

**UNIwersYTET WARMIŃSKO-MAZURSKI**  
**W OLSZTYNIE**  
**WYDZIAŁ MATEMATYKI I INFORMATYKI**

**Kierunek: Informatyka**

**Wiktor Roch**

**Zdecentralizowana aplikacja webowa „Głos  
obywatela” oparta na technologii blockchain  
Ethereum**

Praca inżynierska

Wykonana w Katedra Algebry i Geometrii  
pod kierunkiem

Dr Bogdana Starucha

Olsztyn 2019

**UNIVERSITY OF WARMIA AND MAZURY  
IN OLSZTYN  
FACULTY OF MATHEMATICS AND COMPUTER  
SCIENCE**

**Field of study: Computer Science**

**Wiktor Roch**

**Decentralized web application „Voice of the  
citizen” based on Ethereum blockchain technology**

Engineering Thesis written in  
Chair of Algebra and Geometry under  
supervision of  
Dr Staruch Bogdan

Olsztyn 2019

## Spis treści

1. Wprowadzenie .....	4
2. Podstawy teoretyczne .....	5
2.1 Pojęcia i zagadnienia .....	5
2.2 Języki, narzędzia programistyczne i usługi wykorzystane .....	8
2.3 Różnice między aplikacją webową a zdecentralizowaną aplikacją webową .....	9
3. Analiza.....	12
3.1 Diagram przypadków użycia .....	12
3.2 Model Klas .....	13
4. Realizacja.....	15
4.1 Widoki aplikacji .....	15
4.2 Budowa Smart Kontraktu .....	20
4.3 Testy automatyczne .....	22
5. Historia kryptowaluty na rynku i twórca platformy Ethereum.....	26
6. Instrukcja instalacji.....	28
7. Podsumowanie.....	29
8. Streszczenie .....	30
9. Abstract.....	31
10. Literatura i źródła .....	32

## 1. Wprowadzenie

Praca opisuje zdecentralizowaną aplikację webową do głosowania przez obywateli. Na początku pracy zostały opisane zagadnienia i wybrane technologie. Następnie część analityczna, która przy pomocy diagramów i przypadków użycia przedstawia funkcjonalności aplikacji. Kolejny rozdział pokaże interfejs użytkownika w postaci widoków i ich opisem. Po czym przejdziemy do testów automatycznych i wyjaśnienia zagadnienia smart kontraktu na bazie kodu źródłowego. Aplikacja pod względem technologicznym jest innowacyjna i dobrane technologie posiadają cechy, które znacznie zwiększają bezpieczeństwo procesu głosowania. Bezpieczeństwo i zabezpieczenia będą sprawdzane poprzez testy automatyczne. Tego typu aplikacje będą umożliwiały nam w niedalekiej przyszłości głosowanie bez wychodzenia z domu. Technologia zapoczątkowana przez kryptowalutę Bitcoin była publiczna. Jej kod źródłowy jest do wglądu dla każdego. Szerokie grono programistów i analityków systemowych dostrzegło potencjał i zaczęło tworzyć nowe projekty w oparciu o podobną ideę. Zaowocowało to powstaniem takiej kryptowaluty jak Ethereum. Technologie użyte w pracy zyskują na popularności i są coraz częściej wybierane przez deweloperów.

Aplikacja rozwiąże problem z niskim procentem głosujących obywateli, gdyż dzięki niej nie będzie trzeba wychodzić z domu aby zagłosować. Jest to szczególne ułatwienie dla osób niepełnosprawnych, które mają problem z poruszaniem się. Będzie to kolejna dziedzina życia zoptymalizowana pod względem wygody, kosztów i czasu. Kolejne aplikacje obecnie tworzone w oparciu o tę technologie są dołączane do głównej sieci Ethereum i tworzą coraz większą sieć. Aplikacja w pracy jest oparta na wirtualnej sieci Ethereum, gdyż każdy ruch w sieci generuje prowizję (gaz) i każdy test funkcji wymagałoby waluty Ethereum z głównej sieci. Więc cała aplikacja jest zbudowana na wirtualnej maszynie, która dostarcza walutę do działania aplikacji.

## 2. Podstawy teoretyczne

### 2.1 Pojęcia i zagadnienia

**Aplikacja webowa** - nazwa pochodzi od angielskiego zwrotu „web application”, dlatego w języku polskim aplikacja internetowa nazywana jest zamiennie aplikacją webową. Samo określenie „aplikacja internetowa” oznacza program komputerowy, który pracuje na serwerze i komunikuje się z hostem użytkownika komputera po przez sieć komputerową. Do tego zadania wykorzystuje się przeglądarkę internetową użytkownika, aby aplikacja webowa działała. Użytkownik staje się interaktywnym klientem aplikacji webowej. Aby uruchomić aplikację webową niezbędny jest dostęp do Internetu. To wymaga funkcjonowania serwera – komputera lub aplikacji zainstalowanej na komputerze. Zadaniem serwerów HTTP (z angielskiego Hypertext Transfer Protocol) jest serwowanie stron internetowych. Kiedy w okienko przeglądarki wpisujemy adres strony, np. [www.wp.pl](http://www.wp.pl) do serwera firmy, w której wykupiona jest usługa, zostaje wysłane zapytanie HTTP. W odpowiedzi przeglądarka wyświetla stronę internetową wraz ze znajdującą się na niej treścią [1].

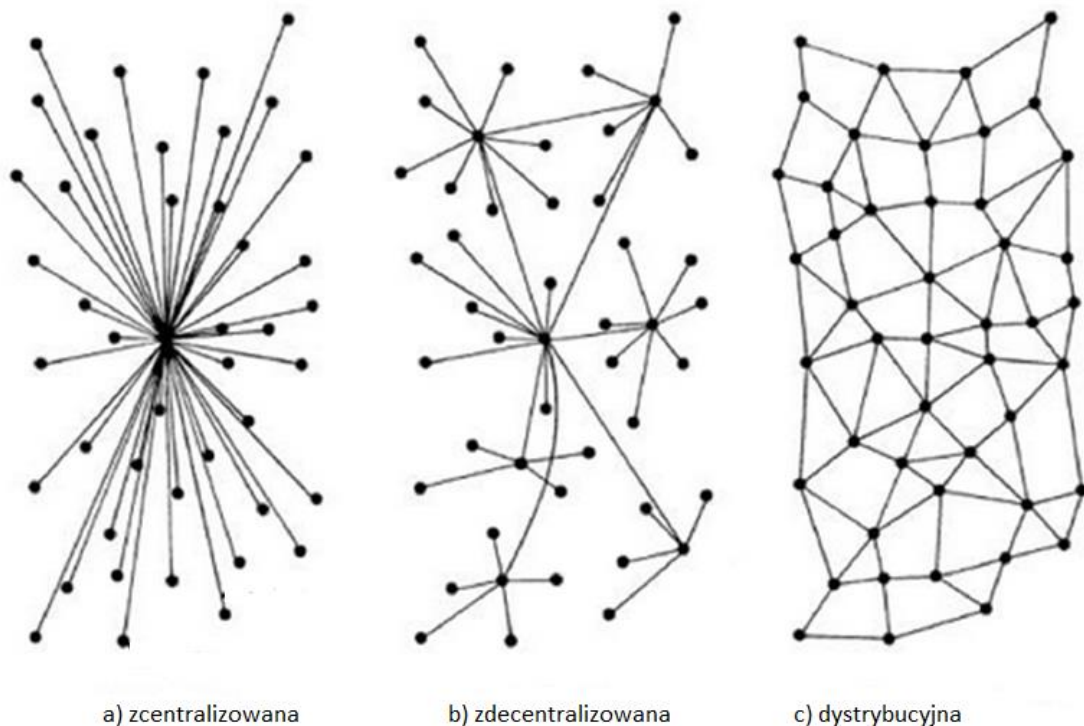
**Serwer** - specjalny komputer zawsze włączony i podłączony do Internetu, na którym zainstalowane są programy do obsługi usług internetowych. Dzięki serwerowi możesz opublikować stronę WWW, używać poczty e-mail czy magazynować pliki. Na serwerze uruchomionych jest kilka, kilkanaście programów, które zapewniają świadczenie usług niezbędnych do twojej obecności w Internecie. Zazwyczaj każdy internetowy komputer wyspecjalizowany jest w innym zadaniu. Jeden odpowiada za wyświetlanie stron WWW, inny za obsługę poczty e-mail, jeszcze inny za tworzenie kopii zapasowych, itd. Takie rozbieżności różnych funkcji świadczy o profesjonalnym podejściu firmy hostingowej do świadczonych usług [2].

**Blockchain** lub łańcuch bloków (czasem też łańcuch blokowy) – zdecentralizowana i rozproszona baza danych najczęściej spotykana w modelu open source w sieci internetowej o architekturze peer-to-peer (P2P) bez centralnych komputerów i niemająca scentralizowanego miejsca do przechowywania danych, służąca do księgowania poszczególnych transakcji, płatności lub zapisów księgowych zakodowana za pomocą algorytmów kryptograficznych. W istocie blockchain to swoisty zdecentralizowany i rozproszony rejestr transakcji. Mówiąc inaczej jest to zdecentralizowana platforma transakcyjna w rozproszonej infrastrukturze sieciowej. Blockchain to publiczny i jawny rejestr, do którego dostęp może uzyskać każdy.

**Kryptowaluta**, rzadziej nazywana waluta kryptograficzna – rozproszony system (rejestr) księgowy bazujący na kryptografii, przechowujący informację o stanie posiadania jednostek o realnej wartości. Stan posiadania związany jest z poszczególnymi węzłami systemu „portfelami” w taki sposób, aby kontrolę nad danym portfelem miał wyłącznie posiadacz odpowiadającego mu klucza prywatnego i niemożliwe było dwukrotne wydanie tej samej jednostki. Kryptowaluta nie jest przez większość państw uznawana za jednostkę walutową, środek płatniczy czy pieniądź elektroniczny, dzięki czemu tworzenie jednostek w ramach wbudowanego w system algorytmu (tzw. wydobywanie lub wykopywanie) jest legalne. Zwykle z tych samych powodów nie korzysta z ulg podatkowych przewidzianych dla handlu walutą, jednak w Unii Europejskiej, wyrokiem Trybunału Sprawiedliwości UE, kryptowaluty są uznawane za prawny środek płatniczy w zakresie opodatkowania VAT i tym samym z VAT zwolnione [3].

**Transakcje** - podpisywane cyfrowo za pomocą kryptografii klucza publicznego przy użyciu kryptografii krzywych eliptycznych (ECDSA) i są publiczne, choć użytkownicy są anonimowi. Transakcje mają charakter rozproszony, co oznacza, że nie są monitorowane lub kontrolowane przez głównego pośrednika. Są przetwarzane przez komputery w sieci peer-to-peer, która nie weryfikuje transakcji poprzez wykorzystanie centralnych komputerów, co ma miejsce w transakcjach w obecnych popularnych systemach bankowych. Transakcje nazywane są węzłami i po potwierdzeniu poprawności, dodawane do replikowanego i znakowanego czasem rejestru transakcji, nazywanego blokiem. Przetwarzanie transakcji nie jest bezpłatne. Wysokość opłaty za transakcję jest uzależniona od ilości transakcji danej kryptowaluty w obecnym momencie, gdyż istnieją ograniczenia przepustowości dla transakcji[3].

