ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název: Návrh back-endu webové crowdfundingové platformy Cityfunder
Student: Bc. Dominik Foral
Vedoucí: Ing. Pavel Krejčí
Studijní program: Informatika
Studijní obor: Webové a softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce letního semestru 2017/18

Pokyny pro vypracování

Analyzujte a zhodnoťte vybraná existující řešení zabývající se komunitními platformami podporujícími městské projekty. Navrhněte a analyzujte funkční a nefunkční požadavky pro vlastní crowdfundingovou platformu pro podporu městských projektů. Stanovte doporučení pro technickou realizaci platformy. Navrhněte datovou strukturu systému. S ohledem na zabezpečení navrhněte propojení systému s platebním systémem vybrané banky (pomocí Open API nebo jinou vhodnou metodou). Analyzujte požadavky na uživatelské rozhraní aplikace a navrhněte odpovídající aplikační rozhraní. Navrhněte a implementujte prototyp systému v rozsahu dle dohody s vedoucím práce. Proveďte vhodné systémové testy vaší implementace. Zhodnoťte závěry implementace a stanovte doporučení pro další rozvoj systému.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 22. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Návrh back-endu webové crowdfundingové platformy Cityfunder

Bc. Dominik Foral

Vedoucí práce: Ing. Pavel Krejčí

2. května 2017

Poděkování

Nejdříve bych rád poděkoval Ing. Pavlovi Krejčímu za spolupráci při vedení této práce. Dále děkuji své přítelkyni, rodině a přátelům, kteří mě během práce velmi podpořili.

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

V Praze dne 2. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2017 Dominik Foral. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Foral, Dominik. *Návrh back-endu webové crowdfundingové platformy Cityfunder.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

V úvodu práce je popsán projekt Cityfunder a porovnán s podobnými městskými crowdfundingovými portály. Poté práce provádí celým softwarovým procesem vývoje backendu této platformy. Od analýzy požadavků, výběru technologií, návrhu celého systému až po testování a dokumentaci. Návrhová část se podrobně věnuje zabezpečení a návrhu REST API a také popisuje propojení s platebním systémem PayU.

Klíčová slova crowdfunding, REST API, zabezpečení, platební systém, backend

Abstract

In the introduction, the Cityfunder project is described and compared with similar urban crowdfunding portals. Then, the thesis describes entire software development process of the backend of this platform. It starts with requirements analysis, technology selection, system design to testing and documentation. The design section details the security and design of the REST API and also describes connection with PayU payment system.

Keywords crowdfunding, REST API, security, payment system, backend

Obsah

Úvod **1**

1 Projekt Cityfunder **3**
1.1 Cityfunder 3
1.2 Cíl projektu 3
1.3 Cílové skupiny a benefity 4
1.4 Možná rizika 5
1.5 Porovnání s konkurenčními platformami 5

2 Analýza požadavků a technologií **9**
2.1 Nefunkční požadavky 9
2.2 Funkční požadavky 10
2.3 Technologie 12

3 Návrh **19**
3.1 Zvolené technologické řešení 19
3.2 Architektura systému 20
3.3 Doménový model 22
3.4 Aplikační rozhraní 23
3.5 Datový model 34
3.6 Třídní diagram 35
3.7 Platební brána 37

4 Implementace **41**
4.1 Rozsah implementace prototypu 41
4.2 Obrana proti základním typům útoků 41
4.3 Ukládání uživatelských hesel 43

5 Testování **45**
5.1 Typy testování 45

5.2 Testování při vývoji Cityfunderu 45

6 Dokumentace **49**
6.1 Základní pravidla komunikace 49
6.2 Kořen rozhraní 51
6.3 Autentizace 51
6.4 Uživatelé 52
6.5 Projekty 53
6.6 Příspěvky na projekt 54
6.7 Komentáře 55
6.8 Novinky 56
6.9 Reakce na projekt 56
6.10 Projektové náklady 57

Závěr **59**

Literatura **61**

A Seznam použitých zkratek **63**

B Obsah přiloženého CD **65**

Seznam obrázků

3.1 Architektura aplikace 21
3.2 Analytický doménový model 22
3.3 Stavový diagram projektu 24
3.4 Autentizace, komunikace a odhlášení uživatele 33
3.5 Datový model 36
3.6 Třídní diagram 37
3.7 Platební proces v PayU 38

Seznam tabulek

2.1 Nefunkční požadavky 10

3.1 Prostředky autentizace 26
3.2 Uživatelské prostředky 27
3.3 Projektové prostředky 28
3.4 Zdroje projektových nákladů 28
3.5 Zdroje projektových novinek 29
3.6 Zdroje projektových komentářů 29
3.7 Zdroje projektových reakcí 29
3.8 Prostředky plateb 30

Úvod

Projekt CityFunder vznikl na jaře roku 2016 v předmětu *Projektové a změnové řízení* společným brainstormingem mě, Markéty Zadražilové, Petra Zavadila a Jakuba Šillera. V tomto předmětu jsme rozvíjeli myšlenku projektu a ověřovali jeho životaschopnost. Po závěrečné prezentaci jsme obdrželi mnoho kladných ohlasů, což nás motivovalo tento projekt neopustit a pokračovat na jeho vývoji.

To se nám podařilo, protože jsme později s Markétou a Petrem navázali spolupráci s bankou. V rámci této spolupráce jsem zejména s Markétou pokračoval v práci na projektu, při níž nám pomáhalo a radilo také mnoho odborníků z této banky. Pomocí kvalitativních a kvantitativních dotazníků jsme zjistili nejen zájem obyvatelstva, ale také problémy, se kterými se projekt takového typu může potýkat. Pravidelnými konzultacemi jsme dolaďovali formu projektu ve všech možných detailech a ze všech možných pohledů.

Tato diplomová práce se zabývá celým procesem vývoje prototypu backendové části, která bude doplněna frontendovou částí, jež je náplní diplomové práce Markéty Zadražilové. Z této skutečnosti plyne určitá vzájemná spolupráce (zejména na aplikačním rozhraní, které bude frontend a backend propojovat), která bude nutná ke správné realizaci této aplikace.

Kapitola 1

Projekt Cityfunder

V této kapitole popisuji, jak je projekt Cityfunder navržen, co je jeho cílem, na jaké cílové skupiny je zaměřen a jaké jim přináší benefity. Poté pokračuji částí, kde zvažuji možná rizika projektu, a závěrem porovnávám Cityfunder s podobnými existujícími platformami.

1.1 Cityfunder

Cityfunder je navržen jako webová crowdfundingová [1] platforma pro podporu občanských projektů, která umožní:

- lidem v obcích prezentovat projekty týkající se veřejného prostoru, kultury, sportu apod.
- získat zpětnou vazbu od ostatních obyvatel a vedení obce
- získat potřebné finanční a nefinanční (materiální) prostředky pro realizaci projektu, a to nejen od ostatních spoluobčanů, ale také přímo od obce
- v neposlední řadě také podporu města při realizaci projektu, zejména při vyřizování povolení, případně i formou finančního, či jiného příspěvku

1.2 Cíl projektu

Hlavním cílem Cityfunderu je aktivizace občanů v obci, a to nejen těch, kteří mají nějaký nápad, který by chtěli realizovat, ale i těch, kteří by tento nápad rádi podpořili, či k němu podali nějakou zpětnou vazbu. Tato aktivizace by

[1] Crowdfunding je populární způsob, jak získat peníze či další prostředky na projekt nebo jinou aktivitu, a to přímo od lidí, kterým se daný nápad líbí. Pěkným příkladem z historie je výběr financí od občanů na nové Národní divadlo, jenž se uskutečnil v 19. století.

měla být dosažena zjednodušením a zefektivněním komunikace mezi občanem a obcí, ježz povede k větší vůli občanů měnit a zlepšovat své okolí.

Vznikajícími projekty se zpestří nejen veřejný prostor, ale také kulturní a jiné dění v obci, které může vést ke spokojenějšímu životu obyvatel a může přilákat nové návštěvníky (turisty).

1.3 Cílové skupiny a benefity

Projekt Cityfunder se zaměřuje na několik cílových skupin, kterým přináší specifické benefity.

1.3.1 Realizátoři projektů

Asi nejdůležitější cílovou skupinou jsou aktivní obyvatelé, kteří mají nápad, jenž by chtěli ve své obci realizovat. I přesto, že zastoupení této skupiny v obci je poměrně malé, vzniká v ní veškerá iniciativa k novým projektům. Pro tuto část obyvatelstva přináší projekt Cityfunder hned několik benefitů.

V první řadě to je zpětná vazba od ostatních obyvatel a vedení obce (případně úředníků), která pomůže s rozhodováním, zdali projekt realizovat, či ne, a také poskytne nápady, jak případně projekt vylepšit.

Druhým, neméně důležitým benefitem, je samotná podpora města, jež povede zejména k jednoduššímu vyřízení všech administrativních úkonů nutných k realizaci projektu. Obec také může poskytnout tvůrci projektu finanční či jinou podporu.

Veřejná prezentace projektu také může realizátorovi pomoci nalézt a spojit se s ostatními aktivními obyvateli, kteří by měli zájem se na projektu podílet.

Z analýzy, co jsme provedli, bude tuto cílovou skupinu tvořit zejména ta část obyvatelstva, která má více volného času. Reálně se by se tedy mohlo jednat o studenty a penzisty. Lidé středního věku vidíme spíše jako potencinální podporovatele projektů.

1.3.2 Obce

Samotná obec je nutná k tomu, aby projekt Cityfunder mohl správně fungovat, protože bez její podpory by se jednalo o klasický crowdfundingový portál, jenž by nepřinášel žádné nové výhody. Ideální velikost obce je, podle průzkumů a rozhovorů, co jsme provedli, nějaké menší město s cca 5 - 20 tisíci obyvateli. Takto veliké město je totiž více vzájemně svázáno, lidé, kteří v něm bydlí, ho celé dobře znají a také se znají mezi sebou. To povede k tomu, že se aktivita jednotlivých občanů ve městě tolik neztratí.

Hlavním benefitem plynoucím z nasazení Cityfunderu v obci je vytvoření komunikačního kanálu mezi občanem a vedením obce, který občanům ukáže, že se obec zajímá o jejich potřeby a nápady. Projekty vznikající z této spolupráce se mohou týkat takových záležitostí, jakými by se vedení obce vůbec

nezabývalo, protože jsou buď natolik drobné, anebo o nich vedení vůbec neví. Společně tedy pomohou vytvářet lepší obec.

Nabízí se také možnost pomocí této platformy získávat obci zpětnou vazbu na projekty, jež chce sama realizovat.

1.3.3 Ostatní obyvatelé

Pro ostatní obyvatele budou vzniklé projekty přinášet jak bohatší dění v obci, tak i samotné lepší prostředí pro život. Možnost zapojení se do projektu může také občanům přinášet určitou formu sebenaplnění a dobrého pocitu.

1.4 Možná rizika

Každý projekt nese určitá rizika a nejinak je to i s Cityfunderem. Cityfunder je založen na iniciativě obyvatelstva, na té to celé začíná i končí. Pokud nebude dostatek aktivních lidí, kteří se budou chtít podílet na tvorbě a realizaci projektů, nebude Cityfunder fungovat. Problém s nezájmem obyvatelstva je dost reálný a bude nutné obyvatelstvo dostatečně motivovat k účasti. Dále bude také náročné si dlouhodobě udržet uživatele, protože na začátku může panovat určité nadšení, které může po nějaké době vyprchat.

Stejně tak je to s obcí, ve které by měl být Cityfunder implementován. Celý princip je založen právě na podpoře od obce, a pokud ta bude chybět, nebude možné Cityfunder v obci spustit.

I přesto, že bude dostatek projektů od občanů a obec bude tyto projekty plně podporovat, se vyskytují další rizika spojená se samotnou realizací projektů. Daný autor projektu může mít problém s organizací projektu, či s motivací projekt dokončit. Projekty totiž často bývají na delší dobu, než jsou původně plánovány a mohou se vyskytnout problémy, které budou realizátory demotivovat.

Dalším rizikem, které má vliv na realizaci projektů, je fakt, že i když se daný projekt někomu líbí, najde se i někdo, komu se nelíbí. Z tohoto důvodu mohou vznikat různé spory mezi obyvateli a obcí, což danému autorovi projektu určitě motivaci nepřidá.

1.5 Porovnání s konkurenčními platformami

Cityfunder je založen na několika principech, které již v praxi fungují. V současné době ale v České republice neexistuje taková platforma, která by takovýmto způsobem tyto principy spojovala do jednoho celku. V zahraničí je ovšem situace jiná, a tak například ve Velké Británii existuje projekt SpaceHive, který se svým zaměřením velmi podobá Cityfunderu.

1.5.1 Spacehive

Tento úspěšný, zprvu pouze londýnský, projekt sdílí s Cityfunderem hlavní myšlenku, což výstižně dokazuje i jejich slogan: „We're a funding platform for projects that make local places better.“. Spacehive vznikl v roce 2012 a v současné době (únor 2017) pomohl při vzniku více než 250 úspěšných projektů. Úspěšnost projektů se pohybuje okolo 53 procent.

Projekty v tomto portálu se musí vždy týkat veřejného prostoru (nemohou být soukromé) a tím pádem přinášet prospěch celé, anebo aspoň většině společnosti. Na projekty lze přispívat financemi, materiálem, know-how či spoluprací. Přispěvateli mohou být nejen obyvatelé, ale také místní firmy a samotná města. Projekty mohou být také podpořeny formou grantu.

Než může být započata crowdfundingová fáze projektu, musí být projekt schválen. O toto schvalování se nestará samo město, nýbrž organizace z třetí strany. V tomto případě se jedná o společnosti *Locality* a *ATCM*, které představují národní sítě komunitních organizátorů. Tyto organizace kontrolují nejen samotný nápad, ale i pozemková povolení a navrhované náklady, aby posoudily proveditelnost a důvěryhodnost projektu. Tento schvalovací proces trvá často pouze několik dnů.

Dalším zajímavým prvkem, který tento portál nabízí, jsou tzv. *Hives* (úly), které představují skupiny obyvatel se společným zájmem, či sdílící stejnou lokalitu. Tyto skupiny tak pomáhají efektivněji najít vhodné přispěvatele a podporovatele projektů.

Spacehive jasně dokazuje, že projekt, s podobným nápadem jako je Cityfunder, je životaschopný a v praxi může fungovat. [1]

1.5.2 Everfund

Everfund je crowdfundingový portál, který vznikl v roce 2015 v rámci projektu Evropské hlavní město kultury Plzeň 2015. Od ostatních crowdfundingových portálů se liší zejména tím, že nabízí možnost zapojit se jinak než finančně. Odlišuje se také tím, že se zaměřuje na menší projekty z Plzně a jejího okolí a je tedy spíše lokálně zaměřen, čímž se právě podobá Cityfunderu.

Everfund fungoval zejména v roce 2015, kdy se pomocí něj podařilo dokončit 14 projektů z celkových 29, a tak se úspěšnost projektů pohybovala okolo 48 %, což je podobné jako u Spacehive. Od té doby již nevznikaly žádné další projekty, což mi bylo jedním členem týmu zdůvodněno tím, že Everfund vznikl zejména v rámci zmiňovaného projektu Evropské hlavní město kultury Plzeň 2015 a že svůj účel splnil. Dále také podotkl, že organizace vznikajících projektů byla velmi časově náročná, protože byly často nepřipravené nebo organizátoři neměli dostatečné kapacity táhnout projekt až do konce. [2]

1.5.3 Participativní rozpočet

S podobným cílem funguje i tzv. *participativní rozpočet*, který se v současné době rozběhl v mnoha českých městech (například v Praze, Ostravě či Žďáře nad Sázavou). Na začátku vyčlení daná obec finanční rozpočet, který bude možné na projekty použít. Poté se rozběhne fáze, během které mohou občané podávat své návrhy na zlepšení v jejich obci. Tato fáze je proložena různými setkáními, kde jsou jednotlivé návrhy diskutovány. Město poté zhodnotí jejich realizovatelnost a nechá obyvatelstvo hlasovat o tom, které projekty mají být uskutečněny. Návrhy, které jsou v hlasování nejúspěšnější, jsou poté městem realizovány.

