

Akademia Finansów i Biznesu Vistula

Wydział: Sztuki, Techniki i Komunikacji

Kierunek studiów: Informatyka

Jakub Pawłowski

Numer albumu 49885

Ethereum jako platforma dla aplikacji zdecentralizowanych

Praca inżynierska
napisana pod kierunkiem
dr. hab. Maciej Janowicz

Warszawa 2022

Spis treści

WSTĘP	4
ROZDZIAŁ I.....	5
1. Ogólna charakterystyka systemów Blockchain	5
1.1 Zastosowanie technologii blockchain	6
1.2 Mechanizm działania architektury Blockchain	7
1.3 Stosowana kryptografia.....	9
1.4 Podział modeli sieci Blockchain	15
1.5 Portfel kryptowalutowy.....	16
1.6 Coin vs Token	21
ROZDZIAŁ II.....	21
2. Blockchain Ethereum	21
2.1 Typy kont w Ethereum.....	22
2.2 Wyznaczanie adresu konta	22
2.3 Transakcje na Ethereum	23
2.4 EVM – Ethereum Virtual Machine	28
2.5 Węzły oraz klienci.....	29
2.6 Algorytm konsensusu	31
2.7 Charakterystyka wykopanych bloków	33
2.8 Standardy ERC.....	34
ROZDZIAŁ III	36
3. Inteligentne kontrakty.....	36
3.1 Ulepszanie kontraktów	37
ROZDZIAŁ IV	39
4. Ewolucja Web 3.0	39
4.1 Web 1.0 vs Web 2.0 vs Web 3.0	39

4.2 Wy tłumaczenie aplikacji zdecentralizowanych w Web 3.0.....	41
ROZDZIAŁ V.....	42
5. Aplikacja kontraktowa – protokół imitujący lokatę.....	42
5.2 Język do pisania kontraktów - Solidity.....	43
5.3 Narzędzie do testowania i wdrażania kontraktów - Brownie	44
5.4 Osobisty blockchain Ethereum - Ganache	44
5.5 Wyrocznie – zdecentralizowane źródła danych.....	45
5.6 Mock – symulator prawdziwych obiektów.....	45
5.7 Testy jednostkowe oraz testy integracyjne	46
ROZDZIAŁ VI.....	46
6. Uruchomienie projektu.....	46
6.1 Instalacja oraz konfiguracja środowiska edytora kodu - Windows	47
6.2 Instalacja Brownie oraz pozostałych komponentów.....	49
6.3 Zalecane polecenia do użycia w projekcie.....	50
PODSUMOWANIE	62
BIBLIOGRAFIA.....	65
Artykuły oraz inne źródła internetowe.....	65

WSTĘP

Branża Blockchain¹ z kryptowalutami na czele przeżywa swoje wzloty i upadki. Głównie za sprawą szalejącej ceny aktualnie największego pod względem kapitalizacji oraz rozpoznawalności Bitcoina². Wiąże to się z cyklami hossy³ oraz bessy³ na rynkach finansowych. Hossa, czyli rynek byka – wzrostowy, winduje cenę Bitcoina do góry. Media na całym świecie rozpisują się o tym fenomenie. Powstaje FOMO⁴, ludzie czują strach, że okazja inwestycyjna ucieka im sprzed nosa. To napędza jeszcze bardziej jego cenę. Ludzie wpadają w euforie i bardzo często kupują po zawyżonej cenie. Bez podstawowej znajomości rynkowej i wskutek zbytnej pewności siebie omijają dogodną okazję na realizację zysków. Następnie pojawia się solidne trzęsienie na rynku w postaci Bessy, czyli rynku niedźwiedzia - spadkowego, cena momentalnie leci w dół. Wiele ludzi po takim doświadczeniu w panice sprzedaje swoje aktywa i ostatecznie jest na stracie. Temat ucicha i wracamy do punktu wyjścia. Tak w skrócie przedstawia się sytuacja co około 4 lata.

Moim głównym celem pracy jest przedstawienie technologii Blockchain od strony teoretycznej oraz technologicznej. To coś o wiele większego niż sam Bitcoin. To dosłownie nowy system finansów odporny na cenzurę, puste drukowanie pieniądza przez banki centralne czy możliwość omijania wszelkiej maści pośredników. To nowa szansa na uzdrowienie systemu finansowego poprzez wszechobecną transparentność.

Oprócz wielkich szaleństw i emocji w czasie rynku wzrostowego są ludzie, przedsiębiorcy, którzy starają się budować rzeczy zmieniające świat. Wbrew pozorom to właśnie w rynku niedźwiedzia powstaje najwięcej wartościowych produktów. Jestem wielkim zwolennikiem ekosystemu Ethereum⁵ dlatego postanowiłem w szczegółowy sposób omówić jego najważniejsze cechy.

¹ Adam Hayes, Blockchain Facts: What is it, How it works, and How it can be used, <https://www.investopedia.com/terms/b/blockchain.asp>

² Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, <https://bitcoin.org/en/bitcoin-paper>

³ Hossa/Bessa długotrwały trend wzrostowy/spadkowy, <https://szybkagotowka.pl/Wiki/hossa-bessa>

⁴ Kate Brush, FOMO (fear of missing out), <https://www.techtarget.com/whatis/definition/FOMO-fear-of-missing-out>

⁵ Vitalik Buterin, Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform, <https://ethereum.org/en/whitepaper/>

ROZDZIAŁ I

1. Ogólna charakterystyka systemów Blockchain

Pierwsze użycie technologii Blockchain miało miejsce w 2009 roku za sprawą opublikowania najpopularniejszej dziś kryptowaluty o nazwie Bitcoin przez tajemniczego twórcę o pseudonimie Satoshi Nakamoto.

Wydarzenie zapoczątkowało trend cyfrowego pieniądza opartego na kryptografii⁶, który miał być rozproszony, zdecentralizowany, nie posiadający centralnego administratora nad sobą. Jego zastosowaniem miały być transfery środków do dowolnego miejsca na świecie z pominięciem zaufanej trzeciej strony w postaci regulatorów, banków czy rządów państw.

Satoshi Nakamoto stworzył Bitcoina ze względu na wielki kryzys finansowy, który miał miejsce w 2008 r. w wyniku, którego upadł jeden z największych banków inwestycyjnych Lehman Brothers⁷. Chciał, aby zwykli ludzie mogli odzyskać kontrolę nad swoimi finansami, przejętą przez elity polityczne i bankowe, dając im szansę wzięcia udziału w zdecentralizowanym systemie, bez udziału odgórnego administratora. Władzę nad nim mieli sprawować jego użytkownicy rozproszeni po całym świecie.

Kluczowa cecha Bitcoina jest jego maksymalna kapitalizacja w ilości 21 milionów sztuk⁸, które zostaną „wykopane” do roku 2140. Od momentu wykopania pierwszego bloku przez jego twórcę występuje zjawisko tzw. **halvingu**⁹, które zmniejsza jego inflację o połowę, aktualnie jest ona na poziomie 1.8% rocznie¹⁰.

Jest to zupełnie inne podejście niż ma to miejsce obecnie w bankach centralnych. Aktualnie **fiat** (PLN, USD, GBP, EURO, etc.), czyli pieniądz fiducjarny¹¹ jest oparty tylko i wyłącznie na wierze w państwa, które za nimi stoją. Jeszcze w ubiegłym wieku w USA dolar miał zabezpieczenie w złocie, ale za kadencji prezydenta Richard’a Nixon’a

⁶ Jakub Pawłowski, 1.3 Stosowana kryptografia, Ethereum jako platforma dla aplikacji zdecentralizowanych

⁷ Nick Lioudis, The Collapse of Lehman Brothers: A Case Study, <https://www.investopedia.com/articles/economics/09/lehman-brothers-collapse.asp>

⁸ 5 Features That Make a Bitcoin A Unique Asset Class, <https://argoblockchain.com/articles/5-features-that-make-bitcoin-a-unique-asset-class>

⁹ Matt Whittaker, What is Bitcoin Halving, <https://www.forbes.com/advisor/investing/cryptocurrency/bitcoin-halving/>

¹⁰ Bitcoin Inflation, <https://charts.woobull.com/bitcoin-inflation/>

¹¹ Weronika Grzywna, Czym jest pieniądz fiducjarny i jaką ma wartość, <https://www.money.pl/pieniadze/czym-jest-pieniadz-fiducjarny-i-jaka-ma-wartosc-6748241007573824a.html>

15 sierpnia 1971 roku¹², wymienialność dolara na złoto została zawieszona i już nigdy nie została wznowiona. Aktualnie od 45 lat żadna waluta narodowa nie posiada zabezpieczenia w kruszcu, a banki centralne posiadają praktycznie nieograniczone możliwości w kwestii drukowania pieniędzy. Dla zobrazowania skali problemu między początkiem stycznia 2020 roku a końcem października 2021 roku – Stany Zjednoczone wydrukowały ponad **80% wszystkich istniejących dolarów**¹³. Trudno to sobie wyobrazić, biorąc pod uwagę fakt, że historia amerykańskiego dolara sięga roku 1792.

1.1 Zastosowanie technologii blockchain

Technologia blockchain daje nam wiele korzyści w walce z cenzurą. Umożliwia walkę z aparatem państwa w skrajnych sytuacjach, jak to się stało w Kanadzie w lutym 2022 roku, podczas protestów kierowców ciężarówek¹⁴. Mimo iż kraj ten góruje w większości zestawień odnoszących się do poziomu wolności, rząd zdecydował się na zamrożenie kont bankowych ludzi, którzy brali w nim udział.

Protestujący zbierali finansowanie na kontynuowanie swoich działań, dlatego w pierwszej kolejności zaczęli korzystać z usług tradycyjnych, scentralizowanych firm tj. (GoFundMe ~ zebrało \$4 miliony, GiveSendGo ~ zebrało \$8.5 milionów), które umożliwiały zbiórkę środków. Wkrótce okazało się jednak, że rząd Kanady z łatwością jest w stanie wpłynąć na administratorów i całkowicie anulować wyżej wspomniane zbiórki.

W tym momencie przechodzimy do wręcz historycznego momentu dla Bitcoina w zakresie walki z „finansową cenzurą”. **Tallycoin** – platforma do pozyskiwania funduszy oparta o Bitcoina, umożliwiła przekazywanie darowizn w postaci kryptoaktywów bezpośrednio do portfela zbiórki na rzecz protestujących.

Rząd Kanady również i wtedy był przekonany, że poradzi sobie bez problemu, ale okazało się, że nie jest w stanie nic zrobić. Wynikło to z tego, że twórcy platformy **Tallycoin** nie przetrzymują żadnych danych odnośnie do ludzi wpłacających z wyjątkiem adresów e-mail. Z tego powodu nie są w stanie zablokować użytkowników, nie mogą

¹² Krzysztof Kolany, 45 lat temu Nixon zamknął „złote okno”, <https://www.bankier.pl/wiadomosc/45-lat-temu-Nixon-zamknal-zlote-okno-7473819.html>

¹³ Daniel Levi, 80% of all US dollars in existence were printed in the last 22 months, <https://techstartups.com/2021/12/18/80-us-dollars-existence-printed-january-2020-october-2021/>

¹⁴ Peter Chawaga, How Bitcoin fueled the freedom convoy and defied government crackdown, <https://bitcoinmagazine.com/culture/how-bitcoin-fueled-canada-trucker-convoy>

również zabronić im transferować wpłaconych środków, w skrócie nie są w stanie tego zatrzymać.

Ten piękny przykład pokazuje prawdziwą moc jaką dają nam kryptoaktywa w zachowaniu wolności czy anonimowości. W czasach, gdy państwo ma możliwość zablokowania całego majątku posiadanego w banku za pomocą jednego przycisku, myślę, że konieczna jest chociaż podstawowa znajomość tych systemów.

W tym miejscu warto podkreślić, że sieci Blockchain tj. Ethereum czy Bitcoin działają 24 godziny na dobę, 7 dni w tygodniu. Mimo swojej aktualnie niedużej przepustowości funkcjonują cały czas. Dlatego z bardzo dużym prawdopodobieństwem możemy być pewni, że nasz przelew dojdzie do skutku w przeciągu kilku minut. Wszystko oczywiście zależy od obciążenia sieci, ale plusem weekendów oraz dni wolnych od pracy jest właśnie jej małe obciążenie.

Nie ma znaczenia również kwota transferu. Opłata transakcyjna będzie taka sama dla transferu \$10 czy \$100 milionów. Ma to swoje plusy i minusy. Plusem jest to, że możemy operować setkami milionów bez zbędnych pytań od strony banków, płacąc jednocześnie niewielkie sumy w wysokości maksymalnie kilku dolarów. W tradycyjnej bankowości coś takiego jest niemożliwe. Zawsze pojawiają się pytania oraz solidna prowizja do zapłaty. Dla dużych graczy opłata transakcyjna nie stanowi żadnego problemu. Jednakże konsument detaliczny nie może sobie pozwolić na opłatę transakcyjną przekraczającą wielokrotnie cenę towaru, za który chce zapłacić.

Oczywiście powstały sieci, które to umożliwiają przy minimalnych opłatach, ale w tej pracy skupiam się wyłącznie nad Ethereum oraz Bitcoinem.

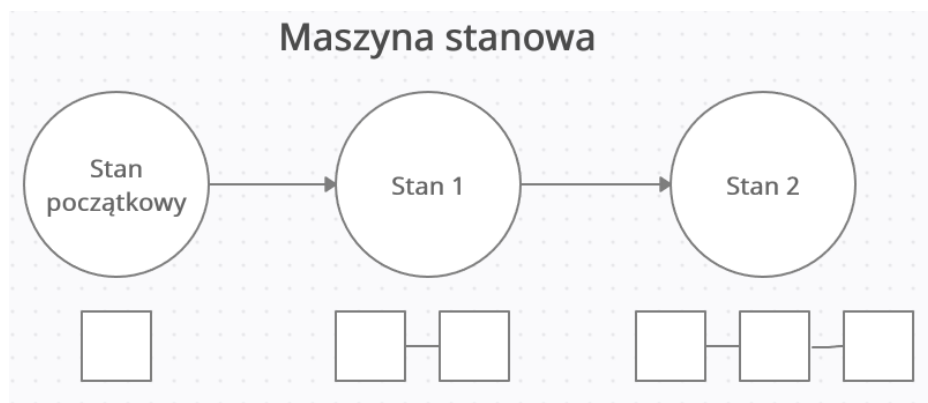
1.2 Mechanizm działania architektury Blockchain

Blockchain jest publiczną formą bazy danych, która jest rozproszona w licznej sieci komputerów, nazywanych węzłami¹⁵. Jest rozproszony, nie występuje tutaj problem pojedynczego punktu awarii. Jeżeli nam „padnie” maszyna, to te same dane są przechowywane jeszcze na wielu innych. Są one zatem bardzo bezpieczne.

„Block” to miejsce, w którym zapisywane są transakcje. Kolejne bloki są dołączane do ich łańcucha – czyli Blockchainu – w różnych odstępach czasowych. W sieci

¹⁵ Jakub Pawłowski, 2.5 Węzły oraz klienci, Ethereum jako platforma dla aplikacji zdecentralizowanych

Bitcoin odstęp ten to około 10 minut¹⁶, natomiast w Ethereum już średnio między 12 a 14 sekund¹⁷. „Blockchain” nazywa się właśnie tak – łańcuchem bloków - ponieważ jest odporny na modyfikacje danych w tym sensie, że transakcje, które miały miejsce w poprzednich blokach nie ulegają zmianie ze względu na zastosowanie kryptografii¹⁸. Blockchain możemy rozumieć jako **maszynę stanową** (state machine), w której jest jakiś **stan początkowy** (Genesis State) i stan zmienia się tylko w momencie dodania nowego bloku.



Rysunek 1: Maszyna stanowa sieci blockchain

Źródło: Opracowanie własne

Bloki zawierają transakcje, które można rozumieć (w przypadku Bitcoina – pierwszego blockchaina) jako przelewy kryptowalut z jednego konta na drugie. Stan aktualny można rozumieć jako zbiór sald wszystkich kont. Stan zmienia się tylko w momencie dołączenia nowego bloku.

Blok 30 liczy funkcję skrótu¹⁹ ze wszystkich danych, które się w nim znajdują. Składają się na to (i) zgromadzone transakcje, (ii) **Timestamp** (znacznik czasu – liczba sekund, które upłynęły od początku „epoki Uniksa”. „Epoka Uniksa”²⁰ rozpoczęła się 1 stycznia 1970 r.) (iii) **Nonce** – licznik, (iv) **Prev Hash** – (wynik funkcji skrótu z poprzedniego bloku). Wynik funkcji skrótu zapisujemy w nagłówku bloku kolejnego – nagłówek bloku 31 – itd.

¹⁶ Prabath Siriwardena, The Mystery Behind Block Time, <https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a?gi=16c0522a3050>

¹⁷ Blocks, <https://ethereum.org/en/developers/docs/blocks/>

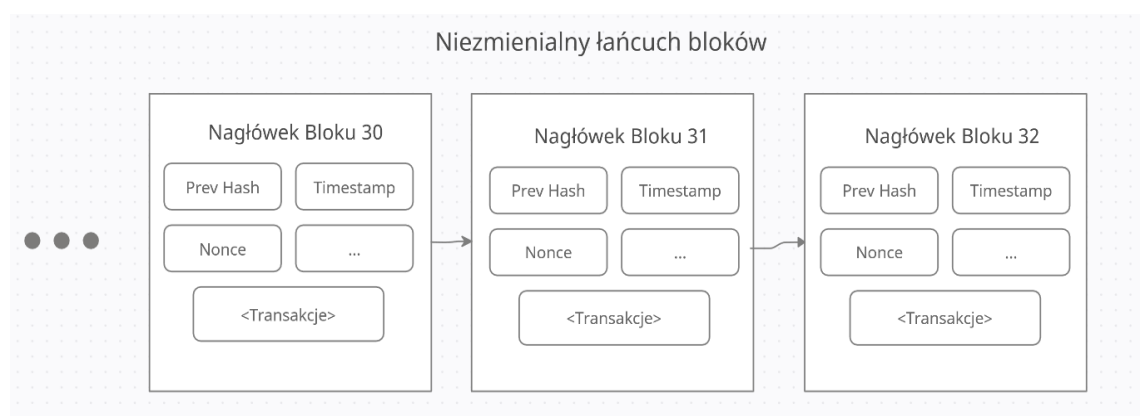
¹⁸ What Makes a Blockchain Secure, <https://academy.binance.com/en/articles/what-makes-a-blockchain-secure>

¹⁹ What is hashing, <https://academy.binance.com/en/articles/what-is-hashing>

²⁰ What is epoch time, <https://www.epochconverter.com/>

Przypuśćmy, że ktoś chciał zmienić jakieś dane, i zmienił chociaż jedną transakcję w bloku 31. Spowoduje, że zmieni się cały blok 31 (funkcja skrótu z tego bloku, będzie miała inny wynik), ponieważ dowolnie mała zmiana daje zupełnie inny wynik, który jest zapisany w nagłówku bloku kolejnego.

Widzimy tutaj kaskadę zależności – tak naprawdę wynik funkcji skrótu najnowszego bloku świadczy o wszystkich danych, jakie zostały kiedykolwiek dodane do blockchaina. Jest to tak bardzo silna zależność, która sprawia, że nie musimy posiadać wszystkich danych lokalnie, żeby mieć pewność co do tego, że one rzeczywiście są w blockchainie.



Rysunek 2: Schemat połączonych łańcuchów bloków

Źródło: Opracowanie własne

1.3 Stosowana kryptografia

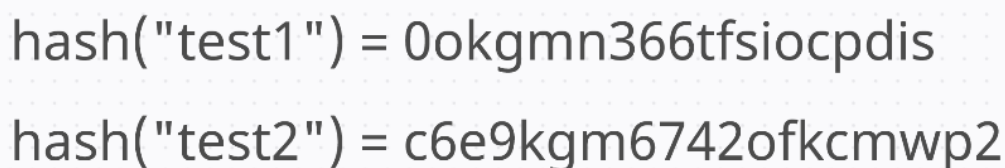
Sieci blockchain są zabezpieczone poprzez zastosowanie całej gamy algorytmów kryptograficznych²¹. W zależności od blockchaina część algorytmów pokrywa się lub używa jakiejś formy wariacji danego algorytmu. Zdarza się również, że sieci dodają swoje autorskie implementacje algorytmów, które mają na celu poprawić chociażby anonimowość. Każda sieć ma swoje zastosowanie oraz grupę odbiorców, dlatego nie ma sensu wsadzać wszystkich do jednego worka. Ich głównym celem jest zapewnienie bezpieczeństwa całej sieci. Rozwiązują wiele problemów oraz umożliwiają wiele ciekawych rozwiązań.

²¹ History of Cryptography, <https://academy.binance.com/en/articles/history-of-cryptography>

Dzięki nim blockchain jest niezmienny tzn. raz już potwierdzona transakcja nie może zostać cofnięta. Inny z kolei zapewnia, że użytkownicy nie mogą wydawać nawzajem swoich środków. Jeszcze inna gwarantuje zachowanie anonimowości w tym transparentnym świecie. Zdecydowanie w tej kulturze króluje wolne oprogramowanie²², które sprzyja rozwojowi implementacji nowych rozwiązań, jak i poprawianiu aktualnych.

Twórcy zdają sobie z tego sprawę i już bardzo poważnie przygotowują się na konieczność ulepszenia chociażby funkcji haszujących w kontekście obrony przed komputerami kwantowymi. W czerwcu na konferencji EthCC²³ Vitalik Buterin, który stoi na czele zespołu Ethereum Foundation stwierdził, że on i jego zespół będą nad tym pracować w ciągu najbliższych lat.

Pierwszym ze wspomnianych kluczowych algorytmów jest kryptograficzna funkcja skrótu to specjalna funkcja, która jest wykorzystywana w wielu miejscach w Blockchainie. Jej głównym celem jest bezpieczne szyfrowanie danych. Blockchain Ethereum korzysta z funkcji skrótu o nazwie Keccak-256²⁴.



```
hash("test1") = 0okgmn366tfsiocpdis
hash("test2") = c6e9kgm6742ofkcmwp2
```

Rysunek 3: Wizualizacja działania funkcji skrótu

Źródło: Opracowanie własne

Funkcja skrótu jest:

1. **Deterministyczna**, czyli zawsze te same dane wejściowe produkują te same dane wyjściowe.
2. **Jednokierunkowa**, czyli na podstawie wyniku funkcji skrótu, niemożliwe jest dowiedzenie się jakie były jej dane wejściowe. Z tego względu, że zmiana nawet jednego znaku powoduje zupełnie inny wynik danych wyjściowych.
3. **Jest odporna na kolizje**, czyli bardzo ciężko jest znaleźć dwa dane wejściowe, które wyprodukują nam te same dane wyjściowe.

²² What is open source, <https://opensource.com/resources/what-open-source>

²³ Vitalik Buterin, Thoughts about the longterm future of the Ethereum protocol, <https://www.youtube.com/watch?v=kGjFTzRTH3Q&t=1891s>

²⁴ Keccak256 hash function in Solidity, <https://cryptomarketpool.com/keccak256/>

4. Może przyjąć dowolnie duże dane a produkuje zawsze dane wyjściowe o stałej długości (fizycznie nie jest możliwe uniknięcie kolizji, ale zdarzają się one tak rzadko, że zakładamy, że ich po prostu nie ma). Są dla nas statystycznie nie ważne.
5. Jakakolwiek dowolna mała zmiana w danych wejściowych powoduje zupełnie inne, różne dane wyjściowe. W ten sposób nie możemy na podstawie podobieństwa danych wyjściowych wnioskować o tym, że jakieś dane wejściowe byłyby równe.

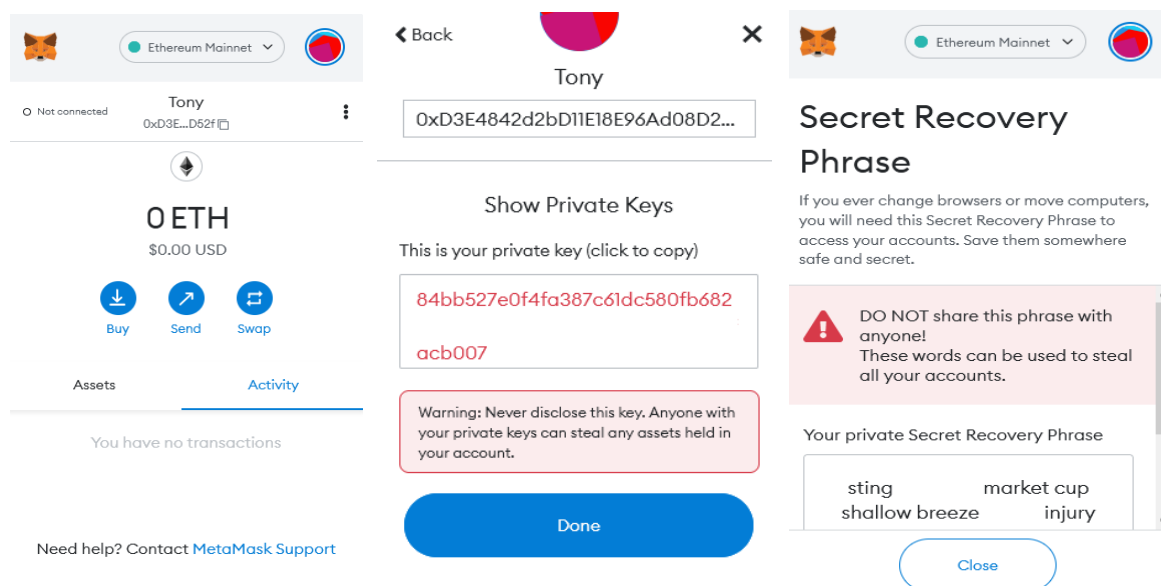
Kolejnym bazowym algorytmem jest kryptografia klucza publicznego²⁵, znana również jako kryptografia asymetryczna. Jest to struktura, w której wyróżniamy parę kluczy – prywatny oraz publiczny. W przeciwieństwie do pojedynczego klucza, który jest używany w kryptografii symetrycznej. Aby wejść w posiadanie takiego zestawu, konieczne jest utworzenie portfela kryptowalutowego²⁶. Aktualnie najpopularniejszy, kompatybilny z łańcuchami z rodziny EVM²⁷ to MetaMask²⁸, który jest dostępny na większość popularnych przeglądarek w postaci rozszerzenia. MetaMask funkcjonuje zarówno na urządzeniach stacjonarnych, jak i mobilnych.

²⁵ What is Public Key Cryptography, <https://academy.binance.com/en/articles/what-is-public-key-cryptography>

²⁶ Jakub Pawłowski, 1.5 Portfel Kryptowalutowy, Ethereum jako platforma dla aplikacji zdecentralizowanych

²⁷ Jakub Pawłowski, 2.4 EVM, Ethereum jako platforma dla aplikacji zdecentralizowanych

²⁸ Matt Hussey, What is MetaMask, <https://decrypt.co/resources/metamask>



Rysunek 4: Od lewej – główny panel portfela MetaMask, środek – klucz prywatny konta Tony, od prawej – słowa ziarna (seed phrase)

Źródło: Opracowanie własne

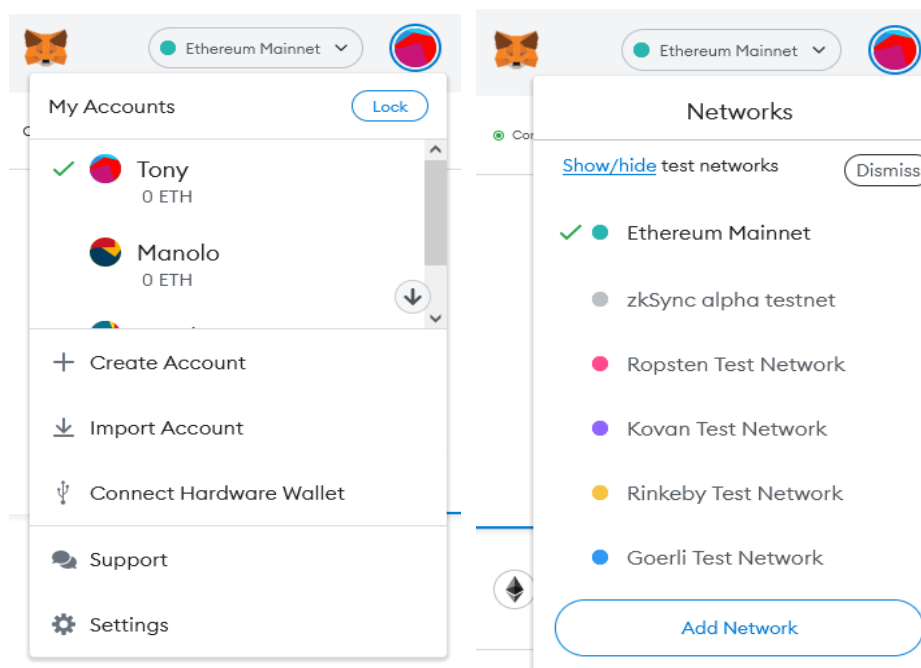
Po jego utworzeniu portfela i spisaniu **seed phrase**²⁹ (zdjęcie 3), naszym oczom ukazuje się domyślne konto przypisane do naszych słów ziaren (zdjęcie 1). To konto posiada oczywiście klucz prywatny, który możemy eksportować z portfela (zdjęcie 2).

Na drugim rzucie ekranu wyciąłem całą drugą linijkę, aby nie podawać całej wartości. Tak samo na trzecim widnieje tylko połowa słów ziaren, które są potrzebne do zaimportowania portfela. Powyższego konta używam tylko do celów edukacyjnych, ale i tak warto pamiętać o zasadach bezpieczeństwa i utrzymywać dobre nawyki poprzez zalecane praktyki.

Za pomocą klucza prywatnego możemy odtworzyć przypisane do niego konto, na dowolnie innym portfelu. **Seed phrase** z kolei umożliwia odtworzenie wszystkich kont, które zostały utworzone na portfelu, wraz z ich kluczami prywatnymi.

W skrócie, jeżeli na jednym portfelu mieliśmy kilka kont, ale ktoś wszedł w posiadanie jednego klucza prywatnego to ma dostęp do tylko jednego konta. Jeżeli ma cały **seed phrase** ~, czyli 12 lub 24 słowa to ma dostęp do wszystkich kont.

²⁹ Seed Phrase, <https://academy.binance.com/en/glossary/seed-phrase>



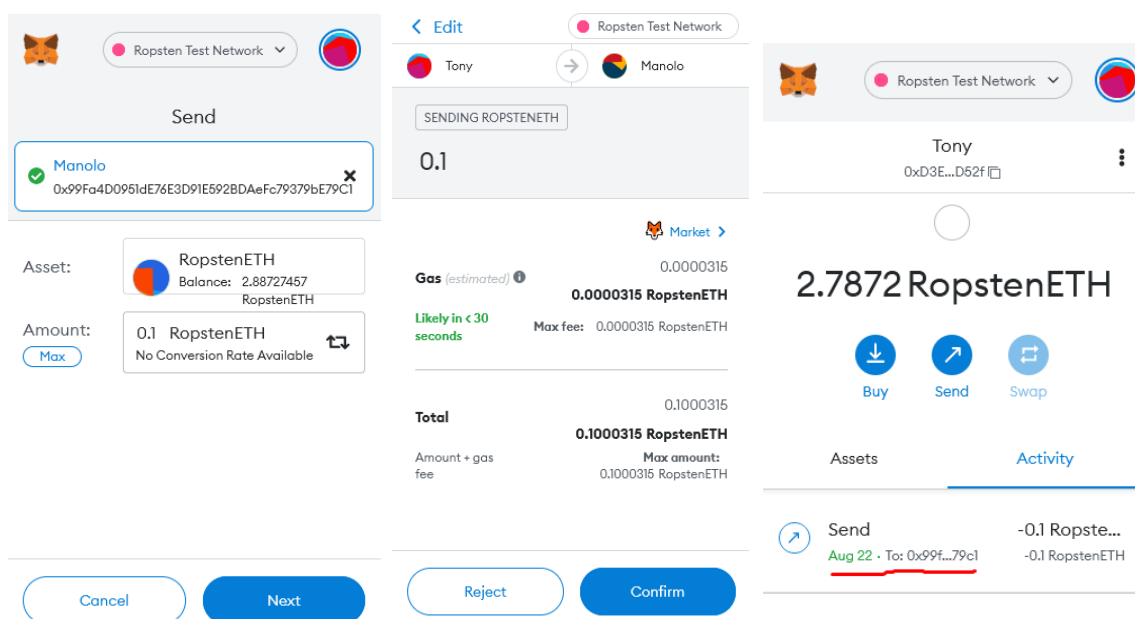
Rysunek 5: Od lewej – widok na dostępne konta w portfelu MM, od prawej – aktualnie dostępne sieci w portfelu

Źródło: Opracowanie własne

Warto w tym miejscu dodać, że w portfelu MetaMask posługujemy się tym samym kluczem publicznym i prywatnym obsługując różne sieci blockchain, kompatybilne z łańcuchem EVM. Na drugim zdjęciu jako pierwszy rekord widzimy sieć główną Ethereum. Cała reszta to sieci testowe, których używam do nauki. Każda para kluczy jest unikalna. Jak sama nazwa wskazuje – klucz publiczny można bezpiecznie udostępniać. W świecie Ethereum jak i innych sieci blockchain - klucz publiczny to swego rodzaju numer konta na który możemy przysyłać środki z innych kont w obrębie jednej sieci. Klucza prywatnego używamy do podpisów cyfrowych³⁰ oraz weryfikacji transakcji – możemy go przyrównać do numeru PIN, którym posługujemy się w tradycyjnej bankowości. Z perspektywy portfela kryptowalutowego widzimy niewiele, głównie ze względu na UX³¹ użytkowników. Nie każdy jest osobą techniczną. Temat i tak jest dosyć skomplikowany dla osób początkujących, dlatego nie ma sensu go zbytnio jeszcze bardziej utrudniać.

³⁰ What is a digital signature, <https://academy.binance.com/en/articles/what-is-a-digital-signature>

³¹ What is User Experience (UX) Design, <https://www.interaction-design.org/literature/topics/ux-design>



Rysunek 6: Schemat wysyłania testowej kryptowaluty Ether na inne konto w tym samym portfelu – od lewej do prawej

Źródło: Opracowanie własne

Dopiero w momencie pracy developerskiej z bibliotekami nisko poziomowymi tj. **Web3.py**³² w kontekście pracy z inteligentnymi kontraktami³³ poprzez interakcję z Ethereum – musimy klucz prywatny zawrzeć w kodzie, aby dokonać jakiegokolwiek transakcji z poziomu edytora kodu. Najlepiej w ukrytym module³⁴, aby nikt nie miał do niego dostępu.

Trzecim mechanizmem kryptograficznym jest podpis cyfrowy. Używamy go do weryfikowania autentyczności oraz integralności cyfrowych danych. Możemy go przyrównać do odręcznie pisanych sygnatur, ale z wyższym poziomem skomplikowania oraz bezpieczeństwa. Możemy go opisać jako kod, który jest załączany jako wiadomość. Po wygenerowaniu, kod zachowuje się jako dowód, że wiadomość nie została naruszona w drodze od nadawcy do odbiorcy. Dzięki temu podpisy cyfrowe są niezaprzeczalne. Fundamentem podpisów cyfrowych są funkcje skrótu oraz kryptografia asymetryczna.

³² Introduction, <https://web3py.readthedocs.io/en/latest/>

³³ Jakub Pałowski, 3. Inteligentne Kontrakty, Ethereum jako platforma dla aplikacji zdecentralizowanych

³⁴ Modular Programming, <https://www.techopedia.com/definition/25972/modular-programming>

W kontekście użyteczności w blockchainie – podpisy cyfrowe składają się z:

1. **Haszowania** – zbieramy w całość wszystkie dane z transakcji, które następnie przesyłamy do funkcji skrótu. Z tego miejsca warto odnotować, że podpisy cyfrowe są ściśle związane z zawartością każdej wiadomości. W przeciwieństwie do tradycyjnych podpisów, które są takie same niezależnie od wiadomości – tutaj każdy jest inny.
2. **Podpisywania** – na tym etapie wyróżniamy wiele różnych algorytmów podpisywania cyfrowego, dlatego ich działanie może się różnić. Zasadniczo w następnej fazie, wynik funkcji skrótu musi zostać podpisany przez nadawcę za pomocą jego klucza prywatnego.
3. **Weryfikowania** – odbiorca weryfikuje transakcje za pomocą klucza publicznego nadawcy. Dzięki temu mamy pewność, że podpis został wykonany przez osobę X, ponieważ tylko ona posiada klucz prywatny korelujący z kluczem publicznym, którym weryfikowaliśmy transakcje.

1.4 Podział modeli sieci Blockchain

Pierwszym z modeli to **UTXO**³⁵ (Unspent transaction output) – aktualny stan jest pojęciem abstrakcyjnym i można go wyznaczyć poprzez zaaplikowanie wszystkich transakcji jakie miały miejsce. Model ten występuje w sieci Bitcoina.

W Ethereum zastosowany jest **account-based transaction model**³⁵. W tej sieci zapisywany jest aktualny stan, czyli np. Magda ma 1 ETH³⁶, Jacek ma 0 ETH w chwili pierwszej. Po zaaplikowaniu jednej transakcji, mamy zapisywany nowy stan, czyli Magda ma teraz 0.6 ETH, a Jacek 0.4 ETH.

Przechowujemy dwie różne informacje:

- a) Jaki był stan przed i po transakcji
- b) Przechowujemy dane transakcji

³⁵ UTXO vs Account Model, <https://www.horizen.io/blockchain-academy/technology/expert/utxo-vs-account-model/>

³⁶ Jake Frankenfield, Ether (ETH), <https://www.investopedia.com/terms/e/ether-cryptocurrency.asp>

Jest to swego rodzaju redundancja³⁷, która zapewnia podwójne zabezpieczenie sieci, na wypadek jakichś nieścisłości. W Bitcoinie z kolei nie przechowujemy aktualnego stanu, tylko jego transakcje. Czyli mamy jakiś stan początkowy i później dopiero poprzez aplikowanie kolejnych transakcji można dojść do jakiegoś stanu X, który nas interesuje. Ważne jest to, że sam stan w tym systemie nie jest zapisywany tak jak w Blockchainie Ethereum, są zapisywane tylko dane transakcji.

1.5 Portfel kryptowalutowy

W sekcji o kryptografii asymetrycznej poruszyłem podstawy związane z portfelem kryptowalutowym³⁸ w kontekście kluczy publicznych oraz prywatnych. Po co jest on nam w ogóle potrzebny? Wszystko zależy od tego co dokładnie chcemy robić w świecie kryptowalut. Jeżeli zależy nam tylko na długoterminowym trzymaniu aktywów lub rozgrywaniu pozycji na giełdzie CEX³⁹ (giełda scentralizowana tj. Binance, Coinbase, Kanga Exchange) to w zasadzie nie będzie nam on potrzebny. Utarło się takie powiedzenie w kontekście trzymania środków na giełdzie – nie twoje klucze prywatne, nie twoje krypto.

Wzięło się to z tego, że giełdy trzymają swoje środki na pełnoprawnych portfelach kryptowalutowych, ale już użytkownicy są jedynie rekordem w ich bazie danych. W takiej sytuacji wierzymy na słowo, że giełda będzie cały czas funkcjonować, a jej właściciele nie uciekną ze wszystkimi pieniędzmi gdzieś w świat. Nawet jeżeli się tak nie stanie i właściciele są uczciwi to giełda może zostać zaatakowana przez grupę hackerów, których zadaniem jest zgarnięcie środków. Zarówno pierwsza jak i druga opcja wydarzyła się wielokrotnie, dlatego naprawdę nie zalecam trzymania tam większych środków. Dlatego giełdę powinniśmy traktować jako miejsce, w którym kupujemy krypto aktywa a następnie przesyłamy je na nasz docelowy portfel. Portfel kryptowalutowy oprócz roli naszego osobistego konta bankowego służy również do interakcji z różnymi aplikacjami zdecentralizowanymi, czyli takimi, które korzystają z systemów blockchain. Warto również pamiętać, że wchodząc do tego świata jesteśmy zdani tak naprawdę tylko na siebie. Nie mamy możliwości zadzwonienia do banku z prośbą o cofnięcie danej transakcji. A w momencie, w którym stracimy środki w wyniku naszej nieuwagi musimy to wziąć na

³⁷ Redundancja, <https://mfiles.pl/pl/index.php/Redundancja>

³⁸ Kirsty Moreland, What is a Crypto Wallet, <https://www.ledger.com/academy/what-is-a-crypto-wallet>

³⁹ Benedict George, What is a CEX, <https://www.coindesk.com/learn/what-is-a-cex-centralized-exchanges-explained/>

nasze barki. Warto w tym miejscu dodać, że żaden portfel kryptowalutowy nie przechowuje fizycznie naszych kryptowalut, tylko jego klucze prywatne, które są potwierdzeniem, że jesteśmy ich właścicielem.

Portfel kryptowalutowy MetaMask, którego wcześniej pokazałem należy do rodziny **hot wallet'ów**⁴⁰. Znaczy to tyle, że przetrzymuje klucze prywatne w aplikacji, która ma stały dostęp do Internetu. Założenie takiego **gorącego portfela** to dosłownie 2 minuty. W dodatku jest w pełni darmowe. Z tego powodu jest to najpopularniejsze rozwiązanie dla większości użytkowników, z najmniejszym progiem wejścia. Mało kto na samym początku swojej przygody zastanawia się nad kwestią bezpieczeństwa. To raczej przychodzi z czasem, kiedy jako użytkownicy stajemy się bardziej świadomi jak to wszystko działa i dlaczego jest to takie ważne.



Rysunek 7: Portfel sprzętowy (cold wallet) – Ledger Nano S

Źródło: Opracowanie własne

⁴⁰ Jake Frankenfield, Hot Wallet, <https://www.investopedia.com/terms/h/hot-wallet.asp>

Zalecaną opcją dla każdego użytkownika jest posiadanie portfela sprzętowego, czyli **cold wallet** ⁴¹. Jego podstawową zaletą jest utrzymywanie kluczy prywatnych offline, odłączonych od Internetu. Największy prym wiodą dwie marki - Ledger⁴² oraz Trezor⁴³. Na powyższej grafice widzimy Ledger Nano S. Osobiście posiadam dokładnie taki sam model, jedynie w innej wersji kolorystycznej.

Z komputerem łączymy się poprzez złącze USB. Pierwszą warstwą zabezpieczenia jest kod PIN, który może mieć maksymalnie 8 liczb od 0-9. Następnie wchodzimy już do wnętrza urządzenia. W tym konkretnym modelu możemy mieć maksymalnie kilka aplikacji, ponieważ urządzenie nie posiada zbyt dużo pamięci. Osobiście korzystam tylko z aplikacji Ethereum oraz Bitcoin. Aby wykonać jakąkolwiek transakcję wysyłania środków lub interakcji z kontraktem, konieczne jest potwierdzenie jej w urządzeniu. Stanowi to kolejną warstwę ochrony, która daje nam potrzebny czas nad zastanowieniem się czy to co aktualnie robimy jest dla nas bezpieczne. Nie działamy pod wpływem emocji.

W ramach ciekawostki - domyślnie za pomocą Ledger'a w Ethereum nie możemy wchodzić w interakcje z żadnym kontraktem. Jest to możliwe dopiero jak wejdziemy do ustawień aplikacji Ethereum i zaznaczymy opcję **blind signing**⁴⁴ jako enabled.

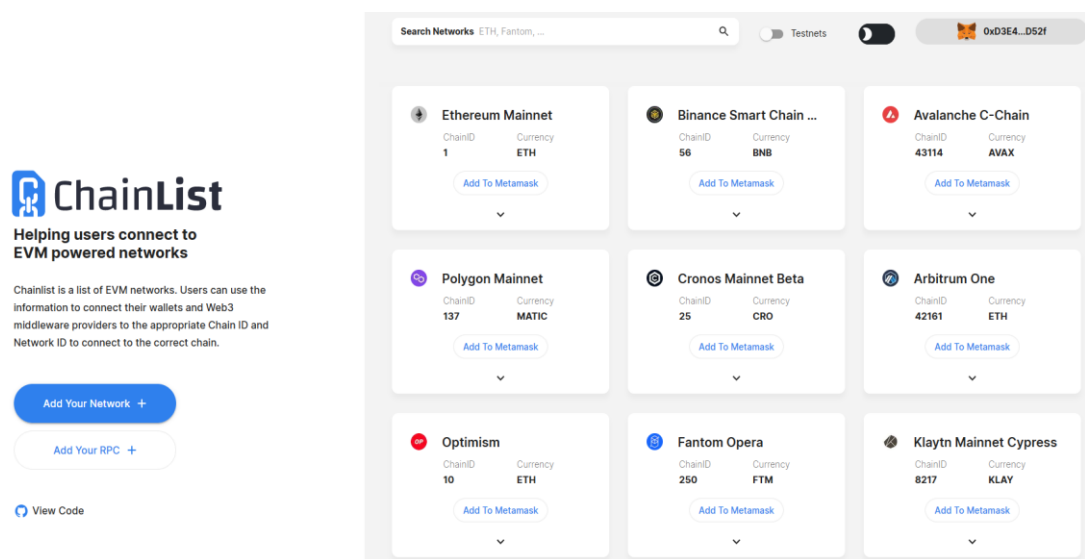
Musimy pamiętać, że nawet portfel sprzętowy nie ochroni nas przed interakcją ze złośliwym kontraktem. Dlatego w pełni rozumiem i szanuję decyzję twórców w kwestii domyślnego zablokowania tej możliwości.

⁴¹ What are Cold Wallets, <https://academy.bit2me.com/en/what-are-cold-wallets/>

⁴² Rakesh Sharma, Ledger Wallet, <https://www.investopedia.com/terms/l/ledger-wallet.asp>

⁴³ Trezor Wallet, <https://www.bitdeal.net/trezor-wallet>

⁴⁴ Kristy Moreland, Enable Blind Signing: Why, When and How to Stay Safe, <https://www.ledger.com/academy/enable-blind-signing-why-when-and-how-to-stay-safe>

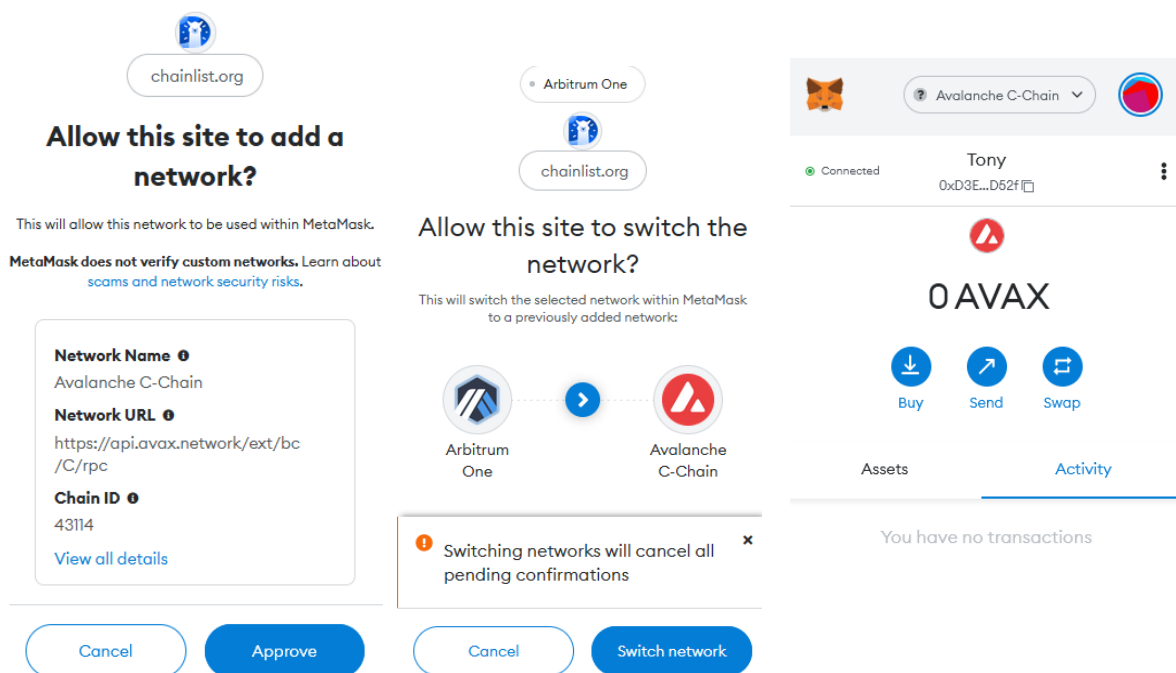


Rysunek 8: Aplikacja webowa umożliwiająca szybkie podłączenie różnych sieci z rodziny EVM do portfela MetaMaska

Źródło: <https://chainlist.org/>

W sekcji o kryptografii asymetrycznej wspomniałem o pojęciu słów ziaren, inaczej **seed phrase**. W momencie utworzenia portfela kryptowalutowego. Zarówno na **gorącym** jak i **zimnym** portfelu – otrzymujemy specjalną listę, która składa się z 12 lub 24 słów. W przypadku **MetaMaska** będzie to 12 słów, natomiast **Ledger** daje już 24 słowa. Jeżeli aktualny sprzęt zostanie nam ukradziony lub zniszczony. Możemy odtworzyć nasz portfel na dowolnie innym urządzeniu, wliczając w to urządzenia mobilne. Z tego powodu powinniśmy od razu zapisać naszą listę słów na osobnej kartce papieru i trzymać ją w bezpiecznym miejscu. Portfel MetaMask obsługuje bardzo dużą liczbę sieci, które są kompatybilne z **EVM**, czyli maszyną wirtualną Ethereum. W dowolnej chwili możemy dodać do MetaMaska interesującą nas sieć i zacząć z niej korzystać. Mechanizm dodawania nowej sieci do portfela MM przedstawiłem poniżej na przykładzie **Avalanche**⁴⁵.

⁴⁵ Nathan Reiff, What is Avalanche (AVAX), <https://www.investopedia.com/avalanche-avax-definition-5217374>



Rysunek 9: Przykładowy schemat dodawania nowej sieci do portfela MetaMask w trzech krokach – od lewej do prawej

Źródło: Opracowanie własne

Wyróżniamy dwa typy, czyli sieci **główne**⁴⁶ i sieci **testowe**⁴⁶. Różnica między nimi jest taka, że na sieciach testowych, wszystkie tokeny⁴⁷ nie mają żadnej realnej wartości. Sieci testowe służą przede wszystkim do eksperymentów nad nowymi funkcjonalnościami inteligentnych kontraktów, ponieważ w pełni odwzorowują sieci główne. Z racji tego, że są w pełni darmowe, developerzy na całym świecie chętnie z nich korzystają, aby sprawdzić czy ich produkt nie zawiera krytycznych błędów.

⁴⁶ Networks, <https://ethereum.org/en/developers/docs/networks/>

⁴⁷ Jakub Pawłowski, 1.6. Coin vs Token, Ethereum jako platforma dla aplikacji zdecentralizowanych

1.6 Coin vs Token

Istnieje mylne przekonanie, że dwa powyższe terminy są sobie równe i często nieświadomi użytkownicy używają ich zamiennie co jest dużym błędem.

Coinem⁴⁸ możemy nazwać kryptowalutę, która posiada własną sieć blockchain. Do tej kategorii oczywiście zalicza się Ethereum, Bitcoin, Solana oraz wiele innych.

Tokenem⁴⁸ z kolei nazywamy projekty, które są umieszczane w blockchainach coinów. Jest to bardzo popularna praktyka, ponieważ tworzenie blockchained od podstaw jest bardzo skomplikowane, czasochłonne oraz kosztowne. Nie ma sensu za każdym razem wymyślać koła na nowo. Kolejną kwestią jest, że wiele projektów chce korzystać z dobrodziejstw istniejących już blockchainów i nie mają potrzeby tworzenia własnych sieci.

ROZDZIAŁ II

2. Blockchain Ethereum

Bitcoin mimo swojej wielkiej rewolucji posiada jedynie podstawową funkcjonalność w postaci transferów środków na różne adresy. Jego dużym atutem jest skończona liczba Bitcoinów, które może zostać wydobyta. Czyli jego inflacja jest z góry określona. Nigdy nie powstanie więcej niż 21 milionów sztuk⁸.

Vitalik Buterin, jeden z twórców Ethereum poszedł o krok dalej. W 2013 roku stworzył wizję zdecentralizowanego komputera EVM²⁷, za pomocą którego możemy uruchamiać dowolne aplikacje. W przypadku blockchained Ethereum aplikacjami nazywamy inteligentne kontrakty³³. Jeżeli kontrakty umieścimy w sieci Ethereum to nabywają one cechy blockchained. Czyli raz umieszczona aplikacja w Ethereum zawsze będzie działała w taki sam sposób. Nikt nie może tego zmienić, włącznie z jego twórcami. W odróżnieniu od platform tj. Google Play⁴⁹ czy App Store⁵⁰, nie potrzebujemy niczyjej zgody na opublikowanie naszej aplikacji. Automatycznie po jej umieszczeniu, staje się publicznie dostępna dla wszystkich użytkowników.

⁴⁸ Crypto tokens vs coins – What's the Difference, <https://crypto.com/university/crypto-tokens-vs-coins-difference>

⁴⁹ How Google Play works, <https://play.google.com/about/howplayworks/>

⁵⁰ App store, <https://www.computerhope.com/jargon/a/app-store.htm>

Połączenie tego wszystkiego sprawia, że odpowiednio zaprogramowane inteligentne kontrakty mogą pełnić rolę zaufanej trzeciej strony, ponieważ raz zakodowane są niezamienialne. Co wiąże się z tym, że nie potrzebujemy pośredników, którzy pełnili rolę gwaranta dla obu stron. Przykładem może być chociażby giełda oparta o inteligentne kontrakty, w skrócie **DEX**⁵¹, czyli giełda zdecentralizowana. W odróżnieniu od tradycyjnej giełdy **CEX**³⁹, czyli giełdy zcentralizowanej nie wymaga ona zaufanej trzeciej strony. W sieci Ethereum posługujemy się walutą **ether**, która przyjmuje formę skrótową **ETH**³⁶.

2.1 Typy kont w Ethereum

Rozróżniamy dwa typy kont. Pierwsze z nich to konta zewnętrzne (**Externally Owned Accounts - EOA**⁵²), są one kontrolowane przez ludzi, osoby, które posiadają do nich klucz prywatny. Drugie z kolei to konta kontraktowe (**Contract Accounts**⁵²), które są sterowane przez kod, który został użyty do ich stworzenia.

Oba typy kont mają dwa pola:

- a) Saldo (balance) – czyli ile mają w danym momencie ETH
- b) Nonce – licznik transakcji

Na początku każde konto ma **nonce** równe zero, jeżeli wykona pierwszą transakcję to wtedy **nonce=1**, etc. Konta kontraktowe z kolei posiadają swój kod źródłowy, który nimi steruje oraz swoją pamięć.

2.2 Wyznaczanie adresu konta

W przypadku konta zewnętrznego (**EOA**), adres publiczny jest wyznaczany na podstawie klucza prywatnego. Aby stworzyć nowe konto, trzeba wygenerować parę kluczy (publiczny oraz prywatny) poprzez utworzenie portfela kryptowalutowego.

Z kolei adres konta kontraktowego wyznacza się w inny sposób. Konto kontraktowe zawsze tworzone jest przez jakieś konto zewnętrzne **EOA**, więc adres konta kontraktowego to wynik funkcji skrótu z dwóch informacji:

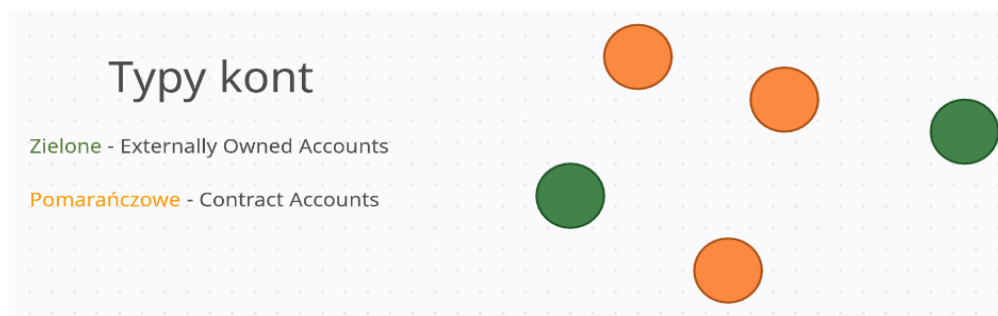
⁵¹ What is a Decentralized Exchange (DEX), <https://www.gemini.com/cryptopedia/decentralized-exchange-crypto-dex>

⁵² The difference between an EOA and a contract, <https://www.oreilly.com/library/view/mastering-blockchain-programming/9781839218262/00403615-d0ec-4c80-94a0-dc1d24fd3246.xhtml>

- a) Adresu konta, które je utworzyło
- b) Licznik transakcji (**nonce**) tego konta

2.3 Transakcje na Ethereum

Transakcje mogą być inicjowane tylko przez konta zewnętrzne (**EOA**), inteligentny kontrakt (**smart contract**) – sam nic nie robi. Wyróżniamy dwa rodzaje transakcji. Pierwsza z nich to utworzenie nowego konta kontraktowego, czyli stworzenie nowego **smart contractu**. W wyniku transakcji dodawane jest nowe konto kontraktowe, które ma swój kod źródłowy oraz pamięć. Druga z kolei to **message call**⁵³. Jest to taki rodzaj transakcji w wyniku, którego zmieniany jest stan jakiegoś konta. Transakcja do istniejącego już konta zewnętrznego.

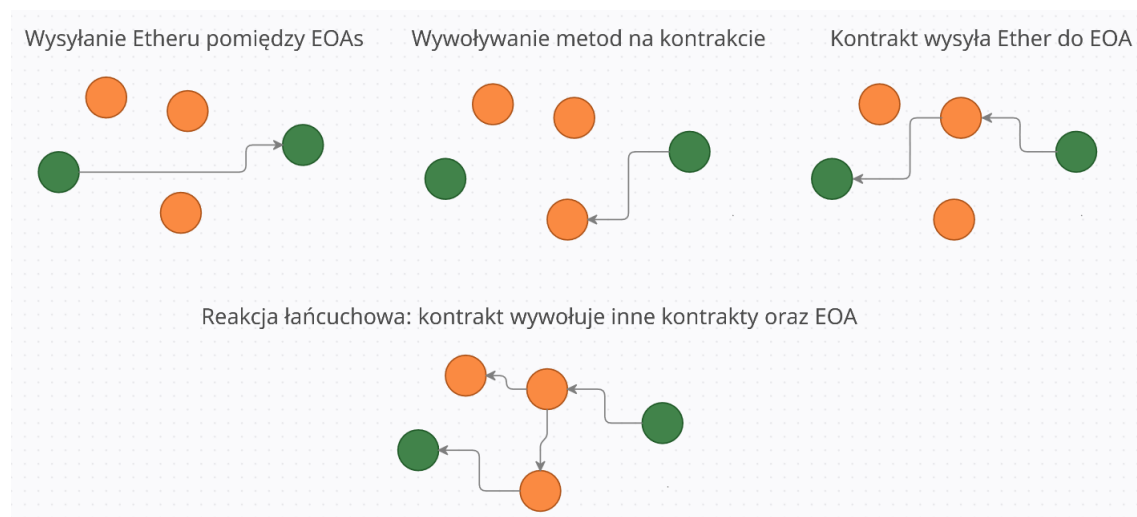


Rysunek 10: Wizualizacja typów kont w sieci Ethereum

Źródło: Opracowanie własne

Tylko **EOA** mogą tworzyć nowe transakcje (są jedynym punktem wejścia do naszego systemu). W wyniku transakcji tworzonych przez te konta można wprowadzić dane do blockchained. Pierwszą najprostszą możliwą transakcją jest przesyłanie Etheru z jednego konta zewnętrznego na drugie. Jest to odpowiednik transakcji przelewu z jednego konta na drugie. Drugą możliwością jest wywoływanie metod na kontraktach. W jej wyniku może on przelać Ether do jakiegoś konta zewnętrznego w momencie spełnienia konkretnych warunków, które są zapisane w kodzie takiego kontraktu.

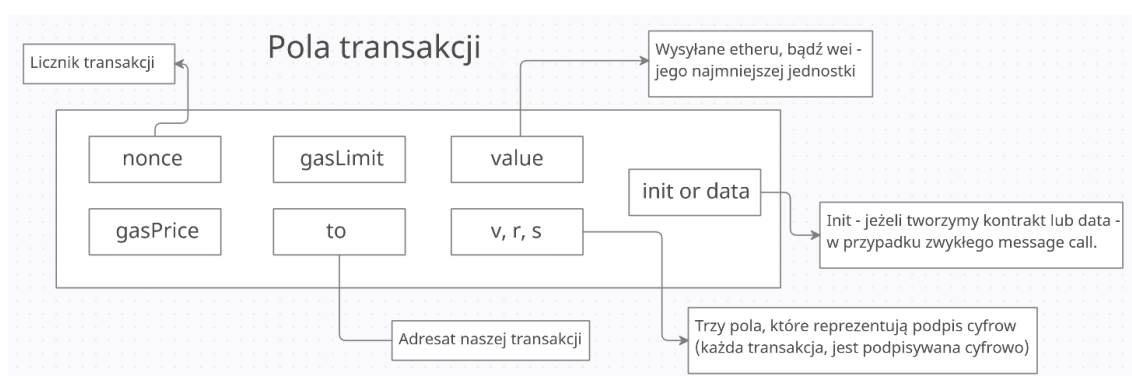
⁵³ What is the difference between a transaction and a call, <https://ethereum.stackexchange.com/questions/765/what-is-the-difference-between-a-transaction-and-a-call>



Rysunek 11: Wizualizacja możliwych wariacji transakcji w sieci Ethereum

Źródło: Opracowanie własne

Konta kontraktowe mają również możliwość przechowywania Etheru i wypłacenia ich tylko w określonych warunkach. Konta kontraktowe mogą również wywoływać metodę na innych kontach kontraktowych – reakcja łańcuchowa. Natomiast bezwzględnie trzeba pamiętać, że zawsze inicjatorem transakcji jest konto zewnętrzne. Transakcja to obiekt, która zawiera kilka pól.



Rysunek 12: Wizualizacja pól, które posiada każda transakcja w sieci Ethereum

Źródło: Opracowanie własne

Pierwszy z nich to **nonce**, czyli licznik transakcji. Druga i trzecia z kolei odpowiada za **gas**⁵⁴, który jest charakterystyczny dla sieci Ethereum. Każdy blok na tym blockchainie ma limit transakcji, które może pomieścić. **Gas** to jednostka wysiłku jaką ponosi sieć na wykonanie transakcji. Trudność, poziom skomplikowania transakcji liczymy w jednostkach **gas**. **Ether** jest podzielny do 18-stu miejsc po przecinku. Jego najmniejszą jednostką jest **wei**, czyli $1 \text{ ETH} = 1\,000\,000\,000\,000\,000\,000 \text{ wei}$.

Prowizje za transakcję (**Tx fee**) liczymy z iloczynu zużytego **gas'u** oraz **gasPrice**⁵⁴. Cena jaką płacimy za jedną jednostkę **gas'u** sami ustalamy. W zależności od tego jak szybko chcemy, aby nasza transakcja doszła do bloku.

Transakcje są dobierane do bloku przez górników⁵⁵ (jednym ze źródeł dochodu górników są prowizje z transakcji), także górnicy dołączają do bloku takie transakcje, które przyniosą im największy zysk (mamy wolny rynek – jest dużo chętnych na dołączenie nowych transakcji – transakcji jest o wiele więcej niż miejsca w blokach). Często zdarza się, że jak wyślemy transakcję, to nie trafia ona do pierwszego bloku, ponieważ się po prostu w tym bloku nie zmieści. Ustalając wyższy **gasPrice**, mamy ciut większe prawdopodobieństwo, że szybciej zostanie wykopana, ponieważ górnikowi będzie się to bardziej opłacać.

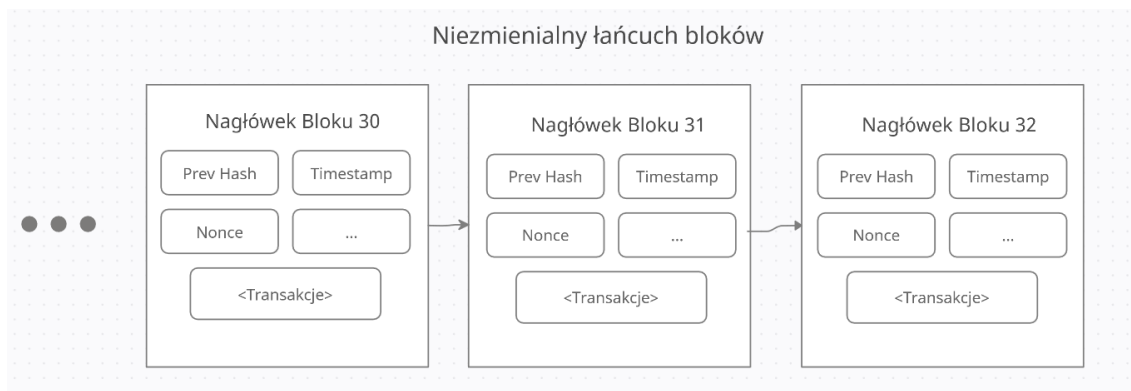
Jednakże jest również **gasLimit**⁵⁴, który jest po to, żeby ustalić jakąś maksymalną cenę jaką zapłacimy za transakcję (często wysyłając transakcję, nie do końca wiemy, ile ona spali **gas'u**, także aby się zabezpieczyć, przed taką sytuacją, w której transakcja kosztuje nas za dużo, mamy pole **gasLimit**, wtedy maksymalny koszt prowizji wyniesie **gasPrice * gasLimit**.

W tym wypadku nie ma czegoś takiego jak czas transakcji, ponieważ kto miałby go ustalić? Osoba wysyłająca transakcję może mieszkać w innej strefie czasowej niż osoba kopiująca. W komputerach mamy zawsze czas, jest on brany z Internetu, jakiegoś centralnego serwera. W naszym wypadku mamy systemy zdecentralizowane, w których nie mamy zależności od żadnego centralnego serwera. Także nie da się ustalić aktualnego czasu. To, że górnik⁵⁵ myśli, że dana transakcja jest pierwsza, to niekoniecznie tak jest. On może myśleć, że dana transakcja jest pierwsza, bo jako pierwsza do niego dotarła. Jednakże to wcale nie musi być prawdą z tego względu, że górnik może być z Chin i w momencie czasowym **x1**, wysłał ktoś z Ameryki, a w momencie czasowym **x2**, wysłał

⁵⁴ What is Ethereum Gas, <https://www.sofi.com/learn/content/what-is-ethereum-gas/>

⁵⁵ Jakub Pawłowski, 2.6. Algorytm konsensusu, Ethereum jako platforma dla aplikacji zdecentralizowanych

ktoś z Japonii. Załóżmy, że do Chin najpierw doleci transakcja z Japonii, a później ta z Ameryki. Górnik myśli, że ta z Japonii była pierwsza, ale tak naprawdę to była ta z Ameryki.



Rysunek 13: Wizualizacja przykładowego ciągu bloków w sieci blockchain

Źródło: Opracowanie własne

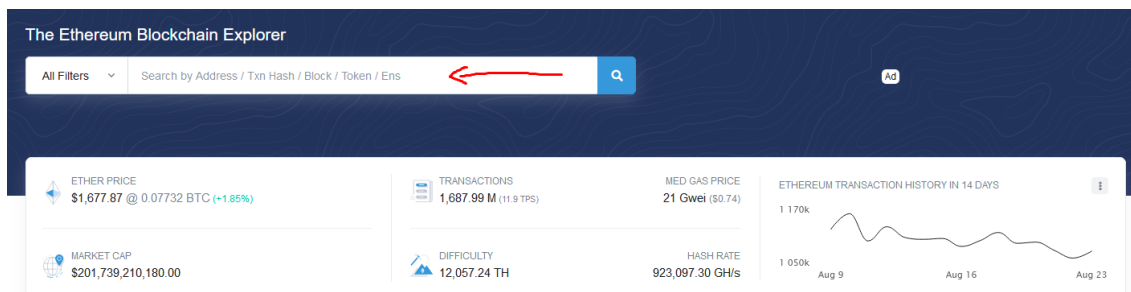
Powyżej widać, że mamy **Timestamp**⁵⁶, ustala go górnik (skoro wybraliśmy jedną osobę z całej sieci, która powie jakie są transakcje osadzone w bloku), to również dajemy tej osobie możliwość ustalenia **Timestamp'u** bloku. Bloki są kopane co 15 sekund, także w **Timestamp**, musi być ta różnica pomiędzy blokami ~ 14-16 sekund. **Timestamp** jest nam potrzebny, ponieważ możemy tworzyć smart kontrakty, które polegają na czasie, czyli np. jakieś środki są uwalniane po upływie paru miesięcy. Transakcje są atomiczne, tak jak w zwykłej bazie danych. To znaczy, że transakcja wykonała się albo nie wykonała. Transakcje wykonują się jedna, po drugiej. Także mamy globalny komputer, który ma jeden wątek. Nie da się wykonywać dwóch transakcji naraz. Jest to jeden z powodów, dlaczego **blockchain'y** słabo skalują się do większej liczby transakcji na sekundę.

Jednym z bardzo użytecznych narzędzi dla użytkowników platformy Ethereum z pewnością jest strona Etherscan.io⁵⁷. Jest to eksplorator bloków technologii blockchain dla sieci Ethereum. Możemy w nim śledzić na bieżąco nowe transakcje dodawane do

⁵⁶ How Accurate and Reliable is the Block Timestamp in Ethereum, <https://origin-stamp.com/blog/how-accurate-and-reliable-is-the-block-timestamp-in-ethereum/>

⁵⁷ Valerio Puggioni, What is Etherscan, and how does it work, <https://cointelegraph.com/news/what-is-etherscan-and-how-does-it-work>

bloków, obserwować historię transakcji, które miały miejsce w przeszłości. W tym również prześledzić historię naszego konta, wystarczy wkleić do zakładki nasz adres publiczny.



Rysunek 14: Podgląd na interfejs strony głównej etherscan'u

Źródło: <https://etherscan.io/>

Etherscan na bieżąco informuję również o tym jakie są średnie opłaty za gas w jednostce gwei, która wynosi **1 gwei = 0.000000001 ether**. Wszystkie bieżące transakcje, w tym wykopane bloki są bieżąco pokazywane, ponieważ wszystko jest transparentne.

Latest Blocks			
Bk	<div>15404919</div> <div>28 secs ago</div>	<div>Miner F2Pool Old</div> <div>335 txns in 5 secs</div>	2,05923 Eth
Bk	<div>15404918</div> <div>33 secs ago</div>	<div>Miner Miner: 0x5dc...435</div> <div>51 txns in 16 secs</div>	2,00648 Eth
Bk	<div>15404917</div> <div>49 secs ago</div>	<div>Miner Poolin 2</div> <div>206 txns in 5 secs</div>	2,04843 Eth
Bk	<div>15404916</div> <div>54 secs ago</div>	<div>Miner Ethereum</div> <div>91 txns in 8 secs</div>	2,1371 Eth
Bk	<div>15404915</div> <div>1 min ago</div>	<div>Miner Ethereum</div> <div>115 txns in 4 secs</div>	2,02412 Eth
Bk	<div>15404914</div> <div>1 min ago</div>	<div>Miner Ezil.me : Ezil Pool 4</div> <div>52 txns in 7 secs</div>	2,01114 Eth
View all blocks			
Latest Transactions			
Tx	<div>0x1bb7fe187be3...</div> <div>28 secs ago</div>	<div>From 0x36f76741813117a0...</div> <div>To 0x000000000006c3852cb...</div>	0.001 Eth
Tx	<div>0x2470d8fb25c1...</div> <div>28 secs ago</div>	<div>From 0xefd3a7459f375033c7f...</div> <div>To 0x000000000006c3852cb...</div>	0.293 Eth
Tx	<div>0xf40d81d88bafe...</div> <div>28 secs ago</div>	<div>From 0x7ee87e9361d0912ab...</div> <div>To 0x27d9086cb8a9f82cbf5...</div>	0 Eth
Tx	<div>0x2d1bc6ad3149...</div> <div>28 secs ago</div>	<div>From 0x01142f38f11df85f982...</div> <div>To 0x283aaf0b28c62c092cb...</div>	0.006 Eth
Tx	<div>0xf7d0b25d28a9...</div> <div>28 secs ago</div>	<div>From 0xd135be23e34bf96a7...</div> <div>To 0x02d67a988653ee16e...</div>	0.27492 Eth
Tx	<div>0x9fbae3a7a5e...</div> <div>79 mins ago</div>	<div>From 0x64f2a11e1158f3eeef...</div> <div>To 0x445c3cf3b34a20c4d4ff...</div>	0 Eth
View all transactions			

Rysunek 15: Podgląd na ostatnie wydobyte bloki oraz transakcje

Źródło: <https://etherscan.io/>

2.4 EVM – Ethereum Virtual Machine

Celem maszyny wirtualnej Ethereum była wizja globalnego komputera. Aby lepiej zrozumieć architekturę systemu konieczne jest poznanie kilku elementów.

Pierwszym z nich to pojęcie maszyny wirtualnej. Ich cechą charakterystyczną jest to, że działają na wyższym poziomie abstrakcji niż tradycyjne systemy operacyjne. W odróżnieniu od Windowsa czy Mac OS, maszyny wirtualne są tworzone na bazie zwykłych systemów operacyjnych, dzięki czemu mogą działać jak ich fizyczne odpowiedniki. Mogą być uruchamiane na różnych systemach operacyjnych oraz różnym sprzęcie komputerowym. Sprawia to, że są idealnym silnikiem dla zdecentralizowanych systemów. Dzięki maszynie wirtualnej można korzystać z zasobów uczestników sieci niezależnie od lokalizacji lub położenia geograficznego. W tym sensie EVM⁵⁸ działa jak globalny procesor, który udostępnia swoją zakumulowaną moc obliczeniową developerom. Programiści wykorzystują ten zasób do tworzenia inteligentnych kontraktów. Maszyna wirtualna może zostać użyta gdziekolwiek na świecie poprzez uczestniczące węzły¹⁵ sieci Ethereum. Biorąc pod uwagę wszystkie zalety maszyn wirtualnych, wybranie przez twórców Ethereum takiej architektury wydaje się być bardzo sensowne. Kluczową funkcjonalnością protokołu Ethereum jest wykorzystywanie inteligentnych kontraktów, które są z kodowaną instrukcją, która wchodzi w interakcję z EVM. Maszyna wirtualna Ethereum jest w stanie wykonać kod bajtowy⁵⁹ dla aplikacji zdecentralizowanych, czyli inteligentnych kontraktów. W ekosystemie Ethereum kontrakty tworzone są przez programistów w wysokopoziomowym języku o nazwie Solidity. Po napisaniu kontraktu, aby móc go umieścić oraz uruchomić na sieci głównej lub testowej, konieczne jest jego skompilowanie do kodu bajtowego. Kod bajtowy jest kodem pośrednim między kodem źródłowym (w tym wypadku Solidity) a kodem maszynowym⁵⁹. Jest kodem niskopoziomowym, który otrzymujemy w wyniku kompilacji kodu źródłowego. Aby kontrakty były w stanie działać przez wiele węzłów bez narażenia na niebezpieczeństwo, EVM ma następujące funkcje:

- a) Tak samo jak funkcja skrótu, EVM również jest deterministyczne. Funkcje w kontraktach w interakcji z użytkownikami zawsze będą zachowywać się w ten sam sposób.

⁵⁸ EVM Explained – What is Ethereum Virtual Machine, <https://moralis.io/evm-explained-what-is-ethereum-virtual-machine/>

⁵⁹ Difference between Byte Code and Machine Code, <https://www.geeksforgeeks.org/difference-between-byte-code-and-machine-code/>

- b) Maszyna wirtualna Ethereum jest izolowana. Czyli jakiegokolwiek krytyczne błędy zawarte w kontraktach, które hackerzy będą chcieli wykorzystać obejmują tylko ten jeden konkretny kontrakt a nie cały protokół Ethereum.
- c) Kontrakty w teorii mogą rozwiązać każdy możliwy problem. Nie jest określone jednak w jakim czasie. Dlatego konieczne jest wprowadzenie mechanizmu kończącego, aby stworzyć dokładne limity. Z tego powodu stworzono termin **gas**, który służy do ułatwienia ruchu w sieci. Opłaty za gas są wykorzystywane do selektywnego określania, które funkcje powinny być uruchamiane lub traktowane priorytetowo. Jego limity są określane w momencie zawierania transakcji. Jeżeli zostaną one przekroczone to maszyna zatrzymuje operacje.

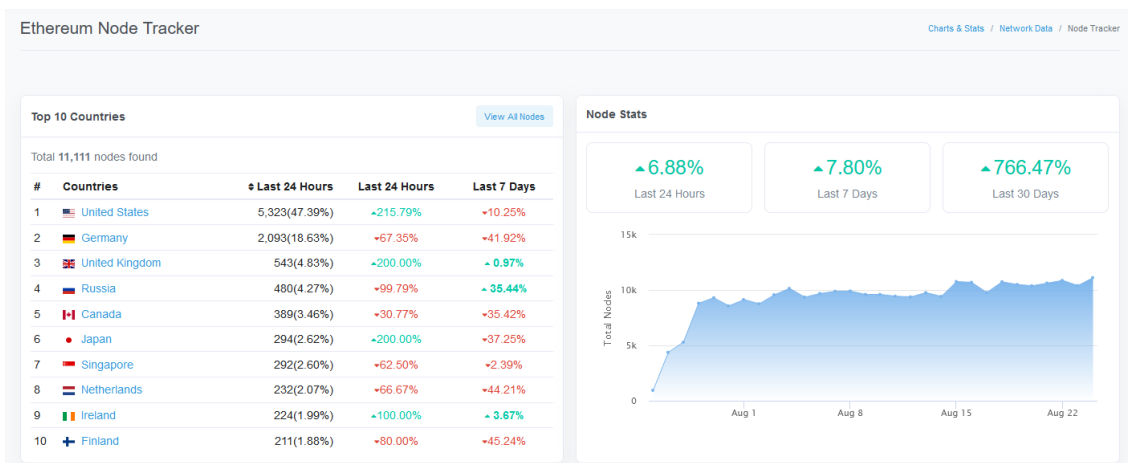
Znaczenie EVM dla protokołu Ethereum to zezwolenie każdemu dołączającemu do sieci na interakcję z interesującym go kontraktem, bez potrzeby zaufanej trzeciej strony. Wynik każdego wykonania jest gwarantowany przez w pełni deterministyczne inteligentne kontrakty.

2.5 Węzły oraz klienci

Ethereum to rozproszona sieć komputerów, czyli węzłów⁶⁰. Uruchamiają one oprogramowanie, których zadaniem jest weryfikacja bloków oraz transakcji umieszczanych do kolejnych bloków. Każdy z węzłów działa na równych prawach, uczestnicząc w sieci. Aplikacja oprogramowania, inaczej nazywana klient⁶⁰ musi być uruchamiana na komputerze, aby przekształcić go w węzeł Ethereum.

Węzeł to instancja oprogramowania klienta Ethereum, który jest połączony z innymi komputerami, które również uruchamiają oprogramowanie Ethereum. Razem tworzą zdecentralizowaną sieć. Klient to implementacja Ethereum, która weryfikuje dane pod kątem reguł aktualnego protokołu, zapewniając bezpieczeństwo sieci.

⁶⁰ Nodes and clients, <https://ethereum.org/en/developers/docs/nodes-and-clients/>



Rysunek 16: Statystyki wszystkich pełnych węzłów wraz z ich rozmieszczeniem na świecie z uwzględnieniem krajów, w których się znajdują

Źródło: <https://etherscan.io/nodetracker#>

Rozmieszczenie węzłów na świecie stale się zmienia. Głównie ze względu na politykę danego kraju. Jakiś czas temu Chiny całkowicie zakazały⁶¹ handlowania oraz wydobywania kryptowalut, dlatego wszyscy górnicy zmienili swoje położenie i w tym wypadku w znacznej większości przenieśli się do Stanów Zjednoczonych, głównie Teksasu⁶².

Węzły nie kopią łańcucha bloków Ethereum. Jednakże, wszyscy górnicy zazwyczaj uruchamiają własne węzły, aby skutecznie weryfikować i przekazywać transakcje Ethereum. Prowadzenie własnego węzła umożliwia korzystanie z Ethereum w sposób bezpośredni, prywatny, samowystarczalny oraz nie wymagający zaufania. Jednocześnie wspieramy sieć, utrzymując ją bardziej niezawodną i zdecentralizowaną.

Full Node⁶³ przechowuje całą historię blockchaina Ethereum. Uczestniczy w walidacji bloków, weryfikuje wszystkie bloki i stany. Obsługuje sieć i udostępnia dane na żądanie. Stanowi aktualnie lwią część wszystkich istniejących węzłów, ponieważ wszyscy górnicy uruchamiają pełny węzeł i uczestniczą w konsensusie.

Alternatywą dla pełnego węzła jest wersja Light⁶⁰. Zamiast pobierać wszystkie bloki jakie miały miejsce, węzły lekkie pobierają jedynie nagłówki bloków. Posiadają

⁶¹ Sofia Brooke, China Makes Cryptocurrency Transactions Illegal: An Explainer, <https://www.china-briefing.com/news/china-makes-cryptocurrency-transactions-illegal-an-explainer/>

⁶² Zhaoyin Feng, Why China's bitcoin miners are moving to Texas, <https://www.bbc.com/news/world-us-canada-58414555>

⁶³ Fares Alkudmani, What are the benefits of being an Ethereum Full Node, <https://www.quora.com/What-are-the-benefits-of-being-an-Ethereum-full-node>

one streszczone informacje o zawartości poszczególnych bloków. Każda inna informacja wymagana przez węzły Light pobierana jest z węzłów pełnych. Następnie węzeł lekki może niezależnie weryfikować dane, które otrzymuje. Węzły Light umożliwiają uczestnictwo użytkownikom w sieci Ethereum bez wydajnego sprzętu czy dużej przepustowości wymaganej do uruchomienia węzłów pełnych. Taki stan rzeczy drastycznie zmniejsza próg wejścia, co sprzyja większej decentralizacji węzłów. W przyszłości celem jest uruchamianie węzłów Light nawet na urządzeniach mobilnych. Warto zaznaczyć, że węzły lekkie nie uczestniczą w konsensusie. Czyli nie mogą być górnika⁵⁵ czy walidatorami⁵⁵, ale mogą uzyskać dostęp do łańcucha bloków Ethereum z taką samą funkcjonalnością jak węzły pełne. Na ten moment Ethereum nie obsługuje jeszcze dużej liczby węzłów Light, ale ich obsługa to obszar, który w bliskiej przyszłości ma się prężnie rozwijać.

Archive Node są mniej atrakcyjne dla zwykłych użytkowników, ponieważ przechowują historię, która jest trzymana w węzłach pełnych. Tworzą archiwalną historię stanów. Dane te reprezentują jednostki terabajtów, które są użyteczne dla wszelkich eksploratorów bloków tj. etherscan oraz dostawców portfeli kryptowalutowych czy narzędzi do analizy historycznych danych łańcucha.

2.6 Algorytm konsensusu

W każdym blockchainie, czyli rozproszonej bazie danych, wszystkie węzły sieci muszą dojść do wspólnego porozumienia odnośnie do aktualnego stanu sieci. Porozumienie to osiąga się za pomocą mechanizmów konsensusu⁶⁴. Jako konsensus rozumiemy to, że generalne porozumienie zostało spełnione. Weźmy za przykład pójście do kina z grupą znajomych. Jeżeli wszyscy zdecydują, że idą na konkretny film, wtedy konsensus jest osiągnięty. W najgorszym przypadku dochodzi do rozdzielenia. W kontekście blockchainów proces jest sformalizowany, aby osiągnąć konsensus potrzeba 51% wszystkich węzłów w sieci, które zgodzą się na kolejny globalny stan sieci. Mechanizm konsensusu pozwala rozproszonemu systemowi na wspólną pracę w bezpieczny sposób. Jest systemem, który pomaga również zapobiegać pewnym atakom. W teorii możliwy jest atak wykorzystujący 51% wszystkich węzłów, który umożliwi kontrolę nad konsensem sieci. Jednakże sieć zaprojektowana jest w taki sposób, żeby taki rodzaj ataku uniemożliwić. Na pewno pomaga również fakt wielkości całej sieci. Jeżeli weźmiemy za przykład

⁶⁴ Consensus mechanisms, <https://ethereum.org/en/developers/docs/consensus-mechanisms/>

blockchain Bitcoina oraz Ethereum, zgromadzenie zasobów w postaci mocy obliczeniowej oraz przede wszystkim prądu potrzebnego do jej zasilenia staje się niesamowicie drogi oraz wręcz niemożliwy do osiągnięcia. Szacunki takiego ataku są liczone dziesiątkach miliardów dolarów.

Najpopularniejszym oraz najłatwiejszym do zaimplementowania jest algorytm konsensusu o nazwie Proof of Work⁶⁵, w skrócie PoW, czyli dowód pracy. Aktualnie wykorzystują go Bitcoin oraz Ethereum. To jest właśnie tzw. kopanie, mining⁶⁶. Ono służy temu, żeby powiedzieć kto doda nowy blok do łańcucha bloków. W tej sieci mamy różne rodzaje węzłów. Są takie węzły, które biorą udział w tworzeniu nowych bloków, one są nazywane „górnikami”⁶⁶ oraz są takie węzły, które tylko obserwują. W tym wypadku mówimy o węźle górnika – nasłuchuje na nowe transakcje, czyli inne węzły mogą zgłosić, że chcą przelać środki, następnie transakcje są dystrybuowane po sieci, górnicy nasłuchują na nie i zbierają je do swojej puli, następnie wybierają jakąś część transakcji i próbują stworzyć nowy blok. Żeby nie było problemu, że 10 węzłów tworzy nowy blok w tym samym czasie, musimy spowolnić ten proces w jakiś sposób, także górnik musi dostarczyć jakiś dowód, że wykonał jakąś pracę polegającą na rozwiązaniu problemu obliczeniowego. W tym wypadku problem obliczeniowy to znalezienie takiego wyniku funkcji skrótu (liczonej z nagłówka bloku), który ma odpowiednią ilość zer na początku.

Jedyną możliwością odgadnięcia wyniku jest metoda prób i błędów, to się dzieje w pętli, górnik kopie blok zmieniając jedno pole nagłówka bloku, które powoduje, że zmienia się wynik funkcji skrótu. To pole nazywa się **Nonce**, znajduje się w każdym bloku. W momencie, gdy takiemu górnikowi uda się znaleźć taki **Nonce**, który powoduje, że wynik funkcji skrótu nagłówka bloku ma odpowiednią ilość zer na początku. Informuję sieć, że znalazł nowy blok, następnie inne węzły dodają ten blok do swojego łańcucha. Jeżeli w trakcie kopania zauważy, że komuś innemu udało się stworzyć nowy blok, to niestety musi przerwać swoją pracę i zacząć od nowa. Wszystkie inne węzły sieci po otrzymaniu nowego bloku, weryfikują poprawność wszystkich transakcji, muszą to zrobić, żeby sprawdzić, czy górnik nie oszukuje. Także mamy taką sieć peer-to-peer wielu węzłów, które nawzajem weryfikują wszystkie transakcje, które odbywają się w tej sieci oraz weryfikują, czy górnik wykonał pracę, która była wymagana. Jedną z zachęt dla

⁶⁵ Jake Frankenfield, Proof of Work (PoW), <https://www.investopedia.com/terms/p/proof-work.asp>

⁶⁶ Euny Hong, How Does Bitcoin Mining Work, <https://www.investopedia.com/tech/how-does-bitcoin-mining-work/>

górników za wykopanie nowego bloku jest przydzielenie sobie kryptowaluty. Górnik w nagrodzie za wykopanie nowego bloku, może dołączyć transakcje, która przelewa kryptowalutę z nieistniejącego konta na swoje konto (tworzy kryptowalutę). Drugą zachętą są prowizje od transakcji, które trafiają do górników.

W połowie września 2022, Ethereum zmienia swój algorytm konsensusu na Proof of Stake⁶⁷. Diametralnie zmieni to funkcjonowanie sieci. Wiele mówi się o tym, że wykorzystywanie PoW jest złe dla środowiska ze względu na ogromne zużycie energii. PoS rozwiązuje ten problem i zmniejsza go o ponad 99%, jednocześnie zmniejszając użycie specjalistycznego sprzętu. Przyczyni się to do znacznego ograniczenia problemu odpadów elektronicznych. Ethereum stanie się przyjazne dla środowiska oraz umożliwi jeszcze większą decentralizację poprzez zmniejszenie wymagań sprzętowych. PoS jest wykonywane przez walidatorów⁶⁸, które stakują posiadane ETH (umieszczają swoje środki na specjalnym kontrakcie), aby uczestniczyć w systemie. Walidator jest wybierany losowo, aby tworzyć nowe bloki oraz udostępniać je w sieci. Dzięki temu zdobywa nagrody w postaci etheru. Twórcy Ethereum już od momentu powstania w 2014 roku otwarcie mówili o przejściu z PoW na PoS. Zajęło to tyle czasu, ponieważ zrobienie tego w bezpieczny sposób jest znacznie trudniejsze technicznie.

2.7 Charakterystyka wykopanych bloków

Generalnie jest limit nowych transakcji, które mogą zmieścić się w bloku. Przede wszystkim wynika on z mocy obliczeniowej komputerów. Potrzebujemy ograniczyć wielkość bloków, żeby wszystkie komputery w sieci nadążały z ich weryfikacją. Kiedy węzeł w sieci dostaje nowy blok, on musi zweryfikować poprawność wszystkich transakcji w nim zawartych. Nowy blok w sieci Ethereum jest wykopywany co 12-14 sekund. Jest to ustalane poprzez parametr trudności⁶⁹. Parametr trudności jest dobierany dynamicznie, tak, żeby te bloki, były kopane w równym odstępie czasowym. Wyobraźmy sobie sytuację w której pojawia się coraz więcej górników, coraz więcej ludzi kupuje koparki i kopie. W związku z tym sumaryczna moc sieci rośnie. Sieć wymaga, aby te bloki były wykopywane w równych odstępach czasowych, co 12-14 sekund. Dlatego zwiększamy trudność

⁶⁷ Proof-of-Stake (PoS), <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>

⁶⁸ Jay Kurahashi-Sofue, What is a blockchain validator, <https://support.avax.network/en/articles/4064704-what-is-a-blockchain-validator>

⁶⁹ What is Ethereum difficulty, <https://2miners.com/eth-network-difficulty>

wykopywania kolejnych bloków poprzez żądanie większej ilości zer na początku (w wyniku funkcji skrótu). Stała długość między powstawaniem nowych bloków wynika z tego, żeby cała sieć nadgoniła, chcemy dać równe szanse wszystkim górnikom na świecie. Jeżeli zmniejszylibyśmy czas pomiędzy blokami to w tym momencie eliminujemy z kopania górników, którzy mają słabszy sprzęt, ponieważ nie mieliby żadnych szans na wykopanie nowego bloku.

2.8 Standardy ERC

Token jest reprezentacją czegoś w blockchainie. Tym czymś mogą być pieniądze, grafiki, akty własności nieruchomości, udziały w spółce, serwisy oraz wiele innych. Dzięki reprezentacji w postaci tokenów, możemy pozwolić inteligentnym kontraktom na wchodzenie z nimi w interakcje poprzez wymianę, tworzenie czy niszczenie.

Tokeny dzielimy na dwie główne kategorie:

a) Fungible⁷⁰ – wymienialne

Czyli takie, które są równoważne oraz wymienne 1:1 w stosunku do siebie. Na takiej samej zasadzie jak zwykle pieniądze, czyli 5 złotych to zawsze będzie 5 złotych. Możemy wymienić monetę 5 złotych na inną o takim samym nominale, ale ostatecznie będziemy mieć tyle samo pieniędzy.

b) Non-Fungible⁷⁰ – niewymienialne

Tokeny niewymienialne są niepowtarzalne i odrębne, takie jak akty własności, dzieła sztuki. Nigdy nie uświadczymy dwóch takich samych.

Istnieją również tokeny hybrydowe, które łączą cechy tokenów wymienialnych oraz niewymienialnych. Ich główne zastosowanie to chociażby wirtualne przedmioty w branży gier wideo.

Mimo iż koncept tokenów jest prosty, istnieją różne wariacje ich implementacji. Aby nie wymyślać koła na nowo za każdym razem oraz w celu zwiększenia bezpieczeństwa oraz ustrukturyzowania funkcjonalności poszczególnych tokenów powstały ich standardy⁷¹. Są to gotowe, w pełni darmowe szablony inteligentnych kontraktów, stworzone przez najwybitniejszych programistów do wykorzystania przez twórców nowych

⁷⁰ Andrea Knezovic, The Difference Between Fungible and Non-Fungible Tokens, <https://medium.com/udonis/the-difference-between-fungible-and-non-fungible-tokens-nfts-123df237b892>

⁷¹ Standards, <https://docs.openzeppelin.com/contracts/4.x/tokens>

projektów. Z racji tego, że są szeroko wykorzystywane zostały wielokrotnie przetestowane pod kątem podatności na wszelkie ataki. Co czyni je wyjątkowo bezpiecznymi.

```
// contracts/VLAToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.5;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract VLAToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Vistula", "VLA") {
        _mint(msg.sender, initialSupply);
    }
}
```

Rysunek 17: Kod kontraktu, którego celem jest stworzenie tokena Vistula w standardzie ERC-20

Źródło: Opracowanie własne

ERC-20⁷² jest aktualnie najbardziej rozpowszechniony standard dla tokenów wymiennych. Napisanie kontraktu w oparciu o ten standard daje nam możliwość utworzenia kryptowaluty na blockchainie Ethereum.







Na załączonym obrazu widać w jak łatwy sposób można utworzyć token o nazwie Vistula, którego skrót to VLA. W momencie umieszczania go na łańcuchu konieczne jest podanie w konstruktorze jaka ma być początkowa pula tokenów. Aby nasz kontrakt VLA-Token uzyskał wszystkie funkcjonalności ze standardu ERC-20, konieczne jest skorzystanie z biblioteki OpenZeppelin⁷³. W kodzie korzystamy z mechanizmu dziedziczenia, aby obdarować nasz kontrakt rozszerzeniami ze standardu ERC-20.

Ostatnimi czasy bardzo dużą popularność zyskało pojęcie NFT (non-fungible token), czyli w skrócie niewymienne tokeny. Jest to nic innego jak ładniejsza nazwa na standard ERC-721⁷⁴. Dzięki tej technologii artyści mają możliwość tokenizowania oraz stałego zarabiania na swoich dziełach na rynku wtórnym. Zamiast otrzymywać jednorazową zapłatę ze sprzedaży danego obrazu, grafiki czy animacji. Dostają również ustalony procent od każdej kolejnej transakcji sprzedaży na rynku wtórnym.

⁷² ERC20, <https://docs.openzeppelin.com/contracts/4.x/erc20>

⁷³ OpenZeppelin, <https://moonbeam.network/community/projects/openzeppelin/>

⁷⁴ ERC721, <https://docs.openzeppelin.com/contracts/4.x/erc721>

COLLECTION	VOLUME ▾	% CHANGE ↕	FLOOR PRICE ↕	SALES ↕
 ENS: Ethereum Name Service	298	-24%	< 0.01	2,981
 Women Ape Yacht Club	264	-9%	0.07	234
 Otherdeed for Otherside	262	-2%	1.80	70
 Bored Ape Yacht Club	245	-55%	80	3
 Mutant Ape Yacht Club	231	+47%	14.90	15
 RENG Black Box	207	+103%	0.45	403

Rysunek 18: Ranking projektów NFT, które osiągnęły największy wolumen w przeciągu ostatnich 24h

Źródło: <https://opensea.io/rankings>

Aby kupić NFT konieczne musimy posiadać natywną kryptowalutę Ethereum, czyli Ether. Wielkość tantiemów dla twórców zależy od ustawionego przez nich procenta od sprzedaży konkretnego NFT w danej kolekcji. Zwykle wachają się od 2.5 do nawet 10%. Nie jest to stricte wbudowane w standardzie, ale jest to stała praktyka na rynkach do sprzedaży NFT⁷⁵ tj. OpenSea, LooksRare.

ROZDZIAŁ III

3. Inteligentne kontrakty

Inteligentne kontrakty⁷⁶ to podstawowe elementy budulcowe aplikacji Ethereum. Są to programy komputerowe przechowywane na blockchainie, które pozwalają przekształcić tradycyjne umowy na ich cyfrowe odpowiedniki. Termin inteligentnych kontraktów powstał już w 1994 roku⁷⁷ za sprawą amerykańskiego informatyka Nick'a Szabo. Zdefiniował je jako skomputeryzowane protokoły transakcyjne, które realizują warunki

⁷⁵ Eric James Beyer, Dive into three of the top NFT marketplaces, <https://nftnow.com/guides/dive-into-three-of-the-top-nft-marketplaces/>

⁷⁶ Jake Frankenfield, What Are Smart Contracts on the Blockchain and How They Work, <https://www.investopedia.com/terms/s/smart-contracts.asp>

⁷⁷ Smart Contracts, <https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>

w nich zawarte. Twórcy Ethereum poszli dokładnie w tym kierunku. Kontrakty zachowują się tak, jak zostały zaprogramowane. W skrócie kod stanowi prawo. Mogą być alternatywną formą pośrednictwa, co znacznie redukuje koszty. Im mniej pośredników po drodze, tym więcej zyskuje użytkownik.

3.1 Ulepszanie kontraktów

Umieszczając kontrakty na łańcuchy głównym lub testowym liczymy się z tym, że są od tego momentu niezamienialne. Sama w sobie logika poszczególnych funkcji nie może zostać zmieniona i zostanie ona na zawsze w takiej postaci na łańcuchu. Jednakże wartości zmiennych czy aktualny balans środków kontraktu ulegają ciągłym zmianom ze względu na to, że użytkownicy wchodzi w interakcje z funkcjami kontraktu.

Z jednej strony jest to jedna z ich największych zalet. Jednak każdy kij ma dwa końce. Może być to wielki problem, jeżeli w kodzie kontraktu są krytyczne błędy, które nie zostały zauważone na etapie ich tworzenia oraz audytu bezpieczeństwa. Kod kontraktów wszystkich największych projektów są z reguły transparentne, aby wzbudzić zaufanie wśród użytkowników.

Są jednak hackerzy, którzy wykorzystują to do złych celów. Specjalizują się w wykrywaniu luk w kodzie, które umożliwiają im przywłaszczenie sobie zdeponowanych środków na kontrakcie. Niestety jest to dosyć częsta praktyka i często straty idą w setki milionów dolarów. Szczególnie, gdy pominie się etap audytu bezpieczeństwa. Istnieją różne metody rozwiązania tego problemu, część z nich jest mniej lub bardziej elastyczna. Każda ma swoje wady i zalety.

Pierwszym, jednocześnie najprostszym oraz najmniej elastycznym sposobem na zobrazowanie ulepszenia kontraktu jest metoda parametryzacji. Nie zmienimy samej logiki konkretnych funkcji, ale możemy wpływać na wartości zmiennych w kontrakcie. Jako przykład może posłużyć nam typowy projekt NFT. W skrócie twórcy tworzą kontrakt, w którym decydują się na określoną liczbę NFT w danej kolekcji np. 10 000 sztuk.

Każda z nich jest oczywiście unikalna, ale razem tworzą cały projekt. Następnie decydują kto będzie mógł je stworzyć w fazie **mintu**⁷⁸, czyli tworzenia. Robią to poprzez

⁷⁸ What is minting an NFT, <https://www.nfi.edu/what-is-minting-an-nft/>

rozdawanie **whitelist**⁷⁹, w skrócie **WL**. Jest to lista adresów publicznych, które w pierwszej kolejności mogą dane NFT wybić, stworzyć.

```
function batchAddToWhitelist(  
    address [] calldata recipients,  
    uint64 [] calldata quantities  
) public onlyOwner {  
    for(uint64 i = 0; i < recipients.length; i++) {  
        _addressData[recipients[i]].aux = quantities[i];  
    }  
}
```

Rysunek 19: Funkcja w kontrakcie projektu NFT, która odpowiada za dodawanie adresów publicznych użytkowników po uprzednim umieszczeniu kontraktu na sieci głównej. Do późniejszego pierwszego stworzenia tokenów NFT przez użytkowników.

Źródło: <https://etherscan.io/address/0xecdeb3fec697649e08b63d93cab0bb168c35eec5#code>

Z tego powodu twórcy tworzą funkcje w kontrakcie, która przyjmuje argumenty tj. listę adresów publicznych oraz listę ilości NFT, które dany adres może utworzyć. W taki sposób twórcy nie muszą znać wszystkich adresów od samego początku. Tylko mogą na spokojnie umieścić kontrakt na sieci głównej oraz z czasem sukcesywnie dodawać partiami kolejne adresy oraz korespondujące ilości sztuk.

Największym minusem tej metody jest to, że jeżeli na samym początku tworzenia kontraktu nie wpadliśmy na zaprogramowanie konkretnej logiki czy funkcjonalności to niestety już nic nie możemy zrobić. Chyba, że napiszemy kontrakt od nowa.

Drugim sposobem jest migracja do nowego zestawu kontraktów. Twórcy informują swoich użytkowników, że następuje aktualizacja protokołu i zmieniają przekierowywania w swoich aplikacjach webowych do ich nowych, ulepszonych wersji. Nowe wersje nie mają żadnego powiązania z tymi poprzednimi, tworzą osobny byt. Przykładem może być najpopularniejszy DEX, czyli giełda zdecentralizowana – Uniswap⁸⁰. Wyróżniamy trzy wersje Uniswapa v1, v2 oraz najnowszą v3.

⁷⁹ What is an NFT Whitelist and How do you get whitelisted, <https://cyberscrilla.com/what-is-an-nft-whitelist-and-how-do-you-get-whitelisted/>

⁸⁰ What is Uniswap, <https://www.coinbase.com/pl/learn/crypto-basics/what-is-uniswap>

Trzecim, najprawdziwszym sposobem na ulepszenie kontraktów są **proxies**⁸¹. Użytkownicy, którzy korzystają z kontraktów zbudowanych w oparciu o ten mechanizm nawet nie zdają sobie sprawy z tego, że coś się zmieniło czy zostało zaktualizowane. Skonfigurowanie architektury kontraktu proxy pozwoli skorzystać z nowych wdrożonych kontraktów, jeśli zajdzie potrzeba aktualizacji. Kontrakty proxy przechowują adresy wcześniej wdrożonych kontraktów i przekierowują wywołania do aktualnie obowiązującej logiki. Jeżeli ktoś zaktualizuje logikę kontraktu wdrażając go do sieci głównej, wystarczy zmienić zmienną referencyjną w kontrakcie proxy za pomocą nowego adresu kontraktu. Warto wspomnieć również o tym, że zmiany wprowadzane za pomocą architektury proxy często podlegają głosowaniu. Użytkownicy protokołów mają prawo głosu, jeżeli posiadają dedykowane tokeny projektu. Czyni ich to pełnoprawnymi udziałowcami. Bez zatwierdzenia w głosowaniu zmian, twórcy nie są w stanie nic zrobić, a użytkownicy mogą ocenić, czy zgadzają się na proponowane ulepszenia.

ROZDZIAŁ IV

4. Ewolucja Web 3.0

Web, czyli w rozwinięciu **World Wide Web** to ogólnosiwiatowa sieć internetowa. Początki Internetu sięgają końcówki lat 60⁸², jednakże w celu zrozumienia powyższego terminu musimy przenieść się do początku lat 90. Wtedy zaczęły powstawać pierwsze strony internetowe.

4.1 Web 1.0 vs Web 2.0 vs Web 3.0

Od tamtego momentu uznajemy początek Internetu pierwszej generacji, w skrócie Web 1.0⁸³. Na dzień dzisiejszy wydają się być bardzo prymitywne, w ogóle nie przypominają dzisiejszych bardzo rozbudowanych aplikacji webowych. Były robione w formie **read-only**, czyli mogliśmy jedynie czytać ich zawartość.

⁸¹ Andrej Rakic, Upgradeable Smart Contracts – Proxy Pattern, <https://mvpworkshop.co/blog/upgradeable-smart-contracts-proxy-pattern/>

⁸² A Brief History of the Internet, https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml

⁸³ Comparison between Web 1.0, Web 2.0 and Web 3.0, <https://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference/>

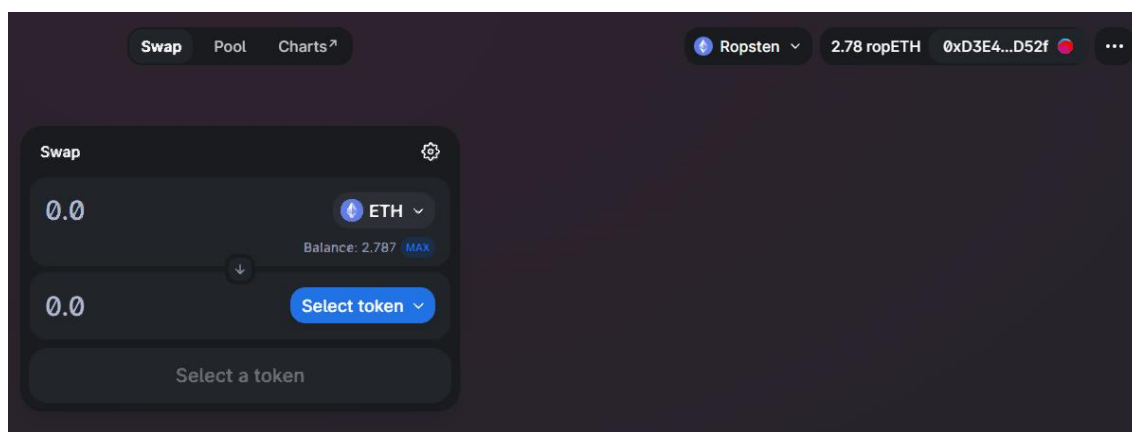
Kolejnym krokiem ewolucyjnym było umożliwienie użytkownikom większą interakcję z portalem, na którym się znajdują. Poprzez zastosowanie formatu **read-write**, czyli zarówno czytanie jak i zapisywanie treści. Okres Web 2.0⁸³ rozpoczął się wraz z pojawieniem platform mediów społecznościowych tj. Twitter czy Facebook. Wspomniane strony umożliwiły sprawne komunikowanie się, generowanie oraz przesyłanie treści między użytkownikami. W dzisiejszych czasach bardzo trudno znaleźć osoby, które z takich aplikacji nie korzystają. To pokazuje jak bardzo stały się one nieodłącznym elementem w naszym codziennym funkcjonowaniu. Z jednej strony nie ma w tym nic złego, ponieważ dzięki nim możemy w łatwy sposób utrzymywać znajomości, które w ciągu życia nawiązaliśmy. Z drugiej strony twórcy platform mają niebywale dużą kontrolę nad użytkownikami. To od nich zależy jakie treści zostaną nam wyświetlone a jakie zablokowane. Z pozoru błaha sprawa, ale jeżeli wejdziemy głębiej i weźmiemy pod uwagę, że setki milionów użytkowników korzystają z nich codziennie. To czyni je niebywale silnym instrumentem politycznym, manipulacyjnym. Na pierwszy rzut oka wszystko jest darmowe, ale tylko z pozoru. Twórcy śledzą każdy nasz ruch, nawet zwykłe wyszukiwanie interesujących nas treści w takich aplikacjach bardzo dużo mówi o użytkowniku. Dzięki temu mogą sprawniej układać reklamy, aby nas bardziej zachęcić do zakupu konkretnych produktów. W modelu **Web 2.0** model przychodów opiera się o reklamy. Mimo iż użytkownicy mogą tworzyć treści to niestety nie są ich właścicielami, tylko właściele portali.

Najnowszą generacją Internetu jest Web 3.0⁸⁴, czyli miejsce stworzone w formie **read-write-own**. Posiadamy te same funkcjonalności co generację wcześniej, ale dodatkowo stajemy się właścicielem treści, którą tworzymy. Jest on odpowiedzią na to, że aktualnie większość Internetu, który ludzie znają i używają, opiera się na zaufaniu kilku prywatnym firmom, które działają w swoim interesie maksymalizacji zysków. Główne cechy Web 3.0 to decentralizacja, czyli w skrócie równość użytkowników. Każdy ma równe prawo w niej uczestniczyć oraz nikt nie może być z niej wykluczony. Dzięki technologii blockchain, nowa generacja Internetu ma wbudowany natywny system płatności w postaci kryptowalut. Kluczowe w aplikacjach Web 3.0 jest to, że wszelkie dane są na blockchainie, a nie na serwerze u konkretnego dostawcy. To sprawia, że możemy korzystać z tych samych danych w obrębie różnych aplikacji. Tworzy to zdrową rywalizację pomiędzy aplikacjami, ponieważ twórcy zdają sobie sprawę z tego, że użytkownik może

⁸⁴ Introduction to Web3, <https://ethereum.org/en/web3/>

w dowolnym momencie pójść do innego usługodawcy w przeciwieństwie do Facebooka, z którym wiele ludzi jest przywiązanych. Przez lata zbierali znajomości na tym portalu i jeżeli zdecydują się go opuścić to cała jego zawartość zostanie na tym portalu.

Podsumowując, Internet nowej generacji nie ma na celu wyeliminowanie jego poprzedniej wersji. Ma jedynie dać wybór użytkownikom, inną alternatywę. Docelowo obie generacje będą funkcjonować równolegle, ale zdecydowanie będziemy obserwować dużą migrację ludzi w kierunku Web 3.0, ponieważ daje wiele benefitów.



Rysunek 20: Przykład aplikacji zrobionej w modelu Web 3.0 – giełda zdecentralizowana Uniswap. W skrócie DEX - decentralized exchange.

Źródło: <https://app.uniswap.org/#/swap>

4.2 Wytłumaczenie aplikacji zdecentralizowanych w Web 3.0

Do tej pory aplikację zdecentralizowaną rozumieliśmy jako inteligentny kontrakt w sieci Ethereum. Jednakże w Internecie Web 3.0, aplikacje zdecentralizowaną⁸⁵ rozumiemy jako platformę, która łączy cechy aplikacji Web 2.0 z technologią blockchain.

Cechą wspólną jest frontend, czyli interfejs, który umożliwia użytkownikom wchodzenie w interakcje z aplikacją. W części ukrytej jest backend, czyli warstwa logiki aplikacji, która wykorzystuje inteligentne kontrakty. W standardowej wersji aplikacji internetowej za warstwą logiki stoi jakiś scentralizowany serwer, który przetrzymuje wszystkie

⁸⁵ Jake Frankenfield, Decentralized Applications (dApps), <https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp>

potrzebne informacje. Tutaj czegoś takiego nie ma, przynajmniej w powyższym przykładzie, który pokazuje interfejs aktualnie najpopularniejszej giełdy zdecentralizowanej – Uniswap.

W tradycyjnych aplikacjach często logujemy się za pomocą Facebooka, Google. Nie musimy zakładać konta od zera, w dużej części wystarczy, że posiadamy konto jednej z dwóch wyżej wymienionych platformach. Aplikacje zdecentralizowane z kolei eliminują problem logowania. W tym przypadku – naszym id, tożsamością jest nasz adres sieci Ethereum – nie musimy się nigdzie logować. W sieci Ethereum najczęściej korzystamy z wcześniej opisanego MetaMaska. Posiada on nasz klucz prywatny, który tylko my posiadamy i to jest wystarczające, żeby nas uwierzytelnić w dowolnej aplikacji zdecentralizowanej. Nie trzeba weryfikować żadnych maili, przypominać haseł, etc.

ROZDZIAŁ V

5. Aplikacja kontraktowa – protokół imitujący lokatę

Aplikacja z sektora zdecentralizowanych finansów zrealizowana w systemie blockchain. Skupia się na mechanizmie protokołu, który generuje nagrody w zamian za umieszczenie dopuszczonych tokenów na określony czas. Trzonem aplikacji są dwa kontrakty. Pierwszym z nich jest **VLAToken**, który odpowiada za utworzenie kryptowaluty o nazwie Vistula w standardzie ERC-20. Drugim kontraktem jest **TokenYield**, który odpowiada za mechanizm imitujący lokatę bankową. Użytkownicy protokołu są wynagradzani tokenami Vistula. Wielkość nagrody jest zależna od wartości depozytu środków, które umieścili w kontrakcie. Przyjmujemy zasadę, że 1 ETH jest równoważny pod względem ceny tokenowi Vistula.

Aplikacja działa w ten sposób, że najpierw wysyłamy transakcje do sieci, aby umieściła kontrakt **VLAToken**, który generuje określoną ilość kryptowaluty Vistula. W kolejnej transakcji umieszczamy w sieci kontrakt **TokenYield**. Do którego przesyłane jest 99.9% wszystkich wygenerowanych tokenów Vistula. Następnie w zakresie funkcji **add_allowed_tokens** dajemy zielone światło dla tokenów Vistula, WETH oraz FAU. Od tego momentu powyższe tokeny mogą być brać udział w stake'ingu (czyli wspomniana wcześniej lokata). Równolegle korzystamy z wyroczni i każdemu tokenowi nadajemy wartość. W tym wypadku WETH będzie 1:1 w stosunku do ceny ETH. Natomiast Vistula

token oraz FAU token będzie 1:1 do DAI. Czyli stablecoina, którego wartość jest równa 1:1 do dolara amerykańskiego. Czyli 1 DAI = 1 dolar.

5.2 Język do pisania kontraktów - Solidity

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.5;
3
4 import '@openzeppelin/contracts/token/ERC20/ERC20.sol';
5
6 contract VLAToken is ERC20 {
7     constructor(uint256 initialSupply) ERC20('Vistula', 'VLA') {
8         _mint(msg.sender, initialSupply);
9     }
10 }
```

Rysunek 21: Kod kontraktu umożliwiający stworzenie kryptowaluty Vistula w standardzie ERC 20

Źródło: Opracowanie własne

Jest najpopularniejszym językiem do pisania kontraktów w ekosystemie Ethereum. Jest zorientowany obiektowo. Słowo kluczowe **contract** jest odpowiednikiem **class** w innych językach. Należy do kategorii języków wysokopoziomowych, tak jak chociażby użyty w projekcie Python⁸⁶. Co sprawia, że jest stosunkowo prosty w opanowaniu. Oprócz tego jest językiem statycznie typowanym. Wspiera mechanizm dziedziczenia, co zresztą widać na powyższym obrazku. Kontrakt **VLAToken** dziedziczy funkcjonalności z kontraktu **ERC 20**, który jest jednocześnie najpopularniejszym standardem tokenu z biblioteki OpenZeppelin. Dodatkowo Solidity⁸⁷ umożliwia tworzenie własnych, zaawansowanych struktur danych, które bardzo często przydają się w tworzeniu funkcjonalności kontraktu. Bardzo ważnym aspektem języka Solidity, są jego używane wersje. W powyższym przykładzie korzystam z wersji nie starszej niż **0.8.5**, ale równie dobrze w kompilatorze mógłbym wybrać wersję najnowszą, czyli **0.8.16** a i tak edytor kodu nie zwróciłby błędu. Aby zabezpieczyć wsteczną kompatybilność kontraktów posługujemy się konkretną wersją. Jeżeli w **pragma** wybierzemy starszą wersję Solidity to zwyczajnie nie będziemy mieli do dyspozycji nowych usprawnień języka.

⁸⁶ About Python, <https://www.python.org/about/>

⁸⁷ Solidity, <https://docs.soliditylang.org/en/v0.8.15/>

Koniecznym elementem każdego kontraktu jest jego licencja, którą specyfikujemy w pierwszej linijce pliku. W ten sposób przyczyniamy się do rozwoju wolnego oprogramowania i udostępniamy innym nasz kod, który stworzyliśmy. Rozwiązujemy problem związany z prawami autorskimi, aby zwiększyć zaufanie do naszego kodu.

Ciekawą cechą jest to, że język nie posiada struktury danych typu **float**. Powód jest bardzo prosty. Dzielenie przez liczby zmiennoprzecinkowe jest bardzo niedokładne. Jest to niedopuszczalnie, jeżeli chodzi o aspekt finansów. Dlatego główna kryptowaluta Ethereum, Ether jest podzielna do 18 miejsc po przecinku.

5.3 Narzędzie do testowania i wdrażania kontraktów - Brownie

Brownie⁸⁸ jest framework’iem, który stanowi szkielet całego projektu. To platforma programistyczna oparta o język Python, która służy do pracy z inteligentnymi kontraktami. Bardzo ułatwia pracę oraz pozwala skupić się na najważniejszych aspektach rozwoju oprogramowania tj. chociażby odpowiednie napisanie testów automatyzacyjnych czy skryptów, które mają umieścić naszą aplikację w sieci blockchain.

5.4 Osobisty blockchain Ethereum - Ganache

Pisanie skryptów, które przeprowadzają transakcje wymaga bardzo częstego uruchamiania aktualnie posiadanego kodu. W takiej sytuacji optymalnym środowiskiem jest osobista sieć blockchain, którą udostępnia nam Ganache⁸⁹. Jest to narzędzie, które umożliwia w bardzo prosty sposób uruchomienie sieci blockchain Ethereum w naszym lokalnym środowisku. Jest ekstremalnie szybkie, dzięki czemu mamy błyskawicznie odpowiedź (w porównaniu do sieci testowych) czy nasze skrypty działają poprawnie. Ganache jest dostępny w dwóch wersjach. W formie aplikacji desktopowej lub jako aplikacja w terminalu, czyli **ganache-cli**. Ta druga jest wbudowana w Brownie, dlatego nie musimy jej dodatkowo pobierać.

⁸⁸ Brownie, <https://eth-brownie.readthedocs.io/en/stable/>

⁸⁹ What is Ganache Blockchain, <https://101blockchains.com/ganache-blockchain/>

5.5 Wyrocznie – zdecentralizowane źródła danych

W projekcie korzystam z wyroczni⁹⁰ od ChainLink⁹¹. Umożliwiają one dostarczenie danych ze świata zewnętrznego do sieci Ethereum. Blockchain sam w sobie nie ma takiej funkcjonalności, dlatego powstało wiele firm, które to umożliwiają. Dane oczywiście są zbierane z wielu źródeł jednocześnie, a następnie porównywane ze sobą, aby nie doszło do nadużyć. Dzięki temu znacznie rozszerzają się możliwości tworzenia inteligentnych kontraktów, ponieważ w oparciu o takie dane możemy stworzyć wiele ciekawych rozwiązań. W aplikacji wykorzystałem wyrocznie w celu uzyskania informacji o cenie tokena, konkretnie ETH oraz DAI. W tym WETH, który cenę ma 1:1 do ETH.

WETH to nakładka standardu ERC-20 na ETH.

5.6 Mock – symulator prawdziwych obiektów

Korzystanie z Mock'ów⁹² to bardzo popularna metoda, która symuluje zachowanie rzeczywistych obiektów w kontrolowany sposób. Aplikacja wykorzystuje Mocki w celu imitacji warunków środowiska sieci testowych oraz sieci głównej. W protokole aplikacji kontraktowej, Mocki wykorzystywane są wtedy, kiedy korzystamy z łańcuchu sieci lokalnej Ganache. Jest to spowodowane tym, że w naszej sieci lokalnej nie ma dosłownie nic, ponieważ działa ona jedynie na naszym komputerze. Za każdym razem uruchamiana jest od zera i wszystkie poprzednio wykonane akcje są zwyczajnie zapominane. Dlatego, żeby poprawnie zasymulować warunki sieci głównej, konieczne jest skorzystanie z nich. Sieci testowe z kolei mają swój stan, który jest trwały i ulega ciągłym zmianom poprzez dodawanie coraz to nowszych kontraktów. Dlatego możemy skorzystać z nich, które zostały w niej wcześniej umieszczone.

⁹⁰ What is a Blockchain Oracle, <https://chain.link/education/blockchain-oracles>

⁹¹ Chainlink, <https://www.investopedia.com/chainlink-link-definition-5217559>

⁹² How to mock Solidity smart contracts for testing, <https://ethereum.org/en/developers/tutorials/how-to-mock-solidity-contracts-for-testing/>

5.7 Testy jednostkowe oraz testy integracyjne

Inteligentne kontrakty są świetnym przykładem aplikacji w których testowanie⁹³ jest absolutnie obowiązkowym elementem. Głównym powodem jest to, że po umieszczeniu ich na sieci głównej są niezamienialne (oprócz architektury Proxy) oraz ich kod jest transparentny. Czyli ludzie o złych zamiarach mogą od razu przejść do działania w kierunku znalezienia ewentualnych błędów, które umożliwią im przejęcie środków z kontraktu lub zwyczajnie jego przejęcie. Lista możliwych wektorów ataków jest naprawdę długa i zależy od tego jaki rodzaj aplikacji tworzymy.

Z przedstawionych wyżej powodów w branży blockchain – testowanie jest rzeczą niesamowicie istotną. W aplikacji dzielimy testy na jednostkowe oraz integracyjne. Te pierwsze sprawdzają czy konkretna, pojedyncza funkcja działa tak jak zakładaliśmy. Drugie z kolei łączą kilka funkcji, tworząc osobne transakcje – sprawdzając w ten sposób czy wszystko działa prawidłowo.

ROZDZIAŁ VI

6. Uruchomienie projektu

Jestem wielkim zwolennikiem Visual Studio Code⁹⁴, który traktuję jako swoje ulubione narzędzie do pracy z projektami. Jego możliwości są praktycznie nieograniczone, ponieważ ma wbudowaną opcję rozszerzeń, czyli przeróżnych dodatków tworzonych przez ludzi z całego świata. Dosłownie do każdego języka znajdzie się wiele ciekawych rozwiązań, które ułatwiają kodowanie. Ma wbudowaną konsolę, w której możemy uruchamiać skrypty co znacznie ułatwia pracę, ponieważ nie potrzebujemy dodatkowego okienka. Świetnie współpracuje z gitem oraz formatowanie kodu jest dziecinne proste. Posiada również wiele skrótów klawiszowych, które umożliwiają szybką modyfikację w wielu miejscach naraz.

⁹³ Testing smart contracts, <https://ethereum.org/en/developers/docs/smart-contracts/testing/>

⁹⁴ What is Visual Studio Code, <https://www.educba.com/what-is-visual-studio-code/>

6.1 Instalacja oraz konfiguracja środowiska edytora kodu - Windows

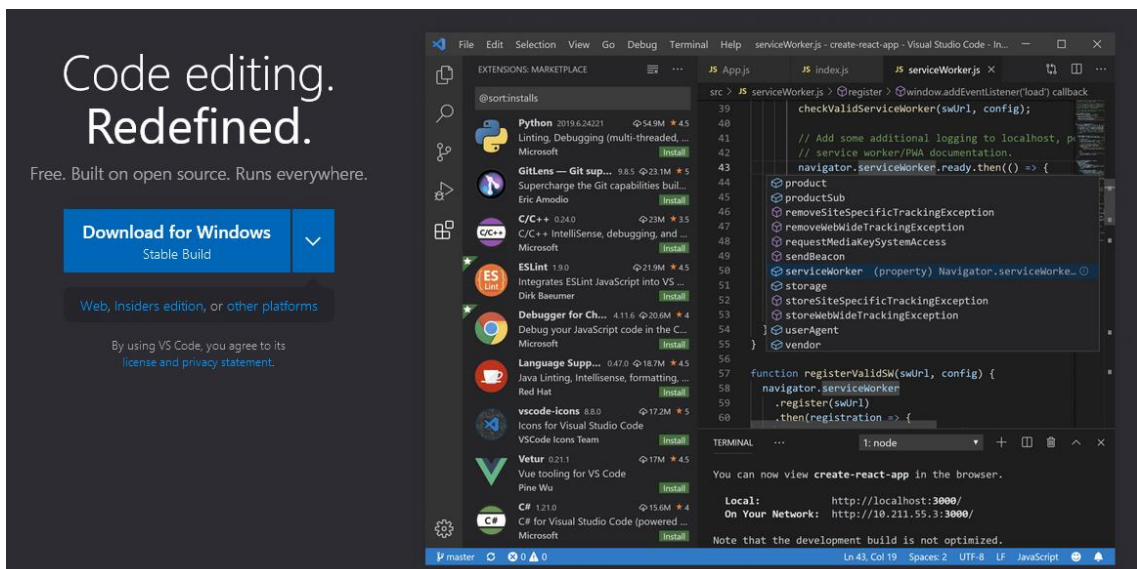
W pierwszej kolejności musimy wejść na oficjalną stronę Pythona, aby pobrać jego najnowszą wersję.



Rysunek 22: Zakładka z oficjalnej strony Pythona z której możemy pobrać jego najnowszą wersję

Źródło: <https://www.python.org/downloads/>

Następnie możemy przejść do instalacji Visual Studio Code, wraz z potrzebnymi rozszerzeniami. W tym celu musimy udać się na oficjalną stronę celem pobrania najnowszej wersji edytora kodu.

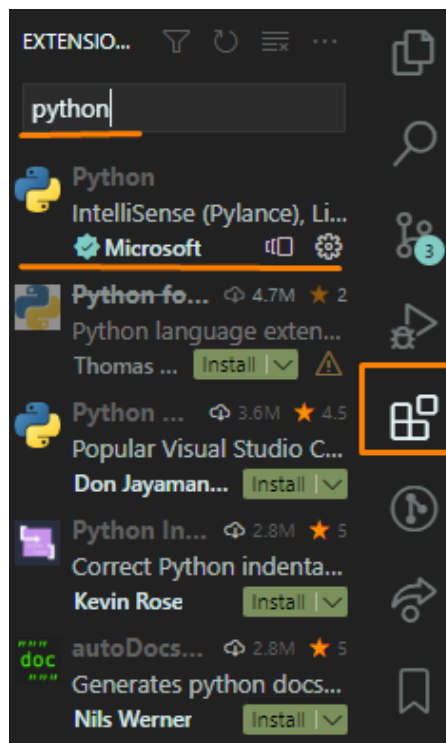


Rysunek 23: Oficjalna strona Visual Studio Code z której możemy pobrać najnowszą wersję edytora kodu

Źródło: <https://code.visualstudio.com/>

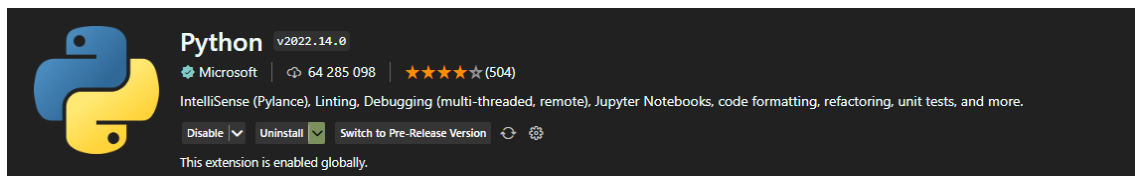
Po wstępnej instalacji możemy przejść do instalacji dodatkowych rozszerzeń, które umożliwią pracę z projektem. W tym wypadku wystarczy załadować jedno, które umożliwi pracę z interpreterem Pythona. Skrót klawiszowy **ctrl + shift + x** przeniesie nas do okienka, w którym możemy wybrać interesujące nas rozszerzenie oraz pobrać je.

W lewym górnym rogu klikamy w zaznaczony element i wpisujemy interesujące nas rozszerzenie. Następnie wybieramy i instalujemy.



Rysunek 24: Okienko, w którym musimy wpisać nazwę rozszerzenia, które chcemy zainstalować

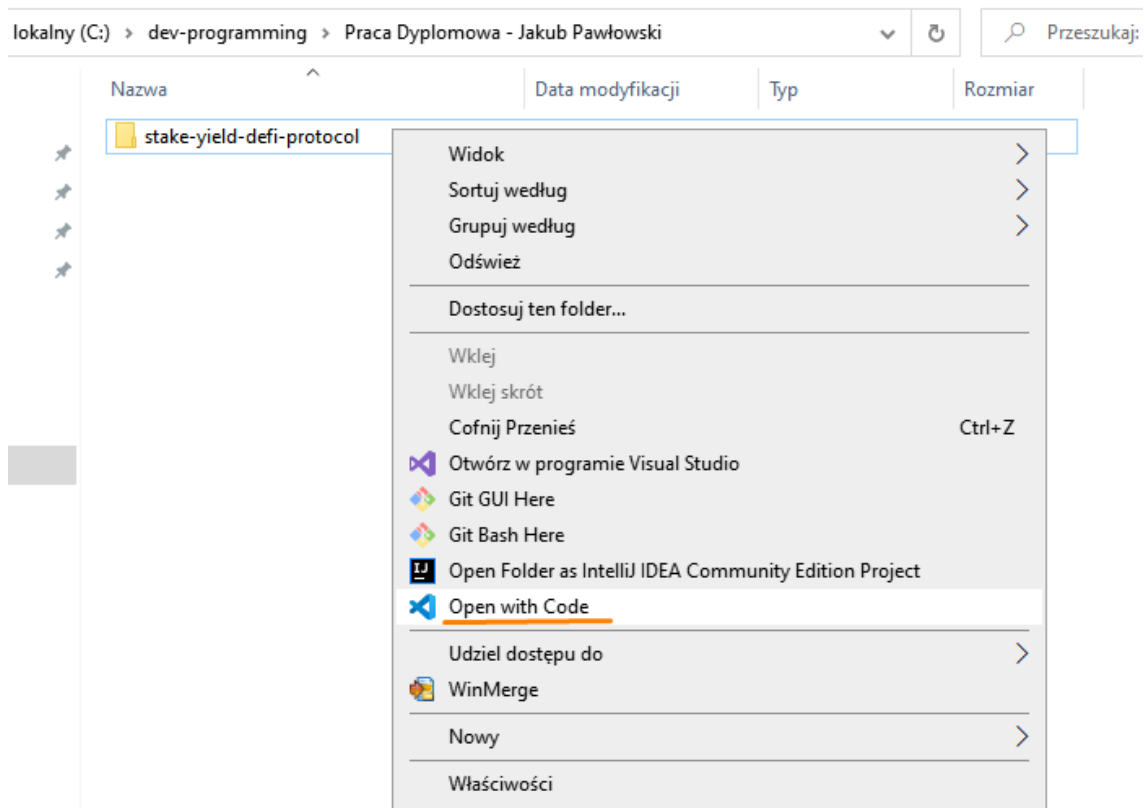
Źródło: Opracowanie własne



Rysunek 25: Rozszerzenie, które musimy zainstalować, żeby VSC mógł obsługiwać język Python

Źródło: Opracowanie własne

Kolejnym krokiem jest kliknięcie prawego przyciska myszy na folder, w którym znajduje się projekt. W taki sposób możemy otworzyć aplikację w pobranym VSC.



Rysunek 26: Najszybsza metoda uruchomienia pliku z aplikacją w środowisku Visual Studio Code

Źródło: Opracowanie własne

6.2 Instalacja Brownie oraz pozostałych komponentów

W celu uruchomienia aplikacji na komputerze konieczne jest wykonanie kilku kluczowych kroków. Pierwszym z nich jest instalacja potrzebnych bibliotek z których korzystam w projekcie. W celu zagregowania, większość z nich umieściłem w pliku **requirements.txt** wraz z wymaganymi wersjami.

Bardzo proszę wejść do folderu z aplikacją za pomocą Visual Studio Code oraz otworzyć terminal za pomocą klawiszowego skrótu **ctrl + `** Następnie wpisać następujące komendy:

```
pip install -r requirements.txt
```

```
pip install eth-brownie
```

Niestety framework Brownie jest na tyle rozbudowany, że konieczne jest uruchomienie dodatkowej komendy, stricte dla tej paczki. Bardzo ważna jest kolejność, ponieważ zależności z pliku tekstowego są wymagane dla poprawnej instalacji Brownie.

6.3 Zalecane polecenia do użycia w projekcie

Z racji tego, że jest to projekt czysto backendowy nie ujrzymy przejrzystego UI⁹⁵ z którym możemy wejść w interakcje. Dlatego kluczowe jest umiejętne posługiwanie się konsolą w celu prześledzenia jego funkcjonalności.

Dla ułatwienia skorzystałem z dodatkowego pliku **package.json** w którym utworzyłem skróty dla szybszego wprowadzania komend do konsoli.

```
"scripts": {  
  "compile": "brownie compile",  
  "ganache": "brownie run scripts/main.py",  
  "goerli": "brownie run scripts/main.py --network goerli",  
  "rinkeby": "brownie run scripts/main.py --network rinkeby",  
  "test": "brownie test"  
},
```

Rysunek 27: Wycinek z pliku `package.json`, który znajduje się w folderze z aplikacją

Źródło: Opracowanie własne

W pierwszej kolejności musimy skompilować kontrakty do kodu bajtowego, aby móc wykonać jakiegokolwiek działania w celu umieszczenia ich do sieci lokalnej lub testowej.

npm run compile

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> npm run compile  
  
> stake-yield-defi-protocol@1.0.0 compile  
> brownie compile  
  
Brownie v1.16.4 - Python development framework for Ethereum  
  
Project has been compiled. Build artifacts saved at C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol\build\contracts
```

Rysunek 28: Wynik prawidłowego skompilowanego kodu kontraktów w konsoli w edytorze VSC

Źródło: Opracowanie własne

⁹⁵ User interface (UI), <https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI>

Następnie możemy umieścić kontrakty składające się na protokół w łańcuchu lokalnym Ganache. Jest to najczęściej używana przeze mnie komenda podczas tworzenia projektu, ponieważ w bardzo szybki sposób możemy uzyskać odpowiedź czy wszystko działa poprawnie. Wykonanie wszystkich transakcji zajmuje najmniej czasu.

npm run ganache

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> npm run ganache
> stake-yield-defi-protocol@1.0.0 ganache
> brownie run scripts/main.py

Brownie v1.16.4 - Python development framework for Ethereum

StakeYieldDefiProtocolProject is the active project.

Launching 'ganache-cli.cmd --accounts 10 --hardfork istanbul --gasLimit 12000000 --mnemonic brownie --port 8545'...

Running 'scripts\main.py::main'...
Transaction sent: 0xf9bd848f552923278f51919b01ab6301de4b9571817b68e414a2aa9b6b5397cb
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 0
VLAToken.constructor confirmed Block: 1 Gas used: 635093 (5.29%)
VLAToken deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

Transaction sent: 0xb45763364540b6f0b26c51e6a02c0751d4659dbc1fb6419bd5e8448a77df76e6
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 1
TokenYield.constructor confirmed Block: 2 Gas used: 1026154 (8.55%)
TokenYield deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent: 0xe358cee4223bfab38ea773c42e5bbb30796755b900159f12b5c9b523c060cb58
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 2
VLAToken.transfer confirmed Block: 3 Gas used: 51109 (0.43%)

VLAToken.transfer confirmed Block: 3 Gas used: 51109 (0.43%)

The active network is development
Deploying Mocks...
Deploying Mock Link Token...
Transaction sent: 0xae6ad8804a232ac933784eb60e38f5c8d4fec516c51acb8160f65a40bbe82315
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 3
LinkToken.constructor confirmed Block: 4 Gas used: 669136 (5.58%)
LinkToken deployed at: 0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85
```

Rysunek 29: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci lokalnej Ganache w edytorze kodu VSC 1/3

Źródło: Opracowanie własne

```

Deployed to 0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85
Deploying Mock Price Feed...
Transaction sent: 0x2e4f53da132afbe8df1be210f3fdc379697e11b23f5215230aa697772078cfe4
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 4
MockV3Aggregator.constructor confirmed Block: 5 Gas used: 430695 (3.59%)
MockV3Aggregator deployed at: 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9

Deployed to 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9
Deploying Mock DAI...
Transaction sent: 0xeac29e1d64b7f8beda3ab310cf319a08a79280f6c0536b018af657e1fd8f1d87
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 5
MockDAI.constructor confirmed Block: 6 Gas used: 585447 (4.88%)
MockDAI deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Deployed to 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3
Deploying Mock WETH
Transaction sent: 0xaa4e4157359265522a39c88f5af6e1a7fc7a8646d009da15869e6ba760523558
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 6
MockWETH.constructor confirmed Block: 7 Gas used: 585288 (4.88%)
MockWETH deployed at: 0x9E4c14403d7d9A8A782044E86a93CAE09D7B2ac9

Deployed to 0x9E4c14403d7d9A8A782044E86a93CAE09D7B2ac9
Transaction sent: 0x0843835eb6bd64b634848d316a4a74e25e20d4740a35cdffdba6464cc7d3f63
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 7
TokenYield.addAllowedTokens confirmed Block: 8 Gas used: 64325 (0.54%)

TokenYield.addAllowedTokens confirmed Block: 8 Gas used: 64325 (0.54%)

Transaction sent: 0x27e2fb73678e45092dabe7b1af36f9691402d3eeec4962e3f38bbb4337d7422
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 8
TokenYield.setPriceFeedContract confirmed Block: 9 Gas used: 44002 (0.37%)

TokenYield.setPriceFeedContract confirmed Block: 9 Gas used: 44002 (0.37%)

```

Rysunek 30: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci lokalnej Ganache w edytorze kodu VSC 2/3

Źródło: Opracowanie własne

```

Transaction sent: 0xbcb76312dfdc1ec78fa5748dd6a9999f6f512252dab956f621850da24aff76950
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 9
TokenYield.addAllowedTokens confirmed Block: 10 Gas used: 49325 (0.41%)

TokenYield.addAllowedTokens confirmed Block: 10 Gas used: 49325 (0.41%)

Transaction sent: 0xca5b6470d2f6ba52949dea5eaafa4a61056032b0f0696b7d379df1e26b1dec9c
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 10
TokenYield.setPriceFeedContract confirmed Block: 11 Gas used: 44002 (0.37%)

TokenYield.setPriceFeedContract confirmed Block: 11 Gas used: 44002 (0.37%)

Transaction sent: 0x32a17d7707bc843e306b5512faae1504963cba8ee7464e6bac25d387663deb1a
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 11
TokenYield.addAllowedTokens confirmed Block: 12 Gas used: 49325 (0.41%)

TokenYield.addAllowedTokens confirmed Block: 12 Gas used: 49325 (0.41%)

Transaction sent: 0x19fb62b09fe25555e955a43ab38b25508718f75b87a26e4925063528d729308a
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 12
TokenYield.setPriceFeedContract confirmed Block: 13 Gas used: 44002 (0.37%)

TokenYield.setPriceFeedContract confirmed Block: 13 Gas used: 44002 (0.37%)

```

Rysunek 31: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci lokalnej Ganache w edytorze kodu VSC 3/3

Źródło: Opracowanie własne

```

networks:
  development:
    verify: False
  ganache:
    verify: False
  goerli:
    verify: True
    weth_token: '0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6'
    fau_token: '0xBA62BCfcAaFc6622853cca2BE6Ac7d845BC0f2Dc'
    dai_usd_price_feed: '0x0d79df66BE487753B02D015Fb622DED7f0E9798d'
    eth_usd_price_feed: '0xD4a33860578De61DBAbDc8BfDb98FD742fA7028e'
  rinkeby:
    verify: True
    weth_token: '0xc778417E063141139Fce010982780140Aa0cD5Ab'
    fau_token: '0xFab46E002BbF0b4509813474841E0716E6730136'
    dai_usd_price_feed: '0x2bA49Aaa16E6afD2a993473cfB70Fa8559B523cF'
    eth_usd_price_feed: '0x8A753747A1Fa494EC906cE90E9f37563A8AF630e'
wallets:
  from_key: ${PRIVATE_KEY}

```

Rysunek 32: Wycinek pliku brownie-config.yaml w którym do każdej sieci testowej przypisane są inne adresy kontraktów, ponieważ są to dwa osobne byty, nie mające ze sobą połączenia. Uwzględniona jest również flaga verify, która świadczy o tym czy wykonywana jest weryfikacja kodu źródłowego. Etherscan działa tylko na sieciach testowych, dlatego jest True, w lokalnej sieci Ganache tego nie uświadczymy, ponieważ działa ona jedynie w naszym środowisku. Jest symulowaną siecią blockchain. Na dole jest odnośnik do klucza prywatnego, który znajduje się w pliku .env

Źródło: Opracowanie własne

Kolejnym już bardziej zaawansowanym krokiem jest umieszczenie kontraktów na sieci testowej. W swoim projekcie umożliwiam wybranie jednego z dwóch łańcuchów Rinkeby lub Goerli.

Każda z sieci korzysta z różnych adresów kontraktów wyroczni, która daje informacje o aktualnej cenie ETH w stosunku do dolara oraz stablecoina DAI (kryptowaluta imitująca dolara w stosunku 1:1, czyli jeden DAI to jeden dolar). Oprócz tego korzystam również z adresów kontraktów, na których znajdują się tokeny WETH oraz FAU. Korzystam z nich, ponieważ wykorzystuje Mocki sieci głównej Ethereum.

W folderze projektu znajduje się również plik o nazwie .env w którym przetrzymuje klucz prywatny do mojego konta w portfelu kryptowalutowym. Jest on konieczny, ponieważ na koncie znajduje się testowy ether, który jest potrzebny, aby móc umieścić zestaw kontraktów w sieci testowej.

```

1 export PRIVATE_KEY=0x84bb527e0f4fa387c61dc580fb6827f4623533988c7dd93a5b908287bfac007
2 export WEB3_INFURA_PROJECT_ID=7abda71ad2fa49b18ca946c72c6b558a
3 export ETHERSCAN_TOKEN=K45Z7IG144SHU135KT6UVNDT8H26YA4KS

```

Rysunek 33: Plik .env w którym przechowywane są wartości zmiennych, które nie powinny być widoczne dla zewnętrznych użytkowników. Z tego względu korzystam z pliku .gitignore, który pomija ten plik w momencie wysyłania commitów do gita. Teoretycznie w produktach, które nie są komercyjne można pominąć etap .gitignore, ale warto utrzymywać dobre praktyki.

Źródło: Opracowanie własne

Oprócz tego widnieje tam **ETHERSCAN_TOKEN**, który służy do weryfikacji kodu źródłowego kontraktów na platformie etherscan. Można się bez tego obejść, ale naprawdę warto z tego korzystać, ponieważ dzięki temu możemy wchodzić z kontraktem w interakcje z poziomem etherscana. Oprócz tego inni użytkownicy widzą z jakim kodem mają do czynienia co sprzyja większemu zaufaniu dla twórców, ponieważ w ten sposób pokazują transparentność. W skrócie nie mają nic do ukrycia. Warto utrzymywać dobre praktyki.

Contract Name: TokenYield

Compiler Version: v0.8.12+commit.f00d7308

Optimization Enabled: Yes with 200 runs

Other Settings: default evmVersion, MIT license

Contract Source Code (Solidity)

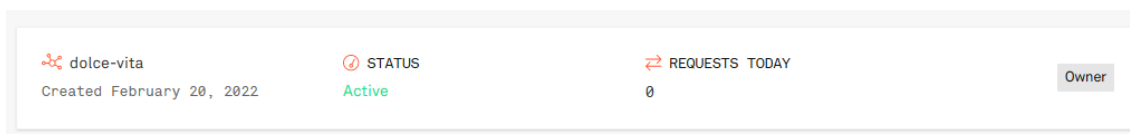
```

1 //
2 *Submitted for verification at Etherscan.io on 2022-09-09
3 */
4
5 // SPDX-License-Identifier: MIT
6
7 pragma solidity 0.8.12;
8
9
10
11 // Part: OpenZeppelin/openzeppelin-contracts@4.2.0/Context
12
13 /*
14  * @dev Provides information about the current execution context, including the
15  * sender of the transaction and its data. While these are generally available
16  * via msg.sender and msg.data, they should not be accessed in such a direct
17  * manner, since when dealing with meta-transactions the account sending and
18  * paying for execution may not be the actual sender (as far as an application
19  * is concerned).
20  *
21  * This contract is only required for intermediate, library-like contracts.
22  */
23 abstract contract Context {
24     function msgSender() internal view virtual returns (address) {
25         return msg.sender;

```

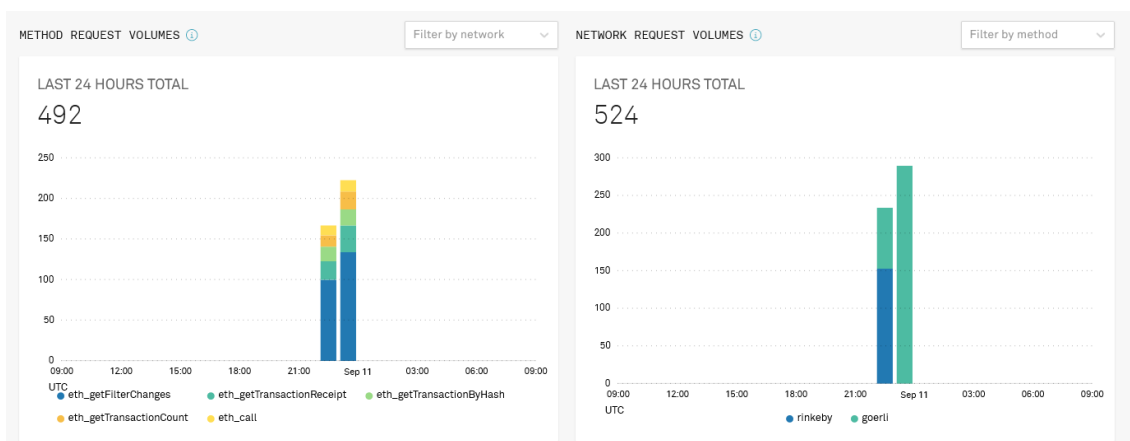
Rysunek 34: Zweryfikowany kod kontraktu TokenYield.sol umieszczony na sieci testowej Goerli

Źródło: <https://goerli.etherscan.io/address/0x38de8b1c32f12dc81334323a92cd99adc354b8e7#code>



Rysunek 35: Utworzona instancja węzła, która umożliwia umieszczenie kontraktów na sieciach testowych. Dostępna jedynie po uprzednim zalogowaniu na konto. Każdy może utworzyć swój węzeł, ale najpierw trzeba stworzyć konto.

Źródło: <https://infura.io/dashboard>



Rysunek 36: Wykresy, które pokazują liczby ostatnich wywołań instancji węzła. Za każdym razem, gdy przeprowadzam transakcje na sieciach testowych to wykres jest aktualizowany. Podgląd dostępny jest wyłącznie po uprzednim zalogowaniu się na konto.

Źródło: <https://infura.io/dashboard/stats/ethereum/24h/7abda71ad2fa49b18ca946c72c6b558a>

REQUESTS ACTIVITY				
METHOD	NETWORK	REQUESTS VOLUME	SUCCESSFUL REQUESTS (%)	FAILED REQUESTS (%)
eth_getFilterChanges	Goerli	134	100.00%	0.00%
eth_getFilterChanges	Rinkeby	52	100.00%	0.00%
eth_getFilterChanges	Goerli	48	100.00%	0.00%
eth_getTransactionReceipt	Goerli	33	100.00%	0.00%
eth_getTransactionCount	Goerli	22	100.00%	0.00%
eth_getTransactionByHash	Goerli	20	100.00%	0.00%
eth_getTransactionReceipt	Rinkeby	17	100.00%	0.00%
eth_getTransactionByHash	Rinkeby	16	100.00%	0.00%
eth_call	Goerli	14	100.00%	0.00%
eth_estimateGas	Goerli	12	100.00%	0.00%

Rysunek 37: Szczegółowa tabela mówiąca o tym z jakich sieci ostatnio korzystałem wraz z liczbą wywołań oraz procentowym zestawieniem udanych oraz nieudanych żądań

Źródło: <https://infura.io/dashboard/stats/ethereum/24h/7abda71ad2fa49b18ca946c72c6b558a>

Całość dopełnia **WEB3_INFURA_PROJECT_ID**, która jest akurat kluczowa, ponieważ platforma Infura⁹⁶ jest węzłem RPC. Dzięki niemu korzystamy z zewnętrznego węzła, który umożliwi nam przesłanie kontraktów do sieci testowej.

Powodem korzystania z możliwości uruchomienia kontraktów na dwóch różnych sieciach jest taki, że Rinkeby wkrótce przestanie być utrzymywane. Wiąże się to z tym, że Ethereum zmienia swój algorytm konsensusu i twórcy otwarcie mówią, że nie będą tej sieci utrzymywać. Ja z kolei zgromadziłem pokaźną liczbę testowych coinów, dlatego większość pracy wykonywałem z poziomu Rinkeby. Goerli natomiast jako jedna z nie-licznych sieci testowych takie wsparcie od twórców otrzymało, dlatego, żeby projekt był używalny zdecydowałem się na rozszerzenie. Jedynym minusem jest to, że coiny sieci Goerli są dosyć trudne do zdobycia, ale całe szczęście udało mi się uzyskać ich minimalną ilość do komfortowej pracy oraz testów.

```
1 TESTNET_NETWORKS = ["goerli", "rinkeby"]
2
3
4 def extractLinkToEtherscanWebsite(testnet, contractAddress):
5     if testnet in TESTNET_NETWORKS:
6         print(f"\nhttps://{testnet}.etherscan.io/address/{contractAddress}\n")
7
```

Rysunek 38: Funkcja drukująca dokładny adres kontraktu do strony internetowej etherscan'u

Źródło: Opracowanie własne

Dodatkowo utworzyłem funkcję, która zostaje aktywowana w momencie umieszczenia kontraktu na sieci testowej. Czyli w czasie egzekucji programu zostanie uruchomiona dwukrotnie. Raz dla kontraktu **VLAToken** oraz drugi dla **TokenYield**. Jej zadaniem jest wydrukowanie pełnej ścieżki do strony etherscanu na którym znajduje się konkretny kontrakt. Ma to na celu szybki dostęp do portalu, na którym możemy w bardzo przejrzysty sposób zobaczyć historie wszystkich transakcji jakie miały miejsce w powyższych kontraktach. Zamiast manualnie przepisywać adres kontraktu do etherscanu, wystarczy kliknąć w konsoli na link przy pomocy **ctrl** + lewy przycisk myszy.

Przechodząc do komend, są one następujące:

⁹⁶ What is Infura, <https://infura.io/faq/general>

npm run rinkeby

Zaznaczone fragmenty to właśnie linki do etherscanu na których możemy podejrzeć kod źródłowy kontraktu, historie transakcji oraz wykonać dodatkowe interakcje z kontraktem.

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> npm run rinkeby

> stake-yield-defi-protocol@1.0.0 rinkeby
> brownie run scripts/main.py --network rinkeby

Brownie v1.16.4 - Python development framework for Ethereum

StakeYieldDefiProtocolProject is the active project.

Running 'scripts\main.py::main'...
Transaction sent: 0x48da4e8272aea8a24982a155ce6046827b820b2a17736c4f11b7a8784f6eeb02
  Gas price: 1.500000008 gwei   Gas limit: 704322   Nonce: 158
  VLAToken.constructor confirmed   Block: 11364020   Gas used: 640293 (90.91%)
  VLAToken deployed at: 0xCB350be5192d283351Ce88c3b7512741fD8b4F0E

Waiting for https://api-rinkeby.etherscan.io/api to process contract...
Verification submitted successfully. Waiting for result...
Verification complete. Result: Already Verified

https://rinkeby.etherscan.io/address/0xCB350be5192d283351Ce88c3b7512741fD8b4F0E

Transaction sent: 0x2fbba5f97c90055aa2a6f196f7ea253b47eaeaa7bb6c1b9e12c734d829787e1f
  Gas price: 1.500000008 gwei   Gas limit: 1131629   Nonce: 159
  TokenYield.constructor confirmed   Block: 11364022   Gas used: 1028754 (90.91%)
  TokenYield deployed at: 0x5cf159ecc3740aFEcF38CC6e4eaF3332Fe0D1FB

Waiting for https://api-rinkeby.etherscan.io/api to process contract...
Verification submitted successfully. Waiting for result...
Verification complete. Result: Already Verified

https://rinkeby.etherscan.io/address/0xCB350be5192d283351Ce88c3b7512741fD8b4F0E

Transaction sent: 0xc7fc178c2a4b619444de32231bac23324be2b079de7eb9da2f1f0336e7e3f56a
  Gas price: 1.500000008 gwei   Gas limit: 56769   Nonce: 160
  VLAToken.transfer confirmed   Block: 11364024   Gas used: 51609 (90.91%)

  VLAToken.transfer confirmed   Block: 11364024   Gas used: 51609 (90.91%)
```

Rysunek 39: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci testowej Rinkeby w edytorze kodu VSC 1/2

Źródło: Opracowanie własne

```

Transaction sent: 0xcb758c67f23ba083f37ee91ae72211efa1defe2914fb66001704732d2ca4cf30
Gas price: 1.500000008 gwei Gas limit: 75047 Nonce: 161
TokenYield.addAllowedTokens confirmed Block: 11364025 Gas used: 68225 (90.91%)

TokenYield.addAllowedTokens confirmed Block: 11364025 Gas used: 68225 (90.91%)

Transaction sent: 0x9c3b8d7bbf53dee226abc68ab35df2cab9eb50bafb4f581d50b0d9efdd810290
Gas price: 1.500000008 gwei Gas limit: 51262 Nonce: 162
TokenYield.setPriceFeedContract confirmed Block: 11364026 Gas used: 46602 (90.91%)

TokenYield.setPriceFeedContract confirmed Block: 11364026 Gas used: 46602 (90.91%)

Transaction sent: 0xde79306704abbc76160d2d78d22787e86f8304ab419f00bd5454a11a3182514e
Gas price: 1.500000008 gwei Gas limit: 56224 Nonce: 163
TokenYield.addAllowedTokens confirmed Block: 11364027 Gas used: 51113 (90.91%)

TokenYield.addAllowedTokens confirmed Block: 11364027 Gas used: 51113 (90.91%)

Transaction sent: 0x02ad308b69c06c937e8d0202f0e1a37150d55b64f4a428a6de692ac2e82d0eb7
Gas price: 1.500000008 gwei Gas limit: 51249 Nonce: 164
TokenYield.setPriceFeedContract confirmed Block: 11364028 Gas used: 46590 (90.91%)

TokenYield.setPriceFeedContract confirmed Block: 11364028 Gas used: 46590 (90.91%)
Transaction sent: 0x38df43923be2fd3ca827b0a7265833ac91d2ff781f2b6aad0225cb05e9b03e90
Gas price: 1.500000008 gwei Gas limit: 56237 Nonce: 165
TokenYield.addAllowedTokens confirmed Block: 11364029 Gas used: 51125 (90.91%)

TokenYield.addAllowedTokens confirmed Block: 11364029 Gas used: 51125 (90.91%)

Transaction sent: 0x0bf64991c405f7d3b8d368e6578a31cf0b01934a8e4e0eca7482745e2e109656
Gas price: 1.500000008 gwei Gas limit: 51262 Nonce: 166
TokenYield.setPriceFeedContract confirmed Block: 11364030 Gas used: 46602 (90.91%)

TokenYield.setPriceFeedContract confirmed Block: 11364030 Gas used: 46602 (90.91%)

```

Rysunek 40: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci testowej Rinkeby w edytorze kodu VSC 2/2

Źródło: Opracowanie własne

npm run goerli

Dosłownie ta sama sytuacja co w sieci Rinkeby. Również mamy możliwość bezpośredniego wejścia na stronę etherscana i podejrzenie jego kodu źródłowego, historii transakcji czy przeprowadzenie dodatkowej interakcji z dostępnymi funkcjami.

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> npm run goerli

> stake-yield-defi-protocol@1.0.0 goerli
> brownie run scripts/main.py --network goerli

Brownie v1.16.4 - Python development framework for Ethereum

StakeYieldDefiProtocolProject is the active project.

