

# Návrh architektury a vývoj využitelné decentralizované aplikace

František Hromek

---

Bakalářská práce  
2022



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

**\*\*\*Do tištěné verze zde vložte oficiální zadání práce, do PDF verze, která se nahrává do IS/STAG vložte zadání bez podpisů!\*\*\***

### **Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

### **Prohlašuji,**

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

.....  
podpis studenta

## **ABSTRAKT**

Tato práce se věnuje decentralizovaným aplikacím a jejich použití a technologiím používaných při jejich vývoji. V práci popisuji technologii blockchainu, smart kontrakty a decentralizované aplikace, princip jejich funkce těchto aplikací, jejich vlastnosti, případy užití a jednotlivé platformy a technologie které se používají k jejich provozu. Zároveň byla vyvinutá decentralizovaná aplikace zabývající se tématem z oblasti crowdfundingu jako ověření konceptu implementování technologií používaným v tomto oblasti vývoje.

Klíčová slova: blockchain, decentralizované aplikace, Ethereum, smart kontrakty, Solidity

## **ABSTRACT**

This thesis deals with decentralized applications and their use and the technologies used in their development. I describe blockchain technology, smart contracts and decentralized applications, the principle of function of these applications, their properties, use cases and individual platforms and technologies that are used for their operation. At the same time, a decentralized application dealing with the topic of crowdfunding was developed as a verification of the concept of implementing technologies used in this area of development.

Keywords: blockchain, decentralized applications, Ethereum, smart contracts, Solidity

Poděkování, motto a čestné prohlášení, že odevzdaná verze bakalářské práce a verze elektronická, nahraná do IS/STAG jsou totožné ve znění:

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH

<b>ÚVOD.....</b>	<b>8</b>
<b>I TEORETICKÁ ČÁST.....</b>	<b>9</b>
<b>1 BLOCKCHAIN .....</b>	<b>10</b>
1.1 PRINCIP FUNKCE.....	10
1.2 ALGORITMY KONSENSU SÍTĚ .....	12
1.2.1 Proof of Work (PoW).....	12
1.2.2 Proof of Stake (PoS).....	13
<b>2 SMART CONTRACTS (CHYTRÉ SMLOUVY) .....</b>	<b>14</b>
2.1 HISTORIE .....	14
2.2 VLASTNOSTI SMART KONTRAKTŮ.....	14
2.3 ORACLES .....	15
2.4 IMPLEMENTACE NA PLATFORMĚ ETHEREUM .....	16
<b>3 DECENTRALIZOVANÉ APLIKACE .....</b>	<b>17</b>
3.1 HISTORIE VÝVOJE WEBU .....	17
3.2 CHARAKTERISTIKA DECENTRALIZOVANÝCH APLIKACÍ.....	17
3.3 VLASTNOSTI DECENTRALIZOVANÝCH APLIKACÍ.....	18
3.4 TYPY ARCHITEKTURY A MÍRA DECENTRALIZACE.....	19
3.5 KRYPTOMĚNOVÉ PENĚŽENKY .....	22
<b>4 PLATFORMY .....</b>	<b>23</b>
4.1 POŽADAVKY NA PLATFORMU .....	23
4.2 ETHEREUM .....	23
4.3 ŘEŠENÍ PROBLÉMU ŠKÁLOVATELNOSTI (ETHEREUM DRUHÁ VRSTVA) .....	24
4.4 SOLANA.....	25
<b>5 DECENTRALIZOVANÉ APLIKACE V PRAXI.....</b>	<b>26</b>
5.1 DeFi (DECENTRALIZOVANÉ FINANCE) .....	26
5.2 HLASOVÁNÍ A SPRÁVA ORGANIZACÍ.....	26
5.3 VLASTNĚNÍ DIGITÁLNÍCH PŘEDMĚTŮ, NFT .....	27
5.4 HRY.....	27
<b>II PRAKTICKÁ ČÁST .....</b>	<b>28</b>
<b>6 ŘEŠENÝ PROBLÉM.....</b>	<b>29</b>
<b>7 ANALÝZA POŽADAVKŮ .....</b>	<b>30</b>
7.1 FUNKČNÍ POŽADAVKY .....	30
7.2 NEFUNKČNÍ POŽADAVKY .....	30
7.3 DIAGRAM PŘÍPADU UŽITÍ A SCÉNÁŘE .....	31
<b>8 VÝVOJ PROTOTYPOVÉ APLIKACE .....</b>	<b>35</b>
8.1 INSTALACE NÁSTROJŮ NUTNÝCH PRO VÝVOJ.....	35
8.1.1 Nástroje použité pro vývoj .....	35
8.1.1.1 Editor Visual Studio Code .....	35
8.1.1.2 Node package manager .....	35
8.1.1.3 Metamask.....	35

8.1.2	Specifické nástroje použité pro vývoj na blockchainu.....	36
8.1.3	Struktura nástroje Truffle .....	36
8.1.4	Popis nástroje Ganache .....	37
8.1.5	Knihovny .....	38
8.2	POSTUP VÝVOJE.....	39
8.2.1	Volba vhodné platformy .....	39
8.2.2	Implementace technologie smart kontraktů .....	39
8.2.3	Testování smart kontraktů.....	42
8.3	WEBOVÁ APLIKACE.....	43
8.3.1	Tvorba kostry projektu .....	43
8.3.2	Práce s vnitřním stavem aplikace Vuex .....	45
8.3.3	Připojení blockchainu k aplikaci a interakce s kontrakty.....	46
8.3.4	Navigace a jednotlivé cesty v aplikaci .....	50
8.3.5	Komponenty a zobrazení.....	50
8.4	NÁVRH UŽIVATELSKÉHO ROZHRANÍ.....	51
8.4.1	Hlavní obrazovka .....	51
8.4.2	Stránka pro vytvoření projektu.....	52
8.4.3	Detail projektu.....	53
8.4.4	Stránka zobrazující informace o aplikaci .....	54
<b>ZÁVĚR .....</b>		<b>56</b>
<b>SEZNAM POUŽITÉ LITERATURY.....</b>		<b>57</b>
<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>		<b>60</b>
<b>SEZNAM OBRÁZKŮ .....</b>		<b>61</b>
<b>SEZNAM TABULEK.....</b>		<b>62</b>
<b>SEZNAM PŘÍLOH.....</b>		<b>63</b>

## ÚVOD

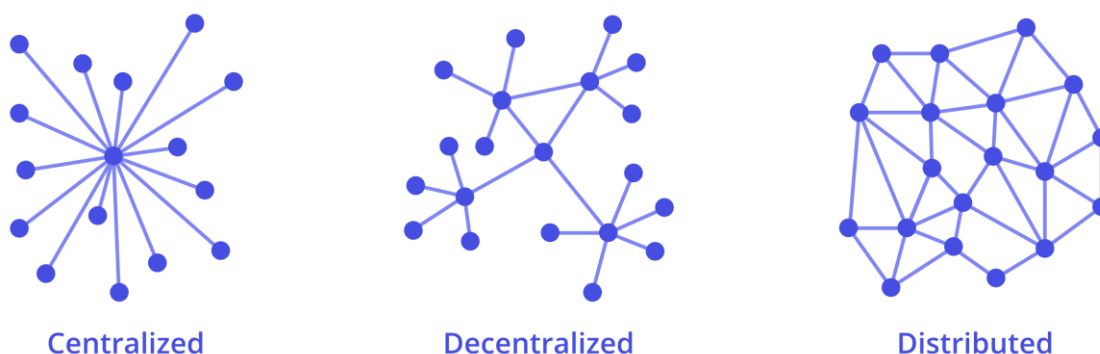
Blockchain a kryptoměny jsou jedním z nejskloňovanějších témat posledních let. Mnoha lidí si myslí že je toto odvětví určeno jen pro spekulace a rychlé zbohatnutí. Mimo spekulace se ale v tomto oboru vyvíjí mnoho decentralizovaných aplikací, které poskytují nová inovativní řešení pro stávající problémy.



## **I. TEORETICKÁ ČÁST**

## 1 BLOCKCHAIN

Technologie blockchain je na vzestupu v posledních letech, přestože nejde o zas tak novou technologii. Blockchain můžeme jednoduše popsat jako typ databáze, nicméně má několik unikátních vlastností, blockchain je také decentralizovaná distribuovaná a neměnná technologie pro ukládání dat a transakcí. Dnes je běžná praxe taková že všechna data jsou uložena v centralizovaných databázích poskytovaných státními či finančními institucemi, v tomto případě se uživatelé spoléhají na tyto centralizované systémy, aby udržovali jejich záznamy v pořádku a kontrolovali správnost jejich stavu. Instituce mohou ukončit svoji činnost a v případě že je obecná důvěra v tyto subjekty a jejich úložiště dat nízká, blockchain může zprostředkovat systém důvěry mezi více stranami, bez nutnosti použití třetí strany. Jak již bylo zmíněno technologie blockchain není výstřelem posledních let, byla navržena v roce 2008 jistým Satoshi Nakomtem, který ji popsal v práci Bitcoin: A Peer-to-Peer Electronic Cash System. V roce 2009 Nakomoto vytvořil síť Bitcoin zapomocí blockchainu. [1]



Obrázek 1 Typy počítačových sítí [19]

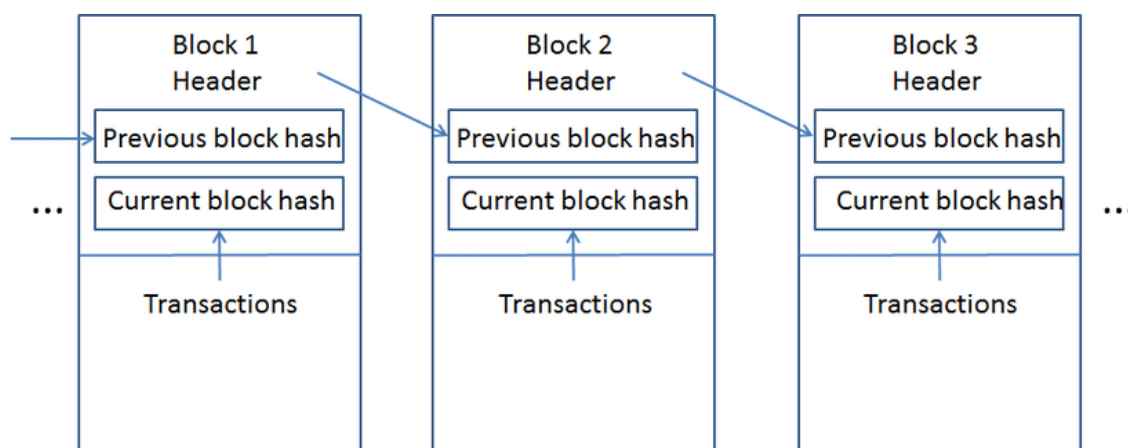
### 1.1 Princip funkce

Blockchain je postavený na kryptografických funkcích, konkrétně na asymetrické kryptografii a hashovacích funkcích. Aby mohl uživatel používat blockchain potřebuje klíčový pár, který obsahuje veřejný a privátní klíč. Veřejný klíč běžně nazýváme jako adresa a je používán pro identifikaci uživatele. Soukromý klíč se používá pro přístup k účtu, v praxi si to můžeme představit veřejný klíč jako číslo bankovního účtu a soukromý klíč jako PIN. Tudíž při ztrátě soukromého klíče přicházíme i o přístup k prostředkům na dané adrese.

Blockchain je v podstatě databáze skládající se záznamů o transakcích. Transakce obsahuje odesílatele, příjemce a data transakce. Odesílatel a příjemce jsou určeni pomocí veřejného klíče a každá transakce je podepsána soukromým klíčem odesílatele. Tyto transakce jsou sestaveny do bloků. Každý blok obsahuje určitý počet transakcí a hlavičku. Hlavička obsahuje časové razítko, nonce (náhodné číslo které lze použít pouze jednou), hash bloku a hash předchozího bloku. Díky tomuto vzniká z bloků řetězec. Bloky se přidávají do blockchainu chronologicky, díky tomuto je zaručena posloupnost transakcí v čase. Transakce jsou validovány dříve, než jsou přidány do blockchainu, způsob, jakým jsou bloky validovány závisí na konsensu sítě a jeho algoritmu potvrzování. Jakmile se jednou blok přidá do sítě nemůže být změněn, transakce jednou zapsané na blockchain jsou tedy nezaměnitelné. Pokud by chtěl někdo změnit data v blockchainu musel by změnit také každý blok po bloku přidaném, protože by přidaný blok změnil hash bloků všech následujících.

Blockchain je založen na tzv. uzlech. Uzel je počítač v síti, na kterém běží software blockchainu. Každý uzel obsahuje kopii blockchainu a také každý uzel validuje transakce. Nově vytvořená transakce je rozeslána do všech uzlů a čeká na validaci, některý z uzlů transakci ověří a přidá ji do řetězce bloků, tato akce můžeme trvat i několik minut v závislosti na typu blockchainu a momentální poptávce transakcí po validaci. Všechny uzly se snaží vyřešit proces validace co nejdříve a soutěží proti sobě. V případě že některý uzel přidá nový blok, upozorní ostatní. Ostatní uzly poté ověří přidanou transakci a pokud většina uzlů transakci schválí, nový blok je přidán do blockchainu ve všech uzlech. Tento proces zajišťuje že pokud by v síti byl uzel se zájmem podvádět, tak to nedokáže provést. Jelikož všechny uzly mají stejnou váhu, není zde potřeba žádné centrální autority. Systém tohoto ověřování také zamezuje tzv. problému dvojí útraty (double spending problem), ten spočívá v situaci, když se někdo pokusí utratit tu samou částku dvakrát (například token nějaké kryptoměny) ve dvou odlišných transakcích. V tomto případě je po první transakci přidán blok a při potvrzení další transakce s těmi samými prostředky, lze vidět že daná transakce je neplatná, protože tyto tokeny byly už utraceny, tudíž je transakce odmítnuta. Proces potvrzování bloků se nazývá těžba a jednotlivé uzly nazýváme těžaři. Těžař vytvoří nový blok, který se nazývá candidate block, ale aby tento blok mohl být připojen do blockchainu musí mít unikátní vlastnosti, které se rozlišují podle algoritmu konsensu sítě.

