Decentralizovaná aplikace na Ethereum Blockchainu

Petr Lovecký

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Lovecký**
Osobní číslo:	**A20310**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Decentralizovaná aplikace na Ethereum Blockchainu**
Téma práce anglicky:	**Decentralized Application on Ethereum Blockchain**

Zásady pro vypracování

1. Specifikujte požadavky na systém s ohledem na jeho zabezpečení.
2. Uveďte výhody použití blockchainové technologie pro vyvíjenou aplikaci.
3. Navrhněte samotný systém pro online správu prodeje vstupenek.
4. Navržený systém implementujte v testovacích prostředí a ověřte jeho funkčnost.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Debajani Mohanty, Ethereum for Architects and Developers: With Case Studies and Code Samples in Solidity, 2018, ISBN 9781484240755.
2. Sylvain Kerkour, Black Hat Rust, 2021.
3. Bashir Imran, Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks, 2017,ISBN 9781787129290.
4. Chris Dannen, Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners, 2017, ISBN 9781484225356.
5. Andreas M. Antonopoulos, Gavin Wood, Mastering Ethereum, 2018, ISBN: 9781491971949.

Vedoucí bakalářské práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

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 22. 05. 2023

Petr Lovecký, v. r.
podpis studenta

ABSTRAKT

Bakalářská práce se zabývá možností využití blokchainové technologie pro řešení konkrétního problému, v podobě decentralizované aplikace na Ethereum blockchainu s funkcí systému pro online správu prodeje vstupenek. V teoretické části jsou vysvětleny principy fungování technologií Blockhainu a Ethereum. Praktická část se již zabývá návrhem a vývojem konkrétní aplikace.

Klíčová slova: blockchain, Ethereum, smart kontrakt, decentralizovaná aplikace, NFT

ABSTRACT

The bachelor thesis deals with the possibility of using blockchain technology to solve a specific problem in the form of a decentralized application on the Ethereum blockchain with the function of an online ticketing management system. The theoretical part explains the principles of Blockhain and Ethereum technologies. The practical part deals with the development of the specific application.

Keywords: blockchain, Ethereum, smart contract, decentralized application, NFT

Tímto bych chtěl poděkovat vedoucímu, panu Ing. Davidu Malaníkovi, Ph.D., za odborné vedení a rady, které mi poskytoval v průběhu tvorby mé bakalářské práce.

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

OBSAH

ÚVOD

Technologie blockchain se v naší společnosti poprvé objevila v roce 2008 a to v podobě kryptoměny Bitcoin. Ze začátku Bitcoin neměl velkou pozornost, postupem času se ale dostal do povědomí lidí po celém světě.

Inspirován Bitcoinem přišel v roce 2013 programátor Vitalik Buterin s myšlenkou decentralizované výpočetní jednotky, se zaměřením na běh programů využívající blockchain pro ukládání stavů a dat programu. Tuto decentralizovanou platformu nazval Ethereum a kryptoměnu, která je používána jako platidlo za služby této platformy, pojmenoval Ether.

Od roku 2015, kdy byla síť Ethereum spuštěna, se myšlenka rozšířila mezi vývojáře a vytváření programů využívajících blockchain nabralo na popularitě. Tyto programy jsou nazývány smart kontrakty a díky kreativitě vývojářů se využití blockchainu rozšířilo od zprostředkování kryptoměny po běh decentralizovaných aplikací.

Dnes je technologie blockchain známá právě díky kryptoměnám jako je Bitcoin a Ether. Proto si většina lidí pod pojmem blockchain představí pouze kryptoměnu a neuvědomuje si potenciál této technologie.

Cílem práce je tedy ukázat možnost využití blockchainu i jiným způsobem, než je kryptoměna, a to v podobě decentralizované aplikace na Ethereum blockchainu, která má sloužit jako systém pro online správu prodeje vstupenek.

Teoretická část se zabývá představením technologiemi Blockchain a Ethereum, které tvoří základ navrhovaného systému.

Praktická část je poté zaměřena na vývoj a ověření funkčnosti této aplikace.

I. TEORETICKÁ ČÁST

1 BLOCKCHAIN

Aby bylo možné blockchainu plně porozumět, je nutné si nejprve rozebrat síťové systémy umožňující propojení a komunikaci více uzlů, na kterých je blockchain založen.

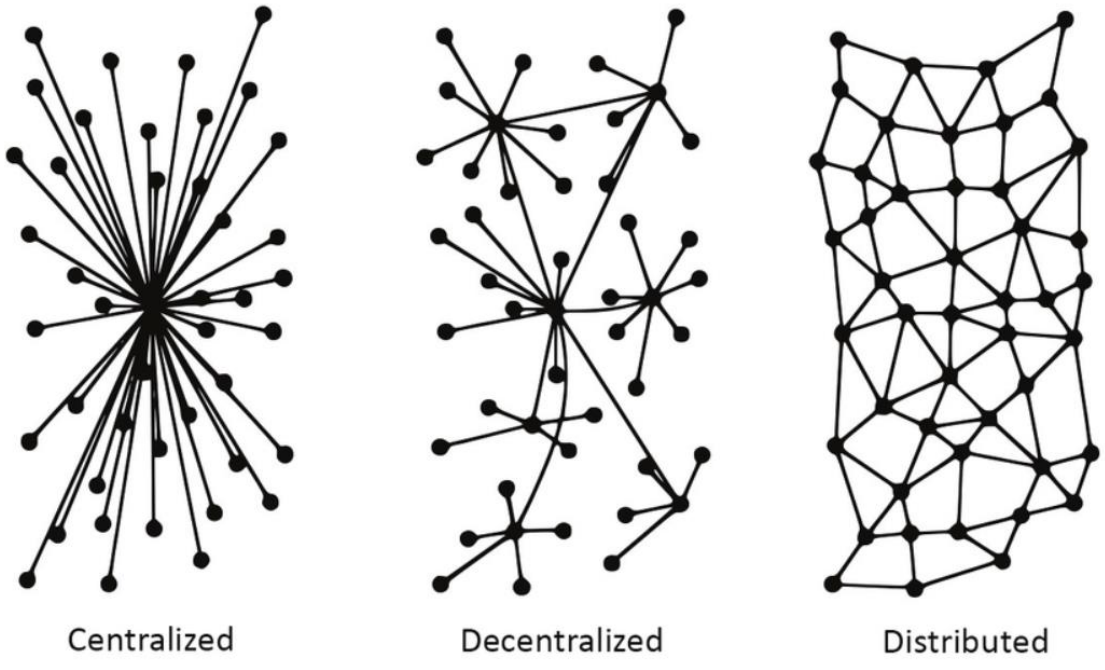
1.1 Síťové systémy

1.1.1 Centralizovaný systém

Koncoví uživatelé jsou připojeni k centrálnímu prvku (serveru), přes který probíhá veškerý provoz sítě. Pokud server selže, přestane systém fungovat. [2]

1.1.2 Decentralizovaný systém

Jedná se o systém složený ze dvou a více centrálních prvků (serverů), které můžou fungovat samostatně a zastanou veškeré funkce systému. To umožňuje rozdělit koncové uživatele mezi jednotlivé severy a v případě výpadku jednoho ze serverů zajistit pokračování chodu systému. [2]



Obrázek 1 Počítačové sítě [3]

1.1.3 Distribuovaný systém

Jedná se o systém složený ze dvou a více uzlů, které spolu komunikují a podílejí se na výpočetním výkonu systému takovým způsobem, že se navenek jeví jako jedna logická jednotka. [2]

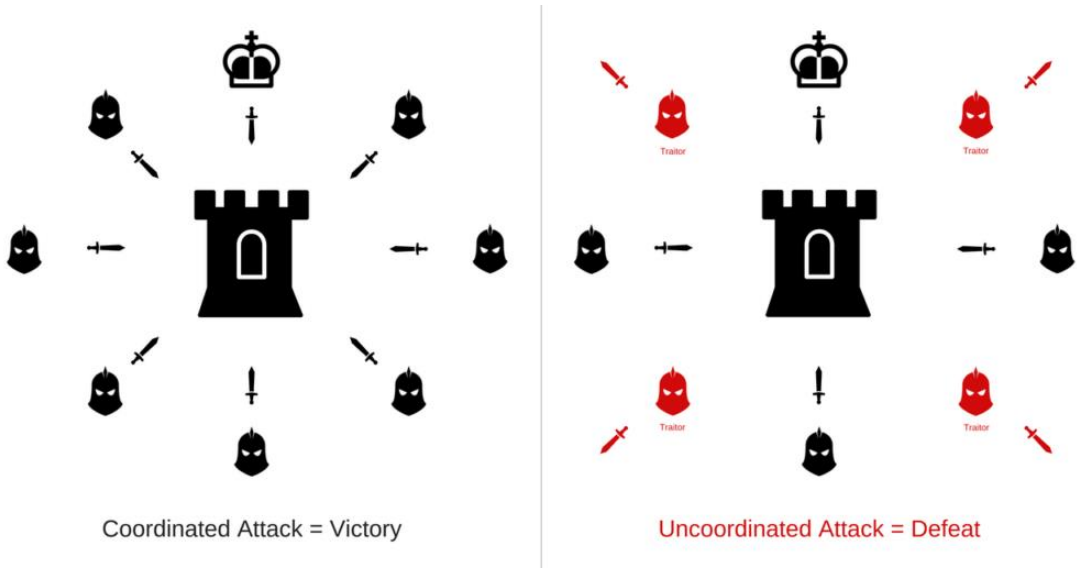
Tím že je systém závislí na jednotlivých uzlech, vzniká nebezpečí v podobě chybného, nebo škodlivého uzlu, který by narušil fungování systému. Takový uzel, který má neobvyklé chování je označován jako Byzantský též „Byzantine node“. Toto označení vzniklo z myšlenkového experimentu *Problém Byzantských generálů*, který v roce 1982 navrhl matematik a informatik Leslie Lamport. [1]

1.1.3.1 Problém Byzantských generálů

V tomto experimentu se dva a více generálů Byzantské armády snaží dobýt opevněné město. Generálové jsou rozmístěny takovým způsobem, že každý stojí na jiné straně města. Jediným způsobem komunikace je vyslat posla, který musí projet kolem opevněného města. Aby generálové měli možnost vyhrát, musí zaútočit současně.

Ze uvedené situace vznikají dva zásadní problémy:

1) Generál, který poslal zprávu, nemá jistotu, že zpráva dorazila
2) Jeden z generálů může být zrádce a stáhnout své vojsko



Obrázek 2 Problém Byzantských generálů [4]

Pokud generál A pošle zprávu generálovi B o čase útoku, generál A neví, jestli zpráva skutečně dorazila. Generál B sice může poslat zprávu s potvrzením, ale znovu nemá jistotu, že zpráva dorazila a generál A si ji přečetl.

Protože ani jeden z generálů nemá jistotu, že jejich zpráva dorazila, může se některý z nich rozhodnout ve smluvený čas nezaútočit, jelikož v případě samostatného útoku hrozí porážka a ztráta vojska útočícího generála.

Tento myšlenkový experiment lze přirovnat ke komunikaci uzlů v distribuovaném systému, kde uzly můžeme považovat za generály a posla za komunikační kanál.

Tyto problémy vyřešili v roce 1999 odborníci Miguel Castrov a Barbara Liskov představením *Practical Byzantine Fault Tolerance* (PBFT) algoritmu.

Později v roce 2009 byla prvně použita praktická implementace algoritmu s vynalezením Bitcoinu, v podobě *Proof of Work* algoritmu. [1]

1.2 Blockchain

Blockchain můžeme popsat jako decentralizovanou distribuovanou databázi, která je kryptograficky zabezpečená. Je možné do ní přidávat nové záznamy a není možné je následně měnit. Změna záznamu je možná pouze tehdy, pokud s ní souhlasí většina účastníků blockchainu. Jednotlivé záznamy jsou pak shlukovány do bloků, které se mezi sebou kryptograficky propojují, od této funkcionality je odvozen název „Blockchain“ neboli řetězec bloků. Jeden záznam v bloku se nazývá transakce. Síť blockchainu je tvořena uzly, kde každý uzel má kopii blockchainu, díky čemuž je zajištěn nepřetržitý chod sítě i v případě výpadku několika uzlů. Uzly se také starají o ověřování transakcí a bloků přidaných do blockchainu. [1] [5]

1.2.1 Struktura bloků v blockchainu

I když se struktura bloků může lišit dle typu blockchainu, má většina z nich stejné základní části, které jsou fundamentální pro jejich fungování. Tyto části jsou rozděleny na hlavičku bloku a tělo bloku. Hlavička bloku obsahuje ukazatel na předchozí blok, timestamp, nonce a merkle root hash. Tělo bloku je složeno z jednotlivých transakcí a jejich počtu. Stejnou strukturu mají všechny bloky v blockchainu kromě prvního bloku, který je označovaný jako

Genesis blok. Na rozdíl od ostatních nemá první blok v blockchainu ukazatel na předchozí blok a neobsahuje transakce. [1] [5] [6]

1.2.1.1 Popis částí bloku

Ukazatel na přechozí blok

Ukazatel na předchozí blok, je ve své podstatě hash vytvořený z veškerých dat předchozího bloku pomocí hashovací funkce. Skrze jednotlivé ukazatele je propojený celý blockchain a díky jeho vlastnostem zabezpečuje neměnnost dat v blockchainu. Pokud by došlo k jakékoliv změně dat v bloku, změnil by se i jeho hash, který by se nerovnal ukazateli v následujícím bloku.

Timestamp

Jedná se o časové razítko, které nese informaci o datumu a času vytvoření daného bloku. Časový formát se může lišit dle typu blockchainu, například Bitcoin a Ethereum ukládají datum a čas v podobě Unix formátu, jedná se tedy o počet sekund uplynulých od předem určeného data a času.

Nonce

Nonce je náhodné číslo, které slouží k úpravě výsledného hashe bloku, aby splňoval požadavky daného blockchainu.

Merkle root hash

Merkle root hash je root datové struktury merkle tree, též známé jako hash tree, která je v bloku použita k uložení jednotlivých transakcí. Merkle root je hash všech transakcí uložených v daném bloku. Díky hashovací funkci je zajištěna validita transakcí. [5] [6]

1.2.2 Konsensus

Jelikož blockchain nemá centrální prvek, musí se uzly v síti domluvit mezi sebou samy. Konsensus je dohoda uzlů na konečném stavu dat, které se mají přidat do blockchainu. Pro dosažení této dohody byli vytvořeny mechanismy konsensu. Tyto mechanismy jsou tvořeny souborem pravidel a kroků, které určují, jakým způsobem se mají uzly v síti dohodnout. Různé blockchainy mohou používat různé mechanismy, nejpoužívanějšími jsou Proof of Work a Proof of Stake. [1] [5]

1.2.2.1 Proof of Work

Proof of Work je první mechanismem který byl vytvořen a je založen na spotřebě elektrické energie, kterou uzly používají k vyřešení daného problému. Problémem je najít takové číslo nonce, které zajistí požadovanou obtížnost hashe celého bloku. Uzly mezi sebou soupeří, kdo problém vyřeší jako první. Vítěz potom získá právo na vytvoření nového bloku a odměnu za provedenou práci v kryptoměně daného blockchainu. Tomuto procesu se říká těžení (mining). Validita nového bloku je lehce ověřitelná ostatními uzly. Pokud by chtěl vítězný uzel vložit do bloku nevalidní data, musel by kontrolovat víc něž 51 % všech uzlů, které blok ověřují. Tento mechanismus používá síť Bitcoin a používala ji i síť Ethereum. [1] [5] [7]

1.2.2.2 Proof of Stake

U mechanismu Proof of Stake zastavují uzly určitou částku kryptoměny daného blockchainu jako zálohu, o kterou by přišli v případě vytvoření nevalidního bloku. Uzel neboli validátor je vybírán náhodně, ale šance na zvolení je ovlivněna množstvím zastavené kryptoměny. Po zvolení má Validátor za úkol vytvořit nový blok, který je ověřen ostatními uzly a za splnění úkolu je mu vyplacena odměna v podobě kryptoměny daného blockchainu. Tento způsob je použitý jako základ mechanismu, který v současnosti používá síť Ethereum. [1] [5] [8]

1.2.3 Interakce uživatelů s blockchainem

Pro interakci s blockchainem používají uživatelé primární a veřejný klíč. Veřejný klíč se používá jako adresa, která představuje identifikaci uživatele. Privátní klíč se používá pro podepsání transakce, aby šlo ověřit, kdo transakci poslal. Veřejný klíč je odvozen z Privátního klíče. Ve své podstatě se jedná o elektronický podpis. Privátní klíč slouží také pro přístup k účtu, proto by ho měl znát pouze jeho majitel.

