Studia i Materiały Informatyki Stosowanej, Tom 17, Nr 1, 2025
str.30-37

MENEDŻER HASEŁ Z FUNKCJĄ SYNCHRONIZACJI OPARTY O STANDARD ZERO-KNOWLEDGE

Jakub Knioła[1], Katarzyna Kazimierska-Drobny[2], Piotr Prokopowicz[3]

Uniwersytet Kazimierza Wielkiego
[1] Wydział Informatyki, [2] Wydział Mechatroniki
ul. Kopernika 1, 85-074 Bydgoszcz
e-mail: jakub.kniola@student.ukw.edu.pl

Streszczenie: W pracy zaprezentowano projekt oraz implementację menedżera haseł, który spełnia standardy end-to-end encryption oraz zero-knowledge encryption, gwarantując wysoki poziom bezpieczeństwa przetwarzanych danych. Aplikacja składa się z trzech komponentów: serwera, interfejsu webowego oraz rozszerzenia do przeglądarki. Do ochrony danych przed atakami wykorzystano algorytmy kryptograficzne PBKDF2 oraz AES.

Słowa kluczowe: menedżer haseł, szyfrowanie end-to-end, szyfrowanie zero-knowledge

PASSWORD MANAGER WITH ZERO-KNOWLEDGE SYNC FUNCTIONALITY

Abstract: The study presents the design and implementation of a password manager that meets the standards of end-to-end encryption and zero-knowledge encryption, ensuring a high level of security for processed data. The application consists of three components: a server, a web interface, and a browser extension. Cryptographic algorithms PBKDF2 and AES were utilized to protect the data against attacks.

Keywords: password manager, end-to-end encryption, zero-knowledge encryption

1. Wstęp

Rozwój internetu stawia przed użytkownikami nowe wyzwania związane z bezpieczeństwem. Jednym z kluczowych aspektów utrzymania bezpieczeństwa jest efektywne zarządzanie hasłami, które są podstawowym mechanizmem uwierzytelniania w większości systemów informatycznych. Niestety, z biegiem czasu ten mechanizm staje się coraz bardziej zawodny i uciążliwy. Wymagania dotyczące nowych haseł oraz ich liczba przekraczają możliwości użytkowników, co prowadzi do powszechnego stosowania tych samych haseł lub ich kombinacji w różnych systemach, a także do tworzenia pozornie unikalnych haseł opartych na publicznych informacjach czy łatwych do odgadnięcia frazach i schematach [1]. Takie praktyki znacząco obniżają skuteczność ochrony, ponieważ wyciek hasła z jednego systemu może zagrażać bezpieczeństwu innych, a pozornie unikalne hasła są łatwe do złamania, jeśli posiadamy wystarczającą wiedzę o użytkowniku, który je stworzył.

Zgodnie z najlepszymi standardami, hasło powinno być w pełni losowe i używane tylko w jednym systemie jednocześnie. Aby ułatwić użytkownikom przestrzeganie tych zasad, powstały menedżery haseł. Oprogramowanie to rozwiązuje większość problemów związanych z zarządzaniem hasłami i oferuje dodatkowe funkcje, takie jak synchronizacja haseł między urządzeniami, automatyczne uzupełnianie haseł na stronach oraz generowanie losowych haseł.

Menedżery haseł przechowują zapisane dane w odpowiednio zabezpieczonej bazie danych, która najczęściej jest szyfrowana hasłem głównym, znanym tylko użytkownikowi. Zamiast zapamiętywać loginy i hasła do wielu serwisów, użytkownik musi zapamiętać jedynie jedno hasło główne. Dzięki temu jest w stanie stworzyć jedno wystarczająco silne hasło, które będzie w stanie zapamiętać [2]. Potencjalnym problemem jest jednak utrata lub uszkodzenie bazy danych, co wiąże się z automatyczną utratą wszystkich zapisanych danych. Aby temu zapobiec, niektóre menedżery haseł umożliwiają przechowywanie kopii bazy danych na serwerze oraz synchronizację z kopią lokalną.

Synchronizacja haseł opiera się zazwyczaj na dwóch standardach: end-to-end encryption oraz zero-knowledge encryption. Pierwszy z nich polega na tym, że baza danych z hasłami jest szyfrowana lub deszyfrowana lokalnie, po stronie klienta, przed wysłaniem lub po pobraniu z serwera. Serwer przechowuje wyłącznie zaszyfrowaną kopię bazy danych, więc nie ma dostępu do jej zawartości [3]. Do szyfrowania bazy danych używane jest hasło główne, które zna tylko użytkownik. Drugi standard pozwala serwerowi na weryfikację tożsamości użytkownika na podstawie hasła głównego, bez poznawania jego treści [4]. Hasło jest haszowane lokalnie, zanim zostanie przesłane do serwera, który przechowuje tylko jego przekształconą wersję.

Na rynku dostępnych jest wiele menedżerów haseł, oferujących różne funkcjonalności. Można je podzielić na proste aplikacje komputerowe, takie jak

KeePassXC, bardziej zaawansowane rozwiązania z interfejsem webowym, aplikacjami na komputery i urządzenia mobilne oraz synchronizacją, jak 1Password czy Bitwarden, oraz rozwiązania wbudowane w systemy operacyjne i przeglądarki, takie jak Google Password Manager czy Apple Keychain. Wszystkie menedżery haseł chronią zapisane dane, najczęściej przy pomocy hasła głównego, ale pojawiają się także opcje wykorzystujące inne mechanizmy ochrony. Przykładem są menedżery wspierające rozwiązania bezhasłowe, takie jak protokół FIDO2 [5], który umożliwia użytkownikom zabezpieczenie bazy danych oraz weryfikację tożsamości przy użyciu klucza bezpieczeństwa, odcisku palca lub skanu twarzy, zamiast tradycyjnego hasła głównego.