Typy serwerów ze względu na rodzaj współpracy z aplikacją.



Rys. 1. Typy serwerów

a) Zcentralizowana- najbardziej popularny model współpracy aplikacji webowych lub innych programów z bazą danych. Niskie koszty utrzymania i duża ilość pamięci sprawiają, że jest najczęściej wykorzystywany do przechowywania dużych ilości danych, które nie wymagają najwyższego poziomu bezpieczeństwa.

b) Zdecentralizowana- rodzaj zyskujący na popularności ze względu na swoje cechy unikalne. Bezpieczeństwo i odporność na ataki hakerskie, drogie utrzymanie i mała ilość przechowywania danych, sprawiają, że jest idealnym narzędziem do aplikacji stworzonych do płatności, weryfikacji użytkownika i innych zaufanych usług.

c) Dystrybucyjna- mało popularna ze względu na wysoki koszt utrzymania i duże obciążenie w sieci, przedstawiona w ramach ciekawostki.

**Smart Contract** - inteligentne umowy, podobnie jak tradycyjne papierowe, są gwarantem zaufania. Są jednak pewne ich wyróżniki. Nie są stosem papieru wypełnionym prawnym i trudnym do zrozumienia językiem. Zwykle, dużo czasu pochłania prawnikom weryfikacja umów pod kątem prawnym. Inteligentne umowy to cyfrowe wersje tych tradycyjnych. Są to najzwyczajniej programy, które działają na platformie Ethereum w ramach technologii blockchain i mają to samo znaczenie: prawnie zbindzać każdą osobę, która ma zamiar uczestniczyć w zawarciu umowy. To nic innego jak linie kodu zapisane w języku Solidity porównywalnym do języka Java Script. Kod ten jest następnie przekształcany w kod bajtowy i wydany do blokady jako inteligentny kontrakt. Każda umowa ma własny adres blokowy. Oznacza to, że jeśli zaczynasz porozumiewanie się z kimś umowa zostanie utworzona w ramach bloku, a jej adres będzie dostępny dla wszystkich zainteresowanych stron. Następnie można użyć go do interakcji z inteligentną umową i wypełnić zobowiązania do rozpoczęcia jej realizacji. Warto zwrócić uwagę, że inteligentne umowy mogą również łączyć się i współpracować z innymi inteligentnymi umowami. Cały proces przebiega na zasadzie blokady, a zatem realizacja inteligentnej umowy jest przejrzysta, pewna, niezmienna, niedroga i zdecentralizowana[4].

**Ethereum** - zdecentralizowana platforma, która obsługuje inteligentne kontrakty: aplikacje działające dokładnie tak, jak zaprogramowano bez możliwości przestojów, cenzury, oszustw lub ingerencji stron trzecich. Te aplikacje działają na specjalnie zbudowanym blockchainie, niezwykle potężnej, współużytkowanej globalnej infrastrukturze, która może przenosić wartość wokół i reprezentować własność nieruchomości. Umożliwia to programistom tworzenie rynków, przechowywanie rejestrów długów lub obietnic, przenoszenie funduszy zgodnie z instrukcjami udzielonymi w przeszłości (jak testament lub kontrakt terminowy) i wiele innych rzeczy, które nie zostały jeszcze wynalezione, wszystkie bez pośrednika lub ryzyka kontrahenta.

Projekt został rozpakowany przez eterową premierę w sierpniu 2014 r. przez fanów na całym świecie. Jest rozwijany przez Ethereum Foundation, szwajcarską organizację non-profit, z udziałem wielkich umysłów na całym świecie.

## 2.2 Języki, narzędzia programistyczne i usługi wykorzystane

**Javascript** - Najczęściej spotykanym zastosowaniem języka JavaScript są witryny internetowe. Skrypty te służą najczęściej do zapewnienia interakcji poprzez reagowanie na zdarzenia(eventy), walidacji danych wprowadzanych w formularzach lub tworzenia złożonych efektów wizualnych. Skrypty JavaScriptu uruchamiane po przez strony internetowe mają znacznie ograniczony dostęp do komputera użytkownika. Po stronie serwera JavaScript może działać w postaci node.js lub Ringo

**HTML** - pozwala opisać strukturę informacji zawartych wewnątrz witryny internetowej, nadając znaczenie poszczególnym fragmentom tekstu – formując hiperłącza, akapity, nagłówki, listy – oraz osadza w tekście dokumentu obiekty plikowe np. multimedia bądź elementy baz danych np. interaktywne formularze danych. HTML daje możliwość określenia wyglądu dokumentu na stronie internetowej. Do szczegółowego opisu formatowania akapitów, nagłówków, użytych czcionek i kolorów, zalecane jest wykorzystywanie kaskadowych arkuszy stylów.

**Pragma Solidity** - wysokopoziomowy język programowania, zorientowany na tworzenie i implementację smart kontraktów. Współpracuje z C++, Python i Javascript, zaprojektowany do współpracy z wirtualną maszyną Ethereum. Jest statycznie typowany, wspiera dziedziczenie i biblioteki kompleksowo wspierające między innymi definiowanie użytkownika.

**Node.js** – środowisko uruchomieniowe stworzone do tworzenia wysoce skalowalnych aplikacji internetowych, głównie serwerów www napisanych w języku JavaScript. Umożliwia tworzenie aplikacji sterowanych zdarzeniami wykorzystujących asynchroniczny system wejścia-wyjścia.

**Truffle** - narzędzie umożliwiające napisanie zdecentralizowanej aplikacji poruszające się po sieci Ethereum, napisanie w języku Solidity smart kontraktu, następnie skompilowania go i wykorzystanie do transakcji w aplikacji. Dodatkowo, udostępnia narzędzia do testowania naszego smartkontraktu.

**Ganache** - program do pobrania, niezbędny dla początkowego rozwoju zdecentralizowanej aplikacji. Tworzy on wirtualną zdecentralizowaną sieć na potrzeby dewelopera i testów zanim aplikacja trafi na główną sieć Ethereum. Udostępnia również szereg użytkowników z kluczami prywatnymi do wprowadzenia w słudze Metamask.

**Metamask** - rozszerzenie dla przeglądarki Google, umożliwiające przełączanie się między użytkownikami, tworzenie nowych i ich importowanie na podstawie klucza prywatnego z programu Ganache. Jest niezbędnym połączeniem użytkownika i wirtualnej sieci do interakcji między nimi po przez aplikację.

**Notepad++** - program do pisania kodu aplikacji w językach: HTML, CSS, Java Script, Pragma Solidity

**Google Chrom** - przeglądarka internetowa, niezbędna do zainstalowania rozszerzenia Metamask i wyświetlenia interfejsu użytkownika aplikacji.

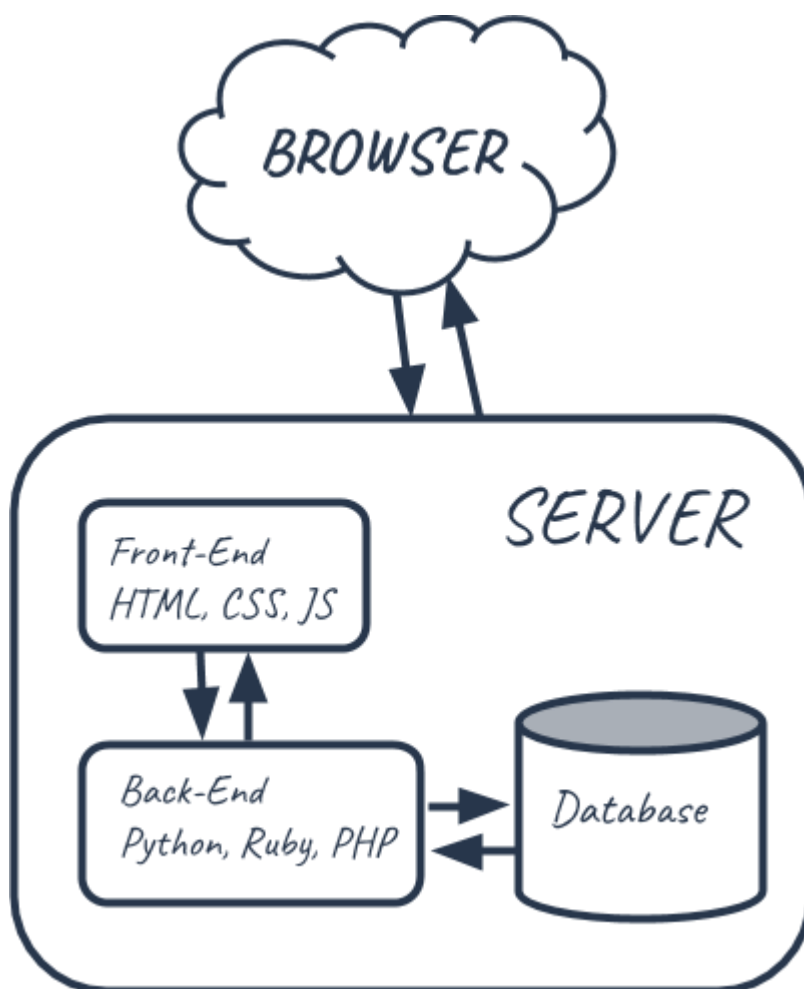
**Chai** - biblioteka dla środowiska node.js, dostarcza funkcje do łatwego i wygodnego testowania procedur i funkcji oprogramowania napisanego w języku Java Script



## 2.3 Różnice między aplikacją webową a zdecentralizowaną aplikacją webową

Aplikacja webowa posiada strukturę zbudowaną z front-endu, który jest interpretowany przez przeglądarkę (przykładowo Google chrom). Front-end jest tworzony za pomocą języków HTML, CSS i JS. HTML odpowiada za strukturę strony internetowej, przykładowo jej zakładki, tabele, pola tekstowe itp. Każdy z tych elementów można edytować, mowa tu o czcionce marginesach kolorach i rozmiarach, lecz nie można zmienić wartości za pomocą kodu CSS. Następnie zmianę wartości za pomocą funkcji front-endowych realizujemy za pomocą języka JS, przykładowo zegar na stronie.

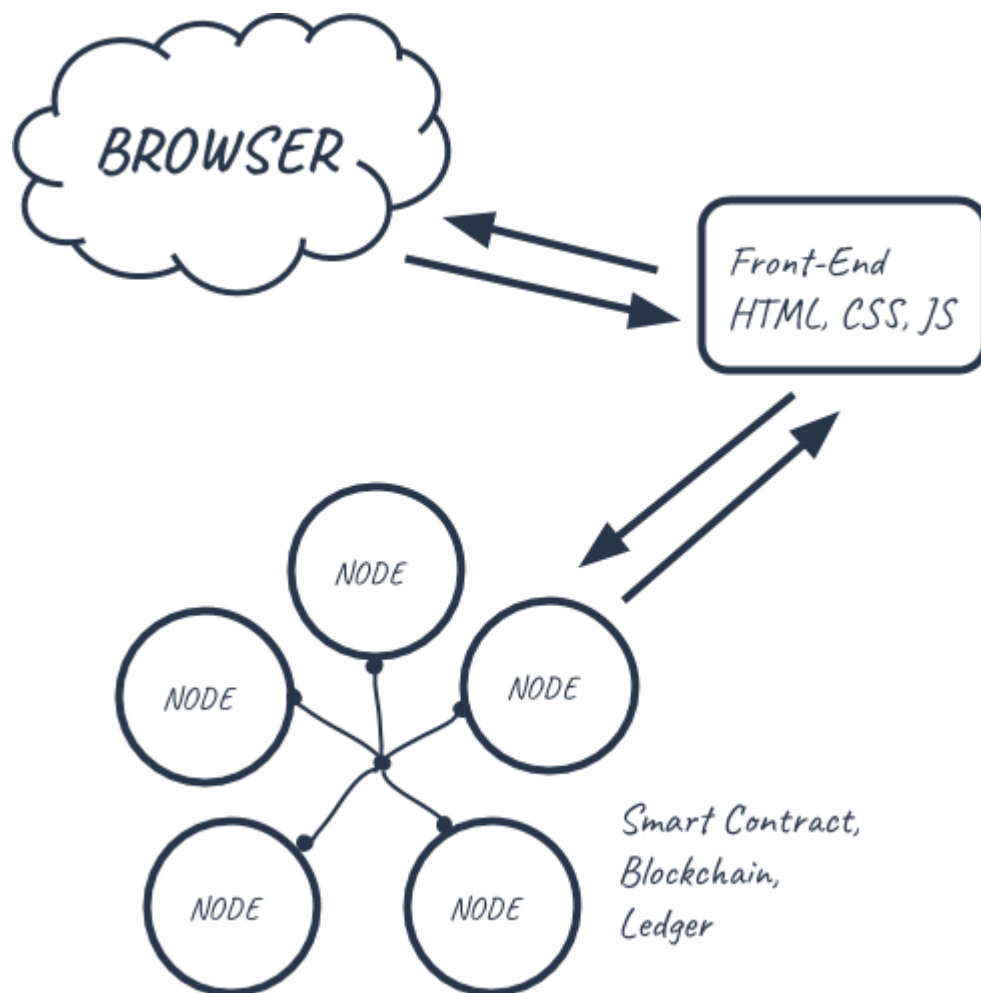
Kolejnym niezbędnym elementem aplikacji webowej jest komunikacja z bazą danych, aby przykładowo zapisać klienta sklepu internetowego w celu realizacji zakupów internetowych. Przykładowymi językami są PHP, Python, Ruby. To dzięki nim informacje z formularzy są przekazywane do bazy danych, a tam zapisywane, edytowane lub usuwane. Bazy danych również mogą posiadać swoje procedury bazodanowe, ale nie ma sensu zagłębiać się aż tak. Języki back-endowe głównie są wykorzystywane do tworzenia CRUDa na bazach.



Rys. 2. Schemat aplikacji webowej

Zdecentralizowana aplikacja webowa na pierwszy rzut oka z poziomu interfejsu użytkownika może wydawać się wręcz identyczna. Mamy przeglądarkę i języki front-end-owe jak w przypadku klasycznej aplikacji webowej. Kluczowa różnica technologiczna kryje się niżej, na poziomie back-end. Konkretnie komunikacja między serwerem a warstwą front-endu. Serwerem, w przypadku zdecentralizowanej aplikacji, są nody lub łańcuch bloków[6]. Są one połączone i przechowują informacje. Są one tylko dopisywane, z każdym kolejnym kontraktem powstaje nowy blok z danymi na jego temat, a następnie warstwa front-end je odczytuje i prezentuje. Wspomniana idea blockchain to nieustanny rozrost sieci i jego optymalizacja. Im większa sieć tym szybciej działa, gdyż jest więcej połączeń i kontrakt zapisze się optymalnie szybko po najmniejszej ilości przesiadek do konkretnego noda. Mowa tu o wielkich sieciach takich jak sky-net rozwijany przez Bitcoina.

Poniżej mamy schemat prezentujący ogólną budowę takiej aplikacji, kluczowym elementem są tu smart kontrakty które przedstawię w dalszej części pracy. To one są odpowiedzialne za zapis danych w aplikacji.



Rys. 3. Schemat zdecentralizowanej aplikacji

Aplikacja wykorzystuje schemat zdecentralizowanej aplikacji zilustrowanej wyżej. Przeglądarka (Browser) została wykorzystana firmy Google. Została ona wybrana dlatego, że posiada niezbędne rozszerzenia (Metamask), które współpracują z aplikacją umożliwiając logowanie użytkownika. Zaletą gotowego rozwiązania jest zaoszczędzenie czasu na

programowaniu mechanizmów rejestracji i logowania, a zaoszczędzony czas można wykorzystać na testy manualne lub automatyczne gotowego rozwiązania.

Powiązanie usługi logowania odbywa się na poziomie front-endu. Poziom ten jest napisany głównie w za pomocą HTML i Java Script. Struktura stron i mechanizmy wywołujące wydarzenia (event) aplikacji zostały napisane w języku HTML w wersji piątej. Funkcje, mechanizmy i logika zostały napisane w Java Script. Najważniejszym elementem na poziomie front-endu są funkcje łączące aplikacje z rozszerzeniem Metamask, które reagują dynamicznie na stan zalogowany lub nie dla użytkownika. Aplikacja, w sposób dynamiczny nasłuchuje wywołania funkcji takie jak, ładowanie danych zalogowanego konta, ładowanie danych kontraktu. Zostały zrealizowane na tym poziomie.

W klasycznym podejściu do aplikacji webowych kolejnym poziomem jest back-end, lecz w opisywanej w pracy aplikacji mamy tu znaczną zmianę, która pomija programowanie back-endu. Mamy w tym miejscu smart kontrakt. To one są odpowiedzialne za zapis danych w blokach danych. Założenie serwera również nie występuje w tej aplikacji. Zamiast niego mamy bloki danych, do których dane zapisywane są za pomocą kontraktów, między użytkownikiem a systemem, a nie zapytań SQL. Programowanie smart kontraktu można porównać do pisania funkcji w języku Java, gdzie przycisk wywołuje funkcje, w której jest szablon zapytania. Do szablonu dodawane są często dodatkowe dane które użytkownik wybrał z poziomu interfejsu i następnie baza otrzymuje dane. Kontrakty to większe funkcje, które tworzą duży obiekt z informacjami, również jest schematyczny. Różni się od zapytania SQL. Główną różnicą jest tworzenie bloku na podstawie schematu z kontraktu i wiązanie bloków w oparciu o informację z kontraktu.

Ostatnim poziomem jest baza danych, w klasycznym podejściu serwer, a w aplikacji zdecentralizowanej blockchain. Serwer i jego tabele są przeglądane przykładowo w programie SQL Developer. Blockchain będziemy przeglądać w programie Ganache.

### 3. Analiza

#### 3.1 Diagram przypadków użycia

W tym rozdziale zostanie przedstawiona aplikacja ze strony analitycznej. Diagram przypadków użycia przedstawia aktorów i ich funkcjonalności w aplikacji ‘Głos obywatela’.

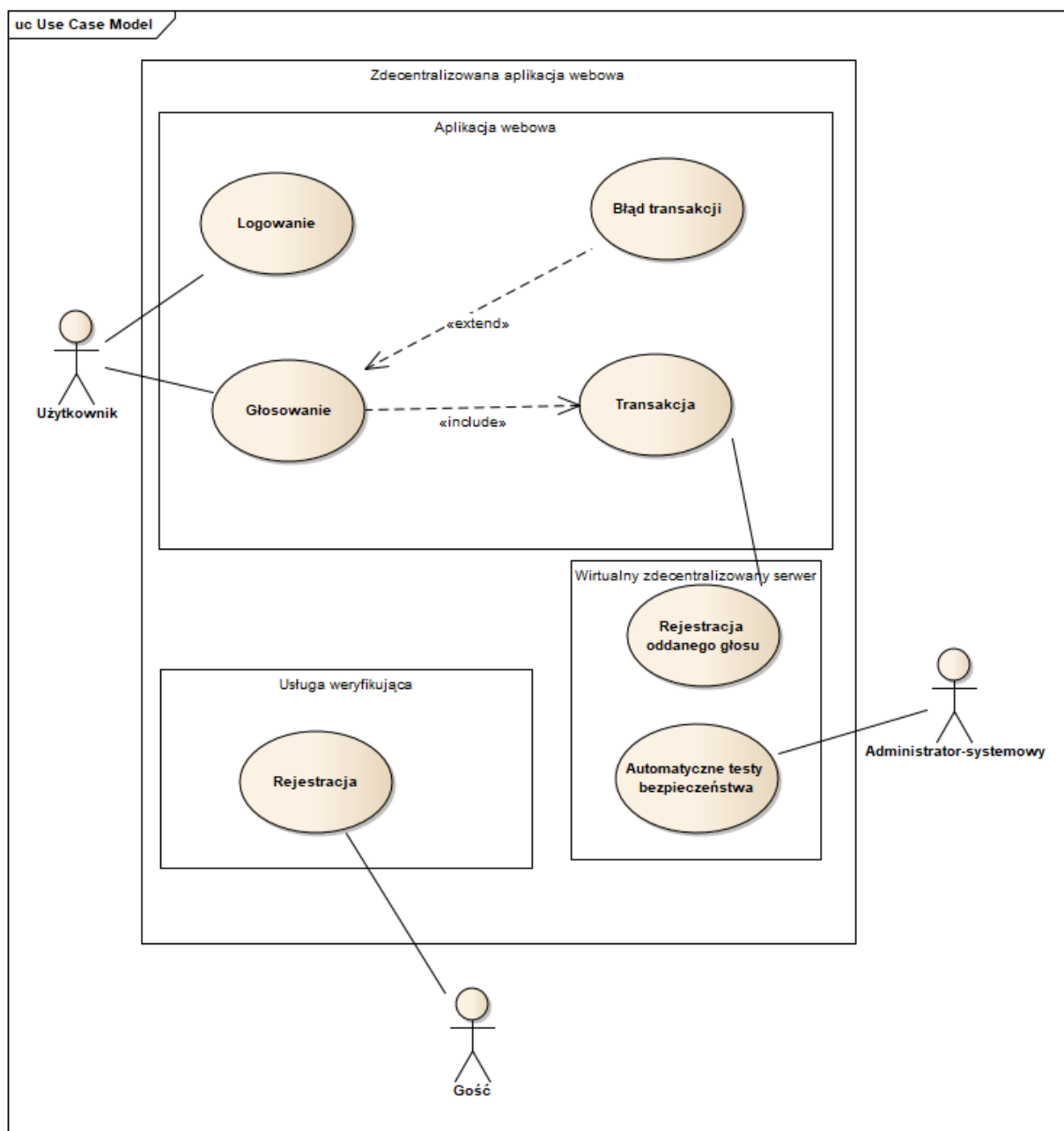


Diagram 1. Diagram przypadków użycia

Aktorzy:

**Gość**- nie zweryfikowany użytkownik, który wszedł na adres strony i widzi widok ładowania bez żadnych wrażliwych elementów aplikacji:

- Rejestracja - aktor 'Gość', może przejść pomyślnie weryfikację i stać się użytkownikiem aplikacji, poprzez sukces procesu weryfikacji w usłudze Metamask. Niezbędne jest zaimportowanie klucza prywatnego z wirtualnej bazy użytkowników (Ganache).

**Użytkownik** - zarejestrowany gość w usłudze weryfikującej Gościa(Metamask).

- Logowanie - proces, w którym użytkownik wprowadza hasło do usługi Metamask i wybiera odpowiednie konto wcześniej stworzone. Widok ładowania w momencie wybrania odpowiedniego konta zmienia się na widok główny aplikacji.
- Głosowanie - sukces procesu jest możliwy po zalogowaniu. Widok główny posiada listę kandydatów i zalogowany użytkownik dokonuje indywidualnego wyboru, następnie zatwierdza go przyciskiem funkcyjnym 'Głosuj'.
- Transakcja - widok pokazujący się po wybraniu przycisku funkcyjnego 'Głosuj'. Użytkownikowi pokazuje się widok smart kontraktu z usługi Metamask, który finalizuje oddanie głosu na kandydata.
- Błąd transakcji - przypadek systemowy w sytuacji, gdy użytkownik już oddał głos na kandydata z konta obecnie, na którym korzysta z aplikacji.
- Rejestracja oddanego głosu- systemowy przypadek użycia. Zrealizowany w momencie zawarcia smart kontraktu, między użytkownikiem a systemem. Następnie ilość oddanych głosów na konkretnego kandydata jest inkrementowana.

**Administrator-systemowy** - użytkownik odpowiedzialny za bezpieczeństwo i poprawny przebieg głosowania:

- Automatyczne testy bezpieczeństwa - seria testów wykonywanych na początku głosowania przez Administratora-systemowego, w celu weryfikacji poprawności działania całego systemu aplikacji, poprawnych wartości kandydatów, funkcji aplikacji i usług współpracujących z aplikacją.

### 3.2 Model Klas

W tym podrozdziale zostanie opisany diagram modelu klas, który zawiera nazwę klasy, ich relację, pola i funkcje.

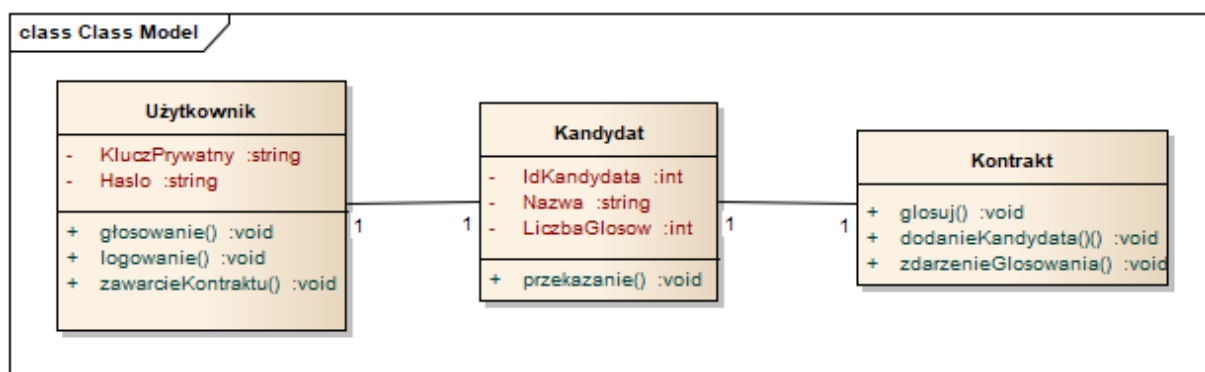


Diagram 2. Model Klas

## Słownik:

**Użytkownik** - klasa w systemie, która przechodzi główną ścieżkę przypadków funkcyjnych systemu, posiada takie wartości i procedury funkcyjne jak:

- **KluczPrywatny** - wartość typu napisowego, każdy użytkownik na jej podstawie ma stworzone konto w usłudze Metamask, co uprawnia do głosowania.
- **Hasło**- wartość typu napisowego, niezbędne do zalogowania się do usługi Metamask
- **logowanie()** - funkcja odpowiedzialna za weryfikację użytkownika i pozwala na przejście do głównego widoku aplikacji, niezbędne jest hasło wprowadzone w usłudze Metamask, następnie wybranie wcześniej zaimportowanego konta lub stworzenie nowego dzięki kluczowi prywatnemu.
- **głosowanie()** - funkcja odpowiedzialna rozwinięcie się elementu 'droplist' z kandydatami do wyboru. Następnie użytkownik dokonuje wyboru i zatwierdź przyciskiem funkcyjnym 'Głosuj'.
- **zawarcieKontraktu()** - funkcja wywołana przyciskiem 'Głosuj'. Użytkownikowi wyświetla się widok kontraktu. Kontrakt jest wygenerowany przez usługę Metamask, Użytkownik ma wybór, przycisk potwierdzenia kontraktu tworzy umowę między użytkownikiem a systemem i zapisuje szczegóły kontraktu w bloku zdecentralizowanej sieci. Jeżeli użytkownik odrzuci kontrakt to system wróci do widoku głównego aplikacji. Użytkownik może ponownie spróbować zawrzeć kontrakt, aby wybrać jednak innego kandydata. Raz dokonany wybór (zawarty kontrakt) blokuje możliwość ponownego głosowania w obecnych wyborach.

**Kandydat** - klasa w systemie, przechowująca informacje niezbędne do prawidłowego działania systemu głosowania, posiada takie pola i procedury funkcyjne jak:

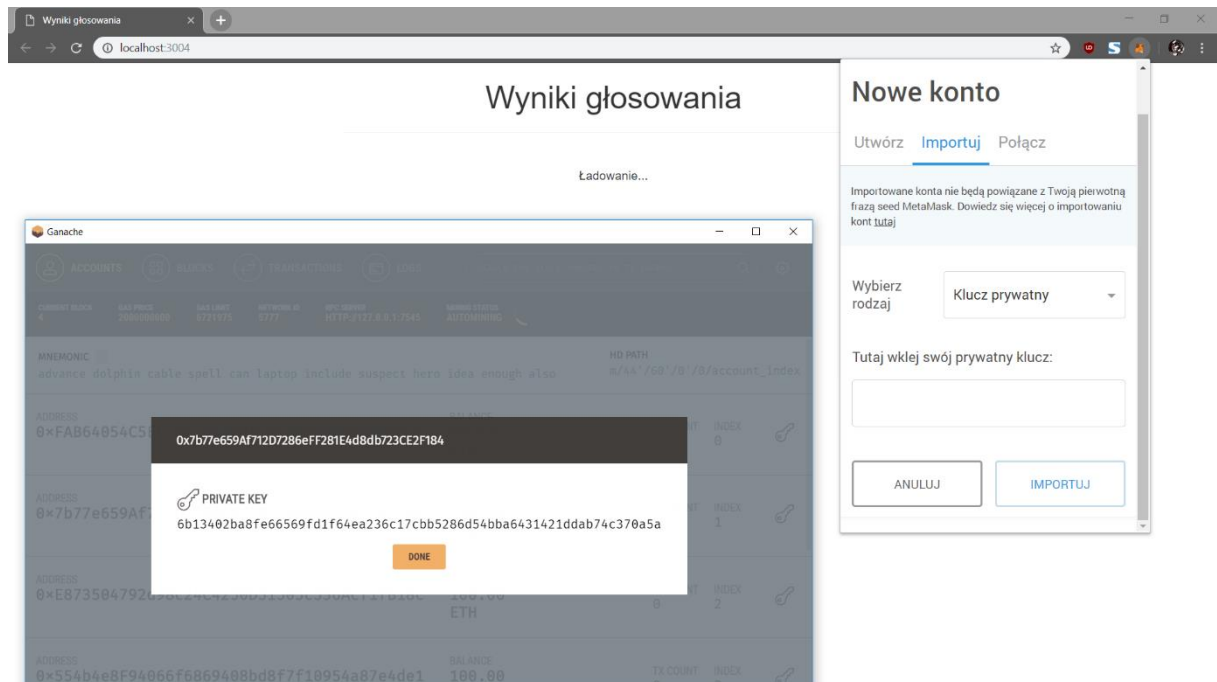
- **IdKandydata** - wartość typu liczbowego, zawiera unikalną wartość niezbędną do identyfikacji kandydatów biorących udział w wyborach.
- **Nazwa**- wartość typu napisowego, jest widoczny w 'droplist' przy funkcji głosowania przez użytkownika. Zawiera przykładową nazwę kandydata, po którym jest identyfikowany przez osobę głosującą.
- **LiczbaGłosów** - wartość typu liczbowego, przechowuje liczbę oddanych głosów na konkretnego kandydata, wartość jest inkrementowana po zawarciu kontraktu przez użytkownika i system (funkcja **zawarcieKontraktu()**).
- **Przekazanie()** - funkcja odpowiedzialna za przekazanie danych kandydata i użytkownika głosującego na niego po zawarciu kontraktu. Suma tych informacji jest przekazana do funkcji **głosuj()** klasy kontrakt.

**Kontrakt** - klasa w systemie, przechowująca funkcje odpowiedzialne za zawieranie smart kontraktów między systemem a użytkownikiem, odnotowanie zdarzenia głosowania. Zawiera funkcje:

- **głosuj()** - funkcja odpowiedzialna za sprawdzenie czy użytkownik głosował, walidacja poprawności kandydata na którego głosujemy, aktualizacja ilości głosów oddanych na kandydata, wydarzenie oddania głosu.
- **dodanieKandydata()** - funkcja odpowiedzialna za dodanie do listy odpowiednich kandydatów.
- **Zdarzenie głosowania()** - funkcja typu wydarzenie, zaczyna głosowanie z indeksem konkretnego kandydata.

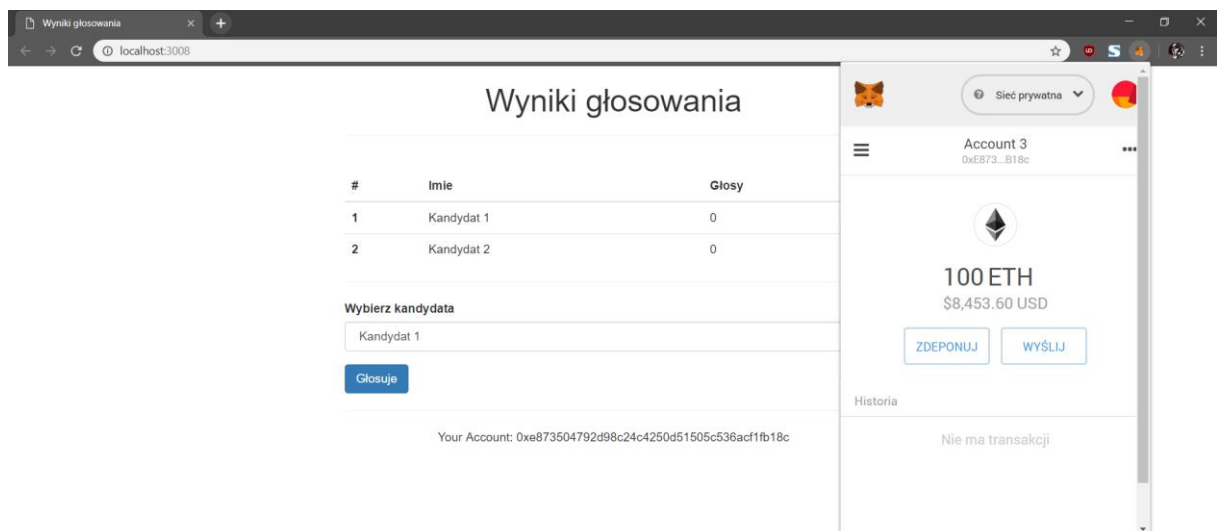
## 4. Realizacja

### 4.1 Widoki aplikacji



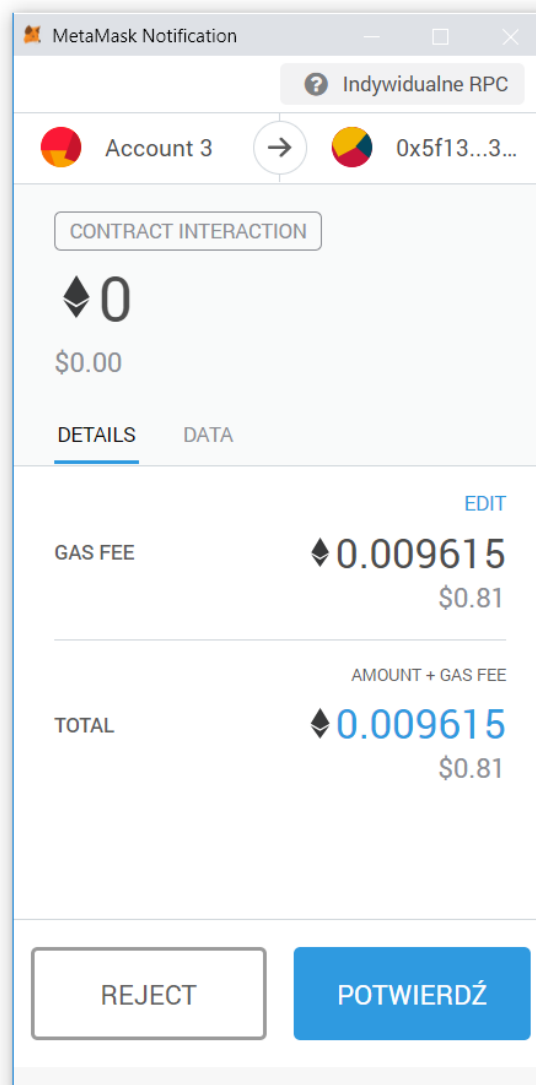
Widok 1. Rejestracja nowego użytkownika

Widok przedstawia ekran ładowania aplikacji, którą widzi Gość. Rejestracja wymaga posiadania zainstalowanej wtyczki do przeglądarki Google o nazwie Metamask. Następnie utworzenie nowego konta po przez import klucza prywatnego. Program Ganache poza wirtualnym zdecentralizowanym serwerem udostępnia pulę kont użytkowników z niezbędną do transakcji walutą Ethereum. Okno z kluczem pokazuje się po wybraniu ikony klucza w programie Ganache. Usługa Metamask deszyfruje klucz prywatny i tworzy konto.



Widok 2. Zalogowany użytkownik

Po zalogowaniu się przy pomocy hasła, jeżeli wcześniej było konto stworzone lub po imporcie klucza prywatnego, użytkownik zostaje przeniesiony do widoku głosowania. Adres naszego konta jest widoczny w usłudze Metamask i stronie aplikacji.




Widok 3. Transakcja (smart kontrakt)

Funkcja głosowania wywołuje widok kontraktu. Widok prezentuje dwa przyciski funkcyjne, potwierdzenia i odrzucenia. Przycisk odrzucenia transakcji wraca do widoku wyboru kandydata, na przykład dla sytuacji zmiany zdania w ostatniej chwili. Przycisk potwierdzający transakcję, tworzy umowę między użytkownikiem, a systemem i nalicza 'Gaz' jako opłatę za transakcję (porównać można do opłaty za przelew wyrażoną w złotych). Dane dotyczące użytkownika, opłaty, szczegóły kandydata, czas, zostaje zapisany w bloku zdecentralizowanego serwera.



#0 - 12/15/2018 at 14:58


**Contract Interaction**
-0 ETH

POTWIERDZONE
-\$0.00 USD

Szczegóły [↗](#)

Z: 0xE873504792d98c24... > Do: 0x5F1394a118FEE2f...

Transaction

Ilość	0 ETH
Limit Gazu (Units)	96151
Gas Used (Units)	64101
Cena gazu (GWEI)	100
Suma	0.00641 ETH \$0.54 USD

Activity Log

- + Transaction created with a value of 0 ETH at 14:58 on 3/16/2014.
- ↗ Transaction submitted with gas fee of 0 WEI at 15:01 on 3/16/2014.
- ✓ Transaction confirmed at 15:01 on 3/16/2014.

Widok 4. Szczegóły transakcji w usłudze Metamask

Użytkownik w każdej chwili może przejrzeć historię swoich transakcji w usłudze Metamask. Widok prezentuje takie szczegóły jak: godzina, kwota wyrażona w walucie wirtualnej Ethereum, Gas, który jest obliczany na podstawie obciążenia sieci (im więcej użytkowników transferuje swoje środki(walutę wirtualna) w sieci tym wyższy współczynnik na podstawie, którego jest obliczany gaz końcowo wyrażony w walucie Ethereum). Suma to byłby realny koszt takiej transakcji w głównej sieci Ethereum, obliczona na podstawie obecnego natężenia transakcji w sieci. Dlatego do tworzenia aplikacji niezbędna jest maszyna wirtualna z udostępnioną walutą.

Wyniki głosowania

#	Imię	Głosy
1	Kandydat 1	0
2	Kandydat 2	1

Ganache

ACCOUNTS BLOCKS TRANSACTIONS LOGS

CURRENT BLOCK 5 GAS PRICE 200000000 GAS LIMIT 6721975 NETWORK ID 5777 RPC SERVER HTTP://127.0.0.1:7545 MINING STATUS AUTOMINING

MINEMONIC advance dolphin cable spell can laptop include suspect hero idea enough also HD PATH m/44'/60'/0'/0/account\_index

ADDRESS 0xFAB64054C5ECaA1531d2B4C24e8D4bB850635DCe BALANCE 99.92 ETH TX COUNT 4 INDEX 0

ADDRESS 0x7b77e659Af712D7286eFF281E4d8db723CE2F184 BALANCE 100.00 ETH TX COUNT 0 INDEX 1

ADDRESS 0xE873504792d98c24C4250D51505c536Acf1fB18c BALANCE 99.99 ETH TX COUNT 1 INDEX 2

Account 3  
0xE873...B18c

99.9936 ETH  
\$8,455.26 USD

ZDEPONUJ WYŚLIJ

Historia

#0 - 12/15/2018 at 14:58  
Contract Interaction -0 ETH  
-\$0.00 USD

Widok 5. Zmiany po zawarciu kontraktu(zagłosowanie)

Widok przedstawia sytuację po potwierdzeniu przez użytkownika transakcji w efekcie oddanie głosu na 'kandydata 2'. Stan konta zmniejszył się o opłatę transakcji. Program Ganache przedstawia pierwsze konto administratora-systemowego (index 0) który testuje funkcje w testach i zaczyna działanie aplikacji, użytkownik głosujący (index 2) na pozycji 3 w liście. Użytkownik dokonał transakcji, stworzony blok symbolizowany jest w widoku pod opisem 'TX COUNT'.

Wyniki głosowania

#	Imię	Głosy
1	Kandydat 1	1
2	Kandydat 2	2

Your Account: 0xac67798d762e2ee92663819dd9a0e65fe1fe21c1

Account 4  
0xac67...21C1

99.9951 ETH  
\$8,452.58 USD

ZDEPONUJ WYŚLIJ

Historia

#0 - 12/15/2018 at 15:12  
Contract Interaction -0 ETH  
-\$0.00 USD

Widok 6. Stan po kilku oddanych głosach

Kilku użytkowników zagłosowało na swoich kandydatów w systemie. Każde konto ma swoją historię transakcji i aplikacja zabezpiecza sytuację, gdy konto zagłosowało już wcześniej (w oparciu o historię transakcji) nawet po restarcie smart kontraktu i ponownej budowie aplikacji.

TX HASH	FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE	STATUS
0x05c8321dc488dbf685fe98ba4734c27f5e35e593f9a1350233664a0a3b9f3fa	0xa0c6779d762e2e92663819dd9a0e65fe1fe21c1	0x5f139aa118fEE2f468b1f0515C66E3AB0192300a	49101	0	CONTRACT CALL
0xed5275518437c966ff773d08300b6ff26507c41a3790d7509a1cb2892cb7a42e	0x554b4e8f94066f6869408bd8f7f10954a87e4de1	0x5f139aa118fEE2f468b1f0515C66E3AB0192300a	64101	0	CONTRACT CALL
0xa404db01d7bd84122b54a26654ebff0147c352d795efe897afc4a5d9b613cec0	0xe873504792d98c24c4250d51505c536ac1f1b18c	0x5f139aa118fEE2f468b1f0515C66E3AB0192300a	64101	0	CONTRACT CALL
0xc8fdc8646d88a792f976b5d5aedb60de14a3e72054183f0ceb815c544b74b182	0xfAB64054C5ECaA1531d2B4C24e8D4bB850635DCe	0xc7b7fb6777047E06C4E4e5e5943E7f4C4A236adb	27008	0	CONTRACT CALL
0xc11a5ee9ae50c1daf7407b07c302bd3882aec16cb93295f7ae62dd13c1e30b9f	0xfAB64054C5ECaA1531d2B4C24e8D4bB850635DCe	0x5f139aa118fEE2f468b1f0515C66E3AB0192300a	469289	0	CONTRACT CREATION
0xe0953b45ddf92f2a0f58e8803928fe79c0315ff5bb2e260d7413f8297ac2f2b9					CONTRACT CALL

Widok 7. Lista transakcji

Program Ganache posiada zakładkę ‘Transakcje’. W widoku można przejrzeć wszystkie zawarte transakcje od początku działania aplikacji, skompilowanie i wdrożenie do blockchain smart kontraktu (oznaczony czerwonym statusem ‘CONTRACT CREATION’). Pierwsze 4 bloki posiadają ten sam adres należący do administratora-systemowego który rozpoczął proces budowy aplikacji i kompilacji smart kontraktu.

BLOCK	MINED ON	GAS USED	TRANSACTIONS
7	2018-12-15 15:12:09	49101	1 TRANSACTION
6	2018-12-15 15:10:39	64101	1 TRANSACTION
5	2018-12-15 15:01:40	64101	1 TRANSACTION
4	2018-12-15 14:50:16	27008	1 TRANSACTION
3	2018-12-15 14:50:16	469289	1 TRANSACTION
2	2018-12-15 14:50:15	42008	1 TRANSACTION
1	2018-12-15 14:50:15	277462	1 TRANSACTION
0	2018-12-15 14:49:55	0	NO TRANSACTIONS

Widok 8. Lista bloków

Program Ganache posiada zakładkę 'Bloki'. Zakładka przedstawia pełną listę utworzonych bloków od początku działania aplikacji. Dodatkowo, można wejść w szczegóły i zobaczyć serie zakodowanych napisów, gdyż właśnie tym są bloki, zakodowanymi długimi napisami połączonymi ze sobą w skutek transakcji.

## 4.2 Budowa Smart Kontraktu

```
pragma solidity ^0.4.24;

contract Election {
    // Model obiektu Kandydat
    struct Candidate {
        uint id;
        string name;
        uint voteCount;
    }

    // Przechowuje konta ktore zaglosowaly
    mapping(address => bool) public voters;

    //
    mapping(uint => Candidate) public candidates;
    // Przechowuje liczbe kandydatow
    uint public candidatesCount;

    // Zdarzenie zaglosowania
    event votedEvent (
        uint indexed _candidateId
    );

    function Election () public {
        addCandidate("Kandydat 1");
        addCandidate("Kandydat 2");
    }

    //Funkcja dodawania kandydatow
    function addCandidate (string _name) private {
        candidatesCount ++;
        candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
    }
}
```

### Widok 9. Smart kontrakt 1/2

Budowa smart kontraktu, zaczyna się od zadeklarowania wersji języka programowania `pragma solidity` stworzonego przez Ethereum. Następnie ciało kontraktu składa się z struktury obiektu i funkcji, które będą wykonywane do operacji związanych z głosowaniem. Komentarze krótko opisują zadania funkcji. Do napisania wykorzystana została stabilna wersja 0.4.24. Język przypomina `java script` zmodyfikowany na potrzebę tworzenia tylko i wyłącznie smart kontraktów. To dzięki niemu użytkownik i system zawierają umowę i zostaje to zapisane w blokach, powiązanych ze sobą. Podgląd rejestru bloków można zobaczyć w programie Ganache, co zostało pokazane w wcześniejszym rozdziale. To kontrakt buduje bazę danych i na jej podstawie przedstawiane są wartości w interfejsie aplikacji.

```

//Funkcja odpowiedzialna za oddanie glosu,
function vote (uint _candidateId) public {
    // wymaganie, ze konto nie zaglosowalo jeszcze
    require(!voters[msg.sender]);

    // walidacja, czy glosujemy na prawidlowych kandydatow
    require(_candidateId > 0 && _candidateId <= candidatesCount);

    // odnotowanie, ze glosujacy zaglosowal
    voters[msg.sender] = true;

    // aktualizacja ilosci glosow oddanych na kandydata
    candidates[_candidateId].voteCount ++;

    // trigger, do wydarzenia oddania glosu
    votedEvent(_candidateId);
}
}

```

#### Widok 10. Smart kontrakt 2/2

Główna funkcja odpowiedzialna za głosowanie. Składa się z:

- Wymaganie, że jeżeli konto nie będzie zalogowane to głosowanie pokaże błąd
- Sprawdzenie ilości kandydatów na podstawie ich wartości 'Id'
- Przekazanie informacji, że konto zagłosowało
- Aktualizacja ilości oddanych głosów na kandydata
- Wydarzenie inicjujące proces głosowania

W funkcji zostało wykorzystane wiele gotowych rozwiązań z bibliotek języka Pragma Solidity. Komentarze krótko opisują zadania funkcji w ciele głównej funkcji 'vote'. Kontrakt ściśle współpracuje z kodem źródłowym napisanym w Java Script.

### 4.3 Testy automatyczne

```
var Election = artifacts.require("../Election.sol");

contract("Election", function(accounts) {
  var electionInstance;

  it("Test0, sprawdza początkowa ilość kandydatów", function() {
    return Election.deployed().then(function(instance) {
      return instance.candidatesCount();
    }).then(function(count) {
      assert.equal(count, 2);
    });
  });

  it("Test1, sprawdza czy obiekt kandydat posiada prawidłowe wartości", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      return electionInstance.candidates(1);
    }).then(function(candidate) {
      assert.equal(candidate[0], 1, "zawiera prawidłowe id");
      assert.equal(candidate[1], "Kandydat 1", "zawiera prawidłową nazwę");
      assert.equal(candidate[2], 0, "zawiera prawidłową liczbę głosów");
      return electionInstance.candidates(2);
    }).then(function(candidate) {
      assert.equal(candidate[0], 2, "zawiera prawidłowe id");
      assert.equal(candidate[1], "Kandydat 2", "zawiera prawidłową nazwę");
      assert.equal(candidate[2], 0, "zawiera prawidłową liczbę głosów");
    });
  });
});
```

Widok 11. Testy 0-1

Widok przedstawia kod źródłowy testów automatycznych, które sprawdzają bezpieczeństwo i prawidłowe działania aplikacji. Test 0 sprawdza początkową ilość kandydatów biorących udział w głosowaniu. Wszystkie testy działają na zasadzie tworzenia nowej instancji kontraktu i sprawdza poszczególne funkcje. W pierwszym teście są sprawdzane podstawowe parametry dla głosownia. Każdy kolejny test zagłębia się w coraz bardziej szczegółowe scenariusze, które mogą doprowadzić do błędnych wyników.

Test 1 sprawdza wszystkie wartości obiektu 'Kandydat': numer id, nazwę i liczbę głosów. W dalszej części rozdziału zostanie zmodyfikowany kod, aby pokazać poprawne działanie testów w przypadku porażki i sukcesu. Testy korzystają głównie z funkcji 'equal' języka JavaScript, który porównuje wartość wyciągniętą z tablicy do wartości po przecinku. Trzeci element funkcji to komentarz wyświetlony w konsoli, naprowadzający administratora-systemowego opisem na błąd.

```

it("Test2, sprawdza czy glosujacy moze zaglosowac na kandydata", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    candidateId = 1;
    return electionInstance.vote(candidateId, { from: accounts[0] });
  }).then(function(receipt) {
    assert.equal(receipt.logs.length, 1, "wydarzenie zostalo zainicjowane");
    assert.equal(receipt.logs[0].event, "votedEvent", "typ wydarzenia jest poprawny");
    assert.equal(receipt.logs[0].args._candidateId.toNumber(), candidateId, "id kandydata jest poprawne");
    return electionInstance.voters(accounts[0]);
  }).then(function(voted) {
    assert(voted, "glosujacy zostal oznaczony jako uzytkownik ktory oddal glos");
    return electionInstance.candidates(candidateId);
  }).then(function(candidate) {
    var voteCount = candidate[2];
    assert.equal(voteCount, 1, "inkrementacja liczby glosow kandydata");
  })
});

it("Test3, pokazuje wyjatek dla nieprawidlowego kandydata", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    return electionInstance.vote(99, { from: accounts[1] });
  }).then(assert.fail).catch(function(error) {
    assert(error.message.indexOf('revert') >= 0, "komunikat bledu musi zawierac odwrocenie");
    return electionInstance.candidates(1);
  }).then(function(candidate1) {
    var voteCount = candidate1[2];
    assert.equal(voteCount, 1, "kandydat 1 nie otrzymal zadnych glosow");
    return electionInstance.candidates(2);
  }).then(function(candidate2) {
    var voteCount = candidate2[2];
    assert.equal(voteCount, 0, "kandydat 2 nie otrzymal zadnych glosow");
  })
});

```

## Widok 12. Test 2-3

Test 2 przeprowadza sprawdzenie funkcji odpowiedzialnej za głosowanie. Kandydat o id=1 z założenia na początku posiada ilość głosów równą 0. Administrator-systemowy korzysta z testowego konta z programu Ganache który oddaje głos na tego kandydata bez interfejsu użytkownika, na poziomie logów i sprawdza czy po zagłosowaniu ilość głosów jest równa 1. Dodatkowo sprawdza funkcję czy typ=event i czy po sformatowaniu wartości id będzie równe 1. Następnie test sprawdza, czy konto, które oddało głos zostało oznaczone, co jest bardzo ważnym punktem, gdyż głosowanie nie miało by sensu, gdy konta mogły by głosować bez końca. Ostatnim punktem testu 2 jest sprawdzenie poprawnej ilości głosów dla kandydata o id=1.

Test 3 sprawdza możliwość zagłosowania na kandydata o id=99 przez konto 1(konto przeznaczone do testów). Otrzymujemy komunikat błędu który jest oczekiwany przez test i odwrócony. Końcowy wynik to sukces, gdyż nie ma kandydata o id=99. Następnie test sprawdza, czy kandydaci mają odpowiednią ilość głosów po serii testów.



```

it("Test4, pokazuje wyjątek dla podwójnego głosowania przez tego samego głosującego", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    candidateId = 2;
    electionInstance.vote(candidateId, { from: accounts[1] });
    return electionInstance.candidates(candidateId);
  }).then(function(candidate) {
    var voteCount = candidate[2];
    assert.equal(voteCount, 1, "akceptacja pierwszego głosu");
    // Proba ponownego głosowania
    return electionInstance.vote(candidateId, { from: accounts[1] });
  }).then(assert.fail).catch(function(error) {
    assert(error.message.indexOf('revert') >= 0, "komunikat błędu musi zawierać odwołanie");
    return electionInstance.candidates(1);
  }).then(function(candidate1) {
    var voteCount = candidate1[2];
    assert.equal(voteCount, 1, "kandydat 1 nie otrzymał żadnych głosów");
    return electionInstance.candidates(2);
  }).then(function(candidate2) {
    var voteCount = candidate2[2];
    assert.equal(voteCount, 1, "kandydat 2 nie otrzymał żadnych głosów");
  });
});
});

