Spis treści

[1 Wstęp 4](#_Toc135515443)

[1.1 Cel 5](#_Toc135515444)

[1.2 Zakres pracy 5](#_Toc135515445)

[2 Zagadnienia teoretyczne 6](#_Toc135515446)

[2.1 Szyfrowanie 6](#_Toc135515447)

[2.1.1 Szyfrowanie jednokierunkowe 6](#_Toc135515448)

[2.1.2 Szyfrowanie dwukierunkowe 6](#_Toc135515449)

[2.1.2.1 Podpis cyfrowy 6](#_Toc135515450)

[2.2 DNS Service Discovery i Multicast DNS 7](#_Toc135515451)

[3 Projekt oraz implementacja 8](#_Toc135515452)

[3.1 Technologie 8](#_Toc135515453)

[3.1.1 .NET 7 8](#_Toc135515454)

[3.1.2 LiteDB 8](#_Toc135515455)

[3.1.3 SQLite 8](#_Toc135515456)

[3.1.4 Hangfire 8](#_Toc135515457)

[3.1.5 Serilog 8](#_Toc135515458)

[3.1.6 ASP .NET 9](#_Toc135515459)

[3.1.7 Swagger 9](#_Toc135515460)

[3.1.8 React 9](#_Toc135515461)

[3.2 Komponenty 10](#_Toc135515462)

[3.2.1 Blockchain 10](#_Toc135515463)

[3.2.1.1 Context 13](#_Toc135515464)

[3.2.1.2 PublicContext 13](#_Toc135515465)

[3.2.1.3 TempContext 13](#_Toc135515466)

[3.2.2 Networking 13](#_Toc135515467)

[3.2.2.1 Synchronizacja 13](#_Toc135515468)

[3.2.2.2 Blokowanie 15](#_Toc135515469)

[3.2.3 Model 17](#_Toc135515470)

[3.3 Aplikacja 18](#_Toc135515471)

[3.3.1 Synchronizacja 18](#_Toc135515472)

[3.3.2 Tworzenie użytkownika 18](#_Toc135515473)

[3.3.3 Tworzenie postów 21](#_Toc135515474)

[3.3.4 Format przechowywanych danych 22](#_Toc135515475)

[4 Bezpieczeństwo danych 24](#_Toc135515476)

[4.1 Dane początkowe 24](#_Toc135515477)

[4.2 Test – Usunięcie obiektu 24](#_Toc135515478)

[4.3 Test – Modyfikacja 25](#_Toc135515479)

[5 Bezpieczeństwo aplikacji 26](#_Toc135515480)

[5.1 Dane początkowe 26](#_Toc135515481)

[5.2 Test – Nieautoryzowane odblokowanie 26](#_Toc135515482)

[5.3 Test – Nieautoryzowane potwierdzenie 27](#_Toc135515483)

[6 Podsumowanie 28](#_Toc135515484)

[7 Spis obrazków 29](#_Toc135515485)

[7.1 Diagramy UML 29](#_Toc135515486)

[7.2 Rysunki 29](#_Toc135515487)

[7.3 29](#_Toc135515488)

[7.4 Schematy blokowe 29](#_Toc135515489)

[7.5 Zrzuty ekranu 29](#_Toc135515490)

[8 Bibliografia 30](#_Toc135515491)

# Wstęp

Od kilkunastu lat znaczącą popularnością cieszy się technologia blockchain. Stało się to głównie za sprawą kryptowalut takich jak Bitcoin (BTC) (1), które ją wykorzystują do działania.

Bitcoin został przedstawiony w 2008 przez tajemniczą osobę lub grupę osób korzystającą z pseudonimu Satoshi Nakamoto (2). Zgodnie z przedstawionymi założeniami jest to sieć typu peer-to-peer (3), gdzie każdy użytkownik jest sobie równy i każdy z nich przetrzymuje informacje o transakcjach przeprowadzanych w sieci. Informacje te są przechowywane w postaci zdecentralizowanego, publicznego rejestru księgowego, w którym każdy rekord jest powiązany z poprzednim. Tworzy to łańcuch nazywany blockchain’em. Dzięki decentralizacji waluta nie może być w żaden sposób kontrolowana przez państwa, urzędy i jednostki. W połączeniu z brakiem możliwości modyfikacji wpisów, na pierwszy rzut oka mogłoby się wydawać, że jest to idealny środek płatniczy. Dowolna osoba na świecie, która ma połączenie do internetu może anonimowo przeprowadzać transakcje z osobami po drugiej stronie świata z minimalnym opóźnieniem, minimalną prowizją oraz brakiem podatków (4). Niestety brak kontroli ma też swoje wady, możliwe jest bezkarne przeprowadzanie oszustw od prania pieniędzy po piramidy finansowe obiecujące niemożliwe do zrealizowania oprocentowania (5), które w dzisiejszych czasach są bardzo trudne do zrealizowana przy użyciu tradycyjnych walut.

Kryptowaluty zaczęły się szybko rozwijać, ich wartość rosła, a kopacze kryptowalut (6) rośli w siłę. Pojęcie kopania kryptowalut opisuje mechanizm, w którym urządzenia należące do użytkowników sieci ścigają się w znalezieniu odpowiedzi aby potwierdzić transakcję i otrzymać nagrodę.

Działa to w następujący sposób: sieć użytkowników ustala wartość nazywaną trudnością. Jest ona uzależniona od zdolności obliczeniowej całej sieci. Każdy nowo stworzony blok łańcucha blockchain zawiera pole nazwane „Nonce” – „Number used only once”. Wartość dla tego pola jest odpowiedzią jakiej szukają kopacze. W celu znalezienia odpowiedzi każdy kopacz generuje przypadkowe liczby, a następnie aplikuje je kolejno do tego pola i generuje skrót (2.1.1 Szyfrowanie jednokierunkowe) całego bloku. Jeżeli wartość skrótu jest mniejsza lub równa wartości aktualnej trudności to odpowiedź została znaleziona. W nagrodę użytkownik lub ich grupa otrzymuje nagrodę, gdy jako pierwsi rozwiążą zadanie.

Przetworzenie tych operacji w przypadku BTC wymaga bardzo dużej mocy obliczeniowej, a zatem dużej ilości energii. Ze względu na fakt, że otrzymywana nagroda znacząco przeważała koszt energii wielu kopaczy bez większego zastanowienia podejmowało decyzję o przystąpieniu do wyścigu. Ta tendencja spowodowała, że w zeszłym roku pobór mocy przeznaczony tylko i wyłącznie do kopania kryptowalut osiągnął 127 TWh (7). Jest to więcej niż całe zużycie energii Norwegii, a w kontekście globalnym jest to 0,55% poboru mocy całego świata.

Pomimo złej sławy jaką cieszą się kryptowaluty w niektórych sektorach (8), w pewnym momencie technologia blockchain stała się na tyle popularna, że każdy biznes, który skupiał się na byciu na czasie chciał wykorzystać blockchain w swoich produktach. Z reguły nie miało to większego sensu, ponieważ ta technologia jest odpowiednia tylko dla wąskiej grupy zastosowań. Należą do niej aplikacje typowo zdecentralizowane oraz aplikacje, których dane z założenia powinny być często audytowane.

## Cel

Ta praca ma na celu przedstawienie projektu oraz implementacji aplikacji wykorzystującej kluczowe mechanizmy technologii blockchain. Będzie to przykład właściwego zastosowania wspomnianej technologii oraz pokazanie jej wad i zalet.

## Zakres pracy

Zostanie zaprojektowana oraz zaimplementowana baza danych typu blockchain, która będzie działać w oparciu o istniejącą bazę danych typu NOSQL.

Następnie powstanie biblioteka zawierająca mechanizmy pozwalające na zdecentralizowanie aplikacji w lokalnej sieci. Aplikacja za pomocą tej biblioteki będzie mogła odnaleźć inne komputery, które mają uruchomioną instancję tej samej aplikacji bez wykorzystania serwera pośredniczącego oraz wymieniać informacje z tymi instancjami.

Na koniec, bazując na stworzonych komponentach, dobudowana zostanie aplikacja webowa, która umożliwi użytkownikowi interakcję z aplikacją. Strona ma imitować w bardzo limitowany sposób znaną na całym świecie aplikację jaką jest Twitter (9). Użytkownik będzie mógł zarejestrować nazwę swojego profilu oraz wysyłać i odbierać wiadomości (posty).

