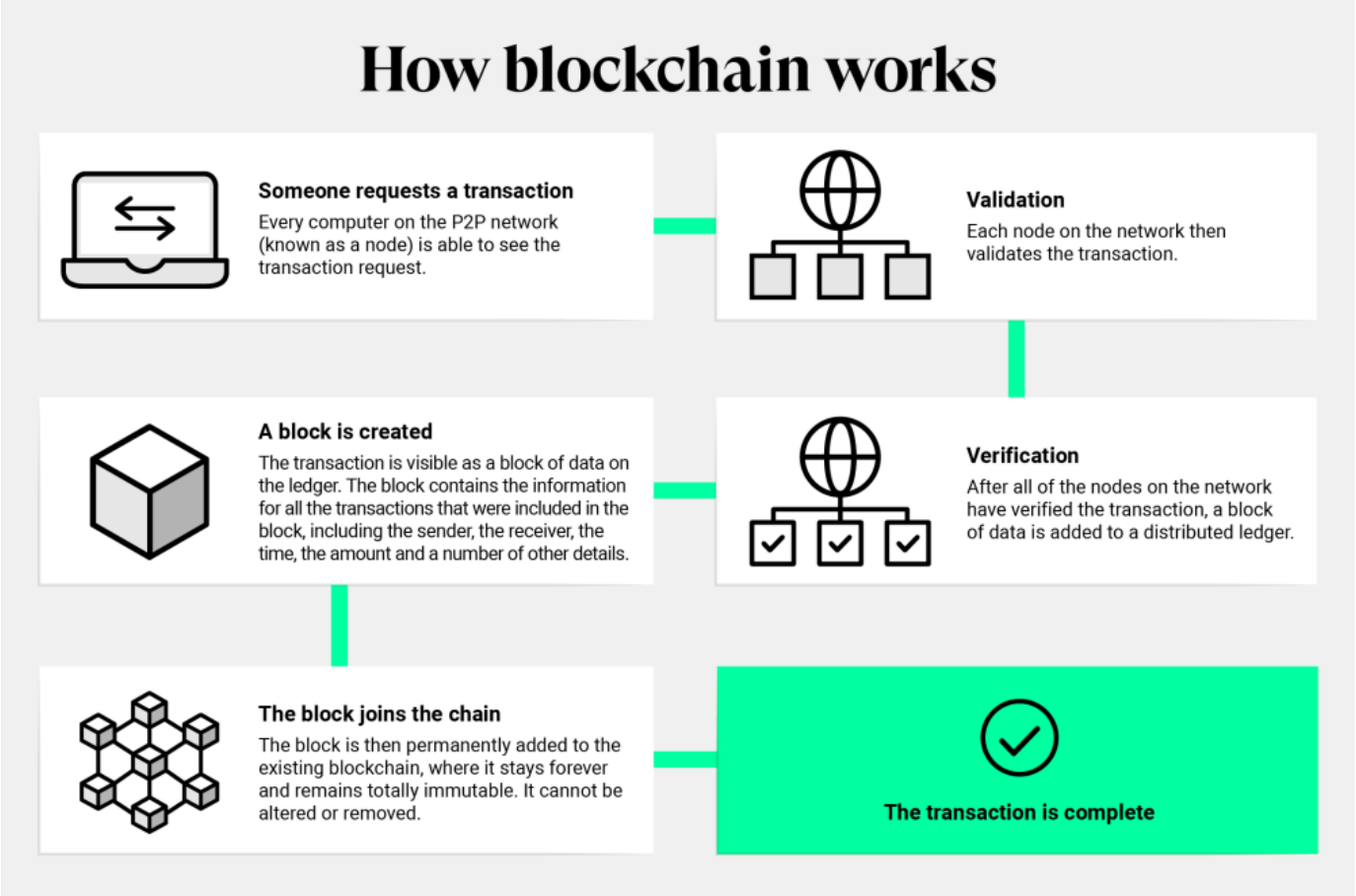
Mateusz Pieróg
Łukasz Sochacki
Jakub Klimczak
Michał Jasiński

Temat projektu:

Implementacja przykładowego systemu opartego na blockchain. System na służyć do obsługi zajęć prowadzonych przez wykładowcę. Zakładana funkcjonalność: rejestrowanie obecności, przetrzymywanie certyfikatów, oceny cząstkowe, oceny końcowe, protokoły z ocen.