Celem niniejszej pracy jest zaprojektowanie i implementacja menedżera haseł składającego się z trzech odrębnych, ale współpracujących ze sobą aplikacji. Pierwszą z nich jest serwer, który umożliwia synchronizację zaszyfrowanej bazy danych haseł między urządzeniami użytkowników. Drugą aplikacją jest interfejs webowy, zawierający formularze rejestracji i logowania, stronę ustawień oraz panel zarządzania hasłami. Panel ten pozwala na edycję bazy danych haseł oraz oferuje dodatkowe funkcje, takie jak generator losowych haseł oraz powiadomienia bezpieczeństwa, które informują użytkownika o potencjalnym zagrożeniu związanym z zapisanymi hasłami. Ostatnią aplikacją jest rozszerzenie do przeglądarki Google Chrome, które zapewnia dwie kluczowe funkcjonalności ułatwiające codzienne korzystanie z menedżera: automatyczne uzupełnianie zapisanych haseł na powiązanych stronach oraz automatyczne dodawanie nowych haseł do bazy danych podczas logowania lub rejestracji na stronach internetowych. Cała implementacja menedżera opiera się na zaprezentowanym algorytmie, który spełnia wymagania standardów end-to-end encryption oraz zero-knowledge encryption.

2. Tworzenie managera haseł

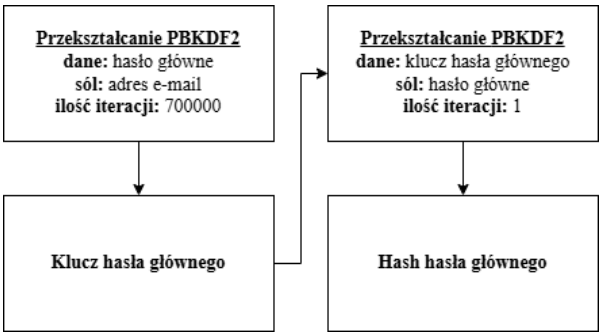
2.1. Projekt algorytmu

Przed przystąpieniem do implementacji aplikacji, opracowano algorytm, który spełnia standardy określone w pierwszym rozdziale pracy. W algorytmie tym wykorzystano dwa istniejące i szeroko przebadane algorytmy kryptograficzne, kierując się jednocześnie najlepszymi praktykami oraz powszechnie przyjętymi standardami w branży, wzorując się na rozwiązaniach stosowanych przez Bitwardena [6]. Algorytm został podzielony na dwa etapy: proces rejestracji oraz proces logowania użytkownika.

Proces przekształcania hasła realizowany jest za pomocą algorytmu PBKDF2 z funkcją haszującą SHA256 [7-8]. Algorytm ten przyjmuje trzy parametry: hasło, sól oraz liczbę iteracji, a następnie zwraca wynik o stałej długości, niezależny od długości oryginalnego hasła, który nie może zostać odwrócony do pierwotnej postaci. Zastosowanie soli pozwala uzyskać różne wyniki nawet dla identycznych haseł, co utrudnia przeprowadzenie ataków słownikowych. Liczba iteracji określa, ile razy hasło wraz z solą będą przetwarzane przez funkcję haszującą. Im wyższa liczba iteracji, tym dłużej trwa proces uzyskiwania wyniku, co znacząco utrudnia ataki typu brute force.

Proces szyfrowania danych realizowany jest za pomocą algorytmu AES w trybie CBC z kluczem o długości 256 bitów [7-8]. Algorytm ten przyjmuje dwa parametry: dane do zaszyfrowania oraz klucz, a zwraca szyfrogram, który może zostać odszyfrowany przy użyciu tego samego klucza. W celu zapewnienia odpowiedniej długości klucza w aplikacji, przed jego przesłaniem do algorytmu szyfrującego, jest on najpierw przekształcany przy użyciu algorytmu PBKDF2. Zastosowany algorytm wymaga również podania wektora inicjującego, jednak szczegóły dotyczące jego obsługi zostały pominięte w niniejszym rozdziale, ponieważ pełną obsługę wektora inicjującego zapewniają biblioteki narzędziowe opisane w rozdziale dotyczącym interfejsu webowego.