[4] [9]



Obrázek 2 Zjednodušená struktura blockchainu [20]

## 1.2 Algoritmy konsensu sítě

Úkolem algoritmů konsensu je zabezpečení toho, aby se uzly v síti shodly bez centrální autority, zároveň také zajišťují že údaje jsou retrospektivně neměnitelné a poskytují síti neoklamatelný matematický důkaz, na základě, kterého se uzly shodnou na vykonání akce. Těchto konceptů je velké množství, potřebnější jsou v této kapitole popsány dva nejpopulárnější. [36]

### 1.2.1 Proof of Work (PoW)

PoW umožňuje transakcím, aby byly potvrzeny a bloky s nimi přidány do blockchainové sítě. Protože PoW vyžaduje výkon, a spotřebovává elektrickou energii, je motivace pro útok či zahlcení sítě mnohem nižší. Ekonomická hodnota PoW tudíž přímo souvisí s cenou elektrické energie, používané během těžby. Při těžbě kryptoměn se používá hashovací funkce pro ověření dat. Hash je výstupem na blockchainu jako veřejný důkaz použití hashovacího algoritmu. Různé platformy často používají různé druhy hashovacích algoritmů, bitcoin používá SHA-256 hashovací algoritmus, Ethereum používá algoritmus Keccak-256. [4]. PoW také zajišťuje přísun nové kryptoměny do sítě a motivuje těžaře, aby potvrzovali bloky. Těžaři, kteří úspěšně přidají blok jsou odměněni nově “vyraženou” kryptoměnou. Hlavní kritikou sítě PoW je její spotřeba elektřiny, pro zajištění bezpečné fungování sítě. Bitcoin spotřebuje ročně okolo 205 TWh energie [34] všechny elektrárny v České republice pro porovnání vyrobí 81,4 TWh ročně. [2] Výhodou tohoto způsobu ověřování je neutralita, nepotřebujete vlastnit žádné finanční prostředky dané kryptoměny, abyste začali těžit, je to také už dlouhodobě otestovaný způsob potvrzování na sítích jako Bitcoin a Ethereum. Další nevýhodou po spotřebě energii, je potřebný výpočetní výkon,

tudíž těžařská uskupení z velkým hardwarovým výkon, mohou časem dominovat celé síti, což ústí v centralizaci sítě a bezpečnostním rizikům. [3] [5] [9]

### 1.2.2 Proof of Stake (PoS)

V PoS systém vybere automaticky vybere jednoho validátora, který schválí následující blok. Bezpečnost je zaručena tím je validátor v rámci svého uzlu uzamče určitý počet mincí. Každý validační uzel ručí za to, že nebude podvádět právě těmito uzamčenými mincemi. Pokud je validační uzel vybrán, ověří všechny transakce a pokud jsou správné, označí validátor blok za platný, uzavře ho, přidá na blockchain a jako odměnu obdrží poplatky za transakce. Pokud by chtěl podvádět a přidal do bloku nějaké nevalidní transakce, tak přijde o své záruční mince, protože ostatní uzly můžou tuto validaci snadno ověřit. V případě že si validátor už nechce validovat další bloky, může si vybrat všechny mince co zde uzamčel jako záruku a k tomu mince které obdržel v poplatcích. Je zde ale stanovena časová prodleva, aby v případě že validátor, který v minulosti vytvořil neplatný blok nemohl mince vybrat ihned. Nevýhodou této sítě může být to, že jedna společnost si může vytvořit více validačních uzlů a čím více validačních uzlů má tím větší šanci má na to že budou její uzly zvoleny pro validaci transakcí, záleží ale přímo na nastavení jednotlivých sítí. Další kritikou je to že PoS nemá žádné vstupy z reálného světa, jelikož se zde ručí pouze digitálními tokeny. [6]

## 2 SMART CONTRACTS (CHYTRÉ SMLOUVY)

Přestože smart kontrakt, není moc chytrý ani to vlastně není kontrakt, je tento termín používán pro softwarový skript, který žije na blockchainu. Interakce s těmito kontrakty probíhá pomocí transakcí, které spouští kód. Cílem těchto smluv je dohoda a vytvoření důvěry mezi všemi zúčastněnými stranami. Těchto vlastností se snaží smart kontrakty dosáhnout pomocí transparentnosti kódu, který je veřejně nahrán na blockchain, jeho neměnností a deterministickou povahou vykonávání kódu. Smart kontrakty můžeme snadno pochopit na příkladu výdejního automatu, se správnými vstupy je kýžený výstup jistý. Abychom dostali zboží z automatu musíme do něj vložit peníze a vybrat si které zboží chceme, a pokud jsou vstupní podmínky ve formě ceny produktu splněny, zboží dostaneme. Stejný typ logiky programujeme do smart kontraktů. Odesláním transakce na danou adresu můžeme spustit implementovaný kód a provést danou akci. Na rozdíl od automatu zde není potřeba prostředníka v podobě zaměstnance, který by se o daný automat staral. Smart kontrakty na sítích jako je Ethereum si můžeme představit jako otevřené API služby, pomocí volání z nich získáváme data, tato volání můžeme dokonce provádět i z jiných kontraktů. [7]

### 2.1 Historie

V 90. letech 20. století Nick Szabo poprvé teoreticky popsal smart kontrakty. Poprvé byla teorie smart kontraktu implementována v omezeném měřítku na síti Bitcoin v roce 2009. Ale celkově smart kontrakty nabrali na popularitě až s příchodem sítě Ethereum a programovacího jazyka Solidity v roce 2013. [8]

### 2.2 Vlastnosti smart kontraktů

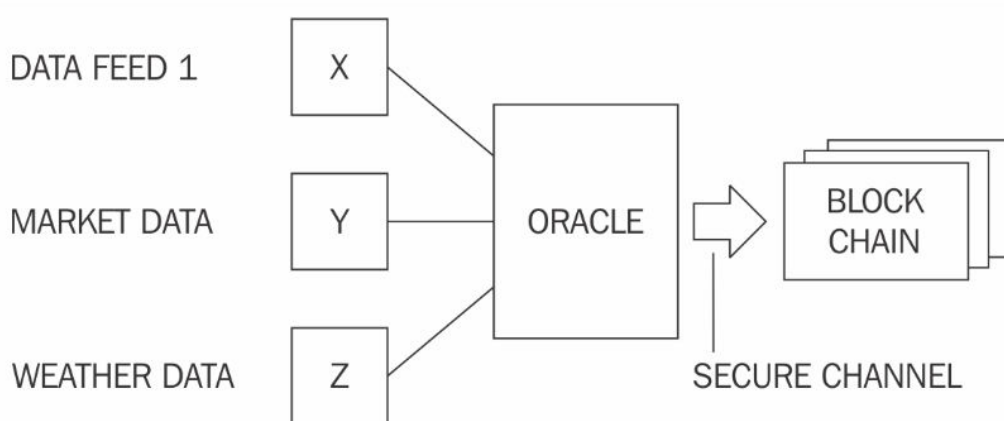
Hlavními charakteristikami smart kontraktů je jejich **bezpečnost**, díky použití kryptografických principů jsou odolné proti manipulaci a napadení. Díky nasazení na blockchainu, kde si každý může ověřit jejich obsah jsou **transparentní** a **nefiguruje v nich žádný prostředník**, který by byl zapojen v procesu ověření či vykonání kontraktů, díky tomu že nepotřebují tohoto prostředníka jsou **samospustitelné a autonomní**. Při naplnění stejných vstupních podmínek, jsou výstupní podmínky vždy stejné, jsou tedy **deterministické**. Pokud by se výsledky v jakémkoliv případě lišili, byla by ohrožena bezpečnost sítě a její validace. Deterministický musí být i programovací jazyk ve kterém se chytré kontrakty píšou. V případě, že by několik různých jazyků smart kontraktů

implementovalo například funkce s desetinnou čárkou, které by každá dávaly, byť i minimálně různé výsledky, byla by opět ohrožena celistvost sítě. Díky transparentnosti a deterministické povaze kontraktů jsou také **předvídatelné**. Přestože může záviset i na propustnosti sítě, kontrakty se obvykle **vykonávají v reálném čase**, je tudíž zajištěno že informace pro všechny strany jsou ihned stejné. Díky kombinaci těchto faktů jsou smart kontrakty, ale stále dobré jenom tak, jak jsou jejich pravidla a logika v nich naprogramovaná. Bezpečnost smart kontraktů tudíž může být ohrožena znalostmi programátora a mírou bezpečnostního testování a auditování daného kontraktu. [<https://scalablesolutions.io/news/smart-contracts-and-their-characteristics/>] [5]

### 2.3 Oracles

Představme si smart kontrakt, který operuje s kurzem nějaké měny z reálného světa, jeho hodnotu potřebujeme získat z nějakého externího zdroje, pokud bychom ale přímo volali například API z blockchainu, tak problém spočívá v tom, že každý uzel blockchainu musí mít všechny transakce identické, a pokud by každý uzel musel použít stejné API volání, tak by se data z tohoto volání mohla lišit, takže tato data nemůžou být validována a všechny uzly blockchainu by nebyli ve stejném stavu. Tento problém se snaží vyřešit služby oracle.

Můžeme si je představit jako rozhraní, či most mezi blockchainem a reálným světem. Oracle volají API a jejich výsledky zapisují na blockchain pomocí transakcí, díky tomu můžeme udržet všechny uzly ve stejném stavu.



Obrázek 3 Schéma funkce oraclů v blockchainové síti [5]

Tento most může být i obousměrný a stejným způsobem můžeme i pomocí transakcí získávat informace zevnitř blockchainu do reálného světa. V závislosti na typu průmyslového odvětví, ve kterém jsou kontrakty nasazeny, potřebuje z oracles získávat

různá data od předpovědi počasí, různých finančních hodnot až po data která nám poskytují IoT zařízení. Naskytá se zde problém s důvěryhodností oracles, pro oracle je žádoucí aby nebylo možné manipulovat s daty které poskytují a aby byla tato data důvěryhodná. Pokud bychom brali data jen z jednoho zdroje, prakticky tím ztrácíme decentralizovanou povahu kontraktu a vytváříme jediný kritický bod selhání kontraktu. Řešení tohoto problému je například získávání stejné informace z mnoho rozdílných zdrojů, v tomto případě by útočníkovi nestačilo ovlivnit kurz například jen jedné burzy, ale musel by vynaložit mnohem více prostředků pro manipulaci více burz. Hojně používaným řešením je decentralizovaný systém oraclů jako <https://chain.link/>. [5].

## 2.4 Implementace na platformě Ethereum

Při vytváření sítě Ethereum, byla implementace smart kontraktů jedním z jejich hlavních cílů. Momentálně je tento typ implementace jedním z nejčastějších. Smart kontrakty jsou zde typicky napsané v nějakém vysokoúrovňovém jazyce jako Solidity. Pro běh na síti je ale nutné zkompilovat vysokoúrovňový kód do bytekódu, který běží v Ethereum Virtual Machine (EVM). Jako obranu proti útokům či nekonečným smyčkám, musíme za každou operaci, kterou náš program vykoná zaplatit virtuální měnou Ether (ETH), velikosti těchto poplatků vyjadřujeme v jednotce gas, která odpovídá hodnotě  $\text{ETH} * 10^{-9}$ , tudíž výše gasu určuje cenu vykonání smart contractu.

Na platformě Ethereum máme 2 typy adres. Externally Owned Account (EOA) jsou adresy uživatelů, kteří mají privátní klíč k dané adrese a Contract account jsou adresy řízené smart kontrakty, které mají přiřazený kód a úložiště dat, nemají svůj privátní klíč, tudíž tyto adresy vlastní sami sebe. Všechny kontrakty jsou zde spouštěny vždy transakcí, která pochází z EOA adresy. Kontrakty můžou volat další kontrakty, které mohou dále volat další kontrakty, a tak dále, ale první interakce s nimi vždy pochází z EOA adresy. Kontrakty se nikdy nespustí jejich vlastní vůlí nebo někde v pozadí, vždy jsou pouze nahrány na síť, a zůstávají nečinné, dokud nejsou spuštěny nějakou transakcí. Kontrakty také nedokážou běžet paralelně, Ethereum můžeme považovat jako jednovláknový počítač. [4]



### 3 DECENTRALIZOVANÉ APLIKACE

#### 3.1 Historie vývoje webu

Abychom pochopili, proč vůbec decentralizované aplikace vznikli musíme znát historii vývoje webu, který se dělí do třech částí.

První část tzv. Web 1.0 je první verzí, který probíhala okolo let 1991–2004, většina účastníků webu byli konzumenti obsahu a tvůrci obsahu byli typicky vývojáři, kteří vyvíjeli webové stránky, které většinou obsahovali informace v textovém formátu či zde byli obrázky. Tuto éru webu tvořili tedy zejména statické stránky, se kterými se dalo minimálně interagovat, tuto éru reprezentovali zejména stránky pouze pro čtení.

Web 2.0 je éra webu ve které se pohybujeme právě teď, je to éra interaktivních a sociálních webů. Oproti minulému období zde nemusí být uživatel vývojářem, aby se zapojil do tvůrčího procesu, portály jako YouTube či Twitter umožňují každému, aby jednoduše tvořil obsah pro miliony lidí. Web 2.0 je jednoduchý a uživatelsky přívětivý a díky těmto vlastnostem se z mnohých stávají tvůrci, což je skvělé, ale tato éra webu sebou nese i některé problémy jako způsob monetizace obsahu, a také otázky soukromí a soukromí. Dnes je prodej osobních data za účelem výdělků běžným obchodním modelem a úniky dat z velkých databází nejsou také ničím nečekaným. Uživatel dnes při používání nejpopulárnějších webů nemá žádnou kontrolu nad svými daty a všechna tato data jsou vlastněna velkými platformami, v případě úniku těchto se můžou uživatelé dostat do potíží. Pokud se nelíbí nějaká situace centrální autoritě, může jednoduše vypnout všechny účty a zastavit běžící platformy i s daty běžného uživatele.

Web 3.0 je novou dobou webu, který se snaží vyřešit nedostatky Webu 2.0, pomocí změnou architektur aplikace a způsobů jakým s nimi pracuje, těchto změn je více, ale hlavním bodem je decentralizace. [10] [11]

#### 3.2 Vlastnosti decentralizovaných aplikací

Decentralizované aplikace se od jejich centralizovaných verzí liší v mnoha aspektech, ale jak je u všeho běžné mají také své výhody a nevýhody. Jednou z hlavních výhod decentralizovaných aplikací je **stálá dostupnost** aplikace, jakmile je aplikace nasazena na blockchain, tak by útočníci museli úspěšně zaútočit na celou síť, aby aplikaci znepřístupnili. Uživatelé nepotřebují pro interakci s aplikací poskytovat svoji identitu

v reálném světě, aplikace nepotřebují zásah do **soukromí** uživatele a uživatel zůstává **anonymní**. Díky decentralizované povaze aplikací, žádná entita či autorita nemůže nikoho blokovat proti publikaci aplikací, posílání transakcí či nahlížení do blockchainu, aplikace jsou díky tomuto **odolné proti cenzuře**, pokud bychom uvažovali decentralizovanou aplikaci typu sociální sítě, nikdo není schopný rozhodnout o smazání našich příspěvků. Data uložená na blockchain jsou **neměnná a nesporná**, díky zabezpečení kryptografickými primitivy. Narušitel sítě není schopen zfalšovat data, která již byla zveřejněna. Díky interakci s aplikacemi pomocí kryptoměnových peněženek zůstávají naše **přihlašovací údaje v bezpečí**. Díky tomu že smart kontrakty jsou nasazeny na veřejném blockchainu, může si je každý uživatel prověřit a díky tomu je jejich **výstup předvídatelný a** není v nich potřeba prostředníka. Decentralizované aplikace se ale neobejdou bez řady nevýhod a nedostatků. Tyto aplikace jsou velmi **náročné na údržbu**, protože kód jednou publikovaný na blockchain je velmi těžké změnit, pro vývojáře je velmi náročné změnit chod decentralizovaných aplikací, jak jsou jednou nasazeny na síť, i v případě že nasazená verze obsahuje bezpečnostní rizika a chyby. Zachování bezpečnosti a decentralizovanosti sítě je velmi náročné, protože ukládání každé transakce na každý uzel v síti je časově a výpočetně intenzivní úkon, tudíž **škálovatelnost sítí je problémová**. V případě aplikace produkují více transakcí za sekundu, než je síť schopná zpracovat, tak se zvyšuje **zahlcenost sítě**, a doba za kterou se transakce zapíše rapidně roste. Díky použití nových technologií, může pro běžného uživatele interakce s těmito aplikacemi přinášet **horší uživatelský zážitek**. V současné době je vytvoření plně decentralizovaných aplikací na úrovni běžných centralizovaných aplikací velmi složité a je využíváno kompromisů pro dosažení plného uživatelského zážitku, tyto **částečně centralizovaná řešení** a jejich prvky mohou představovat úzké hrdlo v bezpečnosti a stabilitě aplikace. [15]

