Spis treści

[1 Wstęp 4](#_Toc133186020)

[1.1 Cel 4](#_Toc133186021)

[1.2 Zakres pracy 4](#_Toc133186022)

[2 Zagadnienia teoretyczne 5](#_Toc133186023)

[2.1 Szyfrowanie 5](#_Toc133186024)

[2.1.1 Szyfrowanie jednokierunkowe 5](#_Toc133186025)

[2.1.2 Szyfrowanie dwukierunkowe 5](#_Toc133186026)

[2.1.2.1 Podpis cyfrowy 6](#_Toc133186027)

[2.2 Blockchain 6](#_Toc133186028)

[2.3 DNS Service Discovery i Multicast DNS 6](#_Toc133186029)

[3 Projekt oraz implementacja 7](#_Toc133186030)

[3.1 Technologie 7](#_Toc133186031)

[3.1.1 .NET 7 7](#_Toc133186032)

[3.1.2 LiteDB 7](#_Toc133186033)

[3.1.3 SQLite 7](#_Toc133186034)

[3.1.4 Hangfire 7](#_Toc133186035)

[3.1.5 Serilog 7](#_Toc133186036)

[3.1.6 ASP .NET 8](#_Toc133186037)

[3.1.7 Swagger 8](#_Toc133186038)

[3.1.8 React 8](#_Toc133186039)

[3.2 Komponenty 9](#_Toc133186040)

[3.2.1 Blockchain 9](#_Toc133186041)

[3.2.1.1 Context 12](#_Toc133186042)

[3.2.1.2 PublicContext 12](#_Toc133186043)

[3.2.1.3 TempContext 12](#_Toc133186044)

[3.2.1.4 Tworzenie nowego ogniwa 12](#_Toc133186045)

[3.2.1.5 Weryfikacja łańcucha 13](#_Toc133186046)

[3.2.1.6 Transfer ogniw pomiędzy łańcuchami 14](#_Toc133186047)

[3.2.2 Networking 15](#_Toc133186048)

[3.2.2.1 Synchronizacja 15](#_Toc133186049)

[3.2.2.2 Blokowanie 16](#_Toc133186050)

[3.2.3 Model 17](#_Toc133186051)

[3.3 Application 17](#_Toc133186052)

[3.3.1 Synchronizacja 17](#_Toc133186053)

[3.3.2 Tworzenie użytkownika 18](#_Toc133186054)

[3.3.3 Tworzenie postów 20](#_Toc133186055)

[3.3.4 Format przechowywanych danych 21](#_Toc133186056)

[4 Bezpieczeństwo danych 23](#_Toc133186057)

[4.1 Dane początkowe 23](#_Toc133186058)

[4.2 Test – Usunięcie obiektu 23](#_Toc133186059)

[4.3 Test – Modyfikacja 24](#_Toc133186060)

[5 Bezpieczeństwo aplikacji 24](#_Toc133186061)

[5.1 Środowisko oraz dane początkowe 24](#_Toc133186062)

[5.2 Test – Nieautoryzowane odblokowanie 24](#_Toc133186063)

# Wstęp

Od kilku lat znaczącą popularnością cieszą się dwa terminy: blockchain oraz krypto waluta. Są to pojęcia ze sobą powiązane, krypto waluta jest oparta o technologię blockchain. To oznacza, że blockchain może zostać wykorzystany do innych zastosowań.

## Cel

Przykładem takiego zastosowania może być aplikacja umożliwiająca publikację wiadomości na publicznie dostępnej wirtualnej tablicy. Oczywiście brzmi to jak bardzo prosta aplikacja, którą można napisać przy użyciu standardowej bazy danych oraz serwera. Jednak, jeżeli aplikacja miałaby nie korzystać z typowego serwera, to musi być zdecentralizowana. Co za tym idzie użytkownicy aplikacji muszą być pewni, że dane, z których korzysta aplikacja są prawdziwe i spójne, a to oznacza konieczność użycia technologii blockchain.

Stworzenie takiej aplikacji jest dokładnie celem tej pracy.

## Zakres pracy

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

Następnie powstanie mechanizm pozwalający na zdecentralizowanie aplikacji.