Aby každý uživatel nemusel mít vlastní kopii blockchainu, přistupuje k němu pomocí uzlu, který tuto interakci umožňuje. [5]

1.2.4 Kryptoměny

Kryptoměna je decentralizovaná digitální měna, která je kryptograficky zabezpečena proti padělání. Kryptoměny jsou obecně postaveny na Distributed Ledger technologii zkráceně DLT. Blockchain je typem DLT, který kryptoměny používají nejčastěji. Existují však kryptoměny, které přímo blockchain nepoužívají.

Blockchain pro svůj chod nemusí nutně použít kryptoměny, pokud dokáže motivovat uzly jinačím způsobem k dosažení konsensu. Například v případě blockchainu, který by uchovával záznamy volebního systému, by byli všechny uzly motivovány spotřebovávat svoje zdroje výměnou za autentičnost voleb. Většina veřejných blockchainů však kryptoměny používá, protože se jedná o nejlepší způsob motivace pro uzly, a navíc slouží jako prostředek směny a uchovatel hodnoty. [9] [10] [11]

1.2.5 Kryptoměnové peněženky

Kryptoměnová peněženka je software nebo zařízení, které umožňuje uživateli interakci s blockchainem. Peněženka neslouží k uložení kryptoměn, jelikož ty jsou uloženy na blockchainu, ale slouží pro přístup uživatele k jeho kryptoměnám za pomocí privátního klíče. Ukládá privátní a veřejný klíč, pomocí kterých posílá a přijímá kryptoměny. Informuje uživatele o množství jeho kryptoměn, a proběhlých transakcí. [1] [12]

1.2.6 Větvení blockchainu

Větvení je událost, při které dojde k vytvoření nové větve blockchainu, která běží nezávisle na hlavní větvi, se kterou má společnou minulost, kořen. Tento proces je běžný u open source softwarů a umožňuje vývojářům použít část nebo celý program jako základ pro vytvoření nového softwaru.

Větvení blockchainu může být náhodné nebo záměrně. Náhodné větvení vzniká v případě vytvoření více bloků ve stejný čas, v důsledků čehož se síť nedokáže shodnout, který blok byl první, což vede k větvení blockchainu. Nakonec se zachová jen jedna větev vybraná většinou uzlů a ostatní se přestanou používat. Záměrné větvení vzniká za účelem úpravy pravidel nebo kódu softwaru blockchainu. Dle zpětné kompatibility větví je rozděleno na dva typy větvení, v angličtině označované jako Fork. Soft fork vzniká v rámci takové změny, která je zpětně kompatibilní s původním blockchainem. Hard fork vzniká v případě zpětně nekompatibilní změny. [13]

1.2.7 Smart kontrakt

Smart kontrakt je program uložený na blockchainu, který vykoná zadanou činnost, v případě splnění předem daných podmínek. Teorii smart kontraktu popsal již v 90. letech 20. století vědec Nick Szabo, nicméně k její implementaci došlo až v roce 2009 při vzniku Bitcoinu, kde kontrakty zajišťují převod kryptoměny mezi uživateli. Protože je vytváření nových smart kontraktů v síti Bitcoin složité a jejich použití limitované, zaznamenali kontrakty popularitu až s příchodem sítě Ethereum, která se specializuje na snadnou tvorbu nových kontraktů. [1] [14] [15]

1.2.8 Typy blockchainu

Dle udělení přístupu k blockchainu je rozdělen na 4 typy. Veřejné blockchainy může používat kdokoliv s připojením k internetu. Soukromé blockchainy jsou určené pouze povoleným uzlům, proto tento typ potřebuje centrální prvek, který se stará o udělování přístupu a přichází tak o decentralizaci. Dalším je hybridní typ, který má část blockchainu soukromou a část veřejnou. A posledním je konsorční blockchain, což je hybridní typ upravený na požadavky organizace. [16]

2 ETHEREUM

Ethereum je platforma založená na blockchainu se specializací na vytváření a běh nových smart kontraktů. Tato platforma se chová jako jeden velký decentralizovaný počítač, za jehož služby se platí kryptoměnou Ether. Ethereum se dá přirovnat k online tržišti služeb, které poskytují smart kontrakty, například finanční služby, hry a další aplikace. Výhodou těchto služeb je zajištění anonymity uživatele díky principům blockchainu.

Ethereum je první Turingovsky kompletní blockchain platforma. To znamená že je schopna vyřešit jakýkoliv výpočetní problém, pokud je poskytnuto dostatek paměti. Oproti předcházejícím blockchainovým sítím, které používají skriptovací jazyky, implementuje Ethereum plnohodnotný programovací jazyk Solidity. Představení této technologie přitáhlo zájem programátorů, jelikož značně ulehčuje vývoj smart kontraktů. Následoval rozvoj v oblasti decentralizovaných aplikacích a způsobů využití blockchainu, dle kreativity vývojáře. [17] [18]

2.1 Vývoj

Myšlenka Etherea vznikla v roce 2013, kdy programátor Vitalik Buterin vydal *whitepaper*, kde představil teorii této technologie. Po jeho přečtení nabídlo několik programátorů pomoc s vývojem a někteří byli vybráni a přijati do týmu. Síť Ethereum byla spuštěna 30. června 2015.

Od spuštění je technologie stále vyvíjena a prošla již spoustou změnami. Každá větší změna funkcionality má za následek hard fork, neboli vytvoření nové větve, která není zpětně kompatibilní s předchozí částí blockchainu. Ne vždy je však hard fork způsoben plánovanou aktualizací. Nejznámějším případem je *DAO hack* z roku 2016, kdy došlo k úniku 3,6 milionů ETH, který byl způsobený chybou v kontraktu. Krádež byla možná zvrátit za pomocí hard forku, pro který se rozhodla většina sítě. Uživatelé, kteří hlasovali proti, se rozhodli pokračovat v původním blockchainu a chybu zachovat. Vznikla tak oddělená platforma nazvaná *Ethereum Classic*.

Nejdůležitější změnou, kterou technologie Ethereum prošla, byl přechod z *Proof of Work* na *Proof of Stake* mechanizmus, známá pod označením *The Merge*. Tento přechod se uskutečnil v roce 2022 a zajistil snížení spotřeby elektrické energie Etherea o 99,95 %. Kromě snížení spotřeby, zvýšila aktualizace zabezpečení a škálovatelnost sítě. [17] [18] [19]

2.2 Ether

Kryptoměna v Ethereum síti je nazývána Ether, zkráceně ETH, Ξ a ♦. Tvoří základ pro fungování celé platformy, motivuje validátory, používá se jako prostředek směny a investiční aktivum. Od události *The Merge* je nedílnou součástí ověřovacího mechanismu. Jednotka Ether je dělitelná na několik částí. Nejmenší z nich je *wei* a slouží jako základní jednotka, v které je měna zapsaná na blockchainu. [18] [20] [23]

Tabulka 1. Jednotky Etherea [20]

Jednotka	**Wei**	**Exponent**
Wei	1	1
Kwei (babbage)	1 000	10^3
Mwei (lovelace)	1 000 000	10^6
Gwei (shannon)	1 000 000 000	10^9
Microether (szabbo)	1 000 000 000 000	10^{12}
Miliether (finney)	1 000 000 000 000 000	10^{15}
Ether	1 000 000 000 000 000 000	10^{18}

Na rozdíl od Bitcoinu, nemá Ether končené množství, ale malá částka je vytvořena za každý nový blok, jako odměna validátorům, tento proces je nazýván *minting*. Kromě vytváření nového Etheru dochází i k jeho ničení, označované jako *burning*. K odstranění částky kryptoměny dochází při každé transakci a její výše se odvíjí od vytíženosti sítě. Tato dynamika množství kryptoměny v oběhu dovoluje platformě reagovat na její potřeby. Kryptoměna byla v roční inflaci 4,61 % do události *The Merge*, kdy přešla do mírné deflace. [21] [22]

2.3 Účty

Ethereum má dva druhy účtů, uživatelský účet a účet kontraktu. Oba účty mohou přijímat a posílat Ether nebo tokeny a oba můžou interagovat s dalšími kontrakty. Jsou však mezi nimi určité rozdíly. Uživatelský účet lze vytvořit zdarma, může založit transakci, a je tvořen veřejným a soukromým klíčem. Vytvoření účtu kontraktu již něco stojí, protože bude uložen na úložiště platformy. Kontrakt může posílat transakci pouze v rámci reakce na příchozí transakci. Účet kontraktu nemá soukromý klíč, je totiž kontrolován logikou kódu kontraktu. [26]

2.4 Ethereum Virtual Machine

Ethereum Virtual Machine zkráceně EVM, je virtuální stroj, na jehož fungování se podílejí jednotlivé uzly v síti. Ve své podstatě se jedná o distribuovaný stavový automat, který mění a ukládá stavy jednotlivý kontraktů na blockchain.

EVM se specializuje na spouštění kódu smart kontraktů. I když jsou kontrakty psány v programovacím jazyku, EVM umí překládat pouze byte kód, proto se kontrakt nejdříve musí zkompilovat, aby mohl být přidán na blockchain. Když dojde k zavolání funkce kontraktu, EVM spustí uložený byte kód. [27]

2.5 Transakce

Transakce je sada instrukcí, nesoucí informaci o změně stavu Ethereum sítě, které jsou kryptograficky podepsané odesílajícím účtem. Nejjednodušší transakce je převod kryptoměny Ether z jednoho účtu na druhý.

Provedení transakce vyžaduje poplatek validátorům za přidání transakce do nového bloku a její ověření. Tento poplatek je nazýván Gas. [28]

2.5.1 Gas

Jednotka, ve které se udává množství výpočetního výkonu potřebného pro vykonání instrukcí uvedené v transakci. Jedná se o poplatek za provedení transakce. Gas je placen v kryptoměně Ether a jelikož se jedná o menší částky, tak se uvádí v jednotce *gwei*. Jednotka gas se skládá ze dvou položek Base fee a Priority fee.

Block size

> Udává v hodnotě gas množství instrukcí, které může být uloženo na jeden blok. Velikost bloku se dynamicky mění v závislosti na požadavcích sítě. Základní velikost bloku je 15 milionů gasu, v případě potřeby zvýší svoji kapacitu na 30 milionů gasu.

Base fee

> Základní poplatek, jehož částka se mění v závislosti na velikosti daného bloku. Pokud velikost bloku dosáhne 30 milionů gasu, dojde k navýšení základního poplatku o 12,5 %. Tento proces se opakuje, dokud nedojde ke snížení cílené velikosti bloku na 15 milionů gasu. Potom co je blok vytěžen, je poplatek zničen.

Priority fee

Poplatek určený validátorům. Výše poplatku určuje prioritu transakce čili pořadí, ve kterém bude zpracovaná.

Gas used

Počet gas jednotek potřebných pro vykonání dané transakce.

Max fee

Umožňuje uživateli zadat maximální částku, kterou je ochoten zaplatit za jeden gas. V případě levnější transakce je přebytek poplatku vrácen zpět uživateli.

Výsledný poplatek za transakci se poté počítá následovně:

gas used * (base fee + priority fee)

Díky gasu je zajištěno zabezpečení sítě proti spamu, nebo zacyklení kontraktu, jelikož by takové akce byly příliš drahé. [24]

2.5.2 Struktura transakce

Přijatá transakce obsahuje tyto informace:

From – adresa odesílatele

Recipient – adresa příjemce

Signature – podpis transakce, identifikace odesílatele

Nonce – číslo označující transakci odesílatele, toto číslo se navyšuje za každou transakci, která byla z účtu odeslána

Value – hodnota ETH poslána v transakci, uváděna v jednotce wei

Data – volitelná informace, obsahuje libovolná data

GasLimit – maximální množství gas jednotek, které může být spotřebováno pro zpracování odeslané transakce

MaxPriorityFeePerGas – maximální částka, kterou je odesílatel ochoten zaplatit validátorům

MaxFeePerGas – maximální částka, kterou je uživatel ochoten zaplatit za jednotku gas [28]

2.6 Tokenizace

Ether není jediným aktivem Etherum platformy. Kdokoliv může vytvořit nové aktivum, obecně označované jako tokeny, se kterým může v síti obchodovat. Pro jejich implementaci bylo vytvořeno několik standardů. [23] [25]

2.6.1 ERC-20

Rozhraní pro zaměnitelné tokeny jako jsou virtuální měny, hlasovací tokeny, staking tokeny.

2.6.2 ERC-721

Rozhraní pro nezaměnitelné tokeny, též známé pod názvem *NFT*, token může představovat dokument, umění či píseň.

2.6.3 ERC-777

Rozhraní pro rozšíření funkcionality zaměnitelného tokenu.

2.6.4 ERC-1155

Rozhraní pro zaměnitelné i nezaměnitelné tokeny umožňující sdružování transakcí a tím šetří náklady.

2.6.5 ERC-4626

Rozhraní pro sjednocení technických parametrů speciálních tokenů, které se používají pro uzamknutí částky kryptoměny na určitou dobu za slíbený výnos. [25]

II. PRAKTICKÁ ČÁST

3 NÁVRH DECENTRALIZOVANÉ APLIKACE

3.1 Řešený problém

V dnešní době máme možnost fyzické vstupenky na události či lístky, nahradit virtuálními, díky online vstupenkovým systémům. Takový systém pořadatelé událostí používají skrze prostředníka, který systém navrhl a nabízí jeho služby za poplatek, nebo si pořadatelé nechávají systém vytvořit podle jejich potřeb. Druhá možnost se ale pořadatelům vyplatí pouze v určitých případech, kdy dochází k vytváření události či poskytování služby opakovaně.

Problémem systémů třetích stran je, že jejich použití závisí na důvěře mezi systémem a pořadateli. Pořadatelé musí důvěřovat poskytovatelům systému, že zabezpečí systém proti padělání vstupenek a zajistí tak jejich pravost. Jelikož má poskytovatel systému absolutní moc nad tvorbou vstupenek, má možnost vytvořit falešné vstupenky nad rámec zadaným pořadatelem, aniž by to pořadatel zpozoroval, protože veškeré akce probíhají uvnitř centrálního systému.

3.2 Navrhované řešení

Jako řešení problému se nabízí možnost využití blockchain technologie v podobě smart kontraktu, který by vykonával požadované funkce systému. Chování kontraktu je předem nadefinováno jeho zdrojovým kódem, který nelze po nahrání na blockchain měnit. Možnost vytvoření falešných vstupenek je tak znemožněna, jelikož se systém nemůže zachovat jinak, než mu bylo předem naprogramováno.

3.3 Analýza požadavků na systém

Na začátku vývoje je potřeba provést analýzu požadavků na systém, aby byla zjištěna představa, jaké funkce by měl systém zajišťovat a v jakém prostředí bude existovat.

3.3.1 Funkční požadavky

FP1 – Systém musí umět nabízet seznam aktuálních událostí.

FP2 – Systém musí umět zobrazit detail události s možností koupení vstupenky.

FP3 – Vstupenky se budou prodávat za kryptoměnu Ethereum.

FP4 – Uživateli musí být umožněno vytvoření nové události.

FP5 – Uživateli musí být umožněno koupit vstupenku na událost.

FP6 – Systém musí umět zobrazit vstupenky patřící danému uživateli.

FP8 – Systém musí umožňovat uživateli vrácení vstupenky.

FP9 – Uživateli musí být umožněno připojit se pomocí MetaMask peněženky.

FP10 – Uživatel může používat systém až po připojení MetaMask peněženky.

3.3.2 Nefunkční požadavky

NP1 – Logiku systému bude zajišťovat smart kontrakt.

NP2 – Smart kontrakt bude naprogramovaný v jazyku Solidity.

NP3 – Smart kontrakt bude umístěn na Ethereum blockchainu.

NP4 – Pro přístup ke kontraktu bude systém používat webovou aplikaci.

NP5 – Webová aplikace bude vytvořená ve frameworku Vue.

