

**WYDZIAŁ ELEKTRONIKI
POLITECHNIKI WROCŁAWSKIEJ**

**ZASTOSOWANIE MECHANIZMÓW KRYPTOGRAFICZNYCH
PRZY PROJEKTOWANIU
WIELOPLATFORMOWYCH APLIKACJI ROZPROSZONYCH**

INŻ. MICHAŁ LANGE

Praca magisterska napisana
pod kierunkiem
dr inż. Roberta Wójcika

WROCŁAW 2016

Spis treści

Spis rysunków	2
1 Wstęp	3
1.1 Cel pracy	5
1.2 Zakres pracy	5
2 Analiza wymagań i założenia projektowe	6
2.1 Koncepcja działania systemu	6
2.1.1 Komponenty systemu	7
2.2 Wymagania funkcjonalne	8
2.2.1 Diagram przypadków użycia	10
2.2.2 Scenariusze wybranych przypadków użycia	11
2.3 Wymagania niefunkcjonalne	14
2.3.1 Bezpieczeństwo	14
2.3.2 Wykorzystywane środowiska, technologie i narzędzia	14
2.3.3 Dostępność	15
2.3.4 Inne wymagania	15
3 Analiza zastosowanych metod i technologii	16
3.1 Aplikacja Enterprise	16
3.1.1 Technologie Java Enterprise Edition	17
3.1.2 Serwer i kontenery Java EE	19
3.1.3 Enterprise JavaBeans	21
3.1.4 Mapowanie obiektowo-relacyjne	21
3.2 Usługi sieciowe typu REST	22
3.2.1 Interfejs JAX-RS oraz implementacja Jersey	22
3.2.2 Asynchroniczność i Long Pooling	23
3.2.3 Filtry	24
3.3 Podpis cyfrowy	26
3.3.1 Podpis cyfrowy z mediatorem	26
3.3.2 Model matematyczny i weryfikowanie podpisu	27
3.3.3 Generowanie i podział kluczy	29
3.3.4 Podpis pliku	31
4 Projekt systemu	32
4.1 Użytkownicy systemu	32
4.2 Projekt komunikacji pomiędzy węzłami	32
4.2.1 Logowanie	33
4.2.2 Wysyłanie wiadomości	36
4.2.3 Odbiór wiadomości	39
4.2.4 Generowanie pary kluczy	42

4.2.5	Podpisywanie pliku	45
4.2.6	Weryfikacja podpisu	48
4.3	Projekt serwera dostępowego	48
4.3.1	Komponenty serwera dostępowego	48
4.4	Projekt serwera mediatora	49
4.4.1	Komponenty mediatora	49
4.5	Projekt aplikacji klienta	50
4.5.1	Komponenty klienta	50
5	Implementacja systemu	51
5.1	Implementacja serwera dostępowego	51
5.1.1	Realizacja modułu	51
5.1.2	Realizacja filtrów zapytań	51
5.1.3	Filtr odpowiedzi	53
6	Testowanie i ocena jakości systemu	54
6.1	Instalowanie i konfiguracja systemu	54
6.1.1	Środowisko testowe	54
6.1.2	Przygotowanie bazy danych	55
6.2	Realizacja testów	56
6.2.1	Testy jednostkowe	56
6.2.2	Testy integracyjne	56
6.2.3	Testy akceptacyjne	56
6.3	Ocena wydajności operacji generowania kluczy	56
6.4	Ocena wydajności transmisji danych	56
6.5	Ocena zachowania w przypadku awarii	56
6.6	Ocena bezpieczeństwa	56
7	Podsumowanie	56

Spis rysunków

1	Kierunek wywoływania komponentów systemu.	7
2	Diagram przypadków użycia.	10
3	Wielowarstwowa aplikacja w technologii Java EE[13].	18
4	Serwer i kontenery Javy EE[13].	20
5	Przetwarzanie potokowe filtrów w JAX-RS 2.0[14]	25
6	Logowanie do systemu - diagram aktywności.	34
7	Logowanie do systemu - diagram sekwencji.	35
8	Wysyłanie wiadomości - diagram aktywności.	37
9	Wysyłanie wiadomości - diagram sekwencji.	38
10	Odbiór wiadomości - diagram aktywności.	40
11	Odbiór wiadomości - diagram sekwencji.	41
12	Diagram aktywności - generowanie pary kluczy.	43
13	Diagram sekwencji - generowanie pary kluczy.	44
14	Diagram aktywności - podpisywanie pliku.	46
15	Diagram sekwencji - podpisywanie pliku.	47
16	Interakcje między komponentami serwera dostępowego.	48
17	Interakcje między komponentami mediatora.	49
18	Interakcje między komponentami Klienta.	50
19	Środowisko testowe.	55

1 Wstęp

Słowo „kryptografia” pochodzi z języka greckiego i może być przetłumaczone jako „ukryte pismo”. Kryptografia początkowo opierała się na tajności sposobu przekazywania informacji oraz na steganograficznych sposobach zapisywania informacji w tekstach lub grafikach w taki sposób, aby nawet osoba przewożąca wiadomości nie wiedziała, że przewozi jakąś tajną wiadomość. Obecnie kryptografia zajmuje się przekształcaniem wiadomości w taki sposób, aby stała się niemożliwa do odczytania bez użycia specjalnego tajnego klucza [10].

Przekształcona w powyższy sposób wiadomość nazywana jest szyfrogramem lub kryptogramem, a cały proces nazywany jest szyfrowaniem.

W dzisiejszych czasach wiadomości wysyłane, przetwarzane i odczytywane są przy pomocy komputerów lub innych urządzeń elektronicznych. Kryptografia w kontekście dnia dzisiejszego nie zajmuje się zatem tylko szyfrowaniem tekstów, ale może być również i jest stosowana do szyfrowania wszelakich informacji przetwarzanych przez systemy komputerowe, takich jak: teksty, obrazy, klipy wideo, muzyka. Dane w systemach komputerowych przechowywane są w postaci cyfrowej i każde takie dane mogą być zaszyfrowane.

Współczesną kryptografię można podzielić na:

- symetryczną - do szyfrowania i deszyfrowania używany jest ten sam klucz,
- asymetryczną - do szyfrowania i deszyfrowania używane są różne klucze.

Kryptografia nie zajmuje się jednak tylko szyfrowaniem. Poza szyfrowaniem, czyli poufnością metody kryptograficzne mogą zapewnić:

- integralność - niezmiennosc danych w czasie procesu,
- uwierzytelnianie - autentyczność, pewność co do pochodzenia danych,
- niezaprzeczalność - nadawca nie może się wyprzec przesłania komunikatu o ustalonej treści.

Powyższe cechy mogą być dostarczone przy pomocy podpisu cyfrowego. Podpis cyfrowy to dodatkowa informacja dołączona do danych służąca do weryfikacji ich źródła, oraz zapewniająca integralność, autentyczność i niezaprzeczalność. Podpisy cyfrowe korzystają z kryptografii asymetrycznej - tworzona jest para kluczy:

- klucz prywatny - służący do podpisywania wiadomości,
- klucz publiczny - służący do weryfikacji wiadomości.

Ponadto domeną kryptografii są takie zagadnienia jak:

- dowód z wiedzą zerową,

- głosowanie elektroniczne,
- współdzielenie tajemnic,
- obliczenia wielopodmiotowe.

Kolejnym zagadnieniem będącym przedmiotem tej pracy są „systemy rozproszone”. Ze względu na różnorodność rozważanych aspektów nie istnieje jednolita definicja systemu rozproszonego. Definicja ta ewoluowała wraz z rozwojem komputerów oraz sieci komputerowych. Za system rozproszony uważa się powszechnie zbiór niezależnych urządzeń technicznych (np. komputerów) połączonych w jedną, spójną logiczną całość (np. przy pomocy sieci komputerowej).

Cechy systemów rozproszonych to[11]:

- Ukrycie przed użytkownikami systemu:
 - różnic pomiędzy poszczególnymi komputerami,
 - sposobów komunikowania się komputerów,
 - wewnętrznej organizacji systemu rozproszonego.
- Jednolity i spójny interfejs dla użytkownika - niezależnie od czasu i miejsca interakcji.

Systemy rozproszone są tworzone ze względu na istotne potencjalne zalety tych systemów. Efektywne zagospodarowanie tych zalet może sprawić, że będą one bardziej atrakcyjne dla użytkownika końcowego niż systemy scentralizowane[11].

Następnym elementem występującym w temacie, a także będącym nieodłącznym elementem tego opracowania jest „wieloplatformowość”. Wieloplatformowość jest to cecha systemu lub aplikacji, który działa na więcej niż jednej platformie. Platforma to kombinacja sprzętu i oprogramowania, na której uruchamiana jest aplikacja lub system. Platforma sprzętowa może odnosić się do architektury procesora lub architektury komputera. Platforma systemowa odnosi się do systemu operacyjnego lub maszyny wirtualnej, może być także kombinacją obydwu. Użytkownik stosuje określony program po to, aby zrealizować konkretne zadanie. Tak długo, dopóki funkcjonowanie aplikacji nie zależy od platformy w której się wykonuje, dla użytkownika jest całkowicie obojętne na jakiej platformie jest uruchamiana. Platforma sprzętowa i programowa stanowią jedynie środowisko dla uruchamiania systemów i aplikacji, same w sobie bez oprogramowania są bezużyteczne.

1.1 Cel pracy

Celem pracy jest zaprojektowanie, zaimplementowanie oraz wdrożenie systemu komputerowego umożliwiającego bezpieczną komunikację użytkowników oraz cyfrowe podpisywanie plików, a także weryfikację cyfrowo podpisanych plików. Dla uzyskania większego bezpieczeństwa generowanie klucza dla podpisu cyfrowego oraz podpisywanie plików będzie rozproszone, a więc odbywać się ono będzie w kilku miejscach systemu.

Wynikiem pracy będzie system rozproszony, którego użytkownicy końcowi będą mogli z niego korzystać pracując na różnych platformach sprzętowych i programowych. Sam system będzie zaprojektowany również w taki sposób, aby przy możliwie małych zmianach lub bez nich mógł być uruchamiany na różnych platformach. W pracy zostaną przeprowadzone testy aplikacji oraz zastosowanych mechanizmów kryptograficznych.

1.2 Zakres pracy

Pierwszym elementem pracy jest zaprojektowanie i zaimplementowanie aplikacji działającej na serwerze, która funkcjonować będzie jako punkt dostępowy dla użytkowników systemu czyli klientów. Kolejnym elementem będzie mediator, będzie to część systemu, która będzie uczestniczyć przy generowaniu klucza dla podpisu cyfrowego oraz będzie brała udział przy podpisywaniu plików. Klienci chcący korzystać z systemu będą musieli posiadać aplikacje kliencką, która będzie kolejnym elementem systemu. Administratorzy będą posiadali specjalną wersję klienta, za pomocą której będzie można dodawać, edytować lub usuwać użytkowników systemu.

2 Analiza wymagań i założenia projektowe

W niniejszym rozdziale zawarto analizę wymagań funkcjonalnych i нефункциональных oraz założenia stawiane przed systemem.

2.1 Koncepcja działania systemu

W skład systemu wchodzi następujące komponenty:

- Serwer Dostępowy,
- Mediator,
- Klient,
- Administrator Klient.

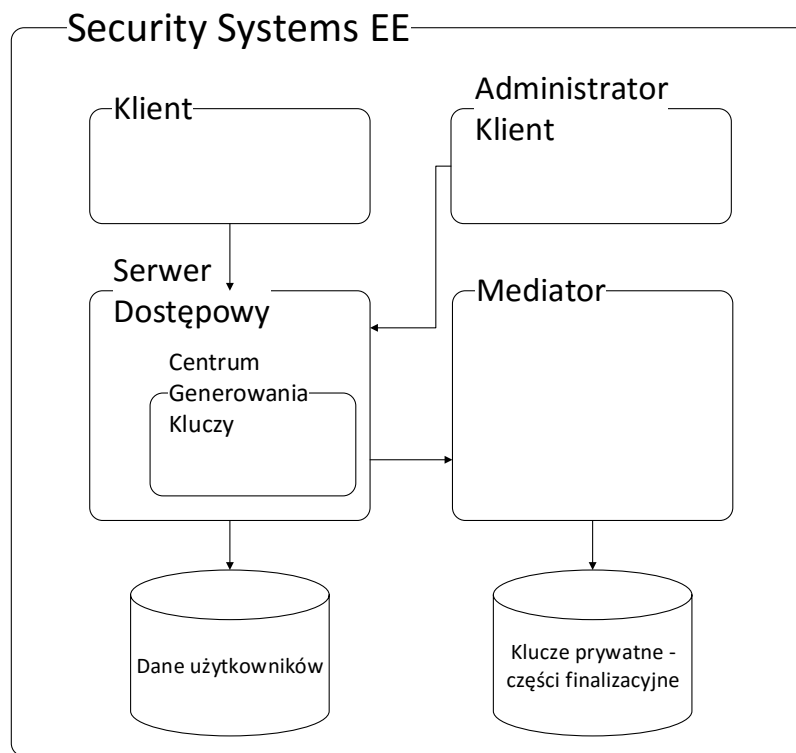
Głównym węzłem systemu jest serwer dostępowy. Serwer ten przechowuje informacje o zarejestrowanych użytkownikach systemu w bazie danych oraz implementuje logikę biznesową. Serwer ten jest punktem dostępowym do którego przy pomocy aplikacji klienckiej łączą się użytkownicy.

Użytkownik końcowy przy dostępie do systemu korzysta z aplikacji Klient, która oferuje graficzny interfejs użytkownika. Aplikacja kliencka komunikuje się bezpośrednio jedynie z serwerem dostępowym. Aplikacja Klient umożliwia wymianę wiadomości z innymi użytkownikami systemu, a także podpisywanie cyfrowe plików oraz weryfikację podpisów cyfrowych.

Mediator jest to osobny serwer, a zarazem węzeł systemu, który bierze udział podczas generowania kluczy dla podpisu cyfrowego, a także podczas procesu podpisywania pliku. Mediator posiada własną bazę danych, w której przechowuje swoje części kluczy prywatnych użytkowników.

