Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych

Praca dyplomowa inżynierska

na kierunku Informatyka stosowana

w specjalności Inżynieria Oprogramowania

Wykorzystanie technologii blockchain do podpisywania autentyczności prac dyplomowych

Wojciech Szade

numer albumu 391110

promotor

dr inż. Robert Szmurło

WARSZAWA 2025

Wykorzystanie technologii blockchain do podpisywania autentyczności prac dyplomowych

Streszczenie

To jest streszczenie. To jest trochę za krótkie, jako że powinno zająć całą stronę.

Słowa kluczowe: A, B, C

Using blockchain technology to authenticate thesis works

Abstract

This is abstract. This one is a little too short as it should occupy the whole page.

Keywords: X, Y, Z

Spis treści

1 Wstęp **9**

2 Potrzeby systemu **11**

3 Analiza istniejących rozwiązań **13**

4 Technologie **15**

5 Architektura systemu **17**
5.1 Moduły systemu 17
5.2 Struktura bloku 18
5.3 Komunikacja w systemie 20

6 Podsumowanie **25**

Bibliografia **27**

Wykaz skrótów i symboli **29**

Spis rysunków **31**

Spis tabel **33**

Spis załączników **35**

Rozdział 1

Wstęp

Fałszowanie dokumentów akademickich to poważny problem, który dotyka systemy edukacyjne na całym świecie [5]. Wzrost dostępności zaawansowanych technologii edycji i druku sprawia, że tworzenie fałszywych dyplomów i certyfikatów staje się coraz łatwiejsze. W odpowiedzi na to rosnące zagrożenie, instytucje edukacyjne i pracodawcy poszukują nowoczesnych rozwiązań, które zapewnią wiarygodne potwierdzenie autentyczności dokumentów.

Jedyne rozwiązania oferowane przez uczelnie w zakresie autentykacji prac dyplomowych w momencie publikowania pracy opierają się o udostępnianie informacji o pracach w formie repozytorium, ale systemy te nie skupiają się na możliwości łatwej walidacji prawdziwości pracy i nie oferują narzędzi w tym celu.

W tym kontekście technologia blockchain zyskuje coraz większe zainteresowanie jako potencjalne rozwiązanie. Blockchain oferuje unikalne cechy, które czynią go idealnym narzędziem do weryfikacji autentyczności dokumentów. Decentralizacja blockchaina, eliminująca potrzebę centralnego organu kontrolnego, zwiększa bezpieczeństwo i transparentność systemu, a także gwarantuje jego długowieczność.[4] Niezmienność danych zapisanych w blockchainie gwarantuje ich integralność i autentyczność.

Badania pokazują, że blockchain ma potencjał zrewolucjonizować sposób, w jaki dokumenty akademickie są wydawane i weryfikowane [12]. Wdrożenie systemu opartego na blockchainie może przynieść wiele korzyści: zwiększenie bezpieczeństwa i zaufania do dokumentów, uproszczenie procesu weryfikacji, zmniejszenie kosztów administracyjnych oraz stworzenie globalnie dostępnego systemu weryfikacji.

Takie rozwiązanie musiałoby jednak rozwiązać problemy z którymi boryka się blockchain - wysokie zużycie energii[6], czy ryzyko przejęcia kontroli nad łańcuchem przez niepożądanych użytkowników[11]. Kwestie te są możliwe do rozwiązania, a sam blockchain, jak i jego kluczowa część - mechanizmy konsensusu bardzo szybko się rozwijają[7].

W dalszej części pracy zostaną poruszone i rozwinięte kwestie dokładnych potrzeb i ograniczeń takiego systemu. Czytelnikowi zostanie przybliżone pojęcie blockchaina i części z których się składa, zostanie przeprowadzona analiza zalet i wad zastosowania go dla naszego problemu. Autor

przeprowadzi porównanie obecnych mechanizmów konsensusu i wskaże te, które nadają się do zastosowania w tym projekcie. Opisane zostaną istniejące rozwiązania stosujące blockchain w podobnych zastosowaniach.
Po zrozumieniu i opisaniu tych zagadnień Autor wskaże rozwiązania i technologie, których użyje do stworzenie rozwiązania opisywanego problemu i dokona ich implementacji, skupiając się w pracy na przedstawienie Czytelnikowi architektury i metody działania powstałego systemu.

Rozdział 2

Potrzeby systemu

System służący podpisywaniu autentyczności prac dyplomowych musi spełniać określone potrzeby i wymagania Uczelni, żeby jego wprowadzenie było rozważane.
Kluczową cechą takiego systemu jest bezpieczeństwo - system nie może być podatny na ataki z zewnątrz, a w przypadku przejęcia jednego z uczestników powinno być możliwe jego wyeliminowanie - poprzez zabranie mu uprawnień do tworzenia bloków do czasu potwierdzenia przywrócenia nad nim kontroli. To co istotne - system nie powinien być niepotrzebnie skomplikowany, a korzystanie z niego powinno być stosunkowo proste. Możliwe powinno być zintegrowanie tego rozwiązania z obecnymi systemami stosowanymi na uczelni. Proces dodawania pracy nie powinien zajmować niepotrzebnie dużo czasu - należy przyjąć maksymalny czas dodawania pracy do systemu na 24 godziny. Samo dodawanie pracy do systemu - do momentu w którym pracownik wykonujący zadanie może uznać je za wykonane - i tylko oczekiwać, aż system wykona odpowiednie operacje - powinien być maksymalnie krótki, tak żeby dodawanie wielu prac do systemu nie było przedłużane. System nie powinien korzystać też z nieproporcjonalnej ilości zasobów - żeby nie powodować nadmiarowego zużycia prądu. Zaletą prac dyplomowych jest uporządkowanie powiązanych z nimi danych - każda praca ma autora lub autorów, promotora, datę obrony i inne pola - i nie powinno dojść do sytuacji w której któregoś z nich brakuje lub potrzebne są dodatkowe zaawansowane pola. Dzięki temu można w łatwy sposób walidować dodawane prace. Niemożliwe powinno być dodanie pracy identycznej do takiej, która już została dodana. System powinien możliwe szybko zwracać informację o duplikacie, gdy taki nastąpi.

Od strony użytkowników uzyskujących informację o pracy - system udostępniający im informacje musi działać szybko, dostarczać wystarczającą liczbę informacji do potwierdzenia autentyczności zadanej pracy, bez żadnych wątpliwości i powinien nie pozostawiać wątpliwości - niezawodność systemu jest tutaj ważna - nie można dopuścić do sytuacji w której problem z odnalezieniem pracy sprawia, że użytkownik stwierdza jej nieautentyczność, bo rozwiązanie straciło by wtedy wiarygodność. Rozsądnym pomysłem wydaje się identyfikowanie pracy unikalnymi kluczami, które użytkownicy udostępniali by, w celu łatwego odnalezienia pracy. Jednocześnie system powinien osobom posiadającym taki unikalny klucz - który powinien być bardzo trudny do zgadnięcia,

wymyślenia - umożliwiać uzyskanie dodatkowych informacji o autorze pracy, tak żeby potwierdzić identyfikację tej osoby wykluczając problem osób o tych samych imionach i nazwiskach.

Rozdział 3

Analiza istniejących rozwiązań

Należy zauważyć, że technologia blockchain jest wszechstronna i może być dostosowywana do różnorodnych zastosowań, a systemy oparte na niej, nawet te służące podobnym celom, mogą znacznie różnić się pod względem zalet i wad.
Najpopularniejszym systemem opartym o blockchain jest Bitcoin, który jest kryptowalutą opartą na rozproszonej księdze rachunkowej, umożliwiającą anonimowe płatności bez udziału rządów i banków [13]. Bitcoin, w przeciwieństwie do wielu innych blockchainów, wykorzystuje mechanizm konsensusu *Proof-of-Work* i skupia się na działaniu jako *system płatności*, a nie jako uniwersalna platforma dla aplikacji [8]. Jego idea polega na zapisywaniu w środku bloku transakcji dokonywanych z jego użyciem - dlatego nazywa się go cyfrowym rejestrem, księgą. Jest to już dosyć stara kryptowaluta, a jej mechanizm konsensusu powoduje nadmiarowe zużycie prądu [6], przez co często jest krytykowany - nowe popularne kryptowaluty takie jak *Etherum* rozwiązują ten problem zmieniając mechanizm konsensusu.

Systemem, który implementuje podobną idee, co nasz system korzystając przy tym z blockchainu jest BigchainDB 2.0.
Jest to oprogramowanie, które łączy w sobie cechy blockchainu, takie jak: decentralizacja, niezmienność i aktywa kontrolowane przez właściciela, z cechami bazy danych, takimi jak: wysoka szybkość transakcji, niskie opóźnienia oraz możliwość indeksowania i przeszukiwania uporządkowanych danych. BigchainDB został po raz pierwszy udostępniony jako oprogramowanie open-source w lutym 2016 roku, a wersja 2.0 wprowadziła istotne ulepszenia w stosunku do wcześniejszych wersji [1]. Głównym celem BigchainDB 2.0 było stworzenie systemu, który łączyłby zalety obu technologii, jednocześnie eliminując ich wady. W poprzednich wersjach istniały problemy z odpornością na awarie oraz scentralizowanym zarządzaniem zapisem danych, gdzie jeden węzeł pełnił rolę węzła głównego [1]. Wersja 2.0 wyeliminowała te problemy poprzez:

- Pełną decentralizację: Każdy węzeł ma swoją lokalną bazę danych *MongoDB*, a komunikacja między węzłami odbywa się za pomocą protokołów *Tendermint*. Dzięki temu system jest odporny na awarie, a nawet jeśli 1/3 węzłów ulegnie awarii, pozostałe węzły będą kontynuować działanie. [1]

- Niezmienność danych: Dane przechowywane w sieci BigchainDB nie mogą być zmieniane ani usuwane, a próby ich zmiany są wykrywalne. Każda transakcja jest kryptograficznie podpisana. [1]
- Aktywa kontrolowane przez właściciela: Tylko właściciel aktywa, który posiada odpowiednie klucze prywatne, może nim zarządzać i je transferować. [1]
- Wysoką szybkość transakcji: BigchainDB 2.0 jest zaprojektowany tak, aby obsługiwać dużą liczbę transakcji na sekundę. [1]
- Niskie opóźnienie i szybką finalizację transakcji: Transakcje są zatwierdzane w ciągu kilku sekund, a po zatwierdzeniu nie mogą być cofnięte. [1]
- Indeksowanie i przeszukiwanie strukturalnych danych: wykorzystanie lokalnych baz umożliwia wykorzystanie zalet *MongoDB* do indeksowania i przeszukiwania danych. [1]
- Odporność na ataki Sybil: Sieć BigchainDB jest kontrolowana przez organizację zarządzającą, co minimalizuje ryzyko ataków Sybil. [1]

BigchainDB 2.0 może być używany w wielu różnych zastosowaniach, w tym w zarządzaniu łańcuchem dostaw, ochronie praw własności intelektualnej, cyfrowych bliźniakach i IoT, zarządzaniu tożsamością, zarządzaniu danymi oraz tworzeniu niezmiennych ścieżek audytu [1]. Dzięki połączeniu cech blockchain i bazy danych, BigchainDB 2.0 stanowi wszechstronne narzędzie do budowy zdecentralizowanych i bezpiecznych systemów.

System ten korzysta z mechanizmu konsensusu *Raft*. Algorytm ten polega na wybieraniu lidera wśród uczestników systemu, który przyjmuje polecenia i zapisuje je w swoim logu. Następnie wysyła on te polecenia do pozostałych uczestników, którzy po zatwierdzeniu ich przez większość uczestników zostają zapisane. Jeżeli lider przestanie być dostępnie - uczestnicy wybierają nowego lidera. Lider wybierany jest na podstawie głosowania wszystkich uczestników. [15]

Mimo, że BigchainDB 2.0 jest uniwersalnym systemem z szeregiem zalet nie spełnia on potrzeb naszego systemu, ponieważ brakuje mu kluczowych funkcjonalności, które są istotne dla naszego rozwiązania. System BigchainDB nie pozwala na kontrolę dostępu i eliminację uczestników systemu, ponieważ opiera się on na równorzędnych węzłach z identycznymi uprawnieniami [1]. Nasz system musi w jakiś sposób pozwalać na zarządzanie permisjami uczestników, najlepiej opierając się na wiarygodnych informacjach spoza systemu - wśród uczelni znalezienie takiej metody nie powinno być problematyczne. BigchainDB jest też kompleksowy i złożony - integracja z istniejącymi rozwiązaniami może być skomplikowana i kosztowna. Dostosowanie go do systemów uczelnianych może okazać się zbyt znaczącym problemem. Bigchain DB 2.0 to system, który implementuje wiele dobrych rozwiązań, a jego otwarto źródłowość ułatwia inspirowanie się tym systemem przy tworzeniu własnej implementacji blockchaina. Pomaga też w tym jego dobre udokumentowanie.

Rozdział 4

Technologie

W ramach realizacji projektu inżynierskiego zdecydowano się wykorzystać język Python, który jest odpowiednio dostosowany do potrzeb zadania. Umożliwia też efektywnego rozwiązywanie problemów programistycznych, jednocześnie eliminując potrzebę skupiania się na kwestiach istotnych w niskopoziomowych językach - takich jak dokładne zarządzanie pamięcią. Dla Pythona istnieje też szereg bibliotek wspierających zarówno komunikację P2P, jak i tworzenie REST API, łączenie się z różnymi bazami danych i implementowania blockchaina.

Jako bazę danych do przechowywania bloków wybrałem nierelacyjne *MongoDB*. Baza ta przechowuje informacje w formie dokumentów w formacie *JSON*, pogrupowanych w kolekcje. Bazy nierelacyjne są lepsze w przechowywaniu bloków blockchain od tych opartych na SQL [9]. W dodatku mogą one umożliwić w dużo łatwiejszy sposób dodanie możliwości przechowywania tylko części łańcucha na wybranej maszynie. Przy użyciu bazy SQLowej było by to bardzo trudne. *MongoDB* pozwala na większą swobodę przy tworzeniu struktury bazy danych, jest szybkie i wydajne, a jego elastyczna architektura umożliwia łatwe skalowanie oraz integrację z innymi systemami, co sprawia, że jest idealnym rozwiązaniem dla aplikacji blockchain, gdzie szybki dostęp do danych oraz możliwość przechowywania wybranych fragmentów łańcucha mają kluczowe znaczenie [2]. *MongoDB* jest też używane przez opisywany w rozdziale 3 system BigchainDB 2.0 [1].

Początkowo rozważano oparcie aplikacji o bibliotekę *Django*, która oferuje dużo wbudowanych rozwiązań ułatwiających tworzenie rozwiązania. Biblioteka ta zawiera chociażby panel administratora umożliwiający zarządzanie modelami, ułatwia podział projektu na moduły oraz wspiera zarządzenie obiektami w sposób *ORM*. Jednak wszystkie te rozwiązania w *Django* zostały stworzone z myślą o relacyjnych bazach danych wykorzystujących *SQL* [3]. Niewspierane jest połączenie w wbudowany sposób z *MongoDB* [3]. Wykorzystywanie tej bazy z biblioteką *Django* sprawiałoby, że nie możliwe było by korzystanie z wbudowanych metod zarządzania modelami i bazą, a także z panelu administracyjnego - korzystając z prostego silnika połączenia z bazą, takiego jak *PyMongo*. Istnieje biblioteka *Djongo*, która integruje *Django* z bazą *MongoDB* w „niezauważalny" sposób - mapując działania do konkretnych funkcji. Niestety ta biblioteka nie jest aktywnie wspierana i brakuje jej wielu funkcjonalności. Jej założenia zakładają tłumaczenie kwerend *SQL* na takie zrozumiałe dla Mongo -

co nie jest optymalnym rozwiązaniem i może skutkować w problemach z wydajnością.
Alternatywnymi bibliotekami od *Django* stworzonymi dla *Pythona*, która także umożliwia tworzenie aplikacji udostępniających API są *Flask* i *FastAPI*. Ich porównanie zostało opisane w tabeli 1 .

Kryterium	**FastAPI**	**Flask**
Wydajność	Wysoka - dzięki natywnemu wsparciu dla programowania asynchronicznego; idealna dla aplikacji o dużej liczbie równoczesnych połączeń.	Niższa - przez domyślą synchroniczność; wsparcie dla asynchroniczności wymaga dodatkowych bibliotek, takich jak *Gevent*.
Dokumentacja API	Automatycznie generuje dokumentację API (*Swagger UI*, *ReDoc*).	Wymaga dodatkowych narzędzi, np. *Flask-RESTX*, do generowania dokumentacji.
Walidacja danych	Wbudowana walidacja dzięki *Pydantic* i podpowiedziom typów.	Brak wbudowanej walidacji; wymaga użycia bibliotek zewnętrznych, takich jak *Marshmallow*.
Przypadki użycia	Idealny dla API o wysokiej wydajności, mikrousług i aplikacji czasu rzeczywistego.	Wszechstronny; odpowiedni dla różnych typów aplikacji webowych.

Tabela 1. Porównanie bibliotek FastAPI i Flask [14]

Biblioteki nie są bardzo odmienne od siebie i obie mogłyby być zastosowane, ale autor zdecydował się skorzystać z *FastAPI*, które jest nowszym rozwiązaniem udostępniającym więcej rozwiązań w sposób wbudowany - bez potrzeby korzystania z kolejnych bibliotek. W dodatku biblioteka ta oferuje imponującą wydajność. Do połączeń pomiędzy użytkownikami wykorzystana zostanie biblioteka *p2pd*, służąca do obsługi połączeń Peer2Peer. Jej zaletą jest zaawansowany system wyszukiwania połączeń między węzłami, obsługujący różne rodzaje połączeń i wyszukiwania ich.

Rozdział 5

Architektura systemu

5.1 Moduły systemu

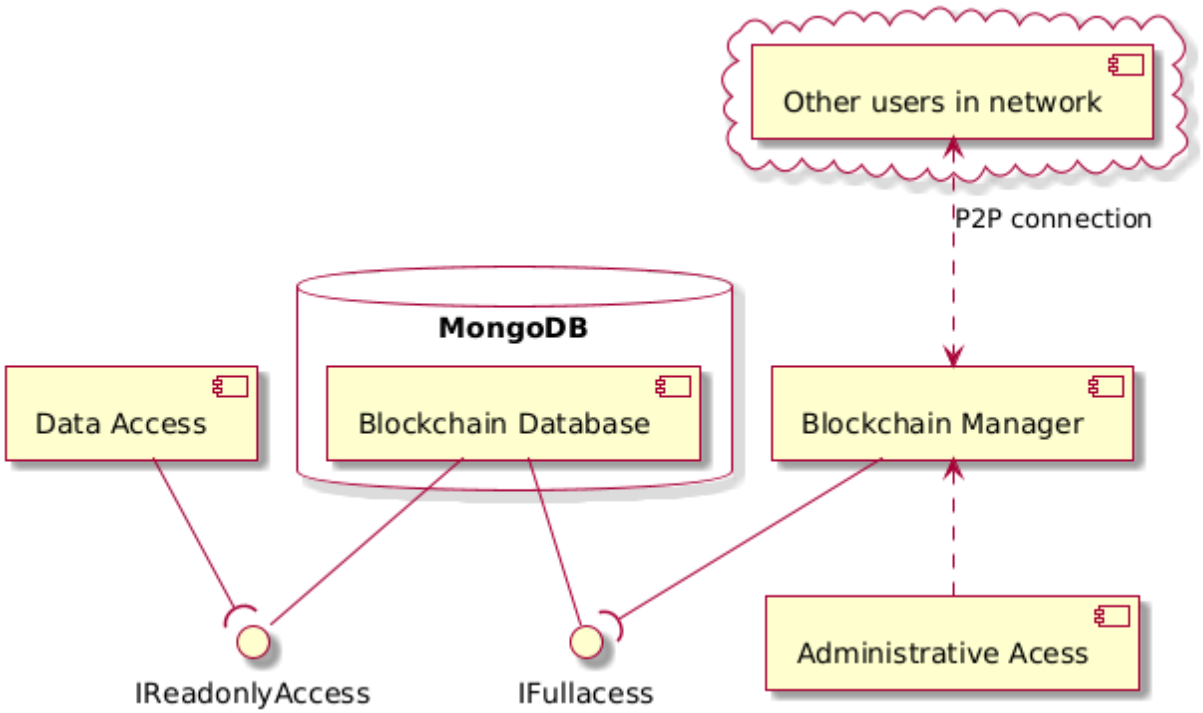
Na rysunku 1 przedstawiony został diagram architektury systemu w uproszczonej formie - skupiający się na modułach systemu. Pokazuje on podział systemu na moduły oraz komunikację pomiędzy nimi.

Komponent *Data Access* przedstawia moduł systemu skupiającą się na użytkownikach, którym będą udostępniane informacje o pracach dyplomowych. Jak widać komponent ten komunikuje się z bazą danych przez połączenie typu „read-only", czyli umożliwiające tylko odczytywanie danych. Zwiększa to bezpieczeństwo systemu i jasno rozdziela cele poszczególnych jego części. Na kolejnych diagramach zostanie przedstawiony dokładniejszy wygląd komponentów składających się na ten moduł systemu.

Komponent *Administrative Access* to moduł służący pracownikom uczelni do dodawania nowych prac dyplomowych do systemu - a administratorom systemu do sterowania działaniem systemu. Moduł ten komunikuje się z komponentem *Blockchain Manager*, który odpowiedzialny jest za całą logikę dodawania nowych bloków, synchronizacji ich i innych zadań związanych z zarządzaniem blockchainem.

Blockchain Manager prowadzi komunikację z innymi uczestnikami sieci poprzez połączenie *Peer to peer (P2P)*, czyli bezpośrednie połączenie pomiędzy maszynami, bez użycia centralnego serwera. To przez to połączenie synchronizowany jest łańcuch, dodawane są nowe bloki. Komponent ten korzysta z połączenia z bazą, przez które ma możliwość modyfikacji zawartości bazy - zapisywania nowych bloków.

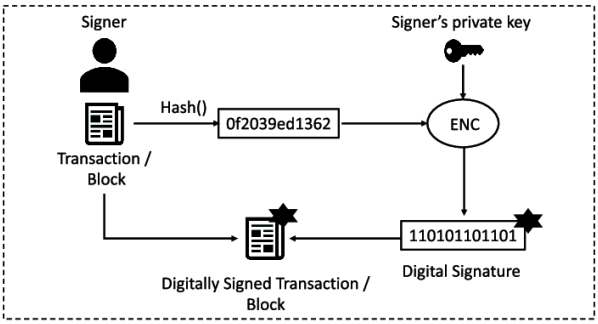
Centralnym elementem systemu oczywiście jest nasza nierelacyjna baza danych *MongoDB*. Udostępnia ona połączenia - umożliwiające tylko odczyt, jak i to które pozwala na modyfikowanie jej zawartości dla dwóch modułów które z niej korzystają. Sama baza przechowuje bloki z informacjami o pracach dyplomowych. W strukturze *MongoDB* takie bloki będą występować jako dokumenty, a należeć będą do kolekcji łańcucha. Bazę można też wykorzystać do przechowywania adresów IP i informacji o uczestnikach sieci.



Rysunek 1. Diagram architektury systemu
Źródło: Opracowanie własne

5.2 Struktura bloku

Informacje o pracach dyplomowych będą przechowywane w naszym systemie w formie bloków. Jedna praca dyplomowa przechowywana jest w jednym bloku. Uporządkowane są one w łańcuchu w kolejności dodawania ich, a wszystkie przynależą do tego samego łańcucha. Każdemu blokowi przypisany jest hasz. Jest on wyliczany na podstawie zawartości bloku, czyli informacji, które on przechowuje, takich jak tytuł pracy, czy autorzy, ale też informacji o poprzednim haszu. To, co kluczowe - to że zmiana którejkolwiek wartości w bloku będzie oznaczała zmianę hasza. Każdy blok posiada też podpis. Jest to hasz podpisany przy użyciu prywatnego klucza uczestnika łańcucha. Proces podpisywania bloku został przedstawiony na rysunku 2. Dzięki temu, że zależny od zawartości hasz zostaje dodatkowo podpisany przez prywatny klucz, zagwarantowana zostaje niezmienność bloku. Modyfikacja go oznaczałaby zmianę hasza, co oznaczałoby potrzebę ponownego podpisu - który nie jest możliwy bez znajomości klucza prywatnego. Pozostali uczestnicy walidują podpis na podstawie przekazanego im hasza i posiadanego klucza publicznego autora bloku [10]. Zawarcie w bloku informacji o haszu bloku poprzedzającego gwarantuje ciągłość łańcucha i zapobiega dodaniu niepożądanego bloku do łańcucha - zmiana informacji o bloku poprzedzającym spowodowałaby to samo, co zmiana każdej innej informacji.



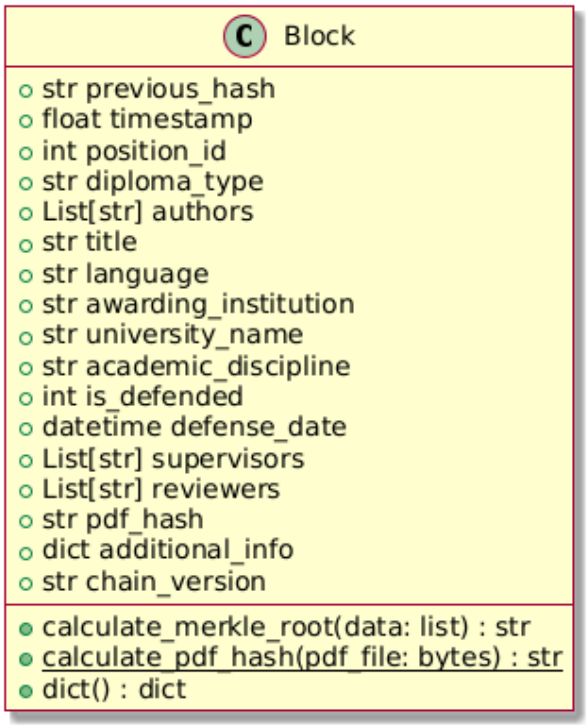
Rysunek 2. Proces podpisywania bloku [10]

Blok musi mieć określoną strukturę określającą zawartość bloku. Zmiana struktury w trakcie funkcjonowania systemu jest możliwa, ale dobrze, jeżeli przewidujemy takie zmiany, wprowadzić pole z informacją o wersji łańcucha. Nasz blok będzie zawierał następujące pola:

- Hasz poprzedniego bloku,
- Znacznik czasu (czas utworzenia bloku w formie timestamp),
- Identyfikator pozycji,
- Rodzaj dyplomu,
- Autora lub autorów pracy,
- Tytuł,
- Język,
- Jednostkę dyplomującą,
- Nazwę uczelni,
- Dyscyplina nauki,
- Status pracy (czy została obroniona),
- Datę obronienia pracy,
- Promotora lub promotorów,
- Recenzent lub recenzenci,
- Hasz PDF (hasz utworzony na podstawie pliku PDF z pracą dyplomową),
- Dodatkowe informacje

Diagram klasy bloku został przedstawiony na rysunku 3. Przedstawione zostały na nim wymienione wyżej pola dostosowane do implementacji oraz metody tej klasy.
calculate_merkle_root(data: list) - metoda, która tworzy drzewo merkle dla podanych danych. Zwraca ona hasz korzenia drzewa. Jest to metoda uzyskiwania hasza dla bloku biorąca pod uwagę każdą przechowywaną daną. Lista danych dzielona jest na pary, które są ze sobą haszowane, poprzez zsumowanie ich i shaszowanie. Powstałe wyniki haszuje się ze sobą, aż do uzyskania tylko jednego wyniku. Dla tych samych danych powstanie zawsze dokładnie taki sam hasz - jednak, jeżeli zmieni się chociaż jedna informacja - hasz ten się zmieni.

Metoda *calculate_pdf_hash(pdf_file: bytes)* służy do obliczania hasza załączonego pliku pdf. Metoda *dict()* zwraca blok w formie *dictionary*.



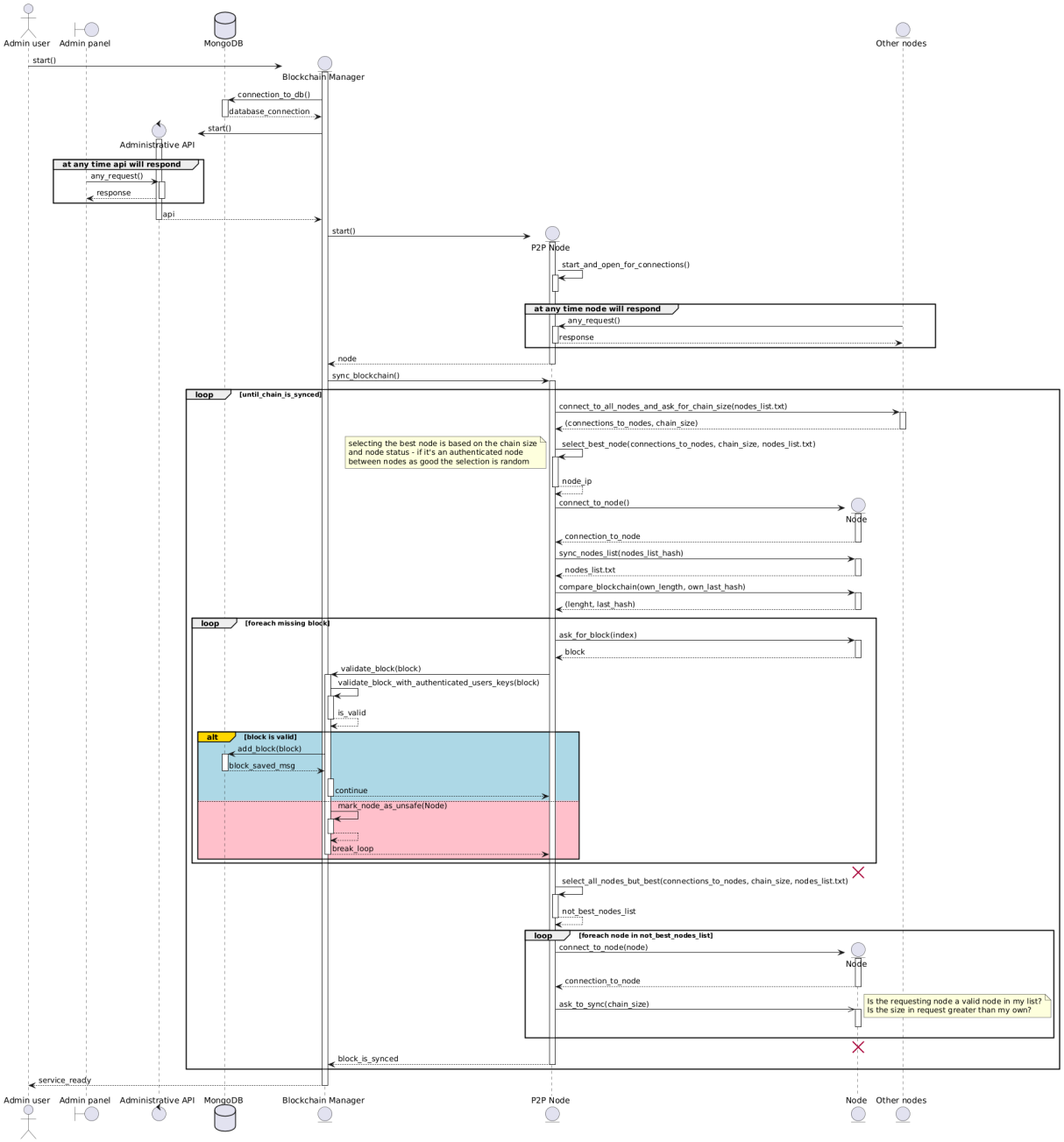
Rysunek 3. Diagram klasy bloku

5.3 Komunikacja w systemie

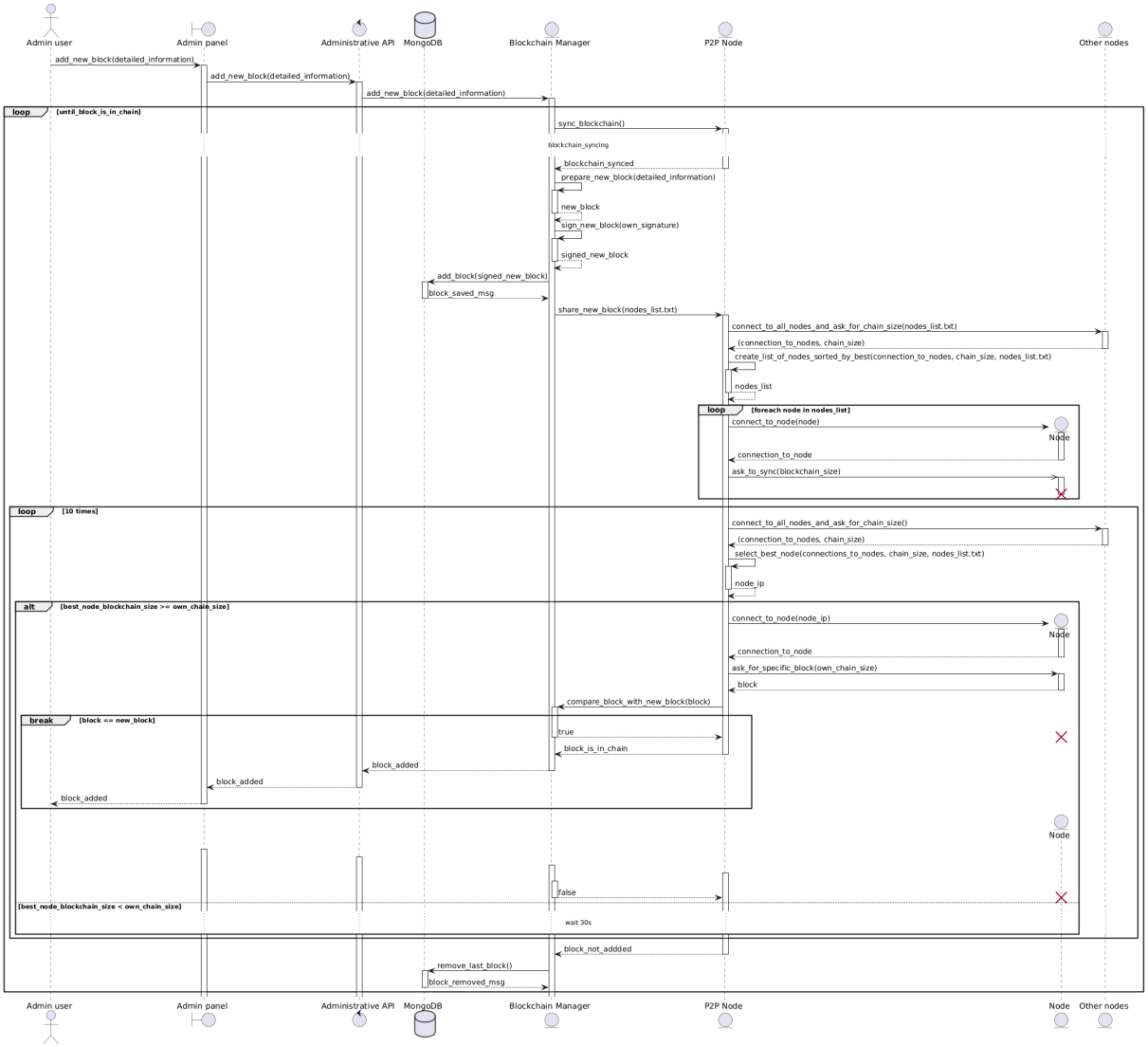
Na rysunku 4 przedstawiony został diagram sekwencji dla uruchomienia systemu. Ilustruje on, co dzieje się i jak przepływa komunikacja środku systemu po uruchomieniu go. Jak widać na diagramie - pierwszą instrukcją wydawaną przez Admina bezpośrednio na samej maszynie - jest prośba o uruchomienie go. System jeszcze nie działa - więc niemożliwe jest komunikowanie się z nim poprzez panel administratora, czy nawet powiązane z nim API. Kolejno, system ustanowi połączenie z bazą danych (jak wspomniane w rozdziale 5 - połączenie to umożliwia zarówno odczyt jak i modyfikację zawartości bazy danych). Następnym krokiem jest uruchomienie API dla panelu administratora - od tego momentu będzie ono przyjmować dowolne zapytania - skutkujące wywoływaniem określonych funkcji, takich jak chociażby wymuszenie synchronizacji łańcucha. Następnie system stworzy instancję węzła sieci P2P - który otworzy się na połączenia od innych węzłów w sieci - każdy węzeł to uczestnik systemu. Węzeł ten będzie od uruchomienia dostępny na zapytania od innych użytkowników. Kolejnym krokiem będzie zsynchronizowanie łańcucha blockchain. W tym celu nasz węzeł połączy się z wszystkimi węzłami, które zapisane są na liście węzłów i zapyta je o długość ich łańcucha. Na podstawie tej informacji oraz zapisanych lokalnie danych o

samym łańcuchu ustali on od którego węzła uzyskać informacje o łańcuchu. Na wybór węzła wpływa przekazana długość łańcucha oraz wiarygodność węzła - autoryzowane węzły są bardziej wiarygodne. Wybór jest też częściowo losowy, żeby uniknąć sytuacji w której wszystkie węzły wybierają zawsze ten sam cel. Po ustanowieniu połączenia z celem nasz węzeł zsynchronizuje z nim listę znanych węzłów w sieci, po czym zacznie proces porównywania i synchronizacji łańcucha. Zostanie wysłane zapytanie dla każdego brakującego bloku, a sam blok będzie sprawdzany przed dodaniem go do naszej bazy. Sprawdzany będzie jego podpis - czy został wykonany przez autoryzowanego uczestnika oraz czy podpisany hasz nie uległ zmianie - a także to czy wskazuje on prawidłowo na poprzednika. Jeżeli blok zostanie uznany za nieprawidłowy - połączenie zostanie przerwane, a węzeł z którym wykonywaliśmy synchronizację zostanie uznany za niewiarygodny. Proces wybierania wiarygodnego węzła do wykonania synchronizacji ponowi się. Po zakończeniu synchronizacji system przygotuje listę wszystkich węzłów, które nie posiadały kompletnego łańcucha i wyśle im polecenie wykonania synchronizacji łańcucha. Uczestniczy który otrzymali takie polecenie nie wykonują go bezwarunkowo - rozważają wiarygodność węzła który wysłał im to polecenie, oraz sprawdzają czy rzeczywiście posiadają brakujące węzły. Finalnie system zwraca administratorowi informację o tym, że serwis jest zsynchronizowany i gotowy do dalszej pracy.

Na rysunku 5 przedstawiony został diagram sekwencji dla dodania nowego bloku (pracy dyplomowej do łańcucha. Jak widać na tym diagramie - panel admina razem z API już działają, więc to przez panel admin - lub pracownik posiadający odpowiednie uprawnienia dodaje nową pracę dyplomową uzupełniając wszystkie związane z nią informacje. Zapytanie to wraz z danymi trafia poprzez API do blockchain managera. Pierwszym krokiem który on wykonuje jest zsynchronizowanie łańcucha. Proces ten został pokazany dokładniej na rysunku 4. Po zsynchronizowaniu łańcucha przygotowywany jest blok z informacjami o pracy dyplomowej, który następnie jest podpisywany przy użyciu prywatnego klucza uczestnika. Blok ten jest następnie dodawany do bazy i udostępniany innych uczestnikom. Proces ten składa się z uzyskania od węzłów informacji o ich stanie łańcucha, posortowania ich na podstawie informacji o wiarygodności i obecnej ich długości bloku, a następnie przesyłana jest im prośba o zsynchronizowanie łańcucha. Następnie nasz węzeł wielokrotnie sprawdza czy nowy blok został już dodany do łańcucha innych uczestników. Robi to odpytując o niego węzły - o ile zwrócą mu informację, że ich obecna długość łańcucha wskazuje na to, że mogą one zawierać nowo dodany blok lub więcej nowych bloków. Po uzyskaniu odpowiedzi system sprawdza czy blok występujący u innych węzłów jest taki sam jak ten, który sam próbuje dodać. Jeżeli tak jest - proces zostaje zakończony. Jeżeli jednak uzyskany blok różni się od tego, który system próbuje dodać lub minęło już zbyt dużo czasu od dodania tego bloku - uznajemy, że dodawanie bloku nie powiodło się. W takiej sytuacji usuwamy z naszej bazy dodany blok i ponawiamy cały proces - zaczynając znowu od synchronizacji i stworzeniu nowego bloku. Blok który powstanie przy takiej kolejnej próbie będzie różnił się od poprzednika jedynie datą utworzenia. Jednak jego ponowne stworzenie jest potrzebne, żeby wyeliminować ryzyko błędu przy tworzeniu bloku.



Rysunek 4. Diagram sekwencji dla uruchomienia aplikacji



Rysunek 5. Diagram sekwencji dla dodania nowego bloku

Rozdział 6

Podsumowanie

Bibliografia

[1] BigchainDB GmbH, „BigchainDB 2.0 The Blockchain Database", BigchainDB GmbH, Berlin, Germany, spraw. tech., maj 2018, Paper version 1.0.

[2] Chauhan, A., „A Review on Various Aspects of MongoDb Databases", *International Journal of Engineering Research & Technology (IJERT)*, t. 8, nr. 05, maj 2019, IJERTV8IS050031. Published by: http://www.ijert.org. Licensed under a Creative Commons Attribution 4.0 International License., ISSN: 2278-0181.

[3] Django Software Foundation, *Django: The web framework for perfectionists with deadlines — Databases*, Documentation version 5.1, accessed 3 February 2025, 2022. adr.: https://docs.djangoproject.com/en/5.1/ref/databases/#.

[4] Golosova, J. i Romanovs, A., „The Advantages and Disadvantages of the Blockchain Technology", w *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2018, s. 1–6. DOI: 10.1109/AIEEE.2018.8592253.

[5] Grolleau, G., Lakhal, L. i Mzoughi, N., „An introduction to the economics of fake degrees", *Journal of Economic Issues*, t. 42, nr. 3, s. 673–693, 2008.

[6] Hayes, A. S., „A Cost of Production Model for Bitcoin", Department of Economics The New School for Social Research, New York, NY, spraw. tech., lut. 2015.

[7] Hussein, Z., Salama, M. i El-Rahman, S., „Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms", *Cybersecurity*, t. 6, s. 30, 2023. DOI: 10.1186/s42400-023-00163-y. adr.: https://doi.org/10.1186/s42400-023-00163-y.

[8] John, K., O'Hara, M. i Saleh, F., „Bitcoin and Beyond", *Annual Review of Financial Economics*, t. 14, s. 95–115, 2022.

[9] Kalajdjieski, J., Raikwar, M., Arsov, N., Velinov, G. i Gligoroski, D., „Databases fit for blockchain technology: A complete overview", *Blockchain: Research and Applications*, t. 4, nr. 1, s. 100 116, 2023, ISSN: 2096-7209. DOI: https://doi.org/10.1016/j.bcra.2022.100116. adr.: https://www.sciencedirect.com/science/article/pii/S2096720922000574.

[10] Raikwar, M., Gligoroski, D. i Kralevska, K., „SoK of Used Cryptography in Blockchain", *IEEE Access*, t. 7, s. 1–1, paź. 2019. DOI: 10.1109/ACCESS.2019.2946983.

[11] Saad, M., Spaulding, J., Njilla, L., Kamhoua, C. A., Nyang, D. i Mohaisen, A., „Overview of attack surfaces in blockchain", w *Blockchain for Distributed Systems Security*, Shetty, S. S., Kamhoua, C. A. i Njilla, L. L., red., Wiley-IEEE Computer Society Pr, 2019, s. 51–62, ISBN: 978-1-119-51960-7.

[12] Sayed, R. H., „Potential of Blockchain Technology to Solve Fake Diploma Problem", prac. mag., University of Jyväskylä, Department of Computer Science i Information Systems, 2019.

[13] Segendorf, B., „Have virtual currencies affected the retail payments market?", *Economic Commentaries, no. 2, Sveriges Riksbank*, 2014.

[14] Turing, *Python FastAPI vs Flask: A Detailed Comparison*. adr.: https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison.

[15] Wikipedia contributors, *Raft (algorithm) — Wikipedia, The Free Encyclopedia*, [Online; accessed 3-February-2025], 2025. adr.: https://en.wikipedia.org/w/index.php?title=Raft_(algorithm)&oldid=1270154233.

Wykaz skrótów i symboli

Spis rysunków

1 Diagram architektury systemu 18
2 Proces podpisywania bloku [10] 19
3 Diagram klasy bloku 20
4 Diagram sekwencji dla uruchomienia aplikacji 22
5 Diagram sekwencji dla dodania nowego bloku 23

Spis tabel

1 Porównanie bibliotek FastAPI i Flask [14] 16

Spis załączników