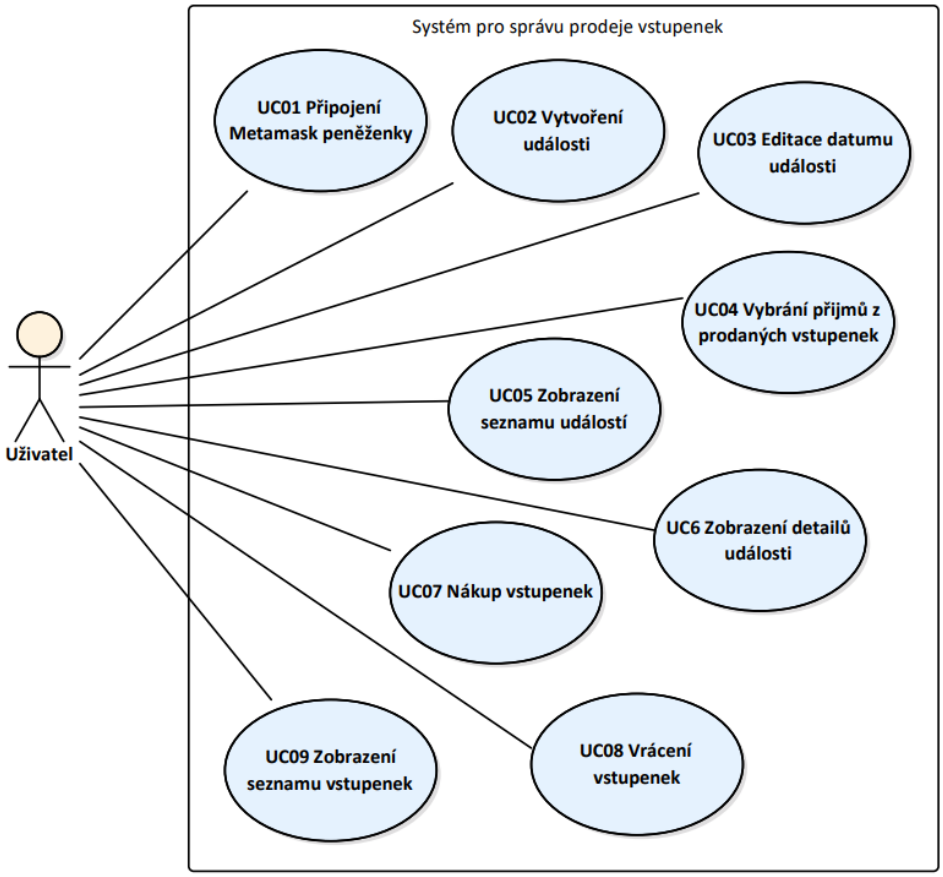
NP6 – Webová aplikace bude používat protokol HTTPS.

NP7 – Vstupenky budou vytvářeny jako NFT.

NP8 – Metadata a QR kódy vstupenek budou uloženy na decentralizovaném úložišti.

3.4 Diagram případů užití

V diagramu případů užití je ukázáno, jakým způsobem aktéři budou systém používat. V tomto případě má systém pouze jednoho aktéra, označovaného obecně jako Uživatel.



Obrázek 3 Diagram případů užití

3.4.1 Případy užití

UC01 Připojení Metamask peněženky

UC02 Vytvoření události

UC03 Editace datumu události

UC04 Vybrání příjmů z prodaných vstupenek

UC05 Zobrazení seznamu událostí

UC06 Zobrazení detailů události

UC07 Nákup vstupenek

UC08 Vrácení vstupenek

UC09 Zobrazení seznamu vstupenek

3.4.2 Scénáře

Název: Připojení Metamask peněženky		
ID: UC01		
Charakteristika:		
Systém připojí uživatele skrze Metamask peněženku		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro připojení peněženky
2	Systém	Připojí uživatele do systému a zobrazí nabídku jeho možností

Tabulka 2 UC01 Připojení Metamask peněženky

Název: Vytvoření události		
ID: UC02		
Charakteristika:		
Systém vytvoří novou událost		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro vytvoření nové události
2	Systém	Zobrazí formulář pro vytvoření nové události
3	Uživatel	Vyplní formulář a potvrdí vytvoření nové události
4	Systém	Validuje zadaná data
5	Systém	Vytvoří novou událost a upozorní uživatele o úspěšném vytvoření
Alternativní scénáře:		
UC02-4A		

Tabulka 3 UC02 Vytvoření události

Název: Alternativní scénář k UC02 – Nevalidní data		
ID: UC02-4A		
Charakteristika:		
Systém upozorní uživatele o neúspěšném vytvoření události		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Systém	Upozorní uživatele o neúspěšném vytvoření události
2	Uživatel	Vyplní formulář znovu

Tabulka 4 UC02-4A Alternativní scénář k UC02 – Nevalidní data

Název: Editace datumu události		
ID: UC03		
Charakteristika:		
Systém změní datum konání události		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro editaci datumu konání události
2	Systém	Zobrazí formulář pro editaci datumu
3	Uživatel	Vyplní formulář a potvrdí editaci
4	Systém	Validuje zadané datum
5	Systém	Změní datum a upozorní uživatele o úspěšné editaci
Alternativní scénáře:		
UC03-4A		

Tabulka 5 UC03 Editace datumu události

Název: Alternativní scénář k UC03 – Nevalidní datum		
ID: UC03-4A		
Charakteristika:		
Systém upozorní uživatele o neúspěšné editace datumu události		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Systém	Upozorní uživatele o neúspěšné editaci datumu
2	Uživatel	Vyplní formulář znovu

Tabulka 6 UC03-4A Alternativní scénář k UC03 – Nevalidní datum

Název: Vybrání příjmů z prodaných vstupenek		
ID: UC04		
Charakteristika:		
Systém pošle příjmy z prodaných vstupenek uživateli		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro vybrání příjmů z prodaných vstupenek
2	Systém	Pošle příjmy v kryptoměně na peněženku uživatele
3	Systém	Upozorní uživatele o úspěšném provedení akce

Tabulka 7 UC04 Vybrání příjmů z prodaných vstupenek

Název: Zobrazení seznamu událostí		
ID: UC05		
Charakteristika:		
Systém zobrazí seznam událostí		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost zobrazení seznamu událostí
2	Systém	Zobrazí seznam událostí

Tabulka 8 UC05 Zobrazení seznamu událostí

Název: Zobrazení detailů událostí		
ID: UC06		
Charakteristika:		
Systém zobrazí detaily události		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost zobrazení detailů události
2	Systém	Zobrazí detaily události

Tabulka 9 UC06 Zobrazení detailů událostí

Název: Nákup vstupenek		
ID: UC07		
Charakteristika:		
Systém prodá vstupenku uživateli		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro nákup vstupenky
2	Systém	Prodá vstupenku uživateli a upozorní na úspěšný nákup

Tabulka 10 UC07 Nákup vstupenek

Název: Vrácení vstupenek		
ID: UC08		
Charakteristika:		
Uživatel vrátí vstupenku systému		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro vrácení vstupenky
2	Systém	Koupí vstupenku od uživatele zpět

Tabulka 11 UC08 Vrácení vstupenek

Název: Zobrazení seznamu vstupenek		
ID: UC09		
Charakteristika:		
Systém zobrazí seznam vstupenek připojeného uživatele		
Primární aktér: Uživatel		
Hlavní scénář:		
Krok	Aktér	Popis
1	Uživatel	Vybere možnost pro zobrazení seznamu vstupenek
2	Systém	Zobrazí seznam vstupenek

Tabulka 12 UC09 Zobrazení seznamu vstupenek

4 VÝHODY POUŽITÍ BLOCKCHAIN TECHNOLOGIE PRO VYVÍJENOU APLIKACI

Použití blockchainu jako backend prostředí pro systém správy prodeje vstupenek nabízí určité výhody, ale i limitace, které je třeba si uvědomovat.

Decentralizace

Systém není závislý na centrálním prvku, díky čemuž je odolný proti hrozbám týkajících se centrálních systémů. To má za následek šetření zdrojů, které by byly potřeba k zabezpečení centrálního prvku proti takovým hrozbám.

Transparentnost

Akce probíhající na blockchainu jsou veřejné, což znemožňuje systému vytvořit více vstupenek, než je zadáno pořadatelem události. Transparentnost tak zajišťuje důvěru mezi pořadatelem a poskytovatelem systému.

Neměnitelnost

Data zapsaná na blockchain je natolik obtížné měnit, že lze považovat blockchain jako neměnný. Tato vlastnost přináší i nevýhodu v podobě nemožnosti editace programu, jakmile je nahraný na blockchain. Proto je důležité při vývoji myslet dostatečně dopředu a program před nasazením důsledně otestovat, aby byly objeveny veškeré chyby systému.

Dostupnost

Blockchain je distribuovaný mezi tisícovkami uzlů. Data jsou kopírována a aktualizována na každém uzlu sítě, což zajišťuje vysokou dostupnost systému.

Kryptografické zabezpečení

Veškeré transakce jsou na blockchainu kryptograficky zabezpečeny. Způsob, jakým blockchain zabezpečuje jednotlivé bloky zajišťuje neměnnost blockchainu.

Šetření nákladů

Poskytovatelům systému odpadají náklady spojené se správou centrálního serveru, což se může projevit na snížení poplatků, které jsou účtovány pořadatelům.

4.1 Výhody použití NFT

Důležitou blockchainovou technologií, na které je systém založen, je NFT. Nezaměnitelné tokeny zde představují jednotlivé vstupenky na událost, z toho vyplívá že každá vstupenka je jedinečná. Vlastnictví vstupenky je zapsáno na blockchainu a je jednoduché ověřit, komu patří. Výhodou je, že si vstupenky může uživatel přidat do své kryptoměnové peněženky a tím je řešena přenositelnost vstupenek. Po skončení události vstupenky nezanikají, ale mohou se stát sběratelskými předměty a mohou si tak uchovat část svojí původní hodnoty.

5 VÝVOJ DECENTRALIZOVANÉ APLIKACE

Při vývoji aplikace se postupovalo následovně. Nejdříve došlo k implementaci backendu takovým způsobem, aby splňoval navrhnuté požadavky na systém. Pro ověření splnění těchto požadavků byly napsány unit testy, kterými byl kontrakt otestován. Poté, co byla ověřena funkčnost logiky systému, byl vyvinut frontend. Nejprve bylo vytvořeno uživatelské prostředí webové aplikace, po kterém bylo vyřešeno propojení s backendem systému. Nakonec došlo k otestování celkové aplikace z pohledu běžného uživatele a opravě nalezených chyb.

5.1 Backend

Backend systému je napsaný v programovací jazyce *Solidity* v podobě smart kontraktu. Implementace kontraktu probíhala ve vývojovém prostředí *Remix Online IDE* z důvodu přehledné a jednoduché interakce s kontraktem. Následně byl kód přesunut do vývojového prostředí *Truffle*, kde byly napsány unit testy v jazyku *Javascript* za pomocí testovacího frameworku *Mocha*. Po otestování byl kontrakt nahrán na soukromý lokální blockchain, který byl vytvořený aplikací *Ganache*. Kontrakt musí být nahrán na blockchainu, pokud s ním chceme komunikovat z frontendu.

5.1.1 Rozbor kódu

Kód používá licenci *MIT* od společnosti SPDX a je psán v *Solidity* verzi 0.8.9.

5.1.1.1 Knihovny

Při vývoji byly použity knihovny od společnosti *OpenZeppelin*, která se zabývá automatizací a bezpečností smart kontraktů. Konkrétně byly použity nástroje související se standardem ERC 721 pro usnadnění a urychlení vývoje, ale také pro zvýšení bezpečnosti, jelikož všechny nabízené knihovny prošli bezpečnostním auditem.

Seznam použitých knihoven

- ERC721
- IERC721Receiver
- ERC721URIStorage
- Ownable
- Counters

5.1.1.2 *Eventy*

Event je nástroj programovacího jazyka *Solidity*, jak uložit vybraná data do logu blockchainu. Kontrakt má vytvořené dva eventy, jeden dává informace o vytvoření události a druhý o výběru příjmů pořadatelem. Kromě těchto eventů, používá kontrakt ještě převzatý event z rozhraní IERC721, který ukládá informace o převodu NFT, tudíž vstupenky.

```
// Solidity Events
event CreateEvent(uint id, string name, address owner, uint maxTicketCount,
                  uint price, uint date, uint returnDate);
event Withdraw(uint eventId, address owner, uint withdrawedAmount);
```

Obrázek 4 Solidity kód – eventy kontraktu

5.1.1.3 *Modifikátory funkcí*

Modifikátor funkce je nástroj, který umožňuje nadefinovat jaký kód se má provést před, nebo po provedení dané funkce. Kód funkce je v modifikátoru označovaný znakem _;.

```
// Modifier of reentrancy security
modifier noReentrant() {
    require(!locked, "No re-entrancy");
    locked = true;
    _;
    locked = false;
}
```

Obrázek 5 Solidity kód – příklad modifikátoru

V kontraktu jsou používané dva modifikátory.

noReentrant

Zabezpečuje funkci proti útoku známém jako *Reentrancy attack*. Tento útok jde použít na metody kontraktu, které posílají kryptoměnu, například metoda pro výběr kryptoměny z kontraktu. *Reentrancy attack* umožňuje útočníkovy takovou metodu volat opakovaně a tím vybrat z kontraktu více kryptoměny, než mu bylo povoleno. Princip fungování útoku spočívá v tom, že je funkce volaná rekurzivně před tím, než se provede zápis o výběru kryptoměny

a tím dochází k opakovanému posílání kryptoměny útočníkovi. Modifikátor *noReentrant* brání funkci takovým způsobem, že ji znemožňuje zavolat před jejím dokončením. K tomu používá proměnou *locked,* která funkci takzvaně uzamkne.

onlyOwner

Modifikátor dovoluje vykonat funkci pouze v případě, že je volaná majitelem kontraktu.

5.1.1.4 Struktury

V kontraktu je vytvořena struktura představující událost, pojmenovaná *Event*.

```
// Struct of event
struct Event {
    uint Id;
    string Name;
    address Owner;
    uint MaxTicketCount;
    uint Price;
    uint Date;
    uint StopReturnDate;
    uint[] TicketsId;
    uint[] ReturnedTicketsId;
    bool Withdrawed;
}
```

Obrázek 6 Solidity kód – struktura Event

Každá událost má svůj identifikátor, název, adresu uživatele, který událost vytvořil, počet vstupenek, cenu za jednu vstupenku, datum konání, datum, do kdy je možné vracet vstupenky, pole identifikátorů vstupenek, pole identifikátorů vrácených vstupenek a bool hodnotu, která ukazuje, zda pořadatel události vybral příjmy z prodaných vstupenek.

5.1.1.5 Proměnné kontraktu

_tokenIdCounter: nese informaci o počtu NFT (vstupenek) vytvořené kontraktem

_Events: umožňuje přistupovat k jednotlivým událostem za pomocí jejich identifikátorů, slouží k uložení událostí

_TicketToEvent: nese informaci přiřazení vstupenky k události

_EventCount: eviduje počet událostí vytvořených v kontraktu

_UserToEvents: umožňuje přistupovat k identifikátorům událostem skrze adresu jejich pořadatele, seskupuje události uživatele

_UserToTickets: umožňuje přistupovat k identifikátorům vstupenek skrze adresu jejich majitele, seskupuje vstupenky uživatele

locked: proměnná používaná v *noReentrant* modifikátoru

withdrawBalance: nese informaci o množství kryptoměny ETH udávané ve WEI, kolik si může majitel kontraktu vybrat za poskytnuté služby

```
// Counter of tokens
using Counters for Counters.Counter;
Counters.Counter private _tokenIdCounter;

// EventId => Event
mapping(uint => Event) private _Events;
// TicketId => EventId
mapping(uint => uint) private _TicketToEvent;

uint private _EventCount = 0;

// User => array of EventsId
mapping(address => uint[]) private _UserToEvents;

// User => array of TicketsId
mapping(address => uint[]) private _UserToTickets;

// Variable for reentrancy modifier
bool private locked;

// Variable for withdraw contract balance
uint private withdrawBalance = 0;
```

Obrázek 7 Solidity kód – proměnné kontraktu

5.1.1.6 Funkce kontraktu

createEvent

Vytvoří novou událost a uloží ji na blockchain.

```
function createEvent(string calldata name, uint maxTicketCount,
                     uint price, uint date, uint stopReturnDate)
                     external
{
    require(date > block.timestamp, "Date from the past");
    require(stopReturnDate > block.timestamp, "Date from the past");
    require(maxTicketCount > 0, "Count of tickets must be at least 1");

    _EventCount ++;

    // Event creation
    Event memory e = Event(_EventCount, name, msg.sender, maxTicketCount, price,
                           date, stopReturnDate, new uint[](0), new uint[](0), false);
    _Events[_EventCount] = e;

    // Add Event to user
    _UserToEvents[msg.sender].push(_EventCount);

    emit CreateEvent(_EventCount, name, msg.sender, maxTicketCount,
                     price, date, stopReturnDate);
}
```

Obrázek 8 Solidity kód – funkce createEvent

Funkce má ošetřené, aby nebylo možné vytvořit událost s datumem, který už nastal a je zamezené vytvoření události bez vstupenek. Na začátku je provedeno navýšení proměnné *_EventCount*, jelikož je používaná jako identifikátor události. Následně dojde k vytvoření nové události, uložení do seznamu událostí a přiřazení události k adrese pořadatele. Na konci funkce se zavolá event a zapíše informace o vytvoření události do logu blockchainu.