### 3.3 Architektura decentralizovaných aplikací

Decentralizované aplikace jsou podobné dnešním aplikacím, ale aplikace nejsou vyvinuty a nasazeny na jeden server a data z nich nejsou uložena v jedné databázi, která je dnes běžně na cloudovém serveru jediného providera. Tyto aplikace používají blockchain a technologii smart kontraktů jako backend pro běh své logiky, hlavním rozdílem oproti běžným aplikacím je cena, kterou musíme za každou provedenou operaci, která mění vnitřní stav backendu zaplatit, tudíž bychom měli každý výpočetní úkon zvážit a minimalizovat jejich počet. Je nutné identifikovat které aspekty aplikace potřebují být

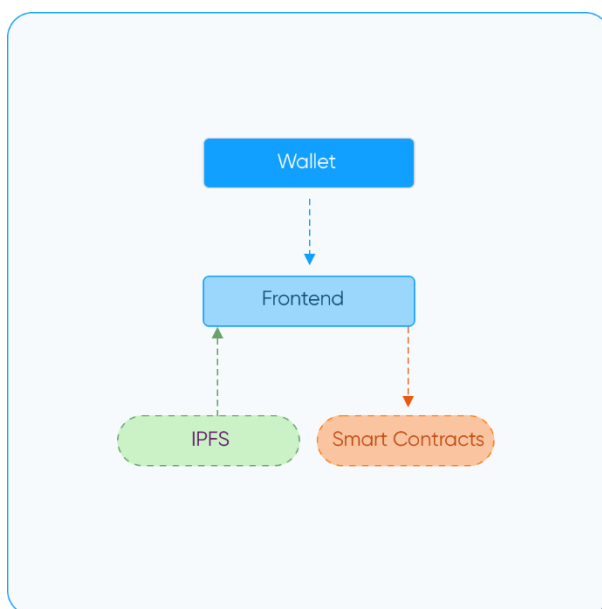
vykonány na důvěryhodné a decentralizované platformě. Na rozdíl od backendu, kde musí vývojář těchto aplikací znát prostředí daného blockchainu a jeho vývojové nástroje, je frontend v případě decentralizovaných aplikací vyvíjen pomocí běžných webových technologií jako HTML, CSS a Javascript. Tento fakt umožňuje použití standardních vývojových nástrojů, knihoven a frameworků. Většinou je frontend spojen s blockchainem pomocí javascriptové knihovny jako je web3.js. Díky vysokým poplatkům za transakce, nebo limitům poplatků pro jednotlivé bloky, nejsou smart kontrakty dobře použitelné pro ukládání a zpracování velkého množství dat, většina decentralizovaných aplikací tedy ukládá své velké objemy dat mimo samotný blockchain. Tato platforma pro ukládání dat mimo blockchain může být centralizovaná (typicky nějaká cloudová databáze), nebo mohou být data decentralizovaná a uložena na P2P platformách jako je IPFS nebo lze použít vlastní implementaci pro danou platformu, pro Ethereum je to například Swarm. Ve velkém množství těchto aplikací také hrají velkou roli kryptoměny, které mohou poskytovat motivaci ve formě tokenů dané aplikace, kterou uživatelé obdrží za používání aplikace či podílením se na jejím chodu. Běžní uživatelé těchto webů platí za jejich služby, ale tyto prostředky mohou jít přímo k účastníkům kteří se podílejí na chodu služby, je zde odstraněna nutnost prostředníka. [4] [10]

IPFS a Swarm jsou decentralizované P2P úložiště, které oproti běžným sítím, které používají location based addressing, používají content-based addressing, což znamená že každá část souboru je zahashovaná a výsledný hash je použit k identifikaci daného souboru. Z každého IPFS uzlu můžeme pak dostat daný soubor pomocí hashe hledaného souboru. Swarm je obdobná implementace P2P datového úložiště, která byla vytvořena přímo projektem Ethereum. [4]

### 3.4 Míra decentralizace

Další otázkou u decentralizovaných aplikací je míra jejich decentralizace, u každé z těchto aplikací můžeme decentralizovat různé části jako backend, frontend, úložiště dat, komunikaci či systém vytváření domén. Přestože smart kontrakty jsou určeny k tomu, aby nahradili backendové servery, tak potřebujeme ve většině případů aspoň server a API abychom zajistili hladký běh decentralizované aplikace pro uživatele. Decentralizované aplikace můžeme rozdělit na několik typů:

- **Plně decentralizované** aplikace mají výhodu, že využívají bezpečnosti decentralizace, každý uživatel je vlastníkem svých prostředků, jedinou povinností aplikace je zaručení fungování základních funkcionalit aplikace, pokud je klient či frontend aplikace jednoduchý je správa těchto aplikací po nasazení velmi jednoduchá. Frontend musí být v tomto případě hostován na platformě jako je IPFS, a ne na centralizovaném úložišti, abychom zajistili úplnou decentralizaci a nevytvářeli kritický bod v aplikaci. V případě této architektury v podstatě nejde aplikaci zrušit. Nevýhodou těchto aplikací je nekvalitní uživatelský zážitek, problémy s výkonem a spolehlivostí aplikace a nedostupnost běžných vlastností moderních webových aplikací. V této době neexistuje mnoho aplikací, které by se vydávali touto cestou. [12]

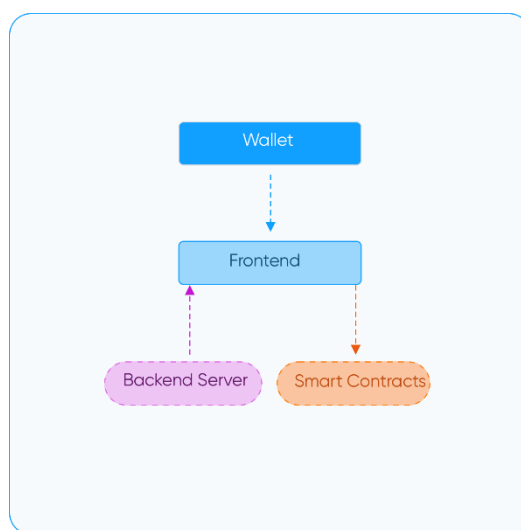


Obrázek 4 Schéma architektury plně decentralizované aplikace [12]

- **Centralizované** aplikace jako například centralizované burzy kryptoměn, dělají všechny operace uvnitř aplikace mimo decentralizovanou platformu. Všechny kryptoměny, s kterými obchodujete uvnitř aplikace jsou zaznamenány v tradiční databázi. Dvnitř burzy si můžeme poslat finance z vnějšího prostředí, či poslat jednotlivé kryptoměny na vlastní decentralizovanou adresu směrem z burzy. Všechny operace vně burzy ale neinteragují s decentralizovanou platformou. [13]

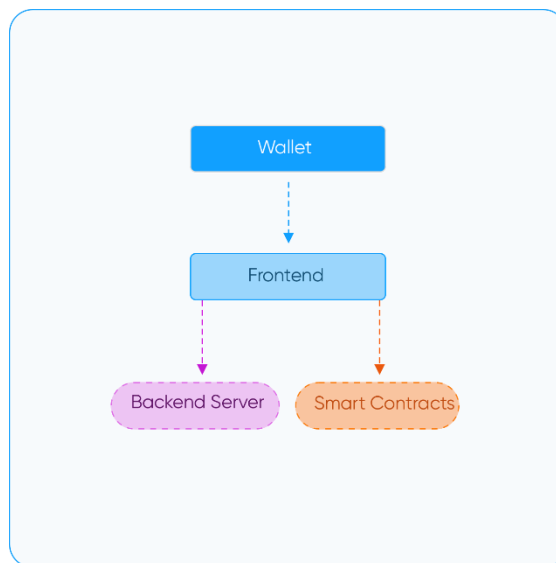
- **Částečně decentralizované** aplikace jsou aplikace, které vyměňují výhody úplné decentralizace za užitečné vlastnosti centralizovaných řešení. V případě plně decentralizovaných aplikací, nám smart kontrakty odvádějí všechnu práci na backendu a můžou nám omezovat možnosti potencionálních funkcí platformy. Částečně decentralizované aplikace rozdělují práci právě mezi tyto smart kontrakty a tradiční backend server. Uživatelský zážitek je totiž velmi důležitý pro získání uživatelské základny a plně decentralizovaná řešení ho v této době nedokážou naplnit.

Důležité funkce aplikace jsou ale stále decentralizované abychom zachovali integritu transakcí. Díky tomu že tato řešení obsahují backendový server, využívá se zde pro obsluhu frontendu , běžných cloudových řešení, které jsou spolehlivější než decentralizované úložiště typu IPFS. Bohužel nám tento cloudový server poté vytváří bod selhání, takže musí být obzvlášť dbáno na dodržování bezpečnosti. [12]



Obrázek 5 Schéma architektury decentralizované aplikace s použitím centralizovaného serveru [12]

Další variantou může být komunikace backendu se smart kontraktem, v tomto případě neposíláme z frontendu požadavky přímo do smart kontraktu, ale backend je zodpovědný za verifikaci a validování transakcí, uživatelé ale mohou mít možnost v tomto případě interagovat i přímo se smart kontraktem za pomoci peněženky. [12]



Obrázek 6 Schéma architektury aplikace s použitím centralizovaného serveru pro filtraci a odesílání transakcí [12]

### 3.5 Kryptoměnové peněženky

Tyto peněženky se používají k ukládání privátních klíčů, což jsou vlastně hesla, které nám dávají přístup k našim kryptoměnám, v bezpečném a lehce přístupném způsobu. Peněženky mohou být jak hardwarové (Ledger, či například český Trezor) či jako čistě softwarová řešení, jako například rozšíření do prohlížeče Metamask. Na rozdíl od běžné peněženky, ve které své finance přímo držíme, tak kryptopeněženky tyto finance nedrží, naše finance jsou na blockchainu a mohou být zpřístupněny pouze privátním klíčem, který v peněžence držíme, tudíž v případě že ztratíme přístup k peněžence ztratíme i přístup ke svým financím. Pomocí těchto peněženek se přihlašujeme do decentralizovaných aplikací a pracujeme s nimi. Peněženky vlastně zajišťují uživatelský účet pro přístup do těchto aplikací. [<https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-wallet>]

## 4 PLATFORMY

Přestože dnes existuje nespočet různých blockchainů, pro vývojáře nemusí být všechny výhodné. Při volbě blockchainu si musíme vytyčit požadavky, které jsou pro nás důležité.

### 4.1 Požadavky na platformu

Jedním z prvních požadavků, které musíme brát v potaz je **typ** blockchain sítě, kterou použijeme, pokud vyvíjíme aplikaci pro veřejnou síť a chceme, aby se mohl připojit kdokoli s běžným přístupem, zvolíme typ veřejného (public blockchainu), v případě že viditelnost záznamů v síti je nežádoucí (na blockchain sdílíme osobní údaje, obchodní tajemství) musíme aplikaci nasadit na soukromý (private) blockchain. V případě že je nutné použít kombinaci těchto dvou sítí a vytvořit různá práva pro různé typy uživatelů je vhodné použít permissioned blockchain. [16] Dalším faktorem pro výběr sítě je její **rychlost**, respektive kolik transakcí za vteřinu je síť schopna zvládnout, s čím se pojí i doba za jakou se jedna transakce zpracuje. Problém rychlosti se také pojí se škálovatelností sítě, při strmém nárůstu uživatelů se může transakční doba zvýšit. Pokud bychom museli kvůli bezpečnosti aplikace provozovat plný uzel blockchainu, tak při vysokých transakčních rychlostech, potřebujeme na provoz uzlu velmi rychlé připojení a úložiště. [17] Jestliže naše aplikace musí komunikovat s ostatními blockchaine musíme brát v potaz také **kapacitu přemostění a možnosti přemostění**. Další důležitou součástí pro vývoj aplikace jsou **nástroje použitelné v jeho ekosystému**, otázkou je, jestli jsou vůbec dostupné nástroje či způsob jakým jeho schopné náš případ užití na dané platformě zrealizovat. V případě že se jedná o zavedený blockchain, vývojové nástroje budou mít kvalitní podporu a vývoj aplikace bude časově méně nákladný. **Celková cena provozu** aplikace je jedním z nejdůležitějších faktorů pro výběr platformy, otázky, které si klademe jsou kolik stojí uložení dat na síť, kolik stojí jednotlivé transakce a jejich poplatky, kolik stojí provoz hardwaru pro běh aplikace na dané platformě či kolik stojí vývoj aplikace. [18]

### 4.2 Ethereum

Ethereum je open-source blockchain a zároveň platforma pro provoz decentralizovaných aplikací, která byla založena v roce 2014 Vitalikem Buterinem [21]. Je to platforma na s nejvyšší celkovou uzamčenou hodnotou a to okolo 113 miliard dolarů [22] a používá se pro různé typy decentralizovaných aplikací, není nijak specificky zaměřen [23]. Z tohoto

faktu vyplývá že Ethereum není žádným výstřelem do tmy ve světě decentralizovaných platforem, naopak je jednou s nejstálenějších platforem. Mechanismus konsensu je PoW, ale v druhé polovině roku 2022 by měla celá síť migrovat na konsensus PoS, kterým si vývojáři slibují zvýšení škálovatelnosti sítě, nyní je Ethereum síť s obrovským provozem a transakční rychlostí okolo 15 až 30 transakcí za sekundu. Obrovská poptávka po transakcích, žene výši poplatků do neúnosných výšin pro běžné uživatele. Důležitou součástí Etherea je EVM, což je technologie a architektura, které umožňuje bezpečný překlad kódu smart kontraktů pro běh na síti. Spoustu ostatních blockchainů implementuje EVM kompatibilitu, vývojář může tedy napsat jedinou aplikaci a nasadit ji snadno na jakýkoliv blockchain podporující EVM. Kryptoměnou Etherea je 1 Ether (ETH), který se používá k provádění transakcí na síti a pro placení poplatků s nimi spojených. Nejpoužívanějším jazykem pro vývoj kontraktů je zde Solidity, ale existují i alternativy jako například Vyper a Yul. [24]

### 4.3 Řešení problému škálovatelnosti (Ethereum druhá vrstva)

Ethereum i Bitcoin jsou základní vrstvou svého blockchainu, kterou označujeme jako první vrstva. Druhá vrstva je vrstvou takzvaných rollupů, které stavějí na základech první vrstvy. Třemi klíčovými prvky blockchainu je decentralizovanost, bezpečnost a škálovatelnost, blockchainové trilema nám ale říká že běžná blockchainová architektura může dosáhnout jen 2 z těchto 3 vlastností. Pokud chceme bezpečný a decentralizovaný blockchain, musíme tedy obětovat škálovatelnost. [35]. Hlavním cílem druhé vrstvy je tedy škálovatelnost ve formě zvýšení rychlosti vyřizování transakcí a jejich zvýšená průchodnost, tedy kolik transakcí je schopno se provést za vteřinu. Hlavními výhodami druhé vrstvy jsou nižší poplatky, kterých lze dosáhnout díky kombinaci mnoha transakcí, do jedné transakce, které se vykoná na první vrstvě. Druhá vrstva si zachovává bezpečnost první vrstvy, ale díky vyšší rychlosti, nižším poplatkům a dalším technologiím rozšiřuje možné případy užití decentralizovaných aplikací. Projekty druhé vrstvy můžeme rozlišovat podle typu architektury druhé vrstvy. (optimistické rollupy, rollupy důkazu s nulovou znalostí). [25] Mezi nejpopulárnější řešení druhé vrstvy se řadí projekty Polygon, Arbitrum či Loopring. Rychlost transakcí v těchto sítích je několikanásobná a cena transakcí je také zlomková oproti první vrstvě. Použití těchto platforem je pro velmi široké spektrum decentralizovaných aplikací, zejména v tom případě že chceme zpracovávat velké množství transakcí. [26]



## 4.4 Solana

Solana je open-source blockchain první vrstvy, který se snaží vyřešit problém škálovatelnosti, je to jedna z nejrychlejších sítí a udává se propustnost až 65000 transakcí za sekundu. Využívá nové výpočetní technologie, které mohou podporovat tisíce uzlů, což umožňuje vysokou propustnost škálovatelnou se šířkou pásma sítě. Solana využívá kombinaci mechanismů konsensu Proof of Stake a Proof of History. Solana implementuje své vlastní řešení decentralizovaných aplikací, smart kontrakty jsou zde vyvíjeny v jazycích Rust či C++. Solana je opět blockchainem s širokou škálou použití a decentralizované aplikace se na ní vyvíjejí v nejrůznějších odvětvích. Jejím měnou a tokenem je SOL. [27]

## 5 DECENTRALIZOVANÉ APLIKACE V PRAXI

Decentralizované aplikace můžeme rozdělit do několika kategorií jako hry, DeFi (decentralizované finance), směnárny, tržiště či virtuální světy.