Decentralizacja aplikacji pod niektórymi względami jest lepsza od tradycyjnej architektury klient-serwer, ponieważ zapewnia do pewnego stopnia niezależność od wszelkich zewnętrznych czynników. Przykładami takich czynników są na przykład finanse – architektura klient-serwer dla aplikacji mających wielu użytkowników jest dużym problem, aby zapewnić nieprzerwane działanie aplikacji konieczne jest stałe źródło finansowania. W przypadku aplikacji Facebook firmy Meta nowe centra danych są ciągle dobudowywane w celu zachowania optymalnej mocy obliczeniowej oraz przestrzeni dyskowej (10). Decentralizacja aplikacji zapewnia praktycznie nielimitowaną skalowalność, ponieważ to użytkownicy zapewniają infrastrukturą. Jednak idą za tym pewne wady, nie każdy użytkownik musi mieć dobre połączenie z Internetem, a zatem doznania tego użytkownika oraz wszystkich porozumiewających się z instancją jego aplikacji będą pozostawiały dużo do życzenia. Dodatkową wadą jest ograniczona przestrzeń dyskowa, jeżeli aplikacja ma być również mobilna to przechowywane w blockchainie dane muszą być ograniczone do minimum.   
Innym przykładem mogą być czynniki polityczne - organizacjom politycznym jest bardzo ciężko zablokować dostęp do zdecentralizowanej aplikacji oraz kontrolować przetwarzane dane, ponieważ w zdecentralizowanej aplikacji nie ma serwera, którego adres IP można po prostu zablokować. Jedyną możliwością jest analiza ruchu sieciowego i blokowanie pakietów, które zostały zidentyfikowane jako te należące do danej aplikacji.

Technologia blockchain jest kluczowa dla decentralizacji tej aplikacji. Bez niej przechowywane dane nie mogłaby być bezpieczne będąc rozproszone pomiędzy wielu użytkowników. Każdy z nich mógłby w swojej wersji danych wprowadzić modyfikacje, które byłyby niewykrywalne.

# Zagadnienia teoretyczne

## Szyfrowanie

Od wieków ludzie próbują ukryć poufne informacje przed innymi. W tym celu powstało wiele algorytmów szyfrujących. Pozwalają one na zniekształcenie informacji w taki sposób, aby tylko właściwe osoby mogły ją wykorzystać.

### Szyfrowanie jednokierunkowe

Szyfrowanie jednokierunkowe cechuje się tym, że informacja poddana działaniu algorytmu szyfrującego (funkcji haszującej) prawdopodobnie nigdy nie będzie mogła zostać odszyfrowana. Mogłoby się wydawać, że tego typu szyfr jest zbędny, jednak jest bardzo przydatny, do przechowania haseł oraz innego rodzaju poufnych informacji, których nikt oprócz właściciela nie powinien znać.

Przykładem takiej funkcji haszującej – inaczej funkcji skrótu jest algorytm SHA. Został stworzony przez NSA, pierwsza wersja algorytmu została opublikowana w 1993 roku. Ze względu na znalezione luki bezpieczeństwa w ciągu kolejnych 8 lat algorytm został zaktualizowany i nazwany SHA-2. Tej wersji używamy do dziś. (10)

### Szyfrowanie dwukierunkowe

W przypadkach gdy odbiorca ma być w stanie odczytać oryginalną wersję informacji stosuje się szyfrowanie dwukierunkowe.

Jednym z algorytmów, który pozwala na taki efekt jest algorytm RSA. Stworzyli go Rona Rivesta, Adiego Shamina oraz Leonard Adleman w 1977 roku.

Z założenia algorytm wykorzystuje dwa klucze, publiczny oraz prywatny. Ten pierwszy pozwala na zaszyfrowanie informacji, jego zaletą jest to, że nie jest poufny – można go udostępnić każdemu. Klucz prywatny natomiast pozwala na zaszyfrowanie informacji i powinien być znany tylko odbiorcy informacji. Klucz prywatny może również posłużyć do wygenerowania klucza publicznego – klucza publicznego natomiast nie może użyć do wygenerowania klucza prywatnego.

#### Podpis cyfrowy

Dodatkową zaletą algorytmu RSA jest możliwość wykonania podpisu cyfrowego. Taki podpis jest tworzony za pomocą klucza prywatnego i może zostać zweryfikowany poprzez użycie powiązanego klucza publicznego.

Korzystając z podpisu cyfrowego można zweryfikować autora informacji oraz to czy informacja została zmodyfikowana w międzyczasie.

## DNS Service Discovery i Multicast DNS

Założenie, że aplikacja będzie działać jedynie w sieci lokalnej, umożliwiło użycie dwóch serwisów sieciowych: mDNS oraz DNS SD.

DNS Service Discovery pozwala klientowi sieci znaleźć listę nazw instancji danego serwisu. Każda instancja serwisu jest opisana za pomocą rekordu DNS SRV oraz DNS TXT. Klient wyszukuje instancje korzystając z rekordu DNS PTR (11).

Multicast DNS umożliwia przetworzenie nazwy domenowej do adresu IP. Serwis nie wymaga żadnej konfiguracji. Wykorzystuje do działania User Datagram Protocol (protokół UDP). Został pierwszy raz zaproponowany przez Bill Woodcock i Bill Manning w IETF w 2000 roku. Trzynaście lat później został wdrożony do IETF RFC 6762 przez Stuarta Cheshire oraz Marc Krochmal. (12)

Łącząc funkcjonalności obu serwisów zasada działania wygląda następująco – każda instancja serwisu nasłuchuje portu 5353, na który przekazywane są zapytania multicast zawierające rekordy DNS. Instancja odpowiada we właściwy sposób na odpowiednio otrzymany rekord DNS.

# Projekt oraz implementacja

## Technologie

### .NET 7

Do implementacji aplikacji, poza częścią interfejsu użytkownika, została wykorzystana najnowsza, w pełni wdrożona wersja najpopularniejszej platformy Microsoft’u - .NET 7. Dostarcza ona język C# w wersji 11. Jest to wersja, o krótkim okresie wsparcia (13).

### LiteDB

Aplikacja potrzebowała bazy danych typu NOSQL. Wybrana została implementacja LiteDB (14), głównie ze względu na fakt, że umożliwia zapisanie danych w postaci pliku. Dodatkowo dane mogą być przetwarzane za pomocą kwerend biblioteki Linq (15), które w znaczący sposób ułatwia pracę z danymi na najnowszych wersjach platformy .NET.

### SQLite

Dodatkowo, ze względu na potrzebę wykorzystania Hangfire’a została wykorzystana również typowa baza danych typu SQL. W tym przypadku został wybrany SQLite (16), który również umożliwia zapisanie danych do pliku.

### Hangfire

Hangfire to komponent dla platformy.NET, który umożliwia na przetwarzanie zadań równolegle. Pozwala na normalne uruchomienie zadania jak lub jego zaplanowanie. Kontrolowane jest również to czy zadanie zostało zrealizowane pozytywnie, w zależności od konfiguracji zadania, jeżeli wystąpi niepowodzenie ponowna próba wykonania zadania może nastąpić określoną ilość razy po określonej ilości czasu opóźnienia. Ta funkcjonalność jest kluczowa dla działania aplikacji. (17)