Technologia:
Backend: Node.js, Express.js,
Frontend: React
biblioteki:
"body-parser": "^1.19.0",
"crypto-js": "^4.1.1",
“uuid”:^8.0.0

Opis działania blockchainu:

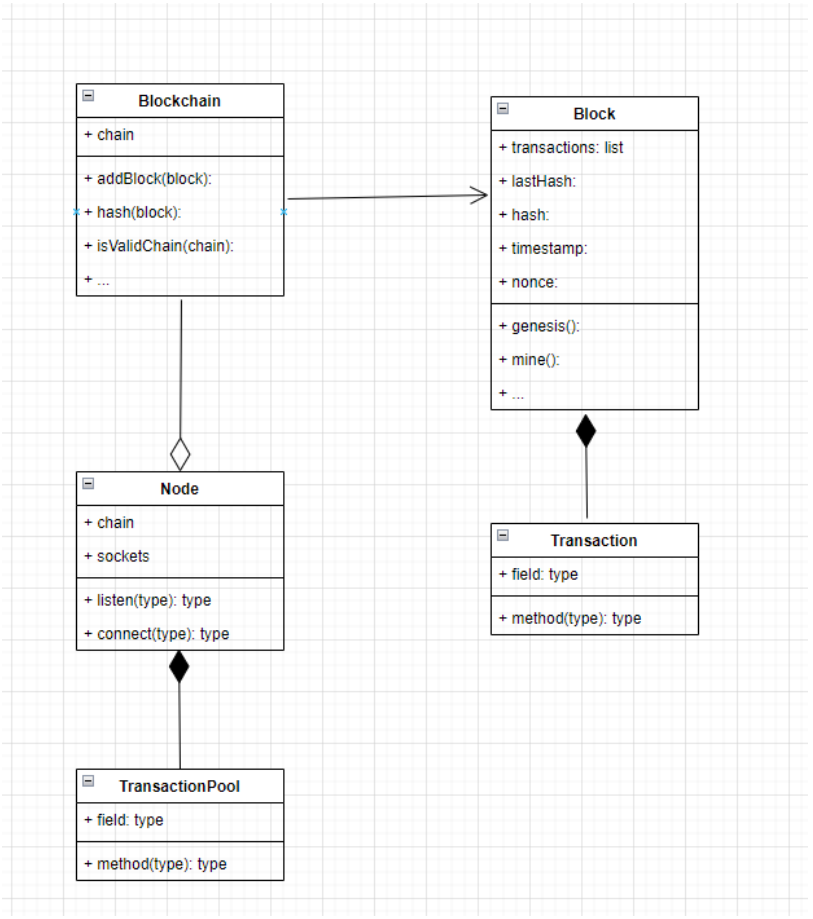


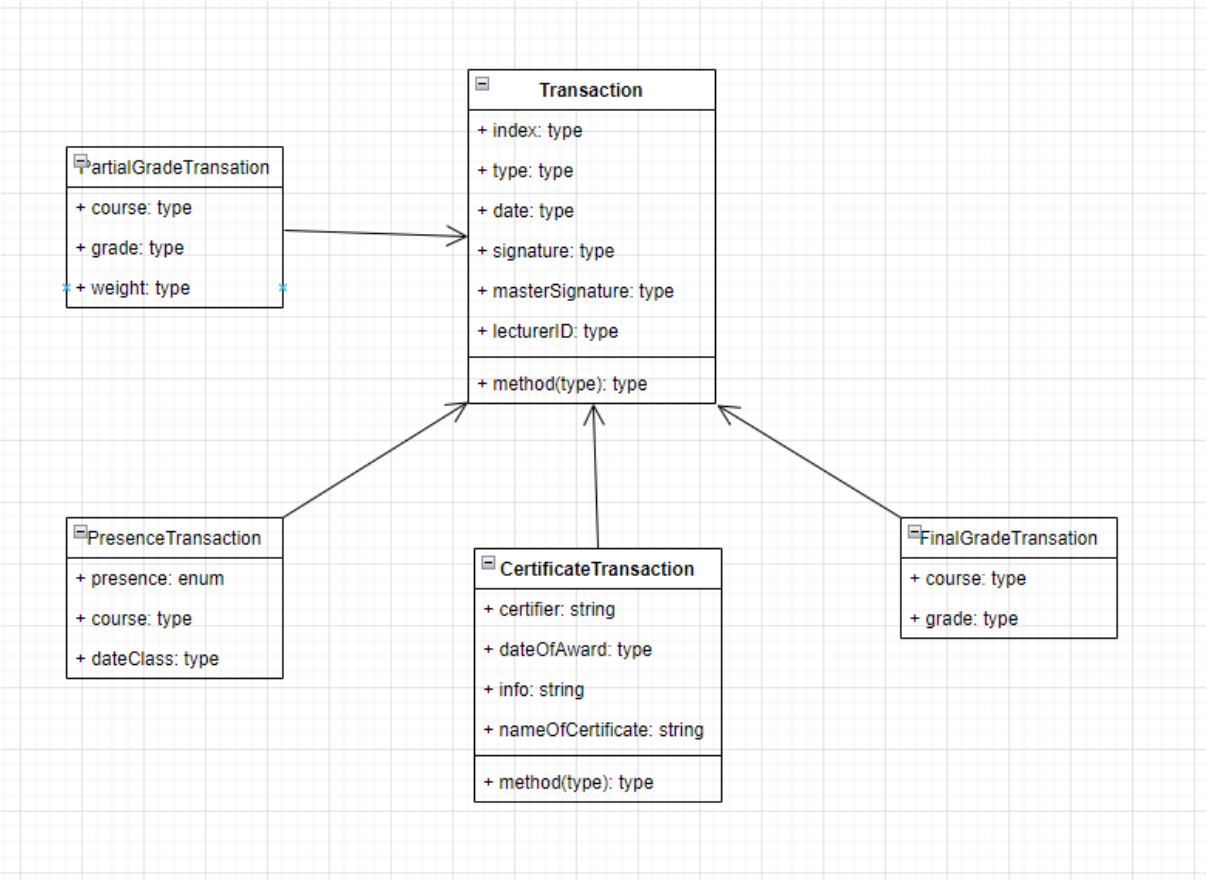
Ogólna idea naszego przykładowego systemu opartego o Blockchain wygląda następująco. Dowolny wykładowca może zadeklarować chęć utworzenia transakcji (przykładowo zarejestrowanie obecności studenta na zajęciach lub wpisanie jego oceny z danego przedmiotu).

