Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace pro evidenci klientů projektu „Úspěšný prvňáček“

Lukáš Rod

Katedra softwarového inženýrství
Vedoucí práce: Ing. Stanislav Kuznetsov

4. května 2018

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 4. května 2018 ……

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Rod, Lukáš. *Webová aplikace pro evidenci klientů projektu „Úspěšný prvňáček“*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`https://github.com/rodlukas/bachelors-thesis`⟩.

Abstrakt

Tato práce si klade za cíl vytvořit webovou aplikaci pro projekt, který nabízí doučování a kurzy pro předškoláky. Výsledná aplikace má sloužit jako evidence klientů, jejich docházky, skupin, plateb za lekce a evidování celé historie klienta. Serverová část aplikace je napsána v Pythonu s webovým frameworkem Django. Klientská část je v Reactu a se serverovou částí komunikuje přes REST API díky Django REST Framework. Na závěr bylo úspěšně provedeno akceptační testování, na jehož základě proběhlo vylepšení zjištěných nedostatků. Aplikace je nasazena na hosting a lektorka ji denně používá.

Klíčová slova webová aplikace, evidence klientů a lekcí, kurzy pro předškoláky

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords web application, client and lectures management, courses for preschoolers

Obsah

Seznam obrázků

Seznam ukázek kódu

Úvod

Úspěšný prvňáček[1] (dále jen ÚP) je soubor kurzů vedených speciální pedagožkou PaedDr. Janou Rodovou. Cílem kurzů je pomoci budoucímu nebo nastupujícímu prvňáčkovi rozvíjet se tak, aby byl připraven na školní docházku. Kurzy se zabývají například prevencí selhávání v oblasti čtení a psaní, správným úchopem a držením tužky, rozvojem dovedností leváka nebo prací s neklidným či hyperaktivním dítětem. Některé z kurzů využívají kromě vlastních zkušeností lektorky také revoluční metodiky např. od ruského psychologa D. B. Elkonina nebo izraelského psychologa prof. Reuvena Feuersteina, PhD., tyto metodiky se do českých končin dostávají až v posledních měsících a letech a mezi rodiči jsou především díky jasným a viditelným úspěchům v rozvoji dovedností dítěte hravou formou velmi vyhledávány.

U projektu ÚP jsem již od úplného počátku (červenec 2014), kdy mimo jiné spatřily díky mně světlo světa jeho první webové stránky. Postupem času se projekt rozšiřoval až do fáze, kdy se na základě poptávky rozrostla nabídka kurzů tak, že bylo potřeba celý web od základů předělat. Tento fakt jsem využil i k rozšíření znalostí o novinky v HTML5, CSS3 a vybudoval jsem plně responzivní stránky přesně na míru projektu.

Ruku v ruce s tímto rozšířením samozřejmě opět vzrůstal počet klientů a bylo mimo jiné potřeba přehledně evidovat klienty a lekce. Jako nejrychlejší a v danou chvíli nejjednodušší řešení byla na počátku zvolena jednoduchá tabulka v Excelu doplněná o pár barev. Díky dalšímu nárůstu klientů a zvýšenému zájmu o skupinové kurzy je ale pro lektorku velmi složité udržet evidenci jakkoliv konzistentní, praktickou a přehlednou. Nemluvě o faktu, že ji téměř nelze rozšířit o další funkcionality a rozumnou evidenci skupinových lekcí. Práce s touto tabulkou je zbytečně zdlouhavá, neefektivní, data jsou duplikována ve více souborech a také ve více formách (papír) a jakákoliv změna vyžaduje pevné nervy.

[1] `https://uspesnyprvnacek.cz/`

Přáním lektorky, a tedy i mým cílem, je vytvořit webovou aplikaci, která umožní evidovat klienty, jejich docházku, skupiny, platby za lekce, historii lekcí klienta a další funkcionalitu zjištěnou při analýze požadavků.

Mojí motivací pro vypracování práce na toto téma je především snaha využít technologie jako užitečný a podpůrný prvek projektu, díky kterému se jednotlivé každodenní procesy usnadní a lektorka tak bude moci ušetřený čas využít pro gró projektu, tedy samotné lekce, a neděsit se případnou prací s jakoukoliv aplikací. Druhou a neméně důležitou motivací je možnost prozkoumat, zmapovat a osvojit si některé z moderních technologií, ke kterým se v rámci této práce dostanu a rozšířit si tak své vědomosti a dovednosti.

V teoretické části nejprve zhodnotím aktuální stav evidence a pokusím se najít možná řešení a jejich výhody a nevýhody. Poté představím technologie, vzory a hostingy, které mohou být použity, zhodnotím je a zvolím ty vhodné pro tuto práci.

V praktické části zanalyzuji a popíši požadavky a související procesy v ÚP, navrhnu samotnou aplikaci a její části včetně struktury databáze a postupně uvedu své kroky při implementaci. Poté se budu věnovat průběhu testování včetně závěrečného akceptačního testování a provedeným úpravám. Na konec popíši nasazení výsledné aplikace na zvolený hosting a uvedu možná rozšíření v budoucnu.

Kapitola 1

Cíle práce

Cílem této práce je navrhnout a implementovat webovou aplikaci, která lektorce ÚP umožní a usnadní evidenci klientů, jejich docházky, skupin, plateb za jednotlivé lekce a zobrazení historie absolvovaných kurzů klientů spolu s jejich docházkou (zda chodili pravidelně apod.).

Dalším cílem, který je nutnou podmínkou pro samotnou implementaci, je analyzovat současné řešení, požadavky a související procesy v ÚP a na základě toho zvolit technologie, které budou pro tvorbu aplikace použity (databáze, jazyky, frameworky, hosting). Poté je potřeba navrhnout strukturu databáze s ohledem na požadavky.

Cílovým bodem je provedení akceptačních testů na výsledné aplikaci, příslušné úpravy a opravy zjištěných nedostatků, nasazení na hosting do běžného provozu a návrh možných rozšíření v budoucnu.

Část I

Teoretická část

Kapitola 2

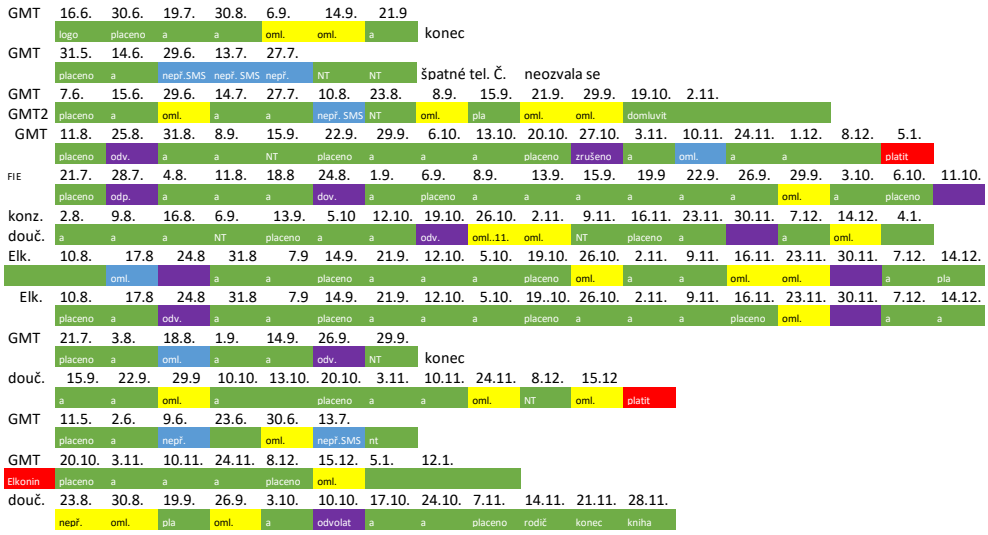
Existující řešení

V této kapitole nejprve popíši, jak je v současné době v projektu ÚP řešena evidence klientů a uvedu nevýhody a výhody řešení. Ve druhé části pak krátce uvedu, zda existují již hotové nástroje, které by se daly pro účely evidence uplatnit.

2.1 Aktuální řešení

Lektorka částečně eviduje klienty a lekce ve svém notebooku v tabulce v programu Microsoft Excel, která je uložená v cloudu na Dropboxu. Ukázka výřezu z tabulky je na obrázku 2.1 (jména klientů jsou úmyslně skryta, dále některé řádky pokračují i mimo výřez). V prvním sloupci jsou jména a příjmení klientů a v dalších buňkách jsou pak vždy v prvním řádku datum lekce a ve druhém barevně (případně i textově další poznámky) informace o stavu účasti a termínu. Pro lepší pochopení je potřeba vysvětlit jednotlivé barvy a zkratky v tabulce:

- **zelená:** zaplacený termín – dále zde může být uvedeno „NT“ (náhradní termín), „a“ (dorazil), „placeno“ (na tomto termínu došlo k zaplacení), pokud není uveden text, mělo by se jednat o předplacený termín,
- **modrá:** klient nedorazil a neomluvil se („nepř.“) – často je doplněno „SMS“ (byla odeslána SMS klientovi, jak to s ním vypadá), nebo „oml.“ (omluveno, ale pozdě, to je potřeba znát kvůli spolehlivosti klienta),
- **červená:** nezaplacený termín – na dané lekci potřeba platit,
- **žlutá:** omluvený termín (značeno „oml.“) – v případě zájmu možno využít náhradní termín,
- **fialová:** termín zrušen z osobních důvodů lektorky (značeno „odv“) – např. dovolená („dov“).



Obrázek 2.1: Výřez ze stávající tabulky s evidencí klientů

Zpočátku tento způsob evidence samozřejmě fungoval (prakticky okamžitě se mohl bez většího učení a práce začít používat) a dokud nebylo potřeba evidovat historii docházky na jiné kurzy klienta a klientů bylo méně, byl soubor relativně udržitelný. Po několika letech ale situace dospěla do stavu, že je třeba evidovat naráz hodně klientů (běží hodně kurzů, i těch dlouhodobějších) a přidávají se i další informace, které je potřeba evidovat, ale už to není v rámci tabulky možné. Rychlé, ale samozřejmě naprosto neudržitelné řešení, je evidování těchto údajů jinde, byla zvolena papírová forma. Tedy v tabulce jsou vedeny jednotlivé lekce (jen datumy bez času), zda je zaplaceno/předplaceno, zda klient došel, omluvil se, nebo je kurz uzavřený. V papírové formě jsou poté evidovány kontakty na klienta, historie kurzů, konkrétní lekce i s časem.

Pro tuto práci je vhodné si přehledně shrnout nevýhody a výhody tohoto vedení informací a tyto závěry pak zužitkovat při tvorbě webové aplikace, která neduhy vyřeší.

2.1.1 Nevýhody řešení

- **nepřehlednost:** když klient chodí dlouhodobě, musí se tabulka posouvat vpravo, pak ale nejsou vidět termíny lekcí ostatních klientů, kteří chodí kratší dobu,

- **nekonzistentnost:** nikdo nehlídá, že červená barva nutně znamená zaplatit, v souborech se často objevují chyby, lektorka omylem přepíše data jiného klienta, nemluvě o nekonzistentních textech a barvách (jiný

font, velikost, zarovnání, někde napsáno „placeno“ a někde pouze „pla“) apod.,

- **neefektivní a nekomfortní práce:** pokud má klient více kurzů najednou nebo je potřeba evidovat skupinové lekce, celá evidence je prakticky neudržitelná a je pracné ji udržet v pořádku,

- **chybí historie klienta:** nelze vést přehledně historii kurzů (a příslušných lekcí) klienta, prakticky řešeno tak, že se jednou ročně v září vytvoří nový soubor, kam se překopírují pokračující klienti a připisují se noví klienti, pokud ještě navíc chodí klient na více kurzů v jednom roce, tak se musí složitě rozlišovat jednotlivé lekce,

- **chybí údaje o klientovi:** neevidují se kontaktní a další informace o klientovi, řešeno papírovou evidencí,

- **chybí jiné pohledy:** chybí přehled pro aktuální den a týden a tato informace se nedá ani snadno dohledat, také chybí čas lekce, řeší se papírovým diářem,

- **neresponzivita:** na tabletu se tabulka špatně upravuje, na mobilním zařízení ještě hůře.

2.1.2 Výhody řešení

- **rychlý rozjezd**: počáteční rozjezd evidence byl rychlý, nebyly potřeba žádné znalosti,

- **žádné poplatky**: lektorka již má Excel nainstalovaný, Dropbox je ve verzi zdarma.

2.1.3 Shrnutí

Je jasné, že i přes to, že je Excel velmi silný nástroj, tak není vhodný k evidenci tohoto typu. Poskytuje sice další pokročilé funkce, díky kterým by se tato tabulka dala vylepšit, ale stále by nebyla schopna pokrýt veškeré požadavky a potřeby lektorky a také by se rozhodně nejednalo o intuitivní a jednoduchou správu (o těžkopádné editovatelnosti v mobilních zařízeních ani nemluvě).

2.2 Podobné aplikace

Po pečlivém hledání jsem nenalezl žádnou webovou aplikaci, která by pokrývala alespoň většinu požadavků. Z nalezených aplikací, které se alespoň

vzdáleně podobají některým požadavkům lze jmenovat snad jen online nástroj RAYNET.cz, což je cloudový CRM (Customer Relationship Management[2]) systém pro řízení vztahů se zákazníky – poskytuje např. databázi kontaktů, historii vztahu s klientem (schůzky, dokumenty, e-maily, reklamace, poznámky), seznam aktuálních zakázek, obchodní výsledky, kalkulace nabídek, kalendář, spolupráci kolegů, fakturaci [2]. Jedná se o moderní, robustní systém, který sice nabízí alespoň nějakou formu evidence klientů s kalendářem a další, ale je nerozšiřitelný a nesplňuje z velké části ani funkční požadavky, které jsou specifické a je tedy potřeba vyvinout webovou aplikaci na míru podle potřeb lektorky: jednoduchou, pokrývající všechny požadavky a neobsahující zbytečné funkce navíc.

[2]systém pro řízení vztahů se zákazníky pro sledování a vyhodnocování obchodních aktivit v rámci společnosti [1]

Kapitola 3

Architektonické vzory ve webových aplikacích

Než se dostanu k samotnému hledání technologií, je potřeba pochopit, jaké architektonické vzory se ve webových aplikacích používají a proč. V této kapitole nejprve popíši dva vzory, se kterými se u různých webových technologií lze setkat. Díky tomu pak budu moci nalezené technologie v další kapitole rozlišovat i podle vzorů, na kterých jsou postaveny.

Tvorba (nejen) webové aplikace nutně nevyžaduje volbu jakéhokoliv architektonického vzoru, ale je dost pravděpodobné, že bez takovéto berličky bude sice možná aplikace docela fungovat, ale může být hůře rozšiřitelná, spravovatelná a pochopitelná. Jakýkoliv zásah pak může vyústit v přepisování kódu celé aplikace (mluvím i ze své zkušenosti).

V poslední části této kapitoly popíši možné přístupy k řešení interakce mezi serverovou a klientskou částí. Díky tomuto rozdělení získám dostatečný přehled k tomu, abych mohl v další kapitole pochopit rozdílné přístupy jednotlivých technologií a rozhodnout se, které možnosti zvolit pro řešení výsledné aplikace.

3.1 MVC a další jeho varianty

MVC (Model-view-controller) vzor dle [3] původně vznikl pro desktopová GUI, ale později se rozšířil zejména mezi vývojáři webových aplikací. Mnoho populárních webových frameworků pro různé jazyky a pro serverovou i klientskou část stojí buď přímo na MVC [3], nebo využívají jeho odvozeninu: MVP (Model-view-presenter), MVT (Model-view-template) ad. – souhrnně jsou někdy tyto architektury označovány jako MVW (Model-view-whatever) nebo MV* [4], pro odkazy na tyto technologie v dalším textu budu používat MVW.

„MVC vychází z teorie, že části kódu, které vykonávají různé úkoly, by měly být od sebe oddělené.“ [5] Nedodržení tohoto přístupu často pak může i díky

absenci disciplíny programátora vést ke špagetovému kódu, vše je smícháno a výsledná aplikace je neudržitelná a těžko rozšířitelná [5][6]. Díky tomuto rozdělení zodpovědnosti se při vývoji může programátor zaměřit pouze na příslušnou část a kód splňuje principy vysoké soudržnosti a nízké provázanosti [7]. Na druhou stranu je ale třeba říci, že se menší aplikace při použití MVC může stát poměrně robustní, je to ale daň za dekompozici do tří částí a s tím spojené již zmíněné výhody [7]. Další výhodou je možnost mít například více pohledů na stejný model [8]. Aplikace je, jak je uvedeno v [5], rozdělena do tří částí:

- modely – starají se o logiku (výpočty, výsledky, získání a uložení dat, validace), při použití ORM (Object-relational mapping) korespondují modely s tabulkami v databázi,
- pohledy – mají na starost vykreslení stránky v HTML (Hyper Text Markup Language) a JS (Javascriptu), do pohledů jsou nejčastěji pomocí šablonovacího systému dodána všechna potřebná data. Modely se
- kontrolery – prostředníci mezi modely a pohledy, propojuje je a říká, co vše je potřeba provést pro výsledek.

Jednoduchým příkladem znázorňujícím tento vzor je např. otevření URL `http://domena.cz/uzivatele/15` – požadavek je zachycen tzv. routerem a na základě parametrů se použije příslušný kontroler, ten zavolá model, který vyhledá daného uživatele v databázi a vrátí jeho údaje kontroleru, který vytvoří pohled a předá mu získaná data [3].

Jak bylo již naznačeno v úvodu této sekce, existují mírné úpravy MVC vzoru. Tento případ lze ilustrovat na frameworku Django pro Python (podrobněji se tomuto frameworku budu věnovat v podsekci 4.2.5), který využívá MVT. Dle jeho autorů [9] pohled nepopisuje podobu dat, ale která data (resp. model) na základě URL jsou zobrazena. Jak se zobrazí data je delegováno z pohledu na šablonu, která je tvořena HTML a šablonovacím jazykem. V tomto pojetí nebyl explicitně zmíněn kontroler, ten je tvořen samotným frameworkem, který odešle požadavek na příslušný pohled podle konfigurace. Zkráceně lze tedy říci, že programátor poskytne model, pohled a šablonu, které poté sváže s příslušnou URL a o zbytek se postará Django [10].

3.2 CBA

V posledních letech nastal poměrně velký zlom v oblasti frameworků pro klientskou část aplikací. Vznikají nové a nové javascriptové frameworky a ty již zaběhlé se přizpůsobují poptávce a často radikálně mění své přístupy k architektuře (např. Angular, viz. podsekce 4.1.3). Více a více se na straně klienta upouští od architektur MVW mj. z důvodu velmi úzké svázanosti kontroleru

a pohledu a také porušování principu jedné odpovědnosti[3] (protože se kontroler stará jak o logiku, tak o zpracování událostí) [12].

Vývoj vyústil ve využití v tomto odvětví doposud nepoužívané architektury CBA (Component-Based Architecture) a s tím spojených „unidirectional" architektur (tedy architektur pro jednostrannou komunikaci ke spravování stavu aplikace). Tyto architektury dokáží nejen pokrýt klasický MVW přístup, ale poskytují i mnohem lepší oddělení odpovědnosti [12].

CBA je podle [13] založeno na rozdělení částí kódu na jednotlivé nezávislé, snadno testovatelné, rozšiřitelné a znovupoužitelné komponenty (v tomto případě tvořící UI), které obsahují všechny potřebné závislosti. Komponenta zapouzdřuje svou funkcionalitu a chování a zároveň navenek poskytuje jednotné rozhraní pro přístup. Díky komponentám tak může být tvorba nebo úprava UI rychleji hotová (lze používat např. i komponenty od jiných autorů) a je zajištěna konzistence napříč celou aplikací. Dalším uplatněním komponent je například jejich souvislost s procesy analýzy požadavků a návrhu během tvorby aplikace, kdy se při přidávání funkcí nemusí zdlouhavě popisovat samotná komponenta a pak teprve její rozšíření, ale stačí definovat pouze rozšiřující funkce [14]. Použití komponent také pomáhá dodržet princip DRY („Don't repeat yourself") [15].

3.3 Interakce mezi serverovou a klientskou částí

Cílem této sekce je ukázat aktuální možnosti řešení interakce mezi serverovou a klientskou částí. Díky tomu, že jsou weby mnohem interaktivnější než dříve, nebylo už často možné úzce svazovat klientskou a serverovou část a bylo potřeba je oddělit prostřednictvím API. To vše v této sekci představím.

3.3.1 Skriptování na straně serveru

Webové aplikace byly typické využíváním skriptování na straně serveru [16], tedy na základě požadavku byl na webovém serveru připraven HTML soubor, který byl poté odeslán do klientova prohlížeče [17] (viz. příklad Djanga na konci sekce 3.1). Používání tohoto způsobu samostatně se hodilo a hodí zejména pro stránky, které například přímo spolupracují s databází, vrací texty a obrázky, ale nevyžadují příliš interaktivity [18].

3.3.2 Skriptování na straně klienta

S trochou nadsázky by se, jak říká autor v [18], dalo říci, že dnes jsou webové stránky spíše aplikace předstírající, že jsou stránky – web je mnohem pokročilejší než dříve, umožňuje chatování, nakupování, prohlížení aktuálního proudu

[3]Princip jedné odpovědnosti je jedno z pravidel objektově orientovaného programování, říká, že by každý objekt měl mít jen jednu odpovědnost (tedy jediný důvod ke změně) [11]

novinek atd. To znamená mnohem větší zátěž na server, který vše musí obstarat. Pro pokročilejší interaktivní aplikace bylo tedy potřeba začít ve větší míře začít používat Javascript, který umožní určitou část zátěže přenést na klientskou část [19] a skriptování na serveru doplnit o skriptování na straně klienta, tedy v prohlížeči. Je ale potřeba říci, že robustní aplikace využívající skriptování na straně klienta může být složitější implementovat, oproti těm využívající především skriptování na straně serveru [20].

Vývoj dnes pokročil až do situace, že pro pokročilé možnosti interaktivity byly připraveny přímo JS frameworky [19], server tedy prakticky jen odešle jednoduchý HTML soubor s odkazy na Javascriptové knihovny, které obstarají zbytek práce u klienta a obsah stránky dynamicky generují a komunikují se serverem, od kterého obdrží pouze potřebné informace bez zbytečné režie navíc ve formě HTML tagů a struktury celé stránky [20]. Tyto frameworky obvykle pracují s API (Application Programming Interface, blíže popíši v podsekci 3.3.3) a AJAX (Asynchronous JavaScript and XML)[4] dotazy [20] a představují tak jednoduchou možnost okamžité a efektivní asynchronní komunikace se serverovou částí [19]. Podobných výsledků lze dosáhnout i bez JS frameworků a explicitně vytvořeného API, jak uvádí např. autor v [21] – jednotlivé pohledy s pomocí šablon na serverové straně vrací buď klasicky celou stránku a nebo pouze JSON, tyto jsou pak využity pro AJAX volání v JS na straně klienta, tento přístup se ovšem hodí pro méně interaktivní aplikace. AJAX a další operace v JS také zaznamenaly zvýšení popularity i díky knihovně jQuery, která přinesla značné zjednodušení práce s JS [19].