### Serilog

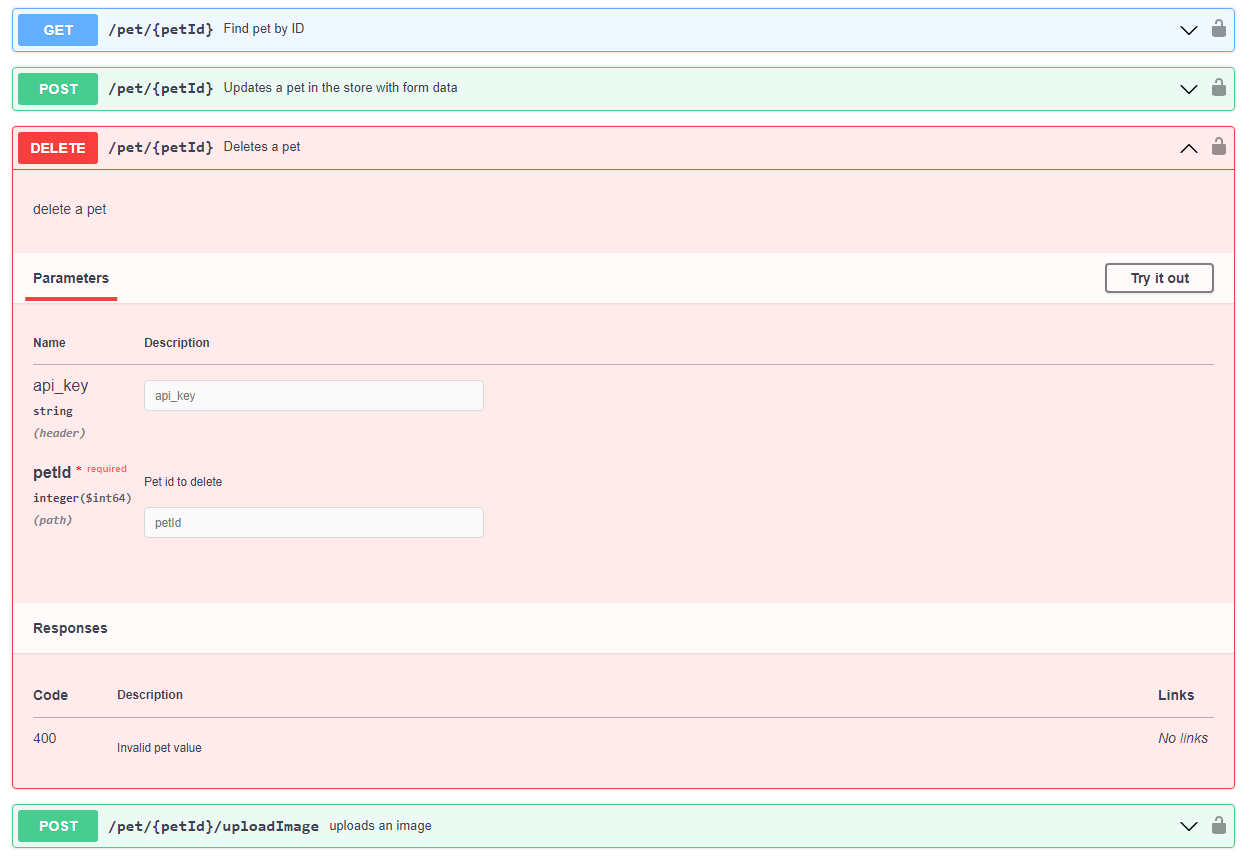
Serilog jako komponent platformy .NET oferuje przejrzyste formatowanie logów zwracanych przez aplikację platformy .NET. Umożliwia na zapisanie stworzonych przez aplikację logów na różne sposoby, w tym zapisanie do pliku tekstowego lub bazy danych (18)

### ASP .NET

Ze względów implementacyjnych, aby pozwolić aplikacji na przetwarzanie żądań typu http wykorzystana została również platforma ASP.NET (19).

### Swagger

W przypadku tworzenia aplikacji web’owych popularnym i pomocnym narzędziem jest komponent platformy .NET – Swagger. Pozwala on przede wszystkim na tworzenie specyfikacji interfejsu REST API zawartego w aplikacji, według standardu OpenAPI. Dodatkowo umożliwia na przystępne dla programisty testowanie działania punktów dostępowych interfejsu w czasie tworzenia oprogramowania. Zrzut ekranu 1 przedstawia przykład takiego interfejsu. (20)



Zrzut ekranu Przykładowy interfejs użytkownika komponentu Swagger (źródło zewnętrzne)

### React

Do wykonania interfejsu użytkownika wykorzystana została platforma React, tworzona w języku JavaScript. Platforma ta pozwala na tworzenie widoków, które umożliwiają definiowanie zmiennych o modyfikowalnym stanie. Co oznacza, że w przypadku aktualizacji takiej zmiennej, widok zostaje ponownie wyrenderowany.

Dodatkowo aby zapewnić schludność aplikacji dodana została biblioteka Reactstrap. Jest ona oparta o Bootstrap 5, jest to biblioteka zawierająca definicje elementów HTML wraz z implementacjami zachowania oraz wyglądu w odpowiednio JavaScript oraz CSS (21).

## Komponenty

W celu zachowania przejrzystości kodu, główne funkcjonalności aplikacji zostały podzielone na trzy komponenty: Model, Networking oraz Blockchain. Zależności między nimi są przedstawione na Rysunek 1.

Obraz zawierający diagram

Opis wygenerowany automatycznie

Rysunek Zależności pomiędzy komponentami aplikacji (opracowanie własne)

### Blockchain

Pierwszym z nich jest Blockchain. Podstawowym zadaniem tego komponentu jest bezpośrednia kontrola bazy danych LiteDB. Jest to kluczowe ze względu na fakt, że baza danych jest przechowywana w postaci pliku, więc może być modyfikowana przez tylko jeden wątek. Dlatego implementacja zawiera dedykowaną klasę, która zapewnia dostęp do danych przechowywanych w bazie dla wszystkich pozostałych elementów komponentu.

Komponent odpowiada również za odczytanie lub w przypadku, gdy nie istnieją stworzeniu dwóch kolekcji NOSQL – chain oraz temp. Jak nazwy wskazują chain to kolekcja zawierająca w pełni potwierdzony i spójny łańcuch danych, natomiast temp zawiera ogniwa łańcucha, które nie zostały jeszcze potwierdzone i oczekują na to.

Głównym zadaniem komponentu jest realizacja operacji oraz przeprowadzanie weryfikacji danych zawartych w bazie danych typu blockchain.  
Stworzony model danych (Diagram UML 1) jest bardzo prosty, składa się z trzech klas:

* Link – reprezentuje ogniwo łańcucha informacji, zawiera podstawowe dane takie jak poprzednie ogniwo oraz datę utworzenia, a także dane, które mają być zapisane w bazie danych.
* Lock – ta klasa jest odpowiedzialna za identyfikację czy dane ogniwo zostało zablokowane, przez kogo, jakie będzie kolejne ogniwo, kiedy blokada wygasa i czy blokada została potwierdzona.
* Signature – jest to klasa, która zawiera informację pozwalające na weryfikację danego jak i poprzedzającego ogniwa, dodatkowo identyfikuje twórcę.

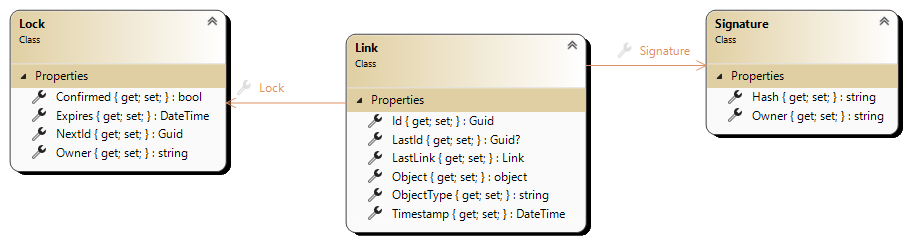


Diagram UML Zależności między klasami modelu komponentu Blockchain (opracowanie własne)

Mechanizmy technologii blockchain zostały zaimplementowane w kontekstach (Diagram UML 2), które mają przewidziane zastosowania.

* Context
  + PublicContext
  + TempContext
    - CreateContext
    - SyncContext
    - LockContext

