DELTA – Střední škola informatiky a ekonomie, s.r.o.

Ke Kamenci 151, PARDUBICE



Maturitní projekt
Termite - decentralizovaná digitální měna

Příjmení, jméno: Shestakov Kyrylo

Třída: 4. B

Studijní obor: Informační technologie 18-20-M/01

Školní rok: 2024/2025

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení: Kyrylo Shestakov

Pro školní rok:2024/2025

Třída: 4. B

Obor: Informační technologie 18-20-M/01

Téma práce: Termite – Decentralizovaná digitální měna

Vedoucí práce: Ing. Tomáš Pešek

Způsob zpracování, Pokyny k obsahu , Cíle práce:

Cílem projektu je vybudovat funkční systém pro provádění finančních transakcí v síti.

Síť bude realizována prostřednictvím jednotlivých programů na počítačích uživatelů, kteří mají mezi sebou spojení a distribuují informace.

Obsah projektu:

• Desktop Application – Node: připojení k dalším uzlům sítě, šifrované připojení ,distribuci dat, potvrzení a validace informací.• Blockchain pro uchování historie transakcí, vytváření bloků, plnění konsensu PoW• Mobilní aplikace pro provádění finančních převodů v síti, management adres a zůstatků, převody mezi adresami, Šifrování a bezpečnost

Harmonogram:

- • Září: Vytvoření komunikačního a bezpečnostního modulu
- • Říjen: Vytvoření distribučního modulu, modul pro ukládání dat
- Listopad: Vytvoření blockchainu a ověřovacího modulu
- Prosinec: Mobilní aplikace
- Leden: Dokumentace a prezentace

Prohlašuji, že jsem maturitní projekt vypracoval(a) samostatně, výhradně s použitím uvedené literatury. V Pardubicích 31.3.2025

Poděkování

Děkuji Ing. Tomáši Peškovi za vedení mého maturitního projektu a jeho trpělivost. Dále

Anotace

Tato závěrečná práce je zaměřena na vývoj decentralizované sítě a blockchainu pro provádění finančních transakcí. Cílem projektu je vytvoření spolehlivého a bezpečného systému, který umožňuje chráněnou výměnu dat mezi uzly a zaručuje integritu transakcí.

V rámci projektu jsou implementovány bezpečnostní mechanismy založené na kryptografických metodách a také konsensuální algoritmy, které zajišťují konzistenci dat mezi uzly sítě. Architektura systému zahrnuje desktopovou aplikaci pro správu uzlů a mobilní aplikaci pro Android, která je určena pro uživatelskou interakci se sítí.

Vyvinutý systém je zaměřen na zvýšení úrovně ochrany dat, odolnost vůči útokům a decentralizaci zpracování transakcí, což jej činí relevantním pro použití ve finančním sektoru a dalších kriticky důležitých oblastech.

Obsah

1 Úvod 1

1.1 Relevance decentralizovaných systémů 1

1.2 Etické aspekty a prevence nelegálního využití 1

1.3 Závěr 2

2. Teoretický úvod 3

2.1 Blockchain 3

2.2 Block 4

2.3 Transaction 4

2.4 Node 5

2.5 Kryptoměna 5

3 Popis projektu 6

3.1 Architektura serveru TCP 6

3.1.1 Hlavní třídy a jejich interakce 7

3.1.1.1 TcpServer (hlavní server) 7

3.1.1.2 TcpRequest (TCP požadavek) 7

3.1.1.3 Response (Odpověď) 8

3.1.1.4 Controller (Řadič požadavků) 8

3.1.1.5 RequestHandlerFactory (Metoda na zpracovatele požadavků) 8

3.1.1.6 ServerResponseService (Služba odpovědí serveru) 8

3.1.1.7 IController (Rozhraní řadiče) 8

3.1.1.8 IRequestHandler (Rozhraní zpracovatele požadavků) 8
3.1.1.9 Zpracovatelé požadavků 9
3.1.1.10 Vztahy mezi třídami 9
3.1.2 Závěr 9
3.2 Architektura clientu TCP 9
3.2.1 Hlavní třídy a jejich interakce 10
3.2.1.1ClientTcp 10
3.2.1.2 IDbProcessor (Rozhraní) 10
3.2.1.3 DataSynchronizer 11
3.2.1.4 RequestExecutor 11
3.2.1.5 RequestHandler 11
3.1.2.6 ResponseHandler 11
3.1.2.7 ConnectionManager 12
3.1.2.8 Request 12
3.1.2.9 RequestFactory 12
3.1.2.10 RequestPool 12
3.1.2.11 DbProcessor 13
3.1.2.12 AppDbContext 13
3.1.2.13 Vztahy mezi třídami 13
3.2.2 Závěr 13
3.3 Architektura blockchainu 13
3.3.1 Hlavní třídy a jejich interakce 15
3.3.1.1 Blockchain 15

3.3.1.2 Block 16
3.3.1.3 BlockBuilder 16
3.3.1.4 BlockManager 17
3.3.1.5 Transaction 17
3.3.1.6 MerkleRoot 18
3.3.1.7 TransactionMemoryPool 18
3.3.1.8 BlockchainService 19
3.3.1.9 TransactionService 19
3.3.1.10 TransactionManager 20
3.3.2 Závěr: 20
3.4 Architektura API 21
3.4.1 Hlavní třídy a jejich interakce 22
3.4.1.1 Service (Služba) 22
3.4.1.2 MainController (Hlavní kontroler) 23
3.4.1.3 IRepository (Repozitář) 24
3.4.1.4 Repository (Repozitář) 24
3.4.2 Zavěr: 25
3.5 Architektura protocolu 26
3.5.1 Hlavní třídy a jejich interakce 26
3.5.1.1 TerProtocol<T> 26
3.5.1.6 TerPayload<T> 27
3.5.1.7 TerHeader 27
3.5.1.8 IpHelper 28

3.5.1.9 RequestSerializer 28
3.5.2 Typy požadavků a dat 29
3.5.2.1 BlockRequest 29
3.5.2.2 InfoSyncRequest 29
3.5.2.3 KeyExchangeRequest 29
3.5.2.4 PeerInfoRequest 29
3.5.2.5 TransactionRequest 30
3.5.2.6 Popis interakce 30
3.6 Mobilní aplikace 30
3.6.1 Hlavní funkce: 31
3.6.2 Příklad použití kryptopeněženky 32
Vytvoření peněženky 32
Dobití zůstatku 33
Odesílání kryptoměny 34
Zobrazení historie transakcí 35
3.7 Desktopová aplikace 35
3.7.1 Příklad použití uzlu sítě 36
4 Použité technologie 37
4.1 Programovací jazyk 37
4.2 Síťové technologie 37
4.3 Bezpečnost a kryptografie 37
4.4 Blockchain a Mining 37
4.5 Databáze 38

4.6 Rámce a knihovny 38

4.7 Vývoj pro mobilní zařízení 38

4.8 Správa verzí 38

4.9 Další technologie 38

5 Závěr 39

6 Seznam literatury 39

1 Úvod

V dnešní době inflace vede k znehodnocení úspor a vlády zavádějí nové poplatky za bankovní transakce (například na Slovensku). Nestabilní globální situace zároveň zvyšuje poptávku po decentralizovaných systémech, které uživatelům poskytují finanční nezávislost a soukromí.

1.1 Relevance decentralizovaných systémů

1. **Finanční svoboda**
   - Centrální banky a vlády mohou ovládat tradiční finanční systémy, zavádět nové daně a poplatky.
   - Decentralizovaná řešení umožňují uživatelům plnou kontrolu nad jejich aktivy bez prostředníků.
2. **Snížení poplatků**
   - Bankovní převody často zahrnují vysoké poplatky, zejména u mezinárodních transakcí.
   - Decentralizované sítě nabízejí rychlejší a levnější transakce.
3. **Ochrana proti inflaci**
   - V době rostoucí inflace ztrácejí tradiční měny svou hodnotu.
   - Decentralizovaná aktiva mohou sloužit jako alternativa k uchování hodnoty.
4. **Spolehlivost a odolnost vůči výpadkům**
   - Centralizované systémy jsou náchylné k cenzuře, zmrazení účtů a technickým výpadkům.
   - V decentralizovaných sítích jsou informace rozloženy mezi mnoho uzlů, což zajišťuje jejich stabilitu a bezpečnost.

1.2 Etické aspekty a prevence nelegálního využití

Při diskusích o decentralizovaných řešeních se často objevují otázky ohledně jejich potenciálního zneužití k nelegálním aktivitám (např. černý trh na dark webu). Důležité je zvážit následující aspekty:

- **Decentralizace ≠ anonymita**
  - Ve většině blockchainů jsou transakce veřejně dohledatelné, což je činí méně anonymními než hotovostní platby.
  - Pro prevenci nelegálních aktivit lze využít analytické nástroje pro sledování transakcí a detekci podezřelé aktivity.
- **Legální využití technologie**
  - Decentralizované systémy pomáhají milionům lidí v zemích se slabou ekonomikou a represivními režimy získat přístup k finančním službám.
  - Umožňují mezinárodní platby, chrání uživatele před cenzurou a politickým tlakem.
- **Odpovědnost za použití**
  - Stejně jako u internetu nebo kryptoměn je technologie sama o sobě neutrální – její využití závisí na účelu, ke kterému ji uživatelé používají.
  - Pro prevenci nezákonných aktivit mohou být do systému integrovány regulační mechanismy a nástroje pro dodržování legislativy.