Na koniec, bazując na stworzonych już komponentach, dobudowana zostanie aplikacja webowa, która umożliwi użytkownikowi interakcję z aplikacją

# 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 (haszującego) 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 powinien znać.

Przykładem takiego algorytmu haszującego, inaczej – funkcji skróty jest funkcja SHA. Została stworzona 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.

// Szczegółowe omówienie działania?

### 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. Klucz prywatny natomiast pozwala na zaszyfrowanie informacji i powinien być znany tylko odbiorcy informacji. Poboczną zaletą klucza prywatnego jest to, że można na jego podstawie wygenerować klucz publiczny.

// Szczegółowe omówienie działania?

#### 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.

// Szczegółowe omówienie działania?

## Blockchain

Pojęcie blockchain pierwszy raz zostało zaproponowane razem z propozycją nowego, wirtualnego środka płatniczego nazwanego Bitcoin (BTC). Koncept został przedstawiony w 2008 roku przez osobę lub grupę osób posługującą się pseudonimem Satoshi Nakamoto.

Bitcoin jest siecią typu peer-to-peer, 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.

## DNS Service Discovery i Multicast DNS

Ze względu na 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 instancji korzystając z rekordu DNS PTR.

Multicast DNS umożliwia przetworzenie nazwy domenowej do adresu IP. Serwis nie wymaga żadnej konfiguracji. Wykorzystuje 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.

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 język C# w wersji 11. Jest to wersja, o krótkim okresie wsparcia.

### LiteDB

Aplikacja potrzebowała bazy danych typu NOSQL. Wybrana została implementacja LiteDB, głównie ze względu na fakt, że umożliwia zapisanie danych w postaci pliku. Dodatkowo dane mogą być przetwarzane za pomocą Linq, 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, który również umożliwia zapisanie danych do pliku.

### Hangfire

Hangfire to komponent dla .NET’u, który umożliwia na przetwarzanie zadań równolegle. Pozwala na normalne uruchomienie zadania jak i jego zaplanowanie. Kontrolowane jest również to czy zadanie zostało zrealizowane pozytywnie, w zależności od konfiguracji zadania, jeżeli wystąpi nie powodzenie ponowna próba wykonania zadania może nastąpić określoną ile razy po określonej ilości czasu opóźnienia. Ta funkcjonalność jest kluczowa dla działania aplikacji.