Hlavním rozdílem oproti Cityfunderu je to, že realizátorem v tomto případě je daná obec, a ne autor návrhu. Na druhou stranu je participativní rozpočet, podobně jako Cityfunder, úzce spjat s danou obcí a propojuje tak přímo obyvatele a radnici. [3]

1.5.4 Ostatní crowdfundingové portály

Existuje mnoho crowdfundingových portálů jako jsou například HitHit, Startovač atd., které umožňují vybírat finanční prostředky od ostatních lidí. Tyto portály ovšem nejsou zaměřeny na občanské projekty a chybí jim podpora konkrétních obcí.

Kapitola 2

Analýza požadavků a technologií

Tato kapitola pojednává o funkčních a nefunkčních požadavcích kladených zejména na backendovou část aplikace. Dále popisuje vybrané programovací jazyky, databázové systémy, platební systémy a přístupy k aplikačním rozhraním, které lze v rámci implementace využít.

2.1 Nefunkční požadavky

Nefunkční požadavky popisují obecné nároky na vznikající aplikaci, které nesouvisí s funkcemi, které má aplikace nabízet. V této části je postupně zanalyzuji a poté ještě přehledně shrnu v tabulce.

Výkon určitě není primárním požadavkem, protože není pravděpodobné, že by byla aplikace, vzhledem k její povaze, příliš zatěžovaná. Je ovšem nutné zajistit alespoň přijatelnou dobu odezvy, aby jednotlivé požadavky netrvaly moc dlouho a neobtěžovaly tím uživatele. U běžných požadavků, u kterých neprobíhá velký přenos dat (například nahrávání souborů) či náročný výpočet, by doba odezvy měla být do 1 sekundy, a to pro řádově desítky souběžných požadavků.

Škálovatelnost není podobně jako výkon důležitým nárokem kladeným na výslednou aplikaci.

Rozšiřitelnost a modifikovatelnost je určitě jedním z klíčových požadavků. Vzhledem k počáteční fázi projektu je jasné, že bude časem potřeba provést větší nebo menší změny v systému, a tak je nutné, aby na to byla aplikace dostatečně připravena. Případné změny lze očekávat zejména u platebního systému, kde by měla být zajištěna snadná výměna jeho implementace. Na změny by mělo být také připraveno aplikační rozhraní, které zastřešuje celou backendovou část.

Tabulka 2.1: Nefukční požadavky

Typ	Požadavek	Priorita
Výkon	Odezva běžných požadavků pod 1s pro desítky současných požadavků	Střední
Škálovatelnost	dfsd	sdfsfd
Rozšiřitelnost a modifikovatelnost	Změna implementace platebního systému Změna aplikačního rozhraní	Vysoká Vysoká
Bezpečnost	Ochrana citlivých informací	Vysoká

Bezpečnost je u tohoto druhu aplikace bezesporu nejdůležitějším požadavkem. V systému budou probíhat finanční transakce, a tak je nutné zabránit jejich zneužití a úniku citlivých informací. Kromě toho je samozřejmě nutné dostatečně chránit i ostatní neveřejná data, se kterými bude aplikace pracovat.

2.2 Funkční požadavky

Funkční požadavky definuji pro následující 3 role:

- Nepřihlášený uživatel - návštěvník, který není přihlášený
- Přihlášený uživatel - registrovaný uživatel, který je přihlášený
- Administrátor - schvalovatel projektů, administrátor

Jednotlivé role od sebe dědí pravomoce, takže administrátor může vše uvedené, přihlášený uživatel může také to, co nepřihlášený (až na registraci, která pro tuto roli nedává smysl), a nepřihlášený uživatel může jen to, co je u něj uvedené.

2.2.1 Nepřihlášený uživatel

Registrace uživatele - vytvoření uživatelského účtu v aplikaci pomocí emailu a hesla

Přihlášení uživatele - přihlášení do systému pomocí emailu a hesla zadaného během registrace

Zobrazení veřejného profilu uživatele - zobrazení veřejného profilu jiného uživatele

Zobrazení seznamu projektů - zobrazení projektů v Cityfunderu, třídění podle stavu

Zobrazení projektů uživatele - zobrazení projektů daného uživatele

Třídění projektů - zobrazení veřejných projektů podle jejich stavu

Zobrazení projektu - zobrazení všech veřejných informací o projektu - včetně projektových nákladů, novinek, komentářů, přispěvatelů a reakcí

Kontaktovat uživatele - odeslání emailu uživateli (např. autorovi projektu)

2.2.2 Přihlášený uživatel

Zobrazení profilu - zobrazení svého uživatelského profilu

Změna údajů - změna osobních údajů

Změna hesla - změna přihlašovacího hesla

Přispění finanční částky - přispění libovolné finanční částky na projekt pomocí platební brány

Přispění materiálu - přispění materiálu pro projekt

Příslib spoluúčasti - přislíbení spoluúčasti na realizaci projektu

Okomentování projektu - vložení komentáře k projektu

Úprava/smazání komentáře projektu - úprava či smazání svého komentáře u projektu

Like/dislike projektu - ohodnocení projektu formou *like* (líbí) nebo *dislike* (nelíbí)

Vytvoření nového projektu - vytvoření nového projektu v aplikaci, zadání základních údajů, nákladů a dalších informací

Zobrazení svých projektů - zobrazení svých projektů v jakékoli fázi

Upravování údajů o projektu - upravování údajů o projektu, některé údaje lze měnit pouze v některých fázích projektu, viz dále stavový diagram projektu v sekci 3.3.2

Předání projektu ke schválení - uživatel dá svůj nový projekt ke schválení

Vkládání novinek k projektu - vkládání novinek k vlastnímu projektu

2.2.3 Administrátor

Zobrazení projektů - zobrazení všech projektů (i nezveřejněných) podle jejich stavu, zejména těch ke schválení

Schválení/neschválení projektu - schválení nebo neschválení projektu, který čeká na schválení (libovolná změna stavu projektu)

Přidělení nutných povolení k projektu - přidělení povolení k projektu, která jsou nutná vyřídit k jeho realizaci

Smazání jakéhokoli komentáře - smazání jakéhokoli komentáře u projektu

2.3 Technologie

V současné době existuje mnoho různých technologií, s pomocí kterých lze naimplementovat backend takovéto aplikace. V této části postupně analyzuji různé programovací jazyky, databázové systémy a další vhodné technologie.

2.3.1 Programovací jazyk

Vzhledem k tomu, že budu implementovat backend, nezajímá mě, jaké nabízí daný jazyk možnosti pro tvorbu uživatelského rozhraní, ale spíše, jak umožňuje vytvořit aplikační rozhraní a jaké nabízí možnosti pro komunikaci s databází.

2.3.1.1 PHP

PHP je v současnosti suverénně nejpoužívanější server-side programovací jazyk na webu [4]. Díky tomu má širokou základnu programátorů a mnoho různých frameworků a knihoven, které poskytují většinu funkcí, jež vývojář potřebuje.

Pro objektově-relační mapování (ORM) se nabízí oblíbený framework *Doctrine* nebo také jeho alternativa *Propel*. K vytvoření aplikačního rozhraní se nabízí nepřeberné množství knihoven. Pro REST variantu API zmíním například knihovny *Silex* nebo *Slim* a pro SOAP API existuje například knihovna *Nusoap*.

Jako největší výhodu tohoto programovacího jazyka vidím jeho snadnou nasaditelnost na web, protože je pro PHP nabízeno obrovské množství webhostingů, které jsou často i zadarmo.

O nevýhodách PHP toho bylo napsáno opravdu hodně. Zdá se, že největší slabinou PHP jsou samotní programátoři, kteří, díky tomu, že je PHP poměrně jednoduché na naučení, tvoří a publikují nekvalitní (zranitelné) programy. PHP také díky své flexibilitě (a skriptovací nátuře) nenutí programátory psát kdovíjak kvalitně navržené programy, a tak často vzniká špatně udržovatelný kód (tzv. „spagetti code“) [5]. Tyto často uváděné nevýhody ovšem nepovažuji za tolik závažné, protože nezpůsobují žádné větší problémy, pokud si jich je programátor vědom.

Další často uváděnou nevýhodou PHP je slabší výkon, který je způsoben zejména tím, že PHP je interpretovaný jazyk. Ovšem i tento problém se dá řešit například paralelizací, kterou nabízí knihovna *pthreads*, případně horizontálním škálováním, použitím *HipHop virtual machine* od společnosti Facebook, která implementuje *Just in time* kompilaci [6], a v neposlední řadě využitím cache paměti.

Mně osobně na PHP nejvíce vadí, že se nejedná o silně typovaný programovací jazyk, čímž se pro mě snižuje čitelnost kódu a hůře se pracuje s daty předávanými v aplikaci.

2.3.1.2 Java Enterprise Edition

Java Enterprise Edition (dále Java EE) není ani zdaleka tak používaným jazykem na webu jako PHP [4]. Jedná se o sadu specifikací, která rozšiřuje standartní verzi Javy kupříkladu o technologii *Java Server Faces* pro tvorbu webového uživatelského rozhraní, *Java Message Services* pro asynchronní komunikaci pomocí zpráv nebo *Enterprise Java Beans*. Pro práci s databází je vytvořena specifikace *Java Persistence API*, která je spolu s *Java API for RESTful Web Services* (JAX-RS) na tvorbu RESTful webových služeb také součástí Javy SE [7].

Mezi implementace technologie JPA patří například framework *Hibernate*, jenž umožňuje objektově-relační mapování [8]. Implementací JAX-RS je poměrně více, přičemž zmíním například *Jersey* jako referenční implementaci [2] od *Oracle* [9] nebo *Apache CXF*.

Výhodou Javy oproti PHP jsou právě zmiňované specifikace, které dávají jazyku jasný řád a pořádek. Díky tomu navádí programátora psát aplikace s kvalitnějším softwarovým návrhem (objektově orientované, vícevrstvé a modulární).

Nevýhodou Javy je bezesporu menší nabídka hostingových služeb, zejména těch zadarmo, které bych rád využil pro protypování této aplikace. Problémem také může být samotné nastavení aplikace, které může sice fungovat na lokálním počítači, ale jinak tomu bude na hostingu. Hlavní roli zde mohou hrát různé verze Javy či samotného aplikačního serveru.

2.3.2 Databázový systém

Existuje několik typů databázových systémů. Nejznámějším a nejpoužívanějším typem jsou relační databáze, kde jsou entity reprezentovány pomocí tabulek, kde sloupce reprezentují entitní atributy a řádky zastupují jednotlivé záznamy v databázi. Pro práci s relačními databázemi slouží jazyk *SQL*. V praxi se často používá objektově-relační mapování (ORM), které spojuje

[2] Referenční implementace je vytvářena se specifikací, ověřuje, zdali je specifikace implementovatelná a také vytváří standard, se kterým lze srovnávat ostatní implementace.

objekt v aplikaci a tabulku v databázi [10]. Mezi nejznámější systémy patří *MySQL*, *Oracle*, *Microsoft SQL Server* a *PostgreSQL*.

Dále také existují objektově orientované databáze, jež reprezentují data ve formě objektů. Umožňují tak zpracování komplexnějších datových struktur, než je tomu u relačních databází, a také lépe zapadají do objektově orientovaného programování. Navzdory tomu nejsou databáze tohoto typu tak oblíbené jako relační. Může za to nejspíše fakt, že jsou úzce spojeny s datovým modelem aplikace a také s programovacím jazykem, v němž je aplikace implementována. Mezi známé implementace patří například *OpenLink Virtuoso* pro PHP nebo *ObjectDB* pro Javu. Některé objektově orientované prvky také obsahuje databáze od Oracle, PostgreSQL či Microsoft SQL Server (jedná se ve skutečnosti o objektově-relační databázové systémy) [11].

Za zmínku určitě také stojí grafové databáze, jež uchovávají data ve formě uzlů a hran. Jako příklad uvedu databázi *Neo4j* či *OrientDB*. Tento typ databází se ovšem pro tento typ aplikace nehodí.

2.3.3 Platební systém

Vzhledem k tomu, že Cityfunder je crowdfundingový portál, je jeho nedílnou součástí je platební systém, pomocí kterého bude nutné implementovat následující funkční požadavky:

- možnost přispět finanční částku na projekt
- zjisit aktuálně vybranou částku na projekt
- odeslat vybranou částku na účet tvůrce projektu (tento požadavek má nízkou prioritu)

V následujících kapitolách rozeberu jednotlivé možnosti plateb na internetu, které jsou v současné době možné, ale i ty, které vypadají, že budou možné v budoucnu.

2.3.3.1 Platební brána

K placení na internetu se v praxi nejčastěji používají platební brány, které zprostředkovávají bankovní přenosy, a to buď pomocí platby kartou, kdy se vyplní pouze údaje z kreditní karty, anebo přednastaveným bankovním převodem u banky přispěvatele, který se jen přihlásí do svého internetového bankovnictví a potvrdí finanční částku.

Takovéto platební brány nabízí většina bank a internetových peněženek (např. GoPay nebo PayU) a jsou připraveny na snadnou implementaci do webové aplikace. Běžnou praxí je, že si společnosti za transakce provedené přes jejich platební bránu účtují určité provize (v řádu procent), k tomu často také něco stojí samotná integrace brány. Tyto poplatky se také často odvíjí od měsíčního objemu plateb.

Při vývoji aplikace je často k dispozici testovací mód brány, při němž nedochází k reálným transakcím, a tak je možné celý proces plateb řádně vyzkoušet. Platební brány mimo samotného zprostředkování plateb také například poskytují informace o provedených transakcích, čímž je možné v aplikaci zjistit, které platby proběhly, a tak v případě crowdfundingového portálu vypočítat celkovou vybranou částku [12].

2.3.3.2 Manuální převody

Alternativou k platební bráně může být prosté uvedení údajů bankovního převodu, na základě kterých poté plátce (přispěvatel) provede převod pomocí internetového bankovnictví či složenky. Nevýhodou této metody je zajisté menší uživatelská přívětivost a prostor pro udělání chyby při přepisování údajů. Tyto nevýhody lze ale vcelku elegantně eliminovat pomocí tzv. *QR plateb*, které se v posledních letech staly velmi oblíbenými.

QR platby využívají při iniciaci platby QR kód, který obsahuje řetězec *Short Payment Descriptor* nesoucí číslo cílového účtu, částku a měnu převodu [13]. Plátce pak využívá mobilní aplikaci své banky, v rámci které naskenuje kód a nastaví tak údaje požadované transakce. Tímto způsobem se předejde chybě při přepisování údajů a také se celý proces platby výrazně zrychlí. Díky své povaze ale nejsou QR platby dostupné pro všechny, protože k jejich používání je nutné mít tablet či chytrý telefon. Na druhou stranu, nespornou výhodou je, že se za provedení platby neplatí žádné poplatky.

Nevýhodou těchto manuálních plateb je ale zejména to, že se v aplikacích obtížně kontrolují. Aplikace totiž musí mít přístup k transakcím, které na účtě proběhly, a z těch počítat například již zmiňovanou celkovou vybranou částku na projekt. Aplikační rozhraní, které by umožňovalo přístup k transakcím na účtě, ovšem v současné době nabízí pouze dvě tuzemské banky - Fio banka a Česká spořitelna. Tato situace se má ovšem brzy změnit díky směrnici Evropské unie PSD2 (Payment service directive).

2.3.3.3 Payment service directive

PSD2 byla představena v roce 2013 jako novela k předchozí směrnici PSD z roku 2007 a má nabrat platnosti v roce 2018. Tato směrnice nařizuje evropským bankám poskytnout třetím stranám rozhraní pro iniciaci plateb a získávání informací o účtech. Z bank se tím pádem mají stát platformy, poskytující své služby třetím stranám, které budou mít se svolením zákazníka přístup k jeho účtu. PSD2 by tak pomohla vzniknout úplně novým bankovním službám, které dosud jinak nebyly možné. Příkladem takové služby může být jednotné bankovnictví postavené nad několika účty několika různých bank [14].

V rámci Cityfunderu by se takovéto otevřené bankovnictví dalo využít na kontrolu stavu účtu, případně k automatizovanému zasílání vybraných peněz

na účet tvůrce projektu. K vybírání financí od přispěvatelů se ovšem hodí spíše současný model platební brány, protože případ, kdy by přispěvatel poskytl Cityfunderu dobrovolně přístup ke svému účtu, aby mohla být provedena transakce, se mi zdá pro tento účel nevhodný, neboť platba by měla být iniciována spíše ze strany plátce než aplikace.

2.3.4 Aplikační rozhraní

Aplikační rozhraní (API) bude sloužit jako jediný komunikační kanál s frontendovou částí aplikace a bude implementováno jako webová služba, tzn. že bude dostupné přes internet.