Administrator Klient to aplikacja, dzięki której można dodawać, edytować i usuwać użytkowników systemu. Komunikuje się ona bezpośrednio z serwerem dostępowym.

2.1.1 Komponenty systemu



Rysunek 1: Kierunek wywoływania komponentów systemu.

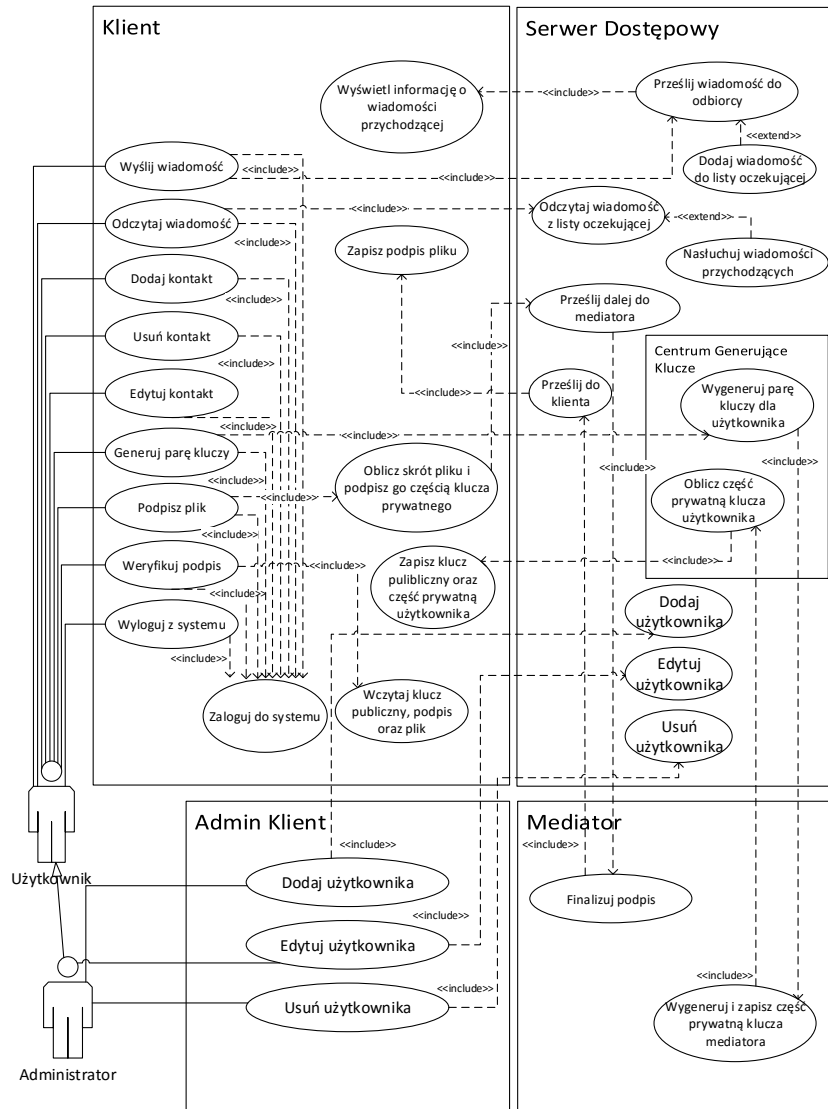
2.2 Wymagania funkcjonalne

Wymagania funkcjonalne dla systemu to:

- Możliwość dostępu do systemu jedynie po pomyślnym zalogowaniu.
- Oprócz nazwy użytkownika i hasła, system przydziela każdemu użytkownikowi specjalny klucz serwisowy, który używany jest przy komunikacji pomiędzy serwerem dostępowym a aplikacją kliencką oraz pomiędzy serwerem dostępowym a mediatorem oraz służy jako dodatkowy element walidacji.
- Wymiana wiadomości tekstowych z innymi użytkownikami w trybie rzeczywistym poprzez sieć internet.
- Dostępna lista kontaktów z możliwością modyfikacji, czyli:
 - dodawanie nowego kontaktu,
 - edytowanie kontaktu,
 - usuwanie kontaktu.
- Przechowywanie listy użytkowników lokalnie w pliku XML.
- Przechowywanie konfiguracji klienta w lokalnym pliku konfiguracyjnym.
- Powiadomienie o otrzymaniu nowej wiadomości od innego użytkownika systemu.
- Wyświetlanie statusu kontaktów:
 - online,
 - offline.
- Generowanie kluczy dla podpisu cyfrowego:
- Generowanie kluczy wymaga ponownej autoryzacji (podanie loginu i hasła)
- Podczas generowania klucz prywatny dzielony jest na dwie części:
 - część prywatna użytkownika - przechowywaną przez aplikację kliencką,
 - część prywatna mediatora - przechowywaną w mediatorze.
- Cyfrowe podpisywanie plików.

- Podpisywanie pliku odbywa się dwuetapowo. Pierwszy etap odbywa się w aplikacji klienckiej, za pomocą części prywatnej klucza użytkownika. Częściowy podpis finalizowany jest przez mediator za pomocą części prywatnej klucza mediatora.
- Weryfikacja podpisanego pliku.
- Weryfikacja odbywa się po stronie aplikacji klienckiej.
- Osobna aplikacja klient do administracji do zarządzania systemem posiadająca następujące funkcjonalności:
 - dodawanie użytkownika systemu,
 - edytowanie użytkownika systemu,
 - kasowanie użytkownika systemu.

2.2.1 Diagram przypadków użycia



Rysunek 2: Diagram przypadków użycia.

2.2.2 Scenariusze wybranych przypadków użycia

W rozdziale tym przedstawione zostały wybrane scenariusze przypadków użycia.

Scenariusz do przypadku użycia: *Generuj parę kluczy*

Warunki początkowe

1. System jest poprawnie skonfigurowany i działa prawidłowo.
2. Użytkownik ma uruchomionego klienta i jest zalogowany do systemu oraz posiada swój własny unikalny klucz serwisowy.

Główny ciąg zdarzeń

1. Użytkownik przechodzi do zakładki konfiguracja.
2. Użytkownik klika w przycisk zarządzaj nową parę kluczy.
3. Użytkownik podaje prawidłowy login i hasło, a następnie klika w przycisk generuj klucze.
4. Zostaje wysłane żądanie do serwera dostępowego o wygenerowanie nowej pary kluczy.
5. Para kluczy zostaje wygenerowana. Klucz prywatny zostaje podzielony pomiędzy mediatorem i użytkownikiem.
6. Zostaje wyświetlona informacja o kluczu oraz jego poprawnym wygenerowaniu.

Scenariusz alternatywny

1. Użytkownik podaje nieprawidłowy login lub hasło.
2. Po wciśnięciu przycisku generuj klucze zostaje wyświetlony komunikat o błędzie.

Scenariusz alternatywny 2

1. Któryś z węzłów systemu: mediator lub serwer dostępowy są niedostępne.
2. Po wciśnięciu przycisku generuj klucze zostaje wyświetlony komunikat o błędzie.

Scenariusz do przypadku użycia: *Podpisz plik***Warunki początkowe**

1. System jest poprawnie skonfigurowany i działa prawidłowo.
2. Użytkownik ma uruchomionego klienta i jest zalogowany do systemu oraz posiada swój własny unikalny klucz serwisowy.
3. Użytkownik wygenerował parę kluczy, posiada swoją część prywatną, a druga część znajduje się na mediatorze.

Główny ciąg zdarzeń

1. Użytkownik przechodzi do zakładki podpis cyfrowy.
2. Użytkownik klika w przycisk wczytaj plik, a następnie wskazuje plik, który ma zostać podpisany.
3. Użytkownik klika w przycisk podpisz plik.
4. Aplikacja kliencka składa częściowy podpis.
5. Zostaje wysłane żądanie do serwera dostępowego o żądanie podpisu.
6. Zwrócony zostaje sfinalizowany podpis pliku
7. Użytkownik dostaje informację o podpisie cyfrowym i ma możliwość jego zapisania.

Scenariusz alternatywny

1. Klucz prywatny użytkownika nie jest kompatybilny z częścią prywatną klucza mediatora, lub klucz został zablokowany.
2. Po wciśnięciu przycisku podpisz plik zostaje wyświetlony komunikat o błędzie.

Scenariusz alternatywny 2

1. Któryś z węzłów systemu: mediator lub serwer dostępowy są niedostępne.
2. Po wciśnięciu przycisku podpisz plik zostaje wyświetlony komunikat o błędzie.

Scenariusz do przypadku użycia: Wyślij wiadomość**Warunki początkowe**

1. System jest poprawnie skonfigurowany i działa prawidłowo.
2. Użytkownik posiada na liście kontaktów kontakt do którego chce wysłać wiadomość.
- 3.
4. Użytkownik ma uruchomionego klienta i jest zalogowany do systemu oraz posiada swój własny unikalny klucz serwisowy.

Główny ciąg zdarzeń

1. Użytkownik przechodzi do zakładki komunikator.
2. Użytkownik klika w kontakt, do którego chce wysłać wiadomość, a następnie w oknie wiadomości wpisuje wiadomość.
3. Użytkownik klika w przycisk wyślij wiadomość.
4. Wiadomość zostaje wysłana do odbiorcy.

Scenariusz alternatywny

1. Któryś z węzłów systemu: mediator lub serwer dostępowy są niedostępne.
2. Po wciśnięciu przycisku podpisz plik zostaje wyświetlony komunikat o błędzie.

2.3 Wymagania niefunkcjonalne

Projektowany system musi spełniać również szereg wymagań niefunkcjonalnych, które zostały przedstawione w kolejnych podpunktach.

2.3.1 Bezpieczeństwo

Kluczowym wymaganiem niefunkcjonalnym projektowanego systemu jest bezpieczeństwo.

- Komunikacja pomiędzy aplikacją kliencką, a serwerem dostępowym powinna być zaszyfrowana.
- Mediator nie powinien być dostępny z sieci internet, ale powinien być umiejscowiony w sieci wewnętrznej za firewallem i być dostępny dla serwera dostępowego.
- Serwer dostępowy powinien zostać umiejscowiony w strefie zdemilitaryzowanej DMZ sieci lokalnej i być dostępny z sieci internet oraz mieć dostęp do Mediatora.

2.3.2 Wykorzystywane środowiska, technologie i narzędzia

Z systemu powinni móc korzystać użytkownicy niezależnie od systemu operacyjnego na którym pracują. Sam system powinien móc być przy znikomych zmianach lub bez nich uruchomiony na innej platformie oraz współpracować z różnymi bazami danych.

- Aplikacja kliencka będzie napisana w technologii Java SE 1.8. Maszyna wirtualna Java działa na systemach Windows, Linux, Mac OS, oraz na wielu platformach sprzętowych.
- Serwer dostępowy oraz mediator będą napisane w technologii Java EE 1.8. Aplikacje Java EE można uruchamiać na wielu platformach serwerowych, które pracują na różnych platformach sprzętowych i systemowych.
- Baza danych dla serwera dostępowego i mediatora powinna być zdefiniowana w serwerze aplikacji.
- Do budowania wszystkich projektów używane będzie narzędzie Maven 3.3.
- Serwer dostępowy oraz mediator domyślnie będą uruchamiane i testowane na serwerze aplikacji Glassfish 4.1 oraz z bazą danych MySQL 5.7.
- Komunikacja między elementami systemu będzie odbywać się za pomocą web serwisów typu REST.

- Wszystkie elementy systemu będą napisane przy pomocy zintegrowanego środowiska programistycznego Eclipse JEE Mars 4.5.1.

2.3.3 Dostępność

- Możliwość korzystania z systemu 7 dni w tygodniu 24h na dobę z wyjątkiem przerwy na zarządzanie i konserwację (średnio 5h w miesiącu).
- Możliwość korzystania z systemu z sieci internet.

2.3.4 Inne wymagania

- Legalne funkcjonowanie systemu bez zakupu licencji na dodatkowe oprogramowanie.
- System napisany modułowo, umożliwiający łatwą modyfikację funkcjonalności, rozszerzalność oraz ponowne użycie kodu.
- Użytkownik powinien logować się do systemu jednocześnie tylko z jednej lokalizacji.
- Aplikacja kliencka posiada intuicyjny i estetyczny interfejs graficzny.

3 Analiza zastosowanych metod i technologii

W tym nieniejszym rozdziale omówione zastostały główne pojęcia i technologie stosowane w systemie oraz metody wykorzystywane podczas jego tworzenia. W ostatnim podrozdziale przedstawiony został model matematyczny podpisu cyfrowego, na podstawie którego utworzona została implementacja w systemie.

3.1 Aplikacja Enterprise

Nie ma jednoznacznej definicji opisującej aplikację typu enterprise, istnieje jednak wiele cech i właściwości, które charakteryzują aplikacje typu enterprise. Są to między innymi:

- skalowalność – to właściwość, która charakteryzuje system przygotowany na wzrost liczby użytkowników zarówno pod względem architektonicznym (rozszerzalność kodu) jak i wydajnościowym (np. load balancing),
- podział na warstwy, istnieją różne podziały, jednak zazwyczaj można wyróżnić:
 - warstwę klienta,
 - warstwę prezentacji,
 - warstwę biznesową,
 - warstwę integracji,
 - warstwę zasobów;
- bezpieczeństwo - system powinien być odporny na ataki z zewnątrz oraz spełniać aktualnie przyjęte normy bezpieczeństwa,
- modularność - elementy systemu powinny być pogrupowane w moduły,
- oddzielenie komponentów technicznych oraz logiki biznesowej,
- wysoce skompikowane systemy implementowane i wdrażane przez wiele lat,
- odporność na awarie - uzyskaną np. poprzez redundancję krytycznych elementów systemu,
- systemy rozproszone,
- łatwość odzyskiwania po awarii - uzyskaną np. poprzez automatyczne i regularne tworzenie kopii zapasowych aktualnego stanu aplikacji.

