

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI



PRACA DYPLOMOWA

STACJONARNE STUDIA I^o

Temat pracy: **PROJEKT I IMPLEMENTACJA SYSTEMU DO OBIEGU I DYSTRYBUCJI BONÓW TOWAROWYCH Z UŻYCIEM TECHNOLOGII BLOCKCHAIN**

INFORMATYKA

(kierunek studiów)

INŻYNIERIA SYSTEMÓW

(specjalność)

Diplomant:

Michał KONOPKA

Promotor pracy:

dr inż. Krzysztof KANCIAK

Oświadczenie

"Wyrażam zgodę na udostępnianie mojej pracy przez Archiwum WAT"

Data 19.01.2022

Michał Konopka

(podpis)

**WOJSKOWA AKADEMIA TECHNICZNA
WYDZIAŁ CYBERNETYKI
INSTYTUT SYSTEMÓW INFORMATYCZNYCH**

"AKCEPTUJĘ"

DZIEKAN WYDZIAŁU CYBERNETYKI

dr hab. inż. Zbigniew TARAPATA

30 CZE. 2021

Warszawa, dnia r.

**ZADANIE
do pracy dyplomowej**

Wydane studentowi: **MICHAŁ KONOPKA**
(stacjonarne studia pierwszego stopnia)

I. Temat pracy:

**PROJEKT I IMPLEMENTACJA SYSTEMU DO OBIEGU I DYSTRYBUCJI BONÓW
TOWAROWYCH Z UŻYCIEM TECHNOLOGII BLOCKCHAIN**

II. Treść zadania:

1. Koncepcja i architektura protokołu obiegu tokenów reprezentujących wartości materialne
2. Przegląd i analiza istniejących publicznych łańcuchów bloków pod kątem zidentyfikowanych wymagań
3. Implementacja systemu obiegu tokenów
4. Analiza wydajności, kosztów działania systemu oraz omówienie wykorzystanych metod anonimizacji użytkowników systemu

III. W rezultacie wykonania pracy należy dodatkowo przedstawić:

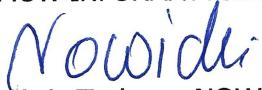
.....
.....

IV. Termin zdania przez studenta ukończonej pracy: **25.01.2022 r.**

V. Data wydania zadania: **30.06.2021 r.**

PROMOTOR
PRACY DYPLOMOWEJ

dr inż. Krzysztof KANCIAK
(tytuł naukowy, imię i nazwisko)

DYREKTOR INSTYTUTU
SYSTEMÓW INFORMATYCZNYCH

dr hab. inż. Tadeusz NOWICKI
(tytuł naukowy, imię i nazwisko)

Zadanie otrzymałem dnia **30.06.2021**

Michał KONOPKA
(podpis studenta)


SPIS TREŚCI

Wstęp	9
1 Wstęp teoretyczny do technologii łańcucha bloków	11
1.1 Architektura scentralizowana a zdecentralizowana	11
1.2 Sieć peer-to-peer	12
1.3 Łańcuch bloków	13
1.4 Rejestr transakcji	14
1.4.1 Funkcja skrótu	14
1.4.2 Łańcuch	15
1.4.3 Binarne drzewo skrótów	16
1.4.4 Kryptografia klucza publicznego	17
1.4.5 Transakcja	18
1.4.6 Blok	19
1.4.7 Niezmienność rejestru transakcji	19
1.5 Konsensus	20
1.5.1 Problem bizantyjskich generałów	21
1.5.2 Algorytm konsensusu w łańcuchach bloków	21
1.5.3 Konsensus dowodu pracy (ang. POW - proof of work) . . .	22
1.5.4 Konsensus dowodu stawki (ang. POS - proof of stake) . . .	23
1.6 Smart kontrakt	23
1.7 Kryptowaluta	24
2 Analiza i założenia systemu obiegu bonów towarowych	25
2.1 Cel systemu	25
2.2 Studium przypadku — bon turystyczny	25
2.2.1 Rejestracja realizatora bonu	26
2.2.2 Dystrybucja bonu do użytkownika	26
2.2.3 Konsumpcja bonu	27
2.2.4 Spieniężenie bonu	29

2.2.5	Aktorzy	29
2.2.6	Cykl życia bonu	31
2.3	Definicja wymagań	31
2.3.1	Funkcjonalność	32
2.3.2	Użyteczność	35
2.3.3	Niezawodność	36
2.3.4	Wydajność	36
2.3.5	Elastyczność	37
3	Przegląd i analiza istniejących publicznych łańcuchów bloków pod kątem zidentyfikowanych wymagań	38
3.1	Analiza wymagań pod kątem łańcucha bloków	38
3.2	Ethereum	39
3.2.1	Konsensus	39
3.2.2	Smart kontrakt	40
3.2.3	Dokumentacja, środowisko deweloperskie	42
3.2.4	Efektywność sieci	42
3.2.5	Ocena	43
3.3	Binance Smart Chain	43
3.3.1	Konsensus	43
3.3.2	Smart kontrakt	45
3.3.3	Dokumentacja, środowisko deweloperskie	45
3.3.4	Efektywność sieci	45
3.3.5	Ocena	46
3.4	Algorand	46
3.4.1	Konsensus	46
3.4.2	Smart kontrakt	48
3.4.3	Dokumentacja, środowisko deweloperskie	51
3.4.4	Efektywność sieci	52
3.4.5	Ocena	53

3.5 Wnioski	53
4 Architektura oprogramowania systemu obiegu tokenów	55
4.1 Styl architektury	55
4.1.1 Architektura mikro usługi	56
4.1.2 Architektura n-warstw	57
4.2 Komponenty i ich interfejsy	59
4.3 Kooperacja komponentów	62
4.3.1 Emisja	62
4.3.2 Nadanie roli realizatora	64
4.3.3 Udział w dystrybucji bonu	65
4.3.4 Spieniężenie bonu	66
5 Implementacja systemu obiegu tokenów	67
5.1 Standardy i interfejsy komunikacji	67
5.1.1 REST	67
5.1.2 Serializacja danych	69
5.2 Warstwa logiki biznesowej	69
5.2.1 Wymagania względem języka programowania	69
5.2.2 Wybrany język programowania	69
5.2.3 Szkielet do budowy aplikacji	71
5.3 Warstwa danych	74
5.3.1 Wybór bazy danych	74
5.3.2 Interakcja z bazą danych	75
5.3.3 Schemat bazy danych	77
5.4 Smart kontrakty	79
5.4.1 Logika smart kontraktu	80
5.5 Pobieranie atrybutów	83
5.5.1 Wybrany dostawca tożsamości	84
5.6 Komunikacja z siecią Algorand	85
5.7 Bramka płatności	85

5.8	Interfejs użytkownika	85
5.8.1	Aplikacja jednostronnicowa	86
5.8.2	Wybrane technologie	86
5.8.3	Portfel	87
6	Wnioski z realizacji pracy dyplomowej	89
6.1	Omówienie wykorzystanych metod anonimizacji użytkowników . . .	89
6.2	Analiza wydajności systemu	90
6.2.1	Środowisko testowe	90
6.2.2	Utworzenie bonu	91
6.2.3	Dystrybucja bonu	91
6.2.4	Transfer bonu	91
6.2.5	Podsumowanie	91
6.3	Analiza kosztów działania systemu	92
Zakończenie		94
Bibliografia		96

WSTĘP

Wiele współczesnych branż odnosi korzyści ze wdrażania nowoczesnych systemów opartych o rozproszoną architekturę. Jedną z branży pozostającej w tyle za autonomicznymi samochodami i inteligentnymi lodówkami, zdaje się być branża finansów. System finansowy zbudowany jest z wielu podsystemów (pojedynczych banków), które są następnie synchronizowane przez nad systemy. Alternatywnym podejściem do hierarchicznej i centralizowanej struktury systemów bankowych jest technologia łańcucha bloków, która oferuje zdecentralizowaną architekturę oraz mechanizm wspólnego konsensusu.

Celem niniejszej pracy dyplomowej jest stworzenie zamkniętego obiegu pieniądza elektronicznego, z programowalną logiką transferu, z wykorzystaniem publicznego łańcucha bloków. Rozwiązanie ma przedstawić alternatywne implementacje, do centralnego systemu bonu turystycznego.

W rozdziale pierwszym zawarto wprowadzenie teoretyczne do technologii łańcucha bloków. Zostały omówione fundamentalne rozwiązania pozwalające tworzyć rozproszone rejesty.

Rozdział drugi poświęcono na analizę istniejącego rozwiązania bonów turystycznych. Zostali w nim zdefiniowani aktorzy występujący w systemie, cykl życia bonu oraz wymagania względem solucji.

Analizę wybranych istniejących publicznych łańcuchów oraz dobranie najlepiej dopasowanego do zdefiniowanych wymagań zawarto w rozdziale trzecim.

W rozdziale czwartym skupiono się na architekturze systemu. Podzielono system na komponenty oraz nakreślono ich kooperacje.

Rozdział piąty dotyczy implementacji systemu. Zostały w nim wybrane odpowiednie technologie, wzorce projektowe, struktury danych, algorytmy oraz standardy. Znalazły się tam również wycinki kodu reprezentujące kluczowe zagadnienia, dotyczące implementacji.

W ostatnim rozdziale zweryfikowano kluczowe aspekty systemu. Przeprowadzono analizę anonimowości użytkownika. Poddano system testom wydajnościowym

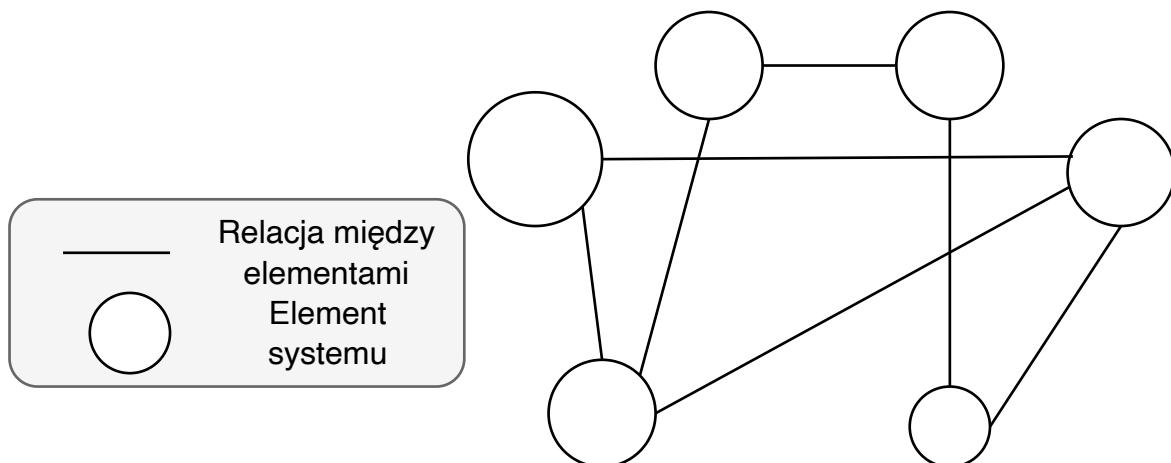
w celu sprawdzenia zgodności z wymaganiami. Przeprowadzono również analizę kosztów związaną z wykorzystaniem publicznego łańcucha bloków.

1 WSTĘP TEORETYCZNY DO TECHNOLOGII ŁAŃCUCHA BLOKÓW

1.1 ARCHITEKTURA SCENTRALIZOWANA A ZDECENTRALIZOWANA

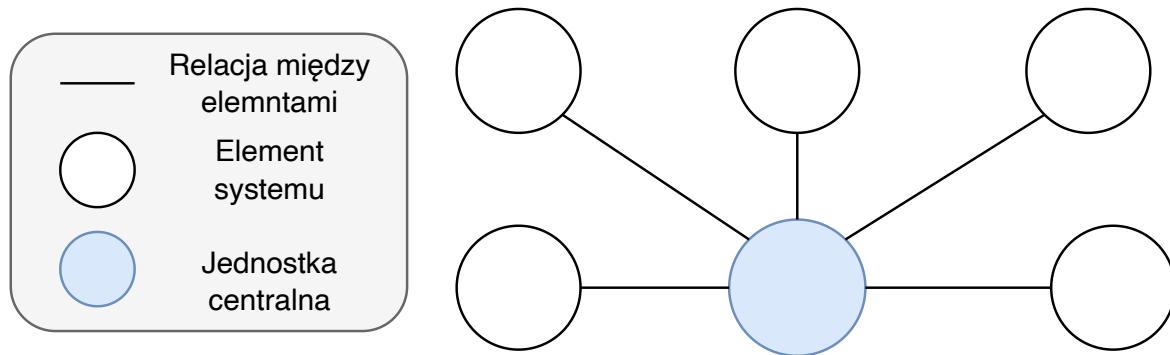
W ogólnym znaczeniu architektura systemu mówi o tym, w jaki sposób zostały zorganizowane jego elementy. To znaczy, jakie są określone relacje pomiędzy tymi elementami oraz jakie cechy posiadają. Jednym z kluczowych kryteriów podziału architektury danego systemu jest fakt, czy dana architektura jest scentralizowana, czy też zdecentralizowana.[4, s. 21-25]

Systemy oparte o architekturę zdecentralizowaną nazywane są systemami rozproszonymi (zdecentralizowanymi). System rozproszony składa się z wielu jednostek sprzętowych połączonych medium transmisyjnym, które dążą do osiągnięcia wspólnego celu. Popularne architektury systemów rozproszonych to między innymi: klient-serwer, klient-multi-serwer oraz peer-to-peer. Ostatnia z nich jest kluczowa w kontekście technologii łańcucha bloków.[9, s. 6]



Rysunek 1: Koncepcja systemu rozprozonego. Źródło: Źródło własne

System scentralizowany jest przeciwnieństwem systemu zdecentralizowanego. Jest on oparty o jednostkę centralną, która stanowi pojedynczy punkt sterowania oraz przechowywania danych.



Rysunek 2: Koncepcja systemu scentralizowanego. Źródło: Źródło własne

Z definicji systemu scentralizowanego i rozproszonego możemy wywnioskować tezę, iż system nie może zostać uznany za rozproszony, dopóki w systemie istnieje chociaż jeden element, którego wyłączenie spowoduje brak możliwości wykonywania zdefiniowanego celu.

1.2 SIEĆ PEER-TO-PEER

Sieć peer-to-peer to jeden z wariantów architektury systemów rozproszonych. Taką organizację systemu wyróżnia równa odpowiedzialność oraz funkcjonalność wszystkich węzłów systemu (przez co węzły pełnią role zarówno producentów jak i konsumentów). Węzły udostępniają pomiędzy sobą moc obliczeniową oraz/lub miejsce na przechowywanie danych. Pomimo pełnienia tej samej funkcji, nie muszą one przechowywać tych samych danych. Odpowiednią metaforą jest architektura rozprozonego systemu plików peer-to-peer, gdzie każdy z uczestników udostępnia pliki do pobrania, równocześnie pobiera pliki od innego węzła, gdzie kryterium wyboru węzła jest dostępność pliku[4, s. 30].

Warto zauważyć, że w praktycznych sieciach peer-to-peer, często porzuca się pełne rozproszenie (równość w sensie odpowiedzialności i funkcji wszystkich węzłów), na rzecz korzyści wydajnościowych wynikających z częściowego scentrali-

zowania. Kontynuując metaforę rozproszonego systemu plików, można sobie wyobrazić istnienie węzłów katalogujących, które posiadają rejestr plików przetrzymywanych przez poszczególne węzły, dzięki czemu węzeł wymagający dostępu do dodanego pliku, może zwrócić się najpierw do węzła katalogującego, który wskaże mu węzły posiadające dany plik. W przypadku pełnego rozproszenia należałoby, odpytać wszystkie węzły.

Taka architektura bezpośredniego kontaktu uczestników systemu pozwala na eliminację pośredników, co niesie za sobą korzyści dla konsumentów i odbiorców. Jednocześnie zagraża biznesowi wielu firm opierających się na pośrednictwie. Zauważył to Satoshi Nakamoto w 2009 roku tworząc “Elektroniczą Gotówkę Peer-to-Peer” zwaną Bitcoin[10], potencjalnie wykluczając tym samym jednego z największych pośredników, czyli banki[13].

1.3 ŁAŃCUCH BLOKÓW

Łańcuch bloków może być definiowany jako pakiet technologii służący do zapewnienia własności integralności systemu rozproszonego opartego o architekturę peer-to-peer, w którym poziom uczciwości poszczególnych węzłów oraz ich ilość jest nieznana. Poprzez własność integralności systemu rozumie się nienaruszalność rejestrów wprowadzanych zmian oraz brak możliwość wprowadzania zmian bez poprawnej autoryzacji.

Na potrzeby tego dokumentu zostanie przyjęta bardziej uniwersalna i holistyczna definicja, określając łańcuch bloków jako system rozproszonych węzłów o architekturze peer-to-peer, które działają według określonego algorytmu. Ponieważ algorytmy, struktury danych mogą znacznie różnić się w zależności od implementacji, przyjęty zostanie ogólny model, który nie jest zoptymalizowany pod konkretny zbiór przypadków użycia.

1.4 REJESTR TRANSAKCJI

Rejestr transakcji to nieedytowalna struktura danych, która pozwala jedynie na dopisywanie następnych transakcji bez możliwości edycji historii transakcji. Jak wskazuje nazwa zawiera on dane transakcyjne, które w przeciwieństwie do danych inwentaryzacyjnych oprócz informacji o aktualnym stanie, uzasadniają pochodzenie i zasadność stanu. Aby taki dowód integralności transakcji był możliwy w niezaufanej rozproszonej sieci, należy to zapewnić na poziomie struktury danych.

1.4.1 FUNKCJA SKRÓTU

Niech dana będzie funkcją skrótu, która odwzorowuje wiadomość o strukturze binarnej m o dowolnej skończonej długości w ciąg bitów q (skrót) o określonej stałej długości l . Opisana jest za pomocą pary probabilistycznych wielomianowych algorytmów (Gen, H) , definiowanych następująco:

1. Gen - jest probabilistycznym algorytmem, który przyjmuje parametr bezpieczeństwa 1^n i produkuje klucz s .
2. H - przyjmuje na wejściu klucz s oraz ciąg $m \in \{0, 1\}^*$ oraz zwraca ciąg $H(s, m) \in \{0, 1\}^{l(n)}$, gdzie n jest wartością parametru bezpieczeństwa zawartego w s [1, s. 154]

Wymaganiem wobec “dobrej” funkcji skrótu jest trudność znalezienia kolizji. Kolizja definiowana jest jako para różnych ciągów m oraz m' dla których $H(m, s) = H(m', s)$. Znalezienie kolizji możemy zdefiniować jako funkcje $Col((Gen, H), n)$, która zwraca wartość 1, tylko gdy zajdzie kolizja.

Należy podkreślić różnice między kluczami symetrycznymi a kluczem s . Przede wszystkim nie wszystkie ciągi są poprawnymi kluczami, dlatego są generowane za pomocą funkcji Gen , a nie losowane. Po drugie, klucz ten nie jest niejawny i funkcja powinna pozostawać odporna na kolizje nawet jeśli jest znany atakującemu.

Funkcja $T = (Gen, H)$ jest uważana za odporną na kolizję jeśli dla każdego ataku o złożoności wielomianowej istnieje funkcja zaniedbywalna [18] $negl$, taka, że:

$$\Pr[Col(T, n) = 1] \leq negl(n)$$

W dalszej części pracy będzie używana uproszczona definicja funkcji skrótu bez klucza $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$. Definicja ta jest problematyczna z teoretycznego punktu (ponieważ w h “zaszyty” jest klucz s), lecz uproszcza dalsze rozważania[1, s. 155].

Funkcje skrótu stanowią nieodłączny element zaplecza technologicznego łańcucha bloków. Metaforą funkcji skrótu jest odcisk palca, tak jak każdy człowiek ma własny odcisk palca tak każdy zestaw danych w wyniku funkcji skrótu otrzymuje swój własny cyfrowy odcisk palca — skrót.

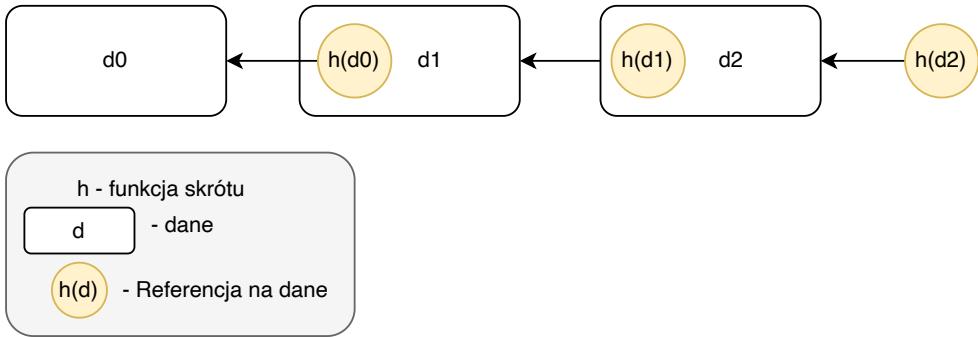
Dzięki funkcji skrótu zyskujemy wzrost efektywności porównywania danych. Struktury danych nie muszą być porównywane atrybut po atrybutie. Wystarczy, że zostaną porównane ich skróty. Niech dane będą struktury danych d_1 oraz d_2 i funkcja skrótu h , wtedy: $d_1 \equiv d_2 \implies h(d_1) \equiv h(d_2)$

Drugą z cech funkcji skrótów jest tworzenie referencji $r = h(d)$ związanych z konkretnymi danymi, które w momencie zmiany stają się niepoprawne, ponieważ wraz z aktualizacją danych, wcześniej utworzona referencja staje się nieaktualna. Zależność ta pozwala organizować dane w struktury wrażliwe na edycję danych. Takie struktury wykorzystane w łańcuchu bloków to:

1. Łańcuch
2. Binarne drzewo skrótów

1.4.2 ŁAŃCUCH

Łańcuch to liniowa struktura danych przypominająca listę jednokierunkową (patrz Rysunek 3.), w której referencją na poprzedni element jest skrót tego elementu. Znajomość wartości pierwszej referencji umożliwia uzyskanie dostępu do wszystkich danych zapisanych na danym łańcuchu. Zmiana zawartości, któregoś elementu byłaby równoważna z przerwaniem łańcucha, a więc i utratą spójności danych. Innymi słowy, zaobserwowanie nieprawidłowej referencji w łańcuchu jest dowodem na modyfikacje danych. Jednocześnie architektura łańcucha pozwala tylko i wyłącznie na dodawanie danych na początek łańcucha. A więc założenia te spełniają wymagania rejestru transakcji wobec struktury danych.



Rysunek 3: Struktura danych łańcuch. Źródło: Źródło własne

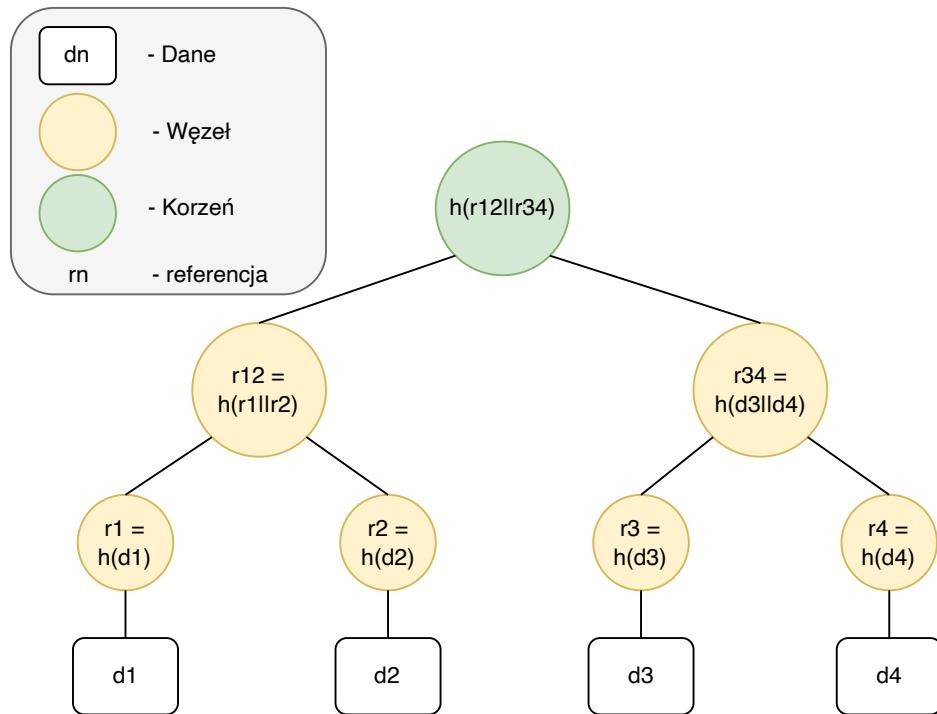
1.4.3 BINARNE DRZEWO SKRÓTÓW

Drzewo skrótów (ang. merkle tree) jest strukturą danych wykorzystywaną w nagłówkach bloków, z których składa się łańcuch bloków.

Niech \parallel oznacza konkatenację ciągu bitów. Co definiuje się następująco dla $m = < m_1, \dots, m_{n-1} >$ oraz $m' = < m'_1, \dots, m'_{w-1} >$, gdzie n i w są odpowiednio długościami danych ciągów, ich konkatenacja jest ciągiem o długości $n + w$ danym przez

$$m \parallel m' = < m_0, \dots, m_{n-1}, m'_0, \dots, m'_{w-1} >$$

W drzewie skrótów liśćmi są skróty danych, zaś każdy wewnętrzny węzeł stanowi skrót konkatenacji dwóch poprzednich (patrz Rysunek 4.). Zatem korzeń drzewa (ang. Merkle root) stanowi skrót wszystkich danych zawartych w drzewie. Dzięki takię konstrukcję, tak jak w przypadku struktury łańcucha struktura staje się wrażliwa na modyfikacje danych. Umożliwia również efektywniejsze sprawdzanie, czy wskażane dane zawarte są w korzeniu drzewa. Na przykładzie rysunku 4, aby sprawdzić, czy d_1 znajduje się w korzeniu drzewa, należałoby obliczyć wyłącznie referencje r_1 , oraz r_{12} oraz r_{34} , aby otrzymać oczekiwany korzeń drzewa (zawierający d_1). Dla drzewa skrótów składających się z n skrótów, najgorsza złożoność obliczeniowa dla sprawdzenia, czy występuje dany element wynosi $2 \log_2(n)$ [14, s. 212].