2.3.4.1 Representational state transfer

V současné době se pro budování webových služeb často používá architektonický styl *REST* (Representational state transfer), který navrhl Roy Fielding ve své dizertační práci v roce 2000. Architektura REST je postavena na několika základních principech:

Klient - server Architektura klient server zajišťuje správné oddělení zodpovědností jednotlivých stran, díky čemuž může docházet k jejich nezávislému vývoji, jednodušší škálovatelnosti a rozvoji klientské části na více platformách.

Bezestavovost Každý klientský požadavek by měl obsahovat všechny informace, které jsou nutné k jeho vykonání. Stav sezení (komunikace) je tedy zcela uložen na straně klienta. Tato bezestavovost přináší hned několik výhod. Hlavním přínosem je snadná škálovatelnost aplikace - díky tomu, že je stav uložen na klientu, můžou být jeho požadavky odesílány na různé servery a ty je mohou stejně obsloužit. Jsou ovšem situace (např. autentizace), kdy je nutné tomuto požadavku mírně ustoupit.

Cache - Odpovědi serveru by měly být označeny, zdali jsou kešovatelné, či nikoliv. Díky tomu, že si mohou klienti kešovat odpovědi, dochází ke snížení počtu požadavků na server.

Jednotné rozhraní Architektura REST uvaluje na rozhraní několik omezení:

- Rozhraní je postaveno na prostředcích (resource based), které jsou identifikovány pomocí URI. Reprezentace prostředků, použita v komunikaci, je pak nezávislá na jejich skutečné reprezentaci.
- Práce s prostředky probíhá skrz jejich reprezentaci.
- Samopopisující se zprávy, které sdělují, jak s nimi pracovat (např. v jakém formátu jsou a zdali jsou kešovatelné).

- Hypermedia as the Engine of Application State (HATEOAS) je velmi důležitý princip, který popisuje, že stav aplikace je přenášen pomocí hypertextu (stručně řečeno textu s hypertextovými odkazy). V praxi to mimo jiné znamená, že v odpovědi serveru není pouze reprezentace požadovaného prostředku, ale také odkazy na další související prostředky či na prostředek samotný. Například při požadavku o kolekci prostředků je navráceno pole prostředků, kde každý obsahuje ještě navíc odkaz na sebe samotného. V ideálním případě by celé aplikační rozhraní mělo být dostatečně samopopisující a provázané odkazy tak, že k jeho porozumění stačí pouze jeho kořenová adresa, jinak řečeno by mělo být rozhraní použitelné bez jakékoli externí dokumentace. Tento princip není v praxi často dodržován a API poskytují často jen data, nikoliv odkazy mezi nimi. Takovéto rozhraní je samozřejmě zcela použitelné, ovšem nemůže se nazývat „RESTful API“.

Vrstvený systém Díky vrstvenému systému lze zvýšit škálovatelnost či bezpečnost aplikace. Klient přitom nepozná, že není připojený ke koncovému serveru, ale k mezivrstvě, která komunikaci zprostředkovává.

Kód na požádání Server může poslat klientovi kus programového kódu, čímž rozšíří jeho program o další funkci. Toto omezení je jako jediné volitelné [15].

Shrnutím těchto základních principů jsem chtěl mimo jiné poukázat na častý omyl týkající se architektury REST. Často se uvádí, že je tato architektura úzce svázána s protokolem HTTP, ovšem v principech není o tomto protokolu ani zmínka. Důvodem tohoto omylu je, že REST se v drtivé většině případů používá právě s tímto protokolem. Jak používat REST s HTTP je pak přímo dáno protokolovou specifikací.

2.3.4.2 Simple object access protocol

Zcela rozdílný přístup přináší při implementaci webových služeb protokol *Simple Object Access Protocol* (SOAP). SOAP a REST se tak nedají přímo porovnávat, protože se jedná o protokol a architektonický styl. Obecně řečeno, SOAP je protokol, který slouží k výměně informací ve formě zpráv mezi počítači v síti. Přenášené zprávy mohou být, narozdíl od RESTu, pouze ve formátu XML a skládají se z několika elementů:

- Obálka (Envelope) je základním (kořenovým) elementem zprávy, který obaluje její ostatní části
- Hlavička (Header), která je volitelnou částí, obsahuje parametry používané při zpracování zprávy

- Tělo (Body) obsahuje XML data posílaná zprávou
- Chyba (Fault) obsahuje informace o chybě, která vznikla při zpracování požadavku

Vzhledem ke své specifikaci je protokol SOAP platformě nezávislý a může plnit mnoho různých funkcí. Velmi často se používá právě pro přenos dat ve webových službách, konkrétně ke vzdálenému volání procedur přes protokol HTTP. Zpráva pak ve svém těle obsahuje parametry, které se mají dané proceduře předat, a je odeslána metodou POST na daný bod na serveru [16].

V porovnání s REST architekturou vede využívání SOAP ve webových službách k většímu svázání klienta se serverem, což není obecně dobrý přístup. Jakékoli změny na serveru mají většinou přímý vliv na chování u klienta, REST je, díky svému principu HATEOAS, vůči těmto změnám odolnější.

Častou uváděnou nevýhodou je také nutnost využití XML zprávy, která je i u jednoduchých požadavků výrazně větší, než jak by byla zpráva například ve formátu JSON.

Kapitola 3

Návrh

Tato stěžejní kapitola popisuje návrh jednotlivých částí systému. V úvodu jsou popsána zvolená technologická řešení backendu a frontendu. Dále následuje architektura systému, která částečně vyplývá ze zvolených technologií. Na to navazuje doménový model a stavový diagram projektu. Nejrozsáhlejší část se týká aplikačního rozhraní a podrobně popisuje návrh struktury rozhraní a jeho zabezpečení při komunikaci s frontendem. Za touto částí následuje návrh tříd aplikace. Kapitolu potom uzavírá návrh propojení s platebním systémem PayU.

3.1 Zvolené technologické řešení

3.1.1 Backend

Pro implementaci jsem se rozhodl zvolit Javu Enterprise Edition, a to zejména z důvodu, že nabízí kvalitní rozhraní pro tvorbu aplikačního rozhraní (JAX-RS, JAX-WS), o kterém jsem se již zmiňoval v části 2.3.1.2. Jak jsem již psal výše, hostingových služeb pro Javu není tolik jako pro PHP, podařilo se mi ovšem nalézt vhodnou nabídku od společnosti RedHat, která nabízí ve svém produktu Openshift hosting javových (ale i mnoha dalších) aplikací v základní verzi na měsíc zdarma. I když není měsíc příliš dlouhá doba, pro testování postačí a navíc se lze opětovně zaregistrovat a využít tak další měsíc.

Při výběru aplikačního serveru jsem přihlížel k tomu, co je v Openshift k dispozici, abych vyvíjel na stejném serveru, na kterém poté aplikace poběží. Aplikační servery jsou sice vyvíjeny podle dané specifikace, ale určité rozdíly mezi nimi jsou a ty by při přesunu aplikace mohly způsobit problémy. V Openshift je k dispozici aplikační server Wildfly (dříve JBoss Application Server), který vyvíjí právě RedHat, a tak jsem zvolil právě jej (konkrétně verzi *9.0.2*).

Nabídka databázových systémů byla rozsáhlejší, a tak jsem měl na výběr mezi *MariaDb*, *MySQL*, *PostgreSQL* a *MongoDB*. Vzhledem k tomu, že na databázi nejsou kladeny žádné konkrétní požadavky, zvolil jsem MySQL, a to

hlavně díky předchozím, poměrně bohatým zkušenostem s tímto databázovým systémem.

Pro vývoj aplikace jsem zvolil, z důvodu osobní preference, prostředí NetBeans. Ke správě verzí jsem pak použil systém Git.

3.1.2 Frontend

Pro implementaci frontendu zvolila Markéta javascriptovou knihovnu *React.js*, pomocí které lze vytvořit zobrazovací vrstvu aplikace. Spolu s dalšími javascriptovými knihovnami poté tvoří funkční frontend, jenž komunikuje s backendem pomocí aplikačního rozhraní.

3.2 Architektura systému

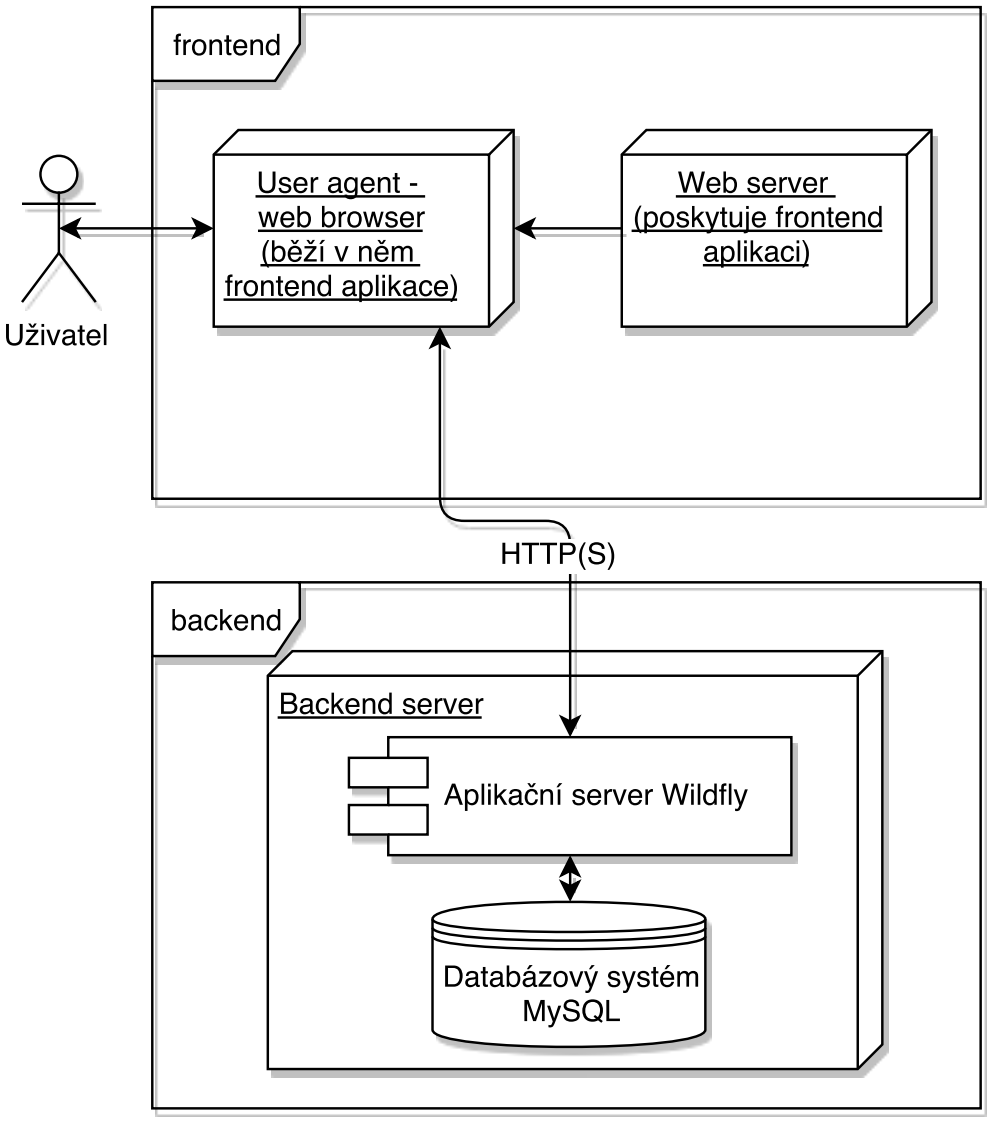
Achitektura systému se skládá ze dvou základních částí - frontend a backend. Backend slouží k ukládání dat a obsahuje základní aplikační logiku. Frontend poté představuje uživatelské rozhraní, které slouží ke komunikaci s uživatelem. Tyto dvě části spolu komunikují pomocí aplikačního rozhraní, které poskytuje backend. Tímto způsobem jsou jednotlivé části minimálně provázané a je možné je v případě potřeby jednoduše vyměnit za zcela jinou implementaci. Například backend lze naimplementovat v PHP, aniž by to mělo jakýkoliv vliv na frontend, nebo frontend lze naimplementovat jako nativní mobilní aplikaci, aniž by bylo nutné upravovat backend.

Základem backendové části je aplikační server Wildfly, v němž běží celá Java aplikace. Aplikační server k tomu komunikuje s databázovým systémem MySQL. Tyto dvě komponenty běží serveru, který konkrétně v produktu Openshift používá operační systém CentOS, což je linuxová distribuce založená na Red Hat Enterprise Linuxu.

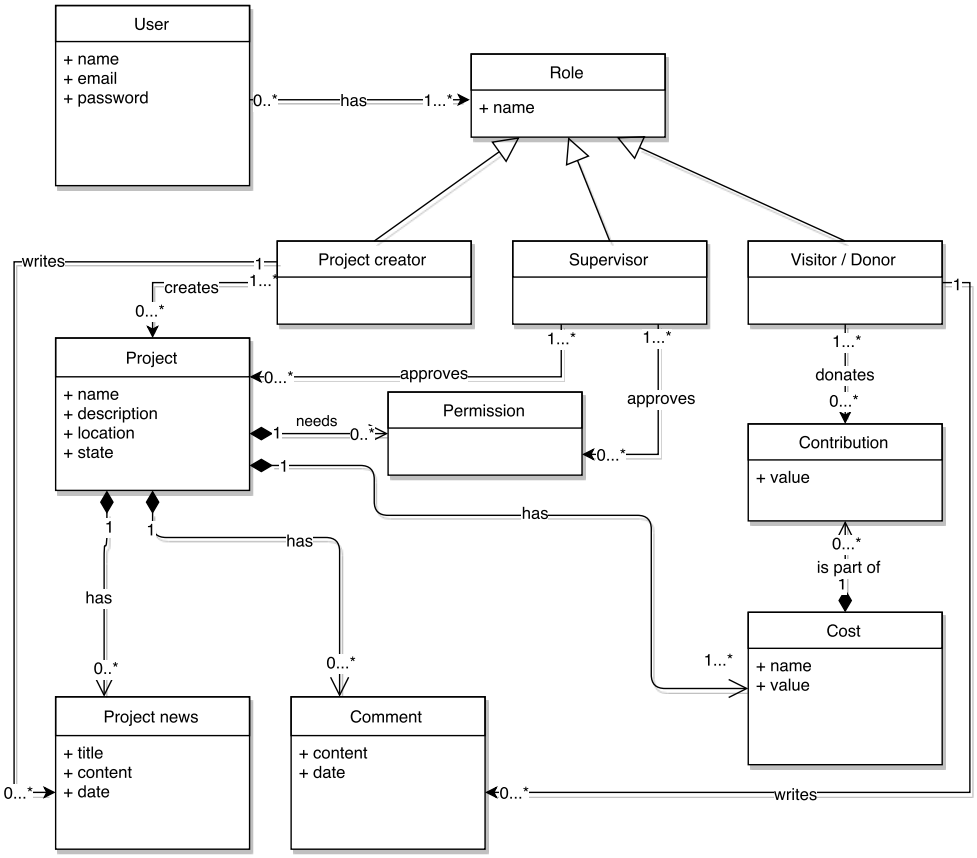
Základem frontendové části je webový server, který obsahuje frontendovou aplikaci. Tuto celou aplikaci pak posílá na žádost webovému prohlížeči uživatele, kde poté běží. Běžící aplikace potom komunikuje s backendovou částí protokolem HTTP. Jedná se o tzv. *user-agent based* aplikaci, protože celá běží v internetovém prohlížeči a se serverem pouze vyměňuje data. Obecněji se poté jedná o veřejného klienta. Veřejný se nazývá proto, že není schopný zajistit důvěrnost [3] svých dat, protože je celý uložen na straně klienta, jenž má tím pádem přístup jak ke zdrojovým kódům, tak i k jakýmkoli datům, se kterými aplikace pracuje. Této problematice se budu ještě dále věnovat.

Architektura aplikace je také přehledně znázorněna na diagramu 3.1.

[3] Důvěrnost je spolu s celistvostí a dostupností základním principem informační bezpečnosti.



Obrázek 3.1: Architektura aplikace



Obrázek 3.2: Analytický doménový model

3.3 Doménový model

3.3.1 Analytický doménový model

Analytický doménový model zobrazuje základní vztahy mezi jednotlivými entitami v rámci dané domény.