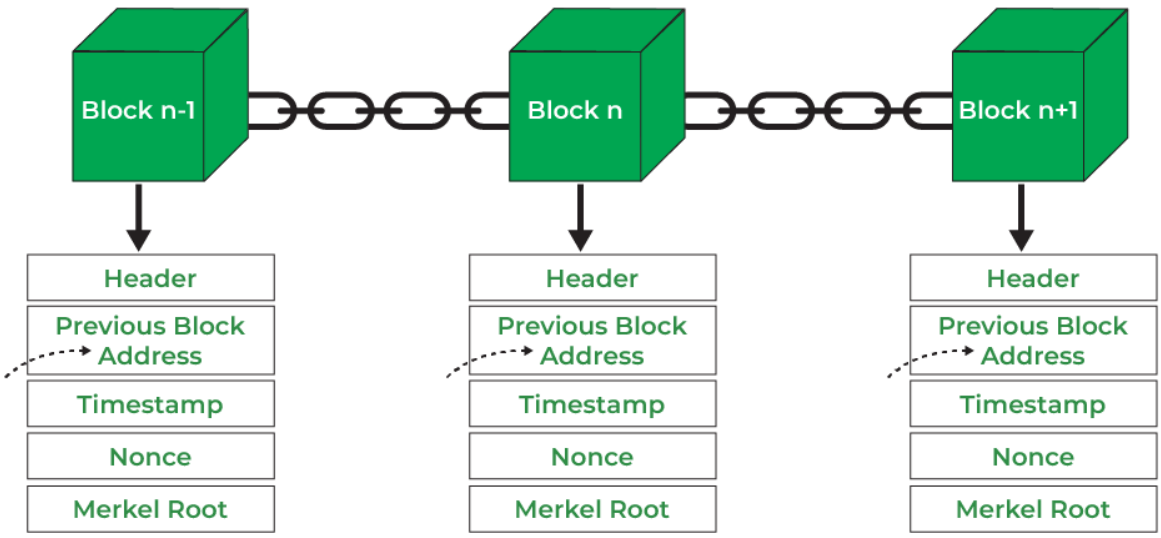
1.3 Závěr

Decentralizované systémy představují krok do budoucnosti, kde si uživatelé mohou spravovat své finance bez závislosti na centralizovaných institucích. Nabízejí nové možnosti po celém světě a pomáhají chránit lidi před inflací, vysokými poplatky a finančními omezeními. Stejně jako každá jiná technologie však vyžadují odpovědné používání a mohou být přizpůsobeny pro legální a bezpečné využití.

2. Teoretický úvod

2.1 Blockchain

Je posloupnost bloků, které jsou vzájemně propojeny pomocí hashování. Každý nový blok zahrnuje hash předchozího bloku, což zajišťuje nezměnitelnost celého řetězce. Blockchain slouží jako distribuovaná databáze pro všechny účastníky systému, kde jsou transakce dostupné a chráněné. Na obrázku 1 vidíte, jak jsou bloky propojeny, a na obrázku 2, jak jsou propojeny v blockchainu tohoto projektu.



Obrázek, 1: jak vypadá blockchain

Blockchain

Index: 0
Date: 30.12.2024 17:51:01
Hash: xibzZ8hl/rRoLzJCnmmEbSezPqbZ2r1r6YlzHW+fb+M=
PreviousHash: 0
Id: 8b055af7-9715-4d2c-9520-e9d2bf00594b
Merkle root: 8B259278929C1C2237B7AD2F78ABC3B2B6CAA89BFD460CFC3561165FA0FADAEF

Index: 1
Date: 28.02.2025 8:30:26
Hash: 00qEqUYlxTBCtLcpVZElopEOieyKWq2bi5PqvLKvzYk=
PreviousHash: xibzZ8hl/rRoLzJCnmmEbSezPqbZ2r1r6YlzHW+fb+M=
Id: f10f919c-f758-4d7e-803f-5356277916c7
Merkle root: 370CBA1697F860C90077AEDFA83710F57D00488D1D1D6791A6F651F96C4924B0

Index: 2
Date: 28.02.2025 17:37:46
Hash: 00+a1OGm5jjNTHYmOy7ExSwSPike0X/l2y+zULv5+Qw=
PreviousHash: 00qEqUYlxTBCtLcpVZElopEOieyKWq2bi5PqvLKvzYk=
Id: 390c73f9-2508-44da-a2ea-612569c49126
Merkle root: A820FC71C6F484D862EB3E7C48297048F7CDE7D3B8BB44360ADD75BC3E348931

Index: 3
Date: 24.03.2025 21:16:49
Hash: 00aL2HAyb/ADF62ZgGACyGTbjb3TknuargSJaOLPj4Q=
PreviousHash: 00+a1OGm5jjNTHYmOy7ExSwSPike0X/l2y+zULv5+Qw=
Id: 67c2eabd-69ea-4fa6-b4e5-2d3409af1281

Obrázek, 2: jak vypadá blockchain v této práci

2.2 Block

je strukturovaná jednotka dat v blockchainu, která obsahuje seznam transakcí a metadata nezbytná k zajištění integrity a bezpečnosti řetězce. Když je blok naplněn, je hashován pomocí SHA-256 a přidán do řetězce, čímž je zajištěna neměnnost dat prostřednictvím kryptografických vazeb.

2.3 Transaction

Je převod digitálních mincí od jednoho uživatele k druhému. Obsahuje informace o odesílateli, příjemci a částce převodu. Po odeslání je transakce ověřena sítí, zapsána do bloku a stává se neměnnou. Všechny operace jsou chráněny kryptografií a probíhají bez zprostředkovatelů, jako jsou banky.

2.4 Node

Je jakékoli zařízení (počítač, server) v decentralizované síti, které se podílí na zpracování, ukládání a ověřování bloků a transakcí.

2.5 Kryptoměna

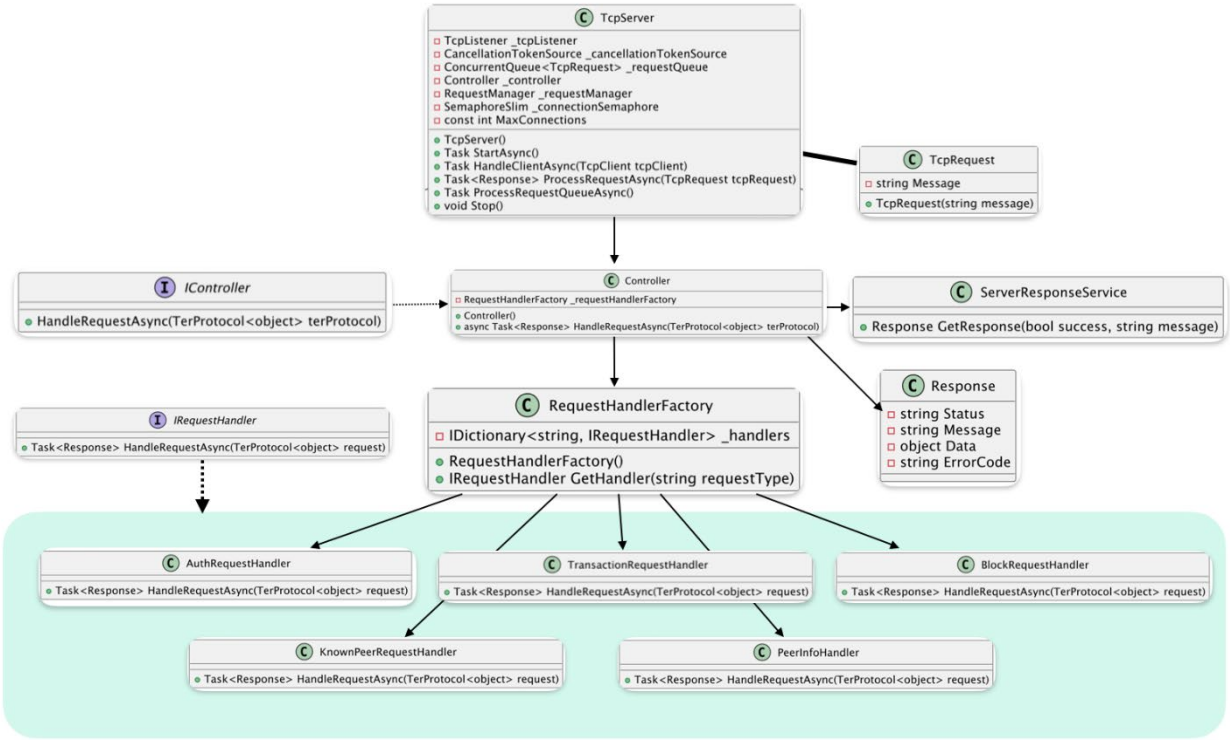
Je digitální měna založená na technologii blockchainu, která využívá kryptografii k zabezpečení transakcí. Je decentralizovaná, není řízena bankami ani státy a umožňuje rychlé a bezpečné převody bez zprostředkovatelů.

3 Popis projektu

3.1 Architektura serveru TCP

Obecný popis

Tento modul představuje TCP server pro zpracování požadavků v decentralizované síti. Přijímá připojení, zpracovává příchozí požadavky, s odešle je příslušným obslužným jednotkám Obslužné jednotky přesměrují požadavky na služby a vrátí odpovědi klientům.



Obrázek, 3: UML diagram serveru

Diagram (na obrázku 3) znázorňuje fungování TCP serveru, který zpracovává příchozí požadavky pomocí kontrolerů a obslužných tříd. Hlavní třída TcpServer (na obrázku 3) naslouchá příchozím připojením pomocí TcpListener, přidává požadavky do fronty _requestQueue, zpracovává je pomocí Controller a omezuje počet připojení pomocí MaxConnections. Server se spouští metodou StartAsync(), zatímco HandleClientAsync() se stará o zpracování klientů. Požadavky jsou reprezentovány třídou TcpRequest (na obrázku 3), která obsahuje textovou zprávu, a odpovědi strukturou Response, jež zahrnuje stav, zprávu, data a chybový kód.

Controller (zobrazený na obrázku 3) řídí zpracování požadavků pomocí RequestHandlerFactory, která vybírá vhodného obslužného zpracovatele, a vrací Response.

RequestHandlerFactory uchovává obslužné třídy (IRequestHandler) v _handlers a poskytuje odpovídající zpracovatele podle typu požadavku. Implementace IRequestHandler zahrnují AuthRequestHandler (autentizace), KnownPeerRequestHandler (zpracování informací o peerech), TransactionRequestHandler (zpracování transakcí), BlockRequestHandler (zpracování bloků) a PeerInfoHandler (práce s informacemi o peerech). ServerResponseService (zobrazený na obrázku 3) odpovídá za vytváření objektů Response.

Pracovní proces (znázorněný na obrázku 3) probíhá následovně: TcpServer přijme připojení a vytvoří TcpRequest, který následně předá do Controller. Ten prostřednictvím RequestHandlerFactory vybere správného zpracovatele. Vybraný IRequestHandler zpracuje požadavek a vytvoří Response, který se vrátí přes Controller zpět na server a následně je odeslán klientovi. Tento přístup zajišťuje flexibilitu a škálovatelnost serveru.

3.1.1 Hlavní třídy a jejich interakce

3.1.1.1 TcpServer (hlavní server)

TcpServer spravuje síťová připojení a zpracovává příchozí TCP požadavky.

Hlavní komponenty:

- _tcpListener – naslouchá příchozím připojením.
- _cancellationTokenSource – spravuje zastavení serveru.
- _requestQueue – fronta příchozích TCP požadavků.
- _controller – hlavní procesor požadavků.
- _requestManager – manažer, který se zabývá dešifrováním požadavku a zpracováním zpráv.
- _connectionSemaphore – omezuje počet aktivních připojení.
- MaxConnections – maximální počet současných připojení.