Running 'scripts/main.py::main'...
Transaction sent: 0x11df08fef31f1c48c2906e80a7abb3e37f5e64f80f46c158c9ad38648dc7c0f1
  Gas price: 0.01271565 gwei  Gas limit: 704322  Nonce: 27
  VLAToken.constructor confirmed  Block: 7574614  Gas used: 640293 (90.91%)
  VLAToken deployed at: 0xea1606fe4d7a46c77521aCf0fb0b393BF5dbc1fF

Waiting for https://api-goerli.etherscan.io/api to process contract...
Verification submitted successfully. Waiting for result...
Verification complete. Result: Already Verified

https://goerli.etherscan.io/address/0xea1606fe4d7a46c77521aCf0fb0b393BF5dbc1fF

Transaction sent: 0x891c1848d049968d4b1c34c2adc5d670f9b03205c8f362a03ef47231287a1634
  Gas price: 0.012715651 gwei  Gas limit: 1131629  Nonce: 28
  TokenYield.constructor confirmed  Block: 7574617  Gas used: 1028754 (90.91%)
  TokenYield deployed at: 0xD9dbbF42f1df41Ddc834f003a9e6BF47DcFB8edA

Waiting for https://api-goerli.etherscan.io/api to process contract...
Verification submitted successfully. Waiting for result...
Verification complete. Result: Already Verified

https://goerli.etherscan.io/address/0xea1606fe4d7a46c77521aCf0fb0b393BF5dbc1fF

Transaction sent: 0xaf5600e996fd20facb288954d050fcc3231abb2d19b5ac391ed8107fd8877856
  Gas price: 0.012715649 gwei  Gas limit: 56769  Nonce: 29
  VLAToken.transfer confirmed  Block: 7574620  Gas used: 51609 (90.91%)
  VLAToken.transfer confirmed  Block: 7574620  Gas used: 51609 (90.91%)
```

Rysunek 41: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci testowej Goerli w edytorze kodu VSC 1/2

Źródło: Opracowanie własne

```

Transaction sent: 0x9912c8e30222add5aafdb6c1fb9dee3b49cdc47bdfacf2e41a3d270b0c96be0
Gas price: 0.09999999 gwei Gas limit: 56237 Nonce: 23
TokenYield.addAllowedTokens confirmed Block: 7568520 Gas used: 51125 (90.91%)

TokenYield.addAllowedTokens confirmed Block: 7568520 Gas used: 51125 (90.91%)

Transaction sent: 0x84bca5b45a52b854105e978e04a4c51e99942972c9f169ab39fc5a52140ae69f
Gas price: 0.099999989 gwei Gas limit: 51262 Nonce: 24
TokenYield.setPriceFeedContract confirmed Block: 7568522 Gas used: 46602 (90.91%)

TokenYield.setPriceFeedContract confirmed Block: 7568522 Gas used: 46602 (90.91%)

Transaction sent: 0xae00e791d337192adc1bed6b53b0a37c21dc61e5e380a0079779196759d2437e
Gas price: 0.099999989 gwei Gas limit: 56237 Nonce: 25
TokenYield.addAllowedTokens confirmed Block: 7568524 Gas used: 51125 (90.91%)

TokenYield.addAllowedTokens confirmed Block: 7568524 Gas used: 51125 (90.91%)

Transaction sent: 0x2b4eae2513b512a827c6c25424785ab9e3f8bbf727588090d1c621f22bd25113
Gas price: 0.099999989 gwei Gas limit: 51262 Nonce: 26
TokenYield.setPriceFeedContract confirmed Block: 7568526 Gas used: 46602 (90.91%)

TokenYield.setPriceFeedContract confirmed Block: 7568526 Gas used: 46602 (90.91%)

```

Rysunek 42: Prawidłowy wynik przeprowadzenia wszystkich transakcji w łańcuchu sieci testowej Goerli w edytorze kodu VSC 2/2

Źródło: Opracowanie własne

Na koniec pozostaje kwestia uruchomienia testów jednostkowych oraz integracyjnych. Do wyboru mamy kilka możliwości. Możemy uruchomić wszystkie testy po kolei przy użyciu jednej komendy

npm run test

```

PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> npm run test

> stake-yield-defi-protocol@1.0.0 test
> brownie test

Brownie v1.16.4 - Python development framework for Ethereum

===== test session starts =====
platform win32 -- Python 3.10.2, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol
plugins: eth-brownie-1.16.4, hypothesis-6.21.6, cov-2.12.1, forked-1.3.0, xdist-1.34.0, web3-5.23.1
collected 9 items

Launching 'ganache-cli.cmd --accounts 10 --hardfork istanbul --gaslimit 12000000 --mnemonic brownie --port 8545'...

tests\integration_testing\test_token_yield_integration.py .
tests\unit_testing\test_allowed_tokens.py ..
tests\unit_testing\test_staked_amount.py ....
tests\unit_testing\test_token_value.py ..

===== 9 passed in 48.83s =====
Terminating local RPC client...

```

Rysunek 43: Wynik prawidłowego wykonania wszystkich testów w kon soli w edytorze VSC

Źródło: Opracowanie własne

Możemy również uruchomić tylko jeden plik skryptowy.

brownie test -k test_staked_amount

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> brownie test -k test_staked_amount
Brownie v1.16.4 - Python development framework for Ethereum

===== test session starts =====
platform win32 -- Python 3.10.2, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol
plugins: eth-brownie-1.16.4, hypothesis-6.21.6, cov-2.12.1, forked-1.3.0, xdist-1.34.0, web3-5.23.1
collected 9 items / 5 deselected / 4 selected

Launching 'ganache-cli.cmd --accounts 10 --hardfork istanbul --gasLimit 12000000 --mnemonic brownie --port 8545'...

tests\unit_testing\test_staked_amount.py ....

===== 4 passed, 5 deselected in 30.33s =====
Terminating local RPC client...
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol>
```

Rysunek 44: Wynik prawidłowego wykonania testów z jednego pliku skryptowego w konsoli w edytorze VSC

Źródło: Opracowanie własne

Lub tylko jedną funkcję testującą.

brownie test -k test_set_price_feed_contract

```
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol> brownie test -k test_set_price_feed_contract
Brownie v1.16.4 - Python development framework for Ethereum

===== test session starts =====
platform win32 -- Python 3.10.2, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol
plugins: eth-brownie-1.16.4, hypothesis-6.21.6, cov-2.12.1, forked-1.3.0, xdist-1.34.0, web3-5.23.1
collected 9 items / 8 deselected / 1 selected

Launching 'ganache-cli.cmd --accounts 10 --hardfork istanbul --gasLimit 12000000 --mnemonic brownie --port 8545'...

tests\unit_testing\test_token_value.py .

===== 1 passed, 8 deselected in 13.29s =====
Terminating local RPC client...
PS C:\dev-programming\Praca Dyplomowa - Jakub Pawłowski\stake-yield-defi-protocol>
```

Rysunek 45: Wynik prawidłowego wykonania pojedynczego testu w konsoli w edytorze VSC

Źródło: Opracowanie własne

PODSUMOWANIE

Cała branża blockchain jest dla mnie niesamowicie interesująca. Na ten moment programowanie w ekosystemie sieci Ethereum jest raczej niszowym zajęciem. Mimo to widzę w tym wielką przyszłość. Cieszę się, że mogę połączyć dwie pasje w postaci programowania oraz finansów.

Cała wizja zdecentralizowanych systemów bez odgórnego administratora jest magnetyzująca. Rozwiązuje wiele problemów oraz wprowadza prawdziwą demokrację. Każdy z użytkowników jest sobie równy, nie ma podziałów na kasty. Wbudowana transparentność jest kolejną bardzo pozytywną cechą. Sprzyja to błyskawicznemu postępowi technologii oraz jeszcze większej propagacji kultu Open Source na świecie.

Wszystko jeszcze jest na początkowym etapie, dlatego jeszcze nie uświadczylimy globalnej adopcji. Wejście w ten świat ma stosunkowo wysoki próg wejścia, ponieważ od samego początku nowi użytkownicy są zawałeni ilością pojęć. Nie ma tutaj miejsca na błędy, ponieważ w razie wpadki nie udamy się do biura obsługi klienta w celu odzyskania środków. Jesteśmy tak naprawdę swoim własnym bankiem, to bardzo duża odpowiedzialność. UX również jest jeszcze daleki od doskonałości, co czyni go mało intuicyjnym.

Całe szczęście firmy na całym świecie firmy na całym świecie pracują na tym, żeby to zmienić na bardziej przystępne dla standardowego użytkownika. Wiele standardowych firm widzi w tym potencjał i tworzą swoje własne produkty. W większości mam tutaj na myśli kolekcje NFT, które w 2021 roku zyskały rozgłos na całym świecie. W ramach ciekawostki Polska przoduje w niechlubnym rankingu⁹⁷ nienawiści w stosunku do tokenów NFT. Zostało to zbadane podczas analizy wpisów Polaków w mediach społecznościowych. Nie przeszkadza to jednak polskim przedsiębiorcom w tworzeniu swoich kolekcji, które służą do gromadzenia społeczności wokół twórców.

Z całą pewnością potrzebujemy w branży więcej tego typu zapalników, które wzniosą technologię blockchain na wyżyny. Jeszcze wiele do odkrycia, a powszechna transparentność na pewno ułatwi to zadanie. Coraz częściej pojawiają się pomysły na nowe standardy tokenów.

⁹⁷ Krystian Ławniczak, Polska nienawidzi NFT, https://ithardware.pl/aktualnosci/polska_nienawidzi_nft_tak_donosza_najnowsze_badania-22089.html

Język Solidity również prężnie się rozwija. W środowisku, w którym kod stanowi prawo, możliwości są praktycznie nieograniczone. Dzięki technologii kontraktów oraz mechanizmów wyroczni możemy działać w sposób niewymagający zaufania trzeciej strony. Wyobraźmy sobie życie bez pośredników nieruchomości, notariuszy. Technologia umożliwia nam chociażby reprezentowanie dowolnej nieruchomości w postaci tokenu NFT, którą następnie możemy wymieniać na dedykowanych aplikacjach webowych zrobionych w standardzie Web 3.0. To nie jest science fiction, tylko rzeczywistość. Ostatnimi czasy wielokrotnie byliśmy świadkami takich sytuacji. Występuje nawet zjawisko tokenizowania nieruchomości w standardzie ERC-20. Dzielimy nieruchomość na konkretną liczbę takich tokenów, przykładowo 100. I każdy z nich odpowiada jednemu procentowi własności. Następnie twórcy rozdysponowują miesięczne zyski z chociażby wynajmu, wszystkim tym, którzy posiadają udział w postaci tokena danej nieruchomości.

W dzisiejszych czasach można dosłownie wszystko stokenizować. Jedynym limitem jest prawo, które nie nadąża za technologią, ale branża blockchain zrzesza również ogromne grono prawników, którzy specjalizują się w tym temacie.

Przed wykonaniem aplikacji zawartej w pracy dyplomowej, udało mi się stworzyć kilka mniejszych projektów. Dużo eksperymentowałem z samymi kontraktami. Jednakże napisanie protokołu wynagradzającego za umieszczenie na nim tokenów było najbardziej kompletną aplikacją z ekosystemu Ethereum jaką wykonałem.

Łączy w sobie wiele rzeczy. Wykorzystanie standardu tokena ERC-20 z biblioteki OpenZeppelin. Główny kontrakt **TokenYield** jest połączony z **VLA Token**. Do tego wykorzystanie wyroczni od ChainLink, która łączy system blockchain ze światem zewnętrznym. W celu zdobycia informacji o aktualnej cenie Etheru. Praca z dwiema sieciami testowymi, co wiąże się z tym, że musiałem uzyskać testowe tokeny z dostępnych kraników (faucet'y). Zrozumienie działania oraz biegle posługiwanie się narzędziem eksploratora bloków, czyli etherscan. Napisanie testów jednostkowych oraz integracyjnych przy użyciu metody AAA⁹⁸. W celu sprawdzenia czy wszystko działa tak jak powinno. Podzielenie wszystkiego na dedykowane moduły, które skupiają się na konkretnej funkcjonalności. Praca z gitem⁹⁹, czyli systemem kontroli wersji oraz oczywiście GitHubem¹⁰⁰. Jest to

⁹⁸ Paulo Gomes, Unit Testing and the Arrange, Act and assert (AAA) Pattern, <https://medium.com/@pjbgr/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80>

⁹⁹ Git, <https://git-scm.com/>

¹⁰⁰ What is GitHub, <https://kinsta.com/knowledgebase/what-is-github/>

podstawowe narzędzie w dosłownie każdej ścieżce programisty jaka istnieje. Odpowiednie pisanie commitów¹⁰¹, opisujących czego dokładnie dotyczy wprowadzana zmiana. Oswojenie się z językiem angielskim, poprzez biegłe czytanie dokumentacji technicznych.

¹⁰¹ Git commit, <https://www.git-tower.com/learn/git/commands/git-commit>

BIBLIOGRAFIA

Artykuły oraz inne źródła internetowe

1. Adam Hayes, Blockchain Facts: What is it, How it works, and How it can be used, <https://www.investopedia.com/terms/b/blockchain.asp>
2. Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, <https://bitcoin.org/en/bitcoin-paper>
3. Hossa/Bessa długotrwały trend wzrostowy/spadkowy, <https://szybkagotowka.pl/Wiki/hossa-bessa>
4. Kate Brush, FOMO (fear of missing out), <https://www.techtarget.com/whatis/definition/FOMO-fear-of-missing-out>
5. Vitalik Buterin, Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform, <https://ethereum.org/en/whitepaper/>
6. Jakub Pawłowski, 1.3 Stosowana kryptografia, Ethereum jako platforma dla aplikacji zdecentralizowanych
7. Nick Lioudis, The Collapse of Lehman Brothers: A Case Study, <https://www.investopedia.com/articles/economics/09/lehman-brothers-collapse.asp>
8. 5 Features That Make a Bitcoin A Unique Asset Class, <https://argoblockchain.com/articles/5-features-that-make-bitcoin-a-unique-asset-class>
9. Matt Whittaker, What is Bitcoin Halving, <https://www.forbes.com/advisor/investing/cryptocurrency/bitcoin-halving/>
10. Bitcoin Inflation, <https://charts.woobull.com/bitcoin-inflation/>
11. Weronika Grzywna, Czym jest pieniądz fiducjarny i jaką ma wartość, <https://www.money.pl/pieniadze/czym-jest-pieniadz-fiducjarny-i-jaka-ma-wartosc-6748241007573824a.html>
12. Krzysztof Kolany, 45 lat temu Nixon zamknął „złote okno”, <https://www.bankier.pl/wiadomosc/45-lat-temu-Nixon-zamknal-zlote-okno-7473819.html>
13. Daniel Levi, 80% of all US dollars in existence were printed in the last 22 months, <https://techstartups.com/2021/12/18/80-us-dollars-existence-printed-january-2020-october-2021/>
14. Peter Chawaga, How Bitcoin fueled the freedom convoy and defied government crackdown, <https://bitcoinmagazine.com/culture/how-bitcoin-fueled-canada-trucker-convoy>
15. Jakub Pawłowski, 2.5 Węzły oraz klienci, Ethereum jako platforma dla aplikacji zdecentralizowanych
16. Prabath Siriwardena, The Mystery Behind Block Time, <https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a?gi=16c0522a3050>
17. Blocks, <https://ethereum.org/en/developers/docs/blocks/>
18. What Makes a Blockchain Secure, <https://academy.binance.com/en/articles/what-makes-a-blockchain-secure>
19. What is hashing, <https://academy.binance.com/en/articles/what-is-hashing>
20. What is epoch time, <https://www.epochconverter.com/>
21. History of Cryptography, <https://academy.binance.com/en/articles/history-of-cryptography>
22. What is open source, <https://opensource.com/resources/what-open-source>

23. Vitalik Buterin, Thoughts about the longerterm future of the Ethereum protocol, <https://www.youtube.com/watch?v=kGjFTzRTH3Q&t=1891s>
24. Keccak256 hash function in Solidity, <https://cryptomarketpool.com/keccak256/>
25. What is Public Key Cryptography, <https://academy.binance.com/en/articles/what-is-public-key-cryptography>
26. Jakub Pawłowski, 1.5 Portfel Kryptowalutowy, Ethereum jako platforma dla aplikacji zdecentralizowa-nych
27. Jakub Pawłowski, 2.4 EVM, Ethereum jako platforma dla aplikacji zdecentralizowanych
28. Matt Hussey, What is MetaMask, <https://decrypt.co/resources/metamask>
29. Seed Phrase, <https://academy.binance.com/en/glossary/seed-phrase>
30. What is a digital signature, <https://academy.binance.com/en/articles/what-is-a-digital-signature>
31. What is Used Experience (UX) Design, <https://www.interaction-design.org/literature/topics/ux-design>
32. Introduction, <https://web3py.readthedocs.io/en/latest/>
33. Jakub Pawłowski, 3. Inteligentne Kontrakty, Ethereum jako platforma dla aplikacji zdecentralizowa-nych
34. Modular Programming, <https://www.techopedia.com/definition/25972/modular-programming>
35. UTXO vs Account Model, <https://www.horizen.io/blockchain-academy/technology/expert/utxo-vs-account-model/>
36. Jake Frankenfield, Ether (ETH), <https://www.investopedia.com/terms/e/ether-cryptocurrency.asp>
37. Redundancja, <https://mfiles.pl/pl/index.php/Redundancja>
38. Kirsty Moreland, What is a Crypto Wallet, <https://www.ledger.com/academy/what-is-a-crypto-wallet>
39. Benedict George, What is a CEX, <https://www.coindesk.com/learn/what-is-a-cex-centralized-exchanges-explained/>
40. Jake Frankenfield, Hot Wallet, <https://www.investopedia.com/terms/h/hot-wallet.asp>
41. What are Cold Wallets, <https://academy.bit2me.com/en/what-are-cold-wallets/>
42. Rakesh Sharma, Ledger Wallet, <https://www.investopedia.com/terms/l/ledger-wallet.asp>
43. Trezor Wallet, <https://www.bitdeal.net/trezor-wallet>
44. Kristy Moreland, Enable Blind Signing: Why, When and How to Stay Safe, <https://www.ledger.com/academy/enable-blind-signing-why-when-and-how-to-stay-safe>
45. Nathan Reiff, What is Avalanche (AVAX), <https://www.investopedia.com/avalanche-avax-definition-5217374>
46. Networks, <https://ethereum.org/en/developers/docs/networks/>
47. Jakub Pawłowski, 1.6. Coin vs Token, Ethereum jako platforma dla aplikacji zdecentralizowanych
48. Crypto tokens vs coins – What’s the Difference, <https://crypto.com/university/crypto-tokens-vs-coins-difference>
49. How Google Play works, <https://play.google.com/about/howplayworks/>
50. App store, <https://www.computerhope.com/jargon/a/app-store.htm>
51. What is a Decentralized Exchange (DEX), <https://www.gemini.com/cryptopedia/decentralized-exchange-crypto-dex>

52. The difference between an EOA and a contract, <https://www.oreilly.com/library/view/mastering-blockchain-programming/9781839218262/00403615-d0ec-4c80-94a0-dc1d24fd3246.xhtml>
53. What is the difference between a transaction and a call, <https://ethereum.stackexchange.com/questions/765/what-is-the-difference-between-a-transaction-and-a-call>
54. What is Ethereum Gas, <https://www.sofi.com/learn/content/what-is-ethereum-gas/>
55. Jakub Pawłowski, 2.6. Algorytm konsensusu, Ethereum jako platforma dla aplikacji zdecentralizowanych
56. How Accurate and Reliable is the Block Timestamp in Ethereum, <https://originstamp.com/blog/how-accurate-and-reliable-is-the-block-timestamp-in-ethereum/>
57. Valerio Puggioni, What is Etherscan, and how does it work, <https://cointelegraph.com/news/what-is-etherscan-and-how-does-it-work>
58. EVM Explained – What is Ethereum Virtual Machine, <https://moralis.io/evm-explained-what-is-ethereum-virtual-machine/>
59. Difference between Byte Code and Machine Code, <https://www.geeksforgeeks.org/difference-between-byte-code-and-machine-code/>
60. Nodes and clients, <https://ethereum.org/en/developers/docs/nodes-and-clients/>
61. Sofia Brooke, China Makes Cryptocurrency Transactions Illegal: An Explainer, <https://www.china-briefing.com/news/china-makes-cryptocurrency-transactions-illegal-an-explainer/>
62. Zhaoyin Feng, Why China's bitcoin miners are moving to Texas, <https://www.bbc.com/news/world-us-canada-58414555>
63. Fares Alkudmani, What are the benefits of being an Ethereum Full Node, <https://www.quora.com/What-are-the-benefits-of-being-an-Ethereum-full-node>
64. Consensus mechanisms, <https://ethereum.org/en/developers/docs/consensus-mechanisms/>
65. Jake Frankenfield, Proof of Work (PoW), <https://www.investopedia.com/terms/p/proof-work.asp>
66. Euny Hong, How Does Bitcoin Mining Work, <https://www.investopedia.com/tech/how-does-bitcoin-mining-work/>
67. Proof-of-Stake (PoS), <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>
68. Jay Kurahashi-Sofue, What is a blockchain validator, <https://support.avax.network/en/articles/4064704-what-is-a-blockchain-validator>
69. What is Ethereum difficulty, <https://2miners.com/eth-network-difficulty>
70. Andrea Knezovic, The Difference Between Fungible and Non-Fungible Tokens, <https://medium.com/udonis/the-difference-between-fungible-and-non-fungible-tokens-nfts-123df237b892>
71. Standards, <https://docs.openzeppelin.com/contracts/4.x/tokens>
72. ERC20, <https://docs.openzeppelin.com/contracts/4.x/erc20>
73. OpenZeppelin, <https://moonbeam.network/community/projects/openzeppelin/>
74. ERC721, <https://docs.openzeppelin.com/contracts/4.x/erc721>
75. Eric James Beyer, Dive into three of the top NFT marketplaces, <https://nftnow.com/guides/dive-into-three-of-the-top-nft-marketplaces/>
76. Jake Frankenfield, What Are Smart Contracts on the Blockchain and How They Work, <https://www.investopedia.com/terms/s/smart-contracts.asp>

77. Smart Contracts, <https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>
78. What is minting an NFT, <https://www.nfi.edu/what-is-minting-an-nft/>
79. What is an NFT Whitelist and How do you get whitelisted, <https://cyberscrilla.com/what-is-an-nft-whitelist-and-how-do-you-get-whitelisted/>
80. What is Uniswap, <https://www.coinbase.com/pl/learn/crypto-basics/what-is-uniswap>
81. Andrej Rakic, Upgradeable Smart Contracts – Proxy Pattern, <https://mvpworks-hop.co/blog/upgradeable-smart-contracts-proxy-pattern/>
82. A Brief History of the Internet, https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml
83. Comparison between Web 1.0, Web 2.0 and Web 3.0, <https://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference/>
84. Introduction to Web3, <https://ethereum.org/en/web3/>
85. Jake Frankenfield, Decentralized Applications (dApps), <https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp>
86. About Python, <https://www.python.org/about/>
87. Solidity, <https://docs.soliditylang.org/en/v0.8.15/>
88. Brownie, <https://eth-brownie.readthedocs.io/en/stable/>
89. What is Ganache Blockchain, <https://101blockchains.com/ganache-blockchain/>
90. What is a Blockchain Oracle, <https://chain.link/education/blockchain-oracles>
91. Chainlink, <https://www.investopedia.com/chainlink-link-definition-5217559>
92. How to mock Solidity smart contracts for testing, <https://ethereum.org/en/developers/tutorials/how-to-mock-solidity-contracts-for-testing/>
93. Testing smart contracts, <https://ethereum.org/en/developers/docs/smart-contracts/testing/>
94. What is Visual Studio Code, <https://www.educba.com/what-is-visual-studio-code/>
95. User interface (UI), <https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI>
96. What is Infura, <https://infura.io/faq/general>
97. Krystian Ławniczak, Polska nienawidzi NFT, <https://ithardware.pl/aktualnosci/polska-nienawidzi-nft-tak-donosza-najnowsze-badania-22089.html>
98. Paulo Gomes, Unit Testing and the Arrange, Act and assert (AAA) Pattern, <https://medium.com/@pjbfgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80>
99. Git, <https://git-scm.com/>
100. What is GitHub, <https://kinsta.com/knowledgebase/what-is-github/>
101. Git commit, <https://www.git-tower.com/learn/git/commands/git-commit>