AKADEMIA NAUK STOSOWANYCH W TARNOWIE

Wydział Politechniczny

Kierunek: *Informatyka*

Specjalność: Informatyka Stosowana

Specjalizacja: Inżynieria systemów informatycznych

2021/2022

Mieszko Maciura

PRACA INŻYNIERSKA

Oprogramowanie do bezpiecznego zarządzania hasłami oraz poufnymi danymi.

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2022

Spis treści

1. Wstęp

Przedmiotem niniejszej pracy dyplomowej jest system zarządzania hasłami składający się z aplikacji webowej, serwerowej i klienckiej. W treści zostaną przedstawione wykorzystane algorytmy, technologie oraz podgląd zaimplementowanych aplikacji.

1.1. Motywacja

W XXI wieku korzystanie z różnych usług internetowych coraz częściej przybiera bezalternatywną formę, stąd rosnąca ilość użytkowników różnych serwisów. Niemalże każdy użytkownik sieci Internet posiada kilkadziesiąt lub kilkaset kont i profili w wielu serwisach internetowych, począwszy od forów o wąskiej tematyce, przez serwisy społecznościowe na usługach finansowych takich jak bankowość elektroniczna kończąc. Konta te często przedstawiają dla użytkowników dużą wartość, stąd istnieje naturalna potrzeba ich ochrony przed nieuprawnionym dostępem.

Aby zapobiegać negatywnym skutkom wycieku danych, a także różnego rodzaju atakom na użytkowników serwisów internetowych powinno się każde konto w sieci zabezpieczać za pomocą innego hasła. W przypadku niestarannego przechowywania tej wrażliwej danej przez podmiot świadczący taką usługę, pojedyncze hasło nie będzie w żaden sposób użyteczne dla potencjalnego przestępcy.

Stosowanie się do powyższej reguły dla przeciętnego użytkownika może być bardzo kłopotliwe, stąd często zdarza się korzystanie z jednego hasła do wielu serwisów i/lub stosowanie słabych haseł. Powszechne praktyki przechowywania haseł pozostawiają wiele do życzenia.

1.2. Cel pracy

Celem pracy było zaprojektowanie i zaimplementowanie systemu, który pozwala na bezpieczne przechowywanie danych użytkownika takich jak loginy i hasła przy zachowaniu prostoty użytkowania, a także zapewnieniu minimalnego wkładu użytkownika w cały proces. Ponadto jednym z priorytetów jest szyfrowanie danych w taki sposób, żeby nawet podmiot przechowujący te dane nie miał w żadnym wypadku dostępu do ich bezpośredniej postaci.

Na rynku istnieje sporo programów i usług o podobnej specyfice i zastosowaniu. Wiele z nich opiera swoją działalność o zaufanie użytkowników poparte na przykład audytami niezależnych ekspertów. Część z usług zapewnia program dostarczający algorytmy dla klienta

oraz przesyłanie zaszyfrowanych baz danych między urządzeniami użytkownika, tak aby miał on do nich dostęp. Kolejny fragment rynku zajmują usługi, które wszystkie niezbędne algorytmy i dane przechowują na swoich serwerach. Oba podejścia mają swoje zalety, ale także i wady. Niniejszy projekt stanowi niejako fuzję, obu rozwiązań co zostanie opisane w kolejnych rozdziałach.

1.3. Zakres pracy

Na projekt inżynierski składa się system złożony z oddzielnie zaimplementowanych, następujących modułów:

- biblioteka do obsługi algorytmów przetwarzających dane,
- aplikacja kliencka (desktopowa),
- aplikacja serwerowa,
- aplikacja internetowa (webowa).

Wymieniona na liście modułów biblioteka służy do przetwarzania danych zgodnie z przeznaczonym do tego algorytmem, a także odpowiada za zapis tych danych do odpowiednio spreparowanego pliku, stąd aplikacja kliencka oraz serwerowa korzystają z tej samej biblioteki. Ponadto aplikacja internetowa również odwzorowuje sposób działania ww. biblioteki w celu zapewnienia przetwarzania wrażliwych danych po stronie klienta.

2. Postawione wymagania

Podstawową funkcją jest możliwość zapisywania i odczytywania loginów i haseł użytkownika. Aplikacja kliencka (dalej zwana desktopową) powinna zapewniać możliwość tworzenia nowego, zabezpieczonego hasłem pliku służącego do przechowywania haseł (dalej zwanego sejfem) i wszelkich danych potrzebnych do ich bezpiecznego odszyfrowania. Aplikacja desktopowa powinna także umożliwiać odtwarzanie sejfu, weryfikację poprawności hasła, możliwość dodawania nowych par loginów i haseł (dla uproszczenia dalej zwanymi dokumentami sejfu), a także dodawania nowych pól na dane w każdym dokumencie sejfu.

Podstawowy dokument sejfu powinien domyślnie zawierać pola login i hasło. Aplikacja desktopowa powinna zapewnić również modyfikację tych pól oraz poszerzanie je o nowe, zdefiniowane przez użytkownika (np. pytanie weryfikacyjne, numer pin, klucz tokena itd.).

Aplikacja internetowa powinna przede wszystkim zapewnić możliwość tworzenia konta w serwisie internetowym, logowania się na to konto. Po zalogowaniu użytkownik powinien uzyskać dostęp do takich samych funkcjonalności jak w przypadku aplikacji desktopowej, przy czym sejfy nie mają być zapisywane lokalnie, na urządzeniu użytkownika, lecz w plikach umieszczonych na serwerze. Dodatkowo aplikacja internetowa powinna zapewnić możliwość pobrania sejfu, aby można go było odtworzyć przy pomocy aplikacji desktopowej.

Podstawowym zadaniem biblioteki do obsługi algorytmów przetwarzających dane (dalej zwanej rdzeniem) jest zapewnienie bezpieczeństwa wprowadzanych i odczytywanych przez użytkownika danych.

3. Mechanizm zabezpieczania danych

Zabezpieczanie danych użytkownika opiera się w głównej mierze o szyfrowanie symetryczne AES-CBC przy pomocy 128-bitowego klucza (dalej zwany kluczem głównym sejfu) i z zastosowaniem wektora inicjalizacyjnego o takiej samej długości. Wspomniany klucz jest generowany całkowicie losowo podczas tworzenia nowego sejfu. Natomiast każdy dokument sejfu ma oddzielny, również generowany losowo wektor inicjalizacyjny. Za pomocą klucza głównego i odpowiedniego wektora program wykonuje operacje szyfrowania i odszyfrowywania danych zapisanych w dokumencie sejfu z wykorzystaniem algorytmu AES-CBC. Klucz główny jest zapisywany w sejfie w postaci zaszyfrowanej przy użyciu podobnej operacji, jednak z wykorzystaniem innego klucza (dalej zwanego kluczem dostępu). Klucz dostępu jest generowany przy pomocy algorytmu iteracyjnego PBKDF2 z haszowaniem HmacSHA1. Na wejście algorytmu podaje się hasło (dalej zwane hasłem dostępu) oraz losowy wektor inicjalizacyjny o długości 128-bitów. W tym rozdziale przedstawione zostaną procesy, które odpowiadają za zabezpieczenie danych. Przetwarzanie danych może się nieznacznie różnić w zależności czy omawiana jest aplikacja desktopowa, czy internetowa.

W przypadku aplikacji internetowej wszelkie dane pobierane są z serwera, a w przypadku aplikacji desktopowej bezpośrednio z pliku.

3.1. Tworzenie sejfu

Proces zaczyna się od dwukrotnego przyjęcia od użytkownika hasła dostępu. Jeżeli hasła są zgodne program generuje trzy losowe wartości:

- UUID – unikatowy identyfikator sejfu,
- 128-bitowy wektor inicjalizacyjny,
- 128-bitowy klucz główny.

Następnie wektor inicjalizacyjny wraz z hasłem jest przekazywany do funkcji realizującej algorytm PBKDF2 z haszowaniem HmacSHA1. Liczba iteracji jest ustalana arbitralnie – w tym przypadku jest to tysiąc. Funkcja ta zwraca klucz dostępu.

W następnym kroku za pomocą klucza dostępu program szyfruje algorytmem AES-CBC identyfikator sejfu i klucz główny. Zaszyfrowany identyfikator stanowi wiadomość weryfikacyjną potrzebną podczas uzyskiwania dostępu do sejfu.

Ostatecznie program zapisuje dane w nowym pliku w standardzie SQLite.

3.2. Uzyskiwanie dostępu do sejfu

Uzyskanie dostępu rozpoczyna się od podania hasła dostępu. Po podaniu hasła program odczytuje z pliku wektor inicjalizacyjny, który wraz z hasłem przekazywany jest do funkcji algorytmu PBKDF2 z takimi samymi parametrami jak podczas tworzenia sejfu. W ten sposób uzyskuje się klucz dostępu, którego poprawność zależy od wprowadzonego hasła. Następnie w celu weryfikacji klucza program próbuje za jego pomocą odszyfrować wiadomość weryfikacyjną. Jeżeli odszyfrowany ciąg znaków jest identyczny jak UUID sejfu – oznacza to, że hasło jest poprawne i dostęp do dokumentów sejfu powinien zostać przyznany, a sam klucz pozwoli na odczytanie danych.

Gdyby okazało się jednak, że wynik operacji jest inny niż zamierzony, jest to znak, że wprowadzone hasło jest niepoprawne, o czym użytkownik zostanie powiadomiony.

W ostatnim kroku program za pomocą odtworzonego klucza dostępu odszyfrowuje klucz główny, który jest zapisywany tymczasowo w pamięci.

3.3. Modyfikowanie dokumentów w sejfie

3.3.1. Tworzenie nowych dokumentów

Tworzenie dokumentu zaczyna się od wygenerowania nowego wektora inicjalizacyjnego, powiązanego tylko z tym dokumentem. Następnie przy znanym schemacie wykorzystania algorytmu AES-CBC wykonywane są operacje szyfrowania nazwy tego dokumentu oraz jego pól. Dane o polach zapisane są w formacie przypominającym składnię JSON.

3.3.2. Modyfikowanie dokumentów

Modyfikowanie dokumentów odbywa się poprzez odszyfrowanie nazwy oraz pól przy pomocy algorytmu AES-CBC i przetwarzaniu ich w postaci jawnej. Po zakończonej operacji dane są serializowane, ponownie szyfrowane i zapisywane w pliku.

4. Algorytmy

W tym rozdziale przedstawione zostaną algorytmy związane z kryptografią, które są wykorzystane w projekcie.

4.1. Algorytm szyfrowania symetrycznego AES-CBC

Algorytm AES (ang. Advanced Encryption Standard) jest symetrycznym szyfrem blokowym opartym na algorytmie Rijandaela. Szyfrowanie symetryczne oznacza, że zarówno do szyfrowania, jak i deszyfrowania wykorzystywany jest dokładnie ten sam klucz.[2]

Działanie algorytmu polega na dzieleniu danych wejściowych na bloki o rozmiarze 128 bitów. Bloki te są reprezentowane jako macierze o rozmiarze 4 bajty x 4 bajty. Każdy następny blok jest przetwarzany w oparciu o wynik operacji na poprzednich blokach.