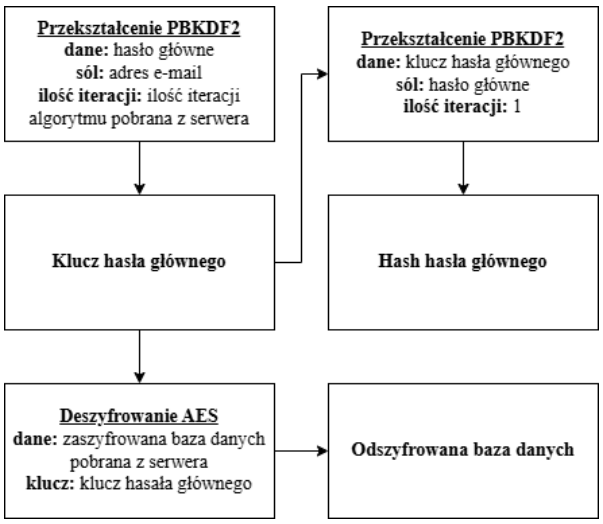
Podczas rejestracji użytkownik podaje adres e-mail oraz hasło główne, które następnie poddawane jest procesowi przekształcania za pomocą algorytmu PBKDF2. W trakcie tego procesu hasło jest solone podanym adresem e-mail oraz przetwarzane przez siedemset tysięcy iteracji algorytmu, co skutkuje utworzeniem klucza hasła głównego, który posłuży do zabezpieczenia bazy danych z hasłami. Następnie ten klucz jest ponownie przetwarzany przy użyciu tego samego algorytmu, ale tym razem solony jest podanym hasłem głównym i poddawany tylko jednej iteracji. W efekcie uzyskiwany jest hasz hasła głównego, który wraz z adresem e-mail jest wysyłany do serwera. Na podstawie tych danych serwer może zweryfikować tożsamość użytkownika podczas logowania.



Rys 1. Proces rejestracji użytkownika.

Podczas procesu logowania użytkownik wprowadza adres e-mail oraz hasło główne, które następnie jest przetwarzane algorytmem PBKDF2. Hasło solone jest podanym adresem e-mail i poddawane liczbie iteracji algorytmu, która została pobrana z serwera i zastosowana podczas rejestracji dla podanego adresu e-mail. W wyniku tego procesu uzyskiwany jest klucz hasła głównego, który ponownie przekształcany jest przy użyciu tego samego algorytmu. Tym razem jest solony podanym hasłem głównym i przechodzi przez tylko jedną iterację. W

efekcie uzyskiwany jest hasz hasła głównego, który wraz z adresem e-mail przesyłany jest do serwera w celu weryfikacji tożsamości użytkownika. Jeśli przesłane dane zgadzają się z danymi zapisanymi podczas rejestracji, serwer odsyła zaszyfrowaną bazę danych z hasłami. Następnie, z wykorzystaniem wcześniej uzyskanego klucza hasła głównego, baza danych zostaje odszyfrowana za pomocą algorytmu AES.



Rys. 2. Proces logowania użytkownika.

Zaprezentowany algorytm spełnia dwa standardy przedstawione we wstępie pracy. Wszystkie dane wprowadzone przez użytkownika są bezpiecznie przesyłane i przechowywane na serwerze. Dostęp do tych danych posiada wyłącznie osoba, która zna hasło główne. Administrator serwera oraz potencjalny atakujący, który próbowałby uzyskać dostęp do przechowywanych na serwerze informacji, nie byliby w stanie odczytać zawartości bazy danych ani treści hasła głównego.

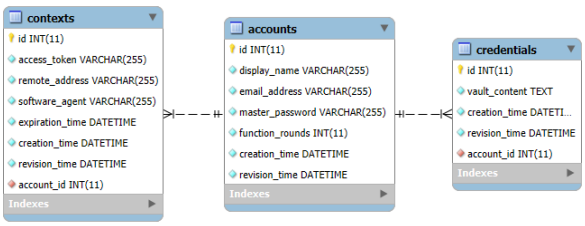
2.2. Implementacja serwera

Pierwszym etapem implementacji menedżera haseł było stworzenie serwera z wykorzystaniem języka programowania Python oraz frameworka Flask. Serwer umożliwia synchronizację zaszyfrowanych haseł po uwierzytelnieniu użytkownika i został zaprojektowany zgodnie z założeniami przedstawionymi we wstępie pracy.

Baza danych

Struktura bazy danych została zaprojektowana z wykorzystaniem biblioteki SQLAlchemy, która umożliwia mapowanie obiektowo-relacyjne [9]. Wykorzystanie tego rozwiązania znacznie uprościło integrację aplikacji z bazą danych, eliminując konieczność używania surowych zapytań SQL na rzecz wysokopoziomowego API dostarczanego przez bibliotekę.

W pliku models.py utworzono trzy klasy reprezentujące odpowiednie tabele w bazie danych: Account, Context, i Credential. Klasa Account przechowuje dane o użytkownikach, klasa Context dane o sesjach, a klasa Credential zaszyfrowane hasła. Tabela Account jest połączona relacją jeden do wielu z tabelami Context i Credential, co oznacza, że jeden użytkownik może mieć przypisane wiele sesji oraz wiele haseł.



Rys. 3. Diagram relacji między tabelami.

Komunikacja klient-serwer

Wykorzystując framework Flask, stworzono architekturę opartą o REST API, umożliwiającą komunikację między klientem a serwerem. Architektura ta, oparta na protokole HTTP, zarządza zasobami poprzez definiowanie ścieżek (punktów końcowych, z ang. endpoints), które pozwalają klientowi na wykonywanie operacji za pomocą odpowiednich metod HTTP oraz przesyłanych danych w formacie JSON [10]. W aplikacji zastosowano również kilka punktów końcowych, które wyrażają akcje zamiast zasobów, co nieco odbiega od standardowych założeń REST.