Transakcja jest weryfikowana korzystając z klucza prywatnego. Następnie hashe kolejnych bloków są sprawdzane w taki sposób, że hash każdego bloku porównywany jest z hashem poprzedniego.

Po udanym procesie walidacji zostaje uruchomiony proces weryfikacji czyli dodania informacji zawartych w deklaracji transakcji do pewnego bloku, który jest tworzony i następnie dodawany do Blockchainu.

Po zakończeniu wszystkich operacji z powodzeniem transakcja zostaje uznana za zakończoną czyli dane utworzone przez wykładowcę zostały skutecznie dodane do naszego Blockchaina. W przypadku niezgodności hashów, peer pobiera z sieci prawidłowy łańcuch. Szyfrowanie które będzie używane to SHA256.


Blockchain
+ chain
+ addBlock(block):
+ hash(block):
+ isValidChain(chain):
+ ...
Block
+ transactions: list
+ lastHash:
+ hash:
+ timestamp:
+ nonce:
+ genesis():
+ mine():
+ ...
Node
+ chain
+ sockets
+ listen(type): type
+ connect(type): type
Transaction
+ field: type
+ method(type): type
TransactionPool
+ field: type
+ method(type): type


Transaction
+ index: type
+ type: type
+ date: type
+ signature: type
+ masterSignature: type
+ lecturerID: type
+ method(type): type
PartialGradeTransation
+ course: type
+ grade: type
+ weight: type
PresenceTransaction
+ presence: enum
+ course: type
+ dateClass: type
CertificateTransaction
+ certifier: string
+ dateOfAward: type
+ info: string
+ nameOfCertificate: string
+ method(type): type
FinalGradeTransation
+ course: type
+ grade: type