Nevýhody spojené se skriptováním na straně klienta

Nevýhodou skriptování na straně klienta je samozřejmě nutnost mít povolený příslušný skriptovací jazyk (tedy většinou Javascript) [17]. Dalším potenciálním problémem může být prvotní čas načtení stránky, především JS souborů, a s tím spojený problém se SEO (Search Engine Optimization)[5], pokud není aplikace implementována korektně nebo její načtení trvá déle (roboti obdrží prázdnou stránku s načítací animací a nemusí vědět, že mají čekat na její načtení) [18][23]. Tento klasický přístup bývá obvykle nazýván jako CSR (Client-Side Rendering) a je obvykle frameworky v základu používán. Na přelomu roku 2016 a 17 se tak začaly objevovat řešení umožňující SSR (Server-Side Rendering), např. balíček Angular Universal, mini-framework next.js pro React [23]. Díky těmto dodatečným úpravám je, jak je ukázáno v [24], aplikace vykreslena na serveru a ve formě stringu je do výchozí kostry HTML vložen předvykreslený kód komponenty a takovýto HTML soubor je zaslán klientovi, který tak vidí bez prodlení stránku a její obsah a pouze čeká na načtení JS, aby byla stránka interaktivní, další interakce v aplikaci pak už probíhají klasicky ve stylu CSR prohlížečem. Tento přístup ale nutně nemusí znamenat zrychlení

[4] technologie pro asynchronní přenos dat na pozadí bez potřeby načíst celou stránku [20]
[5] optimalizace nalezitelnosti na internetu [22]

výkonu a načítání, pokud je například string, který se vygeneruje na serveru velmi dlouhý, dojde naopak ke zvýšení velikosti přenášeného HTML souboru ke klientovi a zvýšení zátěže serveru (kvůli které se na CSR přechází) a tím v důsledku použití SSR naopak ke zpomalení načítání [23]. Při využití SSR je tedy potřeba brát v úvahu mnoho faktorů, mimo již zmíněných také finance a s tím související výkon serveru (server bude více zatížený a nemusí být tolik výkonný), v úvahu také připadá kromě řešení teoretických věcí i praktické testování obou přístupů, tento přístup například zvolili vývojáři v [25] a po vyhodnocení výsledků pro většinu stránek zvolili SSR.

Na závěr si ještě dovolím menší vsuvku k terminologii, skriptování na straně klienta a serveru je často zaměňováno za renderování a naopak (dokonce je i občas špatně interpretován smysl renderování JS na serveru [26]), někteří autoři tyto termíny považují za stejné [18], ale pro tuto práci je vzhledem k tomu, že se častěji opravdu používají v odlišných souvislostech, považuji za odlišné. Renderování (tedy CSR a SSR) je obvykle spojováno pouze s JS frameworky a předvykreslením stránky díky skriptování na straně serveru, kdežto skriptování na straně klienta a serveru je obecně spojováno se všemi možnými jazyky.

3.3.3 API

Obecně uznávanou praxi při vytváření moderní webové aplikace (zejména s JS frameworkem a SPA, viz. další podsekce 3.3.4) je zaslání úvodní stránky od serveru ke klientovi a poté načítání a ukládání dat prostřednictvím jednoduchých zpráv z klienta, neboli prostřednictvím definovaného rozhraní API [27]. Takové rozhraní poskytuje konzistentní, univerzální a flexibilní možnost sdílení dat a funkcionality napříč různými technologiemi a systémy [28]. Klientská a serverová část jsou od sebe odděleny a jsou na sobě prakticky nezávislé, změna na jedné straně (např. v databázi) by neměla vyvolat nutnost změny na straně druhé [29] – například lze snadno vyměnit GUI (JS framework) a použít jiné prakticky bez nutnosti zasáhnout do logiky a funkcionality na serveru.

Ve webových aplikacích se velmi často používá REST (Representational State Transfer) API [27]. Tato architektura podle [30] umožňuje přistupovat k datům na určitém místě (každý zdroj má jeden koncový bod, na který přistoupíme) pomocí metod HTTP (Hypertext Transfer Protocol): GET (získání dat), POST (vytvoření), PUT (úpravy celého zdroje), DELETE (smazání), PATCH (částečné úpravy) a provádět nad nimi CRUD (create-read-update-delete) operace. Součástí HTTP jsou i stavové kódy, na základě požadavku tak součástí odpovědi jsou dle [30] stavové kódy, například:

- **200 OK**: požadavek byl úspěšně proveden,
- **201 Created**: nový obsah byl vytvořen (pro POST),

- **400 Bad Request**: požadavek je nečitelný (např. špatný formát zaslaných dat),
- **401 Unauthorized**: požadavek není autorizován,
- **404 Not Found**: zdroj nenalezen,
- **405 Method Not Allowed**: zdroj není dostupný pro tuto metodu.

Podle [31] se REST spolu s JSON (JavaScript Object Notation) stává defacto standardem pro API webových služeb a spolu s frameworky pro vývoj aplikací na straně serveru poskytuje snadnou možnost jak si vytvořit vlastní REST rozhraní. JSON je formát pro výměnu dat, který se zařadil mezi nejdůležitější formáty na webu. Jeho úspěch tkví v tom, že se s ním díky zápisu dat v souladu s JS pracuje v oblasti zápisu krátkých strukturovaných dat mnohem lépe než s konkurenčním XML (eXtensible Markup Language) [32].

3.3.4 SPA a MPA

V současné době existují dvě možnosti, jak implementovat procházení jednotlivých stránek webovou aplikací: SPA (Single-Page Application) a MPA (Multi-Page Application). Obě tato řešení krátce představím a uvedu jejich výhody, nevýhody a použití.

MPA je podle [33] označení pro tradiční způsob známý již od prvních webových stránek, tedy každá změna vyvolá vykreslení nové stránky v prohlížeči (tedy v panelu prohlížeče se zobrazí např. načítací kolečko, dokud není přenos nové stránky ze serveru dokončen). Využívá se především pro stránky, které mají mnoho úrovní v menu, tedy jsou co se týče struktury velmi rozmanité a bohaté, dále velmi dobře fungují co se týče SEO. Nevýhodou může být úzké propojení mezi klientskou a serverovou částí. MPA nevyžaduje z podstaty věci žádné dodatečné technologie a je velmi jednoduché takovouto aplikaci vytvořit (oproti SPA).

SPA je podle [33] aplikace, která běží přímo u klienta v prohlížeči a nevyžaduje při procházení jakékoliv znovunačítání (indikované samotným prohlížečem), tento způsob používá např. Facebook, Gmail, Google Mapy, Stream, e-shop Arriva ad. Díky této vlastnosti lze poskytnout uživateli vynikající UX (User Experience[6]), protože se najednou aplikace chová přirozeně, neprobliká při přechodu mezi stránkami, prohlížeč neukazuje v panelu načítání – uživatel je stále v jednom prostoru a cítí se tak pohodlně. Aplikace vlastně napodobuje desktopové programy, kde také nedochází k načítání při přechodech [35]. Autorka v [35] dokonce přímo píše „*Nothing beats user experience offered by SPA.*“ . Mezi další výhody SPA dle [33] a [35] patří:

[6] celkový prožitek z používání např. webové aplikace [34]

- rychlé přecházení mezi stránkami – většina zdrojových souborů (HTML, CSS, skripty) je načtena pouze na začátku a během přechodů tak dochází pouze k posílání a přijímání dat jako takových (informace z databáze apod.),
- jednoduchá výměna klientské části (která se v praxi mění častěji) při zachování stejné serverové části,
- možnost znovupoužití kódu serverové části pro nativní mobilní aplikaci, protože veškeré přechody mezi stránkami (kromě zaslání úvodní stránky ze serveru) jsou spravovány na straně klienta a ne serveru, jako v případě MPA – server poskytuje pouze API, které se právě se SPA používá,
- snadnější tvorba offline aplikací.

Pro jednoduchou implementaci SPA se používají JS frameworky, z čehož plyne samozřejmě požadavek na zapnutý JS u uživatele. Také je potřeba počítat s delším prvotním načtením (kód aplikace je delší), už jsem ale zmínil řešení pomocí SSR (viz. sekce 3.3). Dalším potenciálním problémem ve SPA aplikaci je nekorektní odchycení problému, kvůli kterému může dojít ke znemožnění ovládání celé aplikace a minimálně je ji potřeba znovu načíst.

Kapitola 4

Volba technologií

Důležitou volbou při tvorbě aplikace jsou technologie. Odvětví webových aplikací je oproti jiným specifické především svou rychlostí rozvoje. Díky tomu, že jsou webové technologie stále populárnější (například pro svou nezávislost na platformě), rostou i další požadavky programátorů a s tím ruku v ruce vzniká nepřeberné množství dalších technologií, frameworků a knihoven [16][36]. Cílem této kapitoly je poskytnout ucelený základní přehled technologií používaných v současné době pro tvorbu webových aplikací a jejich provozování. Nejprve představím možnosti řešení na straně klienta a serveru, poté zmapuji oblast databází a srovnám hostingy pro provozování výsledné aplikace a v závěrečné části zvolím řešení pro připravovanou aplikaci ÚP.

Ještě před začátkem si dovolím jednu poznámku. Jak uvádí autoři v [37], výběr technologií pouze na základě procházení článků a srovnání je velmi ošemetný a často bude programátor stát na rozcestí, kde budou proti sobě stát dva autoři se svými tvrzeními, která se budou navzájem vylučovat a občas nemusí být snadné najít bez hlubších znalostí a zkušeností to pravdivé. Při popisu technologií (a zároveň pak i výběru) tedy budu klást důraz, jak je řečeno v [37] a [38], na účel a požadavky na aplikaci, snadnost a jednoduchost řešení (např. existenci nástrojů umožňujících rychlý start, tzv. „starter boilerplates“), ekosystém a komunitu kolem dané technologie (čím více lidí, tím je pružnější vývoj, vzniká více knihoven a objeví se více otázek a odpovědí), kvalitní dokumentaci, dospělost (začínající technologie může změnit svůj směr) a také na to, kdo za technologií stojí.

Občas budu uvádět také data z některých průzkumů, aby bylo jasné, nakolik jsou vypovídající, uvádím zde, na jakém vzorku byly prováděny:

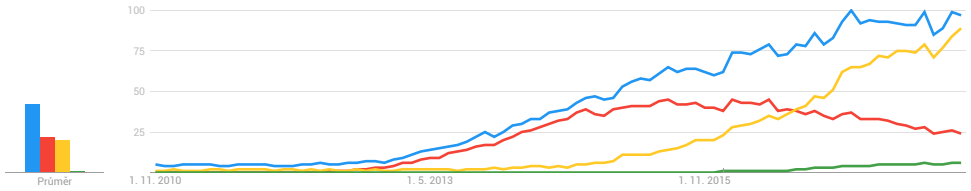
- průzkum Stack Overflow [39] na přelomu ledna a února 2017 absolvovalo přes 64 227 vývojářů z 213 zemí, z toho 70 % zaměstnaných na plný úvazek,
- průzkum Stack Overflow [40] v lednu 2018 absolvovalo přes 101 592 vývojářů ze 183 zemí, z toho 74 % zaměstnaných na plný úvazek,

- průzkum Jetbrains [41] na přelomu roku 2016 a 2017 absolvovalo přes 9 000 vývojářů z 20 zemí, z toho 67 % zaměstnaných ve firmě či organizaci.

4.1 Klientská část

V této části popíši základní technologie, které se používají při tvorbě webových aplikací: HTML, CSS a JS. Jak už jsem zmínil v podsekci 3.3.4, pro pokročilejší interakci a SPA se volí JS frameworky a knihovny, nejprve uvedu dříve velmi často používanou knihovnu jQuery, poté se zaměřím přímo na aktuální frameworky a knihovny. Pochopit rozdíly v tak rychle měnícím se prostředí je náročné (frameworky mění během svých verzí i celou architekturu), informace tak snadno zastarají, stejně jako samotné aplikace. Jedním z aktuálních a velmi dobře hodnocených článků je [42], kde autor velmi rozsáhle shrnuje výhody a nevýhody jednotlivých frameworků. Cílem této práce není ale hloubková analýza všech frameworků, proto se zaměřím na jejich základní popis a výtah z některých článků, aby bylo jasné, čím se frameworky odlišují.

Pro úplnost dodávám, že výčet frameworků zdaleka není úplný, z důvodů uvedených před touto částí v kapitole 4 jsou zvoleny tři v současné době nejpopulárnější podle [43] (seřazeno podle průměru z počtu hvězdiček na GitHub a počtu otázek na Stack Overflow). Pro představu o jejich popularitě ve vyhledávání Google přikládám obrázek 4.1 aktuálního grafu s trendy (je na místě poznamenat, že terminologie kolem verzování Angularu je mírně krkolomná, což je vidět zejména před rokem 2016, kdy existoval jen AngularJS a přesto byl Angular vyhledáván, dále „vue“ je obecné slovo široce užívané i mimo JS frameworky, z toho důvodu je zvolen jako vyhledávací dotaz „vuejs“, což mohlo mírně tomuto frameworku uškodit).



Obrázek 4.1: Celosvětová popularita JS frameworků ve vyhledávači Google od 20. 10. 2010 do 31. 3. 2018 v kategorii Počítače a Elektronika – modře *angular*, červeně *angularjs*, oranžově *react*, zeleně *vuejs* [44]

4.1.1 HTML a CSS

HTML (Hyper Text Markup Language) je značkovací jazyk, bez kterého se dnes při tvorbě webové aplikace nedá obejít, popisuje strukturu webové stránky pomocí HTML značek [45]. V současné době je aktuální používání HTML5, které přináší spoustu dlouho očekávaných funkcí pro moderní web [46], je třeba ale brát na vědomí stále ještě neúplnou podporu všech novinek ze strany všech prohlížečů [47].

CSS (Cascading Style Sheets) je jazyk pro zápis způsobu zobrazení elementů HTML stránky [48]. V současné době se používá CSS3, které přineslo mj. možnost jednoduše vytvářet responzivní stránky, animace a přechody [49]. Stejně jako u HTML5 je třeba brát v úvahu neúplnou podporu všech novinek ze strany všech prohlížečů [50].

4.1.2 Javascript, jQuery

Javascript je skriptovací interpretovaný jazyk, který se obvykle používá pro skriptování na straně klienta (na straně serveru viz. podsekce 4.2.6) – pro jakoukoliv interakci a animace na stránce [51]. Během let se stala mezi programátory velmi populární knihovna jQuery (padla o ní řeč už v podsekci 3.3.2), která výrazně zjednodušuje a zaobaluje syntaxi JS [19]. JS je nejpoužívanější jazyk vycházející ze specifikací ECMAScript, v posledních letech díky implementaci specifikací ECMAScript 5, 5.1, 6, 7 a 8 obdržel spoustu důležitých funkcionalit a syntaktické nadstavby včetně tříd [52].

Opět je zde samozřejmě problém s kompatibilitou v prohlížečích, to se ale obvykle řeší použitím transpileru (překladač mezi dvěma jazyky na stejné úrovni abstrakce) Babel, který může kromě transpilace poskytnout i polyfilly (nahrazují nativní API prohlížeče) [53][54]. Další často používanou nadstavbou pro větší JS aplikace jsou jazyky jako TypeScript, CoffeeScript [55] nebo rozšíření JSX (aby HTML v JS vypadalo jako HTML, ačkoliv se jedná o prosté JS funkce [56]) [57].

4.1.3 Angular, AngularJS

Angular a AngularJS jsou JS frameworky od Google. AngularJS byl, jak je uvedeno v [58], vytvořen v roce 2009 a stavěl na technologii MVC, postupem času se ale začalo ukazovat, že je potřeba použít spíše CBA (viz. sekce 3.2) a změnit přístup i v jiných oblastech.

V roce 2016 byl tak vydán Angular, později nazývaný Angular 2, vzhledem ke zpětné nekompatibilitě a mnoha změnám i v dalších verzích [42] se to z některých stran nesetkalo s kladným přijetím [59][60]. Nabízí bohatou knihovnu včetně API pro HTTP požadavky ad., používá Typescript (popsaný v předchozí podsekci 4.1.2).

4.1.4 React

React je podle [61] knihovna (ne framework, viz. další rozebrání v [60]) od Facebooku, která umožňuje budovat interaktivní uživatelská prostředí. Oproti Angularu je React velmi jednoduchý a malý, protože neobsahuje prakticky žádné dodatečné funkce, v tom spoléhá na komunitu [60], tedy například i pro HTTP požadavky je zde potřeba najít a zvolit vhodnou knihovnu, jinak lze použít pouze běžnou JS syntaxi. Pro tvorbu HTML lze volitelně použít JSX (popis viz. podsekce 4.1.2), je to výhodné zejména z důvodu jednoduššího a přehlednějšího zápisu.

4.1.5 VueJS

VueJS by se s trochou nadsázky dal označit za kombinaci Angularu a Reactu [62], z obou přináší skvělé věci a je velmi kladně hodnocen lidmi, kteří před ním pracovali s Angularem nebo Reactem [60]. Nevýhodou je zatím jeho menší rozšířenost a používanost [60] a také fakt, že za ním nestojí firma, ale jednotlivec, i z toho důvodu jsou zatím ze strany firem zatím upřednostňovány React a Angular [63], jeho popularita se ale pomalu zvyšuje a objevují se týmy, které ho na své větší projekty reálně používají [60].

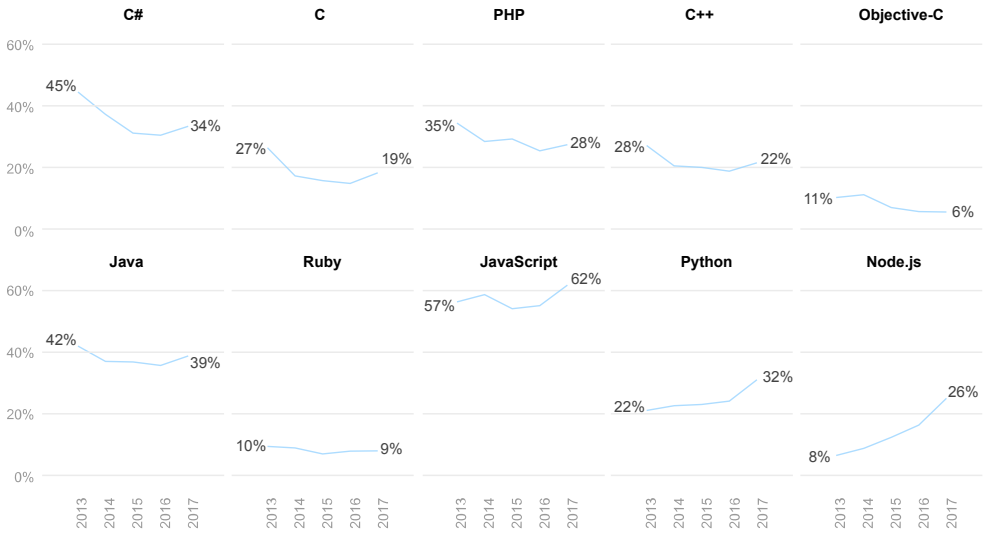
4.2 Serverová část

Volba programovacího jazyka pro serverovou část jde často ruku v ruce s volbou frameworku pro příslušný jazyk. Cílem této části je uvést nejpoužívanější řešení pro webové aplikace včetně možností frameworků. Pro úplnost opět dodávám, že výčet technologií není úplný, podrobněji se nezaobírám např. jazyky C++, Go, Perl a příslušnými frameworky, a to především buď z důvodu velmi specifických možností využití [64] nebo menší rozšířenosti [40][41]. Ze stejných důvodů také volím pouze nejpopulárnější frameworky podle [43].

4.2.1 PHP

PHP je slabě typovaný jazyk, který se obvykle používá pouze ve světě webu [65]. Podle [66] jej využívá přes 80 % webů. Výhodou PHP oproti všem ostatním jazykům je nepřeberné množství levných či bezplatných hostingů s PHP [67] (například velmi populární tuzemský hosting Endora) a také dostupnost nejpopulárnějšího CMS (Content Management System) Wordpress (taktéž napsaného v PHP) [68].

Hlavní výhodou PHP je možnost snadno a jednoduše začít tvořit webovou aplikaci i bez frameworku, to je ale i hlavní nevýhoda, protože mnoho programátorů pak píše špagetový kód a takové aplikace jsou téměř neudržovatelné [69]. I z těchto důvodů se často dle [70] používají frameworky, kterých je oproti jiným jazykům pro PHP mnohem více, např. Laravel, Zend, Symfony a také



Obrázek 4.2: Popularita technologií mezi lety 2013 a 2017 dle průzkumu Stack Overflow, procenta uvádí počet respondentů pracujících s daným jazykem či technologií [39]

tuzemský Nette. Často zmiňovanými nevýhodami jsou např. poměrně nezorganizovaná standardní knihovna [65], názvosloví funkcí a nepohodlná práce při použití UTF-8 [71].

4.2.2 Java

Java je jazyk obecně používaný jak pro webové, mobilní, tak desktopové aplikace, často se používá pro rozsáhlé podnikové systémy díky bezpečnosti a výkonu [72]. Dle [66] je Java třetí nejpoužívanější technologie na straně serveru. Pro webové aplikace nabízí mnoho nástrojů a frameworků, např. Spring MVC, Play, JSF, Google Web Toolkit ad. Zde vyzdvihnu framework Play, který umožňuje stavět jednoduché aplikace v jazycích Java nebo Scala a je vhodný i menší projekty (oproti některým jiným frameworkům [73]) [74].

4.2.3 C#

C# je jazyk s širokým použitím pro vytváření bezpečných a robustních aplikací, aplikace (a dále zmiňované frameworky) pak běží na platformě .NET Framework na Windows nebo na multiplatformním .NET Core [75]. Dle [66] je ASP.NET druhá nejpoužívanější technologie na straně serveru. V roce 2016 Microsoft vydal nový moderní framework ASP.NET Core, který spojuje starší frameworky ASP.NET MVC a ASP.NET Web API [75]. I díky robustnosti celého prostředí se často používá v korporátním prostředí [76].

