FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Autentizace, autorizace a evidence zařízení pomocí platformy Ethereum
Student:	Bc. Václav Chmel
Vedoucí:	Ing. MSc. Martin Jakl
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Vytvořte smart kontrakt (kontrakty) a navazující knihovnu umožňující autorizaci a autentizaci osob a zařízení pomocí platformy Ethereum. Jedná se jak o autentizaci osob tak o autentizaci M2M (machine-to-machine). Dále prozkoumejte možnosti platby za tyto služby.

Pokyny k vypracování práce:
- Podrobně se seznamte s fungováním platformy Ethereum a fungováním Smart Contracts a popište je.
- Prozkoumejte vhodnost využít blockchain (Ethereum) jako technologie pro autentizaci osob či zařízení (M2M). Využití základních crypto funkcí, případně i Smart Contracts.
- Prozkoumejte a popište evidenci osob/zařízení pomocí standardu ERC Identity.
- Prozkoumejte možnosti platby za tyto služby pomocí ERC20 Coins.
- Proveďte analýzu, návrh, implementaci a testování smart kontraktů a knihoven, které je budou zapouzdřovat.
- Zdokumentujte výstupy práce.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 20. dubna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Autentizace, autorizace a evidence zařízení pomocí platformy Ethereum

Bc. Václav Chmel

Katedra softwarového inženýrství
Vedoucí práce: Ing. MSc. Martin Jakl

7. května 2019

Poděkování

Děkuji svému vedoucímu práce panu Martinovi Jaklovi, že mi umožnil podílet se v rámci mé diplomové práce na takto zajímavém tématu a za trpělivost při vysvětlování problematiky.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. května 2019 .....

Odkaz na tuto práci

Chmel, Václav. *Autentizace, autorizace a evidence zařízení pomocí platformy Ethereum*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce představuje technologii blockchain a platformu Ethereum. Dále se věnuje analýze, návrhu, implementaci a testování prototypu mobilní aplikace AEW. Tato aplikace umožňuje autentizaci uživatelů pomocí této platformy a její využití pro mobilní aplikace OS Android.

Klíčová slova Ethereum, autentizace, peněženka, decentralizované aplikace, blockchain, podepisování transakcí, Android

Abstract

This thesis introduces the blockchain technology and the Ethereum platform. It goes over analysis, design, implementation and testing of AEW mobile application prototype which allows authentication of users via Ethereum platform. Furthermore it allows Ethereum platfrom to be used in Android applications.

Keywords Ethereum, authentication, wallet, decentralised applications, blockchain, transaction signing, Android

Obsah

Seznam obrázků

Úvod

V posledních letech dochází k rozvíjení technologie blockchain. Tato technolgie je pro většinu lidí známa pouze s jejím spojením vzniku kryptoměn a vidiny možné finanční investice za účelem zhodnocení zisku. Tito lidé pak většinou nevidí, že některé kryptoměny jako třeba Ethereum, jsou pouze vedlejším produktem technologie, která přináší nové možnosti.

V současné době externě pracuji pro firmu MobitelcoLTD, kde jsme se začali zabývat platformou Ethereum, učit se, zkoušet a zkoumat možnosti jejího využití v rámci mobilních zařízení s OS Android.

Platforma Ethereum umožňuje vývoj decentralizovaných aplikací. Použití takovýchto aplikací má jeden společný prvek, a to nutnost správy uživatelských účtů použitých na této platformě. V současné době již existují komponenty, které toto umožňují. Jedná se o speciální typ peněženek na Ethereum jimiž jsou chytrý prohlížeč Mist nebo rozšíření do prohlížeče Chrome s názvem Metamask. Tyto prvky pak umožňují užití decentralizovaných aplikací v rámci daného prohlížeče.

Bohužel v současné době neexistuje aplikace, která by takovouto funkcionalitu umožnila na mobilních zařízeních pro aplikace operačního systému Android. Nejblíže k tomu má aplikace Trust Wallet, která poskytuje vestavěný webový prohlížeč poskytující tuto funkcionalitu. Jmenované řešení není dostačující, protože umožňuje použít pouze webové aplikace a nikoliv aplikace určené pro OS Android.

Ukázalo se, že pouhá knihovna by pro naše potřeby nebyla dostačující. Vznikl proto první prototyp mobilní aplikace. Oproti původně zamýšlené knihovně má rozšířenou funkcionalitu o správu Ethereum účtů, která je důležitá pro další použití.

Dále se ukázalo, že daná problematika je obsáhlá, neustále se rozvíjí a každé následující další implementační kroky přesahují rozsah této diplomové práce. Z toho důvodu se práce zaměřuje hlavně na autentizaci uživatele pomocí této platformy.

Jedná se pouze o první část, která je nezbytná pro jakékoliv úvahy o složitější autentizaci pomocí smart kontraktů a používání platformy Ethereum pro mobilní aplikace. Bylo důležité, aby výstup práce měl praktické použití.

Hmatatelným výstupem této práce je prototyp aplikace AndroidEtherumWallet - AEW. Tento prototyp má využití jak pro uživatele, kterým poskytuje funkce peněženky na Ethereum a umožňuje se prokazovat Ethereum účtem i v jiných aplikacích, tak pro vývojáře, kterým slouží jako komponent pro správu Ethereum účtů, o kterou se pak nemusí starat ve vlastních aplikacích.

Cíle této práce jsou následující:

- seznámit čtenáře s fungováním technologie blockchain a fungováním platformy Ethereum
- seznámit čtenáře se standardy ERC725(Identity) a ERC20 (Tokens)
- prozkoumat vhodnost platformy Ethereum pro autentizaci a autorizaci uživatelů a zařízení
- provést analýzu, návrh, implementaci a testování prototypu aplikace AEW
- zdokumentovat výstupy práce.

Kapitola 1

Blockchain

1.1 Technologie Blockchain

1.1.1 Co je to blockchain?

Lze najít nejrůznější definice toho, jak si kdo představuje blokchain a co to vlastně je. Uvedu teď proto několik definic z různých zdrojů a pak se pokusím dát dohromady společné prvky.

[1] uvádí blokchain takovýmto způsobem cituji: "A blockchain is, in the simplest of terms, a time-stamped series of immutable record of data that is managed by cluster of computers not owned by any single entity. Each of these blocks of data (i.e. block) are secured and bound to each other using cryptographic principles (i.e. chain)."

[2] uvádí blockchain takovýmto způsobem cituji: "Blockchain is a type of distributed ledger for maintaining a permanent and tamper-proof record of transactional data. A blockchain functions as a decentralized database that is managed by computers belonging to a peer-to-peer (P2P) network. Each of the computers in the distributed network maintains a copy of the ledger to prevent a single point of failure (SPOF) and all copies are updated and validated simultaneously."

Dále podle tohoto zdroje má blockchain následující vlastnosti: data jsou skládána do bloku. Bloky jsou opatřeny časovou značkou a hashem, který se počítá z této časové značky a hashe předchozího bloku. Než je blok přidán do sítě musí projít procesem validace, během tohoto procesu se většina uzlů musí shodnout na tom, že byl hash nového bloku spočítán korektně. Jakmile je blok přidán do sítě, nemůže být změněn, pouze na něj může být odkazováno ostatními bloky.

Podle [3] se dá blockchain shrnout takovýmto způsobem: Blockchain je veřejný deník záznamů získaných přes síť fungující přes internet. Jedná se tedy o strukturu pro ukládání informací, kde informace jsou ukládány do bloků, které jsou navazovány za sebe a má čtyři následující vlastnosti: Uložené infor-

mace jsou opatřené časovou značkou. Uložené informace v blocích nejde změnit beze změny všech následujících bloků. Toto nelze provést bez povšimnutí ostatních účastníků sítě. Blokchain je veřejný tj. každý může prohlížet záznamy transakcí, a dále je decentralizovaný - kopie deníku informací existuje na různých počítačích.

Podle [4] cituji "It is a shared, decentralized, and open ledger of transactions. This ledger database is replicated across a large number of nodes."

Podle tohoto zdroje jsou informace opět ukládány do bloků a bloky jsou spojovány dohromady. Data do blokchainu lze přidávat, ale již zapsaná data nelze měnit.

Uvedu ještě jednu definici podle [5]. Cituji: "A blockchain is a tamper-evident, shared digital ledger that records transactions in a public or private peer-to-peer network. Distributed to all member nodes in the network, the ledger permanently records, in a sequential chain of cryptographic hash-linked blocks, the history of asset exchanges that take place between the peers in the network."

Na základě těchto definic a zdrojů lze tedy blokchain definovat jako techologii sloužící pro ukládání dat, která má následující vlastnosti:

- Je distribuovaná a decentralizovaná.
- Data uložená v blockchainu jsou zpravidla veřejná. Přestože většina výše citovaných zdrojů uvádí, že jsou data veřejná, přiklonil bych se v tomto k definici podle [5]. Blockchain může být realizován na privátní síti a pak data veřejná být nemusí.
- Data jsou uložena pomocí transakcí, které jsou skládány do bloků. Bloky jsou opatřeny časovou značkou.

1.1.2 Fungování blockchainu

1.1.2.1 Distribuovanost a decentralizovanost

Data v blokchainu jsou rozprostřena přes zařízení, která v něm participují. Tato zařízení nazýváme uzly. Uzly jsou si rovnocenné z hlediska připojení a komunikují mezi sebou. Výpadek jednoho uzlu neznamená výpadek celé sítě.

V praxi je toto implementováno jako peer-to-peer síť přes internet. Jakým způsobem jsou data na participujících zařízeních rozložena (zda má každý uzel úplná data, nebo jejich části atd.) pak záleží na konkrétní implementaci této technologie.

1.1.2.2 Zápis dat

Data jsou do blokchainu zapisována pomocí transakcí. Transakce definuje původce, data a příjemce transakce. To jakým způsobem tato transakce vy-

padá, jak je definován původce, příjemce a zapisovaná data, záleží na konkrétní implementaci.

Transakce jsou dále sdružovány do bloků. Bloky jsou navazovány za sebe a nové bloky jsou přidávány vždy nakonec.

Blok kromě transakcí obsahuje i hash informací obsažených v předchozím bloku (a ten zas hash informací v dalším předchozím bloku atd.). Bloky jsou takto propojeny do jednoho celku. Důsledkem toho je, že pokud by někdo chtěl přepsat informace v bloku B, změna by se promítla i ve všech následujících blocích. Aby nová informace byla považována za platnou, musely by být přepsány všechny tyto bloky. [6]

1.1.2.3 Mechanismus konsenzu

Mechanismus konsenzu je protokol, který zajišťuje, že uzly v rámci blockchainu jsou synchronizované a dohodnou se, které transakce jsou legitimní. Toto je potřeba z hlediska bezpečnosti, protože transakce do blockchainu zasílá velký počet uzlů a je nutné zajistit, aby do blockchainu nebyly přidány transakce, jež jsou nelegitimní.

Správné fungování tohoto protokolu je velice důležité pro správný chod sítě. V čem se ovšem různé implementace technologie blockchain mohou lišit je právě zvolený mechanismus konsenzu. [7]

Proof of Work Proof of work přidává ke zpracování bloků umělou výpočetní zátěž. V případě, že uzly narazí na dva konfliktní stavy sítě, berou jako platný ten stav, který ke vzniku potřeboval větší výpočetní výkon. Tato zátěž je generována pomocí procesu těžení. Jedná se o proces, kdy uzly, které chtějí přidat nový blok do sítě, musí najít číslo, které po přidaní k informacím, vytvoří hash bloku splňující určité vlastnosti. Toto musí uzly provádět hrubou silou a obětovat k tomu svůj výpočetní výkon. Verifikace platnosti hashe daného bloku ovšem vyžaduje pouze konstantní čas. Důsledkem podstoupení tohoto procesu při vytvoření každého bloku je, že transakce v blocích určitého stáří, by ke své změně vyžadovaly příliš velký výpočetní výkon (vzhledem k vlastnostem blockchainu) než aby se toto vyplatilo. Tímto je zabezpečena jejich neměnnost. [8]

Tento mechanismus je velmi náročný na výpočetní výkon a tím spotřebu elektrické energie, má negativní vliv na rychlost zpracování transakcí a tím na škálování použitelnosti sítě. [9] Je tedy snaha najít mechanismus, který tato omezení mít nebude. Různé alternativy k tomuto mechanismu již existují. Jejich fungování není natolik prověřené, aby jim zajistilo obecné použití.

Tento mechanismus používají dvě nejznámější implementace technologie blockchain: Bitcoin a Ethereum.

1.1.3 Výhody technologie blockchain

- Decentralizace eliminuje potřebu důvěry v centrální autoritu (např. stát, banku, firmu, které svěřujeme data apod.).
- Distribuovanost - výpadek uzlu nezpůsobí výpadek celku, maximálně dojde k lokálnímu ochromení.
- Veřejně přístupná data - toto může být bráno jako výhoda i nevýhoda, zaleží na kontextu a způsobu použití.
- Blockchainové technologie obsahují vestavné platební metody.
- Data jsou dobře zabezpečena, jakmile se jednou do blockchainu dostanou nelze změnit a smazat.

1.1.4 Nevýhody technologie blockchain

- Technologie není stavěná na ukládání velkého objemu dat, protože data jsou replikována napříč sítí.
- Zpracování transakcí oproti centralizovanému řešení je mnohem pomalejší. [9]
- Každé ukládání/změna stavu blockchainu je zpoplatněna.

1.2 Smart kontrakt

V informačních zdrojích se lze potkat s obecnou definicí smart kontraktu. Vybral jsem si definici podle [10]. Cituji: "A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs."

Příkladem takového smart kontraktu je třeba zakoupení a užívání domény v systému ENS popsaném níže. Za úplatu lze zakoupit doménu použitelnou na platformě Ethereum a tu používat místo Ethereum adresy. Po ukončení platnosti toto využití přestane automaticky fungovat. Jedná se o kontrakt mezi provozovateli ENS a kupcem domény. Podmínky této smlouvy a možné interakce jsou zakódovány ve skupině smart kontraktů, která má za úkol fungování cílové služby.

Dalším příkladem z [11] je use case crowdfundingového smart kontraktu. Tento kontrakt má nastaven cílový objem peněz a deadline pro jejich výběr. Přispěvatelé mohou do tohoto kontrakt ukládat prostředky jimiž chtějí kampaň

podpořit. V případě neúspěšné kampaně jsou pak prostředky automaticky vráceny jejich vlastníkům. V případě úspěšné kampaně jsou odeslány jejímu pořadateli.

Dalším příkladem je zakoupení elektronického předplatného. Cílová osoba si zakoupí digitální předplatné prostřednictvím určeného smart kontraktu. V podmínkách tohoto kontraktu se nachází doba předplatného. Cílová služba po autentizaci uživatele pomocí Ethereum účtu, může zkontrolovat, zda je daný uživatel autorizován k jejímu použití (vlastník předplatného).

Smart kontrakt má tyto důležité vlastnosti podle [12].

Obecně platí, že kód kontraktu je po jeho nasazení neměnný. Na platformě Ethereum existuje postup, který umožní upgradování smart kontraktů. Toto neznamená změnu kódu daných kontraktů, ale použití speciálního návrhového vzoru, kdy upgradem se rozumí nasazení nové implementace požadovaného kontraktu. Díky tomu podmínka neměnosti jejich kódu zůstává neporušena.

Logika kontraktu se provádí automaticky na základě vstupů a podmínek, které jsou naprogramovány a tím kontrakt automaticky zastřešuje svoje plnění.

Důsledkem užití smart kontraktu je šetření finančních prostředků, času a eliminace nutnosti třetích stran.

1.3 Ethereum Blockchain

Ethereum je otevřená softwarová platforma, která umožňuje vývoj a nasazení decentralizovaných aplikací. Tato platforma je založena na technologii blockchain. [13] Hlavním platidlem na této platformě je kryptoměna nazývaná Ether. Často se také lze setkat s denominací Etheru nazývanou Wei. 1 Wei = $1*10^{18}$ Etheru. Tuto platformu budu dále nazývat Ethereum nebo Ethereum blockchain.

1.3.1 Účty v Ethereum blokchainu

Účty jsou v Ethereum blokchainu identifikovány 20 bytovým identifikátorem zvaným Ethereum adresa. Ethereum adresa bývá reprezentována jako hexadecimální řetězec. Příkladem takovéto adresy je tedy:

```
0xB657FECB62Ae2CC3476241E161847FF315bf89f7
```

V ethereum blokchainu jsou rozlišené dva typy účtů:

- externí účet
- účet smartkontraktu

Externí účet je v Etherum blokchainu spjatý s privátním klíčem. Ethereum používá v rámci asymetrické kryptografie eliptickou křivku secp256k1. [14] V praxi to znamená, že privátní klíč k Ethereum účtu je dlouhý 32 bytů a často

se reprezentuje jako hexadecimální řetězec. Externí účet může zasílat transakce na zpracování do blockchainu. Tyto transakce podepisuje svým privátním klíčem. Dále obsahuje vlastněný Ether a hodnotu nonce, což je čítadlo, kolik transakcí daný účet celkem zaslal do sítě. Externí účet neobsahuje žádný kód.

Účet smart kontraktu je vytvořen při nasazení nové instance smart kontraktu do blockchainu. Tento účet má také svůj zůstatek, což umožňuje zasílání Etheru do smart kontraktu. Tento účet také obsahuje parametr nonce. Narozdíl od účtu zde parametr značí kolik ostatních smart kontraktů daný smart kontrakt vytvořil. Účet dále obsahuje uložená data a kód daného smart kontraktu, který je tímto kódem kontrolován a nemůže tedy samovolně vytvářet nové transakce jako je tomu v případě externího účtu. Může vytvářet transkace pouze jako reakci na volání svých funkcí. [15]

Důsledkem toho je, že počáteční transakce musí být iniciovány externímy účty a platforma jako taková nemá zabudovanou funcionalitu rozvrhování transakcí. Naivní řešení by v tomto případě spočívalo v serveru, který by dané transakce spouštěl. Takový server by nesl všechny náklady s tím spojené. Toto řeší služba Ethereum Alarm Clock (viz dále).

1.3.2 Transakce v Ethereum blokchainu

Změny stavu se v Ethreum blokchainu dějí pomocí transkací. Transakce podle [16] obsahuje následující položky:

- nonce - jedná se o celkové pořadí transakce odeslané z daného externího účtu
- to - adresa účtu příjemce (externí účet, účet smart kontraktu, nebo prázdná hodnota v případě nasazení nového kontraktu)
- gasPrice - cena v jednotkách wei, kterou je ochoten původce zaplatit za jednotku Gas (viz dále)
- gasLimit - maximální povolený počet čerpaného Gas pro tuto transakci (viz dále)
- value - přenášená hodnota Etheru v denominaci Wei
- data - v případě, že se jedná o přenos Etheru, tato data jsou prázdná. V případě, že se jedná o volání funkce smart kontraktu, obsahuje tato položka zakódovaný název funkce spolu s parametry jejího volání. V poslední řadě jedná-li se o vytvoření nového smart kontraktu, obsahuje položka k tomu nutná data (bytekód a zakódované parametry konstruktoru).

Aby takováto transakce mohla být přijata do blockchainu, musí být nejprve podepsána privátním klíčem externího Ethereum účtu. Takto podepsaná transakce pak jednoznačně identifikuje svého původce.

V rámci podepisování transakcí existují dvě možnosti. Transakci lze podepsat účtem, který se nachází na Etherum uzlu, jenž je pod správou původce transakce, nebo lze transakci podepsat lokálně a odeslat na některý veřejný Ethereum uzel.

1.3.3 Mechanismus plateb za transakce

Jak si lze všimnout v předchozí části, transakce obsahuje dvě položky, které nebyly v rámci předchozího textu dostatečně vysvětleny. Jedná se o položky gasLimit a gasPrice. Veškeré provádění transakcí je v Ethereum blockchainu zpoplatněno. Míra výpočetního výkonu a paměťová náročnost potřebná ke zpracování transakce se měří pomocí jednotky Gas.

Původce transakce pak určuje maximální limit Gas, který transakce může spotřebovat a také množství Etheru (v jednotkách Wei), který je ochoten za každou spotřebovanou jednotku Gas zaplatit. Celkový poplatek za zpracování transakce je určen jako gasPrice * gasLimit.

V případě, že je Gas spotřebovaný transakcí menší než hodnota gasLimit, transakce proběhne správně a původci transakce je vrácen polatek za nespotřebovaný Gas. V případě, že transakce překročí svůj gasLimit, transakce selže, změny způsobené touto transakcí jsou vráceny zpět, ale poplatek za zpracovaný Gas propadá. Poplatky za zpracování transakcí jsou přičteny uzlu, který transakci zpracuje (vytěží).

Hodnota gasPrice ovlivňuje rychlost zpracování transakce v Ethereum blockchainu. [17]

Účel takovéhoto modelu poplatků za zpracování transakcí je dvojí. Zaprvé slouží jako pobídka, aby se uzly podílely na zpracování transakcí. Zadruhé slouží jako pojistka, aby transakce měla omezenou dobu běhu. [18]

1.3.4 Smart kontrakty na platformě Ethereum

V předchozí části jsem se snažil popsat smart kontrakt obecně. Pojmem smart kontrakt se na platformě Ethereum označuje jakýkoliv kus počítačového kódu, který má za úkol plnit aplikační logiku a běží na této platformě (přestože nemusí splňovat obecnou definici smart kontraktu).

Aby se dalo interagovat s daným smart kontraktem, je potřeba znát dvě věci: jeho Ethereum adresu a jeho rozhraní. (Často se s tímto rozhraním lze setkat ve formátu JSON. Obsahuje podrobné informace o funkcích daného smart kontraktu potřebných pro překlad do strojového kódu).