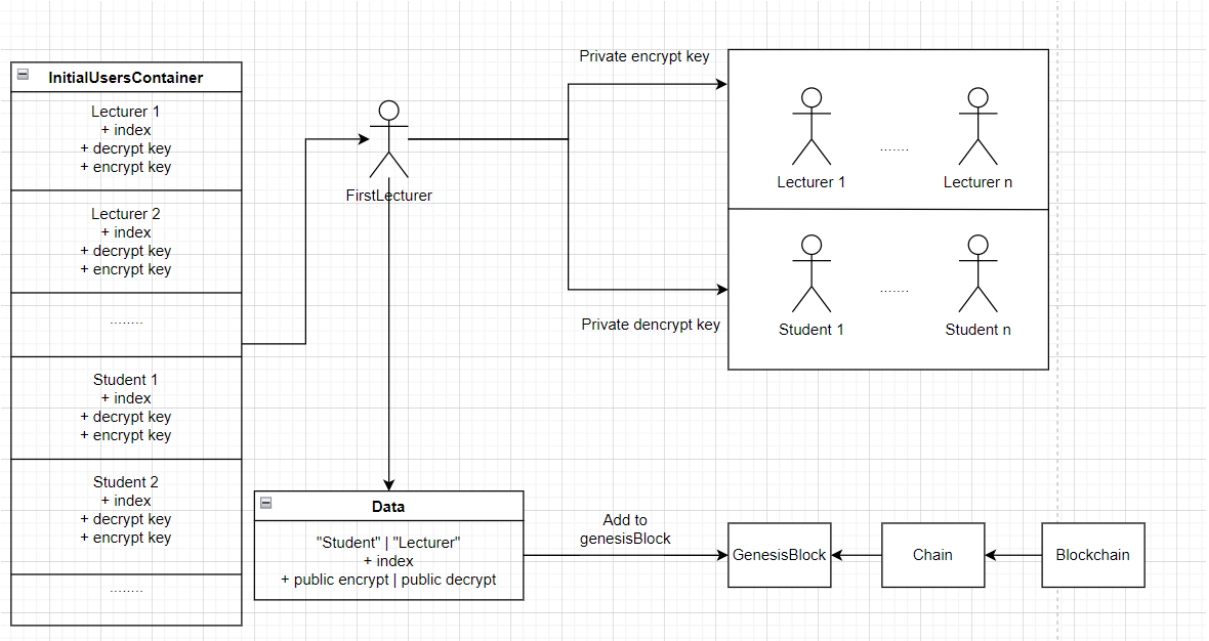
Zakładanie blockchaina

Wykładowca tworzy blok “genesis”, który zawiera informacje o:

- Osobach uprawnionych do wystawiania ocen:
    - Ich identyfikatory
    - Ich klucze deszyfrujące (służące do weryfikacji transakcji)
    - Kursy które prowadzą

  Wygenerowane klucze szyfrujące (do podpisywania transakcji) są prywatne i muszą być przekazane odpowiednim nauczycielom
- Studentach
    - Numery ich indeksów
    - Ich klucze szyfrujące (umożliwią nauczycielom zaszyfrowanie oceny studenta)
    - Kursy w których uczestniczą

  Klucze deszyfrujące (umożliwią studentowi odczyt swoich ocen) są prywatne i muszą być przekazane studentom



Finder

Proces odczytu danych wygląda następująco:

1. Przeszukiwane są wszystkie bloki po kolei, w blokach przeszukiwane są transakcje
2. Pole *signature* transakcji jest odszyfrowane kluczem prywatnym
3. Jeśli odszyfrowane pole zawiera numer indeksu danego studenta, znaczy że transakcja dotyczy jego, a obecność lub ocena jest zapamiętywana.
4. Jeśli odszyfrowane pole jest nieczytelne, jest pomijane gdyż transakcja nie dotyczy odczytującego studenta

Odczyt danych przez wykładowcę przebiega analogicznie, przy czym pole *masterSignature* jest szyfrowane jednym kluczem wykładowcy, zatem wykładowca może odczytać wszystkie identyfikatory studentów.

Podpisy

W transakcji znajdują się pola *signature* i *masterSignature*. Oba zawierają połączony identyfikator studenta i wartość timestamp/losową wartość nonce. Pole *signature* jest szyfrowane kluczem studenta, a pole *masterSignature* kluczem wykładowcy. Wartość obok identyfikatora ma na celu utrudnienie zdeterminowania przez osobę trzecią zaszyfrowanej wartości identyfikatora, ponieważ identyfikator i klucz są niezmienne. Ponadto, dane pozostaną bezpieczne w przypadku upublicznienia kluczy szyfrujących. Dzięki temu student może odczytać informacje jedynie o swoich obecnościach i ocenach, a wykładowca o wszystkich.

Weryfikacja

Pole *verification* w transakcji służy do zweryfikowania uprawnień dodającego transakcję. Wykładowca wpisuje w nie zaszyfrowany swoim kluczem prywatnym (innym niż tym służącym do szyfrowania ocen) dane: znak/słowo służące do sprawdzania poprawności i timestamp. Klucze deszyfrujące każdego wykładowcy są w bloku genesis, zatem jeśli transakcja była dodana przez wykładowcę #2, procedura weryfikacji będzie polegać na odszyfrowaniu pola verification kluczem deszyfrującym wykładowcy #2. Jeśli wartość po odszyfrowaniu jest poprawna, transakcja jest autentyczna.