buyTicket

Po tom, co uživatel pošle dostatečné množství ETH, vytvoří funkce novou vstupenku, přiřadí ji dané události a přidá vstupenku do vlastnictví uživatele.

```
function buyTicket(uint eventId, string memory ticketUri)
external payable
{
    require(_Events[eventId].Date > block.timestamp, "Ticket sale is over");
    require((_Events[eventId].TicketsId.length < _Events[eventId].MaxTicketCount),
     "Tickets are sold");
    require(msg.value >= _Events[eventId].Price, "sent ETH < TicketPrice");

    // Ticket payment
    (bool sent, bytes memory data) = address(this).call{value: _Events[eventId].Price}("");
    require(sent, "failed to sent ETH");

    // Refund of overpayment
    if(msg.value > _Events[eventId].Price){
        uint overpayment = msg.value - _Events[eventId].Price;
        (bool sent2, bytes memory data2) = msg.sender.call{value: overpayment}("");
        require(sent2, "failed to sent overpayment");
    }

    uint256 ticketId = _tokenIdCounter.current();
    _tokenIdCounter.increment();

    // Ticket mint
    _safeMint(msg.sender, ticketId);
    _setTokenURI(ticketId, ticketUri);

    // Add ticket to event
    _Events[eventId].TicketsId.push(ticketId);
    _TicketToEvent[ticketId] = eventId;
    // Add ticket to user
    _UserToTickets[msg.sender].push(ticketId);
}
```

Obrázek 9 Solidity kód – funkce buyTicket

Na začátku funkce kontroluje, zda ještě neskončila doba prodeje, zda vstupenky nejsou vyprodané a jestli uživatel poslal dostatečné množství ETH. Následně kontrakt přijme poslané ETH a v případě přeplatku vrátí nadbytek ETH zpět uživateli. Jako další se zjistí počet NFT v kontraktu a použije se jako identifikátor nově vytvořené vstupenky. Po vytvoření se vstupence přiřadí URI, kde jsou uložená její metadata. Na konec se přiřadí vstupenka události a přidá se do seznamu vstupenek připojeného uživatele.

buyReturnedTicket

V případě, že existují vrácené vstupenky, převede vlastnictví vstupenky od kontraktu k uživateli, který vstupenku koupil.

```
function buyReturnedTicket(uint eventId) external payable {
    require(_Events[eventId].Date > block.timestamp, "Ticket sale is over");
    require(_Events[eventId].ReturnedTicketsId.length > 0, "No tickets left");
    require(msg.value >= _Events[eventId].Price, "sent ETH < TicketPrice");

    // Ticket payment
    (bool sent, bytes memory data) = address(this).call{value: _Events[eventId].Price}("");
    require(sent, "failed to sent ETH");

    // Refund of overpayment
    if(msg.value > _Events[eventId].Price){
        uint overpayment = msg.value - _Events[eventId].Price;
        (bool sent2, bytes memory data2) = msg.sender.call{value: overpayment}("");
        require(sent2, "failed to sent overpayment");
    }

    // Get last of returned tickets
    uint lastIndex = _Events[eventId].ReturnedTicketsId.length - 1;
    uint ticketId = _Events[eventId].ReturnedTicketsId[lastIndex];

    // Ticket transfer
    ERC721(address(this)).safeTransferFrom(address(this), msg.sender, ticketId);

    // Removing ticket from returned tickets array
    _Events[eventId].ReturnedTicketsId.pop();

    // Add ticket to user
    _UserToTickets[msg.sender].push(ticketId);
}
```

Obrázek 10 Solidity kód – funkce buyReturnedTicket

Funkce se provede pouze pokud existují vrácené vstupenky, dále je ošetřeno, aby se funkce neprovedla po skončení prodejní doby a pokud uživatel neposlal dostatečné množství ETH. Pokud jsou splněny uvedené podmínky, přijme kontrakt platbu a v případě přeplatku, vrátí nadbytek ETH uživateli. Následně vybere poslední ID z pole vrácených vstupenek a přesune vlastnictví z kontraktu na kupujícího uživatele. Poté odstraní ID vstupenky z pole vrácených vstupenek a přiřadí vstupenku uživateli.

retrunTicket

Funkce umožňuje uživateli vrátit vstupenku. Vlastnictví vstupenky se převede z uživatele na kontrakt a kontrakt vrátí uživateli takovou částku ETH, kterou zaplatil při koupení vstupenky.

```
function returnTicket(uint ticketId) external noReentrant
{
    require(msg.sender == ownerOf(ticketId), "You are not owner of Ticket");
    // Get ticket by Event
    uint eventId = _TicketToEvent[ticketId];
    require(_Events[eventId].StopReturnDate > block.timestamp);

    // Payment of return
    (bool sent, bytes memory data) = msg.sender.call{value: _Events[eventId].Price}("");
    require(sent, "failed to send ETH");

    // Transfer of ticket from user to contract
    safeTransferFrom(msg.sender, address(this), ticketId);

    // Add ticket to returned ticket array
    _Events[eventId].ReturnedTicketsId.push(ticketId);

    // Remove ticket from user tickets
    for(uint i = _UserToTickets[msg.sender].length-1; i >= 0; i--){
        if(_UserToTickets[msg.sender][i] == ticketId){
            _UserToTickets[msg.sender][i] = _UserToTickets[msg.sender][_UserToTickets[msg.sender].length-1];
            _UserToTickets[msg.sender].pop();
            break;
        }
    }
}
```

Obrázek 11 Solidity kód – funkce returnTicket

Funkce nejdříve ověří, jestli vstupenku opravdu vrací její majitel, poté zjistí, které události vstupenka patří a ověří, zda již neuplynula doba umožňující vrácení vstupenky. Po ověření těchto podmínek pošle kontrakt uživateli částku určenou cenou vstupenky a převede vlastnictví vstupenky z uživatele na kontrakt. Následně přidá vstupenku do pole vrácených vstupenek a odstraní vstupenku ze seznamu vstupenek daného uživatele. Funkce používá modifikátor *noReentrant*, protože by na ni bylo možné použít útok *Reentrancy attack*.

withdraw

Pošle pořadateli události příjmy za prodané vstupenky snížené o poplatek kontraktu za poskytnuté služby.

```
function withdraw(uint eventId) external noReentrant{
    require(_Events[eventId].Date < block.timestamp, "Ticket sale is not over");
    require(msg.sender == _Events[eventId].Owner, "You are not owner of Event");
    require(_Events[eventId].Withdrawed == false, "Alredy withdrawed");

    (uint finalAmount, uint finalFee) = getFinalAmountOfWithdraw(eventId);

    // Withdraw payment to owner of Event
    (bool sent, bytes memory data) = msg.sender.call{value: finalAmount}("");
    require(sent, "Failed to send ETH");

    _Events[eventId].Withdrawed = true;

    withdrawBalance = withdrawBalance + finalFee;

    emit Withdraw(eventId, msg.sender, finalAmount);

}
```

Obrázek 12 Solidity kód – funkce withdraw

Funkci může použít pouze pořadatel události a může být provedena až po skončení prodejní doby. Dále může být funkce vykonána jen jednou pro každou událost, aby nemohl pořadatel vybrat prostředky z kontraktu vícekrát. Toho je docíleno změnou hodnoty proměnné *Withdrawed*. Pomocí funkce *getFinalAmountOfWithdraw* se zjistí částka, kterou si uživatel může vybrat a poplatek který si ponechá kontrakt. Poté kontrakt pošle uživateli příjmy z prodaných vstupenek a změní proměnou události *Withdrawed* na hodnotu *true*, aby nebylo možné funkci volat opakovaně. Dále je proměnná *withdrawBalance* navýšená o poplatek, aby se evidovalo množství ETH, které si majitel kontaktu může vybrat. Nakonec se zavolá event, který zapíše informace o provedené akci do logu blockchainu.

withdrawContract

Pošle vlastníkovi kontraktu příjmy za poskytnuté služby kontraktu.

```
function withdrawContract() external onlyOwner{
    (bool sent, bytes memory data) = msg.sender.call{value: withdrawBalance}("");
    withdrawBalance = 0;
}
```

Obrázek 13 Solidity kód – funkce withdrawContract

Funkce používá modifikátor *onlyOwner*, který zajišťuje, že funkci může použít pouze vlastník kontraktu. Poté co kontrakt pošle požadovanou částku, vynuluje proměnou *withdrawBalance*, aby si vlastník kontraktu nemohl vybrat více prostředků, než je určeno.

Gettery

getEvent

Vrátí objekt představující událost získanou z proměnné *_Events* pomocí identifikátoru události, který byl funkci předaný jako parametr.

getMyEvents

Vrátí pole identifikátorů událostí patřící danému uživateli, které funkce získá z proměnné *_UserToEvents* skrze adresu připojeného uživatele.

getEventCount

Vrátí počet vytvořených událostí uložený v proměnné *_EventCount*.

getCountOfSoldTickets

Vrátí počet prodaných vstupenek události určené identifikátorem, který byl funkci předán jako parametr. Tuto informaci funkce získá rozdílem délky pole identifikátorů vytvořených vstupenek *TicketsId* a délky pole identifikátorů vrácených vstupenek *ReturnedTicketsId*.

getMyTickets

Vrátí pole identifikátorů vstupenek patřící danému uživateli získané z proměnné *_UserToTickets* pomocí adresy připojeného uživatele.

getEventByTicket

Vrátí identifikátor události patřící k dané vstupence získané z proměnné *_TicketToEvent* pomocí identifikátoru vstupenky, který byl předaný jako parametr funkce.

getCurrentTicketId

Vrátí identifikátor vstupenky, která se má vytvořit jako další, získaný pomocí proměnné *_tokenIdCounter*.

getCountOfReturnedTickets

Vrátí počet vrácených vstupenek dané události určené identifikátorem, předaným jako parametr funkce. V podstatě vrací délku pole identifikátorů vrácených vstupenek *ReturnedTicketsId*.

```
//GETTERS
function getEvent(uint eventId) public view returns(Event memory){
    return _Events[eventId];
}

function getMyEvents() public view returns(uint [] memory){
    return _UserToEvents[msg.sender];
}

function getEventCount() public view returns(uint){
    return _EventCount;
}

function getCountOfSoldTickets(uint eventId)public view returns(uint){
    return  _Events[eventId].TicketsId.length - _Events[eventId].ReturnedTicketsId.length;
}

function getMyTickets() public view returns(uint [] memory){
    return _UserToTickets[msg.sender];
}

function getEventByTicket(uint ticketId) public view returns(uint){
    return _TicketToEvent[ticketId];
}

function getCurrentTicketId() public view returns(uint){
    return _tokenIdCounter.current();
}

function getCountOfReturnedTickets(uint eventId) public view returns (uint){
    return _Events[eventId].ReturnedTicketsId.length;
}
```

Obrázek 14 Solidity kód – gettery

getFinalAmountOfWithdraw

Vrátí dvě proměnné, konečnou částku kryptoměny, kterou si pořadatel události může vybrat z kontraktu a poplatek, o který je konečná částka snížena za poskytnuté služby. Výše poplatku je jedno procento z ceny každé prodané vstupenky. Celkový poplatek je tak vypočítán vynásobením poplatku za jednu vstupenku a počtu prodaných vstupenek. Celkové příjmy události se vypočítají vynásobením ceny vstupenky počtem prodaných vstupenek. Konečná částka jsou celkové příjmy události snížené o celkový poplatek.

```
function getFinalAmountOfWithdraw(uint eventId) public view returns (uint, uint){
    //VÝPOČET WITHDRAW
    uint soldTickets = _Events[eventId].TicketsId.length - _Events[eventId].ReturnedTicketsId.length;
    //POPLATEK 1% z prodané vstupenky
    uint feePerTicket = _Events[eventId].Price / 100;
    uint finalFee = feePerTicket * soldTickets;
    uint finalAmount = soldTickets * _Events[eventId].Price - finalFee;
    return (finalAmount, finalFee);
}
```

Obrázek 15 Solidity kód – getter getFinalAmountOfWithdraw

Settery

setDate

Změní datum konání události určené identifikátorem, který je předán jako první parametr funkce. Druhým parametrem je nové datum. Funkci může volat pouze pořadatel události a datum nesmí být z minulosti.

setStopReturnDate

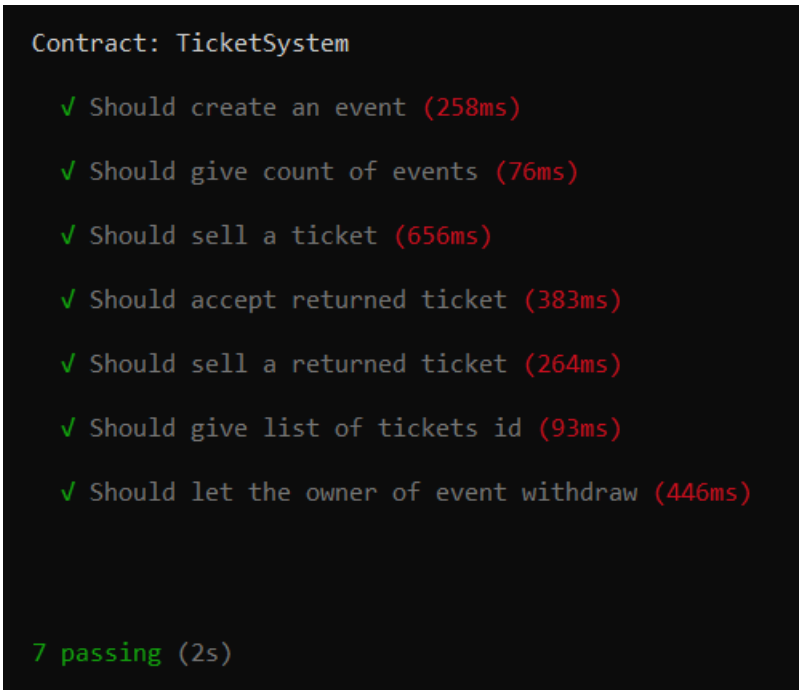
Změní datum ukončující možnost vrácení vstupenek. Funkce funguje stejným způsobem jako předcházející setter *setDate*.

```
//SETTERS
function setDate(uint eventId, uint date) public {
    require(msg.sender == _Events[eventId].Owner, "You are not owner of Event");
    require(date > block.timestamp, "Date from the past");
    _Events[eventId].Date = date;
}


function setStopReturnDate(uint eventId, uint date) public {
    require(msg.sender == _Events[eventId].Owner, "You are not owner of Event");
    require(date > block.timestamp, "Date from the past");
    _Events[eventId].StopReturnDate = date;
}
```

Obrázek 16 Solidity kód – settery

5.1.2 Testování kontraktu

Testování kontraktu bylo provedeno pomocí frameworku *Mocha* v prostředí *Truffle*.



Obrázek 17 Testování kontraktu

K otestování bylo napsáno 7 testů, které ověřují základní funkce systému.

Should create an event
Testuje, zda kontrakt umí vytvořit událost a jestli dokáže vypsat informace o detailech události.

Should give count of events
Testuje, zda kontrakt dokáže vypsat počet událostí. Tato informace je důležitá pro výpis všech událostí. Jelikož jsou události číslované od 1, můžeme snadno vypsat všechny události, když známe pouze jejich celkový počet.

Should sell a ticket
Testuje, zda kontrakt umožňuje uživateli koupit vstupenku.

Should accept returned ticket
Testuje, zda kontrakt umožňuje uživateli vrátit vstupenku.

Should sell a returned ticket
Testuje, zda kontrakt umí prodat vrácenou vstupenku.

Should give list of tickets id
Testuje, zda kontrakt dokáže vypsat seznam identifikátorů vstupenek patřící danému uživateli.

Should let the owner of event withdraw
Testuje, zda kontrakt umožňuje pořadateli vybrat příjmy z prodaných vstupenek.

5.2 Frontend

Frontend byl napsán v programovacím jazyce Javascript za pomocí frameworku *Vue*. Komunikaci s kontraktem zajišťuje knihovna *Ethers*. Pomocí této knihovny je umožněno připojení *MetaMask* peněženky, přes kterou probíhá komunikace s kontraktem. Jelikož je potřeba připojení kryptoměnové peněženky sdílet mezi komponentami *Vue*, ukládá se v podobě stavu, pomocí knihovny *Pinia*, která toto sdílení umožňuje. Pro design aplikace byl použit framework *Bootstrap*.

S každou vytvořenou vstupenkou je generován QR kód a metadata, která jsou ukládána na decentralizované úložiště *IPFS*. Data jsou nahrávána pomocí služby *NFT.Storage*, která nabízí uložení na *IPFS* zdarma.