1.3.4.1 Solidity

Pro vytváření smart kontraktů na platformě Ethereum slouží programovací jazyk Solidity. Je to objektově orientovaný staticky typovaný jazyk. Pro úvod do tohoto jazyka uvádím příklad smart kontraktu v něm napsaný (obrázek

```
contract TestIdentity is ERC725 {

  uint256 constant MANAGEMENT_KEY = 1;
  uint256 constant ACTION_KEY = 2;
  uint256 constant CLAIM_SIGNER_KEY = 3;
  uint256 constant ENCRYPTION_KEY = 4;

  struct Key {=
  }

  struct Execution {=
  }

  mapping (bytes32 => Key) key_table_values;
  bytes32[] key_table_keys;

  bytes32 keyDef;
  uint256[] ppDef;

  mapping (uint => Execution) executions;
  uint executionId;

  constructor() public {=
  }

  modifier ownerOrSelf() {=
  }

  function getKey(bytes32 _key) public constant returns(uint256[] purposes, uint256 keyType, bytes32 key) {=
  }

  function keyHasPurpose(bytes32 _key, uint256 purpose) public constant returns(bool exists) {=
  }

  function getKeysByPurposeCount(uint256 _purpose) private constant returns(uint256 count) {=
  }

  function getKeysByPurpose(uint256 _purpose) public constant returns(bytes32[] keys){=
  }

  function addKey(bytes32 _key, uint256 _purpose, uint256 _keyType) public ownerOrSelf returns (bool success) {
    key_table_keys.push(_key);
    key_table_values[_key].key = _key;
    key_table_values[_key].purposes.push(_purpose);
    key_table_values[_key].keyType = _keyType;

    emit KeyAdded(_key, _purpose, _keyType);

    return true;
  }

  function removeKey(bytes32 _key, uint256 _purpose) public ownerOrSelf returns (bool success) {=
  }

  function execute(address _to, uint256 _value, bytes _data) public returns (uint256 res) {=
  }


  function approve(uint256 _id, bool _approved) public ownerOrSelf returns (bool result) {=
  }

  function toBytes(address a) private pure returns (bytes32 b){=
  }

  function bytesToBytes32(bytes b, uint offset) private pure returns (bytes32) {=
  }

}
```

Obrázek 1.1: Příklad smart kontraktu v jazyce Solidity

1.1). Tento příklad lze vidět na obrázku 1.1. Jedná se o mojí pokusnou implementaci standardu ERC 734 (dříve ERC725) popsanou níže.

Jak je vidět na obrázku, kontrakt je v podstatě třída v objektově orientovaném programování. Třídní proměnné se nazývají storage proměnné a zachycují aktuální stav kontraktu v době jeho životnosti. Statické proměnné zde neexistují, protože se vždy interaguje se smart kontraktem na konkrétní adrese. Storage proměnné jsou definovány mezi strukturou Executions a konstruktorem.

Dále následuje konstruktor, který je volán při nasazení kontraktu. Poté následuje funkce označená jako modifier. Jedná se o specifický typ funkce v jazyce Solidity, jímž lze odekorovat ostatní funkce. Modifier určuje podmínku, kterou lze zkontrolovat na určitém místě během provádění funkce (zpravidla na jejím úplném začátku). Tímto lze omezovat použití funkcí pro konkrétní adresy, jejich skupiny nebo pomocí jiných podmínek (čas bloku aj.). Toto se hodí z hlediska autorizace v distribuovaných aplikacích.

Dále následují jednotlivé funkce. Tyto funkce se dělí v zásadě na dvě kategorie. Funkce, které nemodifikují stav kontraktu a jsou označené jako constant, view, nebo pure. Takovéto funkce lze volat bez použití transakcí a mohou vracet hodnoty. Funkce, které takto označené nejsou, lze volat pouze pomocí odeslání transakce do blockchainu.

Přestože takto označené funkce mohou mít návratovou hodnotu, slouží tato hodnota pouze pokud by tento kontrakt, volal jiný kontrakt. Jinak se musí hodnota vrátit pomocí vyvolání události. Tato událost se uloží do stejného bloku jako transakce, která ji vyvolala.

Funkce kontraktu mohou obsahovat modifikátor payable umožňující danému kontraktu obdržet Ether během volaní funkce s tímto modifikátorem. Ten se přičte k celkovému zůstatku kontraktu. Pokud je potřeba držet bilanci, kolik která adresa přispěla do daného kontraktu, musí to zajistit jeho vývojář.

Kontrakt dále může obsahovat funkci, které se říká defaultní funkce. Tato funkce nenese žádný název a je spojována s modifikátorem payable. Ta umožňuje zasílání Etheru do daného kontraktu ve stejné formě, jako by cílem nebyl kontrakt, ale externí účet.

Stejně tak je povinností vývojáře zajistit, aby daný kontrakt obsahoval funkci zajišťující výběr Etheru z daného kontraktu. Jinak se může stát, že Ether obsažený v daném kontraktu již nepůjde získat zpět.

1.3.4.2 Vyper

Solidity není jediným jazykem, který lze využít pro vytváření smart kontraktů. Dalším jazykem je Vyper. Tento jazyk se zaměřuje na tvorbu smart kontraktů z hlediska bezpečnosti, jednoduchosti a schopnosti kontrakty v tomto jazyce lépe auditovat. Z těchto důvodů neposkytuje tak širokou funkcionalitu jako jazyk Solidity. Je to za cenu zvýšené bezpečnosti pro vytvořené smart kontrakty v tomto jazyce. [19]

```
// ----------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public view returns (uint);
    function balanceOf(address tokenOwner) public view returns (uint balance);
    function allowance(address tokenOwner, address spender) public view returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Obrázek 1.2: Rozhraní definované standardem ERC20 [21]

```
    string public constant name = "Token Name";
    string public constant symbol = "SYM";
    uint8 public constant decimals = 18;  // 18 is the most common number of decimal places
```

Obrázek 1.3: Členy ERC20 [21]

S tímto jazykem na rozdíl od Solidity nemám osobní zkušenost.

1.4 ERC20

ERC20 je technický standard, který slouží k implementaci smart kontraktů jež reprezentují obchodovatelné tokeny na síti Ethereum. [20] V praxi se jedná o rozhraní v jazyce Solidity, které tyto obchodovatelné tokeny musí dodržet. Výhoda tohoto standardu plyne hlavně pro peněženky, jež mají podporu těchto tokenů, protože umožňuje spravovat přes stejné rozhraní mnoho různých tokenů. Jediné co je k tomu potřeba je Ethereum adresa daného tokenu. Rozhraní lze vidět na obrázku 1.2

Rozhraní umožňuje získat celkové množství tokenů, zjistit zůstatek daného účtu, zjistit zůstatek s kterým danému účtu umožnil disponovat jiný účet. Dále umožňuje zaslání tokenu, povolení jinému účtu disponovat a obchodovat s částí vlastních tokenů a přenos tokenu z cizího účtu, za předpokladu, že je tam účet, který chce využít této funkcionality, disponentem. Standard dále obsahuje 3 veřejné členy obr 1.3, které jsou nepovinné, ale tokeny je často implementují. Těmi jsou jméno tokenu, značka tokenu a počet desetinných míst tokenu. [22]

Co se týče použitelnosti ERC20 za platby služeb, existují čtyři možnosti:

1. použít již existující token,
2. implementovat vlastní token,
3. vydat vlastní token na základě již existující implementace,

```
pragma solidity ^0.5.4;

interface ERC725 {
    event DataChanged(bytes32 indexed key, bytes32 indexed value);
    event OwnerChanged(address indexed ownerAddress);
    event ContractCreated(address indexed contractAddress);

    // address public owner;

    function changeOwner(address _owner) external;
    function getData(bytes32 _key) external view returns (bytes32 _value);
    function setData(bytes32 _key, bytes32 _value) external;
    function execute(uint256 _operationType, address _to, uint256 _value, bytes calldata _data) external;
}
```

Obrázek 1.4: Rozhraní definované standardem ERC725 [23]

4. token vůbec nepoužít a použít pouze Ethereum, které tyto tokeny kryje

Která z následujících možností je správná nelze v tuto chvíli diskutovat. Tokeny slouží k odstínění uživatele od kolísavého kurzu Etheru a umožňují vyrovnanější ceny poplatků za služby.

Obecně ale vhodnost použití tokenu nelze posoudit nýbrž závisí na konkrétní situaci.

Aplikace AEW v prvotní fázi podporu pro manipulaci a správu ERC20 tokenů mít nebude. Do budoucna se s ní však počítá.

1.5 ERC Identity

Standard s označením ERC725 přichází s návrhem rozhraní smart kontraktu, který má sloužit jako proxy identita pro osoby, zařízení či společnosti na platformě Ethereum. Tento standard má dvě funkce. Slouží jako proxy kontrakt na Ethereum síti a jako úložiště dat ve formátu klíč-hodnota definovaných podle tohoto standardu. Standard je ve fázi návrhu. V praxi se jedná o Solidity rozhraní stejně jako u standardu ERC 20. Rozhraní lze vidět na obrázku 1.4

Vlastníkem tohoto kontraktu může být externí účet, kontrakt samotný, nebo jiný smart kontrakt. Pomocí funkce execute pak lze volat jakoukoliv jinou funkci libovolného smart kontraktu. Její parametry jsou následující:

1. _operationType - určuje jestli se jedná o operaci call, nebo operaci create. První operace slouží k volání funkce druhá k založení nového smart kontraktu

2. _to - cílová Ethereum adresa

3. _data - při operaci call se jedná o kódovaná data označující název cílové funkce smart kontraktu a její argumenty, při operaci create půjde o byte data nového smart kontraktu

Od chvíle, kdy jsem se začal Etherem zabývat tak se standard a z něj plynoucí rozhraní velice změnilo. Původně byl pod názvem ERC725 představen standard, který se dnes nachází pod zkratkou ERC734 a názvem KeyManager.

Tento standard a z něj plynoucí rozhraní, má podobnou funkci tj. slouží jako proxy identita s tím rozdílem, že definuje navíc typy veřejných klíčů, myšlenka tohoto standardu je taková, že uživatel má různé klíče s kterými provádí různé akce. Má klíč pro správu své identity, klíč pro běžné užívání atd. Standard zatím definuje pouze dva typy těchto klíčů - klíč pro správu identity a klíč pro použití funkce execute, která z hlediska tohoto standardu může vyžadovat schválení pomocí příslušných klíčů a teprve poté je provedena. [24]

Důvodem vzniku takovýchto standardů je opět stejný důvod jako při vzniku standardu erc20 tj. ulehčit integraci takovýchto prvků do aplikací, které by je chtěly použít. Přesněji řečeno, aby ta samá identita definovaná tímto standardem se dala použít v různých klientských aplikacích.

Jelikož standard není konečný, stále se vyvíjí a připomínkuje, rozhodně nebude zahrnut v aplikaci AEW. Tato varianta začne být diskutovatelná až v době, kdy bude tento standard dotažený a začne se obecně používat.

1.6 Vhodnost platformy Ethereum k autentizaci a autorizaci uživatelů a zařízení

Z hlediska platformy Ethereum není žádný rozdíl mezi fyzickým uživatelem a zařízením. Oba jsou identifikování stejně pomocí Ethereum adresy. Může se jednat buď o adresu externího účtu, nebo adresu smart kontraktu viz níže.

Rozdíl mezi nimi je takový, že aplikace, kterou bude ke své autentizaci využívat uživatel bude mít jinou formu než aplikace, kterou ke své autentizaci bude užívat zařízení.

1.6.1 Autentizace

Nejjednoduším způsobem jakým lze využít platformu Ethereum k autentizaci uživatele, nebo zařízení je využití jejího vestavěného mechanismu externích účtů.

V tomto případě musíme během autentizace dokázat, že je daný subjekt disponentem privátního klíče k danému účtu. Tento proces má následující podobu:

1. Ten, kdo požaduje autentizaci, vygeneruje náhodná data a předá je autentizovanému subjektu.

2. Autentizovaný subjekt tato data podepíše pomocí elektronického podpisu a podpis předá zpátky původci dat.

3. Ten ověří platnost elektronického podpisu. Pokud je podpis platný, autentizovaný subjekt je vlastníkem privátního klíče a autentizace byla úspěšná. V opačném případě nikoliv.

V pokročilejším případě má tato síť předpoklady na autentizaci uživatele nebo zařízení pomocí smart kontraktů. Z tohoto důvodu vznikají standardy jako ERC725 a ERC734, které ještě nejsou dotažené a rozšířené na obecné použití. Z tohoto důvodu jsme se zatím rozhodli cestou využití vestavěného mechanismu externích Ethereum účtů.

Aby takováto autentizace byla možná a přenositelná do ostatních aplikací, jak obyčejných tak decentralizovaných, musela pro mobilní zařízení nejprve vzniknout aplikace, která bude podporovat správu externích Ethereum účtů a tuto jednoduchou autentizaci bude podporovat pro uživatele.

1.6.2 Autorizace

Platforma Ethereum je také vhodná k autorizaci uživatelů i zařízení. Na úrovni smart kontraktů lze přímo nadefinovat, které adresy či jejich skupiny a za jakých podmínek budou moci používat určité funkce daného smart kontraktu. Tímto lze autorizovat externí účty i smart kontrakty k využití určité funkcionality. V tomto případě je vhodné mít cílené řešení autorizace na úrovni konkrétní aplikace.

1.7 Služby na platformě Ethereum

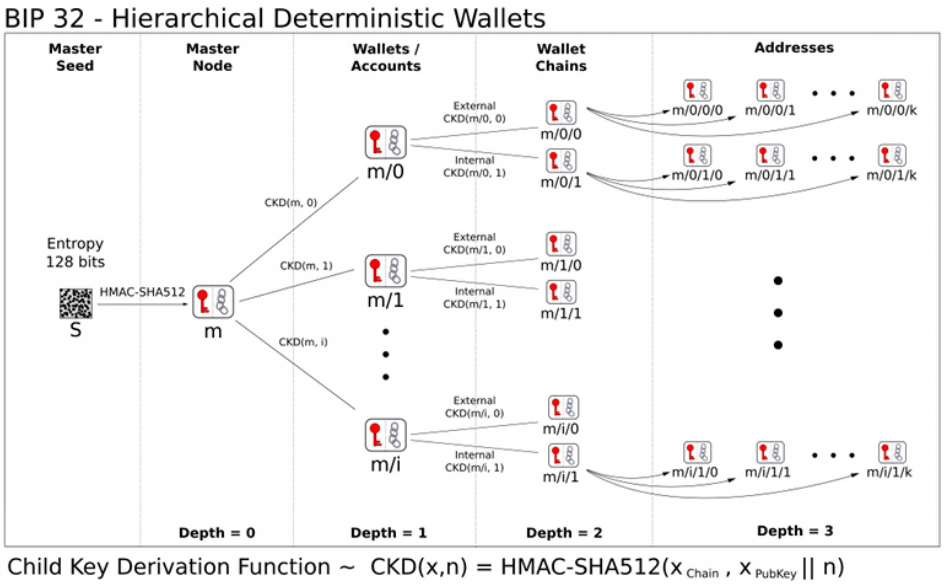
1.7.1 Ethereum Name Service

Ethereum Name Service je služba, jejíž účelem je mapování normálně čitelných jmen na strojové identifikátory a obráceně. Nemusí se jednat pouze o Ethereum adresy.

Tato služba funguje obdobně jako DNS tj. jedná se o domény s tečkovou notací. Hlavní doména na hlavní síti je ".eth". Tato služba funguje pomocí Ethereum blockchainu. Jedná se o sadu smart kontraktů a navazující knihovny. ENS se skládá z těchto komponent: registrů, registrářů a resolverů. Tyto komponety jsou tvořené smart kontrakty.

Registry mají za úkol pro každou doménu držet adresu vlastníka, adresu resolveru a hodnotu TTL. Úkol resolverů je překlad domén na identifikátory a opačně. Registráře umožňují registrovat nové domény do systému. [25]

V minulosti tato registrace probíhala pomocí aukce. Nyní již bude probíhat zakoupením domény. Cena se bude odvíjet od její délky. Přechod na tento systém je naplánován na 4.5.2019. [26]



Obrázek 1.5: Diagram struktury HD Wallet [28]

1.7.2 Ethereum Alarm Clock

Ethereum Alarm Clock je služba umožňující objednávku Ethereum transakcí, tak aby se provedly v budoucnu. Jedná se o sadu smart kontraktů a navazující aplikaci. Ten, kdo si chce transakci objednat, musí předem zaplatit stanovenou částku. V této částce jsou zahrnuté náklady na transakci a odměna pro toho, kdo transakci provede. Teoreticky je možné, že transakce nebude provedena vůbec, protože se spoléhá na své provedení třetí stranou. Služba se snaží zajistit, aby tomu tak nebylo. [27]

Toto je zejména důležité v případě užití služby na soukromém blockchainu.

1.8 HD Wallet a proces generování Ethereum účtů

HD Wallet je stromová struktura (obrázek 1.5) definovaná standardem BIP32 [28]. Uzly v tomto stromě obsahují kryptografické klíče. Kromě standardních veřejných a privátních klíčů definuje ještě extended klíče, což jsou klíče rozšířené o 256bitů entropie. Tvorba uzlu probíhá buď pomocí derivace z rodiče, nebo zadáním extended klíče. Pomocí uzlu v tomto stromě pak můžeme vygenerovat kryptografické klíče pro všechny jeho potomky podle tří následujících možností:

- PK rodič→PK potomek
- PK rodič→VK potomek

- VK rodič→VK potomek

Jak přesně toto generování probíhá, je uvedené v [28]. Z hlediska pochopení problematiky není potřeba zabíhat do přesných kryptografických funkcí.

Uzel v tomto stromě je definovaný svojí cestou. Podle BIP 44 [29] má tato cesta následující podobu:

```
m / purpose' / coin_type' / account' / change / address_index
```

Apostrof v cestě značí hardened potomka.

- Purpose bude konstantní a bude mít hodnotu 44. Značí použití podle BIP44 specifikace.
- Coin type značí o jakou kryptoměnu se jedná. Jedna takováto struktura umožňuje generovat klíče potažmo účty pro různé kryptoměny. Seznam všech registrovaných kryptoměn je k dispozici zde[1]. V případě aplikace AEW toto pole bude také konstantní a bude mít hodnotu 60 značící Ethereum. Toto pole používají ostatní peněženky, které sdružují různé kryptoměny.
- Account umožňuje mít pod jednou kryptoměnou víc kolekcí příslušných účtů. V rámci aplikace AEW se počítá s jednou kolekcí pro jednu peněženku. Pole tedy bude mít opět konstantní hodnotu 0.
- Change značí, zda se jedná o externí nebo interní adresy pro danou kolekci. Ethereum na rozdíl od Bitcoinu interní adresy nepoužívá. Pole, které bude mít hodnotu 0 značí externí adresu.
- Address index značí index účtu v kolekci. Číslování začíná od 0.

Jak je vidět na obrázku 1.5, je potřeba definovat Master Seed neboli veřejný klíč k inicializaci celé této struktury. K tomu se používá seed (nebo menmonická) fráze. Tu popisuje standard BIP 39. [30] Jedná se o sousled 12 a více slov v běžném jazyce pomocí kterého se pak vygeneruje privátní klíč. Fráze vypadá na příklad takto:

```
place tape flip inherit elite turn father combine extend
unlock report pudding
```

[1] https://github.com/satoshilabs/slips/blob/master/slip-0044.md

Kapitola 2

Analýza

2.1 Názvosloví

Peněženka (Wallet) Peněženkou se rozumí instance HD Wallet popsanou v předchozí kapitole. V případě, že uživatel přijde o svoje účty, například ztrátou mobilního telefonu, tak je mu umožněno získat účty zpět importem peněženky do nového zařízení pomocí mnemonické (seed) fráze.

Účet (Account) Účtem se rozumí externí Ethereum účet, jak byl popsán v předchozí kapitole. Tento účet může být vytvořen vygenerováním pomocí peněženky nebo importem pomocí privátního klíče. Pokud dojde ke ztrátě aplikace, tak účty vytvořené pomocí importování privátního klíče nelze obnovit.

Adresářová položka (AddressBook Item) Adresářovou položkou se rozumí dvojice "název" a "ethereum adresa". Adresářová položka slouží k tomu, aby si uživatel nemusel pamatovat ethereum adresu.

Transakce (Transaction) Transakcí se rozumí entita, definovaná v Ethereum blokchainu. Jedná se o změnu stavu Ethereum blokchainu převodem Etheru nebo provedením nekonstantní funkce smart kontraktu či kombinací obojího dohromady.

Budoucí Transakce (Pending Transaction) Jedná se o transakci přenosu Etheru, která má proběhnout v budoucnu.

2.2 Požadavky

2.2.1 Funkční požadavky

F1 Aplikace umožní evidovat peněženky. Peněženka má svůj název, mnemonickou frázi, barevné označení a počet účtů pod svojí správou.

F2 Aplikace umožní evidovat účty. Účet je buď vytvořený pomocí peněženky, do které patří nebo je importovaný. Účet má svůj privátní klíč pro podepisování transakcí a dat a z něho derivovanou Ethereum adresu. Účet dále může mít nastavený alias adresy pro pohodlnější zacházení. Dále má každý účet svůj zůstatek. V poslední řadě jsou pro každý účet evidovány dvě hodnoty: Transfer Threshold a Pending Transaction Length.

V případě, že je z účtu posíláno množství Etheru, které překračuje hodnotu Transfer Threshold, převod neproběhne okamžitě, ale uloží se do smart kontraktu. Po uplynutí doby specifikované hodnotou Pending Transaction Length se obnos uložený ve smart kontraktu převede na cílový účet. Po tuto dobu je možné transakci zrušit.

F3 Aplikace umožní evidovat adresářové položky. Adresářová položka má jméno a Ethereum adresu. Kdykoliv má aplikace zobrazit Ethereum adresu, která je spojená s některou adresářovou položkou, zobrazí místo ní název této položky.

F4 Aplikace umožní evidovat transakce. Transakce má jednoznačný identifikátor na úrovni Ethereum blokchainu "Transaction Hash". Dále má zdrojový účet, cílový účet, převáděnou částku a poplatek za transakci. Poplatek za transakci se spočítá jako gas_limit * gas_price. V poslední řadě má transakce časovou značku, kdy byla zpracována a typ. Transakce může být trojího typu:

- Transfer - převod etheru
- Contract Call - provedení funkce smart kontraktu
- Transfer + Contract Call - kombinace dvou předchozích. Jedná se o volání funkce smart kontraktu s modifikátorem payable, která není fallback funkcí.

F5 Aplikace umožní uživateli podepsat transakce. Aplikace obdrží potřebné údaje, kterými jsou: cílový účet, množství Etheru k přenosu, kódovaná data pro volání smart kontraktu. Transakce může obsahovat poznámku pro větší informovanost uživatele, popřípadě rovnou účet, ze kterého má být transakce provedena. Uživatel smí transakci podepsat, tím je daná transakce finalizována a odeslána do Ethereum blockchainu, nebo smí transakci zamítnout. Tímto daná transakce nebude odeslána.

F6 Aplikace umožní uživateli podepsat data. Data k podpisu obsahují jednak vlastní data a dále mohou obsahovat poznámku pro lepší orientaci uživatele. V poslední řadě mohou obsahovat požadovaný účet k podpisu.

F7 Aplikace umožní ostatním aplikacím ověření pravosti Ethereum podpisu.

F8 Aplikace umožní, aby se uživatel mohl autentizovat pomocí Ethereum účtu. Vstupem do tohoto procesu je požadavek o autentizaci, který může obsahovat adresu účtu, který se k tomu má použít. Výstupem pak je stav autentizace.

F9 Aplikace eviduje budoucí transakce. Budoucí transakce obsahuje zdrojový účet, cílový účet, převáděnou částku a časovou značku plánovaného provedení transakce.

F10 Aplikace umožní zaslání Etheru na libovolnou adresu. V případě, že je částka vyšší než hodnota definovaná v TransferThreshold u zdrojového účtu, převod se provede přes smart kontrakt po uplynutí doby definované vzniknuvší budoucí transakcí. V opačném případě se převod provede okamžitě.

F11 Aplikace zajistí, aby ji mohl užívat jen uživatel, který vlastní/užívá zařízení na kterém je aplikace nainstalovaná a zabrání vstupu neoprávněným osobám.

F12 Aplikace umožní uživateli stáhnout transakce z blokchainu do zařízení zpětně. Toto se děje na úrovni účtu. Zpětně se prochází pevný počet bloků.

2.2.2 Nefunkční požadavky

N1 Jedná se o nativní Android aplikaci.

N2 Položky gasLimit a gasPrice jsou vyplněny automaticky a uživatel je nemůže ovlivnit.

N3 Aplikace je implementovaná v Jave.

N4 Aplikace je funkční pouze v případě funkčního připojení k danému Ethereum uzlu.

N5 Aplikace dbát na zabezpečení citlivých údajů (privátních klíčů) uložených uvnitř mobilního zařízení.

2.3 Rešerše existujících řešení

V následující části popíši rešerši aplikací, které slouží jako peněženky na kryptoměny a jsou dostupné na OS Android přes aplikaci obchod Play. Sledoval jsem několik věcí.

- jakým způsobem je zabezpečen přístup do aplikace,
- co a jak umožňuje daná aplikace evidovat,
- jakým způsobem probíhá zálohování dané peněženky,
- jakým způsobem je řešen přenos etheru,
- poskytuje-li aplikace další zajímavou funkcionalitu.

2.3.1 Moxi Wallet

Přístup do aplikace je zabezpečen pomocí jména a hesla.

Aplikace umožňuje evidovat jedinou peněženku v jeden čas a v ní mít velké množství účtů pro různé kryptoměny.

K zálohování peněženky je použit textový soubor, pravděpodobně šifrovaný zvoleným heslem. Po odhlášení z aplikace je potřeba danou peněženku znovu importovat z daného souboru.

Aplikace řeší přenos kryptoměn, dále umožňuje zasílání kryptoměn, jak přímo, tak přes externí služby.

Aplikace dostupná zde[2]

2.3.2 Mew Connect

Přístup do aplikace je zabezpečen zvolením hesla.

Aplikace umožňuje evidovat pouze jeden Ethereum účet.

Aplikace umožňuje zálohovat daný účet pomocí mnemonické fráze a sama o sobě neposkytuje žádnou další funkcionalitu kromě toho, že jí lze propojit s webovou aplikací `https://www.myetherwallet.com/`.

Tato aplikace pak poskytuje celou řadu funkcionalit od zasílání Etheru přes interakci se smart kontrakty i některými decentralizovanými aplikacemi jako ENS a Ethereum Alarm Clock.

Mobilní aplikace umožňuje volbu Ethereum sítě. Umožňuje zvolit hlavní síť MainNet a testovací síť Ropsten

Aplikace dostupná zde[3]

[2] https://play.google.com/store/apps/details?id=com.crypto.multiwallethl=ru

[3] https://play.google.com/store/apps/details?id=com.myetherwallet.mewconnect

2.3.3 Trust Wallet

Přístup do aplikace lze zabezpečit dobrovolně pomocí šesti-místného číselného pinu, je možné se prokazovat i otiskem prstu.

Aplikace umožňuje evidenci různých peněženek a v nich různých účtů.

Aplikace dále umožňuje importovat peněženku pomocí mnemonické fráze, pomocí keystore souboru i import jednoho účtu pomocí privátního klíče.

Přenos Etheru vypadá, že je řešen přímo.

Aplikace je napojena na celou řadu decentralizovaných aplikací, které lze využívat pomocí vestavěného prohlížeče, který aplikace obsahuje.

Aplikace dostupná zde[4]

2.3.4 Jaxx Liberty

Aplikace umožňuje nastavit heslo v kombinaci s otiskem prstu. Toto zabezpečení není povinné.

Aplikace umožňuje najednou evidovat pouze jednu peněženku a v ní různé účty pro různé kryptoměny.

Zálohování peněženky probíhá pomocí mnemonické fráze. Přenos etheru je možný přímo i přes externí službu.

Aplikace umožňuje rámcové nastavení poplatků za transakce v režimech nízká, střední, vysoká.

Aplikace dostupná zde[5]

2.3.5 Ethereum Wallet by Freewallet

Aplikace umožňuje nastavit heslo v kombinaci s otiskem prstu. Toto zabezpečení není povinné.

Aplikace umožňuje najednou evidovat pouze jednu peněženku a v ní různé účty pro různé kryptoměny.

Zálohování peněženky probíhá pomocí mnemonické fráze. Přenos Etheru je možný přímo i přes externí službu.

Aplikace umožňuje rámcové nastavení poplatků za transakce v režimech nízká, střední, vysoká.

Aplikace dostupná zde[6]

2.3.6 Blockchain Wallet. Bitcoin, Bitcoin Cash, Ethereum

Do aplikace se přihlašuje pomocí emailu a následně volbou pinu.

Aplikace umožňuje správu účtů.

Dále umožňuje backup pomocí mnemonické fráze.

[4] https://play.google.com/store/apps/details?id=com.wallet.crypto.trustapphl=en
[5] https://play.google.com/store/apps/details?id=com.liberty.jaxx
[6] https://play.google.com/store/apps/details?id=eth.org.freewallet.app

Aplikace dostupná zde[7]

2.3.7 Závěr rešerše

Podle mnou testovaných aplikací neexistuje na OS Android aplikace, která by umožňovala ostatním aplikacím v telefonu podepisovat Ethereum transakce, či vystavovala funkcionalitu v podobě podpisu dat a autentizace pomocí Ethereum účtu.

Nejblíže k výše uvedenému má aplikace Trust Wallet. Tato aplikace poskytuje webový prohlížeč, jehož součástí je web3 provider. Ten umožňuje transakce podepisovat a interagovat s decentralizovanými aplikacemi za podmínky, že daná decentralizovaná aplikace je webová.

K tomu, aby bylo dosaženo kýžené funkcionality musí vzniknout nová aplikace.

2.4 Možnosti řešení interakce Ethereum blokchainu a mobilní aplikace

V této části diskutuji řešení interakce a funkcionality mobilní aplikace s Ethereum blokchainem. Z hlediska řešení nejprve nadefinuji požadovanou funkcionalitu, dále představím možnosti a na závěr zvolím variantu a uvedu její výhody.

2.4.1 Definování požadované funkcionality

Abych porovnal vhodnost jednotlivých řešení musím nejprve definovat požadovanou funkcionalitu.

1. Připojení na konkrétní Ethereum uzel.
2. Odeslání podepsané transakce.
3. Odhadnutí gas limitu transakce.
4. Interakce se smart kontraktem.
5. Podpis dat pomocí Ethereum podpisu.
6. Ověření platnosti Ethereum podpisu.
7. Získání aktuálního zůstatku na daném Ethereum účtu.
8. Podpis transakce na zařízení.

[7] https://play.google.com/store/apps/details?id=piuk.blockchain.android

2.4.2 Interakce s Ethereum uzlem

Ethereum uzly jsou spravované různými programy. Nejčastěji se jedná o programy Geth (Implementace ethereum uzlu v jazyce Go. Geth je složenina slov Go a Ethereum) a Parity (implementace ethereum uzlu, kterou vyrobila firma Parity Technologies). Existují i další implementace. Ethereum uzly pak mají vystavené JSON-RPC API, pomocí kterého lze manipulovat blokchainem. RPC (Remote Procedure Call) je forma komunikace mezi dvěma nezávislými systémy zpravidla ve formě klient-server, kdy klientovi je umožněno na serveru spouštět vystavovanou funkcionalitu. JSON-RPC je lehký rychlý protokol, kde jsou přenášené informace uložené ve formátu JSON. [31]

Vystavované api je nízkoúrovňové. Obsahuje například metody jsou `eth_getBalance`, `eth_sendRawTransaction`, `eth_estimate_gas`.

2.4.3 Vlastní řešení

Vytvoření vlastního řešení zahrnuje tvorbu knihovny, která se bude připojovat na konkrétní uzel a pro část své funkcionality konzumuje jeho JSON-RPC API. Tímto jsou pokryty body 1, 2, 3 a 7. Další část (body 4, 5, 6, 8) by bylo nutné implementovat, nebo delegovat na jiné knihovny. Ačkoliv tento postup je možný, považuji ho za krajně nevhodný, jelikož již existují nástroje, které se k tomu dají použít.

2.4.4 Využití knihoven pro interakci s Ethereum uzlem

Jak jsem popsal výše, na manipulaci s blokchainem se používá JSON-RPC API. Dnes již existují implementace, které obalují jeho funkcionalitu a zajišťují s ním komunikaci. Souhrnně je nazývám knihovny Web3. Množství poskytované funkcionality a její forma závisí na implementaci konkrétní Web3 knihovny. Knihovny se mezi sebou liší podle programovacího jazyka zvoleného k jejich implementaci. Mezi nejběžnější používané knihovny patří Web3js, Web3Py, Web3J.

2.4.4.1 Web3js

Jedná se o implementaci Web3 knihovny v jazyce Javascript. Je to dnes nejběžnější a nejpoužívanější knihovna, vzhledem k tomu, že většina vznikajících decentralizovaných aplikací má formu webových aplikací. Poskytuje také největší množství funkcionality od správy účtů, přes podepisování transakcí, po interakci se smartkontrakty. Konkrétně podporuje všechny výše stanovené body.

Výhodou použití této knihovny je rozsah poskytované funkcionality a jednoduchost jejího použití. Dále pak, že tato knihovna je udržována a práce na ní stále probíhá.

Nevýhodou tohoto řešení je nutnost integrace Javascriptu a nativní Android aplikace, která je napsána v Jave.

2.4.4.2 Web3Py

Jedná se o implementaci Web3 v jazyce Python. Poskytuje podobný subset funkcionality jako knihovna Web3js, protože z ní vychází a snaží se dodržovat její API. Knihovna podporuje všechny body, kromě bodů 5 a 6.

Nevýhodou tohoto řešení je nutnost vyřešení integrace Pythonu a nativní android aplikace, která je napsána v Jave.

2.4.4.3 Web3J

Jedná se o implementaci Web3 knihovny v Javě. Co se týče poskytované funkcionality, splňuje všechny body kromě 5 a 6, avšak použití této funkcionality není tak jednoduché jako u předchozích knihoven.

Výhodou použití tohoto řešení je, že je určené přímo pro tuto platformu a tím odpadá nutnost řešení jakékoliv integrace.

Nevýhodou je netriviálnost a zátěž při delegování požadovaných funkcí na tuto knihovnu.

2.4.5 Závěr

Vytvoření kompletního řešení od úplného počátku by bylo zbytečně složité pokud jsou k dispozici knihovny s částečnou funkcionalitou. Z výše popsaných knihoven jsem vybíral mezi Web3J a Web3js. Vybral jsem si knihovnu Web3js - tedy její javascriptovou variantu. Přestože bude nutné vyřešit její integraci na mobilním zařízení. Výběr jsme provedl z těchto důvodů:

- jedná se o oficiální a nejpoužívanější variantu knihovny na interakci s Ethereum blokchain,
- jednoduchost využití,
- udržovanost tzn. že vývoj této knihovny je nyní podporován a bude i v následujících letech.

2.5 Možnosti integrace Javascriptu a nativní Android aplikace s ohledem na Web3js

V následující sekci popíši řešení, která jsem zkoušel při integraci Javascriptu a nativní aplikace OS Android. Cílem bylo zdárně integrovat funkcionalitu Web3JS knihovny do aplikace AEW.

2.5.1 LiquidCore

První knihovna, kterou jsem na integraci Javascriptu testoval se jmenuje LiquidCore. Jedná se o NodeJS virtuální stroj pro Android a IOS. Zkoušel jsem doporučované MicroService API. Toto API má dvě části serverovou a klientskou. Serverová část má na starosti přípravu a zpracování Javascriptu, který si klientská část stáhne k sobě, aby jej mohla provádět. Klientská část má dále dvě další části - Android část a Javascript část. Android (Java) a Javascript potom spolu komunikují pomocí eventů, které lze definovat.

Toto řešení má omezenou podporu nativních addonů. Nativní addony jsou javascriptové knihovny, které slouží jako přemostění pro knihovny napsané v c++ a umožňují jejich použití v NodeJS [32].

V současné době LiquidCore podporuje pouze nativní addon pro SQLite3 databázi. Knihovna web3 používá nativní addony hlavně pro kryptografické funkce. Pro všechny takto použité addony bych musel implementovat jejich přemostění do LiquidCore. Z tohoto důvodu je toto řešení nefunkční.

2.5.2 J2V8

Dále jsem zkoušel integraci pomocí J2V8, což je knhiovna sloužící k provázání Javy a knihovny V8. V8 je Javascript a WebAssembly engine os společnosti Google. Povedlo se mi spustit základní verzi J2V8. Pak jsem zkoušel spustit NodeJS instanci pomocí této knihovny a zjistil jsem, že verze knihovny dostupná v Maven repozitáři má tuto funkcionalitu vypnutou. Zkoušel jsem tedy knihovnu J2V8 s podporou NodeJS zkompilovat, a to se mi nepovedlo. Tímto jsem toto řešení zavrhnul.

2.5.3 Android WebView