Oryginalny blok


```
{
     "index":10,
     "timestamp":2,
"previousHash":"0c5f8a6689332ce6d5f30a2e3de4f34dfa8bb97dd25dc58153f6d6858742039a",
"hash":"e05dcdad02e9d653ef75e82d3f83cedd9b69339979485987c886ca0860ae5def",
     "nonce":1814,
     "data":[{
        "index": 0,
        "type":"PartialGrade",
        "date":"29/11/21",
        "signature":"07615278y72yrw0q2175g1qw",
        "masterSignature":"gw4893hy0ahg9aw3h04y42g",
        "verification":"jfgdso4542394203",
        "course": "Algebra",
        "grade":3,
        "weight":1

       },
       {
        "index": 1,
        "type":"Presence",
        "date":"29/11/21",
        "signature":"g3982qhyweg90ahw238t926",
        "masterSignature":"090hgew208h3gew2t3",
        "verification":"ijer034hfg0230",
        "presence": "present",
        "course":"Algebra",
```

```
        "dateClass": "28/11/21"
       }]
}
```

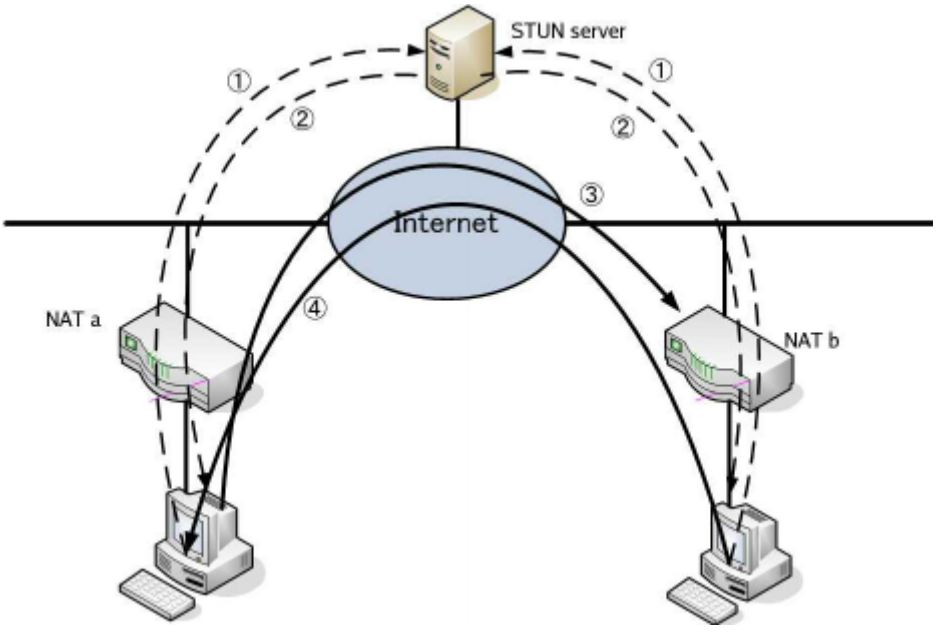
```
JSON.stringify()
```

```
"{"index":10,"timestamp":2,"previousHash":"0c5f8a6689332ce6d5f30a
2e3de4f34dfa8bb97dd25dc58153f6d6858742039a","hash":"e05dcdad02e9d
653ef75e82d3f83cedd9b69339979485987c886ca0860ae5def","nonce":1814
,"data":[{"index":0,"type":"PartialGrade","date":"29/11/21","sign
ature":"07615278y72yrw0q2175g1qw","masterSignature":"gw4893hy0ahg
9aw3h04y42g","course":"Algebra","grade":3,"weight":1},{"index":1,
"type":"Presence","date":"29/11/21","signature":"g3982qhyweg90ahw
238t926","masterSignature":"090hgew208h3gew2t3","presence":"prese
nt","course":"Algebra","dateClass":"28/11/21"}]}"
```