### Serilog

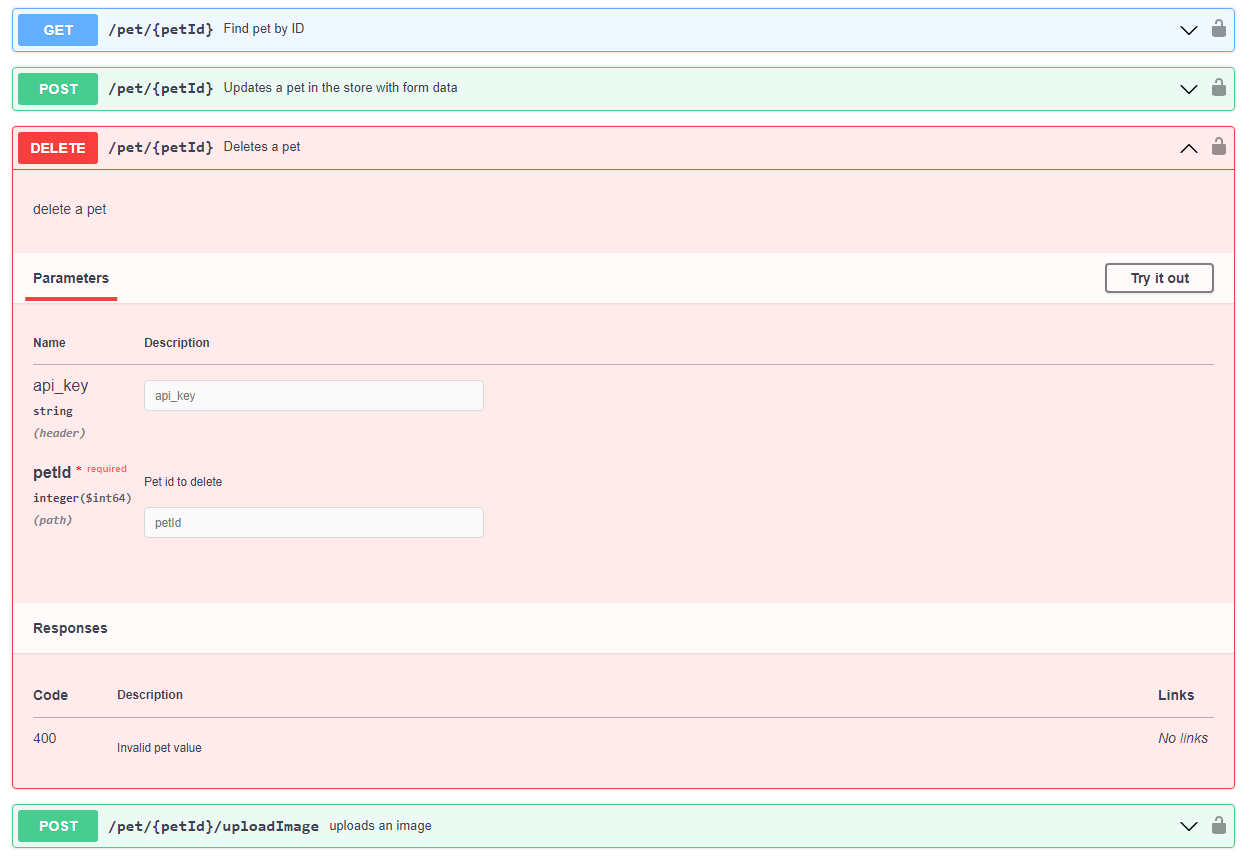
Serilog jako komponent platformy .NET oferuje przejrzyste formatowanie logów zwracanych przez aplikację .NET’ową. Umożliwia na zapisanie stworzonych przez aplikację logów na różne sposoby, w tym zapisanie do pliku lub bazy danych.

### ASP .NET

Ze względów implementacyjnych, aby umożliwić aplikacji na przetwarzanie żądań typu http wykorzystana została również platforma ASP .NET.

### 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.



Zrzut ekranu . Inerfejs użytkownika komponentu Swagger, korzystającego z przykładowej definicji interfejsu API (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 zmiennym stanie. W przypadku aktualizacji takiej zmiennej, widok zostaje ponownie renderowany.

Dodatkowo w celu ulepszenia wyglądu 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.

## Komponenty

W celu zachowania przejrzystości kodu, główne funkcjonalności aplikacji zostały podzielone na trzy komponenty.

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 w postaci pliku, więc może zostać odczytana tylko raz, aby uniknąć konfliktów.  
Komponent odpowiada również za stworzenie lub odczytanie 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 jest bardzo prosty, składa się z trzech klas:

* Link – reprezentuje ogniwo łańcucha informacji
* 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 ogniwa jak i poprzedzające, dodatkowo identyfikuje twórcę.

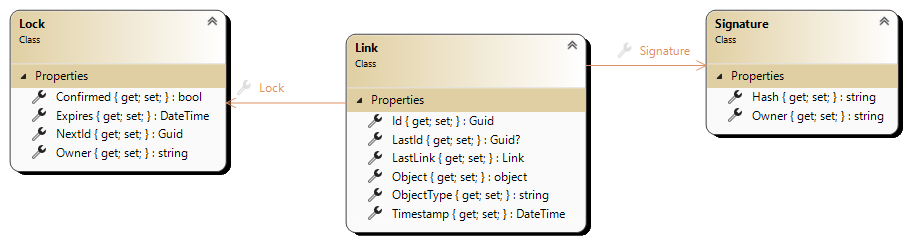


Diagram UML Zależności między klasami modelu komponentu Blockchain (opracowanie własne)

Mechanizmy zostały zaimplementowane w postaci kontekstów:

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

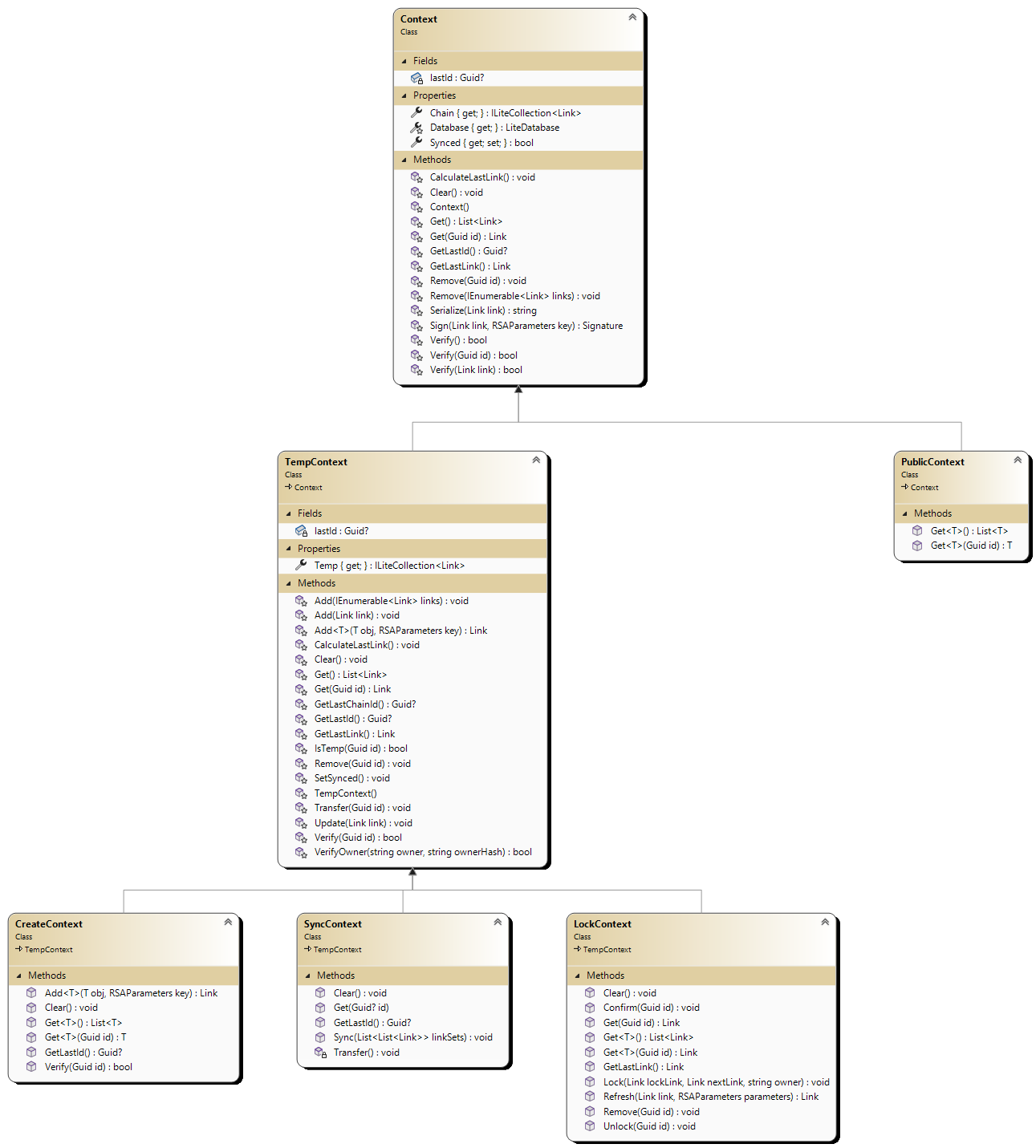


Diagram UML Zależności pomiędzy kontekstami komponentu Blockchain (opracowanie własne)

#### 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 łańcucha do głównego łańcucha. Nadpisuje odpowiednio metody, biorąc pod uwagę dodatkową kolekcję.

#### Tworzenie nowego ogniwa

Obraz zawierający diagram

Opis wygenerowany automatycznie

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

#### Weryfikacja łańcucha

Obraz zawierający diagram

Opis wygenerowany automatycznie

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

#### Transfer ogniw pomiędzy łańcuchami

Obraz zawierający diagram

Opis wygenerowany automatycznie

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

### 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 zsynchronizować się 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.

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 węźle znalezionym lokalnie. Przy pierwszym uruchomieniu aplikacji zapytanie zawiera wyzerowany identyfikator (null).  
Po otrzymaniu zapytania zdalny węzeł odpowiada wszystkimi ogniwami tymi z głównego łańcucha jak i tymczasowego.

Kiedy wszystkie węzły odpowiedziały, każda z odpowiedzi zostaje testowo dodana do łańcucha tymczasowego. Następnie jest przeprowadzana weryfikacja sygnatur wszystkich ogniw. 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 poprzednią zapamiętaną oraz więcej, aktualna kolekcja zostaje zapamiętana. Po przetworzeniu wszystkich odpowiedzi sprawdzane jest czy ilość akceptowalnych 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. Zapewnia to, że jedynie zatwierdzone ogniwa zostały przeniesione.

#### 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 zostało zaimplementowanie blokowanie ogniwa.

W momencie tworzenia nowego ogniwa, 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.

Kiedy aplikacja otrzyma już wszystkie odpowiedzi, zliczane są wszystkie pozytywne, jeśli jest ich więcej 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 obiekt blokady zawiera podpisany algorytmem RSA klucz, który pozwala na weryfikację w momencie potwierdzenia lub wycofania blokady.

### Model

Model aplikacji 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)

## Application

Część projektu poświęcona interfejsowi użytkownika została nazwana Application. Jest złożona z aplikacji webowej napisanej w języku React, który jest obsługiwany przez aplikację serwerową wykonaną w ASP .NET.

Komunikacja pomiędzy oboma elementami jest realizowana za pomocą SPA Proxy, 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.

### Synchronizacja

Pierwszy ekran wyświetlonym użytkownikowi zawiera informację o oczekiwaniu na synchronizację aplikacji.

Obraz zawierający tekst

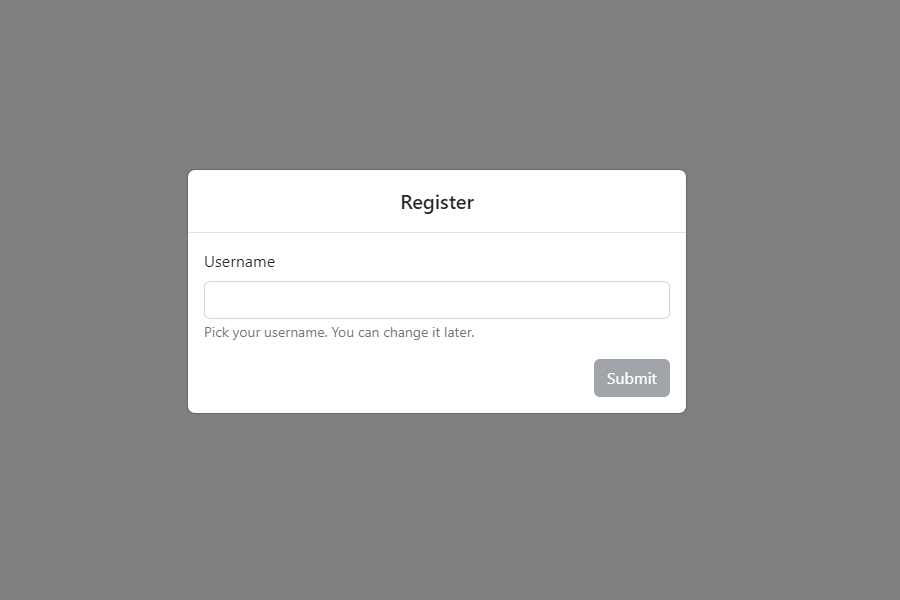
Opis wygenerowany automatycznie

Zrzut ekranu Ekran oczekiwania na synchronizację (opracowanie własne)

### Tworzenie użytkownika

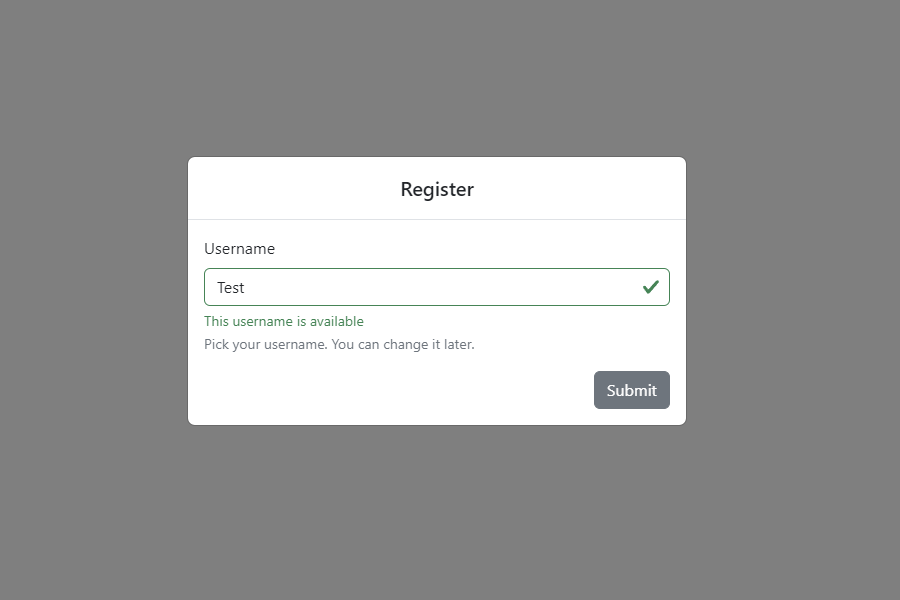
Mając utworzone klucze dla algorytmu RSA oraz zsynchronizowaną bazę danych, aplikacja jest gotowa, aby utworzyć ogniwo reprezentujące użytkownika.

W przeglądarce wyświetli się okno proszące o uzupełnienie nazwy użytkownika.

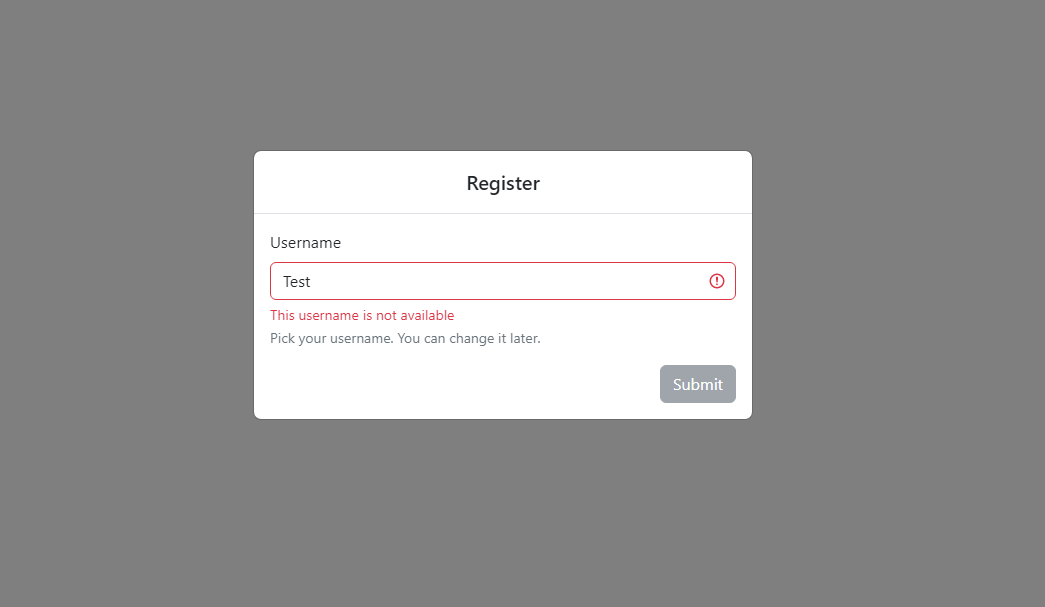


Zrzut ekranu Formularz rejestracyjny (opracowanie własne)

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. W przeciwnym wypadku odpowiednia informacja w kolorze czerwonym zostaje wyświetlona w tym samym miejscu.

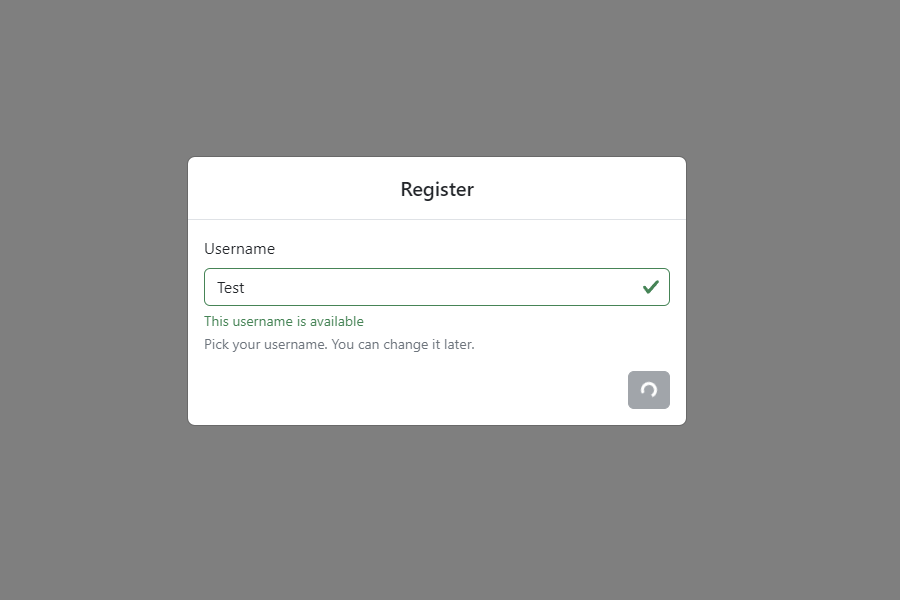


Zrzut ekranu Formularz rejestracyjny - weryfikacja pozytywna (opracowanie własne)



Zrzut ekranu Formularz rejestracyjny - weryfikacja negatywna (opracowanie własne)

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

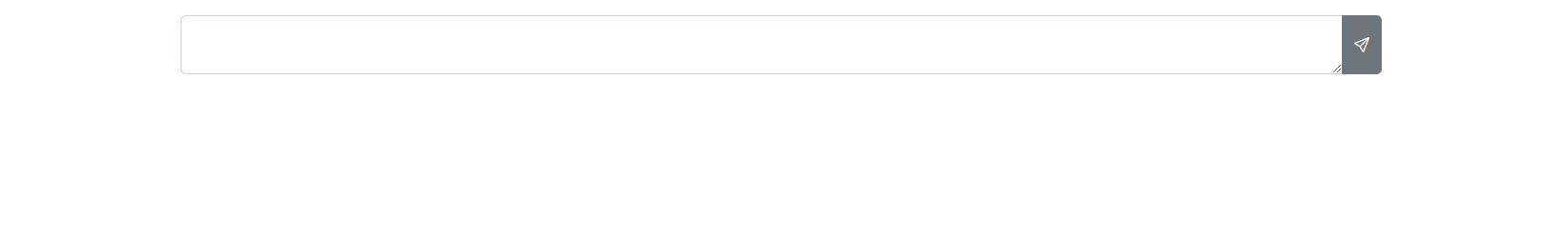


Zrzut ekranu Panel rejestracji - przetwarzanie żądania

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

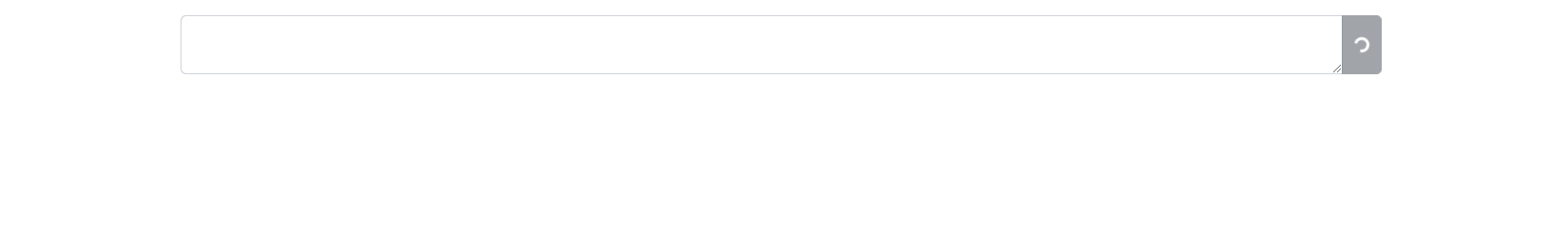
### Tworzenie postów

Otworzy się wtedy główna strona aplikacji, która umożliwia dodawanie postów.



Zrzut ekranu Interfejs użytkownika aplikacji - formularz tworzenia nowej wiadomości (opracowanie własne)

Podobnie jak przy tworzeniu użytkownika, po wprowadzeniu wiadomości i wciśnięciu przycisku z papierowym samolocikiem pokaże się indykator przetwarzania.



Zrzut ekranu Interfejs użytkownika aplikacji - tworzenie wiadomości

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.

Obraz zawierający stół

Opis wygenerowany automatycznie

Zrzut ekranu Interfejs użytkownika aplikacji - stworzona wiadomość (opracowanie własne)

### Format przechowywanych danych

Po przeprowadzeniu wcześniej opisanych czynności, baza danych zawiera dwa rekordy, które występują 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 użytkownik (ObjectType: Model.User), ponieważ nie posiada atrybutu LastId, który reprezentuje identyfikator poprzedniego rekordu. Widać to również po obiekcie Lock który posiada obiekt użytkownika. Atrybut NextId wskazuje na następne ogniwo, którym jest wstawiony post.

Skupiając się na podpisach (obiekt Signature) obu ogniw można stwierdzić z łatwością, że zostały stworzone przez tą samą osobę, ponieważ atrybut Owner jest taki sam. Wartość tego atrybutu reprezentuje klucz publiczny danego użytkownika.  
Natomiast atrybut Hash, który jest kluczowym aspektem 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.

Obraz zawierający tekst, kalendarz

Opis wygenerowany automatycznie

Zrzut ekranu Fragment kodu - dane testowe

Po dodaniu wszystkich obiektów do bazy oraz przeniesieniu ich do głównego łańcucha, przeprowadzona została weryfikacja spójności łańcucha. 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.

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.

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 posta 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, a blokada zostaje zdjęta z ogniwa zawierającego pierwszy post.

Text

Description automatically generated

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

[Diagram UML 1 Zależności między klasami modelu komponentu Blockchain (opracowanie własne) 10](#_Toc133775792)

[Diagram UML 2 Zależności pomiędzy kontekstami komponentu Blockchain (opracowanie własne) 11](#_Toc133775793)

[Diagram UML 3 Klasy modelu wykorzystywanego przez aplikację (opracowanie własne) 17](#_Toc133775794)

[Rysunek 1 Zależności pomiędzy komponentami aplikacji (opracowanie własne) 9](#_Toc133775800)

[Schemat blokowy 1 Tworzenie nowego ogniwa w łańcuchu tymczasowym (opracowanie własne) 13](#_Toc133775874)

[Schemat blokowy 2 Weryfikacja łańcucha na podstawie podpisów ogniw (opracowanie własne) 14](#_Toc133775875)

[Schemat blokowy 3 Transfer ogniw z tymczasowego do głównego łańcucha (opracowanie własne) 14](#_Toc133775876)

**Bibliografia**

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

2. **STANISŁAW DROŻDŻ MARCIN WĄTOREK.** Bitcoin. [Online] 4 Marca 2021. http://www.pauza.krakow.pl/547\_2\_2021.pdf.