CBC (ang. Cipher Block Chaining) – jest trybem pracy algorytmu opisujący sposób powiązania kolejnych bloków.[3]

4.2. Standard wyprowadzania klucza na podstawie hasła PBKDF2

Standard ten opisuje sposób działania algorytmu, który pozwala na wyprowadzenie klucza kryptograficznego na podstawie hasła. Wykorzystywany algorytm jest algorytmem iteracyjnym, w którym każda kolejna iteracja jest zależna od wyniku działania poprzedniej. Im więcej iteracji tym niższa wydajność, jednak większe bezpieczeństwo. Algorytm wykorzystuje funkcje skrótu oraz hasło i w określony sposób wielokrotnie je przetwarza i tworzy klucz kryptograficzny o zadanej długości. Ogromną zaletą algorytmu jest fakt, że łamanie nawet nieskomplikowanych haseł przy użyciu metody brute force jest wyjątkowo czasochłonne. [4]

5. Analiza technologiczna

W niniejszym rozdziale opisane będą technologie wykorzystane do zaimplementowania projektu.

5.1. Rdzeń aplikacji

Rdzeń aplikacji został w całości napisany w języku Java. Do odczytywania i zapisywania danych wykorzystywany jest standard SQLite

5.1.1. Java

Java jest obiektowym językiem programowania, który został opracowany w 1991 roku przez Sun Microsystems. Składnia Javy została zainspirowana językami C i C++.

Głównym założeniem języka jest sposób jego kompilacji. Aplikacja napisana w języku Java jest kompilowana do kodu bajtowego, który następnie przetwarzany jest przez wirtualną maszynę Javy, która z kolei może działać na wielu systemach. W związku z tym język doskonale nadaje się jako uniwersalne narzędzie.

Dodatkową zaletą języka jest otwarty standard SDK (ang. Software Development Kit), który do celów niekomercyjnych jest zawsze darmowy.

Od 1991 roku język Java pozostaje popularnym językiem, jednak jego zastosowanie ulegało zmianie na przestrzeni lat. Początkowo język był wykorzystywany do tworzenia oprogramowania dla urządzeń osobistych, jednak z czasem jego główne zastosowanie przeszło na stronę aplikacji internetowych.[1]

5.1.2. SQLite

SQLite jest otwartoźródłowym systemem zarządzania relacyjną bazą danych, który obsługuje składnię SQL. Projekt sięga roku 2000 i należy do rodziny projektów dostępnych na licencji domeny publicznej. SQLite umożliwia używanie baz danych w ramach jednego pliku. Technologia ta doskonale nadaje się jako system przechowywania danych w systemach wbudowanych, lub jako metoda do przechowywania bardziej złożonych zbiorów danych. Może stanowić alternatywę dla serializacji obiektów (w przypadku gdy potrzebne do zapisania dane są bardziej złożone). [5]

5.2. Aplikacja desktopowa

Aplikacja desktopowa została napisana w języku Java, z użyciem zestawu narzędzi JavaFX. Do działania aplikacji ważny jest także standard pliku SQLite.

5.2.1. JavaFX

Java FX to technologia umożliwiająca między innymi pisanie aplikacji desktopowych z graficznym interfejsem użytkownika. Zbiór narzędzi JavaFX jest de facto biblioteką języka Java. Jego głównymi zaletami są możliwość stosowania plików XML w celu budowania elementów interfejsu aplikacji, a także ograniczona możliwość stosowania kaskadowych arkuszy stylów. JavaFX w naturalny sposób do działania wymaga wirtualnej maszyny Javy. Podobnie jak inne aplikacje napisane w języku Java, aplikacje w JavaFX można uruchamiać na różnych platformach. [6]

5.3. Aplikacja serwerowa

Aplikacja serwerowa działa z wykorzystaniem rdzenia aplikacji napisanego w całości w języku Java

5.3.1. Maven

Maven jest popularnym narzędziem stosowanym do zarządzania zależnościami w projektach opartych o język Java. Umożliwia dostęp do szerokiego wyboru otwartych bibliotek, a także ułatwia korzystanie z nich. [7] Dodatkowymi funkcjami Mavena są także:

- generowanie dokumentacji,
- budowanie projektów,
- generowanie raportów.

5.3.2. Spring

Spring jest popularnym frameworkiem języka Java. Jest otwartą technologią. Jest szkieletem tworzenia aplikacji Java. Jedną z jego bardziej rozpoznawalnych funkcji jest wstrzykiwanie zależności. W praktyce oznacza to, że w przypadku wykorzystania wstrzykiwania zależności to kontener Spring tworzy powiązania między komponentami i zarządza cyklem życia obiektu. Programista ma kontrolę nad tym procesem dzięki wykorzystaniu adnotacji. Rozwiązanie to pozwala na przekazywanie nowo powstałemu obiektowi jego zależności bezpośrednio z kontenera, co przekłada się na fakt, że obiekty same nie muszą tworzyć tych zależności. [8]

5.3.3. Spring Boot

Spring Boot jest nakładką na framework Spring, której zadaniem jest ułatwienie tworzenia aplikacji w oparciu o Spring. [8]

5.3.4. Spring Data JPA

Spring Data JPA to moduł Springa działający w oparciu o standard JPA. Komponent ten odpowiada za dostęp do bazy danych. Jego głównym zadaniem jest ustandaryzowanie dostępu do danych wewnątrz projektu. Dzięki wykorzystaniu JPA można w stosunkowo prosty sposób wykonywać operacje na bazie danych. Sam moduł posiada wbudowane metody, które wykonują operację na bazach danych. Jedną z bardziej rozpoznawalnych funkcjonalności tego rozwiązania jest przekształcanie nazw funkcji w konkretne zapytania SQL. [8]

5.3.5. Spring Security

Spring Security jest modułem Springa odpowiedzialnym za zabezpieczenie aplikacji. Wykorzystywany jest do autoryzacji użytkowników. Konfiguracja Spring Security umożliwia zabezpieczenie endpointów i innych zasobów przed nieautoryzowanym dostępem. W Spring Security można zdefiniować sposób autoryzacji użytkowników, blokować i zezwalać na dostęp do konkretnych zasobów w zależności np. od roli użytkownika lub adresu IP, z którego przychodzi zapytanie. [8]

5.3.6. JSON Web Token

JSON Web Token jest otwartym standardem, który definiuje sposób wymiany danych między stronami. W uproszczeniu służy do autoryzacji użytkownika. Zastosowanie tej technologii coraz częściej wypiera rozwiązanie wymiany danych w aplikacjach klient-serwer w oparciu o sesję. Dzięki wykorzystaniu technologii JWT „ciężar" autoryzacji spoczywa po stronie klienta, który przy każdorazowym zapytaniu wysyła do serwera identyfikujący go token. Serwer weryfikuje ważność tokena i w zależności od statusu tej weryfikacji zwraca dane, które zostały zażądane przez klienta lub odmówi autoryzacji.[9]

5.3.7. Java Persistance API

Java Persistance API jest narzędziem do mapowania obiektowo – relacyjnego. Umożliwia przekształcanie obiektów języka Java na encje bazy danych. Programista ma kontrolę nad tym procesem poprzez stosowanie adnotacji w klasie, która ma reprezentować encje.[8]

5.3.8. PostgresSQL

PostgresSQL jest otwartoźródłowym systemem relacyjnych baz danych. PostgresSQL jest relacyjno-obiektową bazą danych, która została opracowana w 1995 roku na Uniwersytecie Kalifornijskim. Podobnie jak w klasycznej definicji relacyjnej bazy danych jej struktura składa się z tabel, a każda tabela składa się z kolumn, które odpowiadają za atrybuty. Pomiędzy tabelami mogą występować relacje. Relacje te polegają na połączeniu kluczem obcym będący unikalnym atrybutem tabeli z innym atrybutem (najczęściej kluczem głównym). Ogromną zaletą tego systemu bazodanowego jest możliwość definiowania własnych typów danych oraz funkcji, a także spore możliwości skalowania.[10]

5.4. Aplikacja internetowa

Aplikacja internetowa została zaimplementowana z użyciem następujących technologii:

- HTML,
- CSS,
- JavaScript,
- React.js,
- CryptoJS.

5.4.1. HTML i CSS

HTML (ang.HyperText Markup Language) jest językiem znaczników stosowanym przede wszystkim do opisywania struktury dokumentów hipertekstowych, w szczególności stron internetowych. Głównym zadaniem języka jest nadanie znaczenia semantycznego kolejnym fragmentom tekstu. Początki języka sięgają 1980 roku, a swoje zastosowanie znajduje do dzisiaj. Obecnie najpopularniejszą wersją języka jest opatrzona symbolem HTML5.

CSS (ang. Cascading Style Sheets) jest z kolei językiem służącym do opisu formy prezentacji. Swoje zastosowanie ma między innymi w opisywaniu stylów znaczników HTML. Arkusz stylów, czyli dokument języka CSS jest listą dyrektyw, czyli reguł traktujących o formie wyświetlanych treści (np. wielkość, kolor czcionki itp.). Różne formy stylów CSS są wyjątkowo powszechnym narzędziem wśród programistów, dzięki możliwości ich uniwersalnego stosowania. [11]

5.4.2. JavaScript

JavaScript to skryptowy język programowania, który został opracowany w 1995 roku w celu zapewnienia możliwości wprowadzenia pewnej dynamiki do statycznych stron internetowych. Przede wszystkim miał odpowiadać za dynamiczną zmianę stylów, warunkowe pokazywanie lub ukrywanie zawartości oraz dostarczanie niewielkiej ilości logiki na stronach internetowych. Tę formę język zachował przez bardzo długi okres, jednak dynamiczny rozwój i poszerzanie jego możliwości spowodowało, że znalazł on zastosowanie w innych dziedzinach. Dzisiaj za pomocą tego języka można programować aplikacje dostępne na komputery, smartfony, serwery i wiele innych. Dla języka JavaScript powstała ogromna ilość bibliotek i frameworków ułatwiających tworzenie różnego rodzaju aplikacji, w tym aplikacji internetowych. Najpopularniejszymi są: React, Angular, Vue.

5.4.3. React.js

React.js jest frameworkiem języka JavaScript, którego głównym założeniem jest tworzenie „Single Page Applications” czyli aplikacji webowych opartych o jedną stronę. Ideą tego podejścia jest (w odróżnieniu od klasycznych aplikacji internetowych) konieczność pobrania plików strony z serwera tylko raz. Zasada działania jest następująca – zapytanie przeglądarki kierowane pod konkretną podstronę jest przechwytywane przez silnik React, ten następnie przetwarza żądanie i reaguje w odpowiedni sposób, na przykład wyświetlając treść danej podstrony. Dodatkowo React zapewnia relatywnie prostą i przejrzystą budowę aplikacji internetowych dzięki możliwości rozłożenia kodu na przenikające się komponenty. [12]

5.4.4. CryptoJS

CryptoJS jest biblioteką JavaScript zapewniającą dostęp do funkcji odpowiadających za operacje kryptograficzne. Biblioteka dostarcza implementacje popularnych algorytmów i umożliwia wykonywanie ich z parametrami zadanymi przez programistę lub użytkownika.