Základními entitami jsou *User*, jenž představuje uživatele používající aplikaci, a *Project*, což je projekt, který v rámci aplikace vzniká. K těmto entitám se vztahují všechny ostatní. Uživatel má uživatelské role (entita *Role*), kterými mohou být tvůrce projektu (*Project creator*), dohlížitel projektů (*Supervisor*), či návštěvník/dárce (*Visitor/donor*). Typicky bude mít každý návštěvník webu možnost být jak tvůrce projektu, tak přispěvatel. Projekt má přiřazené novinky (*Project news*) vytvářené tvůrcem projektu, komentáře (*Comment*) psané návštěvníky. Projekt dále potřebuje určitá povolení (*Permissions*) a příspěvky (*Contribution*) od dárců, které jsou částmi jeho nákladů (*Cost*).

Tyto vztahy jsou přehledně znázorněny v diagramu 3.2, kde jsou také zahrnuty základní atributy, které jednotlivé entity vlastní.

3.3.2 Stavový diagram projektu

Stavový diagram na obrázku 3.3 zobrazuje jednotlivé stavy projektu, vytvářeného v aplikaci Cityfunder, a přechody mezi nimi. Zobrazené přechody jsou popsány ve formě: *událost [podmínka] / akce*.

Na začátku, když je vytvořen nový projekt, je stav projektu *New*. V této fázi ještě není projekt dostupný veřejnosti a autor ho může libovolně upravovat. Jakmile se rozhodne, že má projekt připravený na zveřejnění, pošle ho ke schválení - stav *For approval*. V tomto stavu je kontrolováno, zdali neporušuje projektová pravidla, či nemá-li nějaké další nedostatky. Pokud pravidla porušuje, je projekt zrušen a jeho stav se změní na *Canceled*. V opačném případě se posouvá do další fáze *In funding*, kdy už je veřejně dostupný. Kontrolor projektu má také možnost vrátit projekt k přepracování zpět do fáze *New* a celý proces tak pak začíná od začátku.

Fáze *In funding* je časově omezená a slouží pro sbírání potřebných prostředků nutných k realizaci projektu. Na konci této fáze je posouzeno, zdali jsou vybrány požadované prostředky, a pokud ano, projekt se posouvá do fáze *Funded*. V případě, že projekt nezískal dostatečnou podporu, je zrušen a tedy jeho stav změněn na *Canceled*.

Pokud nebyla ve fázi *In funding* vyřízena také všechna potřebná povolení, je ještě nutné je získat. Poté je teprve možné začít fázi realizace (*In realization*). V opačném případě je projekt zrušen (přechod do stavu *Canceled*) a vybrané prostředky jsou vráceny jejich přispěvatelům.

Projekt může z nejrůznějších důvodů selhat také ve fází realizace (*In realization*), jeho stav, podobně jako v předchozí fázi, je změnen na *Canceled* a vybrané prostředky jsou vráceny dárcům. Pokud je projekt úspěšně zrealizován, přechází do konečné fáze *Finished*.

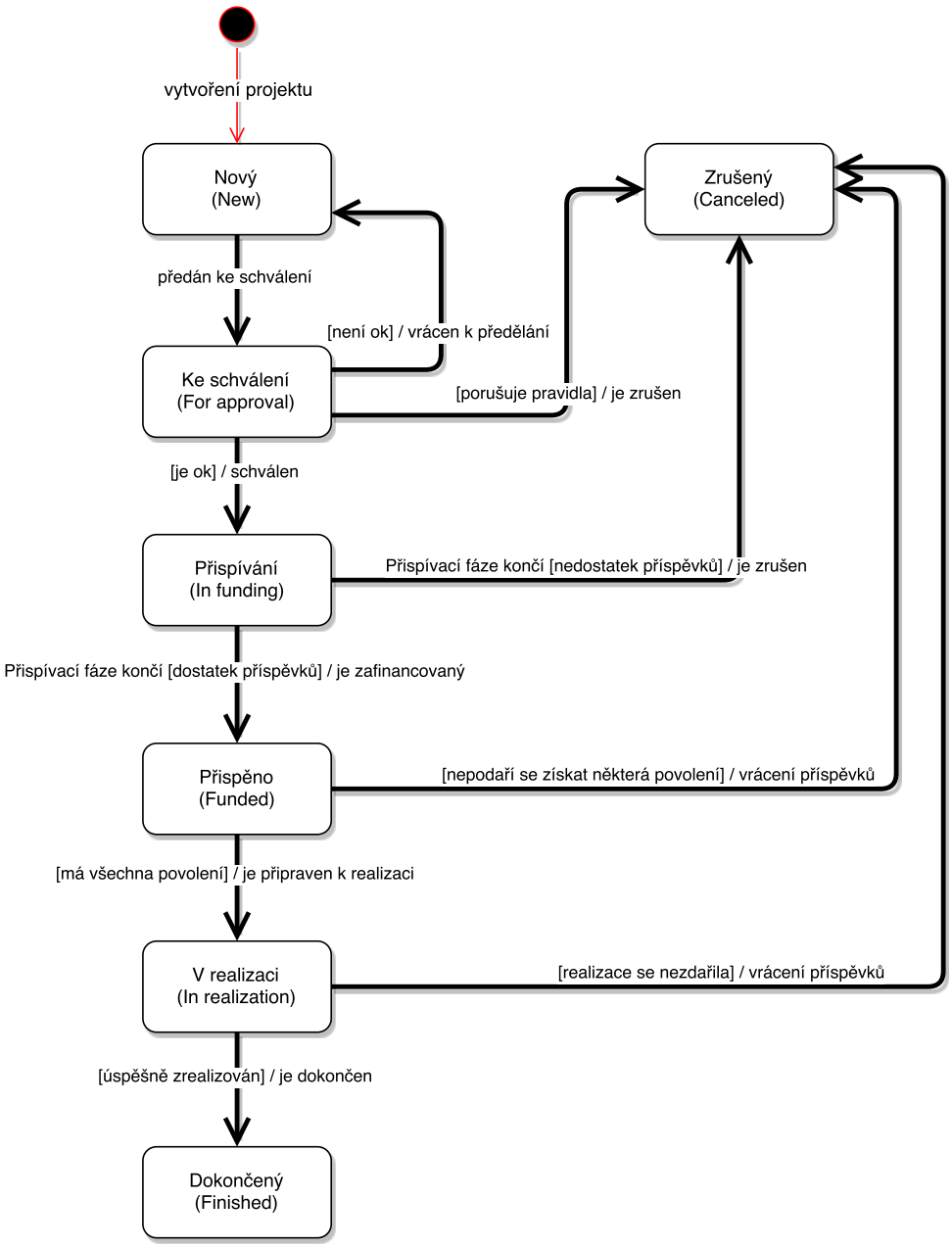
3.4 Aplikační rozhraní

Aplikační rozhraní bude postaveno na architektonickém stylu REST (Representational state transfer), který mimo jiné staví rozhraní na tzv. prostředcích (resources), jež jsou identifikovány jedinečnou URL. Tyto prostředky představují nejčastěji jednotlivé entity, či jejich množiny v systému. Obecný popis této architektury lze nalézt v kapitole 2.3.4.1.

Při typickém použití REST architektury s protokolem HTTP jsou operace s prostředky určovány různými HTTP metodami. Často se základní CRUD[4] operace mapují na metody *POST*, *GET*, *PUT* a *DELETE*, ovšem situace je podle HTTP specifikace trochu složitější, jak rozeberu v následujícím seznamu.

POST - Požaduje zpracování reprezentace entity z požadavku podle sémantiky dané prostředkem. Což může být například vytvoření prostředku,

[4]Create, recieve, update, delete



Obrázek 3.3: Stavový diagram projektu

nebo nějaká jiná operace, pro kterou se nehodí žádná jiná HTTP metoda.

GET - Slouží primárně k získávání informací

DELETE - Požaduje po serveru rozpojení asociace mezi daným prostředkem a jeho funkčností, což může být provedeno například jeho smazáním, či archivací.

PUT - Požaduje po serveru nahrazení nebo vytvoření stavu daného prostředku, který je přiložen v požadavku. Reálně to znamená, že pomocí této metody se mohou prostředky jak vytvářet, tak měnit. Změny jsou vykonány nahrazením celého prostředku reprezentací z požadavku, což se v praxi často nehodí, protože klient musí posílat celou reprezentaci prostředku, i když chce provést pouze změnu například jednoho atributu.

PATCH - Požaduje aplikaci množiny změn uvedených v požadavku na daný prostředek. Hodí se tedy právě pro účely částečných úprav prostředků. [17]

Odpovědí serveru je pak vhodný HTTP kód a případná data. Nejčastěji budu v rozhraní používat následující HTTP kódy:

200 OK - operace proběhla v pořádku, v těle odpovědi jsou požadovaná data, která se odvíjí od typu požadavku (HTTP metody)

201 CREATED - operace proběhla v pořádku, byl vytvořen nový prostředek, v hlavičce *Location* je jeho URL

204 NO CONTENT - operace proběhla v pořádku a nejsou žádná data k odeslání

401 UNAUTHORIZED - požadavku chybí autentizační údaje nebo nebyly autorizovány k provedení dané operace

400 BAD REQUEST - požadavek nemohl být zpracován, protože je neplatný (chyba na straně klienta)

404 NOT FOUND - požadovaný prostředek nebyl nalezen

500 SERVER ERROR - chyba na straně serveru (při zpracování požadavku)

Aby rozhraní splňovalo principy REST architektury (konkrétně *HATEOAS*), budou reprezentace entit odeslané serverem obsahovat hypertextové odkazy na související entity. Základním rozcestníkem bude kořen rozhraní, který bude obsahovat odkazy na autentizaci, projekty a uživatele. Každý projekt bude odkazovat na svoje komentáře, náklady, novinky, reakce a příspěvky. U entit, ke kterým bude možné přistupovat jednotlivě, bude dále odkaz na ně samotné.

Tabulka 3.1: Prostředky autentizace

URL prefix	`/auth`		
URL	Metoda	Funkce	Autorizovaný uživatel
`/login`	POST	přijímá email a heslo, na základě kterého vrací přístupový a obnovovací token	Registrovaný
`/refresh`	POST	přijímá obnovovací token z cookie, na základě kterého vydává nový obnovovací a přístupový token	Přihlášený
`/logout`	POST	odebírá přístupový a obnovovací token	Přihlášený

3.4.1 Struktura rozhraní

Struktura rozhraní je, jak jsem již psal výše, určena jednotlivými prostředky. Prostředky jsem rozdělil do odpovídajících skupin a přehledně zachytil do tabulek. V každé tabulce je ve vrchní části URL prefix, který je první částí URL prostředku. Dále má každý prostředek svou relativní URL vůči zmiňovanému prefixu, metody, které lze nad prostředkem volat, popis těchto metod. V posledním sloupci je uvedeno, kdo je autorizován danou metodu nad daným prostředkem volat.

Názvy jednotlivých prostředků jsem vytvořil na základě uznávaných praktik. Jedná se o anglická podstatná jména, popisující daný prostředek - tedy např. *projects* pro projekty. Nemělo by se jednat o slovesa popisující akci, protože to odporuje REST principům - rozhraní popisuje prostředky (datovou strukturu) a akce jsou pak určeny jednotlivými HTTP metodami (HTTP verbs). Podstatná jména jsou navíc (kromě autentizace) v množném čísle, protože ve všech případech popisují kolekce.

Podrobnější popis prostředků, včetně ukázkových požadavků, je rozebrán v sekci 6.

3.4.2 Zabezpečení

Aby byla zajištěna bezpečnost komunikace mezi frontendem a backendem, je nutné implementovat několik bezpečnostních principů, které postupně popíši v této části.

Tabulka 3.2: Uživatelské prostředky

URL prefix	`/users`		
URL	Metoda	Funkce	Autorizovaný uživatel
	POST	přijímá a registruje nového uživatele	Každý
`/{user_id}`	GET	odesílá veřejně dostupné informace o uživateli s user_id	Každý
	PATCH	přijímá data, která mají být u uživatele změněna (kromě hesla, emailu a uživatelské role)	Vlastník
`/{user_id}/password`	PUT	přijímá nové a staré heslo a mění heslo u daného uživatele	Vlastník
`/{user_id}/projects`	GET	získání všech veřejných projektů uživatele	Každý
	GET	získání všech projektů uživatele	Vlastník
`/current`	GET	odesílá všechny informace kromě hesla o aktuálně přihlášeném uživateli	Přihlášený

3.4.2.1 Autorizace

Prvním principem je autorizace uživatele, která zajišťuje, že se k zabezpečeným prostředkům dostane pouze ten, kdo na to má právo. Během procesu autorizace bude nutné zkontrolovat, zdali daný uživatel má roli, která ho opravňuje k přístupu k prostředkům, a někdy také, zdali je vlastník daného prostředku.

V sekci 2.2 jsem rozdělil jednotlivé funkční požadavky mezi tři uživatelské role: nepřihlášený, přihlášený uživatel a administrátor. Tyto funkční požadavky představují operace, jež se budou skládat z jednoho nebo více volání aplikačního rozhraní. Pro nepřihlášeného uživatele nebude nutná žádná autorizace, protože se jedná o operace, které může provést každý. Pro přihlášeného uživatele a administrátora ovšem bude nutné jednotlivé operace autorizovat na základě jeho uživatelské role a vlastnictví.

Tabulka 3.3: Projektové prostředky

URL prefix	`/projects`		
URL	Metoda	Funkce	Autorizovaný uživatel
	GET	získání všech veřejných projektů	Každý
	POST	vytvoření nového projektu	Přihlášený
`?state={p_state}`	GET	získání veřejných projektů podle jejich stavu	Každý
	GET	získání všech projektů podle stavu	Admin
`/{p_id}`	GET	získání zveřejněného projektu podle p_id	Každý
	GET	získání nezveřejněného projektu podle p_id	Vlastník, Admin
	PATCH	úprava projektu ve fázi NEW	Vlastník
`/{p_id}/state`	PUT	změna stavu projektu na FOR_APPROVAL	Vlastník
	PUT	libovolná změna stavu projektu	Admin

Tabulka 3.4: Zdroje projektových nákladů

URL prefix	`/projects/{p_id}/costs`		
URL	Metoda	Funkce	Autorizovaný uživatel
	GET	získání nákladů daného projektu	Každý
	POST	vytvoření nákladu projektu	Vlastník
`/{cost_id}`	PATCH	úprava nákladu u projektu ve stavu NEW podle cost_id	Vlastník
	DELETE	smazání nákladu u projektu ve stavu NEW podle cost_id	Vlastník
	GET	získání daného projektového nákladu	Každý

Tabulka 3.5: Zdroje projektových novinek

URL prefix	`/projects/{p_id}/news`		
URL	Metoda	Funkce	Autorizovaný uživatel
	GET	získání novinek daného projektu	Každý
	POST	vytvoření novinky projektu	Vlastník
`/{news_id}`	GET	získání novinky projektu podle news_id	Každý

Tabulka 3.6: Zdroje projektových komentářů

URL prefix	`/projects/{p_id}/comments`		
URL	Metoda	Funkce	Autorizovaný uživatel
	GET	získání komentářů daného projektu	Každý
	POST	vytvoření komentáře projektu	Přihlášený
`/{comm_id}`	GET	získání komentáře projektu podle comm_id	Každý
	PATCH	upravení komentáře podle comm_id	Vlastník
	DELETE	smazání komentáře projektu podle comm_id	Vlastník, Admin

Tabulka 3.7: Zdroje projektových reakcí

URL prefix	`/projects/{p_id}/reactions`		
URL	Metoda	Funkce	Autorizovaný uživatel
	POST	vložení/přepsání své reakce k projektu	Přihlášený
	DELETE	odstranění své reakce u projektu	Přihlášený
	GET	získání všech projektových reakcí	Každý
`?type={t}`	GET	získání všech reakcí k projektu podle typu t	Každý
`?type={t}&count`	GET	získání počtu všech reakcí k projektu podle typu t	Každý

Tabulka 3.8: Prostředky plateb

URL prefix	`/projects/{p_id}/contributions`		
URL	Metoda	Funkce	Autorizovaný uživatel
	POST	přispění finanční částky na projekt s p_id	Každý
	GET	získání všech příspěvků na projekt	Každý
`/{c_id}`	GET	získání info o daném příspěvku s c_id	Každý

3.4.2.2 Autentizace

Před tím, než bude možné uživatele autorizovat k provedení dané operace, je nutné provést jeho autentizaci, tzn. ověření identity, podle přihlašovacích údajů. Autentizace funguje nejčastěji tak, že se uživatel prokáže svými údaji a pokud jsou správné, dostane od autentizačního serveru data, jimiž se prokazuje při následných požadavcích. Tato autentizace obvykle probíhá pomocí *cookie* nebo *přístupového tokenu*.