Rysunek 4: Struktura danych drzewo skrótów. Źródło: Źródło własne

1.4.4 KRYPTOGRAFIA KLUCZA PUBLICZNEGO

Kryptografia asymetryczna w odróżnieniu od kryptografii symetrycznej odróżnia się tym, że zamiast jednego klucza istnieje para kluczy, prywatny oraz publiczny. Klucz prywatny jak wskazuje nazwa powinien być utajniony, zaś klucz publiczny ogólnodostępny. Istnieją dwa sposoby wykorzystania tych kluczy:

- publiczno-prywatny, w którym klucz publiczny służy do szyfrowania a klucz prywatny do odszyfrowywania,
- prywatno-publiczny, w którym klucz prywatny służy do złożenia dowodu faktu jego posiadana (bez ujawniania go) a klucz publiczny do jego weryfikacji, w ten sposób jest wykorzystywany w kontekście łańcucha bloków — jako podpis kryptograficzny.

Kryptografia klucza publicznego zwana również asymetryczną w technologii łańcuchów bloków wykorzystywana jest w dwóch aspektach:

1. Identyfikacji autora transakcji — klucz publiczny służy jako jednoznaczny identyfikator, ponieważ jest unikalny
2. Autoryzacji transakcji — transakcje autoryzowane są za pomocą podpisu kryp-

tograficznego

Działanie podpisu kryptograficznego opiera się o dwa algorytmy, podpisywania danych oraz weryfikowania podpisu. Niech dana będzie funkcja h oraz funkcja szyfrowania $E(m, sk)$ i deszyfrowania $D(m, pk)$, gdzie pk to klucz publiczny a sk to klucz prywatny tej samej pary. Wtedy operację generowania podpisu $p(m)$ można zdefiniować jako:

$$p(m) = E(h(m), sk)$$

a operacje weryfikowania podpisu jako:

$$h(m) \equiv D(p(m), pk)$$

Dzięki tej właściwości podpis cyfrowy może służyć jako dowód, że właściciel klucza prywatnego, który został użyty do stworzenia tego podpisu cyfrowego, rzeczywiście zgadza się na określoną transakcję[4, s. 101] oraz dzięki właściwości funkcji skrótu, że dane transakcji nie zostały naruszone. Ze względu na publiczny charakter klucza publicznego każdy uczestnik sieci może to zweryfikować.

1.4.5 TRANSAKCJA

Transakcja to atom, z którego składa się rejestr transakcji. Jedynym sposobem na zmianę stanu reprezentowanego przez rejestr transakcji, jest dodanie nowej transakcji. Struktura transakcji zależna jest od danej implementacji. Uniwersalna postać transakcji została przedstawiona w tabeli 1.

Tabela 1: Ogólna struktura transakcji. Źródło: Źródło własne

Lp.	Nazwa pola	Opis pola
1	Opłata transakcyjna	Opłata jaką twórcą transakcji chce uiścić na cześć utrzymania systemu peer-to-peer
2	Odbiorca	Odbiorca transakcji, w szczególnym przypadku może być to adres smart kontraktu (patrz rozdział 1.6)
3	Wartość	Wartość przekazywanych środków

4	Dane	To pole przeznaczone jest na potrzeby wywoływanego smart kontraktu, (mogą znajdować się tu parametry wywoływanej metody kontraktu lub kod smart kontraktu w przypadku tworzenia)
5	Identyfikator nadawcy	Klucz publiczny nadawcy
6	Podpis	Podpis kryptograficzny całej transakcji

1.4.6 BLOK

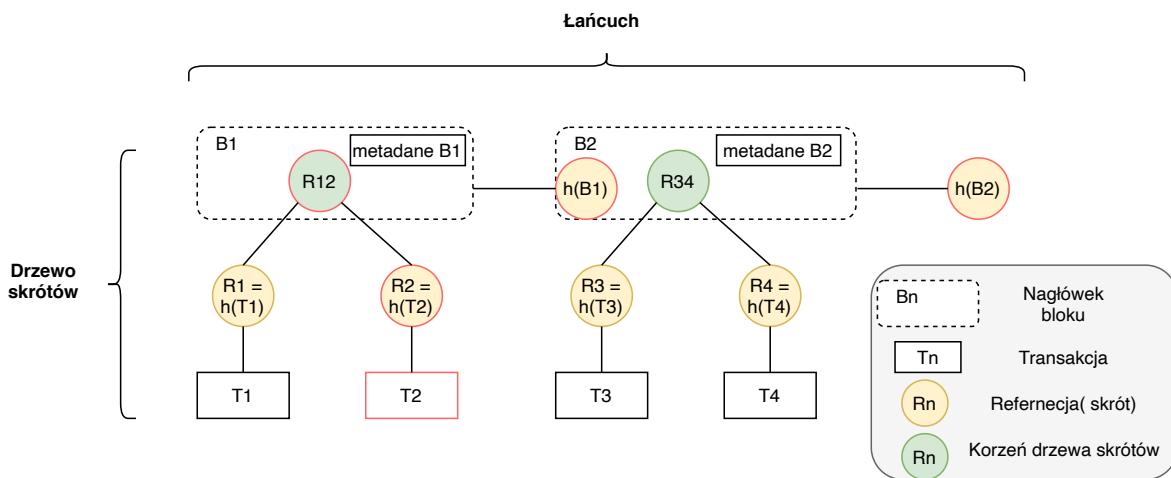
Blok to struktura danych agregująca transakcje. Składa się on z nagłówka bloku oraz transakcji zawartych w danym bloku. Nagłówek bloku składa się z referencji (skrótu nagłówka poprzedniego bloku) wskazujące na poprzedni blok, korzenia drzewa skrótów wszystkich transakcji znajdujących się w bloku oraz znacznika czasowego. W zależności od implementacji konsensusu mogą pojawiać się dodatkowe pola takie jak poziom trudności, z jakim został wydobyty blok czy wartość nonce (patrz rozdział 1.5.1).

1.4.7 NIEZMIENNOŚĆ REJESTRU TRANSAKCJI

Rejestr transakcji stanowi rdzeń architektury łańcucha bloków, ponieważ reprezentuje on stan całego systemu oraz jest jedynym źródłem prawdy o systemie. Jego zasada działania opiera się na dwóch prostych założeniach. Jego zawartość jest nieedytowalna, a dokładniej jej edycja jest łatwo wykrywalna poprzez zidentyfikowanie zerwanej referencji. Założenia drugiego mówiącego, że do rejestru transakcji dodawane są jedynie poprawnie zweryfikowane transakcje. Pierwsze założenie zostało zrealizowane poprzez zastosowanie specjalnych struktur danych, w głównej mierze opartych o własności funkcji skrótu. Drugie założenie opiera się o algorytm weryfikacji transakcji oraz sposobu ich dołączania, zwany konsensusem.

Wykażmy wrażliwości rejestru transakcji na zmianę danych, niech dana będzie następująca struktura rejestru transakcji opisana na rysunku 5 (czerwoną ramką zo-

stały oznaczone wyróżnione elementy). Założymy, że zostałyby zmienione dane transakcji T_2 wtedy referencja $R_2 = h(T_2)$ przestanie być aktualna, a więc zmienimy również referencje R_2 wtedy korzeń drzewa $R_{12} = h(h(T_1) \parallel h(T_2))$ również zostanie zdezaktualizowany. Podążając tą drogą należałoby zmienić drzewo skrótów R_{12} co z kolei wpłynęłoby na unieważnienie referencji $h(B_1)$, z kolei zmiana referencje $h(B_1)$ wpłynęłaby na unieważnienie referencji $h(B_2)$, co dowodzi faktu, aby zmienić dane elementy transakcji T_2 należałoby, by stworzyć nowy rejestr transakcji[4, s. 118].



Rysunek 5: Rejestr transakcji. Źródło: Źródło własne

1.5 KONSENSUS

Pojęcie konsensus jest znanym pojęciem w dziedzinie informatyki, dotyczy ono problemu ustalenia wspólnego stanu dla wszystkich uczestników zdecentralizowanego systemu. Problem ten jest szczególnie trudny do osiągnięcia w systemie pozabawionym jednostek centralnych stanowiących źródło prawdy, czyli w rozważanej sieci peer-to-peer. Aby osiągnąć konsensus, wszystkie jednostki systemu muszą stosować się do tych samych reguł. Aby niezależnie od siebie podejmować tożsame decyzje na temat akceptacji nowych transakcji. Rdzeniem osiągnięcia konsensu jest oddanie na pewien moment w jakiś sposób władzy grupie uczestników sieci i pozwolenie im na podjęcie decyzji, po czym rozproszenie tej decyzji po wszystkich uczestnikach sieci[5, s. 325].

1.5.1 PROBLEM BIZANTYJSKICH GENERAŁÓW

Powstaje problem jak rozproszona sieć węzłów może podejmować wspólne decyzji, przy odgórny założeniu, że węzły mogą ulec awarii lub intencjonalne negatywnie wpływać na działanie sieci. Szczególny przypadek tego problem został zdefiniowany w 1982 roku w formie logicznego dylematu, nazwanego Problemem Bizantyjskich Generalów. Dylemat ten opiera się na założeniu, iż istnieje trzech generałów, każdy z nich koordynuje podległą mu armią. Muszą podjąć jednolitą decyzję w sprawie ataku na miasto, innymi słowy osiągnąć konsensus. W przypadku braku koordynacji poniosą porażkę. Problem objawia się w niepewnej komunikacji, pomiędzy generałami, która przekazywana jest za pośrednictwem gońców. Goniec może zgubić wiadomość, opóźnić jej dostarczenie w czasie lub uszkodzić jej treść. Nawet jeśli wiadomość zostanie dostarczona poprawnie, to generał może skłamać i wysłać fałszywą wiadomość o swoich zamiarach, celowo prowadząc do porażki. Aby osiągnąć konsensus generałowie muszą spełnić następujące warunki:

- każdy z generałów musi podjąć decyzje,
- decyzja nie może ulec zmianie,
- wszyscy generałowie muszą podjąć wspólną decyzję, a następnie wykonać ją w zsynchronizowany sposób.

W łańcuchach bloków analogią generała jest węzeł sieci. A konsensusem jest podjęcie przez większość sieci tej samej decyzji na temat stanu systemu[8].

1.5.2 ALGORYTM KONSENSUSU W ŁAŃCUCHACH BLOKÓW

Algorytm konsensus to mechanizm, który ma zadanie rozwiązać problem bizantyjskich generałów w sieciach peer-to-peer.

Typ algorytmu konsensusu jest tym, co w głównej mierze odróżnia od siebie istniejące publiczne łańcuchy bloków. Każdy z nich posiada swoje wady i zalety, co skutkuje brakiem optymalnego rozwiązania i wyłonienia najlepszego publicznego łańcucha bloków. Dlatego należy dobierać implementacje łańcucha bloków odpowiednio do zdefiniowanych wymagań. Najczęściej spotykanyimi implementacjami są

algorytmy dowodu pracy oraz stawki.

1.5.3 KONSENSUS DOWODU PRACY (ANG. POW - PROOF OF WORK)

Konsensus dowodu pracy opiera się na zagadce kryptograficznej oraz na teorii gier. W zależności od implementacji tego konsensusu typy zagadek mogą się różnić. Dla lepszego zobrazowania dalsze rozważania będą oparte na typie zagadki wykorzystanej w sieci Bitcoin. Zagadka ta polega na znalezieniu takiego skrótu nowego nagłówka bloku, że będzie się on zaczynał od x zer. Ilość zer x nazywana jest również poziomem trudności wydobycia bloku, ponieważ im więcej zer tym mniejsze prawdopodobieństwo na odnalezienie klucza. Aby możliwe było generowanie różnych skrótów dla tego samego bloku, dodaje się do nagłówka bloku tak zwany *nonce*, który jest wartością służącą jedynie do manipulacji wartości skrótu. Jedynym sposobem na znalezienie skrótu o odpowiedniej ilości zer jest atak siłowy, czyli sprawdzenie każdego *nonce* po kolei, jest to zapewnione przez słabą bezkolizyjność funkcji skrótu. Taki atak wymaga pracy, stąd nazwa dowód pracy. Znalezienie takiego skrótu, nazywane jest wykopaniem bloku, i wiąże się z nagrodą dla węzła systemu, któremu się to uda. Węzeł, któremu udało się wykopać blok, rozpowszechnia go do innych węzłów, które obliczając skrót nagłówka, są w stanie efektywnie zweryfikować jego poprawność oraz zaimportować, czyli dopisać do swojego łańcucha bloków. Oprócz sprawdzenia poprawności wartości *nonce*, sprawdzana jest również poprawność wszystkich transakcji, np. czy dane środki nie zostały wydane podwójnie, czy wszystkie transakcje posiadają poprawny podpis.

Element teorii gier polega na tym, że uczestnikowi rozproszonego rejestru, nie opłaca się oszukiwać, ponieważ oszustwo może zostać w łatwy sposób wykryte. Wystarczy, że znajdą się uczciwi uczestnicy, którzy efektywnie zweryfikują błąd, a cała praca poświęcona na wykopanie bloku zostanie stracona. Oznacza to, że szkodliwy uczestnik utraci energię i nie uzyska nic w zamian. Z drugiej strony gdyby tę samą moc poświecił na poprawne wykopanie bloku, wiążałoby się to dla niego z nagrodą za wydobycie poprawnego bloku.

1.5.4 KONSENSUS DOWODU STAWKI (ANG. POS - PROOF OF STAKE)

Jednym z problemów dowodu pracy jest jego energochłonność, która bierze się z rozwiązywania zagadki matematycznej. Algorytm dowodu stawki jest jedną za najpopularniejszych alternatyw, rozwiązującą ten problem. Polega on na pseudolosowym wyborze węzła, który stanie się validatorem następnego bloku. Na wybór węzła w zależności od implementacji mogą wpływać różne czynniki. Możemy do nich zaliczyć czystą losowość lub/i ilość stawki zastawionej przez węzeł. Stawka może być reprezentowana przez kryptowalute lub inne aktywo wymienialne w ramach danego łańcucha bloków. Gdy węzeł zostanie wybrany do wydobycia bloku, sprawdza on poprawność transakcji zawartych w bloku, podpisuje go, a następnie rozsyła do pozostałych uczestników. Za wykonaną pracę otrzymuję nagrodę, pobieraną z opłat transakcyjnych.

Aby węzeł mógł stać się validatorem, musi zablokować pewną wartość stawki. Zablokowana stawka jest pod kontrolą algorytmu konsensusu. Węzeł może zaprzestać być validatorem, lecz jego stawka oraz uzyskane nagrody zostaną zwolnione dopiero po określonym okresie. W tym czasie inne węzły mogą zweryfikować poprawność pracy validatorsa.

Bezpieczeństwo modelu stawki opiera się na modelu finansowym. Przyjęte zostało założenie, że węzłom bardziej opłaca się być uczciwym i produkować poprawne bloki, następnie dostawać za nie nagrodę. Aniżeli dopuszczać niepoprawne transakcje i ryzykować wykrycie przez sieć, w czego konsekwencji validator zostanie pozbaowany części zastawionej stawki oraz straci możliwość bycia validatorem. Podsumowując, tak dług jak węzłom opłaca się być uczciwym, sieć pozostaje bezpieczna[43].

1.6 SMART KONTRAKT

Smart kontrakty to zdecentralizowane programy uruchamiane na wszystkich węzłach w sieci. Dzięki temu, że wykonywane są na wszystkich węzłach, można ufać wykonywanemu programowi tak długo, jak długo ufa się danej sieci. Nie trzeba zdawać się na pojedyncze środowisko obliczeniowe lub mediatora.

W zależności od implementacji programy te mogą być kompletne lub niekompletne w sensie Turinga[15]. Wykonywanie takich programów obarczone jest opłatą tak samo jak egzekwowanie standardowych transakcji.

Do wywoływania i tworzenia smart kontraktów wykorzystywane jest dodatkowe pole w transakcji (patrz tabela 1.) *dane*, pole to może zawierać w zależności od implementacji kod lub skompilowany bajt kod[12] w przypadku tworzenia smart kontraktu. W przypadku wywołania kontraktu zawiera ono zakodowaną sygnaturę metody oraz jej parametry. Smart kontrakt może aktualizować stan rozproszonego rejestru na tej samej zasadzie co w przypadku zwykłych transakcji. Innymi słowy, wywołanie smart kontraktu jest zwykłą transakcją, jedynie logika sprawdzenia transakcji ulega zmianie, ponieważ wymaga wywołania określonego programu. Smart kontrakt podczas wykonywania ma dostęp do stanu reprezentowanego przez rejestr transakcji, a także metadanych aktualnego bloku oraz zawartości transakcji, która go wywołała.

1.7 KRYPTOWALUTA

Większość publiczny łańcuchów bloków posiada natywną walutę, która odgrywa krytyczną rolę w motywowaniu prawidłowego zachowania w sieci. Walidatorzy bloków są nią nagradzani za swoją pracę. A użytkownicy sieci płacą za jej pomoc koszty związane z włączeniem transakcji do bloku. Przesyłanie kryptowalut to najczęściej kojarzony przypadek użycia z technologią łańcucha bloków, lecz stanowi on jedynie wierzchołek góry lodowej.

2 ANALIZA I ZAŁOŻENIA SYSTEMU OBIEGU BONÓW TOWAROWYCH

2.1 CEL SYSTEMU

Celem systemu jest obsługa dystrybucji oraz wymiany bonu cyfrowego. Bon jest środkiem płatniczym tak jak na przykład karta płatnicza, z tą różnicą, że możliwość jego przekazania jest ograniczona przez odgórnie określone warunki. Takimi warunkami może być np. data ważności lub kontrolowana grupa odbiorców. W postaci fizycznej bony często występują w postaci plastikowych kart z unikalnym kodem, reprezentując pewną wartość PLN, którą możemy wydać tylko we wskazanym sklepie.

Celem tego systemu jest przeniesienie bonów do postaci cyfrowej zapisywanej na publicznym łańcuchu bloków z dodatkową funkcją pozwalającą na automatyczną dystrybucję bonów do użytkowników spełniających określone wymagania.

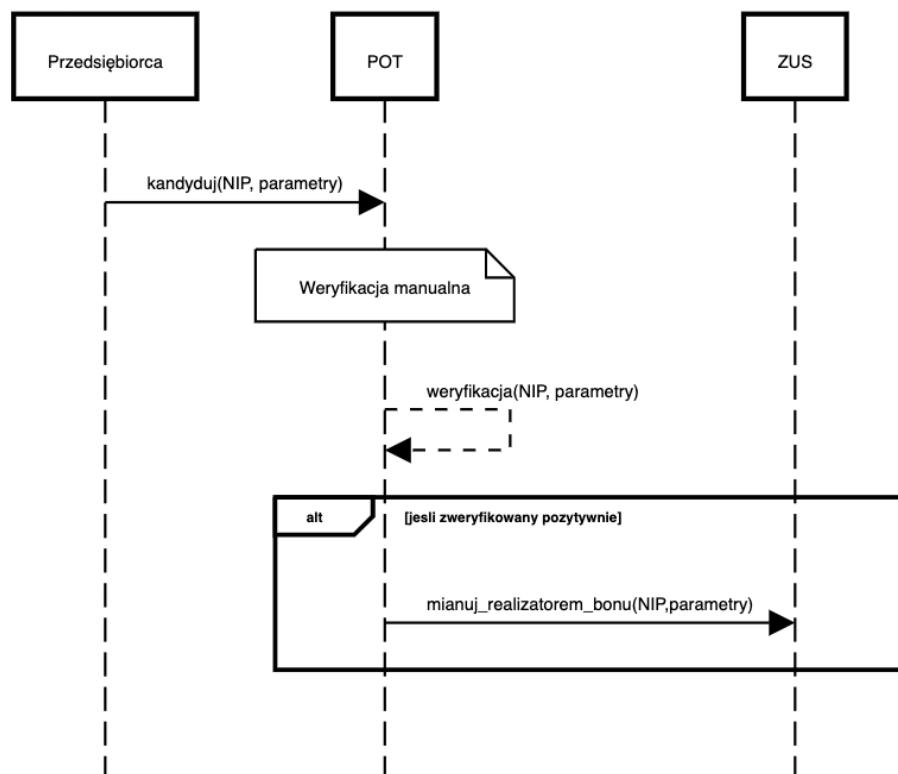
2.2 STUDIUM PRZYPADKU — BON TURYSTYCZNY

Aby lepiej zrozumieć procesy bonu oraz zdefiniować wymagania przeanalizowany został przypadek obecnie istniejącego systemu dystrybucji bonu. Analiza została oparta na danych publicznych znajdujących się w internecie, z tego też powodu jest ograniczona w szczegółach. Skupia się ona na koncepcji realizacji tak zwanej pozytywnej ścieżki. Omawiany przypadek to bon turystyczny, który funkcjonuje w polskim systemie prawnym za sprawą ustawy z dnia 15 lipca 2020 r. Bon ma na celu zaadresować problemy ekonomiczne przedsiębiorstw z branży turystycznej, poprzez dystrybucję do określonej grupy obywateli przez państwo kuponu z datą ważności realizowanego tylko u wybranych przedsiębiorstw.

Na podstawie analizy zidentyfikowano 4 kluczowe procesy omówione w kolejnych sekcjach.

2.2.1 REJESTRACJA REALIZATORA BONU

W procesie występuje trzech aktorów: ZUS jako centralny system obsługi bonu, Polska Organizacja Turystyczna jako instytucja dopuszczająca nowych realizatorów bonu oraz kandydat na realizatora bonu (przedsiębiorca). Proces polega na zweryfikowaniu kandydata przez Polską Organizację Turystyczną - w przypadku pozytywnej weryfikacji kandydat zostaje dodany w systemie ZUS jako realizator bonu. Proces został przedstawiony na rysunku 6.

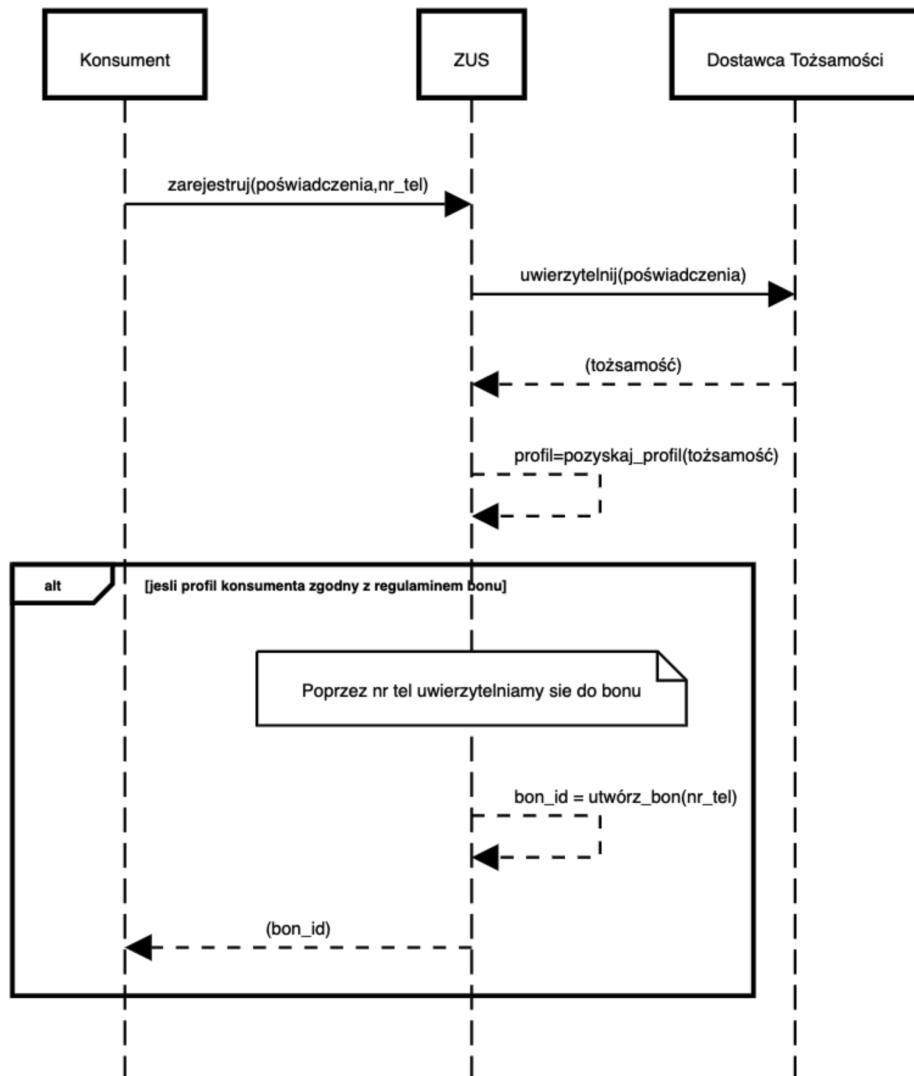


Rysunek 6: Rejestracja realizatora bonu. Źródło: Źródło własne

2.2.2 DYSTRYBUCJA BONU DO UŻYTKOWNIKA

Proces jest realizowany między trzema podmiotami: konsument bonu, ZUS, dostawca tożsamości. Efektem jest uzyskanie przez konsumenta unikalnego identyfikatora jego bonu. W procesie kluczowy jest dostawca tożsamości, który jest w stanie zidentyfikować i uwierzytelnić konsumenta do atrybutu, ze względu na który przysługuje mu bon (w tym wypadku jest to fakt posiadania dzieci). Następnie ZUS

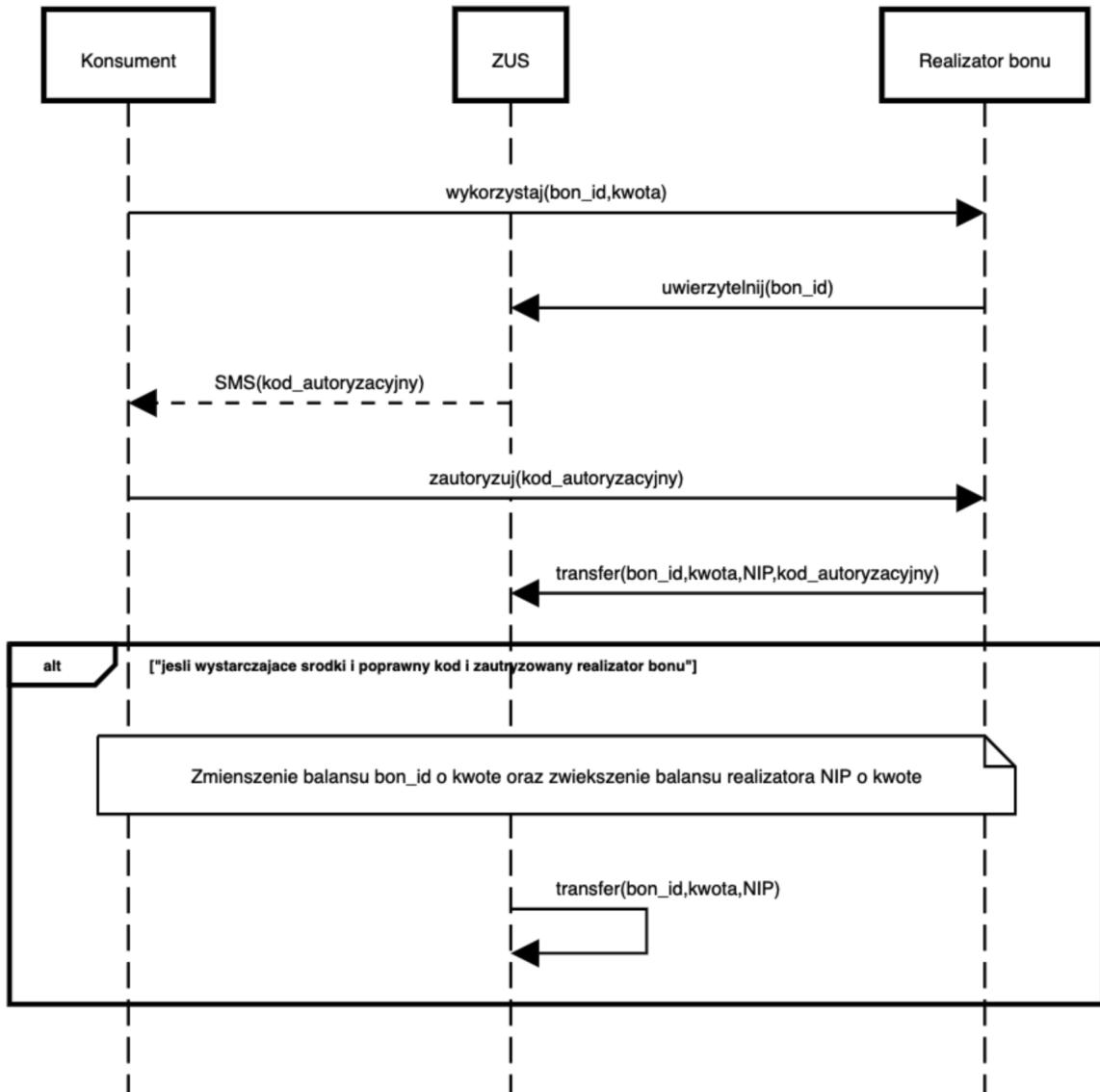
weryfikuje, że profil konsumenta spełnia założenia dystrybucji bonu. Rolę dostawcy tożsamości może pełnić np. system ePUAP [53] lub bank. Proces został przedstawiony na rysunku 7.