Další řešení, které jsem zkoušel v průběhu práce, bylo Android WebView. [33] [34] Je nutné rozlišovat Android WebView jako komponentu OS Android a jako třídu sloužící pro komunikaci s touto komponentou. Komponenta WebView je samostatná aplikace nainstalovaná na cílovém zařízení. Tato aplikace má funkcionalitu omezeného webového prohlížeče. Umožňuje ostatním aplikacím nainstalovaným v telefonu načítat a zobrazovat webové stránky a Javascript.

WebView je dále třída a součást prezentační vrstvy OS Android. Tato třída je potomek View a slouží k fyzickému zobrazení webové stránky tzn. komunikuje se systémovou komponentou WebView a zobrazuje obsah, který od ní dostane. Jiná možnost komunikace se systémovou komponentou WebView než pomocí třídy WebView neexistuje.

Aplikace AEW webový obsah nezobrazuje, ale potřebuje Javascriptový engine na provádění Javascriptu.

Zjistil jsem, že novou instanci třídy WebView lze vytvořit i v aplikačním kontextu a sdílet ji pro celou aplikaci. Při takovémto zacházení, přestože se jedná o View, nebude takto inicializované WebView nikde zobrazeno a uživateli

s ním nebude umožněno interagovat. Přitom ho ovšem půjde použít pro načtení a provádění Javascriptu, což je definovaný cíl.

Dále v textu beru WebView jako třídu. WebView má standardně zakázaný Javascript z bezpečnostních důvodů, ten však v případě potřeby lze povolit. Pro komunikaci směrem do WebView lze použít pouze hlavní vlákno, ale tato komunikace není blokující a přepínání vláken díky použití RxJavy nepředstavuje problém. WebView do sebe umožňuje před načtením stránky injektovat Java třídu, která obsahuje funkce anotované @JavaScriptInterface. Takto anotované funkce lze pak volat z Javascriptu. Toto se používá ke komunikaci směrem z WebView do aplikace.

Potřebnou funkcionalitu Web3js knihovny se mi podařilo pomocí WebView zprovoznit.

2.5.4 Závěr

Pro aplikaci AEW jsem ze zkoušených řešení integrace Javascriptu do OS Android vybral funkční řešení pomocí WebView. Při návrhu jsem dal důraz, aby WebView (resp. komunikace s ním) byla dostatečně abstrahovaná v případě, že by v budoucnu přišla jiná varianta, kterou by bylo možno použít.

2.6 Zabezpečení aplikace AEW

Vzhledem k tomu, že aplikace zachází s peněžními a dalšími citlivými prostředky, je potřeba ji zabezpečit. Toto zabezpečení rozdělím na dvě části:

1. zabezpečení přístupu do aplikace z hlediska autentizace a autorizace uživatele,

2. zabezpečení citlivých údajů (privátních klíčů) uložených v aplikaci.

V následující části popíši oba problémy a zvolená řešení pro aplikaci AEW.

2.6.1 Zabezpečení přístupu do AEW z hlediska autentizace a autorizace uživatele

Cílem zabezpečení přístupu do aplikace je povolit přístup pouze autorizovaným osobám. Z pohledu aplikace AEW se jedná o osobu vlastnící/používající zařízení, na kterém je aplikace nainstalovaná.

Jedná se čistě o přístup a autentizaci/autorizaci na úrovni aplikace AEW a s Ethereem nemá nic společného. (Aplikace AEW ovšem autentizaci na úrovni Ethereum pro ostatní aplikace umožňuje.)

Toto řešení může mít různé formy. V následujících odstavcích tyto formy představím a zasadím jejich vhodnost do kontextu aplikace AEW.

2.6.1.1 Uživatelsky zvolené heslo

První forma, která připadá v úvahu je, že uživatel si sám zvolí při prvním spuštění (nebo při registraci) heslo, kterým se pak prokazuje při autentizaci. Toto řešení používá většina mnou testovaných peněženek. Má však následující nevýhody:

1. Zatěžuje uživatele dalším heslem, které si musí pamatovat.
2. Přidává další zabezpečený údaj, o který se aplikace musí starat.

2.6.1.2 OAuth

Z pohledu předchozích řešení je vhodnější tuto autentizaci přenechat jiné službě, se kterou aplikace vykomunikuje, jestli autentizace proběhla úspěšně, či nikoliv.

Z hlediska bodu 1. tak většina uživatelů bude mít externí účet vybrané společnosti poskytující autentizaci a autorizaci přes OAuth (tj. Facebook, Google atd. . .). Což by tento problém určitě vyřešilo.

Co se týče bodu 2. a dalších případných nevýhod tohoto řešení, si nejsem vědom kvůli mé rámcové znalosti tohoto protokolu. I když by řešení pomocí tohoto protokolu mohlo připadat v úvahu, je z mého úhlu pohledu složité na ověření i implementaci.

2.6.1.3 Autentizace pomocí údajů pro zámek obrazovky

Existuje další a elegantnější řešení. Tím je autentizace uživatele na úrovni zařízení. OS Android nabízí svým uživatelům možnost zabezpečit své zařízení pomocí zámku obrazovky.

Tímto nemám na mysli zámek obrazovky jako ochranu proti náhodnému vytočení telefonu v okamžiku umístění v kapse uživatele, ale to, že se uživatel musí autentizovat vždy při odemykání svého zařízení. Tuto funkcionalitu v dnešní době používá naprostá většina uživatelů. Forma této autentizace závisí na uživateli a možnostech jeho zařízení.

V době psaní této práce jsou dostupné tyto formy:

1. číselný PIN
2. gesto (vzor)
3. heslo
4. otisk prstu

Otisk prstu lze kombinovat s předchozími variantami. Toto řešení je elegantní z následujících důvodů. Údaje pro tuto autentizaci jsou velmi dobře zabezpečené na úrovni OS Android. Pro ověření této autentizace se používá

Android KeyStore, o kterém níže píši podrobněji. Jednoduše lze nastavit délku platnosti autentizace. Nevýhodou tohoto řešení může být to, že o uživateli během autentizace nejsou získány žádné informace, avšak aplikace AEW žádné takové informace nepotřebuje.

2.6.1.4 Závěr

V předchozí sekci jsem představil tři možné způsoby řešení zabezpečení přístupu do aplikace AEW. Jedná se o uživatelsky zadávané heslo, OAuth a autentizaci pomocí zámku obrazovky. S ohledem na výhody a nevýhody se mi zdá jako nejlepší řešení posledně jmenovaný způsob, jednak z hlediska bezpečnosti a dále vzhledem k nároku na uživatele. Z těchto důvodů jsem zvolil toto řešení pro aplikaci AEW.

2.6.2 Zabezpečení citlivých údajů uložených v aplikaci

Citlivými údaji se v aplikaci AEW rozumí privátní klíče. Jednak privátní klíče od externích Ethereum účtů, které slouží k podepisování transakcí a dále privátní klíče od uložených HD Wallets sloužících ke generování nových účtů.

Pro ukládání dat se v Android aplikacích používají standardně dvě možnosti. První jsou takzvané Shared Preferences. Jedná se o obyčejný textový soubor ve formátu klíč-hodnota. Tento textový soubor lze následně přímo získat z telefonu a přečíst tzn. neposkytuje žádnou míru zabezpečení. Používá se pro drobná uživatelská nastavení v rámci aplikace.

Druhou možností je uložení dat v databázi aplikace. Android používá SQLite3. Jedná se o tradiční SQL databázi, která je celá uložena do souboru. Tato databáze není standardně (i když možnost tu je viz níže) nijak zabezpečena. Soubor s databází lze opět extrahovat ze zařízení a přečíst např. pomocí řádkového programu sqlite3. Pouhé uložení privátních klíčů pomocí těchto dvou možností by bylo naprosto nedostačující z hlediska bezpečnosti.

Android poskytuje komponentu Android KeyStore která, jak už název napovídá, je určena pro ukládání a manipulaci s kryptografickými klíči. Podle [35] je poměrně dobře zabezpečena a její průlom by znamenal velké problémy pro celý OS Android. Klíčové vlastnosti:

1. Klíče uložené v AndroidKeystore nejsou součástí aplikačního procesu a nejde je z této komponenty získat zpět a použít v aplikaci. Lze je pouze použít pomocí této komponenty.

2. Klíče v AndroidKeyStore může použít jen aplikace která je vlastní, tzn. že jiná aplikace nainstalovaná v zařízení je nemůže použít.

3. Použití klíčů v AndroidKeyStore může být navázáno na další zabezpečovací prvky. Těmito prvky jsou časová platnost a autentizace uživatele.

4. V případě, že klíč je navázán na autentizaci uživatele, a uživatel změní svoje bezpečnostní nastavení (sám ho změní, vypne nebo provede obnovení továrního nastavení), tento klíč je zneplatněn.

2.6.2.1 Optimální řešení

Optimální řešení by vypadalo tak, že všechny privátní klíče (externích Ethereum účtů a klíče peněženek) jsou uložené v AndroidKeyStore. Toto řešení je nejlepší z hlediska bezpečnosti. Klíče v AndroidKeyStore považuji za dobře zabezpečené. Ovšem toto řešení má následující problémy. První problém je, že vložit vlastní klíč do tohoto komponentu není jednoduché a mě se to nepovedlo udělat.

Druhý problém vyplývá z bodu číslo 1 definice AndroidKeyStore. Klíče z tohoto komponentu nelze vyextrahovat, což je dobré z hlediska bezpečnosti, špatné pro další možnou potřebnou práci s klíči. Toto by znemožnilo delegování určitých funkcionalit na ostatní knihovny. Konkrétně by to znamenalo implementovat derivaci privátního klíče účtu z privátního klíče peněženky tak, jak je popsaná v BIP32 (viz kapitola 1), pomocí AndroidKeyStore.

Dále by to znamenalo nutnost implementace podepisování Ethereum transakcí a dat pomocí tohoto KeyStore.

2.6.2.2 Zabezpečení SQLite databáze pomocí CWAC-SafeRoom

Pro manipulaci s SQLite databází aplikace se běžně používá knihovna Room ze sady Android Architecture Components (viz implementace). Zkoumal jsem způsob zabezpečení manipulace s databází pomocí této knihovny. Knihovna Room sama o sobě nenabízí žádné bezpečnostní prvky, protože je vytvořena pro ten nejběžnější případ, kterým je ukládání běžných dat.

Narazil jsem na knihovnu CWAC-SafeRoom, což je napojení knihovny Room na knihovnu SQLiteCipher, která umožňuje šifrování SQLite databáze jako celku. Toto řešení by problém nezabezpečené SQLite databáze umožnilo i s uložením privátního klíče. Použití této knihovny vyžaduje při použití databáze zvolit heslo, které je pak použité pro šifrování databáze. Zde vznikají dva problémy: Jak takové heslo získat? A kde ho poté následně uložit?

Heslo lze samozřejmě získat od uživatele, což ale z hlediska bezpečnosti, jak jsem popsal výše, není žádoucí. Lepší je heslo vygenerovat.

Palčivější problém je s uložením vygenerovaného hesla. Není možné použít databázi z důvodu jejího šifrování. Shared Preferences nejsou zabezpečené. Optimální je takové heslo opět uložit do AndroidKeyStore. Toto ale nelze, protože se heslo nedá získat zpět. To znamená, že heslo musí být uložené podobným způsobem jako se ukládají API klíče pomocí NDK.

2.6.2.3 Šifrování údajů v databázi pomocí AndroidKeyStore

Když se k uložení klíčů nepoužije AndroidKeyStore a ani se nezabezpečí databáze jako celek, přichází na řadu zabezpečení konkrétních dat v databázi. Toto zabezpečení se provádí pomocí šifrování. Zde už opět přichází na řadu AndroidKeyStore, který umožňuje generování, skladování a použití klíčů pro symetrické šifrování. Řešení, které se nabízí, je data již při uložení do databáze zašifrovat a skladovat je v databázi šifrované. Při použití je pak zpátky dešifrovat.

Toto řešení si zachovává část bezpečnosti poskytovanou AndroidKeyStore, protože symterický klíč, uložený v AndroidKeyStore, je dobře zabezpečený. Bezpečnost dat závisí také na volbě použité šifry a jejich parametrech. Dále toto řešení má výhodu v tom, že použití tohoto klíče lze velmi jednoduše navázat na autentizaci uživatele.

Třetí a poslední věcí, kterou bych zmínil je, že jak bylo popsáno výše, tak při změně nastavení bezpečnosti je tento klíč trvale zneplatněn. To znamená ztrátu šifrovaných klíčů v databázi a je to důležité při ztrátě/odcizení zařízení.

2.6.2.4 Závěr

Po dohodě s vedoucím své práce (vzhledem k tomu, že se jedná o prototyp a vzhledem k efektivitě práce) jsem zvolil poslední popsané řešení, přestože z hlediska bezpečnosti by bylo optimální použít první řešení. Bude-li v budoucnu toto řešení shledáno jako nedostačující, bude nutnost použít první navrhované řešení.

Aplikace používá šifrovací algoritmus AES s provozním módem CBC a PKCS7 paddingem. Inicializační vektor je skladován v databázi přímo u šifrovaných dat. Tyto údaje jsem zvolil výběrem, správnost výběru je diskutovatelná. Volba těchto parametrů není pro prototyp aplikace AEW důležitá. Když bude zjišťěno lepší nastavení, dají se parametry šifry překonfigurovat.

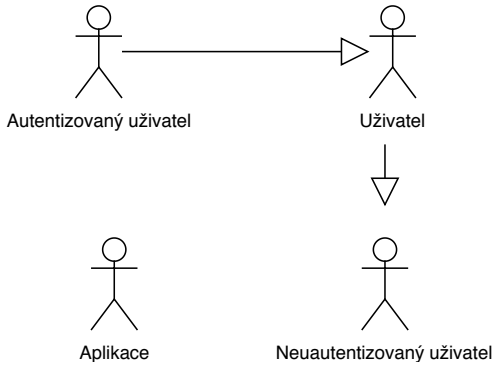
2.7 Případy užití

V této části popíši nejprve aktéry případů užití v aplikaci AEW a i samotné případy užití.

2.7.1 Aktéři v rámci aplikace AEW

V rámci aplikace jsou rozlišeni čtyři aktéři. Jsou jimi:

- Neautentizovaný uživatel - Jedná se o uživatele, který neprošel prvotní autentizací při vstupu do aplikace. Takovému uživateli je zobrazena pouze obrazovka s požadavkem o přihlášení.



Obrázek 2.1: Diagram aktérů v aplikaci AEW

- Uživatel - jedná se o uživatele, který prošel, prvotní autentizací při vstupu do aplikace, ale doba platnosti jeho autentizace již vypršela.
- Autentizovaný uživatel - jedná se o uživatele, který se autentizoval během posledních 2 minut.
- Aplikace - jedná se o aplikace nainstalované v uživatelově telefonu, které budou chtít využít funkcionalit aplikace AEW, které jsou poskytnuty pro ostatní aplikace. (Může se jednat i o samotnou aplikaci AEW).

Aktéry a jejich vztahy zachycuje obrázek 2.1

2.7.2 Případy užití

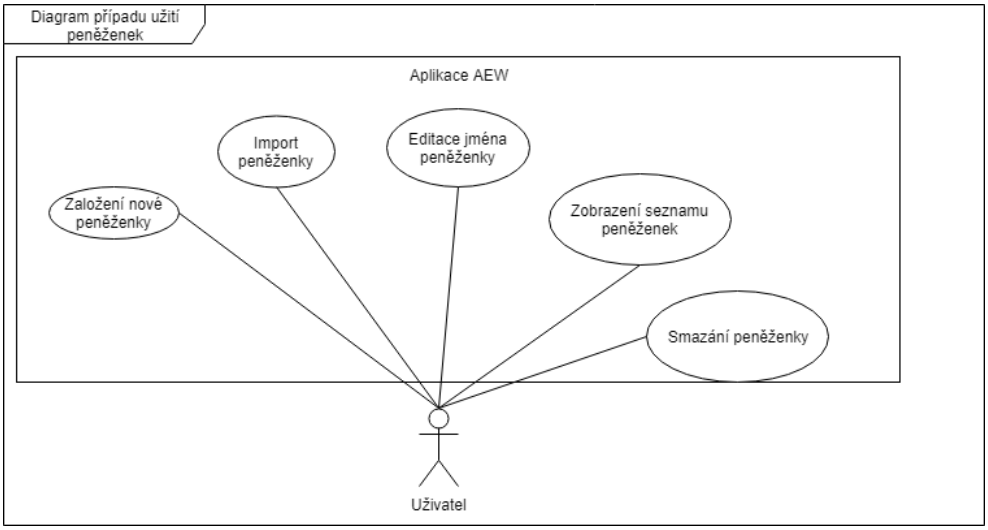
2.7.2.1 Případy užití týkající se peněženek

Diagram je vidět na obrázku 2.2

Založení nové peněženky Uživatel má možnost založit novou peněženku. Při jejím založení vyplní její jméno, které nesmí být prázdné. Aplikace mu poté vygeneruje novou menmonickou frázi a tuto frázi mu zobrazí. Uživatel musí potvrdit, že byl s touto frází obeznámen. To má uživateli připomenout, že si má tuto frázi dobře schovat mimo používané zařízení pro možnost obnovy peněženky na jiném zařízení. Když toto potvrdí, aplikace pro danou peněženku vygeneruje její barvu a peněženku uloží.

Import peněženky Uživatel má možnost peněženku importovat. Při jejím importu vyplní její jméno a mnemonickou frázi. Jméno nesmí být prázdné. Mnemonická fráze musí obsahovat dvanáct slov oddělených mezerou. Aplikace vygeneruje barvu, která symbolizuje danou peněženku, poté peněženku uloží.

Editace jména peněženky Uživatel má možnost editovat jméno peněženky.



Obrázek 2.2: Diagram případů užití peněženek

Zobrazení seznamu všech peněženek Uživatel má možnost zobrazit si seznam všech peněženek, které aplikace spravuje. U každé peněženky je vidět její barva, název a počet účtů, které obsahuje.

Import peněženky Uživatel má možnost peněženku smazat, ale jen v případě, že neobsahuje žádné účty.

2.7.2.2 Případy užití týkající se účtů

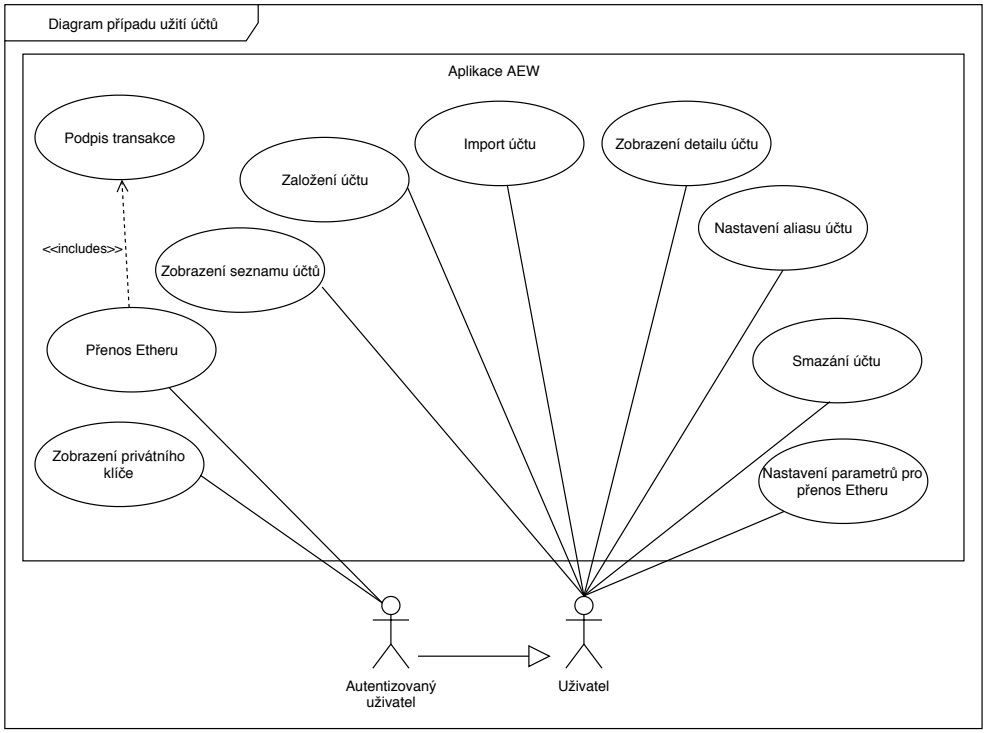
Diagram je vidět na obrázku 2.3

Zobrazení seznamu účtů Uživatel má možnost si zobrazit seznam všech účtů. V seznamu je pro každý účet vidět adresa nebo alias, dále barva dle příslušnosti k peněžence a množství Etheru na účtě. Importované účty jsou odlišené ikonou.

Založení účtu Uživatel má možnost si založit nový účet. Při založení zvolí pouze do které peněženky účet bude patřit a aplikace mu ho založí.

Import účtu Uživatel má možnost importovat existující účet. Při importu vyplní privátní klíč patřící k účtu. Takto importovaný účet nepatří do žádné peněženky a je označen speciální barvou a ikonou.

Zobrazení detailu účtu Uživatel má možnost si zobrazit detail účtu. V detailu účtu je vidět jeho adresa, alias, typ a příslušnost k peněžence, množství Etheru na účtě a hodnoty Transfer Threshold a Pending Tansaction Length



Obrázek 2.3: Diagram případů užití účtů

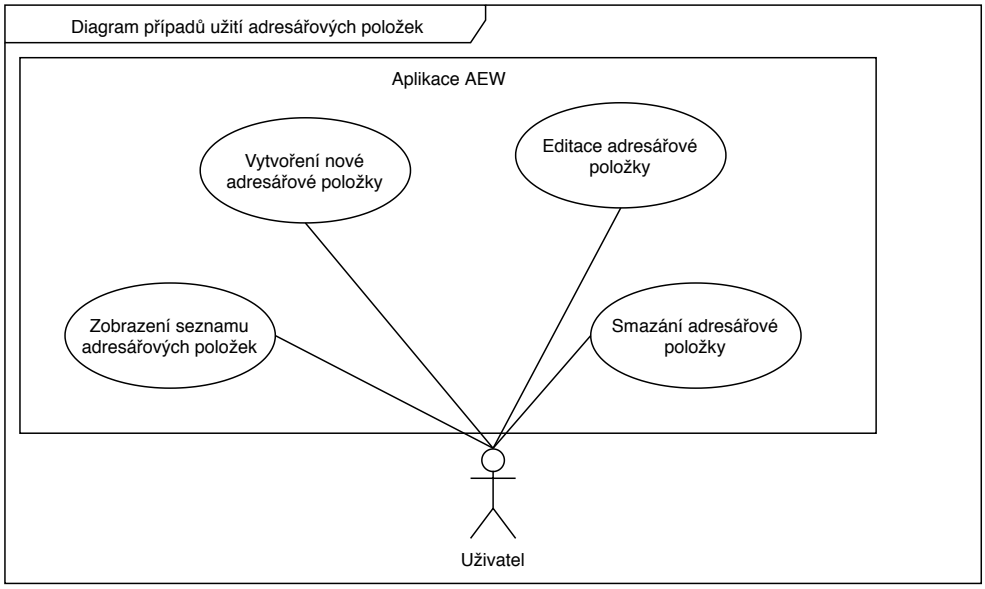
Nastavení aliasu účtu Uživatel má možnost si pro účet nastavit alias. Pokud je nastaven, zobrazuje se místo adresy.

Smazání účtu Uživatel má možnost smazat účet, ale pouze v případě, že množství Etheru nepřekračuje hodnotu 0.002.

Zobrazení privátního klíče Autentizovaný uživatel má možnost si pro daný účet zobrazit privátní klíč. Pokud se o tuto akci pokusí uživatel, je podroben autentizaci.

Nastavení parametrů pro přenos Etheru Uživatel má možnost pro daný účet nastavit hodnoty Transfer Threshold a Pending Transaction Length. Účel těchto hodnot byl popsán výše. Pokud je hodnota Transfer Threshold záporná, nebude aktivována. Hodnota Pending Transaction Length může být nastavena na 12, 24, nebo 36 hodin.

Přenos Etheru Autentizovaný uživatel má možnost z daného účtu posílat Ether na libovolnou adresu. Cílovou destinaci má možnost zadat buď jako Ethereum adresu ve formátu 0x... nebo názvem účtu či adresářové položky.