Principy fungování cookies a přístupových tokenů se mohou zdát na první pohled velmi podobné, protože jsou zasílány s každým požadavkem na zabezpečený prostředek, ovšem je mezi nimi vcelku výrazný rozdíl.

Cookie obsahuje identifikátor, který musí mít server uložen v databázi. K tomuto identifikátoru mohou být uloženy další informace o uživateli, kterému cookie patří. Server při každém požadavku musí nahlédnout do databáze a najít informace k danému identifikátoru. Ve zkratce řečeno, všechny informace o cookie jsou uloženy na straně serveru, čímž je vytvářen stav, který musí server během životního cyklu cookie udržovat. Tento princip fungování přináší například problémy se škálovatelností, protože je náročné udržovat tyto informace sdílené mezi více servery.

Oproti tomu přístupový token přímo identifikuje uživatele (či aplikaci) a reálně může obsahovat jakékoli další informace (kromě důvěrných). Ověření na straně serveru poté spočívá pouze v přečtení informací z tokenu, tudíž odpadá nutnost mít token uložený v databázi, což také snižuje výpočetní a paměťovou náročnost aplikace. Server tak nemusí udržovat žádný stav (stav drží klient), což umožňuje mimo jiné jednoduchou škálovatelnost. Tato bezestavovost také lépe vyhovuje architektuře REST, která ji má jako jeden z hlavních principů.

Častou implementací přístupových tokenů v prostředí webu jsou *JSON web tokens (JWT)*, které se skládají ze tří částí. První částí je hlavička, která obsahuje základní informace o tokenu. Druhá část obsahuje data (claims). Typicky se jedná o čas expirace, poskytovatele tokenu nebo také jméno a role uživatele. Tyto první dvě části jsou zakódovány (nikoliv zašifrovány) do formátu *url64encoded*. Vše je na konci doplněno podpisem, který zajišťuje, že

pochází od pověřeného zdroje, a ujišťuje, že nebyl obsah tokenu jakkoli změněn [18].

Ochrana přístupového tokenu proti zneužití spočívá zejména v jeho časově omezené platnosti. I když se útočníkovi podaří token nějakým způsobem získat, může ho zneužít pouze, dokud neexpiruje. Doba platnosti se liší podle případů užití. Například u internetových bankovnictví to často bývá pouze několik minut, oproti tomu u aplikací, jež nevyžadují tak silné zabezpečení, bývá platnost několik hodin až dní.

Problém nastává, jakmile přístupový token vyprší a uživatel stále aplikaci používá. Token je v danou chvíli neplatný, a tudíž není použitelný k autentizaci. Aplikace může sice vyzvat uživatele, aby se znovu autentizoval, ovšem tento způsob je z uživatelského hlediska velmi obtěžující.

V praxi se k obnovení přístupového tokenu používá tzv. *refresh (obnovovací) token*, jenž je spolu s přístupovým tokenem poskytnut serverem během autentizace uživatele. Tento obnovovací token je poté klientem zaslán v případě, že ten přístupový vypršel, a server na jeho základě vystaví nový přístupový a často i obnovovací token.

Z principu fungování obnovovacích tokenů ale vyplývá, že musí být dostatečně ochráněny před zcizením. Na rozdíl od přístupových tokenů získává totiž útočník ukradením obnovovacího tokenu (téměř) neomezený přístup k zabezpečeným prostředkům. Jak jsem již částečně naznačil v sekci 3.2, na straně veřejného klienta nelze zaručit zabezpečené ukládání důvěrných dat a tím pádem ani obnovovacího tokenu.

V tuto chvíli by se mohlo zdát, že ukládání obnovovacího tokenu na straně klienta nelze provést. Ovšem pokud si uvědomíme, že klient je v tomto případě i internetový prohlížeč, zjistíme, že řešení existuje. Server totiž může poslat data, která jsou dostupná pouze internetovému prohlížeči, nikoliv ovšem programu, který v něm běží. Konkrétně se jedná o tzv. *HttpOnly cookie*, ke kterému nelze přistoupit pomocí JavaScriptu v prohlížeči, čímž zabraňuje rozšířenému typu útoku *Cross Site Scripting* [19]. Pokud tedy server uloží obnovovací token do tohoto typu cookie, posílá ho pak internetový prohlížeč s každým požadavkem na server, kde může být ověřen a v případě potřeby použit k vygenerování nového přístupového tokenu. Klientská aplikace tím pádem pracuje pouze s přístupovým tokenem a o obnovovacím tokenu nic neví. Cookie s parametrem HttpOnly zde tedy není použito k autentizaci, jak tomu tak často bývá, ale jako bezpečné úložiště pro obnovovací token. K tomu lze ještě nastavit cookie parametr *secure*, jenž nařizuje prohlížeči posílat dané cookie pouze přes zabezpečený protokol HTTPS.

Existuje nejspíše řešení, jak implementovat i obnovovací tokeny bezestavově (na straně serveru) - napadá mě například mít uložený hash obnovovacího tokenu v přístupovém JWT tokenu, jenž by server v případě požadavku o obnovení přístupového tokenu porovnal s hashí obnovovacího tokenu v HttpOnly cookie. Hash obnovovacího tokenu by ani v případě ukradení přístupového tokenu a v případě, že byla použita kvalitní hashovací funkce, neumožnil

útočníkovi získat nový token a server by si stále nemusel žádný token ukládat.

Je ale několik důvodů, proč je dobré alespoň obnovovací tokeny na serveru ukládat. V porovnání s počtem všech požadavků na server, je počet požadavků o obnovení tokenů výrazně menší, a tak by bylo možné, mít pro tuto činnost vyhrazen jeden centrální server nebo databázi, aniž by to více zkomplikovalo škálování aplikace. Škálování aplikace ale není důležitým požadavkem u této práce, a tak se jím již nebudu dále zabývat.

Výhodou, kterou přináší ukládání obnovovacích tokenů, ale je, že v případě potřeby je možné odebrat uživateli přístup k aplikaci. Nelze sice přímo zneplatnit přístupový token (bylo by nutné ho mít uložený někde na straně serveru), je ale možné zakázat daný obnovovací token, a tak znemožnit uživateli získat nový. Uživatel se bude poté muset znovu autentizovat. Odebrání přístupu k aplikaci je potřeba například v situaci, kdy se uživatel chce odhlásit, nebo v případě bezpečnostního incidentu, jakým může být například ztráta zařízení.

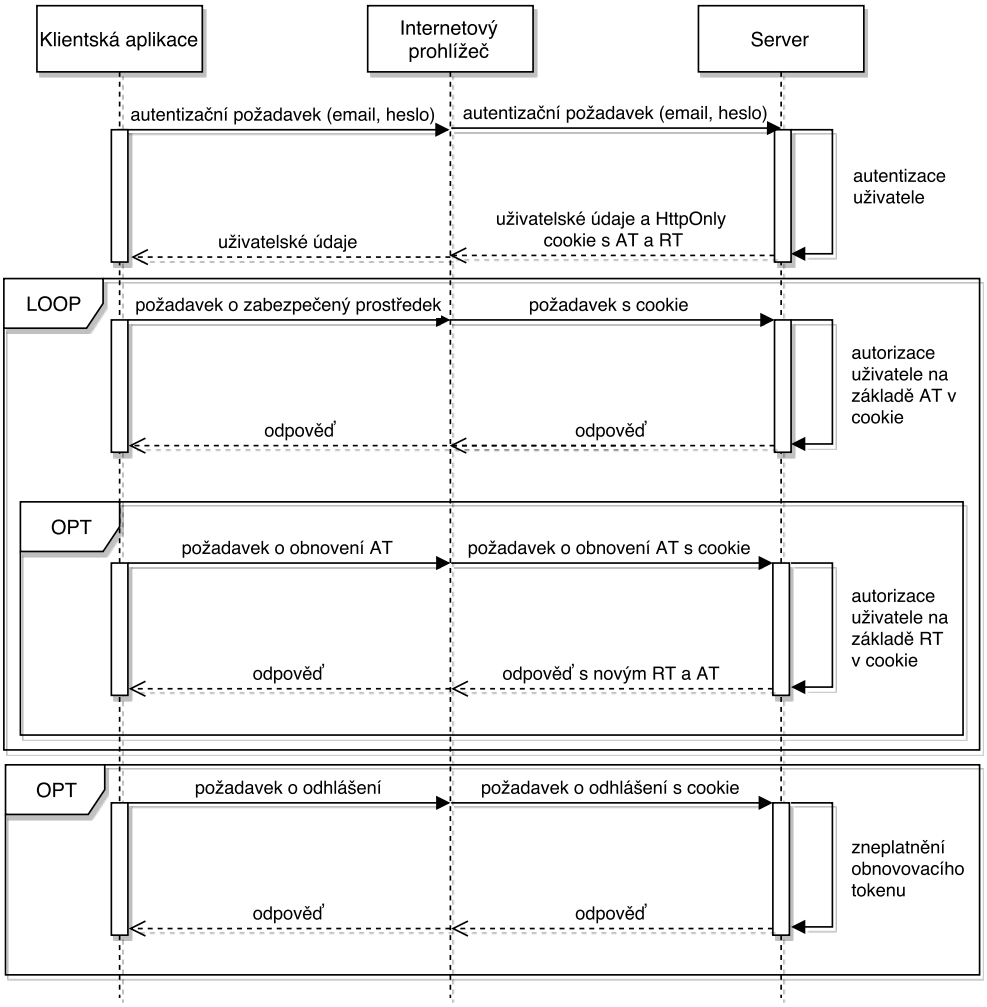
Když už ukládáme obnovovací token do onoho zabezpečeného cookie, nabízí se možnost tam ukládat i ten přístupový. Tím pádem už se aplikace nemusí starat o správu tokenů (o vše se postará prohlížeč). Tento přístup ale zcela odstíní klienta od JWT tokenu, a tak nemůže tímto způsobem zjistit informace o aktuálně přihlášeném uživateli. Řešení této situace je ale triviální, protože server může buď posílat informace o uživateli v odpovědi na autentizační požadavek (a klient si je poté uloží), anebo vystaví metodu v API, jež bude tyto informace poskytovat.

Ukládání obnovovacího tokenu do HttpOnly cookie je tedy dobrý způsob, jak ho ochránit před útoky Cross site scripting. Jinak je tomu bohužel s útoky typu *Cross Site Request Forgery*, které provádí nechtěné požadavky z útočníkova webu na uživatelův web s využitím cookies uložených v jeho prohlížeči. I proti tomuto typu útoků ale existuje obrana, kterou zde ale nebudu rozebírat, protože ji nemám v plánu v rámci prototypu implementovat.

Na diagram 3.4 jsem zobrazil celý navržený proces autentizace. Vystupují v něm tři aktéři:

- Klientská aplikace - JavaScriptový frontend
- Internetový prohlížeč - runtime prostředí pro aplikaci, všechny požadavky z a do aplikace jdou přes něj
- Server - backend, v tomto případě v roli autentizačního serveru

V první fázi probíhá samotná autentizace uživatele, jenž zadá své údaje do přihlašovacího formuláře. Data jsou dále odeslána na server, kde je uživatel autentizován. V případě úspěchu posílá server zpět uživatelské údaje a v HttpOnly cookie dále přístupový (AT) a obnovovací (RT) token. Prohlížeč si uloží přijatá cookie a aplikace obdrží pouze uživatelská data.



Obrázek 3.4: Autentizace, komunikace a odhlášení uživatele

V další fázi, jež je na diagramu zobrazená v rámci *LOOP*, probíhá už požadovaná komunikace se serverem. Aplikace pošle požadavek o zabezpečený prostředek, prohlížeč k němu připojí uložená cookie a na jejich základě poté proběhne na straně serveru autorizace uživatele. V případě úspěchu jsou zpět zaslána požadovaná data.

Mezi jednotlivými požadavky může dojít k expiraci přístupového tokenu. Aplikace tak může zažádat o obnovení tokenu, komunikace probíhá podobně jako v předchozích případech (cookie připojuje k požadavku prohlížeč). Tento typ požadavku je zobrazen v rámci *OPT* uvnitř rámce *LOOP* a je volitelný (je pouze nutný k udržení komunikace).

Poslední (volitelnou) fázi odhlášení zobrazuje spodní rámec *OPT*. Komunikace je opět podobná jako v předchozích případech a dojde v ní na straně

serveru ke zrušení obou cookie a zneplatnění obnovovacího tokenu v databázi.

3.4.2.3 Zabezpečení transportní vrstvy

K tomu, aby byla zajištěna bezpečná komunikace, je také nutné zabezpečit data během jejich přenosu po síti. Při komunikaci přes internet k tomu typicky slouží protokol HTTPS, který je rozšířením protokolu HTTP o šifrování pomocí TLS (Transport security layer). Tento protokol zajišťuje, že data nemohou být přečtena, změněna či jinak zneužita třetí stranou a také autentizuje webový server, se kterým komunikace probíhá [20].

3.5 Datový model

K ukládání dat do databáze využiji ORM Hibernate, díky kterému nemusím ručně vytvářet databázový model. Stačí vytvořit požadované entitní třídy, definovat mezi nimi vztahy a o mapování do databáze se postará framework.

Entitní třídy se v JPA označují pomocí anotace *@Entity*, objekty těchto tříd bude možné ukládat do databáze. Tyto třídy musí mít veřejný (public) konstruktor bez parametrů a atribut označený anotací *@Id*, který představuje jejich jedinečný identifikátor (v jazyce databází se jedná o primární klíč). K jednotlivým atributům se přistupuje pomocí veřejných metod *get* a *set* vytvořených pro každý z nich.

Identifikátor typu *Long* spolu s metodami *getId* a *setId* jsem umístil do abstraktní třídy *AbstractEntity*. Třídy, jimž identifikátor tohoto typu vyhovuje, jsou pak potomky této abstraktní třídy. Třída *AbstractEntity* není označena anotací *Entity*, nýbrž anotací *@MappedSuperclass*, protože se nejedná o entitní třídu jako takovou, ale pouze o rodičovskou třídu ostatních entit. V databázi tím pádem nebude vůbec samostatně ukládána, ale její atribut *Id* bude součástí každého jejího entitního potomka. Mezi potomky *AbstractEntity* patří tyto třídy:

Project - projekt vznikající v CityFunderu, mimo své atributy definuje také výčet (enum) *ProjectState*, který představuje jednotlivé fáze projektu, a dále podtřídy *Location* a *ProjectProgress*, které vyjadřují místo a vývoj (vybrané prostředky, cílová částka) projektu

Cost - náklad, který je součástí projektu, definuje také výčet *CostType*, který určuje typ nákladu (finance, materiál, čas)

News - novinky přidávané k projektu

Comment - komentáře přidávané k projektu

Permission - povolení nutná pro realizaci projektu

ProjectPermission - entita vyjadřující vztah *n:n* mezi projektem a povolením, obsahuje navíc informaci, zdali bylo povolení již vydáno.

User - definuje výčet *UserRole*, určující uživatelskou roli

Contribution - příspěvek ke konkrétnímu nákladu

Výjimky, jimž nevyhovuje identifikátor typu *long*, tudíž nejsou potomky třídy *AbstractEntity*, jsou:

Reaction - reakce uživatele na projekt. Definuje také výčet typů reakcí *ReactionType* (líbí, nelíbí). Tato entita není definována svým identifikátorem, ale dvojicí identifikátorů uživatel-projekt. Jedná se o tzv. slabou entitu, která může být nejvýše jedna pro danou dvojici identifikátorů, což v tomto případě přesně vyhovuje, protože reakce uživatele na daný projekt je pouze jedna.

Token - refresh token, jeho identifikátorem je jeho hodnota (řetězec)

Jednotlivé vztahy a atributy (spolu s jejich typem) tříd jsou znázorněny v diagramu 3.5.