Rysunek 7: Dystrybucja bonu użytkownika. Źródło: Źródło własne

2.2.3 KONSUMPCJA BONU

Proces konsumpcji polega na wykorzystaniu wcześniej uzyskanego identyfikatora bonu przez konsumenta u realizatora bonu uwierzytelniając się do niego poprzez fakt posiadania numeru telefonu. Cały proces jest koordynowany przez ZUS, który przetrzymuje stan dotyczący bonów. Proces został przedstawiony na rysunku 8.



Rysunek 8: Konsumpcja bonu. Źródło: Źródło własne

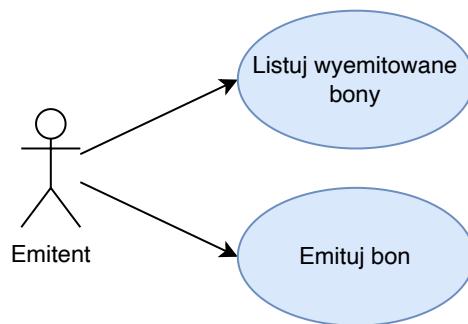
2.2.4 SPIENIĘŻENIE BONU

Jest to najprostszy proces z punktu widzenia funkcjonalnego, w którym Realizator bonu ubiega się o przelew bankowy środków uzyskanych w ramach realizowania bonów, przelew jest realizowany przez ZUS.

2.2.5 AKTORZY

Przed definicją wymagań systemu zostaną zdefiniowani występujący w nim aktorzy oraz ich przypadki użycia. Został również nakreślony cykl życia bonu, który w dalszej analizie będzie zdekomponowany do poziomu procesów.

Emitent Aktor odpowiedzialny za generowanie bonu ze wskazanymi warunkami wykorzystania. Przypadki użycia zostały przedstawione na rysunku 9.

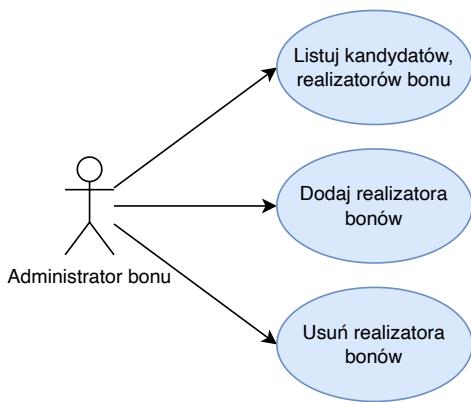


Rysunek 9: Diagram przypadków użycia Emitenta bonu. Źródło: Źródło własne

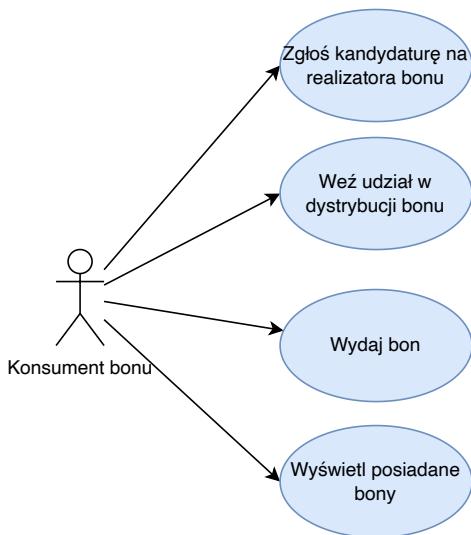
Administrator bonu Aktor odpowiedzialny za zarządzanie bonem w czasie trwania cyklu jego życia. W szczególności zarządzania listą podmiotów upoważnionych do realizacji bonu. Przypadki użycia zostały przedstawione na rysunku 10.

Konsument bonu Aktor, który może ubiegać się o otrzymanie wyemitowanego bonu oraz następnie go wykorzystać u upoważnionych podmiotów. Jest to główny beneficjent systemu generujący obrót bonami. Przypadki użycia zostały przedstawione na rysunku 11.

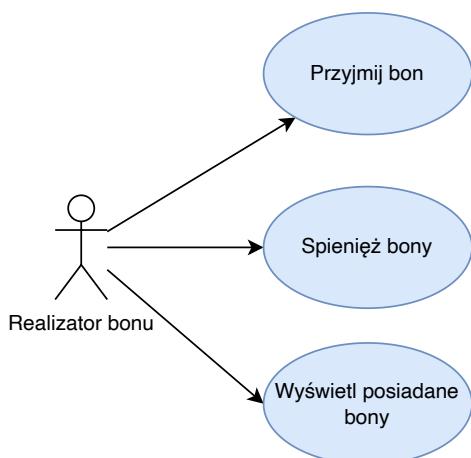
Realizator bonu Podmiot upoważniony przez administratora bonu do wymiany bonów na dobra cyfrowe lub materialne w zależności od charakterystyki danego bonu. Przypadki użycia zostały przedstawione na rysunku 12.



Rysunek 10: Diagrama przypadków Administrator bonu. Źródło: Źródło własne



Rysunek 11: Konsument bonu. Źródło: Źródło własne



Rysunek 12: Realizator bonu. Źródło: Źródło własne

Dostawca tożsamości System zewnętrzny zapewniający identyfikację oraz uwierzytelnienie użytkownika do systemu. Posiada on również wiedzę na temat wybranych atrybutów opisujących użytkownika takich jak wiek, czy miejsce zamieszkania. Warto podkreślić, że nie jest to aktor, lecz dopełnia przedstawiany obraz systemu.

2.2.6 CYKL ŻYCIA BONU

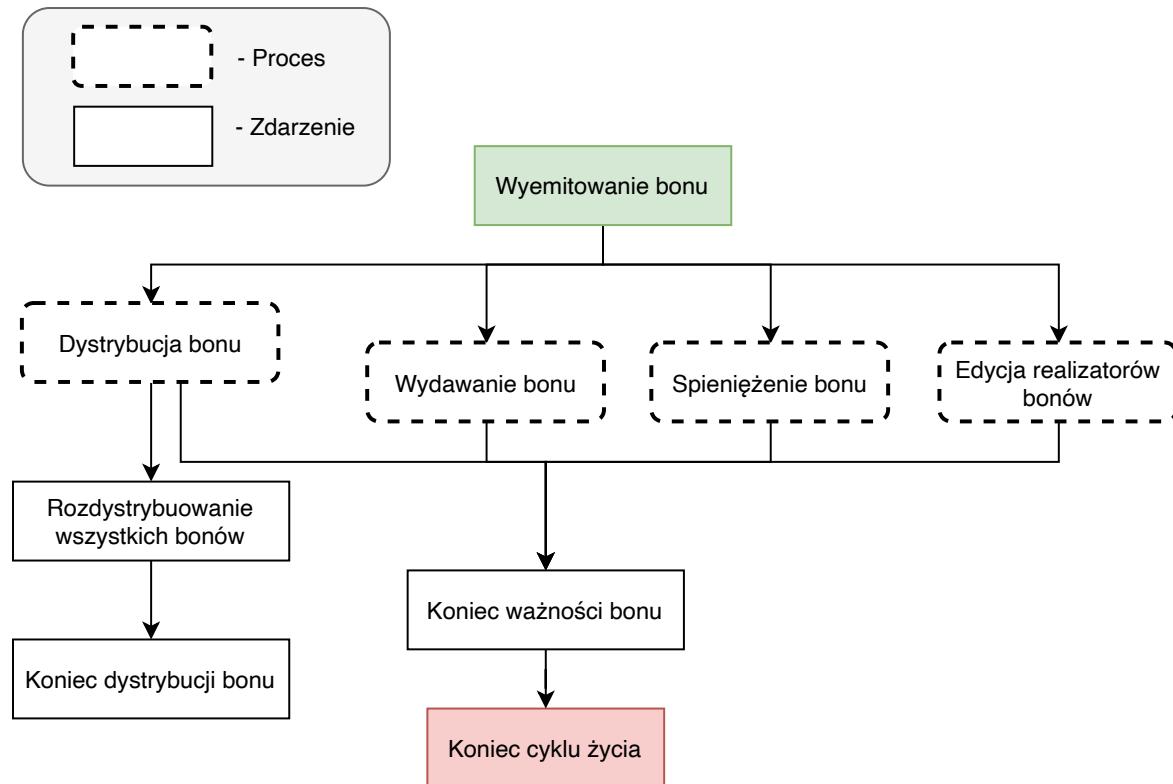
Cykł życia bonu ma na celu przedstawienie kluczowych procesów oraz ich relacji ze zdarzeniami. Diagram z rysunku 13 stanowi źródło do dalszej dekompozycji na procesy.

1. **Wyemitowanie bonu** Zdarzenie inicjujące cykl życia nowego bonu.
2. **Dystrybucje bonu** Przekazanie środków wygenerowanych przez emitenta do konsumentów bonu spełniających kryteria.
3. **Wydawanie bonu** Wymiana bonu za dobra, realizowana pomiędzy konsumenatem bonu posiadającego bony a realizatorem bonu.
4. **Spieniężenie bonu** Wymiana zebranych bonów przez realizatora bonu na PLN.
5. **Edycja realizatorów bonów** Dodanie lub usuwanie nowego realizatora bonu.
6. **Wydano wszystkie bony** Kończy proces dystrybucji bonów, ponieważ wszystkie bony zostały już rozdysytrybuowane.
7. **Koniec ważności bonu** Efekt końca czasu ważności bonu ustawionego podczas emisji bonu.
8. **Koniec cyklu życia** Zdarzenie definitywne kończące cykl życia bonu.

Należy podkreślić, że proces dystrybucji nie jest oddzielnym zdarzeniem, które musi ulec zakończeniu przed rozpoczęciem następnego, a toczy się równolegle z pozostałymi kluczowymi procesami.

2.3 DEFINICJA WYMAGAŃ

W celu wytworzenia spójnego i całościowego rozwiązania należy zdefiniować zbiór wymagań, które będą stanowiły podstawę do podejmowania decyzji w kwestii architektury oraz użytych technologii. Ze względu na akademicki charakter wdrożenia, lista wymagań opisuje minimalny użyteczny produkt. Wymagania zostały zdefi-



Rysunek 13: Cykl życia bonu. Źródło: Źródło własne

niowane według modelu FURPS, jest to akronimem słów:

1. Funkcjonalność (ang. **Functionality**) zestaw funkcji udostępnianych przez system
2. Użyteczność (ang. **Usability**) definiuje wymagania względem interfejsu użytkownika
3. Niezawodność (ang. **Reliability**) skwantyfikowany opis niezawodności systemu
4. Wydajność (ang. **Performance**) skwantyfikowany opis wydajności systemu
5. Elastyczność (ang. **Supportability**) opisuje takie cechy systemu jak rozszerzalność, skalowalność, testowalność [54]

2.3.1 FUNKCJONALNOŚĆ

1. **Emisja bonu**
 - a) **ID:** WF01
 - b) **Opis:** Emitent bonu za pośrednictwem strony www, może wygenerować nowy bon.

c) **Warunki:** Użytkownik jest uwierzytelniony jako Emitent.

2. Listowanie bonów przez emitenta

a) **ID:** WF02

b) **Opis:** Emitent bonów za pośrednictwem strony www, może zobaczyć wygenerowane bony.

c) **Warunki:** Użytkownik jest uwierzytelniony jako Emitent.

3. Nadanie roli realizatora bonu

a) **ID:** WF03

b) **Opis:** Administrator bonu za pośrednictwem strony www, może nadać użytkownikowi rolę realizatora bonu poprzez wybranie go z listy kandydatów.

c) **Warunki:** Użytkownik jest uwierzytelniony jako administrator obsługiwanej bonu. Potencjalny realizator bonu, nie jest realizatorem tego bonu.

4. Usunięcie realizatora bonu

a) **ID:** WF04

b) **Opis:** Administrator bonu za pośrednictwem strony www, może odebrać użytkownikowi rolę realizatora bonu.

c) **Warunki:** Użytkownik jest uwierzytelniony jako administrator obsługiwanej bonu.

5. Listowanie aktywnych realizatorów bonów

a) **ID:** WF05

b) **Opis:** Administrator bonu za pośrednictwem strony www, może przeglądać listę aktywnych realizatorów bonu.

c) **Warunki:** Użytkownik jest uwierzytelniony jako administrator obsługiwanej bonu.

6. Udział w dystrybucji bonu

a) **ID:** WF06

b) **Opis:** Konsument bonu może wziąć udział w dystrybucji każdego bonu, za pośrednictwem strony www.

c) **Warunki:** Użytkownik jest uwierzytelniony jako konsument. Użytkownik

nie brał jeszcze udziału w dystrybucji danego bonu.

7. Wydanie bonu

- a) **ID:** WF07
- b) **Opis:** Konsument bonu może wydać bon u realizatora danego bonu za pośrednictwem strony www, poprzez zeskanowanie kodu QR wyświetlanego przez realizatora bonu.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako konsument. Użytkownik posiada dany bon co najmniej w ilości, w której chce go wydać.

8. Listowanie posiadanych bonów przez konsumenta

- a) **ID:** WF08
- b) **Opis:** Konsument bonu może przeglądać posiadane bony oraz ich ilość, za pośrednictwem strony www.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako konsument.

9. Spieniężenie bonu

- a) **ID:** WF09
- b) **Opis:** Realizator może spieniężyć posiadane bony poprzez wskazanie numeru konta bankowego, za pośrednictwem strony www.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako realizator danego bonu. Użytkownik posiada dany bon.

10. Zgłoszenie kandydatury

- a) **ID:** WF10
- b) **Opis:** Konsument bonu może zgłosić kandydaturę do wybranego bonu, za pośrednictwem strony www.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako realizator. Użytkownika nie jest aktualnie realizatorem tego bonu.

11. Listowanie posiadanych bonów przez realizatora

- a) **ID:** WF11
- b) **Opis:** Realizator bonu może przeglądać posiadane bony oraz ich ilość, za pośrednictwem strony www.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako realizator.

12. Przyjęcie bony

- a) **ID:** WF16
- b) **Opis:** Realizator bonu może przyjąć bon poprzez wyświetlenie kodu QR bonu.
- c) **Warunki:** Użytkownik jest uwierzytelniony jako realizator. Użytkownik jest realizatorem danego bonu.

13. Uwierzytelnienie

- a) **ID:** WF12
- b) **Opis:** Użytkownik uwierzytelnia się do aplikacji poprzez dostawcę tożsamości.
- c) **Warunki:** Użytkownik posiada konto u dostawcy tożsamości.

14. Bezpieczeństwo kluczy prywatnych

- a) **ID:** WF13
- b) **Opis:** Aplikacja przechowuje klucze prywatne w tekście niejawnym. Klucze szyfrowane są po stronie klienta. Serwer nie ma dostępu do kluczy prywatnych w formie jawnej. Klucze szyfrowane są według standardu RFC 8018[55].

15. Anonimowość konsumenta

- a) **ID:** WF14
- b) **Opis:** Jako użytkownik systemu, który nie jest jego administratorem. Mając dostęp do publicznego łańcucha bloku, nie jesteśmy w stanie jednoznacznie powiązać adresu publicznego z konkretną osobą.

16. Audytowalność, przejrzystość rozwiązania

- a) **ID:** WF15
- b) **Opis:** Kluczowe procesy: dystrybucja, wydanie, emitowanie powinny być realizowane za pomocą smart-kontraktu.

2.3.2 UŻYTECZNOŚĆ

1. Standard interfejsu użytkownika

- a) **ID:** WU01

- b) **Opis:** Aplikacja implementuje standard tworzenie interfejsu użytkownika Material Design [56]

2. Responsywność na rozdzielczość

- a) **ID:** WU02
- b) **Opis:** Aplikacja wyświetla się poprawnie na wyświetlaczach o szerokości od 375px do 2560px, oraz o wysokości większej niż 600px.

3. Orientacja

- a) **ID:** WU03
- b) **Opis:** Aplikacja wyświetla się zawsze w orientacji pionowej.

2.3.3 NIEZAWODNOŚĆ

Brak, system nieprodukcyjny.

2.3.4 WYDAJNOŚĆ

1. Czas transferu

- a) **ID:** WW01
- b) **Opis:** Mediana czasu przelewu bonu nie powinna być dłuższa niż 12s przy 100 użytkownikach wykonujących tę samą operację równocześnie
- c) **Warunki:** Liczone od czasu zlecenia przelewu do otrzymania środków na koncie z pominięciem interfejsu użytkownika.

2. Czas dystrybucji

- a) **ID:** WW02
- b) **Opis:** Mediana czasu dystrybucji bonu nie powinna być dłuższa niż 40s przy 20 użytkownikach wykonujących tę samą operację równocześnie.
- c) **Warunki:** Liczone od zwrócenia atrybutów tożsamości od dostawcy tożsamości.

3. Czas stworzenia bonu

- a) **ID:** WW03
- b) **Opis:** Mediana czasu utworzenia bonu nie powinna być dłuższa niż 60s przy 10 użytkowników wykonującym tę samą operację równocześnie.

- c) **Warunki:** Liczone od czasu zlecenia utworzenia bonu do posiadania bonu gotowego do dystrybucji z pominięciem interfejsu użytkownika.

4. Skalowanie

- a) **ID:** WW04
- b) **Opis:** Wszystkie serwisy systemu powinny być bezstanowe lub oddelegować przechowywanie stanu do baz danych. Aby w łatwy sposób umożliwić skalowanie horyzontalne komponentów.
- c) **Warunki:** Brak.

2.3.5 ELASTYCZNOŚĆ

1. Testy integracyjne - warstwa danych

- a) **ID:** WS02
- b) **Opis:** Zestaw testów integracyjnych z warstwą danych.

2. Testy integracyjne - łańcucha bloków

- a) **ID:** WS03
- b) **Opis:** Zestaw testów integracyjnych z warstwą łańcuchów bloków.

3. Elastyczna architektura

- a) **ID:** WS04
- b) **Opis:** Architektura oprogramowania powinna być oparta na spójnych interfejsach, tak aby łatwo dokonywać zmian.

3 PRZEGŁĄD I ANALIZA ISTNIEJĄCYCH PUBLICZNYCH ŁAŃCUCHÓW BLOKÓW POD KĄTEM ZIDENTYFIKOWANYCH WYMAGAŃ

3.1 ANALIZA WYMAGAŃ POD KĄTEM ŁAŃCUCHA BLOKÓW

Główną technologią implementowanego systemu jest łańcuch bloków. Zakodowana jest w nim logika biznesowa odpowiedzialna za podstawowe procesy takie jak emisja bonu czy przelew bonu. Ze względu na krytyczność tych procesów, powinna zostać dobrana implementacja publicznego łańcucha bloków, która pozwoli sprostać zdefiniowanym wymaganiom. Na potrzeby analizy przygotowano zestaw wymagań wynikający bezpośrednio z wymagań zdefiniowanych w rozdziale 2.3, który został dodatkowo rozszerzony o dwa stopnie priorytetyzacji: wysoki z wagą 3 oraz niski z wagą 1. Ze względu na bardzo duży wybór publicznych łańcuchów bloków (powyżej 1000) do analizy zostały wybrane trzy popularne publiczne łańcuchy:

1. **Ethereum**[52] - Najpopularniejszy publiczny łańcuch bloków jako pierwszy udostępnił możliwość pisania smart kontraktów. Nie jest zoptymalizowany pod konkretne zastosowanie.
2. **Binance Smart Chain**[49] - Efektywny łańcuch bloków skupiony wokół eko-systemu firmy Binance. Zoptymalizowany pod kątem smart kontraktów wymagających szybkiego i płynnego działania względem użytkownika końcowego.
3. **Algorand**[51] - Nowatorski łańcuch bloków skupiony wokół zastosowań dotyczących finansów.

Tabela 2: Wymagania łańcucha bloków. Źródło: Źródło własne

Priorytet	Nazwa	Opis
Wysoki	Energoszczędność	Łańcuch bloków powinnien cechować się wysoką wydajnością energetyczną
Wysoki	Koszt transakcji	Koszt transakcji powinnien być mniejszy od 0.01 PLN
Wysoki	Czas finalizacji	Czas finalizacji transakcji powinnien być mniejszy od 10s
Wysoki	Język smart kontraktów	Jakość język smart kontraktów jako: ekspresywność języka, próg wejścia
Niski	Dokumentacja i narzędzia	Zasobność dokumentacji oraz narzędzi ułatwiających wytwarzanie rozwiązania
Niski	Stopień decentralizacji	W jakim stopniu sieć jest zdecentralizowana

3.2 ETHEREUM

3.2.1 KONSENSUS

Ethereum (ETH) to drugi najpopularniejszy łańcuch bloków, który jako pierwszy obsługiwał smart kontrakty kompletne w sensie Turinga.

Jest to łańcuch bloków ogólnego zastosowania, który opiera się na konsensusie dowodu pracy (patrz podrozdział 1.5.3) o nazwie “Ethash”. Problem obliczeniowy polega na analizie dużego zbioru danych, dokładniej acyklicznego grafu skierowanego (AGS). AGS na początku ma wielkość 1 GB, po czym rośnie liniowo. Aktualizacja rozmiaru odbywa się co 30 000 wydobytych bloków, czyli około 100 godzin.[5, s. 327] Zastosowanie tego typu zagadki miało na celu wykluczyć układy ASIC (specjalizowany układ scalony), które są o kilka rzędów wydajniejsze od kart graficznych. Przyczyną takiego zabiegu było uniknięcie centralizacji sieci, która wynikałaby z tego, że grupy z dużym budżetem i dostępem do specjalistycznych fabryk mogą zdominować infrastrukturę wydobywania bloków i zagrozić bezpieczeństwu algorytmu osiągania

konsensu.[5, s. 327] W algorytmie konsensu bitcoin'a nie został zastosowany taki mechanizm, przez co 51% mocy obliczeniowej bitcoin'a jest rozdzielona między 4 organizacje, co stawia pod znakiem zapytania jego decentralizację. Ponadto ETH w swoich planach ma przejść na konsensus oparty na dowodzie stawki (patrz podrozdział 1.5.4), który w odróżnieniu od algorytmu dowodu pracy nie opiera się na złożoności mocy obliczeniowej, co skutecznie odwleka konstrukcje ASIC przeznaczonych tylko i wyłącznie do algorytmu Ethash, ponieważ przestaną one być użyteczne wraz ze zmianą algorytmu konsensusu.

Aktualnie publiczny łańcuch bloków Ethereum jest rozproszony pomiędzy trzy tysiące węzłów utrzymujących rejestr wszystkich transakcji oraz weryfikujących każdą nowo powstałą transakcję[46]. Ponadto, algorytm dowodu stawki pracy, uważany jest za najbezpieczniejszy algorytm konsensusu. Co przy stopniu zdecentralizowania sieci Ethereum czyni ją jedną z najbezpieczniejszych.

3.2.2 SMART KONTRAKT

Smart kontrakty kompatybilne z publicznym łańcuchem bloków Ethereum mogą być pisane w dowolnym języku programowania, które kompilują się do bajt kodu interpretowanego przez wirtualną maszynę Ethereum (EVM). Jednak przodującym językiem, który został utworzony przez twórców EVM oraz zyskał największą popularność jest język Solidity.

Podstawowym konstruktorem języka jest kontrakt. Ze swojej definicji przypomina on klasę znaną z paradygmatu programowania obiektowego. Instancje kontraktu, posiadają swój zenkapsulowany stan, który można modyfikować poprzez wywołanie metod obiektu (instancji kontraktu). Kontrakty mogą też komunikować się z innymi instancjami kontraktów poprzez komunikaty.

```
contract Bon {
    address public minter;
    mapping (address => uint) public balances;
    constructor() {
        minter = msg.sender;
    }
}
```

```

}

function mint(address receiver, uint amount) public {
    require(msg.sender == minter);
    balances[receiver] += amount;
}

error InsufficientBalance(uint requested, uint available);

function send(address receiver, uint amount) public {
    if (amount > balances[msg.sender])
        revert InsufficientBalance({
            requested: amount,
            available: balances[msg.sender]
        });

    balances[msg.sender] -= amount;
    balances[receiver] += amount;
}
}

```

Kod 1: Smart Kontrakt w języku Solidity

Kod 1 ma za zadanie przedstawić istotę pisania smart kontraktów w języku Solidity. Kod zaczyna się od deklaracji samego smart kontraktu, poprzez użycie słowa kluczowego `contract`. Następnie deklarowany jest stan, a dokładniej dwie zmienne. Pierwsza reprezentuje adres użytkownika, który utworzył kontrakt, a druga słownik, gdzie kluczem jest adres a wartością liczba naturalna, która określa, który adres posiada ile środków. Kolejno deklarowany jest konstruktor, który przypisuje do zmiennej `minter` adres osoby, która utworzyła smart kontrakt. W ten sposób możemy określić administratora danego smart kontraktu. Obiekt `msg` przechowuje dane na temat transakcji, która go wywołała, w powyższym przykładzie skorzystano z pola `sender` które oznacza nadawcę, wartość tego pola to adres publiczny, odpowiadającym mu adresem prywatnym została podpisana sama transakcja. Następnie

zdefiniowane są dwie funkcje, pierwsza z nich umożliwia emisję nowego bonu we wskazanej ilości dla wskazanego odbiorcy. Druga pozwala na przesłanie środków z konta, które wywołało transakcje. W przypadku zbyt małej ilości środków na koncie zgłaszany jest błąd. Pisząc smart kontrakty należy pamiętać, że samo wywołanie smart kontraktu jeszcze nie zmienia jego stanu skutkuje jedynie jego symulacją oraz sprawdzeniem poprawności, zmiana następuje dopiero w momencie gdy transakcja zostanie przez sieć dołączona do rozproszonego rejestru.

3.2.3 DOKUMENTACJA, ŚRODOWISKO DEWELOPERSKIE

Z powodu swojej popularności oraz względnie długiego istnienia projekt ethereum posiada bardzo rozbudowaną dokumentację, oraz szeroką liczbę poradników. Posiada również pakiet narzędzi `ganache` [50], który między innymi pozwala na uruchomienie symulatora łańcucha bloków na własnym komputerze, co znacznie ułatwia wytwarzanie rozwiązania. Dodatkowym atutem jest również fakt istnienia zintegrowanych środowisk programistycznych (ang. IDE) wspierających język Solidity oraz platformę Ethereum. Dużym plusem ekosystemu Ethereum jest narzędzie `remix`, które umożliwia konstruowanie i uruchamianie smart kontraktów wyłącznie z użyciem przeglądarki. Tego typu narzędzia bardzo ułatwiają prototypownie rozwiązania. Sieć Ethereum posiada również sieć testową, w której prowadzone działania są darmowe.

3.2.4 EFEKTYWNOŚĆ SIECI

Elementem najbardziej wpływającym na parametry sieci jest algorytm konsensusu. W sieci Ethereum wykorzystywany jest konsensus dowodu pracy, który jak wskazuje nazwa ma duże zapotrzebowanie na energię. Szacowana ilość energii, jaką potrzebuje sieć Ethereum, aby przetworzyć jedną transakcję to około 70[kWh][47], punktem odniesienia może być standardowy odkurzacz, który zużywa równowartość tej energii przez cały rok[48].

Wyliczając koszt transakcji w sieci Ethereum należy uwzględnić zmienny koszt tzw. `gas`, czyli opłaty za wykonanie pojedynczej instrukcji bajtu kodu wirtualnej ma-

szyny Ethereum. Opłata za transakcję może różnić się w zależności od wywołanego smart kontraktu. Koszt jest zależny od ilości operacji które musi wykonać smart kontrakt. Koszt wykonania 100 operacji na maszynie wirtualnej Ethereum na moment pisania wynosi 0.0018 PLN. Ponadto należy doliczyć koszt transakcji, który zmienia się w zależności od obciążenia sieci i wynosi średnio 20 PLN.

3.2.5 OCENA

Tabela 3: Ocen Ethereum. Źródło: Źródło własne

Nazwa wymagania	Ocena	Uzasadnienie
Energoszczędność	1	Ponieważ algorytm konsensusu oparty jest na dowodzie pracy, jego zużycie energii jest niedopuszczalne
Koszt transakcji	1	Koszt transakcji wynosi conajmniej 20 PLN, a więc przekracza minimalny próg 0.01 PLN
Czas finalizacji	2	Czas finalizacji transakcji wynosi średnio 15s
Język smart kontraktów	3	Język jest wyrazisty oraz łatwy do nauki
Dokumentacja i narzędzia	3	Platforma Ethereum posiada szczegółową dokumentację
Stopień decentralizacji	3	Sieć jest rozproszona pomiędzy ok. 3000 węzłów

3.3 BINANCE SMART CHAIN

3.3.1 KONSENSUS

Binance Smart Chain (BSC) to publiczny łańcuch bloków, utworzony przez firmę Binance. Ma on stanowić alternatywę dla łańcucha Ethereum, która jest wydajniejsza oraz skalowalna.

Wyróżnikiem, który zapewnia te cechy, jest algorytm konsensusu oparty na dowodzie autorytetu i stawki (ang. Proof of Staked Authority (PoSA)). Dowód autorytetu opiera się na założeniu, iż istnieje określona liczba validatorów bloków w tym

wypadku 21. Dodatkowo algorytm ten rozszerzony jest o dowód stawki, polegający na tym, że każdy validator musi zablokować środki w postaci kryptowaluty BNB. Istnieje również możliwość delegowania środków na validatorów, poprzez zablokowanie kryptowaluty w odpowiednim smart kontrakcie. Dzięki tej technice większa ilość osób może pośrednio uczestniczyć w konsensusie, “oddając” swój głos na danego validatora. Jeżeli validator wykonuje pracę poprawnie, dzieli się on zyskami ze swoimi “wyborcami”. Natomiast jeśli będzie próbował propagować fałszywe bloki lub celowo spowalniać sieć, zablokowane przez niego środki zostaną mu odebrane. Głosujący nie ponoszą wtedy żadnych strat. Każdy validator sprawdza dany blok oraz podpisuje się pod nim, jeśli blok zostanie poprawnie zweryfikowany przez wszystkich validatorów jestłączany do łańcucha, a validatorzy otrzymują nagrodę za wydobycie bloku[44].

Każdy algorytm konsensusu to pewien kompromis pomiędzy trzema wymaganiami: skalowalności, bezpieczeństwa oraz decentralizacji. W tym algorytmie po-stawiono na skalowalność oraz bezpieczeństwo. Decentralizacja ogranicza się tylko do zbioru 21 validatorów. Twórcy tej sieci, aby zapobiec zbyt małej decentralizacji, postanowili oddelegować wybieranie validatorów oraz przetrzymywanie ich środków do innego bardziej zdecentralizowanego publicznego łańcucha - Binance Chain (BC)[45].

Podsumowując, konsensus BSC opiera się na zbiorze 21 validatorów, którzy są odpowiedzialni za dodawanie kolejnych bloków do łańcucha. Validatorzy biorą udział w konsensusie poprzez składanie kryptograficznego podpisu pod zweryfikowanym blokiem za pomocą własnego klucza prywatnego. Zbiór validatorów jest wybierany przez system smart kontraktów, gdzie decydentem jest ilość środków przez nich zdeponowanych oraz na nich oddelegowanych. Zbiór ten jest osadzony na łańcuchu Binance Chain i jest aktualizowany na łańcuchu BSC raz dziennie poprzez komunikacje między łańcuchami[42].

W przypadku konsensusu dowodu pracy sieć była zabezpieczona przez infrastrukturę sprzętu, który próbował nieustannie rozwiązywać trudno obliczeniowy problem, o konsensusie sieci stanowiła 51% tej mocy obliczeniowej. W przypadku do-

wodu stawki nie trzeba wykonywać trudno obliczeniowej pracy do wydobycia każdego pojedynczego bloku. Aby nadpisać łańcuch bloków, wystarczy posiadać 51% kryptowaluty “odpowiadających” za validację. Dlatego ważnym czynnikiem jest ekonomia danej kryptowaluty. W przypadku BSC używana jest waluta BNB, która na moment pisania jest trzecią walutą pod względem kapitalizacji z dziennym wolumenem ok. 9 mld PLN, a jej kurs wynosi ok. 2500 PLN. W chwili pisania 21 validatorów agreguje 16301701 BNB, aby przejąć 51% tych środków należałoby posiadać kapitał $0.51 \cdot 16301701 \cdot 2500 = 20,784,668,775$ PLN. W przypadku takiego ruchu zakupów cena waluty gwałtownie by wzrosła co spowodowałoby, że w rzeczywistości należałoby wydać znacznie większą kwotę.

3.3.2 SMART KONTRAKT

Binance Smart Chain jest kompatybilny z wirtualną maszyną Ethereum. Co oznacza, że obowiązują dokładnie te same zasady co w przypadku łańcucha ETH, opisany w poprzednim rozdziale. Ponadto BSC wykorzystuje zbliżoną implementację klienta, który wykorzystywany jest w sieci ETH (geth[41]), przez co również konsumowanie kontraktów jest tożsame z tym występującym na platformie ETH.

3.3.3 DOKUMENTACJA, ŚRODOWISKO DEWELOPERSKIE

Ponieważ Binance Smart Chain jest kompatybilny z Ethereum, większość dokumentacji oraz narzędzi technicznych wytworzonych w ramach projektu ETH może zostać również użyta do wytwarzania oprogramowania na platformę Binance Smart Chain. Pomimo tego BSC posiada własną rozbudowaną dokumentację utrzymywaną przez organizację Binance.

3.3.4 EFEKTYWNOŚĆ SIECI

Konsensus sieci opiera się na 21 validatorach, jest to mały zbiór w porównaniu do Ethereum, gdzie zbiór ten wynosił około 3 000. Każdy z validatorów musi również spełniać wymagania co do wydajności sprzętu, który wykorzystuje, w przypadku

ich niespełnienia zostanie pozbawiony roli validatora. Wymienione cechy łańcucha bloków przekładają się na jego wydajność, która w liczbach wynosi:

- koszt transakcji — ok. 2 PLN,
- koszt wykonania 100 operacji na maszynie wirtualnej gąs ok. 0.00025 PLN,
- Czas finalizacji transakcji ok. 3 sekund.

3.3.5 OCENA

Tabela 4: Ocena Binance Smart Chain. Źródło: Źródło własne

Nazwa wymagania	Ocena	Uzasadnienie
Energoszczędność	3	Ponieważ algorytm konsensusu oparty jest na rozszerzonym dowodzie stawki, w którym tylko 21 validatorów jest aktywnych naraz jego zużycie energii jest akceptowalne.
Koszt transakcji	1	Koszt transakcji wynosi ok. 2 PLN a więc przekracza minimalny próg 0.01 PLN
Czas finalizacji	3	Czas finalizacji transakcji wynosi średnio 3 sekundy
Język smart kontraktów	3	Język jest wyrazisty oraz łatwy do nauki
Dokumentacja i narzędzia	3	Platforma Binance posiada szczegółową dokumentację dzieloną z siecią Ethereum
Stopień decentralizacji	1	Siec jest rozproszona jedynie pomiędzy 21 validatorów. Jest zabezpieczona jedynie przez kapitał, który jest płynny i może być natychmiastowo przekazywany z jednego konta na drugie

3.4 ALGORAND

3.4.1 KONSENSUS

Algorand to publiczny łańcuch bloków, utworzony przez renomowanego kryptografa Silvio Micali. Jest to znany twórca protokołów, między innymi dowodów o

zerowej wiedzy, laureat nagród: Gödel (1993), Turing (2012). Sieć Algorand jest przedstawiana jako solucja dla tak zwanego potrójnego problemu publicznych łańcuchów bloków, w którym każdy z łańcuchów bloków, może mieć maksymalnie dwie z następujących cech: bezpieczeństwo, skalowalność, decentralizacja. Twórcy Algorand'a twierdzą, że ich nowatorski konsensus rozwiązuje wszystkie trzy problemy jednocześnie.

Zastosowany algorytm konsensu nazywa się czystym dowodem stawki (PPoS z ang. Pure Proof of Stake). Sercem tego algorytmu jest weryfikowalna losowa funkcja VRF [40] (z ang. Verifiable Random Function), którą możemy zdefiniować jako zestaw 3 funkcji:

1. $\text{GenerujKlucz}(\text{ziarno}) \rightarrow (VK, SK)$ - Na wejściu losowego ziarna, zwracany jest klucz weryfikacyjny (publiczny) VK oraz klucz tajny SK .
2. $\text{Losuj}(SK, M) \rightarrow (Y, p)$ - Na wejściu podajemy tajny klucz SK oraz wiadomość M , na wyjściu otrzymujemy pseudolosową wartość Y oraz dowód p ,
3. $\text{Zweryfikuj}(VK, M, Y, p) \rightarrow 0|1$ - Na wejściu podajemy klucz weryfikacyjny VK , wiadomość M oraz dwójkę stanowiącą dowód Y, p . Na wyjściu otrzymujemy 1 jeśli dowód jest poprawny 0 w przeciwnym wypadku.

Funkcja ta stosowana jest do przeprowadzania loterii z dowodem poprawności. Każdy z uczestników sieci posiada parę (VK, SK) , która obliczalna jest za pomocą funkcji jednokierunkowej z klucza publicznego danego uczestnika, a następnie rejestrowana za pomocą transakcji. VK jest publicznie znane, zaś SK znane jest wyłącznie uczestnikowi sieci. Aby wziąć udział w loterii, węzeł oblicza $\text{Losuj}(SK, P) \rightarrow (Y, p)$, gdzie P to publiczny statyczny parametr z poprzedniej rundy, jeżeli $Y \in [0, A]$, oznacza to, że użytkownik wygrał w loterii, A oznacza ilość kryptowaluty (Algos), jaką dany uczestnik posiada (im więcej waluty, tym większa szansa na wygranie w loterii). Jednakowo parametr p oraz stan A danego konta jest znany wszystkim węzłom z definicji działania łańcucha bloków.

Konsensus PPoS opiera się trzech krokach

1. **Propozycja bloku** - Każdy z uczestników bierze udział w loterii za pomocą VRF $\text{Losuj}(SK, \text{blok}) \rightarrow (Y, p)$. Jeżeli "wygra" w loterii przekazuje swoją pro-

pozycję bloku wraz z dowodem wygranej ($VK, blok, Y, p$) do pozostałych węzłów.

2. **Miękkie losowanie** - Każdy z wierzchołków otrzymuje wiele propozycji bloku w postaci ($VK, blok, Y, p$). Sprawdzają poprawność VRF za pomocą funkcji $Zwerfikuj(VK, blok, Y, p)$. Następnie ze wszystkich propozycji bloku wybiera ten, którego $h(VK||blok)$ ma najmniejszą wartość oraz rozsyła go dalej. Po stałym czasie t , odbywa się kolejna runda losowania. W której uczestnicy ustalają ostateczny blok. Jeżeli ponad połowa uczestników zgodzi się na ten blok, można przejść do następnego kroku (każdy węzeł podejmuje tę decyzję niezależnie od pozostałych).
3. **Certyfikacja bloku** - Ponownie uruchamiana jest funkcja VRF przez każdego z uczestników, aby tym razem wybrać, węzły, które będą odpowiedzialne za certyfikację bloku. Certyfikacja bloku polega na sprawdzeniu poprawność wszystkich transakcji w zadanym bloku. Jeżeli ponad połowa losowo wybranego komitetu zagłosuje na ten blok, jest on dopisywany do łańcucha bloków.

Konsensus ten zapobiega wielu problemom występujących algorytmach dowodu stawki:

- Brak ograniczania się do X validatorów tak jak w przypadku sieci BSC,
- Każdy uczestnik sieci może stać się validatorem, dzięki VRF,
- Nie jest możliwe tworzenie ataków na konkretne węzły, ponieważ atakujący dowiaduje się, że węzeł jest kluczowy w danej rundzie, w momencie gdy ten już wykonał swoją pracę.

3.4.2 SMART KONTRAKT

Smart kontrakty platformy Algorand, możemy podzielić na dwa typy, kontrakty stanowe oraz bez stanowe. Obydwa typy kontraktów są pisane w języku TEAL. W praktyce wykorzystuje się obydwa typy smart kontraktów, łącząc je ze sobą.

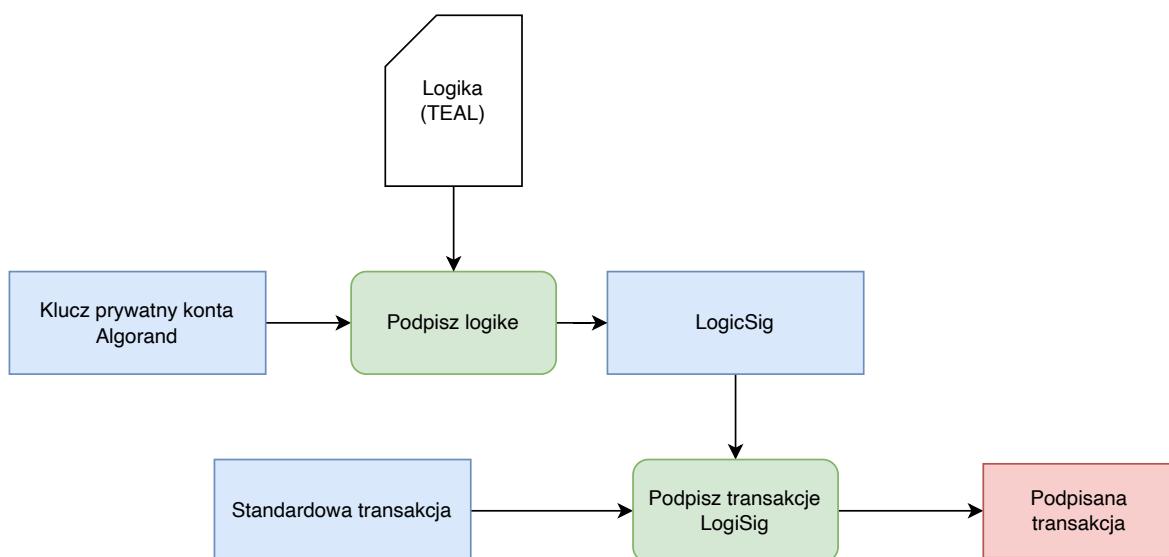
KONTRAKTY BEZ STANOWE Kontrakty bez stanowe nazywane również podpisami logicznymi można dalej podzielić na dwie podkategorie: konta kontraktowe oraz de-

legowane podpisy (przedstawiono na rysunku 14). Zasada działania tych dwóch trybów opiera się na konstrukcie o nazwie **LogicSig**, oznaczającym podpis logiczny[37].

LogicSig można opisać jako następującą strukturę:

- Logika - smart kontrakt napisany w TEAL, musi zakończyć się wynikiem pozytywnym, aby podpisana przez niego transakcja doszła do skutku (wymagane)
- Podpis - Podpisana logika kluczem prywatnym (opcjonalne)

Drugie pole wykorzystywane jest tylko w trybie delegowanego podpisu. Delegowanie podpisu oznacza, że osoba składająca podpis oznajmia: "możesz wykonać transakcje w moim imieniu, jeżeli Logika jest spełniona". Innymi słowy, zezwala na wykorzystywanie swojego konta na zasadach opisanych w Logika.



Rysunek 14: Tryb delegowania. Źródło: Źródło własne

Wynik funkcji skrótu logiki programu stanowi adres konta, tak jak klucz publiczny w przypadku standardowego konta. Dzięki czemu kontrakt ten może być celem innych transakcji. Po wykonaniu transakcji przelewu Algos na taki adres, konto staje się aktywne.

KONTRAKTY STANOWE Kontrakty stanowe nazywane również aplikacjami w ekosystemie Algorand. W przeciwnieństwie do kontraktów bez stanowych nie autoryzują standardowych transakcji, ale manipulują stanem lokalnym oraz globalnym. Stan lokalny jest oddzielny dla każdego konta, które chce wziąć udział w smart kontrakcie,

jego przestrzeń adresowa to 16 par klucz — wartość. Stan globalny dotyczy całej aplikacji. Jego przestrzeń adresowa wynosi 64 pary klucz — wartość. Wszystkie klucze oraz ich wartości mają pojemność 64 bitów[38].

Na kontrakt stanowy składają się dwa kody źródłowe: `ApprovalProgram` - służy do interakcji z aplikacją, edycji stanu oraz `ClearStateProgram` - służy do czyszczenia stanu aplikacji. Istnieje zbiór zdefiniowanych wywołań aplikacji. Wszystkie wywołania oprócz `ClearState`, są przetwarzane przez `ApprovalProgram`. Dostępne wywołania:

- **OptIn** Wzięcie udziału w aplikacji. Aplikacja uzyskuje dostęp do stanu lokalnego konta.
- **NoOp** Standardowe wywołanie aplikacji.
- **UpdateApplication** Aktualizacja kodu smart kontraktu.
- **DeleteApplication** Usunięcie smart kontraktu.
- **CloseOut** Przeciwieństwa operacji **OptIn**, czyli rezygnacja z udziału w aplikacji.
- **ClearState** To samo co powyższe, z tym że zawsze kończy się opuszczeniem aplikacji niezależnie od tego czy logika opuszczania transakcji wykona się prawnie.

Smart kontrakty w systemie Algorand mogą również odczytywać globalny stan maksymalnie dwóch innych aplikacji oraz stan lokalnych dodatkowych czterech kont. Muszą być one określone w chwili tworzenia smart kontraktu.

Językiem smart kontraktów Algorand jest Transaction Execution Approval Language (TEAL). Jest on oparty o maszynę stosową co oznacza, że wywoływane funkcje, zapisują i odczytują argumenty ze stosu[39]. Jest to bardzo prymitywny język przypominający asemblera np. brak w nim konstruktu funkcji. Jest również niekompletny w sensie Turinga — nie występują operacje, które pozwalałyby na “skoki” do tyłu. Taki wybór ma między innymi na celu zwiększenie bezpieczeństwa oraz zmniejszenie złożoności smart kontraktów. W TEAL występują tylko dwa typy danych `unit` 64 oraz tablica bajtów `[] byte`.

`tx` `n Receiver`

```

addr QZBGY6BGX2GA5YCPBY2UJ5HWRLY3U2XBJOPSNVOSIGJMPQYNGFXBCHHQYM
==

txn Amount
int 30000
>
&&

txn Amount
txn Fee
+
int 31000
<
&&

```

Kod 2: Smart Kontrakt w języku TEAL

Przedstawiony powyżej wycinek kodu 2, sprawdza, czy odbiorcą jest wskazany adres, czy kwota transakcji nie przekracza 30000 microAlgos (10^{-6} Algos) oraz czy razem z kosztem transakcji nie przekracza 31000 microAlgos.

Algorand posiada również pojęcie transakcji grupowych. Pozwala ono zgrupować do 16 transakcji. Aby tak zgrupowane transakcje zostały dołączone do bloku, wszystkie z nich muszą wykonać się pozytywnie, gdy choć jedna transakcja zakończy się niepowodzeniem, wszystkie transakcje zostaną odrzucone. Istotny jest fakt, że smart kontrakty mogą odczytywać transakcję, z którymi są zgrupowane, umożliwia to na przykład grupowanie kontraktów stanowych z kontraktami bez stanowymi.

3.4.3 DOKUMENTACJA, ŚRODOWISKO DEWELOPERSKIE

Algorand posiada bardzo szczegółową oraz obszerną dokumentację, niestety na ten moment niedostępna w języku polskim. Na stronie Algorand można także odnaleźć liczne poradniki opisujące konkretne implementacje. Posiada również bibliotekę wspieraną przez Algorand Foundation do takich języków jak: javascript, go, python, C#. Ponadto istnieje oficjalny kanał komunikacyjny, który pozwala programistom w

tym twórcom rozwiązania Algorand dzielić się doświadczeniami oraz pomagać sobie wzajemnie.

Algorand utrzymuje również narzędzie konsolowe - `goal`, które umożliwia interakcje z publicznym łańcuchem bloków, tworzenie transakcji, tworzenie smart kontraktów itd. Społeczność przyczyniła się również do utworzenia programu `sandbox`, który umożliwia utworzenie lokalnej symulacji łańcucha bloków Algorand, dzięki czemu testy można wykonywać na maszynie lokalnej.

3.4.4 EFEKTYWNOŚĆ SIECI

Konsensusy oparte na dowodzie stawki charakteryzują się wysoką wydajnością, w przypadku Algorand dodatkowym atutem są zoptymalizowane na szybkość wykonania smart kontrakty, które dodatkowo przyspieszają działanie sieci w porównaniu do rozbudowanych smart kontraktów stosowanych w ETH. Optymalizacja ta bierze się z licznych ograniczeń opisanych wcześniej. Efektem jest czas finalizacji transakcji, który wynosi ok. 5 sekund.

Wszystkie opłaty w sieci Algorand płacone są z wykorzystaniem natywnej waluty `Algo`. W przeciwieństwie do pozostałych łańcuchów bloków każde aktywne konta musi posiadać co najmniej 0.1 `Algo`. Dodatkowymi opłatami są również obarczone takie operacje jak tworzenie smart kontraktu, zapisywanie się do kontraktu, koszty te wyliczane są z następującego wzoru:

$$100000 + 285000 \cdot \text{ints} + 50000 \cdot \text{bytes}$$

gdzie `bytes` to liczba bajtów, a `ints` to liczba 64-bitowych liczb całkowitoliczbowych użytych w smart kontraktcie. W przypadku tworzenie smart kontraktu parametry te odpowiadają zmiennym globalnym, przy zapisywaniu się do smart kontraktu tyczą się one zmiennych lokalnych. Koszt transakcji jest stały i niezależny od typu transakcji, wynosi 0.001 `Algo`. Wykonywanie smart kontraktów nie jest obarczone dodatkowymi opłatami w postaci `gas`(tak jak w przypadku ETH czy BSC), lecz ograniczona jest ich wielkość. Każda z operacji posiada własny koszt. Maksymalny koszt kontraktu stanowego to 700, zaś bez stanowego to 20 000.

Algorand stara się również promować sieć jako przyjazną środowisku, ponieważ koszt energetyczny jednej transakcji to około 0,000008 kWh, jest to około 9 milionów razy mniej niż w przypadku Ethereum i ok. 116 milionów mniej niż w przypadku Bitcoin'a.

3.4.5 OCENA

Tabela 5: Ocena Algorand. Źródło: Źródło własne

Nazwa wymagania	Ocena	Uzasadnienie
Energoszczędność	3	Zespołowi Algorand udało się wypracować bardzo efektywne energetycznie konsensus łańcucha bloków
Koszt transakcji	3	Koszt transakcji wynosi 0,006 PLN a więc nie przekracza minimalnego progu 0.01 PLN
Czas finalizacji	3	Czas finalizacji transakcji wynosi średnio 5s, nie przekracza progu 15s
Język smart kontraktów	1	Język smart kontraktów posiada sporo ograniczeń zdecydowanie nie jest on przyjazny deweloperom
Dokumentacja i narzędzia	3	Platforma Algorand posiada szczegółową dokumentację oraz aktywnie wspierającą społeczność
Stopień decentralizacji	3	Sieć jest rozproszona pomiędzy ok. 1400 węzłów

3.5 WNIOSKI

Pomimo tego, że wszystkie z publicznych boków podają tą samą ideą budowania zdecentralizowanych sieci peer-to-peer, działających logicznie jako jeden komputer. Wykorzystane w nich technologie, algorytmy oraz protokoły różnią się od siebie, co bezpośrednio przekłada się na cechy danego łańcucha, co z kolei warunkuje możliwość jego wykorzystania w danym systemie.

Na potrzeby tego systemu zostały zestawione trzy łańcuchy bloków: Ethereum,

Binance Smart Chain oraz Algorand. Tabela 6 przedstawia punktację tych łańcuchów względem zdefiniowanych wcześniej wymagań. Każde wymaganie mogło zostać ocenione w skali 1-3, gdzie punktacja danego wymagania jest obliczona jako $P = \text{waga wymagania} * \text{ocena}$, następnie punktacja jest sumowana. W rezultacie otrzymuje się ostateczną ocenę danego łańcucha bloków. Za najbardziej odpowiedni do zdefiniowanych wymagań publiczny łańcuch bloków uznano Algorand.

Tabela 6: Podsumowanie ocen. Źródło: Źródło własne

	Nazwa sieci	Ethereum	Binance Smart Chain	Algorand
Wymagania	Energoszczędność (Wysoki)	1	3	3
	Koszt transakcji (Wysoki)	1	1	3
	Czas finalizacji (Wysoki)	2	3	3
	Język smart kontraktów (Wysoki)	3	3	1
	Dokumentacja i narzędzia (Niski)	3	3	3
	Stopień decentralizacji (Niski)	3	1	3
Suma ważona		27	32	36

4 ARCHITEKTURA OPROGRAMOWANIA SYSTEMU OBIEGU TOKENÓW

Na potrzeby tej pracy architektura oprogramowania definiowana jest według słów Philippe Kruchten, który określa architekturę oprogramowania jako: *zbiór istotnych decyzji dotyczących:*

- *organizacji systemu oprogramowania*
- *wyboru elementów strukturalnych i ich interfejsów, z których system jest zbudowany, razem z ich zachowaniami wyspecyfikowanymi w ich kooperacjach*
- *stylu architektonicznego, według którego tworzy się konstrukcję systemu, to znaczy charakterystyczne elementy i ich interfejsy, od którego zależy kooperacja i składanie[36].*

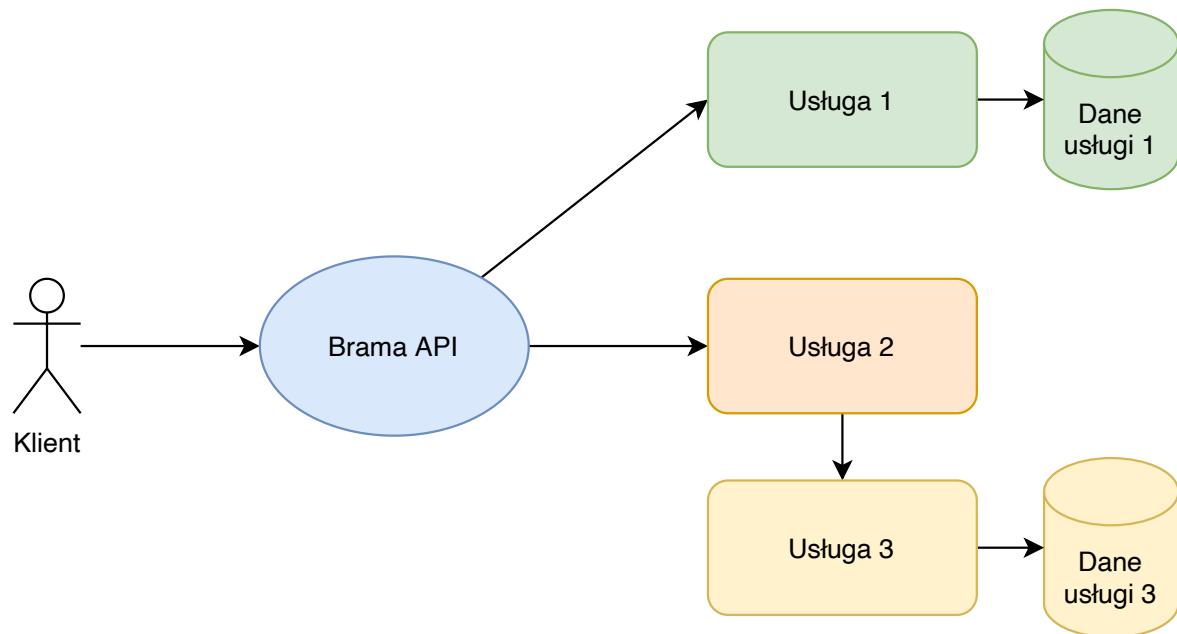
W rozdziale drugim został zdekomponowany cykl życia bonu, który pozwolił na wyróżnienie aktorów (rzeczników) występujących w systemie oraz interakcji (czasowników) między nimi zobrazowanych w postaci diagramów przypadku użycia. Na podstawie tych informacji należy wytworzyć architekturę systemu. Gdzie system definiowany jest jako zbiór komponentów oraz relacji występujących między tymi komponentami, współpracujące w celu osiągnięcia wspólnego celu. Pod postacią wymagań funkcjonalnych oraz niefunkcjonalnych zostały również zebrane cele, które musi spełniać architektura.

4.1 STYL ARCHITEKTURY

Styl architektury nakłada ramy, w jakich będą podejmowane dalsze decyzje dotyczące architektury. Określa główne granice systemu oraz wpływa na sposób wyodrębniania komponentów oraz organizacji oprogramowania. Nie istnieje jeden uniwersalny styl architektury, który pasuje do każdego systemu. Celem wybrania optymalnego stylu dla naszego systemu, zostaną opisane najpopularniejsze style architektoniczne, a następnie wybrany optymalny z nich.

4.1.1 ARCHITEKTURA MIKRO USŁUGI

Architektura oparta na minimalistycznych usługach stała się bardzo popularna w ostatnich czasach, co nie stanowi zaskoczenia przy ilości zalet, jakie za sobą niesie. Styl ten opiera się na rozproszeniu funkcji systemu pomiędzy pojedyncze komponenty, w czego wyniku implementują one tylko i wyłącznie jedną odpowiedzialność systemu. Każda z takich mikro usług musi posiadać własny kod źródłowy niezależny od innych usług. Komponenty komunikują się między sobą za pomocą interfejsu sieciowego. Każda usługa jest również odpowiedzialna za swój stan, to znaczy, że tylko ona może z niego zarówno odczytywać dane jak i zapisywać. Znaczącym komponentem w mikro usługach jest brama API (ang. gateway API). Jest to usługa stanowiąca wejście do systemu dla użytkownika końcowego, takie działanie odciąża klienta z posiadania wiedzy na temat tego, jaką usługę powinien wywołać dla danej funkcji. Schemat architektury mikro usług został przedstawiony na rysunku 15[7, s. 15 -16].



Rysunek 15: Architektura mikrouslug. Źródło: Źródło własne

Jako zalety architektury opartej na mikro usługach możemy wymienić:

1. Granularna skalowalność — Możemy skalować jedynie te usługi, które są najbardziej wykorzystywane w przeciwieństwie do skalowania całego systemu równocześnie.

2. Zakres technologii — Każdy serwis może zostać zaimplementowany w oddzielnym języku. Pozwala to na wybranie języka najlepiej dopasowanego do zadania.
3. Zwinność — pojedyncze usługi mogą być rozwijane niezależnie przez oddzielne zespoły deweloperskie, co optymalizuje organizację pracy w dużych zespołach programistycznych
4. Izolacja błędów — porażka jednej usługi możemy odciąć użytkownika tylko od danej funkcji, nie powoduje awarii całego systemu[7, s. 16-17].

Wszystkie te zalety niosą za sobą jedną wadę, wytwarzanie oprogramowania w rigorze architektury mikro usług jest bardzo pracochłonne. Pracochłonność ta bierze się ze złożonością całej architektury oraz ilością operacji potrzebnych do utrzymywania wielu niezależnych usług. Wada ta znika przy dużych zespołach programistycznych, które posiadają, specjalne zespoły odpowiedzialne za automatyzacje procesów twórczych. Pomimo swoich licznych zalet architektura oparta na mikro usługach niesie zbyt duży narzut złożoności, aby zaimplementować ją w omawianym systemie.

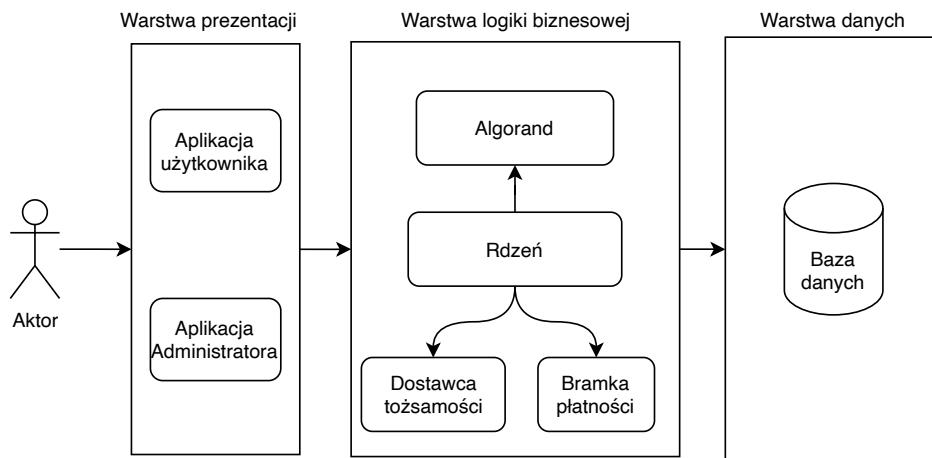
4.1.2 ARCHITEKTURA N-WARSTW

Architektura n-warstw polega na rozdzieleniu odpowiedzialności komponentów między n-grup. Znając zasadę działania architektury opartej o mikro usługi, architekturę n-warstw możemy opisać jako n-usług, gdzie każda z nich może komunikować się tylko z usługą niższej lub w tej samej warstwie[7, s. 7,8]. Jednocześnie porzucając izolacje stanu usługi. Obrazując, z siatki usług przechodzimy do linii. Taka konstrukcja pozwala nam odseparować pomiędzy warstwami odpowiedzialności oraz zależności ze względu na n wydzielonych warstw. W przypadku tego systemu wydzielono trzy warstwy (zobrazowane na rysunku: 16):

1. Warstwa prezentacji — warstwa ta będzie odpowiedzialna za interakcje użytkownika z systemem. Znajdują się tu dwa interfejsy, jeden dla konsumenta i realizatora druga dla administratora i emitenta bonu.
2. Warstwa logiki biznesowej — ponosi odpowiedzialność za wykonywanie wymagań funkcjonalnych. Rdzeń to główna logika biznesowa, która komunikuje

się z zewnętrznymi systemami, jak publiczny łańcuch bloków oraz dostawca tożsamości

3. Warstwa danych — Warstwa służąca do utrwalania danych oraz ich przeszukiwania. Znajduje się w niej jedna baza danych ogólnego przeznaczenia.



Rysunek 16: Architektura 3-warstw. Źródło: Źródło własne

W grupowaniu według warstw brakuje separacji ze względu na funkcję, które występowała w przypadku mikro usług. Można osiągnąć tę zaletę korzystając ze wzorca pakowania według komponentów. Pakowanie według komponentów polega na podzieleniu kodu na hermetyczne komponenty, które udostępniają zwięzły interfejs. Wewnątrz komponentu nadal zachowany jest rozdział odpowiedzialności według modelu warstwowego, czyli warstwy prezentacji, logiki biznesowej oraz warstwa danych, ale stanowi to szczegół implementacyjny ukryty pod interfejsem. Architektura jest bardzo zbliżona do architektury mikro usług, główną różnicą jest sposób rozdzielenia komponentów. W przypadku mikro usług rozdzielenie to było na poziomie oddzielnych kodów źródłowych, technologii implementacji, repozytorium danych. W odniesieniu do pakowania według komponentów separacja jest na poziomie jednego kodu źródłowego, wymuszona przez kompilator poprzez zachowanie odpowiedniej praktyki stosowania modyfikatorów dostępów oraz podziału kodu na moduły[6, s. 314]. Technologia implementacji jest wspólna dla wszystkich komponentów. Komponenty nadal odpowiedzialne są za swój stan, lecz może być on przechowywany w jednolitej bazie danych. Podsumowując pakowanie według kom-

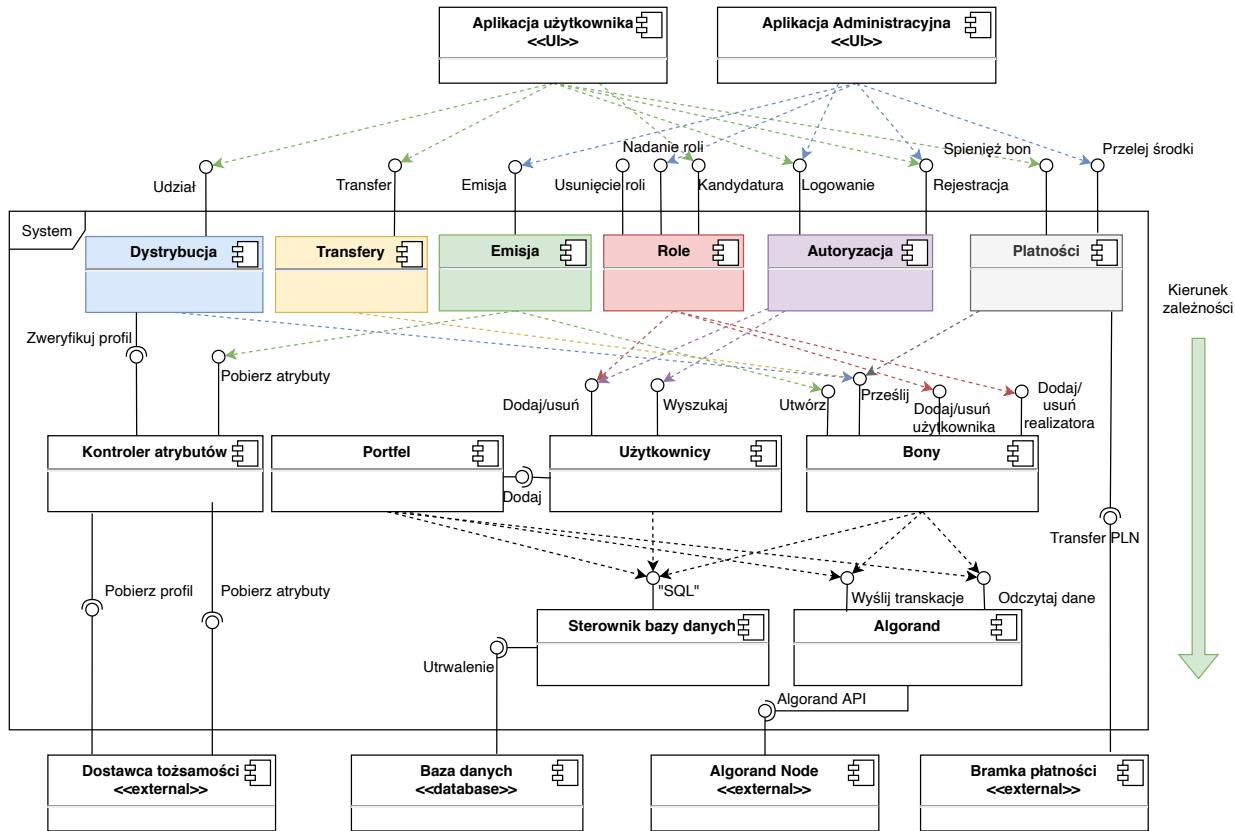
ponentów stanowi dobry wstęp do architektury mikro usług, jednocześnie minimalizując ich złożoność i pracochnośc.

4.2 KOMPONENTY I ICH INTERFEJSY

W poprzednim rozdziale system został zdekomponowany na warstwy. Warstwy podlegają dalszej dekompozycji na komponenty, które odpowiedzialne są za funkcje występujące w systemie. Każdy z komponentów, udostępnia swój interfejs, który pozwala ukryć przed komponentem z niego korzystającym szczegóły implementacyjne. Szczegóły implementacyjne mogą zmieniać się niezależnie od innych komponentów, dopóki interfejs pozostaje niezmodyfikowany. Dlatego prawidłowe zaprojektowanie interfejsów ma kluczowe znaczenie dla systemu. To od nich zależy, jak system będzie elastyczny na zmiany wymagań funkcjonalnych.

W omawianym systemie występuje czterech aktorów: realizator bonów, konsument, administrator bonu oraz emitent. Za każdym razem gdy w systemie będzie dokonywana jakaś zmiana, będzie ona wypływać od jednego z tych interesariuszy. Należy zatem odizolować od siebie komponenty w ten sposób, aby zmiany wprowadzane dla jednego aktora nie wpływały na drugiego[6, s. 298].

Na diagramie 17 widzimy podział komponentów ze względu na funkcje, za które odpowiadają. Funkcje te natomiast biorą się z przypadków użycia określonych w rozdziale drugim. Można zauważyć, że na diagramie nie istnieją żadne zależności acykliczne. Innymi słowy, żadna para komponentów nie zależy od siebie wzajemnie, jest to bardzo pożądana cecha architektury. W przypadku zależności acyklicznych, zmieniając interfejs jednego z komponentów, należy również brać pod uwagę drugi komponent. Ponadto komponenty dziedziczą po sobie wszystkie pozostałe zależności. Systemy tworzone w ten sposób bardzo szybko stają się trudne do modyfikacji. Druga zasadą, jaką zastosowano przy tworzeniu tego diagramu, jest zasada stabilnych zależności. Na diagramie zielona strzałka wskazuje kierunek zależności. Przez zależność rozumie się, że jeżeli komponent A jest zależny od komponentu B - ($A \rightarrow B$) to wtedy zmiana interfejsu w B wymaga również zmiany w A . Intuicyjnie można zauważyć, że komponenty stabilne, które mają najmniej powodów do zmian, powinny



Rysunek 17: Diagram komponentów. Źródło: Źródło własne

znajdować się na samym dole diagramu. Komponenty, które z kolei będą zmieniały się najczęściej, czyli te dotyczące “kruchej” logiki biznesowej powinny znajdować się na samej górze [6, s. 138].

OPIS KOMPONENTÓW Ze względu na złożoność diagramu komponentu pominięto na nim funkcjonalności związane z pobieraniem danych. W systemie istnieją dwie encje Użytkownika oraz Bon, wszystkie dane przechowywane są dostępne z poziomu tych dwóch encji, każdy komponent brzegowy (oznaczone innym kolorem niż biały), udostępnia te dane w postaci skrójonej pod swoją funkcjonalność.

1. Aplikacja administracyjna — To komponent warstwy prezentacji, który służy jako pośrednik w komunikacji z systemem dla aktorów emitent oraz administrator bonu.
2. Aplikacja użytkownika — Podobnie jak aplikacja administratora, ale przeznaczona jest dla użytkownika oraz realizatora bonu.

3. Dystrybucja — Komponent odpowiedzialny za logikę biznesową związaną z dystrybucją bonu.
4. Transfery — Komponent odpowiedzialny za logikę biznesową związaną z przesyłaniem bonu pomiędzy użytkownikiem a realizatorem bonu.
5. Emisja — Komponent odpowiedzialny za emitowanie bonu.
6. Role — Komponent odpowiedzialny za nadawanie i odbieranie roli realizatora.
7. Autoryzacja — Komponent odpowiedzialny za logowanie oraz rejestracje użytkownika w systemie.
8. Płatności — Komponent służący do komunikacji z zewnętrzną bramką płatności, w celu opłacenia emisji bonu lub jego spieniężenia.
9. Kontroler atrybutów — Potrafi rozwiązywać logikę związaną ze zgodnością użytkownika z profilem dystrybucji danego bonu. W zaawansowanym scenariuszu może być on agregatorem wielu dostawców tożsamości oraz weryfikować profil w oparciu o zdania logiczne łączące atrybuty z wielu systemów.
10. Dostawca tożsamości — Zewnętrzny podmiot, który pozwala na integracje na podstawie standardu OAuth 2.0 (patrz podrozdział 5.5).
11. Użytkownicy — Komponent odpowiedzialny za zarządzanie danymi użytkownika.
12. Portfel — Wydzielona encja z Użytkownika, która służy obsłudze przechowywania danych dotyczących portfela użytkownika.
13. Bony — Komponent udostępnia prosty zbiór funkcji bonu, stanowi on abstrakcję smart kontraktów.
14. Sterownik bazy danych — Komponent stanowiący abstrakcję bazy danych. Od uzależnia od konkretnej bazy danych.
15. Algorand — Komponent służący do abstrakcji interfejsu udostępnianego przez węzeł Algorand.
16. Baza danych — Baza danych
17. Algorand Node — Węzeł publicznego łańcucha bloków Algorand.
18. Bramka płatności — Zewnętrzny system do obsługi transferów PLN.

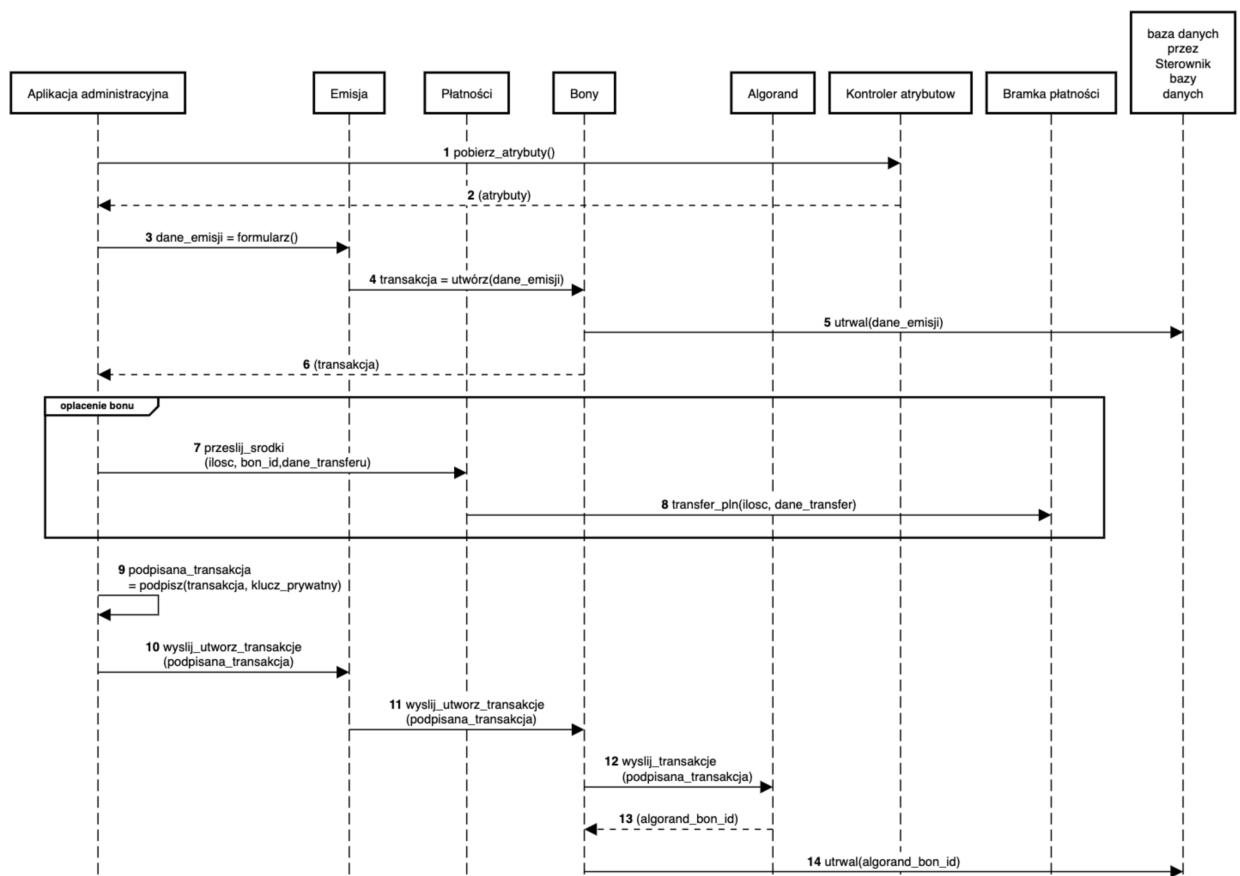
4.3 KOOPERACJA KOMPONENTÓW

Celem lepszego zrozumienia oraz zobrazowania kooperacji poszczególnych komponentów, można zaprezentować ich komunikację za pomocą diagramów sekwencji. Pozwoli to zweryfikować architekturę komponentów oraz posłuży jako cenna wskaźówka przy implementacji systemu. Przedstawione zostaną wyłącznie kluczowe procesy: emisji, nadania roli realizatora, zgłoszenia się do udziału w bonie oraz spieniężenia bonu.

We wszystkich interakcjach z publicznym łańcuchem bloków, nośnikiem danych jest podpisana transakcja, a ponieważ w wymaganiach naszego systemu zdefiniowano, że klucz prywatny musi znajdować się po stronie użytkownika systemu, to każda operacja musi zostać przez niego zauthoryzowana bez trzeciej zaufanej strony, jaką zazwyczaj pełni “system”. Dlatego też niektóre funkcje należy rozdzielić na dwie. Jedna z nich służy do utworzenia transakcji, a druga do wysłania tej transakcji. Z perspektywy architektury komponentów nie stanowi to różnicy, dlatego też ze względu na czytelność podział ten nie został uwzględniony w diagramie komponentów.