Obrázek 2.4: Diagram případů užití adresářových položek

Podmínkou je, že musí mít na účtě dostatek Etheru, jak na přenos, tak na zaplacení transakce.

V případě zadání platných hodnot a potvrzení, je vyvolána obrazovka na podpis transakce.

2.7.2.3 Případy užití týkající se adresářových položek

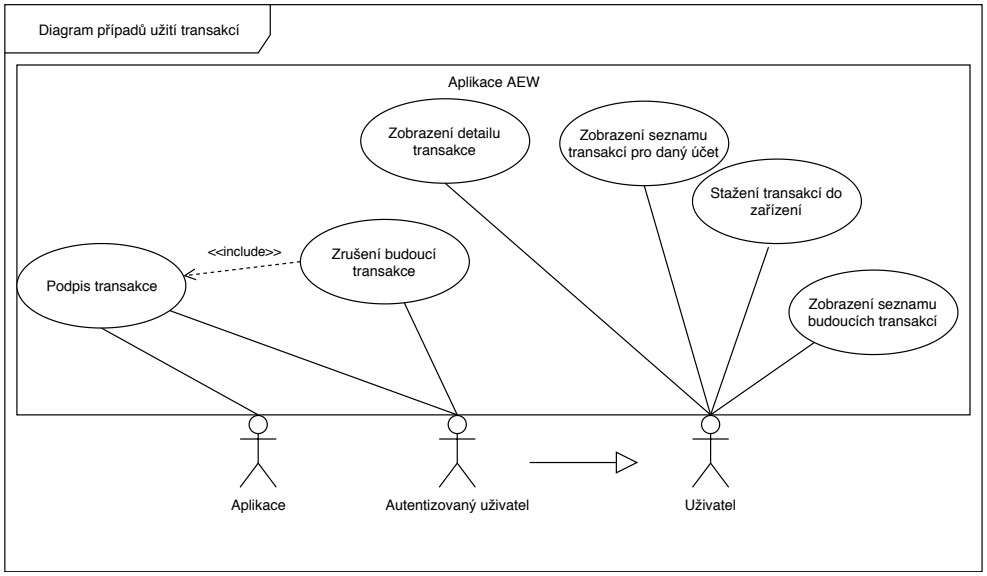
Diagram je vidět na obrázku 2.4

Zobrazení seznamu adresářových položek Uživatel má možnost zobrazit si seznam všech adresářových položek. Položka v seznamu obsahuje svůj název a Ethereum adresu.

Vytvoření nové adresářové položky Uživatel si může vytvořit novou adresářovou položku. Při vytváření vyplňuje oba údaje - jméno a adresu. Jméno musí být unikátní napříč aplikací.

Editace adresářové položky V adresářové položce lze upravit oba údaje. Jméno musí být opět unikátní.

Smazání adresářové položky Uživatel má možnost smazat adresářovou položku.



Obrázek 2.5: Diagram případů užití transakcí

2.7.2.4 Případy užití týkající se transakcí

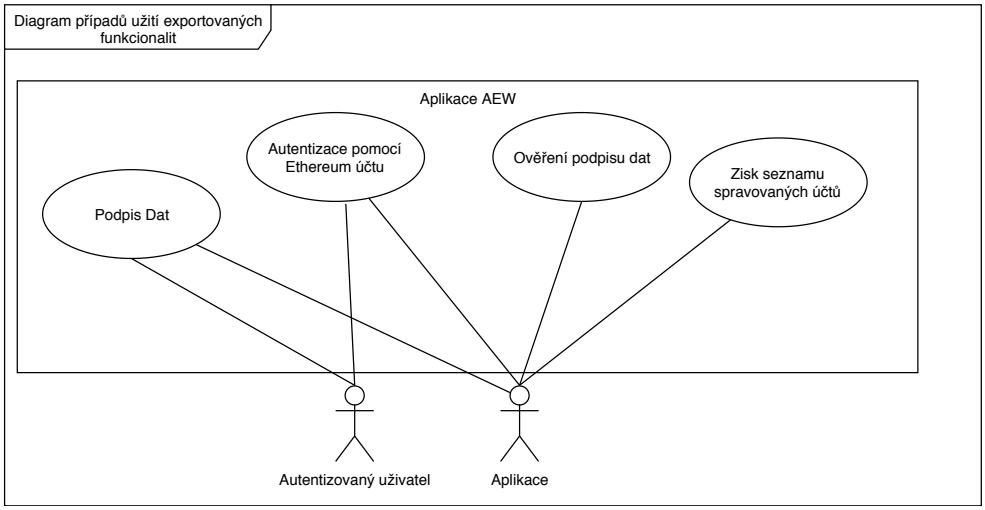
Diagram je vidět na obrázku 2.5

Zobrazení seznamu transakcí pro daný účet Uživatel má možnost zobrazit seznam transakcí pro daný účet. Položka seznamu obsahuje cílový účet a celkové náklady na transakci. Položky jsou barevně odlišeny podle typu transakce. Transakce jsou seřazeny chronologicky od nejnovější. Uživateli jsou zobrazeny pouze transakce, které jsou uloženy v zařízení.

Podpis transakce Aplikace mají možnost iniciovat podpis Ethereum transakce. Tuto transakci podepisuje autentizovaný uživatel. Při podpisu je zobrazen cílový účet transakce, množství přenášeného Etheru, odhadované náklady na transakci, typ transakce. Dále je zde zobrazena poznámka, pokud je přítomna. Pokud je předem vybraný účet, a lze jím transakci podepsat, je také zobrazen. V opačném případě je uživateli umožněno účet zvolit.

Uživatel má možnost transakci podepsat (přijmout), nebo zamítnout. V případě přijetí je transakce podepsána, odeslána na blokchain a uložena do zařízení. Výsledek operace spolu s transakčním hashem je vrácen aplikaci, která je iniciátorem tohoto případu.

Stažení transakcí do zařízení Uživatel má možnost si pro daný účet stáhnout transakce do zařízení zpětně. Do zařízení jsou uloženy jen takové transakce, které se tam nevyskytují. Stahují se transakce jen pro určitý počet bloků zpětně.



Obrázek 2.6: Diagram případů užití exportovaných funkcionalit

Zobrazení detailu transakce Uživatel má možnost si zobrazit detail transakce. V detailu je vidět cílový účet, typ transakce, množství přeneseného Etheru, náklady na transakci a časová značka zpracování transakce.

2.7.2.5 Případy užití týkající se budoucích transakcí

Diagram je vidět na obrázku 2.5

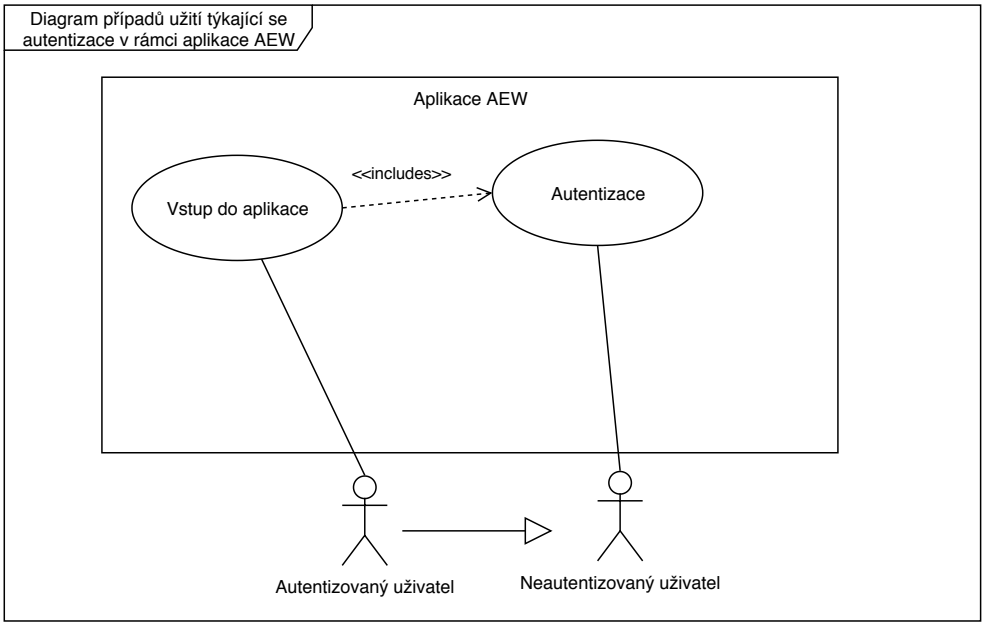
Zobrazení seznamu budoucích transakcí Uživatel má možnost si zobrazit seznam budoucích transakcí pro daný účet. Položka seznamu obsahuje cílový účet, převáděný obnos a zbývající čas do provedení transakce.

Zrušení budoucí transakce Autentizovaný uživatel má možnost budoucí transakci zrušit, to vyvolá podpis transakce.

2.7.2.6 Případy užití exportovaných funkcionalit

Diagram je vidět na obrázku 2.6. Exportovaná funkcionalita Podpis transakce již byla uvedena u případů užití týkajících se transakcí.

Podpis dat Aplikace mají možnost vyvolat podpis dat. Tato data podepisuje autentizovaný uživatel. Při podpisu dat je zobrazena jejich délka, hash a poznámka, byla-li aplikací přidána. Pokud je dodána adresa účtu, která má být použita k podpisu dat, je zobrazena. V opačném případě má uživatel možnost účet zvolit. V případě úspěšného podpisu jsou do aplikace, která tuto akci vyvolala, vrácena podepsaná data a adresa účtu použitého k podpisu.



Obrázek 2.7: Diagram případů užití týkajících se autentizace v rámci aplikace AEW

Ověření platnosti podpisu dat Aplikace zašle aplikaci AEW data, jejich podpis a adresu účtu použitému k podpisu. Aplikace AEW obdržená data zpracuje a vrátí zpět výsledek ve formě podpis platí/neplatí.

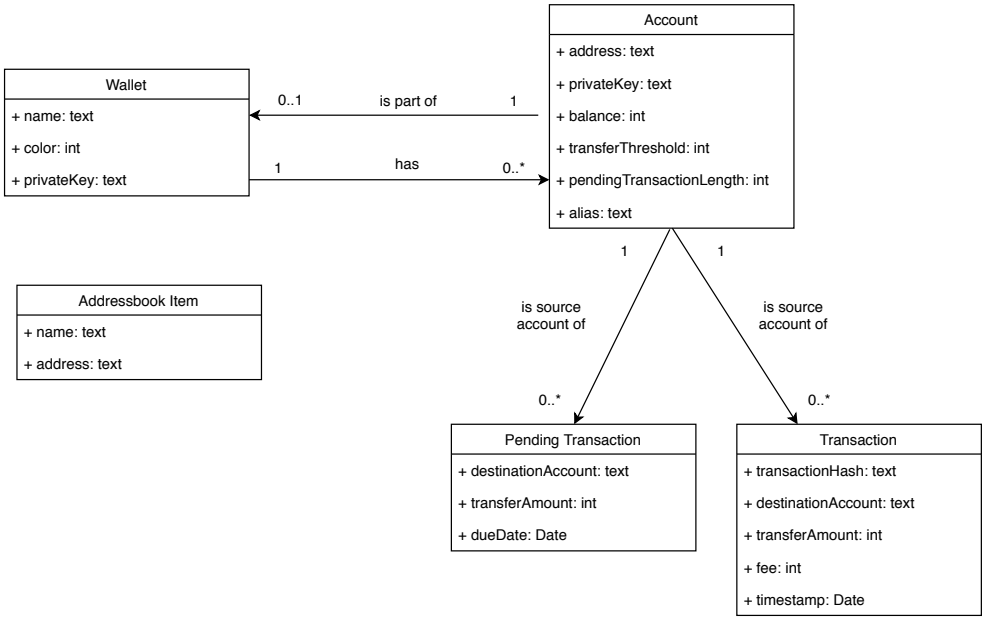
Autentizace pomocí Ethereum účtu Aplikace zašle aplikaci AEW požadavek o autentizaci pomocí Ethereum účtu. Tento požadavek může obsahovat adresu účtu, který má být autentizován. Pokud tato adresa není obsažena, uživatel má možnost zvolit účet použitý k této autentizaci. Uživatel má možnost autentizaci potvrdit, nebo zamítnout. Aplikace, která autentizaci požadovala, obdrží její výsledek.

Získání seznamu spravovaných účtů Aplikace si vyžádá seznam účtů, které aplikace AEW spravuje. Aplikace ho poskytne. Seznam obsahuje adresy a jejich případné aliasy.

2.7.2.7 Případy užití týkající se autentizace v rámci aplikace AEW

Diagram je vidět na obrázku 2.7.

Autentizace Aplikace AEW vybídne uživatele k autentizaci. Uživatel se autentizuje na úrovni zařízení pomocí pinu, hesla nebo otisku prstu. Přesná



Obrázek 2.8: Doménový model

forma autentizace závisí na tom, jaké má uživatel nastavení a zařízení. Při úspěšné autentizaci se z něj na krátkou dobu stává autentizovaný uživatel. V případě neúspěchu je vrácen na původní obrazovku.

Vstup do aplikace Při vstupu do aplikace je neautentizovaný uživatel vybídnut k autentizaci. Pokud je úspěšná, stává se z něj autentizovaný uživatel a je vpuštěn do aplikace. Jinak je vrácen na úvodní obrazovku odkud má možnost podstoupit autentizaci znovu.

2.8 Doménový model

V této sekci představím doménový model, který zachycuje vztahy mezi entitami v aplikaci.

Jak lze vidět na obrázku 2.8, doménový model není komplikovaný. Entity v obrázku jsou totožné s entitami představenými v názvosloví. Jak si lze všimnout, peněženka a účet mají asymetrický vztah. To je způsobeno tím, že aplikace bude umožňovat import účtů pomocí privátního klíče. Takto importované účty jsou brané jako externí a nespadají do žádné peněženky.

Kapitola 3

Návrh

3.1 Architektura aplikace

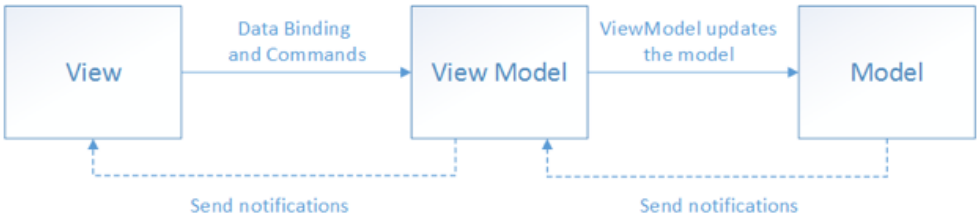
3.1.1 Architektura prezentační vrstvy

V dnešní době již mají Android aplikace doporučovanou architekturu. Jedná se o architekturu podle vzoru MVVM - Model-View-ViewModel. Následně popíši architekturu MVVM podle [36]. Diagram této architektury si lze prohlédnout na obrázku 3.1. Tento obrázek obsahuje následující tři položky:

- Model obsahuje doménová data, která chceme uživateli zobrazit. V mém případě je například seznam Ethereum účtů evidovaných v aplikaci nebo seznam transakcí pro daný účet.
- View je to, co uživatel vidí. Může mít podobu webové stránky, okna v systému či obrazovky v mobilním telefonu. Uživatel s ním má možnost interagovat a měnit tak jeho stav. View ví o existenci ViewModelu (obsahuje ViewModel) a konzumuje jeho a API a jeho data pomocí navázání. V tomto případě budou mít funkci View komponenty OS Android, a to konkrétně aktivita nebo fragment.
- ViewModel je vrstva abstrakce ležící mezi View a Modelem. ViewModel definuje funkcionalitu poskytovanou UI (ale View určuje jak funkcionalita bude dále vypadat). Definuje API, které View může konzumovat a sadu událostí, na které pak View upozorňuje. Na základě toho mění View svůj zobrazený stav. V tomto případě bude mít funkci ViewModelu komponenta ViewModel OS Android. Bude mít funkcionality týkajících konkrétních obrazovek typu: načti data, přidej účet aj.

Tuto architekturu jsem zvolil z těchto důvodů:

1. Řeší jedenu z největších nepříjemností, se kterou se lze na Androidu potkat. Touto je životní cyklus komponent.



Obrázek 3.1: Architektura MVVM [36]

2. Jak je psáno v [36], podporuje prinicp návrhu "sepration of concerns" - každá třída má jasně definovanou funkcionalitu.

3. Přispívá k dobré testovatelnosti. Na Androidu je to ještě navýšeno o to, že komponenty prezentační vrstvy - aktivity, fragmenty, adaptéry a další nelze testovat klasickými jednotkovými testy, ale je potřeba je testovat instrumentačními testy uživatelského rozhraní. ViewModel je ovšem navržen tak, aby takto otestovat šel.

4. Poslední důvod, možná ne zjevný na první pohled, ale velice důležitý je, že se jedná o oficiálně doporučenou architekturu. To znamená, že pokud bude na projektu pracovat, někdo další, kdo bude mít s vývojem Android aplikací zkušenosti, bude mu tato architektura a způsob komunikace známý.

3.1.2 Architektura aplikace jako celku

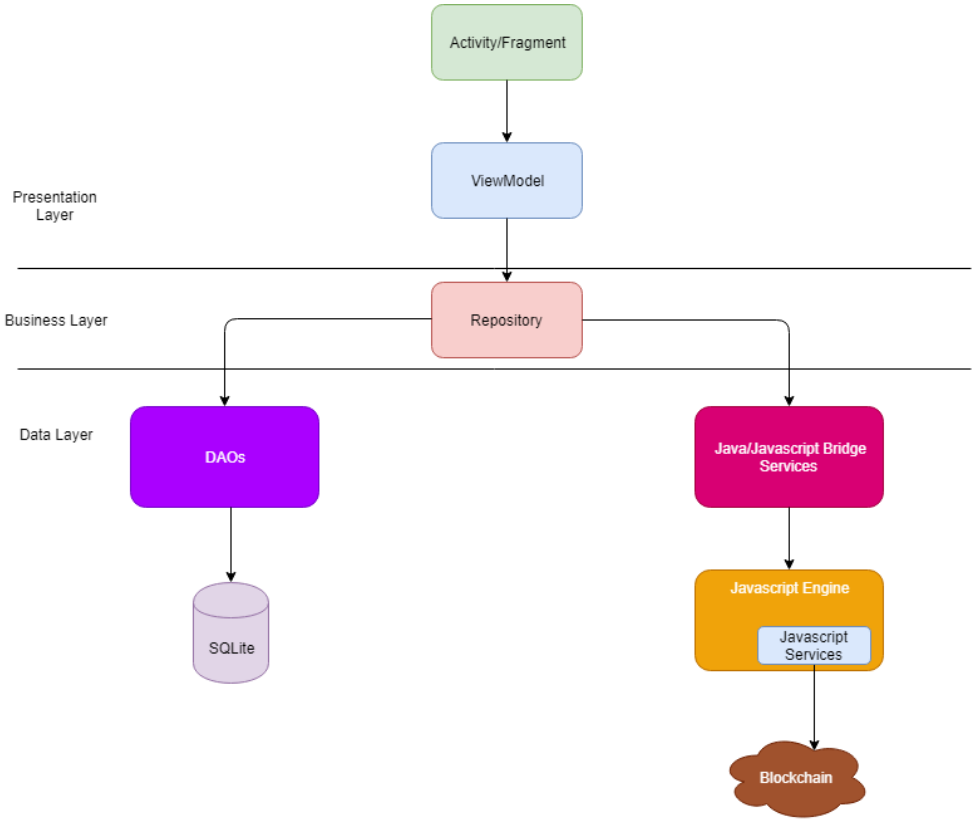
Pokud se na aplikaci podíváme pohledem vícevrstvé architektury, je vidět, že vzor MVVM popsaný výše se týká pouze prezentační vrstvy. V případě návrhu mobilních aplikací vidím v zásadě dvě možnosti:

1. Aplikace je jednoduchá a stačí jí pouze dvě vrstvy - prezentační a datová.

2. Aplikace je složitější a vrstev musí být víc.

Vzhledem k tomu, že tato aplikace agreguje data ze dvou zdrojů - databáze a blockchain a dále obsahuje aplikační logiku navíc, varianta 1. by byla naprosto nedostačující. Zvolil jsem variantu číslo 2. Diagram namodelované architektury je na obrázku 3.2.

Prezentační vrstva První vrstva je prezentační. Obsahuje vzor MVVM popsaný na počátku kapitoly. Má za úkol zobrazování dat a interakci s uživatelem. Obsahuje kmponenty OS Android.



Obrázek 3.2: Architektura aplikace jako celku

Aplikační/Business vrstva Druhá vrstva je aplikační/business. Má několik hlavních funkcionalit. První funkcionalitou, tu lze vidět z obrázku je agregace dat z dvou připojených zdrojů databáze a blokchainu. Druhou funkcionalitou je delegace určitých úkonů na Javascriptové služby běžící v Javascriptovém enginu. Poslední funkcionalitou je, že má na starosti šifrování a dešifrování citlivých údajů (privátních klíčů).

Datová vrstva Datovou vrstvu lze podle obrázku ještě rozdělit na dvě části. První část, na obrázku vlevo, se stará o správu a ukládání dat do SQLite databáze přítomné na zařízení. Druhá část, na obrázku vpravo, se stará o komunikaci s Javascriptovými službami běžícími v Javascriptovém enginu. Tyto služby poskytují komunikaci s blokchainem a logiku s tím související.

3.2 Návrh uživatelského rozhraní

V rámci návrhu aplikace jsem vytvořil hi-fidelity prototyp uživatelského rozhraní.

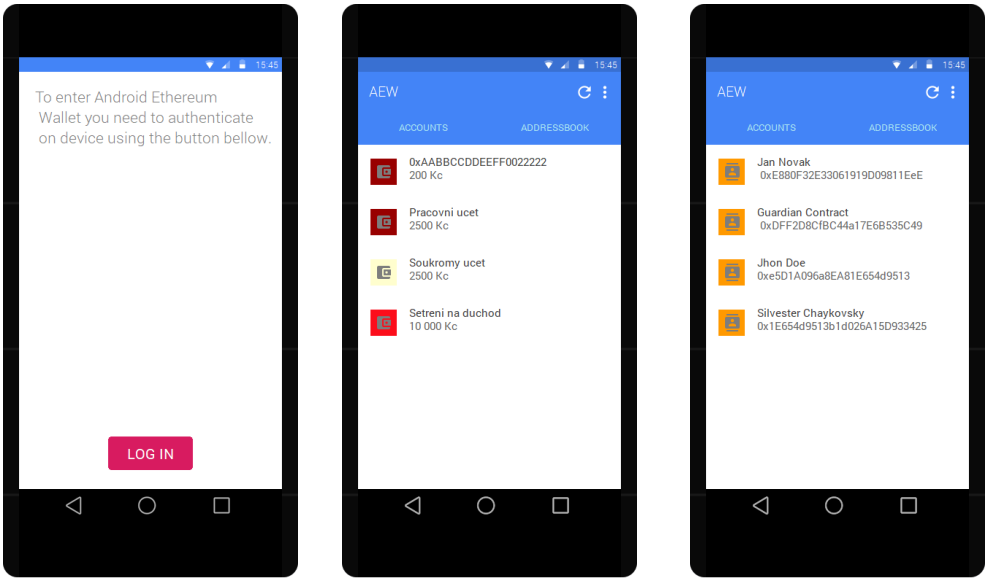
Hi-fidelity prototyp se snaží vypadat a chovat jako finální produkt. Klade důraz na realistický a detailní design a obsahuje velmi podobná nebo stejná data jako finální produkt.[37]

Standardem pro tvorbu uživatelského rozhraní je na OS Android Material design. Prvky tohoto standardu jsem se snažil dodržet, jak při návrhu uživatelského rozhraní, tak při jeho implementaci.

Tento prototyp jsem vytvořil v návrhovém nástroji JustInMind[8], který umožňuje tvorbu hi-fi prototypů mobilních aplikací a webových stránek. Projekt je dispozici na CD, vyžaduje ale buď vývojový nástroj jako takový nebo stejnojmenné rozšíření do prohlížeče Chrome.

3.2.1 Popis Obrazovek

V následující části představím jednotlivé obrazovky a okomentuji umístění určitých prvků. Pojmem menu nazývám menu dostupné v pravém horním rohu obrazovky, zobrazí se po kliknutí na ikonu třech teček. Kontextové menu je menu, které se objeví dlouhým stisknutím na konkrétní položku a umožňuje s ní provádět konkrétní akce jako editaci, nebo mazání.



(a) (b) (c)

Obrázek 3.3: Obrazovky Úvodní, Seznam účtů, Seznam adresářových položek

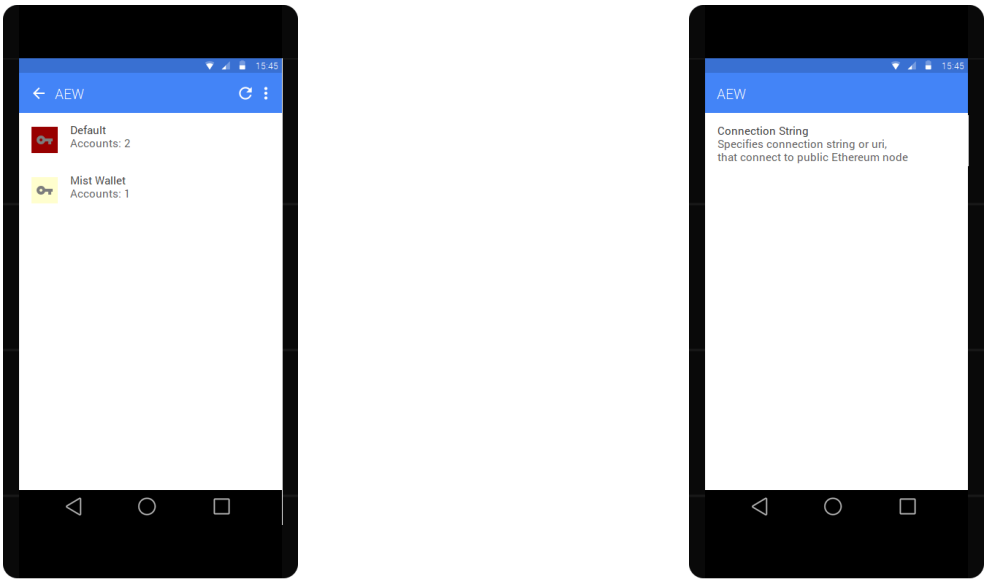
Úvodní obrazovka Úvodní obrazovka obrázek 3.3a obsahuje pouze požadavek o přihlášení (autentizaci) a tlačítko, které toto umožňuje. Autentiazce probíhá pomocí OS Android, tak jak to má nastaveno uživatel v zařízení. V ideálním

[8] JustInMind - `https://www.justinmind.com/`

případě uživatel tuto obrazovku vůbec neuvidí, automaticky se překryje autentizační obrazovkou OS Android. Tato obrazovka slouží jako záchytný bod, pokud uživatel zruší autentizaci OS Android.

Seznam účtů Tuto obrazovku lze vidět na obrázku 3.3b. Je zde zobrazen seznam všech Ethereum účtů, které jsou evidované v aplikaci. Položka seznamu obsahuje barevnou ikonu značící příslušnost účtu k dané peněžence. Externí účty mají lososovou barvu. Položka seznamu dále obsahuje Ethereum adresu, nebo alias je-li k dispozici a aktuální zůstatek účtu. Na této obrazovce je umožněno uživateli vytvořit či importovat nový účet. Tyto akce jsou dostupné v menu v levém horním rohu, protože se jedná o akce, které budou málo časté. Získání potřebných údajů pro tyto akce probíhá zobrazením dialogu. Kliknutím na daný účet se uživatel dostane zobrazení jeho detailu.

Seznam adresářových položek Tuto obrazovku lze vidět na obrázku 3.3c. Je zde zobrazen seznam všech adresářových položek. Adresářová položka obsahuje název a Ethereum adresu. Akce pro vytvoření položky se skrývá v horním menu. Akce na editaci a smazání položky v kontextovém menu.



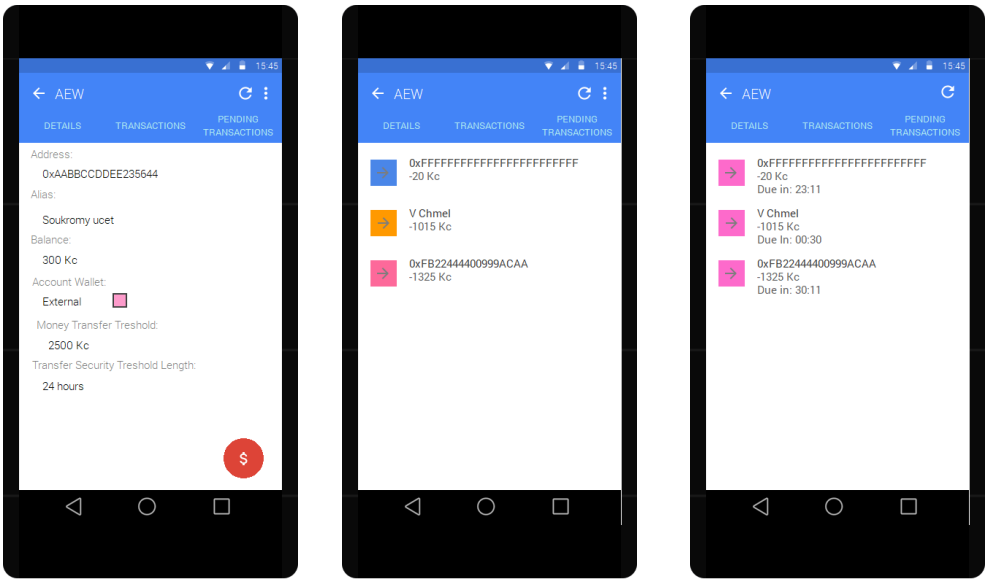
(a) (b)

Obrázek 3.4: Obrazovky Seznam peněženek a Nastavení

Seznam peněženek Tuto obrazovku lze vidět na obrázku 3.4a. Je zde zobrazen uživateli seznam všech peněženek, které jsou evidované v aplikaci. Položka seznamu obsahuje barevnou ikonu, název peněženky a počet účtů, které v ní jsou vytvořené. Akce vytvoření a import peněženky jsou málo časté

a proto jsou skryté v horním menu. Akce editace (názvu) a smazání peněženky jsou dostupné z kontextového menu.

Nastavení Tuto obrazovku lze vidět na obrázku 3.4b. Nastavení obsahuje jedinou položku connection string. Umožňuje uživateli konfiguraci připojení ke konkrétnímu Ethereum blockchainu. Zatím je nutné zadat celou URI tj. ve formátu protokol://host:port



(a) (b) (c)

Obrázek 3.5: Obrazovky zobrazující informace o detaiu účtu

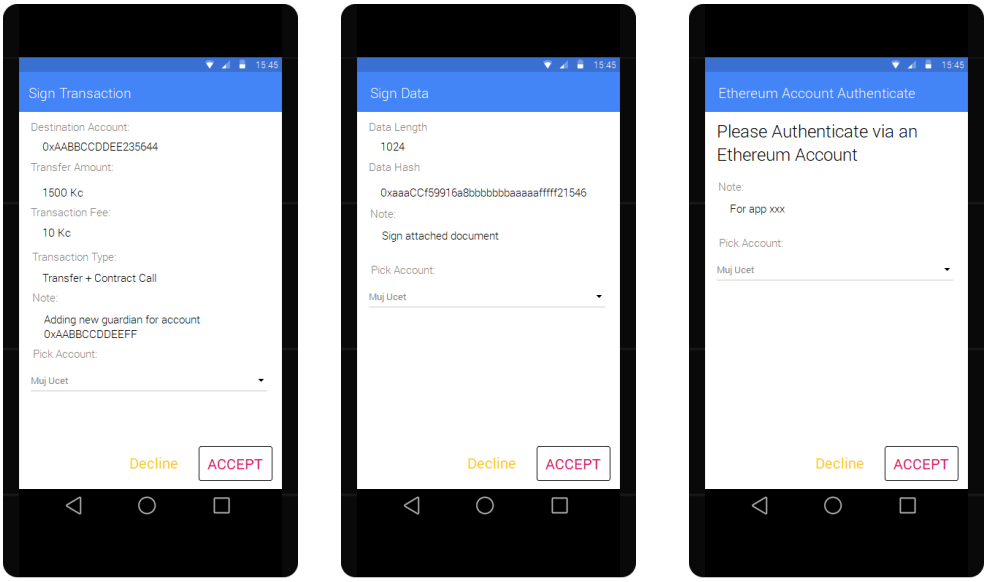
Detail účtu Tuto obrazovku lze vidět na obrázku 3.5a. Obrazovka obsahuje konkrétní údaje o daném účtu. Jedná se o jeho adresu, alias, název peněženky, do které patří, a hodnoty TransferThreshold a PendingTransactionLength. Akce na změnu názvu účtu (alias), nastavení hodnot TransferThreshold, PendingTransactionLength a smazání účtu se nachází v menu. Zadávání hodnot probíhá pomocí dialogů. Dále se v menu nachází možnost zobrazit si privátní klíč pro daný účet. Tento klíč není zobrazen standardně, protože se jedná o citlivý údaj vyžadující autentizaci.

Akce přenosu Etheru je dostupná pomocí kulatého tlačítka ve spodní pravé části obrazovky. Tato akce je takto zdůrazněna, protože jsem počítal s tím, že bude častá.

Seznam transakcí daného účtu Tuto obrazovku lze vidět na obrázku 3.5b. Je zde zobrazen uživateli seznam všech transakcí, které pocházejí z

daného účtu a byly podepsané touto aplikací. Seznam je seřazen chronologicky (nejnovější transakce nahoře). Položka seznamu obsahuje barevnou ikonu značící typ transakce, cílový účet (adresa nebo alias) a celkové náklady na transakci. Po kliknutí na danou transakci se zobrazí dialog s jejími detaily.

Seznam budoucích transakcí daného účtu Tuto obrazovku lze vidět na obrázku 3.5c. Je zde zobrazen uživateli seznam všech budoucích transakcí, které pocházejí z daného účtu. Položka seznamu obsahuje barevnou ikonu, cílový účet (adresa nebo alias), celkové náklady na transakci a dobu za kterou má být provedena. Akce zrušení dané transakce je dostupná z kontextového menu.



(a) (b) (c)

Obrázek 3.6: Obrazovky Podpis transakce, Podpis dat, Autentizace pomocí Ethereum účtu

Podpis transakce Tuto obrazovku lze vidět na obrázku 3.6a. Obrazovka slouží pro podpis transakce. Jsou zde zobrazeny uživateli její údaje. Jedná se o zdrojový účet (je-li připojen), cílový účet, přenášená částka, poplatek za transakci, typ transakce, poznámka (je-li připojena) a v případě, že není dodán zdrojový účet, může ho uživatel vybrat pomocí rozbalovacího seznamu. Uživatel má možnost transakci podepsat, nebo zamítnout/zrušit.

Podpis dat Tuto obrazovku lze vidět na obrázku 3.6b. Obrazovka slouží pro podpis dat. Je zde zobrazena jejich délka a hash, dále poznámka (je-li připojena) a v případě že není dodán účet pro podpis, může ho uživatel

vybrat pomocí rozbalovacího seznamu. Uživatel má možnost data podepsat, nebo podpis zrušit.