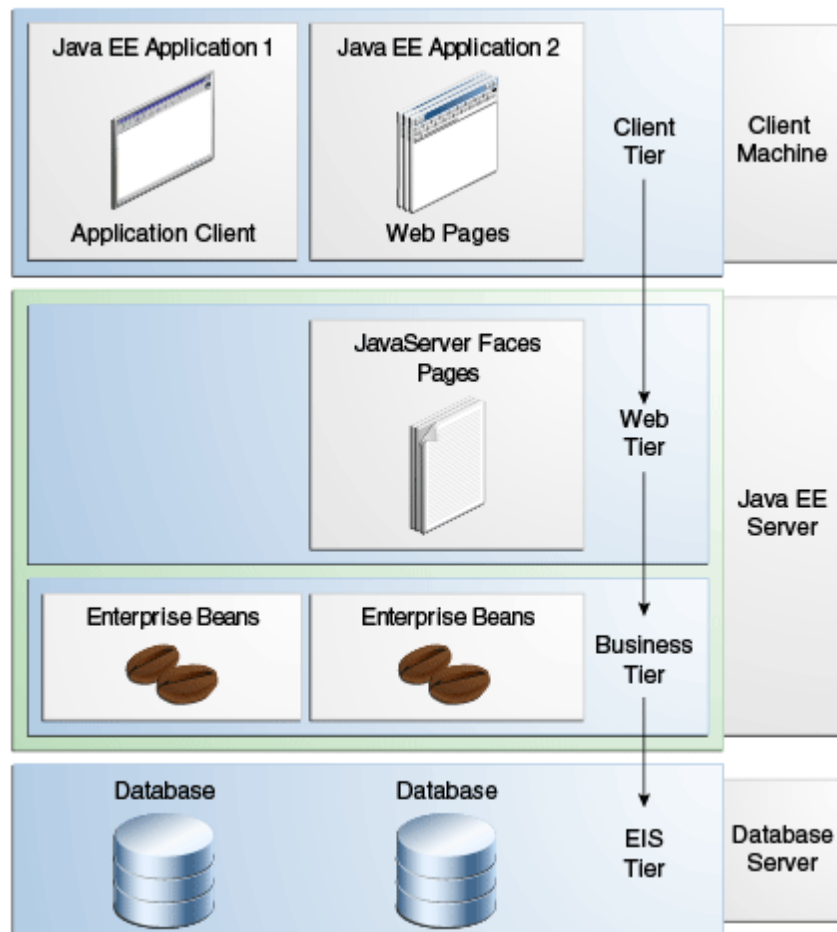
3.1.1 Technologie Java Enterprise Edition

Java EE lub inaczej Java Enterprise Edition jest to platforma programistyczna języka Java służąca do pisania aplikacji biznesowych. Java EE posiada wszystkie elementy oraz zalety platformy Java SE, dzięki czemu jest w pełni przenośna i może być implementowana oraz wdrażana na dowolnym systemie operacyjnym posiadający wirtualną maszynę Javy (JVM). Java EE rozszerza wersję SE o zbiór komponentów przeznaczonych do tworzenia aplikacji klasy enterprise. Komponent Javy EE jest niezależną, funkcjonalną jednostką oprogramowania wbudowywaną w aplikację JEE (wraz z powiązanymi z nim klasami i plikami) i komunikującą się z innymi komponentami[2]. Java EE 7 wprowadziła szereg adnotacji, dzięki którym zbędne staje się przechowywanie dużej części konfiguracji w plikach XML.

W specyfikacji Javy EE zdefiniowano następujące komponenty[2]:

- aplikacje klienckie i aplety są komponentami uruchamianymi po stronie klienta,
- komponenty technologii Java Servlet, JavaServer Faces oraz JavaServer Pages (JSP) są komponentami webowymi działającymi na serwerze,
- komponenty Enterprise JavaBeans (zwane także ziarnami EJB) będące komponentami biznesowymi działającymi na serwerze.

Warstwy aplikacji zostały przedstawione na rysunku 3.



Rysunek 3: Wielowarstwowa aplikacja w technologii Java EE[13].

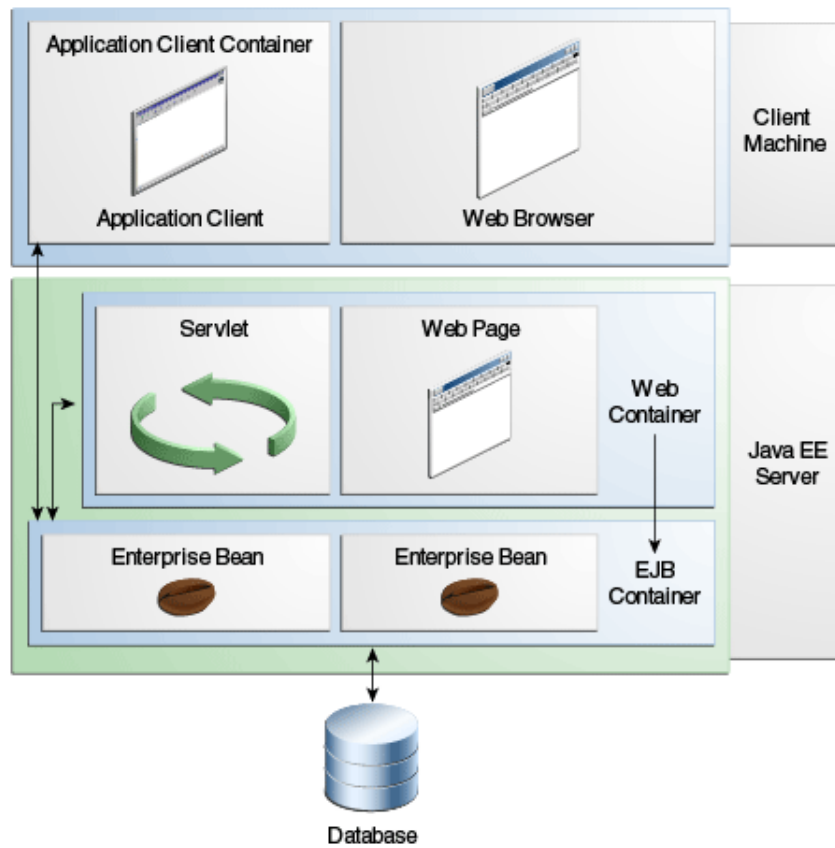
3.1.2 Serwer i kontenery Java EE

Aplikacja Java EE uruchamiana jest na serwerze Javy EE. Serwery aplikacji Javy EE to np:

- Apache Geronimo,
- GlassFish,
- JBoss Application Server,
- ObjectWeb JOnAS,
- IBM WebSphere Application Server,
- Oracle WebLogic Server,
- Oracle AS/OC4J (nierozwijany, porzucony na rzecz WebLogic Server),
- SAP Web Application Server.

Serwer Javy EE zawiera kontenery przeznaczone do przechowywania oraz uruchamiania poszczególnych części aplikacji. Na rysunku 4 zostały przedstawiony został serwer Javy EE oraz osadzone na nim kontenery.

- Serwer Javy EE - środowisko uruchomieniowe dla aplikacji JEE. Serwer Javy EE dostarcza kontenery EJB oraz kontenery webowe.
- Kontener Enterprise JavaBeans(EJB) - zarządza wykonywaniem ziaren EJB w aplikacjach JEE. Komponenty EJB i ich kontener działają na serwerze Javy EE.
- Kontener webowy - zarządza działaniem stron internetowych, serwletów i niektórych komponentów EJB aplikacji JEE. Komponenty webowe i ich kontener działają na serwerze Javy EE
- Kontener aplikacji klienckich - zarządza działaniem komponentów aplikacji klienckiej. Aplikacje klienckie i ich kontener działają po stronie klienta.



Rysunek 4: Serwer i kontenery Javy EE[13].

3.1.3 Enterprise JavaBeans

Komponenty EJB (Enterprise JavaBeans) zwane także ziarnami EJB, są fragmentami kodu zawierającymi pola oraz metody implementujące część logiki biznesowej. Komponenty EJB umożliwiają tworzenie i wdrażanie rozproszonych aplikacji na bazie komponentów, które oferują skalowalność, możliwość przetwarzania transakcyjnego i bezpieczeństwo. Za zarządzanie instancją komponentu w czasie wykonywania odpowiada kontener. Klient uzyskuje dostęp do komponentu za pośrednictwem kontenera, w którym dany komponent został wdrożony. Klient może też działać po stronie serwera i mieć na przykład postać komponentu zarządzalnego, komponentu CDI lub serwletu[1].

Istnieją trzy rodzaje komponentów EJB:

- komponenty sesyjne, które dzielone na:
 - stanowe - każdy komponent przechowuje unikalny stan, np. stan komunikacji z klientem;
 - bezstanowe - komponenty te wbrew nazwie też posiadają swój unikalny stan, jednak nie może być on powiązany bezpośrednio z żądaniem, na które może zostać zwrócony inny komponent tego samego typu.
 - singletony - unikalne komponent na skalę środowiska uruchomieniowego, przechowujące np stan aplikacji;
- komponenty sterowane komunikatami - komponenty wykorzystywane podczas komunikacji w technologii JMS;
- komponenty encyjne - komponenty przechowujące encje danych, od wersji EJB 3.0 oznaczone są jako przestarzałe, a do przechowywania encji zaleca się stosowanie klas z adnotacją @Entity.

3.1.4 Mapowanie obiektowo-relacyjne

Mapowanie obiektowo-relacyjne (ORM) jest to sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze. Implementacja takiego odwzorowania stosowana jest m.in. w przypadku, gdy tworzony system oparty jest na podejściu obiektowym, a system bazy danych operuje na relacjach.

JavaPersistence API (JPA) jest standardem ORM dla języka Java. Z punktu widzenia programisty jest to możliwość operowania na obiektach - zwanych encjami - oraz zapisywania wyników operacji do relacyjnej bazy danych za pomocą obiektu EntityManager. Sposób w jaki obiekty i ich połączenia przekładają się na elementy bazy danych są definiowane za pomocą adnotacji lub dokumentów XML. Poza standardowym zestawem operacji udostępnianych przez obiekt EntityManager standard JPA definiuje język zapytań JPA Query Language podobny do SQL.[15]. Encje w JPA oznaczane są adnotacją @Entity.

3.2 Usługi sieciowe typu REST

REST to akronim od representational state transfer. REST jest usługą sieciową zaimplementowaną na bazie protokołu HTTP. REST to też architektura, która określa model i zasady umożliwiające budowanie rozwiązań, które pozwalają na bardzo dużą skalowalność, wydajność i łatwość modyfikacji. W architekturze REST dane i funkcjonalność reprezentowane są jako zasób. Każdy zasób reprezentowany jest przez oddzielny URL. Rest oprócz zasobów reprezentowanych przez Uri to również kwestia reprezentacji - dany zasób może być reprezentowany jako wynik typu XML, Json, Text, Html, Image oraz wiele innych typów. Komunikacja REST jest bezstanowa. Usługi inaczej web serwisy typu REST nazywane są też serwisami RESTful.

Usługi REST korzystają zazwyczaj z 4 metod. Metody te to:

- Get - używana przy dostępie do zasobu,
- Put - używana przy tworzeniu zasobu oraz przy innych operacjach,
- Update - używana przy aktualizacji zasobu,
- Delete - używana do kasowania zasobu.

3.2.1 Interfejs JAX-RS oraz implementacja Jersey

JAX-RS (od ang. Java API for RESTful web services) to API języka Java, które powstało w celu ułatwienia tworzenia aplikacji opartych o architekturę REST. Od wersji 2.0 JAX-RS definiuje interfejs sterowany adnotacjami. Adnotacje JAX-RS są adnotacjami środowiska uruchomieniowego Java EE. Po uruchomieniu w kontenerze Java EE w środowisku uruchomieniowym zostają wygenerowane klasy pomocnicze oraz artefakty niezbędne do działania web serwisu, następnie usługa jest automatycznie konfigurowana, po czym zostaje udostępniona[2].

JAX-RS oferuje między innymi następujące adnotacje:

- @Path - jest względną ścieżką URI wskazującą miejsce hostowania klasy Javy,
- @GET - metody oznaczone tym desygnatorem będą obsługiwały żądania GET,
- @POST - metody oznaczone tym desygnatorem będą obsługiwały żądania POST,
- @PUT - metody oznaczone tym desygnatorem będą obsługiwały żądania PUT,

- @DELETE - metody oznaczone tym desygnatorem będą obsługiwały żądania DELETE,
- @HEAD - metody oznaczone tym desygnatorem będą obsługiwały żądania HEAD,
- @PathParam - adnotacją tą służy do wyekstrahowania parametru z URI. nazwy parametrów odpowiadają nazwom zmiennych określonych we wzorcu URI;
- @QueryParam - adnotacją tą służy do wyekstrahowania parametru z URI, parametry zapytania są ekstrahowane z parametrów zapytania adresu URI;
- @Consumes - używana do określenia przetwarzanych przez zasób typów MIME,
- @Produces - używana do określenia zwracanych przez zasób typów MIME
- @Provider - adnotacją tą oznacza się wszystkie elementy, które mogą być potrzebne w środowisku uruchomieniowym.

Istnieje szereg dalszych adnotacji np: @MatrixParam, @HeaderParam, @CookieParam, @FormParam, @DefaultValue, @Context.

Jersey jest to referencyjna implementacja JAX-RS w Javie. Jersey to framework o otwartym źródle, oferujący również szereg funkcjonalności wykraczających poza standardowe API JAX-RS np. implementację klienta REST.

