**AKADEMIA NAUK STOSOWANYCH**

**W NOWYM SĄCZU**

**WYDZIAŁ NAUK INŻYNIERYJNYCH**

**PRACA DYPLOMOWA**

**APLIKACJA WEBOWA PRZECHOWUJĄCA PLIKI Z DOSTĘPEM CHRONIONYM**

**Autor: Bartłomiej Cetera**

**Kierunek: Informatyka**

**Nr albumu: 31230**

**Promotor: ……………………………………….**

**Akceptacja promotora: …………………………………………………… data i podpis**

**NOWY SĄCZ 2025**

Spis treści

[Wstęp 4](#_Toc186908748)

[1. Wprowadzenie 5](#_Toc186908749)

[1.1. Przegląd stanu wiedzy 5](#_Toc186908750)

[1.2. Zagrożenia w chmurze 6](#_Toc186908751)

[1.3 Analiza dostępnych rozwiązań 7](#_Toc186908752)

[2. Cel i zakres pracy 10](#_Toc186908753)

[3. Projekt 11](#_Toc186908754)

[3.1. Wymagania funkcjonalne 11](#_Toc186908755)

[3.2. Wymagania niefunkcjonalne 13](#_Toc186908756)

[3.3. Technologie 13](#_Toc186908757)

[3.4. Diagramy 17](#_Toc186908758)

[3.4.1. Diagram przypadków użycia 17](#_Toc186908759)

[3.4.2. Diagram ERD 19](#_Toc186908760)

[3.4.3. Diagramy aktywności 22](#_Toc186908761)

[3.5. Scenariusze użycia 26](#_Toc186908762)

[4. Implementacja 32](#_Toc186908763)

[4.1. System logowania i rejestracji 32](#_Toc186908764)

[4.2. System zarządzania plikami 36](#_Toc186908765)

[4.3. System znajomych 41](#_Toc186908766)

[4.4. System magazynów 46](#_Toc186908767)

[4.5. System powiadomień i wiadomości 56](#_Toc186908768)

[4.6. System enkrypcji i degrupcji danych 59](#_Toc186908769)

[5. Podsumowanie 62](#_Toc186908770)

[6. Bibliografia 63](#_Toc186908771)

[Spis tabel 66](#_Toc186908772)

[Spis rysunków 67](#_Toc186908773)

# Wstęp

Rozwój technologii oraz powszechna dostępność internetu przyczyniły się do tego,   
że przechowywanie plików w chmurze stało się integralnym elementem współczesnego życia. Systemy chmurowe umożliwiają wygodny dostęp do danych z dowolnego miejsca oraz ich synchronizację pomiędzy różnymi urządzeniami. Popularność tych rozwiązań wynika z ich uniwersalności, elastyczności i możliwości integracji z nowoczesnymi narzędziami oraz procesami. Dzięki chmurze użytkownicy mogą efektywniej organizować swoją pracę, optymalizując jednocześnie wykorzystanie zasobów technologicznych. Chociaż systemy chmurowe oferują liczne zalety, takie jak dostępność, skalowalność czy wysoki poziom bezpieczeństwa, wiążą się również z istotnymi wyzwaniami. Kluczowe kwestie dotyczą ochrony danych i prywatności użytkowników, które stają się coraz bardziej palącym problemem w obliczu rosnącego zagrożenia cyberatakami. Wyzwania te podkreślają znaczenie zaawansowanych mechanizmów ochrony, takich jak szyfrowanie danych czy wielopoziomowe uwierzytelnianie. Wraz z upowszechnieniem przechowywania danych w chmurze, zarówno przez użytkowników indywidualnych, jak i przedsiębiorstwa, wzrosły potrzeby w zakresie bezpiecznych i funkcjonalnych rozwiązań umożliwiających zarządzanie plikami. Projektowanie aplikacji spełniających te wymagania stanowi kluczowe wyzwanie współczesnej inżynierii oprogramowania. Niniejsza praca koncentruje się na problematyce tworzenia aplikacji internetowej, która odpowiada na te potrzeby.

# 1. Wprowadzenie

## 1.1. Przegląd stanu wiedzy

Przechowywanie danych w chmurze to jedna z kluczowych funkcji technologii chmurowych, oferująca użytkownikom możliwość przechowywania i udostępniania danych   
za pośrednictwem platform internetowych. Systemy te charakteryzują się elastycznością, wysoką dostępnością oraz niskimi kosztami użytkowania, dzięki czemu są powszechnie wykorzystywane zarówno przez użytkowników indywidualnych, jak i przedsiębiorstwa różnej wielkości. W zależności od potrzeb, przechowywanie danych w chmurze może przyjmować różne formy, w tym chmurę publiczną, prywatną, hybrydową, osobistą i społecznościową.   
Pod względem technologicznym, systemy przechowywania danych w chmurze opierają się   
na trzech głównych modelach: pamięci blokowej, plikowej oraz obiektowej. Pamięć blokowa, stosowana głównie w środowiskach wymagających wysokiej wydajności, umożliwia szybki dostęp do danych w bazach danych i aplikacjach transakcyjnych. Pamięć plikowa, zintegrowana z technologią NAS, oferuje elastyczne zarządzanie danymi w strukturze plików, co sprawia, że jest intuicyjna dla użytkowników końcowych. Pamięć obiektowa, zoptymalizowana pod kątem przechowywania dużych zbiorów niestrukturalnych danych, takich jak multimedia, wyróżnia się wysoką skalowalnością i efektywnością w obsłudze różnorodnych zastosowań. Rozwój technologii przechowywania w chmurze pozwala   
na przezwyciężenie ograniczeń tradycyjnych metod, takich jak wysokie koszty konserwacji sprzętu czy brak elastyczności w zarządzaniu zasobami. Dzięki swojej wszechstronności   
i efektywności, systemy te stają się integralnym elementem współczesnych rozwiązań technologicznych, wspierając zarówno użytkowników indywidualnych, jak i organizacje   
w zarządzaniu danymi na coraz większą skalę. (Yang, Xiong, Ren 2020) (Odun-Ayo 2017)

Trendy w korzystaniu z usług chmurowych w Unii Europejskiej wskazują na dynamiczny rozwój i rosnącą popularność tych rozwiązań wśród przedsiębiorstw. Według danych Eurostatu z 2023 roku, aż 42,5% firm w UE korzysta z usług przetwarzania w chmurze, co oznacza wzrost o 4,2% w porównaniu z rokiem 2021. W Polsce ten wzrost jest jeszcze bardziej   
zauważalny odsetek firm korzystających z usług chmurowych wyniósł 55,7%, co jest wynikiem wyższym niż średnia unijna (45,2%) i stanowi istotny postęp w adopcji technologii chmurowych. Wzrost wykorzystania chmury w Polsce można przypisać m.in. poprawie dostępności usług chmurowych, co było możliwe dzięki uruchomieniu regionalnych centrów danych przez globalnych dostawców, takich jak Google, Microsoft i Amazon. Firmy w Polsce coraz częściej sięgają po zaawansowane usługi chmurowe, takie jak platformy do tworzenia   
i wdrażania aplikacji, co wskazuje na rosnącą świadomość w zakresie strategicznego wykorzystania tych technologii. Jednocześnie jednak odnotowano stosunkowo niskie zainteresowanie oprogramowaniem zabezpieczającym, co podkreśla potrzebę edukacji   
i zwiększenia inwestycji w obszarze cyberbezpieczeństwa. W całej UE zauważalna jest różnorodność w poziomie wykorzystania usług chmurowych, przy czym największe przedsiębiorstwa (77,6%) dominują pod względem adopcji tych technologii. Większość firm korzysta z modelu SaaS (Software as a Service), który oferuje łatwą integrację i niskie bariery wejścia. Coraz częściej jednak przedsiębiorstwa sięgają po bardziej zaawansowane modele, takie jak IaaS (Infrastructure as a Service) czy PaaS (Platform as a Service), co wskazuje   
na rozwój w kierunku bardziej skomplikowanych i wszechstronnych zastosowań chmurowych. Trendy te odzwierciedlają rosnącą rolę chmury w modernizacji procesów biznesowych, zwiększaniu efektywności operacyjnej oraz wspieraniu innowacji w różnych sektorach gospodarki. Taka dynamika rozwoju wskazuje, że chmura stała się kluczowym elementem strategii cyfrowej przedsiębiorstw w Europie, jednak równocześnie podkreśla konieczność ciągłego rozwoju zabezpieczeń i edukacji w zakresie zarządzania ryzykiem związanym   
z przechowywaniem danych w chmurze. (Itwiz 2024) (Cloudforum 2024)

## 1.2. Zagrożenia w chmurze

Zagrożenia związane z przechowywaniem danych w chmurze obejmują zarówno aspekty technologiczne, jak i organizacyjne. Jednym z kluczowych problemów jest podatność   
na cyberataki, w tym kradzież danych czy ataki typu ransomware. Chociaż wiele firm stosuje zaawansowane technologie szyfrowania danych, nie eliminuje to ryzyka złamania zabezpieczeń przez atakujących wykorzystujących np. ataki brute-force lub phishing w celu zdobycia danych logowania. Kolejnym zagrożeniem jest brak pełnej kontroli   
nad przechowywanymi danymi. Współpracując z zewnętrznymi dostawcami chmury, użytkownicy muszą polegać na ich procedurach bezpieczeństwa, które nie zawsze spełniają najwyższe standardy. Ponadto dane mogą być przechowywane w różnych jurysdykcjach,   
co rodzi ryzyko związane z przepisami prawnymi i dostępem organów państwowych   
do informacji. Niezawodność połączenia internetowego stanowi dodatkowy problem.   
W sytuacji awarii infrastruktury sieciowej użytkownicy mogą utracić dostęp do swoich danych. Brak odpowiednich mechanizmów tworzenia kopii zapasowych przez niektórych dostawców zwiększa ryzyko ich nieodwracalnej utraty. Kwestia prywatności pozostaje jednym   
z największych wyzwań. Wiele platform gromadzi metadane użytkowników, które mogą   
być wykorzystywane do analizy lub sprzedaży stronom trzecim. Taka praktyka rodzi obawy   
o nadmierne gromadzenie informacji i potencjalne naruszenia prywatności. Rozwiązania minimalizujące te zagrożenia obejmują włączenie uwierzytelniania dwuskładnikowego, szyfrowanie danych przed ich przesłaniem do chmury, regularne monitorowanie bezpieczeństwa kont oraz wybór dostawców oferujących jasne polityki prywatności   
i bezpieczeństwa. Ważne jest również systematyczne edukowanie użytkowników na temat cyberbezpieczeństwa, aby unikali ryzykownych zachowań, takich jak otwieranie podejrzanych wiadomości e-mail czy korzystanie z niezabezpieczonych sieci publicznych. (Steinberg 2023) (Schneier 2017) (Huang i in, 2014) (360money 2024) (Bitdefender 2025)

## 1.3 Analiza dostępnych rozwiązań

Obecnie dostępnych jest wiele usług umożliwiających przechowywanie plików   
w chmurze. Ze względu na swoją dostępność, funkcjonalność i wygodę, platformy te są szeroko wykorzystywane zarówno w codziennym życiu, jak i w środowisku zawodowym. Mimo licznych zalet, usługi te napotykają na istotne ograniczenia oraz wyzwania, które mogą wpływać na poziom zaufania użytkowników do tych rozwiązań.

Dysk Google – usługa stworzona przez Google, umożliwiająca przechowywanie, synchronizację i udostępnianie plików. Uruchomiona 24 kwietnia 2012 roku, pozwala użytkownikom przechowywać pliki na serwerach Google, synchronizować je na różnych urządzeniach oraz udostępniać innym. Dysk Google obejmuje pakiet biurowy (Dokumenty, Arkusze i Prezentacje Google), który umożliwia wspólną edycję dokumentów, arkuszy kalkulacyjnych, prezentacji, rysunków, formularzy i innych plików. Wszystkie pliki utworzone w pakiecie biurowym są automatycznie zapisywane na Dysku Google. (Wikipedia – Dysk Google 2024) ( Scroll - Dysk Google 2021)

Dropbox - usługa oferowana przez firmę Dropbox, Inc., umożliwiająca przechowywanie danych na zdalnych serwerach. Dostęp do plików jest możliwy za pośrednictwem przeglądarki internetowej lub dedykowanej aplikacji instalowanej na komputerze. W wersji darmowej użytkownik otrzymuje 2 GB przestrzeni dyskowej, z opcją zwiększenia jej za dodatkową opłatą. Istnieje również możliwość bezpłatnego powiększenia dostępnej przestrzeni poprzez skorzystanie z ofert promocyjnych, takich jak zapraszanie nowych użytkowników   
(do maksymalnie 16 GB) czy instalacja oficjalnej aplikacji. (Wikipedia - Dropbox 2024) (CodziennyEkspert 2023)

Microsoft One Drive - OneDrive oferuje użytkownikom 5 GB darmowej przestrzeni dyskowej z możliwością rozszerzenia pojemności do 100 GB. Subskrybenci usługi Microsoft 365 mają dodatkowo dostęp do 1 TB miejsca na swoje pliki. Pojedyncze pliki mogą mieć rozmiar do 100 GB. Usługa jest oficjalnie kompatybilna z przeglądarkami Microsoft Edge, Google Chrome oraz Mozilla Firefox, choć możliwe jest korzystanie z niej także w innych przeglądarkach. (Wikipedia - OneDrive 2024) (Komputer Świat 2022)

Usługi takie jak Google Drive i OneDrive charakteryzują się konkurencyjną ceną, co czyni je popularnym wyborem wśród użytkowników. Niemniej jednak ich wykorzystanie wiąże   
się z pewnymi kompromisami, które mogą budzić wątpliwości. Niskie koszty często oznaczają ograniczoną kontrolę użytkownika nad przechowywanymi danymi oraz utratę prywatności. Firmy takie jak Google i Microsoft mają dostęp do plików użytkowników oraz rutynowo   
je analizują w celu wykrywania potencjalnych zagrożeń. Działania te, choć uzasadniane troską o bezpieczeństwo, wzbudzają obawy o naruszanie prywatności. Stabilność działania tych usług również pozostawia czasem wiele do życzenia. Zgłaszano przypadki znikania plików, nawet   
w przypadku użytkowników korzystających z kont premium, co negatywnie wpływa   
na zaufanie do tych platform. Brak gwarancji bezpieczeństwa i dostępności danych jest istotnym mankamentem. Dropbox Inc. stosuje inną strategię, która również może być przedmiotem krytyki. Darmowy plan tej usługi oferuje jedynie 2 GB przestrzeni dyskowej,   
co w porównaniu z konkurencyjnymi rozwiązaniami jest niewielką wartością. Dodatkowo, liczba urządzeń, które mogą korzystać z jednego konta, jest ograniczona do dwóch,   
co zmniejsza funkcjonalność. Plany płatne są stosunkowo drogie, zaczynając od około   
500zł rocznie za 2 TB przestrzeni. Brak elastycznych opcji pośrednich między darmowym planem a kosztownymi subskrypcjami premium ogranicza dostępność tej usługi dla szerokiego grona użytkowników. Dodatkowo, Dropbox został skrytykowany za brak przejrzystości   
w zakresie polityki prywatności. Firma dzieli się danymi telemetrycznymi z partnerami biznesowymi, co rodzi pytania o ochronę danych użytkowników. Ponadto zgłaszano przypadki ponownego pojawiania się plików, które miały być trwale usunięte z serwerów, co może   
być postrzegane jako naruszenie zaufania użytkowników. Rosnące zapotrzebowanie   
na alternatywne rozwiązania, które łączą konkurencyjną cenę, funkcjonalność i bezpieczeństwo danych, wskazuje na potrzebę poszukiwania bardziej zrównoważonych modeli usług chmurowych. (Fotc 2022) (Futurebeat 2023) (Youtube - Mateusz Chrobok 2024)

# 2. Cel i zakres pracy

Temat pracy dyplomowej skupia się na zaprojektowaniu oraz implementacji internetowej aplikacji do przechowywania danych, która w odpowie na współczesne potrzeby użytkowników w zakresie bezpieczeństwa, prywatności i zarządzania plikami w środowisku online. Głównym celem aplikacji jest stworzenie narzędzia, które nie tylko zapewni użytkownikom możliwość przechowywania plików, i jednocześnie ułatwi współpracę   
i interakcję z innymi użytkownikami. Projektowana aplikacja powinna być przystosowana   
do funkcjonowania jako prywatny magazyn internetowy, który stawia na ochronę   
prywatności użytkownika. Wykorzystanie algorytmów szyfrowania zapewni bezpieczeństwo przechowywanych informacji, dając użytkownikom pewność, że ich dane są chronione przed zagrożeniami zewnętrznymi. Kolejnym kluczowym aspektem aplikacji jest wsparcie współpracy między użytkownikami. Aplikacja powinna umożliwiać tworzenie dedykowanych przestrzeni współdzielonych, nazywanych magazynami, w których użytkownicy będą mogli gromadzić, organizować i wspólnie edytować pliki. System kontroli wersji plików pozwoli   
na jednoczesną pracę kilku osób nad jednym dokumentem, a w razie wystąpienia konfliktów umożliwi ich rozwiązanie poprzez porównanie różnic między wersjami i wybór najbardziej odpowiedniej. Projektowana aplikacja powinna również odpowiadać na potrzeby związane   
z zarządzaniem plikami w sposób przyjazny użytkownikowi. Użytkownicy powinni mieć możliwość łatwego przesyłania, pobierania, zmieniania nazw oraz usuwania plików. Szczególny nacisk należy położyć na przejrzysty interfejs, który pozwoli na intuicyjne korzystanie z dostępnych funkcji, takich jak zarządzanie znajomymi, wysyłanie wiadomości oraz przeglądanie udostępnionych zasobów. Ważnym celem aplikacji jest również zapewnienie użytkownikom możliwości edycji wybranych typów plików bez konieczności ich pobierania. Dzięki integracji z odpowiednimi narzędziami użytkownicy będą mogli edytować pliki tekstowe i dokumenty Worda bezpośrednio w przeglądarce, co zwiększy wygodę i efektywność korzystania z systemu. Podsumowując, celem projektu jest stworzenie aplikacji, która stanie się alternatywą dla istniejących rozwiązań, takich jak Google Drive czy Dropbox, oferując przy tym większe bezpieczeństwo i prywatność. Projektowana aplikacja ma na celu zbudowanie zaufania użytkowników poprzez wdrożenie funkcji zarządzania plikami i współpracy w ramach środowiska cyfrowego.