3.6 Třídní diagram

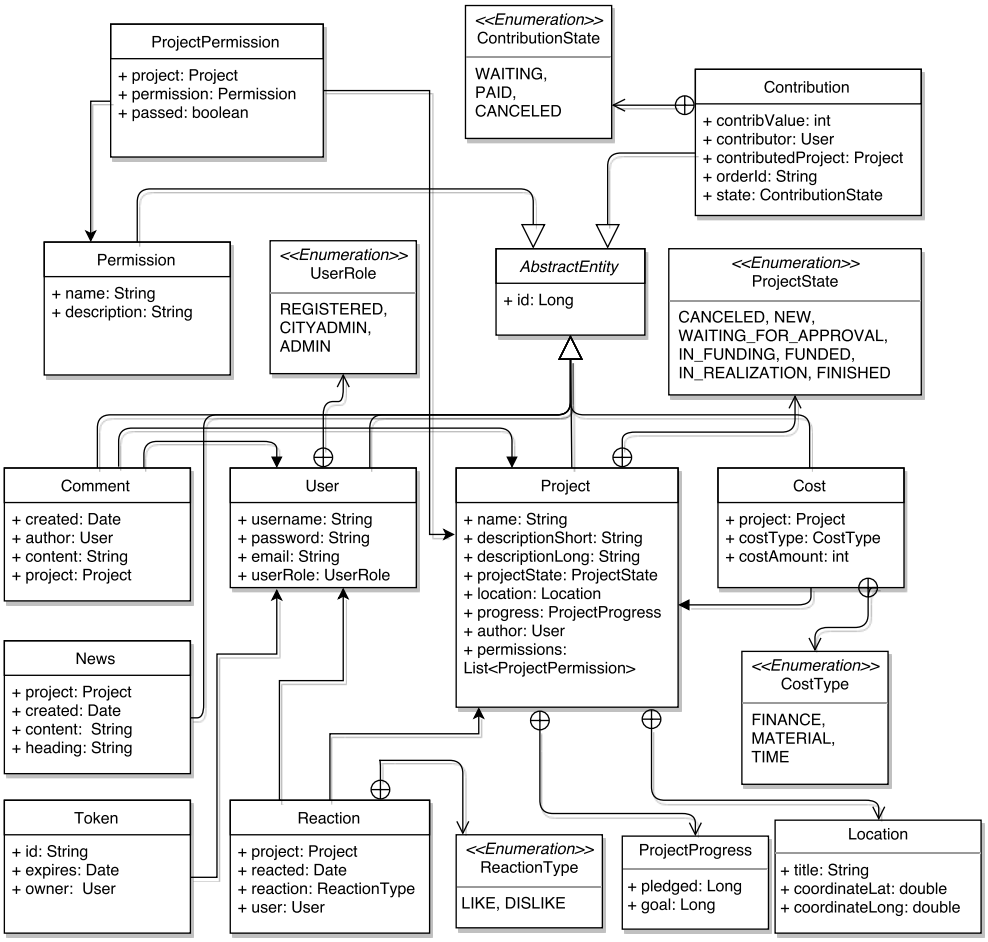
Pokud nepočítáme vrstvu představující datový model aplikace, kterou jsem popsal v předchozí kapitole, skládá se aplikace ze dvou vrstev:

1. Vrstva poskytující REST aplikační rozhraní (API vrstva)

2. Databázová vrstva

Tyto vrstvy jsem pro názornost v diagramu 3.6 rozdělil tlustou vodorovnou čarou. Ve spodní části pod čarou se nacházejí třídy z databázové vrstvy a nad čarou jsou umístěny třídy z API vrstvy. U jednotlivých tříd v diagramu jsem si dovolil vynechat jejich atributy a metody, protože třídy v API vrstvě obsahují metody, které jsem popsal v sekci 3.4.1, a třídy v databázové vrstvě většinou obsahují metody pro získávání, vytváření, mazání a úpravu objektů, které nejsou nijak zajímavé. Výjimkou je pouze třída *AbstractManager*, ve které jsem vypsal jednotlivé metody právě na práci s potomky třídy *AbstractEntity*. U metod této třídy jsem si ale dovolil vynechat jejich parametry.

V následujících sekcích stručně popíši obě zmiňované vrstvy.

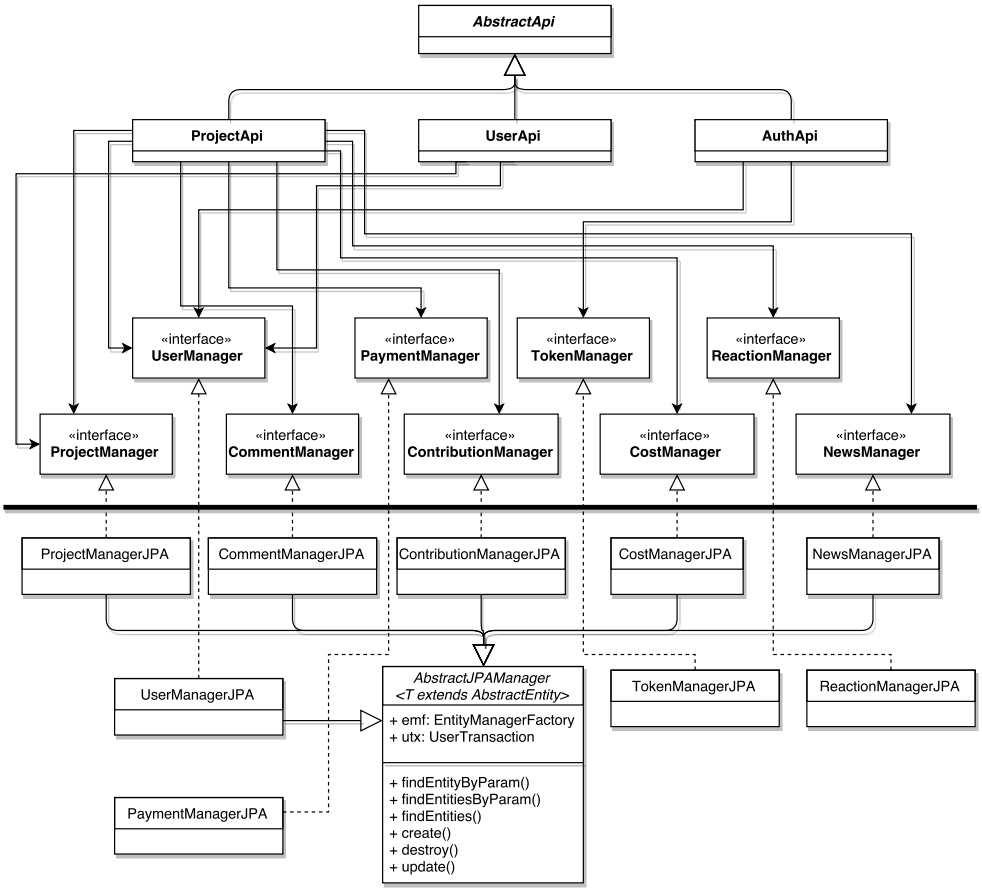


Obrázek 3.5: Datový model

3.6.1 Vrstva aplikačního rozhraní

Tato vrstva obsahuje třídy *ProjectApi*, *UserApi* a *AuthApi*, které poskytují REST rozhraní. Každá třída představuje tu část rozhraní, která je dostupná pod adresou odpovídající první části názvu třídy. V každé metodě těchto tříd je provedena požadovaná autorizace uživatele, zavolání odpovídajících metod z databázové vrstvy a odeslání odpovědi s potřebným HTTP statusem.

Pro operace nad databází jsou v této vrstvě definována rozhraní pro každou modelovou třídu (první část názvu rozhraní odpovídá názvu modelové třídy). Tato rozhraní jsou potom implementována v databázové vrstvě, která tím pádem závisí na API vrstvě (na jejich rozhraních), a nikoliv naopak, jak by tomu bylo v případě použití konkrétních tříd z databázové vrstvy v API vrstvě. Tento princip se nazývá *Dependency inversion* a díky tomu je poté možné jednoduše vyměnit celou implementaci databázové vrstvy, aniž by to mělo vliv na zbytek systému. Konkrétní implementace rozhraní jsou pak v daných



Obrázek 3.6: Třídní diagram

**Api* třídách získávány pomocí *Dependency injection*.

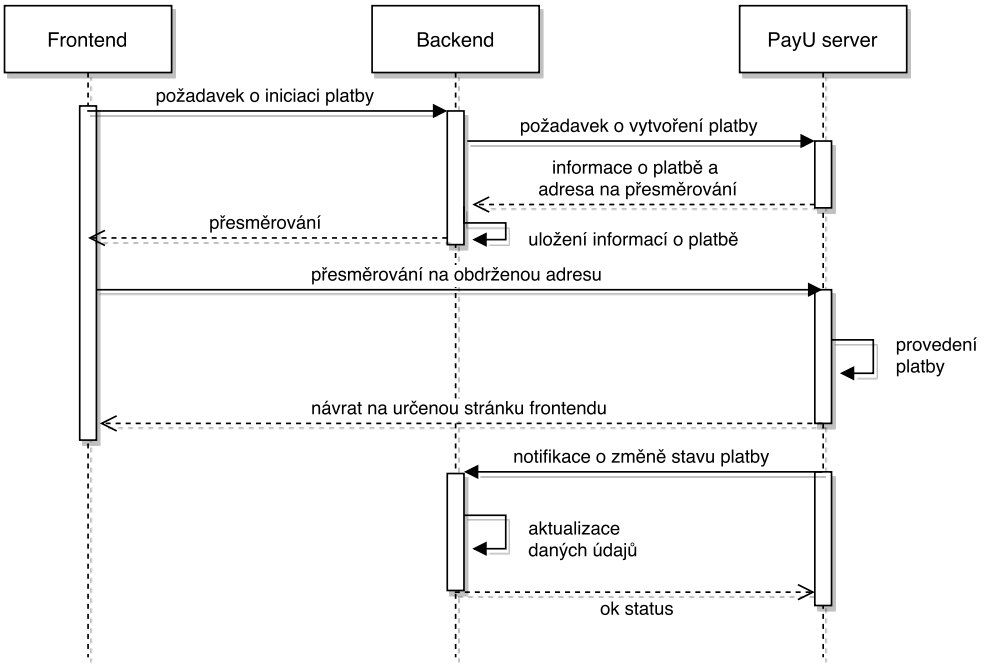
3.6.2 Databázová vrstva

Databázová vrstva obsahuje množinu tříd, které umožňují ukládat a získávat modelové objekty z databáze. Každá třída z této vrstvy implementuje odpovídající rozhraní z API vrstvy. Ty třídy, jež pracují s modelovými objekty, které jsou potomky třídy *AbstractEntity*, jsou potomky třídy *AbstractJPAManager*, v které jsou naimplementovány základní operace s těmito entitami.

Pro práci s databází využívají tyto třídy *Java Persistence Api*, konkrétně implementaci Hibernate.

3.7 Platební brána

Z množství nabízených platebních bran jsem vybral systém PayU, a to zejména z důvodu, že nabízí volné testovací prostředí, a tak je možné bránu jednoduše



Obrázek 3.7: Platební proces v PayU

naimplementovat do systému, aniž by bylo nutné uzavírat jakoukoli smlouvu, či provádět další kroky. Platba přes PayU probíhá z uživatelova hlediska v následujících krocích:

1. Uživatel iniciuje platbu v aplikaci (CityFunder) s danou částkou
2. Uživatel je přesměrován na portál PayU, kde potvrdí danou částku a vybere platební metodu
3. PayU přesměruje uživatele na stránku banky, kde potvrdí danou transakci
4. Uživatel je přesměrován zpět na portál PayU, případně zpět do aplikace

Proces platby z aplikačního hlediska jsem znázornil v sekvenčním diagramu 3.7. Nejdříve je uživatelem z frontendu iniciována platba, která spočívá v odeslání informací backendu. Backend informace zpracuje a vytvoří potřebný požadavek pro PayU server, který mu odešle. Od PayU serveru poté obdrží údaje o vytvořené platbě (identifikátor), které si uloží do databáze jako entitu *Contribution*, a uživateli na frontendu pošle přesměrovací adresu. Frontend po obdržení této adresy zareaguje přesměrováním uživatele (obdrží HTTP požadavek 302). Uživatel poté provede na přesměrované adrese kroky nutné k provedení platby, které jsem popsal v seznamu výše.

O změně stavu platby je pak backend informován požadavkem ze strany PayU. Tento požadavek obsahuje mimo jiné identifikátor platby a stav, do kterého se platba dostala. Díky těmto informacím provede backend potřebnou aktualizaci dat ve své databázi - tedy například u platby, která byla úspěšně provedena, zvýší vybranou částku u daného projektu.

3.7.1 Zabezpečení platby

K provádění požadavků na server PayU je potřebná autentizace, která je v tomto případě implementována podle protokolu OAuth verze 2. Na rozdíl od autentizace klientské javascriptové aplikace, kterou jsem se dlouze zabýval v sekci 3.4.2.2, je autentizace backendu v tomto případě mnohem jednodušší. Backend (aplikace běžící na serveru) je v roli důvěrného klienta, tudíž může obsahovat důvěrné informace (klíče), kterými se může systému PayU autentizovat.

Systém PayU poskytuje klientům tři druhy klíčů:

- POS ID - identifikátor klienta
- OAuthClientSecret - klientské tajemství
- Second key - klíč sloužící k ověřování podpisů

První dva klíče slouží k autentizaci klienta. Jedná se o identický model jako uživatelské jméno/heslo. *POS ID* je veřejná informace (uživatelské jméno) a *OAuthClientSecret* je tajná informace (heslo). Pomocí těchto dvou klíčů získá backend *access token*, který přikládá v hlavičce ke každému požadavku na systém PayU, a tím se dále autentizuje.

Třetí klíč (second key) se používá k ověření integrity notifikační zprávy o změně stavu platby. Server vygeneruje zprávu o platbě (ve formátu JSON), připojí k ní tento klíč a vytvoří z tohoto celku hash. Tento tzv. podpis poté přikládá v hlavičce notifikačního požadavku. Klient si poté stejným způsobem může ověřit, že daná zpráva pochází od ověřeného zdroje a že nebyla po cestě změněna.

Komunikace s PayU samozřejmě probíhá přes protokol HTTPS, který zajišťuje bezpečný přenos dat mezi servery.

Po přesměrování do platebního systému je už bezpečnost zcela na straně PayU a banky, u které platba probíhá. Uživatel si jen musí být vědom, kam zadává údaje a zdali neproběhlo přesměrování na podvodnou stránku (například u nějaké formy útoku na DNS server).

Kapitola 4

Implementace

Tato kapitola popisuje, co bylo v rámci prototypu naimplementováno. Dále také rozebírá implementaci obrany proti základním typům útoků a ukládání uživatelských hesel.

4.1 Rozsah implementace prototypu

V prototypu backendu aplikace Cityfunder, který je součástí přílohy (CD) této diplomové práce, jsem naimplementoval většinu funkcí, které odpovídají funkčním požadavkům vzneseným v kapitole 2.2. Výjimku tvoří pouze několik požadavků.

Prvním z nich je správa povolení nutná k realizaci projektu. V prototypu jsem naimplementoval pouze datovou strukturu těchto povolení. Vynechal jsem jakékoli metody v rozhraní pro práci s těmito povoleními, protože se nám bohužel nepodařilo během tvorby této práce zjistit, jakou formou by bylo případné pilotní město tato povolení ochotno spravovat. Aplikace je ale připravena na implementaci požadovaných funkcí v případě dospecifikování daných požadavků a nasazení v pilotním městě.

Dalším drobným požadavkem, který jsem v diplomové práci neimplementoval, je odesílání emailů ostatním uživatelům portálu. Tento požadavek není moc důležitý, protože email lze při znalosti adresy odeslat i jinými způsoby. Navíc lze i určitým způsobem (pomocí *mailto*) implementovat na frontendu.

4.2 Obrana proti základním typům útoků

Při programování je nutné se vždy řídit pravidlem, že jakýkoli uživatelský vstup je potenciální zlo. Proto je nutné všechny tyto vstupy řádně prověřit a řádně s nimi pracovat. V případě Cityfunderu, jakožto webové aplikace, je třeba dát si pozor zejména na dva typy útoků.

4.2.1 SQL injection

Prvním typem útoku je *SQL* injection, která využívá nevhodného skládání SQL dotazů do databáze. Pokud jsou dotazy skládány pomocí zřetězení SQL kódu a parametrů, které jdou od uživatele (například uživatelské jméno), může být lehce změněn význam těchto dotazů a útočník tak může klidně získat obsah celé databáze. Nebudu zde více popisovat tento typ útoků, protože materiálů na toto téma je opravdu hodně.