3.2.2 Asynchroniczność i Long Pooling

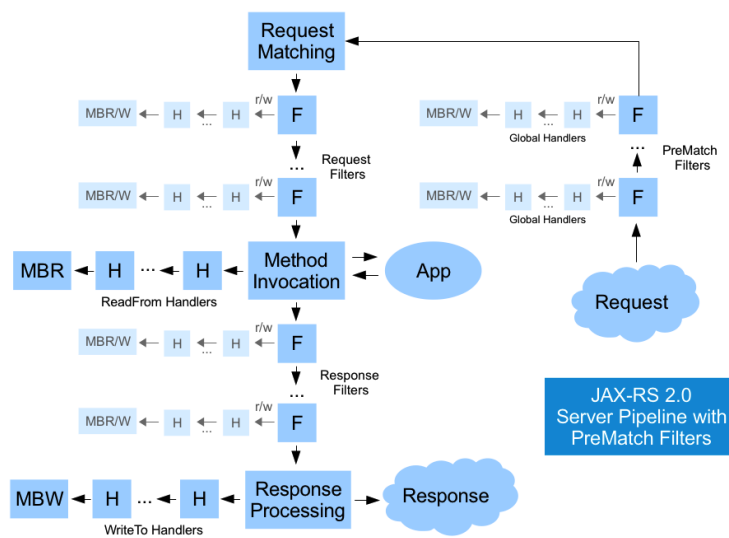
Typowe żądania typu REST przetwarzane są w trybie synchronicznym tzn. klient wysyła żądanie, serwer je przetwarza i zwraca odpowiedź. Żądania synchroniczne stosuje się wtedy gdy reprezentowany przez URI zasób może być od razu zwrócony. W przypadku gdy zasób nie jest od razu dostępny, lub jego przygotowanie trwa zbyt długo żądanie synchroniczne zakończy się niepowodzeniem. W takim przypadku należy stosować żądania asynchroniczne. Przetwarzanie asynchroniczne polega na tym, że serwer nie zwraca od razu żądania, lecz przekazuje je do innego wątku, gdzie odpowiedź na żądanie jest przygotowywana. Gdy zasób zostanie utworzony jest on zwracany do klienta. Istnieje technika przetwarzania asynchronicznego, która nazywa się Long Pooling. Polega ona na tym, że klient wysyła żądanie do serwera. W przypadku gdy zasób nie jest dostępny lub nie może zostać od razu utworzony to żądanie przechodzi w tryb oczekujący i zwrócone zostanie dopiero wtedy, gdy zasób będzie dostępny. Technikę taką stosuje się np. w komunikatorach. Klient wysyła żądanie, które zwrócone zostanie dopiero wtedy, gdy inny klient wyśle do niego wiadomość.

3.2.3 Filtry

Interfejs JAX-RS oferuje szereg filtrów. Filtry mogą być stosowane np. w celu modyfikacji, odrzucenia, czy przekierowania żądania. Przy pomocy filtrów można np. uwierzytelnić nadawcę żądania. Filtrowanie w JAX-RS odbywa się potokowo, tzn. można zdefiniować kilka filtrów oraz ich kolejność. Filtry w JAX-RS dzielą się na trzy grupy:

- PreMatch Filters - są to filtry które stosowane są do żądania jeszcze przed dopasowaniem do danego zasobu.
- Request Filters - filtry te stosowane są po filtrach PreMatch oraz po tym jak żądanie zostało dopasowane do zasobu, ale przed wykonaniem metody zasobu.
- Response Filters - są to filtry, które stosowane są do odpowiedzi na żądanie. Filtry te stosowane są po zakończeniu metody zasobu, ale przed odesłaniem zasobu do klienta. Mogą operować na danych zwróconych przez zasób.

Przetwarzanie potokowe filtrów w JAX-RS 2.0 zostało przedstawione na rysunku 5.



Rysunek 5: Przetwarzanie potokowe filtrów w JAX-RS 2.0[14]

3.3 Podpis cyfrowy

Podpis cyfrowy to matematyczny sposób sprawdzenia autentyczności danych elektronicznych. Zweryfikowany podpis cyfrowy oznacza, że dane pochodzą od właściwego nadawcy, który nie może zaprzeczyć faktowi ich podpisania oraz, że dane od momentu podpisu nie zostały zmienione.

Podpis cyfrowy zapewnia następujące funkcje bezpieczeństwa:

- autentyczności pochodzenia, która daje pewność co do autorstwa danych,
- niezaprzeczalności, która utrudnia wyparcie się autorstwa lub znajomości treści danych,
- integralności, która pozwala wykryć nieautoryzowane modyfikacje danych po ich podpisaniu.

Dwa dominujące standardy podpisu cyfrowego to RSA oraz DSA.

W niniejszym opracowaniu rozpatrywany jest standard RSA. Jego nazwa pochodzi od pierwszych liter nazwisk jego twórców: Rivest, Shamir i Adleman. RSA oparty jest o kryptografię asymetryczną. Bezpieczeństwo szyfrowania opiera się na trudności faktoryzacji dużych liczb złożonych. W podpisie cyfrowym opartym o algorytm RSA stosuje się klucz prywatny do podpisywania, oraz klucz publiczny do weryfikacji podpisu.

3.3.1 Podpis cyfrowy z mediatorem

Podpis elektroniczny z mediatorem to podpis cyfrowy, do którego złożenia potrzebna jest dodatkowa zaufana strona zwana mediatorem. Klucz prywatny dzielony jest między użytkownikiem a mediatorem. Mediator bierze też udział przy generowaniu klucza prywatnego.

Generowanie klucza składa się z następujących faz:

- utworzenie przez diler (generatora kluczy) pary kluczy, klucza prywatnego i publicznego,
- wysłanie do mediatora klucza publicznego oraz identyfikatora klucza,
- wygenerowanie przez mediator z modułu klucza publicznego i identyfikatora klucza swojej części klucza prywatnego i odesłania go do diler kluczy,
- obliczenie przez diler kluczy części prywatnej użytkownika na podstawie części klucza otrzymanego od mediatora i wygenerowanego w pierwszym punkcie klucza prywatnego

Złożenie podpisu cyfrowego przy z pomocą mediatora składa się z dwóch faz:

- utworzenie częściowego podpisu po stronie użytkownika, a następnie wysłanie częściowego podpisu oraz podpisywanych danych do mediatora,
- podpisanie częściowego podpisu (finalizacja podpisu) przez stronę zaufaną, czyli mediatora, a następnie odesłanie podpisu do użytkownika.

Weryfikacja podpisu odbywa się identycznie jak w przypadku podpisu cyfrowego opartego o standardowy algorytm RSA w całości po stronie użytkownika.

3.3.2 Model matematyczny i weryfikowanie podpisu

Użytkownik posiadający w systemie swój własny niepowtarzalny identyfikator ID oznaczony zostanie jako - u .

Dla każdego u może zostać wygenerowana para kluczy:

- (d_u, N_u) - klucz prywatny,
- (e_u, N_u) - klucz publiczny,

gdzie:

- (d_u) - eksponenta klucza prywatnego,
- (e_u) - eksponenta klucza publicznego,
- (N_u) - moduł klucza prywatnego i klucza publicznego.

klucz prywatny dzielony jest na dwie części:

- (d_{K_u}, N_u) - część prywatna klucza strony finalizującej podpis mediatora,
- (d_{M_u}, N_u) - część prywatna klucza strony podpisującej użytkownika,

gdzie:

- d_{K_u} - część prywatna eksponenty strony podpisującej użytkownika,
- d_{M_u} - część prywatna eksponenty strony finalizującej podpis mediatora.

Eksponenty kluczy po podziale muszą spełniać następujący warunek:

- $d_u \equiv d_{K_u} + d_{M_u} \pmod{\varphi(N_u)}$,

gdzie:

- φ - funkcja nosząca nazwisko Eulera przypisująca każdej liczbie naturalnej liczbę liczb względnie z nią pierwszych nie większych od niej samej.

Plik - p podpisany cyfrowo przez użytkownika - u ma postać:

- $(p, h(p)^{d_u} \pmod{N_u})$,

gdzie:

- $h(p)$ - to skrót kryptograficzny pliku - p lub inaczej funkcja haszująca pliku - p,
- $h(p)^{d_u} \pmod{N_u}$ - to skrót pliku - p podniesiony do potęgi d_u modulo N_u
- $h(p)^{d_u} \pmod{N_u} \equiv h(p)^{d_{k_u}} \pmod{N_u} \times h(p)^{d_{M_u}} \pmod{N_u}$.

Aby zweryfikować cyfrowo podpisany plik w postaci $(p, h(p)^{d_u} \pmod{N_u})$, należy dysponować kluczem publicznym (e_u, N_u) użytkownika - u. W przypadku gdy do danego klucza publicznego dostarczony jest certyfikat - c ma on postać:

- (e_u, N_u, c_u) .

Certyfikat c_u ma postać:

- $(u, e_u, N_u, h(u, e_u, N_u)^{d_c} \pmod{N_c}, e_c, N_c)$,

gdzie:

- $h(u, e_u, N_u)^{d_c} \pmod{N_c}$ to skrót klucza publicznego użytkownika oraz identyfikatora użytkownika podniesiony do potęgi eksponenty klucza prywatnego certyfikatu modulo moduł klucza publicznego certyfikatu N_c ,

Klucz publiczny i klucz prywatny certyfikatu - c przechowywany jest w urzędzie certyfikującym:

- U_c .

Weryfikacja podpisu następuje w dwóch etapach. Pierwszy etap to sprawdzenie autentyczności certyfikatu. Następuje to poprzez obliczenie skrótu z pierwszych trzech wartości certyfikatu (u, e_u, N_u) . Otrzymany w ten sposób skrót $h(u, e_u, N_u)$ porównywany jest z wartością $(h(u, e_u, N_u)^{d_c})^{e_c} \pmod{N_c}$, która jest czwartą częścią certyfikatu podniesioną do potęgi e_c modulo N_c . Jeżeli zachodzi równość:

- $h(u, e_u, N_u) = (h(u, e_u, N_u)^{d_c})^{e_c} \pmod{N_c}$.

Oznacza to, że certyfikat pomyślnie przeszedł próbę autoryzacji. Autentyczność certyfikatu implikuje autentyczność klucza publicznego użytkownika - u. Umożliwia to weryfikację podpisu samego pliku:

- $(u, p, h(p)^{d_u} \pmod{N_u})$.

Weryfikacja pliku - p następuje poprzez obliczenie $h(p)$, a następnie porównanie tej wartości z $(h(p)^{d_u})^{e_u} \pmod{N_u}$, gdzie $(h(p)^{d_u})^{e_u} \pmod{N_u}$ to $h(p)^{d_u} \pmod{N_u}$ podniesione do potęgi eksponenty e_u autoryzowanego klucza publicznego (e_u, N_u) modulo N_u . Jeżeli zachodzi równość:

- $h(p) = (h(p)^{d_u})^{e_u} \pmod{N_u}$,

oznacza to, że proces weryfikacji zakończył się pomyślnie. Jeżeli równość ta nie zachodzi może oznaczać to że:

- plik - p od momentu podpisu został zmieniony,
- plik - p został podpisany przy pomocy innego klucza prywatnego,
- klucz publiczny nie należy do użytkownika, który podpisał plik.

Plik może być również zweryfikowany bez certyfikatu klucza publicznego.

3.3.3 Generowanie i podział kluczy

Użytkownik - u chcący podpisać plik - p musi posiadać parę kluczy, która składa się z klucza prywatnego biorącego udział przy składaniu podpisu, oraz z klucza publicznego, który jest niezbędny do weryfikacji podpisu. Za generowanie kluczy odpowiedzialne jest centrum generowania kluczy lub inaczej generator kluczy. Generowanie kluczy przebiega następująco:

- Wybierz losowo dwie duże liczby pierwsze p i q przy uwzględnieniu następujących warunków:
 - liczby powinny mieć zbliżoną długość w bitach, ale jednocześnie powinny być od siebie odległe wartościami;
 - długość bitowa wartości $p \times q$ powinna być równa lub zbliżona do długości bitowej obliczanego klucza.
- Oblicz $N = p \times q$. N jest modułem klucza publicznego i klucza prywatnego.
- Oblicz $\phi(N) = \phi(p) \times \phi(q) = (p-1) \times (q-1) = N - (p+q-1)$.

- Wybierz losowo $e \in \{1, 2, 3, \dots, \phi(N)\}$ względnie pierwsze z $\phi(N)$ tzn. największy wspólny dzielnik $NWD(e, \phi(N)) = 1$. Składnik e jest eksponentą klucza publicznego.

Zaleca się aby wartość e miała krótką długość bitową, oraz małą wagę hamminga, tzn. sumę wartości różnych od 0 w danym alfabecie, w tym przypadku $alfabet = \{0, 1\}$ (rozpatrywany jest zapis bitowy liczby). Typową stosowaną wartością jest np. $e = 2^{16} + 1 = 65,537$.

- Oblicz eksponentę klucza prywatnego $d \in \{1, 2, 3, \dots, \phi(N)\}$ taką że:
 - wartość $d \equiv e^{-1} \pmod{\phi(N)}$, lub inaczej $d \times e = 1 \pmod{\phi(N)}$, tzn. wartość d jest odwrotnością modularną e .
- Wartości p, q oraz $\phi(N)$ powinny zostać niejawne, ponieważ mogą one służyć do obliczenia d .

Z tak obliczonych wartości tworzy się następnie:

- (d, N) - klucz prywatny,
- (e, N) - klucz publiczny.

Po obliczeniu pary kluczy, klucz prywatny dzielony jest na dwie części:

- (d_M, N) - część prywatna klucza strony finalizującej podpis mediatora,
- (d_K, N) - część prywatna klucza strony podpisującej użytkownika.

Klucze mają wspólny moduł, lecz różne eksponenty. Eksponenty te muszą spełniać następujący warunek:

- $d \equiv d_K + d_M \pmod{\phi(N)}$.

Podział klucza prywatnego odbywa się w dwóch etapach. Najpierw generowana jest część prywatna klucza strony finalizującej podpis mediatora. Proces ten przebiega następująco:

- Centrum generowania kluczy wysyła do mediatora wygenerowany klucz publiczny oraz identyfikator ID użytkownika - u , dla którego zostały wygenerowane klucze.
- Mediator posiada swoją własną parę kluczy znaną tylko jemu. Swoim kluczem prywatnym (d_m, N_m) mediator podpisuje ID użytkownika. Podpis ten jest postaci $h(ID)^{d_m} \pmod{N_m}$.

Podpis $h(ID)^{d_m} \pmod{N_m}$ służy jako ziarno dla generatora liczb pseudolosowych.