# 3. Projekt

## 3.1. Wymagania funkcjonalne

System rejestracji - Aplikacja powinna umożliwiać rejestrację użytkowników, którzy podają adres e-mail, hasło oraz imię i nazwisko. Hasła należy bezpiecznie przechowywać, stosując haszowanie. Logowanie powinno odbywać się przy użyciu e-maila i hasła,   
a dla dodatkowego bezpieczeństwa warto dodać dwuetapowe uwierzytelnianie (2FA), które wykorzysta aplikacje typu Google Authenticator. Autoryzacja użytkowników powinna   
być oparta na tokenach JWT (JSON Web Token), które przechowują zaszyfrowane dane użytkownika, takie jak identyfikator. Po zalogowaniu generowany będzie unikalny token, który można przechowywać w przeglądarce (np. w plikach cookies) i używać do autoryzacji żądań do serwera.

Zarządzanie plikami – System zarządzania plikami powinien umożliwiać użytkownikom przesyłanie plików, które będą przechowywane w indywidualnym obszarze. Podczas przesyłania należy zapisywać metadane, takie jak nazwa pliku, data, typ i identyfikator właściciela, w bazie danych, co ułatwi zarządzanie i wyszukiwanie zasobów. Każdy użytkownik powinien mieć dostęp do widoku z listą swoich plików, z możliwością przeglądania szczegółów oraz wykonywania operacji, takich jak pobieranie, zmiana nazwy, usuwanie lub udostępnianie. Zmiana nazwy powinna być odnotowywana w bazie danych   
i systemie plików, aby zachować spójność danych. Usuwanie plików powinno działać poprzez zmianę ich statusu w bazie danych. Udostępnianie plików powinno umożliwiać wybór odbiorców z listy znajomych. Udostępnione pliki powinny być widoczne w dedykowanej sekcji aplikacji.

Szyfrowanie danych - Projektowany system powinien zapewniać pełne bezpieczeństwo   
i prywatność danych użytkowników, dlatego kluczowym elementem funkcjonalności aplikacji musi być implementacja szyfrowania danych. Wszystkie pliki przechowywane   
w aplikacji powinny być szyfrowane na serwerze. Proces ten ma na celu ochronę zawartości plików przed nieautoryzowanym dostępem, nawet w przypadku naruszenia bezpieczeństwa serwera. Szyfrowanie powinno być realizowane za pomocą nowoczesnych i uznanych algorytmów kryptograficznych, takich jak AES (Advanced Encryption Standard). AES,   
ze względu na swoją wyższą wydajność i odporność na ataki, powinien być preferowanym algorytmem. System powinien umożliwiać szyfrowanie plików za pomocą kluczy kryptograficznych generowanych indywidualnie dla każdego użytkownika lub magazynu plików.

Edycja plików Word i TXT - Aplikacja powinna wspierać przeglądanie i edycję plików tekstowych, takich jak .txt oraz dokumentów .docx (Word). Dzięki integracji   
z odpowiednimi narzędziami do konwersji i edycji plików użytkownicy mogliby modyfikować dokumenty bez konieczności pobierania ich na urządzenie lokalne, a zmiany mogłyby być zapisywane bezpośrednio w systemie.

System znajomych - System znajomych w aplikacji powinien oferować użytkownikom łatwe narzędzia do zarządzania relacjami i komunikacji. Użytkownicy powinni móc wyszukiwać innych za pomocą wyszukiwarki (np. po adresie e-mail) i wysyłać zaproszenia   
do znajomych. Akceptacja dodaje osobę do listy znajomych widocznej w aplikacji. Lista znajomych powinna umożliwiać przeglądanie podstawowych informacji, takich jak imię   
i nazwisko, oraz szybkie usuwanie znajomych. System powinien wspierać udostępnianie plików i zasobów – właściciel wybiera osobę z listy znajomych, a udostępnione zasoby   
są widoczne w dedykowanej sekcji. Aplikacja powinna mieć wbudowany system wiadomości do komunikacji między znajomymi oraz zapewniać powiadomienia o nowych zaproszeniach   
i wiadomościach.

System magazynów – System magazynów w aplikacji powinien umożliwiać zarządzanie plikami w dedykowanych zbiorach, które można współdzielić z innymi użytkownikami. Użytkownicy powinni mieć możliwość tworzenia magazynów z nazwami ułatwiającymi identyfikację i organizację plików. Aplikacja powinna wspierać udostępnianie magazynów innym użytkownikom, pozwalając właścicielowi na zarządzanie dostępem, w tym dodawanie   
i usuwanie osób. Kluczową funkcją powinna być kontrola wersji plików, która będzie wspierać współpracę wielu osób nad jednym plikiem. System powinien automatycznie wykrywać konflikty wersji i umożliwiać ich rozwiązanie przez porównanie różnic i wybór odpowiedniej wersji.

## 3.2. Wymagania niefunkcjonalne

Interfejs - Aplikacja powinna być zaprojektowana w taki sposób, aby interfejs   
użytkownika był prosty, przejrzysty i intuicyjny, umożliwiając użytkownikom szybki dostęp   
do wszystkich kluczowych funkcji w maksymalnie pięciu kliknięciach. Oznacza   
to, że użytkownik, korzystając z aplikacji, nie powinien być zmuszony do wykonywania zbędnych czynności lub przechodzenia przez skomplikowane struktury menu, aby osiągnąć zamierzony cel. Interfejs powinien być responsywny, dostosowany do różnych rozdzielczości ekranów. Dzięki temu użytkownicy będą mogli łatwo korzystać z aplikacji niezależnie   
od miejsca i urządzenia, z którego korzystają. Kluczowe funkcje, takie jak przesyłanie plików, zarządzanie magazynami, udostępnianie zasobów czy komunikacja ze znajomymi, powinny być widoczne i dostępne z poziomu głównego ekranu lub intuicyjnych ikon nawigacyjnych.

Dostępność - System powinien być zaprojektowany w sposób zapewniający pełną kompatybilność z wszystkimi popularnymi przeglądarkami internetowymi, takimi jak Google Chrome, Mozilla Firefox, Microsoft Edge oraz Safari, w ich najnowszych stabilnych wersjach. Zapewnienie zgodności z szeroką gamą przeglądarek jest kluczowe, aby użytkownicy mieli dostęp do aplikacji niezależnie od preferowanego środowiska pracy. Dzięki temu aplikacja może dotrzeć do szerszego grona odbiorców i zagwarantować spójne doświadczenie użytkownika bez względu na wybór przeglądarki. Tokeny autoryzacyjne JWT powinny być ważne maksymalnie przez 60 minut od ostatniego żądania użytkownika.

Spójność - W projektowanej aplikacji proces usuwania plików przez użytkowników powinien opierać się na mechanizmie softdelete. Zamiast natychmiastowego, fizycznego usuwania plików z bazy danych lub systemu plików, system powinien jedynie zmieniać status danego pliku w bazie danych. Dzięki temu aplikacja zapewni dodatkowy poziom bezpieczeństwa oraz pozwoli użytkownikom na ewentualne przywrócenie plików   
w przyszłości, minimalizując ryzyko przypadkowej utraty danych.

## 3.3. Technologie

JavaScript - to język programowania wysokiego poziomu używany głównie w tworzeniu dynamicznych stron internetowych. Początkowo zaprojektowany jako język skryptowy   
do przeglądarek, dziś jest jednym z najpopularniejszych języków wykorzystywanych nie tylko po stronie klienta, ale również po stronie serwera (dzięki Node.js). JavaScript pozwala   
na tworzenie interaktywnych aplikacji internetowych, manipulowanie strukturą HTML i CSS w czasie rzeczywistym oraz integrację z backendem aplikacji. Język wspiera różne paradygmaty programowania, w tym programowanie obiektowe i funkcjonalne.   
(Wikipedia - JavaScript 2024) (Justgeekit - node.js 2021)

Node.js - to środowisko uruchomieniowe JavaScript, które umożliwia wykonywanie kodu JavaScript po stronie serwera. Dzięki Node.js programiści mogą tworzyć aplikacje serwerowe w JavaScript, co pozwala na pełne wykorzystanie tego języka zarówno   
po stronie klienta, jak i serwera. Node.js jest oparty na silniku V8 Google, co sprawia,   
że jest bardzo wydajny. Obsługuje asynchroniczne operacje wejścia/wyjścia, co czyni   
go idealnym do obsługi dużych obciążeń i jednoczesnych żądań. Node.js wspiera moduły,   
co pozwala na łatwe zarządzanie i organizowanie kodu. (Justgeekit - node.js 2021)   
(Wikipedia - node.js 2024)

Express.js - to minimalistyczny framework dla Node.js, który umożliwia tworzenie aplikacji sieciowych i API w szybki i przejrzysty sposób. Zapewnia elastyczność i prostotę, umożliwiając programistom łatwe tworzenie aplikacji sieciowych dzięki gotowym funkcjom, takim jak routowanie, obsługa żądań HTTP i middleware. Express pozwala   
na rozwój zarówno prostych aplikacji, jak i złożonych aplikacji typu SPA i mikroserwisów, dzięki czemu jest popularny wśród programistów Node.js. (Aviary 2023)   
( Wikipedia - express.js 2024)

JWT (JSON Web Token) - to otwarty standard (RFC 7519) używany do bezpiecznego przekazywania informacji między stronami jako obiekt JSON. JWT jest szeroko stosowany   
do uwierzytelniania w aplikacjach internetowych, gdzie po pomyślnym zalogowaniu użytkownik otrzymuje token, który jest następnie wysyłany w każdym kolejnym żądaniu jako dowód autentyczności. Tokeny są bezpieczne, ponieważ są podpisane kryptograficznie, dzięki czemu nie można ich łatwo podrobić, co sprawia, że są one często wykorzystywane   
do tworzenia bezpiecznych sesji w aplikacjach sieciowych. (Wikipedia - JWT 2024) (devszczepaniak 2023)

MySQL - to system zarządzania relacyjnymi bazami danych (RDBMS), który wykorzystuje Structured Query Language (SQL) do manipulowania danymi. Został zaprojektowany przez szwedzką firmę MySQL AB w 1995 roku, a jego rozwój jest obecnie kontynuowany przez Oracle Corporation, która przejęła firmę w 2010 roku. MySQL jest znany ze swojej wydajności, niezawodności i łatwości użytkowania. MySQL jest dostępny na licencji GNU General Public License, co oznacza, że użytkownicy mogą go swobodnie pobierać, używać i modyfikować bez ponoszenia kosztów licencyjnych. Dostępna jest również komercyjna edycja MySQL, która zapewnia dodatkowe funkcje i wsparcie, idealne dla firm poszukujących zaawansowanych rozwiązań i bezpieczeństwa operacyjnego. System obsługuje różne typy tabel, które wykorzystują różne silniki pamięci masowej, takie jak InnoDB, który zapewnia obsługę transakcji, oraz MyISAM, znany z wydajnego indeksowania danych. MySQL oferuje również funkcje replikacji, które są kluczowe dla ciągłości biznesowej   
i skalowalności systemu, a także partycjonowanie tabel, które ułatwia zarządzanie dużymi zbiorami danych. (Wikipedia - mysql 2024)( Oracle 2024)

Algorytm AES - (Advanced Encryption Standard) to zaawansowany standard szyfrowania symetrycznego, opracowany w 1998 roku przez belgijskich kryptografów Joan Daemen   
i Vincent Rijmen, znany również jako algorytm "Rijndael". Został zaakceptowany przez amerykański Narodowy Instytut Standaryzacji i Technologii (NIST) w 2001 roku jako standard szyfrowania i obecnie jest używany na całym świecie do ochrony danych. AES działa   
na zasadzie kluczy symetrycznych, co oznacza, że ten sam klucz służy zarówno do szyfrowania, jak i deszyfrowania danych. Algorytm operuje na blokach danych o długości 128 bitów   
i obsługuje klucze o długości 128, 192 lub 256 bitów. Struktura algorytmu opiera się na wielu rundach, w których wykonywane są operacje takie jak zamiana bajtów według tablicy S-box, przesuwanie wierszy w macierzy danych, mieszanie kolumn macierzy oraz dodawanie klucza rundy za pomocą operacji XOR. Dzięki swojej odporności na ataki kryptograficzne AES zapewnia wysoki poziom bezpieczeństwa, co czyni go jednym z najczęściej stosowanych algorytmów szyfrowania w aplikacjach internetowych, bankowości i przechowywaniu danych. (Wikipedia - AES 2024) (Webporadnik 2025)

Google Authenticator- to aplikacja mobilna służąca do generowania jednorazowych kodów, które są używane w celu uwierzytelniania dwuskładnikowego (2FA, Two-Factor Authentication). Ta metoda uwierzytelniania zwiększa bezpieczeństwo, wymagając   
od użytkownika nie tylko podania hasła, ale także wprowadzenia jednorazowego kodu generowanego przez aplikację. Google Authenticator opiera swoje działanie na protokole TOTP (Time-Based One-Time Password) lub HOTP (HMAC-Based One-Time Password), które wykorzystują klucz tajny oraz aktualny czas systemowy do generowania kodów. Klucz tajny jest wprowadzany przez użytkownika poprzez skanowanie kodu QR lub ręczne wpisanie ciągu znaków w aplikacji. Na podstawie tego klucza i aktualnego czasu aplikacja generuje unikalne kody, które są ważne przez krótki czas, najczęściej około 30 sekund. Google Authenticator jest szeroko stosowany w różnych serwisach, takich jak Google, GitHub   
czy Dropbox, zapewniając skuteczny i prosty sposób ochrony kont użytkowników przed nieautoryzowanym dostępem. Dzięki swojej prostocie i efektywności aplikacja stała się jednym z najpopularniejszych rozwiązań wspierających bezpieczeństwo w sieci. (Wikipedia - Google Authentyicator 2024) (Nano.komputronik 2023)

MySQL Workbench - to narzędzie graficzne do zarządzania bazami danych MySQL. Zapewnia użytkownikom interfejs do tworzenia, edytowania, zarządzania i przeglądania struktury baz danych, a także pozwala na pisanie i wykonywanie zapytań SQL. Workbench jest narzędziem wszechstronnym, umożliwiającym tworzenie kopii zapasowych, importowanie danych i administrowanie serwerem. Jest szczególnie przydatny dla programistów   
i administratorów baz danych, którzy mogą w prosty sposób wizualizować struktury baz danych oraz optymalizować ich wydajność. (Wikipedia - MySQL Workbench 2024)   
(Boringowl 2024)

Git - to rozproszony system kontroli wersji zaprojektowany w celu umożliwienia programistom efektywnego zarządzania zmianami w kodzie źródłowym podczas pracy   
nad projektami. Stworzony przez Linusa Torvaldsa w 2005 roku, Git pozwala   
na przechowywanie historii zmian lokalnie, umożliwiając szybką pracę nad kodem nawet   
w trybie offline. Jego główną cechą jest to, że każdy, kto pobierze kopię repozytorium, otrzymuje pełną historię zmian, co oznacza, że każde repozytorium może służyć jako pełna kopia zapasowa. Git jest niezwykle elastyczny w zarządzaniu wieloma gałęziami kodu, umożliwiając eksperymentowanie, rozwój funkcji i naprawianie błędów niezależnie   
od głównej linii kodu. (Wikipedia - Git 2024) (git-scm 2024)

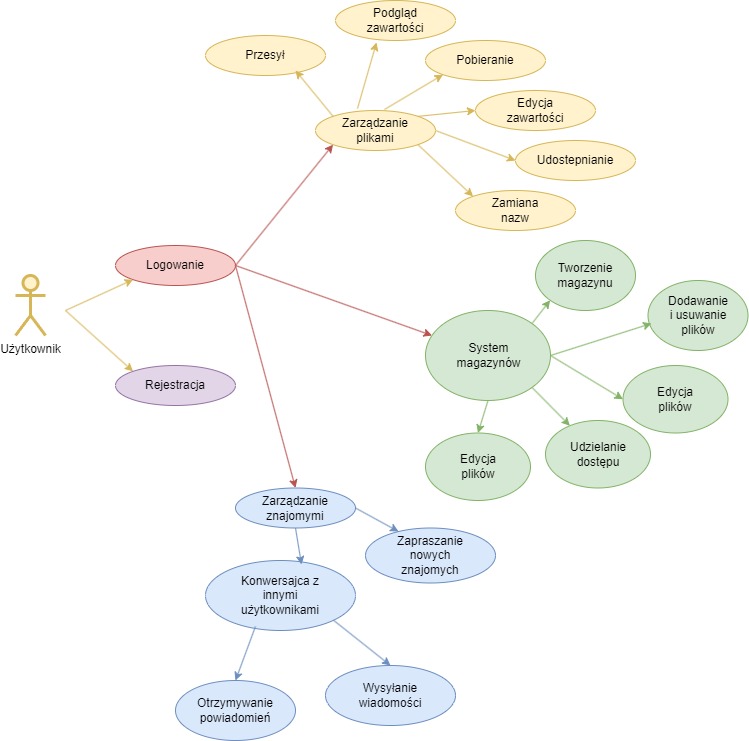
GitHub - to internetowa platforma hostingowa wykorzystująca Git do zarządzania projektami oprogramowania. Uruchomiony w 2008 roku przez Toma Prestona-Wernera, Chrisa Wanstratha i PJ Hyetta, GitHub pozwala programistom i firmom przechowywać, zarządzać   
i kontrolować ich projekty kodu źródłowego. Jest to również społeczność,   
w ramach której deweloperzy mogą współpracować nad projektami, udostępniając kod, pracując nad poprawkami i opracowując nowe funkcje. GitHub oferuje różne funkcje, takie   
jak śledzenie zgłoszeń, pull requesty, zarządzanie projektami i wiki dla każdego repozytorium. (Wikipedia - Github 2024)