Ochrana proti tomuto typu útoků je jednoduchá. Stačí správně zpracovat vstupní parametr, aby nemohl změnit význam SQL dotazu. Zpracování spočívá v tzv. „escapování“ speciálních znaků, čímž dojde ke ztrátě jejich sémantiky. Programátor se většinou o toto escapování nemusí starat, pokud použije vhodnou knihovnu.

V mém případě jsem pro práci s databází používal implementaci specifikace JPA Hibernate, kde se neskládá přímo SQL dotaz, ale dotaz v jazyce HQL (Hibernate Query Language), který je ovšem SQL velmi podobný. Parametry se do tohoto dotazu vkládají pomocí tzv. *placeholderů*, které se vloží do místa, kam má být daný parametr dosazen. Tento placeholder je poté nahrazen escapovanou hodnotou parametru, který se k dotazu přiřazuje. Tímto způsobem je jednoduše zabráněno útokům typu SQL injection.

4.2.2 Cross site scripting (XSS)

Druhým typem útoku, na který je nutné si dávat pozor, je *Cross site scripting*. Jedná se o útočníkem vložený javascriptový kód do stránky (do databáze). K tomuto vložení může dojít kdekoliv, kde není řádně zpracován vstup a návštěvník vytváří obsah stránky (například formou komentáře ke článku). Škodlivý kód je poté vykonáván u každého návštěvníka webu, u kterého dojde k jeho načtení, a je tak možné například ukradnout uživatelovo cookie. V tomto případě sice nemůže dojít k ukradení cookie, vzhledem k tomu, že jsem použil *HttpOnly cookies* (viz kapitola 3.4.2.2), ale může dojít k jiným nepříjemným typům útoků.

Ochranou proti tomuto typu útoků je, podobně jako v předchozím případě, vhodné zpracování (escapování) uživatelského vstupu. Toto escapování jsem opět neimplementoval sám, ale využil jsem knihovnu *StringEscapeUtils* z balíčku *org.apache.commons.lang*. Escapovaní provádím v každém setteru atributu typu *String* všech entitních tříd. Aby nedocházelo k escapování již escapovaných znaků (například < na <), provádím před escapováním ještě opačnou operaci, která vstupní text sjednotí na neescapovaný, a escapování provádím až s tímto řetězcem. Tímto způsobem dochází k ukládání již escapovaného obsahu do databáze. Data předaná aplikačním rozhraním poté mohou být přímo (bez dalšího zpracování) a bezpečně zobrazena na frontendu.

Tento přístup ovšem přináší i jedno úskalí, kterým je nemožnost zobrazení formátovaného HTML kódu. Například u popisu projektu je žádoucí,

aby byl text určitým způsobem formátován - alespoň odstavci, ale vhodné by bylo i základní formátování pomocí tučného nebo podtrženého textu. Jakékoli HTML značky ovšem budou v procesu ukládání escapovány, tudíž neposlouží na frontendu k formátování, ale zobrazí se jako normální text.

Tento problém bude vhodné řešit až na frontendu přepsáním daných formátovacích značek zpět do jejich původního tvaru či definováním vlastních formátovacích značek, které nebudou escapovány.

4.3 Ukládání uživatelských hesel

Ukládání uživatelských hesel je jeden z problémů, který je nutné řešit v každém systému, který s těmito daty pracuje. Uživatelská hesla jsou tajné informace, které mají specifické vlastnosti - často se opakují, a to nejen tak, že může mít více uživatelů stejné heslo v rámci jednoho systému, ale také, že daný uživatel používá stejné heslo napříč několika různými systémy. Získání daného hesla útočníkem tak může mít nejen dopad na daný systém, ale také na další, které uživatel používá. Z těchto důvodů je nutné uživatelská hesla adekvátně chránit, a to takovým způsobem, že je může znát pouze daný uživatel.

Běžnou praxí je ukládat tzv. hashe hesel, což je výstup určité hashovací funkce, které bylo dáno heslo jako vstupní parametr. Hlavní vlastností této hashe je, že neexistuje algoritmus, který by dokázal z dané hashe vypočítat původní vstup. Důvodem je zejména to, že existuje mnoho (až nekonečno) vstupů, pro který daná funkce vrací danou hash. Hashování se také často posiluje solí, což je náhodný, ale pro dané heslo pevný řetězec, který se připojuje k heslu před jeho hashováním. Díky této soli se poté stejná hesla ukládají v databázi různě, a tak i když útočník například zjistí hash a hodnotu hesla u určitého uživatele, tak nemůže zjistit, že nějaký jiný uživatel má stejné heslo pouhým porovnáním hashí těchto hesel.

Celá problematika hashování (nejen hesel) je mnohem složitější, než by se mohlo zdát a vydala by klidně na celou diplomovou práci, a tak ji nebudu dále více rozebírat. Z programátorského hlediska je důležité si uvědomit zejména to, že není vhodné vymýšlet vlastní hashovací algoritmy, či špatně používat ty stávající. Nejlepším způsobem jak provádět hashování hesel je proto využít kvalitní knihovnu, která tuto funkci poskytuje.

Při implementaci Cityfunderu jsem využil algoritmus *BCrypt*, který je v současnosti považován za jeden z nejlepších algoritmů pro hashování hesel. Bcrypt je funkce, jejíž složitost je exponenciálně závislá na jejím parametru (work factor). Výhodou tohoto parametru je, že se se vzrůstající rychlostí počítačů může zvyšovat parametr, aby výpočet zůstal relativně stejně náročný. Delší výpočet hashe je žádoucí kvůli útokům hrubou silou, při nichž může docházet k vyzkoušení výrazně menší množiny řetězců [21].

Hodnotu parametru jsem během vývoje nechal na původní hodnotě (10). Jeho nastavení má smysl až na produkčním serveru, protože jeho výkon je

rozdílný od mého počítače. Nastavení parametru se bude odvíjet od očekávané návštěvnosti aplikace a od maximálně přípustné doby výpočtu hesla. Je pravděpodobné, že přihlašování nebo registrace uživatele bude v Cityfunderu (při nasazení v jednom městě) probíhat pouze několikrát za hodinu, tudíž je možné si dovolit nastavit parametr tak, aby výpočet hesla trval přibližně jednu sekundu.

Kapitola 5

Testování

Tato kapitola v úvodu stručně popisuje rozdělení testů podle jejich typu. Dále se věnuje testování, které bylo při vývoji Cityfunderu prováděno a obsahuje jednotlivé testovací scénáře.

5.1 Typy testování

K vývoji kvalitní aplikace bezesporu patří testování. Existuje několik druhů testů, které je možné během vývoje provádět a které se vztahují nejen k tomu, co chceme testovat, ale také k fázi, ve které se projekt nachází.

Testy lze obecně rozdělit na ruční a automatizované, přičemž ruční testy jsou prováděny člověkem, a to buď testerem, či samotným programátorem a spočívají například v „proklikávání“ aplikací. Automatizované testy jsou pak většinou jiné programy, které ověřují funkčnost testovaného programu.

Testy se také dělí na základě toho, jakou část programu testují. Základními testy, které ověřují kvalitu samotných programových jednotek (například jednotlivých metod dané třídy), se nazývají *Unit testy*. Další úrovní testování jsou pak integrační testy, které ověřují vzájemnou komunikaci mezi aplikačními komponentami. Dalším druh testování představují systémové testy, které ověřují danou aplikaci jako celek. Tyto testy mohou být, z pochopitelných důvodů, prováděny spíše až v pozdějších fázích vývoje softwaru. Jistou formou systémových testů jsou pak akceptační testy, které jsou prováděny zákazníkem při nasazování aplikace [22].

5.2 Testování při vývoji Cityfunderu

Při vývoji backendu Cityfunderu jsem prováděl zejména automatizované testy, které ověřovaly jednotlivé metody aplikačního rozhraní. Z pohledu backendu se tedy jedná o systémové testy, které ověřují backend jako celek. Z pohledu celé aplikace mají tyto testy ale spíše blíž k unit testům.

K implementaci testů jsem využil framework *JUnit*, který se nejčastěji využívá právě pro vytváření unit testů, ovšem hodí se i pro testování aplikačního rozhraní. Každý test spočíval ve vytvoření určitého HTTP požadavku, který byl poté odeslán na danou URI. Test potom ověřoval, zdali byla odpověď serveru (např. stavový kód) taková, jaká má být. Dalším specifikem testů, které jsem vytvářel, bylo, že je nutné je provádět v určitém pořadí, protože představovaly určitý scénář použití aplikace. Na toto není framework JUnit moc dobře uzpůsoben, protože u unit testů většinou nehraje pořadí vykonání roli. Existuje naštěstí anotace testu *@FixMethodOrder(MethodSorters.NAME_ASCENDING)*, která umožňuje vykonávání testů v jejich abecedním pořadí. Pořadí testů jsem tedy ovlivňoval názvem metod, které jednotlivé testy představují.

Usilí vložené do vytvoření testů se určitě vyplatilo, protože mi tyto testy pomáhaly rychle a jednoduše zjistit, zdali backend funguje správně, když jsem prováděl nějaké změny kódu. V následujících sekcích popíši jednotlivé scénáře, kterými jsem prováděl testování.

5.2.1 Testování autentizace

Testování autentizace ověřuje základní funkce autentizační služby jako je přihlášení, obnovení přístupového tokenu a odhlášení uživatele.

1. Přihlášení uživatele s platnými autentizačními údaji a uložení tokenů
2. Obnovení přístupového tokenu - odeslání obnovovacího tokenu a získání nového přístupového tokenu (ověření platnosti obnovovacího tokenu získaného v předchozím kroku)
3. Odhlášení uživatele
4. Ujištění, že byl uživatel odhlášen (že byla smazána cookies)
5. Obnovení přístupového tokenu se starým obnovovacím tokenem - ověřuje, že byl token po odhlášení smazán z databáze
6. Přihlášení uživatele s neplatnými údaji
7. Obnovení přístupového tokenu s neplatným obnovovacím tokenem

5.2.2 Testování uživatelů

Testování uživatelů testuje registraci a následné možné operace s účtem jako je přihlášení, změna údajů a hesla.

1. Registrace uživatele (pokud není uvedeno jinak, další testy pracují s tímto uživatelem)

2. Registrace znovu stejného uživatele - ověření kontroly jedinečnosti uživatelského emailu
3. Získání informací o uživateli
4. Přihlášení uživatele
5. Úprava informací o uživateli
6. Kontrola, že změna informací o uživateli proběhla v pořádku
7. Změna hesla uživatele
8. Opětovná změna hesla uživatele - stejný požadavek jako v předchozím kroku, musí být zamítnut, pokud změna hesla proběhla v pořádku (protože se odesílá staré a nové heslo)
9. Získání informací o uživateli přes `users/current`

5.2.3 Testování projektů

Tento testovací scénář ověřuje všechny operace s projektem a se souvisejícími entitami, které jsem v Cityfunderu implementoval. Na začátku testu je přihlášen testovací uživatel, protože k většině operací s projektem je nutné přihlášení.

1. Vytvoření projektu (pokud není uvedeno jinak, další testy pracují s tímto projektem)
2. Získání projektu autorem (uživatelem, jenž ho vytvořil)
3. Vytvoření projektu bez přihlášení uživatele (bez tokenů) - musí být zamítnuto
4. Získání vytvořeného projektu bez přihlášení - projekt nesmí být viditelný
5. Úprava informací o projektu
6. Kontrola aktualizovaných údajů
7. Předání projektu ke schválení administrátorem
8. Kontrola změny stavu projektu
9. Vrácení projektu do fáze NEW administrátorem
10. Přidání nákladu k projektu
11. Přidání nákladu k cizímu projektu (musí být zavrženo)
12. Získání nákladu k projektu vytvořeného v kroku 5

13. Aktualizace údajů nákladu z předchozího kroku
14. Nalezení změněného nákladu (ověření, že byly upraveny údaje)
15. Odstranění nákladu z předchozích kroků
16. Vložení dvou finančních nákladů k projektu a ověření hodnoty projektové cílové částky
17. Vložení komentáře k projektu
18. Nalezení vloženého komentáře
19. Aktualizace vloženého komentáře
20. Ověření, zda byl komentář upraven
21. Odstranění komentáře
22. Získání odstraněného komentáře (nesmí být nalezen)
23. Vložení komentáře uživatelem a smazání komentáře administrátorem
24. Ověření, že byl komentář opravdu smazán
25. Schválení projektu administrátorem (posun do fáze IN FUNDING)
26. Kontrola, že projekt je nyní veřejný
27. Kontrola, že projekt ve fázi IN FUNDING nelze upravovat
28. Vložení novinky k projektu
29. Získání vložené novinky
30. Přispění finanční částky na projekt
31. Vyvolání umělé notifikace od platebního systému o zaplacení přispěné částky v předchozím kroce
32. Kontrola, že u projektu byla navýšena hodnota vybraných peněz
33. Vložení reakce (like) k projektu
34. Kontrola, zdali je počet like u projektu roven 1

Kapitola 6

Dokumentace

V této kapitole zdokumentuji základní postupy při komunikaci s aplikačním rozhraním, které jsem v rámci této práce navrhnul a naimplementoval. V rámci této dokumentace popisuji zejména datové struktury reprezentací jednotlivých entit a jak s nimi pracovat. Pokud jsou požadavky na rozhraní nějakým způsobem specifické, rozvádím je podrobněji. Kompletní dokumentaci k rozhraní ve formátu *Blueprint*, používaným na portálu *Apiary*, lze nalézt na přiloženém CD (viz B).

6.1 Základní pravidla komunikace

Ukázkové požadavky budou často ve formě vzoru, jenž obsahuje parametry, které je třeba nahradit adekvátními hodnotami, tyto parametry jsou v kódu ohraničeny dvěma hvězdičkami, například: **PARAMETR**.

6.1.1 Content type

Všechny metody aplikačního rozhraní, které přijímají/odesílají data v těle HTTP požadavku/odpovědi, pracují s formátem *JSON*. V hlavičce požadavku tedy musí být vždy uveden parametr *Content-type: application/json* jako například v následujícím vzoru:

```
*HTTP METODA* *ADRESA PROSTŘEDKU* *HTTP VERZE*
Host: *ADRESA HOSTITELE*
Content-Type: application/json
*další parametry*

*tělo požadavku*
```

6.1.2 Cookies

Často je k vyvolání daného požadavku rozhraní nutná autentizace uživatele, která je implementována pomocí tokenů uložených v cookies. Ačkoli jsem volbou HttpOnly cookies ponechal jejich správu zcela na internetovém prohlížeči, uvedu zde pro případ testování či ladění klientské části, jakým způsobem je nutné přikládat cookie k požadavku. Parametr *Cookie* v hlavičce požadavku může obsahovat přístupový token v následujícím tvaru:


```
Cookie: accessTokenCookie=eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIjoiUkVHS
VNURVJFRCIsImlzcyI6ImNpdHlmdW5kZXItYmFja2VuZGFwcCIsImV4cCI6MTQ5M
jY5NDE0MSwidXNlcmlkIjoiMSIsInVzZXJuYW1lIjoibmlja9.Bu2G7mJlmrE0i9
jT04ml6YGAvUqazcCMukd_CZ8XyyY;
```


Nebo pro požadavek obnovení přístupového tokenu může obsahovat obnovovací token:


```
Cookie: refreshTokenCookie=gzQuqN36kAod02gBUPMe7N2HTUlzANxGx7G73
XDk;
```


6.1.3 Reprezentace entit

Základem komunikace je předávání reprezentací entit ve formátu JSON. Pro různé typy požadavků se liší, jaké informace jsou v reprezentaci předávány. Každý prostředek obsahuje ve své reprezentaci identifikátor *id*. Ten je prostředku automaticky přiřazen při jeho vytvoření, tudíž se u tohoto typu požadavku (POST požadavek) identifikátor v reprezentaci neuvádí. Stejně tomu tak je u úpravy prostředku (PATCH požadavek), kdy je identifikátor dán v adrese. Obecně se tedy dá říct, že pokud posíláme na server reprezentaci nějakého prostředku, identifikátor se neuvádí. Při úpravě daného prostředku se navíc posílají pouze ty informace, které mají být změněny. Tedy například, pokud chceme změnit název projektu, posíláme v těle požadavku pouze toto:

```
{
    "name": *UPRAVENY NAZEV PROJEKTU*
}
```

Většina entit obsahuje ve své reprezentaci vrácené serverem pole *links*, které obsahuje hypertextové odkazy. Tyto odkazy nejsou nikdy posílány na server, ani ukládány do databáze, slouží čistě pro klienta. Každý odkaz má pole *rel*, které vyjadřuje vztah k odkazovanému prostředku, a dále pole *href*, které obsahuje relativní URL odkazovaného prostředku vůči kořenu rozhraní. Každá entita v tomto poli obsahuje alespoň odkaz na sebe samou, který může vypadat v případě projektu následovně:

```
[
    {
```

```
    "rel":"self",
    "href":"/projects/61"
    }
]
```

6.2 Kořen rozhraní

Kořen rozhraní na adrese / nabízí s metodou *GET* základní rozcestník rozhraním.

```
{
    "name":"CityFunder API",
    "links":[
        {"rel":"users","href":"/users"},
        {"rel":"projects","href":"/projects"},
        {"rel":"authentication","href":"/auth"}
        ],
    "version":"1.0"
}
```

6.3 Autentizace

Autentizace je základním kamenem celého aplikačního rozhraní a je nutná u většiny metod. Jak přikládat k požadavkům získané cookie jsem popsal v předchozí části. V této části stručně popíši, jak se volají autentizační metody přihlášení, obnovení přístupového tokenu a odhlášení.