Hlavní metody:

- StartAsync() – spouští server a začíná naslouchat připojením.
- HandleClientAsync(TcpClient tcpClient) – zpracovává připojení klienta.
- ProcessRequestAsync(TcpRequest tcpRequest) – odesílá požadavek ke zpracování.
- ProcessRequestQueueAsync() – zpracovává frontu požadavků.
- Stop() – zastavuje server.

3.1.1.2 TcpRequest (TCP požadavek)

Reprezentuje požadavek přijatý od klienta.

Message – řetězec obsahující data požadavku.

3.1.1.3 Response (Odpověď)

Používá se k vrácení výsledku zpracování požadavku.

- Status – stav odpovědi.
- Message – textový popis výsledku.
- Data – dodatečná data.
- ErrorCode – chybový kód (pokud existuje).

3.1.1.4 Controller (Řadič požadavků)

Zodpovídá za směrování a zpracování požadavků.

Hlavní komponenty:

_requestHandlerFactory – metoda na zpracovatele požadavků.

Hlavní metoda:

HandleRequestAsync(TerProtocol<object>) – předává požadavek správnému zpracovateli.

3.1.1.5 RequestHandlerFactory (Metoda na zpracovatele požadavků)

Vybere správného zpracovatele pro každý typ požadavku.

Hlavní metody:

GetHandler(string requestType) – vrací zpracovatele pro zadaný typ požadavku.

3.1.1.6 ServerResponseService (Služba odpovědí serveru)

Zodpovídá za vytvoření odpovědi na požadavek.

- GetResponse(bool success, string message, object data) – vytvoří odpověď s výsledkem operace.

3.1.1.7 IController (Rozhraní řadiče)

Definuje metodu pro zpracování požadavků:

- HandleRequestAsync(TerProtocol<object> terProtocol)

3.1.1.8 IRequestHandler (Rozhraní zpracovatele požadavků)

Definuje metodu pro zpracování požadavků:

- HandleRequestAsync(TerProtocol<object> terProtocol)

3.1.1.9 Zpracovatelé požadavků

Každý zpracovatel implementuje IRequestHandler a je zodpovědný za konkrétní typ požadavku:

- AuthRequestHandler – zpracování autentizačních požadavků.
- KnownPeerRequestHandler – zpracování požadavků o známých uzlech.
- TransactionRequestHandler – zpracování transakcí.
- BlockRequestHandler – zpracování požadavků souvisejících s bloky.
- PeerInfoHandler – zpracování požadavků o informacích o uzlech.
- BlockchainHandler - zpracovává žádosti o informace o stavu blockchainu.

3.1.1.10 Vztahy mezi třídami

- TcpServer používá Controller pro směrování požadavků.
- Controller volá RequestHandlerFactory pro získání zpracovatele.
- RequestHandlerFactory vrací IRequestHandler v závislosti na typu požadavku.
- ServerResponseService vytváří odpověď, která je odeslána klientovi.

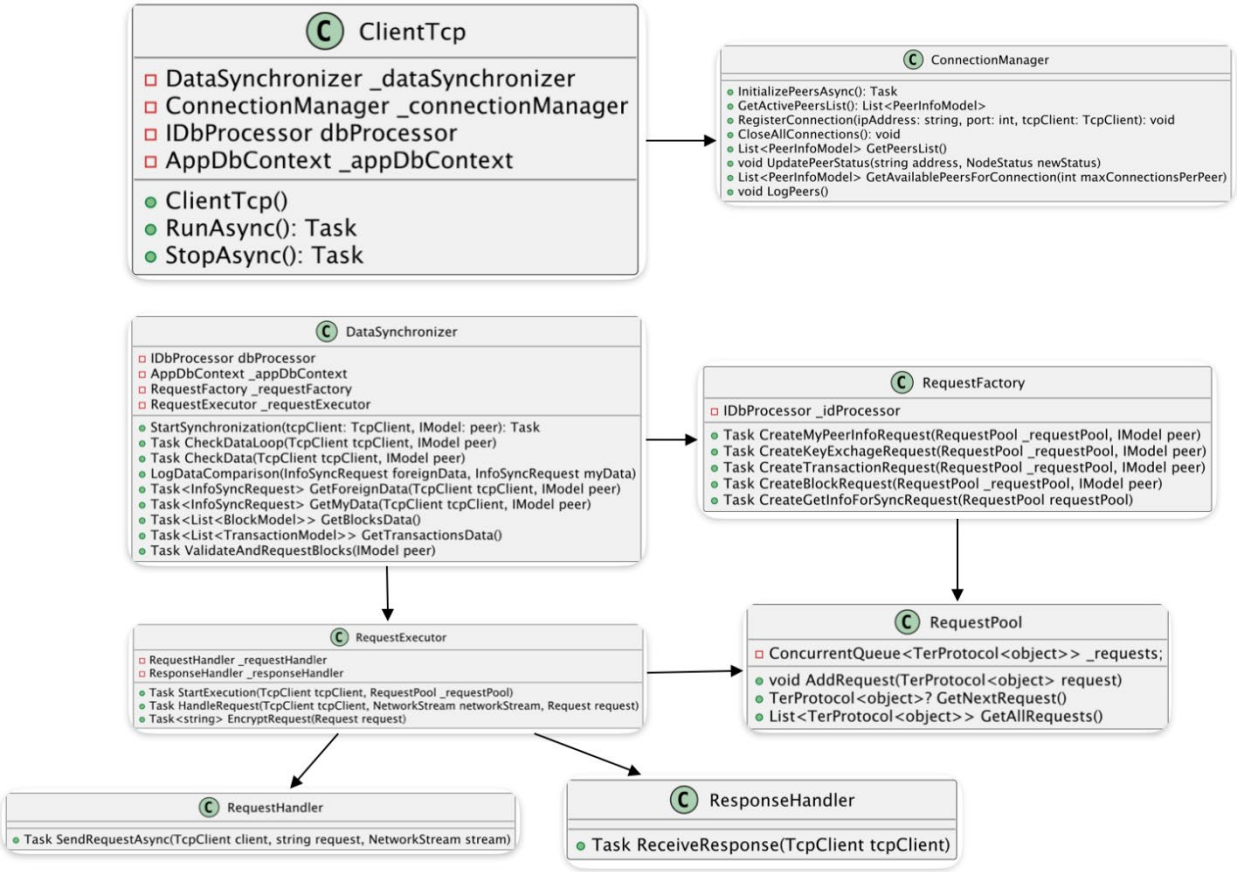
3.1.2 Závěr

Tento server implementuje zpracování TCP požadavků, jejich směrování a předání ke zpracování odpovídajícím handlerům. Systém podporuje modularitu, což umožňuje snadné přidávání nových typů požadavků a zpracovatelů.

3.2 Architektura clientu TCP

Obecný popis

Klient uzlu decentralizované sítě zajišťuje navázání připojení k známým uzlům, správu připojení a synchronizaci dat. Používá ConnectionManager pro inicializaci a registraci připojení a DataSynchronizer pro přenos a aktualizaci dat. Klient se automaticky připojuje k aktivním uzlům a provádí zpracování síťových chyb



Obrázek, 4: UML diagram clientu

3.2.1 Hlavní třídy a jejich interakce

3.2.1.1ClientTcp

Atributy:

- _dataSynchronizer — objekt pro synchronizaci dat.
- _connectionManager — objekt pro správu připojení.
- _dbProcessor — objekt pro práci s databází.
- _appDbContext — objekt kontextu databáze.

Metody:

- RunAsync() — spustí klienta a nastaví připojení k jiným uzlům.
- StopAsync() — ukončí všechna připojení k uzlům.

3.2.1.2 IDbProcessor (Rozhraní)

Popis: Rozhraní pro implementaci zpracování dat v databázi.

Metody:

- ProcessService<T> — zpracovává dotazy k databázi.

3.2.1.3 DataSynchronizer

Popis: Správce synchronizace dat mezi klientem a serverem. Interaguje s databází a provádí dotazy na synchronizaci.

Atributy:

- _dbProcessor — objekt pro zpracování dat.
- _appDbContext — objekt kontextu databáze.
- _requestFactory — továrna na vytváření dotazů.
- _requestExecutor — vykonavatel pro zpracování dotazů.

Metody:

- StartSynchronization() — spustí proces synchronizace dat.
- CheckDataLoop() — provádí cyklickou kontrolu dat.
- CheckData() — kontroluje data pro synchronizaci.
- LogDataComparison() — loguje výsledek porovnání dat.
- GetForeignData() — získává data z jiného uzlu.
- GetMyData() — získává data z aktuálního uzlu.
- GetBlocksData() — získává data bloků.
- GetTransactionsData() — získává data transakcí.
- ValidateAndRequestBlocks() — validuje a požaduje bloky dat.

3.2.1.4 RequestExecutor

Popis: Vykonavatel pro zpracování dotazů a odpovědí.

Atributy:

- _requestHandler — zpracovatel dotazů.
- _responseHandler — zpracovatel odpovědí.

Metody:

- StartExecution() — spustí provedení dotazu.
- HandleRequest() — zpracovává dotaz.
- EncryptRequest() — provádí šifrování dotazu.

3.2.1.5 RequestHandler

Popis: Zpracovatel pro odesílání dotazů.

Metody:

- SendRequestAsync() — odesílá dotaz asynchronně.

3.1.2.6 ResponseHandler

Popis: Zpracovatel pro příjem odpovědí.

Metody:

- ReceiveResponse() — přijímá odpověď od serveru.

3.1.2.7 ConnectionManager

Popis: Správce pro řízení připojení k jiným uzlům.

Metody:

- InitializePeersAsync() — inicializuje seznam dostupných uzlů.
- GetActivePeersList() — získává seznam aktivních uzlů.
- RegisterConnection() — registruje připojení k uzlu.
- CloseAllConnections() — uzavírá všechna připojení.
- GetPeersList() — získává seznam všech uzlů.
- UpdatePeerStatus() — aktualizuje stav uzlu.
- GetAvailablePeersForConnection() — získává uzly pro připojení s ohledem na maximální počet připojení.
- LogPeers() — loguje stav uzlů.

3.1.2.8 Request

Popis: Třída reprezentující dotaz.

Atributy:

- Message — řetězec s zprávou dotazu.

3.1.2.9 RequestFactory

Popis: služba pro vytváření různých typů dotazů.