Draw.io - to narzędzie do tworzenia diagramów online, które oferuje szeroki zakres możliwości wizualizacji danych i procesów. Jest to aplikacja internetowa, która umożliwia użytkownikom łatwe tworzenie diagramów, takich jak diagramy przepływu pracy,   
schematy blokowe, diagramy sieciowe, mapy myśli i wiele innych.   
(Wikipedia - diagrams.net 2024)

## 3.4. Diagramy

### 3.4.1. Diagram przypadków użycia

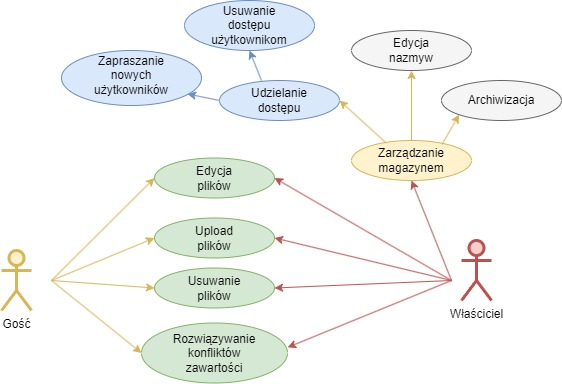
Rysunek 1 przedstawia kluczowe funkcjonalności, które powinna oferować projektowana aplikacja webowa, oraz relacje pomiędzy użytkownikiem a poszczególnymi modułami systemu. Centralnym elementem jest logowanie, które będzie stanowić punkt wyjścia   
do korzystania z pozostałych funkcji aplikacji. Proces ten powinien zapewnić użytkownikowi bezpieczny dostęp do systemu po wcześniejszej rejestracji, podczas której użytkownik będzie miał możliwość utworzenia konta. Rejestracja będzie niezbędnym krokiem, aby zidentyfikować użytkownika i przypisać mu dostęp do zasobów aplikacji. Po zalogowaniu użytkownik powinien zyskać dostęp do modułu zarządzania plikami, który będzie podstawowym obszarem funkcjonalnym aplikacji. W ramach tego modułu użytkownik będzie mógł przesyłać pliki do systemu, pobierać je na swoje urządzenie oraz przeglądać   
ich zawartość. Aplikacja powinna również umożliwić edytowanie plików tekstowych, zmianę ich nazw oraz udostępnianie plików innym użytkownikom. Kolejnym modułem będzie system magazynów, który umożliwi organizację plików. System magazynów powinien oferować funkcje dodawania oraz usuwania plików, a także edycji ich zawartości. Ważnym elementem tej funkcjonalności będzie zarządzanie dostępem do magazynów. Właściciel magazynu będzie mógł udzielać dostępu innym użytkownikom. Ważną częścią projektowanego systemu będzie również moduł zarządzania znajomymi, który będzie wspierał nawiązywanie relacji między użytkownikami aplikacji. Użytkownicy powinni mieć możliwość wyszukiwania innych osób w systemie, zapraszania ich do listy znajomych oraz zarządzania swoimi kontaktami. System powinien pozwalać na prowadzenie konwersacji z innymi użytkownikami, co będzie istotnym elementem komunikacji w kontekście współdzielenia plików i magazynów. Funkcjonalność   
ta ułatwi nawiązywanie współpracy i usprawni komunikację pomiędzy użytkownikami systemu.



Rysunek 1. Diagram przypadków użycia – użytkownik

[źródło: Opracowanie własne]

Rysunek 2 przedstawia funkcjonalności związane z systemem magazynów oraz role użytkowników, którzy mogą z niego korzystać. W systemie wyróżniono dwie główne   
role: Właściciela magazynu oraz Gościa, z których każda posiada określone uprawnienia   
i zakres działań. Właściciel magazynu otrzyma pełną kontrolę nad zarządzaniem magazynem   
i jego zawartością. Podstawową funkcjonalnością dostępną dla właściciela będzie zarządzanie magazynem, które obejmuje takie operacje jak edycja nazwy magazynu oraz archiwizacja. Dodatkowo właściciel powinien mieć możliwość udzielania dostępu innym użytkownikom oraz usuwania dostępu w razie potrzeby. Po udzieleniu dostępu, użytkownicy posiadający rolę Gościa zyskują uprawnienia do pracy z plikami w magazynie. Goście mogą wykonywać operacje takie jak edycja plików, przesyłanie nowych plików do magazynu, oraz usuwanie plików. Kluczową funkcjonalnością będzie także rozwiązywanie konfliktów zawartości, które może wystąpić w sytuacji, gdy kilka osób jednocześnie pracuje nad tym samym plikiem. Mechanizm ten umożliwi porównywanie różnic w wersjach plików i wybór odpowiedniej zawartości, co zapewni spójność danych. Obie role użytkowników otrzymają dostęp do operacji związanych z plikami, jednak to Właściciel posiada nadrzędne uprawnienia   
do zarządzania dostępem, edycji magazynu oraz jego konfiguracji.

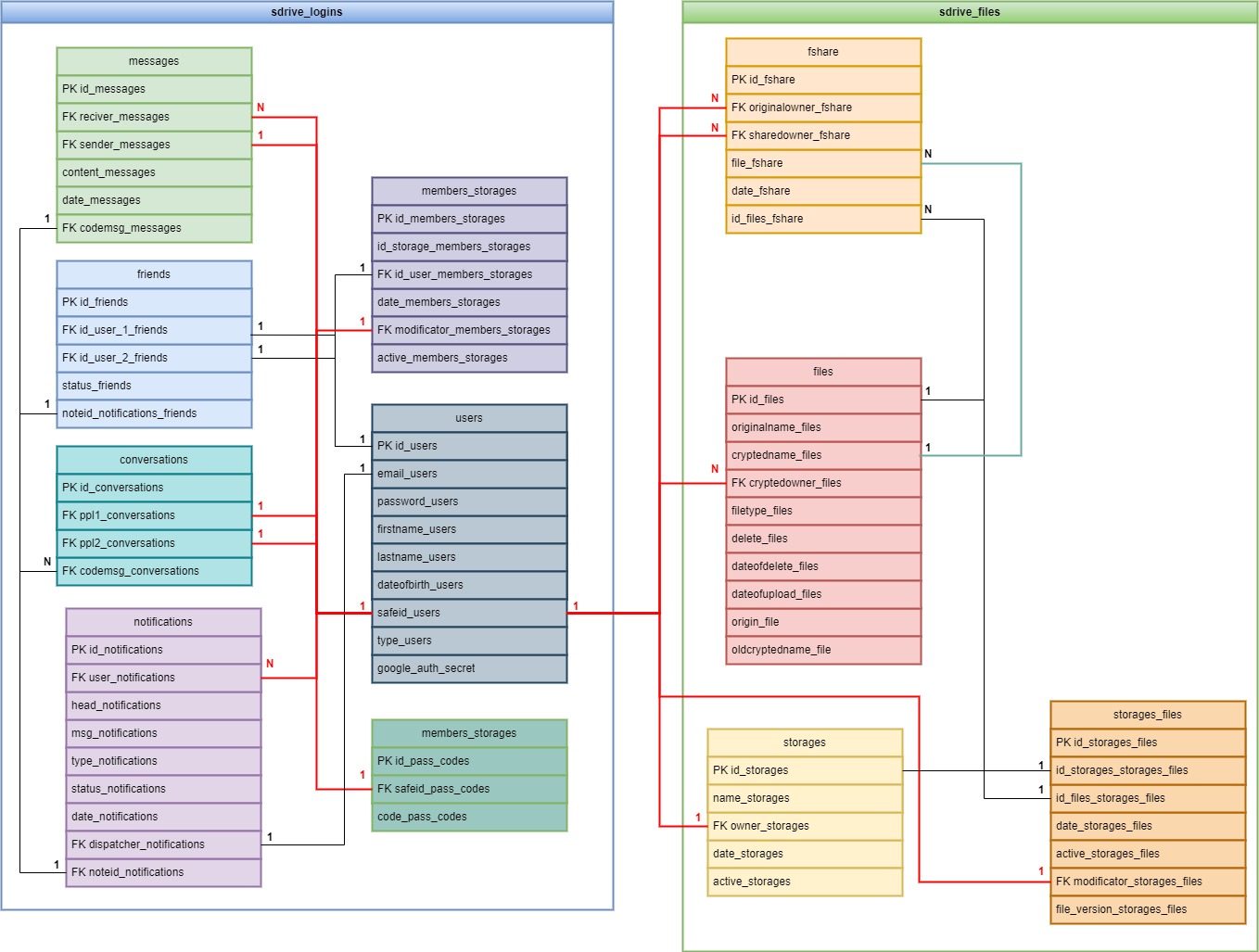


Rysunek 2. Diagram przypadków użycia – magazyny

[źródło: Opracowanie własne]

### 3.4.2. Diagram ERD

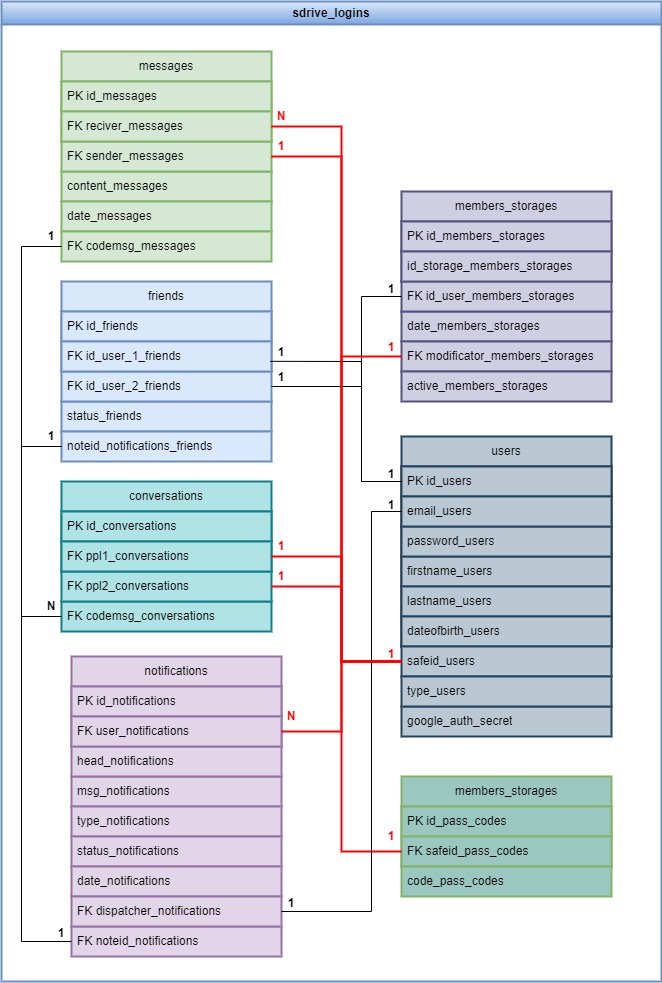
Rysunek 3 ilustruje diagram ERD (Entity-Relationship Diagram), który przedstawia   
pełną strukturę baz danych projektu sDrive. Projekt korzysta z dwóch oddzielnych   
baz danych: sdrive\_logins oraz sdrive\_files.



Rysunek 3. Diagram ERD

[źródło: Opracowanie własne]

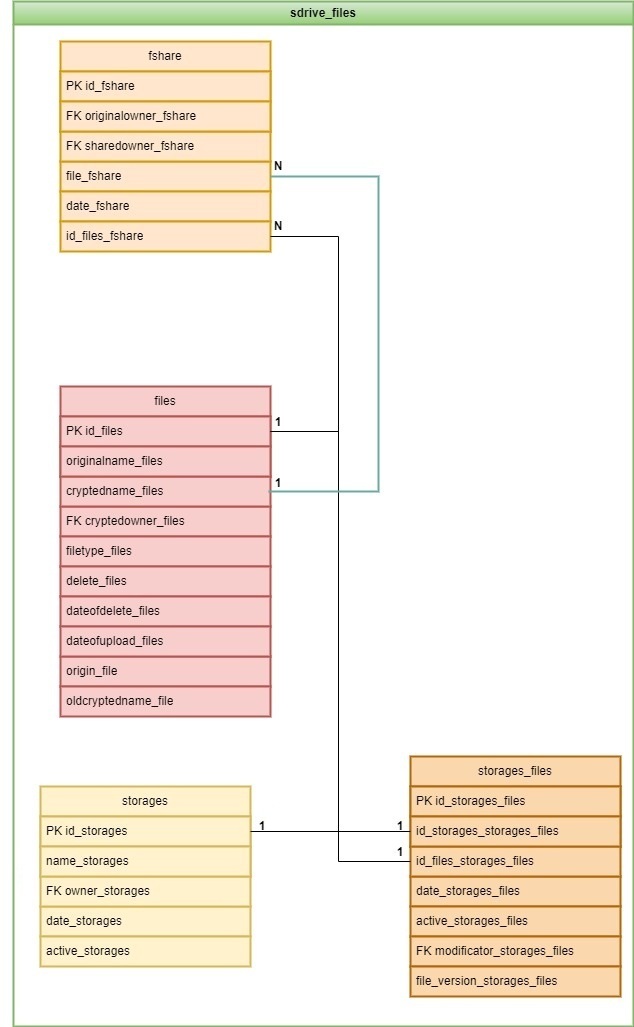
Rysunek 4 przedstawia diagram ERD bazy danych sdrive\_logins systemu sDrive, składający się z siedmiu tabel odpowiedzialnych za przechowywanie danych użytkowników oraz obsługę kluczowych funkcji systemu. Baza zarządza informacjami uwierzytelniającymi, relacjami znajomych, wiadomościami, powiadomieniami, konwersacjami oraz członkami magazynów. Tabele przechowują takie dane jak adresy e-mail, zaszyfrowane hasła, treści wiadomości, statusy relacji, a także klucze zabezpieczające.



Rysunek 4. Diagram ERD – baza sdrive\_logins

[źródło: Opracowanie własne]

Rysunek 5 przedstawia diagram ERD bazy danych sdrive\_files systemu sDrive, który odpowiada za zarządzanie plikami oraz ich powiązaniami z magazynami i udostępnieniami. Struktura bazy składa się z tabel przechowujących informacje o plikach, takich jak oryginalne i szyfrowane nazwy, typy, daty przesłania oraz status usunięcia. Tabele zarządzają również relacjami między plikami a magazynami, wspierając wersjonowanie oraz kontrolę dostępu. System umożliwia udostępnianie plików pomiędzy użytkownikami, przechowując szczegóły dotyczące właścicieli, statusów plików oraz ich historii.

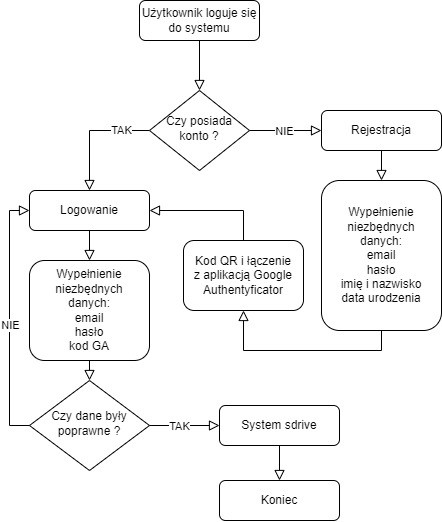


Rysunek 5. Diagram ERD – baza sdrive\_files

[źródło: Opracowanie własne]

### 3.4.3. Diagramy aktywności

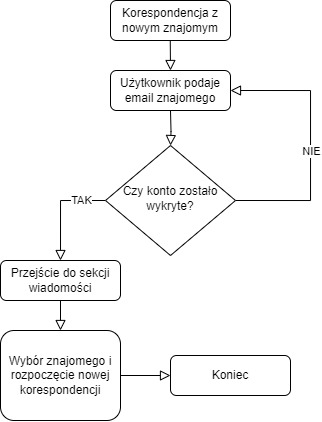
Rysunek 6 przedstawia proces logowania użytkownika do systemu, który rozpoczyna   
się od próby uzyskania dostępu przez użytkownika. Pierwszym krokiem jest weryfikacja,   
czy użytkownik posiada już konto w systemie. W przypadku odpowiedzi twierdzącej, użytkownik przechodzi do etapu logowania, gdzie wprowadza niezbędne dane, takie jak adres e-mail hasło i kod Google Authenticator. Jeżeli podane dane są poprawne, użytkownik uzyskuje dostęp do systemu sDrive, co kończy proces logowania. Jeśli dane okażą się niepoprawne, system ponownie kieruje użytkownika do etapu logowania, umożliwiając ponowne wprowadzenie informacji. W sytuacji, gdy użytkownik nie posiada konta, przechodzi on przez proces rejestracji, który polega na wypełnieniu formularza rejestracyjnego wymagającego podania takich informacji jak adres e-mail, hasło, imię, nazwisko oraz data urodzenia. Następnie należy połączyć konto z aplikacją Google Authenticator. Po zakończeniu rejestracji użytkownik może przystąpić do logowania, aby uzyskać dostęp do systemu.