### 5.1 DeFi (Decentralizované finance)

Obor decentralizovaných financí se snaží řešit mnoho problému, které finanční svět dnes obsahuje. DeFi aplikace poskytují tradiční finanční služby jako platby, půjčky, úvěry, obchodování či investice efektivnějším a otevřenějším způsobem. DeFi aplikace dokážou zpracovat platby téměř hned, což je oproti tradičním systémům, kde je splatnost i několik dní obrovská výhoda. Proti běžnému světu, zde každá transakce stojí stejně bez závislosti na to, kde se příjemce a odesílatel nacházejí ve světě. Námitkou proti používání těchto řešení je volatilita hodnoty kryptoměn, kde poslané peníze dnes nemusí mít stejnou hodnotu již v řádu minut, řešením tohoto problému jsou stablecoiny, kde je jejich hodnota navázána na jiné aktivum jako jsou například dolary. Dalším použitím jsou tradiční finanční služby jako půjčky a úvěry. Jednou z nových forem půjček jsou flash loany, je to typ půjčky, ve které musí být půjčené finance splaceny ve stejné transakci jako byly půjčeny. V decentralizovaném světě můžeme na různých platformách své kryptoměny různě zapůjčovat a získávat mnohem vyšší úrokové sazby než v běžných bankách, to ale samozřejmě nese i rizika, příkladem těchto spořicích aplikací je Aave. Aplikace v tomto odvětví s nejvyšším obchodovaným objemem financí jsou decentralizované směnárny, zkráceně DEX (decentralized exchanges), jako příklad například projekty jako Uniswap či PancakeSwap [28]

### 5.2 Hlasování a správa organizací

Blockchain přináší bezpečný a transparentní systém pro hlasování. Díky decentralizovanosti blockchainu si může každá strana ověřit, že neexistuje žádné centrální řízení určující výsledek hlasování. Což poskytuje mnoho příležitostí pro různé druhy organizací. Stanovy pro akcionáře mohou být například přímo zakódovány v systému, tudíž je tam přesně uvedeno, kdo má jaké práva v subjektu a o čem může hlasovat. V online komunitách může poskytovat blockchainového hlasování pro danou platform, způsob, jakým zajistíme největší spokojenost jejich uživatelů. Těmto komunitám vlastněnými uživateli se v blockchainovém světě říká DAO (decentralized autonomous organizations). V podstatě jde o čistě internetové společnosti, kde je společnost vlastněna

jejími členy a také je pomocí hlasování jejich členů rozhodováno o jejich budoucnosti. Všechny akce v těchto společnostech jsou plně transparentní a veřejné. Základním stavebním kamenem těchto sítí je smart kontrakt, ten definuje pravidla organizace a také drží v sobě veškeré finance společnosti, funguje tedy jako pokladna. [29] [30]

### 5.3 Vlastnění digitálních předmětů, NFT

NFT (non-fungible tokens) jsou tokeny, které reprezentují vlastnění unikátních předmětů. NFT nás nechávají změnit předměty jako umění, sběratelské předměty či nemovitosti v digitální tokeny. Tyto tokeny mohou mít v jednu chvíli pouze jednoho vlastníka. NFT jsou trendem posledních let, kdy zaznamenaly obrovský růst, nejdražší kousky jsou schopny se vyšplhat do výše desítek milionů dolarů. [31]. NFT se snaží vyřešit problém digitálních předmětů, které postrádají vlastnosti jejich fyzických protějšků jako jejich jedinečnost, nedostatek a způsob jakým prokazujeme jejich vlastnictví. Pokud si zakoupíme například hudbu na platformě jako je iTunes či Google Play Music, a dále už ji nechceme vlastnit nemáme možnost jak s ní dál obchodovat, tudíž i přes to že jsme si ji zakoupili s ní nemůžeme dále nakládat.

### 5.4 Hry

Ve hrách typicky figurují nějaké virtuální předměty, se kterými může hráč obchodovat. V případě blockchainových her můžeme tyto předměty vlastnit stejně jako můžeme vlastnit kryptoměny. Obrovskou roli hrají v tomto světě NFT, které reprezentují jednotlivé předměty ve hrách. Tyto digitální aktiva mohou být vaše napořád, pokud je vlastníte. Díky transparentnosti blockchainu si můžeme prohlížet kdo tyto předměty vlastní a pokud hra přestane fungovat, naše předměty z blockchainu nezmizí. Díky tomuto se objevil nový typ her, jsou to takzvané play-to-earn hry, kde hraním hry a následným obchodováním s danými předměty či suroviny hry, můžeme vydělávat peníze. Nejsilnějším příkladem těchto her je hra Axie Infinity, ve které bojují hráči se svými virtuálními mazlíčky. Hra má kolem 2.5 milionu stálých hráčů, z čeho 35 % pochází z Filipín. Lidé z Filipín s nižším příjmem jsou si schopni denně vydělat více než by vydělali klasickou manuální prací. [32] [29] [33]

## **II. PRAKTICKÁ ČÁST**

## 6 ŘEŠENÝ PROBLÉM

Crowdfunding nebo též skupinové financování je alternativní způsob financování podnikatelských aktivit. Použití tohoto způsobu financování je celkově na vzestupu a je úzce spojen s využitím internetu a celkově na sociálních sítích zaznamenává obrovský růst. Crowdfunding jako pojem se skládá ze dvou anglických slov a to *crowd* – dav a *funding* – financování. Lze tedy o crowdfundingu hovořit jako o davovém financování. Jednou z charakteristik crowdfundingu je jeho široký záběr podporovaných aktivit, nápadů či společností. V dnešní době si může zkusit na veřejné crowdfundingové platformě vytvořit projekt téměř každý a už je jen na něm, jak ho sám dokáže zpropagovat. Kouzlo crowdfundingu spočívá v tom že z velkého množství lidí, je část lidí ochotna podpořit malou částkou, a součet těchto malých částek je schopen naplnit finanční cíle projektu. Crowdfunding také poskytuje lidem šanci, podílet se na rozvoji projektů které oni přesně chtějí. Velkou částí z portfolia projektů, které se obecně zakládají jsou čistě digitální projekty, tyto projekty bývají z pohledu tradičního financování moc rizikové či nevýdělečné, a tak se uchylují k tomuto způsobu financování. Jen na českém internetu existuje několik crowdfundingových platforem, které nejsou žádnými zapadlými platformami, na kterých by se vybralo pár příspěvků denně. Pro příklad na portálu Hithit mají nejpopulárnější projekty řady přispěvovatelů v tisících. Crowdfunding také můžeme dělit na více kategorií a to dárcovský, ve kterém lidé peníze pouze darují, odměnový, v němž si lidé předplácují produkt, dluhový, kde lidé půjčují dalším lidem nebo firmám, a podílový, kde investoři získávají podíl ve firmě. Otázkou je, jak velké procentu podílu by i měl ponechat portál, přes které se dané financování děje, ve světě decentralizovaných aplikací teoreticky můžeme toto procentu velmi snížit či úplně vypustit, díky eliminaci prostředníka. Pokud mají obě strany důvěry v daný smart kontrakt, který crowdfunding implementuje, mohou být konečné poplatky, jen ve výši poplatků za transakce. [37]

## 7 ANALÝZA POŽADAVKŮ

Aplikace je určená pro kohokoliv, kdo chce pracovat s metodou crowdfundingu a zajímá se o decentralizovaná řešení. Účel této aplikace je umožnit financování projektů za pomoci kryptoměn a blockchainu.

### 7.1 Funkční požadavky

Funkční požadavky specifikují, co by systém měl dělat (umět).

- FR1. Aplikace musí být schopna zobrazovat projekty které můžu podpořit.
- FR2. Aplikace musí umožnit zobrazit detail daných projektů.
- FR3. Aplikace musí umožňovat finanční podpoření daných projektů.
- FR4. Aplikace musí umožnit připojení kryptoměnové peněženky. (Metamask)
- FR5. Aplikace umožní zpět vybrat přispěvovatelům jejich prostředky, pokud se nenaplní finanční cíl ve stanoveném limitu.
- FR6. Aplikace pošle vybrané prostředky na adresu projektu, pokud se naplní cíl ve stanoveném časovém limitu.
- FR7. Aplikace umožní přidání projektu uživatelům.

### 7.2 Nefunkční požadavky

Nefunkční požadavky specifikují jaký by měl být systém aplikace, spíše než jeho specifické funkce.

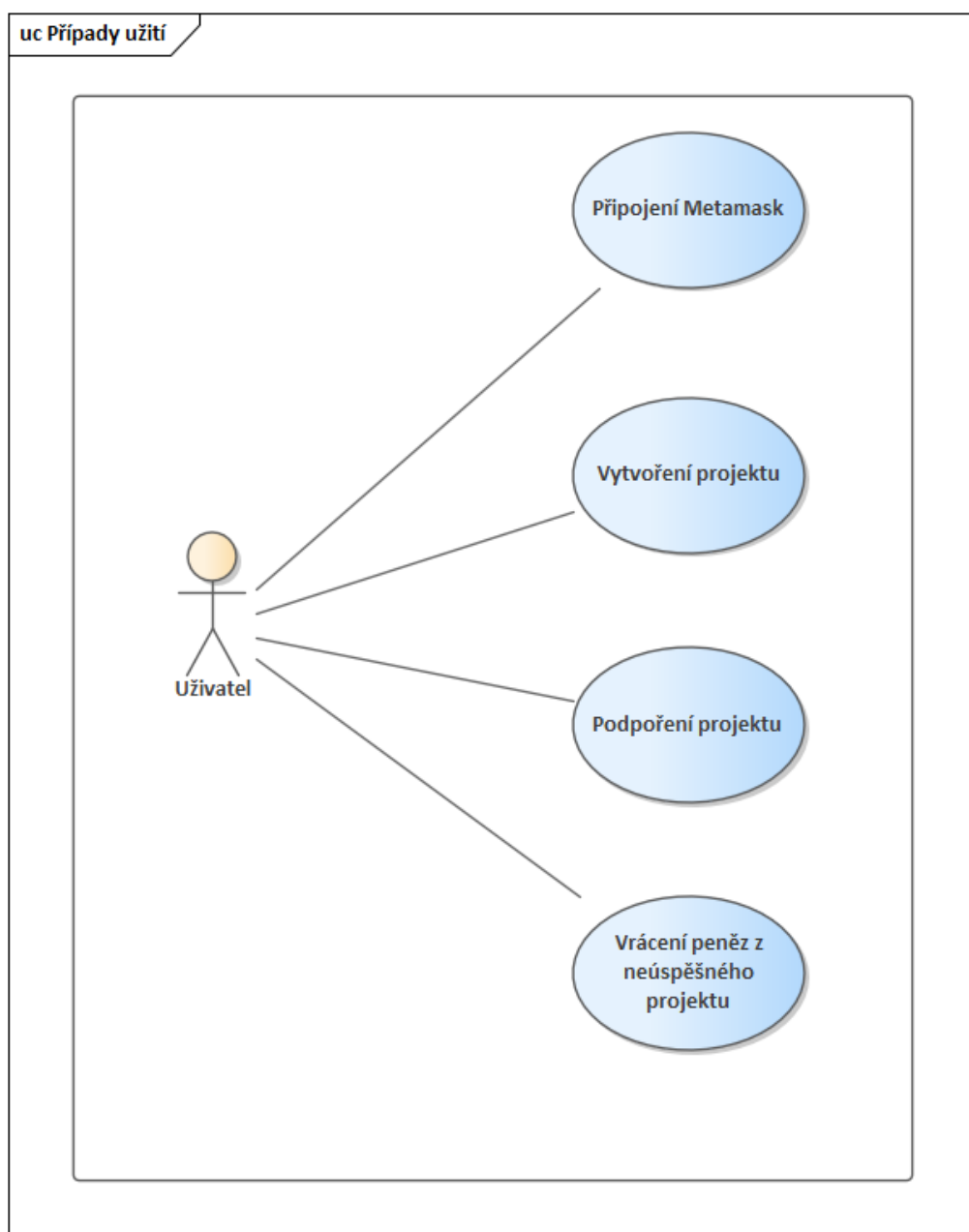
- NR1. Aplikace je cílená pro provoz na desktopech (notebooky a PC).
- NR2. Aplikace bude mít frontend pro interakci vytvořený ve Vue.
- NR3. Aplikace bude mít smart contracty napsané v Solidity.
- NR4. Aplikace bude nasazena na EVM kompatibilním blockchainu.
- NR5. Aplikace bude ukládat obrázky na decentralizovaném úložišti.

### 7.3 Diagram případu užití a scénáře

Případy užití popisují, jak uživatel aplikace dosáhne určitého cíle. Díky tomu je možné detailně popsat, jaké vstupy a výstupy můžeme očekávat.

Scénář případu užití je posloupnost kroků, které popisují interakci aplikace a uživatele, který ji používá.

Aplikace nerozlišuje různé typy uživatelů, má aplikace jediného aktéra který aplikaci používá.



Název: Připojení Metamask		
ID: UC01		
Charakteristika: Uživatel je schopen si připojit svoji kryptopeněženku k aplikaci		
Primární aktér: Uživatel		
Vedlejší aktér: Aplikace		
Vstupní podmínky:  Uživatel má spuštěný webový prohlížeč a v něm aplikaci, má také nainstalované rozšíření Metamask a založený účet v něm		
Výstupní podmínky:  Uživatel má připojenou peněženku k aplikaci, v pravém horním rohu může vidět svoji připojenou adresu		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Klikne na tlačítko připojit peněženku
2	Aplikace	Zobrazí vyskakovací okno Metamask
3	Uživatel	Potvrdí připojení Metamask
4	Aplikace	Zobrazí připojenou adresu v pravém horním rohu

Název: Vytvoření projektu		
ID: UC02		
Charakteristika: Uživatel je schopen vytvořit projekt, pro který chce vybrat finance		
Primární aktér: Uživatel		



Vedlejší aktér: Aplikace		
Vstupní podmínky:  UC01		
Výstupní podmínky:  Uživatel úspěšně přidal projekt, může si zobrazit jeho detail		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Klikne na tlačítko Vytvořit projekt
2	Aplikace	Zobrazí formulář pro vyplnění detailu projektu
3	Uživatel	Vyplní detaily a klikne na tlačítko publikovat projekt
4	Aplikace	Aplikace potvrdí přidání projektu a zobrazí jeho detail

Název: Podpoření projektu
ID: UC03
Charakteristika: Uživatel je schopen poslat finance na projekt, zobrazení daného příspěvku
Primární aktér: Uživatel
Vedlejší aktér: Aplikace
Vstupní podmínky:  UC01  Existuje alespoň jeden vytvořený projekt který lze podpořit
Výstupní podmínky:  Uživatel podpořil projekt svými financemi, jeho příspěvek se zobrazí na stránce projektu
Hlavní scénář:

Krok	Aktér	Popis
1	Uživatel	Klikne na tlačítko Podpořit projekt
2	Aplikace	Zobrazí okno Metamask
3	Uživatel	Vyplní částku a potvrdí odeslání financí
4	Aplikace	Potvrdí přijetí financí a zobrazí daný příspěvek

## 8 VÝVOJ PROTOTYPOVÉ APLIKACE

Cílem této kapitoly je seznámit čtenáře s použitými vývojovými nástroji a s kroky, které bylo nutné podniknout k vytvoření cílové aplikace pomocí specifikovaných vývojových nástrojů.

### 8.1 Instalace nástrojů nutných pro vývoj

Před zahájením vývoje bylo nutné mít nainstalované potřebné nástroje, na zařízení, na kterém probíhal vývoj.

#### 8.1.1 Nástroje použité pro vývoj

##### 8.1.1.1 Editor Visual Studio Code

Editor Visual Studio Code je populární textový editor, který v podobě rozšíření nabízí mnoho nástrojů pro usnadnění a zefektivnění vývoje. V editoru jsem využil rozšíření solidity, které nabízí zvýraznění syntaxu a také statickou analýzu daného kódu a také rozšíření vetur, které nabízí další funkce pro vývoj ve frameworku Vue.

##### 8.1.1.2 Node package manager

Jelikož součástí vývoje byl i vývoj webové aplikace, který zahrnuje instalaci knihoven, bylo nutné použít systém pro správu balíčků Node Package Manager (npm). Pro použití npm je nutná instalace běhového prostředí Node.js. Ke ověření správné instalace nástroje je možné použít příkaz:

```
npm --version
```

##### 8.1.1.3 Metamask

Metamask je softwarová peněženka pro kryptoměny, které slouží pro interakci s blockchainem. Umožňuje uživatelům přístup k jejich peněžence a interakci s decentralizovanými aplikacemi. Peněženku lze také snadno připojit k lokálnímu blockchainu a testovat s ní interakci s decentralizovanou aplikací, lze ji snadno nainstalovat jako rozšíření do velké řady webových prohlížečů.

### 8.1.2 Specifické nástroje použité pro vývoj na blockchainu

Pro vývoj a testování smart kontraktů byl použit nástroj Truffle, který slouží jako lokální vývojové prostředí, testovací nástroj a celkově usnadňuje vývoj v oblasti Ethereum blockchainu. Nástroj jsme schopni nainstalovat pomocí příkazu:

```
npm install -g truffle
```

Pro vytvoření projektové složky nástroje a jeho inicializace použijeme příkaz:

```
truffle init
```

Ganache je nástroj který vytvoří privátní Ethereum blockchain a umožní nám provádět všechny akce, které bychom prováděli na hlavní síti bez reálných poplatků. Poskytuje nám také přehledné grafické prostředí, ve kterém můžeme získat informace o naší síti a akcích na ní prováděných.

Běžný postup při vývoji je vytvoření struktury projektu pomocí Truffle a připojení tohoto projektu do Ganache pomocí konfiguračního souboru *truffle.config.js*. Ganache nám v tuto chvíli slouží jako lokální blockchain na který můžeme snadno kontrakty nasadit pomocí příkazu:

```
truffle migrate
```

Tento příkaz nám automaticky přeloží soubory kontraktů a nasadí je na síť podle toho jak specifikujeme v souborech ve složce migrations. Pokud si uložíme pracovní plochu Ganache připojenou k danému truffle projektu, můžeme jedním kliknutím snadno spustit dříve použitou konfiguraci.

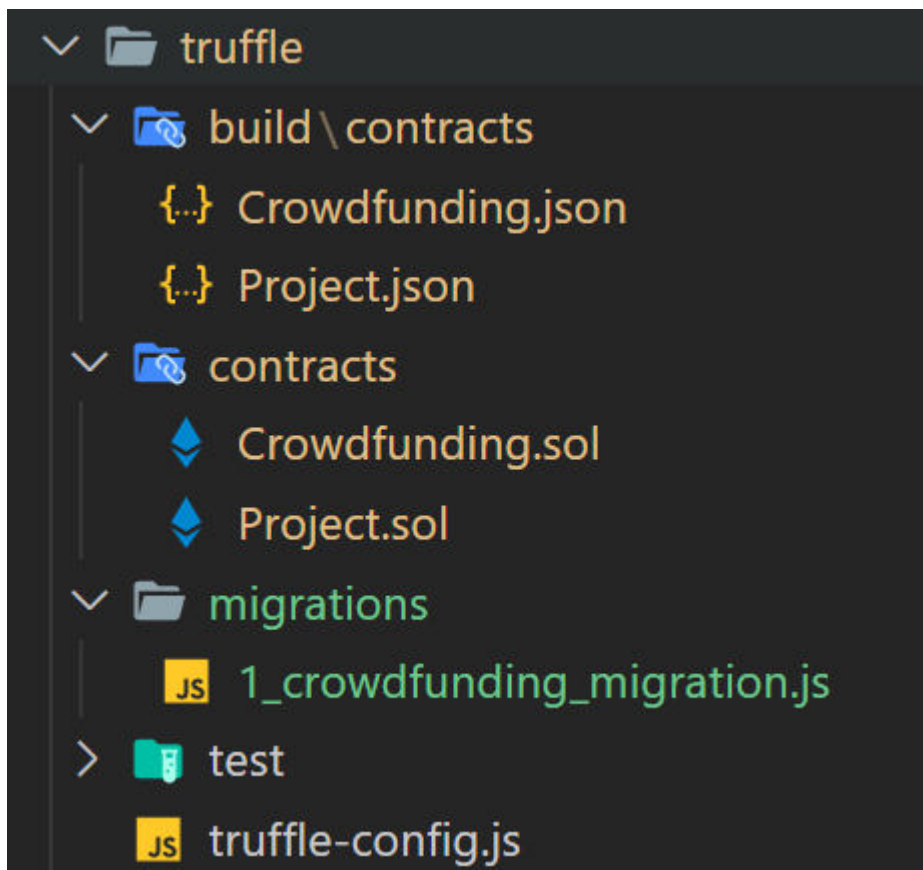
### 8.1.3 Struktura nástroje Truffle

Po provedení příkazu *truffle init*, se nám vygeneruje základní struktura složky, kterou poté dále upravujeme.

- *contracts*: V této složce pracujeme s jednotlivými Solidity soubory s koncovkou *.sol*, ve kterých definujeme logiku kontraktů a jejich funkce
- *migrations*: Migrations jsou javascriptové soubory, které nám pomáhají nasadit kontrakty na Ethereum. V průběhu času a vývoje projektu se tyto migrační skripty upravují a umožňují nám přidávat další kontrakty.
- *build*: Po zkompilování jednotlivých kontraktů se do této složky ukládají informace vzniklé při jejich kompilaci, v těchto souborech se nachází i ABI (application

binary interface), toto aplikační rozhraní potřebujeme pro definici jednotlivých kontraktů a definuje způsob jakým voláme funkce v kontraktu a jaké data z nich dostáváme zpět.

- *test*: Truffle umožňuje psát jednoduché testy ve dvou odlišných způsobech, a to pomocí Javascriptu, ve kterých většinou píšeme testy, které se více přibližují situacím v reálném světě a v Solidity, které nám umožňuje psát testy na podrobné scénáře.
- *truffle-config.js*: Konfigurační soubor se nachází v kořenovém adresáři složky, je to Javascriptový soubor, ve kterém se definují nastavení pro daný projekt jako adresu sítě, její port, id či verzi překladače Solidity.

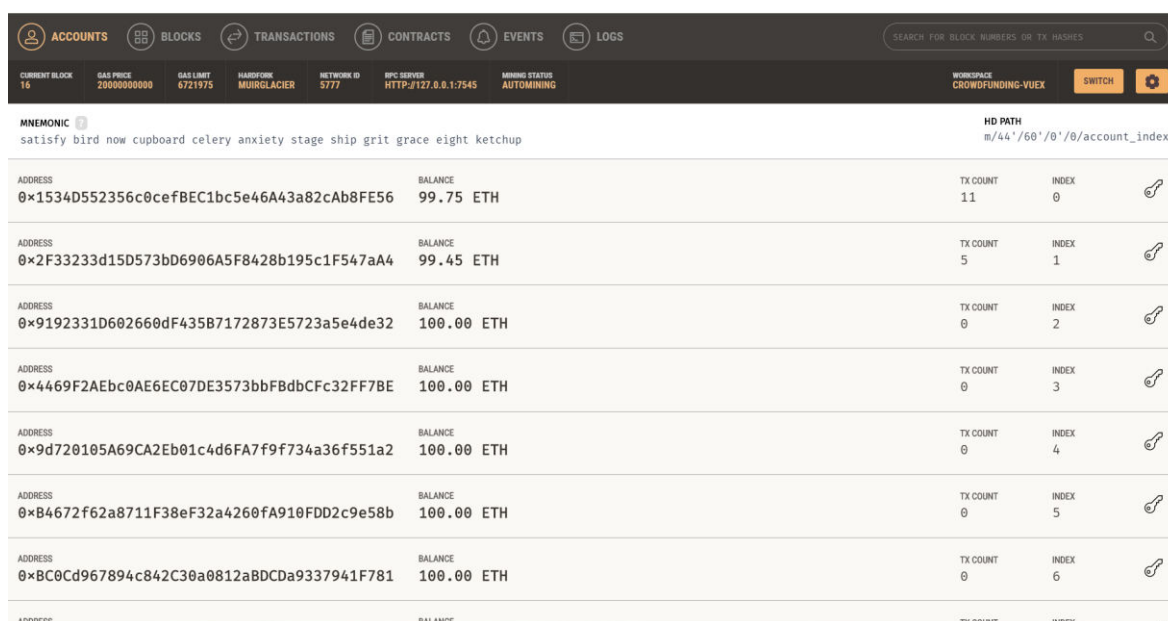


Obrázek 7 Struktura projektu Truffle

#### 8.1.4 Popis nástroje Ganache

Nástroj Ganache je možné jednoduše stáhnout ze stránek tohoto projektu, nástroj je nabízen i v konzolové verzi ganache-cli, pro lepší přehlednost a srozumitelnost jsem ale zvolil verzi desktopovou.

- *Accounts*: Jednotlivé účty a jejich adresy, které jsou použity k interakci z kontrakty, jejich zůstake, počet provedených transakcí a privátní klíč, který se používá pro připojení účtů do rozšíření Metamask.
- *Blocks*: Informace o jednotlivých blocích, datum jejich vytěžení a transakcích v nich obsažených.
- *Transactions*: Je zde zobrazen hash jednotlivých transakcí, adresa ze kterých byly zavolány, adresa kontraktu na kterou byla transakce provedena a typ transakce (např. volání kontraktu, vytvoření kontraktu).
- *Contracts*: Detaily jednotlivých kontraktů, které jsou definované v truffle projektu a informace o nich jako jejich adresa, data v nich uložená, informace o událostech či jestli jsou vůbec na síti nasazený.
- *Events*: Pokud se v kontraktu splní podmínky pro vyslání událostí, najdeme zde informace o těchto událostech
- *Logs*: Informace o záznamech v síti



ADDRESS	BALANCE	TX COUNT	INDEX
0x1534D552356c0cefBEC1bc5e46A43a82cAb8FE56	99.75 ETH	11	0
0x2F33233d15D573bD6906A5F8428b195c1F547aA4	99.45 ETH	5	1
0x9192331D602660dF435B7172873E5723a5e4de32	100.00 ETH	0	2
0x4469F2AEbc0AE6EC07DE3573bbFBdbCfc32FF7BE	100.00 ETH	0	3
0x9d720105A69CA2Eb01c4d6FA7f9f734a36f551a2	100.00 ETH	0	4
0xB4672f62a8711F38eF32a4260fA910FDD2c9e58b	100.00 ETH	0	5
0xBC0Cd967894c842C30a0812aBDCDa9337941F781	100.00 ETH	0	6

Obrázek 8 Ukázka uživatelského prostředí Ganache

### 8.1.5 Knihovny

Pro usnadnění vývoje a zvýšení kvality projektu bylo nutné nainstalovat knihovny jako jsou:

- *Ethers.js*: Tato kompaktní knihovna si klade za cíl snadnou interakci s Ethereum blockchainem a jeho ekosystémem. Díky ní můžeme definovat instance smart kontraktů přímo v javascriptovém kódu a umožňuje připojení k různému typu uzlů sítě.
- *Vue-router*: Umožňuje snadnou navigaci, v rámci Vue propojuje komponenty s cílovými cestami prohlížeče.
- *Vuex*: Implementuje návrhový vzor pro správu stavů aplikace a slouží jako centralizované úložiště pro správu všech komponentů aplikace s předem danými pravidly aby změna stavu aplikace byla předvídatelná a spolehlivá.
- *Ipfs-http-client*: Tato knihovna implementuje API pro interakci se systémem IPFS a také další sady pomocných funkcí.
- *Bootstrap*: Sada nástrojů kaskádových stylů a návrhových šablon založených na CSS a HTML, pro usnadnění vývoje uživatelského rozhraní.
- *Truffle-assertions*: Knihovna umožňující lepší testování smart kontraktů, podporuje testování typů revertu při chybě a testování eventů

## 8.2 Postup vývoje

Tato kapitola popisuje hlavní kroky, které bylo nutné podniknout pro implementaci a realizaci řešení projektu.

### 8.2.1 Volba vhodné platformy

Jako platformu pro nasazení decentralizované aplikace jsem si zvolil Ethereum, kód smart kontraktů, který vyvineme na hlavní síť je totiž snadno přenositelný na další vrstvy hlavní sítě či na jiné platformy, které podporují kompatibilitu tohoto kódu. Díky tomu můžeme operovat s jednou aplikací v rozsáhlém ekosystému řešení a nemusíme výslednou aplikaci omezovat pouze na jedinou síť. Ethereum jsem si také zvolil kvůli rozsáhlé dokumentaci a kvalitě jejich vývojových nástrojů a také díky rozsáhlé vývojářské komunitě, pomocí které je snadnější řešení vzniklých problémů při vývoji.

### 8.2.2 Implementace technologie smart kontraktů

Pro implementaci smart kontraktů na platformě Ethereum jsem si zvolil programovací jazyk Solidity. Tento jazyk je objektově orientovaný, staticky typovaný, podporuje dědičnost, knihovny a uživatelsky definované typy. Tento jazyk jsem si zvolil pro jeho

snadné použití, a i pro jeho popularitu, díky které je dokumentace a popsání případy užití tohoto jazyky větší než u jiných jazyků používaných pro vývoj na tuto platformu.

Jak již vyplývá z funkčních požadavků, je potřeba navrhnout řešení smart kontraktů tak, aby byl schopný uchovávat projekty a jejich finance a manipulovat s nimi. Tato část je rozdělena na dva kontrakty a to kontrakt `Crowdfunding.sol`, který v podstatě reprezentuje celou aplikaci, a všechna její data, pomocí něj jednotlivé projekty zakládáme a přidáváme je do vnitřního pole projektů tohoto kontraktu pomocí metody `createProject`, tato metoda má nastavenou viditelnost na `external`, což znamená že je funkci možno volat pouze z vnějšího prostředí. U každého jednotlivého projektu uchováváme řadu atributů jako název projektu, jeho popis, cílová částka, doba, za kterou chceme danou částku vybrat, hash obrázku, který daný projekt reprezentuje (hash obrázku získáme při jeho nahrání na úložiště IPFS při založení projektu), současný stav financí projektu, adresu, ze které byl projekt vytvořen a jeden ze tří stavů ve kterém se projekt může nacházet (stav získávání finančních prostředků, stav vypršení časového limitu a stav ve kterém byl projekt úspěšně zafinancován), aktuální stav finančních prostředků a jeho stav nelze při jeho vytvoření specifikovat, automaticky se v konstruktoru projektu nastaví na hodnotu 0 respektive na stav vybírání finančních prostředků pro stav projektu. Při vytvoření projektu, posíláme do sítě event, ve kterém jsou obsažené informace o daném projektu, který se poté ukládá do logu na blockchainu. Seznam všech vytvořených projektů z vnitřního pole kontraktu získáváme pomocí metody `returnAllProjects`.



```
contract Crowdfunding {
    Project[] private projects;

    event ProjectCreated (
        address contractAddress,
        address projectOwner,
        string projectTitle,
        string projectDesc,
        uint256 deadline,
        uint256 goalAmount,
        string imgHash
    );

    function createProject(
        string memory title,
        string memory description,
        uint durationInDays,
        uint amountToRaise,
        string memory imgHash
    ) external {
        uint timeGoal = block.timestamp + (durationInDays * 1 days);
        Project newProject = new Project payable(msg.sender), title, description, timeGoal, amountToRaise, imgHash);
        projects.push(newProject);

        emit ProjectCreated(address(newProject), msg.sender, title, description, timeGoal, amountToRaise, imgHash
    );
    }
}
```

Obrázek 9 Ukázka kontraktu Crowdfunding.sol a funkce pro vytvoření projektu

Druhý kontrakt Project.sol reprezentuje instanci každého projektu zvlášť a obsahuje funkce pro manipulaci s projektem, jako funkce pro přispívání, vyplácení úspěšných projektů, vrácení peněz z neúspěšných projektů či funkce pro získávání detailů o daném projektu.

Pro kontrolu předpokladů pro vykonání funkce je zde použito zabudovaných funkcí *modifier* a *require*. *Modifier* je funkce obsahující funkci *require*, která používáme pro snížení redundance kódu, v případě že je potřeba použít stejnou podmínky *require* vícekrát, tato funkce dále používá speciální symbol `_`, který reprezentuje tělo funkce, které se vykoná při naplnění podmínek. Tyto funkce používám pro ověření správného stavu, ve kterém se projekt nachází, k ověření zda adresa ze které je projekt podpořen není identická s adresou vlastníka projektu (v případě anonymity adres a jejich snadných založení je nemožné kontrolovat, zda adresa nijak nesouvisí s vlastníkem projektu a tento problém není možné eliminovat), či jestli je adresa dříve na daný projekt přispěla a je oprávněna k vrácení peněz.

Pro rozlišení adres, které nemohou manipulovat s financemi a adresy které s financemi manipulovat mohou, používáme keyword *payable*, adresa vlastníka projektu je tedy označena tímto keywordem, aby bylo možné později na tuto adresu vyplatit vybrané peníze. Adresy jednotlivých přispěvovatelů ukládáme do struktury *mapping*, která je key,

value strukturou. Solidity podporuje explicitní konverzi adres, používáme tedy při vrácení peněz převod adresy přispěvovatele na adresu *payable*.

Po každém vykonání funkce *contribute*, která obsluhuje přispívání na dané projekty a po naplnění vstupních podmínek přidá adresu a přispěnou částku do struktury přispěvovatelů, voláme funkci *checkStateOfProject*, která buď na základě toho, že vybrané finance již dosahují požadované částky, přepne stav projektu do stavu naplnění částky nebo na základě vypršení časového období přepne stav projektu na vypršení časového limitu.

```
function contribute() external inState(State.Fundraising) payable {
    require(msg.sender != owner, "You are trying to send money to your own project");

    contributions[msg.sender] = msg.value;
    currentBalance += msg.value;
    contributorsArray.push(msg.sender);
    emit FundingReceived(msg.sender, msg.value, currentBalance);
    checkStateOfProject();
}
```

Obrázek 10 Ukázka funkce pro podpoření projektu

Při přepnutí stavu projektu do úspěšného naplnění částky se zavolá funkce *payOut*, která odešle všechny vybrané prostředky na adresu zakladatele projektu a nastaví aktuální stav financí projektu na 0. Po vypršení časového limitu projektu a nenaplnění cílové částky má možnost uživatel, který na projekt přispěl zavolat funkci *getRefund*, pro navrácení financí, které do projektu poslal.

```
function getRefund() public inState(State.Expired) returns (bool) {
    require(contributions[msg.sender] > 0);

    uint amountToRefund = contributions[msg.sender];
    contributions[msg.sender] = 0;

    if (!payable(msg.sender).send(amountToRefund)) {
        contributions[msg.sender] = amountToRefund;
        return false;
    } else {
        currentBalance -= amountToRefund;
    }

    return true;
}
```

Obrázek 11 Ukázka funkce pro vrácení finančních prostředků

Funkce *getDetails*, vrací všechny informace o daném projektu a používáme ji pro získání informací a jejich výpis na frontendu.

### 8.2.3 Testování smart kontraktů

Díky faktu, že smart kontrakty jsou neměnné, než je nasadíme na blockchain je důležité nejdřív jejich funkcionalitu důkladně otestovat, pro jejich testování smart kontraktů používám nástroj truffle a jeho zabudovaný příkaz *truffle test*, po zavolání tohoto příkazu truffle spustí testovací soubory ze složky tests.

```
it("Should let contribute user from different address and emit event FundingReceived", async () => {
  const contributeTx = await contractProject.contribute({'from':accounts[1], 'value': 10})

  let valueFromMapping = await contractProject.contributions(accounts[1]);
  assert(valueFromMapping.words[0] === 10)

  truffleAssert.eventEmitted(contributeTx, 'FundingReceived', (ev) => {
    return ev.contributor == accounts[1] && ev.amount.words[0] == 10;
  });
});
```

Obrázek 12 Ukázka testovací metody z nástroje truffle

Testy jsou napsané v Javascriptu a používají instance kontraktů, které v testech nejdřív nasadíme na testovací blockchainem a poté na nich voláme jejich metody. Testy voláme chronologicky a jak postupně měníme stav uvnitř kontraktu tak ho testujeme.

```
=====
> Compiling .\contracts\Crowdfunding.sol
> Compiling .\contracts\Crowdfunding.sol
> Compiling .\contracts\Project.sol
> Compiling .\contracts\Project.sol
> Artifacts written to C:\Users\hrome\AppData\Local\Temp\test--15764-1Y66LVDpElW0
> Compiled successfully using:
  - solc: 0.8.13+commit.abaa5c0e.Emscripten.clang

Contract: Crowdfunding and Project
  ✓ Should deploy smart contract
  ✓ Let user create project and emit ProjectCreated event (1219ms)
  ✓ Should let contribute user from different address and emit event FundingReceived (1105ms)
  ✓ Should not let contribute the owner of the project (1246ms)
  ✓ Should emit CreatorPaid event and change the state of the project to Successfull (1114ms)

5 passing (5s)
```

Obrázek 13 Ukázka z výpisu funkce truffle test

Pro lepší otestování eventů kontraktu používám knihovnu truffle-assertions, která umožňuje jednoduše získat atributy z vyslaného eventu, které poté mohou testovat.

## 8.3 Webová aplikace

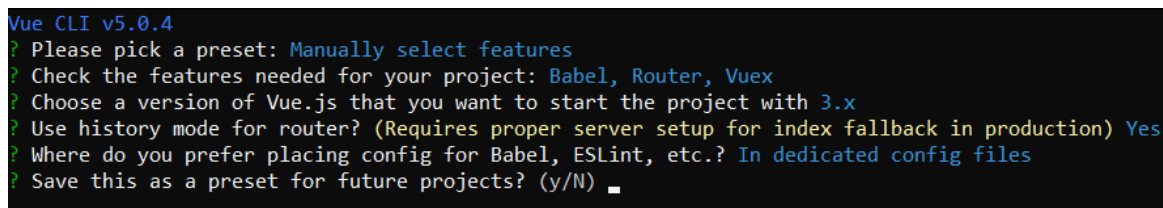
Při vývoji frontendu decentralizovaných aplikací je možno použít běžných knihoven v rámci webových technologií, výsledná aplikace má běžet ve webovém prohlížeči, proto jsem zvolil javascriptový framework Vue.js, který tyto požadavky splňuje.

### 8.3.1 Tvorba kostry projektu

Pro vytvoření projektu ve Vue, jsem použil nástroj pro příkazovou řádku vue-cli, pro vytvoření projektu jsem použil příkaz:

*vue create prototyp-aplikace*

Poté jsem pomocí nabídnutého menu vybral jednotlivé komponenty aplikace, které jsem chtěl použít, a to možnosti Babel (kompilátor javascriptu), router (knihovna pro navigaci) a vuex (správa stavu aplikace), dále jsem vybral verzi Vue, kterou chci vyvíjet a to verzi 3.x.



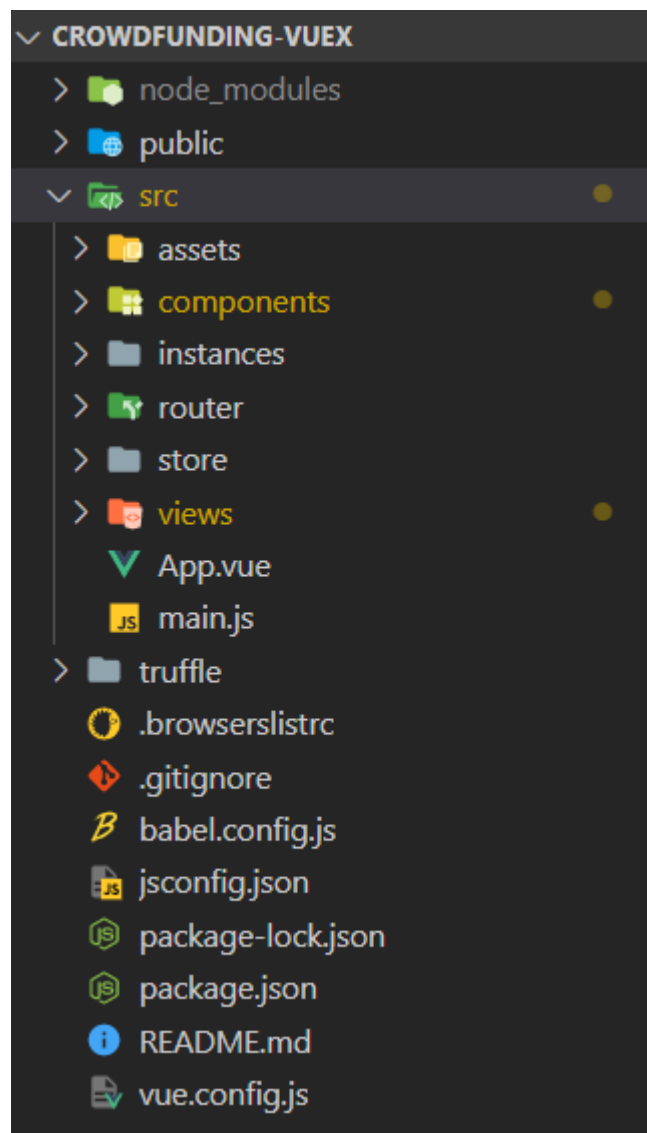
```
Vue CLI v5.0.4
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex
? Choose a version of Vue.js that you want to start the project with 3.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) y
```

Obrázek 14 Ukázka z nástroje vue-cli

Po úspěšném dokončení příkazu se nám vytvořil kořenový adresář projektu, ve kterém se nachází konfigurační soubory projektu a složky ve kterých se nachází další náležitosti aplikace.

- *node\_modules*: Tato složka obsahuje všechny zabudované moduly a externí knihovny
- *public*: Adresář public obsahuje soubor index.html, ve kterém je element `<div id="app"></div>`, který slouží jako nosný prvek všech Vue komponent, při startu aplikace se na něj aplikace připojí pomocí metody *mount*
- *src*: Tento adresář obsahuje další podsložky a soubory, které nesou zdrojový kód aplikace.
  - *assets*: Obrázky použité v aplikaci
  - *components*: Složka obsahující komponenty, pro jednotlivé části aplikace
  - *instances*: Složka obsahující pomocné javascriptové soubory, pro vytváření instancí kontraktů pomocí knihovny ethers.js

- *router*: složka knihovny vue-router, obsahuje soubor *index.js*, ve kterém definujeme jednotlivé cesty pro navigaci v aplikaci
- *store*: Adresář pro soubor knihovny vuex, v souboru *index.js*, definujeme jednotlivé stavy aplikace a metody které tyto stavy obsluhují
- *views*: Obsahuje Vue soubory reprezentující jednotlivé pohledy, reprezentující stránky aplikace
- *App.vue*: Kořenový soubor Vue aplikace
- *truffle*: Projektová složka nástroje Truffle
- *main.js*: Inicializuje kořenový soubor aplikace a je také zodpovědný za registrování knihoven do aplikace
- *package.json*: Obsahuje seznam všech použitých balíčků a knihoven



Obrázek 15 Stromová struktura aplikace

### 8.3.2 Práce s vnitřním stavem aplikace Vuex

Pro správu vnitřního stavu a omezení redundance kódu aplikace používám knihovnu Vuex, způsob, jakým nakládáme s daty v této knihovně se dělí do několika základních konceptů a to:

- *State*: Je to objekt, ve kterém ukládáme se stavem aplikace, tento stav je poté díky systému pluginů Vue vložen do všech potomků hlavního komponentu.

```
export default createStore({
  state: {
    provider: null,
    signer: null,
    account: null,
    projects: [],
    loading: false,
    currentCID: null
  },

```

Obrázek 16 Ukázka definování vnitřních stavů aplikace pomocí knihovny Vuex

- *Mutations*: Metody pomocí, kterých měníme hodnotu stavů aplikace, mutations voláme pomocí funkce *commit()*, která jako argument přebírá název mutation ve formě řetězce, popřípadě další hodnoty na které chceme stav změnit

```
mutations: {
  UPDATE_USER(state, payload) {
    state.provider = payload.provider
    state.signer = payload.signer
    state.account = payload.account
  },
  PUSH_PROJECT(state, project) {
    state.projects.push(project)
  },
  CLEAR_PROJECTS(state) {
    state.projects = []
  },

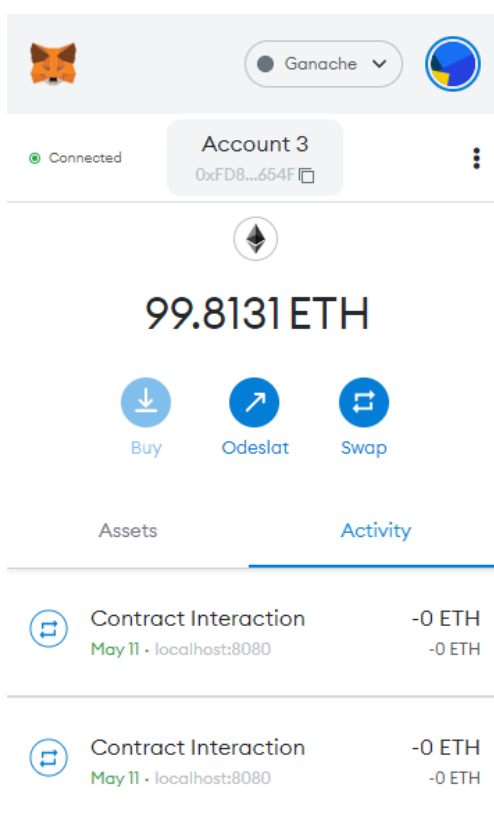
```

Obrázek 17 Ukázka funkcí mutation

- *Actions:* Actions jsou podobné mutations, ale neměníme pomocí nich stavy a dokáží obsluhovat asynchronní operace, pokud chceme změnit stav aplikace vně metody actions, voláme opět funkci commit na danou funkci mutations

### 8.3.3 Připojení blockchainu k aplikaci a interakce s kontrakty

Po nainstalování rozšíření Metamask si zde vytvoříme účet, a poté pro připojení našeho lokálního blockchainu do prohlížeče musíme přidat novou síť do rozšíření, která reflektuje náš lokální blockchain a jeho adresu. Poté je nutné přidat do rozšíření jednotlivé účty vygenerované v Ganache, to provedeme pomocí importu privátního klíče.



Obrázek 18 Rozhraní rozšíření Metamask

K připojení k blockchainu v kódu využíváme objekt *window.ethereum*, který se definuje rozšíření Metamask při spuštění v prohlížeči. Metamask to tohoto objektu přidá aplikační rozhraní, pomocí kterého se můžeme dotazovat na jednotlivé účty a provádět transakce. Při zavolání metody *connectEthereum*, definujeme pomocí metody knihovny ethers.js nového poskytovatele, kterého získáme právě s objektu *window.ethereum*, dalším krokem je poté

dotázání se na účty uživatele pomocí metody *eth\_requestAccounts*, ze získaných hodnot nastavíme stav poskytovatele, podpisovatele a účtu.

```
async connectEthereum({commit}) {  
  try {  
    let ethersProvider = new ethers.providers.Web3Provider(window.ethereum);  
    let provider = markRaw(ethersProvider)  
    if(!provider) {  
      throw Error("Metamask not installed")  
    }  
  
    await provider.send("eth_requestAccounts", [])  
    let signer = provider.getSigner()  
    let account = await signer.getAddress()  
  
    commit('UPDATE_USER', {provider, signer, account})  
  }  
  catch(error) {  
    console.log(error)  
  }  
},
```

Obrázek 19 Ukázka metody pro připojení účtu

Interakci s kontrakty musíme vždy vytvořit danou instanci kontraktu a k tomu potřebujeme aplikační binární rozhraní daného kontraktu (abi), které definuje jednotlivé funkce a vlastnosti kontraktu, adresu kontraktu v síti a instanci poskytovatele, poté vytvoříme novou instanci pomocí knihovny ethers.js a kódu:

```
const instance = new ethers.Contract(address, abi, provider)
```

K vytváření těchto instancí používám pomocné soubory ve složce instances, aby nedocházelo k zbytečnému opakování kódu. Po vytvoření této instance na ni můžeme volat metody definované v abi, pro získání všech projektů voláme metodu *returnAllProjects*, ke každému z vrácených projektů poté přidáme jeho instanci a uložíme do pole projektů, které je definované jako stav aplikace. Pro získání informací, které používáme pro zobrazení seznamu podporovatelů projektu, zde ještě vytváříme nový objekt s těmito daty.



```
crowdfundInstance.returnAllProjects().then((projects) => {
  projects.forEach((address) => {
    const projectInstance = ProjectInstance(state.signer, address)
    projectInstance.getDetails().then((details) => {
      const project = new Object();
      project.details = details
      project.contract = projectInstance;
      let contributorsValue = [];
      project.details.projectContributors.forEach((address) => {
        projectInstance.returnContributorValue(address).then((res) => {
          const value = new Object();
          value.address = address;
          value.value = ethers.utils.formatEther(res.toString())
          contributorsValue.push(value)
        })
      })
      project.contributorsValue = contributorsValue
      commit('PUSH_PROJECT', project)
    })
  })
})
```

Obrázek 20 Ukázka funkce pro získávání projektů z blockchainu

Pro volání funkcí jako je vytvoření projektů musíme z objektu poskytovatele získat tzv. podpisovatele, což je abstrakce ethereového účtu, které umožňuje podpisovat a posílat jednotlivé transakce, transakce je potřeba podpisovat v případě že se mění vnitřní stav kontraktu.

Pro vytvoření nového projektu převezmeme data z formuláře, obrázek nahraný uživatel poté nahrajeme na decentralizované úložiště IPFS, k tomuto účelu využívám knihovnu `ipfs-http-client`, jako poskytovatele služby, abych nemusel provozovat vlastní uzel IPFS, používám službu Infura, která poskytuje vlastní bránu. Nejdříve vytvořím instanci klienta pomocí funkce:

```
const client = create({ host: 'ipfs.infura.io', port: 5001, protocol: 'https' })
```

Pomocí tohoto klienta a volám funkce z API dané knihovny, pro nahrání na síť stačí zavolat funkci `add` na klientovi a přidat požadovaný soubor jako parametr. Z výsledků poté uložím unikátní identifikátor do vnitřního stavu aplikace a ten poté uložím k uložení do smart kontraktu.

```
async uploadFile({commit}, file) {
  const client = create({ host: 'ipfs.infura.io', port: 5001, protocol: 'https' })
  commit('LOADING_TRUE')
  try {
    await client.add(file).then((res) => {
      commit('CHANGE_CID', res.path)
    })
  }
  catch(err) {
    console.log(err)
    commit('LOADING_FALSE')
  }
}
```

Obrázek 21 Funkce pro nahrání obrázků na úložiště IPFS

Další funkcí která vyžaduje změnu hodnot v kontraktu je funkce pro přispívání projektu, tu voláme pomocí asynchronní funkce typu action *fundProject()*, tato funkce pracuje s vnitřním aplikace, kde si nejdřív pomocí indexu z argumentu funkce, vyhledá instanci kontraktu daného projektu a pak z dané instance zavolá metodu *contribute()* a předá do jako argument hodnotu v kryptoměně Ethereum kterou chceme poslat. Jelikož ethers pracuje jako z výchozím hodnotou kryptoměny Ethereum s její hodnotou Wei (1 ETH =  $10^{18}$  Wei), parsujeme tuto hodnotu pomocí funkce knihovny *ethers.utils.parseEther()*.

Poslední funkce, která se nachází v modulu knihovny vuex a přímo se dotýká blockchainu je funkce pro vyplacení částky z neúspěšného projektu. Tato metoda funguje stejně jako metoda předchozí, jen volá funkci kontraktu *getRefund*.

### 8.3.4 Navigace a jednotlivé cesty v aplikaci

Navigaci obsluhuje knihovna vue-router, jednotlivé cesty jsou popsány v souboru *router/index.js*.

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/create',
    name: 'create',
    component: CreateView
  },
  {
    path: '/about',
    name: 'about',
    component: AboutView
  },
  {
    path: '/detail/:index',
    name: 'detail',
    component: DetailView,
    params: true
  },
]

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})

export default router
```

Obrázek 22 Definice jednotlivých cest v aplikaci

### 8.3.5 Komponenty a zobrazení

V rámci projektu jsou jednotlivé části rozděleny do komponent a zobrazení, zobrazení reprezentují celé stránky, na které se můžeme dostat a jsou zaregistrované v navigaci routeru. Každé zobrazení či komponentu reprezentuje soubor *vue*, který se dělí na část *template*, která reprezentuje HTML kód, *script* sekci, ve které voláme jednotlivé funkce, registrujeme další komponenty, které v daném souboru používáme či definujeme vlastnosti souboru. A poslední částí *style*, kde definujeme kaskádové styly souboru. Zobrazení v mém projektu dělím do 4 částí:

- *HomeView*: Toto zobrazení reprezentuje úvodní stránku, na které jsou zobrazeny jednotlivé projekty, je to také stránka, na kterou se uživatel dostane při prvotním načtení aplikace.

- *CreateView*: V tomto zobrazení je formulář, který se používá pro vytvoření nového projektu.
- *DetailView*: Detailní stránka projektu, jsou na ni zobrazeny detaily projektu a také formulář pro příspěví projektu.
- *AboutView*: Stránka obsahující jednoduché informace o webové aplikaci

Komponenty jsou části aplikace, které reprezentují menší úseky funkcionality či HTML kódu, projekt obsahuje těchto několik menších celků a každá je zodpovědná za obsluhu dané funkcionality:

- *CreateProjectForm*: Obsluhuje formulář pro vytváření projektu
- *FundProjectForm*: Formulář pro posílání příspěvků pro projekt
- *LoadingScreen*: Zobrazuje animaci v případě, že se čeká například na potvrzení transakce uživatelem
- *Navbar*: Navigační panel pro přesun mezi zobrazeními
- *ProjectCard*: Zobrazuje seznam všech projektů v podobě karet
- *ProjectDetails*: Zobrazuje detailní informace o projektu

```
<script>
import store from "@/store"
import { Buffer } from "buffer"
import LoadingScreen from "../LoadingScreen.vue"
export default {
  data() {
    return {
      newProject: {}
    };
  },
  methods: {
    async handleSubmit() {
      await store.dispatch("createProject", { newProject });
    },
    updatePhoto(files) {
      const file = files[0];
      const reader = new window.FileReader();
      reader.readAsArrayBuffer(file);
      reader.onloadend = () => {
        const buffer = Buffer(reader.result);
        this.newProject.file = buffer;
        console.log("buffer", this.newProject.file);
      };
    }
  },
  components: { LoadingScreen }
}
</script>
```

Obrázek 23 Ukázka části scriptu komponenty CreateProjectForm

## 8.4 Návrh uživatelského rozhraní

Pro potřeby demonstrace aplikace jsem vytvořil uživatelské rozhraní, které z velké části využívá elementy z knihovny Bootstrap.

### 8.4.1 Hlavní obrazovka

Hlavní obrazovka slouží jako prostor pro zobrazení všech projektů, každý projekt má k dispozici zobrazení základních informací a je aktuální stav. Stavys jsou zobrazeny pomocí 3 barev a každá kategorie má svojí vlastní ikonu. Uživatel také může vidět ukazatel průběhu, který se dynamicky upravuje podle aktuálně vybraných financí. Kliknutím na název projektu je uživatel přesměrován na detailní stránku projektu.

DeCrowdfunding

HomeCreate projectAbout

0x6A3545936414950bd3d85e7a301B6282Cc38d37E

Make arcade games great again

Fundraising

We are aiming to restore some of the old arcane games and present it to you in new space. You can except classic arcade games like Space Invaders or Pinball, but also few hidden gems. We are experienced with restoring old arcade machines, we are doing it since 2008. You can find more info about us at [www.arcaderestore.com](http://www.arcaderestore.com)


From: 0x8aFc918E7601EA956eC01B0CE5540bB8A95ac03D

37 %


Currently raised: 3.64 ETH

Goal: 10.0 ETH

Valid until: 60 days



Successfull




Fund our album - FunkyCrypto

Project owner : 0x8a...


Description : Hello, we our funky crypto and we would like to release our new album focused on...

100 %

Goal : 5.0 ETH



Fundraising



OldMovies - Viewing movie relics from the past every Wednesday

Project owner : 0x8a...


Description : As you know, our old film player suddenly broke and since we are non-profit orga...

Valid until 30 days


96 %

Currently raised: 4.78 ETH

Goal : 5.0 ETH



Expired



Books for the needed - Funding the transportations of our books

Project owner : 0x8a...

Description : Hi, we are currently trying to relocate part of our book library to different pl...

Project already expired

25 %

Currently raised: 1.0 ETH

Goal : 4.0 ETH

Obrázek 24 Ukázka hlavní stránky projektu

#### 8.4.2 Stránka pro vytvoření projektu

Na této stránce může uživatel nalézt formulář pro vytvoření projektu, uživatel má k dispozici na vyplnění standartní informace o projektu, kategorii si volí pomocí zvolení jedné z ikon. Před odesláním formuláře musí uživatel vyplnit všechny jeho náležitosti.

DeCrowdfunding

HomeCreate projectAbout

0x6A3545936414950bd3d85e7a301B6282Cc38d37E

## Enter basic information about the project





Title

Make arcade games great again

Description

We are aiming to restore some of the old arcane games and present it to you in new space. You can except classic arcade games like Space Invaders or Pinball, but also few hidden gems. We are experienced with restoring old arcade machines, we are doing it since 2008. You can find more info about us at [www.arcaderestore.com](http://www.arcaderestore.com)

Categories



Duration in days

60

Amount to raise in ETH

10

Img of your project

Vybrat soubor

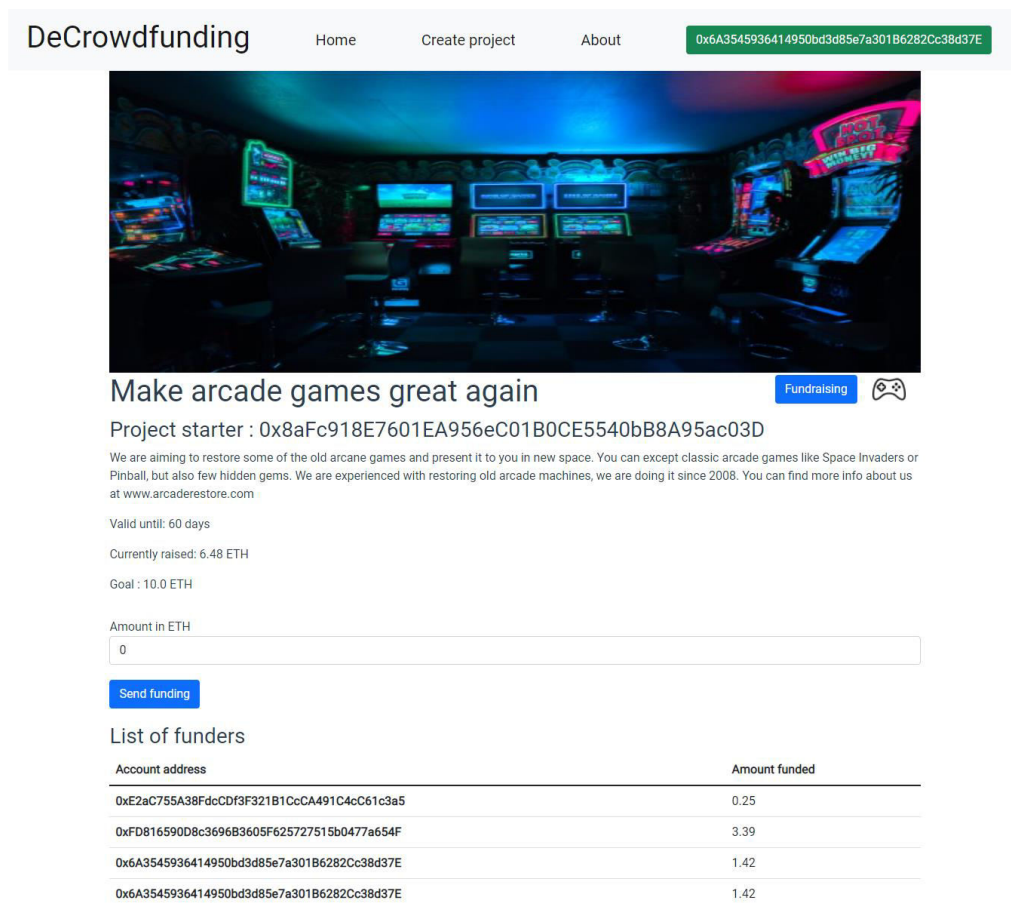
games.jpg

Submit

Obrázek 25 Ukázka stránky s formulářem pro vytvoření projektu

### 8.4.3 Detail projektu

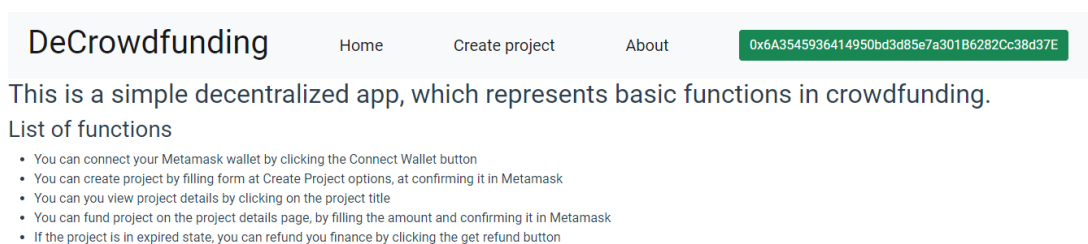
Stránka pro detail projektu zobrazuje detailní informace projektu a umožňuje uživateli poslat příspěvek či případně provést vrácení peněz. V dolní části také zobrazuje seznam adres, které na projekt přispěli a jejich částku.



Obrázek 26 Stránka pro zobrazení detailu projektu

#### 8.4.4 Stránka zobrazující informace o aplikaci

Tato stránka zobrazuje jednoduchý výpis akcí, které uživatel může na stránce provádět.



Obrázek 27 Ukázka stránky informací o projektu



## **8.5 Vyhodnocení významu projektu**

## ZÁVĚR

V této bakalářské práci byl teoreticky i následně prakticky představeny koncepty decentralizovaných aplikací a problematiky jejich vývoje.

Teoretická část seznámila čtenáře s problematikou blockchainu, smart kontraktů a decentralizovaných aplikací. Byl nastíněn způsob, jakým blockchain funguje a jeho jednotlivé algoritmy potvrzování konsensu sítě. Následně byl představen koncept smart kontraktů a jejich specifik. Charakterizována byla taky historie webu, na navazuje popis decentralizovaných aplikací jejich vlastnosti, pro pohled na decentralizované aplikace v dnešní době, byla rozebrána míra jejich decentralizovanosti, byly představeny platformy a jejich požadavky a poté byl vnesen vhléd v podobě rozebrání aplikací z několika odvětví.

V praktické části je popsán vývoj decentralizované aplikace, po úvodním představení řešeného problému, je zadání podrobeno analýze požadavků a scénářů, které se v aplikaci odehrávají. Po specifikaci požadavků následuje část, která popisuje jednotlivé nástroje a jejich instalaci. Poté již následuje popis jednotlivých kroků při vývoji aplikace, jako je návrh smart kontraktů, jejich testování a interakce s nimi ve webové aplikaci, tyto kroky jsou podloženy úryvky kódů, které ukazují způsob komunikace aplikace s blockchainem a způsob práce s funkcemi v aplikaci. V předposlední části práce následuje ukázka návrhu uživatelského rozhraní.

## SEZNAM POUŽITÉ LITERATURY

- [1] NAKAMOTO, Satoshi. *Bitcoin: A peer-to-peer electronic cash system* [online]. 2008 [cit. 2022-04-20]. Dostupné z: <http://www.bitcoin.org/bitcoin.pdf>
- [2] ADAM, Karel. *Výroba a spotřeba elektrické energie v roce 2020* [online]. [cit. 2022-04-20]. Dostupné z: <https://www.czso.cz/csu/xb/vyroba-a-spotreba-elektricke-energie-v-roce-2020>
- [3] *Proof-of-work (PoW)* [online]. 2022 [cit. 2022-04-20]. Dostupné z: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/>
- [4] ANTONOPOULOS, Andreas M. a Gavin WOOD. *Mastering Ethereum: building smart contracts and DApps*. Sebastopol, CA: O'Reilly Media, 2018, 1 online zdroj (xxxv, 384 stran). ISBN 9781491971918. Dostupné také z: <https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=1935734&authtype=ip,shib&custid=s3936755>
- [5] BASHIR, Imran. *Mastering blockchain: a deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, ethereum, and more, 3rd Edition*. 3rd ed. Birmingham: Packt Publishing, Limited, 2020, 1 online zdroj (xx, 788 stran). ISBN 9781839211379. Dostupné také z: <https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=2597859&authtype=ip,shib&custid=s3936755>
- [6] *Proof-of-stake (PoS)* [online]. 2022 [cit. 2022-04-20]. Dostupné z: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>
- [7] *Introduction to smart contracts* [online]. 2022 [cit. 2022-04-20]. Dostupné z: <https://ethereum.org/en/developers/docs/smart-contracts>
- [8] SZABO, Nick. Formalizing and Securing Relationships on Public Networks. *First Monday*. 2(9). Dostupné z: doi:<https://doi.org/10.5210/fm.v2i9.548>
- [9] *Mastering Blockchain*. O'Reilly Media, 2020. ISBN 9781492054702.
- [10] DEBIT, Nader. *What is Web3? The Decentralized Internet of the Future Explained* [online]. 8.září 2021 [cit. 2022-04-20]. Dostupné z: <https://www.freecodecamp.org/news/what-is-web3>
- [11] *Web2 vs Web3* [online]. 2022 [cit. 2022-04-20]. Dostupné z: <https://ethereum.org/en/developers/docs/web2-vs-web3/>
- [12] *Architecting Modern Decentralized Applications* [online]. 15. července 2021 [cit. 2022-04-20]. Dostupné z: <https://medium.com/hexamount/architecting-modern-decentralized-applications-52b3ac3baa5a>
- [13] SAVCHENKO, Nikita. *Decentralized Applications Architecture: Back End, Security and Design Patterns* [online]. 2. dubna 2019 [cit. 2022-04-20]. Dostupné z:

<https://www.freecodecamp.org/news/how-to-design-a-secure-backend-for-your-decentralized-application-9541b5d8bddb>

[14] *DApps: Crypto Basics For All Part 1* [online]. 15.května 2021 [cit. 2022-04-20]. Dostupné z: <https://medium.com/crypto-wisdom/dapps-crypto-basics-for-all-part-1-6fc794e4d765>

[15] *Introduction to dapps* [online]. 2022 [cit. 2022-04-20]. Dostupné z: <https://ethereum.org/en/developers/docs/dapps/>

[16] SETH, Shobith. *Public, Private, Permissioned Blockchains Compared* [online]. 29. ledna 2021 [cit. 2022-04-20]. Dostupné z: <https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/>

[17] STRELENKO, Oleg. *Blockchain and Transaction Speed: Why Does it Matter?* [online]. [cit. 2022-04-20]. Dostupné z: [https://medium.com/@s\\_o\\_s/blockchain-and-transaction-speed-why-does-it-matter-80bfd100fa89](https://medium.com/@s_o_s/blockchain-and-transaction-speed-why-does-it-matter-80bfd100fa89)

[18] *How to design a blockchain application architecture?* [online]. [cit. 2022-04-20]. Dostupné z: <https://www.leewayhertz.com/how-to-design-a-blockchain-app-architecture/>

[19] *Centralized, Decentralized and Distributed networks* [online]. [cit. 2022-04-25]. Dostupné z: <https://berty.tech/fr/faq>

[20] *A simple Blockchain structure.* [online]. In: . [cit. 2022-04-25]. Dostupné z: [https://www.researchgate.net/figure/A-simple-Blockchain-structure\\_fig2\\_325841576](https://www.researchgate.net/figure/A-simple-Blockchain-structure_fig2_325841576)

[21] *5 Ethereum Features* [online]. [cit. 2022-04-25]. Dostupné z: <https://medium.datadriveninvestor.com/5-ethereum-features-76da9462b319>

[22] *Total value locked all chains* [online]. [cit. 2022-04-25]. Dostupné z: <https://defillama.com/chains>

[23] LI, Z., R. ZHONG, Z.G. TIAN, Dai HONG-NING, Ali Vatankhah BARENJI a George Q. HUANG. Industrial Blockchain: A state-of-the-art Survey. *Robotics and Computer-Integrated Manufacturing*. 2021, **2021**(70). ISSN 0736-5845. Dostupné z: doi:<https://doi.org/10.1016/j.rcim.2021.102124>

[24] *Smart contract languages* [online]. [cit. 2022-04-25]. Dostupné z: <https://ethereum.org/en/developers/docs/smart-contracts/languages/>

[25] *Layer 2 - Ethereum for everyone* [online]. [cit. 2022-04-25]. Dostupné z: <https://ethereum.org/en/layer-2/>

[26] NAMBIAMPURATH, Rahul. *The 5 Best Ethereum Layer 2 Solutions* [online]. [cit. 2022-04-25].

[27] *What is Solana and why is it the hottest blockchain of the moment?* [online]. [cit. 2022-04-25]. Dostupné z: <https://forkast.news/what-is-solana-why-hottest-blockchain/>

- [28] *Decentralized finance (DeFi)* [online]. [cit. 2022-04-25]. Dostupné z: <https://ethereum.org/en/defi/>
- [29] *The best use cases for DApps* [online]. [cit. 2022-04-25]. Dostupné z: <https://cennz.net/knowledge-hub/cennznet-blockchain-101/the-best-use-cases-for-dapps/>
- [30] *Decentralized autonomous organizations (DAOs)* [online]. [cit. 2022-04-25]. Dostupné z: <https://ethereum.org/en/dao/>
- [31] Jacob Hale. *Top 10 most expensive NFTs ever sold* [online]. [cit. 2022-04-25]. Dostupné z: <https://www.dexerto.com/tech/top-10-most-expensive-nfts-ever-sold-1670505/>
- [32] *'Life-changing' or scam? Axie Infinity helps Philippines' poor earn* [online]. [cit. 2022-04-25]. Dostupné z: <https://www.france24.com/en/live-news/20220215-life-changing-or-scam-axie-infinity-helps-philippines-poor-earn>
- [33] *Best Blockchain Games* [online]. [cit. 2022-04-25]. Dostupné z: <https://sensoriumxr.com/articles/best-blockchain-games>
- [34] *Bitcoin Energy Consumption Index* [online]. [cit. 2022-04-25]. Dostupné z: <https://digiconomist.net/bitcoin-energy-consumption/>
- [35] *What is the Blockchain Trilemma?* [online]. [cit. 2022-04-25]. Dostupné z: <https://www.ledger.com/academy/what-is-the-blockchain-trilemma>
- [36] *Proof of Work a Proof of Stake (VŠE, CO VÍME)* [online]. 2020 [cit. 2022-05-13]. Dostupné z: <https://www.alza.cz/proof-of-work-a-proof-of-stake>
- [37] *CO JE TO CROWDFUNDING?* [online]. [cit. 2022-05-13]. Dostupné z: <http://www.financeproradost.cz/clanek/co-je-to-crowdfunding>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

PoW	Proof of Work
SHA	Secure Hash Algorithm
PoS	Proof of Stake
API	Application Programming Interface
IOT	Internet of Things
EVM	Ethereum Virtual Machine
ETH	Ether
EOA	Externally Owned Account
P2P	Peer to Peer
IPFS	Interplanetary File System
DeFi	Decentralized Finance
DEX	Decentralized Exchange
DAO	Decentralized Autonomous Organization
NFT	Non Fungible Token
CLI	Command Line Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets

**SEZNAM OBRÁZKŮ**

Obrázek 1 Typy počítačových sítí [19].....	10
Obrázek 2 Zjednodušená struktura blockchainu [20] .....	12
Obrázek 3 Schéma funkce oraclů v blockchainové síti [5] .....	15
Obrázek 4 Schéma architektury plně decentralizované aplikace [12] .....	20
Obrázek 5 Schéma architektury decentralizované aplikace s použitím centralizovaného serveru [12] .....	21
Obrázek 6 Schéma architektury aplikace s použitím centralizovaného serveru pro filtraci a odesílání transakcí [12] .....	22
Obrázek 7 Struktura projektu Truffle .....	37
Obrázek 8 Ukázka uživatelského prostředí Ganache.....	38
Obrázek 9 Ukázka kontraktu Crowdfunding.sol a funkce pro vytvoření projektu.....	40
Obrázek 10 Ukázka funkce pro podpoření projektu .....	41
Obrázek 11 Ukázka funkce pro vrácení finančních prostředků.....	42
Obrázek 12 Ukázka testovací metody z nástroje truffle .....	42
Obrázek 13 Ukázka z výpisu funkce truffle test.....	43
Obrázek 14 Ukázka z nástroje vue-cli .....	44
Obrázek 15 Stromová struktura aplikace.....	45
Obrázek 16 Ukázka definování vnitřních stavů aplikace pomocí knihovny Vuex.....	46
Obrázek 17 Ukázka funkcí mutation .....	46
Obrázek 18 Rozhraní rozšíření Metamask.....	47
Obrázek 19 Ukázka metody pro připojení účtu .....	48
Obrázek 20 Ukázka funkce pro získávání projektů z blockchainu.....	49
Obrázek 21 Funkce pro nahrání obrázků na úložiště IPFS.....	50
Obrázek 22 Definice jednotlivých cest v aplikaci .....	51
Obrázek 23 Ukázka části scriptu komponenty CreateProjectForm .....	53
Obrázek 24 Ukázka hlavní stránky projektu.....	54
Obrázek 25 Ukázka stránky s formulářem pro vytvoření projektu .....	55
Obrázek 26 Stránka pro zobrazení detailu projektu.....	56
Obrázek 27 Ukázka stránky informací o projektu .....	56

**SEZNAM TABULEK**

Tabulka 1. Ukázková tabulka ..... **Chyba! Záložka není definována.**



## SEZNAM PŘÍLOH

## **PŘÍLOHA P I: NÁZEV PŘÍLOHY**