- Generator liczb pseudolosowych o ziarnie $h(ID)^{d_m} \pmod{N_m}$ losuje liczbę pseudolosową d_M , taką że:
 - długość bitowa liczby d_M to suma długości bitowej modułu klucza publicznego N przesłanego przez Centrum generowania kluczy oraz wartości δ , gdzie $\delta \in \{80 \dots 128\}$.
 - Długość bitowa N może być obliczona jako $\text{floor}(\log_2(N)) + 1$.

Liczby (d_M, N) tworzą część prywatna klucza strony finalizującej podpis mediatora. Liczba d_M wygenerowana w mediatorze odsyłana jest następnie do centrum generowania kluczy, gdzie na jej podstawie obliczana jest część prywatna klucza strony podpisującej użytkownika. Odbywa się to w następujący sposób:

- $d_K = d - d_M \pmod{\phi(N)}$.

Liczba d_K wraz z modułem N tworzą część prywatna klucza strony podpisującej użytkownika (d_K, N) .

3.3.4 Podpis pliku

Użytkownik - u chcący podpisać plik - p oblicza skrót pliku $h(p)$, a następnie podnosi go do potęgi d_{K_u} modulo N_u . Tak częściowo podpisany plik ma postać:

- $h(p)^{d_{K_u}} \pmod{N_u}$.

Użytkownik - u wysyła do mediatora skrót pliku - p $h(p)$ oraz częściowy podpis $h(p)^{d_{K_u}} \pmod{N_u}$.

Mediator przy pomocy swojej części klucza prywatnego podpisuje otrzymany skrót $h(p)$ otrzymując w ten sposób $h(p)^{d_u} \pmod{N_u}$, a następnie mnoży go z otrzymanym podpisem modulo N_u . Po takiej operacji podpis jest kompletny i ma postać:

- $h(p)^{d_{K_u}} \pmod{N_u} \times h(p)^{d_{M_u}} \pmod{N_u} = h(p)^{d_u} \pmod{N_u}$.

Podpis taki odsyłany jest do użytkownika - u, który może dołączyć go do pliku - p. Otrzymując podpisany plik w postaci:

- $(p, h(p)^{d_u} \pmod{N_u})$.

4 Projekt systemu

W niniejszym rozdziale został przedstawiony projekt systemu. W pierwszym podrozdziale przedstawiona została komunikacja oraz interakcje pomiędzy poszczególnymi elementami systemu. W kolejnych podrozdziałach przedstawione zostały projekty poszczególnych elementów systemu: serwera dostępowego, mediatora, klienta oraz klienta administratorskiego.

4.1 Użytkownicy systemu

Użytkownik w systemie może być przypisany do następujących ról.

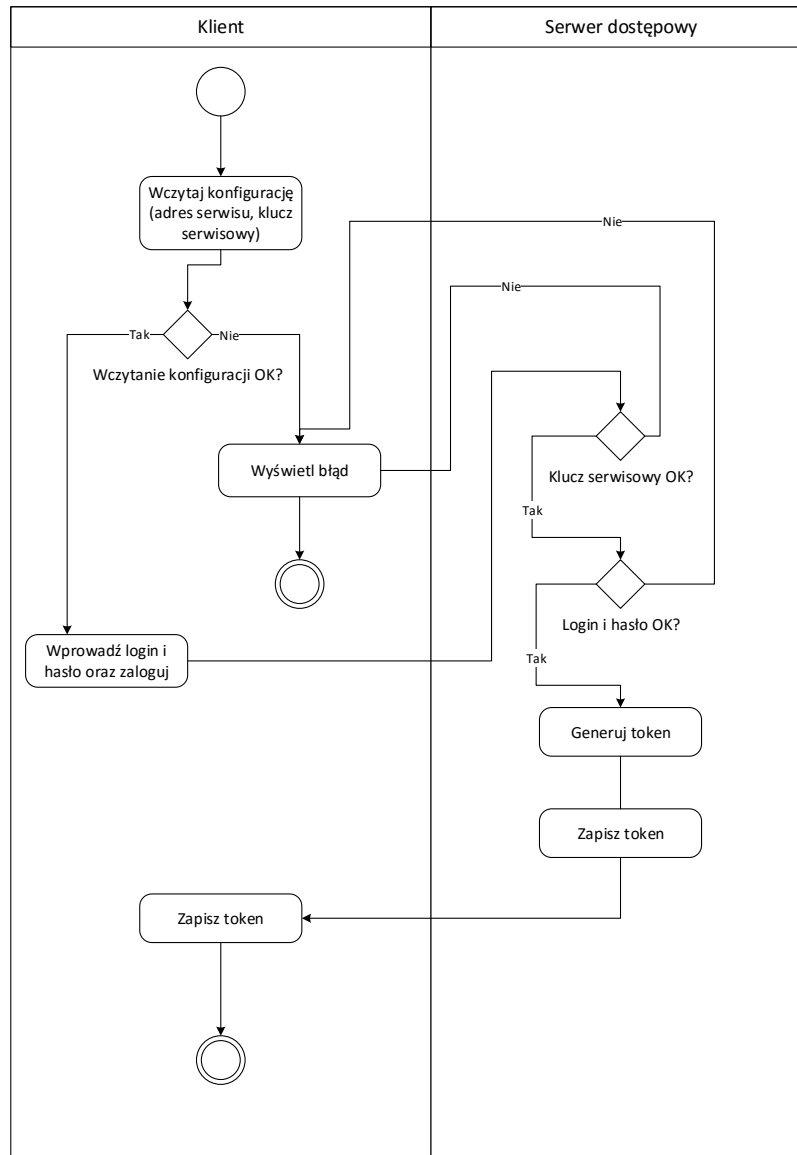
- Użytkownik standardowy - posiada on standardową funkcjonalność, przedstawioną w diagramie przypadku użycia z wyjątkiem, z wyłączeniem działań administracyjnych, a więc zarządzaniem użytkownikami systemu.
- Administrator - posiada on funkcjonalność użytkownika standardowego, a dodatkowo możliwość zarządzania użytkownikami systemu. Administrator ma możliwość logowania się do systemu za pośrednictwem aplikacji administratorskiej.

4.2 Projekt komunikacji pomiędzy węzłami

W podrozdziale tym przedstawiony został przebieg komunikacji pomiędzy elementami systemu dla wybranych przypadków użycia. Komunikacja w systemie odbywa się za pośrednictwem web serwisów typu REST. W przypadku logowania, wylogowywania oraz wysyłania wiadomości są to zapytania synchroniczne. W przypadku odbioru wiadomości, generowania klucza oraz podpisywania pliku są to zapytania asynchroniczne typu Long Pooling. Klient administratorski używa zapytania asynchronicznego do pobrania listy użytkowników. W przypadku gdy jakiś węzeł systemu jest niedostępny klient systemu dostaje komunikat o błędzie. W przypadku łączenia się klientów z sieci Internet, komunikacja pomiędzy elementami systemu powinna być szyfrowana. Konfiguracja szyfrowania odbywa się na serwerze Javy EE. Zaleca się włączenie szyfrowania za pomocą protokołu TLS. Należy wtedy dostarczyć certyfikat serwera lub taki wygenerować.

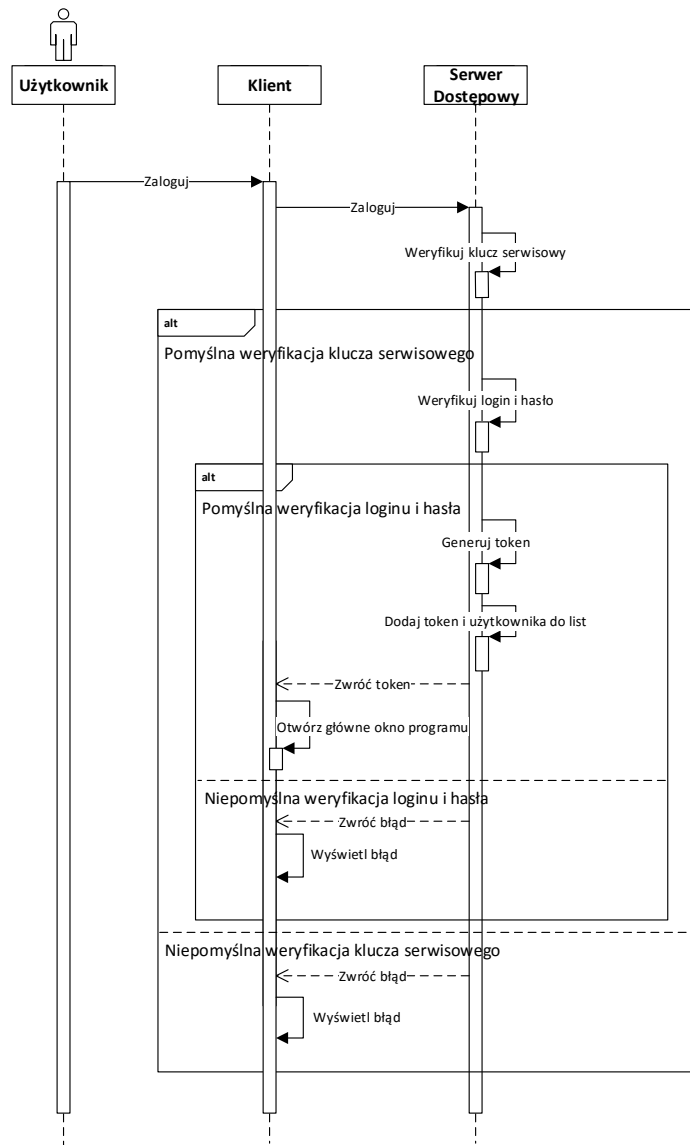
4.2.1 Logowanie

Logowanie to proces, który inicjuje klient systemu. Do zalogowania się do systemu niezbędna jest znajomość loginu, hasła oraz klucza serwisowego. Wszystkie te elementy dostarczane są przez administratora po założeniu konta użytkownika w systemie. Klient wysyła do serwera dostępowego żądanie zalogowania, a wraz z nim użytkownika, hasło i klucz serwisowy, następnie po pomyślnym zautoryzowaniu użytkownika na serwerze dostępowym generowany jest unikalny token. Token dodawany jest do listy tokenów, a następnie zwracany jest do klienta. Po otrzymaniu tokena w odpowiedzi REST klient zapisuje go. Od tego momentu klient jest zalogowany, a każde następne żądanie, aż do wylogowania odbywa się przy pomocy klucza serwisowego oraz tokena bez udziału loginu i hasła. Przy żądaniu wylogowania z systemu serwer dostępowy usuwa z token z listy tokenów. Diagramy na rysunkach 6 i 7 przedstawiają proces logowania.

Diagram aktywności dla logowania

Rysunek 6: Logowanie do systemu - diagram aktywności.

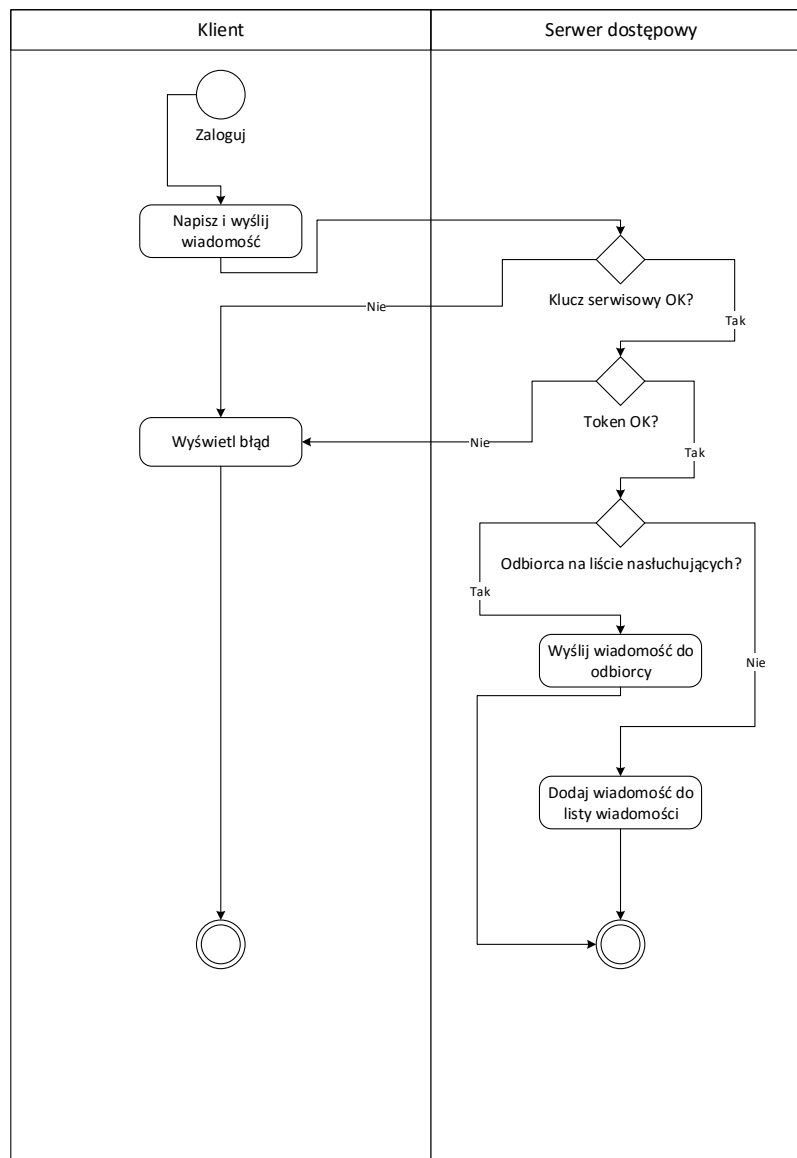
Diagram sekwencji dla logowania



Rysunek 7: Logowanie do systemu - diagram sekwencji.