Rysunek 6. Diagram aktywności – logowanie do systemu

[źródło: Opracowanie własne]

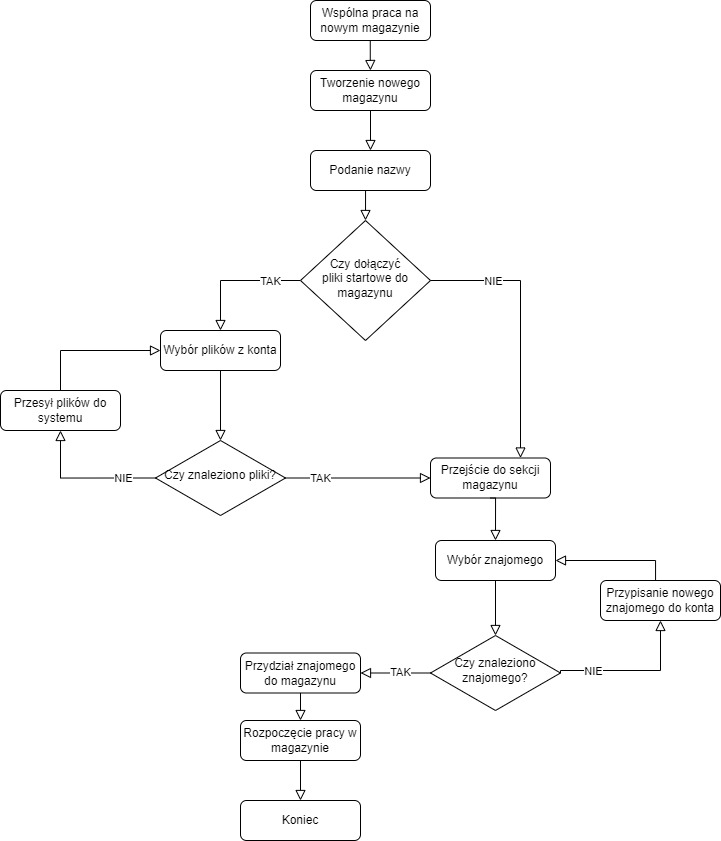
Rysunek 7 przedstawia proces nawiązania korespondencji z nowym znajomym   
w systemie. Całość rozpoczyna się od podania przez użytkownika adresu e-mail znajomego. System weryfikuje, czy konto odpowiadające podanemu e-mailowi istnieje. W przypadku,   
gdy konto zostanie wykryte, użytkownik przechodzi do sekcji wiadomości, gdzie może wybrać znajomego z listy i rozpocząć nową korespondencję. Jeśli system nie znajdzie konta odpowiadającego podanemu adresowi e-mail, proces zostaje przerwany, a użytkownik   
ma możliwość ponownego podania danych znajomego.



Rysunek 7. Diagramy aktywności – korespondencja ze znajomym

[źródło: Opracowanie własne]

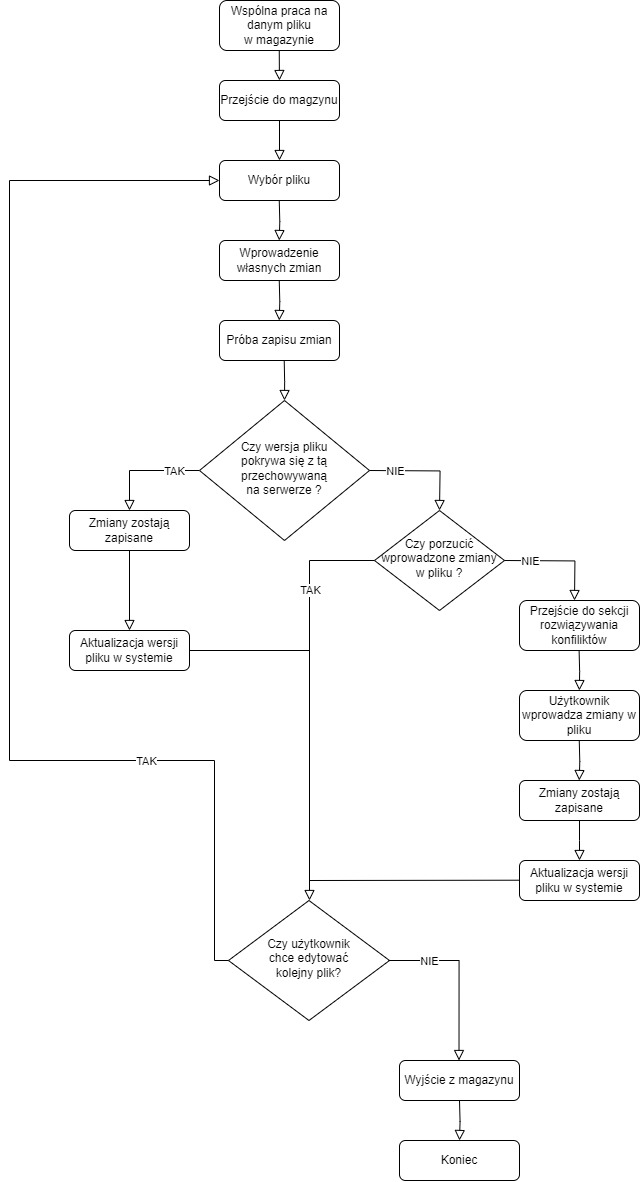
Rysunek 8 przedstawia proces tworzenia nowego magazynu w kontekście współpracy   
z innymi użytkownikami. Całość rozpoczyna się od podania nazwy nowego magazynu. Następnie użytkownik decyduje, czy chce dołączyć pliki startowe do magazynu.   
Jeśli odpowiedź jest twierdząca, system przechodzi do etapu wyboru plików z konta użytkownika, po czym następuje ich przesył do systemu. W przypadku braku plików użytkownik ma możliwość ponownej próby wyboru. Jeśli użytkownik rezygnuje z dodania plików, system przechodzi do sekcji magazynu, gdzie można przypisać znajomego   
do współpracy. W dalszym etapie użytkownik wybiera znajomego. System weryfikuje,   
czy dany znajomy został znaleziony w bazie danych. Jeśli znajomy istnieje, zostaje   
on przypisany do magazynu, co umożliwia rozpoczęcie wspólnej pracy. W przeciwnym wypadku użytkownik może przypisać nowego znajomego do swojego konta. Proces kończy się, gdy znajomy zostanie przydzielony, a praca w magazynie może się rozpocząć.



Rysunek 8. Diagramy aktywności - praca na nowym magazynie

[źródło: Opracowanie własne]

Rysunek 9 przedstawia proces wspólnej pracy nad plikiem w magazynie, uwzględniając zarówno edycję, jak i rozwiązywanie potencjalnych konfliktów. Proces rozpoczyna   
się od przejścia użytkownika do magazynu, gdzie dokonuje on wyboru pliku i wprowadza własne zmiany. W momencie próby zapisu system sprawdza, czy wersja pliku pokrywa   
się z aktualną wersją na serwerze. Jeśli wersje są zgodne, zmiany zostają zapisane, a system dokonuje aktualizacji wersji pliku. W przypadku wykrycia niezgodności użytkownik podejmuje decyzję, czy porzucić wprowadzone zmiany. Jeśli decyduje się na ich odrzucenie, wraca do wcześniejszego etapu. Jeśli natomiast wybierze kontynuację pracy, system przenosi go do sekcji rozwiązywania konfliktów. Tam użytkownik może wprowadzić odpowiednie modyfikacje, które następnie zostają zapisane i zaktualizowane w systemie. Po zakończeniu operacji użytkownik ma możliwość wyboru kolejnego pliku do edycji. Jeśli zdecyduje   
się na kontynuację, proces powtarza się od etapu wyboru pliku. W przeciwnym przypadku użytkownik wychodzi z magazynu, kończąc pracę.



Rysunek 9. Diagramy aktywności - praca na wspólnym pliku w magazynie

[źródło: Opracowanie własne]

## 3.5. Scenariusze użycia

Scenariusze użycia aplikacji sDrive przedstawione w Tabelach 1-10 ilustrują kluczowe funkcjonalności systemu oraz typowe interakcje użytkowników z aplikacją. Opisane przypadki użycia obejmują zarówno podstawowe operacje, takie jak rejestracja, logowanie   
czy przesyłanie plików, jak i bardziej zaawansowane funkcje, takie jak współdzielenie zasobów, edycja plików w magazynach czy zarządzanie znajomymi. Każdy scenariusz został opisany, uwzględniając warunki początkowe, przebieg główny, alternatywne ścieżki oraz końcowe rezultaty. Scenariusze te odzwierciedlają projektowane możliwości systemu.

Tabela 1   
Scenariusz użycia - logowanie do systemu

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Logowanie do systemu |
| Nr. Scenariusza: | 01 |
| Aktorzy: | Użytkownik, system |
| Warunek początkowy: | Użytkownik posiada konto w systemie sdrive. |
| Przebieg główny: | Użytkownik podaje email i hasło. |
| Alternatywny przebieg: | Hasło lub email zostaje odrzucone, podjęcie ponownej próby. |
| Wynik: | Użytkownik zostaje zalogowany do systemu sDrive. |

[źródło: Opracowanie własne]

Tabela 2  
Scenariusz użycia - rejestracja w systemie sdrive

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Rejestracja w systemie sdrive |
| Nr. Scenariusza: | 02 |
| Aktorzy: | Użytkownik, system |
| Warunek początkowy: | Brak --- |
| Przebieg główny: | Użytkownik przechodzi do sekcji rejestracji, podaje email, imię i nazwisko oraz datę urodzenia. Następnie łączy konto z aplikacją Google Authenticator. |
| Alternatywny przebieg: | Wprowadzono błędne dane, należy ponowić próbę rejestracji. |
| Wynik: | Użytkownik tworzy konto w systemie sDrive. |

[źródło: Opracowanie własne]

Tabela 3   
Scenariusz użycia - przesył plików do systemu

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Przesył plików do systemu |
| Nr. Scenariusza: | 003 |
| Aktorzy: | Użytkownik, system |
| Warunek początkowy: | Użytkownik ma konto w systemie i jest zalogowany. |
| Przebieg główny: | Użytkownik w sekcji przesyłu plików wybiera plik z komputera i wysyła  go do swojego konta. |
| Alternatywny przebieg: | Nie udało się przesłać pliku, podjęcie ponownej próby. |
| Wynik: | Udało się użytkownikowi przesłać plik z komputera do systemu. |

[źródło: Opracowanie własne]

Tabela 4   
Scenariusz użycia - udostępnianie plików w systemie

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Udostępnianie plików w systemie |
| Nr. Scenariusza: | 04 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 i 2 mają konto w systemie i są znajomymi,  użytkownik 1 musi być zalogowany. |
| Przebieg główny: | Użytkownik 1 z listy swoich plików lub udostępnianych mu plików  wybiera plik i opcję „udostępnij”, następnie z listy znajomych wybiera  użytkownika 2 i klika przycisk udostępnij. |
| Alternatywny przebieg: | Nie udało się udostępnić pliku, podjęcie ponownej próby. |
| Wynik: | Użytkownik 1 udostępnia plik użytkownikowi 2 |

[źródło: Opracowanie własne]

Tabela 5   
Scenariusz użycia - zapraszanie nowych znajomych

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Zapraszanie nowych znajomych |
| Nr. Scenariusza: | 05 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 i 2 mają konto w systemie, użytkownik 1 musi być zalogowany. |
| Przebieg główny: | Użytkownik 1 w sekcji znajomych wpisuje email użytkownika 2, wyświetla się okno ze szczegółowymi danymi użytkownika 2, użytkownik 1 wybiera opcję „zaproś do znajomych”. |
| Alternatywny przebieg: | Nie udało się zaprosić użytkownika 2 do znajomych (najprawdopodobniej podano zły adres email), podjęcie ponownej próby. |
| Wynik: | Użytkownik 2 i użytkownik 1 zostają znajomymi. |

[źródło: Opracowanie własne]

Tabela 6   
Scenariusz użycia - konwersacja ze znajomym

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Konwersacja ze znajomym |
| Nr. Scenariusza: | 06 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 i 2 mają konto w systemie i są znajomymi,  użytkownik 1 musi być zalogowany. |
| Przebieg główny: | Użytkownik 1 po zalogowaniu przechodzi do sekcji wiadomości, wybiera opcję rozpoczęcia nowej konwersacji, wybiera z listy znajomych użytkownika 2, wpisuje treść wiadomości i wybiera opcję „wyślij wiadomość”. |
| Alternatywny przebieg: | Nie udało się przesłać wiadomości, podjęcie ponownej próby. |
| Wynik: | Użytkownik 2 i użytkownik 1 wymieniają między sobą korespondencję. |

[źródło: Opracowanie własne]

Tabela 7   
Scenariusz użycia - tworzenie magazynu

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Tworzenie magazynu |
| Nr. Scenariusza: | 07 |
| Aktorzy: | Użytkownik 1, system |
| Warunek początkowy: | Użytkownicy ma konto w systemie, użytkownik 1 musi być zalogowany, użytkownik 1 musi mieć przesłany przynajmniej jeden plik do systemu sDrive. |
| Przebieg główny: | Użytkownik 1 po zalogowaniu wybiera opcję „utwórz” w sekcji magazynów, podaje nazwę magazynu, wybiera pliki które mają być załączone do danego magazynu i na koniec klika „Stwórz magazyn”. |
| Alternatywny przebieg: | Nie udało się stworzenie magazynu, podjęcie ponownej próby. |
| Wynik: | Użytkownik 1 tworzy magazyn |

[źródło: Opracowanie własne]

Tabela 8   
Scenariusz użycia - edycja pliku w magazynie

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Edycja pliku w magazynie |
| Nr. Scenariusza: | 08 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 i 2 mają konto w systemie i są znajomymi,  użytkownik 1 musi być zalogowany, użytkownik 1 musi być przypisany przynajmniej do jednego magazynu. |
| Przebieg główny: | Użytkownik 1 przechodzi do sekcji magazynów, wybiera konkretny magazyn, w magazynie wybiera plik, dokonuje zmian, zapisuje zmiany. |
| Alternatywny przebieg: | Wykryto rozbieżność wersji pliku podczas zapisu (najprawdopodobniej użytkownik 2 dokonał zmian i zapisał je w pliku zanim użytkownik 1 zdołał zapisać swoje), użytkownik wybiera opcję rozwiązania konfliktu, przechodzi do sekcji konfliktów, dokonuje odpowiednich zmian w pliku i go zapisuje. |
| Wynik: | Użytkownik 1 edytuje plik w magazynie |

[źródło: Opracowanie własne]

Tabela 9   
Scenariusz użycia - przypisanie znajomego do magazynu

|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Przypisanie znajomego do magazynu |
| Nr. Scenariusza: | 09 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 i 2 mają konto w systemie i są znajomymi,  użytkownik 1 musi być zalogowany, użytkownik 1 musi być właścicielem przynajmniej jednego magazynu. |
| Przebieg główny: | Użytkownik 1 po zalogowaniu przechodzi do swojego magazynu, wybiera opcję udostępniania, z listy znajomych wybiera użytkownika 2 i klika opcję „Udostępnij”. |
| Alternatywny przebieg: | Nie udało się udostępnić dostępu do magazynu, podjęcie ponownej próby. |
| Wynik: | Użytkownik 1 udziela dostępu do swojego magazynu |

[źródło: Opracowanie własne]

Tabela 10   
Scenariusz użycia - przypisywanie nowych plików do magazynu

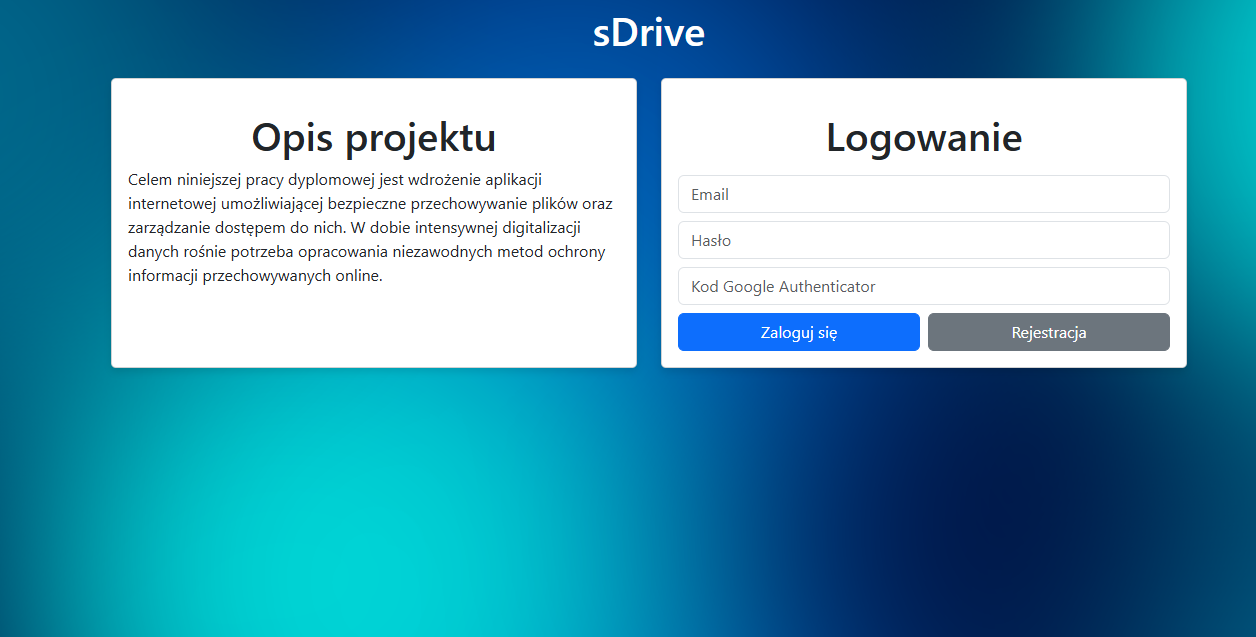
|  |  |
| --- | --- |
| Element | Opis |
| Nazwa scenariusza: | Przypisywanie nowych plików do magazynu |
| Nr. Scenariusza: | 10 |
| Aktorzy: | Użytkownik 1, użytkownik 2, system |
| Warunek początkowy: | Użytkownicy 1 ma konto w systemie, użytkownik 1 musi być zalogowany, użytkownik 1 musi być przypisany przynajmniej do jednego magazynu. |
| Przebieg główny: | Użytkownik 1 po zalogowaniu się przechodzi do magazynu, wybiera opcję dodania pliku, z listy wybiera plik i klika opcję „Dodaj do magazynu”. |
| Alternatywny przebieg: | Nie udało się dodać pliku do magazynu, podjęcie ponownej próby. |
| Wynik: | Użytkownik 1 dodaje plik do magazynu. |

[źródło: Opracowanie własne]

# 4. Implementacja

## 4.1. System logowania i rejestracji

System logowania i rejestracji w aplikacji sDrive został zaimplementowany   
z wykorzystaniem mechanizmów umożliwiających autoryzację użytkowników. Rejestracja polega na wypełnieniu formularza, w którym użytkownik podaje wymagane dane, takie jak adres e-mail i hasło. Po przesłaniu formularza generowany jest kod QR, który służy   
do integracji z aplikacją Google Authenticator, co wprowadza dwuskładnikowe uwierzytelnianie (2FA). Logowanie odbywa się poprzez wprowadzenie adresu e-mail, hasła oraz jednorazowego kodu generowanego przez Google Authenticator. Dane przesyłane   
do serwera są szyfrowane, a hasła zapisywane są w bazie danych w postaci haszowanej przy użyciu algorytmu bcryptjs. Weryfikacja kodu QR i jednorazowego hasła odbywa się w czasie rzeczywistym, co zapewnia dodatkowy poziom zabezpieczeń. Na rysunkach 10–12 przedstawiono poszczególne elementy systemu, w tym widok formularza rejestracji, ekranu logowania oraz integracji kodu QR z Google Authenticator.



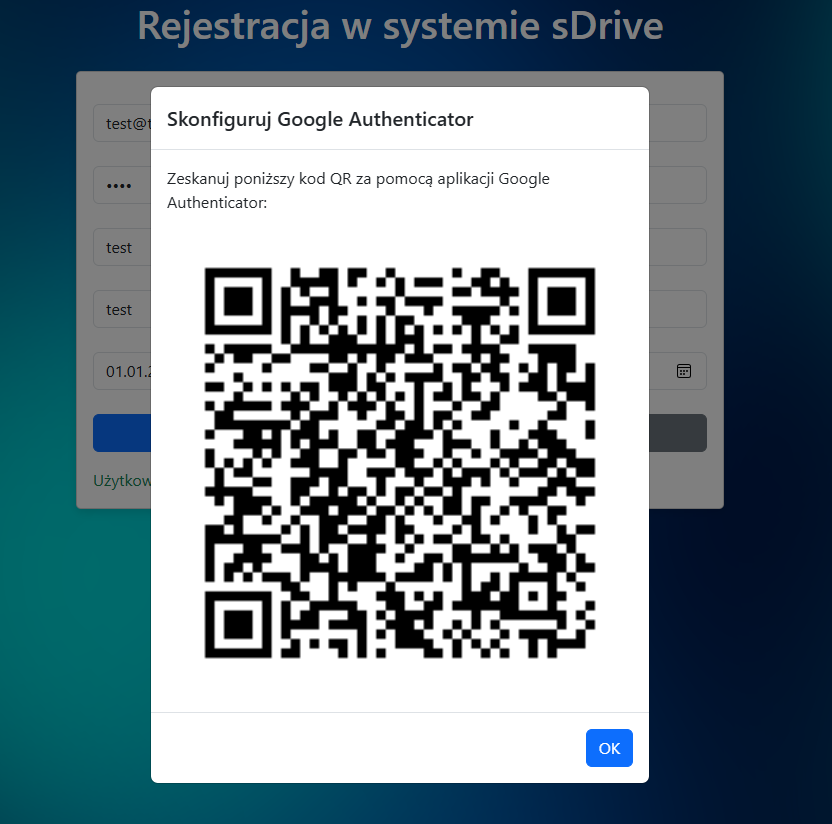
Rysunek 10. System logowania i rejestracji - ekran logowania

[źródło: Opracowanie własne]



Rysunek 11. System logowania i rejestracji - ekran rejestracji

[źródło: Opracowanie własne]



Rysunek 12. System logowania i rejestracji - kod QR

[źródło: Opracowanie własne]

Fragment kodu 1 przedstawia funkcję odpowiedzialną za obsługę logowania użytkownika   
w aplikacji webowej. Funkcja asynchroniczna login przyjmuje żądanie z danymi logowania   
(e-mail i hasło) przesyłanymi przez użytkownika i weryfikuje je w bazie danych. Najpierw sprawdza, czy dane zostały podane, a następnie wykonuje zapytanie SQL, aby pobrać odpowiadający użytkownika rekord z bazy danych. W przypadku poprawnych danych funkcja porównuje hasło z haszowanym hasłem w bazie za pomocą biblioteki bcryptjs. Jeśli logowanie się powiedzie, generowany jest token JWT z identyfikatorem użytkownika, który zostaje zapisany w ciasteczku z ustawionymi parametrami, takimi jak czas wygaśnięcia   
i dostępność jedynie po stronie serwera (httpOnly). Ostatecznie funkcja zwraca odpowiedź   
o sukcesie wraz z informacją o przekierowaniu na stronę domową, a w przypadku błędnych danych zwracany jest odpowiedni komunikat o błędzie.

const login = async (*req*, *res*) => {

    const { email, password, totp } = *req*.body;

    console.log("Odebrane dane logowania:", *req*.body);

    if (!email || !password || !totp) {

        return *res*.json({ status: "error", error: "Podaj swój email, hasło i kod Google Authenticator" });

    } else {

        dbLogins.query('SELECT email\_users, password\_users, safeid\_users, id\_users, google\_auth\_secret FROM users WHERE email\_users = ?', [email], async (*err*, *result*) => {

            if (*err*) throw *err*;

            if (!*result*[0] || !await bcryptjs.compare(password, *result*[0].password\_users)) {

                return *res*.json({ status: "error", error: "Niepoprawne hasło lub email" });

            } else {

                const safeId = *result*[0].safeid\_users;

                const googleAuthSecret = *result*[0].google\_auth\_secret;

                if (!googleAuthSecret) {

                    return *res*.json({ status: "error", error: "Brak skonfigurowanego Google Authenticator dla tego konta" });

               }

                const verified = speakeasy.totp.verify({

                    secret: googleAuthSecret,

                    encoding: "base32",

                    token: totp

                });

                if (!verified) {

                    return *res*.json({ status: "error", error: "Niepoprawny kod Google Authenticator" });

                }

                dbLogins.query('SELECT code\_pass\_codes FROM pass\_codes WHERE

safeid\_pass\_codes = ?', [safeId], async (*err*, *keyResult*) => {

                    if (*err*) throw *err*;

                    if (*keyResult*.length > 0) {

                        const key = Buffer.from(*keyResult*[0].code\_pass\_codes, "hex");

                        const userFolder = path.join(\_\_dirname, "..", "data", "users", safeId);

                        const sharesFolder = path.join(\_\_dirname, "..", "data", "shares", safeId);

                        decryptDirectory(userFolder, key);

                        decryptDirectory(sharesFolder, key);

                    }

                    const token = jwt.sign({ id: *result*[0].id\_users }, process.env.JWT\_SECRET, {

                        expiresIn: process.env.JWT\_EXPIRES

                    });

                    const cookieOptions = {

                        expires: new Date(Date.now() + process.env.COOKIE\_EXPIRES \* 24 \* 60 \* 60 \* 1000),

                        httpOnly: true

                    };

*res*.cookie("userRegistered", token, cookieOptions);

                    console.log("Ciasteczko ustawione: ", token);

                    return *res*.json({ status: "success", redirect: "/home" });

Fragment kodu 1. System logowania i rejestracji – logowanie

[źródło: Opracowanie własne]

Fragment kodu 2 przedstawia rejestrację nowego użytkownika w systemie. Najpierw sprawdzane jest, czy wszystkie wymagane pola zostały wypełnione (e-mail, hasło, imię, nazwisko, data urodzenia), a następnie system weryfikuje, czy podany adres e-mail już istnieje w bazie danych. Jeśli e-mail nie występuje, hasło użytkownika jest haszowane za pomocą biblioteki bcryptjs. Dodatkowo generowany jest unikalny identyfikator safeId, który jest sprawdzany pod kątem unikalności za pomocą pętli, aby upewnić się, że nie występuje on już w systemie. W końcu nowy użytkownik zostaje zapisany do bazy danych z odpowiednimi danymi. Funkcja zwraca komunikat o sukcesie rejestracji lub odpowiedni błąd, jeśli wystąpią nieprawidłowości, takie jak duplikacja e-maila lub błąd serwera.

const register = async (*req*, *res*) => {

    const { email, password: Npassword, firstName, lastName, birthDate } = *req*.body;

    if (!email || !Npassword || !firstName || !lastName || !birthDate) {

        return *res*.json({ status: "error", error: "Proszę wypełnić wszystkie pola" });

    } else {

        dbLogins.query('SELECT email\_users FROM users WHERE email\_users = ?', [email], async (*err*, *result*) => {

            if (*err*) throw *err*;

            if (*result*[0]) {

   return *res*.json({ status: "error", error: "Email już jest w systemie" });

            } else {

                const password = await bcrypt.hash(Npassword, 8);

                let safeId;

                let unique = false;

                while (!unique) {

                    safeId = generateSafeId();

                    unique = await isSafeIdUnique(safeId);}

                const { secret, qrCodeDataUrl } = await generateGoogleAuthSecret(safeId);

                dbLogins.query('INSERT INTO users SET ?', {

                    email\_users: email,

                    password\_users: password,

                    firstname\_users: firstName,

                    lastname\_users: lastName,

                    dateofbirth\_users: birthDate,

                    safeid\_users: safeId,

                    type\_users: "Normal",

                    google\_auth\_secret: secret.base32

                }, (*error*, *results*) => {

                    if (*error*) throw *error*;

                    return *res*.json({

                        status: "success",

                        success: "Użytkownik został zarejestrowany",

                        qrCode: qrCodeDataUrl

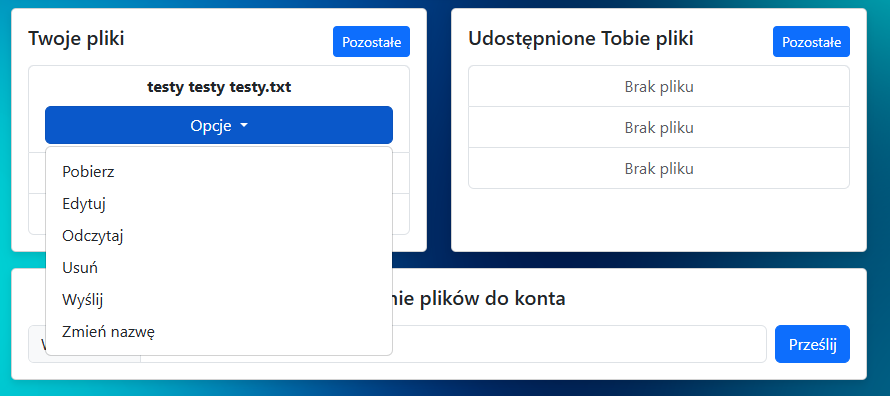
                    });

Fragment kodu 2. System logowania i rejestracji – rejestracja

[źródło: Opracowanie własne]

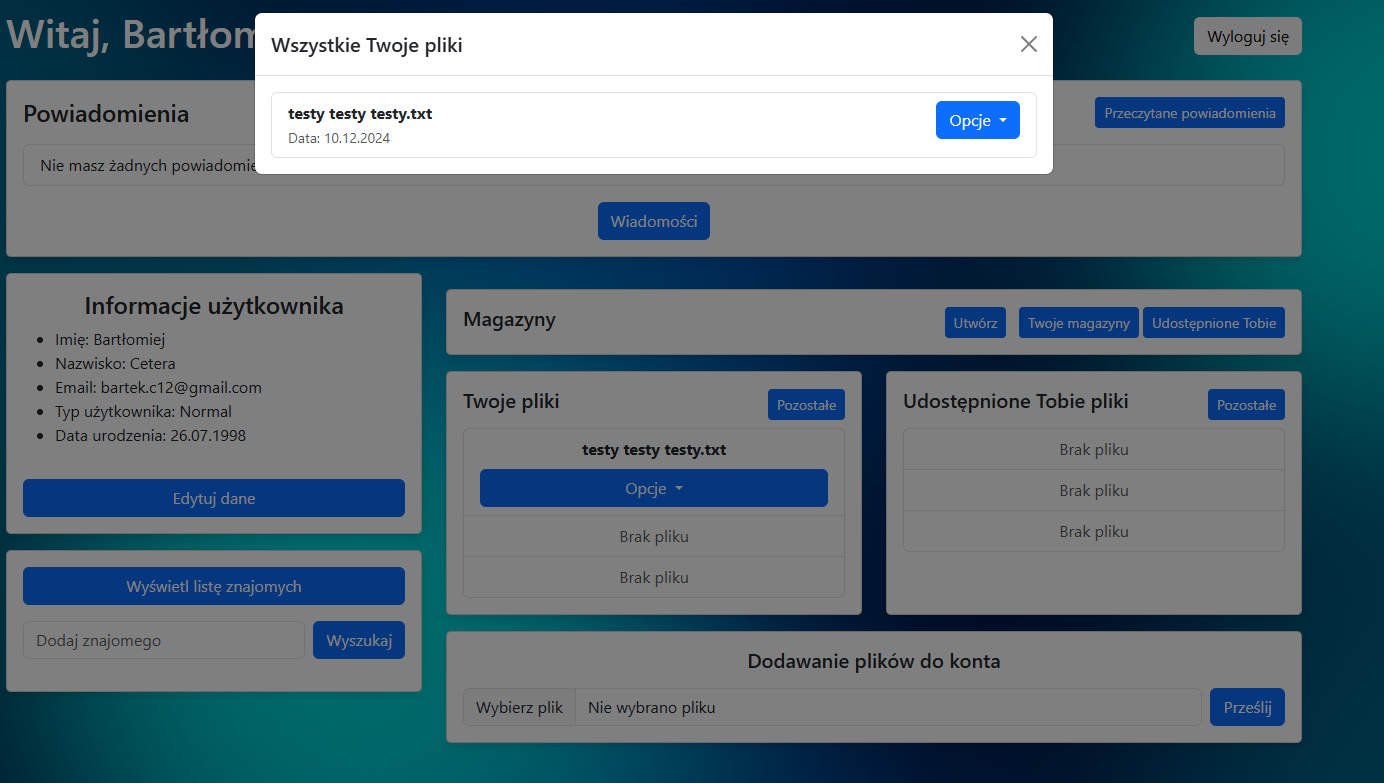
## 4.2. System zarządzania plikami

System zarządzania plikami w aplikacji sDrive został zaimplementowany w sposób umożliwiający użytkownikom wykonywanie operacji na przechowywanych danych. Użytkownicy mają możliwość przesyłania plików na swoje konta, pobierania ich, przeglądania zawartości, edytowania, usuwania oraz zmieniania nazw. Dodatkowo system pozwala   
na udostępnianie plików znajomym przypisanym do konta, a także zarządzanie plikami udostępnionymi przez innych użytkowników. Edycja plików jest możliwa bezpośrednio   
w systemie, co eliminuje potrzebę ponownego przesyłania zmodyfikowanych danych. Funkcja usuwania plików została zrealizowana z użyciem mechanizmu softdelete, który umożliwia odzyskanie przypadkowo usuniętych plików w określonym czasie. Na rysunkach   
13–15 przedstawiono zaimplementowane kluczowe elementy systemu zarządzania plikami, takie jak interfejs sekcji plików, widok szczegółowy dla zarządzania plikami oraz ekran edycji.



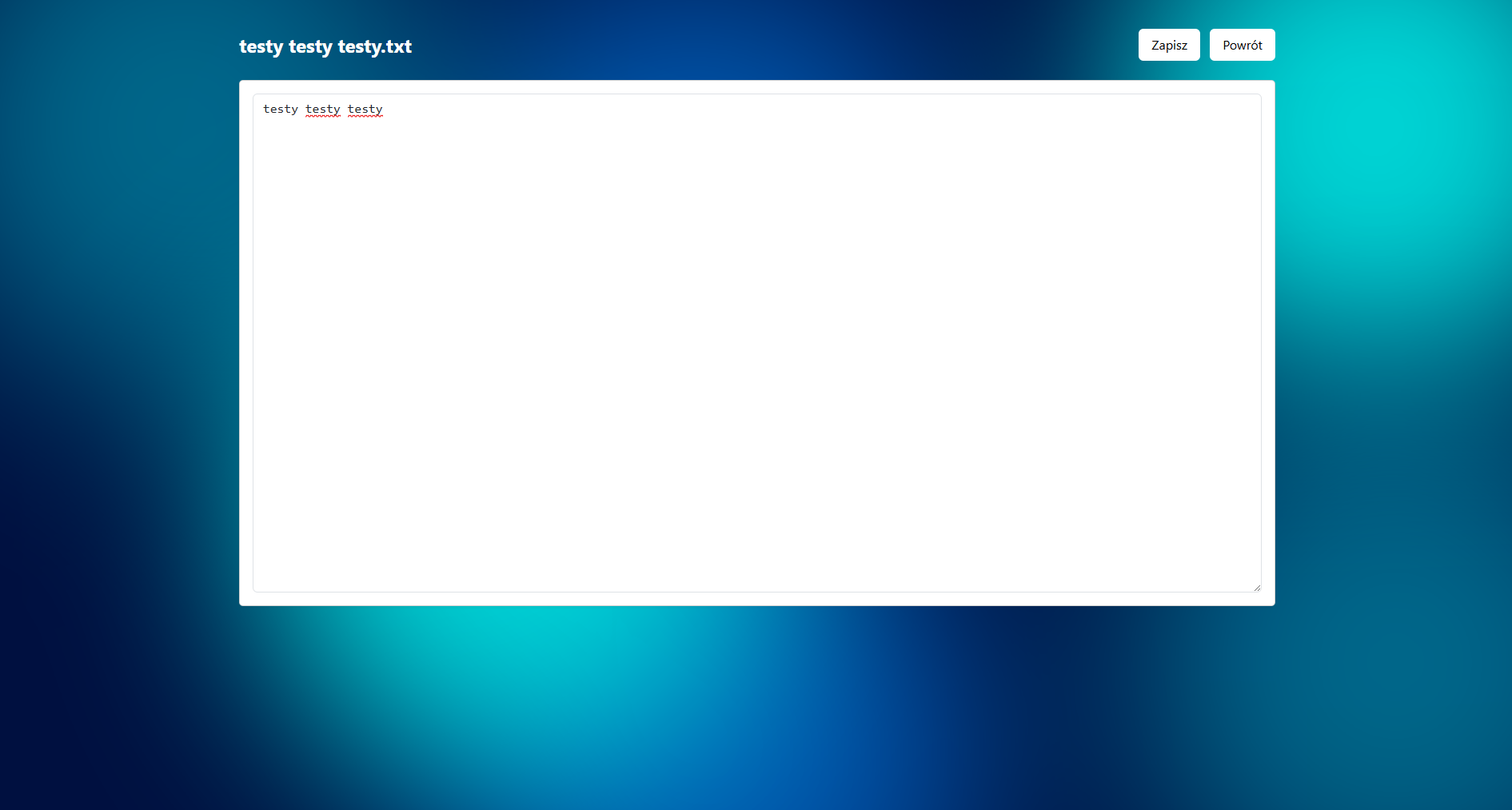
Rysunek 13. System zarządzania plikami - sekcja plików