Autentizace pomocí Ethereum účtu Tuto obrazovku lze vidět na obrázku 3.6c. Obrazovka slouží pro autentizaci pomocí Ethereum účtu. V případě, že je dodán požadovaný účet, je tento účet zobrazen, jinak je zobrazena výzva a rozbalovací seznam, který umožňuje uživateli vybrat požadovaný účet. Také je zde zobrazena poznámka, je-li k dispozici. Autentizaci má uživatel možnost potvrdit, nebo zrušit.

3.2.2 Navigace v aplikaci

Graf obrazovek si lze prohlédnout na obrázku 3.7. Aplikaci jsem rozčlenil horizontálně na 4 úrovně:

úroveň 0 vstupní obrazovka

úroveň 1 zde začíná vlastní aplikace - záložky se seznamem účtů a se seznamem adresářových položek

úroveň 2 záložky s obrazovkami detailu účtu (detail účtu, seznam transakcí a seznam budoucích transakcí), seznam peněženek s nastavení

úroveň 3 podpis transakce

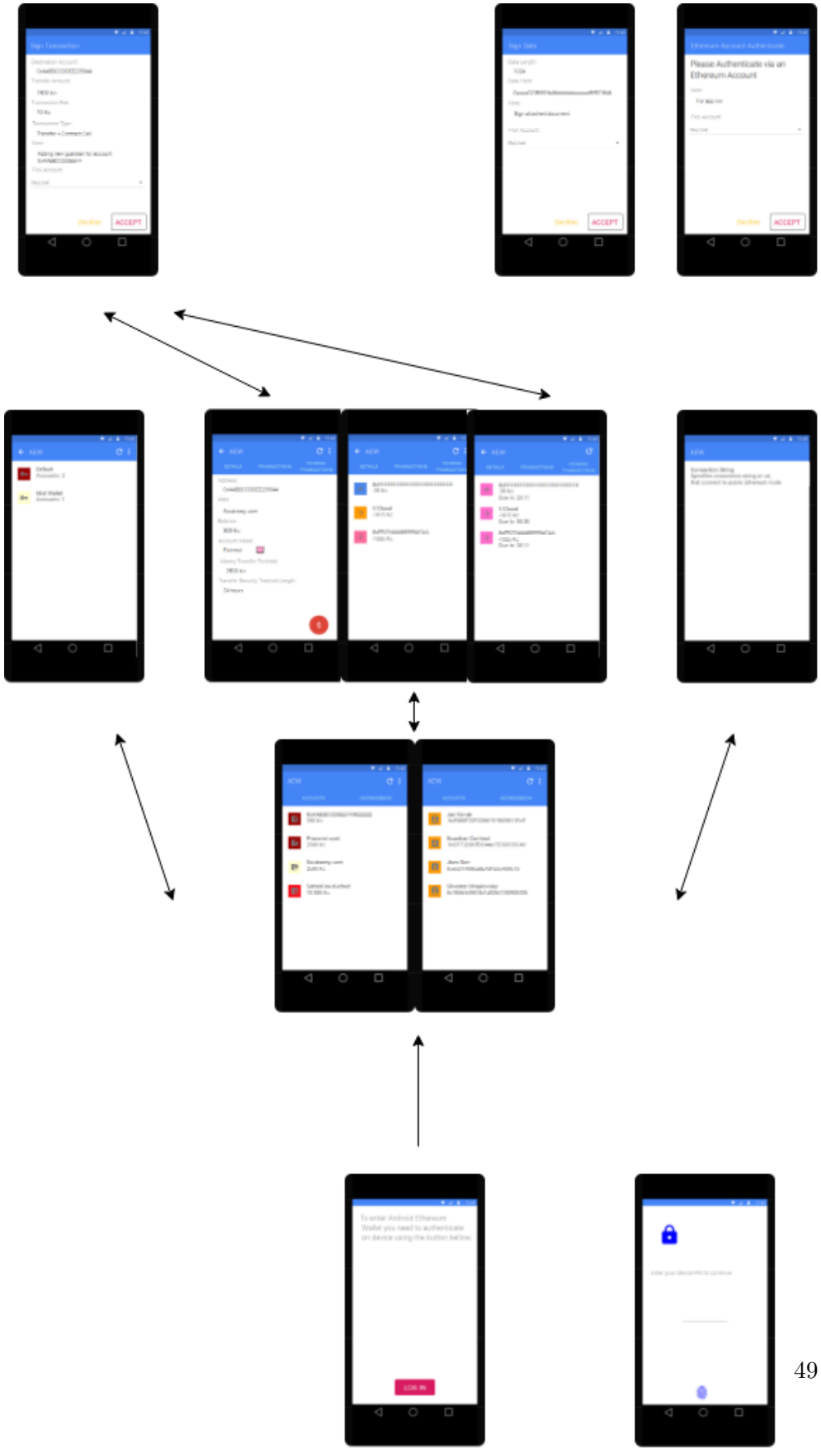
Hlavní úroveň aplikace je úroveň 1. Je k diskuzi jestli na tuto úroveň nedat i obrazovku se seznamem peněženek. Já jsem si zvolil dát ji na druhou úroveň a tím jí skrýt, protože je předpoklad, že běžný uživatel ji použije pouze při prvním spuštění aplikace, aby si založil popřípadě naimportoval peněženku a poté už ji nebude v podstatě využívat. K navigaci na hlavní úrovni aplikace jsou k dispozici podle standardu Material Design 3 prvky [38]. Použití těchto prvků záleží na počtu obrazovek na hlavní úrovni aplikace. Standard Material Design do těchto obrazovek nepočítá Nastavení.

Prvním prvkem je Navigation Drawer. Jedná se o velké vysunovací menu z levé části obrazovky. Je vhodné pro 5 a více položek.

Druhým prvkem je Bottom Navigation Bar. Jedná se o navigační panel ve spodní části obrazovky. Je vhodný pro 3 až 5 položek. [39]

Posledním prvkem jsou Záložky. Jedná se o navigační panel v horní části obrazovky. Je vhodný pro 2 a více obrazovek. Zároveň jde o jediný použitelný prvek na ostatních úrovních aplikace v rámci horizontální navigace.

Podívejme se znovu na obr 3.7. Z přihlašovací obrazovky se po úspěšné autentizaci automaticky dostaneme do aplikace. Z této úrovně (úroveň 1) se pomocí menu lze dostat jak do nastavení (úroveň 2 vpravo), tak i na obrazovku se seznamem peněženek (úroveň 2 vlevo). Kliknutím prstem na položku v seznamu účtů se pak lze dostat na obrazovky tvořící detail účtu (záložka na



Obrázek 3.7: Graf navigace v aplikaci

úrovni 2 uprostřed). Z obrazovky detailu účtu (záložka na úrovni 2 vlevo) se pak při posílání Etheru dostaneme na obrazovku podpisu transakce. Další dvě obrazovky (podpis dat a autentizace pomocí Ethereum účtu) nejsou z aplikace přístupné, protože je nevyužívá, ale poskytuje je pro ostatní aplikace. Tyto obrazovky jsou nahoře vpravo a nejsou přímo připojeny ke grafu navigace aplikací. Na autentizační obrazovku OS Android (vpravo úplně dole) se dostaneme pokaždé, když je potřeba autentizace na úrovni zařízení. Toto je z hlediska návrhu zabezpečení poměrně častá akce a pro zjednodušení jsem ji v grafu neuváděl.

3.3 Návrh API komunikce s ostaními aplikacemi

3.3.1 Komponenty OS Android

V následující části představím komponenty OS Android nutné k pochopení meziaplikační komunikace.

3.3.1.1 Aktivita

Aktivita [40] je komponent, který se v OS Android stará o vytvoření obrazovky a její zobrazení uživateli. Aktivita sama o sobě může a nemusí obsahovat uživatelské rozhraní. Pokud obsahuje uživatelské rozhraní zajišťuje interakci uživatele a aplikace. V opačném případě aktivita zpravidla obsahuje fragment popřípadě fragmenty (viz níže). V případě, že aktivita obsahuje fragmenty, zajišťuje jejich zobrazení a takto obsažené fragmenty mohou s aktivitou komunikovat. V případě, že aktivita obsahuje fragmentů více, slouží pro komunikaci mezi nimi, pokud je to třeba.

Aktivity vytváří a spouští Android OS. Aktivity podléhají životnímu cyklu obr 3.8, což je sada metod které volá Android OS v přesně stanoveném pořadí za daných podmínek. Vývojář má možnost přetěžováním těchto metod ovlivnit, jak se v daný okamžik bude aktivita chovat.

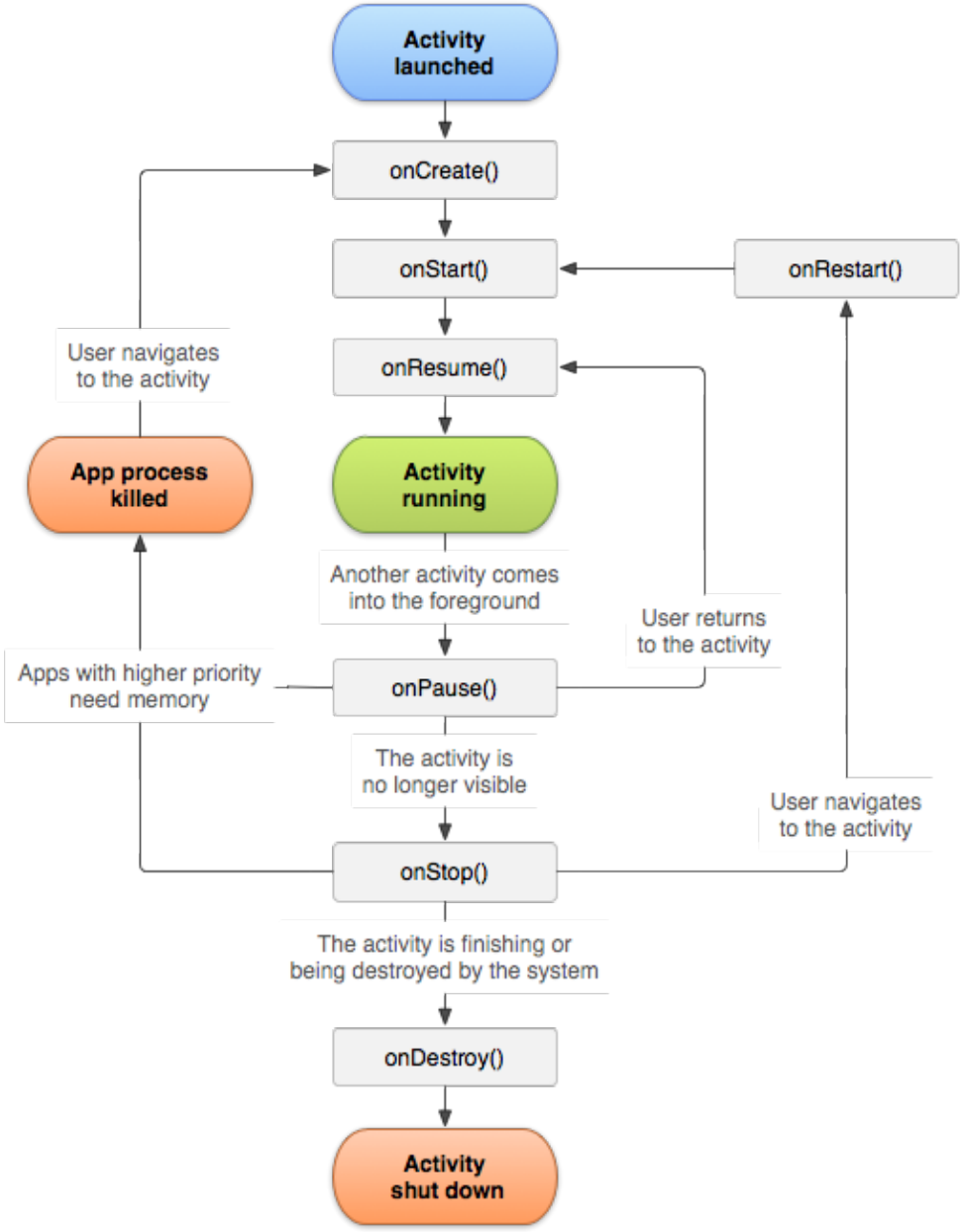
3.3.1.2 Fragment

Fragment je komponenta, která zapouzdřuje konkrétní chování nebo kus uživatelského rozhraní. Fragment také podléhá životnímu cyklu, který je odlišný od životního cyklu Aktivity. [41]

Fragmenty vytváří jak Android OS, tak vývojář dle konkrétní potřeby. Aby byl fragment funkční musí být součástí aktivity nebo jiného fragmentu. V aplikaci AEW je takto implementována většina obrazovek.

3.3.1.3 Služba (Service)

Služba, je komponenta OS Android sloužící k provedení práce na pozadí. Android OS poskytuje různé typy služeb, z hlediska toho, kdy a jak má být



Obrázek 3.8: Diagram životního cyklu Aktivity [40]

práce na pozadí provedena. Některé specifické typy služeb budou diskutovány níže. Službu nespouští vývojář přímo, opět je v režii Android OS a podléhá životnímu cyklu podobně jako aktivita a fragment. [42]

3.3.1.4 Intent

Podle [43] intent zapouzdřuje abstraktní zprávu v OS Android. Má hned několik využití. Slouží ke spouštění výše popsaných komponent jako aktivit a služeb. Dále ho lze použít k posílání dat jak v rámci aplikace, tak napříč Android OS. Intent se standardně dělí na dva typy:

- explicitní - má přesně určenou komponentu jako příjemce toho intentu. Příjemce je plně kvalifikován úplným jménem aplikace (package) a třídy (obsahuje i jméno aplikace).
- Implicitní - nemá přesně určenou komponentu. Ostatní komponenty se mohou v Android OS registrovat pomocí intent filtrů, čímž dávají najevo, že umějí zpracovat intent který odpovídá danému filtru. Z pohledu AEW je z hlediska implicitního intentu nejdůležitější položka ACTION - což je řetězec, který značí jakou akci má intent vyvolat. V Android OS existuje celá řada předdefinovaných akcí a vývojáři mají možnost nadefinovat si akce vlastní. Komponenta, která je pro odpovídající akci zaregistrovaná, tento intent obdrží. Toto se dá využít, jak ke startování aktivit a služeb, tak k posílání dat.

3.3.1.5 Broadcast Receiver

Jednoduchá komponenta, jež má za úkol přijímat intenty a reagovat na ně.

3.3.2 Meziaplikační komunikace v OS Android

Meziaplikační komunikace v OS Android neprobíhá, tak jak na jiných platformách např. pomocí rour či REST API, ale probíhá pomocí intentů. Komunikace se liší podle toho, jestli do ní vstupuje uživatel či nikoliv.

3.3.2.1 Komunikace vyžadující interakci uživatele

Tady je v podstatě jen jedna možnost. Aplikace požadující danou funkcionalitu musí spustit aktivitu aplikace, která ji poskytuje. Tady je k diskuzi jestli aplikace svázat těsně, tedy pomocí explicitního intentu, nebo aktivitu, která poskytuje požadovanou funkcionalitu zaregistrovat v systému na předem danou akci.

Toto druhé řešení má výhodu v tom, že pokud přijde jiná aplikace, která bude poskytovat stejnou funkcionalitu, dostane uživatel možnost volby.

Pro takto vystavené aktivity aplikací AEW je tedy použita tato možnost.

3.3.2.2 Komunikace nevyžadující interakci uživatele

Tato komunikace probíhá pomocí jedné z poskytovaných podtříd třídy Service popsané výše a tady je výběr daleko bohatší.

Intent Service Původní záměr bylo použití IntentService [44]. IntentService je potomek třídy Service, který postupně zpracovává příchozí intenty. Má velmi jednoduchou a přímočarou implementaci, která by k tomuto účelu byla ideální. Bohužel od Androidu 8 začala platit omezení na spouštění takovýchto služeb, pokud se aplikace, do které služba patří nevyskytuje napopředí. [45] Z tohoto důvodu jsem se musel uchýlit k jinému řešení.

Bound Service Bound Services fungují tak, že k nim mohou připojovat klienti a komunikovat s nimi přes Java rozhraní. Služba se spustí, když se k ní připojí první klient a naopak se ukončí, když k ní nejsou připojení žádní klienti. Na tyto služby se nevztahují žádná omezení na spouštění a běh jako na služby odvozené od IntentService. Pro komunikaci mezi aplikacemi lze použít dvě metody komunikaci přes Android A.I.D.L a komunikaci přes Messenger.

Komunikace přes Messenger vystavuje rozhraní pro komunikaci pomocí zpráv (Message). Do zprávy je možné dodat číselný kód, který určuje co se má stát a dále je ke zprávě možné připojit kolekci s daty ve formátu klíč-hodnota.

Komunikace přes A.I.D.L vyžaduje nadefinovat rozhraní pomocí jazyka A.I.D.L a toto rozhraní integrovat do klientské aplikace. Klientská aplikace pak se službou komunikuje přes toto rozhraní. [46]

Rozhodl jsem se využít první variantu, která je jednoduší a nevyžaduje striktní vazbu pomocí rozhraní. Komunikace je pak podobná jako u zasílání dat aktivitám. Komunikace ze služby směrem ven probíhá pomocí implicitního intentu, který zachytává BroadcastReceiver. Akce, kam se posílají výstupní data, je součástí obdržené zprávy ve službě.

3.4 API Poskytované aplikací AEW

V této sekci popíšu API poskytované aplikaci AEW. Nejprve popíši interakci s exportovanými aktivitami a poté interakci s exportovanou službou.

3.4.1 Interakce s exportovanými aktivitami

Vždy bude uvedena komponenta, jež má nastarost zpracování, dále spouštěcí intent a jeho data ve formátu klíč - datový typ - popis, (klíč v závorce znamená nepovinný parametr), dále pak možné výsledky a případný výstupní intent ve formátu klíč - popis. Návratové kódy z Aktivit RESULT_OK a RESULT_CANCELED jsou standardizované v případě potřeby se další takovéto kódy odvozují od RESULT_FIRST_USER.

3.4.1.1 Podpis transakce

- Komponenta: SignTransactionActivity
- Spouštěcí intent:
  - implicitní, akce ”com.chmelva4.ethereumwallet.SIGN_TRANSACTION”
  - TO - String - adresa cílového účtu transakce
  - VALUE - String - hodnota přenášeného Etheru v denominaci WEI
  - DATA - String - serializovaná transakční data
  - (FROM) - String - adresa účtu z něhož má být transakce provedena
  - (NOTE) - String - poznámka pro uživatele
- Výsledky:
  - RESULT_OK
  - RESULT_CANCELED
  - RESULT_FIRST_USER + 1 - chyba autentizace uživatele na vstupu do aktivity
- Výstupní intent:
  - TXHASH - String - transakční hash

3.4.1.2 Podpis dat

- Komponenta: SignDataActivity
- Spouštěcí intent:
  - implicitní, akce ”com.chmelva4.ethereumwallet.SIGN_DATA”
  - DATA - String - data k podpisu
  - (FROM) - String - adresa účtu, který má být použit k podpisu
  - (NOTE) - String - poznámka pro uživatele
- Výsledky:
  - RESULT_OK
  - RESULT_CANCELED
  - RESULT_FIRST_USER + 1 - chyba autentizace uživatele na vstupu do aktivity
- Výstupní intent:
  - SIGNATURE - String - podepsaná data
  - ACCOUNT - String - adresa účtu použitá k podpisu

3.4.1.3 Autentizace pomocí Ethereum účtu

- Komponenta: EthereumAuthActivity
- Spouštěcí intent:
  - implicitní, akce "com.chmelva4.ethereumwallet.ETHEREUM_AUTHENTICATE"
  - (FROM) - String - adresa účtu, který má být použit k autentizaci
  - (NOTE) - String - poznámka pro uživatele
- Výsledky:
  - RESULT_OK
  - RESULT_CANCELED
  - RESULT_FIRST_USER + 1 - chyba autentizace uživatele na vstupu do aktivity
  - RESULT_FIRST_USER + 2 - autentizace pomocí ethereum účtu neúspěšná (vadný privátní klíč)
- Výstupní intent:
  - ACCOUNT - String - adresa účtu použitá k autentizaci

3.4.2 Interakce s exportovanou službou

Zde je komponenta jenom jedna a jedná se o třídu PublicService. K této třídě je potřeba provést navázání (binding) pomocí explicitního intentu. Po úspěšném navázání je obdržen Messenger objekt pomocí kterého lze komunikovat s navázanou službou. Pro komunikaci se pak vytváří Message, jež má parametr WHAT (typu int). Tento parametr určuje, co má služba provést. K této Message lze připojit Bundle (objekt do kterého lze ukládat hodnoty ve formátu klíč-hodnota, podobně jako do Intentu) s očekávanými vstupnímy daty. Popis těchto dat, je ve stejném formátu jako u výše uvedených aktivit.

3.4.2.1 Získání účtů

- hodnota parametru WHAT:1
- Vstupní data:
  - RECEIVER - String - akce na kterou se mají poslat výsledná data
- Výstupní intent:
  - ACCOUNTS - ArrayList<String>- pole s adresami evidovaných účtů
  - NAMES - ArrayList<String>- pole s názvy evidovaných účtů

3.4.2.2 Získání účtů

- hodnota parametru WHAT:2
- Vstupní data:
  - RECEIVER - String - akce na kterou se mají poslat výsledná data
  - DATA - String - data jež byla podepsána
  - SIGNATURE - String - podpis k ověření
  - ACCOUNT - String - adresa účtu použitá k podpisu dat
- Výstupní intent:
  - RESULT - boolean - podpis je platný, či nikoliv

3.5 Technika Dependency Injection

Pří návrhu tříd je v hojném počtu využita technika Dependency Injection. (Dependecy Injection není návrhový vzor jako takový, nýbrž se jedná o aplikaci návrhového vzoru Strategy). Podle [47] je Dependedency Injection programovací technika, která odprostí třídu od jejích závislostí. Jinak řečeno třída, která tuto techniku používá, si nevytváří instance svých závislostí sama o sobě, ale dostává všechny instance závislostí zvenčí. O to se postará buď programátor sám, nebo použije některý z frameworků, který to obslouží. Použití Dependency Injection má následující výhody:

- Přispívá k dobré testovatelnosti.
- Přispívá k lepší udržovatelnosti kódu, protože závislosti u tříd nejsou skryté.
- Odděluje interface od implementace, čímž umožňuje implementaci jednoduše vyměnit.

Dependency Injection má různé formy, teď představím ty nejběžnější a uvedu je do kontextu mobilní aplikace AEW.

3.5.1 Konstruktorová DI

Jedná se o první a asi nejběžnější formu DI. Třída své závislosti obdrží v konstruktoru. Dá se samozřejmě použít tam, kde má programátor nebo framework kontrolu nad vytvářenými objekty. V aplikace AEW jsou takto navržené ViewModely, Repozitáře a Javascript Services.

3.5.2 DI do členů třídy (Field)

Jedná se o DI, kde závislosti jsou injektovány do členů instance třídy. To znamená, že instance třídy volá nějáký objekt, který nazvu injektor. Tento objekt tuto třídu zná a ví jaké má do ní injektovat závislosti. Třída, do které je injektováno, pak sama zavolá Injector.inject.

V aplikaci AEW se takto injektují komponenty OS Android, které mají povinný prázdný konstruktor, protože je vytváří systém a jedná se o aktivity, fragmenty a služby.

3.5.3 Metodová (Setter) DI

Třída své závislosti obdrží pomocí zavolání metody k tomu určené. Toto se využívá pokud třídy potřebují nainjektovat sami sebe do některé své závislosti. Toto by sice šlo udělat v konstruktoru, ale v konstruktoru ještě není objekt plně inicializovaný, a je to nevhodné.

V aplikaci AEW se takto injektují třídy zajišťující komunikaci mezi Javou a Javascriptem. Tyto třídy obsahují interface na komunikaci z Javascriptu do Javy a injektují sami sebe do tohoto enginu. Zároveň ale tento engine potřebují ke svému chodu, takže instanci obdrží v konstruktoru, ale sami sebe pak injektují do enginu v metodě k tomu určené.

Kapitola 4

Implementace a testování

4.1 Složení aplikace

4.1.1 Složení jádra aplikace

Jádrem aplikace jsou ty obrazovky, které jsou dostupné pro uživatele, když spustí aplikaci ze svého mobilního zařízení. Jedná se tedy o neexportovanou funkcionalitu aplikace AEW. Jádro se skládá ze dvou aktivit. `AuthenticationActivity` a `MainActivity`. `AuthenticationActivity` má za úkol prvotní autentizaci uživatele, kterou provádí delegací této autentizace na OS Android. Při úspěšné autentizaci jí uživatel vůbec neuvidí, protože je překryta autentizační obrazovkou Android OS. Zobrazí se pouze v případě neúspěšné autentizace.

`MainActivity` obsahuje vlastní aplikaci. Obrazovky aplikace jsou realizované pomocí fragmentů, které se mění v této aktivitě. K tomu jsem použil knihovnu Navigation, jež je součástí sady pro vývojáře Android Jetpack.

4.1.2 Exportovaná funkcionalita

Exportovaná funkcionalita je implementována dle předchozího popisu návrhu API. Jedná se o tří exportované aktivity - `SignTransactionActivity`, `SignDataActivity` a `EthereumAuthActivity`. Tyto aktivity je možné spouštět pomocí daných implicitních intentů. Dále aplikace exportuje službu `PublicService`.

4.2 Implementace jednotlivých vrstev aplikace

4.2.1 Implementace komponent prezentační vrstvy

Komponenty prezentační vrstvy jsem implementoval pomocí dědičnosti od základních komponent systému OS Android. Tyto komponenty byly popsány v předchozí části. V tomto se jedná o běžnou praxi v rámci vývoje na OS Android.