Zaprogramowano mechanizm ograniczający dostęp do wybranych punktów końcowych wyłącznie dla uwierzytelnionych użytkowników. W tym celu w pliku decorators.py stworzono dekorator authentication_required, który sprawdza obecność nagłówka Authorization zawierającego token dostępu. Dekorator weryfikuje, czy token jest obecny w bazie danych oraz czy nie wygasł. W przypadku pomyślnej weryfikacji, dekorator wykonuje żądany punkt końcowy i wstrzykuje do niego obiekt użytkownika powiązany z tokenem dostępu.

W katalogu routes stworzono dwa pliki, w których zaprogramowano punkty końcowe odpowiedzialne za zarządzanie użytkownikami oraz hasłami. Plik accounts.py zawiera punkty końcowe do logowania, tworzenia oraz aktualizacji użytkowników, natomiast plik credentials.py obejmuje operacje związane z zarządzaniem zapisanymi hasłami. Dodatkowo, wykorzystując bibliotekę marshmallow, w pliku schemas.py stworzono klasy odpowiedzialne za walidację, serializację i deserializację danych przesyłanych między klientem a serwerem.

2.3. Implementacja interfejsu webowego

Drugim etapem implementacji menedżera haseł było stworzenie interfejsu webowego z wykorzystaniem

języka programowanie JavaScript oraz frameworka React. Interfejs umożliwia użytkownikom rejestrację, logowanie oraz zarządzanie hasłami, komunikując się z wcześniej utworzonym serwerem, a całość została zaprojektowana zgodnie z założeniami przedstawionymi we wstępie pracy.

Biblioteki narzędziowe

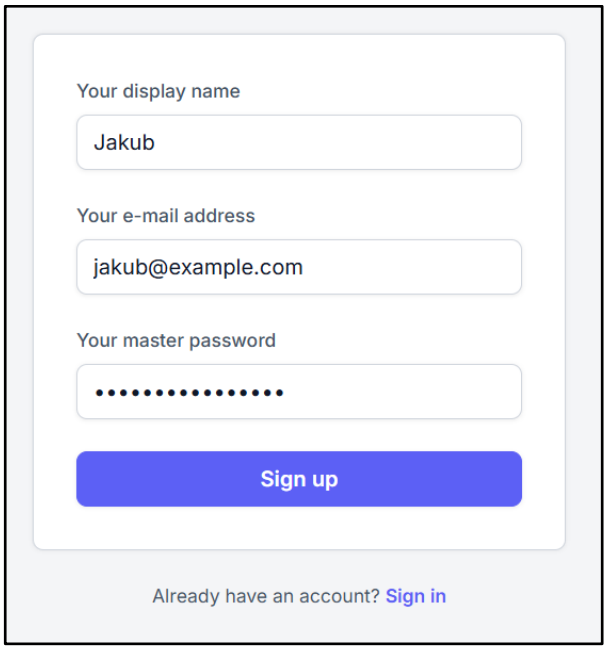
W katalogu utilities stworzono dwa pliki zawierające funkcje narzędziowe, które upraszczają realizację kluczowych działań. W pliku requests.jsx zaimplementowano funkcje obsługujące zapytania do serwera przy użyciu wbudowanej w przeglądarkę funkcji fetch. Funkcje te automatycznie serializują i deserializują przesyłane dane, obsługują potencjalne błędy oraz dołączają token dostępu sesji w celu uwierzytelnienia klienta po stronie serwera.

W pliku crypto.jsx zaprogramowano funkcje odpowiedzialne za operacje kryptograficzne, które wykorzystują interfejs Web Crypto API [11] z wbudowanymi implementacjami wybranych algorytmów. Funkcje te umożliwiają przekształcanie danych za pomocą algorytmu PBKDF2 oraz szyfrowanie i deszyfrowanie danych algorytmem AES. Podczas szyfrowania i deszyfrowania algorytm AES wymaga wektora inicjującego o stałej długości 16 bajtów, który musi być identyczny w obu procesach. Aby efektywnie rozwiązać problem generowania i przechowywania tego wektora, funkcja szyfrująca generuje losowy wektor inicjujący, który jest zapisywany na pierwszych 16 bajtach szyfrogramu. Dzięki temu funkcja deszyfrująca może automatycznie odczytać wektor inicjujący z szyfrogramu i użyć go podczas odszyfrowywania danych.

Komponenty

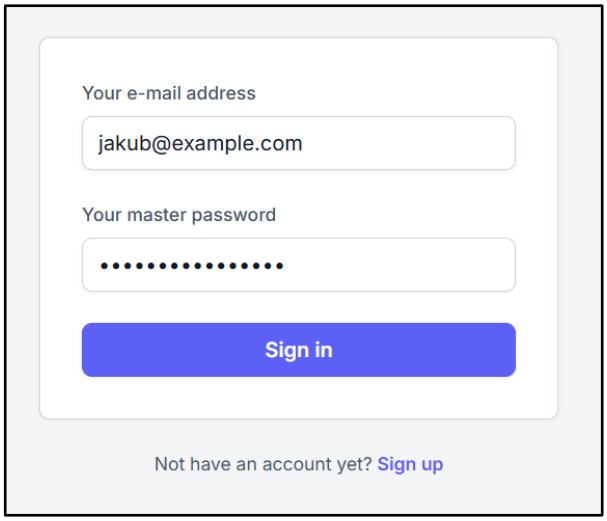
W katalogu components opracowano komponenty odpowiedzialne za interfejs graficzny użytkownika, z których każdy zawiera formularz umożliwiający interakcję z aplikacją. Do uproszczenia implementacji oraz efektywnej obsługi formularzy zastosowano bibliotekę react-hook-form, która znacząco ułatwia tworzenie formularzy w React. Wygląd interfejsu został określony w arkuszu stylów style.css, znajdującym się w głównym katalogu projektu. Motyw graficzny oparto na rozwiązaniu Untitled UI, a ikony pochodzą z biblioteki Font Awesome.