4.2.4 Ruby

Ruby je jazyk se zajímavou a jednoduchou syntaxí a silným objektovým založením [77]. Používá se ve spojení s frameworkem Ruby on Rails, díky kterému je velmi jednoduché začít a po chvíli programování vidět výsledky [70]. Práce s frameworkem je postavena na mnoha návrhových vzorech a konvencích, což někteří považují za výhodu [70] a jiní za nevýhodu [78].

4.2.5 Python

Python je silně typovaný jazyk, vyniká především svou jednoduchou syntaxí a srozumitelnou standardní knihovnou s dobrou organizací [65], jedná se o obecný jazyk a i díky dostupnosti velkého množství knihoven (např. pro strojové učení a umělou inteligenci) je velmi populární [79]. Aktuálně je ve verzi 3, která je záměrně zpětně nekompatibilní s verzí 2 – dle autora jazyka je cílem napravit hříchy jazyka z minulosti [80]. Jak je vidět na obrázku 4.2, jeho popularita se stále zvyšuje.

Python se původně se pro webové aplikace vůbec nepoužíval, pro jednoduché vytvoření webové aplikace je potřeba použít některý z webových frameworků, nejpopulárnější je Django a Flask [67]. Tyto frameworky lze od sebe snadno rozlišit, jak uvádí [81]: Flask je jednoduchý modulární framework a je na uživateli, jaké součástí si do něj přidá, naproti tomu Django je tzv. „batteries included“ framework, tedy má v sobě spoustu nástrojů, díky kterým může uživatel rychle vyvíjet webovou aplikaci bez nutnosti zabývat se volbou a hledáním konkrétních modulů.

4.2.6 Node.js

Node.js je podle [70] prostředí pro efektivní běh JS na straně serveru, díky JS hojně využívá model událostí a asynchronních operací pro maximalizaci výkonu a minimalizaci režie procesoru. Ideou autorů bylo sjednocení jazyka serverové a klientské části, protože moderní aplikace obvykle stály na použití JS na klientovi, ale ne na serveru.

Popularita Node.js, jak je vidět na obrázku 4.2, strmě roste, na druhou stranu se ale objevují názory [82], že JS patří pouze na stranu klienta. Pro jednodušší vývoj webových aplikací se často používá framework Express [83]. I když programátor zvolí na server jinou technologii, často se i tak s Node.js setká např. ve formě nejpoužívanějšího správce balíčků pro JS s názvem npm, který je na Node.js postaven [70].

4.3 Databáze

Vzhledem k tomu, že volba databáze je často úzce svázána jak s použitými technologiemi, tak s nabídkou na straně hostingu, jsem se rozhodl pouze krátce

Obrázek 4.3: Šest nejpopulárnějších databází v roce 2018 dle průzkumu Stack Overflow [40]

uvést možnosti řešení databází v současné době. Průzkum Stack Overflow [40] na obrázku 4.3 ukazuje šest nejpopulárnějších databázových řešení v tomto roce.

MySQL [84], SQLite [85] (která je velmi jednoduchá a často se např. používá pro prototypy aplikací) a SQL Server jsou relační databáze, PostgreSQL [86] je objektově relační databáze, všechny tedy sdílí tradiční práci s SQL (Structured Query Language). MongoDB [87] a Redis [88] jsou naproti tomu NoSQL databáze, jsou to moderní, flexibilní a škálovatelná řešení často se uplatňující v transakčně náročných aplikacích díky své rychlosti. MongoDB je dokumentová databáze a Redis databáze založená na formátu „klíč – hodnota“ a na využívání mezipaměti.

4.4 Srovnání hostingů

Je důležité rozlišit jednotlivé typy hostingů, kde může být aplikace uložena, pro tuto práci jsou možné 3 způsoby: tradiční hosting, PaaS a IaaS.

Tradiční hosting je velmi populární hlavně díky své cenové dostupnosti a rozšířenosti, na serveru, který má nějakou konfiguraci, systém a komponenty, je každému uživateli vyhrazen jeho prostor [89], často je nabízena funkcionalita emailů ad.

PaaS (Platform as a service) a IaaS (Infrastructure as a Service) jsou podle [90] cloudové služby, které se od sebe liší mírou toho, co umožní uživateli, a co dělají za něj, společné mají to, že poskytnou uživateli virtuální prostředí. IaaS je prakticky pronájem hardwaru na dálku, nabízí zákazníkům plnou kontrolu nad vším od systému až po databázi a aplikace. PaaS je v tomto ohledu striktnější a od poskytovatele dostanou uživatelé předpřipravené prostředí s mnoha doplněnými funkcemi včetně propojení s verzovacími systémy, mohou si vybrat konkrétní databázi a jazyk, ale nemají přístup k samotnému

systému a dalším funkcím, díky tomu se ale mohou zaměřit na samotný vývoj aplikace, testování a nasazování.

4.4.1 Heroku

Heroku od firmy Salesforce.com je PaaS služba, která podle [91] podporuje jazyky Ruby, PHP, Go, Python, Java, Scala, Clojure, technologii Node.js a jako výchozí databáze nabízí PostgreSQL a také Redis. Poskytuje několik programů, včetně jednoho zdarma [92], ten má samozřejmě několik omezení:

- po 30 minutách neaktivity aplikace usíná a chvíli trvá, než se probudí,
- je k dispozici až 1 000 hodin provozu měsíčně pro jeden účet (to není problém, protože v rámci účtu neběží žádné další aplikace a 1 000 hodin je v přepočtu přes 41 dnů provozu na měsíc),
- maximálně 10 000 řádků v databázi PostgreSQL (rozšíření na 10 milionů stojí 9 $ měsíčně)

Výhodou Heroku je, že i v programu zdarma dává možnost vybrat servery v Evropě [93], také nabízí velkou škálu předpřipravených doplňků, díky kterým je možné aplikaci rozšířit o další funkcionalitu. Pokud by byl potřeba pokročilejší program mj. bez usínání aplikace, příplatek činí dalších 7 $ měsíčně.

4.4.2 DigitalOcean

DigitalOcean je podle [94] služba IaaS poskytující distribuce Ubuntu, CentOS, Debian, Fedora, CoreOS a FreeBSD. Obsahuje spoustu předpřipravených možností a návodů, které usnadní start aplikace v typických prostředích. Servery má i v Evropě [95]. Nejlevnější řešení stojí podle [96] 5 $ měsíčně.

4.4.3 Openshift

Openshift od firmy Red Hat je služba PaaS, která podle [97] umožňuje hostovat projekty v jazycích Java, Python, Perl, Ruby, PHP, technologiích .NET Core, Node.js na serverech Apache a Tomcat. Co se týče databází, nabízí MySQL, PostgreSQL, Redis, MariaDB a MongoDB. Má dva programy [98]: bezplatný a placený za 50 $ měsíčně. Bezplatné hostování je především limitováno uspáním po 30 minutách neaktivity (probuzení chvíli trvá a zároveň spaní musí zabrat alespoň 18 hodin z každého 72hodinového intervalu), další nevýhodou bezplatné verze jsou servery umístěné v Severní Americe.

4.4.4 PythonAnyWhere

PythonAnyWhere je podle [99] a [100] Paas nabízející hosting pro aplikace v Pythonu – zdarma umožňuje provozovat aplikace s omezeným výkonem a MySQL databází, vyšší výkon včetně dalších možností (např. PostgreSQL) lze získat za příplatek 5 $ nebo více. Servery má v USA [101].

4.4.5 Další možnosti

Dalšími možnostmi jsou například PaaS služba Google App Engine [102] a IaaS služba od Amazon Web Services s názvem EC2 [103], obě mají velmi rozsáhlé ceníky se spoustou pokročilých variant včetně kalkulátorů a není lehké se v nich vyznat (například AWS dává k dispozici 12 měsíců s určitými podmínkami zdarma, ale při jejich překročení nebo vypršení roční lhůty přesune uživatele do placeného programu bez možnosti návratu [104]). Také je potřeba zmínit český hosting Roští, který podle [105] nabízí hosting aplikací v Pythonu, Ruby, PHP a Node.js, nabízí relativně levný program za 99 Kč měsíčně. Vzhledem k tomu, že už jsem našel rozumné varianty (dokonce i zdarma), které pro rozjezd projektu postačí, nebudu se těmito službami hlouběji v této práci zaobírat.

4.5 Zvolené řešení

Na úvod bych chtěl krátce uvést své zkušenosti s technologiemi v oblasti webových aplikací. V minulosti jsem si vyzkoušel práci s Javou a Servlety, častěji jsem také pracoval s čistým PHP, nikdy jsem ale nepracoval s frameworky. Na základě těchto zkušeností jsem se rozhodl zvolit pro tuto práci nějaký framework, který mi usnadní práci. Co se týče klientské části, zde jsem pracoval s čistým JS a případně jQuery, vzhledem k pokročilejší interaktivitě aplikace v rámci této práce a na základě předchozích zkušeností jsem chtěl využít služby frameworku/knihovny i na straně klienta. V oblasti databází jsem pracoval především s MySQL a PostgreSQL. Co se týče hostingů, mám zkušenosti s klasickými tuzemskými hostingy, tedy kombinace Apache, PHP, MySQL, z týmového projektu také mám zkušenosti s Heroku.

Při volbě technologií jsem také využil možnost vidět díky projektu RealWorld[7] reálnou aplikaci včetně kódu využívající různé serverové a klientské technologie. Jak autor popisuje v [106], rozhodli se tento projekt vytvořit především kvůli rychle se rozvíjejícímu odvětví webových technologií a prakticky nemožnosti zmapovat najednou všechny technologie jedním člověkem – existovaly sice projekty ukazující tímto způsobem v mnoha technologiích vytvořenou aplikaci na správu úkolů, ale takovýto typ aplikace se většinou liší od CRUD aplikace, kterou většina lidí vytvoří, zvolili tedy blog. Díky tomu

[7] `https://github.com/gothinkster/realworld`

se na jejich repozitáři nachází implementované aplikace v různých jazycích a frameworcích, a to jak pro klientskou část, tak pro serverovou, a na dalších technologiích se už pracuje [107].

Na základě všech informací zjištěných během této rešerše a dalšího hledání jsem se rozhodl využít na serverovou část Python 3 s frameworkem Django (v čerstvě vydané verzi 2.0 z prosince 2017, která mj. výrazně zjednodušuje zápis URL routování [108]) a React na klientské části (JS a JSX spolu s HTML a CSS) komunikující skrze API vystavené serverem. Klientská část bude realizovaná konceptem SPA a pro začátek bude využívat renderování pouze na straně klienta. Python s Djangem je velmi populární volbou pro spoustu projektů (viz. např. [40]).

Python jsem zvolil, protože se jedná o stále více populární jazyk a díky této práci tak budu mít možnost se ho naučit (podle průzkumu Jetbrains [41] je to jazyk, který by se chtělo naučit nebo na něj přímo přejít nejvíce programátorů, ke stejnému názoru došel i průzkum Stack Overflow [40], kde je vidět, že patří Python patří do trojice nejvíce oblíbených jazyků dle osobních preferencí) a vyzkoušet si jeho jednoduchou syntaxi a práci s ním. Django jsem zvolil z několika důvodů, zaprvé mě při procházení dokumentací jeho dokumentace zaujala svou rozsáhlostí a poměrně přátelským přístupem, zadruhé se jedná o „batteries included“ framework a protože na klientské části bude React, který je přesným opakem (tedy obsahuje jen nutný základ a je na uživateli, co zvolí dalšího), rozhodl jsem se zvolit jeden přístup na straně klienta a druhý přístup na straně serveru.

React jsem zvolil především kvůli jeho relativně stabilní architektuře [42], Angular se výrazně proměnil a vznikl Angular 2 s architekturou bližší Reactu (viz. podsekce 4.1.3), ale dostupné návody, články a literatura je spíše pro starší AngularJS (často jsou tyto názvy zaměňovány, což je docela matoucí pro nově příchozího). Volba také souvisela s Djangem, kdy jsem chtěl zkusit využít pro klientskou část spíše modulárnější technologii i vzhledem k častěji se měnícího UI oproti serverové části. VueJS jsem nezvolil kvůli jeho menší rozšířenosti a nejistotě budoucnosti (není za ním firma).

Protože cílem této práce má být jednoduchá a uživateli srozumitelná aplikace, je třeba poskytnout mu co nejlepší uživatelský prožitek a zkušenost, z toho důvodu je zvolen přístup SPA, který je často ve spojení s JS frameworky používán (viz. podsekce 3.3.4).

Jako databázi použiji PostgreSQL, protože se jedná o základní databázi, kterou poskytuje Heroku, které bylo na základě srovnání vybráno jako nejvhodnější hosting (může být zdarma i se servery v Evropě a zároveň s ním mám už zkušenosti z jiného projektu).

Část II

Praktická část

Kapitola 5

Analýza

Cílem této kapitoly je analyzovat entity, procesy a fungování projektu a sestavit postupně sestavit funkční a nefunkční požadavky na výslednou aplikaci. Na základě toho pak může být započat návrh jednotlivých částí aplikace.

5.1 Procesy a entity

V této sekci popíši fungování projektu, které rozdělím do několika částí, resp. entit. Vzhledem k tomu, že se v projektu pohybuje jediný člověk, lektorka, a tato aplikace slouží jako podpůrný prvek procesů v projektu (tedy např. neřeší objednání klienta, to proběhne např. telefonicky, osobně, e-mailem, pouze zaznamenává potřebná data a umožňuje s nimi pracovat a zobrazovat tak, aby zefektivnila jednotlivé procesy), je upuštěno od modelování procesů ve prospěch podrobného textového popisu, na základě kterého bude vytvořen co nejpřesnější návrh této aplikace.

5.1.1 Kurzy

ÚP nabízí v současné době 6 kurzů, plánují se ale i další, stejně tak ale některé mohou být ukončeny. Každý kurz má svůj název. Jsou individuální a skupinové, neexistuje ale žádná přímá souvislost mezi kurzem a způsobem jeho výuky, protože se vše přizpůsobuje na míru klientovi, tedy některý kurz může sice být obvykle vyučován skupinově, ale někdy také individuálně. Mají variabilní délku, většinou od 1 měsíce až po celoroční. Všechny kurzy jsou placené a většinou je na rodičích, zda zaplatí celý kurz, platí měsíčně, každou lekci a nebo úplně individuálně (platí se v hotovosti nebo převodem). Konají se ve všední dny, v současné době 3x týdně od odpoledne až do večera. Na kurz se klient objednává telefonicky, e-mailem, zprávou nebo osobně.

5.1.2 Klienti

Účastník kurzu se nazývá klient. Kurzy navštěvuje buď sám za sebe, tedy individuálně, nebo je součástí nějaké skupiny (viz. následující podsekce 5.1.3). U klienta je potřeba evidovat jméno a příjmení, e-mailový a telefonní kontakt na rodiče a případně další poznámky, vzhledem k tomu, že v současné době byla evidence používána v podobě uvedené v sekci 2.1, často některé kontaktní údaje schází a je třeba s tím počítat.

5.1.3 Skupiny

Zejména v poslední době se v projektu zvyšuje počet skupinových lekcí. Skupinu tvoří většinou 2 až 4 klienti, kteří v rámci této skupiny dochází na příslušný kurz. Lektorka si pro skupiny vytváří jejich jméno (které vychází z názvu kurzu a pořadového čísla skupiny v tomto kurzu), aby se v nich mohla orientovat. Je potřeba poznamenat, že klient může skupinu opustit a stejně tak se k nějaké stávající připojit, to samozřejmě znamená, že další lekce už budou pouze s těmi, kteří jsou stále ve skupině (včetně nově příchozích). K opuštění skupiny obvykle odchází buď z časových důvodů (tedy docházka dočasně nebo trvale končí), nebo kvůli přechodu na individuální formu téhož kurzu (např. kvůli pomalejšímu tempu). I po opuštění skupiny je stále potřeba evidovat předchozí docházku klienta (pro případ, že se znovu přihlásí, nebo přešel na individuální formu téhož kurzu a je potřeba vidět, že část odchodil skupinově).

5.1.4 Lekce

Každý kurz se skládá z jednotlivých lekcí. Lekce jsou termíny, které každý klient dostane a dochází na ně. Jak bylo uvedeno v podsekci o kurzech 5.1.1, za lekce se platí, většinou je ale na klientovi, jaký způsob placení zvolí. Pokud se jedná o skupinu, většinou každý platí jiným způsobem a v jiné termíny. Vzhledem k různorodosti způsobů placení jsou rodiče většinou rádi, že si nemusí nic pamatovat, dochází na kurz a poprosí lektorku, aby je upozornila, že si mají přinést příště peníze, někteří pak kurzy raději platí rovnou celé. Někdy se stane, že klient na kurz nedorazí, pro účely historie docházky a platby je potřeba rozeznat, zda se omluvil nebo nepřišel bez omluvy. Výjimečně se také může stát, že je termín zrušený ze strany lektorky z osobních důvodů. Kromě této docházky se samozřejmě musí evidovat u klientů, zda mají lekci zaplacenou a případně další poznámky. Vzhledem k tomu, že stále častěji rodiče volí předplácení, je tedy potřeba evidovat jak naplánované lekce, tak nenaplánované (tedy ty, co jsou zaplacené, ale nemají ještě přidělený a domluvený termín).

5.2 Požadavky

V této sekci shrnu funkční a nefunkční požadavky na výslednou aplikaci. Všechny požadavky nebyly známé hned na počátku, některé z nich vznikly díky inkrementálnímu přístupu k analýze, návrhu a implementaci. Při návrhu aplikace jsem například zjistil, že by se lektorce hodil také týdenní pohled na lekce (jako má v diáři), dále přibylo zobrazení čísla lekce ad. Součástí zadání této práce je požadavek, že se má jednat o webovou aplikaci. Tento požadavek tedy neuvádím v přehledu požadavků níže a beru jej jako výchozí.

5.2.1 Funkční požadavky

- **evidence klientů:** systém umožní CRUD operace s klienty, u klienta je třeba zaznamenat jméno a příjmení dítěte, telefonní a e-mailový kontakt na rodiče a poznámku pro další informace,

- **evidence lekcí klientů:** systém umožní evidovat, na které kurzy klient chodí (nebo chodil), a to jak individuální, tak skupinové, bude možné všechny tyto informace zobrazit na jednom místě,

- **evidence údajů o lekci:** součástí evidovaných dat pro každou lekci bude datum a čas, stav účasti (omluven, neomluven, odvolán ze strany lektorky), zda je zaplaceno a další poznámka (například informace, že klientovi byla zapůjčena knížka),

- **evidence předplacených lekcí:** aplikace umožní evidovat předplacené lekce klientů,

- **přehled pro aktuální den:** na hlavní stránce po přihlášení je potřeba zobrazit plán pro aktuální den – klienty spolu s dalšími informacemi o lekci (stav účasti, platba, kurz, datum, čas, číslo lekce, zda mají příště platit a další poznámky), v případě zrušené lekce ze strany lektorky se lekce v denním přehledu nemá zobrazit (zobrazí se pouze v kartě klienta a v týdenním přehledu)

- **další upozornění:** u lekcí, které jsou jako poslední zaplacené je potřeba upozornit klienta, že příště musí platit, tedy je třeba toto upozornění lektorce připomenout u příslušné lekce,

- **počítání lekcí:** u lekcí je potřeba zobrazit, o kolikátou (navštívenou, tedy nepočítají se omluvené) lekci v rámci kurzu se jedná,

- **týdenní přehled:** aplikace umožní zobrazit lekce v týdenním přehledu (pouze pracovní dny) a umožní mezi týdny přecházet jako v diáři.

5.2.2 Nefunkční požadavky

- **podporované prohlížeče:** aplikace bude plně funkční a kompatibilní s posledními verzemi běžně používaných prohlížečů, tedy Mozilla Firefox, Google Chrome, Microsoft Edge, Apple Safari (primárně bude ale využívána na desktopovém Firefoxu),
- **podporovaná zařízení:** aplikaci bude přizpůsobená zejména pro používání na notebooku (kde bude používána primárně, jedná se o rozlišení 1920 × 1080, 15,6“), bude ji ale možno bez omezení používat i na tabletu (iPad s iOS 11.3 a 9,7“ displejem) a chytrém telefonu (s Androidem 8.0 a 5,2“ displejem),
- **připravenost na rozšíření a údržbu:** aplikace bude vytvořena tak, aby byla snadno rozšířitelná a upravitelná, je totiž plánovaný rozvoj projektu a rozšíření aplikace o další součásti (např. evidence prodejů pomůcek a učebnic ad.),
- **srozumitelné a jednoduché rozhraní aplikace:** aplikace tvoří podpůrný systém pro zefektivnění a urychlení práce, musí dát možnost uživateli co nejsnadněji, nejrychleji a nejpochopitelněji provést každý úkon a rychle zjistit potřebné informace,
- **bezpečnost:** aplikace obsahuje osobní data klientů, je tedy třeba zajistit odpovídající úroveň zabezpečení, aby se případný útočník nemohl dostat k citlivým údajům.

Kapitola 6

Návrh

V této kapitole nejprve popíši návrh logického datového modelu aplikace a vysvětlím, které důvody mě vedly k vytvoření modelu zrovna takovýmto způsobem. Poté bude následovat návrh architektury aplikace, při kterém vyjdu z už zvolených technologií v sekci 4.5. Krátce uvedu, jak probíhal návrh uživatelského rozhraní spolu s ukázkou a na závěr popíši navržené komunikační rozhraní API pro komunikaci mezi serverem a klientem.

6.1 Datový model

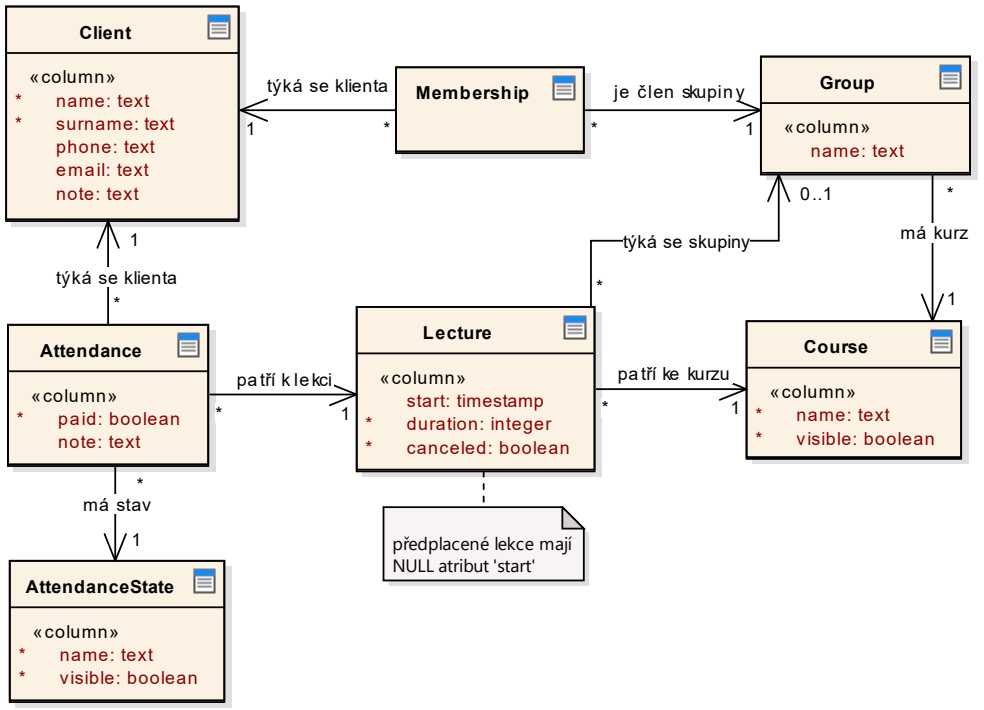
Pro modelování databáze jsem využil logický datový model, který umožní platformní nezávislost, je na obrázku 6.1. Tento model pak může být jednoduše převeden na platformě závislý a také na SQL skript pro vytvoření dané databáze, vzhledem k využití frameworku Django ale ani jeden převod nebude nutný (důvody uvedu v sekci 7.4), a tak pro vizuální představu postačí pouze tento model. Model byl tvořen i s důrazem na budoucí rozšíření a je tedy připraven na zásahy uvedené v kapitole 10.