[źródło: Opracowanie własne]



Rysunek 14. System zarządzania plikami - sekcja wszystkich plików

[źródło: Opracowanie własne]



Rysunek 15. System zarządzania plikami - edycja pliku

[źródło: Opracowanie własne]

Fragment kodu 3 przedstawia funkcję showUserFileContent, odpowiadającą   
za wyświetlanie zawartości pliku użytkownika na podstawie jego unikalnego identyfikatora safeId oraz nazwy pliku przesłanej w parametrach żądania. Najpierw wykonuje zapytanie   
do bazy danych, aby sprawdzić, czy plik o podanej zaszyfrowanej nazwie (cryptedname\_files) należy do użytkownika. Jeśli plik istnieje, funkcja pobiera jego oryginalną nazwę, określa rozszerzenie pliku oraz buduje ścieżkę do pliku w katalogu użytkownika. Następnie sprawdza, czy plik fizycznie istnieje na serwerze, i jeśli tak, wywołuje funkcję handleFileRead, która odpowiednio odczytuje plik i przesyła jego zawartość do klienta. W przypadku braku pliku   
w systemie plików lub bazie danych funkcja zwraca odpowiedź o błędzie z kodem 404. Funkcja zawiera także obsługę błędów serwera oraz informuje użytkownika o problemach z dostępem do pliku lub jego brakiem.

const showUserFileContent = (*req*, *res*) => {

    const safeId = *req*.user.safeid\_users;

    const filename = *req*.params.filename;

    dbFiles.query(

        'SELECT originalname\_files, cryptedname\_files, filetype\_files FROM files WHERE cryptedname\_files = ? AND cryptedowner\_files = ?',

        [filename, safeId],

        (*err*, *result*) => {

            if (*err*) {

                console.error('Błąd podczas pobierania pliku z bazy danych:', *err*);

                return *res*.status(500).json({ status: 'error', error: 'Błąd pobierania pliku z bazy danych' });

            }

            if (*result*.length > 0) {

                const originalName = *result*[0].originalname\_files;

                const fileExtension = path.extname(originalName).toLowerCase();

                const userDir = path.join(\_\_dirname, '..', 'data', 'users', safeId);

                const filePath = path.join(userDir, filename);

                if (fs.existsSync(filePath)) {

                    handleFileRead(filePath, fileExtension, originalName, *res*);

                } else {

*res*.status(404).send("Plik nie istnieje");

                }

            } else {

*res*.status(404).send("Plik nie istnieje w bazie danych");

            }

Fragment kodu 3. System zarządzania plikami - wyświetlanie plików użytkownika

[źródło: Opracowanie własne]

Fragment kodu 4 przedstawia funkcję getFileContent, która jest odpowiedzialna   
za pobieranie zawartości pliku użytkownika z serwera na podstawie jego unikalnego identyfikatora safeId oraz nazwy pliku przekazanej jako parametr żądania. Na początku generowana jest ścieżka do katalogu użytkownika oraz docelowego pliku, po czym system sprawdza, czy plik istnieje na serwerze. Jeśli plik ma rozszerzenie .docx, jego zawartość jest odczytywana przy użyciu biblioteki mammoth, która przekształca dokument Worda na tekst   
w formacie surowym. Dla plików o innych rozszerzeniach funkcja korzysta z metody fs.readFile do odczytania zawartości pliku jako tekstu w formacie UTF-8. W przypadku powodzenia zawartość pliku zostaje przesłana do klienta w formacie JSON z odpowiednim statusem. Jeśli plik nie istnieje lub wystąpią błędy podczas odczytu, system zwraca komunikaty błędów odpowiednio z kodami 404 lub 500, informując o problemie z dostępem lub odczytem pliku.

const getFileContent = (*req*, *res*) => {

    const safeId = *req*.user.safeid\_users;

    const filename = *req*.params.filename;

    const userDir = path.join(\_\_dirname, '..', 'data', 'users', safeId);

    const filePath = path.join(userDir, filename);

    console.log(`Próbuję pobrać zawartość pliku: ${filePath}`);

    if (fs.existsSync(filePath)) {

        const fileExtension = path.extname(filename).toLowerCase();

        if (fileExtension === '.docx') {

            const mammoth = require('mammoth');

            mammoth.extractRawText({ path: filePath })

                .then(*result* => {

*res*.json({ status: 'success', content: *result*.value });

                })

                .catch(*err* => {

               console.error('Błąd podczas odczytu pliku .docx:', *err*);

*res*.status(500).json({ status: 'error', error: 'Błąd odczytu pliku .docx' });

                });

        } else {

            fs.readFile(filePath, 'utf8', (*err*, *data*) => {

                if (*err*) {

                    console.error('Błąd odczytu pliku:', *err*);

                    return *res*.status(500).json({ status: 'error', error: 'Błąd odczytu pliku' });

                }

*res*.json({ status: 'success', content: *data* });

            });

        }

} else {

        console.error('Plik nie istnieje:', filePath);

*res*.status(404).json({ status: 'error', error: 'Plik nie istnieje' });

    }

};

Fragment kodu 4. System zarządzania plikami - wyświetlanie zawartości pliku

[źródło: Opracowanie własne]

Fragment kodu 5 przedstawia funkcję renameFile, która odpowiada za zmianę nazwy pliku należącego do konkretnego użytkownika. Proces rozpoczyna się od walidacji nowej nazwy pliku, aby upewnić się, że jest ona poprawnym, niepustym ciągiem znaków. Następnie funkcja wykonuje zapytanie do bazy danych w celu sprawdzenia, czy plik o podanym identyfikatorze (cryptedname\_files) istnieje i należy do użytkownika na podstawie jego unikalnego identyfikatora safeId. Jeżeli plik zostanie odnaleziony, system aktualizuje jego oryginalną nazwę (originalname\_files) w bazie danych, pozostawiając zaszyfrowaną nazwę pliku   
bez zmian. Po pomyślnym zakończeniu operacji funkcja zwraca odpowiedź potwierdzającą zmianę nazwy pliku. W przypadku wystąpienia błędów, takich jak brak pliku, brak dostępu lub problem z serwerem, funkcja zwraca odpowiedni komunikat błędu z właściwym kodem HTTP.

const renameFile = (*req*, *res*) => {

    const safeId = *req*.user.safeid\_users;

    const { fileId } = *req*.params;

    const { newFileName } = *req*.body;

    if (!newFileName || typeof newFileName !== 'string' || !newFileName.trim()) {

        return *res*.status(400).json({ status: 'error', error: 'Nazwa pliku nie może być pusta.' });

    }

    dbFiles.query(

        'SELECT originalname\_files, cryptedname\_files FROM files WHERE cryptedname\_files = ? AND cryptedowner\_files = ?',

        [fileId, safeId],

        (*err*, *results*) => {

            if (*err*) {

                console.error('Błąd podczas wyszukiwania pliku:', *err*);

                return *res*.status(500).json({ status: 'error', error: 'Błąd serwera.' });

            }

            if (*results*.length === 0) {

                return *res*.status(404).json({ status: 'error', error: 'Plik nie istnieje lub brak dostępu.' });

            }

            const currentOriginalName = *results*[0].originalname\_files;

            const cryptedName = *results*[0].cryptedname\_files;

            const userDir = path.join(\_\_dirname, '..', 'data', 'users', safeId);

            const oldFilePath = path.join(userDir, cryptedName);

            const newFilePath = path.join(userDir, cryptedName);

            dbFiles.query(

                'UPDATE files SET originalname\_files = ? WHERE cryptedname\_files = ? AND cryptedowner\_files = ?',

                [newFileName, fileId, safeId],

                (*err*) => {

                    if (*err*) {

                        console.error('Błąd podczas aktualizacji nazwy pliku:', *err*);

                        return *res*.status(500).json({ status: 'error', error: 'Błąd serwera podczas zmiany nazwy pliku.' });

                    }

                    console.log(`Zmieniono nazwę pliku z "${currentOriginalName}" na "${newFileName}"`);

*res*.json({ status: 'success', message: 'Nazwa pliku została pomyślnie zmieniona.' });

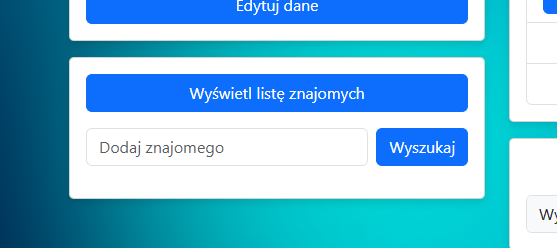
                }

Fragment kodu 5. System zarządzania plikami - zmiana nazwy pliku

[źródło: Opracowanie własne]

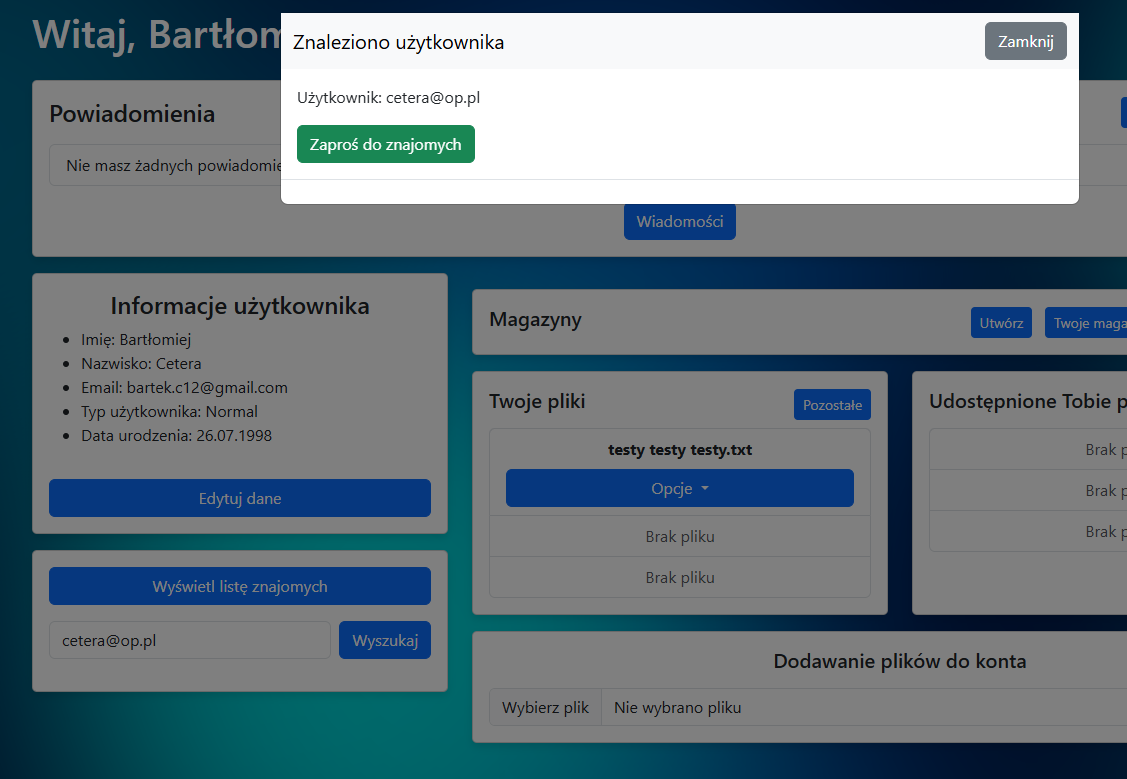
## 4.3. System znajomych

System znajomych w aplikacji sDrive został zaimplementowany w sposób umożliwiający użytkownikom nawiązywanie i zarządzanie relacjami z innymi osobami w ramach aplikacji. Funkcjonalność obejmuje możliwość wyszukiwania użytkowników na podstawie ich adresu   
e-mail oraz wysyłania zaproszeń do znajomych. Po zaakceptowaniu zaproszenia użytkownik uzyskuje dostęp do szeregu dodatkowych opcji, takich jak udostępnianie plików, wysyłanie wiadomości czy przypisywanie znajomych do wspólnych magazynów. Wdrożony system umożliwia również zarządzanie listą kontaktów, co pozwala użytkownikom na szybki dostęp do podstawowych informacji o znajomych oraz efektywne współdzielenie zasobów w obrębie aplikacji. Mechanizmy te wspierają współpracę i komunikację między użytkownikami, zapewniając jednocześnie intuicyjność obsługi i spójność danych. Rysunki 16–17 ilustrują implementację systemu znajomych, w tym interfejsy odpowiedzialne za wyszukiwanie użytkowników, wysyłanie zaproszeń oraz zarządzanie kontaktami.



Rysunek 16. System znajomych - sekcja znajomych

[źródło: Opracowanie własne]



Rysunek 17. System znajomych - wyszukiwanie nowych znajomych

[źródło: Opracowanie własne]

Fragment kodu 6 przedstawia funkcję inviteFriend, która jest odpowiedzialna   
za zapraszanie nowego znajomego w systemie oraz wysyłanie powiadomienia o zaproszeniu. Proces rozpoczyna się od pobrania identyfikatora oraz e-maila użytkownika zapraszającego,   
a także identyfikatora użytkownika, który ma zostać zaproszony. Funkcja sprawdza, czy użytkownicy nie są już znajomymi za pomocą funkcji areUsersAlreadyFriends. Jeśli relacja nie istnieje, wykonywane jest zapytanie do bazy danych w celu znalezienia unikalnego identyfikatora (safeid\_users) oraz e-maila zapraszanego użytkownika. Następnie generowany jest unikalny kod identyfikujący zaproszenie (noteId) przy użyciu funkcji bcryptjs i pętli sprawdzającej unikalność kodu. Po wygenerowaniu kodu zaproszenie zostaje zapisane w tabeli friends z odpowiednim statusem ("NotActive"), co oznacza oczekiwanie na akceptację. Równocześnie tworzona jest wiadomość zawierająca szczegóły zaproszenia, która zostaje zapisana w tabeli notifications jako powiadomienie typu friend\_request. W przypadku powodzenia funkcja zwraca komunikat o sukcesie, informując, że znajomy został zaproszony   
i powiadomienie zostało wysłane. W razie wystąpienia błędów na którymkolwiek etapie (wyszukiwania użytkownika, zapisu zaproszenia lub powiadomienia), funkcja zwraca odpowiedni komunikat błędu w formacie JSON.

const inviteFriend = async (*req*, *res*) => {

    const inviterId = *req*.user.id\_users;

    const inviterEmail = *req*.user.email\_users;

    const { friendId } = *req*.body;

    const alreadyFriends = await areUsersAlreadyFriends(inviterId, friendId);

    if (alreadyFriends) {

        return *res*.json({ status: 'error', error: 'Użytkownicy są już znajomymi.' });

    }

    dbLogins.query('SELECT safeid\_users, email\_users FROM users WHERE id\_users = ?', [friendId], async (*err*, *result*) => {

        if (*err*) {

            return *res*.json({ status: 'error', error: 'Błąd wyszukiwania użytkownika' });

        }

        if (*result*.length === 0) {

            return *res*.json({ status: 'error', error: 'Nie znaleziono zapraszanego użytkownika' });

        }

        const recipientSafeId = *result*[0].safeid\_users;

        const recipientEmail = *result*[0].email\_users;

        let noteId;

        let unique = false;

        while (!unique) {

            noteId = await bcrypt.hash(Date.now().toString(), 10);

            unique = await isNoteIdUnique(noteId);

        }

        dbLogins.query(

            'INSERT INTO friends (id\_user\_1\_friends, id\_user\_2\_friends, status\_friends, noteid\_notifications\_friends) VALUES (?, ?, "NotActive", ?)',

            [inviterId, friendId, noteId],

            (*err*, *result*) => {

                if (*err*) {

                    return *res*.json({ status: 'error', error: 'Błąd zaproszenia znajomego' });

                }

                const message = `Użytkownik ${*req*.user.firstname\_users} ${*req*.user.lastname\_users} (${inviterEmail}) wysłał ci zaproszenie do znajomych.`;

                dbLogins.query(

                    'INSERT INTO notifications (user\_notifications, head\_notifications, msg\_notifications, type\_notifications, status\_notifications, date\_notifications, dispatcher\_notifications, noteid\_notifications) VALUES (?, "Zaproszenie do znajomych", ?, "friend\_request", "unread", CURDATE(), ?, ?)',

                    [recipientSafeId, message, inviterEmail, noteId],

                    (*err*, *result*) => {

                        if (*err*) {

                            return *res*.json({ status: 'error', error: 'Błąd wysyłania powiadomienia' });

                        }

*res*.json({ status: 'success', success: 'Znajomy został zaproszony, a powiadomienie wysłane.' });

Fragment kodu 6. System znajomych - zaproszenie znajomego

[źródło: Opracowanie własne]

Fragment kodu 7 przedstawia funkcję respondToFriendRequest, która odpowiada   
za obsługę odpowiedzi na zaproszenie do znajomych, umożliwiając użytkownikowi jego zaakceptowanie lub odrzucenie. Proces rozpoczyna się od walidacji parametru action, który musi przyjmować wartość „accept” lub „deny”. Na podstawie tej akcji system ustala nowy status relacji („Active” dla akceptacji, „Denied” dla odrzucenia) i sprawdza w bazie danych, czy istnieje zaproszenie odpowiadające przesłanemu identyfikatorowi noteId. Jeśli zaproszenie zostanie znalezione, jego status jest aktualizowany w tabeli friends, a odpowiadające   
mu powiadomienie w tabeli notifications zostaje oznaczone jako „read”. W przypadku akceptacji system dodatkowo pobiera dane użytkownika zapraszającego (safeid\_users)   
i generuje powiadomienie informujące go, że jego zaproszenie zostało zaakceptowane. Powiadomienie to zawiera szczegóły akcji i jest zapisywane w tabeli notifications   
z odpowiednim statusem („unread”). Ostatecznie funkcja zwraca komunikat o sukcesie, informując, czy zaproszenie zostało zaakceptowane, czy odrzucone. W razie wystąpienia błędów na dowolnym etapie, takich jak problem z bazą danych lub brak zaproszenia, funkcja zwraca odpowiedni komunikat błędu.

const respondToFriendRequest = (*req*, *res*) => {

    const { noteId, action } = *req*.body;

    let newStatus = action === 'accept' ? 'Active' : 'Denied';

    if (!['accept', 'deny'].includes(action)) {

        return *res*.json({ status: 'error', error: 'Nieprawidłowa akcja' });

    }

    dbLogins.query(

        'SELECT \* FROM friends WHERE noteid\_notifications\_friends = ?',

        [noteId],

        (*err*, *result*) => {

            if (*err*) {

                return *res*.json({ status: 'error', error: 'Błąd podczas szukania zaproszenia' });

            }

            if (*result*.length === 0) {

                return *res*.json({ status: 'error', error: 'Nie znaleziono zaproszenia' });

            }

            const friendEntryId = *result*[0].id\_friends;

            const inviterId = *result*[0].id\_user\_1\_friends;

            const accepterId = *result*[0].id\_user\_2\_friends;

            dbLogins.query(

                'UPDATE friends SET status\_friends = ? WHERE id\_friends = ?',

                [newStatus, friendEntryId],

                (*err*) => {

                    if (*err*) {

                        return *res*.json({ status: 'error', error: 'Błąd aktualizacji statusu' });

                    }

                    dbLogins.query(

                        'UPDATE notifications SET status\_notifications = "read" WHERE noteid\_notifications = ?',

                        [noteId],

                        (*err*) => {

                            if (*err*) {

                                return *res*.json({ status: 'error', error: 'Błąd aktualizacji statusu powiadomienia' });

                            }

                            if (action === 'accept') {

                                dbLogins.query(

                                    'SELECT safeid\_users, email\_users FROM users WHERE id\_users = ?',

                                    [inviterId],

                                    (*err*, *inviterResult*) => {

                                        if (*err* || *inviterResult*.length === 0) {

                                            return *res*.json({ status: 'error', error: 'Błąd pobierania danych zapraszającego' });

                                        }

                                        const inviterSafeId = *inviterResult*[0].safeid\_users;

                                        const accepterEmail = *req*.user.email\_users;

                                        const message = `Użytkownik o mailu ${accepterEmail} zaakceptował twoje zaproszenie do znajomych.`;

                                        dbLogins.query(

                                            'INSERT INTO notifications (user\_notifications, head\_notifications, msg\_notifications,

type\_notifications, status\_notifications, date\_notifications, dispatcher\_notifications) VALUES (?, "Nowy znajomy", ?, "friend\_response", "unread", CURDATE(), ?)',

                                            [inviterSafeId, message, accepterEmail],

                                            (*err*) => {

                                                if (*err*) {

                                                    return *res*.json({ status: 'error', error: 'Błąd podczas dodawania powiadomienia' });

                                                }

                                                return *res*.json({ status: 'success', success: 'Zaproszenie zaakceptowane i powiadomienie wysłane.' });

                                            }

                                        );

                                    }

                                );

                            } else {

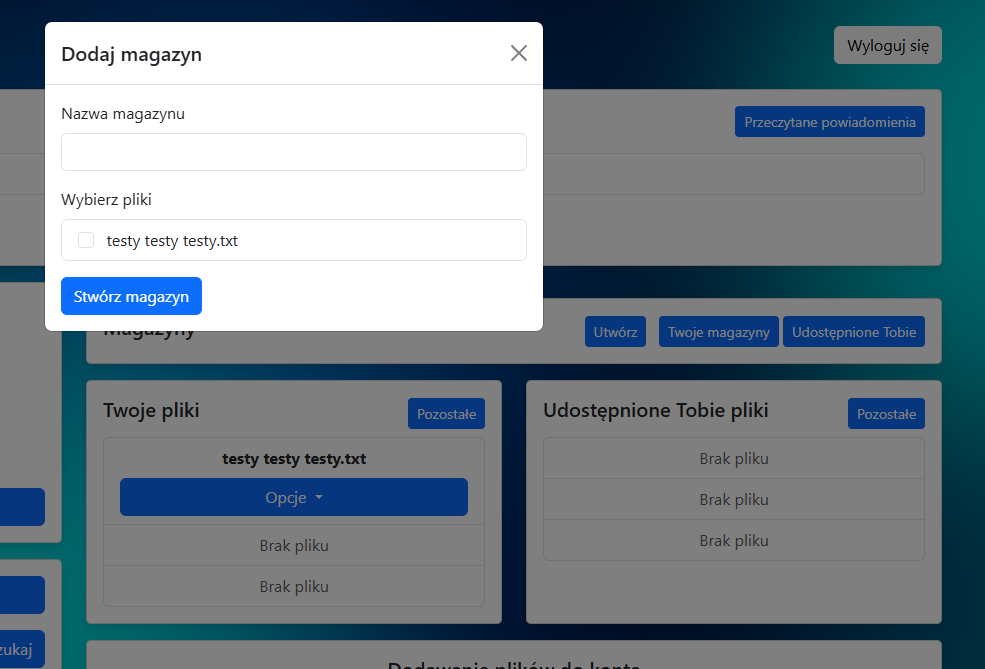
                                return *res*.json({ status: 'success', success: 'Zaproszenie zostało odrzucone.' });

Fragment kodu 7. System znajomych – odpowiedź na zaproszenie

[źródło: Opracowanie własne]

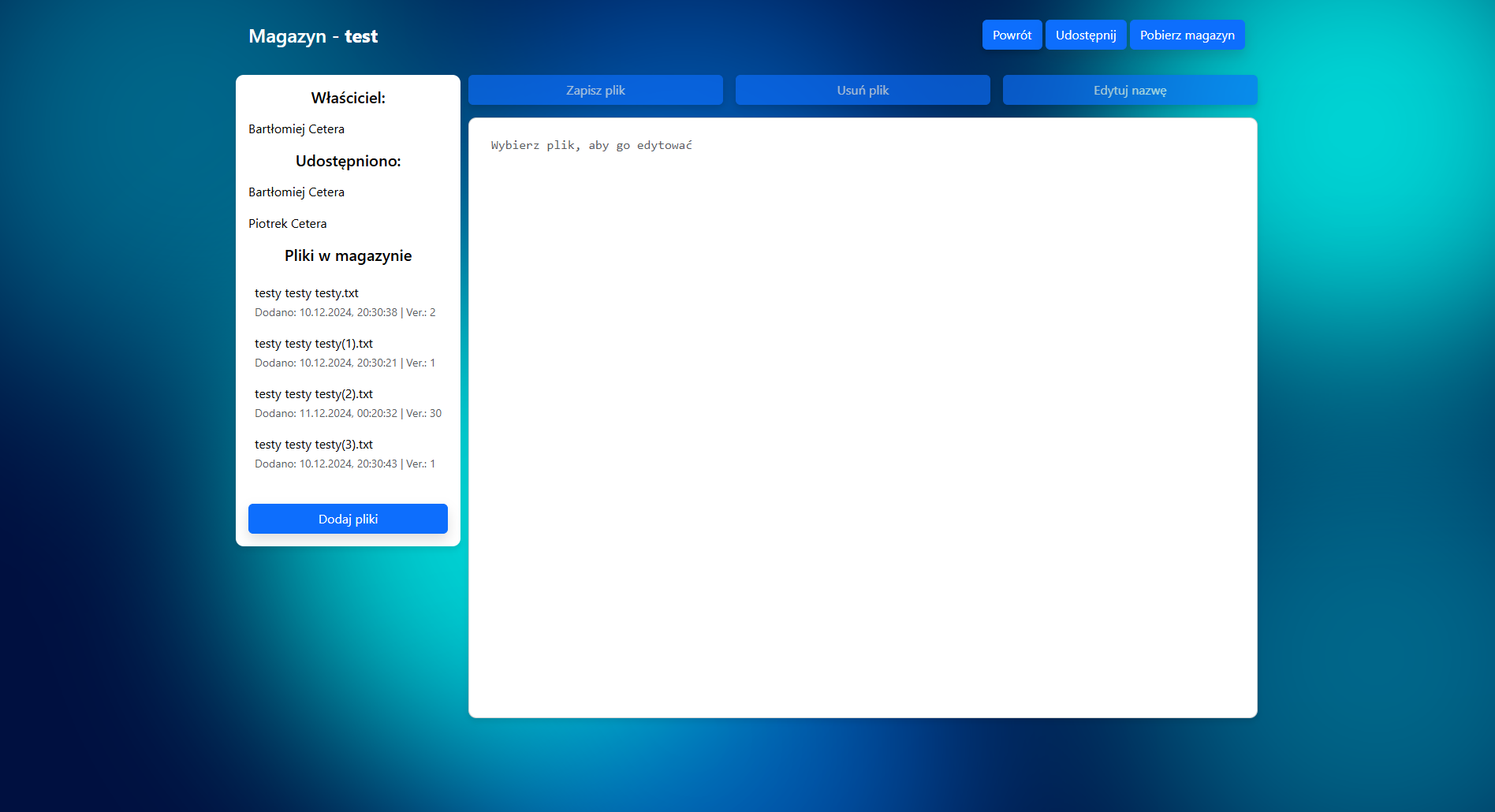
## 4.4. System magazynów

System magazynów w aplikacji sDrive został zaimplementowany jako przestrzeń dedykowana do przechowywania plików oraz wspólnej pracy użytkowników. Umożliwia tworzenie nowych magazynów oraz zarządzanie nimi, w tym usuwanie, zmianę nazwy czy przypisywanie innych użytkowników do współpracy. W ramach magazynu można dodawać pliki, edytować je oraz korzystać z mechanizmu kontroli wersji, który zapewnia spójność danych. W przypadku wystąpienia konfliktu wersji użytkownik ma możliwość jego rozwiązania poprzez dedykowaną stronę, gdzie wyświetlane są szczegóły problemu. Dodatkowo magazyny mogą być pobierane w całości w formie archiwum ZIP. Rysunki 18–20 wizualizują implementację kluczowych funkcjonalności systemu magazynów, takich jak tworzenie przestrzeni przechowywania, zarządzanie zawartością, rozwiązywanie konfliktów wersji oraz wspólna praca użytkowników.



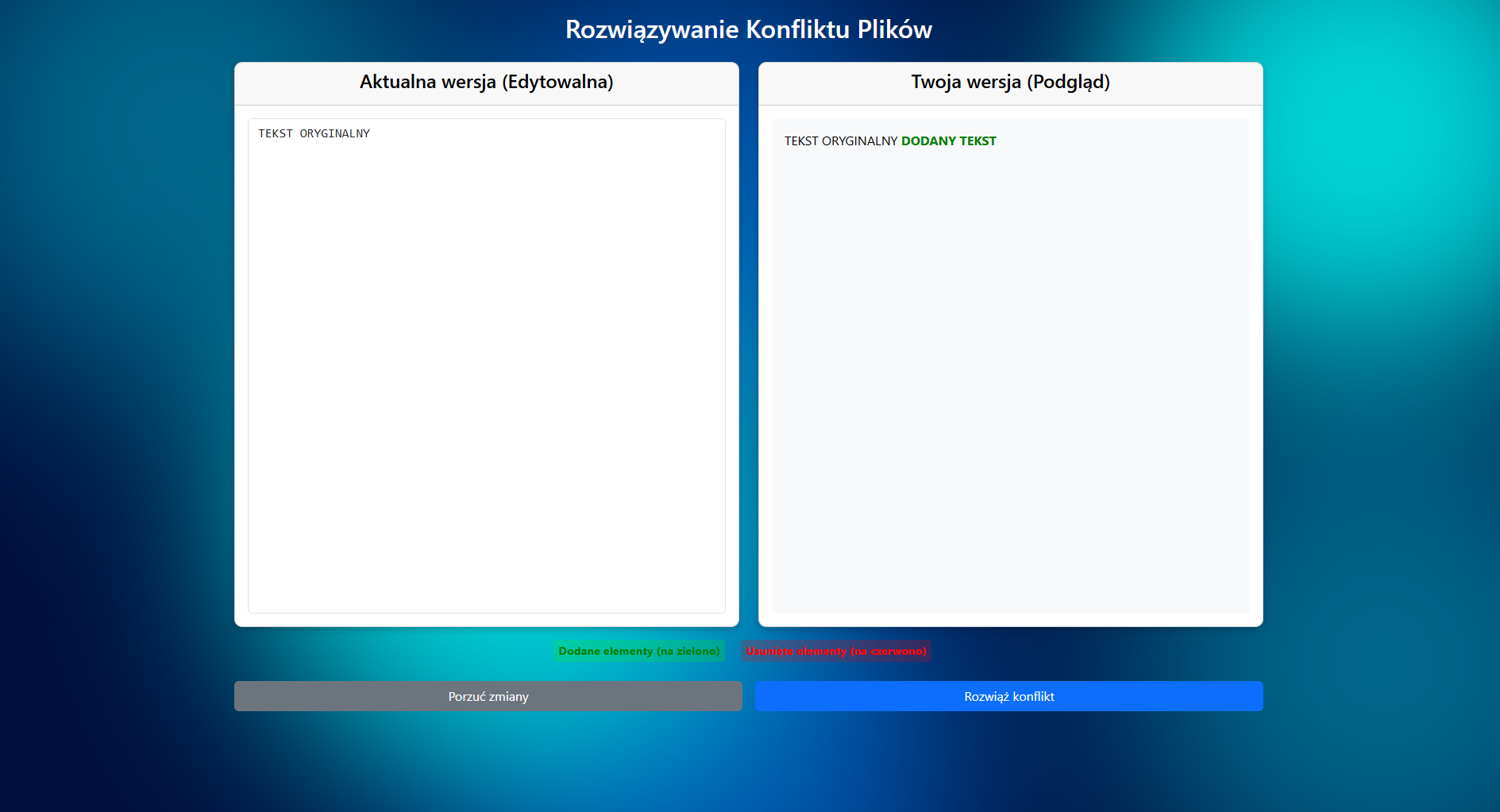
Rysunek 18. System magazynów - tworzenie magazynu

[źródło: Opracowanie własne]



Rysunek 19. System magazynów – magazyn

[źródło: Opracowanie własne]



Rysunek 20. System magazynów - rozwiązywanie konfliktu w magazynie

[źródło: Opracowanie własne]

Fragment kodu 08 przedstawia funkcję saveResolvedFileContent, która zapisuje rozwiązaną zawartość pliku i aktualizuje jego wersję w systemie magazynów. Funkcja pobiera dane, takie jak zaszyfrowana nazwa pliku, identyfikator magazynu, identyfikator użytkownika i nową zawartość pliku, po czym weryfikuje ich obecność oraz istnienie ścieżki zapisu   
na serwerze. Następnie sprawdza w bazie danych, czy plik istnieje, i pobiera jego bieżącą wersję. Jeśli plik istnieje, jego zawartość jest nadpisywana, a wersja zwiększana o jeden. Aktualizacja wersji i daty modyfikacji zapisywana jest w bazie danych. W przypadku sukcesu funkcja zwraca komunikat z potwierdzeniem i nową wersją pliku. W razie błędów, takich jak brak danych, problem z bazą lub zapisem pliku, funkcja zwraca odpowiednie komunikaty błędów, a logowanie szczegółów wspiera debugowanie procesu.

const saveResolvedFileContent = (*req*, *res*) => {

    const cryptedName = *req*.params.fileId;

    const storageId = *req*.query.storageId;

    const safeId = *req*.user.safeid\_users;

    const { content } = *req*.body;

    console.log('DEBUG: Rozpoczęcie zapisu rozwiązanej zawartości pliku.');

    console.log(`DEBUG: fileId: ${cryptedName}`);

    console.log(`DEBUG: storageId: ${storageId}`);

    console.log(`DEBUG: safeId: ${safeId}`);

    console.log(`DEBUG: Długość zawartości: ${content ? content.length : 'brak zawartości'}`);

    if (!cryptedName || !storageId || !content) {

        console.error('DEBUG: Brak wymaganych danych');

        return *res*.status(400).json({ status: 'error', error: 'Brak wymaganych danych' });

    }

    const filePath = path.join(\_\_dirname, '..', 'data', 'storages', safeId, storageId, cryptedName);

    console.log(`DEBUG: Ścieżka do pliku: ${filePath}`);

    const dirPath = path.dirname(filePath);

    if (!fs.existsSync(dirPath)) {

        console.error(`DEBUG: Katalog nie istnieje: ${dirPath}`);

        return *res*.status(500).json({ status: 'error', error: 'Ścieżka do zapisu pliku nie istnieje.' });

    }

    dbFiles.query(

        `SELECT f.id\_files, sf.file\_version\_storages\_files

         FROM storages\_files sf

         JOIN files f ON sf.id\_files = f.id\_files

         WHERE f.cryptedname\_files = ? AND sf.id\_storages = ?`,

        [cryptedName, storageId],

        (*err*, *results*) => {

            if (*err*) {

                console.error(`DEBUG: Błąd podczas pobierania pliku: ${*err*.message}`);

                return *res*.status(500).json({ status: 'error', error: 'Błąd podczas wyszukiwania pliku.' });

            }

            if (*results*.length === 0) {

                console.error('DEBUG: Plik nie został znaleziony.');

                return *res*.status(404).json({ status: 'error', error: 'Plik nie został znaleziony.' });

            }

            const fileId = *results*[0].id\_files;

            const currentVersion = *results*[0].file\_version\_storages\_files || 1;

            const newVersion = currentVersion + 1;

*// Zapisanie pliku na dysku*

            fs.writeFile(filePath, content, 'utf8', (*err*) => {

                if (*err*) {

                    console.error(`DEBUG: Błąd zapisu pliku: ${*err*.message}`);

                    return *res*.status(500).json({ status: 'error', error: 'Błąd podczas zapisywania pliku.' });

                }

                console.log(`DEBUG: Plik zapisany pomyślnie: ${filePath}`);

                dbFiles.query(

                    `UPDATE storages\_files

                     SET file\_version\_storages\_files = ?, date\_storages\_file = NOW()

                     WHERE id\_storages = ? AND id\_files = ?`,

                    [newVersion, storageId, fileId],

                    (*updateErr*) => {

                        if (*updateErr*) {

                            console.error(`DEBUG: Błąd podczas aktualizacji wersji pliku: ${*updateErr*.message}`);