6. Implementacja aplikacji internetowej

6.1. Opis funkcji

Aplikacja internetowa posiada dwóch aktorów:

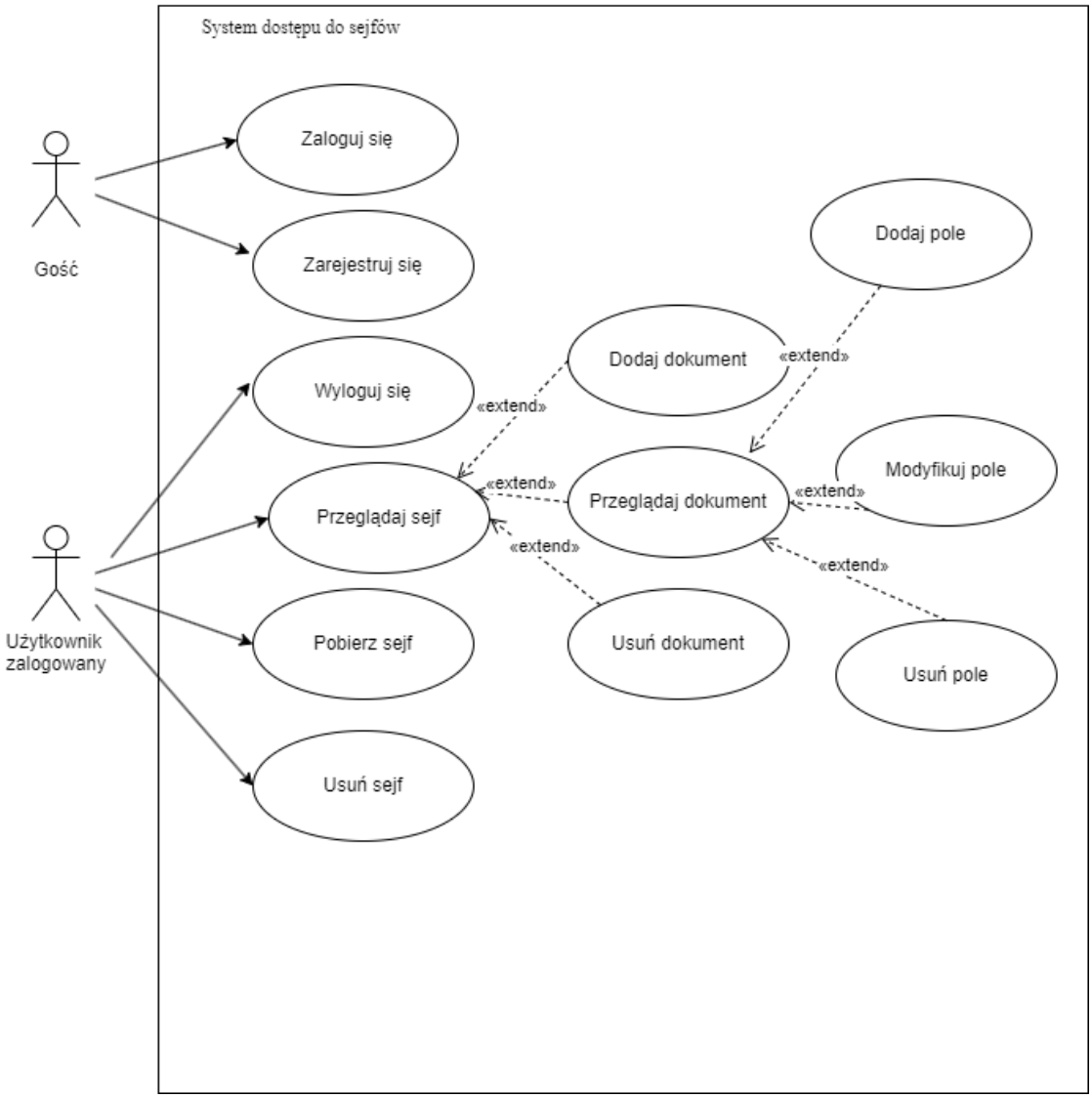
- użytkownik niezalogowany (gość),
- użytkownik zalogowany

Goście nie mają możliwości skorzystania z żadnej funkcji serwisu z wyjątkiem rejestracji i logowania. Po pomyślnej rejestracji gość jest przekierowywany na stronę logowania, a po pomyślnym logowaniu na stronę domową gdzie uzyskuje dostęp do funkcjonalności serwisu.

Użytkownicy zalogowani mają do dyspozycji stronę główną, na której widoczna jest lista ich sejfów, a także formularz, który umożliwia utworzenie nowego pliku sejfu, nadanie mu hasła oraz nazwy. Na liście widoczne są pliki użytkownika, które użytkownik może otworzyć, pobrać lub usunąć. W przypadku wybrania sejfu użytkownik jest przekierowywany na podstronę, na której weryfikowane jest hasło. W przypadku pomyślnej autoryzacji użytkownik zyskuje dostęp do sejfu, który następnie może dowolnie modyfikować poprzez:

- dodawanie nowych dokumentów,
- usuwanie dokumentów,
- modyfikowanie pól dokumentów,
- dodawanie nowych pól w dokumentach,
- usuwanie pól w dokumentach.

6.2. Diagram Przypadków użycia



Rys. 6.1 Diagram przypadków użycia [opracowanie własne]

7. Implementacja aplikacji serwerowej

7.1. Opis funkcji

Aplikacja serwerowa ma za zadanie przyjmować zapytania od aplikacji internetowej. Jej podstawowym zadaniem jest umożliwienie autoryzacji użytkowników i przesyłanie ich danych. Aplikacja serwerowa realizuje następujące funkcje aplikacji internetowej:

Dla użytkowników nieautoryzowanych:

- rejestracja użytkownika (tj. zapisanie jego loginu i hasła poddanemu haszowaniu w bazie danych),
- logowanie użytkownika (tj. wygenerowanie i wysłanie tokena JWT użytkownikowi, który przesłał prawidłowe dane logowania),
- weryfikacja tokena (sprawdzenie czy token jest ważny oraz ustalenie właściciela).

Dla użytkowników autoryzowanych:

- przesyłanie nazw, numerów UUID i numerów ID sejfów skojarzonych z użytkownikiem,
- przesyłanie danych konfiguracyjnych sejfu (zaszyfrowana wiadomość weryfikacyjna, zaszyfrowany klucz główny),
- przesyłanie listy nazw dokumentów sejfu w postaci zaszyfrowanej wraz z ich unikalnym ID,
- przesyłanie wszystkich danych dokumentów sejfu w postaci zaszyfrowanej,
- rejestrowanie nowego UUID sejfu i skojarzenie go z aktualnym użytkownikiem,
- zapisywanie otrzymanych danych konfiguracyjnych sejfu (zaszyfrowanej wiadomości weryfikacyjnej, zaszyfrowanego klucza głównego),
- zapisywanie zaktualizowanych i nowo utworzonych dokumentów sejfu przesłanych przez użytkownika.

Przez użytkownika autoryzowanego rozumie się użytkownika, który wraz z zapytaniem przekazał ważny token JWT, na którego podstawie odbywa się weryfikacja tożsamości.

Serwer realizując większość zapytań zalogowanych użytkowników wykonuje operacje odczytu i zapisu na plikach w standardzie SQLite znajdujących się w jego systemie plików.

7.2. REST API

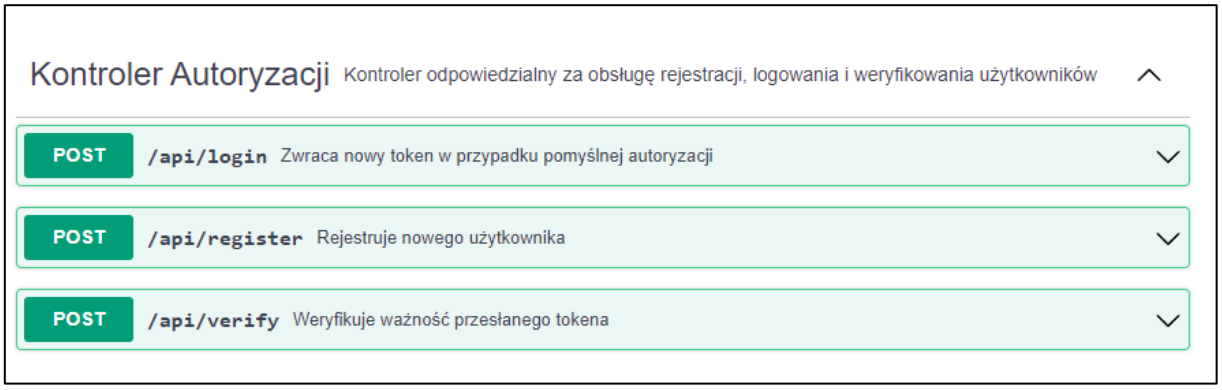
Serwer przetwarza zapytania od aplikacji webowej na podstawie REST API (ang. Representational State Transfer Application Programming Interface). Podstawą tego rozwiązania jest obsługiwanie zapytań przez kontrolery będące endpointami, do których aplikacje klienta mogą się łączyć przez protokół HTTP. Wyróżnia się cztery metody zapytań REST, z których każda nawiązuje do charakteru zapytania:

- GET – pobranie zasobów,
- POST – utworzenie nowego zasobu,
- PUT – modyfikowanie istniejącego zasobu,
- DELETE – usunięcie istniejącego zasobu

Aplikacja serwera będąca przedmiotem omawianego projektu posiada trzy kontrolery. Jeden dotyczy obsługi użytkowników, drugi obsługi konfiguracji sejfów, natomiast trzeci dotyczy obsługi samych danych zawartych w sejfach. Poniżej zaprezentowane zostaną najważniejsze endpointy obsługiwane przez serwer.