Metody:

- CreateMyPeerInfoRequest() — vytváří dotaz pro získání informací o peer uzlu.
- CreateKeyExchageRequest() — vytváří dotaz pro výměnu klíčů.
- CreateTransactionRequest() — vytváří dotaz pro odeslání transakce.
- CreateBlockRequest() — vytváří dotaz pro získání bloku dat.
- CreateGetInfoForSyncRequest() — vytváří dotaz pro synchronizaci dat.

3.1.2.10 RequestPool

Popis: Přípojka dotazů, která řídí pořadí a vykonávání dotazů.

Atributy:

- _requests — fronta dotazů.

Metody:

- AddRequest() — přidává dotaz do fronty.

- GetNextRequest() — získává následující dotaz z fronty.
- GetAllRequests() — získává všechny dotazy z fronty.

3.1.2.11 DbProcessor

Popis: Implementace rozhraní pro zpracování databázových dotazů.

Metody:

- Process() — zpracovává dotazy k databázi.

3.1.2.12 AppDbContext

Popis:

Kontext databáze.

Metody:

- SaveChanges() — ukládá změny do databáze.

3.1.2.13 Vztahy mezi třídami

ClientTcp používá DataSynchronizer, ConnectionManager, DbProcessor, a AppDbContext.

DataSynchronizer používá RequestFactory a RequestExecutor.

RequestExecutor používá RequestHandler a ResponseHandler..

3.2.2 Závěr

Tento proces zajišťuje efektivní synchronizaci dat mezi uzly v decentralizované síti. Díky porovnání dat mezi lokálním úložištěm a vzdáleným uzlem lze identifikovat chybějící nebo neplatná data a automaticky provést jejich aktualizaci. Použitím RequestFactory k vytvoření požadavků a RequestExecutor k jejich odeslání se garantuje, že každý uzel má vždy aktuální a konzistentní informace. Tento mechanismus přispívá k bezpečnosti, spolehlivosti a stabilitě celé sítě.

3.3 Architektura blockchainu

Obecný popis

V projektu blockchain slouží k neměnnému ukládání dat. Každý blok obsahuje platné transakce od uživatelů a zaručuje jejich integritu pomocí kryptografických funkcí.

K potvrzování bloků se používá konsensuální algoritmus **Proof of Work (PoW)**. Ten je realizován hledáním správného **nonce**, což vyžaduje výpočetní výkon. Uzlu, který jako první vyřeší úlohu, je udělena odměna ve formě tokenů sítě. To motivuje uzly k účasti na podpoře sítě a zajišťuje její bezpečnost.

V blockchainových systémech se používají různé algoritmy konsensu, které zajišťují validaci transakcí a ochranu sítě před útoky. Mezi nejběžnější patří:

1. Proof of Work (PoW) – důkaz práce. Účastníci sítě řeší složité matematické úkoly (hledání nonce), což vyžaduje značnou výpočetní sílu.
2. Proof of Stake (PoS) – důkaz podílu. Uzly s největším množstvím mincí mají prioritu při vytváření nových bloků.
3. Delegated Proof of Stake (DPoS) – delegovaný důkaz podílu. Validátoři jsou vybíráni uživateli a bloky jsou vytvářeny omezeným počtem uzlů.
4. Proof of Authority (PoA) – důkaz autority. Pouze předem schválené uzly mohou ověřovat transakce, což zvyšuje rychlost, ale snižuje decentralizaci.
5. Proof of Burn (PoB) – důkaz spálení. Uzly ničí část svých mincí, aby získaly právo ověřovat transakce.
6. Proof of Elapsed Time (PoET) – důkaz uplynulého času. Používá náhodné zpoždění, které určuje, který uzel vytvoří následující blok.

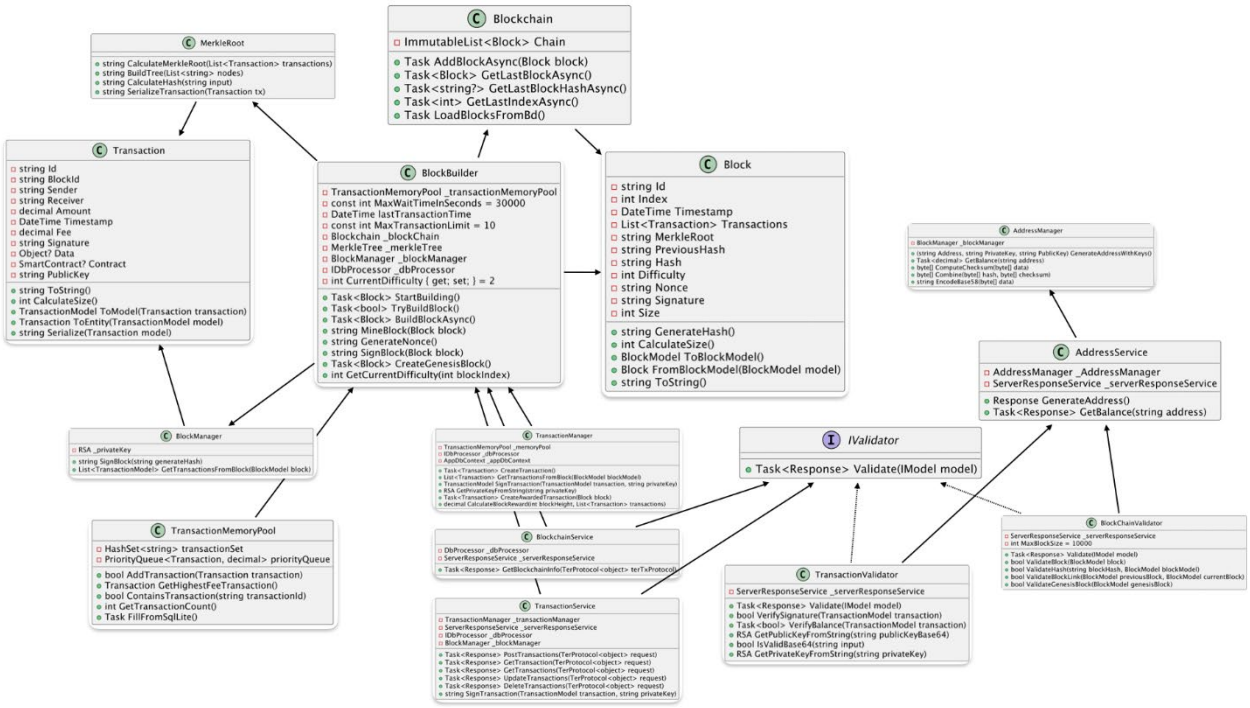
Odůvodnění výběru Proof of Work

Algoritmus **Proof of Work** byl vybrán, protože zajišťuje vysokou úroveň bezpečnosti a decentralizace. Díky nutnosti řešit složité výpočetní úkoly se útok na síť stává velmi nákladným.

Hlavní výhody PoW:

- Spolehlivost – vyžaduje značné výpočetní prostředky, což činí falšování bloků téměř nemožné.
- Decentralizace – je dostupný každému uzlu bez nutnosti vlastnit velké množství mincí.
- Nezměnitelnost dat – vysoká složitost výpočtů brání možnosti změnit historii transakcí.

I přes vysokou spotřebu energie zůstává **Proof of Work** jedním z nejprověřenějších a nejbezpečnějších algoritmů konsensu, který zajišťuje ochranu sítě a spravedlivé rozdělení odměn mezi účastníky.



Obrázek, 5: UML diagram blockchainu

3.3.1 Hlavní třídy a jejich interakce

3.3.1.1 Blockchain

Popis:
Hlavní třída reprezentující blockchain. Řídí řetězec bloků a poskytuje metody pro přidávání bloků a získávání informací o posledním bloku.

Atributy:

- Chain (ImmutableList<Block>) — seznam bloků v blockchainu.

Metody:

- AddBlockAsync(Block block) — přidává nový blok do blockchainu.
- GetLastBlockAsync() — získává poslední blok v blockchainu.
- GetLastBlockHashAsync() — získává hash posledního bloku.
- GetLastIndexAsync() — získává index posledního bloku.
- LoadBlocksFromBd() — načítá bloky z databáze.

3.3.1.2 Block

Popis:
Třída reprezentující jednotlivý blok v blockchainu.

Atributy:

- Id (string) — unikátní identifikátor bloku.
- Index (int) — index bloku v řetězci.
- Timestamp (DateTime) — časová značka bloku.
- Transactions (List<Transaction>) — seznam transakcí v bloku.
- MerkleRoot (string) — Merkle root pro blok.
- PreviousHash (string) — hash předchozího bloku.
- Hash (string) — hash aktuálního bloku.
- Difficulty (int) — obtížnost těžby bloku.
- Nonce (string) — hodnota pro nalezení hashe bloku.
- Signature (string) — podpis bloku.
- Size (int) — velikost bloku v bajtech.

Metody:

- GenerateHash() — generuje hash pro blok.
- CalculateSize() — počítá velikost bloku.
- ToBlockModel() — převádí blok na datový model.
- FromBlockModel(BlockModel model) — vytváří blok z datového modelu.
- ToString() — vrací textovou reprezentaci bloku.

3.3.1.3 BlockBuilder

Popis:
Třída pro vytváření bloků, včetně generování genesis bloku a těžby.

Atributy:

- _transactionMemoryPool (TransactionMemoryPool) — transakční paměťový pool pro blok.
- _blockChain (Blockchain) — instance blockchainu.
- _merkleTree (MerkleTree) — Merkle strom.
- _blockManager (BlockManager) — manažer bloků.
- _dbProcessor (IDbProcessor) — databázový procesor.
- CurrentDifficulty (int) — aktuální obtížnost těžby.

Metody:

- StartBuilding() — spustí proces vytváření bloku.
- TryBuildBlock() — pokouší se vytvořit blok.
- BuildBlockAsync() — asynchronně vytváří blok.
- MineBlock(Block block) — těží blok.
- GenerateNonce() — generuje nonce pro blok.
- SignBlock(Block block) — podepisuje blok.
- CreateGenesisBlock() — vytváří genesis blok.
- GetCurrentDifficulty(int blockIndex) — vrací obtížnost pro zadaný blok.

3.3.1.4 BlockManager

Popis:
Manažer, který spravuje podpisy bloků.

Atributy:

- _privateKey (RSA) — privátní klíč pro podpisy bloků.

Metody:

- SignBlock(string generateHash) — podepisuje blok podle jeho hashe.
- GetTransactionsFromBlock(BlockModel block) — extrahuje transakce z bloku.