4.3.1 EMISJA

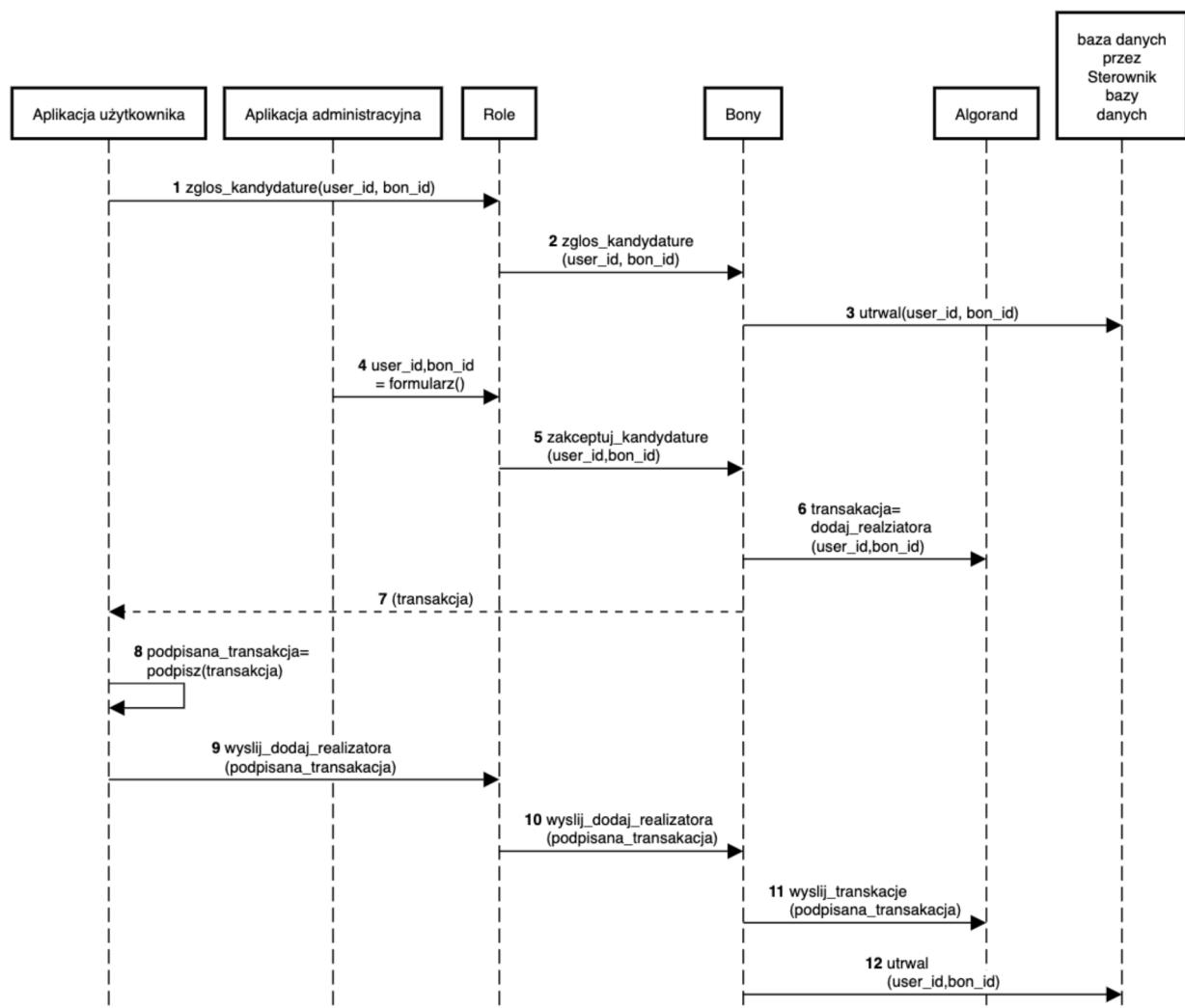
W procesie emisji tworzony jest nowy bon. Inicjatorem jest aktor emitenta, który generuje dany bon. Encja bonu znajduje się w dwóch miejscach. Jednym z nich jest baza danych, w której przechowywane są meta dane na temat bonu oraz uchwyty do kontraktów na publicznym łańcuchu bloków, które odpowiedzialne są za kluczowe procesy bonu. Drugim z nim jest łańcuch bloków, na który przechowywane są kontrakty, balans bonów i role użytkownika. Dane potrzebne do wyemitowania bonu przekazywane są przez emitenta. Tworzona jest encja bonu oraz uiszczany przelew PLN. Następnie generowana jest stosowana transakcja, podpisywana przez emitenta i dołączana do łańcucha bloków.



Rysunek 18: Proces emisji. Źródło: Źródło własne

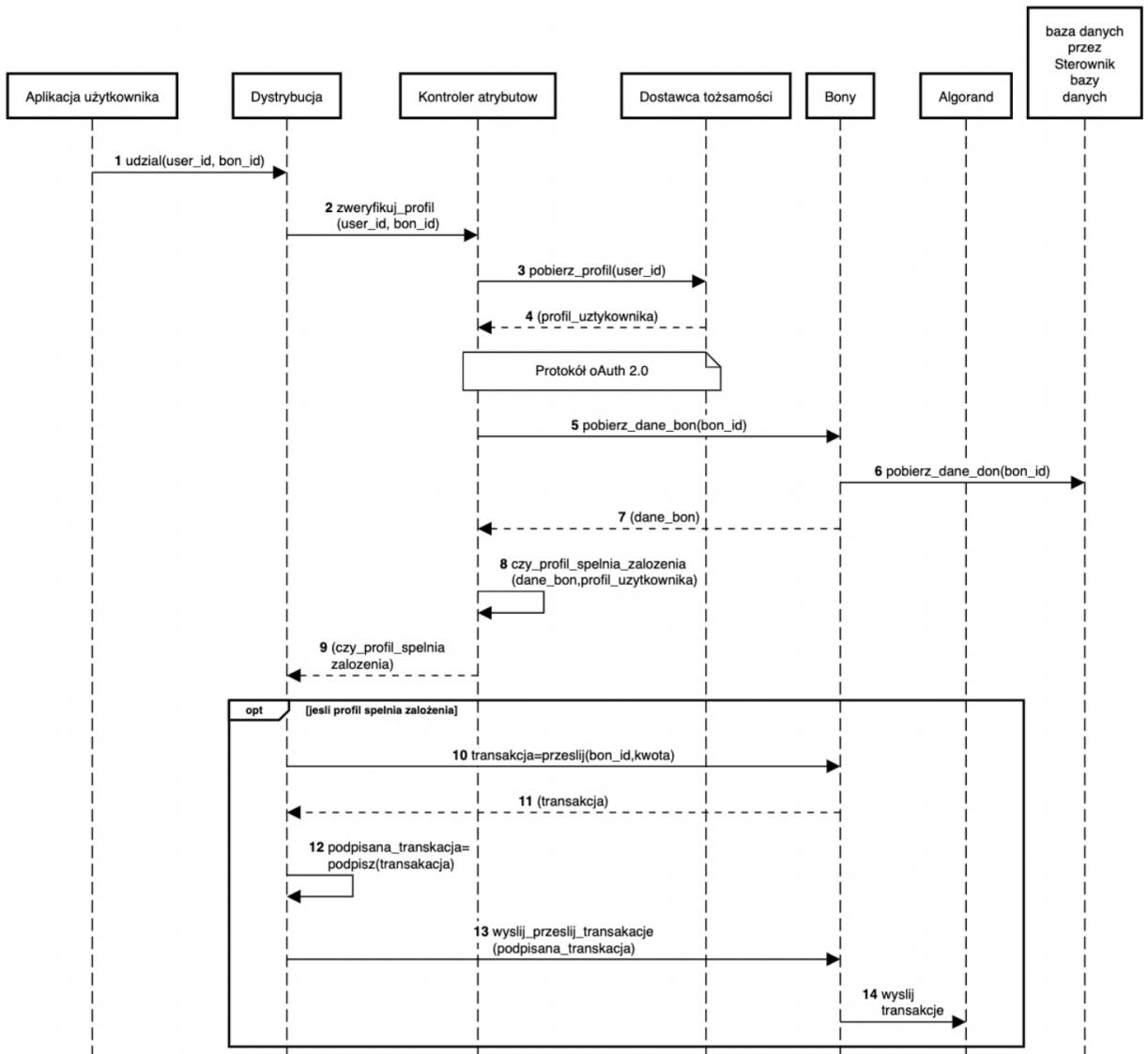
4.3.2 NADANIE ROLI REALIZATORA

Proces nadania użytkownikowi uprawnienia realizatora bonu, polega na wskazaniu przez administratora bonu unikalnego identyfikatora bonu oraz użytkownika. Unikalny identyfikator użytkownika może pojawić się w procesie na dwa sposoby. Zostanie przekazany poza granicami systemu, a następnie ręcznie wpisany przez administratora lub kandydat najpierw zgłosi chęć wzięcia udziału, a następnie może zostać zaakceptowany przez administratora bonu. Następnie zmiana ta zostaje odnotowana w bazie danych oraz rozproszonym rejestrze. Podczas projektowania należy uwzględnić spójność danych pomiędzy bazą danych a publicznym łańcuchem bloków, przy czym za źródło prawdy należy przyjąć publiczny łańcuch bloków, ponieważ to tam odbywa się validacja logiki biznesowej.



Rysunek 19: Proces nadania roli realziatora. Źródło: Źródło własne

4.3.3 UDZIAŁ W DYSTRYBUCJI BONU

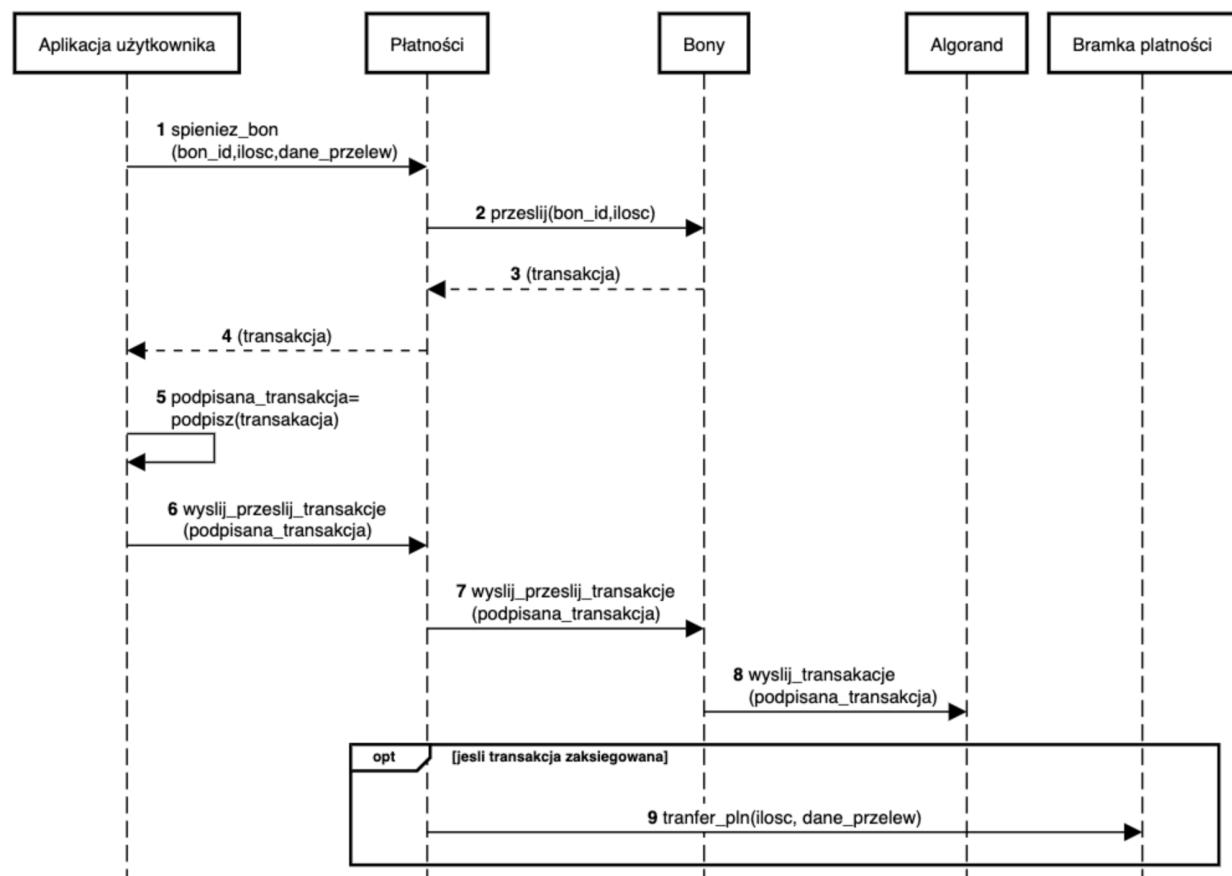


Rysunek 20: Proces dystrybucji bonu. Źródło: Źródło własne

W procesie dystrybucji bonu, użytkownik, który spełnia założenia bonu. Innymi słowy, jego profil dostarczany przez dostawcę tożsamości jest zgodny z atrybutami określonymi podczas emisji bonu, otrzymuje zdefiniowaną wcześniej ilość bonu. Atrybuty, do jakich użytkownik ma możliwość się uwierzytelnienia, są zależne od dostawcy tożsamości. Standardem wykorzystywanym do pobierania tych danych jest OAuth 2.0 [34], pozwala nam on pobrać wskazane dane o użytkowniku z danego serwisu, za pośrednictwem użytkownika końcowego, przez co zyskuje się pewność co do źródła danych, nigdy nie poznaje się poświadczeń użytkownika

oraz pobierane są tylko te zbiór atrybutów, który jest rzeczywiście wymagany. Możliwe jest, aby takie atrybuty pochodziły z wielu serwisów i wynik zdania logicznego na tych atrybutach definiował prawo otrzymania bonu.

4.3.4 SPIENIĘŻENIE BONU



Rysunek 21: Proces spieniężenia bonu. Źródło: Źródło własne

W procesie spieniężania bonu realizator bonu, który w otrzymał bon od użytkownika bonu, może go wymienić na złotówki. W ten sposób dany bon kończy swój cykl, przechodząc od postaci złotówki do postaci cyfrowej na publicznym łańcuchu, a następnie ponownie do postaci złotówki. W tym procesie należy zwrócić uwagę na to, że wysłanie transakcji w kroku o numerze 8 powinno być na adres zerowy, co spowoduje utylizację bonu oraz obydwa kroki 8 i 9 powinny być jedną transakcją, czyli albo obie czynności powinny wykonać się poprawnie albo żadna.

5 IMPLEMENTACJA SYSTEMU OBIEGU TOKENÓW

5.1 STANDARDY I INTERFEJSY KOMUNIKACJI

Rozsądnym podejściem przy wdrażaniu systemów jest wykorzystanie istniejących standardów oraz dobrych praktyk. Dzięki temu można wykorzystywać gotowe przetestowane pomysły, które przeszły już pewien ciąg iteracji doskonalenia. Ponadto w momencie podejmowania decyzji znane są ich realne wady i zalety, co nie jest możliwe w przypadku własnych rozwiązań.

W dobie internetu WWW (ang. World Wide Web), najczęściej wykorzystywany protokołem do komunikacji w relacji klient — serwer jest HTTP 1.1 (ang. Hypertext Transfer Protocol). Protokół określa format żądania oraz odpowiedz. Przykładowe żądanie HTTP, składa się z metody zapytania, adresu zasobu, wersji protokołu, nagłówków oraz opcjonalnie ciała zapytania z danym. Dane mogą być również zakodowane w adresie. Przykładowe żądanie zostało przedstawione na rysunku 22.

```
GET http://localhost:8080/products/search HTTP/1.1
content-type: application/json
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9

{
    "name": "one plus5t",
    "category": "telefon"
}
```

Rysunek 22: Przykładowe zapytanie HTTP. Źródło: Źródło własne

Na odpowiedź serwera składa się kod statusu, który sygnalizuje wynik operacji, ciało odpowiedzi oraz nagłówki.

5.1.1 REST

W jednym żądaniu pojawia się wiele możliwości konfiguracji, dlatego warto posłużyć się doświadczeniem innych programistów, którzy mierzyli się już z takimi

problemami, a swoją wiedzą zawiązały w standardach. Jednym z takich rozwiązań jest standard REST (ang. Representational State Transfer). Definiuje on, w jaki sposób tworzyć bez stanowe interfejsy serwera z wykorzystaniem protokołu HTTP. Zastosowane zasady REST można podsumować w kilku punktach:

- **Separacja warstwy prezentacji oraz logiki biznesowej** Innymi słowy, serwer udostępnia ogólny interfejs, który może być konsumowany przez dowolną aplikację, może być to zarówno aplikacja mobilna, jak i strona WWW. Serwer nie jest w żaden sposób zależny od warstwy prezentacji.
- **Architektura klient serwer** Żądania są wysyłane tylko w jedną stronę od klienta do serwera.
- **Brak stanu** Każde żądanie musi zawierać komplet informacji potrzebny do jego wykonania, nie może polegać na sesyjność połączenia, co jest kompatybilne z zachowaniem protokołu HTTP.
- **Metody żądań**
 - GET — Służy do pobierania informacji. Nie może ona wpływać na stan serwera.
 - POST — Jest używana do tworzenia nowych obiektów lub w ogólniejszym znaczeniu do działań odnoszących jakiś efekt np. transfer bonu.
 - PUT — Służy do aktualizacji danego obiektu. Żądania realizowane w ramach tej metody powinny być idempotentne, co oznacza, że wielokrotne ich wykonanie z tymi samymi parametrami będą zawsze dawały ten sam skutek.
 - DELETE — Oznacza usunięcie danego obiektu.
- **Adresy żądań** Adres można podzielić na dwa części. Pierwszy wskazuje adres hosta, do którego wysyłamy żądanie. Druga część określa komponent, do którego następuje odwołanie, następnie wskazywana jest konkretna funkcja. Przykładowo POST `http://localhost:8989/transfer/send` wskazuje na funkcję transferu bonu[33].

5.1.2 SERIALIZACJA DANYCH

W opisywanym sposobie komunikacji pojawia się zagadnienia formatu przekazywania danych. Ponieważ dane pokonują granicę dwóch systemów, ich format powinien być zrozumiały dla obydwu stron, w tym celu stosuję się serializacją oraz deserializacją danych. Serializacja polega na zmianie obiektów natywnych danego języka do ogólnej postaci rozumianej w wielu systemach np. ciągu znaków. Deserializacja jest natomiast procesem odwrotnym, czyli przekształceniem z postaci ogólnej do natywnej danego języka. Jako format serializacji danych został wykorzystany JSON (ang. Javascript Object Notation). Jest to najczęściej wykorzystywany sposób serializacji, który jest zaimplementowany w licznych językach. Cechuje się on prostotą i jest zrozumiały dla człowieka, co ułatwia proces tworzenia oprogramowania.

5.2 WARSTWA LOGIKI BIZNESOWEJ

5.2.1 WYMAGANIA WZGLEDEM JĘZYKA PROGRAMOWANIA

Kluczową kwestią w implementacji logiki biznesowej, jest dobranie odpowiedniego języka oprogramowania, który w łatwy sposób pozwoli na realizację założeń architektonicznych. Dobierając język, musimy również zwrócić uwagę na dostępność bibliotek, które wykorzystamy do integracji z poszczególnymi komponentami takimi jak węzeł Algorand lub baza danych.

Przy doborze rozwiązania należy skupić uwagę na charakterystyce operacji dla wdrażanego systemu pod względem wydajności. W tym przypadku logika biznesowa spaja ze sobą różne komponenty takie jak baza danych czy węzeł Algorand oddzielając do nich działania, a następnie reaguje na rezultaty. Taka charakterystyka wymaga odpowiedniego środowiska uruchomieniowego zorientowanego na obsługę dużej ilości zapytań wejścia-wyjścia.

5.2.2 WYBRANY JĘZYK PROGRAMOWANIA

Do implementacji systemu wybrano język TypeScript. Jest to nadzbiór języka JavaScript, który rozszerza go o możliwość statycznego typowania. Typo-

wanie statyczne pozwala nam wyłapać część błędów jeszcze na fazie kompilacji, co może znacznie skrócić pętlę zwrotną podczas pisania oprogramowania. Język TypeScript jest bezpośrednio kompilowany do języka JavaScript. Nie posiada oddzielnej maszyny wirtualnej, dzięki czemu jest on jedynie warstwą pośrednią i w żaden sposób nie ogranicza języka JavaScript.

Język JavaScript wraz z HTML jest natywnym językiem przeglądarki, co oznacza, że cała baza kodu zarówno po stronie serwera, jak i przeglądarki będzie mogła być utrzymana w jednym języku.

Krytycznym atutem rozwiązań opartych o język JavaScript po stronie serwera, jest jego środowisko uruchomieniowe node.js. Jest to rozwiązanie zorientowane na procesy wymagające obsługi dużej ilości operacji wejścia-wyjścia. Pomimo że wirtualna maszyna node.js oparta jest o model jednowątkowy, to wykorzystuje tzw. pętle zdarzeń. Program nie oczekuje aktywnie, aż jakąś operację (np. pobrania danych z bazy danych) dojdzie do skutku. Oczekiwanie na rezultat zostaje oddelegowane do jądra systemu (które zazwyczaj jest wielowątkowe). Jądro po otrzymaniu odpowiedzi wraca z tą informacją z powrotem do głównego programu[32]. Taka architektura pozwala w bardzo prosty sposób pracować z asynchronicznymi opercjami, zostało ta zaprezentowane na wycinku kodu 3. Alternatywnym podejściem jest wykorzystanie wielowątkowej maszyny wirtualnej (np. JVM), co jednak niesie za sobą złożoność pisania programów wielowątkowych.

```

1 // synchroniczna operacja
2 const db = db.connection(db_url);
3 // asynchroniczna operacja
4 // zapytanie jest wykonane od razu
5 // i przechodzi do następnej kwerendy (10)
6 db.query("kwerenda_które_zajmuje_2_sekundy")
7     // wykona się dopiero po zakończeniu zapytania
8     .then((result) => console.log('rezultat_1'))
9 // asynchroniczna operacja
10 db.query("kwerenda_która_zajmuje_1_sekunde")

```

```

11 // wykona sie dopiero po zakonczeniu zapytania
12 .then(( result ) => console.log( ' rezultat_2' ))
13
14 // wynik programu na standardowym wyjsciu:
15 // rezultat 2
16 // rezultat 1

```

Kod 3: Programowanie asynchroniczne nodejs

Język JavaScript ma również jeden z największych ekosystemów bibliotek. Dlatego zarówno zawiera on narzędzia do konsumowania większości baz danych jak i oficjalną bibliotekę do Algorand.

5.2.3 SZKIELET DO BUDOWY APLIKACJI

Wiele współczesnych REST serwisów, dzieli podobną strukturę kodu oraz abstrakcje, które rozwiązuje te same problemy. Aby nie powielać tej samej pracy, powstały platformy programistyczne (ang. framework), które zapewniają konfigurowalne szablony programistyczne. Dzięki, którym można zaoszczędzić wiele czasu na pisaniu kodu, który ktoś kiedyś już napisał. Ponadto, często poprzez schematy propagują one dobre praktyki, które wspomagają programistę w wytwarzaniu solidnego oprogramowania.

W przypadku tego rozwiązania została wybrana platforma o nazwie Nest js[31]. Jest to kompleksowe rozwiązanie, które w znaczny sposób ułatwia budowanie modułarnych rozwiązań opartych o architekturę pakowania według komponentów (patrz rozdział 4.1.2). Prócz podstawowych funkcjonalności takich jak tworzenie abstrakcji wokół serwera HTTP, która minimalizuje ilość kodu, jaką trzeba napisać, aby obsługiwać żądanie HTTP. Nest js udostępnia bardzo użyteczny koncept zwany modułami (ang. module). Moduł jest odpowiednikiem komponentu w stosowanej nomenklaturze. Aplikacja jest dokomponowana na moduły, natomiast każdy moduł składa się z czterech części:

- **Dostawcy (ang. providers)** - Jest to zbiór interfejsów, które są implemento-

wane przez ten moduł.

- **Importy (ang. imports)** - Jest to zbiór zewnętrznych modułów, które są wykorzystywane przez ten moduł. Oznacza to, że moduł jest od nich zależny.
- **Kontrolery (ang. controllers)** - Są to końówki udostępnione na zewnątrz przez sieć. Mogą być wywoływanie przez inne podmioty np. warstwę prezentacji.
- **Eksporty (ang. exports)** - Jest to podzbiór dostawców. Znajdują się tu tylko interfejsy brzegowe modułu, które należy udostępnić w ramach tego modułu. Implementacje tych interfejsów będą dostępne w innym module jeśli zostanie zimportowany.

```

1 @Module({
2   imports: [ AttrControlModule , VoucherModule ] ,
3   controllers: [ DistributionController ] ,
4   providers: [ DistributionService , HttpClient ] ,
5   exports: [ DistributionService ]
6 })
7 export default class DistributionModule { }
```

Kod 4: Komponent Dystrybucji w Nestjs

Wycinek kodu 4 stanowi deklaracje modułu odpowiedniego dla komponentu Dystrybucji. Zgodnie z diagramem komponentów, moduł ten jest zależny od Kontrolera atrybutów (AttrControlModule) oraz Bony (VoucherModule), obydwa te moduły w swoich polach exports powinny udostępniać zadeklarowany interfejs. Ponieważ, Dystrybucja może być wywoływana bezpośrednio przez warstwę prezentacji za pomocą żądań HTTP, to posiada ona kontroler, który potrafi przetworzyć takie żądanie. Moduł dostarcza dwa inne interfejsy HttpClient który służy do wysyłania żądań HTTP oraz DistributionService który jest eksportowany i możliwy do użycia przez inne komponenty.

Opisane powyżej moduły stanowią jedynie statyczną konfigurację, która wiąże dany interfejs z jego implementacją w postaci klasy, lecz do działania wymagana jest

instancja tej klasy. Ten obszar również pokryty jest przez Nest.js. Technika ta nazywana jest odwróceniem zależności i polega na zmianie przepływu zależności między dwoma klasami. Jeżeli klasa $A \rightarrow B$ (A zależy od B), to tworząc interfejs IB klasy B , tworzy się granicę architektoniczną, $A \rightarrow BI \leftarrow B$, w której klasy nie zależą już od siebie, lecz od interfejsu. W praktyce jest to realizowane za pomocą techniki nazywanej wstrzykiwaniem zależności (ang. DI). W przypadku Nest.js polega to na tym, że za pomocą adnotacji, oznacza się odpowiednie klasy, a następnie Nest.js automatycznie tworzy instancje tych klas i wstrzykuje je do konstruktora klasy, która ją wykorzystuje. Należy zaznaczyć, że taka realizacja jest jedynie sposobem, efektem jest odwrócenie zależności. Informacja o tym, którą implementację należy wstrzyknąć znajduje się w deklaracji modułu w sekcji providers. Oprócz tego, że zabieg ten pozwala budować architekturę bardziej elastyczną na zmiany, ponieważ ukrywamy szczegóły implementacyjne są ukryte, takie podejście bardzo ułatwia również pisanie testów jednostkowych, gdyż, w bardzo prosty sposób można podmieniać implementacje komponentu na testową.

```

1  @Module({
2      imports: [ AttrControlModule , VoucherModule ] ,
3      controllers: [ DistributionController ] ,
4      providers: [
5          {
6              provide: 'DISTRIBUTION' ,
7              useClass: DistributionService
8          ,
9          HttpClient ],
10         exports: [ DistributionService ]
11     }) export default class DistributionModule { }
12
13 @Controller("distribute")
14 class DistributionController {
15     constructor(
```

```

16     @Inject('DISTRIBUTION')
17     distributionService: DistributionService
18   ) {}
19
20   @Get('opt-in')
21   opt-in(@Body() body): Any {
22     this.distributionService
23       .optIn(body)
24   }
25 }
```

Kod 5: Wstrzyknięcie serwisu Dystrybucji do kontrolera

Na wycinku kodu 5 oprócz wstrzykiwania zależności, pokazano również sposób przekierowania żądań do odpowiedniego kontrolera. Należy wskazać nazwę kontrolera (linia 13), typ metody oraz jej wartość za pomocą odpowiednich adnotacji (linia 20). Taka konstrukcja spowoduje, że wywołanie żądania

GET `http://host:port/distribute/opt-in`, wywoła metodę `opt-in` klasy `DistributionService`. Klasa ta z kolei zostanie skonstruowana i wstrzyknięta przez `Nest.js`.