W pliku Registration.jsx umieszczono komponent zawierający formularz rejestracji użytkownika, który składa się z przycisku oraz pól tekstowych do wprowadzenia nazwy użytkownika, adresu e-mail i hasła głównego. Proces rejestracji został zaimplementowany zgodnie z wcześniej przedstawionym algorytmem. Po przetworzeniu danych z formularza, są one przesyłane do serwera. Po pomyślnej rejestracji użytkownik jest przekierowywany na stronę logowania.



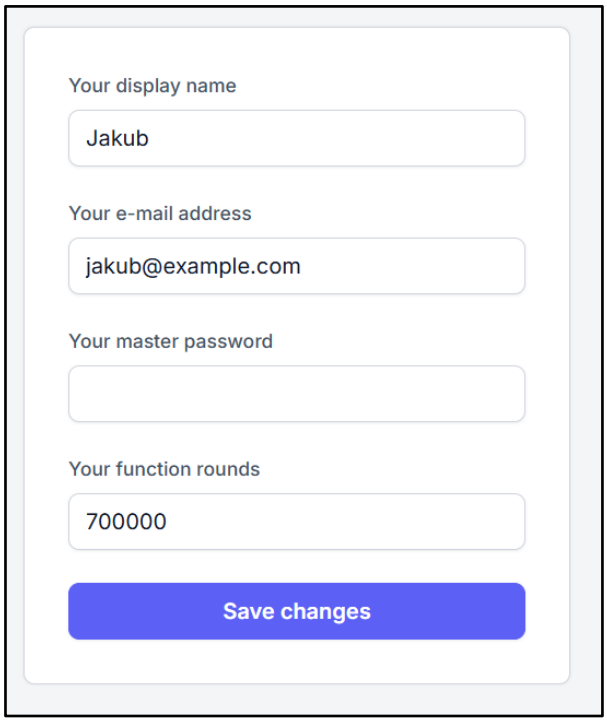
Rys. 4. Formularz rejestracji użytkownika.

W pliku Authentication.jsx umieszczono komponent zawierający formularz logowania użytkownika, który składa się z przycisku oraz pól tekstowych do wprowadzenia adresu e-mail i hasła głównego. Proces logowania został zaimplementowany zgodnie z wcześniej przedstawionym algorytmem. Przetworzone dane z formularza są przesyłane do serwera. Po pomyślnym logowaniu, szczegółowe dane użytkownika oraz token dostępu sesji są zapisywane w pamięci przeglądarki (z wykorzystaniem kontekstu AccountContext.jsx), a użytkownik zostaje przekierowany na stronę panelu z hasłami. Dodatkowo, komponent zapisuje podany adres e-mail w pamięci przeglądarki (z wykorzystaniem mechanizmu localStorage), co pozwala na podanie jedynie hasła głównego przy kolejnym logowaniu, bez potrzeby ponownego wprowadzania adresu e-mail.



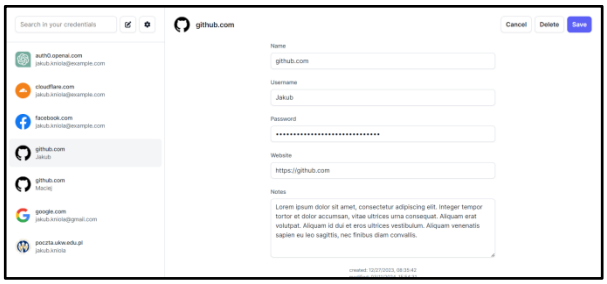
Rys. 5. Formularz logowania użytkownika.

W pliku Settings.jsx umieszczono komponent zawierający formularz umożliwiający aktualizację danych użytkownika, który składa się z przycisku oraz pól tekstowych: nazwy użytkownika, adresu e-mail, hasła głównego oraz liczby iteracji algorytmu. Wszystkie pola, z wyjątkiem hasła głównego, są automatycznie wypełniane aktualnymi danymi użytkownika. Proces przetwarzania danych z formularza jest zgodny z wcześniej przedstawionym algorytmem. Po udanej aktualizacji użytkownik zostaje przekierowany na stronę panelu z hasłami.



Rys. 6. Formularz zmiany ustawień.

W pliku Credentials.jsx umieszczono komponent z panelem haseł, który w górnej lewej części zawiera pole wyszukiwania, przycisk do tworzenia nowego hasła oraz przycisk umożliwiający przejście do strony ustawień. Poniżej znajduje się lista przechowywanych haseł, z której każdy element zawiera ikonę, nazwę własną oraz nazwę użytkownika. Po wybraniu hasła z listy, szczegóły są wyświetlane w prawej części panelu. Widok szczegółowy składa się z nagłówka oraz formularza. Nagłówek zawiera ikonę oraz nazwę własną po lewej stronie, a po prawej znajdują się przyciski umożliwiające zapisanie zmian, odrzucenie zmian oraz usunięcie hasła. Kliknięcie przycisku usuwania wywołuje komunikat potwierdzający zamiar usunięcia wybranego hasła. Z kolei kliknięcie ikony otwiera selektor plików, umożliwiając zmianę ikony. Centralna część widoku szczegółowego to formularz zawierający pola tekstowe: nazwę własną, nazwę użytkownika, hasło, adres witryny oraz notatki. Pole hasła jest domyślnie ukryte, ale po najechaniu kursorem na pole i kliknięciu ikony, hasło staje się widoczne, a poniżej pojawia się generator haseł, który pozwala ustawić długość hasła oraz rodzaj znaków, które mają być w nim zawarte. Na dole formularza wyświetlane są data utworzenia oraz data ostatniej edycji hasła.



Rys. 7. Panel z hasłami.

Jeśli użytkownik zapisze to samo hasło więcej niż raz, poniżej pola z hasłem pojawi się komunikat ostrzegawczy (powiadomienie bezpieczeństwa). Podobny komunikat wyświetli się, gdy zapisane hasło znajdzie się w publicznie dostępnym wycieku danych. W celu weryfikacji, czy hasło zostało upublicznione, wykorzystywane jest API dostarczane przez haveibeenpwned.com. Fragment hasza zapisanych haseł jest wysyłany do tego API, a w odpowiedzi zwracana jest lista upublicznionych haseł zaczynających się od przesłanego fragmentu. Następnie sprawdzane jest, czy na liście znajduje się hasz sprawdzanego hasła. Jeśli tak, wyświetlany jest odpowiedni komunikat ostrzegawczy. To rozwiązanie zapewnia poufność sprawdzanego hasła. Dodatkowo, każde takie ostrzeżenie powoduje pojawienie się czerwonej, migającej kropki obok danego hasła na liście haseł po lewej stronie panelu.

Proces szyfrowania i deszyfrowania danych został zrealizowany zgodnie z wcześniej zaprezentowanym algorytmem. Zaszyfrowane dane są pobierane z serwera, odszyfrowywane lokalnie w celu odczytu lub edycji, a następnie ponownie szyfrowane przed ich przesłaniem na serwer. Dodatkowo, wykrycie zmian w hasłach przechowywanych w pamięci przeglądarki generuje zdarzenie synchronization, które zawiera wszystkie zapisane hasła, co zostało opisane dokładnie w rozdziale poświęconym rozszerzeniu do przeglądarki.

Konteksty i trasy

Podczas tworzenia komponentów zaimplementowano konteksty umożliwiające przechowywanie danych w pamięci konkretnej karty przeglądarki, co oznacza, że dane te są automatycznie usuwane po zamknięciu karty i nie są współdzielone między innymi kartami tej samej witryny. Użytkownik może poruszać się po różnych ścieżkach w obrębie jednej karty, nie tracąc zapisanych informacji. Taki mechanizm jest szczególnie pożądany w aplikacji, ponieważ przechowywane dane są wrażliwe, a ograniczenie ich dostępności minimalizuje ryzyko nieautoryzowanego dostępu. Konteksty zapewniają również reaktywność na zmiany przechowywanych danych, co stanowi przewagę nad standardowym mechanizmem sessionStorage dostępnym w przeglądarkach W katalogu contexts

utworzono dwa pliki: AccountContext.jsx, przechowujący dane o zalogowanym użytkowniku i token sesji używany do uwierzytelniania na serwerze, oraz SaveContext.jsx, który nasłuchuje zdarzeń typu save i przechowuje przesłane dane przez rozszerzenie, wyświetlając je w formularzu na stronie panelu z hasłami, więcej na ten temat w rozdziale dotyczącym rozszerzenia do przeglądarki.

Wykorzystując bibliotekę react-router oraz wcześniej stworzone komponenty i konteksty, w pliku App.jsx zaprojektowano strukturę umożliwiającą dynamiczne zmienianie wyświetlanego komponentu w zależności od aktualnej ścieżki w adresie witryny. Dodatkowo w katalogu routers zaimplementowano element ProtectedRoute.jsx, który sprawdza, czy w kontekście AccountContext.jsx znajdują się dane uwierzytelnionego użytkownika. W przypadku ich braku użytkownik jest przekierowywany na stronę logowania. Komponenty wymagające zalogowanego użytkownika, takie jak Settings.jsx i Credentials.jsx, zostały opakowane w ten element. Stworzono również zestaw ścieżek, które prowadzą do odpowiednich wcześniej zaprojektowanych komponentów.

2.4. Implementacja rozszerzenia do przeglądarki

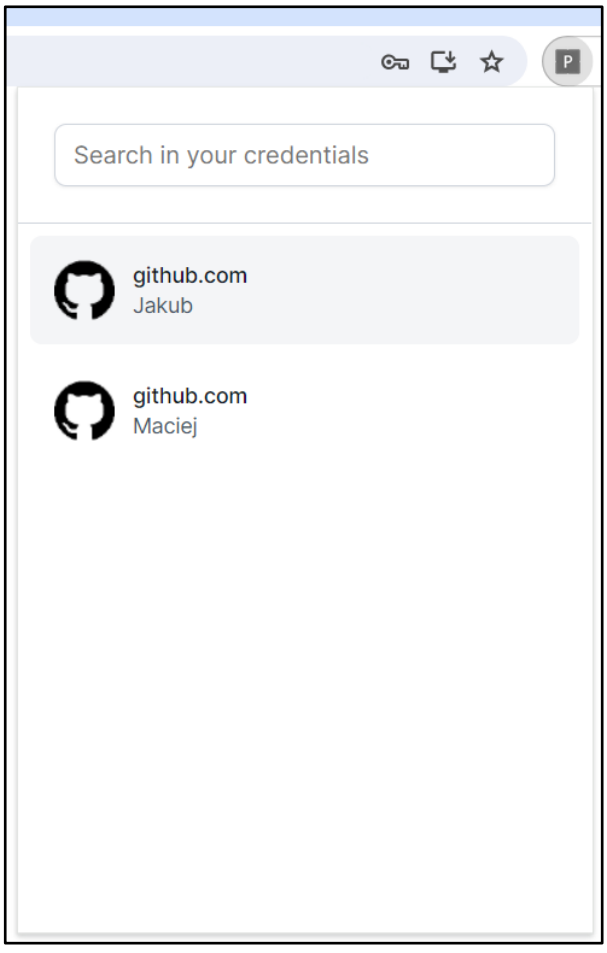
Trzecim etapem implementacji menedżera haseł było stworzenie rozszerzenia do przeglądarki Google Chrome z wykorzystaniem języka programowania JavaScript. Rozszerzenie umożliwia automatyczne uzupełnianie zapisanych haseł na powiązanych stronach oraz automatyczne zapamiętywanie nowych haseł, komunikując się z wcześniej utworzonym interfejsem webowym.

Automatyczne uzupełnianie haseł

Zaimplementowano mechanizm synchronizacji, który przesyła zapisane hasła z pamięci interfejsu webowego do pamięci rozszerzenia, eliminując potrzebę tworzenia dodatkowego formularza logowania w rozszerzeniu i wykorzystując istniejący formularz w interfejsie webowym. Mechanizm ten opiera się na funkcjonalności umożliwiającej wstrzykiwanie plików JavaScript do otwartych kart w przeglądarce oraz wymianę danych między nimi a plikami w rozszerzeniu. Do każdej karty zawierającej interfejs webowy wstrzykiwany jest plik synchronization.js, który nasłuchuje zdarzenia synchronization zawierającego wszystkie zapisane hasła użytkownika. Po wykryciu tego zdarzenia hasła są przesyłane do pliku background.js, który zapisuje je w pamięci rozszerzenia do momentu zamknięcia przeglądarki.

Stworzono również interfejs pozwalający użytkownikowi wybrać nazwę użytkownika i hasło do wprowadzenia w formularzu na aktywnej stronie. W katalogu popup umieszczono następujące pliki: popup.html zawierający strukturę interfejsu, popup.css z arkuszem stylów oraz popup.js implementujący mechanizm autouzupełniania. Interfejs ten jest dostępny po kliknięciu ikony rozszerzenia w liście obok paska adresu przeglądarki. Składa się z dwóch głównych elementów: pola wyszukiwania umieszczonego w górnej części oraz listy haseł poniżej. Każdy element listy zawiera ikonę, nazwę własną, nazwę użytkownika oraz opcjonalnie czerwoną migającą kropkę informującą o ostrzeżeniu. Przy każdym uruchomieniu interfejsu sprawdzana jest aktualna strona użytkownika, a w liście wyświetlane są jedynie hasła powiązane z daną stroną, co zapobiega przypadkowemu użyciu hasła przypisanego do innej witryny.

Wybranie elementu z listy powoduje przesłanie nazwy użytkownika i hasła do wstrzykniętego pliku automation.js, który znajduje się na aktywnej stronie. Plik ten jest wstrzykiwany do każdej strony zaraz po jej otwarciu. Po odebraniu danych wyszukiwane są pola formularza – pole na nazwę użytkownika (typu text lub email) oraz pole na hasło (typu password). Żadne z tych pól nie może posiadać atrybutu hidden. Jeśli oba pola zostaną znalezione, przesłane dane są automatycznie wprowadzane.

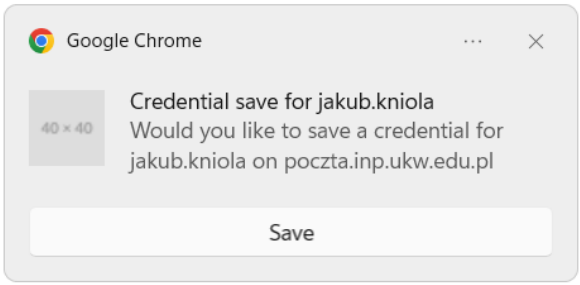


Rys. 8. Lista haseł do autouzupełnienia.

Automatyczne zapisywanie haseł

W pliku automation.js zaimplementowano mechanizm, który przed zamknięciem strony wyszukuje pola z nazwą użytkownika oraz hasłem, wykorzystując ten

sam algorytm, co w procesie autouzupełniania. Jeśli oba pola zostaną znalezione, dane w nich zawarte, wraz z adresem strony, są przesyłane do pliku background.js. Plik ten weryfikuje w pamięci rozszerzenia, czy przesłana nazwa użytkownika i adres strony znajdują się wśród zapisanych haseł. W przypadku ich braku wysyłane jest powiadomienie systemowe, informujące użytkownika o możliwości automatycznego zapisu danych.