6.1.1 Client

Entita klienta slouží k evidenci informací o klientovi, vyžadováno je jméno a příjmení, nepovinně telefonní a e-mailový kontakt na rodiče a poznámka. Kontakty jsou nepovinně zejména kvůli tomu, že doteď nebyly evidovány na jednom místě a někdy dokonce vůbec a bude potřeba již stávající klienty do evidence samozřejmě přidat (viz. podsekce 5.1.2).

6.1.2 Membership

Členství slouží ke dvěma účelům. Zaprvé k dekompozici M:N vztahu klienta a skupiny (klient může být ve více skupinách a skupiny mohou mít více klientů), vím tedy, který klient patří do které skupiny. Zadruhé pro možné



Obrázek 6.1: Logický datový model

budoucí využití (původní záměr byl vytvořit atributy pro začátek a konec členství, na základě další analýzy a návrhů se ukázalo, že ale zatím postačí jednodušší varianta, tedy zda je teď členem nebo není).

6.1.3 Group

Entita skupiny slouží k evidenci jednotlivých skupin. Atribut jméno skupiny je nepovinný (ačkoliv bude obvykle používán, jeho použití není vyžadováno, protože to není nutné). Dále skupina ví, ke kterému kurzu náleží, tato vazba je zde proto, že skupiny už od počátku svého vytvoření patří vždy k právě jednomu kurzu (oproti klientovi, který takovou vazbu nepotřebuje).

6.1.4 Course

Kurzu drží dva atributy: jméno kurzu a viditelnost, oba jsou povinné. Viditelnost slouží k tomu, aby bylo v rámci aplikace možné při přidávání lekcí či skupiny skrýt z možností kurzu příslušný kurz, pokud už není provozován (aby byla zachována předchozí evidence lekcí daného kurzu).

6.1.5 AttendanceState

Stav účasti slouží k evidování možností stavu účasti klientů na kurzu, předpokládanými stavy jsou např. „omluven“ a „nepřišel“. Tato entita drží dva atributy: název stavu účasti a viditelnost, oba jsou povinné. Viditelnost opět slouží k tomu, aby bylo v rámci aplikace možné pro nově přidávané lekce skrýt z možností stavu účasti klienta příslušný stav, pokud už není využívaný (aby byla zachována předchozí evidence lekcí daného kurzu). Tím, že mám jednu entitu, která říká, jaké jsou možné stavy účasti, může být aplikace konzistentní, zároveň pro úpravu názvu není potřeba upravit každý stav zvlášť, dále lze jednoduše stav účasti přidat a začít jej používat, případně jej přestat používat a skrýt z budoucích nabídek (nebo v případě žádného použití i smazat).

6.1.6 Attendance, Lecture

Vzhledem k úzkému propojení těchto dvou entit spojuji jejich popis do jedné podsekce. Účast a lekce jsou jádrem aplikace a vzhledem k tomu, že je k nim potřeba z různých entit přistupovat, bylo potřeba jim věnovat dostatečný čas v návrhu, aby byl kvalitní a nebylo nutné jej v průběhu vývoje měnit.

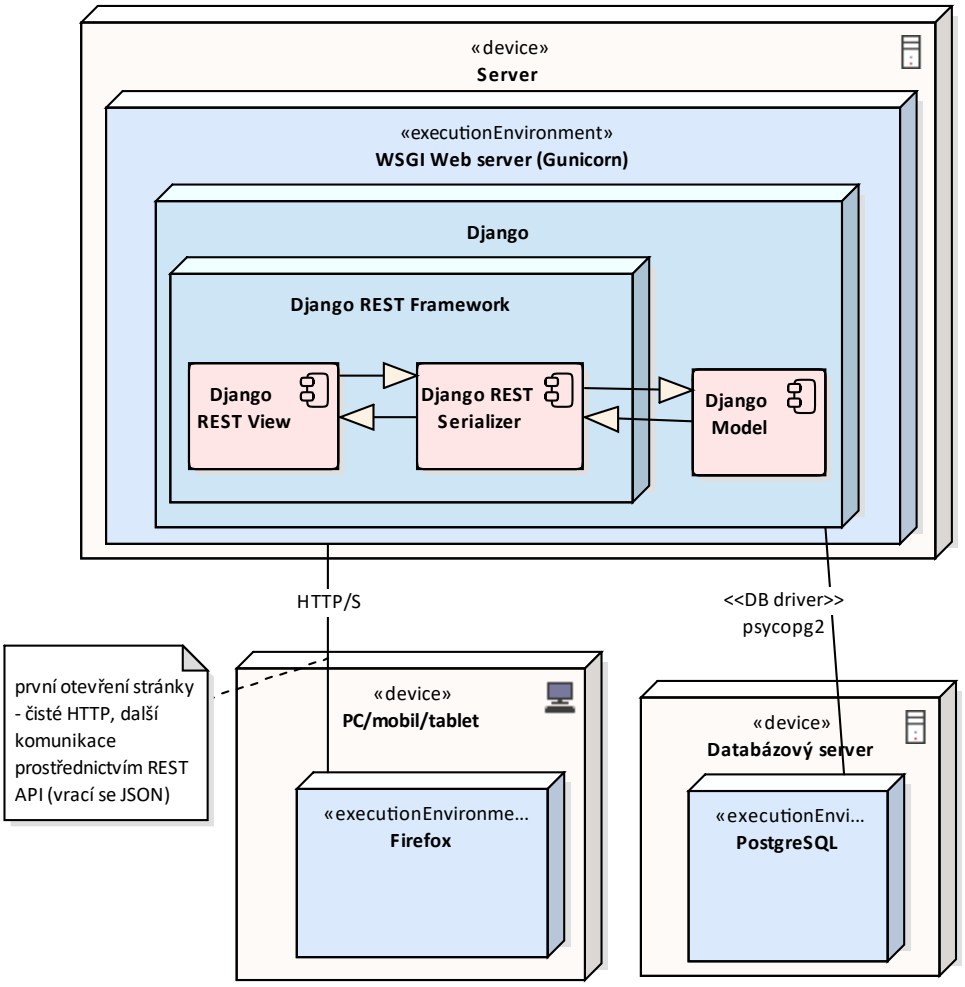
Je potřeba evidovat lekce, každá má povinné atributy trvání (plánuje se funkcionalita upozornění na překryv lekcí), zda je zrušena (díky tomu lze jednoduše zrušit i skupinové lekce, oproti přístupu, kdy bychom zrušení evidovali prostřednictvím entity stavu účasti) a nepovinnou časovou značku začátku lekce (tzv. timestamp). V případě, že se jedná o naplánovanou lekci bez zatím známého konkrétního termínu, atribut začátku obsahuje hodnotu `null`. Ke každé lekci eviduji k ní náležící právě jeden kurz a nepovinně také skupinu, pokud se nejedná o individuální kurz. Tato vazba je potřeba pro individuální lekce (skupiny jsou řešeny vazbou přímo z entity skupin), tento způsob byl shledán po vyhledávání alternativních možností tím nejvhodnějším.

Entita účasti je úmyslně oddělena od lekce, aby každý klient, kterého se lekce týká, mohl mít své údaje o účasti (což je v případě skupiny potřeba). Každá účast tedy ví, ke které jedné lekci a jednomu klientovi náleží, a také je k ní navázán právě jeden stav účasti příslušného klienta.

Příkladem budiž lekce z pohledu entity účasti – pro případ individuální lekce má účast navázaného jednoho klienta, jeden stav účasti a jednu lekci, která je dále navázána na jeden kurz. Pro případ skupinové lekce se čtyřčlennou skupinou jsou využity 4 záznamy v klientech, jeden záznam ve skupině a tyto záznamy jsou spojeny entitou členství. Skupině náleží jeden kurz. Nyní mohu vytvořit pro každého z účastníků účast navázanou na každého z nich, na stav účasti a také všechny navázané na stejnou lekci.

6.2 Architektura

Na obrázku 6.2 je diagram nasazení. Architektura má tři části: server, klient a databázový server. Na klientovi (počítač, telefon, tablet) běží webový prohlížeč. Na serveru běží webový server Gunicorn, což je Python WSGI HTTP Server (WSGI znamená, že splňuje požadavky na rozhraní Web Server Gate Interface, a tedy umožňuje komunikovat Django aplikaci, která WSGI vyžaduje, s webovým serverem). PostgreSQL databáze bude na samostatném serveru.



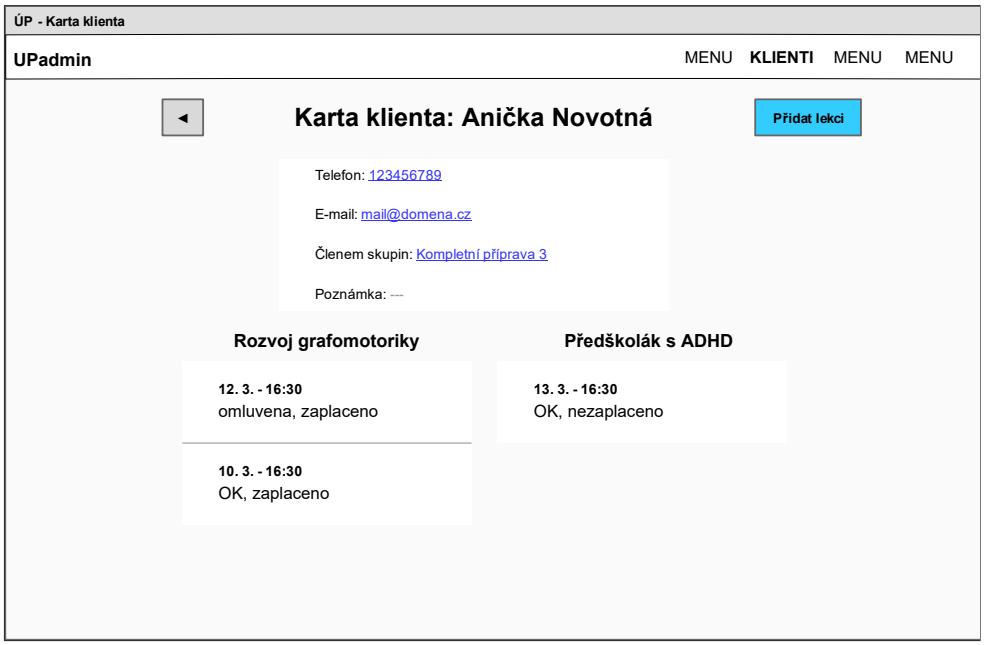
Obrázek 6.2: Diagram nasazení

Jak již bylo řečeno v sekci 4.5, serverová část je v jazyce Python a frameworku Django a s klientskou částí v jazyce Javascript a frameworku React komunikuje přes REST API ve formátu JSON. Django je postavené na archi-

tektuře MVT (popis v sekci 3.1) a React na architektuře CBA (popis v sekci 3.2). První požadavek a odpověď budou čistě v HTTP/S, tedy klient požádá o stránku, server mu ji celou vrátí, další komunikace už bude probíhat přímo přes REST API a odpovědi budou v JSON (díky tomu může být aplikace SPA, viz. 3.3.4). Python, a tedy i Django, umí v základu komunikovat pouze s SQLite databází, pro další databáze je třeba využít adaptér – pro PostgreSQL se používá psycopg2. O vyřízení API požadavků je požádán Django REST Framework, konkrétně část View, která poté prostřednictvím Serializeru využije příslušné Django Modely (tedy ORM) a z databáze tak získá data.

6.3 Uživatelské prostředí

Návrh uživatelského prostředí je neodmyslitelnou součástí tvorby webové aplikace. Vzhledem k tomu, že byl návrh tvořen v několika iteracích, zvolil jsem pohodlnější a rychlejší kreslení na papír. Díky tomu jsme při konverzaci nad budoucí podobou aplikace s lektorkou mohli některé nápady okamžitě zahodit a vytvořit bez problémů nové.



Obrázek 6.3: Návrh karty klienta

Na obrázku 6.3 je jeden z návrhů z poslední iterace překreslený z papíru do aplikace Pencil, jedná se o návrh karty klienta. Na základě těchto návrhů jsme doiterovali až do stavu, kdy jsme přesně věděli, co od aplikace kde a jak čekat.

6.4 Komunikační rozhraní

Bylo potřeba navrhnout API tak, aby vystavilo všechny příslušné body potřebné pro práci na klientské části. Při návrhu jsem vycházel mj. z doporučení v [109] – pro všechny body jsou názvy v množném čísle, využívám naplno všechno dostupné HTTP metody GET, POST, PUT, PATCH a DELETE, nevystavuji zbytečně adresy obsahující prováděné akce (např. vytvoření) a adresa všech níže dále uvedených bodů API vždy začíná `/api/v1/` (obsahuje tedy i označení verze API).

Jak bylo již řečeno v předchozím odstavci, API kromě běžných operací umožňuje také PATCH požadavek, ten slouží k datově méně náročné částečné aktualizaci údajů. Tuto metodu lze použít například v případě označení lekce jako zaplacené – v tomto případě není nutné přenášet další údaje, stačí `id` a upravovaný údaj. GET požadavky často, z důvodu snížení počtu požadavků na server, rovnou obsahují zanořená data (která jsou stejně vždy potřeba).

6.4.1 Klienti

Bod pro klienty pracuje s klíči `id`, `name`, `surname`, `phone`, `email` a `note`. Pro GET požadavky jsou klienti seřazeni vždy podle abecedy vzestupně, a to podle příjmení a poté jména.

`clients/`	**GET**	vrátí všechny klienty
`clients/`	**POST**	vytvoření nového klienta
`clients/:id/`	**GET**	vrátí klienta s `id`
`clients/:id/`	**PUT**	úprava klienta s `id`
`clients/:id/`	**PATCH**	částečná úprava klienta s `id`
`clients/:id/`	**DELETE**	smazání klienta s `id` (lze smazat pouze pokud nemá žádné lekce)

6.4.2 Lekce

Bod pro lekce pracuje s klíči `id`, `start`, `group`, `canceled` a `duration`, to ale není vše. V případě skupinové lekce klíč `group` obsahuje zanořené informace o skupině. Součástí odpovědi jsou i zanořené informace o kurzu (klíč `course`) a jednotlivých účastech klientů (klíč `attendances`), které obsahují platbu (klíč `paid`), poznámku (klíč `note`) a navíc také zanořené informace o každém z klientů (klíč `client`), stavu účasti (klíč `attendancestate`), vypočítané informace o číslu lekce v pořadí (klíč `count`) a zda je potřeba připomenout příští platbu (klíč `remind_pay`). Vypočítané informace jsou pouze v odpovědích a při požadavcích na úpravu či vytvoření se neuvádějí, dále zanořené informace kromě samotných účastí se zadávají pouze formou `klíč_id` (tedy např. `attendancestate_id`).

Každý z dotazů na lekce lze doplnit také o parametr `ordering=start`, resp. `ordering=-start` pro seřazení výsledků dle atributu `start` vzestupně,

resp. sestupně (s posledními třemi dotazy, které již výsledky filtrují, lze toto řazení připojit přes operátor `&`), výchozí řazení je sestupně (tedy od posledních lekcí k nejstarším). Kromě dotazu s uvedeným přesným datumem v parametru se vrací i zrušené lekce.

`lectures/`	**GET**	vrátí všechny lekce
`lectures/`	**POST**	vytvoří novou lekci
`lectures/:id/`	**GET**	vrátí lekci s `id`
`lectures/:id/`	**PUT**	úprava lekce s `id`
`lectures/:id/`	**PATCH**	částečná úprava lekce s `id`
`lectures/:id/`	**DELETE**	smazání lekce s `id`
`lectures/?group=:id`	**GET**	vrátí lekce skupiny s `id`
`lectures/?client=:id`	**GET**	vrátí lekce klienta s `id` (jen individuální)
`lectures/?date=:date`	**GET**	vrátí lekce (bez zrušených) konající se v zadaný datum `date` (ve formátu YYYY-mm-dd)

6.4.3 Skupiny

Bod pro lekce pracuje s klíči `id`, `name` a dále se zanořenými informacemi o kurzu (klíč `course`) a členech skupiny (klíč `memberships`) se zanořenými informacemi o klientovi (klíč `client`). Pro úpravy a vytváření se opět místo zanoření atributy zadávají pouze formou `klíč_id`.

`groups/`	**GET**	vrátí všechny skupiny
`groups/`	**POST**	vytvoří novou skupinu
`groups/:id/`	**GET**	vrátí skupinu s `id`
`groups/:id/`	**PUT**	úprava skupiny s `id`
`groups/:id/`	**PATCH**	částečná úprava skupiny s `id`
`groups/:id/`	**DELETE**	smazání skupiny s `id`
`groups/?client=:id`	**GET**	vrátí skupiny klienta s `id`

6.4.4 Kurzy

Bod pro kurzy pracuje s klíči `id`, `name` a `visible`.

`courses/`	**GET**	vrátí všechny kurzy
`courses/`	**POST**	vytvoří nový kurz
`courses/:id/`	**GET**	vrátí kurz s `id`
`courses/:id/`	**PUT**	úprava kurzu s `id`
`courses/:id/`	**PATCH**	částečná úprava kurzu s `id`
`courses/:id/`	**DELETE**	smazání kurzu s `id` (lze smazat pouze pokud není přiřazený k žádné lekci nebo skupině)

6.4.5 Stavy účasti

Bod pro stavy účasti pracuje s klíči `id`, `name` a `visible`.

`attendancestates/`	**GET**	vrátí všechny stavy účasti
`attendancestates/`	**POST**	vytvoří nový stav účasti
`attendancestates/:id/`	**GET**	vrátí stav účasti s `id`
`attendancestates/:id/`	**PUT**	úprava stavu účasti s `id`
`attendancestates/:id/`	**PATCH**	částečná úprava stavu účasti s `id`
`attendancestates/:id/`	**DELETE**	smazání stavu účasti s `id` (lze smazat pouze pokud není přiřazený k žádné účasti)

6.4.6 Účasti

Bod pro účasti je vytvořen pro snížení datové náročnosti běžné prováděných úprav. Pracuje s klíči `id`, `paid`, `note`, `attendancestate_id` a `client_id`.

`attendances/:id/`	**PUT**	úprava účasti s `id`
`attendances/:id/`	**PATCH**	částečná úprava účasti s `id`

6.4.7 Přihlášení

Bod pro účasti pracuje s klíči `username`, `password` a `token`.

`jwt-auth/`	**POST**	na základě zaslaných údajů uživatele vrátí token
`jwt-refresh/`	**POST**	na základě zaslaného (neexpirovaného) tokenu vrátí nový obnovený token

Kapitola 7

Implementace

V této kapitole představím průběh samotné implementace. Je rozdělena do několika částí, nejprve popíši nástroje použité pro vývoj a přípravu prostředí, poté se zaměřím na vytvoření základního nastavení serverové a klientské části, které bude výchozím krokem pro další sekce, ve kterých postupně ukážu práci s datovou částí a tvorbu API. Tím připravím všechny součásti potřebné k fungování a vytvoření plnohodnotné klientské části v další sekci, uvedu zde také způsob komunikace se serverovou částí, tvorbu UI a další způsoby práce s JS a Reactem. Na závěr uvedu způsoby řešení bezpečnosti v aplikaci.

Nejproblematičtější fází se překvapivě stala konfigurace Djanga a Reactu tak, aby spolu tyto dvě části fungovaly, a také pokročilejší přizpůsobení API. Oběma problémům se budu také v této kapitole věnovat.

7.1 Nástroje pro vývoj

Celá práce byla implementována ve vývojovém prostředí Pycharm Professional Edition od Jetbrains na systému Windows 10, které nabízí nativní podporu pro všechny části tohoto projektu od samotného Pythonu (např. už ve výchozím nastavení podporuje virtuální prostředí pro izolaci jednotlivých prostředí pro různé projekty), Djanga až po React. Díky tomu lze využít všechny funkce tohoto vývojového prostředí pro všechny používané jazyky a frameworky a také lze většina úkonů provádět také pomocí grafického prostředí – integrovaný terminál lze použít jen v případě, kdy je to opravdu potřeba. Dále existuje mnoho doplňků, které funkci IDE rozšíří o podporu dalších funkcí, např. pro lepší práci s verzovacími nástroji a YAML formátem.

Pro rychlejší vývoj, organizaci a také pro rozšíření znalostí a zkušeností jsem se rozhodl využít i další služby. Pro verzování využívám soukromý repozitář na GitHubu. Dále používám nástroj pro průběžnou integraci a dodávání Travis CI. Díky jednoduchému propojení s GitHubem se tak mohou samy spouštět při každém nahrání nové verze testy včetně těch pokročilejších a ná-

ročnějších a v případě úspěchu se aplikace nahraje na produkční server pro zákazníka. K tomu všemu slouží jediný soubor v kořenovém adresáři s názvem `.travis.yml`, podrobněji příslušné části souboru popíši v kapitolách o testování 8 a nasazení 9.