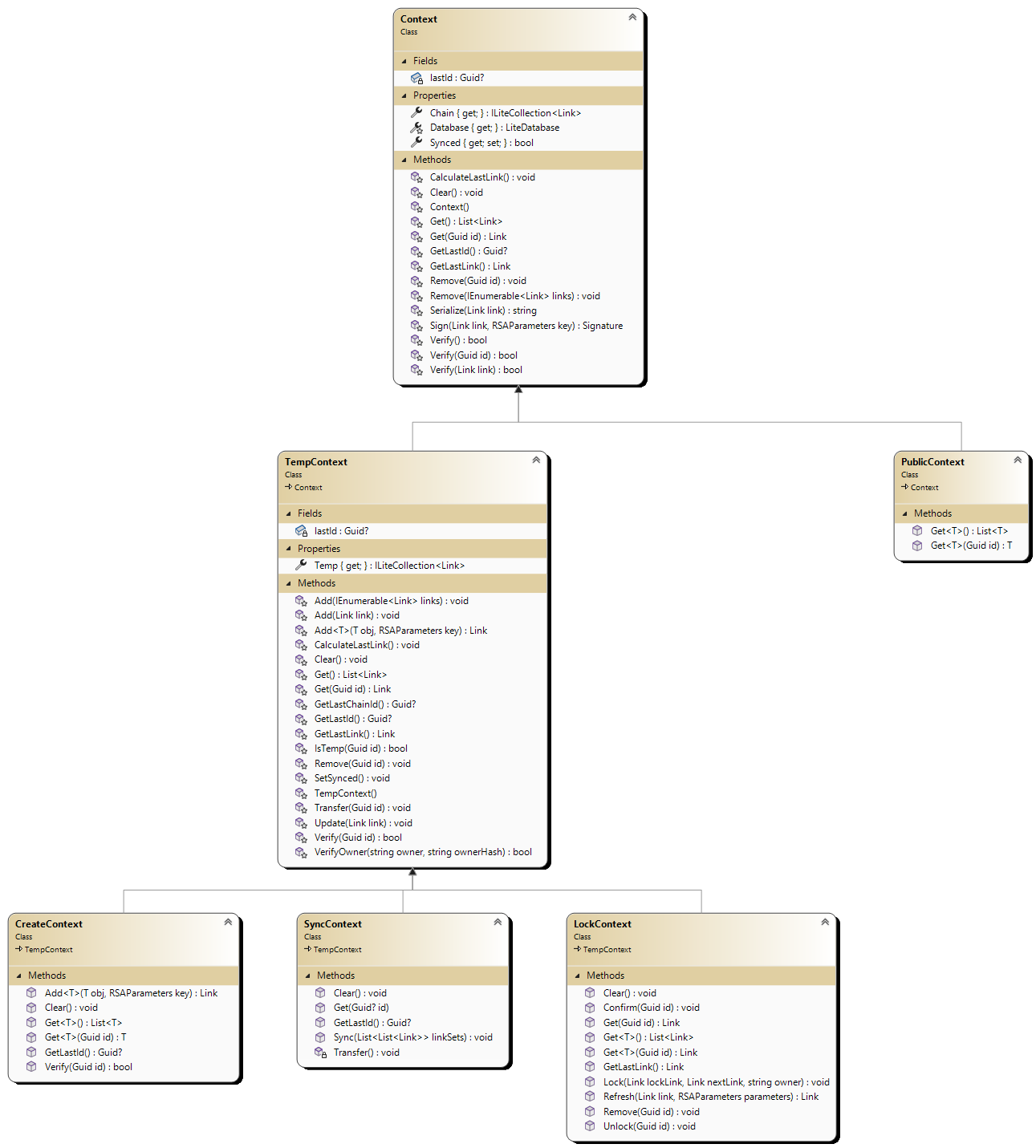


Diagram UML Zależności pomiędzy kontekstami komponentu Blockchain (opracowanie własne)

#### Context

Klasa zabezpieczona, zawiera podstawowe, potrzebne dla pozostałych kontekstów informacje, takie jak:

* ostatni identyfikator ogniwa w łańcuchu
* flaga potwierdzająca synchronizację bazy danych – gotowość do przyjęcia operacji od użytkownika
* referencje do obiektu bazy danych oraz kolekcji chain

Implementuje również podstawowe akcje takie jak:

* pobieranie ogniwa na podstawie identyfikatora
* usunięcie ogniwa na podstawie identyfikatora

#### PublicContext

Klasa publiczna, rozszerza Context. Przeznaczona do pobierania danych na podstawie podanego typu przechowywanych przez ogniwo danych.

#### TempContext

Klasa zabezpieczona, rozszerza Context. Wprowadza pojęcie tymczasowego łańcucha danych, który wykorzystywany jest przed wdrożeniem nowego ogniwa do głównego łańcucha. Nadpisuje odpowiednio metody, biorąc pod uwagę dodatkową kolekcję.

### Networking

Komponent Networking jest odpowiedzialny za komunikację pomiędzy węzłami. Wykorzystuje w tym celu protokół mDNS (Multicast DNS) oraz DNS-DC (DNS Service Discovery).

W momencie uruchomienia aplikacji, zostaje wysłane zapytanie o wszystkie dostępne węzły. W przypadku otrzymania odpowiedzi, wysłana zostaje prośba o doprecyzowanie adresu IP. Po otrzymaniu adresu węzeł zostaje dodany do listy.

#### Synchronizacja

Równolegle wraz z poszukiwaniem węzłów w sieci, Hangfire otrzymuje zadanie, aby uruchomić mechanizm synchronizacji ze znalezionymi węzłami. Z założenia zadanie może zakończyć się niepowodzeniem pięć razy, każde ponowne uruchomienie opóźnione jest o 10 sekund. W ten sposób synchronizacja może zając maksymalnie minutę. Po tym czasie przyjęte zostaje, że aplikacja jest zsynchronizowana, ponieważ jest jedynym aktywnym węzłem w sieci.

Uruchomione zadanie sprawdza ile węzłów zostało znalezionych. Jeżeli żaden węzeł nie został znaleziony, zadanie kończy się niepowodzeniem. W przypadku, gdy węzły zostały znaleziony, do każdego z nich zostaje wysłane zapytanie o wszystkie ogniwa, które zostały dodane po ostatnim ogniwie znalezionym w lokalnej wersji bazy danych. Przy pierwszym uruchomieniu aplikacji zapytanie zawiera wyzerowany identyfikator (null). Po otrzymaniu takiego zapytania zdalny węzeł odpowiada wszystkimi ogniwami, które posiada w głównym oraz tymczasowym łańcuchu.

Kiedy wszystkie węzły odpowiedziały, każda z odpowiedzi zostaje testowo dodana do łańcucha tymczasowego. Następnie jest przeprowadzana weryfikacja sygnatur wszystkich ogniw (Schemat blokowy 1). Po weryfikacji tymczasowy łańcuch zostaje wyczyszczony i operacja zostaje powtórzona dla następnej otrzymanej odpowiedzi. Kolekcja poprawnie zweryfikowanych ogniw, ilość udanych i nieudanych weryfikacji zostają zapamiętane. Jeżeli kolejne kolekcje zawierały te same ogniwa co ta zapamiętana oraz więcej, aktualna kolekcja zostaje zapamiętana. Po przetworzeniu wszystkich odpowiedzi sprawdzane jest czy liczba zaakceptowanych odpowiedzi jest większa. Jeżeli tak jest to zapamiętana kolekcja zostaje ponownie dodana do tymczasowego łańcucha. W następnej kolejności następuje wdrożenie ogniw z łańcucha tymczasowego do głównego łańcucha (Schemat blokowy 2). Zapewnia to, że jedynie zatwierdzone ogniwa zostały przeniesione.

Obraz zawierający diagram

Opis wygenerowany automatycznie

Schemat blokowy Weryfikacja łańcucha na podstawie podpisów ogniw (opracowanie własne)

Obraz zawierający diagram

Opis wygenerowany automatycznie

Schemat blokowy Transfer ogniw z tymczasowego do głównego łańcucha (opracowanie własne)

#### Blokowanie

Ze względu na fakt, że aplikacja jest zdecentralizowana, istnieje możliwość, że dwóch użytkowników stworzy nowe ogniwo w tym samym czasie. To tworzy zagrożenie dwóch różnych wersji łańcucha na przestrzeni sieci węzłów. Aby zapobiec takiemu przypadkowi został zaimplementowany mechanizm blokowania ogniw.

W momencie tworzenia nowego ogniwa (Schemat blokowy 3), aplikacja wysyła do wszystkich członków sieci prośbę o lokalne zablokowanie ostatniego ogniwa. Zewnętrzne węzły sprawdzają wtedy czy otrzymane dane są poprawne oraz czy istnieje możliwość zablokowania tego ogniwa. Jeżeli tak, to blokada zostaje nałożona i zostaje zwrócona odpowiedź pozytywna. W przypadku, gdy weryfikacja danych się nie powiodła odpowiedni błąd zostaje zwrócony. Natomiast, jeżeli blokada już istnieje, to negatywna odpowiedź zawiera również propozycję identyfikatora ogniwa, które powinno zostać zablokowane.

Obraz zawierający diagram

Opis wygenerowany automatycznie

Schemat blokowy Tworzenie nowego ogniwa w łańcuchu tymczasowym (opracowanie własne)

Kiedy aplikacja otrzyma już wszystkie odpowiedzi, zliczane są pozytywne, jeśli jest ich więcej niż negatywnych to blokada zostaje potwierdzona – lokalnie ogniwo zostaje włączone do głównego łańcucha, a do wszystkich węzłów zostaje wysłana prośba o zatwierdzenie blokady. Wtedy wszystkie węzły zewnętrzne powinny również wdrożyć ogniwo do głównego łańcucha.

W przypadku kiedy ilość odpowiedzi negatywnych przeważa, to aplikacja wysyła do wszystkich węzłów, które zwróciły pozytywną odpowiedź prośbę o wycofanie blokady. Następnie, lokalnie dochodzi do aktualizacji ogniwa o identyfikator zaproponowany przez sieć oraz obliczenie nowego podpisu na podstawie zaktualizowanego ogniwa. Po tych operacjach aplikacja przystępuje do ponownej próby blokady.