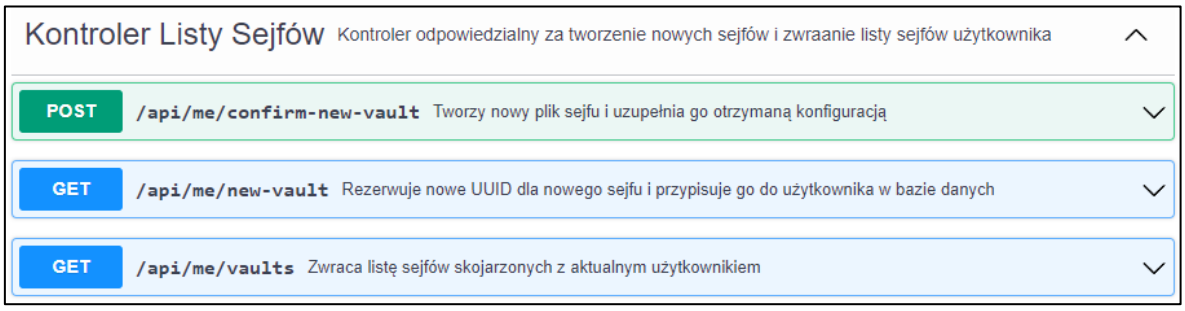
7.2.1. Kontroler autoryzacji

Na rysunku 7.1 przedstawiony jest schemat endpointów wygenerowany za pomocą narzędzia Swagger [13]



Rys. 7.1 Kontroler autoryzacji [opracowanie własne]

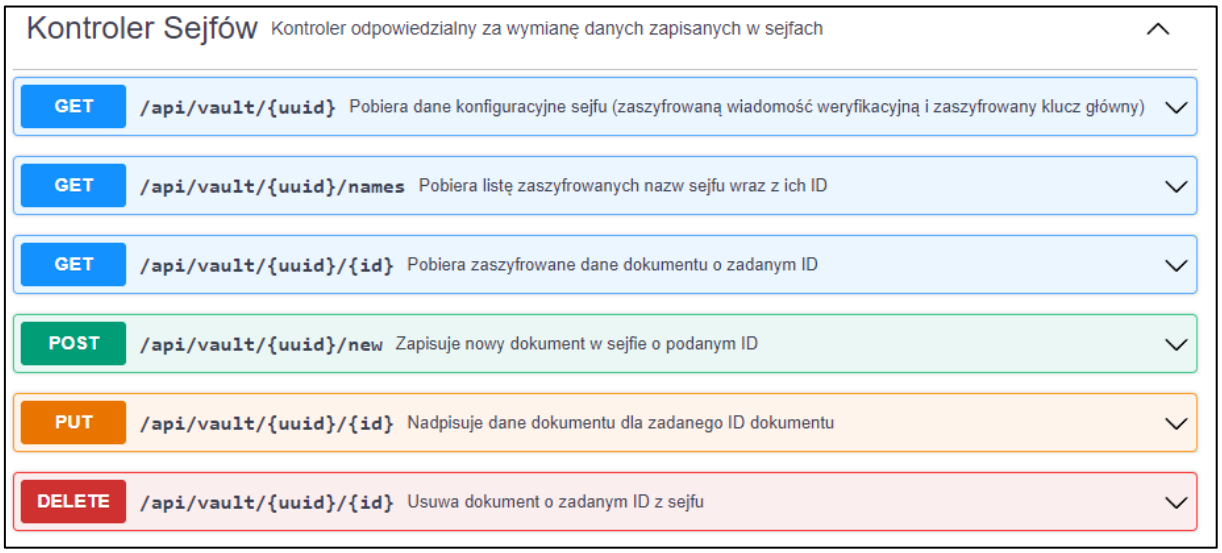
7.2.2. Kontroler listy sejfów



Rys. 7.2 Kontroler listy sejfów [opracowanie własne]

7.2.3. Kontroler sejfów

Kontroler sejfów przy każdym zapytaniu dodatkowo weryfikuje czy zapytanie jest zgodne z wymaganiami, tzn. czy użytkownik, który wysłał zapytanie jest uprawniony do otrzymania danych sejfu, który wskazał.



Rys. 7.3 Kontroler sejfów [opracowanie własne]

8. Implementacja aplikacji desktopowej

8.1. Opis funkcji

Aplikacja desktopowa zapewnia możliwość tworzenia i obsługi plików sejfów znajdujących się na komputerze użytkownika. Po uruchomieniu aplikacji dostępne są dwie możliwości:

- Utworzenie nowego sejfu,
- Otwarcie istniejącego sejfu.

Utworzenie nowego sejfu otwiera okno dialogowe, w którym użytkownik powinien wskazać lokalizację gdzie chce zapisać plik sejfu. Po poprawnym wybraniu lokalizacji aplikacja prosi użytkownika o dwukrotne podanie hasła. Po stwierdzeniu ich zgodności rdzeń aplikacji rozpoczyna procedurę opisaną w rozdziale trzecim.

Otwarcie istniejącego sejfu również rozpoczyna się od okna dialogowego, w którym użytkownik musi wskazać plik sejfu. Po prawidłowym wskazaniu pliku użytkownik proszony jest o podanie hasła do sejfu. Hasło i wszystkie potrzebne dane przetwarzane są przez rdzeń aplikacji w sposób opisany w rozdziale trzecim.

Aplikacja desktopowa umożliwia obsługę nowych sejfów, a także tych utworzonych za pomocą interfejsu aplikacji internetowej, dzięki możliwości pobrania plików z sejfu z serwera na lokalizację lokalną.

9. Implementacja rdzenia aplikacji

Rdzeń aplikacji jest programem – biblioteką, z której korzysta zarówno serwer, jak i aplikacja desktopowa (w różnym zakresie).

9.1. Zadania rdzenia w kontekście aplikacji serwerowej.

W tym przypadku rdzeń odpowiada jedynie za operacje odczytu i zapisu do pliku sejfu. Aplikacja serwerowa przekazuje do rdzenia UUID sejfu oraz wskazuje dane, które muszą być odczytane lub przekazuje dane do zapisania. W powyższym kontekście rdzeń aplikacji w żaden sposób nie modyfikuje danych otrzymanych przez aplikację serwerową, a jedynie pełni rolę sterownika zapisu i odczytu z lub do właściwego pliku.

9.2. Zadanie rdzenia w kontekście aplikacji desktopowej.

W przypadku aplikacji desktopowej rdzeń oprócz funkcji sterownika sejfów zapewnia również algorytmy do przetwarzania danych. Rdzeń przetwarza dane zgodnie z założeniami i wytycznymi przedstawionymi w rozdziale trzecim niniejszej pracy.

10. Sejf

10.1. Rola sejfu

Rolą sejfu jest przechowywanie informacji dotyczących samego sejfu, jak i przechowywanie danych, które użytkownik w nim zapisuje. Sam sejf jest wyłącznie plikiem zapisanym w standardzie SQLite.

10.2. Struktura sejfu

Każdy sejf składa się z dwóch tabel – tabeli zawierającej konfigurację sejfu, oraz tabeli przechowującej dokumenty sejfu, czyli właściwe dane.

10.2.1. Tabela konfiguracji

Tabela zawierająca konfigurację zawiera w sobie tylko jeden rekord o następujących atrybutach:

- UUID – generowany losowo unikalny numer identyfikacyjny,
- wiadomość weryfikacyjna – zaszyfrowany numer UUID zapisany w formacie Base64,
- wektor inicjalizacyjny – zapisany jawnie w formacie heksadecymalnym,
- klucz główny – zaszyfrowany klucz główny zapisany w formacie Base64.

Wiadomość weryfikacyjna po odszyfrowaniu ma format tekstu jawnego w formacie UTF-8, czyli dokładnie taki jaki ma numer UUID.

Klucz główny po odszyfrowaniu ma postać heksadecymalną.

10.2.2. Tabela danych

Tabela danych składa się z następujących kolumn:

- id – identyfikator rekordu,
- wektor inicjalizacyjny – odpowiedni tylko dla jednego rekordu wektor inicjalizacyjny zapisany tekstem jawnym w postaci heksadecymalnej,
- nazwa – zaszyfrowana nazwa dokumentu zapisane w formacie Base64,
- dane – zaszyfrowana zawartość dokumentu zapisane w formacie Base64.

10.3. Metoda przechowywania danych w dokumentach wewnątrz sejfu

Dane w dokumentach są przechowywane w formacie JSON. Nadrzędny obiekt zawiera obiekty, które opatrzone są nazwami pól tekstowych dokumentu (np. „login” lub „hasło”). Każdy z tych podrzędnych obiektów ma dwa pola: *fieldContent* oraz *fieldType*. Pole *fieldContent* zawiera wartość zapisaną w polu tekstowym, natomiast pole *fieldType* zawiera wartość „TEXT” lub „PASSWORD”, dzięki czemu można różnicować metody wyświetlania różnych typów danych zapisanych w dokumencie.

Całość tworzy łańcuch znakowy, który jest szyfrowany zgodnie z opisywaną wcześniej procedurą i zapisywany jako atrybut danych dokumentu w formacie Base64.

11. Testy

Działanie systemu zostało przetestowane za pomocą testów jednostkowych. Dodatkowo endpointy zostały przetestowane przy użyciu oprogramowania Postman, w którym zostały przygotowane kolekcje danych zestawionych w celu przetestowania odpowiedzi serwera.

11.1. Testy jednostkowe

Test jednostkowy jest testem, który sprawdza poprawność działania jednostki. Jednostką w tym kontekście jest najczęściej nazywana funkcja lub metoda, które realizuje określone zadanie. Do testów jednostkowych elementów zaimplementowanych w języku Java wykorzystano bibliotekę JUnit.

11.2. Testy REST API

W ramach testów REST API przygotowana została kolekcja, która po kolei sprawdza działanie różnych scenariuszy, między innymi:

- rejestracja i logowanie,
- otrzymywanie tokena JWT
- weryfikacja uprawnień do sejfów,
- przekazywanie danych konfiguracyjnych sejfów,
- przekazywanie nowych i zaktualizowanych dokumentów sejfów,
- otrzymywanie danych konfiguracyjnych sejfów,
- otrzymywanie danych dokumentów.

11.3. Test odczytu danych konfiguracyjnych sejfu

Na rysunku 11.1 został przedstawiony test odczytywania danych konfiguracyjnych z pliki SQLite. Do tego celu wykorzystano odpowiednio przygotowany plik z przykładowymi danymi. Prawidłowy przebieg testu oznacza, że wszystkie wywołane metody działają bez zarzutu, a dane odczytywane są prawidłowo. Gdyby test się nie powiódł, a wymagany plik znajdowałby się w odpowiedniej lokalizacji, wskazywało by to na problem z działaniem wywoływanych metod.

```
19      @Test
20      void getVaultConfig() {
21          VaultDriverService service = new VaultDriverService(); //Given
22          try {
23              VaultConfig response = service.getVaultConfig(TEST_DB_URL); //When
24              //Then
25              assertEquals(TEST_DB_CHECK_MESSAGE, response.getCheckMessage());
26              assertEquals(TEST_DB_ID, response.getId());
27              assertEquals(TEST_DB_INITIALIZATION_VECTOR, response.getIv());
28              assertEquals(TEST_DB_MASTER_KEY, response.getMasterKey());
29          } catch (VaultDoesNotExistException e) {
30              throw new RuntimeException(e);
31          } catch (SQLException e) {
32              throw new RuntimeException(e);
33          } catch (EmptyResponseException e) {
34              throw new RuntimeException(e);
35          }
36
37      }
38  }
```

Rys. 11.1 Test odczytu konfiguracji [opracowanie własne]

12. Wybrane rozwiązania implementacyjne

W tym rozdziale zostaną przedstawione wybrane rozwiązania zastosowane przy implementacji systemu.