7.2 Příprava prostředí

Pro vývoj je potřeba nainstalovat další potřebné balíčky a závislosti, nejdříve jsem tedy nainstaloval do systému Python, jehož součástí je balíčkovací systém pip pro instalaci knihoven, a Node.js, jehož součástí je balíčkovací systém pro JS s názvem npm (Node.js je potřeba především pro vývoj, vytváření buildů a běh vývojového serveru, npm je vítaným ulehčením práce s knihovnami a závislostmi). V současnosti se často používá také balíčkovací systém yarn, který je ale třeba doinstalovat (například přes npm), jeho výhodou jsou stejné principy jako npm, ale mnohem vyšší efektivita a rychlost [110], pro tento projekt jej tedy z těchto důvodů používám. Pomocí těchto balíčkovacích systémů jsem dále nainstaloval poslední verze frameworku Django a nástroj create-react-app pro jednoduché vytvoření React aplikace. Vzhledem k tomu, že vývoj bude probíhat v Pycharmu, není již potřeba dále řešit virtuální prostředí v Pythonu. Na závěr samozřejmě přidám a připravím v systému databázi PostgreSQL.

7.3 Základní nastavení serverové a klientské části

Začnu nejprve serverovou částí. V Djangu je potřeba nejprve vytvořit projekt a do něj poté přidávat jednotlivé komponenty, kterým se říká aplikace, projekt je tedy soubor aplikací a nastavení na jedné doméně. Pro vytvoření základní kostry projektu s výchozím nastavením jsem využil možností rozhraní Pycharmu, díky kterému stačí vyplnit základní informace o aplikaci a celou kostru připraví za mě, vytvořil jsem tedy projekt `up`. Pro vytvoření aplikací jsem využil vestavěného manage.py terminálu v Pycharmu a v něm jsem přes příkaz `startapp` vytvořil dvě aplikace: `admin` (pro samotnou aplikaci) a `api` (pro API). Ty je poté potřeba v nastavení Djanga přidat do seznamu `INSTALLED_APPS` v souboru s nastavením.

```
re_path(r'^', TemplateView.as_view(template_name="index.html"))
```

Ukázka kódu 1: Základní nastavení routování v urls.py

Do serverové části zatím nebudu v této sekci příliš zasahovat, pro další práci ještě ale upravím aplikaci `admin` tak, aby místo své výchozí stránky zobrazovala mou vlastní. Pro takto jednoduché zobrazení stačí využít již připravený generický pohled `TemplateView` a jeho metodu `as_view`, tedy do souboru `urls.py` vložím kód 1, který zařídí, že každému uživateli na jakékoliv

adrese (kromě API, to nastavím až v sekci 7.5) ukážu daný soubor (zobrazit klientovi správnou stránku v rámci webové aplikace bude zodpovědnost JS na klientské části).

Serverová část je připravena, nyní popíši základní nastavení klientské části. Nástroj create-react-app umožňuje jednoduše vytvořit základní React aplikaci včetně všech základních nastavení a kostry. Takto jsem vytvořil aplikaci „frontend“, tedy složku s tímto názvem, která obsahuje všechny potřebné součásti pro běh React aplikace. Součástí této připravené aplikace jsou mimo jiné tyto předkonfigurované součásti:

- transpiler Babel, který umožní používat bez obav JSX a další nové funkce z novějších ES standardů (viz. sekce 4.1.2),
- bundlovací nástroj Webpack, který umožňuje [111]:
  - rozdělovat kód do modulů a napříč aplikací je importovat a znovupoužívat,
  - spouštět server, který umožní rychlý vývoj díky „hot reloadingu“ (tedy okamžité automatické projevení změn při úpravě kódu),
  - automaticky spouštět transpilaci Babelem,
  - po přeložení vytvořit ze všech modulů a částí kódu jeden či více balíčků, které pak lze jako běžné JS a další soubory (např. CSS) servírovat na produkci, kde už vývojový server neběží,
  - s produkčními soubory provést obfuskace (minifikaci, uglifikaci), komprimaci, označení hashem (aby prohlížeč poznal, zda může využít soubor z cache, nebo došlo ke změnám) – ,
- testovací nástroje a další knihovny umožňující rychlejší vývoj, vyšší kompatibilitu napříč prohlížeči, rychlejší načítání ad.

Všechny tyto součásti používám, včetně nových specifikací ECMAScript, JSX apod., díky všem nástrojům není důvod k obávám z nekompatibility a můžu z jejich použití pouze profitovat. Uvedené operace se soubory, které provádí Webpack a Babel by většinou šly provést až na produkci u uživatele, ale jednalo by se o zbytečně krkolomné a pomalé řešení, proto se provádí již při nahrávání na testovací/produkční server.

Nyní tedy mám na lokálním počítači připravené Django, které mi díky svému vývojovému serveru zobrazí výchozí stránku a React, který mi díky Webpack vývojovému serveru zobrazí na odlišném portu také svou výchozí stránku. Je třeba tyto dvě části propojit. V případě lokálního prostředí musí spolupracovat tak, abych nepřišel o žádné výhody jako např. hot reloading, hash v názvu, tedy aby nebylo při každé úpravě potřeba v Djangu měnit adresu souborů kvůli odlišné hodnotě hashe). V případě produkčního prostředí je třeba přizpůsobit konfiguraci tomu, že zde už poběží pouze jeden server

(Gunicorn), na kterém bude běžet Django. Tato část se vzhledem k mé dosavadní neznalosti těchto technologií ukázala jako poměrně náročná, protože bylo potřeba upravit kódy na obou stranách (tedy jak v Djangu, tak pro React v konfiguraci Webpacku) a ještě k tomu toto udělat prakticky dvakrát, jak pro lokální vývoj, tak pak odlišně pro produkci. Nakonec se mi ale podařilo tuto problematiku vyřešit, řešení vychází z článku [112] a také staršího článku autora obou použitých knihoven [113], v následujícím odstavci jej stručně popíši.

Díky nástroji create-react-app [114] mám k dispozici jednoduchou kostru, která mi ale neumožní hlubší zásahy do další konfigurace vnitřních součástí. Je tedy potřeba pomocí příkazu `yarn eject` „vysunout“ konfigurační soubory a závislosti tak, abych k nim měl přístup a mohl je upravit. Nyní potřebuji dvě další dvě knihovny, každou pro jednu stranu – do JS přidám knihovnu webpack-bundle-tracker (dále jen tracker), která vyextrahuje potřebné informace z Webpacku do JSONu [115] (např. názvy vygenerovaných souborů) a do Djanga přidám knihovnu django-webpack-loader (dále jen loader), která tyto informace bude konzumovat a umožní použít vygenerované soubory Webpackem v Djangu [116]. Tento JSON soubor bude tedy propojovat Webpack s Djangem.

Když mám připravené tyto knihovny a kompletní kostru aplikace po vysunutí, je potřeba upravit konfiguraci Webpacku pro lokální i produkční prostředí tak, aby spolupracoval s trackerem a soubory byly tam, kde je bude očekávat loader, který bude konzumovat výstup trackeru (a aby totéž umožnil i pro lokální prostředí s hot reloadingem). Také je potřeba nakonfigurovat loader, to lze provést v souboru s nastavením Djanga, oproti původní vygenerované kostře Djangem budou ale soubory s nastavením dva, jeden obecný, jehož součástí budou také informace pro lokální prostředí a jeden produkční, který bude obsahovat import předchozího a přetíží konfiguraci potřebných částí tou svojí (to se týká nejen této části, ale později například i různých nastavení databází pro různá prostředí). V těchto nastaveních jsem nakonfiguroval loader tak, aby konzumoval správný soubor ze správného místa a také jsem nastavil adresy statických souborů tak, aby je Django zahrnulo do shromažďování souborů prováděného příkazem `collectstatic` (používá se při nasazení). Posledním krokem je vložení Djangem získaných souborů do webové stránky, aby se zobrazily uživateli, v tomto kroku tedy poprvé a naposledy v celé této práci použiji šablonovací systém Djanga (který by byl naopak hojně používaný v případě, že bych nezvolil architekturu) – do kostry výchozí stránky Djanga vložím tagy, které z loaderu umístí příslušné soubory do stránky (tedy JS a CSS). Je hotovo, po provedení příkazu `manage.py runserver` a `yarn start` Django ukazuje webovou stránku, na které je zobrazena výchozí stránka Reactu, a to jak v lokálním prostředí, tak na produkci. Výchozí zjednodušená stránka, která zaručí zobrazení celé aplikace je na ukázce 2. React aplikace se vkládá do elementu `root`.

```
{% load render_bundle from webpack_loader %}
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width,
                                   initial-scale=1,
                                   shrink-to-fit=no">
    <title>ÚPadmin</title>
</head>
<body>
<div id="root">
    <h2>Načítání...</h2>
</div>
{% render_bundle 'main' %}
</body>
</html>
```

Ukázka kódu 2: Základní stránka webové aplikace

7.4 Datová část

K již nainstalovanému PostgreSQL jsem ještě nainstaloval vývojové prostředí pro databáze od Jetbrains s názvem DataGrip, díky kterému budu moci jednoduše nahlížet jak do lokální databáze, tak do databáze na produkčním serveru a případně provádět i další operace. Jak jsem již zmínil v sekci s návrhem architektury 6.2, je potřeba použít adaptér psycopg2, ten umožní v Pythonu, a tedy i Djangu, používat databázi PostgreSQL. Do nastavení Djanga přidám tuto databázi a zbývá definovat tabulky v databázi.

Při návrhu datového modelu v sekci 6.1 jsem nastínil, že díky frameworku mám výrazně ulehčenou práci s databází (a je třeba říci, že pokud člověk doteď žádný framework s touto funkcionalitou nepoužíval, je to opravdu obrovský krok kupředu), v Djangu není potřeba mít skripty pro vytvoření databáze, jediným zdrojem pravdy jsou Django modely definující jednotlivé entity, jejich atributy a chování. Následné další úpravy databáze na základě modelů se provádí prostřednictvím takzvaných migrací. V ukázce 3 je jeden z modelů, konkrétně lekce, vybral jsem jej proto, že je na něm možné ukázat více věcí – je zde vidět pět atributů, některé jsou nepovinné (`start` a `group`), reprezentují různé typy polí (kladné číslo, cizí klíč, boolean, časová značka) a u cizích klíčů je vidět, na který model odkazují, mohou mít přidělené jméno (použije se při využití opačného vztahu v modelu skupin) a také mají definované chování při smazání příslušného záznamu, tedy kurz se podaří smazat pouze když k němu nejsou žádné lekce (ošetření, aby se omylem nepřišlo o záznamy z historie) a v případě smazání skupiny se smažou mj. všechny její lekce.

```
class Lecture(models.Model):
    start = models.DateTimeField(null=True)
    canceled = models.BooleanField()
    duration = models.PositiveIntegerField()
    course = models.ForeignKey(Course, on_delete=models.PROTECT)
    group = models.ForeignKey(Group, related_name='lectures',
                              on_delete=models.CASCADE,
                              null=True)
```

Ukázka kódu 3: Ukázka modelu lekce ze souboru models.py

Postup při používání zmíněných migrací je jednoduchý. Při vytvoření či úpravě modelů je potřeba zavolat `manage.py makemigrations` (vytvoří soubor s migracemi na základě provedených úprav, v případě nejasností nebo potřebě dalších informací mě vyzve k doplnění, tedy např. pokud měním atribut z nepovinného na povinný, vyzve mě k zadání výchozí hodnoty, která bude doplněna do stávajících záznamů bez vyplněného atributu) a poté změny aplikovat `manage.py migrate`. V praxi tedy při vývoji upravím model, vytvořím příslušným příkazem soubor s migracemi (a s tím související doplňující informace), zkontroluji správné výsledky a funkčnost a výsledek zašlu do verzovacího systému, Travis a Heroku poté provedou příslušné migrace a změny jsou úspěšně provedeny ve všech prostředích konzistentně. Také je třeba dodat, že první migrací nevzniknou pouze mnou definované tabulky, ale také pomocné tabulky Djanga potřebné pro správný chod (např. tabulka s uživateli, provedenými migracemi ad.).

7.5 API

Mám připravený základ serverové části, tedy Djanga, které servíruje stránku s React aplikací a také modely Djanga, které umožní pracovat s entitami v projektu. Je potřeba vytvořit REST API, které umožní Reactu komunikovat se serverovou částí. V této části stručně shrnu, jak tvorba API probíhala. Vzhledem k tomu, že s tvorbou API je spojeno spoustu opakování podobného kódu, rozhodl jsem se využít služby dalšího frameworku, který mi v Djangu obstará jednoduché vytvoření REST API. Frameworků existuje několik, zvolil jsem nejpopulárnější, Django REST Framework (dále DRF), který poskytuje užitečné nástroje a konstrukty k efektivní tvorbě API a zároveň obsahuje velkou škálu doplňkových knihoven od dalších členů komunity, které lze také využít [117]. Pro následné testování a úpravy API jsem používal nástroj Postman.

Obecně se tvorba API v DRF dělí na několik částí: views (pohledy), serializery a URL mapování. Nejprve je potřeba zvolit URL adresu pro API požadavky, tato adresa bude jako jediná mít zvláštní chování, ostatní adresy jsou obslouženy Reactem (viz. nastavení v sekci 7.3). Před řádek v ukázce

kódu 1 vložím další řádek uvedený v ukázce 4 (pokud bych jej vložil za, tak vzhledem k působnosti dříve vloženého kódu by převzal zodpovědnost za API React, což nechci). Nyní už stačí definovat konečné URL body API, kde určím, která adresa využije který pohled, k tomu slouží v DRF router, který umožní v souboru `api/urls.py` jednoduchým způsobem definovat spojení URL adres s pohledy, v ukázce kódu 5 uvádím část pro lepší pochopení – vybral jsem dva pohledy, které v dalším odstavci podrobněji ve zkratce představím a ukážu na nich další kroky při tvorbě API.

```
path('api/v1/', include('api.urls')),
```

Ukázka kódu 4: Nastavení routování pro API v souboru urls.py

```
router = routers.DefaultRouter()
router.register('courses', views.CourseViewSet)
router.register('groups', views.GroupViewSet)
```

Ukázka kódu 5: Ukázka routeru pro API v souboru api/urls.py

DRF nabízí několik možností, jak vyřešit pohledy. Zvolil jsem tu, která by měla umožnit velmi jednoduše definovat celé API a obstarat automaticky pohled jak na detail (instanci entity), tak na kolekci (všechny instance entity) včetně všech operací, nazývá se ViewSet a API pohledy úzce mapuje na Django modely. Pohled může být definován poměrně snadno, jak je vidět v ukázce kódu 6 ze souboru `api/views.py` – vytvořil jsem pohled pro kurzy, definoval jsem `queryset`, tedy dotaz, který získá obsah pro odpověď na požadavek na API (v tomto případě i seřazený podle jména, to je definováno v modelu) a `serializer_class`, tedy třídu použitou pro serializaci, validaci a deserializaci dat. Serializeru se budu věnovat v dalším odstavci. V druhé ukázce kódu pohledu 7 je vidět, že přetěžuji metodu `get_queryset`, to mi umožní vrátit v API odpovědi vyfiltrované výsledky, v tomto případě vrátím buď všechny skupiny (seřazené podle jména, to je definováno v modelu), nebo v případě zadání `id` klienta všechny skupiny (opět seřazené podle jména), ve kterých je členem. Pro pokročilejší filtrování v API a také umožnění řazení přes parametry v adrese je pak u některých dalších pohledů využita knihovna django-filter, díky které stačí zadat příslušné atributy a vše zařídí za mě.

```
class CourseViewSet(viewsets.ModelViewSet):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
```

Ukázka kódu 6: Jednoduchý pohled pro API v souboru api/views.py

```
class GroupViewSet(viewsets.ModelViewSet):
    serializer_class = GroupSerializer
    def get_queryset(self):
        qs = Group.objects.all() # qs znaci queryset
        id_client = self.request.query_params.get('client')
        if id_client is not None:
            qs = qs.filter(memberships__client=id_client)
        return qs
```

Ukázka kódu 7: Pokročilejší pohled pro API v souboru api/views.py

Poslední chybějící součástkou, kterou jsem ještě neukázal, jsou serializery. V ukázkách kódu 6 a 7 je vidět jejich používání, o kterém jsem mluvil v předchozím odstavci. Vzhledem k tomu, že jejich kód je delší, na ukázce 8 je vidět nejjednodušší serializer pro kurzy. K implementaci jsem použil `ModelSerializer`, díky kterému je serializer úzce navázán na model a není potřeba opakovat zbytečně kód. Nevýhodou DRF je, že neposkytuje jednoduchou možnost práce s vnořenými zdroji, se kterými jsem v rámci návrhu počítal – cílem bylo např. získat lekci spolu s údaji klienta, ale při úpravě lekce už údaje klienta nevyžadovat, pouze jeho `id`. Dokumentace tohoto frameworku je sice poměrně rozsáhlá, ale často je také poměrně krkolomná a spoustu potřebných údajů je těžké nebo i nemožné dohledat. Nakonec jsem přišel na způsob, jak by se tento problém měl řešit, v příslušných serializerech bylo potřeba u dotyčných atributů vždy vytvořit dvojici atributů, kdy ten s údaji klienta bude pouze pro čtení a název bude totožný s modelem a druhý atribut bude mít odlišný název (zvolil jsem vždy přidání `_id` k původnímu názvu) a bude určen pouze pro zápis (navíc je se strany DRF pro korektní fungování vyžadován argument `queryset`, také je použit argument `source`, aby se daný atribut choval jako atribut původní), lze vidět na ukázce 9.

```
class CourseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = '__all__'
```

Ukázka kódu 8: Pokročilejší pohled pro API v souboru api/views.py

```
client = ClientSerializer(read_only=True)
client_id = serializers.PrimaryKeyRelatedField(
        queryset=Client.objects.all(), source='client',
        write_only=True)
```

Ukázka kódu 9: Práce se vnořenými zdroji v serializeru

Druhým souvisejícím problémem, který jsem musel vyřešit, byla práce se vnořenými zdroji při požadavcích POST, PUT a PATCH, například při vytvoření lekce je potřeba vytvořit také pro každého člena účast apod. Nakonec i tento problém se mi podařilo vyřešit, konkrétně přetížením metod `create` a `update` v příslušných serializerech (ukázku neuvádím kvůli delším kódům). Vytvoření funkčního API především kvůli těmto dvěma zmíněným problémům trvalo mnohem déle, než jsem očekával.

Součástí některých odpovědí API, například při GET požadavku na lekce, jsou další informace určené pouze pro čtení a vypočítané na základě dat v databázi, jsou jimi například informace o číslu lekce v pořadí (v rámci jednoho klienta a daného kurzu) či informace, zda má příště platit. Bylo tedy potřeba je důkladně promyslet a otestovat, příkladem budiž číslo lekce – abych zjistil počet uběhlých lekcí, a tedy s přičtením jedné pořadové číslo aktuální lekce, je potřeba vzít v úvahu, zda se jedná o naplánovanou lekci bez datumu (ta nemá číslo), o skupinovou lekci (zde se zjišťuje číslo jednodušeji, stačí vzít lekce skupiny, vyselektovat pouze ty, které jsou dříve než datum aktuální lekce, nejsou nenaplánované a ani zrušené) a nebo o individuální lekci (zde vezmu lekce klienta náležící k danému kurzu, pouze individuální, s určeným datumem, který je dříve než datum u aktuální lekce, nezrušené a se stavem účasti odpovídajícím tomu, že se dostavil).

7.6 Klientská část

Pro klientskou část je, jak jsem již zmínil při volbě architektury v sekci 4.5, zvolena knihovna React, která mi umožní vytvořit interaktivní aplikaci a jednoduše ji dále rozšiřovat a rozvíjet díky architektuře CBA (viz. sekce 3.2). V této části stručně popíši, jak jsem při tvorbě klientské části postupoval.

7.6.1 Vzhled

Součástí tvorby klientské části bylo rozhodnutí, zda zvolit pro řešení UI nějakou předpřipravenou šablonu nebo framework. Co se týče šablon, nalezl jsem několik takových, které byly postaveny na Reactu, jako např. react-director-admin-template, ReactJS-AdminLTE, admin-on-rest (který umožňuje napojit administraci přímo na REST API), Ant Design Pro, CoreUI-React, k jejich použití jsem ale nepřistoupil, protože byly buď zbytečně megalomanské a třeba ztížily orientaci v rámci aplikace, nebo měly špatnou (či čínskou) dokumentaci, omezené funkce zdarma apod. Také by se mohly hůře přizpůsobovat pozdějším požadavkům na změny UI.

Rozhodl jsem se využít služby frameworku Bootstrap, který v lednu 2018 přišel s dlouho očekávanou přepracovanou 4. verzí. Díky němu se mohu při vývoji mnohem hlouběji zaměřit na funkcionalitu, protože se postará o spoustu věcí za mě, dalším důvodem použití je také jeho popularita a rozšířenost [118] mezi vývojáři. S Bootstrapem jsem nikdy nepracoval (volil jsem vždy svůj

vlastní kód) a zaujaly mě novinky v poslední verzi, které by vývoj této aplikace usnadnily. Abych mohl Bootstrap pohodlně v Reactu používat, použil jsem nástroj reactstrap, díky kterému mohu Bootstrap komponenty vkládat jako bezstavové komponenty Reactu.

Pro pokročilejší úpravy jsem také použil knihovny react-toastify pro oznámení, react-select pro usnadnění tvorby a práce s poli ve formuláři, kde je potřeba provést násobný výběr (např. členové skupiny) tak, aby byl intuitivní. Napříč celou aplikací také používám balíček a nástroj pro ikony Fontawesome – použiji zde poprvé nejnovější verzi 5, která je zcela přepracovaná a rozšířená oproti předchozím verzím, se kterými mám výborné zkušenosti (balíček bude v mnou zakoupené placené verzi se stylem ikon „solid“) a opět pro jednodušší použití jako komponenty Reactu použiji oficiální knihovnu react-fontawesome.

7.6.2 Základní práce s Reactem

Prošel jsem všechny potřebné závislosti, nástroje, knihovny a konfigurace a nyní tak mohu ukázat, jak jsem postupoval při samotné tvorbě aplikace v Reactu. Vývoj probíhal v nejnovější verzi 16.2, předem bych ale rád podotkl, že na závěr vývoje došlo k vydání verze 16.3, která přinesla několik novinek, zejména v oblasti životního cyklu komponenty (změna API) a Context API, které usnadňuje předávání dat mezi komponentami napříč jejích stromem [119]. U Contextu bych se rád zastavil, teprve se začíná používat (byl představen na konci března 2018), ale už teď lze říci, že doplňuje chybějící článek Reactu, kvůli kterému mnoho vývojářů používalo Redux. Redux se používá se pro správu stavu aplikace, jak ale říká jeho autor a dnes také přední vývojář Reactu v [120], často se používá zbytečně a komplikuje tak vývoj aplikace, který by byl jinak mnohem rychlejší a kratší. Rozhodl jsem se pro tuto práci respektovat jeho doporučení, tedy nejdříve vytvořit čistou aplikaci v Reactu, pochopit všechny aspekty takové tvorby a fungování a pak teprve uvažovat o tom, zda je potřeba Redux využít (případně použít nové Context API). Aplikace nyní běží na verzi 16.3, zatím ale nevyužívá žádné z jejích novinek, vzhledem k úpravě API životního cyklu komponenty je ale potřeba počítat s drobnějšími změnami, které budu muset provést až přijde verze 17 [121].