5.2.1 Rozbor kódu

Webová aplikace je psaná ve frameworku *Vue* ve verzi 3.2.47. Zdrojový kód byl napsán v editoru *Visual Studio Code*. Pro správu knihoven byl použit balíčkovací systém *Npm*.

5.2.1.1 Knihovny

Vue Router: knihovna pro směrování stránek aplikace

Ethers: knihovna pro připojení *MetaMask* peněženky a komunikaci se smart kontraktem

Pinia: knihovna pro uložení a sdílení stavů mezi komponentami

qrcode: knihovna pro generování QR kódů

canvas-to-blob: knihovna pro vytvoření blob souboru, který je převeden na obrázek QR kódu

5.2.1.2 Struktura webové aplikace

Webová aplikace používá strukturu *Single-page application,* To znamená, že se obsah jednotlivých stránek dynamicky načítá do jedné hlavní stránky. Hlavní stránka obsahuje rozvržený layout aplikace a o obsah stránek se starají jednotlivé komponenty, *views*. Tento způsob je označovaný jako směrování na straně klienta, jelikož dochází pouze k načtení obsahu komponent, a nikoliv celé stránky. To má za následek urychlení přechodu mezi stránkami.

5.2.1.3 Seznam view komponent

Každá view komponenta představuje jednu stránku webu, která je načítána uvnitř těla aplikace.

EventsView
Komponenta představující stránku pro zobrazení seznamu událostí.

MyEventsView
Komponenta představující stránku pro zobrazení seznamu událostí, u kterých je připojený uživatel pořadatelem.

MyTicketsView
Komponenta představující stránku pro zobrazení seznamu vstupenek připojeného uživatele.

CreateEventView
Komponenta představující stránku pro vytvoření nové události.

EventDetailView
Komponenta představující stránku pro zobrazení detailů události s možnostmi její editace.

BuyTicketView
Komponenta představující stránku pro zobrazení detailů události s možností koupení vstupenky.

TutorialView
Komponenta představující stránku obsahující návod přidání vstupenky do MetaMask peněženky

TicketOwnersView
Komponenta představující stránku pro zobrazení tabulky majitelů vstupenek dané události.

5.2.1.4 *Propojení s kontraktem*

Připojení MetaMask peněženky a propojení s kontraktem zajišťuje metoda *connectMetamask,* díky použití funkcí knihovny *ethers*. Metoda si získá a uloží do konstanty *provider* poskytovatele připojení do Ethereum sítě. Do konstanty *signer* si načte účet připojovaného uživatele. Pomocí konstanty *signer* jsou podepisovány transakce, požadavky na kontrakt. Do proměnné *signer* je uložena adresa připojeného uživatele a je zobrazována v hlavičce aplikace. Konstanta *contract* představuje propojení kontraktu a webové aplikace. Umožňuje pracovat s kontraktem jako s objektem v Javascriptu. Nakonec metoda tyto konstanty uloží v podobě stavu za pomocí knihovny *Pinia*, aby je bylo možné sdílet mezi jednotlivými komponentami.

```
async connectMetamask(){

  const provider = new ethers.providers.Web3Provider(window.ethereum, "any");
  await provider.send("eth_requestAccounts", []);
  const signer = provider.getSigner();

  this.signer = await signer.getAddress()

  const contract = new ethers.Contract(import.meta.env.VITE_CONTRACT_ADDRESS, TicketSystem.abi, signer)

  //SETTING PINIA STORE
  const store =  useEthConnectionStore()
  store.setConnection(provider, signer, contract)

  this.loaded = true

}
```

Obrázek 18 Vue kód – propojení s kontraktem

5.2.1.5 *Vytvoření událostí*

Jako první se načte stav knihovny *Pinia* obsahující propojení s kontraktem a je uložen do konstanty *store*. Tuto konstantu obsahují všechny následně probírané metody. Aby nedošlo k neustálému opakování informace, nebude u těchto metod dále popisována.

Před zavoláním funkce kontraktu pro vytvoření události, musí být upraveny datumy a cena vstupenky do požadovaného formátu. Datum z formuláře je převeden do UNIX formátu pomocí funkce *Date.parse*. Protože datum ukládáme na blockchain v počtu sekund a převedené datum je v milisekundách, je vydělené tisícem, aby bylo dosaženo požadované jednotky. Cena vstupenky předána z formuláře je v desetinném čísle znázorňující částku v ETH. Jelikož se na blockchain ukládá ve WEI jednotkách, musí být cena do této jednotky převedena. Protože převedené číslo může přesahovat rozsah čísla v Javascriptu, musí být ukládáno pomocí objektu *BigInt*. Po úpravě jsou data z formuláře předána funkci kontraktu, která vytvoří novou událost. Aplikace poté upozorní uživatele na úspěšné vytvoření události a přesměruje na stránku *MyEventsView*, kde je nová událost zobrazena v seznamu událostí.

```
async submit(){

    try{
    const store =  useEthConnectionStore()
    //FORMAT DATE
    //Date is in ms but blockchain saves in s
    let dateUnix = Date.parse(this.event.date) / 1000
    let stopDateUnix = Date.parse(this.event.stopDate) / 1000
    //ETH TO WEI
    let weiPrice = BigInt(this.event.price * Math.pow(10,18))
    //CREATING EVENT
    let txn = await store.contract.createEvent(this.event.name, this.event.countTicket,
                                                weiPrice, dateUnix, stopDateUnix)
    let txnResults = await txn.wait() // await for transaction to be mined
    console.log(txnResults)
    alert("Creation was successful")
    router.push('/my-events')
    }
    catch (error){
        console.log(error)
        if(error.value = ""){
            alert("Please fill in all fields!")
        }
        else if(error.code == -32603){
            alert(error.data.data.reason)
        }
        else{
            alert("Failed to create event")
        }
    }
}
```

Obrázek 19 Vue kód – vytvoření události

5.2.1.6 Editace události

Editace je možná pouze u datumu konání události a datumu ukončení možnosti vrácení vstupenek. Tuto editaci zajišťují metody *editDate* a *editStopDate*. Obě metody fungují stejně jen s rozdílem volání jinačí funkce kontraktu. Aby bylo možné předat datum funkci kontraktu, musí projít převedením ze zobrazovaného formátu na UNIX formát uvedený v sekundách, vysvětleno u metody pro vytvoření události. Po provedení transakce je uživatel informován o úspěšné editaci. Aby byly změny zobrazeny ihned po provedení akce, načte metoda detaily události znovu zavoláním metody *getMyEvent*.

```
async editDate(){
    try{
    const store =  useEthConnectionStore()
    // Date is in ms, whereas date in blockhain is in s
    let txn = await store.contract.setDate(this.event.id, Date.parse(this.event.date)/1000)
    let txnResults = await txn.wait() // await for transaction to be mined
    alert("Date was edited")
    this.getMyEvent()
    }
    catch (error){
        console.log(error)
        alert("Failed to edit date")
    }
},

async editStopDate(){
    try{
    const store =  useEthConnectionStore()
    let txn = await store.contract.setStopReturnDate(this.event.id, Date.parse(this.event.stopDate)/1000)
    let txnResults = await txn.wait() // await for transaction to be mined
    alert("Date was edited")
    this.getMyEvent()
    }
    catch (error){
        console.log(error)
        alert("Failed to edit date")
    }

},
```

Obrázek 20 Vue kód – editace události

5.2.1.7 *Zobrazení seznamu událostí*

Získání seznamu událostí zajišťuje metoda *getEvent*, která je volána při načítání stránky. Metoda získá z kontraktu počet všech událostí, a protože jsou událostí číslované postupně od jedničky, je snadné vytvořit pomocí cyklu pole všech identifikátorů událostí. Identifikátory jsou načítány od největšího, aby bylo pole seřazeno od nejnovějších událostí.

Detaily události jsou získávány pomocí funkce *loadEvent*. V kódu lze vidět, že dochází k načtení pouze prvních devíti událostí. Počet událostí je omezený, aby nedocházelo k dlouhému načítání stránky.

```
async getEvents(){
    const store =  useEthConnectionStore()
    let eventCount = await store.contract.getEventCount()
    let eventsId = []
    //LOADS EVENT IDS FROM THE NEWEST
    for(let i = Number(eventCount)  ; i > 0; i--){
        eventsId.push(i)
    }
    eventsId.forEach(eventId => this.allEventsIds.push(eventId))


    //LOADS EVENTS TO listOfEvents
    let j = 0
    for(let i = 0; i < this.allEventsIds.length; i++){
        if(j > 8){
            break
        }
        this.loadEvent(this.allEventsIds[i], store)
        j++
    }
},
```

Obrázek 21 Vue kód – zobrazení seznamu událostí 1/4

Metoda *loadEvent* získá detaily události zavoláním funkce kontraktu *getEvent*, které předá identifikátor načítané události. Funkce kontraktu vrátí objekt představující událost s jejími vlastnostmi a tento objekt přidá do seznamu událostí *listOfEvents*. Události uložené v proměnné *listOfEvents* jsou zobrazovány v aplikaci, proto pokud aplikace chce načíst další události, musí je přidat do této proměnné.

```
async loadEvent(id, store){
    let event = await store.contract.getEvent(id)
    this.listOfEvents.push(event)
}
```

Obrázek 22 Vue kód – zobrazení seznamu událostí 2/4

K načtení více událostí slouží metoda *loadMore*, která načte dalších devět událostí v pořadí. Tato metoda se volá pomocí tlačítka zobrazovaného pouze v případě, že nejsou všechny události načtené. Tuto informaci aplikace zjistí, když od délky listu všech identifikátorů událostí odečte délku listu načtených událostí. Pokud je výsledek větší než nula, existují události, které ještě nebyli zobrazeny.

```
<dv v-if="allEventsIds.length - listOfEvents.length > 0" class="text-center mt-5">
    <button class="btn btn-secondary" @click="loadMore">Load more</button>
</dv>
```

Obrázek 23 Vue kód – zobrazení seznamu událostí 3/4

```
async loadMore(){
    const store =  useEthConnectionStore()
    //LOADS MORE EVENTS TO listOfEvents
    let j = 0
    for(let i = this.listOfEvents.length; i < this.allEventsIds.length; i++){
        if(j > 8){
            break
        }
        this.loadEvent(this.allEventsIds[i], store)
        j++
    }

},
```

Obrázek 24 Vue kód – zobrazení seznamu událostí 4/4

5.2.1.8 Zobrazení detailů události

Načtení detailů události zajišťuje metoda *getMyEvent*. Detaily události jsou získané pomocí funkce kontraktu *getEvent*, které je předáván identifikátor události. Tento identifikátor je uvedený v URL adrese stránky zobrazované události. Následně jsou detaily události předány struktuře event, jejíž vlastnosti jsou zobrazovány v aplikaci. Získaná cena vstupenky je převedená z WEI na ETH. Datumy události jsou převedeny nejprve ze sekund na milisekundy a následně do klasického formátu datumu zobrazovaného v aplikaci.

```
async getMyEvent(){
    const store =  useEthConnectionStore()
    let event = await store.contract.getEvent(this.$route.params.id)
    console.log(event)
    this.event.id = event.Id
    this.event.name = event.Name
    this.event.owner = event.Owner
    this.event.sold = await store.contract.getCountOfSoldTickets(this.event.id)
    this.event.countTicket = event.MaxTicketCount
    this.event.price = event.Price / Math.pow(10,18)
    this.event.date = new Date(Number(event.Date) * 1000).toDateString()
    this.event.stopDate = new Date(Number(event.StopReturnDate) * 1000).toDateString()
    this.event.withdrawed = event.Withdrawed
```

Obrázek 25 Vue kód – Zobrazení detailů událostí

5.2.1.9 Prodej vstupenek

Prodej vstupenek je nejsložitější akce celé aplikace, jelikož řeší hned několik problémů. Vstupenka se vytváří v momentu, kdy je zakoupena, proto mají vrácené vstupenky přednost v prodeji, jelikož už byli vytvořeny. Nejdříve tak aplikace musí zjistit počet vrácených vstupenek dané události. Jakmile neexistují žádné další vrácené vstupenky, dojde k vytvoření nové.

```
async buyTicket(){
    try{
    this.loading = "Processing..."
    const store =  useEthConnectionStore()
    const countOfReturnedTickets = await store.contract.getCountOfReturnedTickets(this.event.id)
    const countOfReturned = Number(countOfReturnedTickets._hex)

    //IF EVENT HAS RETURNED TICKETS
    if(countOfReturned > 0){
        let txn = await store.contract.buyReturnedTicket(this.event.id, {value: this.event.price})
        let txnResults = await txn.wait() // await for transaction to be mined
    }
    //ELSE CREATE NEW TICKET
    else{
```

Obrázek 26 Vue kód – prodej vstupenek 1/3

Postup vytvoření vstupenky probíhá následovně. Nejprve aplikace zjistí identifikátor následující vstupenky, poté dojde k vygenerování QR kódu, kde je toto ID uloženo. Poté se QR kód uloží do PNG obrázku a společně s metadaty je nahrán na IPFS. URL adresa, kde jsou metadata nahraná je spolu s identifikátorem vstupenky předána funkci kontraktu, která vytvoří novou vstupenku.

Po vytvoření vstupenky upozorní aplikace na úspěšně provedenou akci a přesměruje uživatele na seznam jeho vstupenek, kde je přidána nově vytvořená vstupenka.

```
//ELSE CREATE NEW TICKET
else{
    //GET TICKET ID FOR GENERATING QR CODE
    const ticketId = await store.contract.getCurrentTicketId()
    const ticketIdN = Number(ticketId._hex)

    //CREATING QR CODE
    const qrCode = await QRCode.toDataURL(String(ticketIdN))
    const blob = toBlob(qrCode)
    const file = new File([blob], "image.png", { type: "image/png"});

    //STORING NFT METADATA
    const NFT_STORAGE_KEY = import.meta.env.VITE_NFT_STORAGE_API_KEY
    const client = new NFTStorage({ token: NFT_STORAGE_KEY })

    const nft = {
        name: "Ticket: " + String(ticketIdN),
        image: file,
        description: "This is ticket for event: " + this.event.name
    }

    const cid = await client.store(nft)
    //EDIT URL TO REQUIRED FORM
    const url = 'https://' + cid.ipnft + '.ipfs.dweb.link/metadata.json'

    //CREATING TICKET
    let txn = await store.contract.buyTicket(this.event.id, url, {value: this.event.price})
    let txnResults = await txn.wait() // await for transaction to be mined
}
```

Obrázek 27 Vue kód – prodej vstupenek 2/3

```
    }

    this.loading = ""
    alert("Purchase completed")
    //REDIRECT TO My Tickets
    router.push('/my-tickets')
    }
    catch (error){
        console.log(error)
        alert("Purchase failed")
    }
},
```

Obrázek 28 Vue kód – prodej vstupenek 3/3

5.2.1.10 Zobrazení seznamu vstupenek

Získání seznamu vstupenek zajišťuje metoda *getMyTickets*, která je volána při načítání stránky zobrazující vstupenky připojeného uživatele. Metoda zavolá funkci kontaktu pro získání pole identifikátorů vstupenek patřící připojenému uživateli. Následně dojde k převrácení tohoto pole, aby byli vstupenky seřazeny od nejnovější po nejstarší. K načtení detailů vstupenky je používaná metoda *loadTicket*. V případě většího počtu vstupenek je tato metoda použita pouze desetkrát, aby nedošlo k pomalému načítání stránky. Načítání dalších vstupenek je řešeno stejným způsobem jako u zobrazení seznamu událostí.

```
async getMyTickets(){
    const store =  useEthConnectionStore()
    let tickets = await store.contract.getMyTickets()
    //LOADS TICKET IDS FROM THE NEWEST
    tickets.forEach(ticket => this.allTicketsIds.unshift(Number(ticket._hex)))
    //LOADS TICKETS TO listOfTickets
    let j = 0
    for(let i = 0; i < this.allTicketsIds.length; i++){
        if(j > 9){
            break
        }
        this.loadTicket(this.allTicketsIds[i])
        j++
    }
},
```

Obrázek 29 Vue kód – zobrazení seznamu vstupenek

Protože je u každé vstupenky zobrazen název události, získá metoda *loadTicket* nejprve identifikátor události, ke které vstupenka patří, pomocí funkce kontraktu *getEventByTicket*. Této funkci je předáno ID vstupenky, podle kterého zjistí přiřazenou událost.
Metoda poté zavolá funkci kontraktu *getEvent* k získání detailů dané události. Ty jsou společně s identifikátorem vstupenky uloženy do objektu představující vstupenku Tento objekt je pak je přidán do seznamu seznamu vstupenek *listOfTickets*, z kterého jsou načtené vstupenky zobrazovány v aplikaci.

```
async loadTicket(id){
    const store =  useEthConnectionStore()
    //GET EVENT ID BY TICKET ID
    let eventId = await store.contract.getEventByTicket(id)
    //GET EVENT
    let event = await store.contract.getEvent(eventId)
    //LOAD TICKET WITH EVENT INFO
    let ticket = {ticketId: id, event: event}
    this.listOfTickets.push(ticket)
},
```

Obrázek 30 Vue kód – zobrazení seznamu vstupenek

5.2.1.11 Vrácení vstupenky

Vrácení vstupenky zajišťuje metoda *returnTicket*. Uvnitř této metody je volaná funkce kontraktu pro vrácení vstupenky. Po úspěšném vrácení vstupenky dojde k vyprázdnění a znovu načtení seznamu zobrazovaných vstupenek *listOfTickets*, aby se změny projevili reaktivně ihned po dokončení metody.

```
async returnTicket(ticketId){
    try{
    const store =  useEthConnectionStore()
    let txn = await store.contract.returnTicket(ticketId)
    let txnResults = await txn.wait() // await for transaction to be mined
    console.log(txnResults)
    alert("Return was successful")
    //RELOAD LIST OF TICKETS AFTER RETURN
    this.listOfTickets = []
    this.getMyTickets()
    }
    catch (error){
        console.log(error)
        alert("Failed to create return ticket")
    }
}
```

Obrázek 31 Vue kód – vracení vstupenek

5.2.1.12 Výběr příjmů pořadatelem

Výběr příjmů z prodaných vstupenek zajišťuje metoda *withdraw*. Metoda volá funkci kontraktu pro výběr příjmů pořadatelem a předává jí v parametru identifikátor události. Po provedení transakce je pořadatel upozorněn na úspěšné dokončení akce. Pro zobrazení provedených změn jsou detaily události načteny znovu zavoláním metody *getMyEvent*.

```
async withdraw(){
    try{
    const store =  useEthConnectionStore()
    let txn = await store.contract.withdraw(this.event.id)
    let txnResults = await txn.wait() // await for transaction to be mined
    alert("Withdraw was successful")
    this.getMyEvent()
    }
    catch (error){
        console.log(error)
        alert("Failed to withdraw")
    }
}
```

Obrázek 32 Vue kód – výběr příjmů pořadatelem

5.2.1.13 Zobrazení tabulky majitelů vstupenek

Metoda nejprve získá detaily události, jelikož obsahují seznam identifikátorů přiřazených vstupenek. Poté pro každé ID vstupenky zavolá metodu *loadTicket*. Tato metoda zjistí adresu majitele vstupenky pomocí funkce kontraktu *ownerOf* a uloží ji společně s identifikátorem vstupenky do objektu představující vstupenku. Tento objekt je přidán do seznamu vstupenek *listOfTickets*, z něhož jsou vytvářeny jednotlivé záznamy v zobrazované tabulce vlastníků vstupenek.

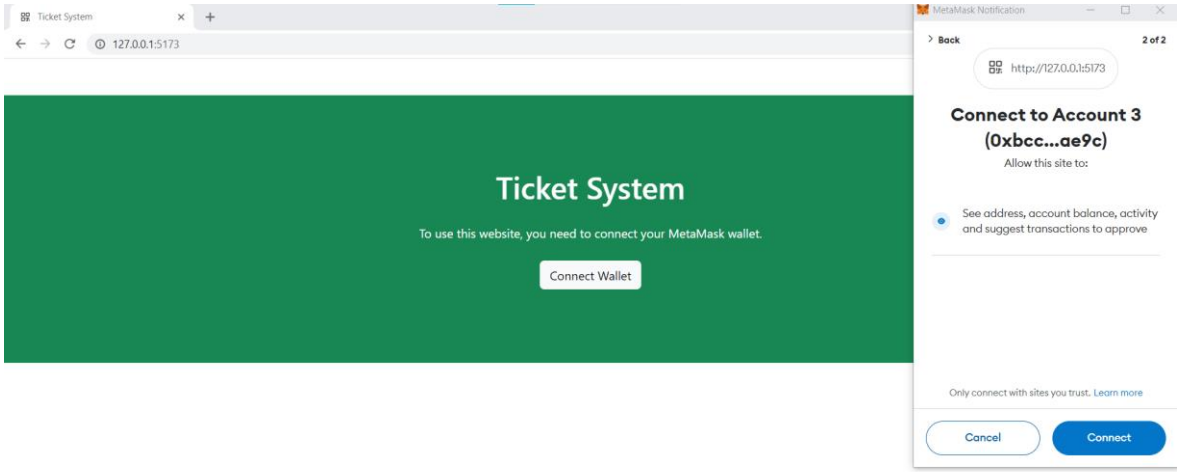
```
async getMyEvent(){
    const store =  useEthConnectionStore()
    let event = await store.contract.getEvent(this.$route.params.id)
    this.event.id = event.Id
    this.event.name = event.Name
    this.event.owner = event.Owner
    this.event.sold = await store.contract.getCountOfSoldTickets(this.event.id)
    this.event.countTicket = event.MaxTicketCount

    this.loading = true
    event.TicketsId.forEach(eventId => this.loadTicket(Number(eventId._hex)))
    this.loading = false

},

async loadTicket(id){
    const store =  useEthConnectionStore()
    //GET OWNER
    let owner = await store.contract.ownerOf(id)
    //LOAD TICKET ID WITH OWNER ADDRESS
    let ticket = {ticketId: id, owner: owner}
    this.listOfTickets.push(ticket)
},
```

Obrázek 33 Vue kód – zobrazení tabulky majitelů vstupenek

5.2.2 Popis jednotlivých stránek

5.2.2.1 Úvodní stránka

Aby mohl uživatel aplikaci používat, musí se nejprve připojit pomocí své MetaMask peněženky, proto se hned po načtení aplikace zobrazí tlačítko, které umožňuje toto připojení.



Obrázek 34 Náhled aplikace – úvodní stránka

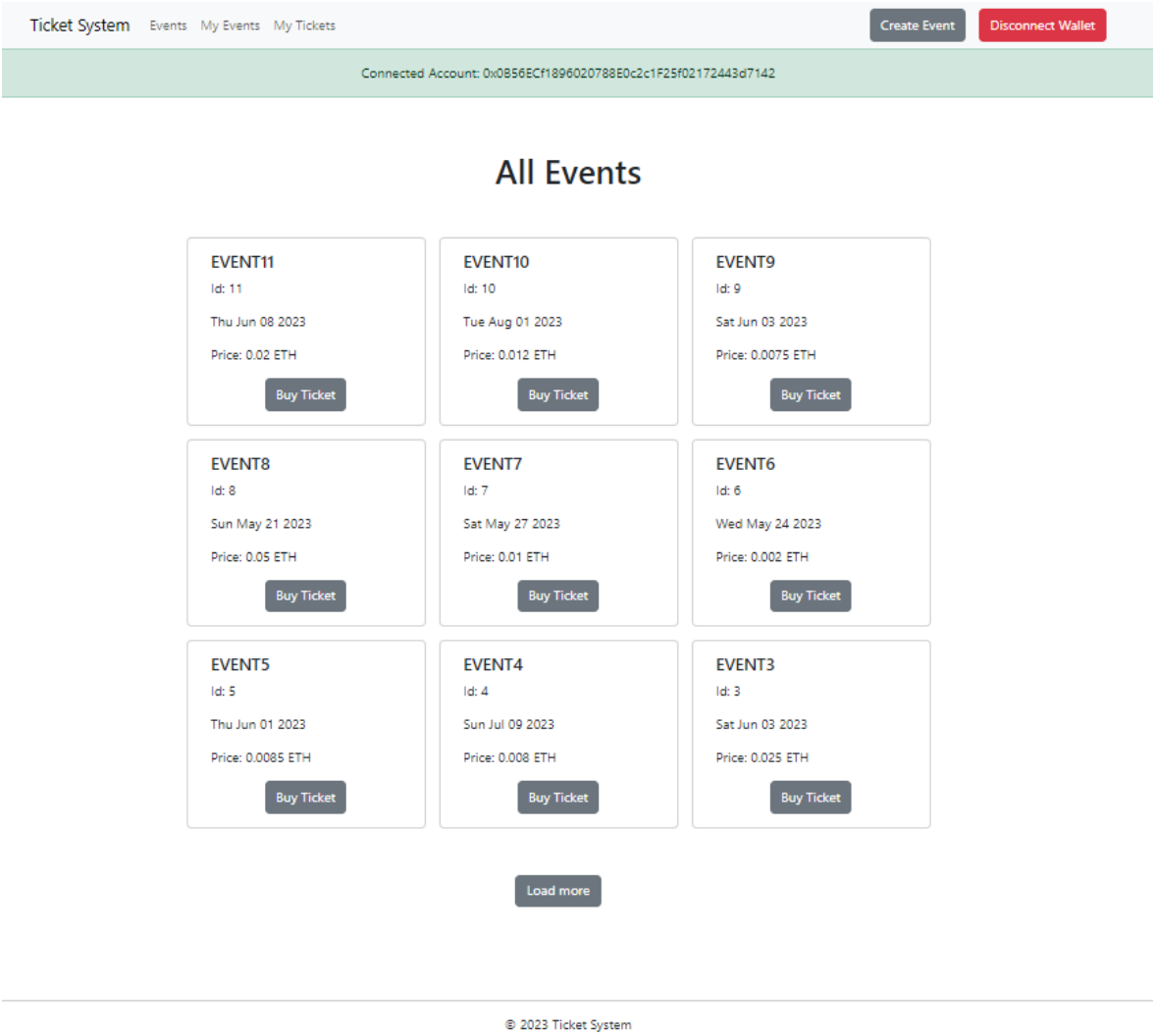
Po kliknutí na tlačítko *Connect Wallet*, se zobrazí MetaMask notifikace pro připojení peněženky. Po potvrzení dojde k připojení peněženky a zobrazení hlavní stránky aplikace.

5.2.2.2 Hlavní stránka

Hlavní stránku tvoří hlavička, dynamicky načítaný obsah a patička. Hlavička je tvořena nabídkou stránek, funkcí aplikace a adresou připojeného uživatele.

Obsah vytváří jednotlivé stránky, komponenty, které jsou v této části zobrazovány pomocí knihovny *Vue Router* v závislosti na zadané URL adrese. Při startu aplikace je nastavena jako původní stránka *EventView*, která načítá jednotlivé události s možností zakoupení vstupenky.

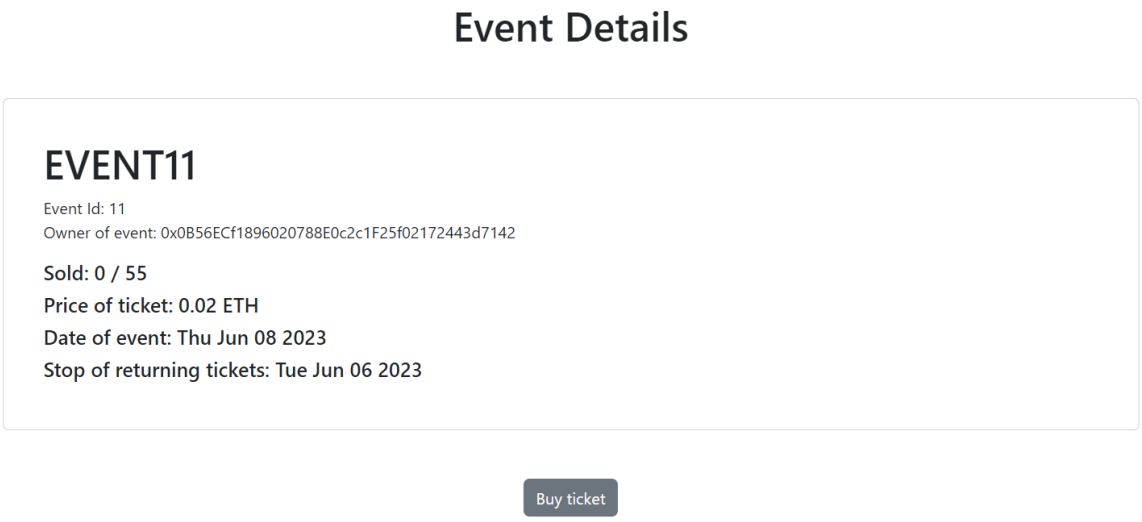
Patička obsahuje pouze autorské právo a rok vytvoření.



Obrázek 35 Náhled aplikace – hlavní stránka

5.2.2.3 Prodej vstupenky

Zobrazí detaily události s možností koupení vstupenky. Uvedené informace jsou název, identifikátor, adresa pořadatele, počet prodaných vstupenek, počet dostupných vstupenek, cena vstupenky, datum konání a datum ukončení možnosti vrácení vstupenek.

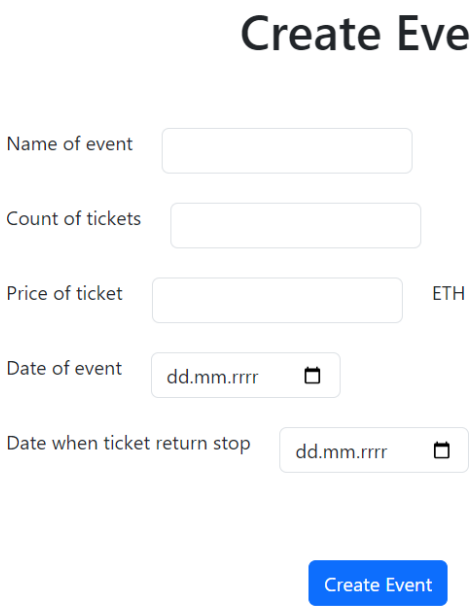


Obrázek 36 Náhled aplikace – prodej vstupenky

5.2.2.4 *Vytvoření události*

Zobrazí formulář pro vytvoření nové události. Formulář obsahuje vstupy pro název události, počet vstupenek, cenu vstupenky, datum konání a datum vrácení vstupenek.

Je umožněno vytvořit více událostí se stejným názvem, jelikož název není identifikátor události. Dále je možné vytvořit událost, s nulovou cenou vstupenky, pokud by chtěl pořadatel dávat vstupenky zdarma.

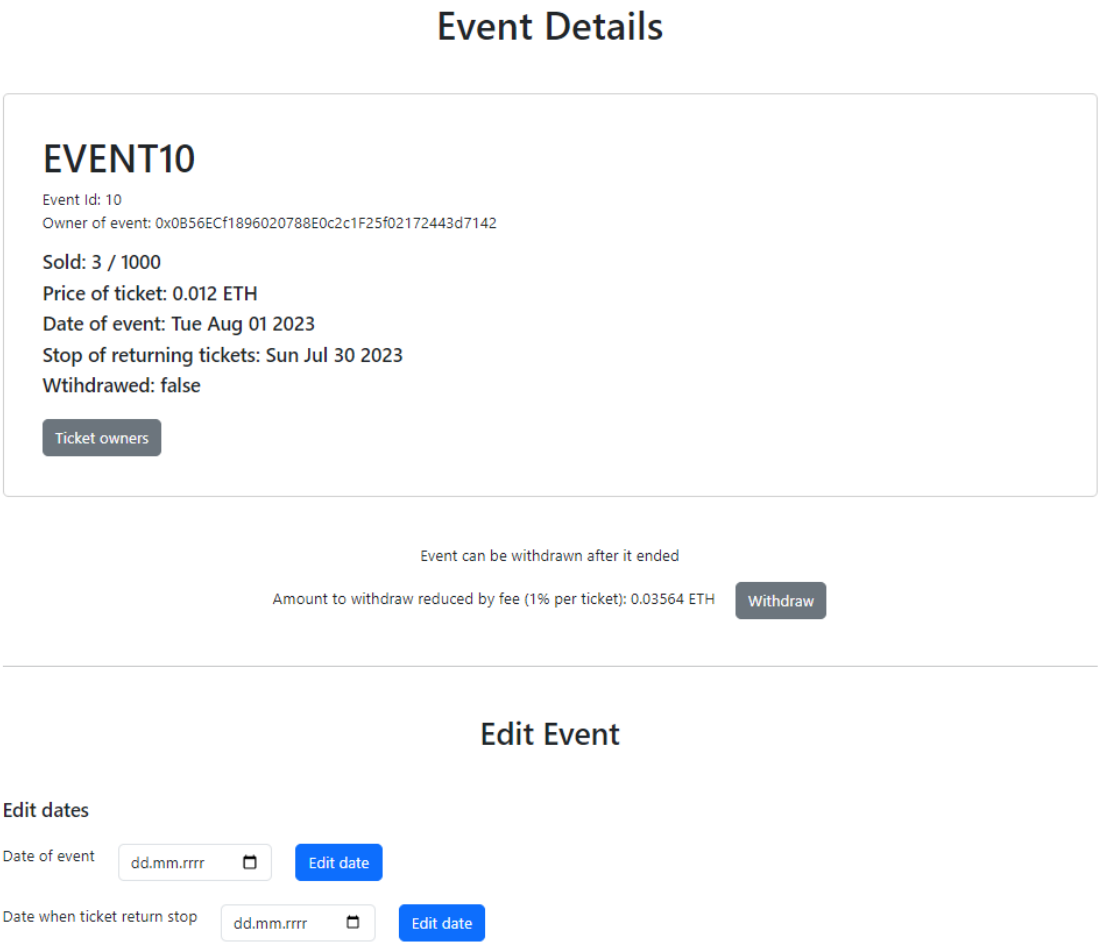


Obrázek 37 Náhled aplikace – vytvoření události

5.2.2.5 Detaily události

Obsahuje detaily události a možnosti její správy pořadatelem. Uvedené informace jsou název události, identifikátor, adresa pořadatele, počet prodaných vstupenek, počet vstupenek celkem, cena, datum konání, datum ukončení možnosti vrácení vstupenek a zda si pořadatel vybral příjmy z prodaných vstupenek.