```
@Dao
public interface WalletsDao {

    @Insert
    Completable insertItem(WalletRoom item);

    @Update
    Completable updateItem(WalletRoom item);

    @Delete
    Completable deleteItem(WalletRoom item);

    @Query("Select * from wallets")
    List<WalletRoom> getAll();

    @Query("Select * from wallets Where id = :id")
    WalletRoom getById(int id);
```

Obrázek 4.1: Ukázka použití knihovny Room

4.2.2 Implementace komponent business vrstvy

Třídy business vrstvy jde rozdělit do dvou částí.

První část tříd slouží k realizaci aplikační logiky. Jedná se o repozitáře a další pomocné třídy. Tyto třídy jsou použité jako Singleton. Toto za mě obstarala knihovna Dagger 2.

Druhá část jsou pak entitní třídy nesoucí data. Jsou jimy třeba třídy `Account` nebo `Wallet`. Tato vrstva agreguje data z lokální databáze a blockchainu a stará se o šifrování a dešifrování dat.

4.2.3 Implementace komponent datové vrstvy

Datovou vrstvu lze rozdělit na dvě části.

První část má na starosti lokální databázi zařízení. Druhá část se skládá z Javascriptového rozhraní aplikace.

K implementaci lokální databáze jsem použil knihovnu Room z vývojové sady Android Jetpack. Knihovna poskytuje úroveň abstrakce nad SQLite databází. [48]. Umožňuje definovat schéma databáze pomocí tříd v jazyce Java a následně definovat rozhraní, které mají splňovat DAO objekty. Knihovna se pak postará o doplnění implementace. Přesto se nejedná o plný objektově-

relační mapper, protože je nutnost dotazy nad databází vytvořit ručně viz obrázek 4.1.

Implementace Javascriptové části aplikace má dvě části Javovou a Javascriptovou. Javascriptová část aplikace, žije uvnitř Webview, jak bylo popsáno v druhé části práce.

Třídy této části opět lze rozdělit na třídy realizující aplikační logiku a entitní třídy. Platí pro ně stejná pravidla jako pro třídy business části.

4.3 Komunikace mezi jednotlivými vrstvami aplikace

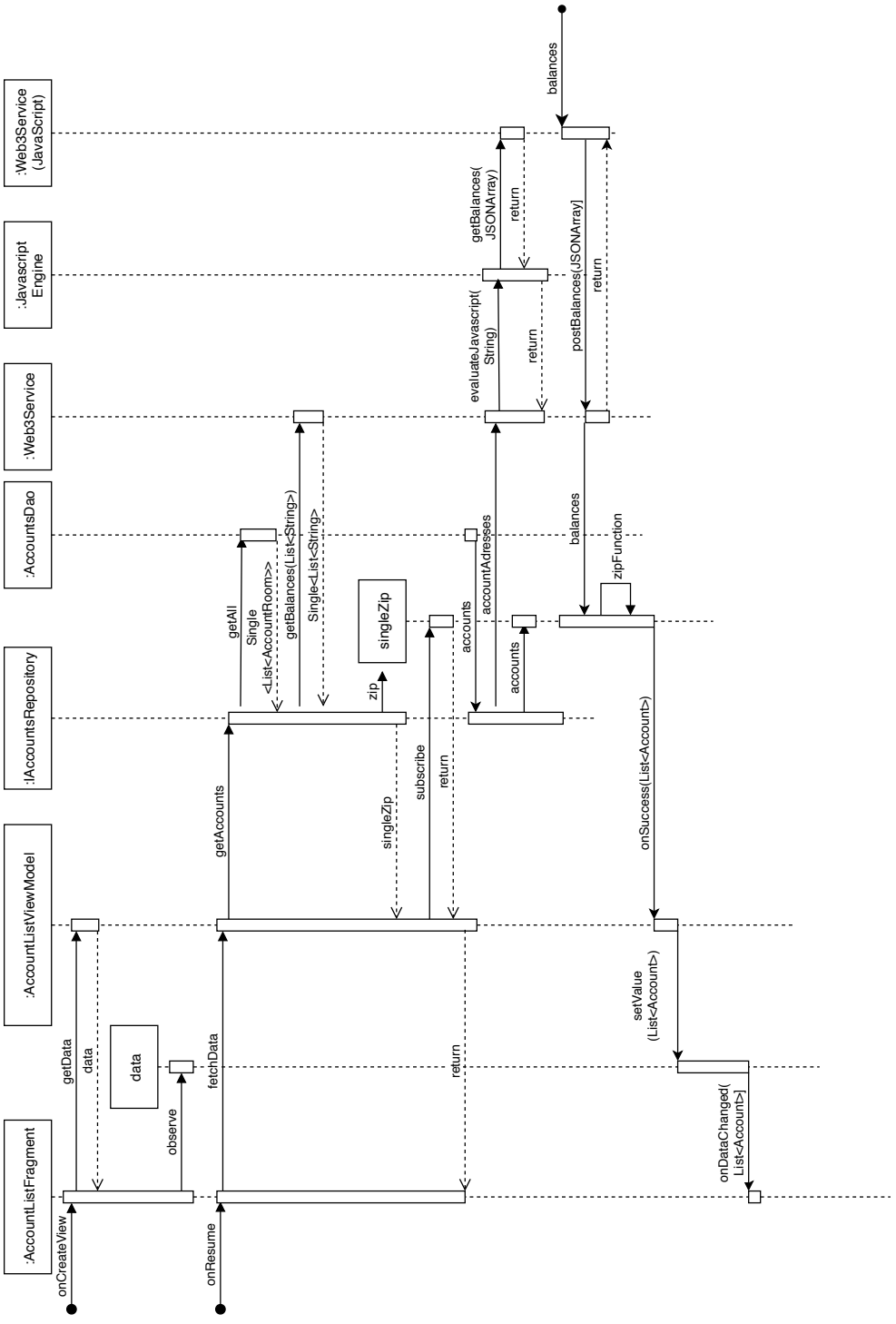
Na komunikaci mezi jednotlivými vrstvami aplikace jsem použil knihovnu RxJava 2. Tato knihovna umožňuje asynchronní komunikaci pomocí návrhového vzoru Observer.

Jedná se behaviorální návrhový vzor, který slouží pro šíření událostí. Má dva komponenty. První komponent je `Observer`, ten se registruje na přijímání změn stavu objektu, který chce monitorovat. Tento druhý objekt se nazývá `Observable` (někdy lze najít `Subject`). Tento komponent pak při změně svého stavu upozorní všechny u sebe zaregistrované Observers. [49]

Nejlépe půjde pochopit jakým způsobem je komunikace v aplikaci zajištěná pomocí příkladu. Uvedu tedy příklad komunikace při zobrazení seznamu účtů v aplikaci.

Diagram 4.2 můžeme rozdělit vodorovně na dvě poloviny voláním metody `subscribe` na objektu `singleZip`. Vrchní polovina probíhá synchroním způsobem a jejím výsledkem je Observer typu Single. Jedná se o komponetu knihovny RxJava, která má za úkol jednorázově obdržet data. V momentě, kdy je na tomto objektu zavolána metoda subscribe, začne se provádět jeho naprogramovaná logika (získání účtů z lokální databáze, poté získání zůstatků daných z blockchainu a jejich výsledná kombinace) asynchroním způsobem (spodní polovina).

Data se získávají mimo hlavní vlákno a aplikace tak není blokovaná. V tomto konkrétním případě se z databáze získá seznam účtů a následně se jejich Ethereum adresy pošlou do javascriptové části aplikace, aby se mohly z blockchainu získat zůstatky odpovídajících účtů. Tyto zůstatky se pošlou zpátky do příslušného observeru a zkombinují s již získanými informacemi o účtech (adresa, jméno). Tyto informace obdrží observer v příslušném viewmodelu, odkud jsou dále propagovány do fragmentu a zobrazeny uživateli.



Obrázek 4.2: Sekvenční diagram získání seznamu účtů

```
const seed = bip39.mnemonicToSeed(mnemonic)
const root = hdkey.fromMasterSeed(seed);
const addrnode = root.derive(`m/44'/60'/0'/0/0`);
// private key for new ethereum account
var privateKey =  Buffer.from(addrnode._privateKey).toString('hex');
```

Obrázek 4.3: Ukázka získání privátního klíče Ethereum účtu pomocí mnemonicé fráze

4.4 Implementace Dependency Injection pomocí knihovny Dagger 2

K implementaci techniky dependency injection, která byla představena v předchozí části, jsem použil knihovnu Dagger 2. Jedná se o plně statický compile-time framework na realizaci této praktiky. [50] Základem toho frameworku jsou tři věci.

Framework umožňuje anotovat konstruktory, členy tříd a metody anotací `@Inject` a sám se pak postará o injektování potřebných závislostí.

V případě, že dané závislosti potřebují zvláštní formu inicializace, nebo je-li potřeba navázání konkrétní implementace na dané rozhraní, musí vývojář z těchto závislostí vytvořit modul. Jedná se o třídu, nebo interface anotovanou anotací `@Module`. V tomto modulu pak poskytuje konkrétní závislosti formou metod, které mají jako návratovou hodnotu požadovanou závislost a jsou anotované anotací `@Provides`. Framework dále u tříd umožňuje nastavit rozsah. Třídy aplikační logiky pak jsou anotovány jako `@Singleton`. Dagger obstará inicializaci takto anotovaných tříd pomocí tohoto návrhového vzoru. Jedná se o vytvářecí návrhový vzor, který zajišťuje, že v aplikaci existuje pouze jediná instance dané třídy. [51]

Poslední důležitou součástí tohoto frameworku je komponent. Jedná se opět o třídu nebo rozhraní, která se skládá z modulů a umí injektovat konkrétní závislosti, na základě jakých modulů se skládá. Aplikace AEW obsahuje jediný komponent, který se inicializuje při startu aplikace.

4.5 Použití HD Wallet v aplikaci AEW

V aplikaci AEW má uživatel dvě možnosti jak založit novou peněženku:

1. nechá si jí vygenerovat aplikací,

2. chce jí importovat, tzn. zadá menmonickou frázi ručně.

V prvním případě mu menmonickou frázi vygeneruje knihovna bip39 - Javascriptová implementace BIP39 (standard představený v první kapitole).

```
const seed = bip39.mnemonicToSeed(mnemonic)
const root = hdkey.fromMasterSeed(seed);
const masterPrivateKey = root.privateKey.toString('hex');
const walletNode = root.derive(`m/44'/60'/0'/0`);
var walletExtendedKey = walletNode.privateExtendedKey
```

Obrázek 4.4: Ukázka získání privátního klíče konkrétní peněženky pomocí mnemonicé fráze