Ještě než se pustím do popisu práce s Reactem, rád bych uvedl jednu z knih – *React Design Patterns and Best Practices* [122], ze které jsem při seznamování se s Reactem vycházel. Kromě podrobných popisů fungování jednotlivých součástí Reactu obsahuje mnoho rad a tipů, jak psát efektivní, rozšiřitelný, znovupoužitelný a čistý kód v Reactu. Vzhledem k její rozsáhlosti a pokročilosti s ní plánuji pracovat i při dalším vývoji této aplikace po odevzdání práce.

V Reactu je vše rozdělené do komponent, které obsahují celý kód a logiku obstarávající její korektní vykreslení a práci. Každá takováto komponenta může mít v sobě dva typy dat [123]:

- „props“ atributy, které může získat od rodiče, komponenta je nemůže měnit (patří do správy nadřazené komponenty), jsou tedy podobné parametrům funkcí,
- „state“, tedy stav, který se nedědí a je spravován uvnitř komponenty, je tedy podobný proměnným ve funkcích.

Díky těmto atributům React zařídí překreslení pouze té části stránky, kde nastala změna. Velmi jednoduchá bezstavová komponenta využívající nejnovější prvky syntaxe JS používaná napříč aplikací k informování, zda má klient příště platit, je v ukázce 10. V aplikaci dále používám i pokročilejší stavové komponenty, pro lepší pochopení a popis vkládám ukázku 11 (z úsporných důvodů nehezky zarovnaná), která obsahuje i komentáře, kde se nachází které části kódu a jejich účel, zdůrazním zde metodu `componentWillReceiveProps`, která je zde proto, že v rodiči se provádí asynchronní požadavek na API a některý z jeho výsledků se předává do této komponenty, tedy do potomka, a zde se na jeho základě komponenta vykresluje, vzhledem k tomu, že jsou „props“ samy o sobě neměnné, je třeba nový stav z potomka zpracovat v této metodě a změny zde projevit do vlastního stavu komponenty.

```
import React from 'react'
import {Badge} from 'reactstrap'
const RemindPay = ({remind_pay}) =>
    (remind_pay &&
        <Badge color="warning" pill>Příště platit</Badge>)
export default RemindPay
```

Ukázka kódu 10: Jednoduchá bezstavová komponenta Reactu

```
import React, {Component} from "react"
// import dalších JS komponent a CSS souborů
export default class NazevKomponenty extends Component {
    constructor(props) {
    /* nastavení stavu a dalších proměnných/props*/}
    dalsiFunkce = () => {/* tělo funkce */}
    componentDidMount() {/* požadavky na API */}
    componentWillReceiveProps(nextProps) {
    /* aktualizace stavu při změně stavu rodiče */}
    render() {
    // příprava dalších komponent a proměnných
        return (<div>{/* vykreslení v JSX */}</div>)}
}
```

Ukázka kódu 11: Kostra pokročilejší komponenty v Reactu

7.6.3 Routování a HTTP požadavky v Reactu

Nyní chybí dořešit poslední dvě části v Reactu: routování a komunikaci přes API, ani jedno totiž React v základu nemá, jak již bylo řečeno v kapitole 4.1.4. Pro routování jsem zvolil knihovnu React Router, která mi umožní v Reactu snadno implementovat SPA a zajistí také fungování tak, jak by uživatel očekával, tedy při přechodech se mění URL adresa v prohlížeči, korektně funguje navigace napříč historií apod. K tomu využívám `BrowserRouter`, který se nově objevil v přepracované poslední čtvrté verzi této knihovny, oproti `HashRouter` předpokládá funkční routování i na straně serveru a využívá HTML5 API pro historii (HashRouter nepoužívá běžnou čistou URL, ale pracuje s fragmentem URL za přidaným znakem `#`), to je díky kódu 1 zařízeno, nic tedy nebrání použít naplno tuto novou komponentu.

```
getClients = () => {
    ClientService
        .getAll()
        .then((response) => {
            this.setState({clients: response, loading: false})
        })}
```

Ukázka kódu 12: Kostra pokročilejší komponenty v Reactu

Pro komunikaci Reactu s vystaveným REST API jsem použil velmi populární a častou volbu vývojářů nejen pro React, knihovnu Axios, ačkoliv se už pomalu začíná rozšiřovat standardizované JS Fetch API, Axios nabízí širší a jednoduší možnosti práce s asynchronními požadavky, např. odchytávání pomocí „interceptorů“, zabudovanou CSRF ochranu (viz. sekce 7.7) a také zaručuje kompatibilitu ve všech prohlížečích [124]. Protože bylo potřeba mít k dispozici jednotnou správu nad všemi požadavky včetně odchytávání chyb a konfigurace, vytvořil jsem jednotné rozhraní pro požadavky, které jsem doplnil o vytvoření služeb pro každou využívanou část API. V samotné stránce, kde probíhá požadavek na API tedy nejsou žádné konfigurační informace (ani URL adresa) a vše lze tak snadno upravit nebo dokonce nahradit knihovnu pro požadavky. Na ukázce 12 přikládám výsledný kód, který využívá importovanou službu, která poté zavolá jednotné rozhraní pro požadavky, kde teprve proběhne vytvoření a zaslání požadavku prostřednictvím Axiosu na API a v případě úspěchu se výsledek uloží do stavu komponenty a komponenta se tak překreslí. V této ukázce je také vidět přenastavení atributu `loading`, díky tomu může být v příslušné komponentě zobrazena načítací animace, dokud komponenta neobdrží příslušná data, tedy například při načítání lekcí v týdenním přehledu jsou okamžitě zobrazeny všechny pracovní dny i s jejich boxy, díky rozdělení na komponenty má každý box své načítání a tato animace zmizí vždy právě tehdy, když se načtou data do příslušného boxu. Uživatel mezitím může provádět další operace a vidí, kdy je požadavek zakončený.

7.6.4 Další práce s JS

Ačkoliv Babel spolu s dalšími polyfilly umožní vytvářet kód s co nejvyšší možností kompatibility napříč různými prohlížeči, bylo třeba i tak dbát na ověření této kompatibility. V rámci vývoje jsem například zjistil, že prohlížeče Safari a Internet Explorer jinak pracují s datumy dle standardu ISO, což vyústilo v několik chyb v aplikaci (nezobrazení datumů ad.), musel jsem tedy upravit práci s datumy tak, aby byl kód kompatibilní i v těchto prohlížečích. Další s ISO standardem související problém byl, že v JS se pracuje s datumy v tomto formátu pouze s časovou zónou UTC, ačkoliv jsou třeba v jiné. To vyústilo v situaci, že vždy mezi půlnocí a jednou hodinou ranní byly v týdenním přehledu zobrazeny špatné dny (tedy při letním čase dokonce od půlnoci do dvou hodin ráno), byl jsem tedy nucen pro tuto práci s datumy vytvořit vlastní funkce, které respektovaly i další časové zóny.

7.7 Bezpečnost

Vzhledem k tomu, že se v aplikaci nacházejí důvěrné informace, je potřeba zvolit adekvátní úroveň zabezpečení. V této sekci stručně shrnu kroky, které jsem učinil k zabezpečení aplikace, zejména zabezpečení komunikace a přihlašování.

7.7.1 Ochrana proti útokům

Je potřeba využívat protokol HTTPS, díky kterému bude komunikace šifrovaná – Heroku HTTPS nabízí pro aplikace bez vlastní domény zdarma SSL certifikát, bylo tedy nutné jen zajistit přesměrování veškeré komunikace na zabezpečenou. To zajistí konfigurace Djanga, konkrétně proměnná `SECURE_SSL_REDIRECT`.

Ve webových aplikacích se často objevují některé bezpečnostní chyby, podle [125] a [126] zejména:

- **SQL Injection:** narušení SQL dotazu z důvodu špatného ošetření parametrů při tvorbě SQL dotazů,
- **XSS (Cross Site Scripting):** vložení škodlivého kódu kvůli nedostatečnému ošetření vstupů od uživatele,
- **CSRF (Cross Site Request Forgery):** tajné vykonání požadavku z důvodu neošetření původu požadavku,
- **Clickjacking:** součástí podvodné stránky je jiná stránka a uživatelem provedená akce vyústí v nezamýšlené akce na jiné stránce.

Využil jsem možnosti Djanga v oblasti bezpečnosti [125] a nakonfiguroval jej tak, aby rizika hrozeb eliminovalo na minimum, díky ORM není problém

s SQL Injection a díky dalším konfiguracím ani s ostatními (a nejen těmi) problémy. Správné nastavení bylo mj. ověřeno i nástrojem `check` v Djangu. Některé problémy vyžadují při komunikování zaslání dodatečných informací sloužících k rozpoznání potenciálních útoků, z toho důvodu bylo potřeba adekvátně nastavit i klienta Axios pro HTTP požadavky z klientské části. V neposlední řadě se o většinu XSS problémů stará i React (funkce, které jsou bez ochrany se v této aplikaci nepoužívají) [127].

7.7.2 Přihlašování

Pro přihlašování jsem zvolil metodu JSON Web Token (JWT), která je standardizovaná v RFC 7519. Mezi serverem a klientem se podle [128] posílá malý JSON objekt (token), který může být ověřen a je důvěryhodný, protože je podepsaný. Díky tomu není potřeba zatěžovat databázi opakujícími se dotazy, protože součástí objektu jsou všechny potřebné informace (hlavička s nastavením, tělo s informacemi o uživateli a expirací a podpis).

DRF obsahuje připravených několik možností pro přihlášení [129], místo JWT ale obsahuje vlastní přihlašování přes tokeny, které oproti JWT používá pro každou validaci tokenu při každém dotazu databázi, zvolil jsem tedy doporučenou knihovnu Django REST framework JWT, která zjednodušuje celou implementaci JWT metody. Vystaví se zde dva koncové body API pro autentikaci a obnovení tokenu, jak je v ukázce 13. Samotná autentikace funguje tak, že se uživatel přihlásí, obdrží JSON Web Token, kterým se pak při další komunikaci prokazuje. Jelikož má token nastavenou určitou dobu expirace, pokud se uživatel v aplikaci pohybuje mezi jednotlivými stránkami a blíží se doba vypršení, pošle se požadavek na API o obnovení tokenu, zašle se původní a pokud je vše v pořádku, server vrátí token s prodlouženou expirací, doba, po jakou lze prodlužovat je opět omezená. Aby se mohla expirace tokenu ověřovat na straně klienta a v případě nutnosti token zaslat před vypršením k obnově, je potřeba převést token v JS do čitelného formátu, k tomu je použita jednoduchá knihovna jwt-decode.

```
path('jwt-auth/', obtain_jwt_token),
path('jwt-refresh/', refresh_jwt_token),
```

Ukázka kódu 13: API pro přihlašování

Kapitola 8

Testování

Testování aplikace bylo rozděleno na několik částí. Jak již bylo řečeno v části o nástrojích pro vývoj 7.1, bylo vytvořena sada základních testů, které se automaticky během celého vývoje spouští na integračním serveru Travis CI, o tom více povím v první části. V další části popíši výstupy a provedené úpravy na základě vlastního testování, které jsem provedl na závěr vývoje, abych ověřil splnění požadavků a dobré fungování aplikace. Vlastní testování probíhalo samozřejmě v průběhu celého projektu, z pohledu této práce jsou ale nejzajímavější výsledky závěrečného testování. Stejně tak v průběhu samotného vývoje jsme s lektorkou procházeli dodané a upravené části, aby bylo zajištěno, že se vývoj ubírá správným směrem. V poslední části popíši a uvedu výsledky akceptačního testování.

8.1 Automatizované testování

Součástí vývoje bylo automatizované testování, které se může spouštět jak na lokálním stroji, tak na integračním serveru. V této části popíši, co vše má zvolený nástroj na integraci Travis CI za úkol a jaké základní testy byly vytvořeny.

Pro konfiguraci prostředí integračního serveru používá konfigurační soubor `.travis.yml`, v ukázce 14 je část souboru (některé příkazy jsou zkráceny třemi tečkami nebo rovnou smazány, ponecháno je jen to nejdůležitější). Práce na Travisu je rozdělena do několika fází, tyto fáze dále nabízí ještě akce, které se provedou před a po. Nejdříve se zvolí jazyk projektu a další technologie, v tomto případě Python, Node.js a PostgreSQL, zaktualizuje se Node.js a balíčkovací systém `npm`, nainstaluje se balíčkovací systém yarn, nastaví se do globální proměnné, které nastavení pro Django se využije, nainstalují se veškeré závislosti jak pro Python, tak pro Javascript (součástí toho se po instalaci automaticky vytvoří ze všech závislostí jeden sestavený JS a CSS soubor), vytvoří se databáze pro testování, díky `migrate` se naplní příslušnými tabul-

kami v souladu s Django modely, `collectstatic` zkopíruje všechny statické soubory do jedné složky, která se použije na produkci a na závěr se spustí všechny připravené testy (spolu s počítáním pokrytí kódu). O některých dalších částech konfigurace týkajících se průběžného nasazování budu hovořit v následující kapitole o nasazení 9, pro účely jak testování, tak i nasazení, je samozřejmě potřeba několik dalších souborů v kořenovém adresáři:

- **requirements.txt** – závislosti na balíčcích Pythonu včetně Djanga,
- **package.json** – závislosti na balíčcích JS včetně Reactu a další konfigurace,

```
language: python
python: '3.6.5'
node_js: '8'
services: postgresql
before_install:
  - nvm install 8.11.1
  - npm i -g npm
  - npm install -g yarn
  - export DJANGO_SETTINGS_MODULE=up.production_settings
install:
  - pip install -r requirements.txt
  - pip install codecov
  - yarn install
before_script: psql -c 'create database ci_test;' -U postgres
script:
  - python manage.py migrate
  - python manage.py collectstatic --noinput
  - coverage run --source=admin,api --omit=... manage.py test
```

Ukázka kódu 14: Část konfigurace Travis CI v souboru .travis.yml

Pro testování bylo vytvořeno několik základních testů, jejich rozšíření a pokrytí co největší části systému je plánováno v blízké budoucnosti, protože vývoj této aplikace bude pokračovat i po této práci. Využívají nástroje pro testování přímo od frameworků Django a Django REST. Testy ověřují především:

- správné vytvoření databáze,
- funkčnost přidávání do databáze přes Django modely,
- správnou funkčnost Django pohledů, zda je uživateli při příchodu na web ukázána správná stránka a její obsah je správný,

- funkčnost API požadavků včetně autorizace, vytvoření uživatele pro administraci, získání tokenu, vytvoření klienta přes API – je tedy vytvořen účet pro uživatele aplikace, pro ten se vyzkouší získání jeho tokenu prostřednictvím API a následně se provede přes API autorizovaný pokus (s tokenem) o přidání klienta a zkontroluje se, zda byl přidán.

8.2 Vlastní testování

Pro ověření toho, že aplikace skutečně splňuje všechny požadavky a funguje korektně jsem ji na závěr vývoje sám otestoval. Jednalo se jak o testy funkční (tedy zda aplikace správně plní vše, k čemu je určena), tak nefunkční (např. responzivita, kompatibilita v prohlížečích), součástí této kontroly byla i kontrola všech kódů, a to jak pro odhalení možných chyb, tak pro ověření, že je aplikace korektně navržená a bude tak v budoucnu díky tomu snadněji rozšířitelná a udržovatelná (tedy např. konstanty, neopakující se kód, potenciální špatné ošetření apod.).

- telefonní číslo lze zadat v neexistujícím formátu (např. osmičíselně) – vyřešeno nastavením minimální délky na 9,

- ve formuláři pro přidání lekce není automaticky předvybrán stav účasti "OK"– opraveno,

- ve formuláři pro přidání lekce není při odeslání vyžadován datum a čas, ačkoliv to tak server vyžaduje – opraveno doplněním atributu `required` pro obě pole, doplněno také kontrolování validity datumu (nastavení minimálního a maximálního možného zadaného roku),

- server akceptuje pouze neprázdné názvy lekcí a kurzů, dotyčné formuláře v aplikaci ale při odeslání toto explicitně nekontrolují – opraveno doplněním atributu `required` pro obě pole,

- pokud skupina nebo klient nemají žádné lekce, je karta s lekcemi prázdná a uživatel není nijak informován o tom, že žádné lekce nejsou – vyřešeno zobrazením "žádné lekce", pokud klient nebo skupina ještě nemají žádné lekce,

- při analýze dotazů na API přes nástroje pro vývojáře v prohlížeči (část pro síťové požadavky) bylo zjištěno, že při zobrazení týdenního přehledu se zbytečně odesílá pětkrát (tedy tolikrát, kolik je zobrazeno dnů) požadavek na zjištění možných stavů účasti, tyto stavy jsou pro všechny dny stejné a stačí je tedy načíst jednou – upraveno, stavy se načtou pouze jednou a poté se předají příslušným komponentám zobrazujícím jednotlivé dny,

- v týdenním přehledu není zobrazený aktuální den, uživatel se tak zbytečně musí zdržovat s jeho hledáním – vyřešeno barevným odlišením aktuálního dne,
- pokud má klient jak skupinové lekce nějakého kurzu, tak i individuální lekce téhož kurzu a zároveň je ve skupině první při seřazení podle abecedy dle příjmení, číslo lekce kurzu je oproti očekávání vyšší (dojde k započítání individuálních i skupinových lekcí) – v API opraven špatný dotaz na databázi, nyní se už filtruje dle individuálních a skupinových lekcí,
- upozornění, že má klient příště platit, je někdy zobrazeno i v případě, že má příští lekci už zaplacenou – vyřešeno úpravou dotazu na databázi (při výpočtu opět nebyl brán ohled na to, zda lekce daného kurzu je skupinová nebo individuální),
- v dotazu na číslo lekce se pro skupinové lekce zbytečně prochází větší část databáze, než je potřeba – vyřešeno zjednodušením dotazu, který nyní pracuje pouze s tabulkou `Lecture`, která obsahuje méně dat než původní tabulka `Attendance`,
- na systému iOS se pole pro zadání datumu a času zobrazovala špatně (měla nízkou výšku) – opraveno.

8.3 Akceptační testování

Po provedení úprav na základě vlastního testování v předchozí části jsem přistoupil k akceptačnímu testování. Aplikace byla díky Travisu již připravená ve své nejnovější verzi na produkčním serveru na Heroku. Pro akceptační testování se obvykle vytvářejí scénáře, které nejprve testuje programátor a poté jsou testovány zákazníkem. Vzhledem k tomu, že bylo potřeba, aby lektorka tak či tak naplnila aplikaci daty ze všech svých zdrojů, kde jednotlivé informace eviduje (tedy z Excelu, diáře, poznámek na papírech apod.), rozhodl jsem se jako scénář použít prakticky toto zadávání stávajících údajů, které vzhledem k počtu údajů pokryje všechny možnosti úkonů.

Akceptační testování bylo prakticky spojeno i s testováním použitelnosti (tedy pozorování uživatele a nalezení nedostatků, které jsem mohl přehlédnout v důsledku toho, že jsem aplikaci sám vytvářel). Testování bylo úspěšně provedeno a na základě pozorování práce lektorky v aplikaci a také jejích postřehů bylo nalezeno několik problémů a možných vylepšení, vybrané nejdůležitější uvádím v seznamu spolu se způsobem jejich řešení:

- ve formuláři pro úpravu klienta je pole moc malé, je potřeba umožnit psát více řádků – vyřešeno zobrazením `textarea` místo stávajícího `input` pole,

- v kartě klienta chybí zobrazení údajů klienta včetně členství ve skupinách – doplněno (stačilo využít již připravené API),
- v kartě skupiny chybí zobrazení aktuálních členů – doplněno (opět stačilo využít již připravené API),
- při přidávání položek není automaticky předvybráno ke psaní první pole ve formuláři a lektorka si často nevšimne, že nikam napíše – nastaveno automatické vybrání prvního pole pro všechny formuláře, kde je to potřeba,
- při přidávání kurzu a stavu účasti je u pole pro název napsáno jméno, to je pro lektorku matoucí (i kvůli tomu, že v přehledu stavů účasti i kurzů je v záhlaví tabulky nekonzistentně uveden „název“) a svádí to ke psaní jména klienta – změněno na název,
- v kartě klienta není poznat, že není ve skupině nebo že není zadán telefon (u skupin dokonce není zobrazeni v řádku ani „Členství ve skupinách“, pokud v žádných skupinách klient není) – upraveno, u skupin i ostatních údajů se zobrazí tři pomlčky,
- skupiny většinou mají délku lekce 45, bylo by tedy vhodné, aby místo výchozích 30 (které zůstanou u jednotlivců) bylo automaticky přednastaveno 45 – doplněno,
- při označení stávající lekce jako nepředplacené se nezaplacené termíny označily jako zaplacené – opraveno,
- skupinová lekce se špatně ruší, protože se musí se u každého člena zadat zrušeno – doplněna nová funkcionalita pro jednoduché rušení lekcí (jak individuálních, tak skupinových).

Kapitola 9

Nasazení

V této části popíši, jak probíhá nasazení aplikace na Heroku, a které kroky k tomu bylo potřeba udělat. Naváži tak na předchozí kapitolu o testování 8, kde je popsán proces testování na integračním serveru, protože Travis CI kromě samotné průběžné integrace v této práci obstará i průběžné dodávání. Na závěr kapitoly vložím několik ukázek aktuální podoby nasazené aplikace.

Způsoby nasazení čerpají z dokumentací jednotlivých knihoven, jež jsou příslušně ocitovány v daných větách, dále vycházím z dokumentace od Heroku pro webové aplikace Python [130] a Django tutoriálu v dokumentaci Mozilla Developer Center [131].

Jak již bylo zmíněno při volbě technologií v kapitole 4.5, pro nasazení bylo zvoleno Heroku, o kterém jsem více psal v kapitole 4.4.1. Heroku poskytuje připravené řešení infrastruktury přímo pro tuto aplikaci a je možné jej používat zdarma, případně za menší poplatek, pokud bude potřeba se zbavit některých omezení. Aplikace na Heroku běží v tzv. dyno kontejnerech, což jsou izolované virtuální linuxové kontejnery, které pro aplikaci poskytnou potřebné prostředí pro běh. Aplikaci, kontejner a doplňky (jako např. databázi PostgreSQL) lze ovládat buď přes základní webové rozhraní a nebo přes Heroku terminál

9.1 Základní nastavení

Jak již bylo naznačeno v ukázce konfiguračního souboru pro Travis 14, pro testování na integračním serveru a pro produkci je vytvořené zvláštní soubor s nastavením Djanga oproti tomu, které se používá při lokálním vývoji a testování. Díky tomu lze pro produkci nastavit vyšší zabezpečení, větší výkon, jinou databázi ad. Autoři Djanga se práci snažili maximálně zjednodušit, a tak mám k dispozici mnoho nástrojů od `manage.py` až po seznam doporučení při nasazení [132] spolu s jednoduchým příkazem `manage.py check --deploy`, který projde projekt a upozorní mě na možná vylepšení (o použití jsem psal v sekci o bezpečnosti 7.7.1). Některé proměnné se z důvodu bezpečnosti do-

poručuje dodávat z proměnných v našem prostředí Heroku, jako např. adresu databáze nebo tajný klíč pro Django. Co se týče databáze, k jednoduchému rozparsování informací o databázi z této proměnné do nastavení Djanga využívám doporučený nástroj dj-database-url.