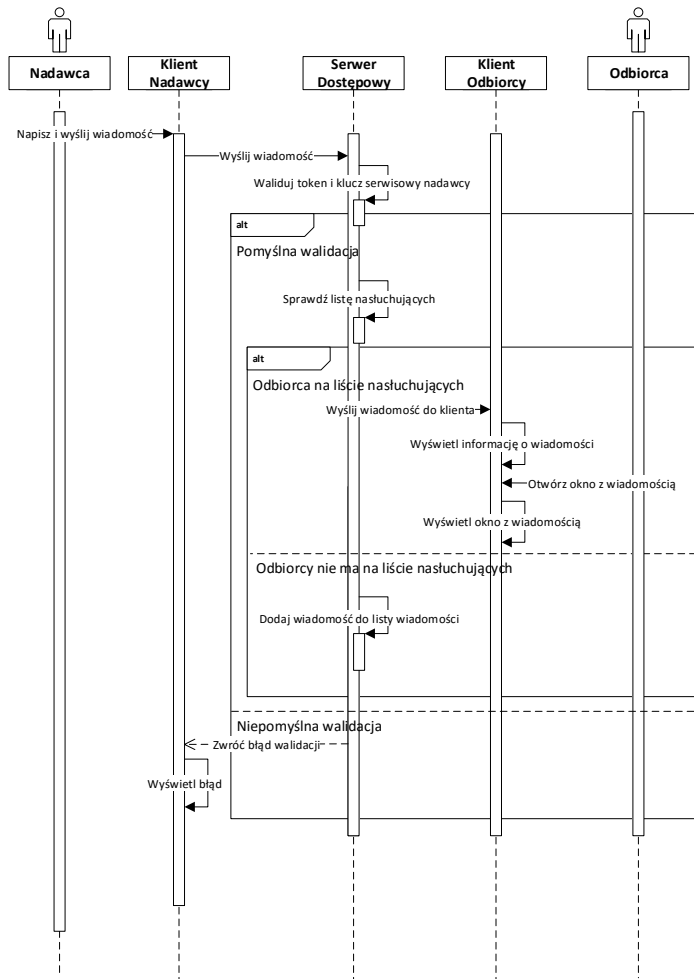
4.2.2 Wysyłanie wiadomości

Użytkownik ma możliwość wysyłania wiadomości tekstowych do innych użytkowników systemu. Aby wysłać wiadomość z klienta należy być zalogowanym do systemu. Klient wysyła żądanie REST do serwera dostępowego, w treści którego znajduje się wiadomość a w nagłówku klucz serwisowy oraz token. Serwer dostępowy następnie autoryzuje użytkownika oraz sprawdza listę nasłuchujących użytkowników. Jeżeli adresat wiadomości znajduje się na tej liście to serwer dostępowy przekazuje mu wiadomość od nadawcy. Jeżeli użytkownika nie ma na liście, to wiadomość dodawana jest do listy wiadomości i będzie tam aż do chwili zalogowania się adresata wiadomości do systemu, po czym zostanie mu przekazana. Na rysunkach 8 i 9 został przedstawiony proces, który zachodzi po wysłaniu wiadomości z aplikacji klienta.

Diagram aktywności dla wysyłania wiadomości

Rysunek 8: Wysyłanie wiadomości - diagram aktywności.

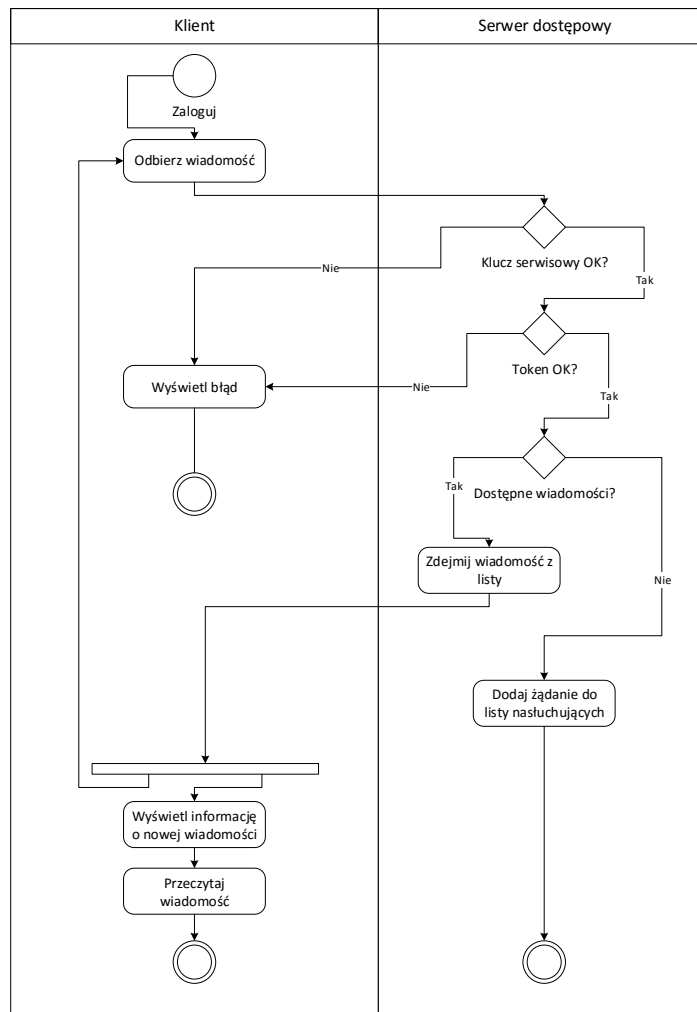
Diagram sekwencji dla wysyłania wiadomości



Rysunek 9: Wysyłanie wiadomości - diagram sekwencji.

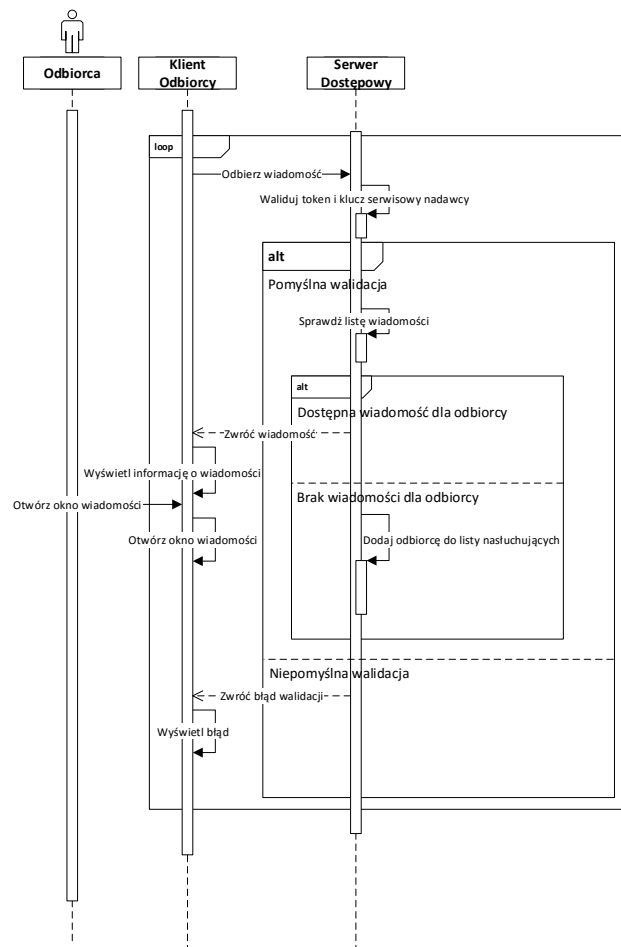
4.2.3 Odbiór wiadomości

Odbiór wiadomości odbywa się również przy pomocy asynchronicznego zapytania REST. Po zalogowaniu klient wysyła do serwera dostępowego żądanie odbioru wiadomości z nagłówkiem zawierającym klucz serwisowy i token. Po pomyślnej autoryzacji serwer dostępowy sprawdza listę wiadomości w poszukiwaniu wiadomości, której adresat jest użytkownikiem wysyłającym żądanie. W przypadku znalezienia takiej wiadomości zostaje ona dołączona do odpowiedzi i jest zwracana doklienta. W przypadku, gdy na liście wiadomości nie ma żadnej wiadomości do użytkownika, to żądanie zapisywane jest na liście nasłuchujących i znajduje się tam aż do chwili pojawienia się wiadomości do użytkownika. Każde żądanie wysyłające wiadomość sprawdza najpierw listę nasłuchujących. W chwili pojawienia się na serwerze dostępowym nowego żądania wysłania wiadomości z wiadomością do użytkownika, którego zapytanie znajduje się na liście nasłuchujących, wiadomość zostaje dołączona do żądania oczekującego, a żądanie zostaje usunięte z listy nasłuchujących i jest odesyłane do klienta jako odpowiedź na żądanie asynchroniczne. Aplikacja klient po otrzymaniu odpowiedzi wyświetla na liście kontaktów obok kontaktu, który jest adresatem wiadomości informację o nowej wiadomości. Proces w aplikacji klient do odbioru wiadomości uruchamiany jest jako osobny wątek w tle. Proces ten działa w pętli tak, że po wysłaniu żądania odbioru i otrzymaniu odpowiedzi wysyła ponowne żądanie odbioru wiadomości. Na rysunkach 10 i 11 przedstawiony został proces odbioru wiadomości.

Diagram aktywności dla odbioru wiadomości

Rysunek 10: Odbiór wiadomości - diagram aktywności.

Diagram sekwencji dla odbioru wiadomości

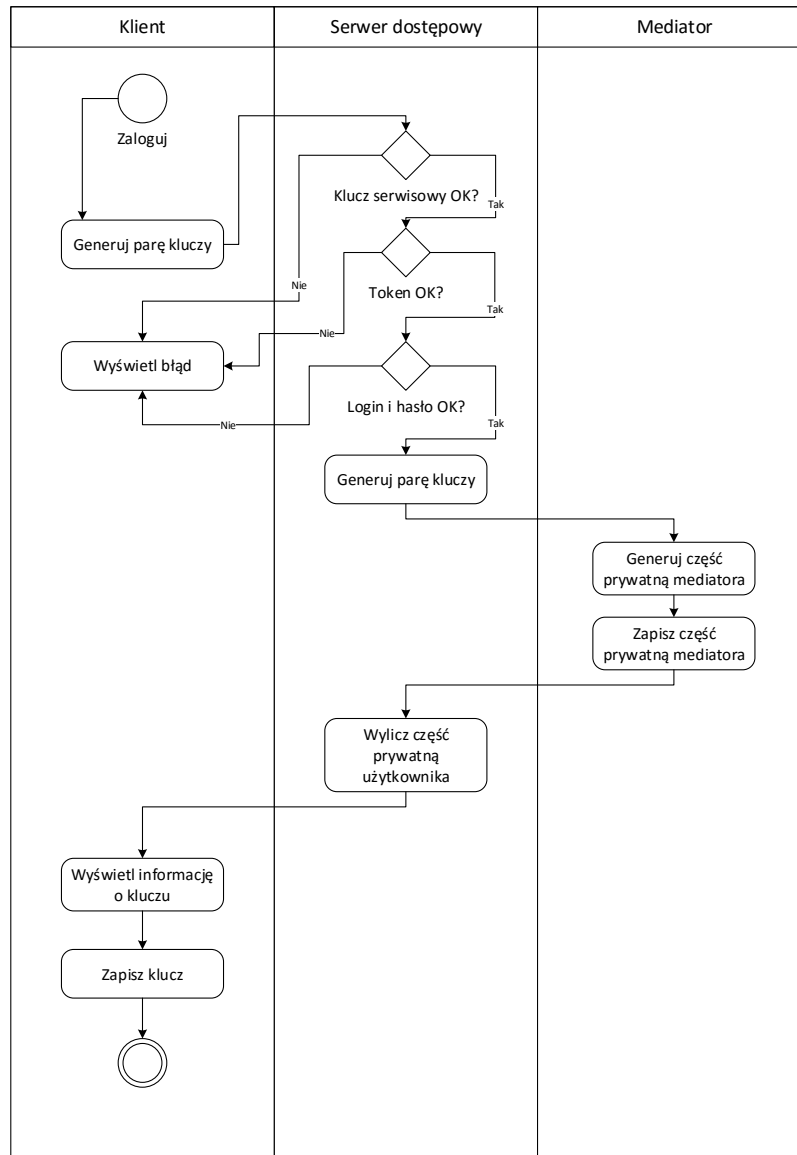


Rysunek 11: Odbiór wiadomości - diagram sekwencji.