```
var walletNode = hdkey.fromExtendedKey(walletExtendedKey)
var accountNode = w2.derive(`m/0`)
var privateKey =  Buffer.from(accountNode._privateKey).toString('hex');
```

Obrázek 4.5: Ukázka získání privátního klíče Ethereum účtu z privátního klíče peněženky

Tuto knihovnu jsem mohl využít díky integraci Javascriptu v aplikaci. V druhém případě je mnemonická fráze získána od uživatele.

Z této fráze se vygeneruje privátní klíč k inicializaci HD Wallet popsané dříve. O to se stará knihovna hdkey, která implementuje BIP32 (HDWallet strukturu). A nyní nastává otázka, jak s touto frází (klíčem) naložit?

Prvotní řešení spočívalo v tom, uložit si při vytváření peněženky menmonickou frázi (samozřejmě šifrovanou) a při vytváření účtů získat privátní klíč odvozením klíče získaného pomocí této fráze. Toto lze vidět na obrázku 4.3 Při vytváření účtů je použitá plná cesta, tak jak byla popsaná dříve.

Prvotní řešení má jednu výhodu - vzhledem k tomu, že je pro toto řešení nutné si uložit mnemonickou frázi, lze ji později ukázat uživateli.

Prvotní řešení je nevýhodné z hlediska bezpečnosti. V případě, že by došlo k narušení bezpečnosti aplikace a odcizení této fráze, je kompromitována nejen celá peněženka jako entita v aplikaci AEW, ale i všechny další peněženky pro všechny další kryptoměny, které lze získat pomocí této fráze (což je důležité z hlediska importu).

Lepší řešení spočívá v tom si při vytvoření peněženky uložit Extended PK uzlu validního pouze pro konkrétní Ethereum peněženku. Toto lze vidět na obrázku 4.4. Poté při vytváření účtu z uloženého extended PK peněženky lze získat uzel struktury HDWallet a pomocí něho vygenerovat nový Ethereum účet. Toto lze vidět na obrázku 4.5. Jedná se o první účet v dané peněžence. Jinak by obrázek obsahoval `w2.derive('m1')` atd.

Druhé řešení má přesně opačný důsledek než řešení předchozí. To znamená není k dispozici mnemonická fráze, ale v případě kompromitace aplikace, je kompromitována jen ta nejmenší možná část ka6d0 peněženky.

V aplikaci je použito druhé řešení.

4.6 Naimplementovaná funkcionalita

- evidence peněženek
- evidence účtů
- evidence Ethereum transakcí
- evidence adresářových položek
- podpis Ethereum transakcí
- zasílání Etheru pomocí Ethereum transakcí
- podpis dat pomocí Ethereum účtu
- autentizace pomocí Ethereum účtu
- ověření platnosti Ethereum podpisu
- získání seznamu Ethereum účtů pro ostatní aplikace

4.7 Neimplementovaná funkcionalita

Níže uvedená funkcionalita nebyla po dohodě s vedoucím práce do prototypu aplikace AEW implementována. S její implementací se počítá až do následující verze.

- evidence budoucích transakcí
- převod Etheru pomocí budoucích transakcí
- stažení transakcí do zařízení

Implementace logiky budoucích transakcí by musela využít služby Ethereum Alarm Clock diskutované v první kapitole a měla by následující postup:

1. Zprovoznit a nakonfigurovat službu Ethereum Alarm Clock na testovacím blockchainu použitém pro vývoj aplikace.
2. Naimplementovat smart kontrakt (`PendingTransactionContract`), který bude sloužit jako úložiště pro budoucí transakce a umožní jejich obsluhu.
3. Integrace služby Ethereum Alarm Clock a výše uvedeného kontraktu do aplikace AEW.

Funkcionalita by poté vypadala, tak, že dojde-li k přenosu Etheru většímu, než je stanovený `TransferThreshold` pro daný účet a má být vytvořena budoucí transakce, stanou se dvě věci.

V první řadě je nová budoucí transakce uložena do `PendingTransaction` kontraktu.

Poté se pomocí služby Ethereum Alarm Clock objedná volání funkce, která uskuteční daný přenos za stanovenou dobu. Hodnoty které jsou k tomuto nutné z hlediska účtů už jsou v aplikaci funkční a připravené. Čtení budoucích transakcí pak bude probíhat ze smart kontraktu `PendingTransactionContract`.

Implementace stahování transakcí by vypadala následujícím způsobem. Uživatel zvolí stáhnout transakce do zařízení pro daný účet. Tato akce spustí službu typu Foreground Service (typ android služby určený pro dlouhotrvající práci, kterou OS Android neukončí ani při upozadění aplikace). Aplikace pomocí Web3 knihovny projde určitý počet bloků zpětně a stáhne do zařízení všechny transakční hashe, které se vyskytují v těchto blocích a mají jako zdrojový účet označený ten, pro který probíhá tato akce. Hashe jsou následně uloženy do databáze.

4.8 Použité prostředky pro vývoj

4.8.1 Android Studio

Jedná se o oficiální vývojové prostředí nativních aplikací na OS Android založené na IntelliJ. Poskytuje celou řadu funkcionality nutné pro vývoj Android aplikací včetně editoru pro návrh uživatelského rozhraní, editoru navigace v aplikaci a správu grafických prvků, které se balí do výsledné aplikace [52].

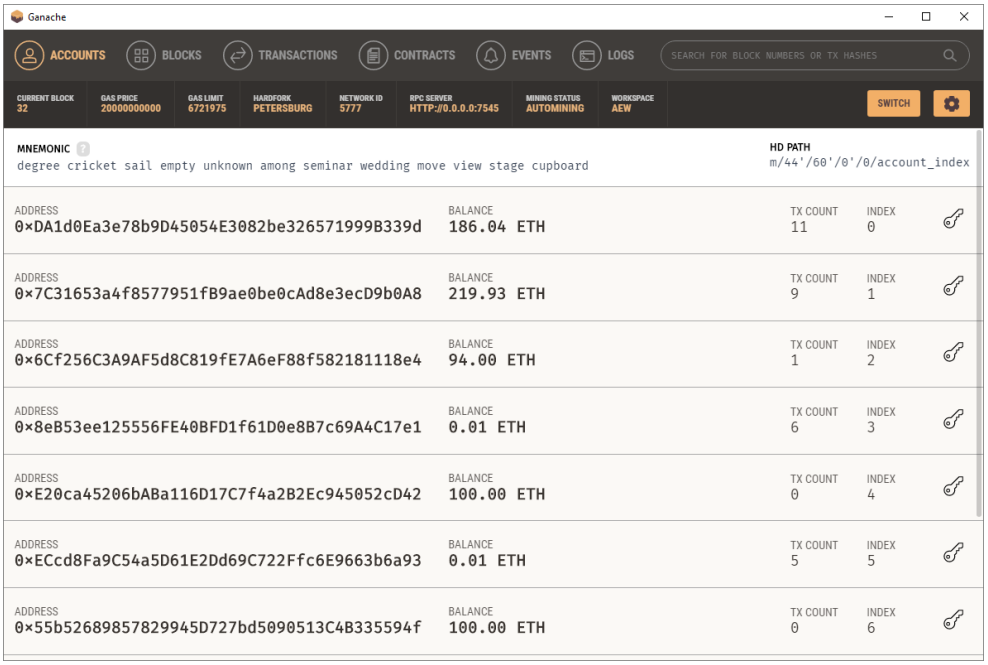
4.8.2 Mobilní telefon

Ačkoliv dnes existují různé emulátory mobilních zařízení, aplikaci jsem vyvíjel a testoval na svém mobilním telefonu Sony Xperia XZ Premium s OS Android 9. Poté otestoval i na mobilním telefonu Samsung Galaxy S5 s OS Android 6 a Sony Xperia Z5 compact s OS Android verze 7.

Vývoj a testování na fyzickém zařízení je z mého úhlu pohledu lepší, protože při vývoji na emulátoru se může stát, že nebudou zachyceny některé nepříjemné interakce při zadávání vstupů do telefonu se softwarovou klávesnicí.

4.8.3 Testovací blockchain Ganache

Jedná se o osobní Ethereum blockchain od společnosti Truffle. Umožňuje vývoj a testování decentralizoavných aplikací bez nutnosti vlastnění opravdového nebo testovacího Etheru.



Obrázek 4.6: Testovací blockchain Ganache

Jde o nepostradatelnou pomůcku při vývoji na Ethereum blockchain. Jelikož aplikace AEW je zatím pouze prototyp, nezkoušel jsem jí testovat ani na testovací síti Ethereum a veškerá funkcionalita byla vyvíjena a testována vůči tomuto blockchainu.

Grafické rozhraní tohoto blockchainu lze vidět na obrázku 4.6. Při použití aplikace je nutné, aby mobilní telefon a tento blockchain byly v jedné síti. V horní části lze vidět mnemonickou frázi, která umožňuje importování účtů do aplikace AEW pomocí importu peněženky. Jedná se zde opět o účty vygenerované pomocí HD Wallet.

4.9 Testování aplikace

4.9.1 Jednotkové testy

Jednotkové testy jsou typem automatických testů. Mají za úkol otestovat jednotlivé jednotky kódu. V objektově orientovaném programování jsou jimi nejčastěji jednotlivé metody. Důležitým prvkem tohoto testování je, že daná jednotka je testovaná v izolaci. [53]

Toho lze docílit několika způsoby. Jedním z nich je technika mockování. Jedná se o nahrazení reálných objektů objekty testovacími, které pouze napodobují jejich chování pro účely testu. [54]

K testování pomocí jednotkových testů jsem použil testovací framework

JUnit V aplikaci AEW jednotkovými testy zcela pokryl business vrstvu aplikace a databázovou část datové vrstvy. Zbytek aplikace jsem testoval manuálně.

4.9.1.1 Testy business vrstvy

Jedná se o lokální jednotkové testy. Lokální testy jsou takové testy, které nepotřebují běžet na mobilním zařízení nebo na emulátoru a mohou být spuštěny na počítači. [55]

K mockování závislostí jsem použil knihovnu Mockito. Jelikož třídu `KeyStoreHelper`, která zapozdřuje práci s AndroidKeyStore pomocí toho frameworku namockovat nešlo, vytvořil jsem speciální třídu pro testování `TestKeyStoreHelper`, která pouze simuluje její chování.

Testy mají klasickou strukturu. Nejprve se nastaví chování mockovaných objektů. Poté se volá testovaná metoda. V poslední části probíhá ověření výsledků.

4.9.1.2 Testy databáze

Jedná se o instrumentační jednotkové testy. Tyto testy musí běžet buď na fyzickém zařízení, nebo na emulátoru. [55]

Databáze, která je použitá během testování existuje pouze v paměti zařízení po dobu testu. K těmto testům nebylo potřeba žádných jiných závislostí, kromě testované databáze a frameworku JUnit.

4.9.2 Manuální testování

Manuální testování je proces během kterého probíhá testování funkcionality aplikace testerem, tak jak by ji používal reálný uživatel. [56]

Část vystavované funkcionality aplikace AEW sama nepoužívá. Musel jsem proto vytvořit jednoduchou aplikaci, pomocí které lze tuto funkcionalitu otestovat. Aplikace s názvem AEW-testapp je k dispozici na přiloženém CD.

Aplikaci jsem z velké části testoval manuálně. Níže uvedu pár příkladů scénářů z toho testování. Tyto scénáře mohou posloužit jako základ pro tvorbu scénářů na případné uživatelské testování.

Vstup do aplikace Uživatel spustí aplikaci a je vybídnut k autentizaci. Tuto autentizaci zruší. Aplikace mu zobrazí úvodní obrazovku. Poté zvolí přihlásit a autentizuje se pomocí validních údajů. Zkontroluje, že má zobrazenu obrazovku se seznamem účtů.

Založení peněženky Uživatel se nachází v seznamu peněženek. Zvolí založit peněženku. Do zobrazeného dialogu vyplní její název. Dialog lze potvrdit pouze pokud název není prázdný. Dialog potvrdí. Aplikace mu zobrazí dialog s mnemoickou frází peněženky. Tento dialog potvrdí 10 stisky na tlačítko OK. V seznamu peněženek se objeví nová peněženka.

Import peněženky Uživatel se nachází v seznamu peněženek. Zvolí importovat peněženku. Do zobrazeného dialogu vyplní její název a její mnemonickou frázi. Dialog lze potvrdit pouze pokud název není prázdný a fráze obsahuje 12 slov oddělených mezerou. Dialog potvrdí. V seznamu peněženek se objeví nová peněženka.

Smazání peněženky Uživatel se nachází v seznamu peněženek. V tomto seznamu se nachází peněženka P, která má 1 účet a peněženka Q, která má 0 účtů. Uživatel se pokusí zvolit smazat peněženku P, to mu aplikace nedovolí, protože obsahuje účet. Uživatel zvolí smazat peněženku Q a potvrdí dialog, který mu aplikace zobrazí. Peněženka zmizí se seznamu.

Založení účtu Uživatel se nachází v seznamu účtů a aplikace obsahuje peněženku P. Zvolí založit účet. Vybere si v nabídnutém dialogu peněženku P. Aplikace mu založí nový účet pomocí této peněženky. Uživatel zkontroluje, že se účet objevil v seznamu a má stejnou barvu jako peněženka P.

Přenos Etheru Uživatel se nachází na obrazovce detailu účtu a není autentizovaný. Pomocí tlačítka zvolí zaslání Etheru. Aplikace mu zobrazí dialog. Uživatel do něj vyplní údaje a zkontroluje, že dialog lze potvrdit pouze při zadání validních údajů. Uživatel potvrdí dialog. Dále mu aplikace zobrazí autentizační obrazovku. Tam se uživatel autentizuje a pokračuje na obrazovku podpisu transakce. Uživatel zkontroluje, že obrazovka obsahuje správné údaje a následně zvolí podepsat transakci. Aplikace ho vrátí na obrazovku detailu účtu. Nakonec uživatel na obrazovce seznamu transakcí pro daný účet zkontroluje, že přibyla nová transakce s odpovídajícími údaji.

Poté uživatel zopakuje celý scénář znovu s tím rozdílem, že transakci zruší. Zkontroluje, že aplikace ho na tuto skutečnost upozorní a že v seznamu transakcí nepřibyla nová transakce.

Získání seznamu účtů pomocí testovací aplikace V aplikaci AEW se nacházejí dva účty a jeden z nich má nastavený alias. Uživatel spustí testovací aplikaci a zvolí možnost "získat účty". Testovací aplikace mu získané účty zobrazí. Uživatel zkontroluje, že byly vráceny obě odpovídající adresy a jeden odpovídající alias účtu.

Závěr

V této práci jsem představil technologii blockchain a platformu Ethereum. Jedná se o novou platformu a technologii, která se neustále vyvíjí.

Platforma Ethereum je využitelná k autentizaci uživatelů i zařízení, jak pomocí vestavěného mechanismu externích účtů, tak pomocí smart kontraktů. V rámci autentizace pomocí smart kontraktů ještě neexistují rozšířené a dotažené standardy, které bychom mohli použít.

Práce splnila všechny cíle, které byly stanoveny v úvodu práce. Jejím výstupem je funkční prototyp aplikace AEW. Byla provedena jeho analýza, návrh, implementace a následné testování. Dokumentace prototypu je k dispozici na přiloženém CD.

Lze očekávat, že do další verze aplikace AEW bude jako první zahrnuta funkcionalita budoucích transakcí, které byly představeny v této práci a dále funkcionalita stahování transakcí do zařízení.

V dalších krocích je možné do aplikace přidat podporu ERC20 Tokenů, které byly popsány v první kapitole a integrovat API, které umožní nákup Etheru a převádění zobrazení hodnot Etheru v jednotlivých měnách.

Je znovu důležité zdůraznit, že se jedná o první prototyp aplikace AEW a tomu odpovídá zahrnutá funkcionalita. Diplomovou prací moje účast na tomto projektu zdaleka neskončila, spíše začala.

Literatura

[1] Blockgeeks: *What is Blockchain Technology? A Step-by-Step Guide For Beginners.* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://blockgeeks.com/guides/what-is-blockchain-technology/

[2] Rouse, M.: *blockchain.* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://searchcio.techtarget.com/definition/blockchain

[3] Lisk: *What is Blockchain?* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://lisk.io/academy/blockchain-basics/what-is-blockchain

[4] Bikramaditya Singhal, P. S. P., Gautam Dhameja: *Beginning Blockchain.* Apress, 2018, ISBN 978-1-4842-3444-0, str. 8.

[5] Sloane Brakeville, B. P.: *Blockchain basics: Introduction to distributed ledgers.* [online], 2019, [cit. 2019-05-03]. Dostupné z: https://developer.ibm.com/tutorials/cl-blockchain-basics-intro-bluemix-trs/

[6] FORTNEY, L.: *Blockchain, Explained.* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://www.investopedia.com/terms/b/blockchain.asp

[7] Blockgenic: *Different Blockchain Consensus Mechanisms.* [online], 2019, [cit. 2019-04-25]. Dostupné z: https://hackernoon.com/different-blockchain-consensus-mechanisms-d19ea6c3bcd6

[8] Bulkin, A.: *Explaining blockchain - how proof of work enables trustless consenzus.* [online], 2019, [cit. 2019-05-03]. Dostupné z: https://keepingstock.net/explaining-blockchain-how-proof-of-work-enables-trustless-consensus-2abed27f0845

[9] Jackson, R.: *Scalability is Blockchain's Biggest Problem But it Can Be Resolved.* [online], 2019, [cit. 2019-05-03]. Dostupné z: https://cryptoslate.com/scalability-is-blockchains-biggest-problem-but-it-can-be-resolved/

[10] Szabo, N.: *Smart Contracts*. [online], 1994, [cit. 2019-05-03]. Dostupné z: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[11] Crypttalker: *Top 10 Ethereum Use Cases – Public and Private*. [online], 2019, [cit. 2019-04-28]. Dostupné z: https://cryptalker.com/ethereum-use-cases/

[12] Khatwani, S.: *What Are Smart Contracts in Relation To Ethereum?* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://coinsutra.com/smart-contracts/

[13] Blockgeeks: *What is Ethereum? The Most Comprehensive Guide Ever!* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://blockgeeks.com/guides/ethereum/

[14] Heide, D. T.: *A Closer Look At Ethereum Signatures*. [online], 2018, [cit. 2019-04-25]. Dostupné z: https://hackernoon.com/a-closer-look-at-ethereum-signatures-5784c14abecc

[15] Hu, K.: *Ethereum account*. [online], 2018, [cit. 2019-04-25]. Dostupné z: https://medium.com/coinmonks/ethereum-account-212feb9c4154

[16] Murthy, M.: *Life Cycle of an Ethereum Transaction*. [online], 2017, [cit. 2019-04-25]. Dostupné z: https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e

[17] Kasireddy, P.: *How does Ethereum work, anyway?* [online], 2017, [cit. 2019-04-28]. Dostupné z: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369

[18] Dannen, C.: *Introducing Ethereum and Solidity*. Apress, 2017, ISBN 978-1-4842-2535-6, str. 59.

[19] Buterin, V.: *Vyper*. [online], 2017, [cit. 2019-05-3]. Dostupné z: https://vyper.readthedocs.io/en/v0.1.0-beta.9/

[20] REIFF, N.: *What is ERC-20 and What Does it Mean for Ethereum?* [online], 2019, [cit. 2019-04-27]. Dostupné z: https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/

[21] *ERC20 Token Standard*. [online], 2016, [cit. 2019-04-27]. Dostupné z: https://theethereum.wiki/w/index.php/ERC20_Token_Standard

[22] Fabian Vogelsteller, V. B.: *EIP 20: ERC-20 Token Standard*. [online], 2015, [cit. 2019-04-27]. Dostupné z: https://eips.ethereum.org/EIPS/eip-20

[23] Fabian Vogelsteller, T. Y.: *Key Manager*. [online], 2017, [cit. 2019-04-27]. Dostupné z: https://github.com/ERC725Alliance/erc725/blob/master/docs/ERC-725.md

[24] Vogelsteller, F.: *Proxy Account*. [online], 2017, [cit. 2019-04-27]. Dostupné z: https://github.com/ethereum/EIPs/issues/734

[25] Ethereum Name Service: *Introduction*. [online], 2019, [cit. 2019-04-28]. Dostupné z: https://docs.ens.domains/

[26] Ethereum Name Service: *eth Permanent Registrar*. [online], 2019, [cit. 2019-04-28]. Dostupné z: https://docs.ens.domains/contract-api-reference/.eth-permanent-registrar

[27] Merriam, P.: *Ethereum Alarm Clock 1.0.0 documentation*. Ethereum Name Service, [online], 2015, [cit. 2019-04-28]. Dostupné z: https://ethereum-alarm-clock-service.readthedocs.io/en/latest/introduction.html

[28] WUILLE, P.: *Hierarchical Deterministic Wallets*. [online], 2019, [cit. 2019-04-19]. Dostupné z: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[29] Marek Palatinus, P. R.: *Multi-Account Hierarchy for Deterministic Wallets*. [online], 2019, [cit. 2019-04-19]. Dostupné z: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

[30] Marek Palatinus, P. R.: *Mnemonic code for generating deterministic keys*. [online], 2019, [cit. 2019-04-19]. Dostupné z: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[31] Ethereum: *JSON RPC API*. [online], 2019, [cit. 2019-04-10]. Dostupné z: https://github.com/ethereum/wiki/wiki/JSON-RPC#json-rpc-api

[32] NodeJS: *C++ Addons*. [online], 2019, [cit. 2019-04-10]. Dostupné z: https://nodejs.org/api/addons.html

[33] Google: *WebView*. [online], 2019, [cit. 2019-04-20]. Dostupné z: https://developer.android.com/reference/android/webkit/WebView

[34] Google: *Building web apps in WebView*. [online], 2019, [cit. 2019-04-20]. Dostupné z: https://developer.android.com/guide/webapps/webview

[35] Google: *Android keystore system*. [online], 2019, [cit. 2019-04-23]. Dostupné z: https://developer.android.com/training/articles/keystore

[36] BIRCH, D.: *The Model-View-ViewModel Pattern*. [online], 2019, [cit. 2019-04-26]. Dostupné z: https://medium.com/@angellopozo/ethereum-signing-and-validating-13a2d7cb0ee3

[37] Babich, N.: *Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each*. Adobe, [online], 2019, [cit. 2019-04-30]. Dostupné z: https://theblog.adobe.com/prototyping-difference-low-fidelity-high-fidelity-prototypes-use/

[38] Google: *Lateral navigation*. [online], 2019, [cit. 2019-04-30]. Dostupné z: https://material.io/design/navigation/understanding-navigation.html#lateral-navigation

[39] Google: *Bottom Navigation*. [online], 2019, [cit. 2019-04-30]. Dostupné z: https://material.io/design/components/bottom-navigation.html

[40] Google: *Activity*. [online], 2019, [cit. 2019-04-26]. Dostupné z: https://developer.android.com/reference/android/app/Activity

[41] Google: *Fragment*. [online], 2019, [cit. 2019-04-26]. Dostupné z: https://developer.android.com/guide/components/fragments

[42] Google: *Services Overview*. [online], 2019, [cit. 2019-04-26]. Dostupné z: https://developer.android.com/guide/components/services

[43] Google: *Intent*. [online], 2019, [cit. 2019-04-26]. Dostupné z: https://developer.android.com/reference/android/content/Intent

[44] Google: *IntentService*. [online], 2019, [cit. 2019-04-24]. Dostupné z: https://developer.android.com/reference/android/app/IntentService

[45] Google: *Background Execution Limits*. [online], 2019, [cit. 2019-04-24]. Dostupné z: https://developer.android.com/about/versions/oreo/background

[46] Google: *Bound services overview*. [online], 2019, [cit. 2019-04-24]. Dostupné z: https://developer.android.com/guide/components/bound-services

[47] THORBEN, J.: *Design Patterns Explained – Dependency Injection with Code Examples*. Stackify, [online], 2019, [cit. 2019-04-24]. Dostupné z: https://stackify.com/dependency-injection/

[48] Google: *Room Persistence Library*. [online], 2019, [cit. 2019-05-01]. Dostupné z: https://developer.android.com/topic/libraries/architecture/room

[49] Poyias, A.: *Design Patterns - A quick guide to Observer pattern.* [online], 2019, [cit. 2019-05-03]. Dostupné z: https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2

[50] Google: *Dagger.* [online], 2019, [cit. 2019-05-02]. Dostupné z: https://google.github.io/dagger/

[51] Tutorialspoint: *Design Pattern - Singleton Pattern.* [online], 2019, [cit. 2019-05-02]. Dostupné z: https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

[52] Google: *Meet Android Studio.* [online], 2019, [cit. 2019-04-29]. Dostupné z: https://developer.android.com/studio/intro

[53] Software Testing Fundamentals: *Unit Testing.* [online], 2019, [cit. 2019-05-02]. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/

[54] Rouse, M.: *mock object.* [online], 2019, [cit. 2019-05-02]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/mock-object

[55] Google: *Build effective unit tests.* [online], 2019, [cit. 2019-05-02]. Dostupné z: https://developer.android.com/training/testing/unit-testing

[56] Bartlett, J.: *What is Manual Testing?* TestLodge, [online], 2018, [cit. 2019-05-02]. Dostupné z: https://blog.testlodge.com/what-is-manual-testing/

Příloha **A**

Seznam použitých zkratek

OS Operační systém

DI Dependency Injection

PK Privátní klíč

VK Veřejný klíč

ENS Ethereum Name Service

UI User Interface - uživatelské rozhraní

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `instalacni-prirucka.pdf` ..... insalační příručka
- `aplikace` ..... adresář s připravenými aplikacemi
  - `AEW.apk` ..... prototyp aplikace AEW
  - `test-app.apk` ... aplikace použitá k testování vystavené funkcionality
- `doc` ..... adresář s dokumentací zdrojových kódů
- `hifi-prototyp` ..... adresář s hi-fi prototypem uživatelského rozhraní
  - `AEW_prototype.vp` ..... soubor s projektem prototypu spustitelným v programu JustInMind
  - `proto_html` ..... adresář s prototypem vyexportovaným do webových stránek
- `src`
  - `android-ethereum-wallet` .... repozitář projektu se zdrojovými kódy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF