



Politechnika  
Śląska

**POLITECHNIKA ŚLĄSKA**  
**WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI**

Praca inżynierska

Zastosowanie technologii Blockchain w implementacji prostego  
systemu do głosowania on-line

autor: Kamil Susek

kierujący pracą: dr inż. Marcin Połomski

Gliwice, styczeń 2021



## Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody\* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej\*.

Gliwice, dnia 6 stycznia 2021

.....  
(podpis)

.....  
(poświadczenie wiarygodności  
podpisu przez Dziekanat)

\* podkreślić właściwe



## Oświadczenie promotora

Oświadczam, że praca „Zastosowanie technologii Blockchain w implementacji prostego systemu do głosowania on-line” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 6 stycznia 2021

.....  
(podpis promotora)



# Spis treści

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Wstęp</b>                                      | <b>1</b>  |
| 1.1      | Cel i zakres pracy . . . . .                      | 2         |
| 1.2      | Charakterystyka rozdziałów . . . . .              | 2         |
| <b>2</b> | <b>Blockchain</b>                                 | <b>3</b>  |
| 2.1      | Architektura Blockchain . . . . .                 | 4         |
| 2.1.1    | Łańcuch bloków . . . . .                          | 4         |
| 2.1.2    | Sieci Peer-to-peer . . . . .                      | 5         |
| 2.1.3    | Konsensus . . . . .                               | 6         |
| 2.1.4    | Funkcja skrótu . . . . .                          | 8         |
| 2.1.5    | Smart kontrakty . . . . .                         | 9         |
| 2.2      | Zastosowania technologii Blockchain . . . . .     | 9         |
| 2.2.1    | Kryptowaluty . . . . .                            | 9         |
| 2.2.2    | Finanse . . . . .                                 | 10        |
| 2.2.3    | Internet of Things . . . . .                      | 10        |
| <b>3</b> | <b>E-voting</b>                                   | <b>11</b> |
| 3.1      | System estoński . . . . .                         | 11        |
| 3.1.1    | Przegląd rozwiązania . . . . .                    | 12        |
| 3.2      | Evoting w Walnych Zgromadzeniach . . . . .        | 14        |
| <b>4</b> | <b>Wymagania systemu i wykorzystane narzędzia</b> | <b>15</b> |
| 4.1      | Wymagania funkcjonalne . . . . .                  | 15        |
| 4.2      | Wymagania pozafunkcjonalne . . . . .              | 16        |
| 4.3      | Diagramy przypadków użycia . . . . .              | 16        |
| 4.4      | Wykorzystane narzędzia i technologie . . . . .    | 17        |
| 4.4.1    | Interfejs użytkownika . . . . .                   | 18        |
| 4.4.2    | Serwer . . . . .                                  | 19        |
| 4.4.3    | Baza danych . . . . .                             | 20        |
| 4.4.4    | Testowanie . . . . .                              | 20        |
| 4.4.5    | Metodyka pracy . . . . .                          | 20        |
| <b>5</b> | <b>Specyfikacja zewnętrzna</b>                    | <b>23</b> |
| 5.1      | Instalacja . . . . .                              | 23        |
| 5.1.1    | Instalacja dla Windows i MacOS . . . . .          | 23        |
| 5.1.2    | Instalacja dla Linux . . . . .                    | 23        |
| 5.2      | Struktura oprogramowania . . . . .                | 24        |

---

|          |   |           |
|----------|---|-----------|
| 5.3      | Uruchamianie modułów . . . . .                  | 24        |
| 5.4      | Aktywacja oprogramowania . . . . .              | 25        |
| 5.5      | Aplikacja administratora . . . . .              | 25        |
| 5.5.1    | Logowanie . . . . .                             | 25        |
| 5.5.2    | Lista wyborców . . . . .                        | 26        |
| 5.5.3    | Konfiguracja sieci Blockchain . . . . .         | 27        |
| 5.5.4    | Lista wyborów . . . . .                         | 27        |
| 5.5.5    | Konfiguracja wyborów . . . . .                  | 28        |
| 5.5.6    | Nadzorowanie wyborów . . . . .                  | 30        |
| 5.6      | Aplikacja użytkownika . . . . .                 | 33        |
| 5.6.1    | Logowanie . . . . .                             | 33        |
| 5.6.2    | Instrukcja obsługi . . . . .                    | 34        |
| 5.6.3    | Głosowanie . . . . .                            | 35        |
| 5.6.4    | Wyniki . . . . .                                | 36        |
| <b>6</b> | <b>Specyfikacja wewnętrzna</b>                  | <b>39</b> |
| 6.1      | Architektura systemu . . . . .                  | 39        |
| 6.1.1    | Architektura pełnego systemu . . . . .          | 39        |
| 6.1.2    | Architektura sieci Blockchain . . . . .         | 40        |
| 6.2      | Struktury danych . . . . .                      | 41        |
| 6.2.1    | Łańcuch bloków . . . . .                        | 41        |
| 6.2.2    | Baza danych . . . . .                           | 44        |
| 6.3      | Algorytmy . . . . .                             | 46        |
| 6.3.1    | Proof of Work . . . . .                         | 46        |
| 6.3.2    | Proof of Authority . . . . .                    | 48        |
| 6.4      | Zastosowane wzorce projektowe . . . . .         | 49        |
| 6.4.1    | Singleton . . . . .                             | 49        |
| 6.4.2    | Repozytorium . . . . .                          | 50        |
| 6.4.3    | Strategia . . . . .                             | 51        |
| 6.4.4    | Redux . . . . .                                 | 53        |
| <b>7</b> | <b>Weryfikacja i walidacja</b>                  | <b>57</b> |
| 7.1      | Testowanie aplikacji serwerowych . . . . .      | 57        |
| 7.2      | Testowanie aplikacji przeglądarkowych . . . . . | 58        |
| 7.3      | Weryfikacja systemu . . . . .                   | 58        |
| <b>8</b> | <b>Podsumowanie i wnioski</b>                   | <b>63</b> |



# Rozdział 1

## Wstęp

Rzeczywista technologia pozwala na tworzenie coraz większych i bardziej bezpiecznych systemów, co prowadzi do przenoszenia niektórych procesów życia społecznego, do świata wirtualnego. Takie działania pozwalają na zwiększenie efektywności i zmniejszenie kosztów tych procesów. Dodatkowo wyeliminowanie wpływu czynnika ludzkiego zwiększa wiarygodność takiego procesu i poprawia bezpieczeństwo. Przykładem takiego procesu jest głosowanie, które w tradycyjnej formie jest czasochłonne i trudne w organizacji. Natomiast przykładem technologii, która doprowadziła do prawdziwej rewolucji jest Blockchain. Blockchain odpowiada za przełomowe rozwiązanie w sektorze finansów, jakim jest Bitcoin, który zapoczątkował rozwój systemów odpowiedzialnych za istotne procesy życia społecznego.

Tematem niniejszej pracy jest głosowanie *on-line*, czyli system głosowania przez internet, który będzie wspierany przez technologię Blockchain. W ramach pracy został zaprojektowany model systemu, którego część została zaimplementowana. Do zaimplementowanej części systemu należą: aplikacje internetowe wyborcy oraz organizatora i grupa serwisów dostarczających REST (Representational State Transfer) API (Application Programming Interface). Aplikacja wyborcy pozwala na zalogowanie się, wysłanie głosu i uzyskanie wyników wyborów. Natomiast aplikacja organizatora dostarcza narzędzia do konfiguracji i nadzorowania procesu wyborczego.

W pracy zastosowano sieć Blockchain, jako środek zabezpieczający głosy przed usunięciem lub podmiianą. Blockchain to rozproszona sieć węzłów przechowujących kopie danych, w łańcuchach bloków. Łańcuch bloków to specjalna struktura danych, która dzięki specjalnym algorytmom kryptograficznym pozwala na zabezpieczenie danych przed podmiianą lub usunięciem. W pracy zaprojektowano schemat działania takiej sieci oraz dostarczono narzędzia do jej konfiguracji i uruchamiania.

Do realizacji projektu zastosowano techniki oraz narzędzia pozwalające na tworzenie aplikacji przeglądarkowych i serwerowych. W pracy przedstawiono narzędzia, wzorce projektowe, algorytmy i struktury danych wykorzystane do implementacji rozwiązania.

## 1.1 Cel i zakres pracy

Głównym celem pracy jest zastosowanie technologii Blockchain w implementacji prostego systemu do głosowania *on-line*. Osięgnięcie celu wymaga zrealizowania zakresu pracy, który obejmuje:

- analizę dziedziny i przegląd literatury dotyczącej wyborów internetowych i technologii Blockchain,
- projekt oraz implementacja systemu do przeprowadzenia wyborów *on-line*,
- implementację części systemu z zastosowaniem wybranych narzędzi,
- weryfikację i walidację aplikacji.

## 1.2 Charakterystyka rozdziałów

Niniejsza praca została podzielona na kilka rozdziałów. W rozdziale pierwszym zostały przybliżone cele projektu, jego założenia, a także zakres pracy. Rozdział drugi poświęcony jest analizie zagadnienia wchodzącego w skład pracy, jakim jest Blockchain. W tym rozdziale przybliżono koncepcję technologii Blockchain, jej budowę i zastosowania. Rozdział trzeci zawiera przegląd rozwiązań dotyczących głosowania *on-line*, w zakresie tematyki, której dotyczy praca. W rozdziale czwartym sformułowano wymagania funkcjonalne i pozafunkcjonalne systemu. W tym rozdziale sformułowano wymagania stawiane systemowi. Rozdział czwarty zawiera również diagramy przypadków użycia i opis wykorzystanych technologii. Rozdział piąty dotyczy specyfikacji zewnętrznej oprogramowania. W tym rozdziale zawarto informacje dotyczące instalacji i uruchamiania oprogramowania, a także scenariusze działania systemu. Rozdział szósty zawiera informacje dotyczące specyfikacji wewnętrznej. W tym rozdziale znajduje się opis architektury systemu, wykorzystane struktury danych i szczegóły implementacyjne wybranych części systemu. Rozdział szósty zawiera informacje o wykorzystanych wzorcach projektowych i algorytmach. Rozdział siódmy dotyczy sposobów testowania poszczególnych modułów oprogramowania. W rozdziale siódmym umieszczono również przebieg weryfikacji systemu, na podstawie scenariusza zastosowania systemu w głosowaniu. Natomiast rozdział ósmy jest podsumowaniem pracy, a w jego skład wchodzi spełnione założenia projektu, napotkane błędy i wnioski.

# Rozdział 2

## Blockchain

Podstawy teoretyczne technologii Blockchain powstały już w roku 1991, a zostały opracowane przez dr Stuart'a Haber'a oraz dr W. Scott Stornett'a [18]. Naukowcy zajmowali się opracowaniem systemu, zabezpieczającego cyfrowe dokumenty przed podrobieniem, bądź podmianą. System opierał się na łańcuchach bloków, gdzie w każdym bloku znajdowały się cyfrowe dane dokumentu. Podczas dodawania nowego dokumentu do łańcucha, podpisywano go za pomocą tak zwanego stempla czasu (ang. timestamp), a następnie dokument był łączony z poprzednim dokumentem. Łączenie polega na przypisaniu nowemu blokowi, wskaźnika na poprzedni dokument. Wartością wskaźnika były określone dane poprzedniego dokumentu, co stanowiło zabezpieczenie dokumentów. Jeżeli zawartość dokumentu w łańcuchu uległaby zmianie, to należałoby zmienić również wskaźnik na ten dokument [19].

Kamieniem milowym w popularyzacji tego pomysłu był open-source'owy projekt o nazwie Bitcoin. Projekt ten powstał w roku 2009, a jego autorstwo przyznawane jest Satoshi'emu Nakamoto. Bitcoin to kryptowaluta, której podstawa działania oparta jest o wykorzystanie systemu Blockchain. Transakcje wykonane za pomocą Bitcoin'a zapisywane są na tak zwanym arkuszu, który jest widoczny dla każdego uczestnika sieci. Blockchain odgrywa swoją rolę w rejestrowaniu nowych transakcji i zapisywaniu ich w arkuszu. Arkusz ma strukturę łańcucha bloków, a sama architektura Blockchain została ulepszona o działanie w rozproszonej sieci *Peer-to-peer*. *Peer-to-peer* pozwala na udostępnienie zawartości Bitcoin'a milionom użytkowników na całym świecie, a dodatkowo zabezpieczenie sieci. Bitcoin wykorzystuje algorytm konsensusu, w celu zapewnienia spójności zawartości węzłów sieci. Jest to algorytm o nazwie *Proof of Work*, który odpowiada za silnie kojarzony z Bitcoin'em proces tzw. "kopania Bitcoinów". Działanie *Proof of Work* polega na włożeniu odpowiedniej ilości mocy obliczeniowej przez węzły sieci, poprzez rozwiązanie zagadki matematycznej, w celu utworzenia nowego bloku. Węzeł który rozwiąże zagadkę jako pierwszy nagradzany jest odpowiednią liczbą środków w Bitcoin'ie. Węzeł, który ułoży najdłuższy łańcuch jest synchronizowany z innymi [7].

Kombinacja rozwiązań tworzących Blockchain, w kryptowalucie Bitcoin wywołała poruszenie na całym świecie. Zaczęto rozmyślać nad nowymi rozwiązaniami, których podstawą miał być Blockchain. Blockchain znalazł swoje zastososo-

wanie w systemach dokumentowania transakcji, ze względu na redukcję kosztów utrzymania oraz wydajność, co jest spowodowane brakiem pośredników [23]. Dodatkowo technologia cały czas się rozwija. Kolejnym rewolucyjnym krokiem dla Blockchainu było utworzenie przez Vitalik’a Buterin’a nowej kryptowaluty, o nazwie Ethereum. Ethereum poza funkcjami płatniczymi kryptowaluty wprowadza *smart* kontrakty, czyli możliwość stworzenia kodu i uruchomienia go w systemie Ethereum. Programy mogą wchodzić w interakcje z węzłami w sieci, co pozwala na tworzenie ogromnych rozproszonych aplikacji.

Technologia cały czas podlega rozwojowi i znajduje zastosowanie w nowych dziedzinach, jednakże przedstawione powyżej koncepty są podstawą obecnego systemu Blockchain. W skrócie podstawę systemu Blockchain tworzą [23]:

- Wspólny arkusz - rozproszony łańcuch bloków,
- Ochrona dostępu,
- *Smart* kontrakty - kontrolowanie i programowanie transakcji,
- Konsensus - zawartość sieci jest bezpieczna, dzięki zgodzie pomiędzy jej uczestnikami.

## 2.1 Architektura Blockchain

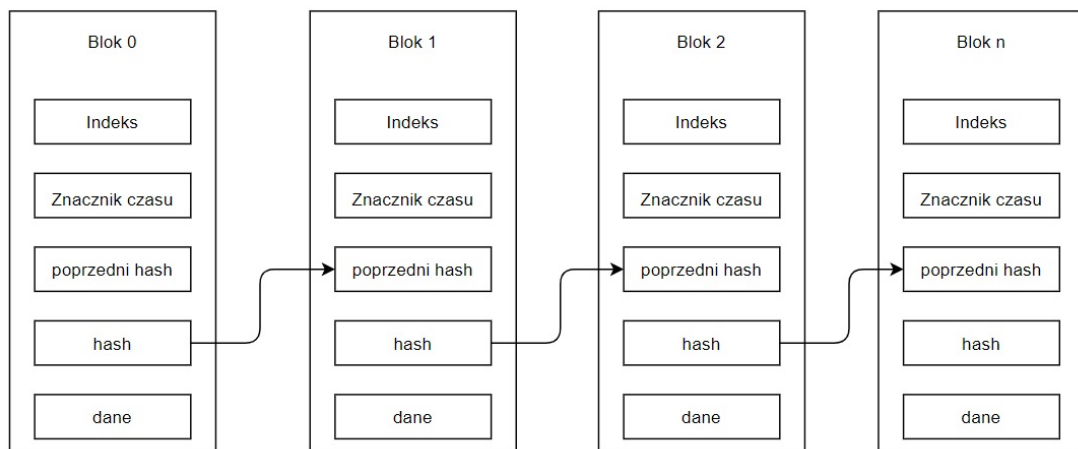
Obecnie pod terminem Blockchain kryje się wiele rozwiązań, co czasami prowadzi do rozmycia definicji Blockchain’u. Mówiąc Blockchain można mieć na myśli sam łańcuch bloków lub rozproszoną sieć urządzeń. Zdarzają się przypadki użycia terminu Blockchain zamiennie, z terminem Bitcoin [11]. Mylenie Bitcoin’a, z Blockchain’em jest błędem. Jednakże ten przypadek pokazuje, jak ważnym elementem Bitcoin’a jest Blockchain. W celu uściślenia rozwiązań charakterystycznych dla Blockchain’u, w tym podrozdziale opisano składniki, które wchodziły w skład technologii Blockchain.

### 2.1.1 Łańcuch bloków

Można powiedzieć, że łańcuch bloków to jeden z filarów, na którym opiera się cała architektura współczesnego Blockchain’u. Jak już wspomniano, idea łańcucha bloków pochodzi z pracy naukowej z 1991 roku. Wykorzystana w tym pomyśle zasada, według której każdy blok wskazuje na pewne dane obecne w poprzednim, cały czas funkcjonuje w systemach Blockchain.

Łańcuch bloków to struktura danych, w której każdy blok wskazuje na określoną wartość poprzedniego bloku. Wartość ta określana jest jako *hash*. *Hash* to ciąg znaków utworzony przez odpowiednią metodę kryptograficzną (zwaną również funkcją skrótu). Ważną cechą *hash*’u jest nieodwracalność i bezkolizyjność. Nieodwracalność oznacza, że znając *hash*, nie można wywnioskować jakie dane zostały *hash*’owane. Natomiast bezkolizyjność oznacza, że nie istnieją dwa różne zestawy danych, które po *hash*’owaniu dają ten sam wynik. Dodatkowo drobna zmiana

zawartości *hash'owanych* danych prowadzi do uzyskania nowego, kompletnie innego *hash'a*. Znalezienie takich danych wymaga dużej mocy obliczeniowej [20]. Dokładny opis działania *hash'owania* będzie miało miejsce w rozdziale 2.1.4.



Rysunek 2.1: Łańcuch bloków.

Na rysunku 2.1 ukazany jest schemat łańcucha bloków. Jak widać dane przechowywane są w blokach, a bloki połączone są za pomocą wskaźników na *hash* poprzedniego bloku. Dokładne działanie procedury *hash'owania* zawarto w rozdziale 2.1.4. Taka budowa stanowi zabezpieczenie łańcucha bloków przed modyfikacją lub usunięciem danych. Modyfikacja dowolnego bloku w sieci jest widoczna i łatwa do wykrycia. Wystarczy policzyć od początku *hash* dla każdego bloku i porównać go z polem następnego bloku, które przechowuje poprzedni *hash*.

### 2.1.2 Sieci Peer-to-peer

Zmodyfikowanie łańcucha bloków jest trudne, lecz nie jest niemożliwe. Wymaga wystarczająco dużej mocy obliczeniowej. Dodatkowo zamiast modyfikować łańcuch, można przejąć kontrolę nad urządzeniem, na którym uruchomiony jest łańcuch bloków, co daje możliwość zakłócenia pracy systemu opartego na działaniu łańcucha.

Twórcy Bitcoin'a rozwiązali podane problemy wykorzystując zdecentralizowaną sieć *Peer-to-peer*. Sieć *Peer-to-peer* składa się z węzłów, które komunikują się ze sobą, bez centralnego serwera. Jeżeli węzły chcą się ze sobą skontaktować, to wysyłają dane bezpośrednio do siebie. Takie rozwiązanie pozwala zwiększyć bezpieczeństwo sieci, ponieważ trudniej jest przejąć kontrolę nad wieloma maszynami tworzącymi całą sieć, niż nad centralnym systemem obsługującym działanie tej sieci. Dodatkowo *Peer-to-peer* zapewnia zabezpieczenie danych zgromadzonych w sieci, przed usunięciem, ponieważ każde urządzenie przechowuje swoje egzemplarze danych, które można traktować jako kopie zapasowe. Rozproszenie sieci na całym świecie zabezpiecza sieć przed cenzurowaniem, z czego korzysta Bitcoin i inne kryptowaluty.

*Peer-to-peer* jest przykładem ciekawego zastosowania architektury klient-serwer. Architektura klient serwer polega na podzieleniu systemu na stronę kliencką, która

zajmuje się wysłaniem żądań do serwera, oraz na stronę serwerową, której zadaniem jest odpowiadanie na żądania, przechowywanie danych i operowanie na tych danych. W sieci *Peer-to-peer* węzeł pełni rolę zarówno serwera, jak i klienta [22]. Przykładowo węzeł może żądać przesłania danych od innego węzła, a także odpowiedzieć na żądanie innego węzła.

### 2.1.3 Konsensus

Konsensus w sieci Blockchain to proces, w którym każdy z jej uczestników zgadza się na przeprowadzenie transakcji. Każda aktywność zmieniająca stan przechowywanych danych powinna być zatwierdzona przez uczestników sieci, aby zapewnić bezpieczeństwo i spójność sieci. Konsensus pozwala na wprowadzenie do sieci zestawu reguł, według których działa cała sieć. Takie działanie pozwala na zweryfikowanie zaufanego źródła i wykluczenie wpływu węzłów działających na szkodę sieci [13]. W kryptowalutach najczęściej wykorzystywane algorytmy konsensusu to *Proof of Work* i *Proof of Stake*. Istnieje również algorytm *Proof of Authority*, który głównie dotyczy rozwiązań prywatnych sieci Blockchain.

#### Proof of Work

*Proof of Work* to popularny w kryptowalucie Bitcoin algorytm uzyskiwania konsensusu. Algorytm ten wykorzystywany jest podczas tworzenia nowego bloku. Nowy blok wysyłany jest do wszystkich węzłów w sieci, które w celu dodania bloku do swojego łańcucha muszą zostać zatwierdzone. Aby blok został zatwierdzony węzeł musi wykonać pracę, czyli dokonać wkładu mocy obliczeniowej. Moc obliczeniowa wykorzystywana jest do rozwiązania zagadki matematycznej. Praca ta wykonywana jest przez węzeł i jest określana jako *mining* (pol. wydobywanie). Ten węzeł, który jako pierwszy rozwiąże zagadkę, otrzyma nagrodę w postaci określonej ilości środków, w kryptowalucie Bitcoin. W typowych rozwiązaniach dla Bitcoin'a poziom trudności dodania nowego bloku jest tak skonstruowany, aby nowe bloki były dodawane, w odstępach około 10 minutowych [2].

Takie rozwiązanie premiuje węzły, o większej mocy obliczeniowej. Dlatego w celu utrzymania spójności danych należy przeprowadzać synchronizację węzłów. Synchronizacja przebiega następująco: każdy węzeł rozsyła do całej sieci swój łańcuch, węzły porównują długość swojego łańcucha z nadesłanym łańcuchem i w węźle zapisywany jest dłuższy łańcuch.

*Proof of work* to rozwiązanie towarzyszące architekturze Blockchain, od momentu jej powstania. Bitcoin poza dokonywaniem transakcji, pozwala na zarabianie poprzez tworzenie tzw. węzłów górniczych. Węzeł górniczy zajmuje się realizacją algorytmu *Proof of Work*, co oznacza rozwiązywanie zagadki matematycznej. Fizycznie jest to urządzenie, ze specjalistycznymi podzespołami i oprogramowaniem, które pozwala na uzyskanie wysokiej efektywności w wydobywaniu Bitcoin'a [17]. Głównym problemem tego rozwiązania jest zużycie energii elektrycznej. W roku 2017 zużycie energii elektrycznej, przez wydobywanie Bitcoin'a było większe, niż w 159 krajach na świecie [1]. W roku 2019 naukowcy z *University of Cambridge* stworzyli specjalny indeks *CBECI* (Cambridge Bitcoin Electricity

Consumption Index), ukazujący dane dotyczące zużycia energii elektrycznej, przez Bitcoin'a [3].

W tym algorytmie czynnik wkładu fizycznych zasobów, jest głównym motywem do uczciwego postępowania. Jednakże głównym czynnikiem sprawiającym, że ten algorytm działa jest pewność, że ponad 50% uczestników sieci działa uczciwie. Ogrom inwestowanych zasobów wpływa na bezpieczeństwo sieci. Duża zdecentralizowana sieć Blockchain dysponuje ogromną mocą obliczeniową, na jej moc składa się moc wszystkich węzłów w sieci. Węzły różnią się mocą obliczeniową, natomiast decentralizacja sieci sprawia, że żaden z węzłów nie może przejąć kontroli nad całą siecią. Im większa decentralizacja tym bardziej zmniejsza się wpływ pojedynczego węzła na całą sieć. Co innego, gdy do sieci dołączona zostanie grupa węzłów, które będą stanowiły 51% mocy obliczeniowej sieci oraz będą ze sobą współpracowały, działając na szkodę sieci. Taki scenariusz nazywany jest atakiem 51%. Dysponując 51% mocy obliczeniowej sieci, można bez wiedzy uczciwych uczestników, w szybszy sposób dodawać nowe bloki, co prowadzi do przejścia obsługi dodawania zawartości, przez szkodliwych uczestników sieci [21].

W przypadku dużych kryptowalut atak 51% jest bardzo rzadkim zjawiskiem, gdyż jest on bardzo drogi. Atak ten może być opłacalny jedynie wtedy, gdy atakujący chce zdestabilizować sieć. W przypadku Bitcoin'a destabilizacja sieci nie ma większego sensu, ponieważ dysponując taką mocą obliczeniową można uzyskać ogromne przychody z kopania Bitcoin'a, dodatkowo uzyskanie takiej mocy obliczeniowej w rozrastających się sieciach Blockchain dla Bitcoin'u jest na tyle drogie, że w teorii jest uznawane za nieosiągalne. Z tego powodu, w przypadku sieci Blockchain, bardzo ważne jest zadbanie o jak największe rozproszenie węzłów w sieci *Peer-to-peer* [6].

## Proof of Stake

*Proof of Stake* to kolejny algorytm używany do osiągnięcia konsensusu w sieciach Blockchain. Jego największą zaletą jest rozwiązanie problemu wykorzystania ogromnych ilości zasobów energii elektrycznej, przez algorytm *Proof of Work*.

Działanie algorytmu *Proof of Stake* polega na wybieraniu spośród dostępnych węzłów sieci Blockchain walidatora. Walidator to węzeł, którego zadaniem jest weryfikacja poprawności bloku dołączanego do łańcucha. Wybór walidatora może się odbywać na różne sposoby, od całkowicie losowego wyboru, po wybór z uwzględnieniem czynników takich jak np. wiek stawki węzła, wysokość stawki. Stawka to zablokowane na koncie danego węzła jednostki waluty (kryptowaluty). Stawka jest wymagana, aby węzeł mógł uczestniczyć w losowaniu.

W przypadku algorytmu *Proof of Stake* głównym motywem uczciwego uczestnictwa w sieci jest możliwość utracenia części stawki. W przypadku wykrycia przez sieć nieprawidłowości w bloku dodanym przez konkretny węzeł, traci on określoną część stawki.

*Proof of Stake* jest o wiele bardziej ekologiczny, jeśli chodzi o zużycie energii elektrycznej, niż *Proof of Work*. W *Proof of Stake* głównym czynnikiem determinującym moc sieci jest zgromadzony w niej wkład finansowy w postaci stawek. W przypadku tego algorytmu również występuje problem ataku 51%, jednakże

tym razem atakujący musi dysponować odpowiednią ilością środków płatniczych, a dokładnie musi włożyć 51% ogólnej stawki, co w przypadku Bitcoin'a jest mało prawdopodobne [9].

## Proof of Authority

*Proof of Authority* to rozwiązanie skłaniające się ku mniej zdecentralizowanym systemom, które są tworzone na prywatne potrzeby i wymagają dużej przepustowości. *Proof of Authority* jest w swoim działaniu nieco zbliżony do *Proof of Stake*. Zamiast stawki podanej w jednostkach odpowiedniej waluty, w *Proof of Authority* używana jest reputacja walidatora. Prowadzi to do skonstruowania sieci, w której uczestniczą tylko zaufani walidatorzy. Uzyskanie statusu walidatora wiąże się z dużym nakładem pracy, w celu spełnienia kryteriów i uzyskania odpowiednio wysokiej reputacji. Podczas projektowania takiego systemu należy zwrócić szczególną uwagę, na stworzenie kryteriów o wysokiej trudności do spełnienia, aby wykluczyć węzły o szkodliwym działaniu.

Trudność uzyskania statusu walidatora bardzo mocno wpływa na stopień zdecentralizowania. Decentralizacja porównywalna z rozwiązaniem *Proof of Stake* jest niemożliwa do osiągnięcia. Niski stopień decentralizacji wymaga pełnego zaufania do walidatorów, co może stanowić słaby punkt systemu, gdy zaufany walidator zostanie przekupiony i dokona szkodliwej manipulacji na zawartości systemu.

Algorytm *Proof of Authority* świetnie sprawdza się w prywatnych sieciach Blockchain, gdy ważna jest wysoka przepustowość łącza. Jednakże należy pamiętać o odpowiednim doborze walidatorów [8].

### 2.1.4 Funkcja skrótu

Funkcje skrótu w systemach informatycznych pełnią rolę zabezpieczenia integralności danych. Zabezpieczenie danych polega na wykryciu ich modyfikacji. Matematyczna definicja funkcji skrótu  $h$  to odwzorowanie wiadomości  $m$  o dowolnej, skończonej długości, w ciąg bitów o określonej, stałej długości  $n$  [20]:

$$h : \{0, 1\}^* \rightarrow \{0, 1\}^n, \text{ gdzie } : \{0, 1\}^* = \bigcup_{i \in N} \{0, 1\}^i, N = \{0, 1, \dots\}, n \in N \quad (2.1)$$

Funkcja skrótu cechuje się wysoką podatnością na zmiany. Zmiana jednego bitu w danych wejściowych powoduje otrzymanie zupełnie innych wyników.

Przykładowo dla ciągu znaków *hash0*, dla funkcji skrótu *SHA256* wynikiem jest ciąg znaków

*6ac73742db534bebcc9af1453c5637ee5bb5d7c9628ec2f26cf9777c89e96d8*. Natomiast

zmieniając ciąg znaków na *hash1*, otrzymanym wynikiem jest *af316ecb91a8ee7ae99210702b2d4758f30cddc3bf61e3d8e787d74681f90a6*. Przykład obrazuje jak na wynikowy ciąg znaków wpływa zmiana jednego symbolu na wejściu.

Głównymi cechami funkcji skrótu są:

- nieodwracalność – na podstawie otrzymanego wyniku nie można bezpośrednio odtworzyć danych.



- bezkolizyjność – nie istnieją dwa różne ciągi znaków, które dają ten sam wynik.

Znalezienie danych wejściowych dla funkcji skrótu jest zadaniem wymagającym dużego nakładu mocy obliczeniowej. Dane można uzyskać losując wejściowy ciąg znaków i *hash’ując* go, a następnie porównując otrzymany *hash* z *hash’em*, który ma być rozszyfrowany. Jest to tak zwany atak brutalny (ang. brute force).

Popularnym zastosowaniem funkcji skrótu jest podpis cyfrowy. Podpis cyfrowy to zaszyfrowany *hash*, wygenerowany na podstawie danych potrzebnych do podpisu. Główną zaletą wykorzystania *hash’owania* w podpisie cyfrowym jest szybkość. *Hash’owanie* wiadomości jest szybsze od szyfrowania, a zaszyfrować wystarczy sam ciąg znaków o stosunkowo małej długości, w porównaniu do tekstu wiadomości.

Funkcje skrótu można spotkać w bazach danych. Hasła są przykładem informacji, które w bazie danych są przechowywane w postaci funkcji skrótu. Takie rozwiązanie zabezpiecza hasła, w przypadku wycieku danych. [20]

### 2.1.5 Smart kontrakty

Wraz z powstaniem kryptowaluty Ethereum do architektury Blockchain została dodana nowa funkcja. Dodano tak zwane *smart* kontrakty, które pozwalają na zaprogramowanie i uruchomienie kodu w sieci Blockchain. Takie programy można stosować, w celu kontrolowania sieci poprzez zdefiniowanie zestawu reguł dla sieci. Zdefiniowanie zestawu reguł pozwala na zwiększenie zaufania w sieci, a także automatyzację niektórych procesów.

*Smart* kontrakty pozwalają na tworzenie aplikacji, których podstawą jest Blockchain. Przykładem środowiska pozwalającego na tworzenie takich programów jest język Solidity, który został stworzony dla sieci Ethereum [5].

## 2.2 Zastosowania technologii Blockchain

Technologia Blockchain jest wykorzystywana w wielu dziedzinach życia, a jej głównym zadaniem jest zapewnienie bezpieczeństwa danych.

### 2.2.1 Kryptowaluty

Kryptowaluta to cyfrowy środek płatniczy. W przeciwieństwie do tradycyjnych walut, kryptowaluty nie są kontrolowane przez bank centralny. Środki płatnicze w kryptowalucie znajdują się w rozproszonej, po całym świecie sieci. Każda transakcja w tym środowisku jest zapisywana we współdzielonym rejestrze. Rejestr pozwala na ustalenie ile środków w danej kryptowalucie jest na koncie danego uczestnika sieci. Podstawą działania kryptowalut jest Blockchain. To Blockchain odpowiada za tworzenie rejestru transakcji, zabezpieczenie danych przed modyfikacją, współdzielenie danych i uzyskanie konsensusu. Przykładem takich kryptowalut są Bitcoin i Ethereum. [7]

### 2.2.2 Finanse

Dynamiczny rozwój technologii zapoczątkował rozwój innych branż, w tym sektora finansów. Powstała osobna gałąź tego sektora o nazwie FinTech. FinTech zajmuje się opracowywaniem i wdrażaniem nowych technologii do systemów finansowych na całym świecie. Korzyści płynące z unowocześnienia sektora finansów mogą skupiają się głównie na zwiększeniu szybkości transakcji przez brak pośredników, zmniejszeniu kosztów przepływu transakcji i zwiększenie bezpieczeństwa. FinTech zajmuje się między innymi szukaniem zastosowań dla technologii Blockchain. Jako przykład rozwijania technologii blockchain przez FinTech może posłużyć przykład trwałego nośnika wdrożonego przez PKO Bank Polski. Trwały nośnik pozwala na dostarczanie klientom banku prywatnych dokumentów w formie cyfrowej [14]. Twórcy tego rozwiązania zastosowali również mechanizm Smart Kontraktów. Smart Kontrakty są wykorzystywane do zautomatyzowania procesu egzekwowania umów ubezpieczeniowych [4]. Pozwala to na zmniejszenie uczestnictwa osób trzecich i redukcję kosztów.

### 2.2.3 Internet of Things

Internet of Things to technologia pozwalająca na łączenie ze sobą wielu urządzeń. Tak skonstruowane sieci znajdują swoje przeznaczenie w systemach czasu rzeczywistego w przemyśle, bądź w inteligentnym domu. IoT wykorzystuje serwery internetowe, do przechowywania i pobierania danych. Blockchain w usługach IoT pomaga zabezpieczyć serwery z danymi, poprzez ich decentralizację. [15]

# Rozdział 3

## E-voting

W rozdziale znajduje się przegląd wybranych istniejących rozwiązań do realizacji wyborów *on-line*. Jednym z przykładów takiego rozwiązania jest System estoński, który jest przykładem wykorzystania wyborów *on-line* przez struktury państwowe. Natomiast drugim przykładem jest oprogramowanie dostarczane przez KDPW (Krajowy Depozyt Papierów Wartościowych). W skład tego oprogramowania wchodzi aplikacja E-voting w Walnych Zgromadzeniach, która umożliwia realizację wyborów z wykorzystaniem technologii Blockchain.

### 3.1 System estoński

Przykładem regularnego wykorzystania e-votingu jest Estonia. Estonia jest krajem, który pierwszy udostępnił możliwość głosowania elektronicznego w wyborach lokalnych, na szczeblu krajowym. W pierwszych wyborach wzięło udział około 1% wyborców. Od roku 2005 w Estonii postępował rozwój systemów e-votingu. Wraz z następnymi wyborami zwiększała się liczba uczestników. W roku 2019 liczba wyborców, którzy skorzystali z e-votingu wyniosła 43,8% wyborców.

Na system e-votingu w Estonii składa się kilka mniejszych rozwiązań:

- identyfikacja za pomocą karty e-obywatela lub profilu e-obywatela w telefonie,
- architektura rozwiązania pozwala na wysłanie wielu głosów, a głosem wiążącym jest zawsze ostatni,
- serwery wykorzystywane podczas głosowania są pod szczególną ochroną i nie można uzyskać do nich dostępu z bezpośrednio z internetu (zabezpieczenie firewall'em),
- wykorzystywanie prywatnych kluczy i narzędzi kryptograficznych w celu zabezpieczenia dostępu do danych. Wykorzystywanie standardu SSL (ang. Secure Sockets Layer).

### 3.1.1 Przegląd rozwiązania

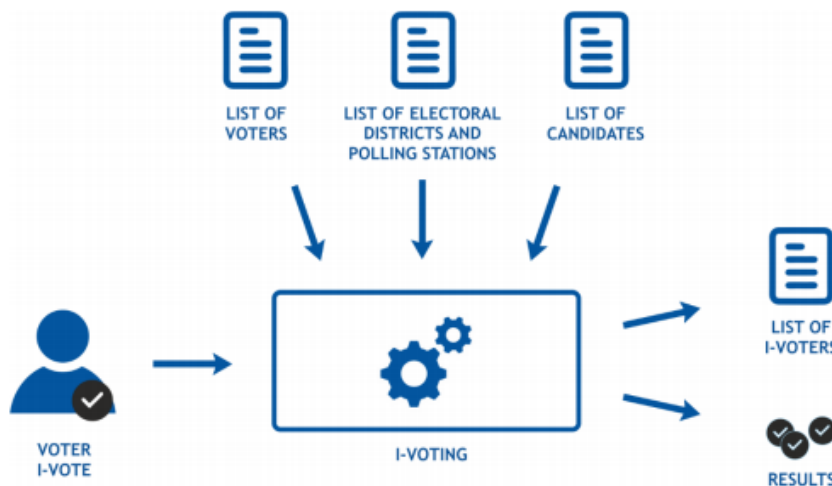
Rysunek 3.1 przedstawia moduły systemu estońskiego, które wchodzi w skład przebiegu całego procesu wyborów. Przebieg całego procesu wyborczego w Estonii można podzielić na kilka etapów:

- ogłoszenie wyborów,
- zarejestrowanie kandydatów,
- przygotowanie list wyborczych,
- głosowanie,
- liczenie głosów,
- ogłoszenie wyników.

Wsparcie e-votingu (w Estonii określanego i-votingiem) obejmuje ostatnie trzy etapy procesu wyborczego.

W skład systemu głosowania wchodzi baza danych (rys. 3.1) przechowująca:

- listę osób uprawnionych do głosowania (na rysunku *List of voters*),
- listę okręgów wyborczych (na rysunku *List of electoral districts and polling stations*),
- listę kandydatów lub opcji wyborczych (na rysunku *List of candidates*),
- listę e-wyborców (na rys. 3.1 *i-voters*).



Rysunek 3.1: Moduły systemu i-votingu [10].

Dostarczenie odpowiednich danych do systemu pozwala na walidację wyborcy, zgłoszenie przez niego głosu oraz zapisanie wyborcy, w bazie danych odpowiedzialnej za przechowanie listy e-wyborców. W skład logiki systemu wchodzi mechanizmy generujące wyniki. Wyniki e-votingu są scalane z wynikami wyborów tradycyjnych, przy czym scalanie nie pozwala na podwójne zliczanie głosu oddanego tradycyjnie i elektronicznie.



Rysunek 3.2: Proces i-votingu [10].

Rysunek 3.2 przedstawia proces wyborczy. W tym procesie można wyróżnić następujących aktorów:

- *voter* (wyborca) - za pośrednictwem aplikacji klienckiej (która w systemie estońskim jest aplikacją desktopową pobieraną przed wyborami) szyfruje i potwierdza swoim podpisem elektronicznym głos, który następnie wysyłany jest do zbieracza głosów,
- *collector* (zbieracz głosów) - to aplikacja serwerowa wyposażona w logikę pozwalającą na utworzenie głosu. Aplikacja waliduje dane wprowadzone przez wyborcę i elektronicznie podpisuje dane, które następnie przesyła do procesora,
- *processor* (procesor głosów) - zajmuje się przetwarzaniem głosów. Sprawdza poprawność danych otrzymanych z zbieracza głosów, wraz z podpisem elektronicznym. Usuwa głosy, które się powtarzają, zarówno głosy oddane elektronicznie, jak i te oddane w lokalach wyborczych. Sortuje głosy według okręgów wyborczych i usuwa z nich podpis elektroniczny, w celu anonimizacji głosu. Tak przetworzone głosy są mieszane według odpowiedniego algorytmu i wysyłane do Licznika,
- *tallier* (licznik głosów) - rolę licznika jest odebranie głosu od procesora, otwarcie go i dodanie przyporządkowanie do odpowiedniego wyniku.

Estoński system e-votingu jest spójny i dobrze zabezpieczony. Dlatego jest on wykorzystywany w ważnych wyborach krajowych. Rozwiązanie jest stale udoskonalane, aby mogło dotrzymać kroku rozwojowi technologii. [10]

## 3.2 Evoting w Walnych Zgromadzeniach

Krajowy Depozyt Papierów Wartościowych to instytucja finansowa, której zadaniem jest nadzorowanie i prowadzenie rejestracji obrotu instrumentami finansowymi w Polsce. KDPW pełni również rolę depozytu papierów wartościowych. KDPW jest spółką akcyjną, której współwłaścicielami są akcjonariusze - państwo (Skarb Państwa) i państwowe spółki finansowe (Giełda Papierów Wartościowych, Narodowy Bank Polski).

Walne Zgromadzenie to organ mający najwyższe uprawnienia w spółce akcyjnej. Walne Zgromadzenie pozwala akcjonariuszom na zarządzanie spółką. Podczas Walnego Zgromadzenia podejmuje się uchwały dotyczące spółki akcyjnej.

Krajowy Depozyt Papierów Wartościowych udostępnia swoim akcjonariuszom aplikacje eVoting oraz Walne Zgromadzenia, które przenoszą proces przeprowadzania Walnego Zgromadzenia do internetu. Zgodnie z art. 402 Kodeksu Spółek Handlowych [12], Walne Zgromadzenie zwoływane jest przez ogłoszenie go, co najmniej trzy tygodnie przed zaplanowanym terminem. W celu zwołania Walnego Zgromadzenia w aplikacji Evoting i Walne Zgromadzenia należy zarejestrować walne zgromadzenie. KDPW przesyła ogłoszenie, o Walnym Zgromadzeniu do jego uczestników. Wymiana informacji odbywa się drogą elektroniczną. Uczestnicy tworzą listę osób uprawnionych do udziału w Walnym Zgromadzeniu, która jest przesyłana do KDPW. KDPW generuje wykaz osób uprawnionych do uczestnictwa w Walnym Zgromadzeniu, a następnie dostarcza informacje o uczestnictwie w Walnym Zgromadzeniu indywidualnie, dla każdego uczestnika. Każdy uczestnik otrzymuje również kod autoryzacyjny, który wykorzystywany jest do potwierdzenia tożsamości uczestnika, w celu przyznania uprawnień do uczestnictwa w walnym zgromadzeniu. Przyznanie uprawnień jest równoznaczne z potwierdzeniem obecności na walnym zgromadzeniu i udostępnieniu uczestnikowi interfejsu do głosowania [16].

Za zapewnienie nieodwracalności wykonanych akcji i ogólnego bezpieczeństwa danych, w aplikacjach eVoting i Walne Zgromadzenia jest odpowiedzialny Blockchain.

# Rozdział 4

## Wymagania systemu i wykorzystane narzędzia

W rozdziale sformułowano i omówiono wymagania funkcjonalne i pozafunkcyjne. Dodatkowo przedstawiono przypadki użycia i aktorów systemu. W ostatniej części rozdziału zostały omówione technologie i narzędzia wykorzystane podczas tworzenia systemu.

### 4.1 Wymagania funkcjonalne

W tej sekcji umieszczono wszystkie funkcjonalności systemu. Do głównych części systemu należą: aplikacja internetowa wyborcy, aplikacja internetowa administratora wyborów, serwis udostępniający REST API i serwery sieci Blockchain. System dostarcza następujące funkcje:

- użytkownicy systemu muszą się zalogować, aby uzyskać dostęp do swoich zasobów,
- wyborca ma dostęp tylko do wyborów, do których jest przydzielony,
- wyborca ma dostęp do instrukcji obsługi,
- proces głosowania składa się z wyboru odpowiedniego głosowania, wyboru kandydatury i wysłania głosu,
- każdy wyborca może wysłać tylko jeden głos,
- tworzenie wyborów, poprzez konfigurator wyborów,
- dodawanie nowych wyborców,
- konfigurowanie sieci Blockchain,
- nadzorowanie wyborów,
- publikacja wyników wyborów.

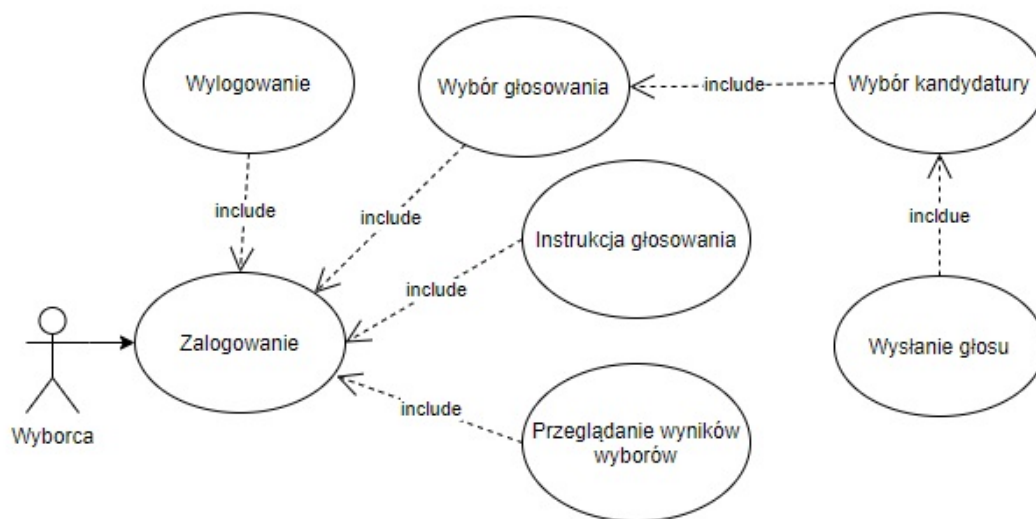
## 4.2 Wymagania pozafunkcjonalne

Wymagania pozafunkcjonalne definiują jakość usług dostarczanych przez system. W systemie e-votingu ważnymi elementami są bezpieczeństwo i czytelność procedury głosowania. Dlatego w celu uściślenia oczekiwań jakościowych systemu sformułowano następujące wymagania pozafunkcjonalne:

- dane logowania użytkowników są zabezpieczone, za pomocą algorytmu *SHA256*,
- przechowywane głosy są zabezpieczone przed modyfikacją, za pomocą technologii Blockchain,
- przejrzyste treści na stronie wyborcy,
- czytelna i łatwa w obsłudze procedura wyborcza na stronie wyborcy,
- przejrzyste treści na stronie administratora wyborów,
- czytelna i łatwa w obsłudze konfiguracja wyborów, w aplikacji internetowej,
- komunikacja pomiędzy modułami, z wykorzystaniem konwencji REST API,
- aplikacje internetowe wyborcy i administratora działają na najpopularniejszych przeglądarkach internetowych.

## 4.3 Diagramy przypadków użycia

W systemie można wyróżnić dwóch aktorów. W tym podrozdziale zostały przedstawione i omówione diagramy przypadków użycia dla tych aktorów.



Rysunek 4.1: Diagram przypadków użycia wyborcy.

Pierwszym z wymienionych aktorów jest użytkownik aplikacji, czyli wyborca. Przewidziane akcje dostępne dla wyborcy to zalogowanie, wylogowanie,

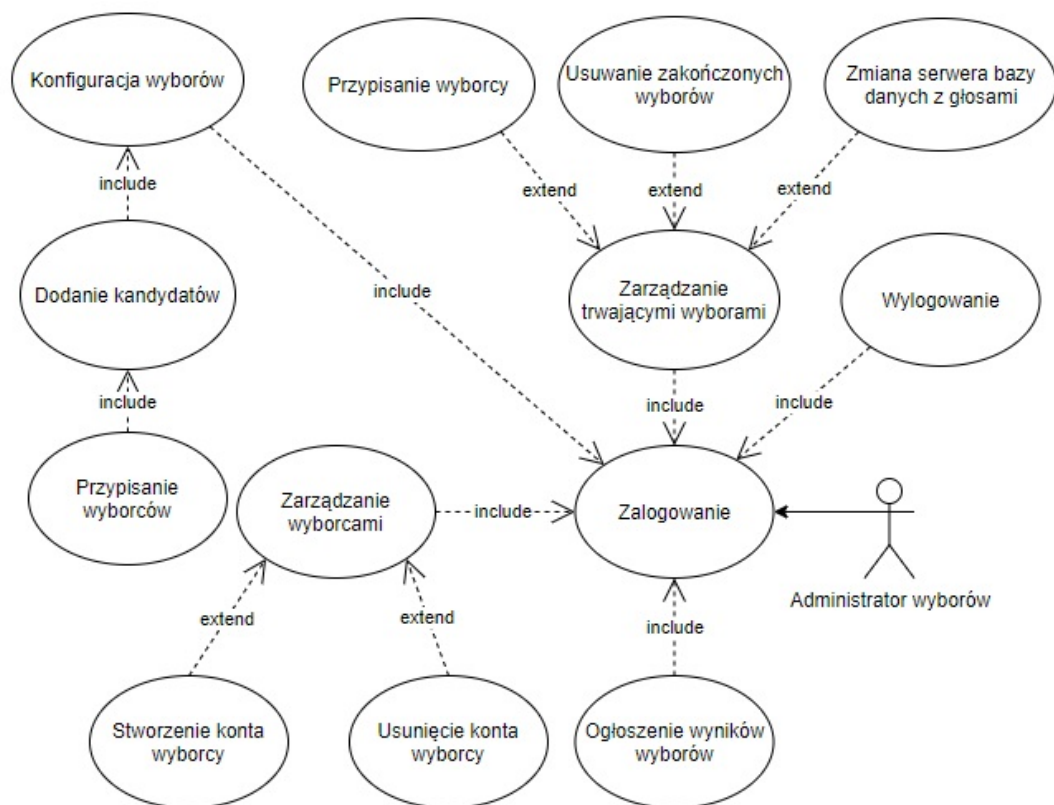


dostęp do instrukcji głosowania, głosowanie i odbiór wyników wyborów. Rysunek 4.1 przedstawia diagram przypadków użycia dla wyborcy.

Drugim aktorem występującym w systemie jest administrator wyborów. Do zadań administratora należą:

- tworzenie wyborów - konfiguracja składników wyborów,
- modyfikacja wyborów - nadzorowanie procesu wyborczego,
- dodawanie wyborców do bazy danych - zarządzanie kontami wyborców,
- ogłoszenie wyników wyborów.

Dodatkowo administrator posiada swoje konto w systemie. Aby mógł korzystać z przydzielonych mu zasobów musi się zalogować. Rysunek 4.2 przedstawia diagram przypadków użycia dla administratora wyborów.



Rysunek 4.2: Diagram przypadków użycia administratora.

## 4.4 Wykorzystane narzędzia i technologie

System e-votingu składa się z wielu modułów, które dotyczą różnych rodzajów oprogramowania (aplikacje przeglądarkowe i serwerowe). W systemie są elementy odpowiedzialne za interfejs użytkownika, aplikację serwerową i bazę danych. Każdy

z tych rodzajów oprogramowania wymaga innego podejścia. Są to popularne dziedziny programowania, dlatego znalezienie rozwiązań nie stanowi problemu. W tym podrozdziale przedstawiono wykorzystane narzędzia i technologie.

#### 4.4.1 Interfejs użytkownika

W celu stworzenia rozbudowanego interfejsu użytkownika wykorzystano bibliotekę javascript-ową - ReactJs. React pozwala na tworzenie aplikacji internetowych z wykorzystaniem języków JavaScript, HTML i CSS. Takie podejście do tworzenia aplikacji określane jest jako SPA (Single Page Application), taka aplikacja posiada jeden plik *.html*, którego zawartość jest dynamicznie generowana w trakcie działania aplikacji. Podejście typu SPA pozwala na zredukowanie liczby zasobów potrzebnych do generowania strony, poprzez wykorzystanie tylko jednego pliku *.html*. Strona typu SPA działa płynnie, ponieważ nie wymaga ona ładowania szablonów z plików *.html*. Ważną cechą ReactJs jest podejście do tworzenia interfejsu użytkownika, jak do grupy komponentów. Komponent może być dowolnym elementem interfejsu, od paska nawigacji do przycisku. Każdy komponent posiada swój stan, czyli zestaw zmiennych kontrolujących jego funkcjonalności. Takie podejście pozwala na tworzenie uniwersalnych i konfigurowalnych komponentów, które można wykorzystywać w wielu miejscach aplikacji.

Biblioteka ReactJs to bardzo rozbudowane narzędzie, które dzięki swojej popularności posiada wiele dodatkowych rozwiązań współpracujących z całym środowiskiem ReactJs. Przykładem takiego dodatku jest biblioteka Redux, która pozwala na zarządzanie stanem aplikacji. Redux pozwala przenieść stan z komponentu do osobnego kontekstu. Takie rozwiązanie pozwala na umieszczenie tych części stanu aplikacji, które odnoszą się do różnych komponentów w jednym kontenerze. Stan aplikacji jest dostępny globalnie, a nie tylko dla komponentów potomnych. Przykładowo stan informujący o zalogowaniu do aplikacji powinien być dostępny dla całej aplikacji. Przeniesienie tego stanu do osobnego kontekstu pozwala na zdefiniowanie go w jednym miejscu i dostarczenie do odpowiednich komponentów. W projekcie wykorzystano opartą na Redux, bibliotekę Redux-Toolkit. Głównym argumentem za wykorzystaniem Redux-Toolkit jest mniejsza objętość kodu, która powstaje do uzyskania rozwiązania.

Podstawowym narzędziem odpowiedzialnym za wygląd interfejsu aplikacji internetowej są kaskadowe arkusze stylów, czyli popularny CSS (Cascade Style Sheets). Język CSS jest jedynym językiem odpowiedzialnym za określenie wyglądu strony internetowej, który jest rozumiany przez przeglądarkę. Jednakże istnieją rozwiązania, które pozwalają na wprowadzenie nowej składni z zachowaniem kompatybilności z danym środowiskiem uruchomieniowym (w tym przypadku z przeglądarką). Są to tzw. *preprocesory*. Preprocesor pozwala na przetworzenie danego kodu źródłowego, na określony kod wyjściowy. Przykładem preprocesora języka CSS, który został wykorzystany w projekcie jest SASS (Syntactically Awesome Style Sheets). SASS dostarcza funkcjonalności, które nie są dostępne w języku CSS. Przykładowo SASS pozwala na zagnieżdżanie instrukcji, deklarowanie i wywoływanie zmiennych oraz tzw. *mixiny*, które pozwalają na zadeklarowanie powtarzających się fragmentów kodu w jednym miejscu i ich wywoływanie bez koniecz-

ności przepisywania kodu. Sass pozwala na lepsze wykorzystanie dobrych praktyk programowania, takich jak uniknięcie powtarzania kodu.

Jak już wspomniano w tej sekcji ReactJs jest popularną biblioteką, co przekłada się na dużą liczbę rozszerzeń i dodatków. Dodatki obejmują również stylowanie aplikacji. W projekcie została wykorzystana biblioteka Material-Ui, która dostarcza gotowe zestawy komponentów i narzędzia do ich stylowania. Zaletą tego rozwiązania jest wykorzystywanie komponentów, które stworzono z użyciem dobrych praktyk tworzenia interfejsu użytkownika.

Aplikacje internetowe często wymagają komunikacji z zewnętrznym serwerem, w celu pobrania lub wysłania odpowiednich zasobów. Istnieje wiele rozwiązań, które pozwalają na wysyłanie żądań w standardzie HTTP (Hypertext Transfer Protocol). W projekcie wykorzystano javascript-ową bibliotekę Axios. Biblioteka dostarcza wiele mechanizmów do wysyłania żądań HTTP, a jej wielką zaletą jest łatwość obsługi.

JavaScript to język programowania, który świetnie sprawdza się w środowisku aplikacji przeglądarkowych. Język umożliwia wieloaspektowe podejście do programowania. Można pisać programy w sposób deklaratywny lub obiektowy. JavaScript spełnia standardy nowoczesnych języków programowania, o czym świadczyć może ogromna popularność tego języka. Jednakże niektóre właściwości tego języka mogą utrudniać tworzenie oprogramowania. Przykładem jest dynamiczne typowanie. W JavaScript każda zmienna nie ma przypisanego typu na stałe. Takie rozwiązanie ma swoje wady i zalety, jednak są przypadki, w których programista powinien mieć wybór, kiedy użyć zmiennej typowanej dynamicznie, a kiedy nie. Rozwiązaniem tego problemu jest język Typescript. Typescript jest nadzbiorem JavaScript co oznacza, że dostarcza on wszystkie funkcje JavaScript'u i jednocześnie rozszerza go o nowe. W Typescript można deklarować zmienne z określonym typem, a także używać typu *any*. Typ *any* określa zmienne, które mogą być typowane dynamicznie. Język Typescript dostarcza również możliwość definiowania własnych typów. W projekcie, w celu uniknięcia problemów związanych z dynamicznym typowaniem wykorzystano język Typescript.

#### 4.4.2 Serwer

Za stronę serwerową aplikacji odpowiada silnik NodeJs. NodeJs pozwala na tworzenie aplikacji w JavaScript, poza środowiskiem przeglądarkowym. Silnik stanowi również rozbudowane narzędzie pozwalające na tworzenie REST API. Zaletą NodeJs jest ogromna popularność, co jest zauważalne podczas szukania rozwiązań w internecie na napotkane problemy. Popularność silnika przekłada się również na jego bogatą dokumentację i dodatkowe biblioteki, które usprawniają proces tworzenia oprogramowania.

ExpressJs to biblioteka działająca na silniku NodeJs, która została wykorzystana w projekcie do tworzenia REST API i węzłów sieci Blockchain. ExpressJs dostarcza podstawowe narzędzia do tworzenia REST API, takie jak tworzenie punktów końcowych serwera, które pozwalają na obsługiwanie żądań HTTP. Zaletą biblioteki ExpressJs jest kompatybilność z wieloma innymi bibliotekami przeznaczonymi dla silnika NodeJs.

Prisma to jedna z bibliotek NodeJs, które są odpowiedzialne za obsługę bazy danych. Prisma dostarcza mechanizm ORM (Object Relational Mapping), czyli możliwość odwzorowania relacyjnej bazy danych, na obiekty danego języka programowania. Prisma generuje moduły odpowiedzialne za obsługę bazy danych i dostarcza je programiście. Takie rozwiązanie pozwala na utrzymanie całości aplikacji w jednym języku programowania, ponieważ nie wymaga korzystania z języka SQL (Structured Query Language). Wszystkie moduły obsługi bazy danych tworzone są automatycznie, dlatego Prisma pozwala oddzielić implementację bazy danych, od innych funkcjonalności aplikacji.

### 4.4.3 Baza danych

Wykorzystana baza danych działa na silniku SQLite. Baza danych znajduje się w pliku i jest umieszczona na serwerze wraz z REST API. SQLite to szybki, mały i łatwy w konfiguracji silnik baz danych. Jest to świetny silnik do prototypowania i tworzenia małych, a zarazem przenośnych baz danych.

Silnik SQLite można obsługiwać z poziomu konsoli lub za pomocą specjalnej aplikacji, która dostarcza interfejs graficzny prezentujący zawartość bazy danych. W projekcie wykorzystano aplikację SQLiteStudio, która pozwala na połączenie się z bazą danych, w celu zarządzania nią. SQLiteStudio pozwala na tworzenie tabel, dodawanie zawartości, usuwanie danych i przeszukiwanie bazy danych. Aplikacja pozwala również na pisanie własnych zapytań w języku SQL. Tego rodzaju aplikacje są przydatne podczas tworzenia tabel w bazie danych i testowania aplikacji.

### 4.4.4 Testowanie

Testowanie strony serwerowej systemu odbywało się przez aplikację Postman, która pozwala na wysyłanie żądań HTTP. W aplikacji można skonfigurować żądanie HTTP, tak aby zasymulować odpowiednie scenariusze żądania. Pozwala to przetestować działanie wszystkich funkcjonalności API.

Testowanie aplikacji w środowisku lokalnym pozwala odpowiedzieć na pytanie, czy całość komponentów działa w kontrolowanym środowisku. Natomiast przeniesienie aplikacji do sieci pozwala na przetestowanie wydajności aplikacji. W projekcie, w celu udostępnienia aplikacji w internecie wykorzystano serwisy Netlify oraz Heroku. Netlify pozwala na udostępnienie aplikacji internetowej, pod adresem generowanym przez serwis. Heroku zajmuje się uruchomieniem aplikacji serwerowej, która również dostępna jest w sieci pod odpowiednim adresem.

### 4.4.5 Metodyka pracy

Do prowadzenia projektu wykorzystano repozytorium kontroli wersji GitHub. GitHub to narzędzie, które pozwala przechowywać kod źródłowy aplikacji, a także zapewnia jego archiwizację i wersjonowanie. Do przydatnych funkcji należy również tworzenie rozgałęzień na różnych etapach tworzenia aplikacji. Pozwala to na oddzielenie poszczególnych funkcji na odrębne gałęzie projektu. Takie podejście

pozwała na uporządkowanie pracy nad projektem i większą przejrzystość w procesie rozwoju modułów.

GitHub pozwala również na synchronizację ze wspomnianymi Heroku i Netlify, co pozwala na automatyzację procesu umieszczania nowych wersji aplikacji w środowisku internetowym. Wraz z umieszczeniem nowej wersji aplikacji w repozytorium, serwisy otrzymują kod źródłowy za pośrednictwem GitHuba, a następnie budują i uruchamiają aplikację.



# Rozdział 5

## Specyfikacja zewnętrzna

Rozdział zawiera szczegółowy opis funkcjonalności systemu. Zawarto w nim opis instalacji, uruchomienia i korzystania z systemu. Opis dotyczy pracy systemu w środowisku lokalnym.

### 5.1 Instalacja

Przed rozpoczęciem instalacji poszczególnych modułów systemu należy zainstalować silnik NodeJs oraz środowisko NPM(Node Packet Module). Silnik NodeJs jest niezbędny do uruchomienia poszczególnych serwerów systemu. NPM to menedżer pakietów, który dostarcza mechanizmy pozwalające uruchamiać odpowiednie skrypty, które pozwalają na konfigurację serwera.

#### 5.1.1 Instalacja dla Windows i MacOS

Dla systemów operacyjnych Windows i MacOS pakiet instalacyjny znajduje się na stronie NodeJs - <https://nodejs.org/en/download/>. Na stronie należy wybrać opcję *Windows Installer* lub *macOs Installer*, a pobieranie rozpocznie się automatycznie. Pobrany instalator przeprowadzi również instalację NPM.

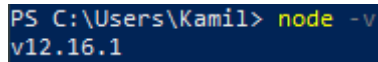
#### 5.1.2 Instalacja dla Linux

Instalacja wymaganego oprogramowania dla systemu Linux może zostać przeprowadzona z wykorzystaniem polecenia *sudo*. W terminalu należy wpisać następujące polecenia:

```
sudo apt install nodejs
```

```
sudo apt install npm
```

W celu sprawdzenia poprawności instalacji dla każdego systemu można wykonać terminal (lub konsolę w przypadku Windows). W terminalu należy wpisać: *node -v*, co powinno skutkować wyświetleniem aktualnie zainstalowanej wersji NodeJs. Tak jak na rysunku 5.1.

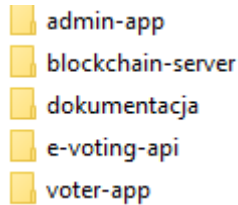


```
PS C:\Users\Kamil> node -v
v12.16.1
```

Rysunek 5.1: Rezultat polecenia `node -v`

## 5.2 Struktura oprogramowania

System składa się z kilku modułów, dlatego dostarczone oprogramowanie jest podzielone na kilka folderów. Struktura folderów została przedstawiona na rysunku 5.2.



Rysunek 5.2: Struktura folderów

Foldery przechowują następujące moduły:

- dokumentacja przechowuje dokumentację projektu,
- admin-app przechowuje aplikację przeglądarkową administratora,
- voter-app przechowuje aplikację przeglądarkową użytkownika,
- e-voting-api przechowuje aplikację serwerową dostarczającą API systemu,
- blockchain-server przechowuje program, który realizuje działanie węzła w sieci Blockchain.

## 5.3 Uruchamianie modułów

Za pomocą menadżera pakietów NPM uruchamiane są poszczególne moduły systemu. Aby uruchomić dany moduł należy otworzyć katalog z tym modułem. W katalogu znajduje się plik *package.json*, który zawiera konfigurację i skrypty uruchamiające program. Programy uruchamia się za pomocą poleceń dostarczanych przez pakiet NPM, które należy wpisać w konsoli. Konsola musi być uruchomiona w folderze z plikiem *package.json*.

Moduły *admin-app* i *voter-app* uruchamiane są za pomocą polecenia *npm-start*. Aplikacja *admin-app* w środowisku lokalnym uruchamiana jest pod adresem `http://localhost:4000`, a aplikacja *voter-app* pod adresem `http://localhost:3000`.

Podczas uruchamiania aplikacji serwerowych, należy określić port, na którym widoczny będzie określony serwer.

Aplikacja serwerowa *e-voting-api* jest domyślnie uruchamiana na porcie 8080, jednak można to zmienić za pomocą następującego polecenia:



`$env:PORT=NUMER_PORTU; npm start.` (5.1)

Użycie tego polecenia jest obligatoryjne podczas uruchamiania węzła sieci Blockchain, ponieważ węzeł nie uruchamia się na domyślnym porcie. W celu uruchomienia węzła należy wybrać dowolny nieużywany port komputera.

Podczas uruchamiania węzła sieci Blockchain można wybrać algorytm konsensusu, który będzie wykorzystywał dany węzeł. Pełne polecenie uruchamiające węzeł Blockchain wygląda następująco:

`$env:PORT=NUMER_PORTU; $env:CONSENSUS=KOD; npm start.` (5.2)

W polu *KOD* należy wpisać kod algorytmu. Dostępne są dwa kody: *POA* i *POW*. Kod *POA* odpowiada algorytmowi *Proof of Authority*, a kod *POW* algorytmowi *Proof of Work*. W przypadku podania błędnego kodu lub jego braku, domyślnym algorytmem używanym przez węzeł jest *Proof of Authority*.

## 5.4 Aktywacja oprogramowania

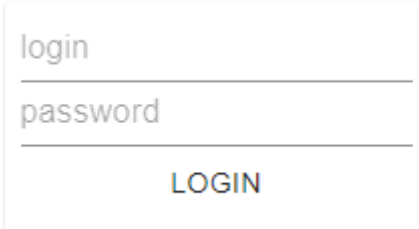
Wraz z oprogramowaniem dostarczono konto administratora. Na konto administratora należy się zalogować wpisując *admin*, w pole *login* i *test*, w pole *password*.

## 5.5 Aplikacja administratora

Aplikacja administratora jest najbardziej rozbudowanym elementem systemu. W tym podrozdziale przedstawiono dostępne scenariusze obsługi tej aplikacji.

### 5.5.1 Logowanie

Wszystkie funkcjonalności administratora dostępne są po zalogowaniu. Dlatego na stronie administratora pierwszym dostępnym widokiem jest formularz logowania (rys. 5.3).



login

password

LOGIN

Rysunek 5.3: Formularz logowania administratora.

Po zalogowaniu ukazuje się strona administratora. Nawigacja po stronie administratora odbywa się za pomocą górnego paska nawigacji (rys. 5.4). Z paska nawigacji można przejść do listy wyborców (*Voters*), listy wyborów (*Elections*)

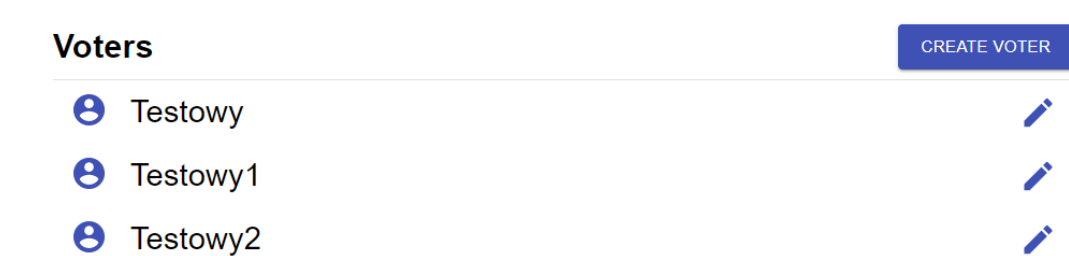
i panelu konfiguracji sieci Blockchain (*Servers*). Na pasku nawigacji znajduje się również przycisk służący do wylogowania administratora z aplikacji (*Logout*).



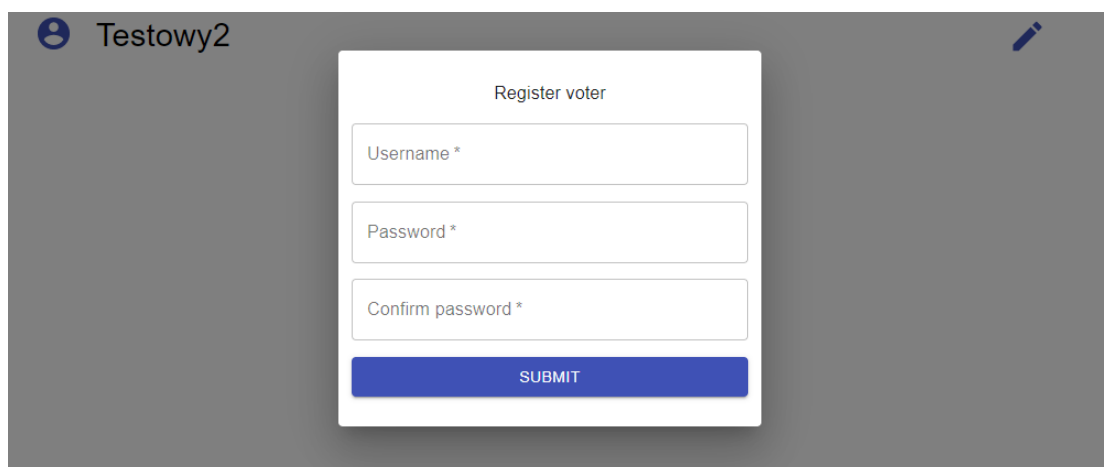
Rysunek 5.4: Pasek nawigacji administratora.

### 5.5.2 Lista wyborców

Lista wyborców dostarcza informacje o wyborcach utworzonych w systemie. Cały widok przedstawia rysunek 5.5. Przycisk *Create Voter* pozwala na dodanie nowego wyborcy do systemu. Po kliknięciu tego przycisku pojawia się okno 5.6, które zawiera formularz dodawania nowego wyborcy. Do formularza należy wpisać dane nowego wyborcy.



Rysunek 5.5: Lista wyborców.



Rysunek 5.6: Okno dodawania wyborcy.

### 5.5.3 Konfiguracja sieci Blockchain

Administrator posiada interfejs pozwalający na tworzenie sieci Blockchain. Węzły Blockchain to oprogramowanie działające na pojedynczym serwerze. Za pomocą konfiguratora, administrator systemu może połączyć się z danym węzłem i połączyć z nim inne serwery. Interfejs jest dostępny po kliknięciu na pasku nawigacji opcji *Servers*. Rysunek 5.7 przedstawia wyszukiwarkę węzłów Blockchain. W pole *Server url* należy wpisać adres serwera, na którym pracuje węzeł sieci Blockchain.



Rysunek 5.7: Wyszukiwarka węzłów.

Pomyślne połączenie sygnalizowane jest zmianą koloru przycisku *check*, tak jak na rysunku 5.8. Na rysunku widać również zakładkę *Peers*, która wyświetla wszystkie połączone węzły. Po podłączeniu do węzła administrator uzyskuje dostęp, do możliwości połączenia ze sobą dwóch węzłów. Do tego służy pole z etykietą *Server url*, znajdujące się pod zakładką *Peers*.



Rysunek 5.8: Konfiguracja podłączonego węzła.

W taki sposób można łączyć węzły i tworzyć sieci Blockchain. Bardzo ważne jest, aby każdy węzeł był połączony z węzłem głównym, który komunikuje się z API systemu.

### 5.5.4 Lista wyborów

Widok prezentujący listę wyborów pozwala na uzyskanie dostępu do menedżera wyborów, opcji tworzenia nowych wyborów i usunięcia wyborów. Rysunek 5.9 przedstawia omawiany widok. Ikona kosza pozwala na usunięcie wyborów, jeżeli wyniki wyborów są opublikowane. Ikona długopisu pozwala na zarządzanie wyborami. Przycisk *Create election* przenosi użytkownika do konfiguratora wyborów, który pozwala na tworzenie nowych wyborów.

# Elections

[CREATE ELECTION](#)

## Wybory prezydenckie 2077



Rysunek 5.9: Widok listy wyborów.

### 5.5.5 Konfiguracja wyborów

Konfiguracja wyborów to dosyć długi proces, dlatego został on podzielony na kilka etapów. Widok na rysunku 5.10 przedstawia pierwszy etap konfiguracji. W tym etapie administrator musi wypełnić formularz. W formularzu należy wpisać tytuł wyborów, opis, datę rozpoczęcia, datę zakończenia i adres głównego serwera sieci Blockchain, który jest punktem dostępu do całej sieci.

1 Provide elections informations. 2 Add candidates. 3 Add voters.

Elections title \*  
Wybory prezydenckie 2077

Description \*  
Wybory na prezydenta 2077.

Start date \*  
20.12.2020

End date \*  
24.12.2020

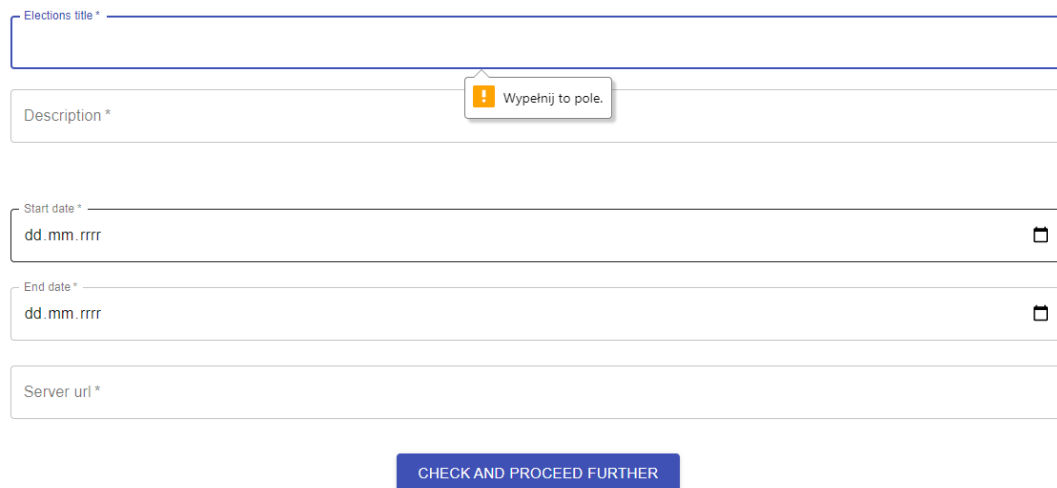
Server url \*  
http://localhost:3001

[CHECK AND PROCEED FURTHER](#)

Rysunek 5.10: Konfiguracja wyborów etap pierwszy.

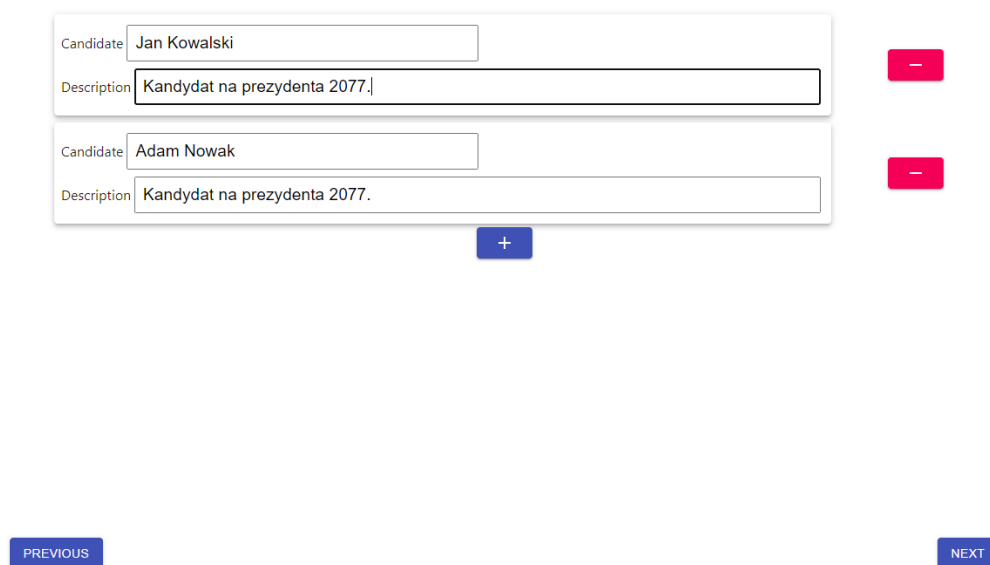
Po wypełnieniu formularza należy kliknąć przycisk *Check and proceed further*. Aplikacja sprawdzi poprawność formularza i poinformuje o błędach (rys. 5.11) lub wyświetli następny widok.

Następnym widokiem jest formularz pozwalający na utworzenie kandydatów. Widok został przedstawiony na rysunku *votescandidate*. Formularz jest zbudowany z dwóch pól tekstowych *Candidate* i *Description*, w których należy umieścić nazwę kandydata i jego opis. Dodatkowo w skład formularza wchodzi przyciski oznaczone plusem i minusem. Minus usuwa okno, a plus dodaje nowe okno do formularza. Przejście do następnego widoku następuje po kliknięciu przycisku *Next*. Dodatkowo można powrócić do widoku pierwszego za pomocą przycisku *Previous*.



The screenshot shows a form with several input fields. The first field, 'Elections title \*', is empty and has a blue border. The second field, 'Description \*', contains the text 'Wypełnij to pole.' and has a red border. The third field, 'Start date \*', contains 'dd.mm.rrrr' and has a red border. The fourth field, 'End date \*', contains 'dd.mm.rrrr' and has a red border. The fifth field, 'Server url \*', is empty and has a blue border. A blue button labeled 'CHECK AND PROCEED FURTHER' is at the bottom.

Rysunek 5.11: Wyświetlanie błędów formularza.



The screenshot shows a form for adding candidates. It has two main sections. The first section has a 'Candidate' field with 'Jan Kowalski' and a 'Description' field with 'Kandydat na prezydenta 2077.'. The second section has a 'Candidate' field with 'Adam Nowak' and a 'Description' field with 'Kandydat na prezydenta 2077.'. There are red minus buttons to the right of each section and a blue plus button at the bottom. At the bottom of the form are 'PREVIOUS' and 'NEXT' buttons.

Rysunek 5.12: Formularz dodawania kandydatów. Etap drugi

Następnym i ostatnim krokiem jest przypisanie wyborców do wyborów. Rysunek 5.13 przedstawia formularz dodawania wyborców. Wyborców dodaje się poprzez zaznaczenie ich na liście. Żeby zakończyć proces konfiguracji wyborów należy kliknąć przycisk *Submit*.

|          |                                     |
|----------|-------------------------------------|
| Testowy  | <input checked="" type="checkbox"/> |
| Testowy1 | <input checked="" type="checkbox"/> |
| Testowy2 | <input checked="" type="checkbox"/> |
| Testowy3 | <input checked="" type="checkbox"/> |

PREVIOUS

SUBMIT

Rysunek 5.13: Lista dostępnych wyborów.

Po utworzeniu wyborów następuje przeniesienie do widoku z listą wyborów (rys. 5.14).

## Elections

CREATE ELECTION

Wybory prezydenckie 2077



Rysunek 5.14: Lista dostępnych wyborów.

### 5.5.6 Nadzorowanie wyborów

Żeby przejść do panelu nadzorowania wyborów, z poziomu widoku listy wyborów (rys. *electionsview*) należy kliknąć ikonę edycji (rys. 5.15).

Po kliknięciu przycisku ukazuje się menu zarządzania wyborami (rys. 5.16). Na górze menu widać rozszerzenie paska nawigacji, które pozwala na przełączanie się pomiędzy funkcjami menadżera wyborów. Widokiem, który ukazuje się jako



Rysunek 5.15: Ikona edycji.

pierwszy jest komponent o tytule *Status*. Ten komponent wyświetla tytuł i opis wyborów, a także adres głównego węzła sieci Blockchain odpowiedzialnego za dane wybory. Na rysunku 5.16 widać przycisk *Change Url*, którego kliknięcie aktywuje pole tekstowe z etykietą *Server Url* (rys. 5.17). W pole tekstowe należy wpisać adres nowego serwera sieci Blockchain, który ma być traktowany jako węzeł główny. Zmiana serwera następuje po kliknięciu przycisku *Submit*. Pozwala to na zmianę serwera, jeżeli używany serwer uległ awarii.

STATUS VOTERS CANDIDATES BLOCKCHAIN RESULTS

Wybory prezydenckie 2077

Wybory na prezydenta 2077.

Server url \*  
http://localhost:3001

CHANGE URL

Rysunek 5.16: Komponent Status.

Wybory na prezydenta 2077.

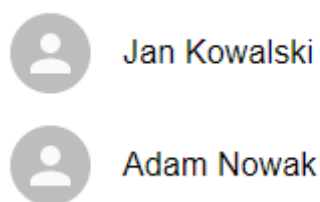
Server url \*  
http://localhost:3003

SUBMIT

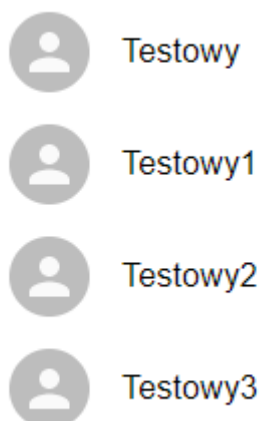
CHANGE URL

Rysunek 5.17: Zmiana adresu serwera.

Kolejne komponenty, które można wyświetlić poprzez pasek nawigacji (rys. 5.16) to *Voters* i *Candidates*. *Voters* prezentuje listę wyborców przypisanych do głosowania, a *Candidates* listę kandydatów. Są to komponenty informacyjne, które przedstawiono na rysunkach 5.19 i 5.18.



Rysunek 5.18: Lista kandydatów.



Rysunek 5.19: Lista wyborców.

Komponent Blockchain (rys. 5.20) zawiera informacje dotyczące liczby stworzonych bloków (*Blocks created*) i węzłów (*Nodes*) w sieci Blockchain.

**Blockchain network status**

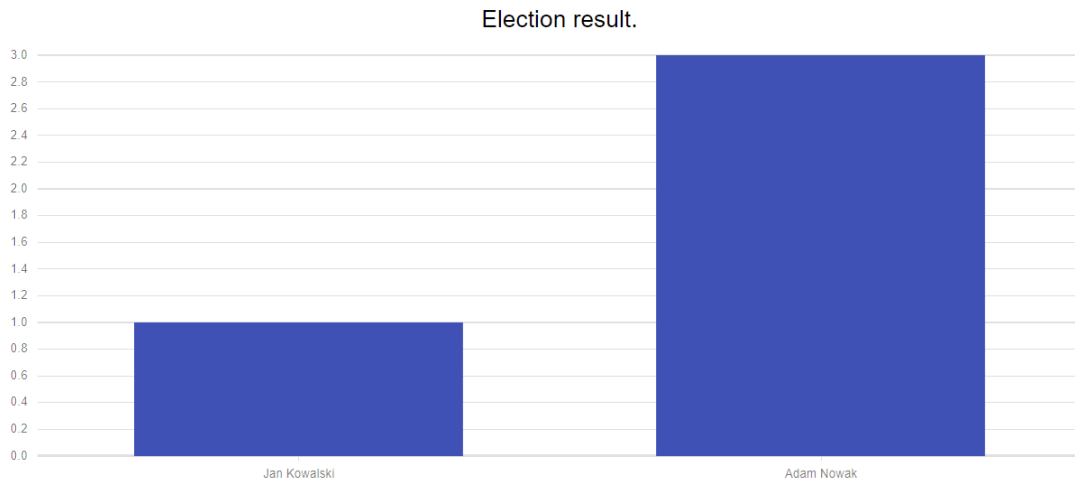
Rysunek 5.20: Dane dotyczące sieci Blockchain.

Ostatnim komponentem menadżera wyborów jest Results . Ten komponent od- powiada za publikację wyników wyborów. Wyniki wyborów są tajne, dla każdego aktora systemu, do momentu ich opublikowania. Po otwarciu komponentu, jeżeli wyniki wyborów nie zostały opublikowane wyświetlony zostaje przycisk 5.21. Po kliknięciu tego przycisku wyniki wyborów zostaną opublikowane i pobierane z serwera. Wyświetlany jest wykres słupkowy prezentujący wyniki wyborów (rys. 5.22).





Rysunek 5.21: Przycisk publikowania wyborów.



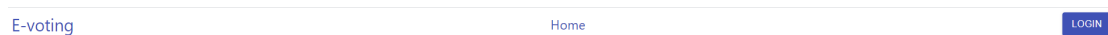
Rysunek 5.22: Diagram słupkowy prezentujący wyniki.

## 5.6 Aplikacja użytkownika

Aplikacja użytkownika dostarcza interfejs pozwalający na wysłanie głosu i odczytanie wyników wyborów. Ten podrozdział przedstawia scenariusze obsługi aplikacji wyborcy.

### 5.6.1 Logowanie

Dostęp do zasobów wyborcy wymaga zalogowania. W celu zalogowania się należy kliknąć przycisk *login*, który znajduje w prawym górnym rogu, na pasku nawigacji (rysunek 5.23).



Rysunek 5.23: Pasek nawigacji.

Po kliknięciu przycisku *login* na ekranie pojawia się okno, które wymaga podania loginu i hasła wyborcy. Okno przedstawiono na rysunku 5.24. Jeżeli dane logowania są poprawne, to następuje przeniesienie do strony wyborcy (rys. 5.26).

Jeżeli użytkownik poda błędne dane logowania, to zostanie o tym poinformowany poprzez wyświetlenie stosownego komunikatu w oknie logowania (rys. 5.25).

1

Login

Login  
Testowy

Password  
....

LOGIN

Rysunek 5.24: Okno logowania.

Wylogowanie wymaga kliknięcia przycisku *Logout* w prawym górnym rogu strony wyborcy (rys. 5.26).

### 5.6.2 Instrukcja obsługi

Instrukcja obsługi, to krótki opis wykonywania podstawowych akcji na stronie użytkownika. Można do niej przejść klikając *help* na pasku nawigacji (rys. 5.26). Informacje zawarte w instrukcji dotyczą: wyboru głosowania, procesu głosowania i wyników wyborów.

E-voting Elections Help LOGOUT

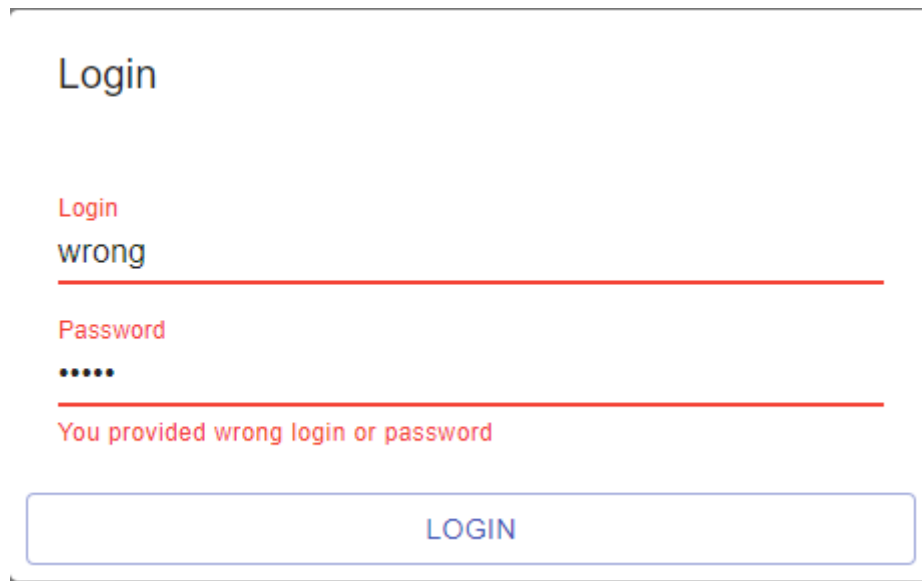
Voting manual.

1. Elections  
To open elections menu click 'Elections' on the top navigation panel.  
After opening elections menu you will see all elections.  
If the elections menu is empty, you are not participating in any elections.

2. Vote  
To proceed to voting panel, click vote button at the election item.  
It will open new panel. This is voting panel. You can go back to elections menu by clicking 'go back' button.  
Voting panel contains all candidates from selected voting. You can select candidate by clicking checkbox. If you want to send your vote, click 'send vote' button.

5. Results  
If the voting is finished you can see its results. To check results click the 'results' button.  
It will show you a diagram with results. You can go back by clicking 'go back' button beneath the diagram.

Rysunek 5.26: Strona wyborcy. Instrukcja obsługi.



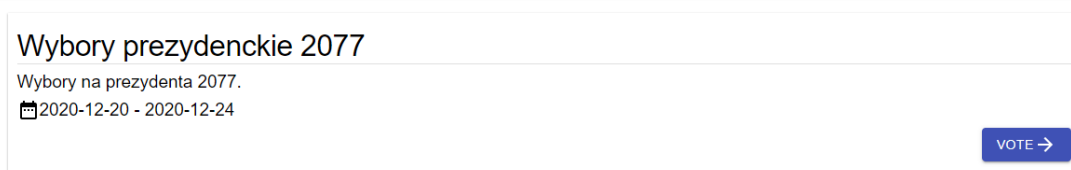
The screenshot shows a login form with the title "Login". It contains two input fields: "Login" and "Password". The "Login" field has the text "wrong" below it, and the "Password" field has "....." below it. A red horizontal line spans the width of the form below the password field. Below this line, the error message "You provided wrong login or password" is displayed in red. At the bottom of the form is a blue button labeled "LOGIN".

Rysunek 5.25: Obsługa błędnych danych logowania.

### 5.6.3 Głosowanie

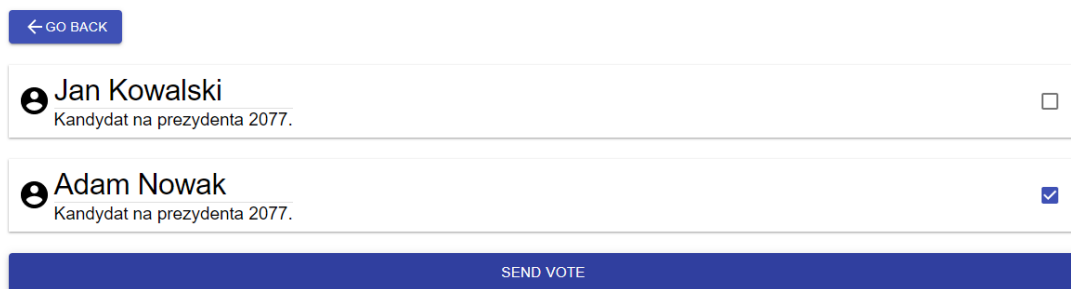
Głosowanie dostępne jest po kliknięciu *Elections* na pasku nawigacji. Po wybraniu tej opcji z paska nawigacji następuje przekierowanie do strony, której zawartość przedstawiona jest na rysunku 5.27. Jeżeli wyborca bierze udział w większej liczbie wyborów, to będzie miał do wyboru więcej opcji, niż na rysunku 5.27.

Okno przedstawione na rysunku 5.27 zawiera przycisk oznaczony *Vote*. Kliknięcie tego przycisku przenosi użytkownika do głównego panelu wyborów 5.28. W tym miejscu wyborca dokonuje wyboru kandydata i przesyła swój głos, po kliknięciu przycisku *Send Vote*. Wyborca może cofnąć widok, do poprzedniego widoku, za pomocą przycisku *Go Back*.



The screenshot shows a voting interface titled "Wybory prezydenckie 2077". Below the title, it says "Wybory na prezydenta 2077." and "2020-12-20 - 2020-12-24". At the bottom right, there is a blue button labeled "VOTE →".

Rysunek 5.27: Okno wyborcze.



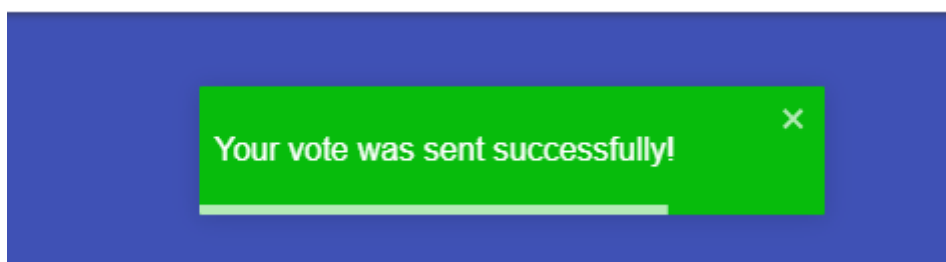
Panel wyborów. W górnej części znajduje się przycisk "← GO BACK". Poniżej lista kandydatów na prezydenta 2077. Każdy kandydat ma ikonę osoby, nazwę i tytuł. Przy każdym kandydacie znajduje się pole wyboru. W przypadku Adam Nowak pole wyboru jest zaznaczone. Na dole znajduje się przycisk "SEND VOTE".

| Kandydat                                     | Status                              |
|--|-------------------------------------|
| Jan Kowalski<br>Kandydat na prezydenta 2077. | <input type="checkbox"/>            |
| Adam Nowak<br>Kandydat na prezydenta 2077.   | <input checked="" type="checkbox"/> |

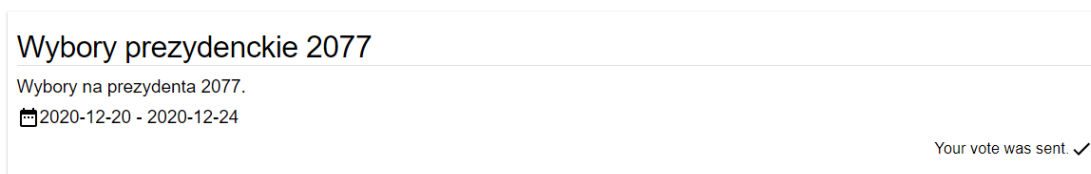
SEND VOTE

Rysunek 5.28: Panel wyborów.

Po wysłaniu głosu na ekranie pojawia się powiadomienie 5.29 i następuje przeniesienie do widoku 5.30.



Rysunek 5.29: Pomyślnie wysłany głos.

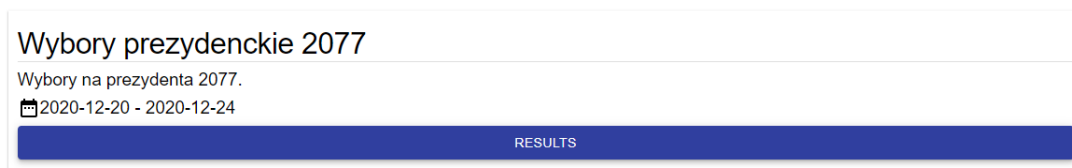


Okno wyborów po zagłosowaniu. Tytuł "Wybory prezydenckie 2077". Podtytuł "Wybory na prezydenta 2077.". Data "2020-12-20 - 2020-12-24". W prawym dolnym rogu napis "Your vote was sent. ✓".

Rysunek 5.30: Okno wyborów. Po zagłosowaniu.

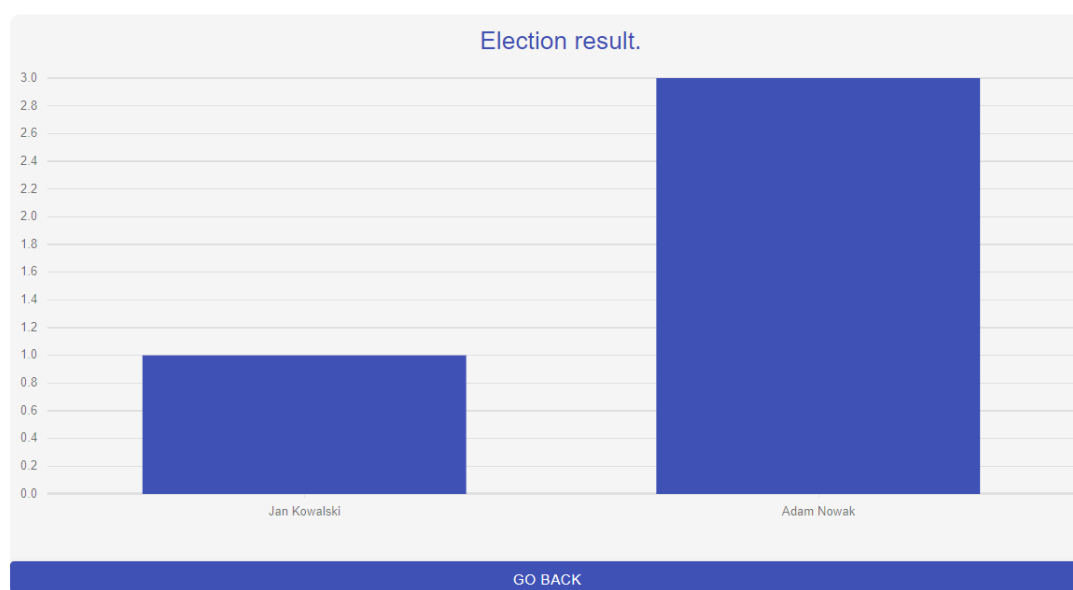
#### 5.6.4 Wyniki

Dostęp do wyników wyborów uzyskiwany jest po ogłoszeniu wyników. Wybory, których wyniki są dostępne oznaczone są tak jak na rysunku 5.31. Okno zawiera przycisk *results*, którego kliknięcie przenosi użytkownika do okna 5.32. Wykres jest podobny do wykresu administratora (rys. 5.22). Wrócić do poprzedniego widoku można za pomocą przycisku pod wykresem *Go Back*.



Okno wyborów po ogłoszeniu wyników. Tytuł "Wybory prezydenckie 2077". Podtytuł "Wybory na prezydenta 2077.". Data "2020-12-20 - 2020-12-24". Na dole znajduje się przycisk "RESULTS".

Rysunek 5.31: Okno wyborów. Po ogłoszeniu wyników.



Rysunek 5.32: Wykres wyników wyborów.



# Rozdział 6

## Specyfikacja wewnętrzna

W tym rozdziale umieszczono wszystkie informacje dotyczące architektury systemu, wykorzystanych struktur danych, algorytmów i wykorzystania wzorców projektowych.

### 6.1 Architektura systemu

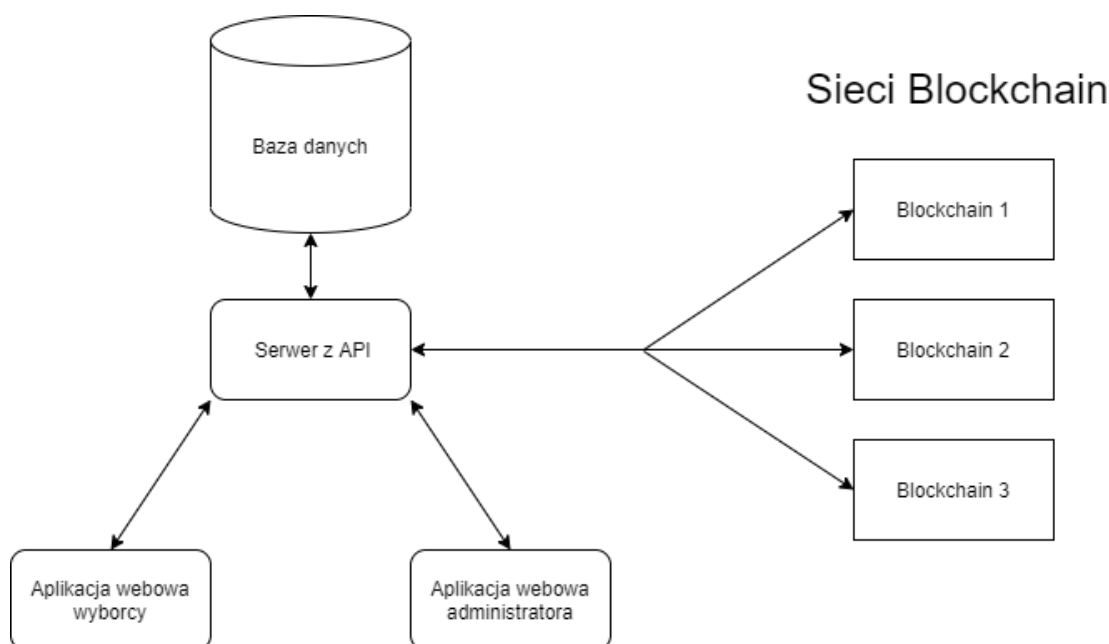
Podrozdział architektura systemu zawiera przegląd komponentów systemu, także nakreśla idee projektu. Podrozdział podzielono na dwie części. Pierwsza część dotyczy całego systemu, natomiast druga opisuje charakterystykę sieci Blockchain.

#### 6.1.1 Architektura pełnego systemu

System składa się z pięciu głównych komponentów:

- aplikacja wyborcy - pozwala na dostęp do zasobów wyborcy,
- aplikacja administratora - umożliwia konfigurację wyborów, audytowanie wyborów, zarządzanie sieciami Blockchain i zarządzanie bazą danych,
- serwer z API - serwer, na którym uruchomione jest API. API dostarcza odpowiednie zasoby określonym użytkownikom i posiada dostęp do bazy danych,
- baza danych - przechowuje informacje dotyczące konfiguracji wyborów. Listy użytkowników, wyborów, kandydatów i określa relacje pomiędzy nimi. W bazie danych znajdują się również hasła użytkowników,
- sieć Blockchain - odpowiada za przechowywanie głosów i udostępnianie ich na życzenie administratora. Blockchain pełni również funkcje zabezpieczenia głosów.

Komponenty są przedstawione na rysunku 6.1.



Rysunek 6.1: Schemat przedstawiający komponenty systemu.

Taki podział systemu umożliwia wykorzystanie sieci Blockchain. Proces przeprowadzania wyborów jest scentralizowany, a sieć Blockchain, która odpowiada za zapisanie wyników wyborów jest rozproszona.

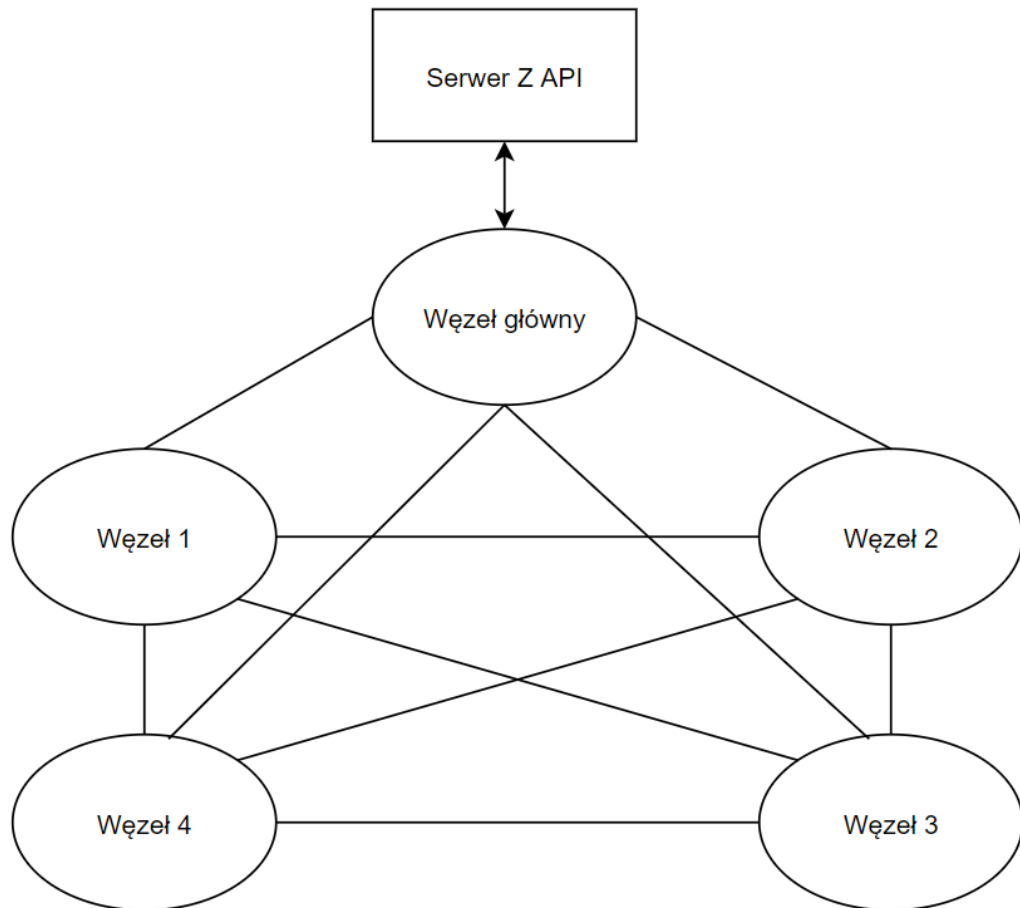
### 6.1.2 Architektura sieci Blockchain

Sieć Blockchain to grupa serwerów, które wymieniają się informacjami. Komunikacja głównie odbywa się w celu przesłania zawartości nowego bloku, synchronizacji danych i dodania nowego węzła do sieci.

Rysunek 6.2 przedstawia sieć Blockchain złożoną z pięciu węzłów. Wszystkie węzły połączone są ze sobą. W sieci można wyróżnić węzeł główny, którego zadaniem jest utrzymanie połączenia z REST API systemu wyborczego.

Drogą komunikacyjną pomiędzy REST API, a węzłem głównym przesyłane są głosy i żądania dotyczące wyniku wyborów oraz statusu sieci.





Rysunek 6.2: Schemat przedstawiający przykładową sieć Blockchain.

## 6.2 Struktury danych

W systemie można wyróżnić dwie struktury danych, które w znaczny sposób wpływają na zachowanie systemu. W tym podrozdziale omówiono implementację i zastosowania tych struktur danych.

### 6.2.1 Łańcuch bloków

Za dostarczenie odpowiedniej struktury łańcucha bloków odpowiedzialne są klasy *Block* i *ChainRepository*. Listing 6.3 przedstawia klasę *Block*, która jest reprezentacją pojedynczego bloku. Każdy blok jest odpowiedzialny za przechowywanie głosu wyborczego. Głos jest przechowywany w polu *data*. Pole *timestamp* przechowuje datę utworzenia bloku w łańcuchu. W łańcuchu bloków, każdy blok posiada odniesienie do *hash'u* poprzedniego bloku. W przedstawionej implementacji zajmuje się tym pole *prevHash*. Klasa *Block* posiada również pole *nonce*, którego zastosowanie dotyczy algorytmu *Proof of Work*.

```
1 class Block {  
2   index: number  
3   timestamp: string  
4   nonce: number  
5   prevHash: string  
6   data: string  
7  
8   constructor (index: number,  
9     timestamp: string,  
10    nonce: number,  
11    prevHash: string,  
12    data: string) {  
13     this.index = index  
14     this.timestamp = timestamp  
15     this.nonce = nonce  
16     this.prevHash = prevHash  
17     this.data = data  
18   }  
19 }
```

Rysunek 6.3: Klasa *Block*.

Klasa *ChainRepository* (przedstawiona w listingu 6.4) odpowiada za przechowywanie łańcucha bloków i jego udostępnianie. Aby zapewnić spójność danych, ułatwienie dostępu i ograniczenie zużycia pamięci, w programie istnieje tylko jedna instancja tej klasy. W tym celu wykorzystano wzorzec projektowy Singleton. Z punktu widzenia łańcucha bloków istotne jest pole *chain*. Jest to tablica bloków. Klasa udostępnia metodę *addBlock*, która pozwala na dodawanie nowego bloku na początek tablicy, zgodnie z regułą tworzenia łańcucha bloków. Dodatkowo klasa posiada prywatną metodę *createGenesisBlock*, która tworzy pierwszy blok łańcucha, którego zawartość nie przechowuje istotnych danych. Metoda wywoływana jest podczas tworzenia instancji *ChainRepository*.

```
1 class ChainRepository {
2     public static instance: ChainRepository
3     private chain: Block[]
4
5     private constructor () {
6         this.chain = new Array<Block>()
7         this.createGenesisBlock()
8     }
9
10    public static getInstance (): ChainRepository {
11        if (!ChainRepository.instance) {
12            ChainRepository.instance = new ChainRepository()
13        }
14        return ChainRepository.instance
15    }
16
17    public addBlock (block: Block) {
18        this.chain.push(block)
19    }
20
21    private createGenesisBlock () {
22        const block
23        = new Block(this.chain.length, '0', 0, '0', 'genesis')
24
25        this.chain.push(block)
26    }
27    /* ... Getters and setters */
28 }
```

Rysunek 6.4: Klasa *ChainRepository*.

Za zapewnienie poprawnego utworzenia nowego bloku odpowiedzialna jest metoda *createNewBlock* (listing 6.5), która należy do klasy *Blockchain*. Metoda przyjmuje następujące parametry: *nonce*, *prevBlock* i *data*. Z parametru *prevBlock* wyliczany jest *hash*, za pomocą metody *getHash*. Jest to *hash* poprzedniego bloku, który przypisywany jest do pola *prevHash* tworzonego bloku.

```
1 abstract class Blockchain {
2   protected blockchain: ChainRepository
3
4   public createNewBlock (nonce: number, prevBlock: Block,
5     data: string): Block {
6     const blockchain = this.blockchain.getChain()
7     const chainLength = blockchain.length
8     const prevHash = this.getHash(prevBlock)
9     const date = Date.now().toString()
10
11     const block = new Block(chainLength, date, nonce,
12       prevHash, data)
13
14     return block
15   }
16
17   protected getHash (block: Block): string {
18     const inputWord = block.toString()
19     const hash = SHA256(inputWord).toString()
20     return hash
21   }
22   /* ... */
23 }
```

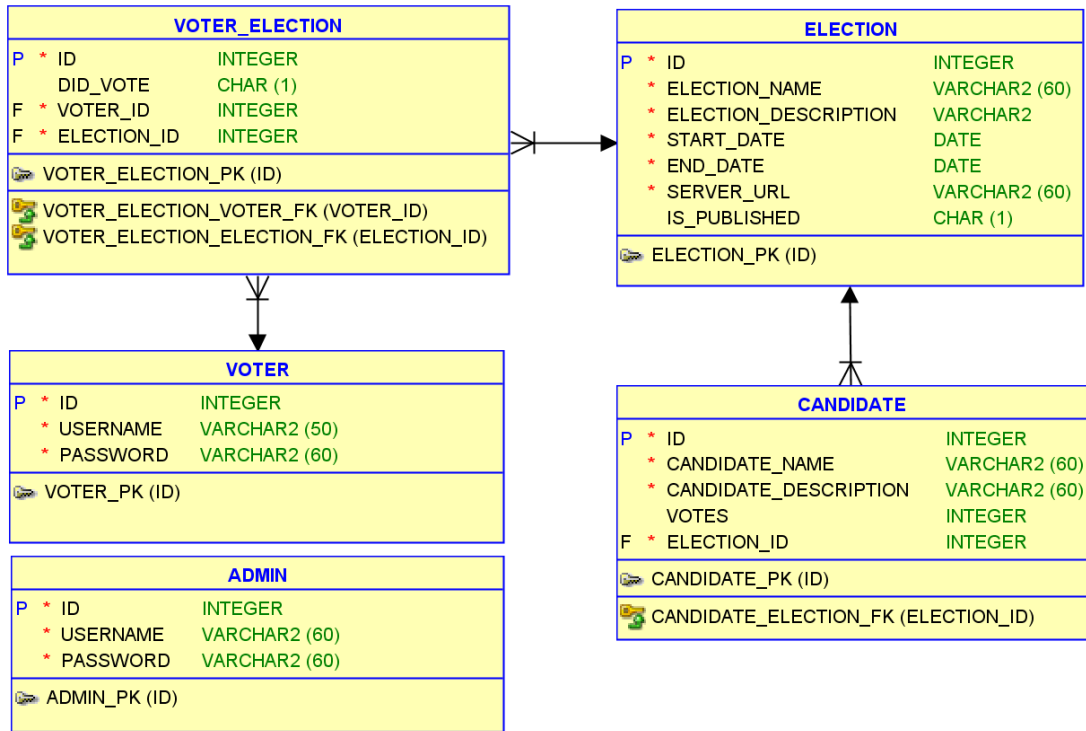
Rysunek 6.5: Fragment klasy Blockchain.

Tak skonstruowany blok jest gotowy do dodania do łańcucha. Dodawanie bloku jest realizowane za pomocą metody *addBlock*, klasy *ChainRepository*.

### 6.2.2 Baza danych

W projekcie wykorzystano relacyjną bazę danych. Taka struktura danych pozwala na uporządkowanie danych, w relacje pomiędzy tabelami. Rysunek 6.6 przedstawia schemat relacyjnego modelu wykorzystanej bazy danych.

Bazę danych można podzielić na część odpowiadającą za przechowywanie konfiguracji wyborów oraz część przechowującą hasła i role użytkowników. Tabele *ADMIN* i *VOTER* przechowują dane logowania aktorów systemu. Na podstawie tych tabel można dokonać autoryzacji użytkownika systemu i przydzielić mu odpowiednią rolę, która zapewnia dostęp do zasobów. Dodatkowo tabela *VOTER* jest połączona relacją wiele do wielu z tabelą *ELECTION*. W wyniku relacji wiele do wielu powstała tabela łącznikowa *VOTER\_ELECTION*, która pozwala podzielić relację wiele do wielu, na dwie relacje jeden do wielu. Tabela *VOTER\_ELECTION* pozwala na przypisanie wyborcy do odpowiednich wyborów. Dodatkowo tabela przechowuje informację o oddaniu głosu przez wyborcę, w polu *DID\_VOTE*.



Rysunek 6.6: Schemat przedstawiający relacyjny schemat bazy danych.

Pozostała część tabel odpowiada za przechowywanie danych dotyczących bezpośrednio wyborów. Zadaniem tabeli *ELECTION* jest składowanie danych dotyczących konfiguracji wyborów.

Konfiguracji mogą podlegać:

- *ELECTION\_NAME* - tytuł wyborów,
- *ELECTION\_DESCRIPTION* - opis dotyczący wyborów,
- *START\_DATE* - data rozpoczęcia wyborów,
- *END\_DATE* - data zakończenia wyborów,
- *SERVER\_URL* - adres zewnętrznego serwera sieci Blockchain, na który wysłane są głosy wyborców,
- kandydaci - zbiór kandydatów, którzy biorą udział w wyborach, jest reprezentowany jako relacja jeden do wielu z tabelą *CANDIDATE*.

Dodatkowo w tabeli znajduje się również pole *IS\_PUBLISHED* typu *Boolean*, które odpowiada za oznaczenie stanu wyborów jako opublikowane lub nieopublikowane. System wykorzystuje tę informację do zakończenia wyborów i udostępnienia ich wyników.

Tabela *CANDIDATE* reprezentuje kandydata, którego dane i opis definiują pola *CANDIDATE\_NAME* i *CANDIDATE\_DESCRIPTION*. Tabela posiada również pole *VOTES*, które po zakończeniu wyborów uzupełniane jest o liczbę zdobytych głosów, przez danego kandydata.

## 6.3 Algorytmy

W podrozdziale Algorytmy opisano implementację algorytmów konsensusu, w sieci Blockchain. Węzeł sieci Blockchain może pracować w dwóch algorytmach konsensu: *Proof of Work* lub uproszczonej wersji algorytmu *Proof of Authority*. Każdy węzeł sieci Blockchain pozwala na wybranie algorytmu konsensusu, w którym będzie pracował. Algorytmy różnią się sposobem dodawania nowego bloku i synchronizacją sieci. Wybór algorytmu jest możliwy, poprzez wykorzystanie wzorca projektowego strategia. Implementacja tego wzorca projektowego została opisana w rozdziale 6.4.

### 6.3.1 Proof of Work

Dodawanie nowego bloku w algorytmie *Proof of Work* poprzedzone jest rozwiązaniem zagadki matematycznej. W listingu 6.7 przedstawiono metodę *validateNonce*, która zajmuje się weryfikacją rozwiązania zagadki.

```
1 private validateNonce (  
2     lastNonce: number,  
3     nonce: number,  
4     prevHash: string  
5 ): boolean {  
6     const attempt = `${lastNonce}${nonce}${prevHash}`  
7     const hash = SHA256(attempt).toString()  
8  
9     const isValid = hash.startsWith('000')  
10    return isValid  
11 }
```

Rysunek 6.7: Metoda *validateNonce*.

Rozwiązanie zagadki matematycznej, to znalezienie odpowiedniej liczby (*nonce*), która po dołączeniu do określonego stałego ciągu znaków i *hash'owaniu* go, daje ciąg określonej liczby zer na początku. W przypadku tej implementacji są to trzy zera.

Metoda *solveNonce* (listing 6.8) zajmuje się znalezieniem liczby *nonce*. W pętli wywołana jest metoda *validateNonce*, do momentu rozwiązania zagadki. Wraz z każdą iteracją pętli wykonywana jest inkrementacja zmiennej *nonce*.

```
1 private solveNonce (lastNonce: number, prevHash: string) {  
2     let nonce = 0  
3  
4     while (!this.validateNonce(lastNonce, nonce, prevHash)) {  
5         nonce++  
6     }  
7     return nonce  
8 }
```

Rysunek 6.8: Metoda *solveNonce*.

Dodawanie nowego bloku jest przeprowadzane w metodzie *mine*. Metodę przedstawiono w listingu 6.9.

```
1 public mine (data: any) {  
2     const blockchain = this.blockchain.getChain()  
3     const lastBlock = blockchain[blockchain.length - 1]  
4     const prevHash = this.getHash(lastBlock)  
5     const nonce = this.solveNonce(lastBlock.nonce, prevHash)  
6  
7     const newBlock = this.createNewBlock(nonce, lastBlock, data)  
8     this.blockchain.addBlock(newBlock)  
9 }
```

Rysunek 6.9: Dodawanie nowego bloku. Metoda *mine*.

Przed wywołaniem metody *createNewBlock* wywoływana jest metoda *solveNonce*, która rozwiązuje zagadkę matematyczną. Następnie zwracany jest wynik zagadki i tworzony jest nowy blok. Nowy blok jest dodawany do łańcucha.

Synchronizacja łańcucha odbywa się poprzez porównanie jego długości z innym łańcuchem. Listing 6.10 przedstawia metodę *synchronize*, która przyjmuje jako argument długość porównywanego łańcucha znaków. Następnie porównuje ją z łańcuchem bloków węzła.

```
1 public synchronize (syncValue: number): boolean {  
2     if (this.blockchain.getChain().length < syncValue) {  
3         return true  
4     } else {  
5         return false  
6     }  
7 }
```

Rysunek 6.10: Metoda *synchronize*.

Jeżeli metoda *synchronize* zwróci wartość *true*, to obecny łańcuch jest podmieniany, na łańcuch przesłany do synchronizacji.

### 6.3.2 Proof of Authority

Algorytm *Proof of Authority* pozwala zwiększyć przepustowość sieci Blockchain, w porównaniu z algorytmem *Proof of Work*. Ten algorytm nie wymaga rozwiązywania żadnej zagadki logicznej, dlatego dodawanie nowego bloku sprowadza się tylko do umieszczenia nowego bloku na końcu tablicy.

W projekcie zastosowano bardzo uproszczoną implementację algorytmu *PoA*. W implementacji zrezygnowano z mechanizmu penalizacji węzłów, za nieprawidłowe operacje w sieci. Dlatego należy mieć na uwadze, że węzły wchodzące w skład sieci muszą być dobrze zabezpieczone. Do sieci należy dodawać tylko zaufane węzły.

Proces dodawania nowego bloku przedstawia listing 6.11. Klasa odpowiedzialna za algorytm *Proof of Authority* posiada dodatkowe pole - *authorityFactor*. Jest to współczynnik określający zaufanie danego węzła w sieci. Im wyższy współczynnik, tym wyższe zaufanie. Zaufanie jest zwiększane, wraz z dodawaniem nowych bloków.

```
1 private authorityFactor: number  
2  
3 public mine (data: any) {  
4     const blockchain = this.blockchain.getChain()  
5     const lastBlock = blockchain[blockchain.length - 1]  
6     const newBlock = this.createNewBlock(0, lastBlock, data)  
7     this.blockchain.addBlock(newBlock)  
8     this.authorityFactor++  
9 }
```

Rysunek 6.11: Dodawanie nowego bloku w algorytmie Proof of Authority.

W przypadku algorytmu *Proof of Authority* synchronizacja łańcucha dokony-



wana jest poprzez porównanie wartości *authorityFactor* dwóch węzłów. Listing 6.12 przedstawia ten proces.

```
1 public synchronize (auth: number) {  
2     if (this.authorityFactor < auth) {  
3         return true  
4     } else {  
5         return false  
6     }  
7 }
```

Rysunek 6.12: Synchronizacja w algorytmie Proof of Authority.

Podmiana węzłów odbywa się analogicznie, jak dla algorytmu *Proof of Work*.

## 6.4 Zastosowane wzorce projektowe

W projekcie zastosowano kilka wzorców projektowych dotyczących zarówno programowania obiektowego, jak i podejścia funkcyjnego towarzyszącego ReactJs. W tym podrozdziale przedstawiono przykłady implementacji tych wzorców w projekcie.

### 6.4.1 Singleton

Singleton to najczęściej występujący wzorec projektowy, w projekcie. Jego zadaniem jest bezpośrednio przechowywanie danych lub utrzymywanie połączenia i dostępu do bazy danych. W Listing 6.13 przedstawiono klasę *Database*, która jest Singletonem. Klasa zajmuje się udostępnianiem połączenia z bazą danych, za pośrednictwem obiektu *PrismaClient*. Prywatny konstruktor otwiera połączenie z bazą danych tylko raz, a statyczna metoda *getInstance* zwraca instancję klasy *Database*. Publiczna metoda *getDatabase* zwraca obiekt *PrismaClient*, który umożliwia dostęp do bazy danych.

```
1 class Database {  
2     private static instance: Database;  
3     private db: PrismaClient;  
4  
5     private constructor() {  
6         this.db = new PrismaClient();  
7     }  
8  
9     public static getInstance(): Database {  
10         if (!Database.instance) {  
11             Database.instance = new Database();  
12         }  
13  
14         return Database.instance;  
15     }  
16  
17     public getDatabase(): PrismaClient {  
18         return this.db;  
19     }  
20 }
```

Rysunek 6.13: Klasa *Database*. Singleton.

### 6.4.2 Repozytorium

Repozytorium pozwala na udzielenie dostępu do danych warstwom biznesowym aplikacji, oddzielając jednocześnie implementację modelu dostępu do bazy danych, od reszty warstw. Repozytorium posiada wszelkie metody wyszukiujące i zwracające żądane obiekty lub kolekcje obiektów. Listing 6.14 przedstawia klasę *CandidateRepository*. Jest to jedno z wielu repozytoriów, które dostarcza podstawowe metody służące do wyszukiwania i zapisywania danych. Klasa posiada pole *db*, które jest instancją udzielającą dostępu do bazy danych.

```
1 class CandidateRepository {
2     private db: PrismaClient
3
4     constructor () {
5         this.db = Database.getInstance().getDatabase()
6     }
7
8     public async findAll () { /* Implementacja */ }
9
10    public async findByName (candidateName: string) {
11        /* Implementacja */
12    }
13
14    public async findByElectionId (electionId: number) {
15        /* Implementacja */
16    }
17
18    public async save (candidate: CandidateDTO, election: Election) {
19        /* Implementacja */
20    }
21 }
```

Rysunek 6.14: Klasa *CandidateRepository*.

W projekcie repozytoria mają za zadanie dostarczać obiekty z bazy danych w klasach serwisowych, które zawierają implementację logiki biznesowej.

### 6.4.3 Strategia

Aplikacja serwerowa węzła sieci Blockchain pozwala na wybór jednego z dwóch algorytmów konsensusu. Algorytmy zajmują się rozwiązaniem tego samego problemu, dlatego można je wykorzystać, w ten sam sposób. Wzorec projektowy strategia pozwala na dokonanie takiego podstawienia, bez konieczności zmiany struktury programu. Listing 6.15 przedstawia klasę bazową *Blockchain*. Jest to klasa abstrakcyjna, która posiada implementację metod uniwersalnych dla każdego algorytmu. Metody *mine*, *getSyncValue* i *synchronize* są abstrakcyjne.

```
1 abstract class Blockchain {
2     protected blockchain: ChainRepository
3
4     constructor () {
5         this.blockchain = ChainRepository.getInstance()
6     }
7
8     abstract mine (data: any): void
9     abstract getSyncValue (): number
10    abstract synchronize (syncValue: number): boolean
11
12    public createNewBlock (nonce: number, prevBlock: Block,
13    data: any): Block {
14        const blockchain = this.blockchain.getChain()
15        const chainLength = blockchain.length
16        const prevHash = this.getHash(prevBlock)
17        const date = Date.now().toString()
18
19        const block = new Block(chainLength, date, nonce,
20    prevHash, data)
21
22        return block
23    }
24
25    public setChain (chain: Block[]) {
26        this.blockchain.setChain(chain)
27    }
28
29    public getScore (): Block[] {
30        return this.blockchain.getChain()
31    }
32
33    protected getHash (block: Block): string {
34        const inputWord = block.toString()
35        const hash = SHA256(inputWord).toString()
36        return hash
37    }
38 }
```

Rysunek 6.15: Klasa *Blockchain*.

Implementacja metod zależy od wykorzystywanego algorytmu. Implementacje znajdują się w klasach *PoWAlgorithm* i *PoAAlgorithm*. Klasy rozszerzają klasę *Blockchain*. Szczegółowa implementacja metod abstrakcyjnych została omówiona

w podrozdziale 6.3. Wykorzystanie mechanizmu dziedziczenia pozwala na podstawienie w miejsce klasy *Blockchain* dowolnej klasy potomnej, która posiada zdefiniowaną implementację metod odpowiedzialnych za realizację algorytmu.

#### 6.4.4 Redux

Redux to specjalna architektura wykorzystywana w aplikacjach ReactJs. W projekcie wykorzystano bibliotekę Redux’a, o nazwie Redux-Toolkit, która upraszcza niektóre elementy Redux’a. Architektura pozwala na przeniesienie stanu aplikacji do jednego magazynu, który jest dostępny globalnie. Listingi 6.16 i 6.17 przedstawiają jeden z wycinków magazynu. Wycinek (ang. *slice*) to oznaczony odpowiednim identyfikatorem fragment magazynu. Każdy wycinek przechowuje określony stan (listing 6.16).

```
1 const initialState = {  
2   electionsList: [],  
3   candidatesList: [],  
4   selectedElection: '',  
5   selectedVote: '',  
6   isCandidatesShown: false,  
7   isResultsShown: false,  
8   electionResults: []  
9 }
```

Rysunek 6.16: Stan początkowy wycinka *electionsSlice*.

Wycinek posiada zestaw metod, które pozwalają na modyfikowanie stanu wycinka. W kodzie są to metody obiektu *reducers*, przedstawione w listingu 6.17.

```
1 const electionsSlice = createSlice({
2   name: 'elections',
3   initialState,
4   reducers: {
5     setElectionsList (state, action) {
6       state.electionsList = action.payload
7     },
8     selectElection (state, action) {
9       state.selectedElection = action.payload
10    },
11    setCandidates (state, action) {
12      state.candidatesList = action.payload
13    },
14    setElectionResults (state, action) {
15      state.electionResults = action.payload
16    },
17    showCandidates (state) {
18      state.isCandidatesShown = true
19      state.isResultsShown = false
20    },
21    showElections (state) {
22      state.isCandidatesShown = false
23      state.isResultsShown = false
24    },
25    showResults (state) {
26      state.isCandidatesShown = false
27      state.isResultsShown = true
28    },
29    selectVote (state, action) {
30      state.selectedVote = action.payload
31    }
32  }
33 })
```

Rysunek 6.17: Wycinek *electionsSlice*.

Tak stworzony kontener ułatwia zarządzanie stanem aplikacji, ponieważ pozwala na odseparowanie implementacji stanu, od kodu odpowiedzialnego za użycie stanu. Architektura Redux posiada również specjalny mechanizm do wykonywania zadań asynchronicznych, takich jak np. pobranie danych z API. Takie zadania nazywane są Thunk’ami. Listing 6.18 przedstawia Thunk’a, który pobiera listę wyborów dostępnych dla określonego użytkownika.

```
1 export const fetchElections = (): AppThunk => async dispatch => {
2   try {
3     const username = localStorage.getItem('username')
4     const response = await axios.get(`/elections/${username}`, {
5       headers: {
6         role: localStorage.getItem('role'),
7         authorization: `Bearer ${localStorage.getItem('token')}`
8       }
9     })
10    const electionList = response.data
11
12    dispatch(setElectionsList(electionList))
13  } catch (error) {
14
15  }
16 }
```

Rysunek 6.18: Thunk *fetchElection*.

W Thunk'ach można wykorzystywać metody obiektu *reducers*, a następnie wywoływać Thunk'i w komponentach. Przykład przedstawiono w listingu 6.19. Wywołanie każdego reducer'a zawsze znajduje się wewnątrz funkcji *dispatch*. *Dispatch* odpowiada za wysłanie odpowiedniej metody obiektu *reducers*, do punktu obsługi tych metod.

```
1 function ElectionsPage () {
2   const dispatch = useDispatch()
3   const { electionsList, isCandidatesShown, isResultsShown }
4   = useSelector(
5     (state: RootState) => state.elections
6   )
7
8   useEffect(() => {
9     dispatch(fetchElections())
10  }, [])
11
12  /* ... */
13 }
```

Rysunek 6.19: Wykorzystanie Thunk'a, w komponencie *ElectionsPage*.





# Rozdział 7

## Weryfikacja i walidacja

W rozdziale przedstawiono proces testowania systemu, a także wykryte i naprawione błędy. Testowanie systemu można podzielić na dwa etapy: testowanie aplikacji serwerowych i testowanie aplikacji przeglądarkowych. W ostatniej części rozdziału przedstawiono proces weryfikacji systemu, wraz z pokazaniem jego przykładowego zastosowania.

### 7.1 Testowanie aplikacji serwerowych

Testowanie aplikacji serwerowych odbywało się przez aplikację *Postman*, która została omówiona w rozdziale 4.4. Aplikacja umożliwia wysyłanie żądań HTTP do serwera, co pozwala na utworzenie kontrolowanego środowiska do testowania. W ten sposób można tworzyć scenariusze testowania. W projekcie stworzone scenariusze obejmowały wszystkie funkcjonalności systemu.

Podczas testowania aplikacji serwerowych wykryto błędy wynikające z nieobsłużonego wyjątku w kodzie, które zostały naprawione. Wykryto również dosyć poważne ograniczenie systemu, które wynika z wykorzystania języka Typescript.

Ograniczenie systemu występuje w oprogramowaniu węzła sieci Blockchain. Sieć wykorzystująca algorytm konsensusu typu *Proof of Work* wymaga przeprowadzania wymagających obliczeń przez serwer. Język Typescript jest językiem operującym na jednym wątku. W Typescript kod wykonywany jest w tzw. pętli zdarzeń. Pętla zdarzeń dzieli zdarzenia według ich typu i w określonej kolejności umieszcza je na stosie wywołań. Stos wywołań to główny stos, w którym ostatecznie wykonywane są zdarzenia. Taki sposób wykonywania zadań pozwala na uzyskanie pewnego rodzaju współbieżności dla programu działającego na jednym wątku. Jednakże wykonywanie zadań, które wymagają dużego wykorzystania procesora powoduje zatrzymanie pętli zdarzeń na długi czas na jednym zadaniu. Do momentu zakończenia obliczeń nie zostaną wykonane inne operacje. Oznacza to, że w tym czasie serwer nie jest w stanie odpowiedzieć na przychodzące żądania, co powoduje zablokowanie portów wejścia/wyjścia serwera. Przychodzące żądania są umieszczane na końcu kolejki, która posiada skończoną pojemność. W przypadku przepełnienia kolejki, część żądań nie zostanie obsłużona. Jest to dosyć poważne ograniczenie, którego częściowym rozwiązaniem było zmniejszenie wykorzystania mocy obliczeniowej przez algorytm *Proof of Work*. Dużo lepszym rozwiązaniem

problemu było wprowadzenie możliwości wykorzystania uproszczonego algorytmu *Proof of Authority*, który nie obciąża procesora.

## 7.2 Testowanie aplikacji przeglądarkowych

Aplikacje przeglądarkowe zostały przetestowane, pod kątem działania na różnych przeglądarkach. Wykorzystane przeglądarki to *Google Chrome*, *Mozilla Firefox* i *Microsoft Edge*. Podczas testowania aplikacje działały zgodnie z założeniami. Dodatkowo przetestowano wszystkie elementy, które podlegają interakcji z użytkownikiem. Do tych elementów należą formularze, przyciski i elementy służące do nawigowania widokami aplikacji. Wykryto kilka błędów dotyczących obsługi formularzy, jednakże były to błędy łatwe w naprawie.

Aplikacja wyborcy została przetestowana pod kątem responsywności na *smartfonach*. Aplikacje przetestowano na modelach *Lenovo P2* i *Samsung Galaxy Note 2*. Podczas testowania responsywności nie wykryto błędów.

Zakres testów oprogramowania należy uznać za niepełny, ponieważ podczas testowania nie przeprowadzono testów jednostkowych oprogramowania, które zapewniają duże pokrycie przypadków testowych.

## 7.3 Weryfikacja systemu

Weryfikacja systemu pozwala na przetestowanie funkcji systemu, a także przedstawienie przykładowego zastosowania. Do weryfikacji systemu przygotowano scenariusz, który zakładał konfigurację i przeprowadzenie wyborów na najlepszego piłkarza sezonu 2019 - 2020. W tym celu stworzono konta dla dwudziestu-jeden wyborców, a następnie przystąpiono do konfiguracji sieci Blockchain.

Dodatkowym założeniem scenariusza było uruchomienie węzłów Blockchain na osobnych maszynach wirtualnych, aby przetestować system w środowisku bardziej rozproszonym. Maszyny wirtualne operowały na systemie Linux Ubuntu. Po zainstalowaniu wymaganych składników uruchomiono program odpowiedzialny za węzeł Blockchain, w konfiguracji wykorzystującej algorytm *Proof of Authority*, na każdej maszynie. Program wykorzystywał port *3001*. Do konfiguracji sieci wymagane są adresy IP węzłów. W celu uzyskania adresów wykorzystano polecenie terminalowe *ifconfig*, które dostarcza informacje dotyczące interfejsów sieciowych systemu operacyjnego. Wśród tych informacji znajduje się adres IP karty sieciowej, jest on oznaczony poprzez *inet* (rys. 7.1).

```
kamil@kamil-VirtualBox:~/Desktop/e-voting/blockchain-server$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.55.102 netmask 255.255.255.0 broadcast 192.168.55.255
```

Rysunek 7.1: Rezultat polecenia *ifconfig*.

Następnie zalogowano się na stronę administratora i połączono ze sobą utworzone węzły (rys. 7.2).



THIS BLOCKCHAIN NETWORK

Server url \*  
http://192.168.55.102:3001

CHECK

Peers

http://192.168.55.104:3001

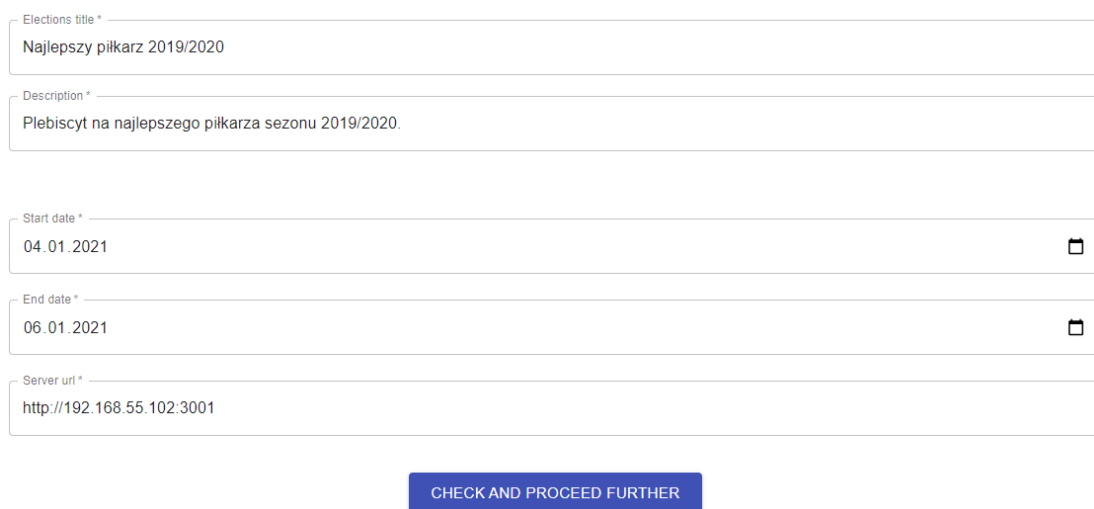
http://192.168.55.107:3001

Server url \*

CONNECT

Rysunek 7.2: Konfiguracja węzłów.