Blokady nie są wieczne dopóki nie zostały potwierdzone. Aplikacja ma jedną minutę na potwierdzenie. Blokady mogą być potwierdzone jedynie przez aplikację, która o nią poprosiła. Wynika to z faktu, że w momencie prośby o wycofanie blokady lub jej potwierdzenie zostaje załączona informacja podpisana kluczem prywatnym osoby blokującej. Jeżeli klucz publiczny zawarty w blokadzie umożliwi potwierdzenie poprawności podpisu to proces jest kontynuowany.

### Model

Model aplikacji (Diagram UML 3) został dobrany tak, aby w prosty sposób reprezentować dwa elementy wykorzystywane w aplikacjach typu tablicy ogłoszeń lub bloga.

Komponent zawiera dwie klasy:

* User
* Post

Obraz zawierający tekst

Opis wygenerowany automatycznie

Diagram UML Klasy modelu wykorzystywanego przez aplikację (opracowanie własne)

## Aplikacja

Część projektu poświęcona interfejsowi użytkownika została nazwana Application. Jest złożona z aplikacji webowej napisanej w technologii React, która jest obsługiwany przez aplikację serwerową wykonaną za pomocą platformy ASP .NET.

Komunikacja pomiędzy oboma elementami jest realizowana za pomocą SPA Proxy (22), które umożliwia działanie obu aplikacji na tym samym porcie, a zapytania z części przeglądarkowej do serwerowej mogą zostać wykonane poprzez zdefiniowane punkty dostępowe. W tym przypadku jest to ścieżka /api.

Przy pierwszym uruchomieniu aplikacji generowane są klucze publiczny oraz prywatny dla algorytmu RSA. Posłużą one jako dane logowania dla użytkownika, który zostanie stworzony w następnych krokach.

Sama aplikacja ma na celu imitować działanie globalnie znanej aplikacji – Twitter. Pozwala zatem na stworzenie użytkownika o wybranej nazwie oraz wstawianie wpisów na tzw. ścianie. Wpisy, inaczej wiadomości są wyświetlane w kolejności od najnowszych do najstarszych.

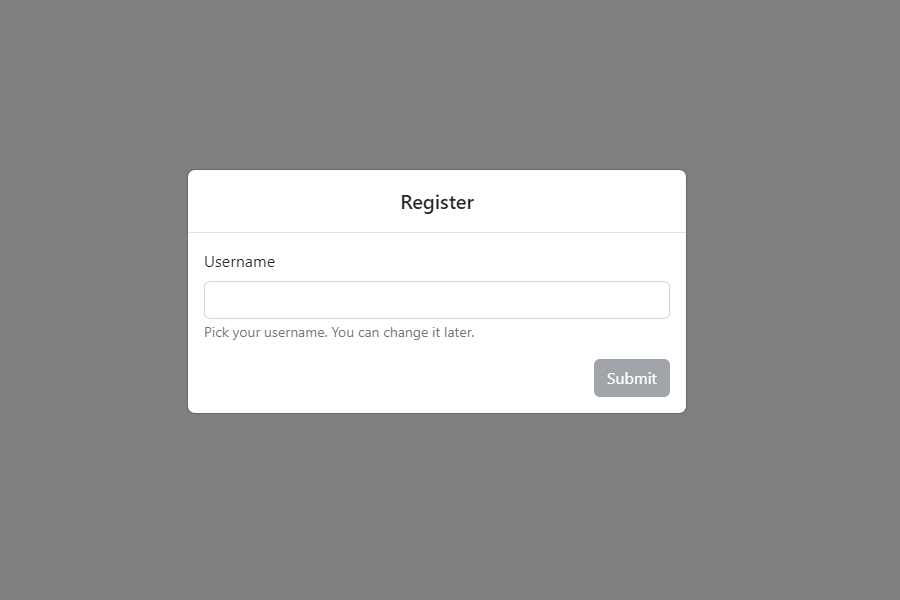
### Synchronizacja

Pierwszy ekran wyświetlonym użytkownikowi zawiera informację o oczekiwaniu na synchronizację aplikacji z pozostałymi węzłami w sieci.

### Tworzenie użytkownika

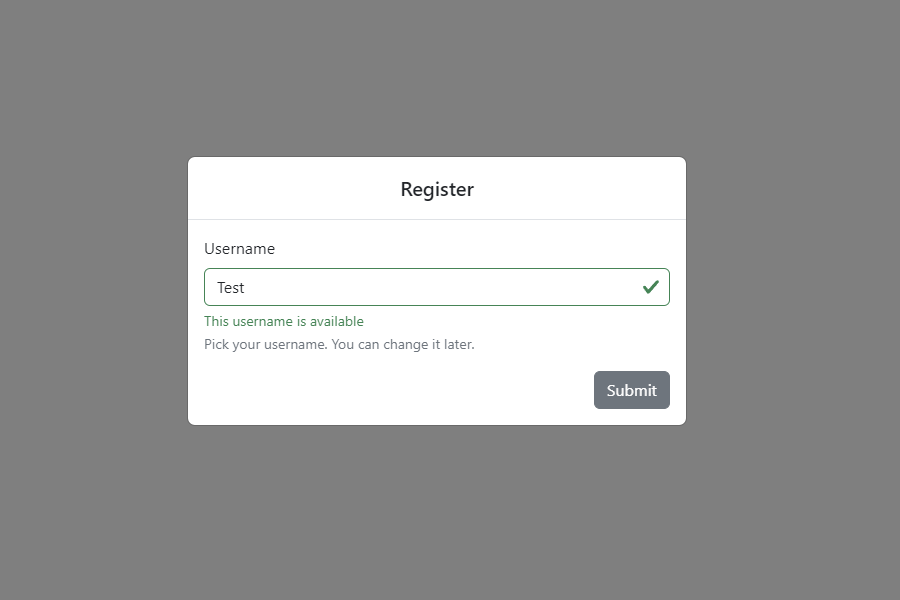
Mając utworzone klucze dla algorytmu RSA oraz zsynchronizowaną bazę danych, aplikacja jest gotowa, aby utworzyć ogniwo reprezentujące użytkownika.

W przeglądarce wyświetli się okno proszące o uzupełnienie nazwy użytkownika (Zrzut ekranu 2).

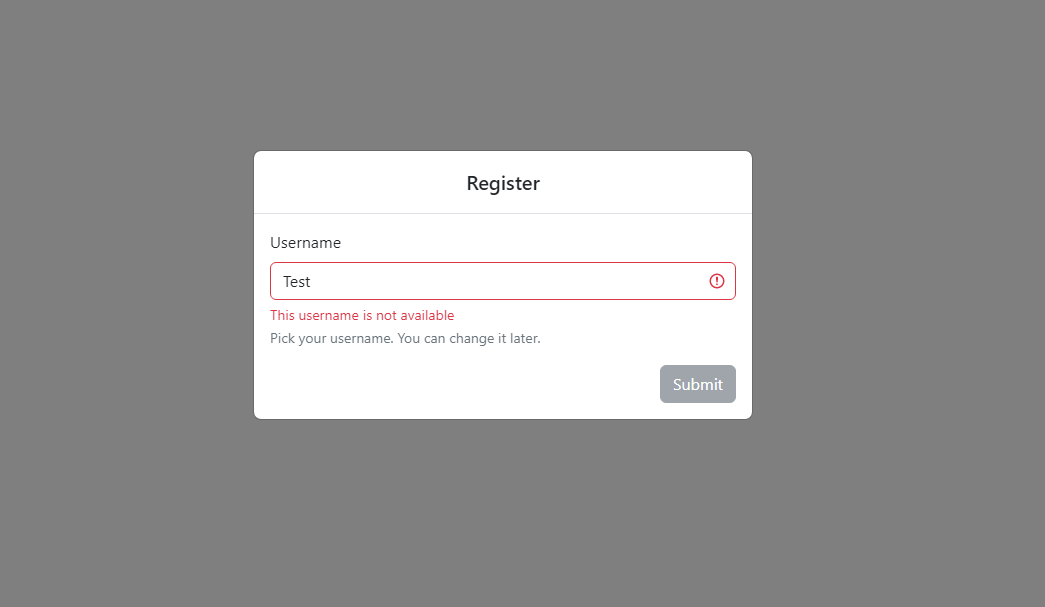


Zrzut ekranu Formularz rejestracyjny (opracowanie własne)

Po wprowadzeniu nazwy użytkownika, zostaje przeprowadzona weryfikacja czy dana nazwa jest dostępna. Jeżeli tak jest, to zielona wiadomość potwierdzająca ten fakt zostanie wyświetlona pod polem tekstowym (Zrzut ekranu 3). W przeciwnym wypadku odpowiednia informacja w kolorze czerwonym zostaje wyświetlona w tym samym miejscu (Zrzut ekranu 4).