Prostředky a metody pro autentizaci jsou následující:

- `/auth/login` (POST) - přihlášení uživatele
- `/auth/refresh` (POST) - obnovení přístupového tokenu
- `/auth/logout` (POST) - odhlášení uživatele

6.3.1 Přihlášení

Požadavek o přihlášení obsahuje email a heslo a vypadá v základní (minimální) formě následujícím způsobem:

```
POST *ADRESA API*/auth/login HTTP/1.1
Host: *ADRESA HOSTITELE*
Content-Type: application/json
Content-Length: 35

{"email":"nick","password":"psswd"}
```

Odpověď (se statusem *OK*) v případě úspěšné autentizace uživatele obsahuje hlavičky *Set-Cookie*, pomocí nichž jsou klientovi předány nově vytvořené cookies. V těle odpovědi jsou přiloženy informace o právě autentizovaném uživateli, které může klient ihned využít:

```
{
    "id":1,
    "links":[{"rel":"self","href":"users/1"}],
    "name":"nick",
    "email":"tajny@email.cz",
    "userRole":"REGISTERED"
}
```

6.3.2 Obnovení tokenu a odhlášení

Požadavek o obnovovení přístupového tokenu obsahuje v hlavičce obnovovací token, tudíž vypadá například následovně:

```
POST *ADRESA API*/auth/refresh HTTP/1.1
Host: *ADRESA HOSTITELE*
Cookie: refreshTokenCookie=*REFRESH TOKEN*
```

Odpověd serveru je pak podobná jako při přihlášení, protože obsahuje nová cookie, která nahrazují ta stará. V těle odpovědi ovšem nejsou žádná data, proto je stavový kód odpovědi *NO_CONTENT*.

Požadavek o odhlášení uživatele je v podstatě totožný s tím o obnovení tokenu, pouze směřuje na adresu `/auth/logout`. V odpovědi jsou poté odeslána neplatná cookie, která přepíšou ta stávající.

6.4 Uživatelé

Uživatelské prostředky a podporované metody jsou:

- `/users` (POST)
- `/users/{user_id}` (GET, PATCH)
- `/users/{user_id}/password` (PUT)
- `/users/current` (GET)
- `/users/{user_id}/projects` (GET)

Reprezentace uživatelské entity má následující tvar (s ukázkovými hodnotami):

```
{
    "id":1,
    "name":"Happy User",
    "password": "passwordhash"
    "email":"happyuser@email.xyz",
    "userRole":"REGISTERED",
    "links":[{"rel":"self","href":"users/1"}]
}
```

Pole s heslem *password* je ve všech požadavcích skryto a posílá se pouze v případě registrace uživatele. Jediným atributem uživatele, který lze měnit, je jméno. Je ale pravděpodobné, že se v budoucnu tato entita rozšíří i o další atributy.

6.4.1 Změna hesla

Změna hesla je specifický požadavek, který ve svém těle obsahuje dva parametry:

```
{
    "oldPassword": *STARE HESLO*,
    "newPassword": *NOVE HESLO*
}
```

6.5 Projekty

Projektové prostředky a podporované metody jsou:

- `/projects` (GET, POST)
- `/projects/{project_id}` (GET, PATCH)

Reprezentace projektové entity předávaná serverem má tvar (s ukázkovými hodnotami):

```
 {
    "id":61,
    "name":"test project",
    "descriptionShort":"short description",
    "descriptionLong":"long description",
    "projectState":"IN_FUNDING",
    "progress":
        {
            "pledged": 39108,
            "goal": 150000
        },
```

```
    "location":
        {
            "title":"Test Location",
            "coordinateLat":50.0,
            "coordinateLong":50.0
        },
    "author":
        {
            "id":1,
            "name":"nick",
            "email":"tajny@email.cz",
            "userRole":"REGISTERED",
            "links": [{"rel":"self","href":"/users/1"}]
        }
    "links": [
        {"rel":"self","href":"/projects/61"},
        {"rel":"comments","href":"/projects/61/comments"},
        {"rel":"costs","href":"/projects/61/costs"},
        {   "rel":"contributions",
            "href":"/projects/61/contributions"
        },
        {"rel":"reactions","href":"/projects/61/reactions"},
        {"rel":"news","href":"/projects/61/news"}
    ]
}
```

Atribut *progress* představuje průběh crowdfundingu projektu (cílová a vybraná částka) a je vypočítáván serverem. Součástí je také reprezentace autora projektu. Tyto dva atributy (společně s *id*) tedy nelze měnit. Ostatní údaje je možné měnit metodou *PATCH*. Autor projektu je autorizován měnit stav projektu pouze ze stavu *NEW* do *FOR_APPROVAL*.

Pole links obsahuje odkazy na všechny související entity, kterými jsou komentáře, náklady, příspěvky, reakce a novinky.

6.6 Příspěvky na projekt

Prostředky projektových příspěvků a podporované metody jsou:

- `/projects/{p_id}/contributions` (GET, POST)
- `/projects/{p_id}/contributions/{contribution_id}` (GET)

Reprezentace entity příspěvku na projekt (s ukázkovými hodnotami) má následující tvar:

```
{
    "id":48,
    "links": [
        {"rel":"self","href":"/projects/43/contributions/48"}
        ],
    "contribValue":1000,
    "contributed":"2017-04-23T09:52:47Z"
    "contributor":
        {
        "id":1,
        "links":[{"rel":"self","href":"users/1"}],
        "name":"nick",
        "email":"tajny@email.cz",
        "userRole":"REGISTERED"},
}
```


Při posílání požadavku o přispění na projekt se posílá pouze hodnota *contributedValue*.

6.7 Komentáře

Prostředky komentářů a podporované metody jsou:

- `/projects/{p_id}/comments` (GET, POST)
- `/projects/{p_id}/comments/{comment_id}` (GET, PATCH, DELETE)

Reprezentace entity komentáře k projektu (s ukázkovými hodnotami) má následující tvar:


```
{
    "id":1,
    "links":
        [{"rel":"self","href":"/projects/18/comments/1"}],
    "created":"2017-04-23T09:52:47Z",
    "author":
        {
        "id":1,
        "links":[{"rel":"self","href":"users/1"}],
        "name":"nick",
        "email":"tajny@email.cz",
        "userRole":"REGISTERED"
        },
    "content":"comment content"
}
```

V požadavku o vložení/úpravu komentáře se posílá pouze obsah komentáře v poli *content*.

6.8 Novinky

Prostředky projektových novinek a podporované metody jsou:

- `/projects/{p_id}/news` (GET, POST)
- `/projects/{p_id}/news/{news_id}` (GET)

Reprezentace entity novinky u projektu (s ukázkovými hodnotami) má následující tvar:

```
{
    "id":24,
    "links":[{"rel":"self","href":"/projects/18/news/24"}],
    "created":"2017-04-23T09:52:47Z",
    "content":"test news content",
    "heading":"test news heading"
}
```

V požadavku o vložení novinky se posílá pouze obsah a nadpis novinky v polích *content* a *heading*.

6.9 Reakce na projekt

Prostředky reakcí na projekt a podporované metody jsou:

- `/projects/{p_id}/reactions` (POST, GET)

Reprezentace entity reakce na projekt (s ukázkovými hodnotami) má následující tvar:

```
{
    "user":
        {
        "id":1,
        "links":[{"rel":"self","href":"users/1"}],
        "name":"nick",
        "email":"tajny@email.cz",
        "userRole":"REGISTERED"
        },
    "reaction":"like",
    "reacted":"2017-04-23T09:52:47Z"
}
```

V požadavku o vložení/úpravu se posílá pouze typ reakce v poli *reaction*.

Pro získání určitého typu reakce se do adresy požadavku přidává parametr *type* s hodnotou požadovaného typu reakce. Například:

```
/projects/{project\_id}/reactions?type=likes
```

Pro získání pouze počtu určitých reakcí se do adresy ještě navíc přidává parametr *count* bez hodnoty. Například:

```
/projects/{project\_id}/reactions?type=likes&count
```

6.10 Projektové náklady

Prostředky projektových nákladů a podporované metody jsou:

- `/projects/{p_id}/costs` (GET, POST)
- `/projects/{p_id}/costs/{cost_id}` (GET, PATCH, DELETE)

Reprezentace entity projektového nákladu (s ukázkovými hodnotami) má následující tvar:

```
{
    "id":20,
    "links":[{"rel":"self","href":"/projects/18/costs/20"}],
    "costType":"FINANCE",
    "costAmount":10,
    "title":"projektovy naklad",
    "description":"popis projektoveho nakladu"
}
```

Závěr

Výstupem této diplomové práce je funkční prototyp backendu crowdfundingové platformy Cityfunder. V rámci prototypu se podařilo naimplementovat klíčové funkce tohoto portálu a připravit aplikaci na dodělání dalších, jako je například správa povolení projektu. K finálnímu produktu je ovšem ještě poměrně dlouhá cesta, během které bude nutné doplnit systém o další prvky. Jedním z nich je například nahrávání obrázků, které k tomuto typu aplikace neodmyslitelně patří. Dále také bude potřeba uzavřít smlouvu s vybraným poskytovatelem platebního systému (prototyp je připraven na PayU), aby mohly probíhat reálné platby. Úplně nejdůležitější podmínkou k dokončení projektu je navázání spolupráce s pilotním městem (obcí). Bez této spolupráce pozbývá totiž tento projekt smysl.

Při práci na tomto prototypu vznikly také další otázky ohledně stávajících požadavků. Jedná se zejména o vztah finančních a nefinančních nákladů projektu. V prototypu jsou implementovány zcela nezávisle, ovšem v situaci, kdy se nepodaří získat daný nefinanční prostředek, bude autor potřebovat finance, aby ho mohl získat a projekt dokončit. Nabízí se tedy možnost přičítat odhad nefinančních nákladů k těm finančním a v případě získání nefinančního příspěvku snižovat o danou částku celkový finanční náklad.

V nejbližší době je plánováno nasazení backendové části na veřejný hosting (pravděpodobně Openshift) a vyzkoušení spolupráce backend-frontend, která do této doby probíhala pouze přes server *Apiary*, který tvořil frontendu tzv. *mock*[5] backendu. Během tohoto testování se dozajista naleznou nedostatky na obou stranách, které bude nutné odstranit.

Při tvorbě této diplomové práce jsem získal mnoho cenných zkušeností a znalostí. Při návrhu aplikačního rozhraní jsem dopodrobna porozuměl problémům spojeným se zabezpečením komunikace s veřejným klientem, ale také pochopil principy návrhu rozhraní postaveném na architektuře REST. Další, nepochybně cennou, zkušeností byla analýza a návrh propojení s platebním

[5] Mock je anglické slovo, které v tomto kontextu znamená imitaci, napodobení.

systémem. Mnoho praktických zkušeností jsem také získal samotnou implementací prototypu v Javě Enterprise Edition. Velmi přínosnou částí byla také příprava a analýza projektu, která předcházela této diplomové práci.

Literatura

[1] Spacehive: FAQs - Your questions answered [online]. Dostupné z: https://about.spacehive.com/faqs/, [cit. 13.2.2017].

[2] Everfund: Everfund obecně [online]. Dostupné z: http://everfund.cz/cs/napoveda/obecne, [cit. 10.2.2017].

[3] Europe, A. C.: Co je „participativní rozpočet“? [online]. Dostupné z: http://www.participativni-rozpocet.cz, [cit. 13.2.2017].

[4] GmbH., Q.-S. D. G.: Usage of server-side programming languages for websites [online]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all, [cit. 15.2.2017].

[5] O'Dell, J.: 8 Experts Break Down the Pros and Cons of Coding With PHP [online]. Dostupné z: http://mashable.com/2010/11/19/pros-cons-php/, [cit. 20.2.2017].

[6] Source, F. O.: What is HHVM? [online]. Dostupné z: http://hhvm.com/, [cit. 21.2.2017].

[7] Oracle: Your First Cup [online]. Dostupné z: http://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html, [cit. 23.2.2017].

[8] RedHat: Hibernate ORM [online]. Dostupné z: http://hibernate.org/orm/, [cit. 23.2.2017].

[9] Oracle: Java API for RESTful Services (JAX-RS) [online]. Dostupné z: https://jax-rs-spec.java.net/, [cit. 24.2.2017].

[10] Oracle: A Relational Database Overview [online]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html, [cit. 24.2.2017].

[11] Zicari, P. D.-I. R. V.: Introduction to ODBMS [online]. Dostupné z: http://www.odbms.org/introduction-to-odbms/definition/, [cit. 24.2.2017].

[12] Vodička, J.: Velké srovnání platebních bran [online]. Dostupné z: https://blog.gopay.com/velke-srovnani-platebnich-bran, [cit. 30.3.2017].

[13] Neuveden: Specifikace formátu [online]. Dostupné z: http://qr-platba.cz/pro-vyvojare/specifikace-formatu, [cit. 30.3.2017].

[14] Tornová, M. J.: PSD 2 – regulace platebních služeb v novém kabátě [online]. Dostupné z: https://www.epravo.cz/top/clanky/psd-2-regulace-platebnich-sluzeb-v-novem-kabate-100956.html, [cit. 30.3.2017].

[15] Fielding, R. T.: Representational State Transfer (REST). Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.html, [cit. 12.4.2017].

[16] W3C: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Dostupné z: https://www.w3.org/TR/soap12/, [cit. 13.4.2017].

[17] Roy T. Fielding, J. F. R.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Dostupné z: https://tools.ietf.org/html/rfc7231, [cit. 14.4.2017].

[18] Neuveden: Introduction to JSON Web Tokens. Dostupné z: https://jwt.io/introduction, [cit. 8.3.2017].

[19] Neuveden: HttpOnly. Dostupné z: https://www.owasp.org/index.php/HttpOnly, [cit. 9.3.2017].

[20] Rescorla, E.: HTTP over TLS. Dostupné z: https://tools.ietf.org/html/rfc2818, [cit. 9.3.2017].

[21] Neuveden: jBCrypt. Dostupné z: http://www.mindrot.org/projects/jBCrypt/, [cit. 9.4.2017].

[22] Hlava, T.: Fáze a úrovně provádění testů. Dostupné z: http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu/, [cit. 12.4.2017].

Příloha **A**

Seznam použitých zkratek

API Application programming interface

AT Access token

CRUD Create, recieve, update, delete

EE Enterprise edition

HATEOAS Hypermedia as the engine of application state

HQL Hibernate query language

HTTP Hypertext transfer protocol

JAX-RS Java API for RESTful services

JPA Java persistent API

JSON Javascript object notation

JWT JSON web token

ORM Objektově-relační mapování

PHP Hypertext preprocessor

REST Representational state transfer

RT Refresh token

SOAP Simple object access protocol

SQL Structured query language

URI Uniform resource identifier

URL Uniform resource locator

XML Extensible markup language

XSS Cross-site scripting

Příloha **B**

Obsah přiloženého CD

```
readme.txt ..................................stručný popis obsahu CD
doc ....................................... adresář s dokumentací k API
  api_blueprint.txt........dokumentace k API ve formátu Blueprint
src
  impl ..................................zdrojové kódy implementace
  thesis ...................... zdrojová forma práce ve formátu LaTeX
text ..................................................... text práce
  thesis.pdf ........................... text práce ve formátu PDF
  assignment.pdf ....................................zadání práce
```