Kolejnym krokiem była konfiguracja wyborów. Rysunek 7.3 przedstawia nadany tytuł, opis, ramy czasowe i adres głównego węzła Blockchain, dla tworzonych wyborów.



Elections title \*  
Najlepszy piłkarz 2019/2020

Description \*  
Plebiscyt na najlepszego piłkarza sezonu 2019/2020.

Start date \*  
04.01.2021

End date \*  
06.01.2021

Server url \*  
http://192.168.55.102:3001

CHECK AND PROCEED FURTHER

Rysunek 7.3: Konfiguracja wyborów.

Następnie dodano sześciu kandydatów na piłkarza sezonu 2019-2020(rys. 7.4).

|             |  |   |
|-------------|--|---|
| Candidate   | Lionel Messi                           | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |
| Candidate   | Robert Lewandowski                     | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |
| Candidate   | Cristiano Ronaldo                      | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |
| Candidate   | Neymar                                 | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |
| Candidate   | Thiago Alcantara                       | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |
| Candidate   | Kylian Mbappe                          | — |
| Description | Kandydat na piłkarza sezonu 2019/2020. |   |

Rysunek 7.4: Dodawanie kandydatów.

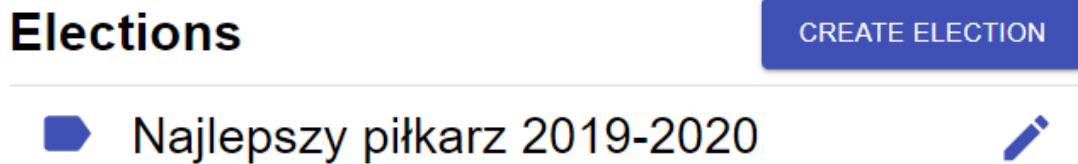
Po dodaniu kandydatów przypisano wyborców do tworzonych wyborów (rys. 7.5).

|                    |                                     |
|--------------------|-------------------------------------|
| Lionel Messi       | <input checked="" type="checkbox"/> |
| Cristiano Ronaldo  | <input checked="" type="checkbox"/> |
| Robert Lewandowski | <input checked="" type="checkbox"/> |
| Jurgen Klopp       | <input checked="" type="checkbox"/> |
| Marcelo Bielsa     | <input checked="" type="checkbox"/> |
| Hans-Dieter Flick  | <input checked="" type="checkbox"/> |
| Kylian Mbappe      | <input checked="" type="checkbox"/> |
| Neymar             | <input checked="" type="checkbox"/> |
| Sergio Ramos       | <input checked="" type="checkbox"/> |

Rysunek 7.5: Przypisanie wyborców.

W przedstawiony sposób utworzono nowe wybory, które są widoczne w menu

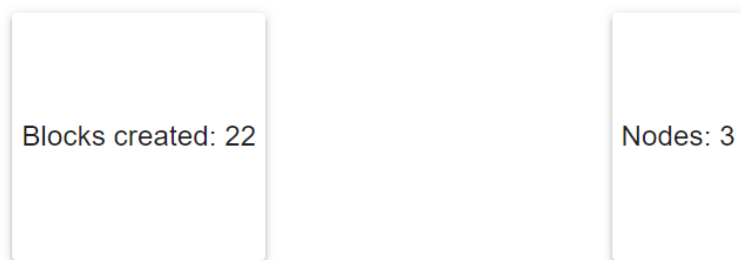
wyborów (rys. 7.6).



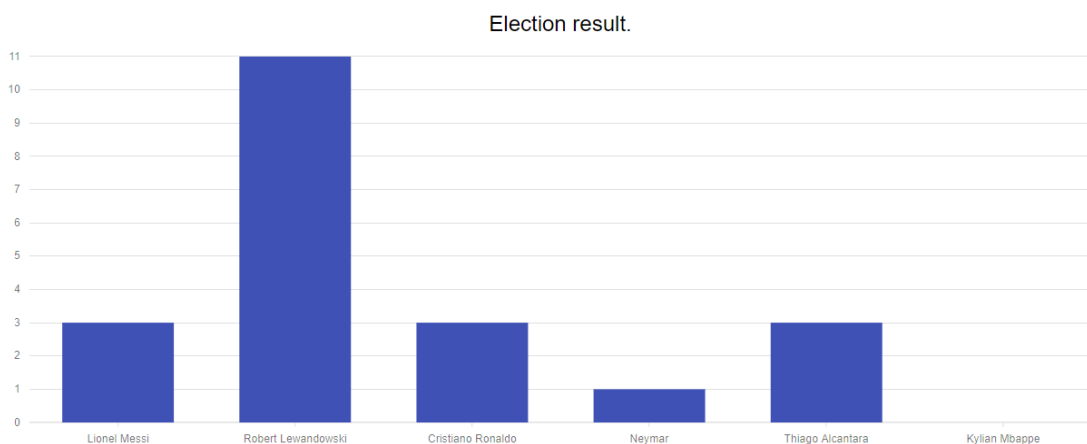
Rysunek 7.6: Menu wyborów.

Po utworzeniu wyborów oddane zostały głosy z utworzonych dwudziestu-jeden kont. System pomyślnie przetworzył wszystkie głosy, o czym świadczy informacja z panelu administratora, dotycząca statusu sieci (rys. 7.7). Sieć składa się z trzech węzłów, a utworzono w niej 22 bloki. Pierwszy blok przechowuje nieistotne dane, dlatego należy go odliczyć. Ostatecznie dodano 21 nowych bloków, czyli tyle ilu było wyborców uprawnionych do głosowania. Wyniki wyborów zostały opublikowane i otrzymano diagram, który przedstawiono na rysunku 7.8.

### Blochain network status



Rysunek 7.7: Status wyborów.



Rysunek 7.8: Wyniki wyborów.



## Rozdział 8

### Podsumowanie i wnioski

Celem pracy było opracowanie oraz zaimplementowanie części systemu głosowania on-line, który wykorzystywał będzie technologię Blockchain. W skład systemu wchodzi cztery moduły: aplikacja wyborcy, aplikacja administratora, serwer z API systemu i sieć Blockchain. System umożliwia konfigurację i przeprowadzenie wyborów elektronicznych, z wykorzystaniem aplikacji przeglądarkowej. W skład systemu wchodzi oprogramowanie, które umożliwia tworzenie węzłów sieci Blockchain. Węzły działają zgodnie z częścią założeń technologii Blockchain. Stworzony system spełnia funkcjonalności oraz realizuje cel pracy.

Zaproponowana implementacja systemu nie wyczerpuje w pełni tematu wyborów *on-line* i technologii Blockchain. Jednym z możliwych kierunków rozbudowy aplikacji jest utworzenie osobnego interfejsu graficznego, dla każdego węzła sieci Blockchain. Takie rozwiązanie zwiększyłoby kontrolę nad siecią Blockchain, a także ułatwiłoby zarządzanie siecią.

Dodatkowo obecny system pozwala na organizowanie wyborów, które dotyczą jednego zagadnienia. Przykładowo wybory prezydenckie pozwalają na wybranie jednej kandydatury z kilku dostępnych. System można rozbudować w taki sposób, aby do jednego procesu wyborczego można było włączyć kilka zagadnień. Przykładowo w wyborach parlamentarnych wybierani są przedstawiciele do parlamentu i senatu, a odbywa się to w ramach jednego procesu wyborczego.

Głównym problemem napotkanym podczas pracy było zaprojektowanie i zaimplementowanie rozproszonej sieci, która działa według standardów Blockchain. Zapewnienie komunikacji pomiędzy rozproszonymi węzłami zostało rozwiązane poprzez zaprojektowanie każdego węzła w konwencji REST. Dodatkowo zastosowanie podejścia REST pozwoliło na łatwe połączenie sieci Blockchain z serwerem zarządzającym wyborami, który również działał w konwencji REST.

Zastosowanie technologii opartych na NodeJs do tworzenia sieci Blockchain okazało się złym pomysłem. NodeJs nie jest wystarczająco wydajny, gdy od serwera wymagana jest duża moc obliczeniowa. Zastosowanie tego rozwiązania negatywnie przekłada się na wydajność sieci Blockchain, które wykorzystują algorytm *Proof of Work*.





# Bibliografia

- [1] Bitcoin mining now consuming more electricity than 159 countries including ireland and most countries in africa. <https://powercompare.co.uk/bitcoin/>, 2018. data dostępu: rok 2020.
- [2] Proof-of-work explained. <https://cointelegraph.com/explained/proof-of-work-explained>, 2018. data dostępu: rok 2020.
- [3] Cambridge bitcoin electricity consumption index (cbeci). <https://www.epe.admin.cam.ac.uk/cambridge-bitcoin-electricity-consumption-index-cbeci>, 2019. data dostępu: rok 2020.
- [4] Blockchain w pko banku polskim. <https://fintech.pkobp.pl/blockchain-w-banku/>, 2020. data dostępu: rok 2020.
- [5] Co to smart kontrakt? <https://academy.binance.com/pl/articles/what-are-smart-contracts>, 2020. data dostępu: rok 2020.
- [6] Czym jest atak 51%? <https://academy.binance.com/pl/articles/what-is-a-51-percent-attack>, 2020. data dostępu: rok 2020.
- [7] Czym jest bitcoin? <https://academy.binance.com/pl/articles/what-is-bitcoin#what-is-bitcoin>, 2020. data dostępu: rok 2020.
- [8] Dowód autorytetu (proof of authority) - jak działa. <https://academy.binance.com/pl/articles/proof-of-authority-explained>, 2020. data dostępu: rok 2020.
- [9] Dowód stawki (proof of stake). <https://academy.binance.com/pl/articles/proof-of-stake-explained>, 2020. data dostępu: rok 2020.
- [10] General framework of electronic voting and implementation thereof at national elections in estonia. [https://www.venice.coe.int/files/13EMB/13EMB\\_Priit\\_Vinkel.pdf](https://www.venice.coe.int/files/13EMB/13EMB_Priit_Vinkel.pdf), 2020. data dostępu: rok 2020.
- [11] Jaka jest różnica między blockchainem a bitcoinem? <https://academy.binance.com/pl/articles/difference-between-blockchain-and-bitcoin>, 2020. data dostępu: rok 2020.

- [12] Kodeks spółek handlowych art. 402. tryb zwoływania walnego zgromadzenia. <https://www.lexlege.pl/ksh/oddzial-3-walne-zgromadzenie/57/>, 2020. data dostępu: rok 2020.
- [13] Leksykon pojęć na temat technologii blockchain i kryptowalut. [https://www.gov.pl/documents/31305/0/leksykon\\_pojec\\_na\\_temat\\_tehnologii\\_blockchain\\_i\\_kryptowalut.pdf/77392774-1180-79ab-4dd5-089ffab37602](https://www.gov.pl/documents/31305/0/leksykon_pojec_na_temat_tehnologii_blockchain_i_kryptowalut.pdf/77392774-1180-79ab-4dd5-089ffab37602), 2020. data dostępu: rok 2020.
- [14] Pko bp wdraża nową wersję trwałego nośnika. to pierwsze takie rozwiązanie w europie. <https://fintek.pl/pko-bp-wdraza-nowa-wersje-trwalego-nosnika-to-pierwsze-takie-rozwiazanie-w-europ>, 2020. data dostępu: rok 2020.
- [15] Przykłady wykorzystania blockchain: Internet rzeczy (iot). <https://academy.binance.com/pl/articles/blockchain-use-cases-the-internet-of-things>, 2020. data dostępu: rok 2020.
- [16] Walne zgromadzenia. evoting - jak to działa? [https://www.kdpw.pl/pl/uslugi/Walne\\_zgromadzenia/Strony/Jak-to-dziala.aspx](https://www.kdpw.pl/pl/uslugi/Walne_zgromadzenia/Strony/Jak-to-dziala.aspx), 2020. data dostępu: rok 2020.
- [17] Węzły (nodes) - czym są i jak działają? <https://academy.binance.com/pl/articles/what-are-nodes>, 2020. data dostępu: rok 2020.
- [18] Stuart Haber, W. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3, 09 1999.
- [19] Ankit Patel. Medium. [https://medium.com/@ankit\\_233/history-of-the-blockchain-1991-38d6d4c3420c](https://medium.com/@ankit_233/history-of-the-blockchain-1991-38d6d4c3420c), 2018. data dostępu: rok 2020.
- [20] P. Rodwald. Kryptograficzne funkcje skrótu. *Zeszyty Naukowe Akademii Marynarki Wojennej*, R. 54 nr 2 (193):91–102, 2013.
- [21] Sarwar Sayeed, Hector Marco-Gisbert. Assessing blockchain consensus and security mechanisms against the 51 *Applied Sciences*, 9(9), 2019. data dostępu: rok 2020.
- [22] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. strony 101 – 102, 09 2001.
- [23] Ravindhar Vadapalli. *BLOCKCHAIN FUNDAMENTALS TEXT BOOK Fundamentals of Blockchain*. 10 2020.

# Dodatki



# Spis skrótów i symboli

REST - Representational State Transfer

API - Application Programming Interface

IoT - Internet of Things

ORM - Object Relational Mapping

JWT - Json Web Token

SPA - Single Page Application

NPM - Node Node Packet Module

GB - Giga Bajt

KDPW - Krajowy Depozyt Papierów Wartościowych

IP - Internet Protocol

Do pracy dołączona jest płyta CD, z następującą zawartością:

- dokumentacja – (źródła  $\text{\LaTeX}$ owe i końcowa wersja w **pdf**),
- admin-app – źródła aplikacji przeglądarkowej administratora,
- voter-app – źródła aplikacji przeglądarkowej wyborcy,
- blockchain-server – źródła aplikacji serwerowej, będącej instancją węzła sieci Blockchain,
- e-voting-api – źródła aplikacji serwerowej, dostarczającej API systemu.

# Spis rysunków

|      |  |    |
|------|--|----|
| 2.1  | Łańcuch bloków. . . . .                              | 5  |
| 3.1  | Moduły systemu i-votingu [10]. . . . .               | 12 |
| 3.2  | Proces i-votingu [10]. . . . .                       | 13 |
| 4.1  | Diagram przypadków użycia wyborcy. . . . .           | 16 |
| 4.2  | Diagram przypadków użycia administratora. . . . .    | 17 |
| 5.1  | Rezultat polecenia <i>node -v</i> . . . . .          | 24 |
| 5.2  | Struktura folderów . . . . .                         | 24 |
| 5.3  | Formularz logowania administratora. . . . .          | 25 |
| 5.4  | Pasek nawigacji administratora. . . . .              | 26 |
| 5.5  | Lista wyborców. . . . .                              | 26 |
| 5.6  | Okno dodawania wyborcy. . . . .                      | 26 |
| 5.7  | Wyszukiwarka węzłów. . . . .                         | 27 |
| 5.8  | Konfiguracja podłączonego węzła. . . . .             | 27 |
| 5.9  | Widok listy wyborów. . . . .                         | 28 |
| 5.10 | Konfiguracja wyborów etap pierwszy. . . . .          | 28 |
| 5.11 | Wyświetlanie błędów formularza. . . . .              | 29 |
| 5.12 | Formularz dodawania kandydatów. Etap drugi . . . . . | 29 |
| 5.13 | Lista dostępnych wyborów. . . . .                    | 30 |
| 5.14 | Lista dostępnych wyborów. . . . .                    | 30 |
| 5.15 | Ikona edycji. . . . .                                | 31 |
| 5.16 | Komponent Status. . . . .                            | 31 |
| 5.17 | Zmiana adresu serwera. . . . .                       | 31 |
| 5.18 | Lista kandydatów. . . . .                            | 32 |
| 5.19 | Lista wyborców. . . . .                              | 32 |
| 5.20 | Dane dotyczące sieci Blockchain. . . . .             | 32 |
| 5.21 | Przycisk publikowania wyborów. . . . .               | 33 |
| 5.22 | Diagram słupkowy prezentujący wyniki. . . . .        | 33 |
| 5.23 | Pasek nawigacji. . . . .                             | 33 |
| 5.24 | Okno logowania. . . . .                              | 34 |
| 5.26 | Strona wyborcy. Instrukcja obsługi. . . . .          | 34 |
| 5.25 | Obsługa błędnych danych logowania. . . . .           | 35 |
| 5.27 | Okno wyborcze. . . . .                               | 35 |
| 5.28 | Panel wyborów. . . . .                               | 36 |
| 5.29 | Pomyślnie wysłany głos. . . . .                      | 36 |

|      |   |    |
|------|---|----|
| 5.30 | Okno wyborów. Po zagłosowaniu. . . . .                              | 36 |
| 5.31 | Okno wyborów. Po ogłoszeniu wyników. . . . .                        | 36 |
| 5.32 | Wykres wyników wyborów. . . . .                                     | 37 |
| 6.1  | Schemat przedstawiający komponenty systemu. . . . .                 | 40 |
| 6.2  | Schemat przedstawiający przykładową sieć Blockchain. . . . .        | 41 |
| 6.3  | Klasa <i>Block</i> . . . . .  | 42 |
| 6.4  | Klasa <i>ChainRepository</i> . . . . .                              | 43 |
| 6.5  | Fragment klasy Blockchain. . . . .                                  | 44 |
| 6.6  | Schemat przedstawiający relacyjny schemat bazy danych. . . . .      | 45 |
| 6.7  | Metoda <i>validateNonce</i> . . . . .                               | 46 |
| 6.8  | Metoda <i>solveNonce</i> . . . . .                                  | 47 |
| 6.9  | Dodawanie nowego bloku. Metoda <i>mine</i> . . . . .                | 47 |
| 6.10 | Metoda <i>synchronize</i> . . . . .                                 | 48 |
| 6.11 | Dodawanie nowego bloku w algorytmie Proof of Authority. . . . .     | 48 |
| 6.12 | Synchronizacja w algorytmie Proof of Authority. . . . .             | 49 |
| 6.13 | Klasa <i>Database</i> . Singleton. . . . .                          | 50 |
| 6.14 | Klasa <i>CandidateRepository</i> . . . . .                          | 51 |
| 6.15 | Klasa <i>Blockchain</i> . . . . .                                   | 52 |
| 6.16 | Stan początkowy wycinka <i>electionsSlice</i> . . . . .             | 53 |
| 6.17 | Wycinek <i>electionsSlice</i> . . . . .                             | 54 |
| 6.18 | Thunk <i>fetchElection</i> . . . . .                                | 55 |
| 6.19 | Wykorzystanie Thunk'a, w komponencie <i>ElectionsPage</i> . . . . . | 55 |
| 7.1  | Rezultat polecenia <i>ifconfig</i> . . . . .                        | 58 |
| 7.2  | Konfiguracja węzłów. . . . .  | 59 |
| 7.3  | Konfiguracja wyborów. . . . .                                       | 59 |
| 7.4  | Dodawanie kandydatów. . . . .                                       | 60 |
| 7.5  | Przypisanie wyborców. . . . .                                       | 60 |
| 7.6  | Menu wyborów. . . . .   | 61 |
| 7.7  | Status wyborów. . . . .   | 61 |
| 7.8  | Wyniki wyborów. . . . .   | 61 |