5.3 WARSTWA DANYCH

5.3.1 WYBÓR BAZY DANYCH

Ostatnią warstwą omawianej architektury stanowi warstwa danych. Jest ona odpowiedzialna za utrwalenie danych oraz ich przeszukiwanie. Przetrzymywane dane mają znaną stabilną strukturę oraz ich pojemność nie przekracza możliwości baz relacyjnych. Dlatego w tym zakresie zdecydowano się na technologię relacyjnych baz danych.

Relacyjne bazy danych jak nazwa wskazuje, opierają się na pojęciu relacji. Relacja definiowana jest jako podzbiór iloczynu kartezjańskiego skończonej liczby zbiorów. W praktycznym podejściu słowo relacji zostało zastąpione słowem tabela, która

tłumaczona jest jako zbiór rekordów o tej samej strukturze[30]. Dzięki zastosowaniu takiego modelu, do przeszukiwania relacyjnych baz danych można użyć algebry relacji oraz rachunku relacyjnego. Praktycznym manifestem tych metod jest język SQL, który w deklaratywny sposób umożliwia wyszukiwanie i wpisywanie informacji do bazy danych.

5.3.2 INTERAKCJA Z BAZĄ DANYCH

Pomimo że język SQL jest standardem w dziedzinie baz danych już od 1986 roku, poszczególne bazy danych różnią się niuansami składni, przez co zapytanie SQL napisane na jedną bazę danych niekoniecznie musi zadziałać na drugiej bazie danych. Z tego powodu oraz ze względu na wygodę programistów powstała technika o nazwie ORM (z ang. mapowanie obiektowo-relacyjne). Nakłada ona abstrakcję na bazę danych, dzięki czemu interakcje z bazą danych można wykonywać za pomocą natywnego języka programowania np. JavaScript, zamiast używać do tego języka SQL[28]. Ponadto rozwiązania takie abstrahują niuanse związane z różnymi implementacjami relacyjnych baz danych, udostępniając jeden spójny interfejs. W ten sposób w fazie rozwoju aplikacji można używać uproszczonej bazy danych np. w pamięci ulotnej, a następnie przejść na rozwiązanie produkcyjne, zmieniając jedynie konfiguracje połączenie do bazy danych. Takie podejście, oczywiście ma również swoje minusy, przy stosowaniu go pojawia się tendencja do przenoszenia operacji związanych z przetwarzaniem danych do logiki biznesowej, która w przeciwieństwie do bazy danych nie jest zoptymalizowana pod ich kątem. Należy również, pamiętać, że jak każda warstwa abstrakcji wnosi ona do oprogramowania pewną złożoność, która komplikuje projekt i zwiększa ilość “poruszających się części”.

Ze świadomością powyższych wad i zalet, do interakcji z bazą danych została wybrana biblioteka TypeOrm[29], która posiada wsparcie dla większości relacyjnych baz danych oraz integrację z Nest.js poprzez dedykowany moduł. Jest to najczęściej wykorzystywana biblioteka ORM w języku TypeScript, posiada wsparcie dla adnotacji, przez co można pisać deklaratywny czytelny kod.

Wybór konkretnej implementacji relacyjnej bazy danych nie był kluczowy, z

uwagi na to, że, podjęcie tej decyzji może zostać odsunięte w czasie, do momentu gdy będzie ona konieczna. Z tego względu na początek zdecydowano na jeden z popularniejszych silników - PostgreSQL[27].

```

1 @Entity()
2 export default class User implements Comparable<User> {
3     @PrimaryGeneratedColumn()
4     public id ?: number;
5     @Column({ unique: true })
6     public email: string;
7     @Exclude()
8     @Column({ select: false })
9     public password: string;
10    @ManyToMany(() => Asa)
11    @JoinTable()
12    public assets: Asa[];
13    @OneToMany(() => Wallet, (wallet: Wallet) => wallet.owner)
14    public wallets: Wallet[];
15    @Column({ default: Role.User })
16    public role: Role;
17    public compare(u: User): boolean {
18        return this.id === u.id;
19    }
20 }
```

Kod 6: Deklarowanie encji użytkownika

W kodzie 6 zaprezentowano stworzenie encji z wykorzystaniem biblioteki TypeOrm. Deklarację encji zaczynamy od adnotacji `@Entity()`. Następnie tworzymy klasy i adnotujemy jej kolejne atrybuty, które będą tworzyły kolumny relacji, do typowania kolumn zostaną użyte natywne typy. Posiadamy też adnotacje typu `@OneToMany`, które pozwalają tworzyć zależności z innymi tabelami, poprzez wskazanie na atrybut

przetrzymujacy klucz obcy.

```

1  @Injectable()
2  export class UserService {
3      constructor(
4          @InjectRepository(User)
5          private userRepository: Repository<User>,
6      ) { }
7
8      async create(userData: CreateUserDto) {
9          const newUser = await this.userRepository
10         .create(userData);
11         await this.userRepository.save(newUser);
12         return newUser;
13     }
14
15     async getById(id: number) {
16         return await this.userRepository
17         .findOneOrFail({ id }, { relations: [ 'wallets' ] });
18     }

```

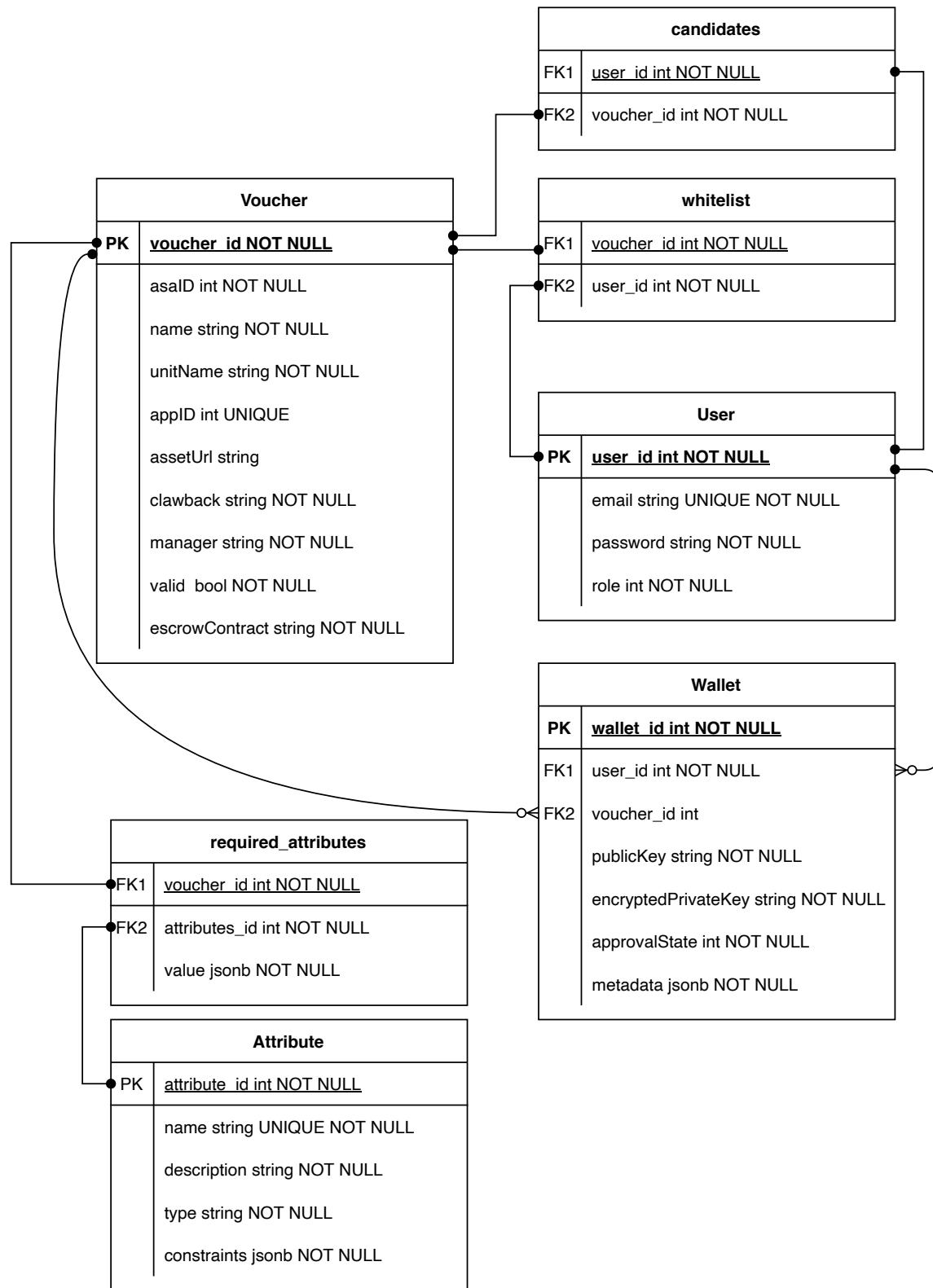
Kod 7: Wstrzykiwanie i używanie repozytorium użytkownika

Do tak utworzonej encji, automatycznie generowane jest repozytorium, które można wstrzyknąć do interesującego nas modułu, a następnie wykonywać operacje na bazie danych. Co zostało zaprezentowane w kodzie 7.

5.3.3 SCHEMAT BAZY DANYCH

Struktura bazy danych musi być znana przed momentem jej użycia, dlatego ważne jest zaprojektowanie schematu bazy danych. Schemat został przedstawiony na rysunku 23.

Schemat bazy danych dla tego projektu jest stosunkowo prosty. Wyróżnia się



Rysunek 23: Schemat bazy danych. Źródło: Źródło własne

cztery tabele

- `User` - Tabela użytkownika, reprezentuje aktorów w systemie rozpoznawanych za pomocą atrybutu `role`. Atrybuty `email` oraz `password` służą do uwierzytelniania użytkownika. Użytkownik posiada związek wiele do wielu z bonem (tabela `whitelist`), określając w ten sposób realizatorów bonów. Jeżeli id użytkownika i id danego bonu znajdują się w tej relacji oznacza to, że użytkownik jest jego realizatorem. Analogicznie jest z tabelą `candidates`, w ramach kandydatów do danego bonu.
- `Wallet` - Tabela portfela, użytkownik może mieć wiele portfeli, przy czym każdy portfel może być przypisany tylko do jednego bonu. Encja ta to para kluczy asymetrycznych, publicznego i prywatnego, przy czym klucz prywatny jest w formie zaszyfrowanej. Atrybut `metaData` przetrzymuje meta dane na temat szyfrowania. `approvalState` informuje o tym czy portfel jest aktywny.
- `Voucher` - Tabela reprezentująca Bon. Jej atrybuty odzwierciedlają bon w sieci Algorand.
- `Attributes` - Tabela ta zawiera listę atrybutów, do jakich potrafi uwierzytelnić system, oraz informację na temat samego atrybutu. Należy wyróżnić, kolumnę `constraints`, w której zawarte są niestrukturyzowane informacje na temat walidacji danego atrybutu w systemie zewnętrznym. Zdecydowano się na taką postać danych, z powodu trudności w utworzeniu generycznego zestawu informacji dla różnorodnych atrybutów. Typ ten to JSON w formacie binarnym.
- `required_attributes` - To tabela zawierająca informację na tematy atrybutów, jakie musi spełniać użytkownik, aby otrzymać bon. Atrybut jest sparametryzowany przez konkretną wartość. Wartość może przybierać różną formę, dlatego wybrano niestrukturyzowany typ.

5.4 SMART KONTRAKTY

Algorand pozwala programistom tworzyć ASA (ang. Algorand Standard Assets), które mogą być cyfrowym odpowiednikiem aktywa, usługi lub w tym przypadku bonu (w dalszej części pojęcia ta będą używane naprzemiennie). Są one czę-

ścią protokołu, można o nich myśleć jak o systemowych smart kontraktach. W domyślnym ustawieniu ASA mogą być wymienialne jak natywna kryptowaluta Algo. Takie rozwiązanie jest akceptowalne w większości przypadków. Jednak na potrzeby tego systemu, aby doszło do przelewu muszą zostać spełnione odpowiednie warunki, określone w kontrakcie:

1. Odbiorca bonu musi być realizatorem bonu. Informacja o fakcie, że jest jego realizatorem, znajduje się w rozproszonym rejestrze lub nadawca bonu musi być jego administratorem.
2. Ważności bonu nie mogła jeszcze upłynąć.

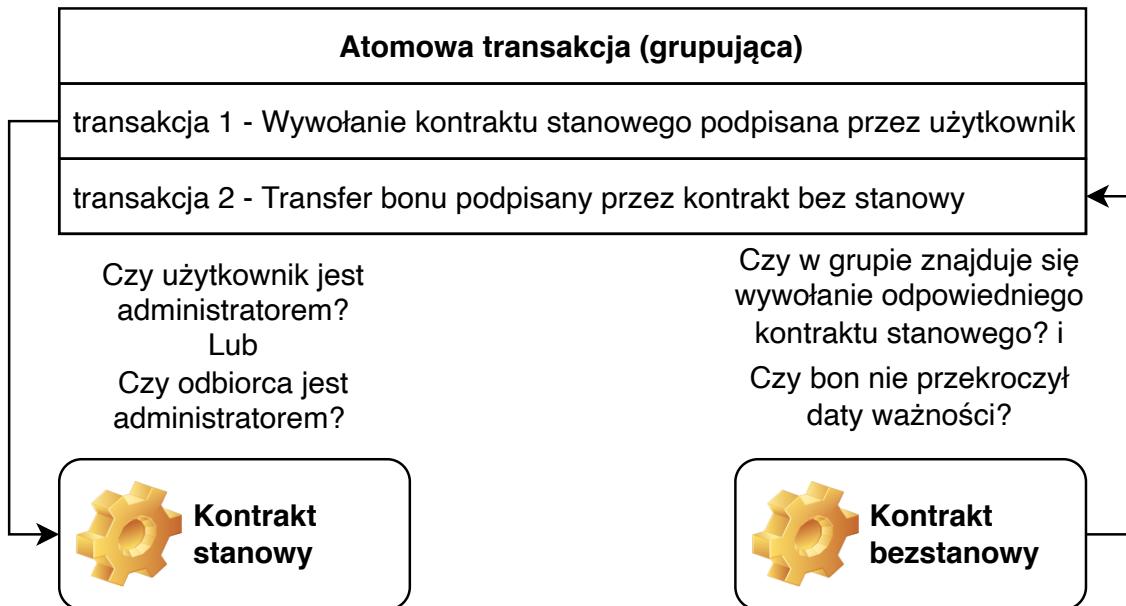
5.4.1 LOGIKA SMART KONTRAKTU

W każdym ASA należy skonfigurować cztery adresy

1. **adres menadżera** - Menadżer jest odpowiedzialny za konfiguracje ASA oraz ma możliwość zmiany pozostałych trzech adresów.
2. **adres rezerwy** - To konto, na którym znajdują się wszystkie nowo utworzone środki.
3. **adres zamrażacza** - Ten adres jest uprawniony do całkowitego zamrożenia wymiany środków lub zamrożenia pojedynczego konta.
4. **adres Clawback** - Adres ten jest upoważniony do transferu ASA z dowolnego konta na inne dowolne konto nawet jeśli ASA jest zamrożone.

Jednym z możliwych rozwiązań, jest stworzenie ASA, które domyślnie jest zamrożone, wówczas tylko adres Clawback może wykonywać przelewy środków. W takim wypadku adresem Clawback powinien być kontrakt bez stanowy, w podrozdziale 3.4.2 taki kontrakt nazwano kontem kontraktowym. Ponieważ, logika wymaga odczytania stanu z rozproszonego rejestru, możemy to wykonać wyłącznie przy użyciu kontraktu stanowego. Aby tego dokonać, należy za pomocą kontraktu bez stanowego wymusić wywołanie kontraktu stanowego (patrz podrozdział 3.4.2). Efekt ten można uzyskać grupując dwie transakcję w jedną. Pierwsza z nich byłaby wywołaniem kontraktu stanowego, która sprawdza, czy odbiorca jest realizatorem bonu lub nadawca administratorem oraz druga transferu ASA, która jest podpisywana przez

kontrakt bez stanowy, w której logice znajduje się sprawdzenie, czy transakcja została zgrupowana z wywołaniem kontraktu stanowego o właściwych parametrach [26]. Pojęcie to zostało zobrazowane na rysunku 24.



Rysunek 24: Koncepcja smart kontraktu. Źródło: Źródło własne

```

1 global GroupSize
2 int 2
3 ==
4 gtxn 0 TypeEnum
5 int appl
6 ==
7 &&
8 gtxn 1 TypeEnum
9 int axfer
10 ==
11 &&
```

Kod 8: Wycinek kodu kontraktu bez stanowego

Kod 8 prezentuje sprawdzenie, czy transakcja znajduje się w grupie transakcji o rozmiarze dwa. Odpowiada również na pytanie, czy pierwsza transakcja jest wywoła-

niem kontraktu bez stanowego a druga transferem ASA. Sprawdzenie to ma charakter pokazowy i jest niekompletne, należałoby dodatkowo sprawdzić adresatów oraz nadawców transakcji, a także argumenty wywołania kontraktu stanowego.

Aby zaimplementować takie rozwiązanie, wymagane są następujące dwie funkcje od stanowego kontraktu:

- Pierwsza z nich wywoływana jest tylko przez administratora kontraktu, pozwala na ustawienie roli specyficzemu użytkownikowi do danego bonu,
- Druga z nich pozwala na odczytanie roli nadawcy i odbiorcy transakcji i sprawdzeniu, czy są one odpowiednie. Czy odbiorca jest realizatorem bonu lub, czy nadawca jest administratorem bonu, który dystrybuuje bon. Funkcja ta może być wywołana przez dowolną osobę.

Taka konstrukcja niesie za sobą następujące implikacje:

1. Para kontraktów powinna zostać utworzona dla każdego nowego bonu,
2. każdy użytkownik powinien mieć tyle kont Algorand w ilu bonach bierze udział,
3. każde z takich kont musi być zasilone minimalną ilością środków pozwalającą na wzięcie udziału w kontrakuie stanowym i ASA
4. domyślnie istnieje tylko jedno konto, które może dystrybuować środki.

Jednym z wymagań funkcjonalnych było przypisanie daty ważności do bonu. W przypadku zdecentralizowanych sieci nie jest to oczywiste zadanie, ponieważ każdy z węzłów może wykonywać transakcję w innym czasie co wynika np. z opóźnień przesyłania danych. Innymi słowy, czas nie jest deterministyczną zmienną w rozproszonym przetwarzaniu transakcji. Zatem należy zastąpić ją substytutem, który będzie deterministyczny. Takim parametrem jest numer bloku. Wszystkie węzły są zgodne co do numeru bloku, w którym procesowana jest dana transakcja. Znając aktualny numer bloku oraz średni czas na wygenerowanie bloku jest możliwe oszacowanie, do którego bloku dany bon powinien być ważny.

δ - średni czas wydobycia bloku w milisekundach (w przypadku Algorand 4350 ms)

b - aktualny numer bloku

t - ważność trafienia bonu w milisekundach

x - szacowany numer bloku końca ważności bonu

$$x = b + \left\lceil \frac{t}{\delta} \right\rceil$$

Jest to jedynie oszacowanie, a więc ważność bonu byłaby obarczona pewnym błędem, zależnym od wahań czasu wydobycia bloku. Alternatywnym rozwiązaniem, mogłoby być udostępnienie funkcji dla administratora bonu, która pozwala na zakończenie bonu. Błąd oszacowania mógłby być w ten sposób zminimalizowany, lecz wymagałoby to zaufania do administratora bonu, że zakończy go w ustalonej chwili.

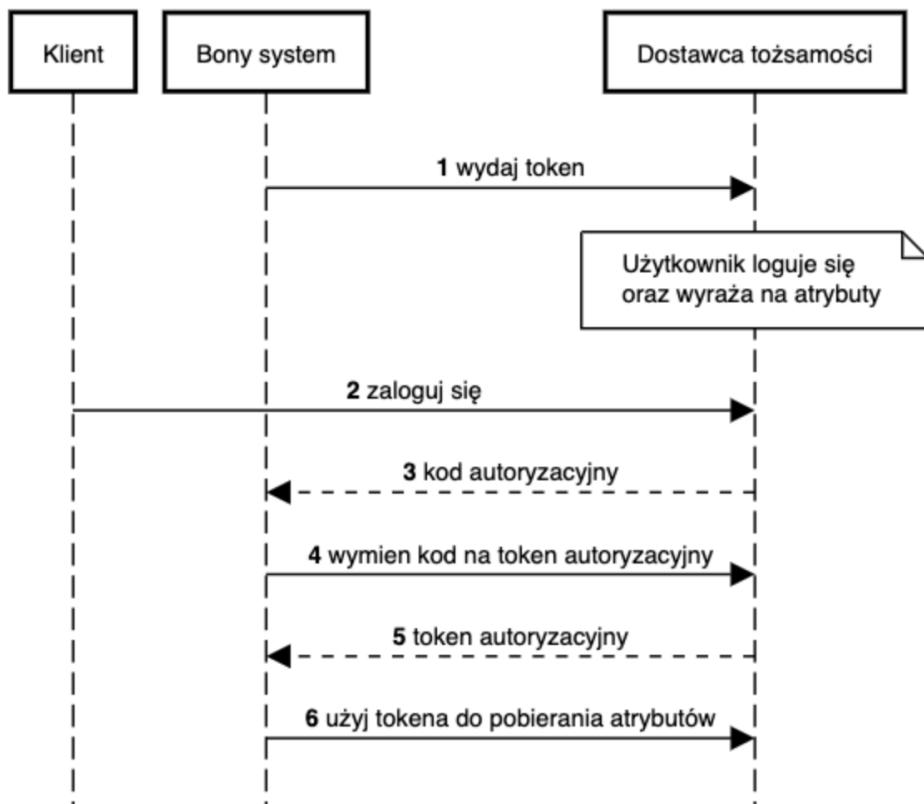
5.5 POBIERANIE ATRYBUTÓW

System integruje się z dostawcą tożsamości w celu pobrania atrybutów o użytkowniku. Integracja jest realizowana z wykorzystaniem protokołu OAuth 2.0.

Protokół OAuth 2.0 stanowi standard branżowy uwierzytelniania. Skupia się na prostocie wdrożenia oraz dokładności specyfikacji, w celu minimalizacji wektora podatności[19].

OAuth jest sposobem przekazywania informacji na temat użytkownika pomiędzy dwoma niezależnymi serwisami, bez użycia jego poświadczeń. Zamiast tego wykorzystywany jest token autoryzacyjny, generowany po stronie serwisu uwierzytelniającego (dostawcy tożsamości). Następnie jest on przekazywany do serwisu żądającego. Serwis na podstawie tokenu przez określony czas może pobierać dane o użytkowniku. Zakres danych, który może zostać pobrany jest zakodowany w tokenie autoryzacyjnym. Użytkownik zgadza się na zakres danych podczas autoryzacji u dostawcy tożsamości[17].

Na rysunku 25 przedstawiono sekwencje czynności potrzebną do uwierzytelnienia atrybutów. Poszczególne komendy są żądaniami HTTP. Wyjątkiem jest krok numer 2, który jest manualnym procesem logowania wykonywanym przez użytkownika. W praktyce użytkownik zostaje przekierowany na stronę dostawcy tożsamości. Po poprawnym zalogowaniu się i udzieleniu zgody na wskazane atrybuty zostaje ponownie przekierowany do pierwszego serwisu, z kodem autoryzacyjnym, na którego podstawie może pobrać token autoryzacyjny.



Rysunek 25: Diagram sekwencji dla oAuth 2.0. Źródło: Źródło własne

5.5.1 WYBRANY DOSTAWCA TOŻSAMOŚCI

Protokół OAuth 2.0 jest implementowany przez największe firmy w branży, między innymi: Amazon, Google, Facebook, Microsoft oraz Twitter. Każda z nich jest potencjalnym kandydatem na dostawcę tożsamości w omawianym systemie. Jednakże z perspektywy systemu kluczowe są atrybuty, do jakich serwis może uwiaryzelić konsumenta bonu. W przypadku wymienionych serwisów gama atrybutów skupia się na tych wygenerowanych wewnętrz serwisów, które z perspektywy biznesowej nie są wysokiej jakości. Z tego względu wybrano projekt naukowo-badawczy, który agreguje wiele atrybutów z kilku źródeł również systemów administracyjnych i bankowych. Projekt nosi nazwę CyberID. *Praktyczną część pracy stanowi opracowanie dwóch kooperujących ze sobą modułów oprogramowania. Pierwszy z nich służyć ma monitorowaniu operacji na danych osobowych, zarządzaniu zgodami profilowanych użytkowników oraz kontrolowaniu dostępu do wspomnianych danych. Do realizacji funkcjonalności modułu wykorzystano platformę blockchain Hyperledger*

Fabric. Natomiast drugi moduł ma za zadanie pozyskiwać dane bankowe z rachunków płatniczych użytkowników od zewnętrznego dostawcy, określonego jako serwis TPP. [16].

Zestaw wybranych atrybutów, do których użytkownik może się uwierzytelnić w ramach tego systemu to kod pocztowy oraz wiek.

5.6 KOMUNIKACJA Z SIECIĄ ALGORAND

Z węzłem Algorand można komunikować się poprzez interfejs REST HTTP. Do autoryzacji wymagany jest token, który powstaje wewnątrz węzła. W przypadku używania testowego węzła token jest odgórnie zdefiniowany. Algorand udostępnia bibliotekę w JavaScript, która ułatwia korzystanie z węzła. Dostępnych jest dużo pomocniczych funkcji kryptograficznych, pozwalających na podpisywanie transakcji, czy generowanie pary kluczy.