Rys. 9. Powiadomienie o autozapisie.

Kliknięcie powiadomienia powoduje przekierowanie użytkownika na stronę panelu zarządzania hasłami oraz przesłanie danych do wstrzykniętego skryptu synchronization.js, który dalej przekazuje je za pomocą zdarzenia save, odbieranego przez kontekst SaveContext.jsx. W panelu z hasłami automatycznie wyświetlany jest formularz wypełniony odebranymi danymi, co umożliwia użytkownikowi ich weryfikację lub edycję przed zapisaniem. Po wprowadzeniu zmian użytkownik może kliknąć przycisk zapisu. Jeśli jednak strona panelu zarządzania hasłami nie była wcześniej otwarta, użytkownik będzie musiał najpierw się zalogować.

3. Podsumowanie i wnioski

Praca skupiała się na zaprojektowaniu i implementacji menedżera haseł, który spełnia wysokie standardy bezpieczeństwa, takie jak end-to-end encryption oraz zero-knowledge encryption. Dzięki zastosowaniu algorytmów PBKDF2 i AES udało się stworzyć rozwiązanie zapewniające poufność danych zarówno podczas ich przesyłania, jak i przechowywania na serwerze, minimalizując ryzyko nieautoryzowanego dostępu, w tym przez administratorów systemu.

Aplikacja składa się z trzech komponentów: serwera, interfejsu webowego oraz rozszerzenia do przeglądarki, które zostały zaprojektowane w sposób umożliwiający ich wzajemną współpracę.

Serwer umożliwia synchronizację danych między urządzeniami użytkownika, zapewniając pełne bezpieczeństwo informacji dzięki przechowywaniu wyłącznie zaszyfrowanych danych. Synchronizacja pozwala na dostęp do zapisanych haseł z dowolnego urządzenia i miejsca, jednocześnie stanowiąc bezpieczną kopię zapasową, co minimalizuje ryzyko utraty danych.

Interfejs webowy aplikacji został zaprojektowany z myślą o wygodzie użytkownika i ułatwia zarządzanie hasłami. Zawiera generator haseł oraz funkcję automatycznych powiadomień, ostrzegających o potencjalnych zagrożeniach, takich jak zapisanie tego samego hasła po raz drugi czy wykrycie haseł w publicznych bazach wycieków.

Rozszerzenie do przeglądarki jest kluczowym elementem, który znacząco usprawnia codzienne korzystanie z aplikacji. Dzięki niemu użytkownik może automatycznie uzupełniać formularze logowania oraz zapisywać nowe hasła, co eliminuje potrzebę ręcznego wprowadzania danych.

Przeprowadzona implementacja potwierdziła, że zastosowanie standardów end-to-end encryption oraz zero-knowledge encryption w menedżerze haseł jest skutecznym podejściem do ochrony danych uwierzytelniających. Rozwiązanie to może być rozwijane o kolejne funkcjonalności, takie jak wsparcie dla technologii bezhasłowych, co otworzyłoby nowe możliwości w zakresie uwierzytelniania i ochrony danych.

Podsumowując, opracowany system spełnia postawione cele projektowe, dostarczając bezpieczne i funkcjonalne narzędzie do zarządzania hasłami. Jego modułowa budowa oraz elastyczność pozwalają na dalszy rozwój i adaptację do zmieniających się potrzeb użytkowników oraz wymagań bezpieczeństwa. Praca ta pokazuje, że innowacyjne podejście do ochrony danych może być skutecznie wdrożone w aplikacjach użytkowych, stanowiąc istotny krok w kierunku poprawy bezpieczeństwa cyfrowego.

Literatura

1. Badanie dotyczące nawyków związanych z bezpieczeństwem haseł przeprowadzone na grupie Amerykanów przez security.org w 2021 roku. Dostępne pod adresem: https://www.security.org/resources/online-password-strategies/
2. Opis mechanizmu zabezpieczania bazy danych z hasłami napisany przez proton.me. Dostępny pod adresem: https://proton.me/blog/how-do-password-managers-work
3. Opis działania mechanizm end-to-end encryption napisany przez ibm.com. Dostępny pod adresem: https://www.ibm.com/think/topics/end-to-end-encryption
4. Opis działania mechanizmu zero-knowledge encryption napisany przez tresorit.com. Dostępny pod adresem: https://tresorit.com/blog/zero-knowledge-encryption/
5. Dokumentacja menedżera haseł Bitwarden dotycząca wykorzystania protokołu FIDO2 do zabezpieczenia bazy danych z hasłami oraz weryfikacji tożsamości użytkownika. Dostępna pod adresem: https://bitwarden.com/help/login-with-passkeys/
6. Dokumentacja menedżera haseł Bitwarden dotycząca zabezpieczania bazy danych z hasłami. Dostępna pod adresem:

https://bitwarden.com/help/bitwarden-security-white-paper/
7. Karbowski M, Podstawy kryptografii, Helion, wyd. III.
8. Houtven L., Crypto101, 2017, Dostępne po adresem: https://www.crypto101.io/
9. Dokumentacja SQLAlchemy. Dostępna pod adresem: https://www.sqlalchemy.org/
10. Opis architektury REST API napisany przez ibm.com. Dostępny pod adresem: https://www.ibm.com/think/topics/rest-apis
11. Dokumentacja Web Crypto API. Dostępna pod adresem: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API