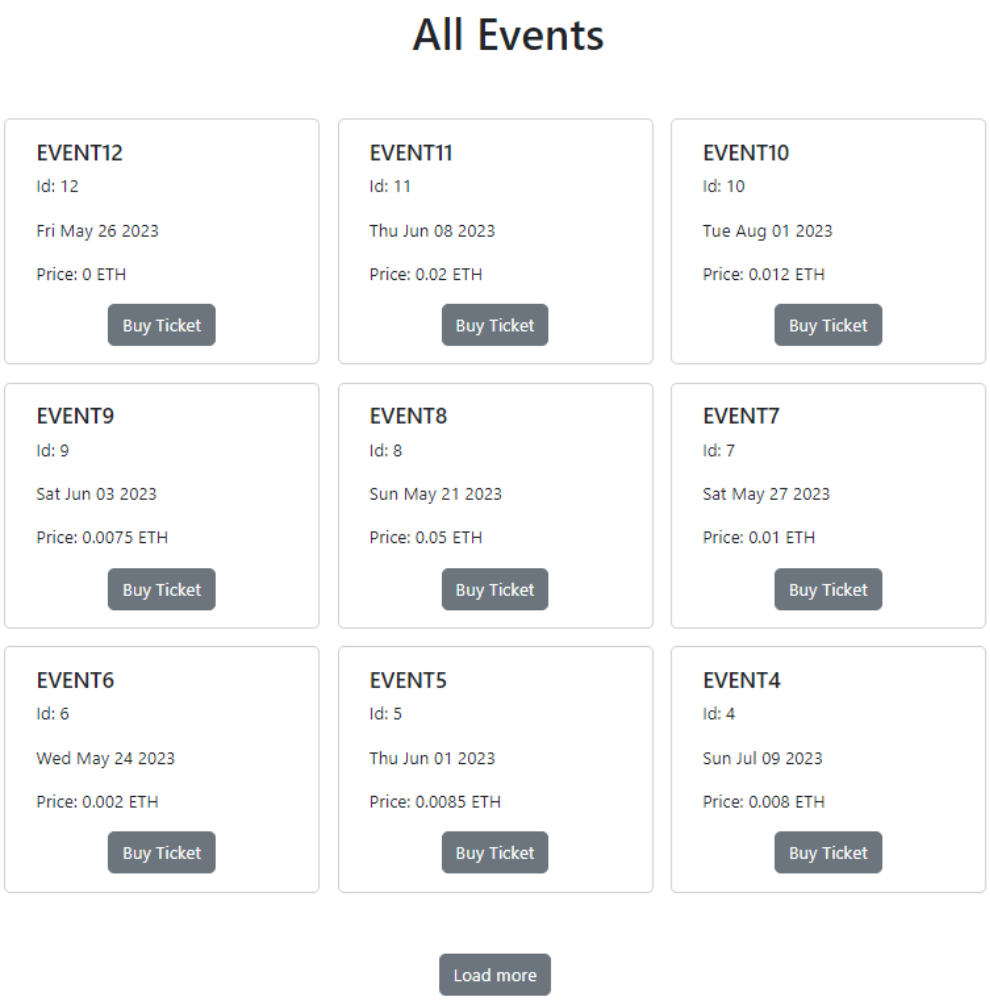
Mezi možnosti správy události patří zobrazení tabulky majitelů vstupenek, výběr příjmů z prodeje vstupenek, editace datumu konání a zastavení vracení vstupenek.



Obrázek 38 Náhled aplikace – detaily události

5.2.2.6 Seznam všech událostí

Zobrazí nabídku všech událostí. Při načtení stránky dojde k zobrazení maximálně devíti událostí. Načtení starších událostí je možné pomocí tlačítka. Každá událost je zobrazena v podobě karty s informacemi o názvu události, identifikátoru, datumu konání, ceně vstupenky a odkazem na stránku prodeje vstupenky.

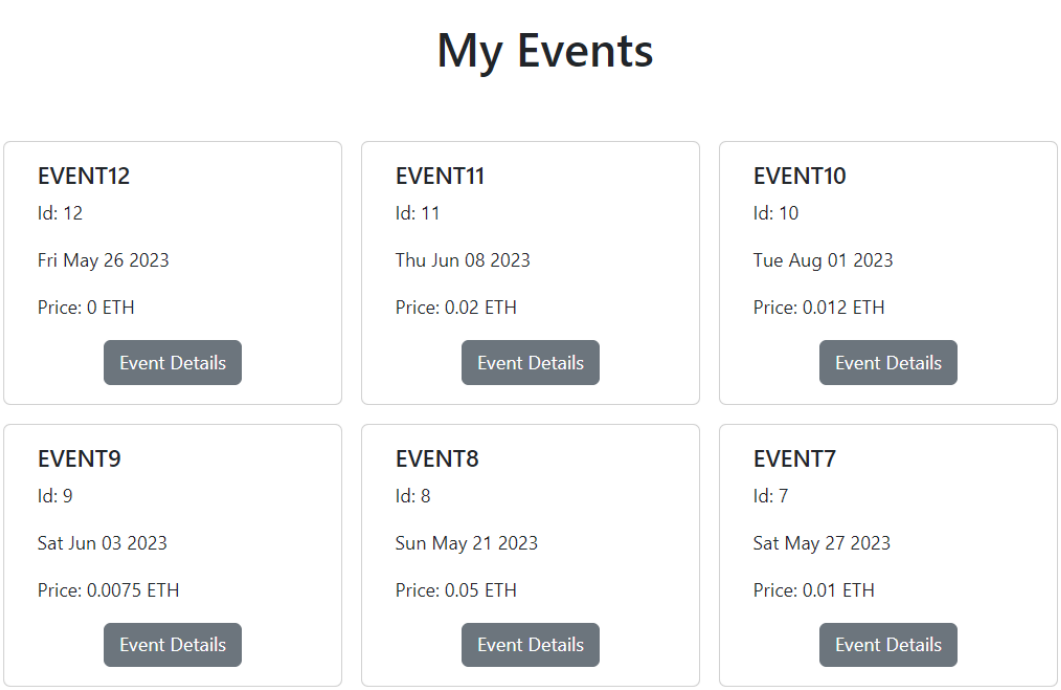


Obrázek 39 Náhled aplikace – seznam všech událostí

5.2.2.7 Seznam událostí pořadatele

Zobrazuje seznam událostí, u kterých je připojený uživatel uveden jako pořadatel. Každá karta události obsahuje informace o názvu, identifikátoru, datumu konání, ceně vstupenky a odkaz na stránku s detaily události.

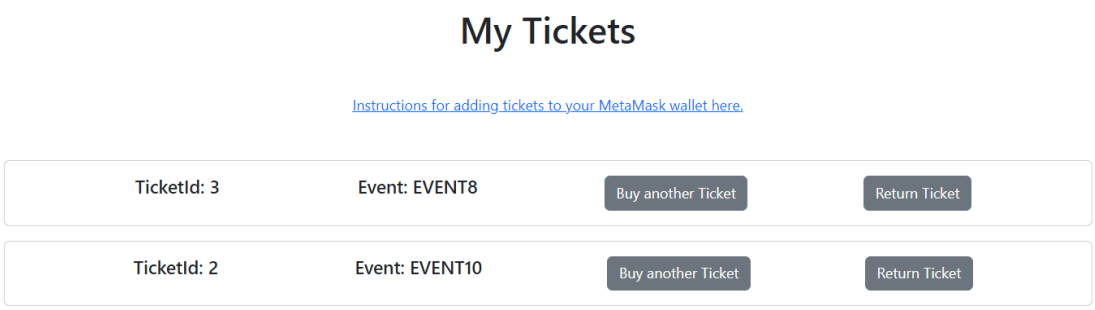
Pokud má uživatel více jak devět událostí, může si načíst starší události pomocí tlačítka. Toto tlačítko se zobrazí pouze v případě, že existují události, které nejsou načtené.



Obrázek 40 Náhled aplikace – seznam událostí pořadatele

5.2.2.8 *Seznam vstupenek*

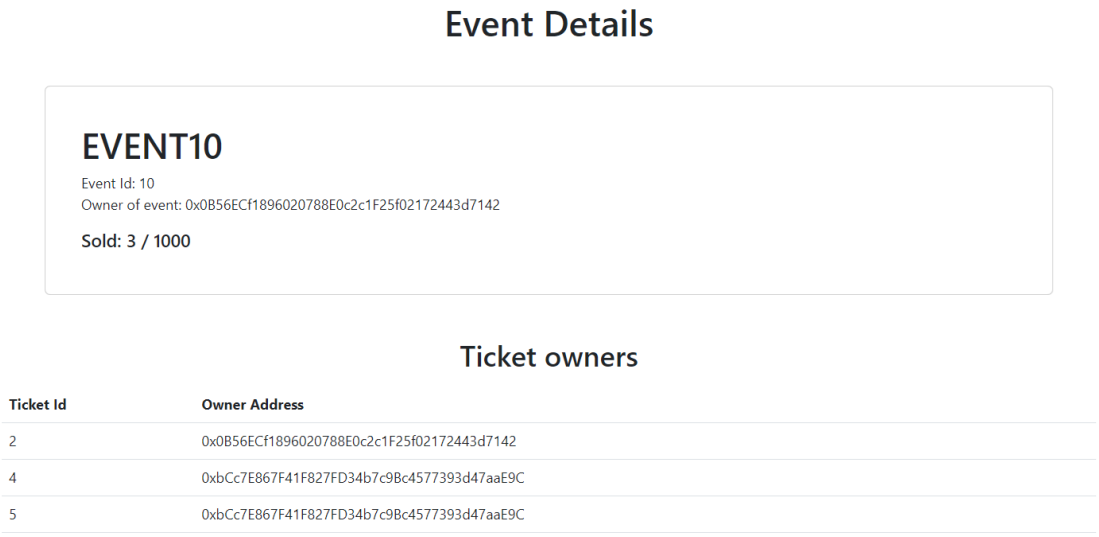
Zobrazuje seznam vstupenek připojeného uživatele. Obsahuje odkaz na návod přidání vstupenky do MetaMask peněženky. U každé vstupenky je možnost vrácení a zakoupení další vstupenky. V případě většího počtu vstupenek se zobrazí pouze prvních deset, starší vstupenky je možné načíst pomocí tlačítka. Toto tlačítko se zobrazí pouze v případě, že existují další vstupenky, které je možné načíst.



Obrázek 41 Náhled aplikace –seznam vstupenek

5.2.2.9 *Tabulka majitelů vstupenek*

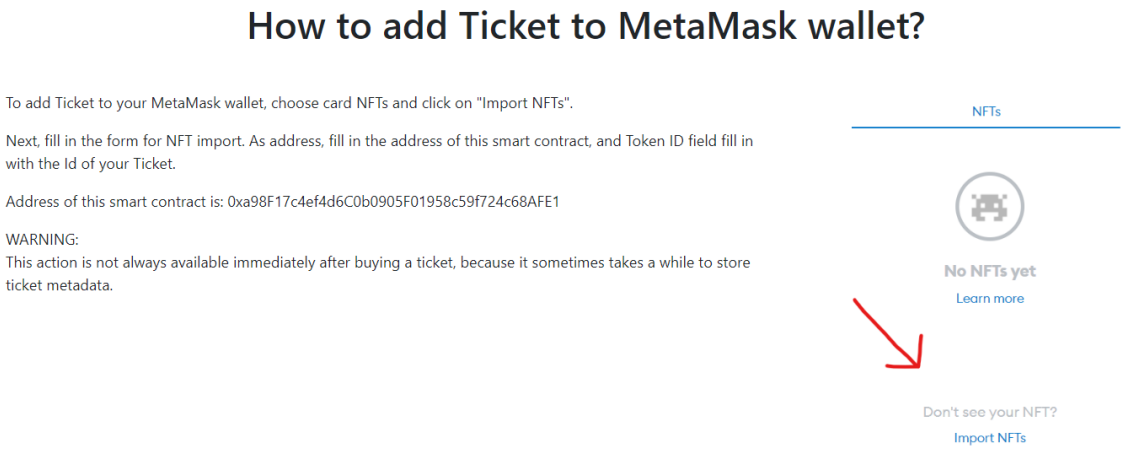
Zobrazí tabulku vstupenek přiřazených dané události. Každý záznam v tabulce je tvořen identifikátorem vstupenky a adresou majitele, který vstupenku koupil.



Obrázek 42 Náhled aplikace – tabulka majitelů vstupenek

5.2.2.10 Návod

Obsahuje návod pro přidání vstupenky do MetaMask peněženky.



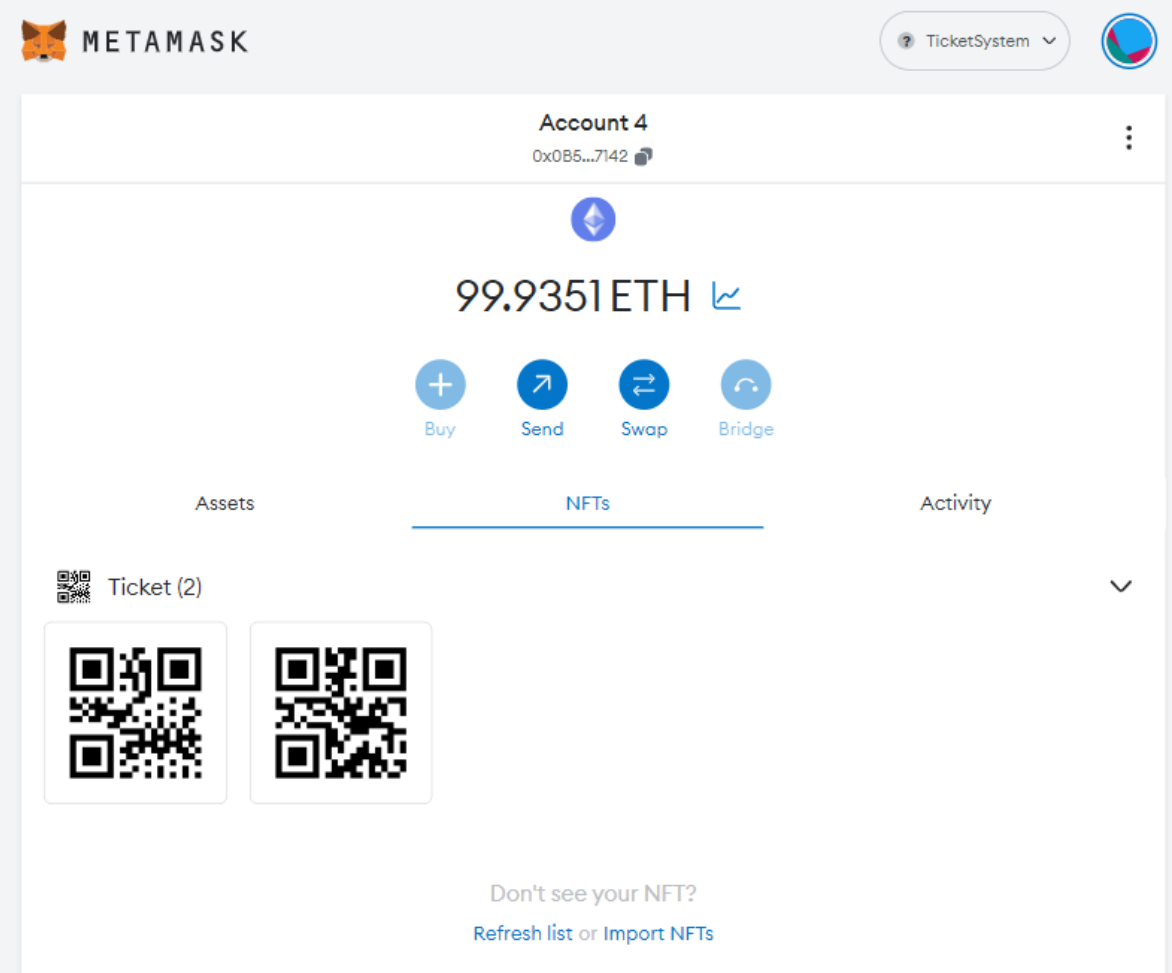
Obrázek 43 Náhled aplikace – návod

5.3 Přidání vstupenky do MetaMask peněženky

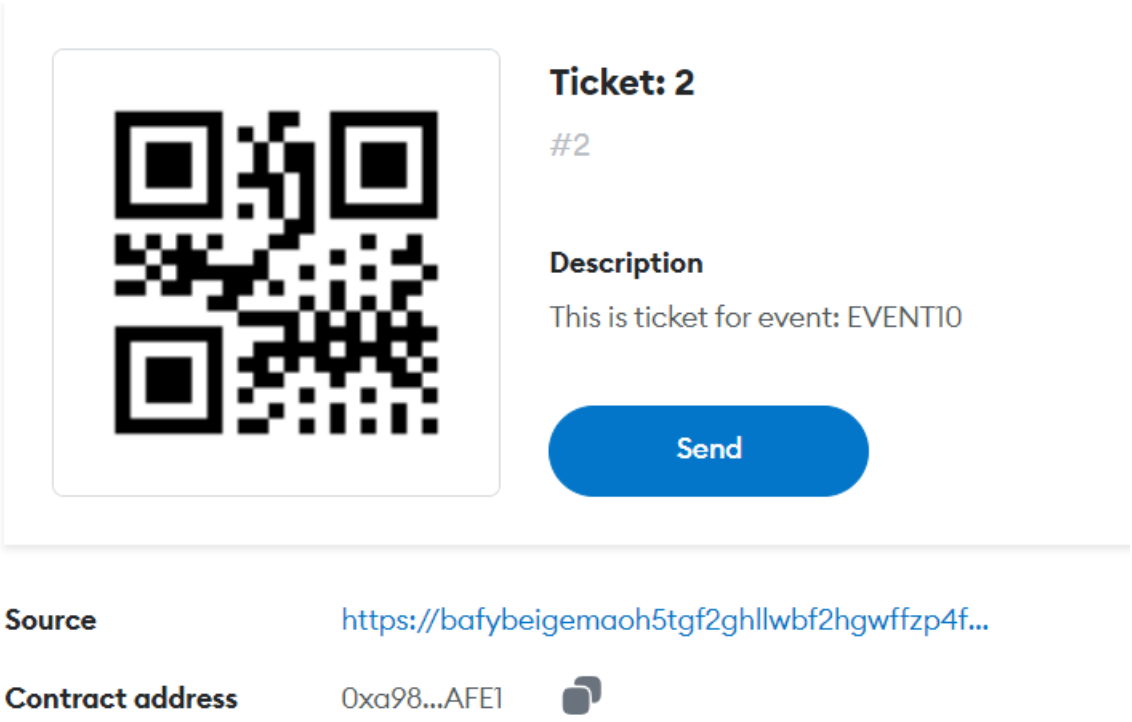
Jedna z výhod NFT vstupenek je jejich přenositelnost v kryptoměnové peněžence. Pro přidání vstupenky je potřeba znát adresu kontraktu a identifikátor vstupenky. Vstupenku lze přidat pomocí možnosti Import NFTs v MetaMask aplikaci. Poté stačí vyplnit formulář a potvrdit přidání NFT. Následně by měla být vstupenka přidána do peněženky uživatele.

Přidání vstupenky do MetaMask peněženky nemusí být úspěšné ihned po zakoupení, jelikož se nejdříve musí uložit QR kód a metadata vstupenky na IPFS. Proto je doporučené po zakoupení vstupenky chvíli počkat.

NFT vstupenky můžou být po skončení události odstraněny pro zvýšení přehlednosti vstupenek v peněžence. Odstraněním vstupenky nepřijde uživatel o její vlastnictví pouze se mu nebude zobrazovat, může ji tak dále používat jako sběratelský předmět.



Obrázek 44 MetaMask – seznam vstupenek



Obrázek 45 MetaMask – detaily vstupenky

6 MOŽNÁ RIZIKA NAVRŽENÉHO SYSTÉMU

Pohyblivá cena vstupenky

Jelikož pořadatel zadává cenu za jednu vstupenku v kryptoměně Ether, vzniká zde problém s pohyblivostí ceny vstupenky, protože hodnota kryptoměny se neustále mění a nejde předpovídat. Proto je potřeba, aby cena vstupenky byla zadávána v klasické měně, například USD a následně při prodeji převedena do kryptoměny. Cenový vztah mezi ETH a měnou nelze brát z klasické API, protože by došlo k oslabení decentralizace systému. Tento problém jde vyřešit použitím služby od společnosti *Chainlink*, která nabízí decentralizovanou databázi s informacemi o cenách jednotlivých měn za ETH. [29]

Pro jednoduchost navrhované aplikace je cena vstupenky zadávaná v kryptoměně Ethereum.

Centralizovaný hosting

Systém používá jako přístupový bod ke kontraktu webovou aplikaci, která je uložena na centrálním webhosting serveru. Tím je bohužel částečně rušena decentralizace aplikace. Pro decentralizovaný webhosting je ale možné použít P2P síť pro ukládání a sdílení dat, jako je *Swarm* nebo *IPFS* a doménu zaregistrovat na decentralizovaném doménovém systému *Ethereum Name System*. Toto řešení ale přináší nevýhodu. Pro přístup k webu je zapotřebí prohlížeč, který je schopen pracovat s výše zmíněnými web3 technologiemi, například *Brave*. [30]

Účtování příjmů za služby systému v kryptoměně

Systém si ponechává malou část ceny prodané vstupenky jako poplatek za poskytnutou službu. Příjmy za poskytnuté služby jsou tedy v kryptoměně. Pokud by aplikace byla nasazena do reálného světa, mohlo by dojít k problémům s účtováním kryptoměn. V takovém případě se nabízí možnost osvobodit systém od původních poplatků a zpoplatnit pořadatele při vytvoření události v tradiční měně. Například v podobě pevné částky odvíjející se od počtu vstupenek, které chce uživatel prodat.

Ověřování vstupenek

Systém nemá dořešené ověřování vstupenek. Vstupenky má uživatel uložené jako NFT na své MetaMask peněžence, kde má každá vstupenka svůj QR kód. Jelikož potřebujeme číst QR kódy vstupenek, jeví se jako nejlepší způsob vytvořit aplikaci na mobil. Na QR kódu je uloženo Id vstupenky, které by mohl mobil číst pomocí fotoaparátu. Následně by se zavolala funkce, která by kontrolovala, zda Id patří k dané události. Podle této informace by se ověřovala platnost vstupenky.

Protože je blockchain veřejný, kdokoliv si může zjistit, které vstupenky patří dané události. Vzniká zde proto bezpečností riziko v podobě vytvoření falešné kopie vstupenky se stejnými metadaty. Toto riziko však zanikne, pokud budeme kontrolovat adresu kontraktu, které NFT vstupenka patří. Adresa kontraktu je uvedena u každé vstupenky v MetaMask peněžence. Další možností je zkontrolovat adresu peněženky a ověřit, zda vstupenka opravdu patří ukázané peněžence. Tyto opatření by šli obejít pouze v případě, že by útočník vytvořil naprosto stejnou kopii aplikace MetaMask peněženky, kde by si vytvořil falešnou peněženku s kopií vstupenky a vydával by se za pravého majitele této peněženky. Takový útočník by prakticky mohl jít na jakoukoliv akci s větším počtem účastníků, jelikož by bylo velmi obtížné na takový podvod přijít. V systému tak existuje bezpečnostní trhlina.

ZÁVĚR

Tato práce se zabývala návrhem a vývojem decentralizované aplikace na Ethereum blockchainu, která má vykonávat funkci systému pro online správu prodeje vstupenek. Práce má teoretickou a praktickou část.

Teoretická část je rozdělena na dvě kapitoly, v nichž byli vysvětleny technologie Blockchain a Ethereum, kterým je nutné porozumět pro pochopení praktické části, jelikož byla psána s vědomím, že čtenář již rozumí principům těchto technologií.

Praktická část obsahuje čtyři kapitoly. V první kapitole došlo k definici řešeného problému a návrhu jeho řešení. Ten je tvořen analýzou požadavků a modelem případů užití. Návrhem řešení došlo k vymezení hranic funkčnosti systému. V druhé kapitole se práce zabývá výhodami použití blockchainové technologie pro vyvíjenou aplikaci, kde je zmíněna i technologie NFT, která hraje důležitou roli v systému správy prodeje vstupenek. Třetí kapitola je nejrozsáhlejší kapitolou celé práce, jelikož zde dochází k popisu vývoje celé aplikace. Vývoj aplikace byl rozdělen do dvou částí. Nejprve došlo k popsání backendu systému v podobě smart kontraktu a jeho otestování, poté došlo k popsání frontendu v podobě webové aplikace. Ve čtvrté kapitole práce rozebírá problémy vyvinutého systému a pojednává o možném řešení těchto problémů.

Cílem, za kterým byla práce napsána, je ukázat možnost využití blokchainové technologie pro řešení konkrétního problému v podobě decentralizované aplikace s funkcí systému pro online správu prodeje vstupenek.

SEZNAM POUŽITÉ LITERATURY

[1] BASHIR, Imran. Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks [online]. Birmingham: Packt Publishing, 2017 [cit. 2023-04-18]. ISBN 9781787125445. Dostupné z: https://www.packtpub.com/product/mastering-blockchain/9781787125445

[2] QADEER, Saad. What are centralized, decentralized, and distributed systems? [online]. [cit. 2023-04-18]. Dostupné z: https://www.educative.io/answers/what-are-centralized-decentralized-and-distributed-systems

[3] ARCOS, L. C. The blockchain technology on the music industry. Brazilian Journal of Operations & Production Management [online]. 2018, (15(3)) [cit. 2023-04-18]. Dostupné z: doi:10.14488/BJOPM.2018.v15.n3.a11

[4] MELKER, Scott. Bitcoin and The Byzantine Generals Problem [online]. MARCH 19, 2021 [cit. 2023-04-18]. Dostupné z: https://www.thewolfofallstreets.io/bitcoin-and-the-byzantine-generals-problem/

[5] MOHANTY, Debajani. Ethereum for Architects and Developers [online]. Noida, Uttar Pradesh, India: Apress, 2018 [cit. 2023-04-18]. ISBN 978-1-4842-4075-5. Dostupné z: https://link.springer.com/book/10.1007/978-1-4842-4075-5

[6] M0HITKIRANGE. Blockchain Structure. GeeksforGeeks [online]. 16 Nov, 2022 [cit. 2023-04-19]. Dostupné z: https://www.geeksforgeeks.org/blockchain-structure/

[7] HOODA, Parikshit. Blockchain – Proof of Work (PoW). GeeksforGeeks [online]. [cit. 2023-04-20]. Dostupné z: https://www.geeksforgeeks.org/blockchain-proof-of-work-pow/

[8] PENNELLA, Luca. PROOF-OF-STAKE (POS). Ethereum.org [online]. March 1, 2023 [cit. 2023-04-20]. Dostupné z: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[9] SINGH, Onkar. Can blockchain be used without cryptocurrency? [online]. JUL 30, 2022 [cit. 2023-04-20]. Dostupné z: https://cointelegraph.com/explained/can-blockchain-be-used-without-cryptocurrency

[10] BITSTAMP. Do all cryptocurrencies use the blockchain? [online]. August 9th, 2022 [cit. 2023-04-20]. Dostupné z: https://www.bitstamp.net/learn/blockchain/do-all-cryptocurrencies-use-the-blockchain/

[11] ASHEFORD, Kate a Benjamin CURRY. What Is Cryptocurrency?. Forbes [online]. Feb 16, 2023 [cit. 2023-04-20]. Dostupné z: https://www.forbes.com/advisor/investing/cryptocurrency/what-is-cryptocurrency/

[12] COINBASE. What is a crypto wallet? [online]. [cit. 2023-04-21]. Dostupné z: https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-wallet

[13] HOODA, Parikshit. Blockchain Forks [online]. 03 Apr, 2023 [cit. 2023-04-21]. Dostupné z: https://www.geeksforgeeks.org/blockchain-forks/

[14] PETERSSON, David. How Smart Contracts Started And Where They Are Heading [online]. Oct 24, 2018 [cit. 2023-04-21]. Dostupné z: https://www.forbes.com/sites/davidpetersson/2018/10/24/how-smart-contracts-started-and-where-they-are-heading/?sh=474bbb2d37b6

[15] IBM. What are smart contracts on blockchain? [online]. [cit. 2023-04-21]. Dostupné z: https://www.ibm.com/topics/smart-contracts

[16] CAMPBELL, Christine. What are the 4 different types of blockchain technology? [online]. 03 Mar 2023 [cit. 2023-04-21]. Dostupné z: https://www.techtarget.com/searchcio/feature/What-are-the-4-different-types-of-blockchain-technology

[17] WHAT IS ETHEREUM?. Ethereum.org [online]. [cit. 2023-04-22]. Dostupné z: https://ethereum.org/en/what-is-ethereum/

[18] ANTONOPOULOS, Andreas M. a Dr. Gavin WOOD. Mastering Ethereum: Building Smart Contracts and DApps [online]. Sebastopol: O'Reilly Media, 2018 [cit. 2023-04-22]. ISBN 978-1-491-97194-9. Dostupné z: https://www.oreilly.com/library/view/mastering-ethereum/9781491971932/

[19] The history of Ethereum. Ethereum.org [online]. April 19, 2023 [cit. 2023-04-22]. Dostupné z: https://ethereum.org/en/history/

[20] DNYANESHWATDOD. Solidity – Ether Units. GeeksforGeeks [online]. 27 Feb, 2023 [cit. 2023-04-23]. Dostupné z: https://www.geeksforgeeks.org/solidity-ether-units/

[21] SMITH, Corwin. INTRO TO ETHER. Ethereum.org [online]. April 12, 2023 [cit. 2023-04-24]. Dostupné z: https://ethereum.org/en/developers/docs/intro-to-ether/

[22] How The Merge impacted ETH supply. Ethereum.org [online]. April 19, 2023 [cit. 2023-04-24]. Dostupné z: https://ethereum.org/en/roadmap/merge/issuance/

[23] WHAT IS ETHER (ETH)?: Currency for our digital future. Ethereum.org [online]. [cit. 2023-04-24]. Dostupné z: https://ethereum.org/en/eth/

[24] SMITH, Corwin. GAS AND FEES [online]. April 12, 2023 [cit. 2023-04-24]. Dostupné z: https://ethereum.org/en/developers/docs/gas/

[25] WACKEROW, Paul. TOKEN STANDARDS. Ethereum.org [online]. August 15, 2022 [cit. 2023-04-25]. Dostupné z: https://ethereum.org/en/developers/docs/standards/tokens/

[26] SMITH, Corwin. ETHEREUM ACCOUNTS [online]. February 9, 2023 [cit. 2023-04-25]. Dostupné z: https://ethereum.org/en/developers/docs/accounts/

[27] MINIMALSM. ETHEREUM VIRTUAL MACHINE (EVM) [online]. January 19, 2023 [cit. 2023-04-25]. Dostupné z: https://ethereum.org/en/developers/docs/evm/

[28] SMITH, Corwin. TRANSACTIONS. Ethereum.org [online]. April 7, 2023 [cit. 2023-04-25]. Dostupné z: https://ethereum.org/en/developers/docs/transactions/

[29] What Is a Blockchain Oracle?. Chainlink [online]. September 14, 2021 [cit. 2023-05-11]. Dostupné z: https://chain.link/education/blockchain-oracles

[30] SINGH, Jagjit. How to host a decentralized website [online]. MAR 11, 2023 [cit. 2023-05-11]. Dostupné z: https://cointelegraph.com/news/how-to-host-a-decentralized-website

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ETH	Ether
EVM	Ethereum Virtual Machine
HTTPS	Hypertext Transfer Protocol Secure
NFT	Non Fungible Token
IPFS	InterPlanetary File Systém
P2P	Peer to Peer
API	Application Programming Interface

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH DVD

fulltext.pdf – elektronická verze bakalářské práce

Složka TicketSystem – obsahuje zdrojové soubory aplikace