5.7 BRAMKA PŁATNOŚCI

Bramka płatności została zaimplementowana jedynie w postaci zaślepki, a więc nie doszło do integracji z prawdziwym systemem. Ma ona jedynie na celu zobrazować ideę rozwiązania. Integracje nawet z testowymi bramkami płatności wymagają odpowiednich zaświadczeń prawnych. Docelowo system mógłby być wkomponowany w system bankowy, który udostępnia gotowe usługi do realizacji przelewów.

5.8 INTERFEJS UŻYTKOWNIKA

Interfejs użytkownika został podzielony na dwie oddzielne aplikacje. Pierwsza aplikacja ma służyć administratorowi bonu oraz emitentowi, pozwalając na jego tworzenie oraz zarządzanie. Druga aplikacja jest przeznaczona dla realizatora bonu oraz użytkownika bonu w celu zgłaszania udziału w bonie, oraz transferu bonu.

Interfejs użytkownika został zaimplementowany w postaci strony WWW. Strona została przystosowana zarówno do urządzeń mobilnych jak i klasycznych desktop. Do implementacji wybrano strony internetowe, ponieważ są one proste w użyciu dla użytkownika.

kownika końcowego, wystarczy wpisać adres strony, aby zacząć interakcje z systemem. Dodatkowo ważnym aspektem jest niezależność rozwiązania od platformy sprzętowej. Stronę można otworzyć na każdej platformie, na której dostępna jest przeglądarka WWW. Oprócz tego w celu ułatwienia ponownego otwierania aplikacji do aplikacji został przypisany manifest (jest to plik specjalnego znaczenia), który zawiera metadane na temat aplikacji, przez co pozwala ją zainstalować na używanym urządzeniu. Po otworzeniu tak zainstalowanej aplikacji wyświetla się ona bez komponentów graficznych przeglądarki, przez co aplikacja przypomina natywną.

5.8.1 APLIKACJA JEDNOSTRONNICOWA

Aplikacja została zrealizowana według wzorca aplikacji jednostronicowej (z ang. single page application). W standardowym podejściu na każdą odpowiedź serwera zwracana jest nowa strona WWW, która następnie jest renderowana przez przeglądarkę. W efekcie użytkownik musi oczekiwać na przeładowanie strony. W celu optymalizacji responsywności stron WWW zaproponowano nowe podejście, w którym cała strona ładowana jest przy pierwszym żądaniu (stąd nazwa jednostronicowa), a następnie jest dynamicznie modyfikowana w odpowiedzi na interakcje użytkownika, przy czym nie każda interakcja użytkownika musi wiązać się z komunikacją z serwerem. Przy obaniu takiej techniki powstaje naturalna granica architektoniczna między warstwą prezentacji a warstwą logiki biznesowej, zgodna z zasadami REST.

5.8.2 WYBRANE TECHNOLOGIE

Aplikacje jednostronicowe, przechodzą z modelu lekkiego klienta, który odpowiadzialny jest tylko i wyłącznie za renderowanie gotowych widoków HTML, do modelu, w którym klient (przeglądarka WWW) odpowiadzialny jest za przetwarzanie danych otrzymywania z serwera i na ich podstawie generowania odpowiednich zmian w widokach. Ze względu na rosnącą złożonością po stronie kodu przeglądarki warto skorzystać z gotowych sprawdzonych rozwiązań, które ułatwią to zadanie.

Jednym z popularniejszych rozwiązań jest biblioteka ReactJS. Pozwala ona na pisanie w sposób deklaratywny, dzięki czemu kod staje się bardziej przejrzysty i prost-

szy do rozumowania. Interfejs użytkownika jest tworzony poprzez łączenie odizolowanych komponentów, które zarządzają własnym stanem[22].

Komponenty w praktycznym sensie to klasy, które implementują metodę `render`, która zwraca widok w postaci kodu HTML. Została opracowana specjalna składnia o nazwie JSX, która jest rozszerzeniem języka JavaScript i pozwala na pisanie widoków w tym języku, które następnie automatycznie przetwarzane są na HTML. Każdy z komponentów składa się z dwóch źródeł danych, są to atrybuty przekazywane od komponentów rodziców oraz stan wewnętrzny komponentu. Każda zmiana stanu powoduje ponowne renderowanie komponentu oraz wszystkich komponentów pochodnych, które są zależne od tego stanu. W przeciwieństwie do atrybutów, które są niemodyfikowalne.

Dzięki izolacji stanu komponentów oraz jednokierunkowemu przepływowi danych, naturalnym stało się tworzenie reużywalnych komponentów. Powstały biblioteki o otwartym kodzie źródłowym, udostępniające gotowe komponenty, z których można budować interfejsy użytkownika. Jedną z takich bibliotek jest Material UI[23], która implementuje standard interfejsu użytkownika Material Design, co stanowiło wymaganie WU01, dlatego też została wybrana w tym projekcie.

5.8.3 PORTFEL

Autoryzacja transakcji w rozproszonych rejestrach oparta jest na kryptografii klucza publicznego (patrz podrozdział 1.4.4). Oznacza to, że wszelkie prawa dostępu do konta związane są z tym kluczem prywatnym. Plusem takiego rozwiązania jest fakt, że dopóki klucz nie zostanie ujawniony, zyskuje się pewność, że nikt nie wykona operacji w imieniu tego konta. Ten plus jest również minusem, ponieważ, to na właściwym kluczu pozostaje odpowiedzialność za jego bezpieczeństwo. W centralnych systemach, podmiot ma zazwyczaj władzę absolutną i może podać się za każdego użytkownika systemu, natomiast niweluje ryzyko po stronie użytkownika, pozwalając mu pamiętać wymyślone przez niego hasło oraz obsługuje procedurę jego resetowania.

Aby wyciągnąć najlepsze cechy z tych dwóch światów zastosowano następujące

podejście. Para kluczy jest generowana po stronie klienta, następnie jest szyfrowana przez bibliotekę kryptograficzną po stronie klienta na wskazane przez niego hasło. W takim formacie jest zapisywana po stronie serwera. W ten sposób użytkownik nie może zgubić klucza prywatnego, ponieważ jest zapisany po stronie serwera. Z kolei serwer nie zna klucza prywatnego, dopóki nie złamie hasła użytkownika, które go zabezpiecza. Następnie przy podpisywaniu transakcji użytkownik jest każdorazowo proszony o podanie hasła, a więc klucz prywatny i hasło w postaci jawniej przechowywane są jedynie w pamięci ulotnej, po stronie przeglądarki. Istnieje jeszcze możliwość zgubienia hasła przez użytkownika, da się jednak stworzyć procedurę, w której to menadżer ASA po utracie hasła przez użytkownika będzie mógł przenieść środki na wskazane przez niego nowe konto.

Od strony interfejsu użytkownika została wykorzystana biblioteka `key-store`. Biblioteka jest zgodna ze specyfikacją PKCS v2.0[25, sekcja 5.2]. Wykorzystuje algorytm PBKDF2 do obniżania skuteczności ataków siłowych na hasło użytkownika, poprzez wielokrotne wykonanie funkcji skrótu na haśle zwiększając tym samym ilością cykli procesora potrzebną na sprawdzenie jednego hasła. W ten deterministyczny sposób otrzymywany jest klucz symetryczny z hasła użytkownika, który następnie wykorzystany jest przez algorytm `xsalsa20-poly1305` do zaszyfrowania klucza prywatnego[24].

6 WNIOSKI Z REALIZACJI PRACY DYPLOMOWEJ

6.1 OMÓWIENIE WYKORZYSTANYCH METOD ANONIMIZACJI UŻYTKOWNIKÓW

Omawiając metody anonimizacji danych należy rozróżnić dwa z pozoru podobne pojęcia anonimizacji danych oraz pseudoanomizacji. Pseudonanimzacja to proces przetworzenia danych w taki sposób, aby nie umożliwić ich powiązanie z daną osobą. Pseudoanonimzacja jest procesem odwracalnym. Dodatkowe informacje, które pozwalają na odwrócenie go, powinny być przechowywane osobno od danych i być objęte środkami technicznymi i organizacyjnymi uniemożliwiającymi ich pozyskanie nieuprawnionym osobom[21]. Anonimizacja różni się od poprzedniej definicji tym, że dane są przetworzone nieodwracalnie. Nie istnieje procedura ani dodatkowe informacje, które pozwolą na ponowne powiązanie danych z osobą[20].

W przypadku publicznych łańcuchów bloków klucze publiczne użytkownika są generowane losowo. Informacje, które pozwalają je powiązać z danym użytkownikiem, znajdują się w bazie danych. Jest to alternatywnym miejscu do publicznego łańcucha bloków, a więc spełnia definicję danych pseudoanonimowych. Takie podejście spełnia wymaganie WF11.

Danymi osobowymi są również atrybuty, do jakich może uwierzytelnić się użytkownik w ramach uczestnictwa w bonie. Atrybuty, nie są zapisywane po stronie serwera, jednakże są przetwarzane. Atrybutami może być wiek czy miejsce zamieszkania, a nawet dane z urzędu skarbowego np. dochód na dany rok. Aby uniknąć ich przetwarzania, można zastosować dwa alternatywne podejścia, obydwa polegające na oddelegowaniu sprawdzania atrybutów do serwisów, które je zapewniają.

W pierwszym wariantie zdanie logiczne agregujące atrybuty wysyłane jest do dostawcy tożsamości, który następnie zwraca odpowiedź z informacją czy dany użytkownik spełnia zdanie logiczne. Wymaga to implementacji dodatkowego modułu w oprogramowaniu dostawcy tożsamości.

W drugim wariantie administrator bonu jest odpowiedzialny za wygenerowanie listy unikalnych identyfikatorów, którym należy przyznać bon oraz przekazanie jej

do systemu. System musi również mieć możliwość uwierzytelnienia do takiego identyfikatora. Przedstawiając na przykładzie, takim identyfikatorem mógłby być spesudoadminizowany numer pesel. Listę stanowiły wszyscy mieszkańcy danej gminy. Zaś bonem byłoby darmowe wejście na basen. Jeżeli przetworzony numer pesel użytkownika znajdowałby się na liście, po poprawnym uwierzytelnieniu użytkownik otrzymałby bon.

W kompleksowym rozwiązaniu istnieje możliwość wykorzystania trzech sposobów, a następnie odpowiednio dobierane metody w zależności od charakterystyki atrybutów. Powyższe rozważania są tylko przemyśleniami i nie zostały zaimplementowane w projekcie.

6.2 ANALIZA WYDAJNOŚCI SYSTEMU

W omawianym systemie zostały określone trzy wymagania co do jego wydajności

1. **WW01** - Odnoszące się do wydajności procesu utworzenie bonu
2. **WW02** - Odnosząca się do wydajności procesu dystrybucji bonu
3. **WW03** - Odnosząca się do wydajności procesu transfer bonu

Wymagania te skupiają się wokół najważniejszych funkcji systemu, w których system wchodzi w interakcje z publicznym łańcuchem bloków.

6.2.1 ŚRODOWISKO TESTOWE

Testy zostały przeprowadzone na pojedynczej maszynie wirtualnej. Uruchomione zostały trzy serwisy: baza danych, główny serwer oraz testowy węzeł Algorand. Testowy węzeł Algorand działa z czasem finalizacji transakcji występującej na głównej sieci publicznego łańcucha. Ponieważ wszystkie urządzenia znajdują się na tym samym komputerze, testy nie uwzględniają opóźnień wynikających z przesyłu danych przez sieć. W testach został również pominięty interfejs użytkownika, skupiono się na wydajności logiki biznesowej. Użyta maszyna posiadała 8GB pamięci RAM oraz procesor — Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz z dwoma rdzeniami.

6.2.2 UTWORZENIE BONU

W celu przetestowania wydajności tworzenia bonu zostało uruchomione “jednocześnie” 10 żądań utworzenia bonu. Wykonano 10 takich testów, w których mediana wynosiła 57.25 sekund, a średnia 56.63 sekund. Z każdą kolejną transakcją czas utworzenia bonu zwiększał się nieznacznie o około 0.1%. Rezultat ten świadczy o spełnieniu wymagania WW01.

6.2.3 DYSTRYBUCJA BONU

W testach dystrybucji bonu zostało uruchomione “jednocześnie” 25 żądań dystrybucji bonu. Proces ten zaczyna się od stworzeniu pary kluczy do konta Algorand, a kończy na posiadaniu bonów przez użytkownika. Wykonano 10 takich testów, w których mediana wynosiła 35 sekund, a średnia 36 sekund. Oznacza to spełnienie wymagania WW02.

6.2.4 TRANSFER BONU

W ostatnim teście przetestowanie żądanie transferu bonu, które powinno być najszybsze, ponieważ, najistotniej wpływa na doświadczenie użytkownika końcowego. Jako że operacja ta będzie wykonywana najczęściej, w teście uruchomiono 100 żądań przesłania transakcji jednocześnie. Przeprowadzono łącznie 10 takich testów, w których mediana wynosiła 11 sekund, a średnia 11.72 sekund. Rezultat ten jest zadowalający, ponieważ oznacza spełnienie wymagania WW03.

6.2.5 PODSUMOWANIE

Z analizy wynika, że najwięcej czasu zabierają operacje związane z włączaniem transakcji do bloku, na czas ten nie da się wpływać, ponieważ jest on zależny od konstrukcji danego łańcucha bloków. Możliwą optymalizacją byłoby grupowanie transakcji wtedy kiedy jest to możliwe, przez co kilka transakcji mogłyby być włączane w ramach jednej.

Ponieważ wszystkie serwisy są bez stanowe, można je skalować poprzez ich

replikacje, również węzły Algorand. Wyjątek stanowi baza danych, która używa innych technik skalowania, jak grupowanie w klastry lub dodawanie warstwy pamięci podręcznej cechującej się krótszy czasem dostępu do danych.

6.3 ANALIZA KOSZTÓW DZIAŁANIA SYSTEMU

Omawiając koszty działania systemu skupiono uwagę wyłącznie na kosztach związanych z wykorzystaniem publicznego łańcucha Algorand. Obecnie nie ma żadnych narzędzi, które mogłyby nam pomóc w wyliczeniu kosztów. Proces ten więc został wykonany manualnie na podstawie dokumentacji Algorand.

Opłaty możemy podzielić na dwie kategorie, opłat transakcyjnych, które pobierane są na rzecz validatorów w sieci za wykonanie konkretnego działania oraz środki, które muszą być ulokowane na danym koncie (zwane dalej blokadą), aby mogło podjąć określone działania. Na pierwszą opłatę (zwaną dalej kosztem) nie mamy wpływu i musimy ją traktować jako koszty stałe. W przypadku blokowania środków możliwe są do wprowadzenia mechanizmy, które pozwoliłyby na ponowne użycie tych środków, na przykład poprzez ponowne użycia konta, w którym bon już wygasł.

Tabela 7: Koszt utworzenia bonu. Źródło: Źródło własne

Operacja	Koszt [microAlgos]	Blokada [microAlgos]
Utworzenie ASA	1 000	100 000
Utworzenie kontraktu stanowego	1 000	184 000
Aktywowanie kontraktu bez stanowego	2 000	0
Aktualizacja adresu clawback ASA	1 000	0
Nadanie roli administratora bonu	1 000	0
Suma	6 000	284 000
W PLN (1 Algos = 6 PLN)	0,04 PLN	1,70 PLN

W tabeli numer 7 wskazano koszty związane z utworzeniem bonu. Po przeliczeniu, na złotówki koszty te są niskie nawet jeśli uwzględnimy blokadę jako koszt. Należy jednak pamiętać, że przelicznik Algos na PLN jest zmienny i zależy od tego

jak zostanie wyceniony przez rynek. Od powstania sieci Algorand jego kurs wahał się od 0.4 PLN do nawet 15 PLN.

Tabela 8: Koszt dystrybucji bonu. Źródło: Źródło własne

Operacja	Koszt [microAlgos])	Blokada [microAlgos]
Wzięcie udziału w ASA	1 000	100 000
Wzięcie udziału w kontrakcie stanowym	1 000	184 000
Zasilenie konta ASA	1 000	0
Zasilenie konta minimalnymi środkami	1 000	1 000
Suma	4 000	284 000
W PLN (1 Algos = 6 PLN)	0,02 PLN	1,70 PLN

W tabeli numer 8 wyróżniono koszty dystrybucji bonu. Blokada wynosi tyle samo co w przypadku utworzenia bonu, blokada mogłaby, by być ponownie użyta w przypadku wygaśnięciu danego bonu. Należałoby, by wtedy uiścić dwie transakcje, jedną odpowiadającą za rezygnację z udziału w ASA, drugą za rezygnację udziału w kontrakterze stanowym. Transakcje możnaby zgrupować, a więc ich koszt wynosiłby 1000 microAlgos.

Koszt i blokada mianowania realizatora bonu wynosi tyle samo co w przypadku dystrybucji bonu pomniejszona jedynie o transakcje zasilenia bonem w wysokości 1000 microAlgos.

Koszt transferu bonu jest równy kosztowi transferu pojedynczej transakcji, czyli 1 000 microAlgos co w przeliczeniu stanowi 0,000006 PLN. Dla porównania opłata z użyciem karty płatniczej wynosi minimalnie ok. 0.2% od kwoty transakcji, co oznacza, że opłata przy kwocie 50 PLN będzie ok. 16666 większa niż przy użyciu sieci Algorand.

ZAKOŃCZENIE

Celem pracy było stworzenie systemu wykorzystującego łańcuch bloków do implementacji cyfrowego bonu. Kompletność rozwiązania została uwarunkowana 16 wymaganiem funkcjonalnymi oraz 10 niefunkcjonalnymi (określonymi w rozdziale 2.3), wszystkie z wymagań zostały zrealizowane. Alternatywne podejście do implementacji systemu, pozwoliło osiągnąć następujące korzyści:

- Utworzono ogólnodostępny interfejs wymiany bonu. Przez co rozwiązanie może być wykorzystywane przez różne podmioty.
- Ograniczono stopień zaufania względem centralnego systemu, poprzez przeniesienie głównej logiki biznesowej do publicznego łańcucha bloków.
- Osiągnięto audytowalność systemu, dzięki prowadzeniu nienaruszalnego rejestru operacji transakcji bonu,
- Uzyskano wysoką niezawodność (tożsamą z wykorzystanym łańcuchem bloków). Wszystkie kluczowe procesy biznesowe mogą być realizowane z wykorzystaniem alternatywnego systemu.

System do osiągnięcia pełnej funkcjonalności należałoby zintegrować z bramką płatności, która pozwoliłaby na konwersje bonu do postaci waluty PLN oraz na zabezpieczenie emisji bonu. Z wymagań niefunkcjonalnych należałoby uwzględnić bezpieczeństwo przechowywania administracyjnych kluczy prywatnych oraz szablonów smart kontraktów. Ponadto warto uwzględnić uwagi związane z optymalizacją kosztów łańcuchów bloków (określone w rozdziale 6.3) oraz stworzyć procesy odzyskiwania bonów z niedostępnych kont z powodu utraty hasła.

Kluczowym biznesowym aspektem, systemu jest lista atrybutów, do jakiej można uwierzytelnić użytkownika. Definiuje ona spektrum zastosowań bonu — ilość możliwych zastosowań bonu jest skorelowana dodatnio z ilością dostępnych atrybutów. Rozszerzenie zbioru atrybutów, poprzez integrację kolejnych dostawców tożsamości, stanowi główny nurt dalszego rozwoju systemu.

Programowalny obieg pieniądza, stanowi kolejny wielki krok w rozwoju świata finansów. Teza ta znalazła potwierdzenie na hackatonie[57], gdzie omawiany system

został wyróżniony drugim miejscem. Ponadto jeden z patronów (PKO BP) dostrzegł potencjał rozwiązania, czego skutkiem są prace badawcze prowadzone wewnątrz organizacji.

BIBLIOGRAFIA

- [1] Jonathan Katz, Yehuda Lindell, Introduction to Modern Cryptography Second Edition, CRC Press, 2014
- [2] Andreas M, Antonopoulos, Bitcoin dla zaawansowanych: programowanie z użyciem otwartego łańcucha bloków, Wydawnictwo HELION, 2017
- [3] Sachin S. Shetty, Charles A. Kamhoua, Laurent L. Njilla, Blockchain i bezpieczeństwo systemów rozproszonych, Wydawnictwo PWN, Warszawa 2020
- [4] Daniel Drescher, Blockchain: podstawy technologii łańcucha bloków w 25 krokach, Wydawnictwo HELION, 2019
- [5] Andreas M. Antonopoulos, Gavin Wood, Ethereum dla zaawansowanych: tworzenie inteligentnych kontraktów i aplikacji zdecentralizowanych, Wydawnictwo HELION, 2019
- [6] Robert C. Martin, Czysta architektura: Struktura i design oprogramowania. Przewodnik dla profesjonalistów, Wydawnictwo HELION, 2018
- [7] Christopher Bennage, Mike Wasson, Masashi Narumoto and the Microsoft Patterns and Practices team, Cloud Application Architecture Guide, Microsoft Press, 2017
- [8] <https://academy.binance.com/pl/articles/byzantine-fault-tolerance-explained>
[10.07.2021]
- [9] Włodzimierz Mosorow ,System rozproszone <http://mosorow.kis.p.lodz.pl/pl/sr/sr.pdf> [10.07.2021]
- [10] Satoshi Nakamoto, Bitcoin Biała księga, <https://bitcoin.org/bitcoin.pdf>
- [11] Gidon Katten, Issuing Green Bonds on the Algorand Blockchain
<https://arxiv.org/pdf/2108.10344.pdf> [12.07.2021]
- [12] Kod bajtowy <https://pl.wikipedia.org/wiki/KodBajtowy> [12.07.2021]
- [13] Peer-to-peer-networks, <https://academy.binance.com/pl/articles/peer-to-peer-networks-explained> [13.07.2021]
- [14] Alin Tomescu, What is A Merkle Tree? <https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/> [16.07.2021]

- [15] Kompletność Turinga, https://pl.wikipedia.org/wiki/Kompletno%C5%9B%C4%87_Turinga [16.07.2021]
- [16] Adrian Kotowski, Modelowanie cyfrowego profilu użytkownika z wykorzystaniem technologii blockchain, <https://repo.pw.edu.pl/info/master/WUT45aaf69ace3c4b7f949c8a7c8808ca8c/> [02.01.2022]
- [17] OAuth 2.0, https://pl.wikipedia.org/wiki/OAuth_2.0 [28.08.2021]
- [18] Rohin Shah, Sanjam Garg, Lecture 1: One-Way Functions, <https://people.eecs.berkeley.edu/~sanjamg/classes/cs276-fall14/scribe/lec02.pdf>
- [19] OAuth 2.0, <https://oauth.net/2/> [27.07.2021]
- [20] Marcin Kujawa, Pseudonimizacja danych – co to właściwie jest, <https://odo24.pl/blog-post.pseudonimizacja-anonimizacja-danych-wyjasniamy-pojecie> [18.07.2021]
- [21] Pseudonimizacja, <https://pl.wikipedia.org/wiki/Pseudonimizacja> [11.07.2021]
- [22] ReactJs, <https://pl.reactjs.org> [11.07.2021]
- [23] Material UI, <https://v4.mui.com> [11.07.2021]
- [24] Key Store, <https://www.npmjs.com/package/key-store> [02.08.2021]
- [25] RFC2898, <https://datatracker.ietf.org/doc/html/rfc2898#section-5.2> [29.07.2021]
- [26] Jason Weathersby , Assets and custom transfer logic, <https://developer.algorand.org/solutions/assets-and-custom-transfer-logic/> [15.08.2021]
- [27] PostgreSql, <https://www.postgresql.org> [09.08.2021]
- [28] https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping [10.08.2021]
- [29] TypeOrm, <https://typeorm.io/#/> [10.08.2021]
- [30] Model relacyjny, https://pl.wikipedia.org/wiki/Model_relacyjny [09.08.2021]
- [31] NestJs, <https://nestjs.com> [01.08.2021]
- [32] The Node.js Event Loop, <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>
- [33] Mateusz Michalski, Czym jest REST? <https://zaprogramujzycie.pl/czym-jest-rest/>

- [34] oAuth 2.0, https://pl.wikipedia.org/wiki/OAuth_2.0 [28.08.2021]
- [35] Styl architektury mikrousług, <https://docs.microsoft.com/pl-pl/azure/architecture/guide/architecture-styles/microservices> [20.07.2021]
- [36] Architektura oprogramowania https://pl.wikipedia.org/wiki/Architektura_oPROGRAMOWANIA [20.07.2021]
- [37] SmartSig details <https://developer.algorand.org/docs/get-details/dapps/smart-contracts/smartsigs/> [22.07.2021]
- [38] Smart contract details, <https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/> [22.07.2021]
- [39] Maszyna stosowa, https://pl.wikipedia.org/wiki/Maszyna_stosowa [27.08.2021]
- [40] Sergey Gorbunov, <https://medium.com/algorand/algorand-releases-first-open-source-code-of-verifiable-random-function-93c2960abd61> [28.08.2021]
- [41] go-ethereum, <https://github.com/ethereum/go-ethereum> [28.08.2021]
- [42] Validator Overview, <https://docs.binance.org/smart-chain/validator/overview.html> [19.08.2021]
- [43] Dowód stawki, <https://academy.binance.com/pl/articles/proof-of-stake-explained>
- [44] Delegated Proof Of Stake, <https://academy.binance.com/pl/articles/delegated-proof-of-stake-explained> [22.08.2021]
- [45] Similarities and Differences — DPoS and PoA, <https://medium.com/metadium/similarities-and-differences-dpos-and-poa-eb03bc23dde8> [23.08.2021]
- [46] <https://etherscan.io/nodetracker> [12.08.2021]
- [47] Cosimo Bassi, Sustainable Blockchain: Estimating the Carbon Footprint of Algorand's Pure Proof-of-Stake, <https://www.algorand.com/resources/blog/sustainable-blockchain-calculating-the-carbon-footprint> [17.08.2021]
- [48] Ranking domowych urządzeń pobierających najwięcej prądu <https://enerad.pl/aktualnosci/ranking-urzadzen-pobierajacych-najwiecej-pradu/> [17.08.2021]
- [49] <https://www.binance.org/en/smartChain> [25.08.2021]
- [50] <https://trufflesuite.com/ganache/> [12.08.2021]

- [51] <https://www.algorand.com> [21.08.2021]
- [52] <https://ethereum.org/en/> [05.08.2021]
- [53] <https://epuap.gov.pl/wps/portal> [03.07.2021]
- [54] Metoda FURPS, czyli 29 rzeczy do przemyślenia w każdym projekcie IT, <https://www.analizait.pl/2014/metoda-furps-czyli-29-rzeczy-do-przemyslenia-w-kazdym-projektie-it/> [02.07.2021]
- [55] RFC 8018, <https://datatracker.ietf.org/doc/html/rfc8018> [03.07.2021]
- [56] Material io, <https://material.io/design> [04.07.2021]
- [57] Hackaton: <https://hack4lem.com> [04.01.2022]