4.2.4 Generowanie pary kluczy

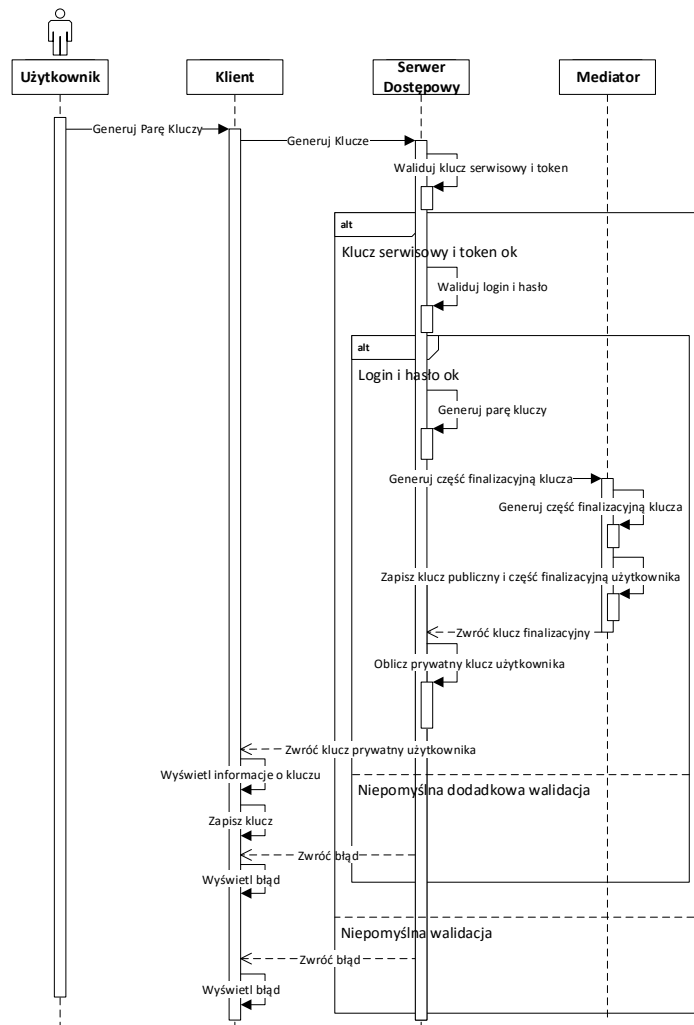
Proces generowania kluczy odbywa się w oparciu o model matematyczny przedstawiony w rozdziale 3. Nowe klucze generowane są na żądanie użytkownika. W aplikacji klienckiej żądanie generowania pary kluczy to jedyne żądanie oprócz logowania, które w celu dodatkowego zabezpieczenia, wymaga podania loginu i hasła. Żądanie generowania kluczy wysyłane jest do serwera dostępowego. Serwer po pomyślnej autoryzacji użytkownika generuje nową parę kluczy: klucz publiczny oraz klucz prywatny. Klucz publiczny oraz klucz serwisowy wysyłany jest następnie do mediatora. Mediator na podstawie długości bitowej klucza publicznego, własnego klucza prywatnego oraz wartości losowej generuje w oparciu o swój algorytm prywatny klucz finalizujący. Mediator zapisuje klucz publiczny użytkownika oraz prywatny klucz finalizujący, następnie prywatny klucz finalizujący odsyłany jest do serwera dostępowego. Serwer na podstawie wcześniej wygenerowanego klucza prywatnego oraz prywatnego klucza finalizującego oblicza prywatny klucz użytkownika. Para kluczy składająca się z Prywatnego klucza użytkownika oraz wcześniej wygenerowanego klucza publicznego odsyłana jest do klienta jako odpowiedź na asynchroniczne zapytanie REST. Klient po zakończeniu żądania zapisuje swoją parę kluczy do plików. W przypadku generowania ponownie pary kluczy, stara para zostaje nadpisana na mediatorze oraz w aplikacji klienta. W takim przypadku, należy zabezpieczyć istniejący plik zawierający klucz publiczny w celu późniejszej weryfikacji plików podpisanych przy pomocy odpowiadającemu mu kluczowi prywatnemu. Rysunki 12 i 13 przedstawiają proces generowania pary kluczy.

Diagram aktywności dla generowania pary kluczy



Rysunek 12: Diagram aktywności - generowanie pary kluczy.

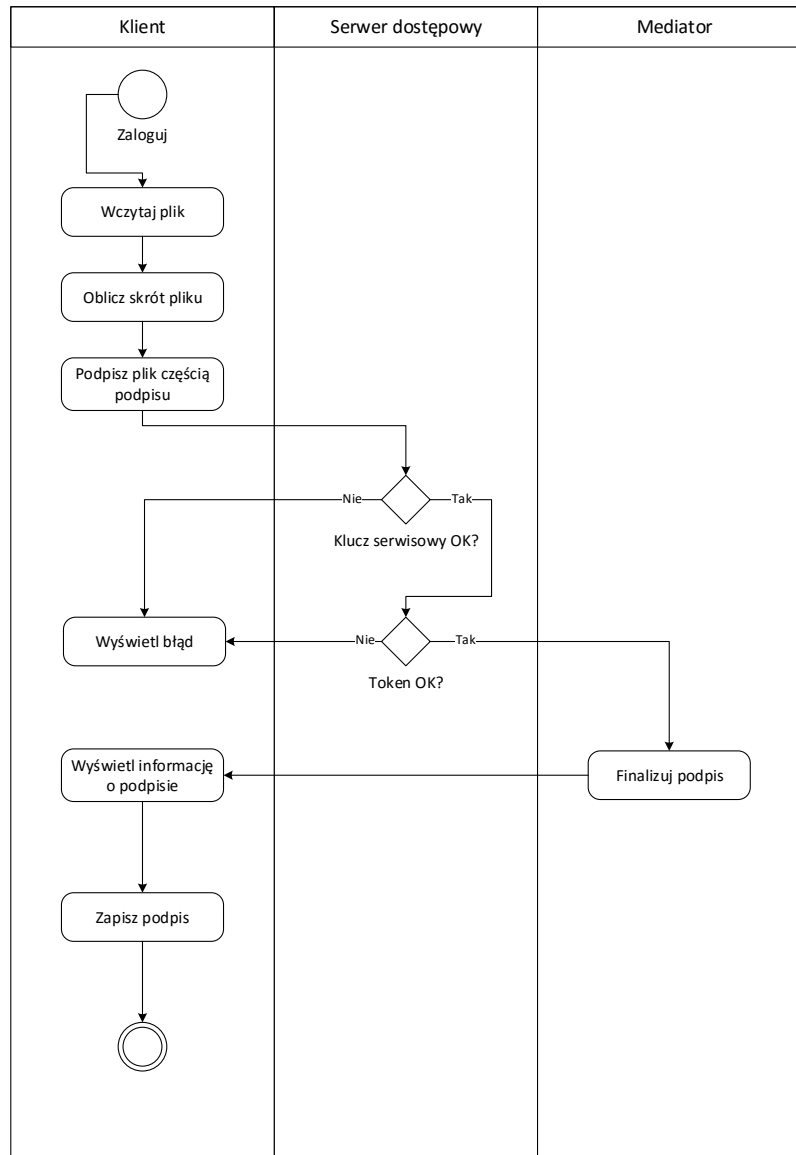
Diagram sekwencji dla generowania pary kluczy



Rysunek 13: Diagram sekwencji - generowanie pary kluczy.

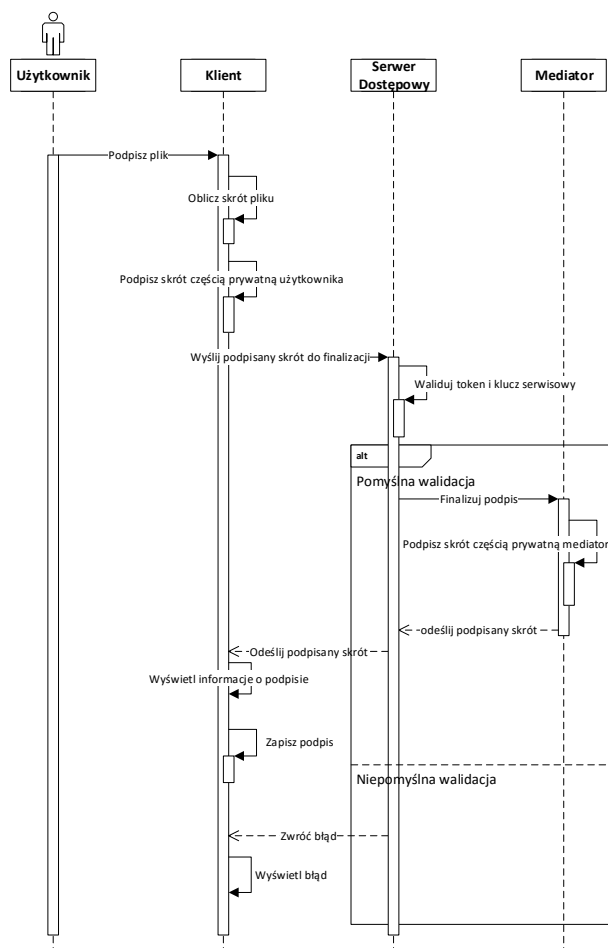
4.2.5 Podpisywanie pliku

Proces podpisywania pliku również odbywa się w oparciu o model matematyczny przedstawiony w rozdziale 3. Aby podpisać plik użytkownik powinien posiadać wcześniej wygenerowaną parę kluczy. Następnie plik do podpisu powinien zostać wczytany w aplikacji klient. Z załadowanego wcześniej pliku zostaje obliczony skrót kryptograficzny SHA512. Skrót ten jest następnie podpisywany kluczem prywatnym użytkownika. Częściowo podpisany plik oraz jego skrót wysyłane są w żądaniu podpisu do serwera dostępowego. Nagłówek żądania zawiera klucz serwisowy i token. Serwera dostępowy po pomyślnym zautoryzowaniu użytkownika przesyła do mediatora otrzymany podpis i skrót oraz klucz serwisowy użytkownika. Mediator podpisuje otrzymany skrót przy pomocy prywatnego klucza finalizującego, a następnie składa go z podpisem dokonany przez klienta. Złożony podpis odsyłany jest jako odpowiedź asynchroniczna do serwera dostępowego, który odsyła go do klienta. Podpis zapisywany jest automatycznie do pliku. Proces podpisywania pliku został przedstawiony na rysunkach 14 i 15

Diagram aktywności dla podpisywania pliku

Rysunek 14: Diagram aktywności - podpisywanie pliku.

Diagram sekwencji dla podpisywania pliku



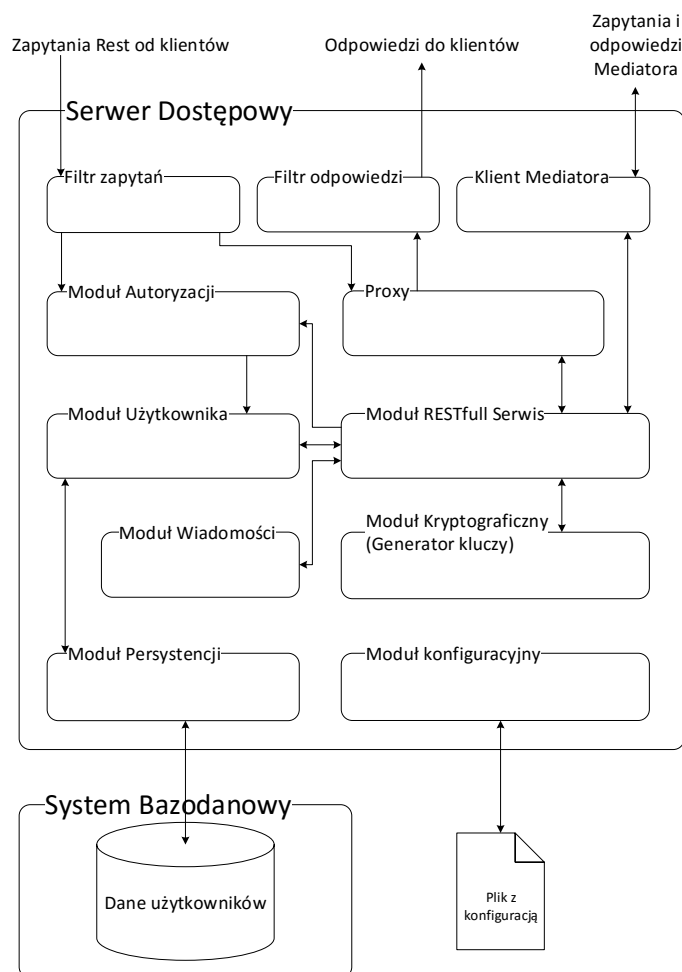
Rysunek 15: Diagram sekwencji - podpisywanie pliku.

4.2.6 Weryfikacja podpisu

Aby zweryfikować podpisany plik należy wczytać go w aplikacji klienckiej, następnie należy wczytać podpis pliku oraz klucz publiczny należący do użytkownika, który ten plik podpisał. Weryfikacja następuje wtedy w całości po stronie klienta. Proces ten został dokładnie przedstawiony w projekcie aplikacji klienta.

4.3 Projekt serwera dostępowego

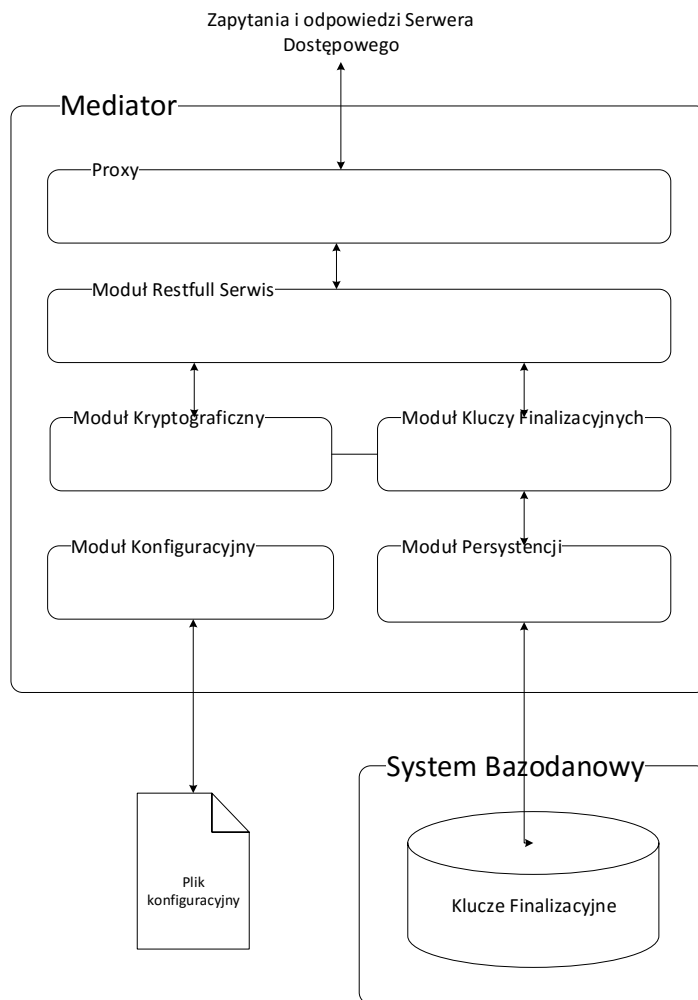
4.3.1 Komponenty serwera dostępowego



Rysunek 16: Interakcje między komponentami serwera dostępowego.