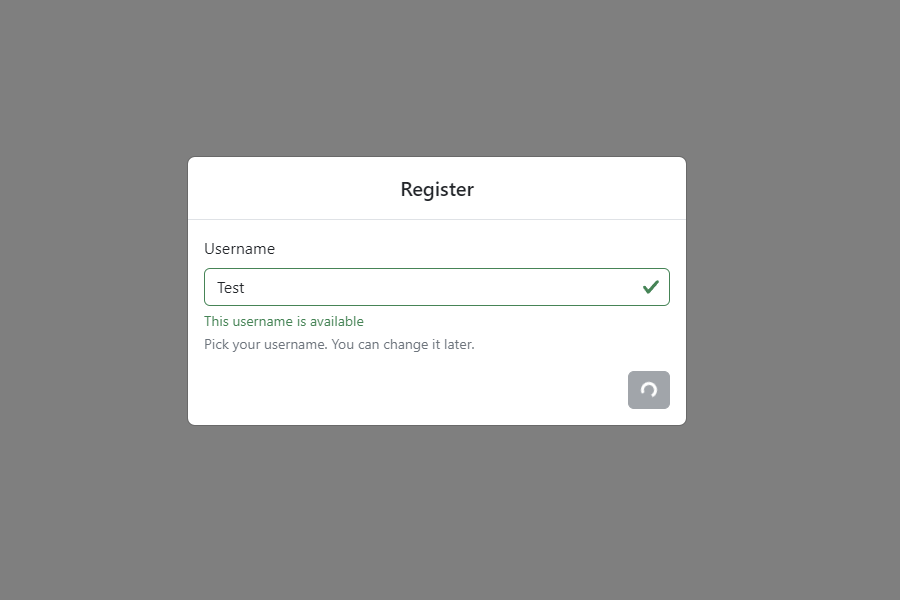


Zrzut ekranu Wynik pozytywnej weryfikacji dostępności nazwy użytkownika (opracowanie własne)



Zrzut ekranu Wynik negatywnej weryfikacji dostępności nazwy użytkownika (opracowanie własne)

Po spełnieniu wszystkich wymagań przycisk zatwierdzenia staje się aktywny i umożliwia przesłanie wybranej nazwy do aplikacji serwerowej. Po wciśnięciu przycisku staje się on ponownie nieaktywny, a indykator przetwarzania zostaje wyświetlony (Zrzut ekranu 5).

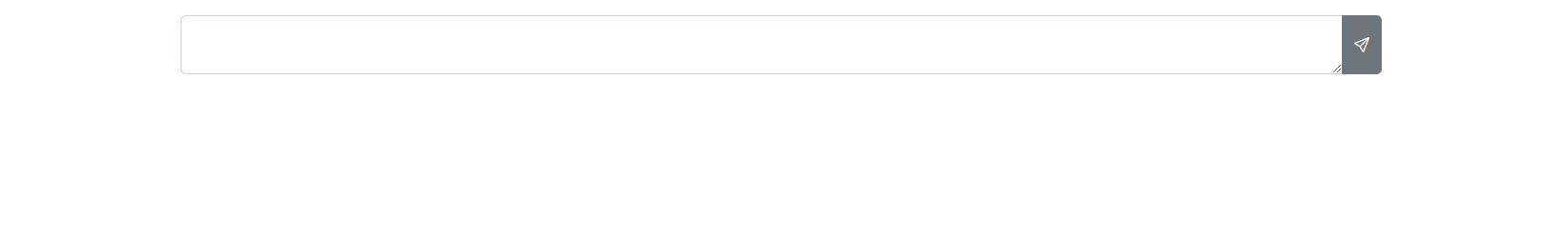


Zrzut ekranu Panel rejestracji - przetwarzanie żądania (opracowanie własne)

W momencie, gdy aplikacja serwerowa zakończy przetwarzanie, a ogniwo użytkownika zostanie dodane do głównego łańcucha, to panel rejestracji zniknie.

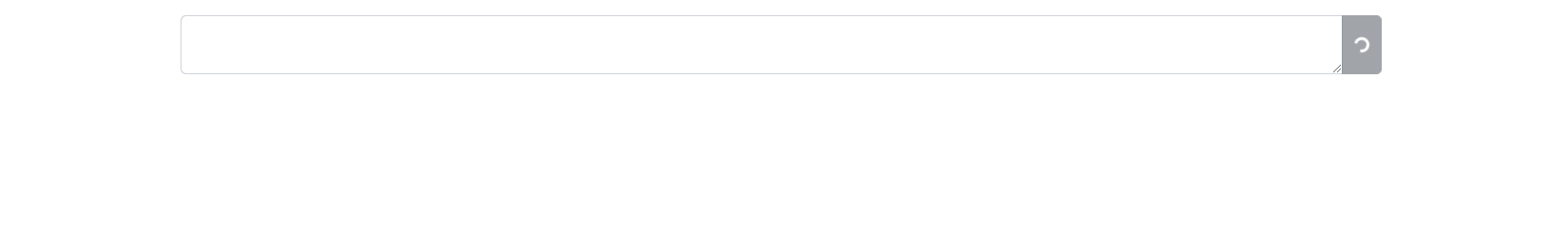
### Tworzenie postów

Pojawi się wtedy główna strona aplikacji, która umożliwia dodawanie postów (Zrzut ekranu 6).



Zrzut ekranu Formularz tworzenia nowej wiadomości (opracowanie własne)

Podobnie jak przy tworzeniu użytkownika, po wprowadzeniu wiadomości i wciśnięciu przycisku z ikoną reprezentującą papierowy samolot, pokaże się indykator przetwarzania (Zrzut ekranu 7).



Zrzut ekranu Tworzenie wiadomości (opracowanie własne)

Kiedy wszystkie procesy związane z dodawaniem nowego ogniwa do głównego łańcucha się zakończą, pod polem tekstowym pokaże się nowo dodana wiadomość, wraz z nazwą użytkownika, który ją dodał (Zrzut ekranu 8).

Obraz zawierający stół

Opis wygenerowany automatycznie

Zrzut ekranu Stworzona wiadomość (opracowanie własne)

### Format przechowywanych danych

Po przeprowadzeniu wcześniej opisanych czynności, baza danych zawiera dwa rekordy, które wyglądają następująco.

{

"\_id": {"$guid": "09e111ff-78d1-4d60-b49b-2cf6dbcf2917"},

"Object":

{

"\_type": "Model.User, Model",

"Name": "Test"

},

"ObjectType": "Model.User",

"Signature":

{

"Hash": "msgH+A/IHLdEA35ffhbmVTk1r1BSzIs5iGAI1P1fGgvOV8tAoQ2vjl6Kd8gyy5Ve7WysezAkhnG…

"Owner": "MIIBCgKCAQEAs0lVlp3dKby65TcgWINj+rfRUx3WGAglTa2bjrIdCSGj1tucvIJRfWR9GVocuv…

},

"Timestamp": "2023-04-18T22:09:22.3687220Z",

"Lock":

{

"NextId": {"$guid": "037f8938-ba83-4d06-b312-b1ff5d4933f8"},

"Owner": "fN/rrSg5o1uTvxCpKdURdXLctnwisMkW0E1FdfQdWyBuEAEVT5G63TYLvoB+aFetb/j4ZYJaSU…

"Expires": "2023-04-18T22:16:10.8445449Z",

"Confirmed": true

}  
},  
{

"\_id": {"$guid": "037f8938-ba83-4d06-b312-b1ff5d4933f8"},

"Object":

{

"\_type": "Model.Post, Model",

"Message": "Wiadomość testowa!"