```

### Widok 13. Test 4

Test 4 sprawdza proces podwójnego głosowania przez to samo konto. Początek testu to zagłosowanie przez konto testowe na kandydata o id=2. Po zaakceptowaniu głosu rozpoczyna się proces ponownego głosowania przez to samo konto. Spodziewamy się błędu, który przechwytyjemy i odwracamy na sukces. Następnie podobnie jak w teście 3 sprawdzamy ilości głosów dla kandydatów. Po wszystkich testach ilość głosów dla kandydatów powinna być równa 1, gdyż na potrzeby testów zostały oddane głosy po 1 na każdego. Te wartości są resetowane po kompilacji kontraktu.

```

$ truffle test
Using network 'development'.

Contract: Election
  ✓ Test0, sprawdza początkowa ilość kandydatów
  ✓ Test1, sprawdza czy obiekt kandydat posiada prawidłowe wartości (61ms)
  ✓ Test2, sprawdza czy głosujący może zagłosować na kandydata (183ms)
  ✓ Test3, pokazuje wyjątek dla nieprawidłowego kandydata (234ms)
  ✓ Test4, pokazuje wyjątek dla podwójnego głosowania przez tego samego głosującego (419ms)

5 passing (1s)

```

### Widok 14. Testy kontraktu sukces

Do przeprowadzenia testu kontraktu niezbędna jest jego kompilacja i praca programu Ganache w tle, gdyż testy opierają się na tworzeniu nowych instancji kontraktu, bloków na zdecentralizowanym serwerze. Serie testów automatycznych zaczyna się komendą 'truffle test'. Następnie kontrakt będzie poddany serii opisanych wyżej testów pod kątem wszelkich możliwych błędów. Przeprowadza ten proces administartor-systemowy aplikacji. Po sukcesie, można rozpocząć bezpieczne głosowanie.



```

truffle test
Using network 'development'.

Contract: Election
  ✓ Test0, sprawdza początkowa ilość kandydatów
  1) Test1, sprawdza czy obiekt kandydat posiada prawidłowe wartości
  > No events were emitted
  ✓ Test2, sprawdza czy głosujący może zagłosować na kandydata (168ms)
  ✓ Test3, pokazuje wyjątek dla nieprawidłowego kandydata (212ms)
  ✓ Test4, pokazuje wyjątek dla podwójnego głosowania przez tego samego głosującego (365ms)

4 passing (929ms)
1 failing

1) Contract: Election
   Test1, sprawdza czy obiekt kandydat posiada prawidłowe wartości:
  AssertionError: zawiera prawidłowa liczbe głosow: expected { Object (s, e, ...) } to equal 1
    at test\election.js:21:14
    at <anonymous>
    at process._tickCallback (internal/process/next_tick.js:188:7)

```

#### Widok 15. Testy kontraktu porażka

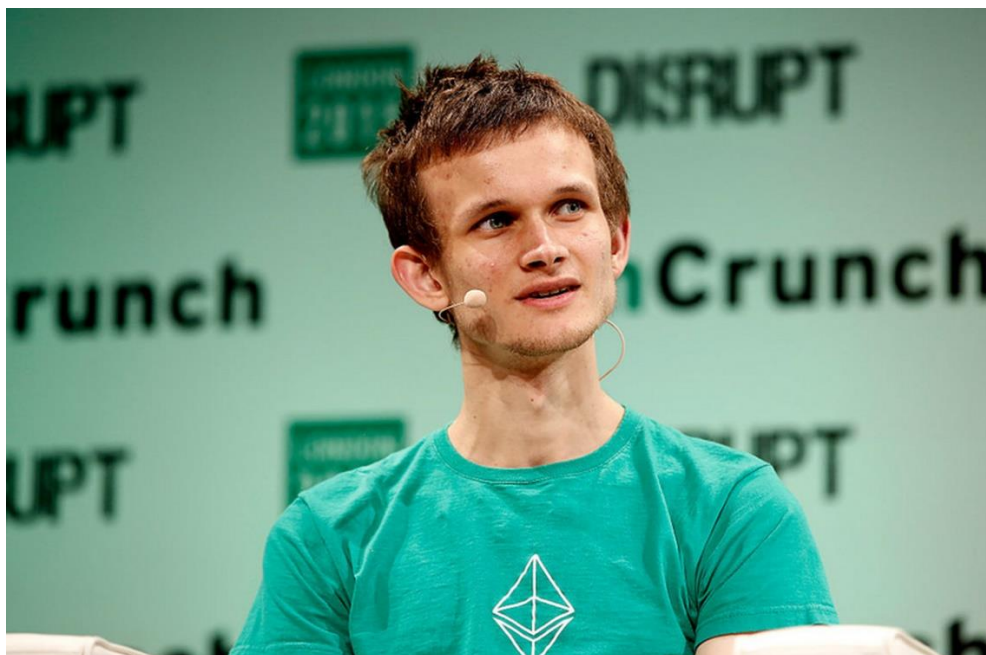
W teście 1 została zmieniona wartość, przy sprawdzeniu poprawnych wartości kandydata, konkretnie czy wartość początkowa, ilości oddanych głosów równa się 0. Ta wartość została zmieniona na poziomie kodu źródłowego na 1. Test 1 pokazał błąd wskazując wartość i wyświetlając odpowiedni komentarz opisujący błąd. Administrator-systemowy w takiej sytuacji sprawdza kontrakt i nie rozpoczyna głosowania, gdyż jej poprawność nie miała by miejsca. Dla zdecentralizowanych aplikacji, testy przed rozpoczęciem działania są niezwykle ważne, gdyż blockchain nie jest do edycji. Raz zapisany błędny blok zostaje. Następnie należało by podprawić błąd i zresetować kontrakt razem z strukturą bloków. Taka sytuacja jest rzadko spotykana. W historii życia platformy Ethereum nastąpił jeden włam. Została wykradziona niewielka część walut użytkowników. Podzieliło to społeczność i deweloperów. Część chciała kontynuować bez resetu na obecnym kodzie i stanie blockchain, a druga do której należy twórca Vitalik Buterin zdecydował że odda użytkownikom skradzione kryptowaluty i zmodyfikuje kod kontraktu. Nastąpił po tym wydarzeniu nowy branch kodów źródłowych i tak powstał Ethereum Classic.

## 5. Historia kryptowaluty na rynku i twórca platformy Ethereum

Ethereum nazywane jest czasem "bitcoinem 2.0" i faktycznie jako kryptowaluta stanowi alternatywę dla bitcoina. Od kilku miesięcy Ethereum notuje też spore wyżki kursu, co czyni z niej dość intratną inwestycję. Jeszcze w grudniu 2016 roku za ethereum płacono 7–8 dolarów, ale w styczniu zaczął się powolny wzrost kryptowaluty. Cena rosła stabilnie – 9, 10, potem 11 dolarów, i tak do końcówki lutego. Wtedy dopiero kurs ethereum wystrzelił, niemal każdego kolejnego dnia bijąc rekordy. 12 marca po raz pierwszy ta kryptowaluta przebiła barierę 20 dol. i w ciągu kilkunastu następnych dni zaliczyła prawdziwy skok do ceny 45 dolarów. Kolejnym kamieniem milowym był magiczny poziom 50 dolarów, przebity 24 marca. Od tamtej pory tylko podczas kilku dni w kwietniu cena ethereum spadała poniżej 45 dolarów. To zaś oznacza wzrost o 273,7 proc. w ciągu ostatnich trzech miesięcy i 350 proc. w 2 miesiące. Tak wygląda wykres Ethereum od jej powstania. Dzięki skokom, które nastąpiły w ostatnich miesiącach, kapitalizacja ethereum to już 4,5 miliarda dolarów. Oczywiście nijak ma się to do bitcoina, który niedawno zrównał się poziomem cen ze złotem i pomimo późniejszych spadków nadal jego wycena jest kilka razy wyższa niż Ethereum. Skąd taki wzrost popularności Ethereum, gdy zdaje się, że Bitcoin święci tryumfy? Paradoksalnie to właśnie problemy Bitcoina były jednym z powodów wzrostu jego konkurenta. Takie problemy mogą ograniczyć ekspansję sieci. Społeczność kryptowalutowa obecnie pracuje nad rozwiązaniem tej sprawy, z związku z czym Ethereum wydaje się efektywniejszą i bardziej skalowalną alternatywą. Większość aktywów, które przyczyniły się do wzrostu cenowego w marcu, pochodziła wprost z inwestycji bitcoinowych. Poza tym jednak, są dwa główne czynniki wzrostu Ethereum. Pierwszym jest bez wątpienia projekt Hyperledger, który zrzesza ponad 100 najważniejszych światowych instytucji finansowych i ma na celu stworzenie globalnego systemu rozliczeń walutowych. Rozważane są opcje, aby projekt bazował na systemie blockchain opartym właśnie na Ethereum. Drugi czynnik przez niego wymieniony to aktualizacja systemu Casper, który ma umożliwić szybsze i sprawniejsze rozliczenia w tej walucie.

Obecnie rynek kryptowalut charakteryzuje się skrajnie wysokim poziomem ryzyka. Z pewnością w przyszłości zobaczymy wzrost ceny Etherum, natomiast warto pamiętać o historii notowań kryptowalut. W marcu 2013 roku cena Bitcoina wzrosła z 3 dolarów do 126 dolarów zaledwie w ciągu jednego tygodnia. Później inwestorzy byli świadkami zwyżek do 1200 dolarów, a następnie spadków do poziomu 200 dolarów. Pamiętajmy jednak, iż to właśnie Bitcoin jest pionierem na rynku kryptowalut. Jest często określany jako cyfrowy zamiennik złota i nie szybko odejdzie w niepamięć.

Ostatnio jesteśmy świadkami zmian legislacyjnych, które mają na celu integrację kryptowalut i to Bitcoin jest ich głównym motorem. Pomimo, że Etherum zyskuje na uwadze, musi pokonać naprawdę długą drogę, zanim stanie się pełnoprawnym konkurentem bitcoina. Ethereum to nie tylko kryptowaluta, ale cała platforma oparta na blockchain, pozwalająca nie tylko na inwestowanie, ale też zawieranie zakładów czy wspieranie crowdfundingu. Nie mam wątpliwości, że sieć Etherum bardzo szybko rośnie w siłę oraz ma duży potencjał. Warto wspomnieć, iż ostatnio Storj, jedna z największych firm zajmujących się technologią blockchain zapowiedziała, że przeprogramuje swój system gromadzenia danych z Bitcoina na Etherum[11].



Rys. 5 Vitalik Buterin

Cała technologia została stworzona przez 22-letniego programistę Vitalika Buterina z Rosji. Buterin urodził się w Kolomnie, w obwodzie moskiewskim w Rosji, syn Dmitrija Buterina, informatyka i Natalii Ameline (z domu Chistyakova). Mieszkał w okolicy do szóstego roku życia, kiedy jego rodzice wyemigrowali do Kanady w poszukiwaniu lepszych możliwości zatrudnienia. Podczas gdy w trzeciej klasie szkoły podstawowej w Kanadzie, Buterin został umieszczony w klasie dla uzdolnionych dzieci i zaczął rozumieć, że miał predyspozycję do matematyki, programowania i ekonomii. Miał także umiejętność do dodawania trzycyfrowych liczb w swojej głowie dwukrotnie szybciej niż jego rówieśnicy.