4.4 Projekt serwera mediatora

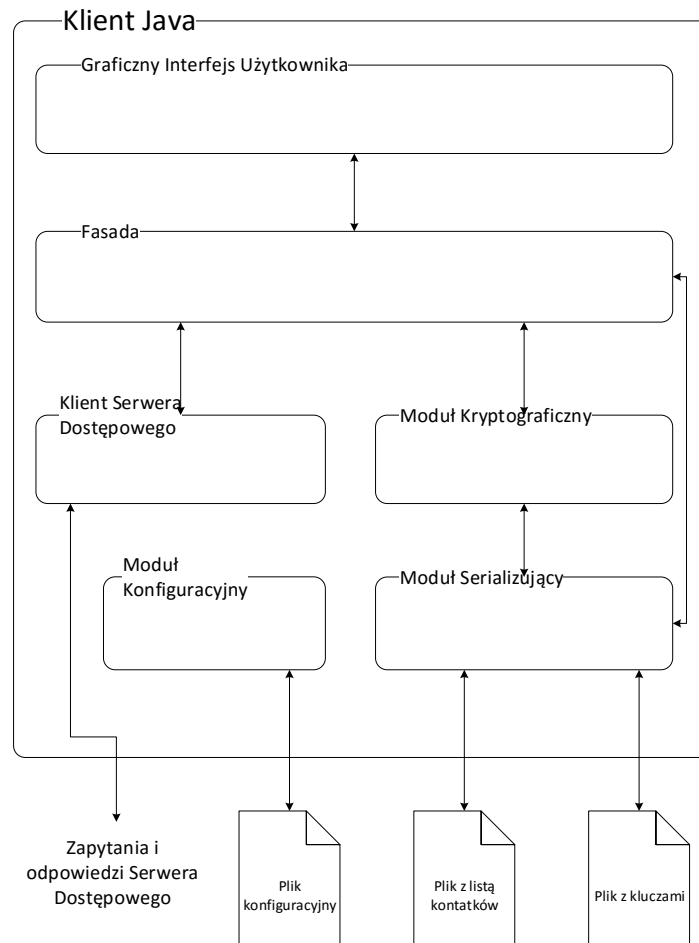
4.4.1 Komponenty mediatora



Rysunek 17: Interakcje między komponentami mediatora.

4.5 Projekt aplikacji klienta

4.5.1 Komponenty klienta



Rysunek 18: Interakcje między komponentami Klienta.

5 Implementacja systemu

5.1 Implementacja serwera dostępowego

5.1.1 Realizacja modułu ...

5.1.2 Realizacja filtrów zapytań

```

1  package net.ddns.falcoboss.registrationserver.security;
2
3
4  import java.io.IOException;
5  import java.util.logging.Logger;
6
7  import javax.ejb.EJB;
8  import javax.ejb.Local;
9  import javax.ejb.Stateless;
10 import javax.ws.rs.container.ContainerRequestContext;
11 import javax.ws.rs.container.ContainerRequestFilter;
12 import javax.ws.rs.container.PreMatching;
13 import javax.ws.rs.core.Response;
14 import javax.ws.rs.ext.Provider;
15
16 import net.ddns.falcoboss.common.HTTPHeaderNames;
17
18 @Provider
19 @PreMatching
20 @Local
21 @Stateless
22 public class RequestFilter implements ContainerRequestFilter {
23
24     private final static Logger log = Logger.getLogger(RequestFilter.class.getName
25         ());
26
27     @EJB
28     AuthenticatorBean authenticatorBean;
29
30     @Override
31     public void filter(ContainerRequestContext requestCtx) throws IOException {
32         String path = requestCtx.getUriInfo().getPath();
33         log.info( "Filtering request path:_" + path );
34         String serviceKey = requestCtx.getHeaderString( HTTPHeaderNames.SERVICE_KEY )
35             ;
36
37         if (!authenticatorBean.isServiceKeyValid(serviceKey)) {
38             requestCtx.abortWith( Response.status( Response.Status.UNAUTHORIZED ).
39                 build() );
40             return;
41         }
42
43         if (!path.startsWith( "service/login/" )) {
44             String authToken = requestCtx.getHeaderString( HTTPHeaderNames.
45                 AUTH_TOKEN );
46             if (!authenticatorBean.isAuthTokenValid(serviceKey, authToken)) {
47                 requestCtx.abortWith( Response.status( Response.Status.UNAUTHORIZED ).
48                     build() );
49             }
50         }
51     }
52 }

```

Listing 1: RequestFilter.java.

5.1.3 Filtr odpowiedzi

```

1 package net.ddns.falcoboss.registrationserver.security;
2
3 import java.io.IOException;
4 import java.util.logging.Logger;
5
6 import javax.ws.rs.container.ContainerRequestContext;
7 import javax.ws.rs.container.ContainerResponseContext;
8 import javax.ws.rs.container.ContainerResponseFilter;
9 import javax.ws.rs.container.PreMatching;
10 import javax.ws.rs.ext.Provider;
11
12 import net.ddns.falcoboss.common.HTTPHeaderNames;
13
14 @Provider
15 @PreMatching
16 public class ResponseFilter implements ContainerResponseFilter {
17
18     private final static Logger log = Logger.getLogger(ResponseFilter.class.getName
19         ());
20
21     @Override
22     public void filter(ContainerRequestContext requestCtx,
23         ContainerResponseContext responseCtx) throws IOException {
24
25         log.info("Filtering_REST_Response");
26
27         responseCtx.getHeaders().add("Access-Control-Allow-Origin", "*");
28         responseCtx.getHeaders().add("Access-Control-Allow-Credentials", "true");
29         responseCtx.getHeaders().add("Access-Control-Allow-Methods", "GET,POST,DELETE,PUT");
30         responseCtx.getHeaders().add("Access-Control-Allow-Headers",
31             HTTPHeaderNames.SERVICE_KEY + "," + HTTPHeaderNames.AUTH_TOKEN);
32     }

```

Listing 2: RequestFilter.java.

6 Testowanie i ocena jakości systemu

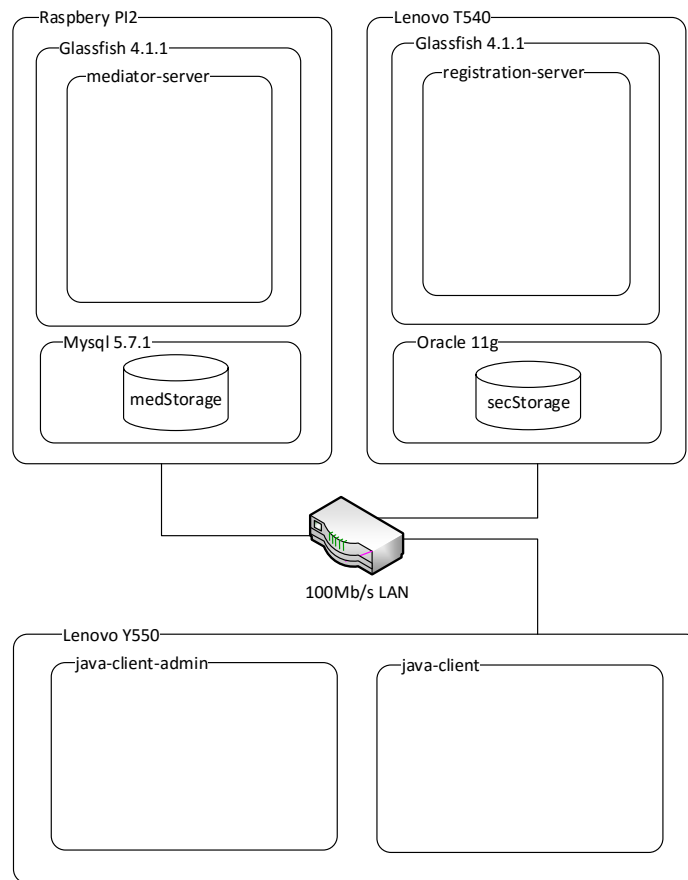
6.1 Instalowanie i konfiguracja systemu

6.1.1 Środowisko testowe

Środowisko testowe składa się z 3 komputerów.

- Raspberry PI 2, 4-rdzeniowy procesor ARM Cortex-A7 900MHz, 1GB RAM;
- Lenovo T540, 4-rdzeniowy procesor i7-4700MQ 2,4GHz, 8GB RAM;
- Lenovo Y550, 2-rdzeniowy procesor Core 2 Duo T6500 2,1GHZ, 4GB RAM;

Mediator uruchomiony został na Raspberry PI2 z baz danych MySQL 5.7. Serwer dostępowy uruchomiony został na Lenovo T540 z baz danych Oracle 11g. Klient i Klient administrator działają na Lenovo Y550. Mediator i Serwer dostępowy uruchamiane są wewnątrz kontenerów hostowanych przez serwer aplikacji Glassfish 4.1.1 Full Platform. Na wszystkich komputerach zainstalowana została Maszyna wirtualna Javy JVM 1.8.0_60-b27. Komputery połączone są za pomocą sieci LAN o maksymalnej przepustowości 100MB/s. Domyślna biblioteka serwera Glassfish 4.1.1 Full Platform odpowiadająca za persystencję obiektów EclipseLink 2.6.1-RC1 zawiera błąd i uniemożliwia poprawne działanie z bazą danych, dlatego wymagana jest aktualizacja biblioteki do wersji 2.6.2. Na rysunku 19 przedstawiona została konfiguracja poszczególnych elementów systemu.



Rysunek 19: Środowisko testowe.

6.1.2 Przygotowanie bazy danych

```

1  -- połączenie z systemem bazodanowym jako root
2  mysql --host=localhost --user=root -p
3
4  -- usuwanie bazy danych jeżeli istnieje
5  mysql> DROP DATABASE IF EXISTS secstorage;
6
7  -- tworzenie bazy danych
8  mysql> CREATE DATABASE secstorage;
9
10 -- połączenie z utworzoną bazą danych
11 mysql> use secstorage;
12
13 -- usuwanie użytkownika
14 mysql> DROP USER 'mainStorage'@'localhost';
15
16 -- utworzenie użytkownika
17 mysql> CREATE USER 'mainStorage'@'localhost' IDENTIFIED BY 'registrationServer1';

```

```
18
19 -- nadanie utworzonemu użytkownikowi wszystkich praw do utworzonej bazy
20 mysql> GRANT ALL PRIVILEGES ON *.* TO 'mainStorage'@'localhost';
```

Listing 3: Tworzenie bazy danych i użytkownika bazy danych

```
1
2 \subsubsection{Konfiguracja serwera Java EE}
3
4 asadmin create-jdbc-connection-pool
5 --datasourceclassname=com.mysql.jdbc.jdbc2.optional.MysqlDataSource
6 --restype=javax.sql.DataSource
7 --property="user=mainStorage:password=registrationServer1:url=
8 jdbc\:mysql\://localhost\:3306/secstorage" mysqlRegPool
9
10 asadmin create-jdbc-resource
11 --connectionpoolid mysqlRegPool jdbc/secstorage
```

Listing 4: Tworzenie zasobu JDBC.

6.2 Realizacja testów

6.2.1 Testy jednostkowe

6.2.2 Testy integracyjne

6.2.3 Testy akceptacyjne

6.3 Ocena wydajności operacji generowania kluczy

6.4 Ocena wydajności transmisji danych

6.5 Ocena zachowania w przypadku awarii

6.6 Ocena bezpieczeństwa

7 Podsumowanie

Literatura

- [1] Arun Gupta. Java EE 6 Leksykon kieszonkowy, Wydawnictwo HELION, Gliwice, 2013. ISBN 978-83-246-6640-9.
- [2] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, Chinmayee Srivathsa. Java EE 6 Przewodnik, Wydanie IV, Wydawnictwo HELION, Gliwice, 2012. ISBN 978-83-246-3847-5.
- [3] Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa. Java EE 6 Zaawansowany Przewodnik, Wydanie IV, Wydawnictwo HELION, Gliwice, 2012. ISBN 978-83-246-7393-3.
- [4] Bhakti Mehta. REST Najlepsze praktyki i wzorce w języku Java, Wydawnictwo HELION, Gliwice, 2015. ISBN 978-83-283-0644-8.
- [5] Andrew S. Tanenbaum. Sieci Komputerowe, Wydawnictwo HELION, Gliwice, 2004. ISBN 83-7361-557-1. Bezpieczeństwo w sieciach komputerowych, s.641-740.
- [6] Bernd Bruegge, Allen H. Dutoit. Inżynieria oprogramowania w ujęciu obiektowym. UML, wzorce projektowe i Java, Wydawnictwo HELION, Gliwice, 2012. ISBN 978-83-246-2872-8.
- [7] William Stallings. Kryptografia i bezpieczeństwo sieci komputerowych. Matematyka szyfrów i techniki kryptologii, Wydawnictwo HELION, Gliwice, 2010. ISBN 978-83-246-2986-2.
- [8] Mirosław Kutylowski, Przemysław Kubiak. Mediated RSA cryptography specification for additive private key splitting (mRSAA), Internet Szkie, 2013 [dostęp 15 grudnia 2015] Dostępne w internecie: <https://tools.ietf.org/html/draft-kutylowski-mrsa-algorithm-03>
- [9] Oracle® Security Developer Tools Reference, Źródło, 2006 [dostęp 15 grudnia 2015] Dostępne w internecie: https://docs.oracle.com/cd/B28196_01/idmanage.1014/b28165/crypto.htm
- [10] Kryptografia asymetryczna i jej zastosowanie w algorytmach komunikacji, Źródło, 2006 [dostęp 29 grudnia 2015] Dostępne w internecie: <http://kis.pwsschelm.pl/publikacje/V/Bartyzel.pdf>
- [11] Systemy rozproszone, Wykład, 2006 [dostęp 28 grudnia 2015] Dostępne w internecie: http://wazniak.mimuw.edu.pl/index.php?title=Systemy_rozproszone

- [12] Zofia Kruczkiewicz. Modelowanie i analiza systemów informatycznych 1, Wykłady, 2014 [dostęp 28 grudnia 2015] Dostępne w internecie: <http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/analizasi/>
- [13] Java Platform, Enterprise Edition: The Java EE Tutorial, Źródło, 2014 [dostęp 15 grudnia 2015] Dostępne w internecie: <https://docs.oracle.com/javaee/7/JEETT.pdf>
- [14] Filters and Handlers Revisited, Źródło, 2011 [dostęp 24 stycznia 2015] Dostępne w internecie: <https://java.net/projects/jax-rs-spec/pages/PreMatchingFilters>
- [15] Wikipedia, Źródło, 2015, [dostęp 24 stycznia 2016] Dostępne w internecie: <http://wikipedia.org>