12.1. Rdzeń aplikacji

12.1.1. Uzyskiwanie klucza przy pomocy hasła

Na rysunku 12.1 przedstawiony jest fragment implementacji metody odpowiedzialnej za generowanie klucza głównego. Metoda wykorzystuje hasło dostępu w postaci jawnej, wektor

inicjalizacyjny w postaci heksadecymalnej oraz wcześniej zdefiniowane stałe – liczbę iteracji oraz długość klucza wynikowego.

```
4 usages  Mieszko
private void generateKey(){
    PBEKeySpec pbeKeySpec = new PBEKeySpec(this.password, this.iv, this.ITERATIONS, this.KEY_LENGTH);
    try {
        SecretKeyFactory secretKeyFactory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        this.mainKey = secretKeyFactory.generateSecret(pbeKeySpec).getEncoded();
    } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
        e.printStackTrace();
    }
}
```

Rys. 12.1 Wyprowadzanie klucza [opracowanie własne]

12.1.2. Metody odpowiedzialne za symetryczne szyfrowanie danych

Rysunki 12.2 oraz 12.3 przedstawiają metody wykorzystywane do szyfrowania większości danych w ramach sejfu. Do szyfrowania strumieniowego wykorzystywany jest wcześniej zapisane w instancji klasy dokumentu wektor inicjalizacyjny oraz klucz główny. Metoda szyfrująca zwraca szyfrogram zakodowany w Base64, a metoda deszyfrująca tekst jawny.

```
@Override
public String encrypt(String plainText){
    try {
        IvParameterSpec iv = new IvParameterSpec(this.iv);
        SecretKeySpec secretKeySpec = new SecretKeySpec(this.mainKey, "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, iv);

        byte[] encrypted = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encrypted);
    } catch (NoSuchAlgorithmException |
             NoSuchPaddingException |
             InvalidKeyException |
             InvalidAlgorithmParameterException |
             IllegalBlockSizeException |
             BadPaddingException e) {
        e.printStackTrace();
        return "";
    }
}
```

Rys. 12.2 Szyfrowanie tekstu jawnego [opracowanie własne]

```java
public String decrypt(String cipherText){
    try {
        IvParameterSpec iv = new IvParameterSpec(this.iv);
        SecretKeySpec secretKeySpec = new SecretKeySpec(this.mainKey, "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, iv);

        byte[] plainText = cipher.doFinal(Base64.getDecoder().decode(cipherText));
        return new String(plainText);

    } catch (NoSuchAlgorithmException
             | NoSuchPaddingException
             | InvalidKeyException
             | InvalidAlgorithmParameterException
             | IllegalBlockSizeException
             | BadPaddingException e) {
        e.printStackTrace();
        return "";
    }
}
```

Rys. 12.3 Deszyfrowanie kryptogramu [opracowanie własne]

12.1.3. Tworzenie nowego dokumentu

Fragment przedstawiony na rysunku 12.4 pokazuje kod odpowiedzialny za zapisywanie nowego dokumentu w pliku sejfu. Do metody przekazywany jest obiekt reprezentujący gotowy dokument zawierający pola zapisane w postaci jawnej. Dane są szyfrowane i zapisywane w formacie JSON.

```
105     public void createNewEntity(Entity entity) throws NoSuchAlgorithmException,
106             IllegalBlockSizeException, InvalidKeyException, BadPaddingException,
107             InvalidAlgorithmParameterException, NoSuchPaddingException, SQLException {
108         this.currentEntity = entity;
109         Gson gson = new Gson();
110         VaultDataKey vaultDataKey = new VaultDataKey(this.dataKey);
111         this.sqLiteHandler.insertNewEntity(vaultDataKey.getIvAsString(),
112                 vaultDataKey.encrypt(gson.toJson(this.currentEntity.getName())),
113                 vaultDataKey.encrypt(gson.toJson(this.currentEntity.getFields())));
114     }
```

Rys. 12.4 Tworzenie nowego dokumentu [opracowanie własne]

12.1.4. Wczytywanie szyfrogramu

Przedstawiona na rysunku 12.5 metoda odpowiada za wczytywanie dokumentu. Na wejściu przekazywane jest id dokumentu. W pierwszej kolejności pobierany jest właściwy dla dokumentu wektor inicjalizacyjny oraz zapisany w instancji klasy sejfu klucz główny.

Następnie odszyfrowywane są nazwa i pola wraz z wartościami dokumentu. W kolejnym kroku dane są mapowane na pary rekordów składające się z nazw pól, ich typu oraz zawartości. Ostatecznie wszystkie dane są zapisywane w atrybucie klasy sejfu przechowującym obecnie przetwarzany dokument.

```
138     public void loadEntity(int id) throws SQLException,
139             EmptyResponseException, NoSuchPaddingException, InvalidKeyException,
140             NoSuchAlgorithmException, IllegalBlockSizeException, BadPaddingException,
141             InvalidAlgorithmParameterException {
142         Gson gson = new Gson();
143         VaultDataKey vaultDataKey = new VaultDataKey(this.sqLiteHandler.getEntityIv(id), this.dataKey);
144         String plainName = vaultDataKey.decrypt(this.sqLiteHandler.getEntityName(id));
145         String plainData = vaultDataKey.decrypt(this.sqLiteHandler.getEntityData(id));
146         this.currentEntity = new Entity(gson.fromJson(plainName, String.class));
147
148         Map tmpMap =  gson.fromJson(plainData, LinkedHashMap.class);
149
150         Map<String, EntityField> tmpFieldsMap = new LinkedHashMap();
151
152         tmpMap.forEach((k, v) -> {
153             EntityField tmpEntityField;
154             Map tmpFieldMap = (LinkedTreeMap) v;
155             String tmpFieldContent = tmpFieldMap.get("fieldContent").toString();
156             String tmpFieldType = tmpFieldMap.get("fieldType").toString();
157             if(tmpFieldType.equals("PASSWORD")){
158                 tmpEntityField = new EntityField(tmpFieldContent, VaultDataFieldType.PASSWORD);
159             }else{
160                 tmpEntityField = new EntityField(tmpFieldContent, VaultDataFieldType.TEXT);
161             }
162             tmpFieldsMap.put(k.toString(), tmpEntityField);
163         });
164         this.currentEntity.setFields(tmpFieldsMap);
165         this.currentEntity.setId(id);
166     }
```

Rys. 12.5 Wczytywanie dokumentu [opracowanie własne]

12.1.5. Weryfikowanie hasła do sejfu

Proces weryfikacji hasła do sejfu przedstawiony na rysunku 12.6 polega na sprawdzeniu, czy odszyfrowana za pomocą wyprowadzonego klucza wiadomość weryfikacyjna jest zgodna z UUID sejfu. W zależności od wyniku operacji metoda zwraca prawdę lub fałsz.

```
168     private boolean confirmPassword() throws SQLException, EmptyResponseException {
169         if(this.vaultId.equals(this.cryptEngine.decrypt(this.sqLiteHandler.retrieveCheckMessage()))) return true;
170         return false;
171     }
```

Rys. 12.6 Weryfikowanie hasła do sejfu [opracowanie własne]

12.1.6. Tworzenie nowego sejfu

Metoda z rysunku 12.7 jest odpowiedzialna za tworzenie nowego sejfu na podstawie otrzymanych danych i jest wykorzystywana zarówno przez aplikację desktopową, jak i serwerową.

```
37    public void initializeNewDatabase(String databaseId, String databaseUrl, String iv, String checkMessage, String vk) throws SQLException {
38        this.databaseId = databaseId;
39        String configStatement = "INSERT INTO config (id, checkMessage, iv, vk)"
40                + "VALUES ('"
41                + this.databaseId + "', '"
42                + checkMessage + "', '"
43                + iv + "', '"
44                + vk + "')";
45        this.databaseUrl = databaseUrl;
46        this.connection = DriverManager.getConnection(this.SQLITE_PREFIX + this.databaseUrl);
47        this.statement = this.connection.createStatement();
48        this.statement.execute(this.DB_INITIAL_STATEMENT_1);
49        this.statement.execute(this.DB_INITIAL_STATEMENT_2);
50        this.statement.execute(configStatement);
51    }
```

Rys. 12.7 Tworzenie nowego pliku sejfu [opracowanie własne]

12.1.7. Łączenie z sejfem

Analogicznie metoda z rysunku 12.8 jest wykorzystywana przez oba warianty aplikacji.

```
53    public void connectToDatabase(String databaseUrl) throws VaultDoesNotExistException, SQLException, EmptyResponseException {
54        this.databaseUrl = databaseUrl;
55        File file = new File(this.databaseUrl);
56        if(!file.exists() || file.isDirectory()){
57            throw new VaultDoesNotExistException();
58        }
59        this.connection = DriverManager.getConnection(this.SQLITE_PREFIX + this.databaseUrl);
60        this.statement = this.connection.createStatement();
61        this.databaseId = this.retrieveId();
62    }
```

Rys. 12.8 Metoda łącząca z plikiem sejfu [opracowanie własne]

12.1.8. Pobieranie zaszyfrowanych nazw dokumentów

Metoda widoczna na rysunku 12.9 jest odpowiedzialna za przekazanie listy nazw dokumentów zapisanych w sejfie. Na początku wybierane są nazwy i wektory inicjalizacyjne poszczególnych dokumentów. Następnie dane mapowane są do formatu obsługiwanego przez inne części aplikacji.

```
100    public Map<Integer, SQLiteHandlerResponse> getEntityNames() throws SQLException {
101        Map<Integer, SQLiteHandlerResponse> map = new HashMap<Integer, SQLiteHandlerResponse>();
102        String selectStatement = "SELECT id, iv, name FROM entries";
103        ResultSet resultSet = statement.executeQuery(selectStatement);
104        SQLiteHandlerResponse sqLiteHandlerResponse = null;
105        while(resultSet.next()){
106            sqLiteHandlerResponse = new SQLiteHandlerResponse();
107            sqLiteHandlerResponse.setResponse(resultSet.getString("name"));
108            sqLiteHandlerResponse.setIv(resultSet.getString("iv"));
109            map.put(resultSet.getInt("id"), sqLiteHandlerResponse);
110        }
111        return map;
112    }
```

Rys. 12.9 Pobieranie zaszyfrowanych nazw dokumentów [opracowanie własne]

12.2. Aplikacja desktopowa

12.2.1. Otwieranie sejfu

Otwieranie pliku z sejfem przedstawione jest na rysunku 12.10. W metodzie odpowiedzialnej za otwarcie sejfu, oprócz metod stricte związanych z JavaFX używane są klasy i metody dostarczone przez rdzeń aplikacji.

```
47    private void connectToVaultDriver() throws IOException {
48        VaultDriver vaultDriver = new VaultDriver();
49        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource( name: "../vaultview/vault_view.fxml"));
50        Parent root = fxmlLoader.load();
51        Scene scene = new Scene(root);
52        this.stage = (Stage)anchor.getScene().getWindow();
53        try {
54            vaultDriver.connectToVault(this.vaultUrl, passwordInput.getText());
55            this.stage.setScene(scene);
56            stage.show();
57            this.errorText.setText("");
58            VaultView vaultView = fxmlLoader.getController();
59            vaultView.setVaultDriver(vaultDriver);
60            vaultView.fillEntityList();
61        } catch (SQLException e) {
62            e.printStackTrace();
63        } catch (IncorrectPasswordException e) {
64            this.errorText.setText("Oops, provided password seems to be wrong");
65        } catch (VaultDoesNotExistException e) {
66            e.printStackTrace();
67        } catch (EmptyResponseException e) {
68            e.printStackTrace();
69        }
70    }
```

Rys. 12.10 Metoda odpowiedzialna za ładowanie sejfu [opracowanie własne]

12.2.2. Ładowanie głównego widoku sejfu

Na rysunku 12.11 widoczna jest metoda ładująca główny widok sejfu. Lista dokumentów jest ładowana w jednej sekcji. Każdy element listy składa się z nazwy dokumentu oraz niewidocznego dla użytkownika ID, potrzebnego do wyświetlenia szczegółów dokumentu.

```
165     public void fillEntityList() throws SQLException {
166         Node[] nodes = new Node[2];
167         nodes[0] = listPane.getChildren().get(0);
168         nodes[1] = listPane.getChildren().get(1);
169         listPane.getChildren().clear();
170         listPane.getChildren().add(nodes[0]);
171         listPane.getChildren().add(nodes[1]);
172         ObservableList<ListElement> data = FXCollections.observableArrayList();
173         this.vaultDriver.listEntities().forEach((id, name) -> {
174             data.add(new ListElement(id, name.substring(1, name.length() - 1)));
175         });
176         final ListView<ListElement> listView = new ListView<ListElement>(data);
177         this.listView = listView;
            Mieszko
178         listView.setCellFactory(new Callback<ListView<ListElement>, ListCell<ListElement>>() {
                Mieszko
179             @Override
180             public ListCell<ListElement> call(ListView<ListElement> listView) { return new CustomListCell(); }
183         });
184         listView.setOnMouseClicked(listClickEvent);
185         listPane.getChildren().add(listView);
186     }
```

Rys. 12.11 Ładowanie głównego widoku sejfu[opracowanie własne]

12.3. Aplikacja serwerowa

12.3.1. Tworzenie nowego sejfu

Na rysunku 12.12 widoczny jest kod odpowiedzialny za zapisanie do pliku danych przesłanych przez klienta internetowego. Przekazywane dane zawierają UUID, wektor inicjalizacyjny, wiadomość weryfikacyjną oraz klucz główny. Wszystkie dane mają format opisany w rozdziale trzecim.

```
27  @   public void createNewVault(RequestVaultConfig requestVaultConfig) throws SQLException {
28          this.vaultDriver.initializeNewDatabase(requestVaultConfig.getUuid(),
29                  requestVaultConfig.getUuid(),
30                  requestVaultConfig.getIv(),
31                  requestVaultConfig.getCheckMessage(),
32                  requestVaultConfig.getEncryptedMasterKey());
33          Long id = this.vaultRepository.findByVaultName(requestVaultConfig.getUuid()).get(0).getId();
34          VaultEntity vaultEntity = this.vaultRepository.getOne(id);
35          vaultEntity.setVaultCustomName(requestVaultConfig.getVaultName());
36          this.vaultRepository.save(vaultEntity);
```

Rys. 12.12 Tworzenie nowego pliku sejfu na serwerze [opracowanie własne]

12.3.2. Tworzenie nowego użytkownika

Metoda rejestrująca nowego użytkownika przyjmuje nazwę użytkownika oraz hasło. Funkcja sprawdza czy użytkownik o podanej nazwie już istnieje. Jeżeli podana nazwa nie znajduje się jeszcze w bazie danych, wówczas użytkownik jest rejestrowany. Do bazy danych zapisywane są nazwa użytkownika i hasło (hasło podlega haszowaniu i w takiej postaci jest zapisywane w bazie danych). Metoda widoczna jest na rysunku 12.13.

```
31    public void insertNewUser(String username, String password) throws UserAlreadyExistsException {
32        final VaultUserEntity vaultUserEntity = new VaultUserEntity();
33        final BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
34        vaultUserEntity.setUsername(username);
35        vaultUserEntity.setPassword(bCryptPasswordEncoder.encode(password));
36        if(this.vaultUserRepository.findByUsername(username).size() > 0) throw new UserAlreadyExistsException();
37        this.vaultUserRepository.save(vaultUserEntity);
38    }
```

Rys. 12.13 Rejestrowanie nowego użytkownika [opracowanie własne]

12.3.3. Pobieranie konfiguracji wybranego sejfu

Na rysunku 12.14 widoczna jest metoda odpowiadająca za przesłanie klientowi konfiguracji sejfu. Na podstawie przekazanego numeru UUID metoda uzyskuje dostęp do właściwego pliku, z którego odczytuje niezbędne dane. Następnie mapuje je na format klasy, której obiekt jest wykorzystywany przez kontroler.

```
39    public VaultConfigResponse getVaultConfig(String uuid)
40            throws SQLException, VaultDoesNotExistException, EmptyResponseException {
41        this.vaultDriver.connectToDatabase(uuid);
42
43        VaultConfigResponse vaultConfigResponse = new VaultConfigResponse(
44                uuid,
45                this.vaultDriver.retrieveCheckMessage(),
46                this.vaultDriver.retrieveIv(),
47                this.vaultDriver.retrieveKey()
48        );
49
50        return vaultConfigResponse;
```

Rys. 12.14 Pobieranie informacji o konfiguracji sejfu [opracowanie własne]

12.4. Aplikacja internetowa

12.4.1. Szyfrowanie i deszyfrowanie danych konfiguracyjnych

Rysunek 12.15 przedstawia metody odpowiedzialne za szyfrowanie i deszyfrowanie danych przy pomocy hasła. Metody te są wykorzystywane do tworzenia i walidacji wiadomości weryfikacyjnej, a także do szyfrowania i odszyfrowywania klucza głównego. Metody pobierają tekst jawny lub szyfrogram, wektor inicjalizacyjny oraz hasło. Na ich podstawie generują klucz i szyfrują bądź odszyfrowują wiadomość, którą następnie zwracają.

```
14      const encryptWithPassword = (message, iv, password) => {
15          const ivWa = CryptoJS.enc.Hex.parse(iv);
16          const messageWa = CryptoJS.enc.Utf8.parse(message);
17          const keyWa = CryptoJS.PBKDF2(password, ivWa, {keySize: 8, iterations:
            1000});
18          const cipher = CryptoJS.AES.encrypt(messageWa, keyWa, {iv: ivWa});
19          return cipher.toString();
20      }
21
22      const decryptWithPassword = (messageB64, iv, password) => {
23          const ivWa = CryptoJS.enc.Hex.parse(iv);
24          const messageWa = CryptoJS.enc.Base64.parse(messageB64);
25          const keyWa = CryptoJS.PBKDF2(password, ivWa, {keySize: 8, iterations:
            1000});
26          const plain = CryptoJS.AES.decrypt({ciphertext: messageWa}, keyWa,
            {iv: ivWa});
27          return plain.toString(CryptoJS.enc.Utf8);
28      }
```

Rys. 12.15 Szyfrowanie i deszyfrowanie na podstawie hasła [opracowanie własne]

12.4.2. Weryfikowanie poprawności hasła dostępu

Na rysunku 12.16 przedstawiona jest funkcja, która weryfikuje poprawność hasła. Działa analogicznie do metody z rysunku 12.6.

```
    const verify = (e) => {
        e.preventDefault();
        setError(null);
        const cipher = encryptWithPassword(vaultConfig.id, vaultConfig.iv,
        password);
        if(cipher === vaultConfig.checkMessage) {
            const masterKey = decryptWithPassword(vaultConfig.masterKey,
            vaultConfig.iv, password);
            history.push({pathname: "/vault", state: {
                uuid: vaultConfig.uuid,
                key: masterKey
            }});
        } else {
            setError('The password is incorrect');
        }
```

Rys. 12.16 Weryfikowanie poprawności hasła dostępu [opracowanie własne]

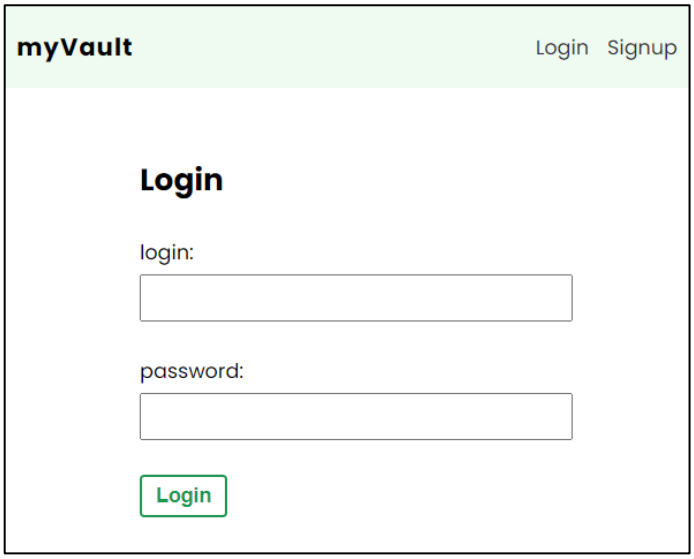
13. Interfejs użytkownika

W tym rozdziale zostaną przedstawione wybrane elementy interfejsu użytkownika.

13.1. Aplikacja internetowa

13.1.1. Strona logowania

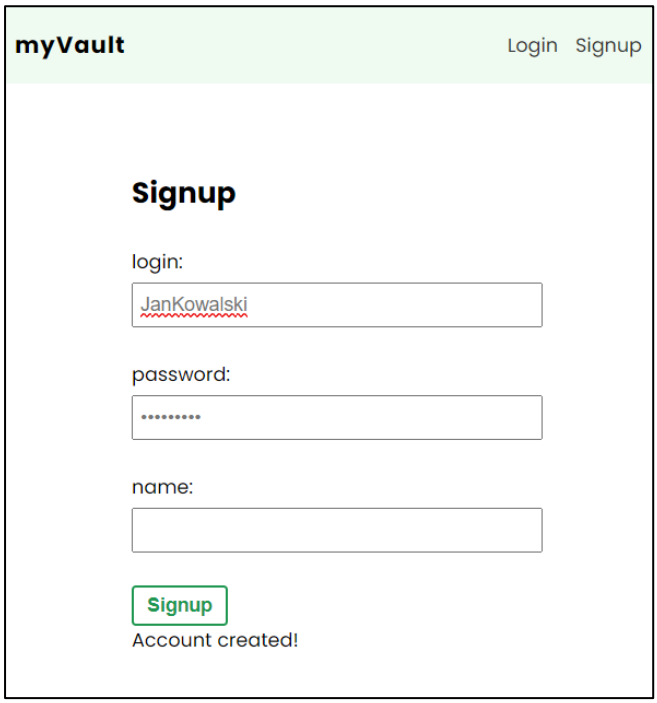
Użytkownik niezalogowany ma możliwość skorzystania ze strony logowania, składającej się z pól *login* oraz *password* oraz przycisku *Login*. Po wpisaniu poprawnych danych użytkownik zmienia status na zalogowanego i uzyskuje dostęp do funkcji aplikacji. Strona logowania widoczna jest na rysunku 13.1.



Rys. 13.1 Strona logowania [opracowanie własne]

13.1.2. Strona rejestracji

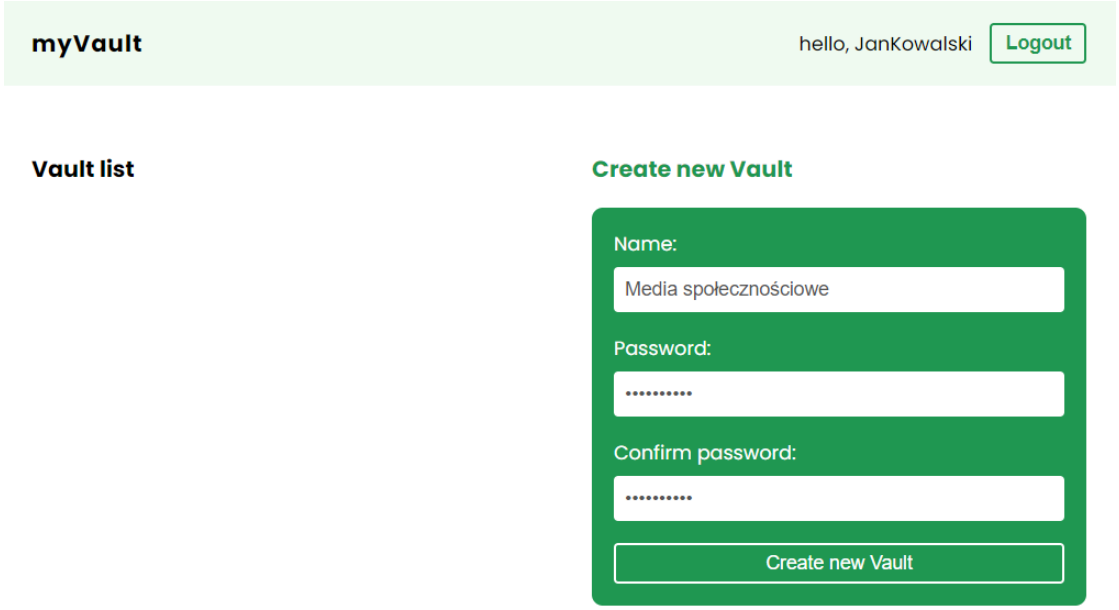
Analogiczne do strony logowania dla użytkownika niezalogowanego dostępna jest strona rejestracji. Formularz rejestracyjny zawiera pola *login*, *password* oraz *name*. Użytkownik podaje login, hasło oraz swoje imię. Po przesłaniu formularza użytkownik otrzymuje informację zwrotną o utworzeniu konta. W przypadku gdyby wystąpiły problemy podczas tworzenia konta, stosowna wiadomość wyświetlana jest pod przyciskiem. Formularz rejestracyjny przedstawiony jest na rysunku 13.2.



Rys. 13.2 Strona rejestracji chwilę po utworzeniu konta [opracowanie własne]

13.1.3. Strona główna zalogowanego użytkownika

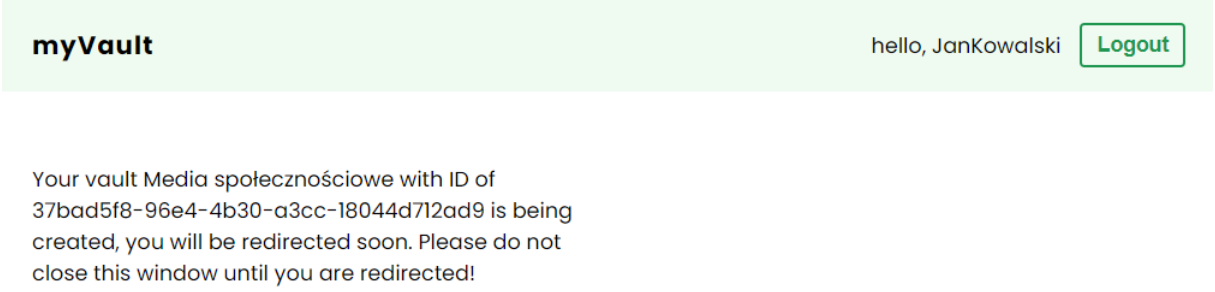
Na rysunku 13.3 widoczna jest strona główna dostępna dla zalogowanego użytkownika. Na samej górze znajduje się pasek nawigacji wyświetlający nazwę użytkownika oraz przycisk do wylogowania. W lewej kolumnie strony domowej widoczna jest lista sejfów użytkownika, a w kolumnie prawej formularz umożliwiający utworzenie nowego sejfu.



Rys. 13.3 Strona główna [opracowanie własne]

13.1.4. Strona przejściowa tworzenia sejfu

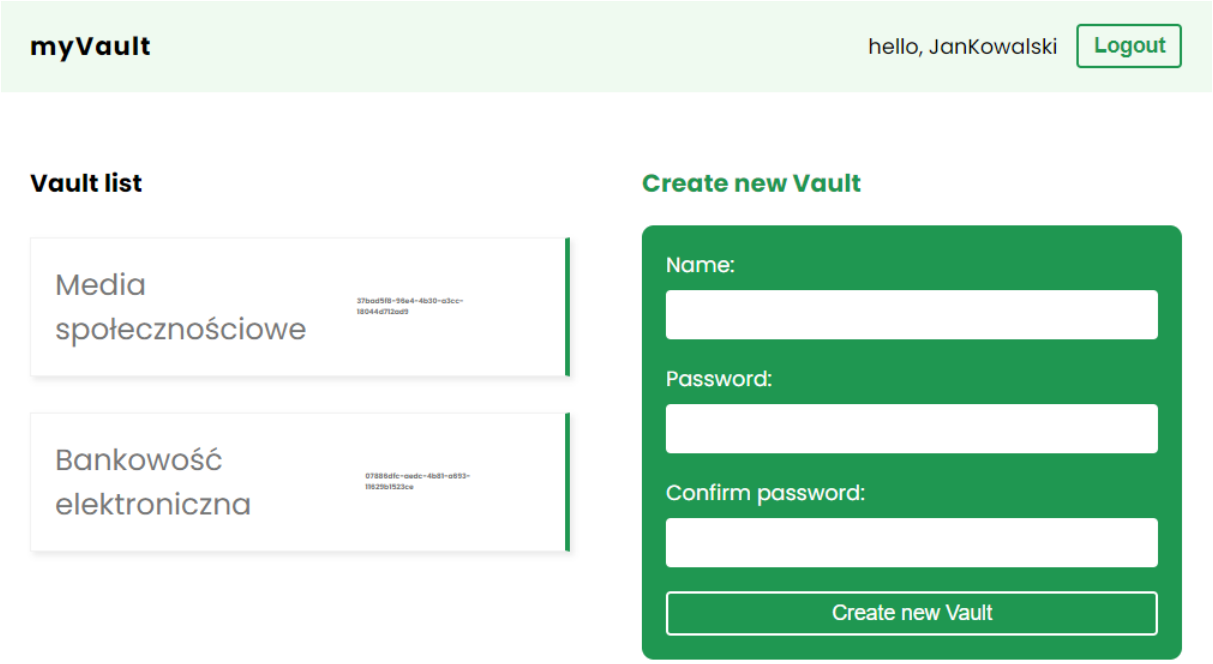
Zaraz po utworzeniu sejfu użytkownik przekierowywany jest do strony informacyjnej, w której widoczna jest nazwa sejfu i jego unikalny numer UUID. Na tej stronie przejściowej (rysunek 13.4) wyświetlany jest komunikat z prośbą o nie zamykanie okna przeglądarki. Gdy proces tworzenia sejfu przebiegnie pomyślnie użytkownik zostaje przeniesiony z powrotem na stronę główną.



Rys. 13.4 Strona przejściowa [opracowanie własne]

13.1.5. Strona główna użytkownika posiadającego sejfy

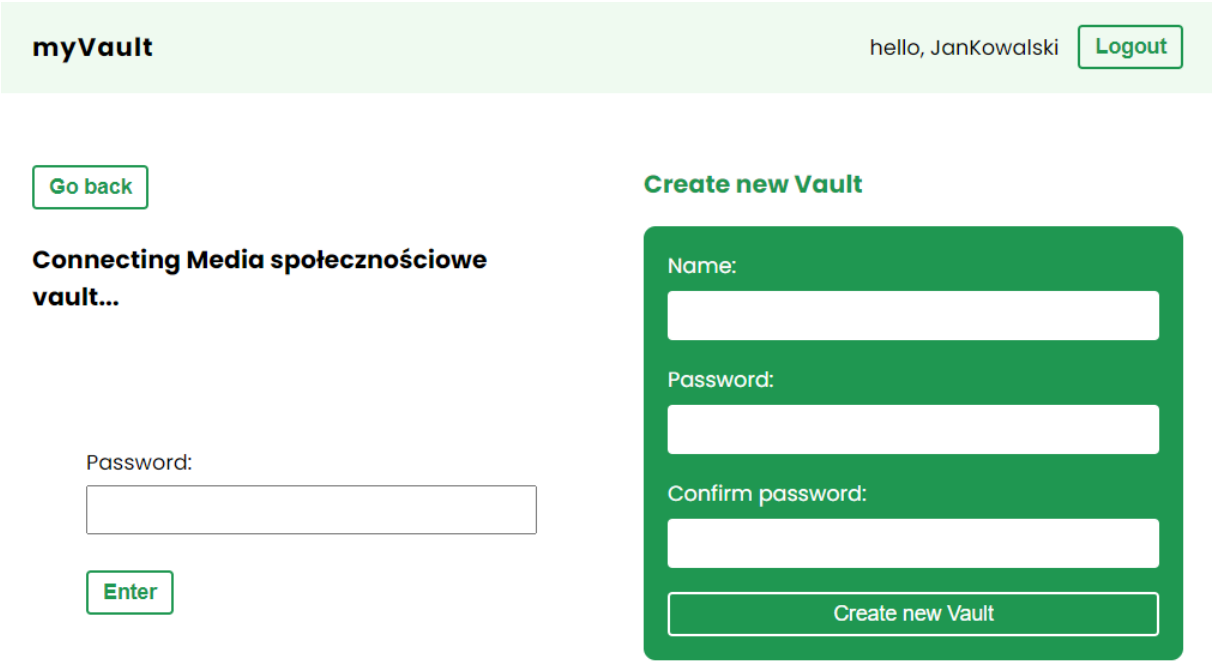
Na stronie głównej w kolumnie listy sejfów, użytkownik może wybrać sejf, który chce otworzyć. Podgląd strony zawierającej sejfy jest widoczny na rysunku 13.5.



Rys. 13.5 Strona główna [opracowanie własne]

13.1.6. Strona logowania do sejfu

Po wybraniu sejfu użytkownik proszony jest o podanie hasła dostępu. Widok tego elementu jest przedstawiony na rysunku 13.6.