Z takiego napisu będzie obliczany hash bloku

Komunikacja

NAT hole punching

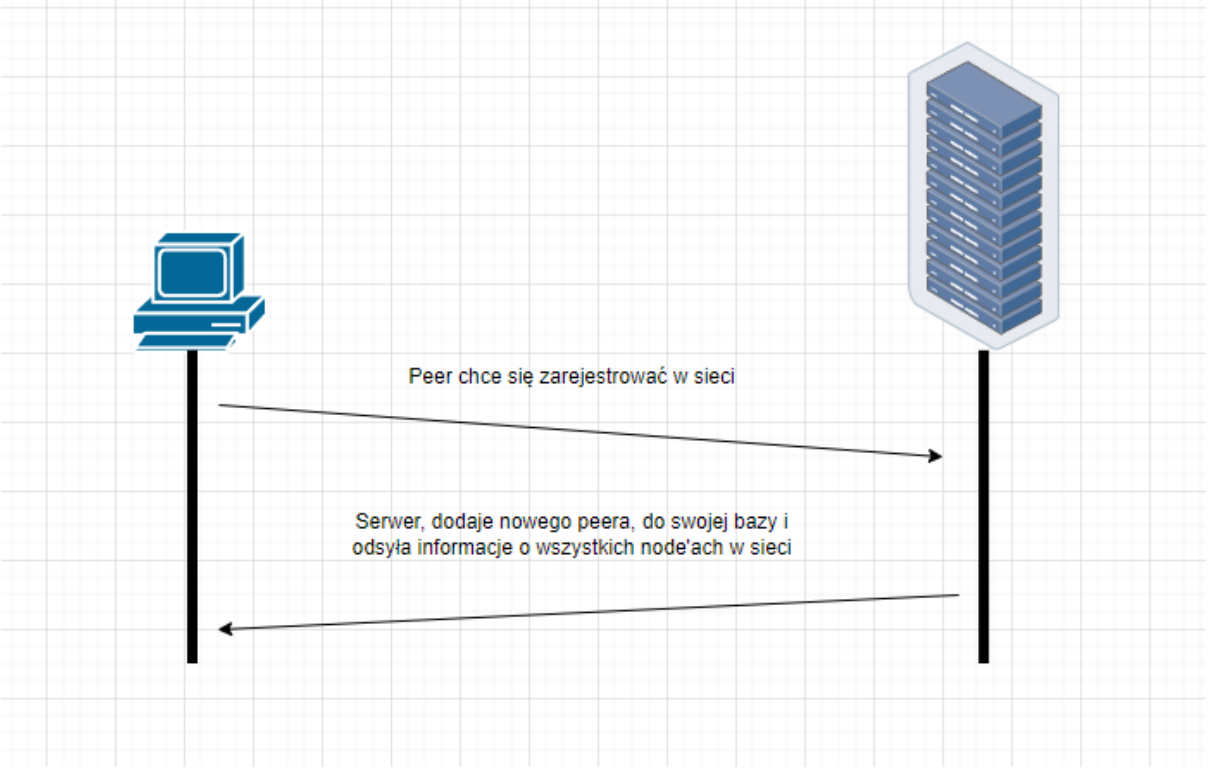


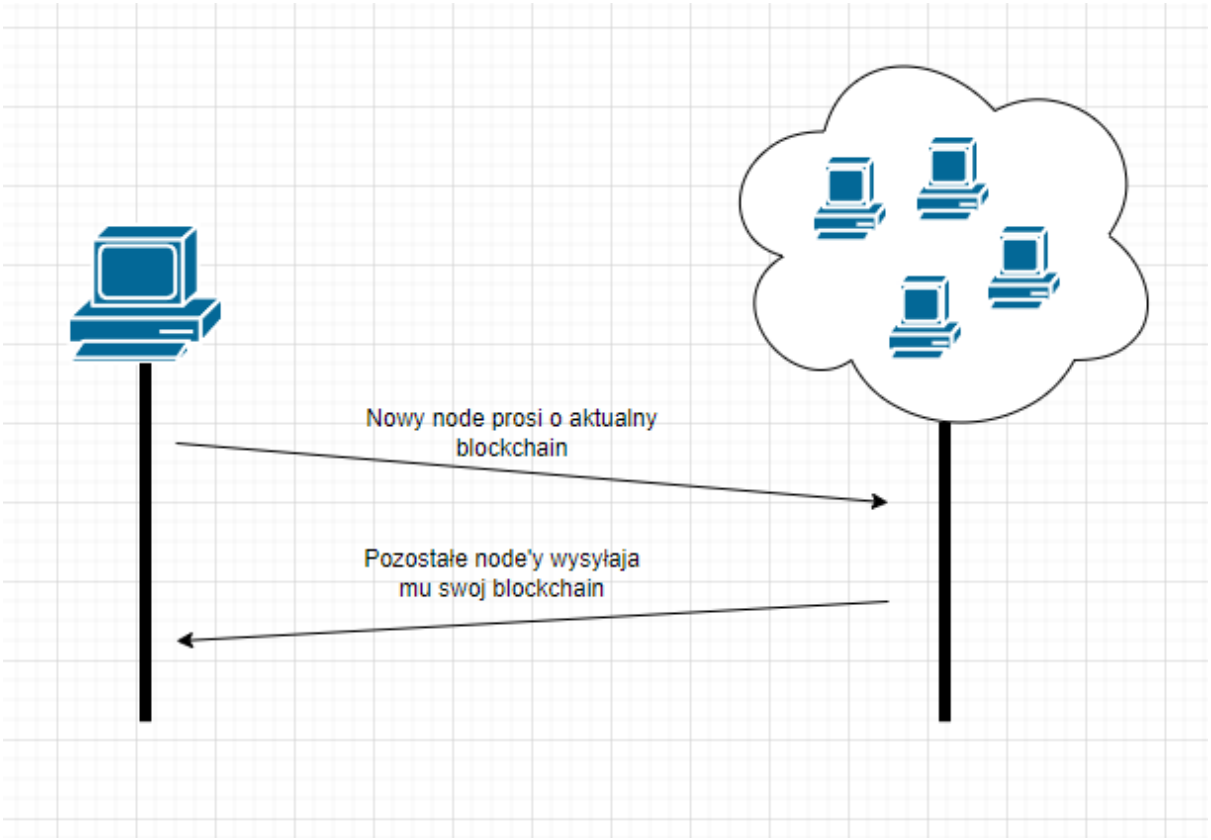
Hole punching służy do ominięcia zapory sieciowej, a co za tym idzie, umożliwienia peerom komunikacji między sobą. “Wybijanie dziury” wygląda przebiega następująco:

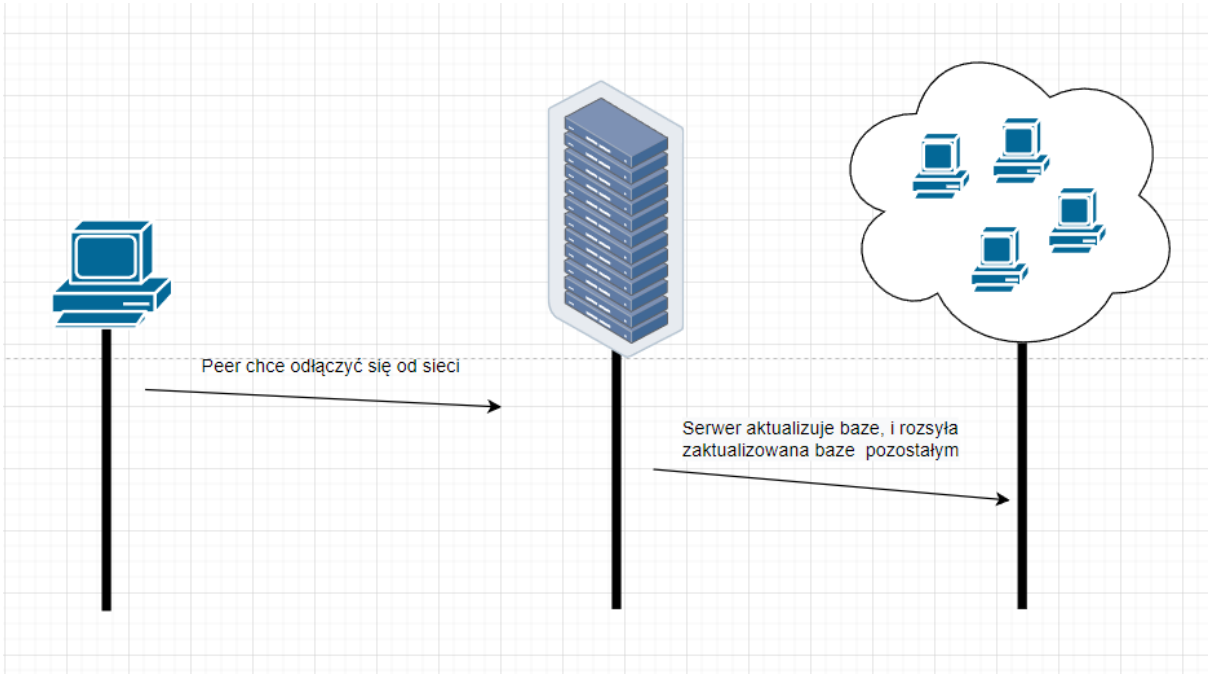
1. Peery wysyłają do serwera STUN żądania z ustalonymi portami. Serwer kojarzy publiczne adresy peerów z ich zewnętrznymi portami.
2. Serwer wysyła peerom zewnętrzne adresy ip wraz z portami, na jakich ma przebiegać komunikacja
3. Peery wysyłają do siebie wiadomości kierowane na odpowiednie porty.
4. Wiadomości są odbierane, gdyż traktowane są jako odpowiedź

Dołączanie do sieci

1. Peer wysyła do serwera żądanie rejestracji
2. Jako odpowiedź otrzymuje adresy (wszystkich/części) pozostałych peerów
3. Odtąd serwer będzie cyklicznie sprawdzał czy zarejestrowany peer jest aktywny
4. Nowo dołączony peer odpytuje peerów w sieci o obecny blockchain. Zaakceptowany może zostać blockchain otrzymany od 50% peerów w sieci oraz mający zgodne hashowanie.
5. Jeśli peer się rozłączy, serwer zauważy jego brak i prześle stosowną informację do pozostałych peerów