3.3.1.5 Transaction

Popis:
Třída pro transakce v blockchainu.

Atributy:

- Id (string) — unikátní identifikátor transakce.
- BlockId (string) — identifikátor bloku, ve kterém se transakce nachází.
- Sender (string) — odesílatel transakce.
- Receiver (string) — příjemce transakce.
- Amount (decimal) — částka transakce.
- Timestamp (DateTime) — časová značka transakce.
- Fee (decimal) — poplatek transakce.
- Signature (string) — podpis transakce.

- Data (object?) — doplňková data transakce.
- Contract (SmartContract?) — chytrý kontrakt, pokud existuje.
- PublicKey (string) — veřejný klíč odesílatele.

Metody:

- ToString() — vrací textovou reprezentaci transakce.
- CalculateSize() — počítá velikost transakce.
- ToModel(Transaction transaction) — převádí transakci na model.
- ToEntity(TransactionModel model) — převádí model zpět na transakci.
- Serialize(Transaction model) — serializuje transakci do řetězce.

3.3.1.6 MerkleRoot

Popis:
Třída pro práci s Merkle stromem a výpočet Merkle kořene.

Metody:

- CalculateMerkleRoot(List<Transaction> transactions) — počítá Merkle root pro seznam transakcí.
- BuildTree(List<string> nodes) — staví Merkle strom ze seznamu uzlů.
- CalculateHash(string input) — počítá hash pro řetězec.
- SerializeTransaction(Transaction tx) — serializuje transakci do řetězce.

3.3.1.7 TransactionMemoryPool

Popis:
Pool transakcí, který spravuje transakce předtím, než budou přidány do bloku.

Atributy:

- transactionSet (HashSet<string>) — množina identifikátorů transakcí.
- priorityQueue (PriorityQueue<Transaction, decimal>) — fronta prioritních transakcí.

Metody:

- AddTransaction(Transaction transaction) — přidává transakci do poolu.
- GetHighestFeeTransaction() — vrací transakci s nejvyššími poplatky.
- ContainsTransaction(string transactionId) — kontroluje, zda existuje transakce s daným ID.
- GetTransactionCount() — získává počet transakcí v poolu.
- FillFromSqlLite() — naplňuje pool transakcemi z databáze.

3.3.1.8 BlockchainService

Popis:
Služba pro práci s blockchainem, poskytující metody pro získání informací o blockchainu.

Atributy:

- _dbProcessor (DbProcessor) — databázový procesor.
- _serverResponseService (ServerResponseService) — služba pro zpracování odpovědí od serveru.

Metody:

- GetBlockchainInfo(TerProtocol<object> terTxProtocol) — získává informace o blockchainu.

3.3.1.9 TransactionService

Popis:
Služba pro práci s transakcemi, včetně vytváření, získávání a aktualizování transakcí.

Atributy:

- _transactionManager (TransactionManager) — manažer transakcí.
- _serverResponseService (ServerResponseService) — služba pro zpracování odpovědí od serveru.
- _dbProcessor (IDbProcessor) — databázový procesor.
- _blockManager (BlockManager) — manažer bloků.

Metody:

- PostTransactions(TerProtocol<object> request) — odesílá transakce na server.
- GetTransaction(TerProtocol<object> request) — získává transakci.
- GetTransactions(TerProtocol<object> request) — získává seznam transakcí.
- UpdateTransactions(TerProtocol<object> request) — aktualizuje transakce.

- DeleteTransactions(TerProtocol<object> request) — odstraňuje transakce.
- SignTransaction(TransactionModel transaction, string privateKey) — podepisuje transakci.

3.3.1.10 TransactionManager

Popis:
Manažer pro vytváření a zpracování transakcí.

Atributy:

- _memoryPool (TransactionMemoryPool) — transakční pool.
- _dbProcessor (IDbProcessor) — databázový procesor.
- _appDbContext (AppDbContext) — kontext databáze.

Metody:

- CreateTransaction() — vytváří transakci.
- GetTransactionsFromBlock(BlockModel blockModel) — získává transakce z bloku.
- SignTransaction(TransactionModel transaction, string privateKey) — podepisuje transakci.
- GetPrivateKeyFromString(string privateKey) — extrahuje privátní klíč z řetězce.
- CreateAwardedTransaction(Block block) — vytváří transakci pro odměnu bloku.
- CalculateBlockReward(int blockHeight, List<Transaction> transactions)

3.3.2 Závěr:

Vytvoření a správa blockchainu je komplexní proces, který zahrnuje interakci několika tříd, jejich atributů a metod, které spolupracují na zajištění správného fungování celé sítě. Hlavní třídy, jako je **Blockchain**, **Block**, **Transaction**, **TransactionManager**, **BlockManager** a další, spolu úzce spolupracují na zajištění integrity, bezpečnosti a efektivity transakcí a bloků v blockchainu.

Třída **Blockchain** poskytuje základní strukturu pro uchovávání bloků a jejich správu, zatímco **Block** definuje strukturu jednotlivého bloku včetně transakcí, hashů a podpisů. **Transaction** a její správa prostřednictvím **TransactionManager** a **TransactionMemoryPool** zajišťují, že transakce jsou správně pořadí a zpracovány před jejich přidáním do bloku. **BlockBuilder** a **BlockManager** se zaměřují na těžbu bloků a jejich validaci prostřednictvím generování nonce a podpisů.

Merkle strom a jeho kořen jsou klíčové pro validaci a optimalizaci transakcí v bloku. **TransactionService** a **BlockchainService** zajišťují interakci s databází a komunikaci s externími servery pro zajištění dostupnosti a správy informací o blockchainu.

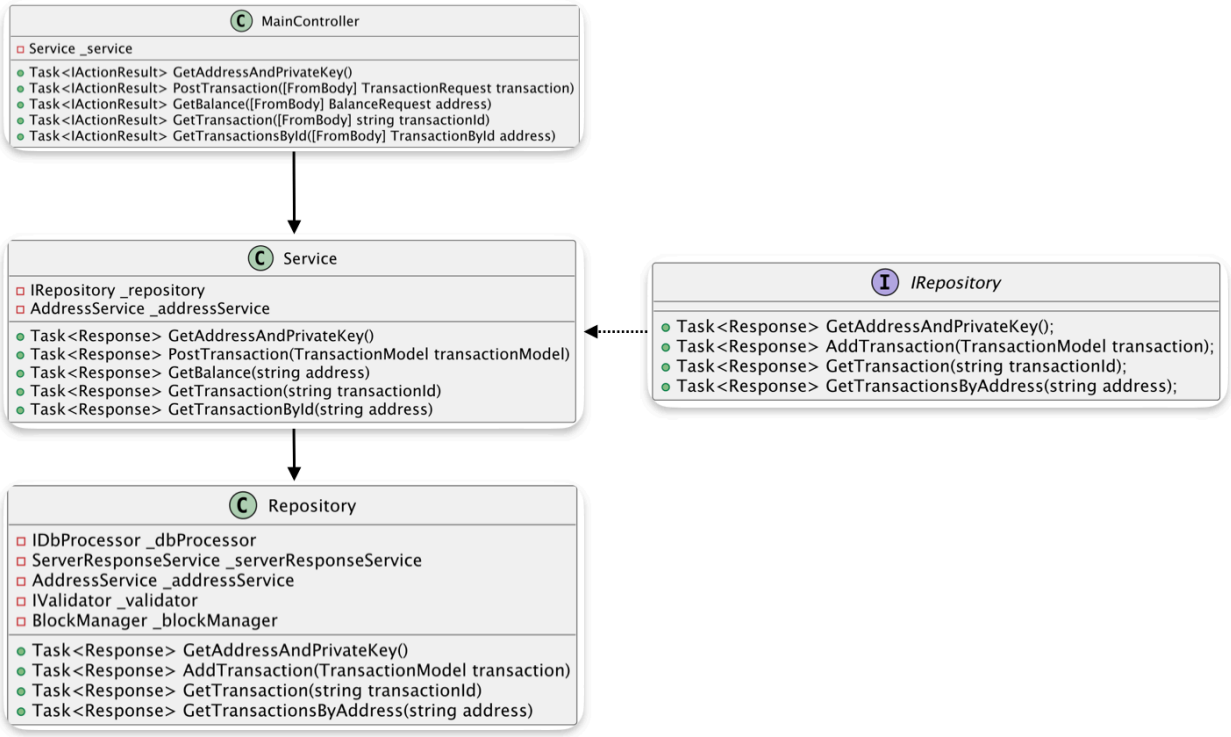
Celý systém je navržen tak, aby byl vysoce modulární, což umožňuje snadné rozšíření a úpravy. Interakce mezi jednotlivými třídami zajišťují bezpečné, efektivní a decentralizované zpracování transakcí a bloků v blockchainu. Tento design je základním stavebním kamenem pro implementaci spolehlivých a škálovatelných blockchainových aplikací.

V budoucnu by bylo možné tento systém vylepšit přidáním dalších funkcionalit, jako je podpora pro chytré kontrakty nebo implementace víceúrovňové škálovatelnosti pro vyšší zatížení. Tato architektura však již poskytuje solidní základ pro vývoj komplexních decentralizovaných aplikací a blockchainových řešení.

3.4 Architektura API

Obecný popis

Toto API uzlu je určeno pro interakci s uzlem blockchainu a umožňuje integraci s jinými aplikacemi, například mobilními aplikacemi pro správu kryptoměnových peněženek. Systém poskytuje různé endpointy, které umožňují aplikacím provádět operace, jako je získání adresy a soukromého klíče, provádění transakcí, kontrola zůstatků a získávání informací o transakcích.



Obrázek, 6: UML diagram API

3.4.1 Hlavní třídy a jejich interakce

3.4.1.1 Service (Služba)

Popis:
Třída Service je zodpovědná za poskytování vyšší úrovně obchodní logiky, která se týká adres, transakcí a zůstatků. Komunikuje s repozitářem a dalšími službami pro provádění potřebných operací.

Atributy:

- _repository (IRepository) — Rozhraní repozitáře, které se používá pro operace s daty.
- _addressService (AddressService) — Služba zodpovědná za operace spojené s adresami.

Metody:

- GetAddressAndPrivateKey() — Asynchronně získá adresu a s ní spojený soukromý klíč.
- PostTransaction(TransactionModel transactionModel) — Asynchronně vloží novou transakci do systému.
- GetBalance(string address) — Asynchronně získá zůstatek pro danou adresu.
- GetTransaction(string transactionId) — Asynchronně získá transakci na základě jejího ID.
- GetTransactionById(string address) — Asynchronně získá transakci na základě adresy.