Buterin uczęszczał do Abelard School, prywatnego liceum w Toronto, przez cztery lata, które, jak powiedział, "okazały się jednym z najciekawszych i najbardziej produktywnych lat mojego życia: bliższy związek między uczniami i nauczycielami, sprawił, że chcę się uczyć i skoncentrować na uczeniu się jako moim głównym celem". Buterin napisał, że chociaż "nigdy nie był szczególnie zainspirowany tradycyjnym systemem edukacji", gdy tematy były prezentowane z "poświęceniem i skupieniem na intelektualnym śledztwie", powiedział: "Zauważyłem, że moje nastawienie i moje wyniki niemal natychmiast ulegają drastycznej zmianie, a wykształcenie jest czymś więcej niż tylko zapamiętywaniem pojedynczych faktów, a nawet uczeniem się poszczególnych pojęć. Co jest najważniejsze: nauka myślenia, nauka rozumowania i nauka uczenia się."

Buterin dowiedział się o bitcoinie od ojca w wieku 17 lat. W 2012 roku zdobył brązowy medal na Międzynarodowej Olimpiadzie Informatycznej. W 2013 r. Odwiedził programistów w innych krajach, którzy podzielali jego entuzjazm dla kodu. Wrócił do Toronto jeszcze w tym roku i opublikował białą księgę, w której zaproponował Ethereum. Uczęszczał na University of Waterloo, ale zrezygnował z niego w 2014 r., Kiedy otrzymał stypendium Thiela w wysokości 100 000 \$ i rozpoczął pracę w Ethereum w pełnym wymiarze godzin. W dniu 25 czerwca 2017 r. Buterin był przedmiotem oszustwa śmierci pochodzącego z 4chan (serwis internetowy z obrazami). W dniu 30 listopada 2018 r. Vitalik Buterin otrzymał honorowy doktorat Wydziału Biznesu i Ekonomii Uniwersytetu w Bazylei z okazji Dies Academicus [10].

## 6. Instrukcja instalacji

Lista kroków do zainstalowania i przetestowania zdecentralizowanej aplikacji 'Głos obywatela' (niezbędne jest połączenie z Internetem):

1. Przekopiować zawartość płyty CD dołączonej do pracy inżynierskiej na dysk lokalny.
2. W folderze 'programy do zainstalowania' znajduje się instalator przeglądarki Google chrom, należy ją zainstalować.
3. W folderze 'programy do zainstalowania' znajduje się program Ganache, należy go zainstalować.
4. Otwieramy przeglądarkę Google, wyszukujemy i instalujemy rozszerzenie Metamask.
5. Przechodzimy do folderu z kodem źródłowym i otwieramy konsolę.
6. Wpisujemy komendę 'brew install node'.
7. Wpisujemy komendę 'npm install -g truffle'.
8. Wpisujemy komendę 'npm install chai'.
8. Uruchamiamy program Ganache.
9. W oknie komend na poziomie kodu źródłowego wpisujemy komendę 'truffle migrate --reset'.
10. Następnie komendę 'npm run dev'.
11. Pokaże się nam Google Chrom z stroną uruchomioną na lokalnym środowisku w stanie ładowania.
12. Tworzymy konto w rozszerzeniu Metamask zgodnie z instrukcją w obecnej wersji.
13. W programie Metamask tworzymy swoją sieć RPC i wprowadzamy adres 'http://localhost:7545' i zapis.
14. Na koniec tworzymy konto użytkownika aplikacji po przez import klucza prywatnego z programu Ganache klikając w ikonę klucza, kopiujemy ją a w programie Metamask wklejamy w pole import konta za pomocą klucza prywatnego.
15. Zalogowany użytkownik zobaczy widok aplikacji i może zrealizować główną funkcjonalność aplikacji.

## 7. Podsumowanie

Stworzony został zdecentralizowany, anonimowy i jawny rejestr zastosowany do aplikacji, w której użytkownik głosuje na swojego kandydata. Bardzo ważnym punktem pracy są testy automatyczne. Zaprogramowane w języku Java Script. Sprawdzają one smart kontrakt i działanie głównych mechanizmów aplikacji, pod kątem poprawności struktury, wartości stworzonego obiektu, następnie funkcje i różne scenariusze zachowania użytkownika. Kolejne testy sprawdzają integrację aplikacji z usługami zewnętrznymi, takimi jak rozszerzenie Metamask.

Aplikację można rozwinąć o nowe funkcjonalności, takie jak głosowanie z większą liczbą kategorii. Rozbudować smart kontrakt o dowolną ilość kandydatów i innych pozycji do przegłosowania przez obywateli. Dodatkowo, w takiej sytuacji należałoby stworzyć nowe testy sprawdzające poprawności i zmodyfikować obecne. Upiększyć w dowolny sposób, za pomocą gotowych bibliotek, ale nie wygląd graficzny był celem aplikacji, lecz konceptualne ukazanie technologii. Istnieje także możliwość modyfikacji aplikacji, aby była ona wykorzystana do faktycznego głosowania na głównej sieci Ethereum, wiąże się to z wykrzesaniem waluty, która posiada realną wartość, co by się wiązało z inwestycją w skali kraju. Dlatego warto opracować swoje technologie. Im wyższa cena waluty Ethereum tym droższy ruch w sieci. Najlepszym rozwiązaniem byłoby stworzenie narodowej kryptowaluty do wymiany lub tylko głosowania. Projekt 'krypto złotego' został już przygotowany, lecz nie wszedł jeszcze w życie w momencie pisanie pracy.

Aplikacja ma szerokie zastosowanie w głosowaniu na swobody obywatelskie, wszelkie wybory na kandydatów i ankiety. Dzięki użytej technologii głosowanie jest odporne na wszelkie manipulacje jego wynikiem, a to jest podstawowy problem przy procesie głosowania. W przyszłości wszystko będzie się odbywać internetowo. Przedstawione narzędzie właśnie do tego służy i za parę lat będzie powszechnie wykorzystywane. Technologia posiada wady i nie nadaje do przechowywania dużych ilości danych typu muzyka, obrazy, wielkie tabele z milionami rekordów przemnożonymi przez kolumny. Do tych typów jeszcze przez lata zostanie wykorzystywany zcentralizowany system serwerów, czyli jeden serwer i tworzone kopie zapasowe. Zaletą wybranej technologii jest jej rozproszenie, odporność na manipulacje i przejście danych. Dlatego aplikacje finansowe i głosowania jako pierwsze zaadaptują tę technologię i sukcesywnie będą z niej czerpać korzyści.

Technologia ta jest coraz popularniejsza, a razem z zapotrzebowaniem na nią rośnie zapotrzebowanie na deweloperów specjalizujących się w zdecentralizowanym podejściu do tworzenia oprogramowania.

Udało się stworzyć aplikację na testowym środowisku wirtualnym dostarczonym przez Ethereum. W dalszej perspektywie taka aplikacja będzie działać w głównej sieci i tam za prowizją, która jest realnym kosztem będzie w pełni funkcjonalna jak na sieci wirtualnej. Dlatego, tak ważne jest aby kraj zainwestował w stworzenie swojej kryptowaluty, aby uniknąć kosztów w dalszej perspektywie i wykorzystać stworzoną technologię do administracji, sądownictwa, finansów, gier itp. Prawdopodobnym obrotem sytuacji, na przestrzeni kilku lat jest koniec pieniądza 'polski złoty' na korzyść 'krypto złotego'. Jego przykładowe zalety to niskie koszty produkcji, utylizacji, transferu, czasu transferu, bezpieczeństwo przechowywania i kontrola przez państwo.

## 8. Streszczenie

Praca prezentuje zdecentralizowaną aplikację webową pod nazwą roboczą ‘Głos obywatela’. Celem jest zaprezentowanie technologii związanych z nowym typem aplikacji webowych. Głównym aspektem jest decentralizacja serwera. Pozwala ona zwiększyć poziom bezpieczeństwa, co w wypadku aplikacji do głosowania jest najważniejsze. Aplikacja wykorzystuje program Ganache do stworzenia zdecentralizowanego serwera wirtualnego i dostarcza pulę kont z wirtualną walutą (Ethereum) niezbędną do realizacji kontraktów w sieci. Klucze prywatne z programu Ganache są wykorzystywane do importu konta w usłudze Metamask. Zalogowanego użytkownika system przepuści do głównego widoku aplikacji i będzie miał możliwość przejścia głównej ścieżki funkcji głosowania. Aplikacja jest wyposażona w serię testów automatycznych sprawdzających poprawność obiektów, smart kontraktu i weryfikuje poprawność przebiegu wszystkich funkcji. Wybrane technologie i rozwiązania mają za zadanie pokazać i przekonać czytelnika, że aplikacja do głosowania powinna być tworzona przy pomocy specjalnych narzędzi programistycznych zastosowanych w pracy.

## 9. Abstract

The work presents a decentralized web application under the working name 'Głos obywatela'. The aim is to present technologies related to the new type of web applications. The main aspect is server decentralization. It allows you to increase the level of security, which is most important in the case of a voting application. The application uses the Ganache program to create a decentralized virtual server and provides pools of accounts with a virtual currency (Ethereum) necessary to carry out contracts on the network. Private keys from the Ganache program are used to import an account in the Metamask service. The logged-in user will pass the system to the main application view and will have the option of passing the main voting path. The application is equipped with a series of automatic tests that verify the correctness of objects, a smart contract and verify the correctness of the course of all functions. Selected technologies and solutions are designed to show and convince the reader that the voting application should be created using special programming tools used in the work.

## 10. Literatura i źródła

- [1] <https://gapper-agencja.pl/blog/aplikacje-internetowe-webowe-czym-sa> (1.12.2018)
- [2] [http://www.hostingfirmowy.pl/Pomoc/Servery/Informacje\\_ogolne](http://www.hostingfirmowy.pl/Pomoc/Servery/Informacje_ogolne) (1.12.2018)
- [3] <https://pl.wikipedia.org/wiki/Kryptowaluta> (1.12.2018)
- [4] <https://bithub.pl/artykuly/czym-smart-contract/> (2.12.2018)
- [5] <https://truffleframework.com/tutorials> (8.12.2018)
- [6] <http://www.dappuniversity.com/articles/code-your-own-cryptocurrency-on-ethereum> (8.12.2018)
- [7] <https://www.chip.pl/2017/12/cryptokitties-gra-korzystajaca-ethereum/> (8.12.2018)
- [8] <https://www.w3schools.com/js/> (8.12.2018)
- [9] <https://www.ethereum.org/> (9.12.2018)
- [10] [https://en.wikipedia.org/wiki/Vitalik\\_Buterin](https://en.wikipedia.org/wiki/Vitalik_Buterin) (15.12.2018)
- [11] <https://businessinsider.com.pl/finanse/kryptowaluty/ethereum-co-to-jest-kryptowaluta-i-platforma> (15.12.2018)