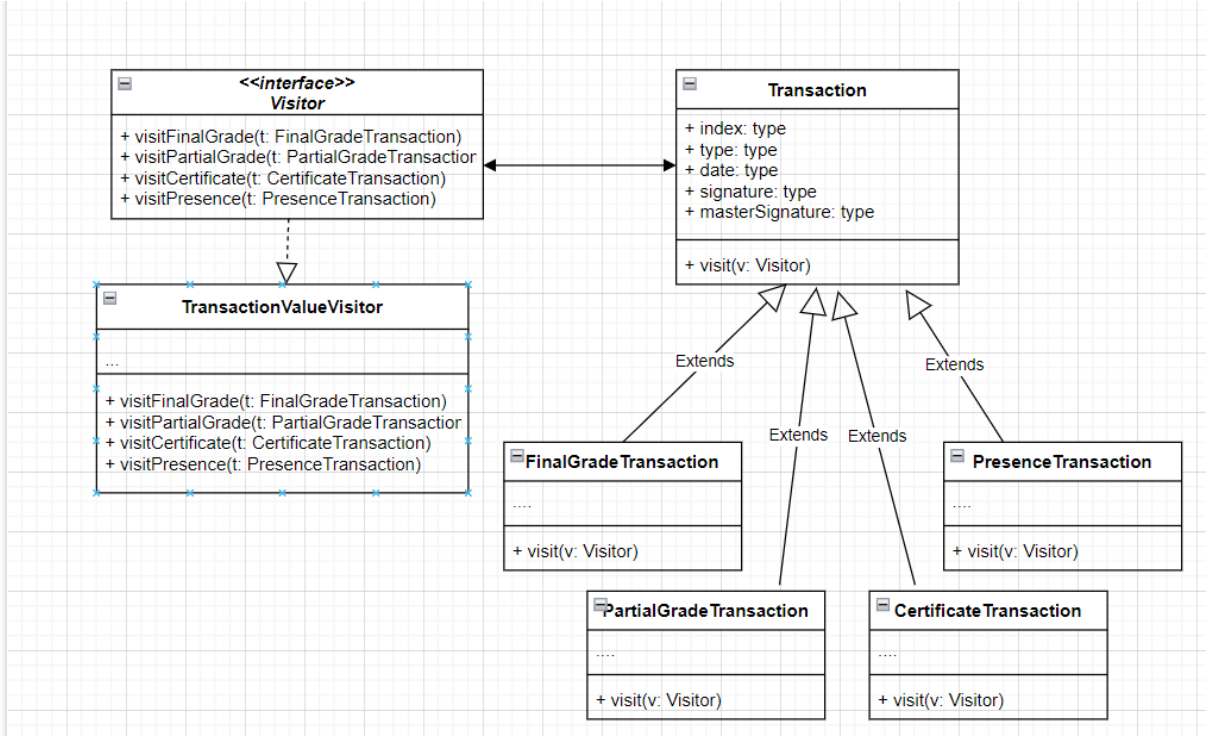
Ogłaszanie nowego bloku

1. Jeśli peer “wykopał” nowy blok, rozgłasza w sieci ten blok
2. Peery sprawdzają poprawność bloku i kompatybilność z istniejącym blockchainem
3. W przypadku konfliktu, peer z konfliktem odpytuje sieć o prawidłowy blockchain

Ogłaszanie nowej transakcji

1. Uprawniony do tego peer rozsyła informację o transakcji
2. Peery sprawdzają prawdziwość transakcji odszyfrowując ją odpowiednim kluczem (klucz danego wykładowcy w genesis bloku)
3. Transakcje fałszywe są odrzucane

Wzorce projektowe:
Visitor - pobieranie interesujących nas danych z transakcji, sprawdzenie poprawnosci blockchaina. Dzięki zastosowaniu visitora będziemy mogli wykonywać np. w jednym ciągu przeszukiwania osobne operacje dla różnych rodzajów transakcji.



Iterator - iteratory zwracające nastepne oceny, obecnosci itp. Dzięki zastosowaniu iteratorów w wygodniejszy sposób bedziemy mogli odszukiwać interesujące nas dane.

Unikniemy również powtarzania kodu.

Builder - na roznego rodzaju transakcje. Konstruktory transakcji składałyby się z wielu parametrów, przez co konstruktory byłyby długie i skomplikowane. Tworzenie transakcji za pomocą buildera będzie bardziej przejrzyste.