3.4.1.2 MainController (Hlavní kontroler)

Popis:
Třída MainController je zodpovědná za zpracování příchozích HTTP požadavků, jejich zpracování a volání příslušných metod třídy Service.

Atributy:

- _service (Service) — Instance služby, která se používá pro zpracování obchodní logiky.

Metody:

- GetAddressAndPrivateKey() — Zpracovává HTTP požadavek pro získání adresy a soukromého klíče.
- PostTransaction(TransactionRequest transaction) — Zpracovává HTTP požadavek pro vložení nové transakce.
- GetBalance(BalanceRequest address) — Zpracovává HTTP požadavek pro získání zůstatku pro zadanou adresu.
- GetTransaction(string transactionId) — Zpracovává HTTP požadavek pro získání transakce podle jejího ID.
- GetTransactionsById(TransactionById address) — Zpracovává HTTP požadavek pro získání transakcí spojených s určitou adresou.

3.4.1.3 IRepository (Repozitář)

Popis:
Rozhraní IRepository definuje operace, které se týkají získávání a ukládání dat pro adresy a transakce. Implementace tohoto rozhraní se starají o interakci s datovou vrstvou.

Metody:

- GetAddressAndPrivateKey() — Asynchronně získá adresu a její soukromý klíč.
- AddTransaction(TransactionModel transaction) — Asynchronně přidá novou transakci do úložiště dat.
- GetTransaction(string transactionId) — Asynchronně získá transakci podle jejího ID.
- GetTransactionsByAddress(string address) — Asynchronně získá všechny transakce spojené s danou adresou.

3.4.1.4 Repository (Repozitář)

Popis:
Třída Repository implementuje rozhraní IRepository a zajišťuje interakci s podkladovou databází a dalšími externími službami. Řídí data týkající se adres, transakcí a blockchainu.

Atributy:

- _dbProcessor (IDbProcessor) — Procesor databáze zodpovědný za interakci s databází.
- _serverResponseService (ServerResponseService) — Služba zodpovědná za zpracování odpovědí serveru.
- _addressService (AddressService) — Služba zodpovědná za operace s adresami.
- _validator (IValidator) — Validátor pro ověřování platnosti dat.
- _blockManager (BlockManager) — Manažer pro operace související s bloky.

Metody:

- GetAddressAndPrivateKey() — Asynchronně získá adresu a její soukromý klíč.
- AddTransaction(TransactionModel transaction) — Asynchronně přidá novou transakci do databáze.
- GetTransaction(string transactionId) — Asynchronně získá transakci podle jejího ID.
- GetTransactionsByAddress(string address) — Asynchronně získá všechny transakce spojené s danou adresou.

3.4.2 Závěr:

Navržená architektura API představuje dobře strukturovaný systém pro správu operací s adresami, transakcemi a zůstatky v kontextu blockchainových aplikací. Každá z uvedených tříd má svou jasně definovanou roli, což zajišťuje flexibilitu, škálovatelnost a podporu různých úrovní logiky.

1. **Service** je zodpovědná za obchodní logiku, která spojuje repozitáře, databázi a externí služby. Inkapsculuje práci s adresami, transakcemi a zůstatky a poskytuje pohodlné metody pro jejich získání a zpracování.
2. **MainController** funguje jako prostředník mezi klientem a systémem, přijímá HTTP požadavky a přesměrovává je na odpovídající metody ve službách. To zjednodušuje zpracování požadavků a činí kód snadněji udržovatelným a testovatelným.
3. **IRepository** a **Repository** zajišťují přístup k datům a poskytují rozhraní pro práci s transakcemi a adresami. Díky oddělení rozhraní a jeho implementace je dosaženo vysoké flexibility, která umožňuje snadno měnit způsoby interakce s daty bez nutnosti měnit základní logiku.
4. **TransactionModel** a **AddressService** poskytují konkrétní mechanismy pro práci s transakcemi a adresami, což umožňuje systému vykonávat nezbytné operace s daty.

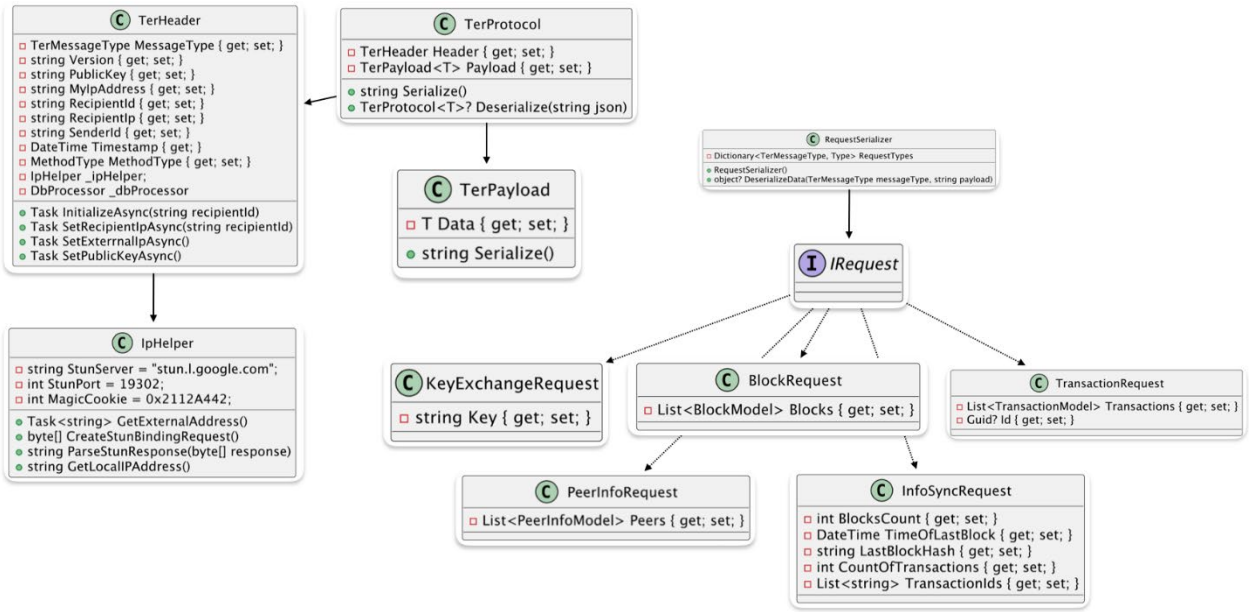
Celý systém funguje asynchronně, což je důležité pro zajištění vysokého výkonu při práci s velkými objemy dat, které jsou charakteristické pro blockchainové aplikace. Dělení na samostatné služby a repozitáře činí systém modulární, což usnadňuje testování a údržbu, a také umožňuje snadné přidávání nových funkčností, jako je správa bloků nebo integrace s jinými službami.

Celkově navržená architektura API poskytuje silný a flexibilní nástroj pro práci s blockchainovými daty, zajišťující všechny potřebné mechanismy pro provádění transakcí, získávání informací o zůstatcích a správu adres, což je základem pro vytváření škálovatelných a vysoce výkonných blockchainových systémů.

3.5 Architektura protocolu

Obecný popis

Protokol TerProtocol je navržen pro výměnu zpráv mezi uzly v decentralizované síti, poskytující bezpečnou a strukturovanou výměnu dat. Zahrnuje různé třídy a struktury pro serializaci a deserializaci dat, stejně jako pro zpracování metadat zpráv. Níže jsou uvedeny klíčové komponenty protokolu.



Obrázek, 7: UML diagram protocolu

3.5.1 Hlavní třídy a jejich interakce

3.5.1.1 TerProtocol<T>

Třída představující zprávu protokolu.

Obsahuje hlavičku (TerHeader) a payload (TerPayload), která může obsahovat libovolné typy dat.

Metody:

- string Serialize(): Serializuje objekt TerProtocol do řetězce JSON.
- TerProtocol<T>? Deserialize(string json): Deserializuje řetězec JSON do objektu TerProtocol.

Atributy:

- TerHeader Header { get; set; }: Hlavička zprávy obsahující metadata.
- TerPayload<T> Payload { get; set; }: Payload zprávy, která může obsahovat libovolná data typu T.

3.5.1.6 TerPayload<T>

Třída představující payload zprávy. Může obsahovat libovolná data typu T.

Metody:

- string Serialize(): Serializuje payload do řetězce JSON.

Atributy:

- T Data { get; set; }: Data obsažená v payload.

3.5.1.7 TerHeader

Třída obsahující metadata, která popisují zprávu.

Metody:

- Task InitializeAsync(string recipientId): Inicializuje hlavičku s údaji o příjemci.
- Task SetRecipientIpAsync(string recipientId): Nastaví IP adresu příjemce.
- Task SetExterrnalIpAsync(): Nastaví externí IP adresu.
- Task SetPublicKeyAsync(): Nastaví veřejný klíč pro podepisování a ověřování.

Atributy:

- TerMessageType MessageType { get; set; }: Typ zprávy (např. požadavek, odpověď).
- string Version { get; set; }: Verze protokolu.
- string PublicKey { get; set; }: Veřejný klíč odesílatele.
- string MyIpAddress { get; set; }: IP adresa odesílatele.
- string RecipientId { get; set; }: ID příjemce.
- string RecipientIp { get; set; }: IP adresa příjemce.
- string SenderId { get; set; }: ID odesílatele.
- DateTime Timestamp { get; }: Čas odeslání zprávy.
- MethodType MethodType { get; set; }: Typ metody (např. požadavek, odpověď, synchronizace).

- IpHelper _ipHelper: Pomocná třída pro práci s IP adresami a STUN serverem.
- DbProcessor _dbProcessor: Zpracovatel dat pro práci s databází.
- IValidator _validator: Validátor pro kontrolu dat.

3.5.1.8 IpHelper

Třída pro práci s IP adresami, včetně komunikace se STUN serverem pro získání externí IP adresy.

Metody:

- Task<string> GetExternalAddress(): Získá externí IP adresu pomocí STUN serveru.
- byte[] CreateStunBindingRequest(): Vytvoří požadavek na STUN binding.
- string ParseStunResponse(byte[] response): Parsuje odpověď od STUN serveru.
- string GetLocalIPAddress(): Získá místní IP adresu zařízení.

Atributy:

- string StunServer = "stun.l.google.com": Adresa STUN serveru.
- int StunPort = 19302: Port STUN serveru.
- int MagicCookie = 0x2112A442: hodnota pro STUN.

3.5.1.9 RequestSerializer

Třída pro serializaci a deserializaci požadavků v závislosti na typu zprávy.