9.2 Produkční server a statické soubory

Vzhledem k tomu, že vývojový server, který Django poskytuje, není určený pro produkční užití [133], je obecně doporučeno využít server Gunicorn, který jsem již popsal v kapitole 6.2. Pro produkci je také nevhodné, aby byly statické soubory jako JS a CSS servírovány prostřednictvím Djanga, jako při vývoji, a tak jsem zvolil efektivnější řešení (opět doporučované přímo od Heroku), tím je knihovna WhiteNoise [134], která umožní efektivně servírovat zkomprimované soubory na produkci přímo z Gunicornu, k tomu stačí stáhnout knihovnu, přidat ji do nastavení Djanga do `MIDDLEWARE` a také zde nastavit kompresi souborů, která je volitelná, naimportovat WhiteNoise do WSGI konfigurace v již existujícím souboru `wsgi.py` a zaobalit v něm již existující aplikaci do instance WhiteNoise (ukázka 15).

9.3 Nastavení Heroku a Travisu

Pro využití Heroku je třeba se zaregistrovat, poté vytvořit aplikaci, v tomto případě s názvem „uspesnyprvnacek“, při tvorbě je možnost zvolit servery v Evropě, díky kterým mám jistotu rychlejší odezvy. Poté jsem ve webovém rozhraní přidal databázi PostgreSQL jako doplněk k aplikaci. Databázi Heroku jsem také připojil do DataGripu, díky tomu pak bylo možné přistupovat k datům ve vzdálené databázi ze stejného místa jako k lokálním, pro správnou funkčnost je ale potřeba přidat do URL databáze parametr `sslmode=require`, protože Heroku umožňuje externím komunikacím přistupovat k databázi jen s aktivním SSL. Pro další konfiguraci nasazení na Heroku je potřeba dodat do kořenové složky několik souborů:

- **runtime.txt** – programovací jazyk projektu a verze,
- **pakage.json** – závislosti pro JS a také skript, který se pustí po instalaci,
- **Procfile** – seznam procesů, které se mají provést pro start webové aplikace, je vidět v ukázce kódu 17, kde je nejdříve spuštěn skript, který provede příkazy `collectstatic` (shromáždění statických souborů, které sice Heroku může provádět automaticky, v případě této aplikace jsem ale tuto možnost deaktivoval, protože je potřeba tomuto příkazu předat správný soubor s produkčním nastavením) a `migrate` (tedy migraci databáze na její připadnou novou verzi) a v kontejneru startuje webový server.

```
from whitenoise.django import DjangoWhiteNoise
...
application = DjangoWhiteNoise(application)
```

Ukázka kódu 15: Přidaný kód do souboru wsgi.py

```
deploy:
  provider: heroku
  api_key:
    secure: ...
  app: uspesnyprvnacek
```

Ukázka kódu 16: Konfigurace Travis CI v souboru .travis.yml

```
release: bash release-tasks.sh
web: gunicorn up.wsgi --log-file -
```

Ukázka kódu 17: Soubor Procfile

Díky možnosti Heroku propojit s Travisem [135] tedy mám možnost zprovoznit i průběžné nasazení v případě, že na integračním serveru nenastane žádný problém a vše se s úspěchem otestuje a vytvoří se úspěšně nový build. Po splnění všech předešlých kroků a konfiguraci Travisu, jejíž část je vidět v ukázce 16 (pro vytvoření této části bylo potřeba nainstalovat Travis terminál a využít zde příkazy pro zašifrování Heroku tokenu), se tedy začne vytvářet build na Heroku a pokud vše proběhne bez problémů, provedou se příkazy podle souboru `Procfile` a aplikace je nasazena. Celý tento proces mám díky konfiguraci Travisu možnost sledovat v jeho logu. Pokud se jedná o první nasazení a v databázi ještě nejsou data, je třeba vytvořit účet pro uživatele, který bude použit pro přihlášení do aplikace, k tomu využijeme, stejně jako v případě lokálního vývoje, příkaz `manage.py createsuperuser`, který zadám do Heroku terminálu, ještě jej ale doplním o název aplikace, interpretr Pythonu a produkční nastavení.

9.4 Ukázka aplikace

V této sekci krátce popíši a ukáži na obrázcích aktuální podobu nasazení aplikace. V ukázkách jsou samozřejmě začerněny osobní údaje klientů.

V příloze na obrázku C.2 je vidět podoba karty klienta, který docházel na Kurz Slabika a také dochází na kurz Feuersteinova metoda. Kromě údajů klienta je zde vidět několik barev lekcí, žluté značí budoucí lekce, červené s přeškrtnutý datumem zrušené lekce ze strany lektorky, bílé jsou běžné lekce bez dalších příznaků.

V příloze na obrázku C.1 je ukázka přehledu pro aktuální den, toto je

hlavní stránka aplikace, která se zobrazí po přihlášení. Lektorka zde vidí veškeré informace k lekcím na aktuální den a jednoduše může jedním kliknutím změnit stav účasti klientů či platbu, případně otevřít kartu klienta nebo skupiny. Bíle jsou individuální lekce, šedou barvou jsou zvýrazněny skupinové lekce. Naprosto stejné prostředí nabízí týdenní přehled v příloze na obrázku C.3 (je to dáno znovupoužitím stejné React komponenty), zde vidíme kromě již zmíněných prvků v denním přehledu modře zvýrazněný aktuální den v záhlaví příslušného dne, díky navigaci v horní části stránky lze bez znovunačítání stránky přecházet mezi jednotlivými týdny a vracet se jedním kliknutím na aktuální týden (tlačítko „Dnes“).

Kapitola 10

Další možná rozšíření

Vytvořená aplikace je základem ÚP a už teď jsou ve fázi plánů a návrhů další funkcionality, které jsou potřeba. Týkají se jak doplnění funkcí do stávající evidence kurzů, lekcí, klientů a skupin, tak rozšíření aplikace o další části. Cílem této krátké kapitoly je nastínit plánovaná a možná rozšíření.

Co se týče stávajících funkcí, v nejbližší době je v plánu vylepšení funkcionality předplacených kurzů, rodiče čím dál více volí možnost předplacení na mnoho lekcí dopředu a je třeba umožnit pohodlnější zaznamenání těchto předplacených lekcí, spolu s tím by se rozumněji evidovaly i předplacené lekce pro jednotlivé účastníky ze skupin, což je sice v současné době také možné, ale ne úplně pohodlně proveditelné. Dále je v plánu kontrola překryvu kurzů, tedy ochrana proti tomu, aby např. nenastal konflikt dvou lekcí v jeden čas. Také se počítá s přidáním vyhledávání do aplikace, díky kterému by se mohli klienti snadněji vyhledávat, stejně jako i další entity (např. pomůcky, viz. dále).

Mezi další funkce, které v budoucnu rozšíří působnost aplikace do dalších částí projektu patří například evidování zájemců o kurz, například skupinové kurzy totiž většinou vznikají tak, že se vytvoří skupina zájemců a když se uvolní blok v týdnu a hodí se klientům z této skupiny, začnou lekce – je tedy potřeba evidovat, kteří rodiče mají zájem o které kurzy pro své děti a jim pak po domluvě umožnit vytvořit lekce. Také je v plánu vytvoření úplně nové části systému pro evidenci pomůcek a učebnic, která je také mírně specifická a mimo administraci je potřeba její část napojit také na web. Dalším důležitým bodem ve vývoji aplikace je doplnění dalších testů a vysoké pokrytí kódu.

Jak jsem již zmínil v kapitole 7.6.2, během vývoje došlo k vydání nové verze Reactu, ruku v ruce s již zmíněnými rozšířeními je v plánu analýza možností využití nového Context API v Reactu, protože existují části aplikace, kde se domnívám, že by se toto API dalo využít k vylepšení aplikace (např. snížení počtu přístupů do REST API). Stejně tak bude potřeba nadále držet krok s novějšími verzemi Reactu a přizpůsobit se tak novému životnímu cyklu komponent a případně dalším změnám.

Mezi další nápady na vylepšení je zpřístupnění údajů offline, bude tedy po-

třeba prozkoumat možnosti řešení jako např. automatické ukládání do Google kalendáře, progresivní webové aplikace apod. a s tím související další oblasti jako SSR (viz. kapitola 3.3.2).

Takto rozšířená aplikace tak ještě více urychlí a zefektivní každodenní práci a pomůže tak lektorce získat čas pro samotné lekce a jejich přípravu a další rozvoj.

Závěr

Úspěšně jsem vytvořil webovou aplikaci na základě požadavků lektorky. Umožňuje evidovat klienty kurzů, jejich docházku, platby, historii a poskytuje přívětivé rozhraní pro úpravu těchto údajů. Součástí aplikace je také správa skupin a jejích kurzů, klientů a lekcí spolu s dalšími funkcionalitami usnadňujícími jednotlivé procesy. V počátcích se objevilo nečekaně hodně problémů při zprovozňování a propojování zvolených technologiích. Některé plynuly z mé nedostatečné znalosti dotyčných frameworků, technologií a knihoven, jiné byly způsobeny špatnou dokumentací a část problémů souvisela s odlišnými přístupy jednotlivých frameworků. Všechny problémy se ale podařilo vyřešit a došlo tak zároveň k ověření toho, že příslušné technologie lze pro takovýto typ aplikace využít a těžit z jejich spojení.

Aplikace je nasazená a je lektorkou každodenně používána. Během akceptačního testování i ostrého provozu bylo vyladěno několik problémů a přidány některé další funkce pro pohodlnější správu klientů.

Samotný úspěch lze pozorovat jak z tvrzení lektorky, která si aplikaci chválí a je s ní nadmíru spokojena, tak i z ušetřeného času, který získala díky aplikaci připravené na míru jejím potřebám.

V plánu je rozšíření aplikace o další součásti s cílem vytvoření přívětivého, jednoduchého, ale mocného systému, který pokryje potřeby ve všech specifických procesech probíhajících v projektu.

Bibliografie

1. ŠTRÁFELDA, Jan. *Co je CRM* [online]. © 2005–2018 [cit. 2018-03-26]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/crm/.
2. RAYNET S.R.O. *Představujeme RAYNET - nejoblíbenější cloudové CRM v ČR* [online]. © 2018 [cit. 2018-03-26]. Dostupné z: https://raynet.cz/cloud-crm/.
3. ČÁPKA, David. *MVC architektura* [online]. © 2018 [cit. 2018-03-28]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.
4. MINAR, Igor. *MVC vs MVVM vs MVP* [online]. 2012 [cit. 2018-03-28]. Dostupné z: https://plus.google.com/+IgorMinar/posts/DRUAkZmXjNV.
5. KLÍMA, Tomáš. *Architektura MVC* [online]. 2017 [cit. 2018-03-28]. Dostupné z: http://jakpsatphp.cz/MVC/.
6. OAMKUMAR, Reema. *This Is Why MVC Is The Most Popular Approach For Building ASP.NET Solutions* [online]. 2016 [cit. 2018-03-28]. Dostupné z: http://www.software-developer-india.com/this-is-why-mvc-is-the-most-popular-approach-for-building-asp-net-solutions/.
7. SOCRATIC SOLUTION. *Why MVC Architecture?* [online]. 2017 [cit. 2018-03-28]. Dostupné z: https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76.
8. BRAINVIRE. *Six benefits of using MVC model for effective web application development* [online]. 2016 [cit. 2018-03-28]. Dostupné z: https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/.

9. DJANGO. *Django appears to be a MVC framework, but you call the Controller the "view", and the View the "template". How come you don't use the standard names?* [online]. © 2005–2018 [cit. 2018-03-28]. Dostupné z: https://docs.djangoproject.com/en/2.0/faq/general/%5C#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.

10. TUTORIALS POINT. *Django Overview* [online]. © 2018 [cit. 2018-03-28]. Dostupné z: https://www.tutorialspoint.com/django/django_overview.htm.

11. OODESIGN. *Single Responsibility Principle* [online] [cit. 2018-03-28]. Dostupné z: http://www.oodesign.com/single-responsibility-principle.html.

12. MOLDOVAN, Alex. *Is Model-View-Controller dead on the front end?* [online]. 2016 [cit. 2018-03-28]. Dostupné z: https://medium.freecodecamp.org/is-mvc-dead-for-the-frontend-35b4d1fe39ec.

13. CRACIUNOIU, Marius. *Component Based Architecture* [online]. © 2018 [cit. 2018-03-28]. Dostupné z: https://www.uplift.agency/blog/posts/2016/05/component-based-architecture.

14. SCOTT, Damien; MORGADO, Mundi. *6 Reasons for Employing Component based UI Development* [online]. © 2017 [cit. 2018-03-28]. Dostupné z: https://www.tandemseven.com/technology/6-reasons-component-based-ui-development/.

15. AFONSO, Francisco. *3 Reasons Why You Should Invest in a Component-based Architecture* [online]. 2017 [cit. 2018-03-28]. Dostupné z: https://www.outsystems.com/blog/3-reasons-invest-component-based-architecture.html.

16. BÍLEK, Lubor. *Vývoj skriptovacích jazyků pro vývoj internetových aplikací* [online]. 2003 [cit. 2018-03-29]. Dostupné z: https://www.fi.muni.cz/usr/jkucera/pv109/2003/xbilek2.htm.

17. SQA. *Differences between Client-side and Server-side Scripting* [online]. © 2007 [cit. 2018-03-29]. Dostupné z: https://www.sqa.org.uk/e-learning/ClientSide01CD/page_18.htm.

18. VEGA, Juan. *Client-side vs. server-side rendering: why it's not all black and white* [online]. 2017 [cit. 2018-03-29]. Dostupné z: https://medium.freecodecamp.org/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d.

19. WODEHOUSE, Carey. *Front-End Web Development: Client-Side Scripting & User Experience* [online] [cit. 2018-03-29]. Dostupné z: https://www.upwork.com/hiring/development/how-scripting-languages-work/.

20. THOMS, Neal. *Client-side vs server-side scripting* [online]. 2017 [cit. 2018-03-29]. Dostupné z: https://www.fasthosts.co.uk/blog/websites/client-side-vs-server-side-scripting.

21. FREITAS, Vitor. *How to Work With AJAX Request With Django* [online]. 2016 [cit. 2018-03-29]. Dostupné z: https://simpleisbetterthancomplex.com/tutorial/2016/08/29/how-to-work-with-ajax-request-with-django.html.

22. HASSMAN, Martin. *Marek Prokop: SEO není optimalizace pro vyhledávače* [online]. 2009 [cit. 2018-03-29]. Dostupné z: https://www.zdrojak.cz/clanky/marek-prokop-seo-neni-optimalizace-pro-vyhledavace/.

23. LASN, Indrek. *Next.js — React Server Side Rendering Done Right* [online]. 2017 [cit. 2018-03-29]. Dostupné z: https://hackernoon.com/next-js-react-server-side-rendering-done-right-f9700078a3b6.

24. EAST, David. *Server-side Rendering React from Scratch! (Server-side Rendering with JavaScript Frameworks)* [online]. 2017 [cit. 2018-03-29]. Dostupné z: https://youtu.be/82tZAPMHfT4?t=11m33s.

25. GRIGORYAN, Alex. *The Benefits of Server Side Rendering Over Client Side Rendering* [online]. 2017 [cit. 2018-03-29]. Dostupné z: https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8.

26. DALE, Tom. *Point of Server-Side Rendered JavaScript Apps* [online]. 2015 [cit. 2018-03-29]. Dostupné z: https://tomdale.net/2015/02/youre-missing-the-point-of-server-side-rendered-javascript-apps/.

27. KLIMUSHYN, Mel. *Web Application Architecture from 10,000 Feet, Part 1 – Client-Side vs. Server-Side* [online]. 2015 [cit. 2018-03-30]. Dostupné z: https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/.

28. MCGREGOR, Andy; FLANDERS, David; RAMSEY, Malcolm. *The advantage of APIs* [online]. 2012 [cit. 2018-03-30]. Dostupné z: https://www.jisc.ac.uk/guides/the-advantage-of-apis.

29. MULESOFT. *What is a REST API?* [online]. © 2018 [cit. 2018-03-30]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-rest-api-design.

30. HANÁK, Drahomír. *Stopařův průvodce REST API* [online] [cit. 2018-03-30]. Dostupné z: https://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api.

31. MALÝ, Martin. *REST: architektura pro webové API* [online]. 2009 [cit. 2018-03-30]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

32. HASSMAN, Martin. *JSON : jednotný formát pro výměnu dat* [online]. 2008 [cit. 2018-03-30]. Dostupné z: https://www.zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/.

33. NEOTERIC. *Single-page application vs. multiple-page application* [online]. 2016 [cit. 2018-03-29]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

34. HARVEY, Amy. *User Experience: What Is It And Why Should I Care?* [online]. 2013 [cit. 2018-03-30]. Dostupné z: https://usabilitygeek.com/user-experience/.

35. JAIN, Shilpa. *Ultimate DEATH Match: SPA Vs. MPA* [online]. 2018 [cit. 2018-03-29]. Dostupné z: https://codeburst.io/ultimate-death-match-spa-vs-mpa-82e0b79ae6b6.

36. BOGH, Chris. *The Evolution of Web Technologies* [online]. 2012 [cit. 2018-03-29]. Dostupné z: https://www.eploy.co.uk/about-eploy/theblog/may-2012/the-evolution-of-web-technologies/.

37. VOROBIOV, Vlad; BRYKSIN, Gleb. *How to Choose a Technology Stack for Web Application Development* [online]. © 2011–2018 [cit. 2018-03-30]. Dostupné z: https://rubygarage.org/blog/technology-stack-for-web-development.

38. NEUHAUS, Jens. *9 Steps: Choosing a tech stack for your web application* [online]. 2017 [cit. 2018-03-30]. Dostupné z: https://medium.com/unicorn-supplies/9-steps-how-to-choose-a-technology-stack-for-your-web-application-a6e302398e55.

39. STACK OVERFLOW. *Developer Survey Results 2017* [online] [cit. 2018-04-01]. Dostupné z: https://insights.stackoverflow.com/survey/2017.

40. STACK OVERFLOW. *Developer Survey Results 2018* [online] [cit. 2018-04-01]. Dostupné z: https://insights.stackoverflow.com/survey/2018.

41. JETBRAINS. *The State of Developer Ecosystem in 2017* [online]. © 2000–2018 [cit. 2018-04-01]. Dostupné z: https://www.jetbrains.com/research/devecosystem-2017/.

42. NEUHAUS, Jens. *Angular vs. React vs. Vue: A 2017 comparison* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176.

43. HOTFRAMEWORKS. *Web framework rankings* [online] [cit. 2018-04-01]. Dostupné z: http://hotframeworks.com/.

44. GOOGLE. *Trendy Google* [online] [cit. 2018-04-01]. Dostupné z: https://trends.google.com/trends/.

45. W3SCHOOLS. *HTML Introduction* [online]. © 1999–2018 [cit. 2018-04-01]. Dostupné z: https://www.w3schools.com/html/html_intro.asp.

46. SLÁDEK, Jan. *Webdesignérův průvodce po HTML5 – díl nultý* [online]. 2010 [cit. 2018-04-01]. Dostupné z: https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-dil-nulty/.

47. HTML5TEST. *how well does your browser support HTML5?* [online] [cit. 2018-04-01]. Dostupné z: https://html5test.com/results/desktop.html.

48. W3SCHOOLS. *CSS Tutorial* [online]. © 1999–2018 [cit. 2018-04-01]. Dostupné z: https://www.w3schools.com/css/default.asp.

49. WODEHOUSE, Carey. *CSS vs. CSS3: New Features in the Evolving Visual Language of the Web* [online]. © 2015–2018 [cit. 2018-04-01]. Dostupné z: https://www.upwork.com/hiring/development/css-vs-css3/.

50. CAN I USE. *Support tables for HTML5, CSS3, etc* [online] [cit. 2018-04-01]. Dostupné z: https://caniuse.com/%5C#comparison.

51. SHIOTSU, Yoshitaka. *Writing a Job Description to Find a Great JavaScript Developer* [online]. © 2015–2018 [cit. 2018-04-01]. Dostupné z: https://www.upwork.com/hiring/development/javascript-developer-job-description/.

52. COPES, Flavio. *Writing a Job Description to Find a Great JavaScript Developer* [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://flaviocopes.com/ecmascript/.

53. BOLF, Petr. *Elm – Hello world on the map – úvod* [online]. 2016 [cit. 2018-04-01]. Dostupné z: https://www.zdrojak.cz/clanky/elm-uvod/.

54. JANČA, Marek. *JavaScript - z prohlížeče do operačního systému* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://www.ackee.cz/blog/javascript%E2%80%8A-%E2%80%8Az-prohlizece-do-operacniho-systemu/.

55. JUUSTILA, Sampo. *CoffeeScript vs. TypeScript vs. Dart* [online]. 2013 [cit. 2018-04-01]. Dostupné z: https://codeforhire.com/2013/06/18/coffeescript-vs-typescript-vs-dart/.

56. STEIGERWALD, Daniel. *React.js Conf 2015 – Co musíte vědět* [online]. 2015 [cit. 2018-04-01]. Dostupné z: https://www.zdrojak.cz/clanky/react-js-conf-2015-co-musite-vedet/.

57. LINDLEY, Cody. *What Is JSX?* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://codeforhire.com/2013/06/18/coffeescript-vs-typescript-vs-dart/.

58. DZIWOKI, Michał. *What's the difference between AngularJS and Angular?* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://gorrion.io/blog/angularjs-vs-angular/.

59. GOLOVIN, Artem. *Why Angular 2 (4, 5, 6) suc*s* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://medium.com/dirtyjs/why-angular-2-4-5-6-sucks-afb36567ad68.

60. ELIZONDO, Luis. *Why we moved from Angular 2 to Vue.js (and why we didn't choose React)* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://medium.com/reverdev/why-we-moved-from-angular-2-to-vue-js-and-why-we-didnt-choose-react-ef807d9f4163.