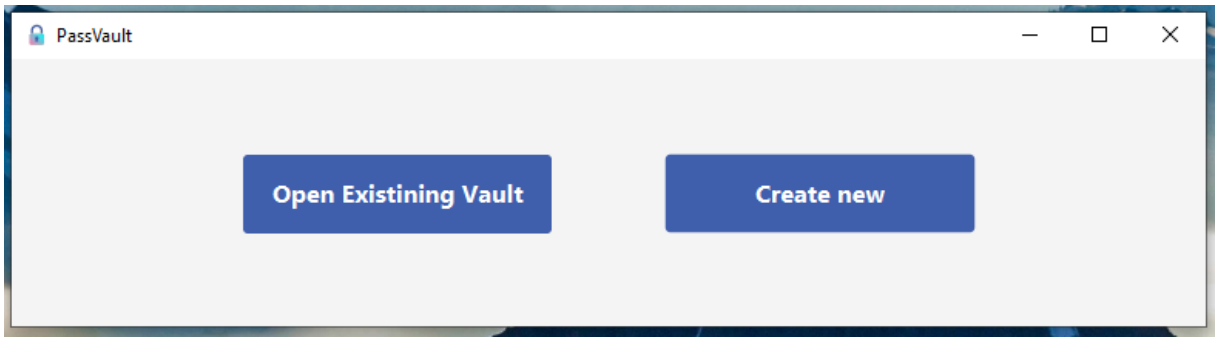


Rys. 13.6 Strona logowania sejfu [opracowanie własne]

13.2. Aplikacja desktopowa

13.2.1. Widok startowy

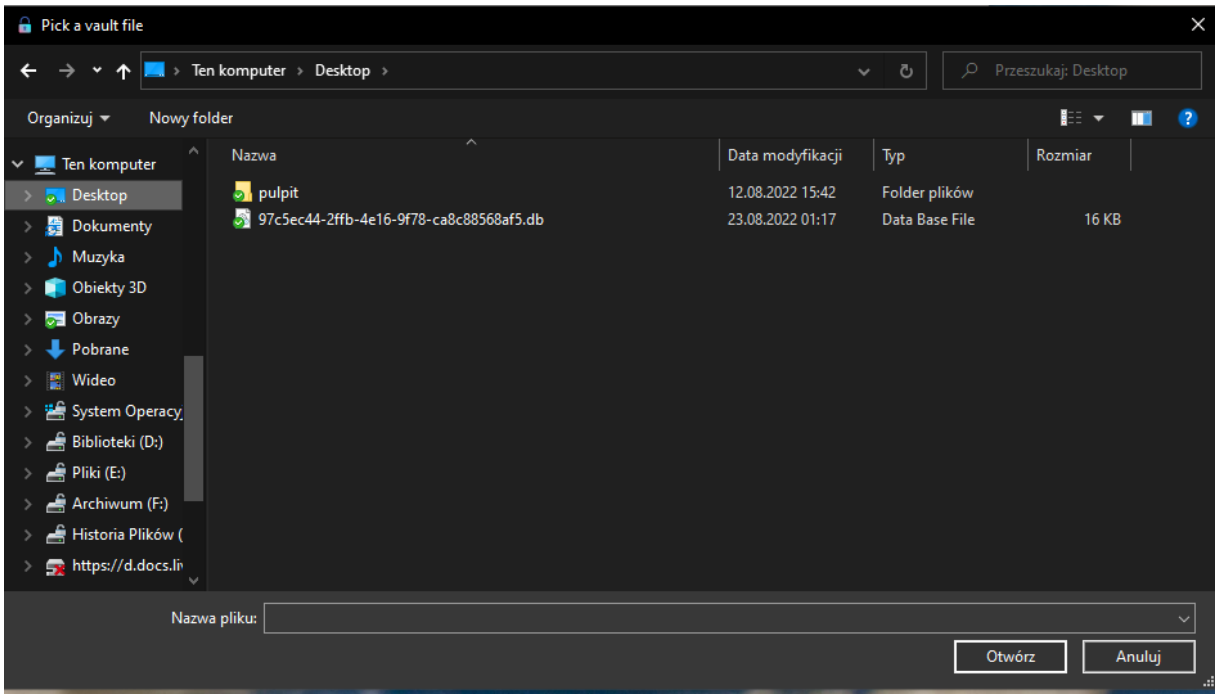
Aplikacja desktopowa po otwarciu udostępnia dwie opcje: otwarcia sejfu i utworzenia nowego. Opcje te dostępne są po wybraniu odpowiedniego przycisku. Okno dostępne po uruchomieniu aplikacji widoczne jest na rysunku 13.7.



Rys. 13.7 Widok startowy [opracowanie własne]

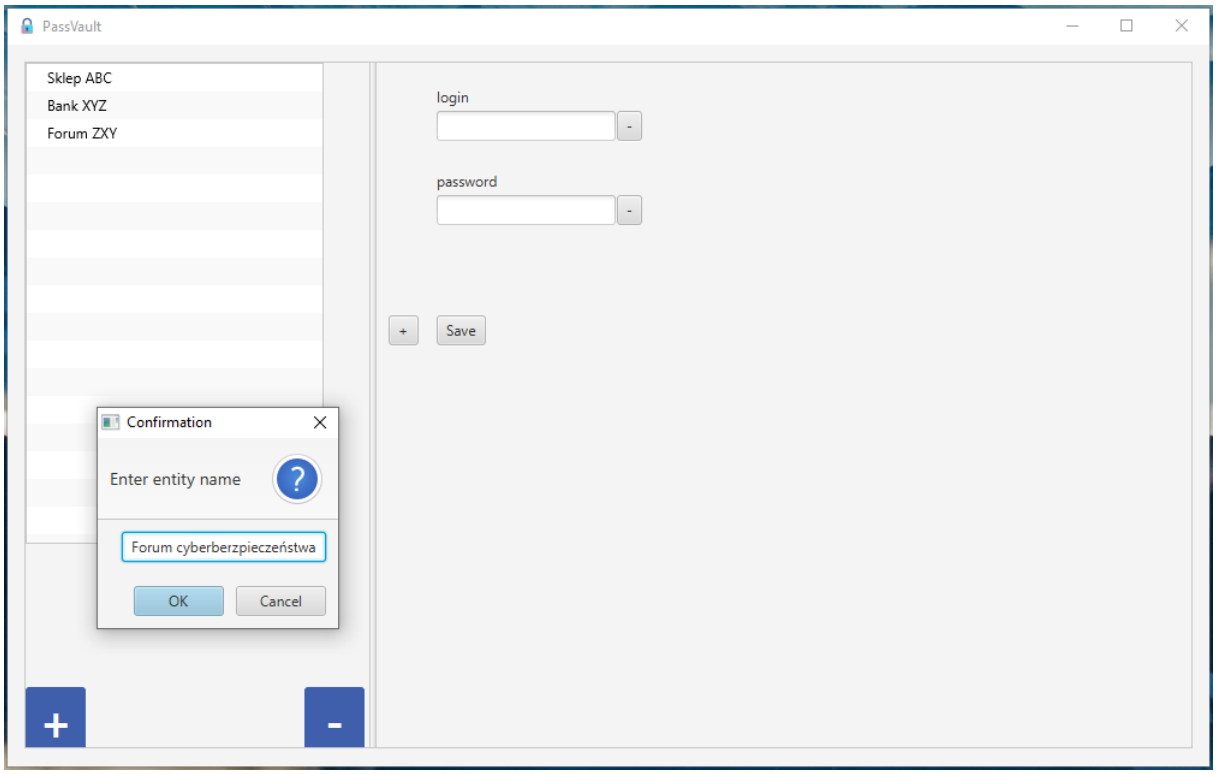
13.2.2. Okno dialogowe wyboru pliku

Po wybraniu którejkolwiek opcji wyświetlane jest okno dialogowe, z którego użytkownik może wybrać lokalizację. Okno dialogowe przedstawione jest na rysunku 13.8.



Rys. 13.8 Okno wyboru lokalizacji [opracowanie własne]

13.2.3. Tworzenie nowego dokumentu



Rys. 13.9 Podgląd sejfu [opracowanie własne]

Za pomocą niebieskiego przycisku „+” użytkownik ma możliwość dodania dokumentu, a za pomocą niebieskiego przycisku z symbolem „-” użytkownik może usunąć zaznaczony dokument. W prawej sekcji użytkownik może zarządzać polami – symbol „-” pozwala na zlikwidowanie pola, natomiast symbol „+” na dodanie nowego. Przycisk *Save* zapisuje postęp. Opisany interfejs ukazany jest na rysunku 13.9.

14. Podsumowanie

Celem pracy dyplomowej było zaprojektowanie i zaimplementowanie systemu, który pozwala w bezpieczny sposób przechowywać wrażliwe dane. Jest to możliwe dzięki zastosowaniu wielu kombinacji algorytmów. Najważniejszym elementem projektu jest zaimplementowany system zabezpieczeń i metodyka, którą się posłużono. Dla celu projektu kwestią drugorzędną jest sposób przedstawienia tych algorytmów w formie przystępnych aplikacji – te spełniają jedynie podstawowe założenia.

Zachowując podejście do sposobu przetwarzania i zapisu danych opisanego w przedmiotowej pracy można rozwijać różnego rodzaju aplikacje klienckie o różnej specyfice. Podejście, które zostało przedstawione w niniejszym projekcie można rozwinąć w wielu kierunkach i na wiele sposobów, między innymi:

- rozszerzyć możliwości aplikacji internetowej o bardziej profesjonalny interfejs użytkownika i bardziej szczegółowe metody zakładania i modyfikacji kont użytkowników,
- zaprojektować całkowicie nową aplikację desktopową w oparciu o bardziej współczesne standardy,
- w ramach różnych aplikacji można zaimplementować dodatkowe funkcjonalności, takie jak: generowanie silnych haseł, monitorowanie wycieków danych,
- zaimplementować rozwiązanie, które pozwalałoby na autouzupełnianie haseł na stronach internetowych.

W przypadku komercjalizacji aplikacji można wprowadzić różne rozwiązania, które pozwoliłyby na przynoszenie korzyści finansowych jej właścicielom, na przykład:

- wyświetlanie reklam w aplikacjach,
- możliwość rozszerzenia funkcjonalności dla użytkowników z płatnymi kontami o podniesionym standardzie.

Omówiona praca pozwoliła na ugruntowanie wiedzy w trakcie studiów z wielu zakresów, w tym przede wszystkim z zakresu tworzenia aplikacji internetowych oraz kryptografii, która w dzisiejszych czasach odgrywa kluczową rolę dla użytkowników.

Bibliografia

[1] Andrzej Zoła. *Programowanie w języku Java.* 2004

[2] William Stallings *Kryptografia i bezpieczeństwo sieci komputerowych. Matematyka szyfrów i technik kryptologii.* 2011

[3] *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.* [Online] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf /dostęp 04.2022

[4] *PKCS #5: Password-Based Cryptography Specification* [Online] https://www.rfc-editor.org/rfc/rfc8018 /dostęp 04.2022

[5] Jay A. Kreibich. *Using SQLite. Small. Fast. Reliable. Choose Any Three.* 2010

[6] Urszula Piechota, Jacek Piechota. *JavaFX. Tworzenie graficznych interfejsów użytkownika.* 2021

[7] *Maven – wprowadzenie* [Online] *https://maven.apache.org/what-is-maven.html* /dostęp 04.2022

[8] Craig Walls. *Spring w akcji. Wydanie IV.* 2015

[9] *JSON Web Token – wprowadzenie* [Online] https://jwt.io/introduction/ /dostęp 04.2022

[10] Zdzisław Dybikowski. *PostgreSQL. Wydanie II. 2012*

[11] Jon Duckett. *HTML i CSS. Zaprojektuj i zbuduj witrynę WWW. Podręcznik Front-End Developera.* 2018

[12] Jakub Kukuryk. *React.js i Node.js. Kurs video. Budowanie serwisu w oparciu o popularne biblioteki języka JavaScript.* 2020

[13] *Swagger - dokumentacja techniczna.* [Online] https://swagger.io/docs/specification/about/ /dostęp 04.2022

Spis rysunków