},

"ObjectType": "Model.Post",

"Signature":

{

"Hash": "TAaRHju2CoTFp4lvvtgHrxcFCNvmhdh2weUGYOCJwYq5BcMj7Uf+kNkz7xc6Par2R9XCP8rglfe…

"Owner": "MIIBCgKCAQEAs0lVlp3dKby65TcgWINj+rfRUx3WGAglTa2bjrIdCSGj1tucvIJRfWR9GVocuv…

},

"Timestamp": "2023-04-18T22:15:05.2599500Z",

"LastId": {"$guid": "09e111ff-78d1-4d60-b49b-2cf6dbcf2917"}

}

Można zauważyć, że pierwszym wpisem był stworzony użytkownik (ObjectType: Model.User), ponieważ to ogniwo nie posiada atrybutu LastId, który reprezentuje identyfikator poprzedniego rekordu. Widać to również w obiekcie zawartym pod atrybutem Lock. Posiada on identyfikator następnego ogniwa pod atrybutem NextId. Identyfikator ten wskazuje na ogwnio zawierające dodaną wiadomość.

Skupiając się na podpisach (obiekt pod atrybutem Signature) obu ogniw można stwierdzić z łatwością, że zostały stworzone przez tę samą osobę, ponieważ atrybut Owner jest taki sam. Wartość tego atrybutu reprezentuje klucz publiczny danego użytkownika.  
Natomiast atrybut Hash, który jest kluczowym aspektem blockchain’u, a zatem całego projektu jest różny pomiędzy ogniwami. Wynika to z faktu, że zawiera on podpisany elektronicznie algorytmem RSA skrót wykonany za pomocą algorytmu SHA256 zawartości danego ogniwa oraz poprzedniego, jeżeli takowe istnieje.  
Na przykład hash obiektu użytkownika zawiera dane jedynie tego ogniwa, ponieważ jest to pierwsze ogniwo i nie ma ono referencji do poprzedniego ogniwa.

# Bezpieczeństwo danych

## Dane początkowe

Przygotowane dane testowe zawierały lokalnie utworzonego użytkownika oraz trzy posty dodane jeden po drugim. Każdy z nich zawierał inny tekst. Poniższy Zrzut ekranu 9 przedstawia opisane dane w postaci implementacji.

Obraz zawierający tekst, kalendarz

Opis wygenerowany automatycznie

Zrzut ekranu Fragment kodu - dane testowe (opracowanie własne)

Po dodaniu wszystkich obiektów do bazy oraz przeniesieniu ich do głównego łańcucha, przeprowadzona została weryfikacja spójności danych. Pozytywny wynik tej weryfikacji był warunkiem koniecznym przystąpienia do faktycznego testu.

## Test – Usunięcie obiektu

W przypadku testu na usunięcie obiektu z łańcucha, przygotowane zostały dwa scenariusze:

* Usunięcie pojedynczego ogniwa w środku łańcucha
* Usunięcie pojedynczego ogniwa na końcu łańcucha

Pierwszy przypadek zakłada, że po usunięciu ogniwa z wewnątrz łańcucha, a następnie po ponownym uruchomieniu weryfikacji zaczynając od końca łańcucha otrzymana zostanie negatywna odpowiedź.

Drugi scenariusz przewiduje, że po usunięciu ostatniego ogniwa weryfikacja zwróci wynik pozytywny ze względu na fakt, że po ostatnim ogniwie nie zostanie ślad. Wzmianka o ostatnim ogniwie znajduje się w atrybucie Lock poprzedzającego ogniwa, jednak ze względu na fakt, że jest to dana pomocnicza i nie jest wliczana w podpis ogniwa, dlatego też będzie pomijana.

Oba scenariusze testów zwracają wynik pozytywny (Zrzut ekranu 10).

Obraz zawierający tekst

Opis wygenerowany automatycznie

Zrzut ekranu Wyniki testów na usunięcie ogniwa (opracowanie własne)

## Test – Modyfikacja

Test na wykrycie przeprowadzenia modyfikacji ogniwa definiuje dwa scenariusze:

* Modyfikacja ogniwa – dane podstawowe
* Modyfikacja ogniwa – dane pomocnicze
* Modyfikacja przechowywanego obiektu (User / Post)

W przypadku modyfikacji danej podstawowej ogniwa wybrany został atrybut Timestamp. Aktualna data i godzina UTC zostaje przypisana temu atrybutowi dla ogniwa zawierającego post numer 1. Ogniwo zostaje następnie zaktualizowane w bazie danych i przeprowadzona jest weryfikacja, której wynik powinien być negatywny.

Przypadek modyfikacji danej pomocniczej zakłada, że zostanie zmodyfikowany atrybut Lock, a zatem został on wyczyszczony dla ogniwa zawierającego post numer 2. Następnie uruchomiona zostaje weryfikacja, której wynik również powinien być negatywny, ponieważ podpis następującego ogniwa zawiera wszystkie dane zawarte w poprzedzającym (rozważanym) ogniwie.

Test na wykrycie przeprowadzonej modyfikacji obiektu zakłada, że post numer jeden zostanie zmodyfikowany poprzez zamianę wiadomości z „Test Message 1” na „Test Message”. Zmiana wydaje się niewielka, jednak dla zasady działania blockchain jest znacząca. Po zaktualizowaniu ogniwa w bazie danych proces weryfikacji zostanie uruchomiony, a oczekiwany wynik to negatywny.

W tym przypadku również test zwrócił wynik pozytywny (Zrzut ekranu 11).

Obraz zawierający tekst

Opis wygenerowany automatycznie

Zrzut ekranu Wyniki testów na modyfikację ogniw łańcucha (opracowanie własne)

# Bezpieczeństwo aplikacji

## Dane początkowe

Do przeprowadzenia poniższych testów przygotowane zostały certyfikaty dla trzech użytkowników testowych. Dla każdego z nich został stworzony rekord w bazie danych. Wszystkie te rekordy zostały zablokowane a następnie potwierdzone korzystając z odpowiednich certyfikatów.

## Test – Nieautoryzowane odblokowanie

W celu przetestowania, czy dowolny użytkownik byłby w stanie odblokować ogniwo (inaczej mówiąc - usunąć obiekt zawierający klucz publiczny, datę ważności blokady oraz identyfikator następnego ogniwa) do bazy danych zostaje dodany post pierwszego użytkownika. Ogniwo zawierające ten post będzie zawierało obiekt blokady, który zostanie stworzony na dalszym etapie testu. Ogniwo zostaje wdrożone do głównego łańcucha. Kolejnym korkiem było stworzenie wiadomości dla drugiego użytkownika. Zostaje nałożona blokada na poprzedzające ogniwo. Następnie korzystając z certyfikatu użytkownika trzy podjęta zostaje próba odblokowania poprzedzającego ogniwa. Próba ta jest nieowocna, ponieważ w celu przeprowadzenia procesu odblokowania należy przedstawić informację podpisaną certyfikatem twórcy blokady. Przy tworzeniu blokady wykorzystany został klucz publiczny użytkownika dwa, a zatem gdy użytkownik trzy próbował przedstawić informację podpisaną swoim certyfikatem to weryfikacja zakończyła się negatywnie.

W celu weryfikacji czy mechanizm zadziałałby dla użytkownika dwa, próba ta zostaje przeprowadzona ponownie tym razem korzystając z certyfikatu użytkownika dwa. Zadanie kończy się powodzeniem (Zrzut ekranu 12), a blokada zostaje zdjęta z ogniwa zawierającego pierwszy post.

Text

Description automatically generated

Zrzut ekranu Wynik testu na nieupoważnioną próbę odblokowania ogniwa (opracowanie własne)

## Test – Nieautoryzowane potwierdzenie

W momencie gdy stworzona blokada oczekuje na potwierdzenie, istnieje krótki okres, w którym użytkownik sieci mający złe zamiary może spróbować zakłócić działanie sieci poprzez przedwczesne zatwierdzenie blokady i wdrożenie ogniwa do łańcucha. W celu przetestowania czy jest to możliwe, podobnie jak w poprzednim przypadku dodany zostaje post użytkownika dwa, który zostaje wdrożony do głównego łańcucha. Następnie stworzony zostaje drugi post, dla którego zakładana jest blokada na ogniwie postu pierwszego. W tym momencie zostaje wywołany mechanizm potwierdzenia korzystając z certyfikatu użytkownika trzy. Przekazana zostaje informacja podpisana certyfikatem wspomnianego użytkownika, blokada zawiera klucz publiczny użytkownika dwa, a zatem weryfikacja podpisu zakończy się niepowodzeniem.

Test zwraca pozytywny wynik (Zrzut ekranu 13).