61. REACT. *A JavaScript library for building user interfaces* [online]. © 2018 [cit. 2018-04-01]. Dostupné z: https://reactjs.org/.

62. ANDRUSHKO, Sviatoslav. *The Best JS Frameworks for Front End* [online]. © 2011–2018 [cit. 2018-04-01]. Dostupné z: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end.

63. BILEJCZYK, Bartosz. *Will Vue.js Become a Giant Like Angular or React?* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://10clouds.com/blog/vuejs-angular-react/.

64. CPPCMS. *When CppCMS Should Be Used.* [online] [cit. 2018-03-30]. Dostupné z: http://cppcms.com/wikipp/en/page/when_to_use_cppcms.

65. MILDE, Daniel. *Python z pohledu PHP programátora* [online]. © 2007–2015 [cit. 2018-04-01]. Dostupné z: https://blog.milde.cz/post/305-python-z-pohledu-php-programatora/.

66. W3TECHS. *Usage of server-side programming languages for websites* [online]. © 2009–2018 [cit. 2018-04-01]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all.

67. DAITYARI, Shaumik. *Python on the Web: Why Frameworks Like Django Are Hot* [online]. 2016 [cit. 2018-04-01]. Dostupné z: https://www.sitepoint.com/python-on-the-web/.

68. CLOUDWAYS. *PHP or Python — Which Language Should You Learn in 2017* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://hackernoon.com/php-or-python-which-language-should-you-learn-in-2017-3ced1fd75ee2.

69. MARUTI TECHLABS. *Choosing the Right Back-end Technology for your Business* [online]. © 2018 [cit. 2018-04-01]. Dostupné z: https://www.marutitech.com/back-end-technology/.

70. ART+LOGIC. *Web Development and Design* [online]. © 2018 [cit. 2018-04-01]. Dostupné z: https://artandlogic.com/web-development/.

71. R.B. *»Why Does PHP S*ck?«* [online]. 2014 [cit. 2018-04-01]. Dostupné z: https://whydoesitsuck.com/why-does-php-suck/.

72. GUPTA, Suraj. *Which is Best for Web Application Development—Dot Net, PHP, Python, Ruby, or Java* [online]. 2016 [cit. 2018-04-01]. Dostupné z: https://www.addonsolutions.com/blog/which-is-best-for-web-application-development-dot-net-php-python-ruby-or-java.html.

73. DAILYRAZOR. *The 10 Best Java Web Frameworks for 2018* [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://www.dailyrazor.com/blog/best-java-web-frameworks/.

74. PLAY FRAMEWORK. *Build Modern & Scalable Web Apps with Java and Scala* [online] [cit. 2018-04-01]. Dostupné z: https://www.playframework.com/.

75. HOLEC, Miroslav. *MVC 6 neexistuje! Ať žije ASP.NET Core!* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://www.miroslavholec.cz/blog/mvc-6-neexistuje-at-zije-aspnet-core.

76. JIMMYWEB. *ASP.NET is almost certainly the wrong technology for your project* [online]. © 2003–2018 [cit. 2018-04-01]. Dostupné z: http://www.jimmyweb.net/insights/aspnet-is-almost-certainly-the-wrong-technology-for-your-website-project/.

77. ITNETWORK. *Ruby* [online]. © 2018 [cit. 2018-04-01]. Dostupné z: https://www.itnetwork.cz/programovani/ruby.

78. GRUDL, David. *Ruby on Rails? Děkuji, nechci.* [online]. 2007 [cit. 2018-04-01]. Dostupné z: https://phpfashion.com/ruby-on-rails-dekuji-nechci.

79. MISCHOOK, Stefan. *PHP vs Python in 2018?* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://www.killerphp.com/articles/php-vs-python-in-2018/.

80. ROSSUM, Guido van. *What's New In Python 3.0* [online]. 2009 [cit. 2018-04-01]. Dostupné z: https://docs.python.org/release/3.0.1/whatsnew/3.0.html.

81. DWYER, Gareth. *Flask vs. Django: Why Flask Might Be Better* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v.

82. SUBEDI, Kaushal. *Node.js S*cks! Here's Why* [online]. 2014 [cit. 2018-04-01]. Dostupné z: https://kaushalsubedi.com/blog/2014/10/15/node-js-sucks-heres-why/.

83. EXPRESS. *Node.js web application framework* [online]. © 2017 [cit. 2018-04-01]. Dostupné z: https://expressjs.com/.

84. MYSQL. *What is MySQL?* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html.

85. SQLITE. *SQLite As An Application File Format* [online] [cit. 2018-04-02]. Dostupné z: https://www.sqlite.org/appfileformat.html.

86. POSTGRESQL. *PostgreSQL: About* [online]. © 1996–2018 [cit. 2018-04-02]. Dostupné z: https://www.postgresql.org/about/.

87. MONGODB. *What is MongoDB?* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://www.mongodb.com/what-is-mongodb.

88. REDIS. *Redis* [online] [cit. 2018-04-02]. Dostupné z: https://redis.io/.

89. NASIR, Amir. *What is Cloud hosting? Difference between Cloud & traditional hosting* [online]. 2016 [cit. 2018-04-02]. Dostupné z: http://blog.webspecia.com/web-hosting/cloud-hosting-difference-between-cloud-traditional-hosting.

90. LAURA BERNHEIM. *IaaS vs. PaaS vs. SaaS Cloud Models (Differences & Examples)* [online]. 2017 [cit. 2018-04-02]. Dostupné z: http://www.hostingadvice.com/how-to/iaas-vs-paas-vs-saas/.

91. HEROKU. *Language Support* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://www.heroku.com/languages.

92. HEROKU. *Simple, flexible pricing* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://www.heroku.com/pricing.

93. HEROKU. *Regions* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://devcenter.heroku.com/articles/regions.

94. DIGITALOCEAN. *Scalable compute services* [online]. © 2018 [cit. 2018-04-02]. Dostupné z: https://www.digitalocean.com/products/droplets/.

95. BASE, Tim. *Scalable compute services* [online]. 2017 [cit. 2018-04-02]. Dostupné z: http://www.webhostwhat.com/digitalocean-datacenter-server-locations-regions-map/.

96. DIGITALOCEAN. *Simple, predictable pricing* [online]. © 2018 [cit. 2018-05-02]. Dostupné z: https://www.digitalocean.com/pricing/.

97. OPENSHIFT. *Technologies* [online] [cit. 2018-04-02]. Dostupné z: https://www.openshift.com/features/technologies.html.

98. OPENSHIFT. *Plans & Pricing* [online] [cit. 2018-04-02]. Dostupné z: https://www.openshift.com/pricing/index.html.

99. PYTHONANYWHERE. *Plans and pricing* [online] [cit. 2018-04-02]. Dostupné z: https://www.pythonanywhere.com/pricing/.

100. PYTHONANYWHERE. *Databases available* [online] [cit. 2018-04-02]. Dostupné z: https://help.pythonanywhere.com/pages/KindsOfDatabases.

101. HARRY. *Forum: co-location* [online]. 2017 [cit. 2018-04-02]. Dostupné z: https://www.pythonanywhere.com/forums/topic/10927/.

102. GOOGLE. *Google App Engine* [online] [cit. 2018-04-02]. Dostupné z: https://cloud.google.com/appengine/.

103. AMAZON. *Amazon EC2 Pricing* [online] [cit. 2018-04-02]. Dostupné z: https://aws.amazon.com/ec2/pricing/.

104. AMAZON. *AWS Free Tier* [online] [cit. 2018-04-02]. Dostupné z: https://aws.amazon.com/free/.

105. ROŠTÍ.CZ. *Roští.cz - chytrý hosting* [online]. © 2012–2018 [cit. 2018-04-02]. Dostupné z: https://rosti.cz/.

106. SIMONS, Eric. *Introducing RealWorld* [online]. 2017 [cit. 2018-04-01]. Dostupné z: https://medium.com/@ericsimons/introducing-realworld-6016654d36b5.

107. THINKSTER. *Introducing RealWorld* [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://github.com/gothinkster/realworld.

108. GRAHAM, Tim. *Django 2.0 released* [online]. 2017 [cit. 2018-03-28]. Dostupné z: https://www.djangoproject.com/weblog/2017/dec/02/django-20-released/.

109. SAHNI, Vinay. *Best Practices for Designing a Pragmatic RESTful API* [online] [cit. 2018-04-24]. Dostupné z: https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api.

110. ARSENAULT, Cody. *npm vs Yarn – Which Package Manager Should You Use?* [online]. 2017 [cit. 2018-05-04]. Dostupné z: https://www.keycdn.com/blog/npm-vs-yarn/.

111. JANČA, Marek. *Webpack – moderní Web Development* [online]. 2017 [cit. 2018-04-24]. Dostupné z: https://www.ackee.cz/blog/moderni-web-development-webpack/.

112. YADAV, Vikas. *Modern Django: Part 1: Setting up Django and React* [online]. 2017 [cit. 2018-04-13]. Dostupné z: http://v1k45.com/blog/modern-django-part-1-setting-up-django-and-react/.

113. LONE, Owais. *Using Webpack transparently with Django + hot reloading React components as a bonus* [online]. 2015 [cit. 2018-04-13]. Dostupné z: http://owaislone.org/blog/webpack-plus-reactjs-and-django/.

114. FACEBOOK. *Create React App* [online]. 2018 [cit. 2018-04-24]. Dostupné z: https://github.com/facebook/create-react-app.

115. LONE, Owais. *webpack-bundle-tracker* [online]. 2018 [cit. 2018-04-24]. Dostupné z: https://github.com/owais/webpack-bundle-tracker.

116. LONE, Owais. *django-webpack-loader* [online]. 2018 [cit. 2018-04-24]. Dostupné z: https://github.com/owais/django-webpack-loader.

117. FRAMEWORK, Django REST. *Home - Django REST framework* [online]. 2018 [cit. 2018-04-18]. Dostupné z: http://www.django-rest-framework.org/.

118. GERCHEV, Ivaylo. *The 5 Most Popular Front-end Frameworks Compared* [online]. 2018 [cit. 2018-05-04]. Dostupné z: https://www.sitepoint.com/most-popular-frontend-frameworks-compared/.

119. VAUGHN, Brian. *Update on Async Rendering* [online]. 2018 [cit. 2018-04-17]. Dostupné z: https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html.

120. ABRAMOV, Dan. *You Might Not Need Redux* [online]. 2016 [cit. 2018-04-17]. Dostupné z: https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367.

121. VAUGHN, Brian. *React v16.3.0: New lifecycles and context API* [online]. 2018 [cit. 2018-04-17]. Dostupné z: https://reactjs.org/blog/2018/03/29/react-v-16-3.html.

122. BERTOLI, Michele. *React Design Patterns and Best Practices: Build easy to scale modular applications using the most powerful components and design patterns*. Birmingham: Packt Publishing, 2017. ISBN 978-1-78646-453-8.

123. VAUGHN, Brian. *Component State* [online]. 2018 [cit. 2018-04-23]. Dostupné z: https://reactjs.org/docs/faq-state.html.

124. AXIOS. *Axios - Promise based HTTP client for the browser and node.js* [online]. 2018 [cit. 2018-05-04]. Dostupné z: https://github.com/axios/axios.

125. DJANGO. *Security in Django* [online]. © 2005–2018 [cit. 2018-04-18]. Dostupné z: https://docs.djangoproject.com/en/2.0/topics/security/.

126. JAHODA, Bohumil. *Bezpečnost webových stránek* [online]. 2013 [cit. 2018-04-18]. Dostupné z: http://jecas.cz/bezpecnost.

127. REACT. *Introducing JSX* [online]. © 2018 [cit. 2018-04-18]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

128. JWT. *Introduction to JSON Web Tokens* [online] [cit. 2018-04-18]. Dostupné z: https://jwt.io/.

129. FRAMEWORK, Django REST. *Authentication* [online]. 2018 [cit. 2018-04-18]. Dostupné z: http://www.django-rest-framework.org/api-guide/authentication/#json-web-token-authentication.

130. HEROKU. *Deploying Python and Django Apps on Heroku* [online]. 2018 [cit. 2018-04-10]. Dostupné z: https://devcenter.heroku.com/articles/deploying-python.

131. MDN. *Django Tutorial Part 11: Deploying Django to production* [online]. 2018 [cit. 2018-04-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Deployment.

132. DJANGO. *Deployment checklist* [online]. © 2005–2018 [cit. 2018-04-10]. Dostupné z: https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/.

133. DJANGO. *django-admin and manage.py* [online]. © 2005–2018 [cit. 2018-04-10]. Dostupné z: https://docs.djangoproject.com/en/2.0/ref/django-admin/.

134. EVANS, Dave. *WhiteNoise 3.3.1 documentation* [online]. © 2013–2017 [cit. 2018-04-10]. Dostupné z: http://whitenoise.evans.io/en/stable/.

135. TRAVIS CI. *Heroku Deployment* [online]. 2018 [cit. 2018-04-10]. Dostupné z: https://docs.travis-ci.com/user/deployment/heroku/.

Příloha A

Seznam použitých zkratek

ADHD Attention Deficit Hyperactivity Disorder

API Application Programming Interface

AJAX Asynchronous JavaScript and XML

CBA Component-Based Architecture

CRM Customer Relationship Management

CRUD create-read-update-delete

CSR Client-Side Rendering

CSS Cascading Style Sheets

DRF Django REST Framework

DRY Don't repeat yourself

GUI Graphical user interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IaaS Infrastructure as a Service

JS Javascript

JSON JavaScript Object Notation

MVC Model–view–controller

MVP Model–view–presenter

MVT Model-view-template

MVW Model-view-whatever

MPA Multi-Page Application

ORM Object-relational mapping

PaaS Platform as a Service

SEO Search Engine Optimization

SPA Single-Page Application

SQL Structured Query Language

SSR Server-Side Rendering

UI User Interface

URL Uniform Resource Locator

ÚP Úspěšný prvňáček

UX User Experience

XML Extensible markup language

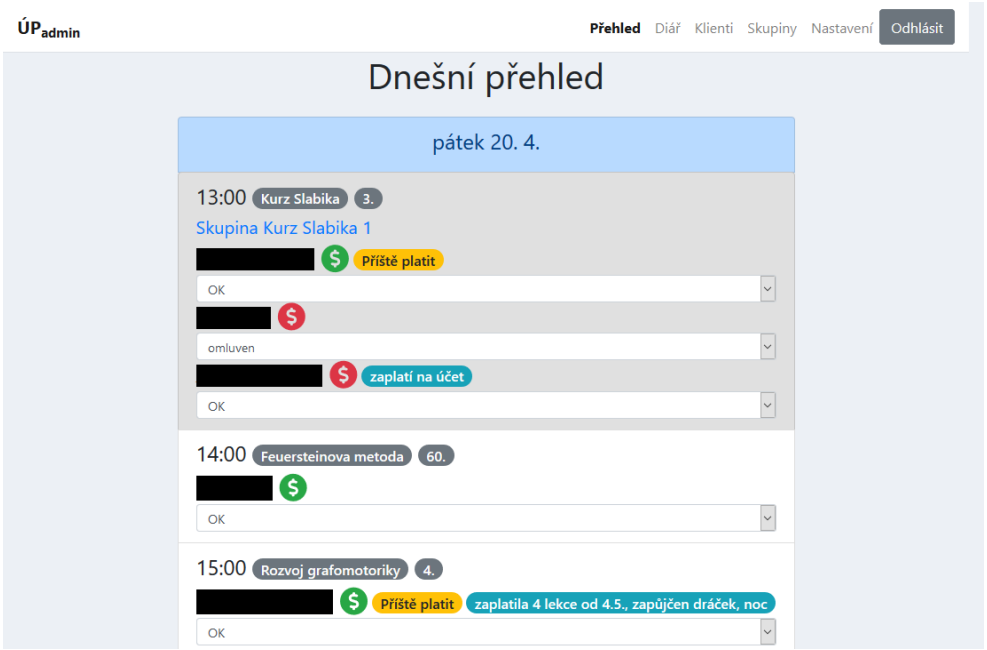
Příloha **B**

Obsah přiloženého CD

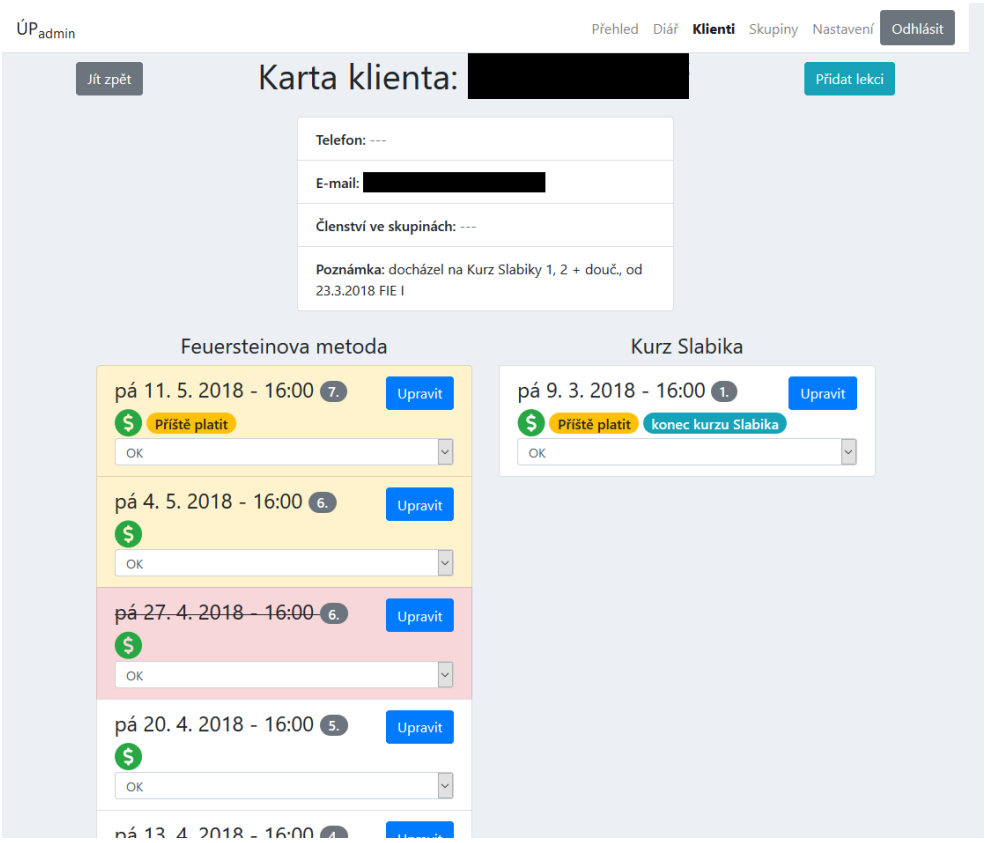
- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS

Příloha C

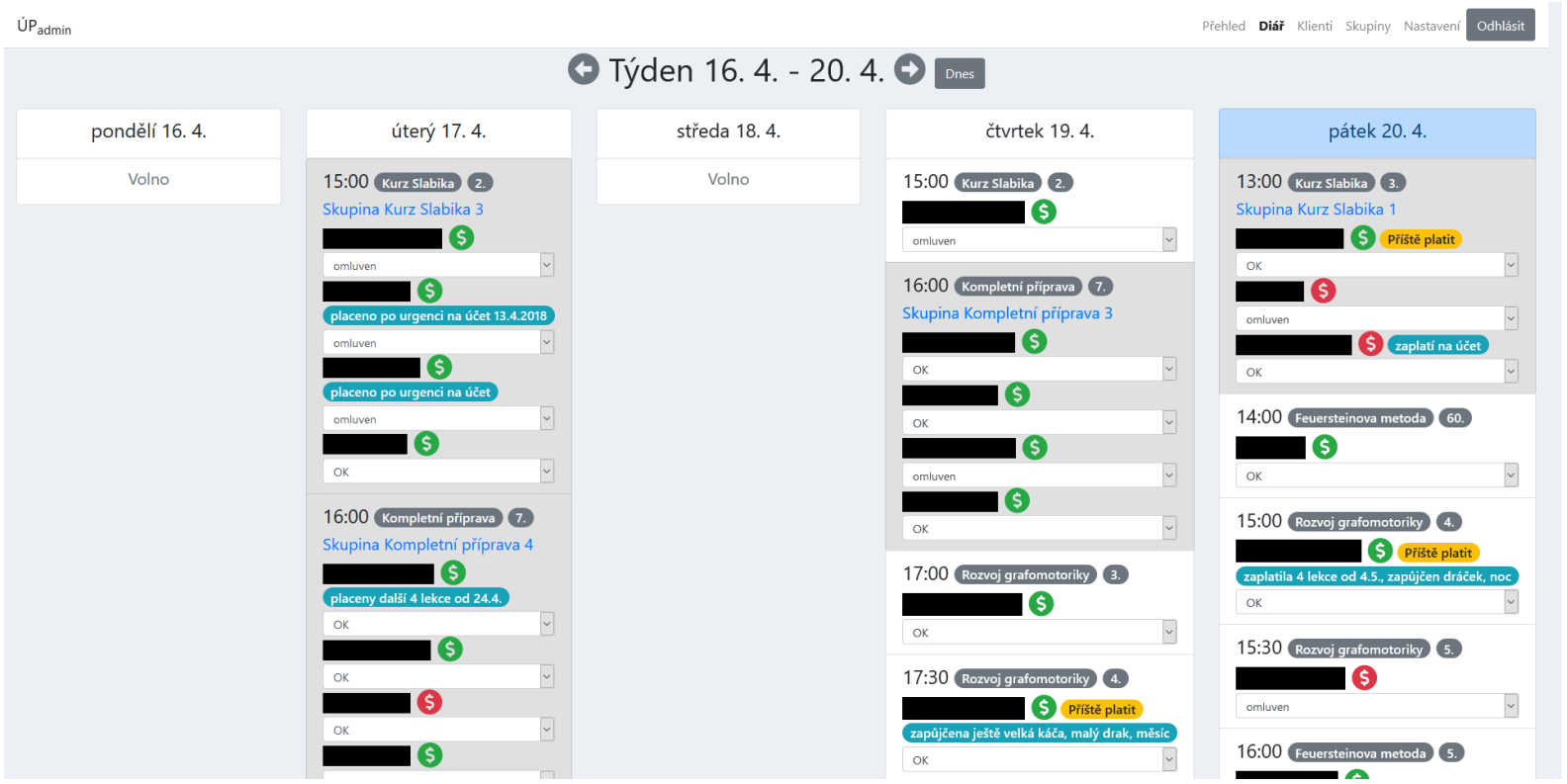
Snímky aplikace



Obrázek C.1: Snímek obrazovky s denním přehledem



Obrázek C.2: Snímek obrazovky s kartou klienta



Obrázek C.3: Snímek obrazovky s týdenním přehledem