Metody:

- RequestSerializer(): Konstruktor, který inicializuje slovník typů požadavků.
- object? DeserializeData(TerMessageType messageType, string payload): Deserializuje data podle typu zprávy.

Atributy:

- Dictionary<TerMessageType, Type> RequestTypes: Slovník spojující typy zpráv s typy dat.

3.5.2 Typy požadavků a dat

3.5.2.1 BlockRequest

Třída pro požadavek bloků.

Atributy:

- List<BlockModel> Blocks { get; set; }: Seznam bloků, které mají být požadovány.

3.5.2.2 InfoSyncRequest

Třída pro požadavek synchronizace informací o blocích.

Atributy:

- int BlocksCount { get; set; }: Počet bloků.
- DateTime TimeOfLastBlock { get; set; }: Čas posledního bloku.
- string LastBlockHash { get; set; }: Hash posledního bloku.
- int CountOfTransactions { get; set; }: Počet transakcí.
- List<string> TransactionIds { get; set; }: Seznam ID transakcí.

3.5.2.3 KeyExchangeRequest

Třída pro požadavek výměny klíčů.

Atributy:

- string Key { get; set; }: Klíč, který je předáván pro výměnu.

3.5.2.4 PeerInfoRequest

Třída pro požadavek informací o peeru (uzlu).

Atributy:

- List<PeerInfoModel> Peers { get; set; }: Seznam informací o peerech.

3.5.2.5 TransactionRequest

Třída pro požadavek transakcí.

Atributy:

- List<TransactionModel> Transactions { get; set; }: Seznam transakcí.
- Guid? Id { get; set; }: ID transakce.

3.5.2.6 Popis interakce

Protokol TerProtocol podporuje serializaci a deserializaci dat, stejně jako výměnu různých typů požadavků a odpovědí mezi uzly v síti. Je určen pro práci s objekty, jako jsou bloky, transakce a informace o peerech, stejně jako pro výměnu veřejných klíčů a synchronizaci dat.

Protokol zajišťuje:

- Výměnu metadat prostřednictvím hlaviček zpráv.
- Předávání dat ve formátu JSON.
- Dynamické zpracování různých typů zpráv a požadavků prostřednictvím serializace a deserializace.

3.6 Mobilní aplikace

Obecný popis

Mobilní aplikace **TermiteWallet** je kryptoměnová peněženka, která je navržena pro práci s decentralizovanou sítí. Umožňuje uživatelům bezpečně spravovat jejich kryptoměnová aktiva a komunikovat s API uzlů sítě. Aplikace poskytuje intuitivní a snadno ovladatelný rozhraní pro vytváření a importování peněženek, stejně jako pro provádění transakcí a dalších operací s kryptoměnami.

Aplikace je postavena pomocí **Android Studio** a napsána v programovacím jazyce **Java**. Implementuje všechny potřebné funkce pro práci s decentralizovanou sítí, včetně:

- **Vytváření nové peněženky**: Uživatelé mohou vytvořit novou kryptoměnovou peněženku s unikátními klíči.
- **Import stávající peněženky**: Možnost obnovy a používání dříve vytvořené peněženky pomocí soukromých klíčů nebo jiných metod obnovy.
- **Rozhraní s API uzlů sítě**: Aplikace se připojuje k uzlům decentralizované sítě pro získání a odesílání dat o transakcích, synchronizaci zůstatků a další informace.

- **Bezpečnost dat**: Aplikace používá moderní šifrovací metody pro zajištění bezpečnosti dat, včetně správy sezení a veřejných klíčů.
- **Uživatelsky přívětivé rozhraní**: Vytvořeno jednoduché a přehledné rozhraní pro uživatele s různou úrovní technických znalostí.

Aplikace umožňuje snadnou interakci s kryptoměnovými blockchainy a zároveň poskytuje vysokou úroveň bezpečnosti a soukromí uživatelů v rámci decentralizované sítě.

3.6.1 Hlavní funkce:

- Připojení k decentralizovaným uzlům sítě
- Vytváření a obnova peněženek
- Autentifikace a šifrování dat
- Podpora základních kryptoměnových operací (např. odesílání a přijímání prostředků)
- Jednoduchost a pohodlí rozhraní pro uživatele

TermiteWallet je vynikající volba pro ty, kteří hledají bezpečnou a uživatelsky přívětivou mobilní aplikaci pro práci s kryptoměnami v decentralizovaných sítích.



Obrázek, 8: Termite wallet

3.6.2 Příklad použití kryptopeněženky

Kryptopeněženka umožňuje uživateli bezpečně ukládat, odesílat a přijímat kryptoměnu. Podívejme se na příklad jejího použití:

Vytvoření peněženky

- Uživatel nainstaluje aplikaci kryptopeněženky a vygeneruje novou peněženku.
- Při tom se vytvoří soukromý a veřejný klíč.

Dobití zůstatku

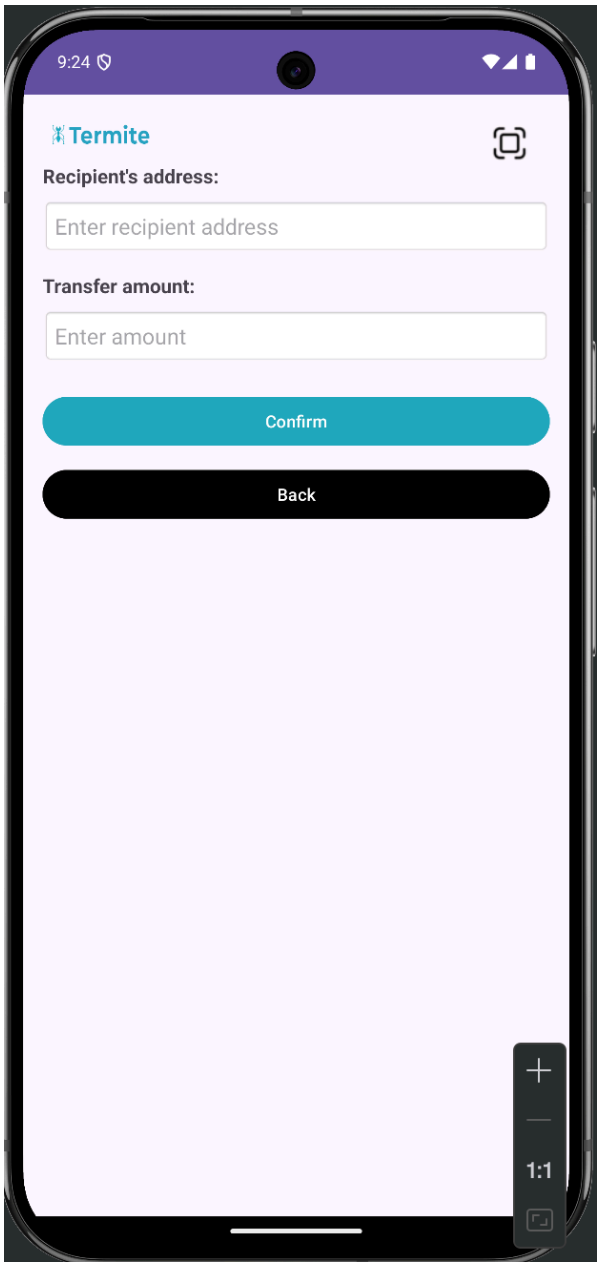
- Uživatel získá svou veřejnou adresu (je vidět na obrázku 8 pod zůstatkem peněženky.) a předá ji odesílateli.
- Jiný uživatel nebo burza pošle prostředky na tuto adresu.
- Po potvrzení transakce se aktualizuje zůstatek peněženky.
- Peněženka může vygenerovat QR kód s adresou, který lze zobrazit na obrazovce pro snadné skenování a odeslání prostředků.



Obrázek, 9: Obrazovka pro příjem

Odesílání kryptoměny

- Uživatel zadá adresu příjemce a částku k odeslání.
- Pro rychlé zadání adresy může použít skener QR kódů. Na obrázku 9 v pravém horním rohu je tlačítko pro zapnutí skeneru.
- Potvrdí transakci podepsáním pomocí svého soukromého klíče.
- Prostředky jsou odeslány a záznam o transakci je uložen v blockchainu.



Obrázek, 10: Obrazovka odeslání

Zobrazení historie transakcí

- V peněžence lze sledovat příchozí i odchozí transakce, jejich stav a poplatky.
- Správa různých aktiv v rámci jednoho rozhraní.

Kryptopeněženka dává uživateli plnou kontrolu nad jeho prostředky a zajišťuje bezpečnost i pohodlnou práci s kryptoměnami.

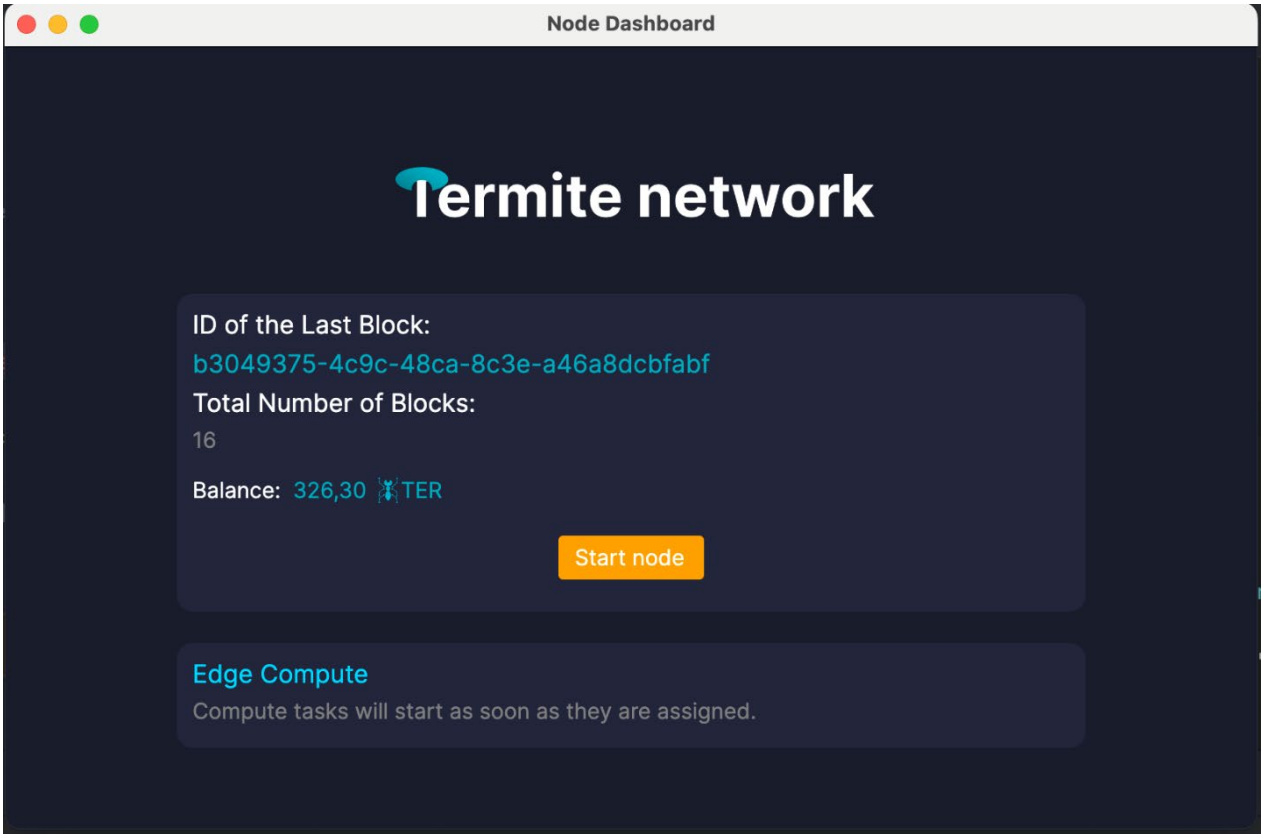
3.7 Desktopová aplikace

Popis uživatelské aplikace pro spuštění uzlu

Uživatelská aplikace pro spuštění uzlu poskytuje jednoduché a efektivní rozhraní pro správu a sledování činnosti uzlu v decentralizované síti. Tato aplikace spojuje všechny potřebné komponenty pro plnohodnotnou činnost uzlu, včetně klienta, serveru, API a těžby.

Hlavní funkce aplikace:

1. **Spuštění uzlu:** Stisknutím tlačítka Start node (jak je znázorněno na obrázku 10) aplikace spouští všechny části uzlu, včetně:
    a. Klientské a serverové části.
    b. API pro komunikaci s jinými uzly.
    c. Modul pro těžbu.
2. **Sledování zůstatku:** Uživatelé mohou sledovat zůstatek své peněženky.
3. **Informace o blocích:** Aplikace poskytuje údaje o aktuálním stavu blockchainu, včetně:
    a. Počtu bloků v blockchainu.
    b. Adresy posledního bloku, což uživatelům umožňuje sledovat aktuálnost sítě.
4. **Jednoduché rozhraní:** Rozhraní aplikace je zaměřeno na uživatele, s jednoduchými a přehlednými ovládacími prvky, které umožňují snadné spuštění a správu uzlu. Všechna hlavní data jsou zobrazena v přehledné a přístupné podobě.



Obrázek, 11: Jak vypadá desktopová aplikace

3.7.1 Příklad použití uzlu sítě

Uživatel, který si nainstaluje uzel sítě, poskytuje svůj výpočetní výkon pro podporu fungování decentralizované sítě. Během své činnosti uzel provádí následující úkoly:

- Zpracovává a ověřuje transakce.
- Šíří informace o nových blocích v síti.
- Podílí se na procesu těžby nebo validace bloků (v závislosti na mechanismu konsenzu).

Za to uživatel získává odměnu ve dvou formách:

1. **Poplatky za transakce** – uživatelé sítě platí poplatek za provádění transakcí a část těchto prostředků je rozdělena mezi uzly, které je zpracovávají.
2. **Odměna z blockchainu** – uzel může získávat dodatečné odměny, například za úspěšné vytvoření nového bloku, v závislosti na použitém konsenzuálním algoritmu (Proof of Work, Proof of Stake atd.).

Tímto způsobem instalace uzlu nejen podporuje stabilitu a bezpečnost sítě, ale také umožňuje získávat pasivní příjem za účast na jejím provozu.

4 Použité technologie

Při vývoji uzlu decentralizované sítě a uživatelské aplikace byly použity následující technologie:

4.1 Programovací jazyk

- **C#** – hlavní programovací jazyk pro vývoj serverové a klientské části uzlu.

4.2 Síťové technologie

- **TCP/IP** – používá se pro komunikaci mezi uzly sítě.
- **Web API** – komunikace s ostatními uzly prostřednictvím HTTP požadavků.

4.3 Bezpečnost a kryptografie

- **RSA** – šifrování dat a zajištění bezpečnosti připojení.
- **SHA-256, SHA-3** – hashovací algoritmy používané pro ochranu dat a zajištění integrity blockchainu.
- **RSA pro podepisování transakcí** – pro podepisování transakcí v kryptoměnové peněžence se používá algoritmus RSA s SHA-256.

4.4 Blockchain a Mining

- **Blockchain** – decentralizovaný a distribuovaný ledger (účetní kniha), který slouží k ukládání transakcí. Každý blok v blockchainu obsahuje transakce a je propojen s předchozím blokem pomocí hashovací funkce. Tato struktura zajišťuje transparentnost a imutabilitu dat. Blockchain je zajištěn pomocí kryptografie a konsensuálních mechanismů, což znamená, že všechny transakce jsou ověřovány a schváleny uzly v síti.
- **Proof-of-Work (PoW)** – algoritmus konsensu, který je využíván k ověřování transakcí a vytváření nových bloků v blockchainu. V tomto systému uzly v síti (tzv. těžaři) soutěží o to, kdo jako první vyřeší matematický problém, což znamená, že "těží" nový blok. Tento problém se nazývá "nonce" a těžaři musí provést výpočet, který obvykle zahrnuje hledání správné hodnoty (tzv. nonce), která splňuje podmínky s požadovanou mírou obtížnosti.

- **Měnová odměna a transakční poplatky** – těžaři, kteří úspěšně vyřeší matematický problém a přidají nový blok do blockchainu, jsou odměněni za svou práci. Odměna se skládá z dvou částí: bloková odměna (nově vytvořené mince) a transakční poplatky (poplatky spojené s transakcemi v daném bloku).
- **Těžební proces** – těžba bloků v systému PoW vyžaduje značnou výpočetní sílu a spotřebu energie. Tento proces nejen zajišťuje přidávání nových bloků do blockchainu, ale také chrání síť před podvody a manipulacemi.

4.5 Databáze

- **SQLite** – lehká relační databáze používaná pro ukládání dat o blockchainu, uzlech a transakcích. Díky malé velikosti a autonomní struktuře je ideální pro použití na uzlu.
- **Redis** – kešování dat a ukládání dočasných sesních klíčů.

4.6 Rámce a knihovny

- **Entity Framework Core** – ORM pro práci s databází.
- **Avalonia UI** – multiplatformní rámec pro tvorbu uživatelského rozhraní.
- **Newtonsoft.Json** – knihovna pro serializaci a deserializaci dat ve formátu JSON.

4.7 Vývoj pro mobilní zařízení

- **Android Studio** – vývojové prostředí pro vytváření mobilních aplikací pro Android.
- **Java** – programovací jazyk pro vývoj mobilní aplikace pro kryptoměnovou peněženku.
- **Google Gson** – knihovna pro práci s JSON ve mobilní aplikaci.

4.8 Správa verzí

- **Git** – systém pro správu verzí.
- **GitHub** – pro ukládání repozitáře a týmovou spolupráci na projektu.

4.9 Další technologie

- **QR kódy** – generování a skenování QR kódů pro usnadnění transakcí.
- **Logování** – systém logování pro sledování činnosti uzlu a analýzu událostí.

Tyto technologie zajišťují stabilní chod uzlu, bezpečnost dat a pohodlnou interakci s uživatelským prostředím.

5 Závěr

V rámci maturitního projektu zaměřeného na vytvoření decentralizované digitální měny "Termite" byl vyvinut funkční systém pro provádění finančních transakcí v síti. Tento systém umožňuje uživatelům bezpečně provádět převody kryptoměn mezi sebou a zároveň zajišťuje vysokou úroveň bezpečnosti a integrity dat prostřednictvím blockchainu.

V průběhu vývoje byly vytvořeny klíčové komponenty systému:

1. **Desktopová aplikace (Node)** - implementace uzlu v decentralizované síti, který se připojuje k dalším uzlům, šifruje komunikaci a zajišťuje distribuci dat mezi uzly. Tento modul také zahrnuje ověřování a validaci informací.
2. **Blockchain pro uchování historie transakcí** - blockchain obsahuje bloky, které uchovávají všechny provedené transakce, včetně mechanismu konsensu založeného na Proof of Work (PoW), což zajišťuje bezpečnost a nezměnitelnost uložených dat.
3. **Mobilní aplikace pro provádění finančních převodů** - mobilní aplikace umožňuje uživatelům provádět finanční převody mezi adresami, spravovat své adresy a zůstatky. Aplikace obsahuje šifrování pro ochranu citlivých dat a bezpečnost transakcí.

Během vývoje byl kladen důraz na bezpečnost systému a ochranu uživatelských dat, což bylo dosaženo použitím šifrování RSA a algoritmu SHA-256 pro podpisy transakcí.

V dalších fázích vývoje je možné systém rozšířit o nové funkcionality, jako jsou **smart kontrakty**, které umožní provádění automatizovaných a podmíněných transakcí, nebo integraci s **kryptoměnovými burzami (exchanges)** pro obchodování a výměnu digitálních aktiv.

Tento projekt nejen že naplňuje cíle uvedené v zadání, ale také ukazuje možnoti dalšího rozvoje v oblasti blockchain technologie a decentralizovaných financí.

6 Seznam literatury

- [1] Nakamoto, S. Bitcoin Whitepaper. 2008. [online]. – Pomohlo pochopit základní principy blockchainových technologií a jejich použití v kryptoměnách. Dostupné z: https://bitcoin.org/bitcoin.pdf
- [2] Antonopoulos, A. M. Mastering Bitcoin. 2017. [online]. – Použito pro studium bezpečnosti transakcí a vytváření kryptoměnových peněženek. Dostupné z: https://aantonop.com/books/

- [3] Shostak, A. Blockchain Basics. 2019. [online]. – Pomohlo pochopit základy blockchainu a jeho použití pro distribuované sítě. Dostupné z: https://www.amazon.com/Blockchain-Basics-Introduction-Blockchain/dp/1492051382
- [4] ChatGPT. 2025. [online]. – Použito pro získání rad ohledně implementace bezpečnosti a šifrování v projektu. Dostupné z: https://chat.openai.com/