Text

Description automatically generated

Zrzut ekranu Wynik przeprowadzonego testu na nieupoważnioną próbę potwierdzenia ogniwa

W czasie tworzenia powyższego testu zauważone zostało potencjalne zagrożenie, które może wystąpić w zaimplementowanej aplikacji. Użytkownicy są w stanie spreparować prośby o potwierdzenie blokady oraz wdrożenie ogniwa do głównego łańcucha, dla węzłów które zwróciły pozytywną odpowiedź w momencie blokowania (3.2.2.2 Blokowanie), nawet jeśli nie otrzymali większej niż 50% pozytywnych odpowiedzi od wszystkich węzłów. Sama aplikacja nie pozwala na taki scenariusz, jednak użytkownik mógłby wysłać zapytania bezpośrednio do innych węzłów poza aplikacją.

Powyższy problem pokazuje jak trudne jest zaprojektowanie zdecentralizowanej aplikacji korzystającej z technologii blockchain. Architektura klient-serwer jest dużo prostsza, ponieważ logika biznesowa znajduje się w pełni po stronie serwera. W przypadku zdecentralizowanej aplikacji każdy użytkownik jest sobie równy oraz każdy z nich jest w stanie poznać strukturę wysyłanych oraz odbieranych danych. To umożliwia użytkownikom mającym złe intencje wysyłanie spreparowanych danych do instancji aplikacji należących do innych użytkowników. Bez odpowiednich mechanizmów bezpieczeństwa aplikacja staje się podatna na takie ataki.

# Podsumowanie

Zaimplementowana aplikacja spełnia założone na początku cele. Część webowa projektu umożliwia użytkownikowi na zdefiniowanie swojej nazwy oraz na tworzenie nowych postów, wyświetlane są również wszystkich wiadomości zawarte w bazie danych wraz z nazwami ich autorów.

Dodatkowo, aplikacja została pomyślnie zdecentralizowana w lokalnej sieci, jest w stanie wykryć dostępne węzły oraz wymieniać się z nimi informacjami.

Ponadto, aplikacja wykorzystuje zaimplementowaną bazę danych typu blockchain. Jak pokazały testy bezpieczeństwa dane przechowywane w tej bazie są spójne, a zatem bezpieczne. Dzięki temu nazwy użytkowników oraz ich wiadomości nie mogą zostać zmodyfikowane po tym jak zostały zatwierdzone do głównego łańcucha.

Została zidentyfikowana luka bezpieczeństwa, która przypomina o konieczności dokładnego testowania tworzonych aplikacji, zwłaszcza w przypadku decentralizacji.

# Spis obrazków

## Diagramy UML

[Zależności między klasami modelu komponentu Blockchain (opracowanie własne) 11](#_Toc135515492)

[Zależności pomiędzy kontekstami komponentu Blockchain (opracowanie własne) 12](#_Toc135515493)

[Klasy modelu wykorzystywanego przez aplikację (opracowanie własne) 17](#_Toc135515494)

## Rysunki

[Zależności pomiędzy komponentami aplikacji (opracowanie własne) 10](#_Toc135515495)

## 

[Weryfikacja łańcucha na podstawie podpisów ogniw (opracowanie własne) 15](#_Toc135515496)

[Transfer ogniw z tymczasowego do głównego łańcucha (opracowanie własne) 15](#_Toc135515497)

[Tworzenie nowego ogniwa w łańcuchu tymczasowym (opracowanie własne) 16](#_Toc135515498)

## Schematy blokowe

[Weryfikacja łańcucha na podstawie podpisów ogniw (opracowanie własne) 15](#_Toc135515499)

[Transfer ogniw z tymczasowego do głównego łańcucha (opracowanie własne) 15](#_Toc135515500)

[Tworzenie nowego ogniwa w łańcuchu tymczasowym (opracowanie własne) 16](#_Toc135515501)

## Zrzuty ekranu

[Przykładowy interfejs użytkownika komponentu Swagger (źródło zewnętrzne) 9](#_Toc135515502)

[Formularz rejestracyjny (opracowanie własne) 19](#_Toc135515503)

[Wynik pozytywnej weryfikacji dostępności nazwy użytkownika (opracowanie własne) 20](#_Toc135515504)

[Wynik negatywnej weryfikacji dostępności nazwy użytkownika (opracowanie własne) 20](#_Toc135515505)

[Panel rejestracji - przetwarzanie żądania (opracowanie własne) 21](#_Toc135515506)

[Formularz tworzenia nowej wiadomości (opracowanie własne) 21](#_Toc135515507)

[Tworzenie wiadomości (opracowanie własne) 22](#_Toc135515508)

[Stworzona wiadomość (opracowanie własne) 22](#_Toc135515509)

[Fragment kodu - dane testowe (opracowanie własne) 24](#_Toc135515510)

[Wyniki testów na usunięcie ogniwa (opracowanie własne) 25](#_Toc135515511)

[Wyniki testów na modyfikację ogniw łańcucha (opracowanie własne) 26](#_Toc135515512)

[Wynik testu na nieupoważnioną próbę odblokowania ogniwa (opracowanie własne) 27](#_Toc135515513)

[Wynik przeprowadzonego testu na nieupoważnioną próbę potwierdzenia ogniwa 27](#_Toc135515514)

# Bibliografia

1. **Wprowadzenie. *Bitcoin.* [Online] https://bitcoin.org/pl/bitcoin-dla-osob-fizycznych.**

**2. Stanisław Drożdż Marcin Wątorek. Bitcoin. [Online] 4 Marca 2021. http://www.pauza.krakow.pl/547\_2\_2021.pdf.**

**3. Warszawska Politechnika. Architektura peer-to-peer. *Ośrodek Kształcenia na Odległość.* [Online] https://esezam.okno.pw.edu.pl/mod/book/view.php?id=8&chapterid=58.**

**4. Nikita Tambe Aashika Jain. Advantages and Disadvantages of Cryptocurrency in 2023. *Forbes.* [Online] https://www.forbes.com/advisor/in/investing/cryptocurrency/advantages-of-cryptocurrency/.**

**5. OneCoin. *Wikipedia.* [Online] https://en.wikipedia.org/wiki/OneCoin.**

**6. Whittaker Matt. How Does Bitcoin Mining Work? *Forbes.* [Online] https://www.forbes.com/advisor/investing/cryptocurrency/bitcoin-mining/.**

**7. Huestis Samuel. Cryptocurrency’s Energy Consumption Problem. *rmi.* [Online] 23 Styczeń 2023. [Zacytowano: 30 Kwietnia 2023.] https://rmi.org/cryptocurrencys-energy-consumption-problem/.**

**8. Yu Song Bo Chen, Xin-Yi Wang. Cryptocurrency technology revolution: are Bitcoin prices and terrorist attacks related? *National Library of Medicine.* [Online] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9860235/.**

**9. Shepherd Jack. 22 Essential Twitter Statistics You Need to Know in 2023. *Social Shepherd.* [Online] https://thesocialshepherd.com/blog/twitter-statistics.**

**10. SHA-1. *Wikipedia.* [Online] https://pl.wikipedia.org/wiki/SHA-1.**

**11. Zero-configuration networking. *Wikipedia.* [Online] https://en.wikipedia.org/wiki/Zero-configuration\_networking#DNS-SD.**

**12. Multicast DNS. *Wikipedia.* [Online] https://en.wikipedia.org/wiki/Multicast\_DNS.**

**13. Microsoft. Co to jest .NET? Wprowadzenie i omówienie. *Microsoft | Learn.* [Online] https://learn.microsoft.com/pl-pl/dotnet/core/introduction.**

**14. LiteDB team. Getting Started. *LiteDB.* [Online] https://www.litedb.org/docs/getting-started/.**

**15. Microsoft. Wprowadzenie do kwerend LINQ (C#). *Microsoft | Learn.* [Online] https://learn.microsoft.com/pl-pl/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries.**

**16. SQLite Consortium. Home. *SQLite.* [Online] https://sqlite.org/index.html.**

**17. Hangfire. Documentation. [Online] https://docs.hangfire.io/en/latest/.**

**18. Serilog. Home. [Online] https://serilog.net/.**

**19. Microsoft. What is ASP.NET? *Microsoft.* [Online] https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet.**

**20. SmartBear. About. *Swagger.* [Online] https://swagger.io/about/.**

**21. Facebook. Home. *React.* [Online] https://react.dev/.**

**22. Microsoft. Overview of Single Page Applications (SPA) in ASP.NET Core. *Microsoft | Learn.* [Online] https://learn.microsoft.com/en-us/aspnet/core/client-side/spa/intro?view=aspnetcore-7.0.**

**23. Computer Science Education Research Group at the University of Canterbury, New Zealand. CSFG. [Online] 29 Kwiecień 2016. https://www.csfieldguide.org.nz/en/.**