                            return *res*.status(500).json({ status: 'error', error: 'Błąd podczas aktualizacji wersji pliku.' });

                        }

                        console.log(`DEBUG: Wersja pliku zaktualizowana do Ver. ${newVersion}`);

*res*.json({ status: 'success', message: `Plik zapisany pomyślnie. Aktualna wersja: Ver. ${newVersion}` });

Fragment kodu 8. System magazynów - rozwiązywanie konfliktu plików

[źródło: Opracowanie własne]

Fragment kodu 9 przedstawia funkcję saveConflictedFile, która obsługuje sytuacje konfliktu plików w systemie, tworząc oddzielne kopie aktualnej wersji pliku oraz zmian wprowadzonych przez użytkownika. Na początku pobierane są wymagane dane, takie jak zaszyfrowana nazwa pliku (cryptedName), identyfikator magazynu (storageId), identyfikator użytkownika (safeId) oraz treść zmian. Funkcja weryfikuje, czy wszystkie niezbędne dane zostały przesłane, a następnie określa ścieżkę do oryginalnego pliku na serwerze oraz   
do katalogu tymczasowego, w którym będą przechowywane pliki konfliktowe. Jeśli katalog tymczasowy nie istnieje, zostaje on utworzony. Następnie funkcja sprawdza, czy oryginalny plik znajduje się na serwerze. Jeśli plik istnieje, jego kopia zostaje zapisana w katalogu tymczasowym. Równocześnie zmiany wprowadzone przez użytkownika są zapisywane jako osobny plik w tym samym katalogu, z dopiskiem \_tmp w nazwie. Na koniec funkcja zwraca komunikat o powodzeniu wraz ze ścieżkami do obu plików konfliktowych. W przypadku błędów, takich jak problem z dostępem do pliku lub operacjami na systemie plików, funkcja zwraca odpowiedni komunikat o błędzie i loguje szczegóły do debugowania.

const saveConflictedFile = (*req*, *res*) => {

    const cryptedName = *req*.query.fileId;

    const storageId = *req*.query.storageId;

    const safeId = *req*.user.safeid\_users;

    const { content } = *req*.body;

    console.log('DEBUG: Rozpoczęcie obsługi konfliktu pliku.');

    console.log(`DEBUG: fileId: ${cryptedName}`);

    console.log(`DEBUG: storageId: ${storageId}`);

    console.log(`DEBUG: safeId: ${safeId}`);

    console.log(`DEBUG: Długość zawartości: ${content ? content.length : 'brak zawartości'}`);

    if (!cryptedName || !storageId || !content) {

        console.error('DEBUG: Brak wymaganych danych');

        return *res*.status(400).json({ status: 'error', error: 'Brak wymaganych danych' });

    }

    const filePath = path.join(\_\_dirname, '..', 'data', 'storages', safeId, storageId, cryptedName);

    const tempDir = path.join(\_\_dirname, '..', 'data', 'temp');

    if (!fs.existsSync(tempDir)) {

        fs.mkdirSync(tempDir);

    }

    const tempFilePath = path.join(tempDir, cryptedName);

    const tempFilePathTmp = path.join(

        tempDir,

        `${path.basename(cryptedName, path.extname(cryptedName))}\_tmp${path.extname(cryptedName)}`

    );

    try {

        if (fs.existsSync(filePath)) {

            fs.copyFileSync(filePath, tempFilePath);

            console.log(`DEBUG: Utworzono kopię pliku z serwera: ${tempFilePath}`);

        } else {

            console.warn(`DEBUG: Plik z serwera nie istnieje: ${filePath}`);

        }

        fs.writeFileSync(tempFilePathTmp, content, 'utf8');

        console.log(`DEBUG: Utworzono kopię zmian użytkownika: ${tempFilePathTmp}`);

*res*.json({

            status: 'success',

            message: 'Pliki konfliktowe zostały utworzone.',

            tempFilePath,

            tempFilePathTmp,

        });

    } catch (err) {

        console.error('DEBUG: Błąd podczas obsługi konfliktu pliku:', err);

*res*.status(500).json({ status: 'error', error: 'Wystąpił błąd podczas obsługi konfliktu pliku.' });

    }

};

Fragment kodu 9. System magazynów - obsługa konfliktu w magazynie

[źródło: Opracowanie własne]

Fragment kodu 10 przedstawia funkcję addFilesToStorage, która odpowiada za dodawanie wybranych plików użytkownika do określonego magazynu danych. Na początku pobierane   
są informacje o identyfikatorze magazynu (storageId), identyfikatorze użytkownika (safeId) oraz liście wybranych plików (selectedFiles). Funkcja sprawdza, czy wszystkie wymagane dane zostały przekazane i czy magazyn istnieje na serwerze. Następnie wykonuje zapytanie   
do bazy danych, aby zweryfikować, czy pliki należą do użytkownika i nie zostały usunięte.   
Dla każdego pliku funkcja sprawdza unikalność nazwy w obrębie magazynu, a w przypadku konfliktu modyfikuje nazwę pliku, dodając odpowiedni sufiks. Pliki są następnie kopiowane do katalogu magazynu z nadaniem nowej zaszyfrowanej nazwy, a informacje o plikach   
są zapisywane w bazie danych w tabelach files i storages\_files. Funkcja zapewnia obsługę błędów na każdym etapie procesu, takich jak problemy z dostępem do plików czy konflikty nazw, a w przypadku powodzenia zwraca komunikat potwierdzający dodanie plików   
do magazynu. Dzięki szczegółowemu logowaniu debugowania administratorzy mogą łatwo monitorować i rozwiązywać ewentualne problemy.

const addFilesToStorage = async (*req*, *res*) => {

    const { storageId } = *req*.params;

    const safeId = *req*.user.safeid\_users;

    const { selectedFiles } = *req*.body;

    console.log(`DEBUG: Rozpoczęcie dodawania plików do magazynu. storageId: ${storageId}, safeId: ${safeId}`);

    console.log(`DEBUG: Wybrane pliki: ${selectedFiles.join(', ')}`);

    if (!selectedFiles || !Array.isArray(selectedFiles) || selectedFiles.length === 0) {

        return *res*.status(400).json({ status: 'error', error: 'Brak wybranych plików do dodania.' });

    }

    const storagePath = path.join(\_\_dirname, '..', 'data', 'storages', safeId, storageId);

    if (!fs.existsSync(storagePath)) {

        return *res*.status(404).json({ status: 'error', error: 'Magazyn nie istnieje.' });

    }

    try {

        const files = await new Promise((*resolve*, *reject*) => {

            dbFiles.query(

                'SELECT \* FROM files WHERE originalname\_files IN (?) AND cryptedowner\_files = ? AND delete\_files = 0',

                [selectedFiles, safeId],

                (*err*, *results*) => {

                    if (*err*) {

                        return reject(*err*);

                    }

                    resolve(*results*);

                }

            );

        });

        const currentDate = new Date();

        const storageFilesValues = [];

        for (const file of files) {

            const sourcePath = path.join(\_\_dirname, '..', 'data', 'users', safeId, file.cryptedname\_files);

            let newOriginalName = file.originalname\_files;

            const fileExtension = path.extname(file.originalname\_files);

            const fileBaseName = path.basename(file.originalname\_files, fileExtension);

            let suffix = 0;

            let isOriginalNameUnique = false;

            while (!isOriginalNameUnique) {

                const conflictingFiles = await new Promise((*resolve*, *reject*) => {

                    dbFiles.query(

                        `SELECT f.originalname\_files

                         FROM storages\_files sf

                         JOIN files f ON sf.id\_files = f.id\_files

                         WHERE sf.id\_storages = ? AND f.originalname\_files = ? AND sf.active\_storages\_files = 1`,

                        [storageId, newOriginalName],

                        (*err*, *results*) => {

                            if (*err*) {

                                return reject(*err*);

                            }

                            resolve(*results*);

                        }

                    );

                });

                if (conflictingFiles.length > 0) {

                    suffix++;

                    newOriginalName = `${fileBaseName}(${suffix})${fileExtension}`;

                } else {

                    isOriginalNameUnique = true;

                }

            }

            if (fs.existsSync(sourcePath)) {

                const newCryptedName = `${crypto.randomBytes(16).toString('hex')}${fileExtension}`;

                const destPath = path.join(storagePath, newCryptedName);

                fs.copyFileSync(sourcePath, destPath);

                console.log(`DEBUG: Skopiowano plik: ${file.cryptedname\_files} -> ${destPath}`);

                const newFileId = await new Promise((*resolve*, *reject*) => {

                    dbFiles.query(

                        `INSERT INTO files (originalname\_files, cryptedname\_files, cryptedowner\_files, filetype\_files, delete\_files, dateofupload\_files, origin\_file, oldcryptedname\_file)

                         VALUES (?, ?, ?, ?, 0, ?, ?, ?)`,

                        [

                            newOriginalName,

                            newCryptedName,

                            safeId,

                            file.filetype\_files,

                            currentDate,

                            'storage',

                            file.cryptedname\_files,

                        ],

                        (*err*, *result*) => {

                            if (*err*) {

                                return reject(*err*);

                            }

                            resolve(*result*.insertId);

                        }

                    );

                });

                storageFilesValues.push([

                    storageId,

                    newFileId,

                    currentDate,

                    1,

                    safeId,

                    1,

                ]);

            } else {

                console.warn(`DEBUG: Plik nie istnieje: ${sourcePath}`);

            }

        }

        if (storageFilesValues.length > 0) {

            await new Promise((*resolve*, *reject*) => {

                dbFiles.query(

                    'INSERT INTO storages\_files (id\_storages, id\_files, date\_storages\_file, active\_storages\_files, modificator\_storages\_files, file\_version\_storages\_files) VALUES ?',

                    [storageFilesValues],

                    (*err*) => {

                        if (*err*) {

                            return reject(*err*);

                        }

                        resolve();

                    }

                );

            });

        }

        console.log('DEBUG: Pliki zostały pomyślnie dodane do magazynu.');

*res*.json({ status: 'success', message: 'Pliki zostały dodane do magazynu.' });

    } catch (err) {

        console.error(`DEBUG: Błąd podczas dodawania plików do magazynu: ${err.message}`);

*res*.status(500).json({ status: 'error', error: 'Błąd podczas dodawania plików do magazynu.' });

    }

};

Fragment kodu 10. System magazynów - dodawanie pliku do magazynu

[źródło: Opracowanie własne]

Fragment kodu 11 przedstawia funkcję saveFileStorageContent, która zajmuje   
się zapisywaniem zawartości pliku w magazynie danych i aktualizacją jego wersji. Na początku pobierane są kluczowe dane, takie jak zaszyfrowana nazwa pliku (cryptedName), identyfikator magazynu (storageId), identyfikator użytkownika (safeId) oraz nowa zawartość pliku i jego aktualna wersja. Funkcja sprawdza, czy wszystkie wymagane dane zostały przekazane, a także weryfikuje, czy ścieżka do magazynu istnieje na serwerze. Następnie system wykonuje zapytania do bazy danych, aby zweryfikować, czy plik istnieje w tabelach files i storages\_files. Po uzyskaniu informacji o wersji pliku zapisanej w bazie danych (dbVersion), następuje porównanie z wersją przesłaną w żądaniu (currentVersion). Jeśli wersje się różnią, funkcja zgłasza konflikt wersji i odrzuca żądanie z odpowiednim komunikatem błędu. W przypadku zgodności wersji zawartość pliku zostaje zapisana na serwerze, a wersja pliku w tabeli storages\_files jest zwiększana o jeden. Po pomyślnym zakończeniu operacji funkcja zwraca odpowiedź potwierdzającą zapis pliku oraz aktualną wersję. Jeśli w dowolnym etapie wystąpią błędy, takie jak problem z dostępem do pliku, zapisaniem zawartości czy aktualizacją wersji, funkcja zwraca odpowiedni komunikat błędu z kodem statusu. Debugowanie funkcji jest wspierane przez szczegółowe logowanie, co ułatwia rozwiązywanie potencjalnych problemów.

const saveFileStorageContent = (*req*, *res*) => {

    const cryptedName = *req*.params.fileId;

    const storageId = *req*.query.storageId;

    const safeId = *req*.user.safeid\_users;

    const { content, currentVersion } = *req*.body;

    console.log('DEBUG: Rozpoczęcie zapisu pliku do magazynu.');

    console.log(`DEBUG: fileId: ${cryptedName}`);

    console.log(`DEBUG: storageId: ${storageId}`);

    console.log(`DEBUG: safeId: ${safeId}`);

    console.log(`DEBUG: Aktualna wersja: ${currentVersion}`);

    console.log(`DEBUG: Długość zawartości: ${content ? content.length : 'brak zawartości'}`);

    if (!cryptedName || !storageId || !content) {

        console.error('DEBUG: Błąd - Brak wymaganych danych');

        return *res*.status(400).json({ status: 'error', error: 'Brak wymaganych danych' });

    }

    const filePath = path.join(\_\_dirname, '..', 'data', 'storages', safeId, storageId, cryptedName);

    console.log(`DEBUG: Ścieżka do pliku: ${filePath}`);

    const dirPath = path.dirname(filePath);

    if (!fs.existsSync(dirPath)) {

        console.error(`DEBUG: Katalog nie istnieje: ${dirPath}`);

        return *res*.status(500).json({ status: 'error', error: 'Ścieżka do zapisu pliku nie istnieje.' });

    }

    dbFiles.query(

        `SELECT id\_files FROM files WHERE cryptedname\_files = ?`,

        [cryptedName],

        (*err*, *fileResults*) => {

            if (*err*) {

                console.error(`DEBUG: Błąd podczas pobierania id\_files: ${*err*.message}`);

                return *res*.status(500).json({ status: 'error', error: 'Błąd podczas wyszukiwania pliku.' });

            }

            if (*fileResults*.length === 0) {

                console.error('DEBUG: Plik nie został znaleziony w tabeli `files`.');

                return *res*.status(404).json({ status: 'error', error: 'Plik nie został znaleziony.' });

            }

            const fileId = *fileResults*[0].id\_files;

            console.log(`DEBUG: Znaleziono id\_files: ${fileId}`);

            dbFiles.query(

                `SELECT file\_version\_storages\_files

                 FROM storages\_files

                 WHERE id\_storages = ? AND id\_files = ? AND active\_storages\_files = 1`,

                [storageId, fileId],

                (*err*, *results*) => {

                    if (*err*) {

                        console.error(`DEBUG: Błąd podczas pobierania wersji pliku: ${*err*.message}`);

                        return *res*.status(500).json({ status: 'error', error: 'Błąd podczas pobierania wersji pliku.' });

                    }

                    if (*results*.length === 0) {

                        console.error('DEBUG: Plik nie został znaleziony w tabeli `storages\_files`.');

                        return *res*.status(404).json({ status: 'error', error: 'Plik nie został znaleziony w magazynie.' });

                    }

                    const dbVersion = *results*[0].file\_version\_storages\_files || 1;

                    console.log(`DEBUG: Wersja pliku w bazie danych: Ver. ${dbVersion}`);

                    if (dbVersion !== currentVersion) {

                        console.error(

                            `DEBUG: Konflikt wersji pliku. Wersja w bazie: ${dbVersion}, przesłana wersja: ${currentVersion}`

                        );

                        return *res*.status(409).json({

                            status: 'error',

                            error: `Konflikt wersji pliku. Oczekiwana wersja: Ver. ${dbVersion}`,

                        });

                    }

                    fs.writeFile(filePath, content, 'utf8', (*err*) => {

                        if (*err*) {

                            console.error(`DEBUG: Błąd zapisu pliku: ${*err*.message}`);

                            return *res*.status(500).json({ status: 'error', error: 'Błąd podczas zapisywania pliku.' });

                        }

                        console.log(`DEBUG: Plik zapisany pomyślnie: ${filePath}`);

                        const newVersion = dbVersion + 1;

                        dbFiles.query(

                            `UPDATE storages\_files

                             SET file\_version\_storages\_files = ?, date\_storages\_file = NOW()

                             WHERE id\_storages = ? AND id\_files = ?`,

                            [newVersion, storageId, fileId],

                            (*updateErr*) => {

                                if (*updateErr*) {

                                    console.error(`DEBUG: Błąd podczas aktualizacji wersji pliku: ${*updateErr*.message}`);

                                    return *res*.status(500).json({ status: 'error', error: 'Błąd podczas aktualizacji wersji pliku.' });

                                }

                                console.log(`DEBUG: Wersja pliku zaktualizowana do Ver. ${newVersion}`);

*res*.json({ status: 'success', message: `Plik został zapisany pomyślnie. Aktualna wersja: Ver. ${newVersion}` });

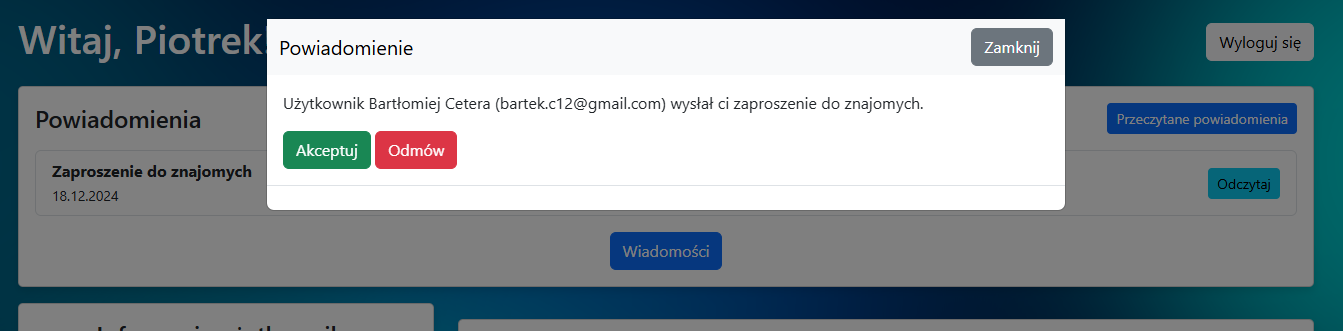
                            }

Fragment kodu 11. System magazynów - zapis zawartości pliku w magazynie

[źródło: Opracowanie własne]

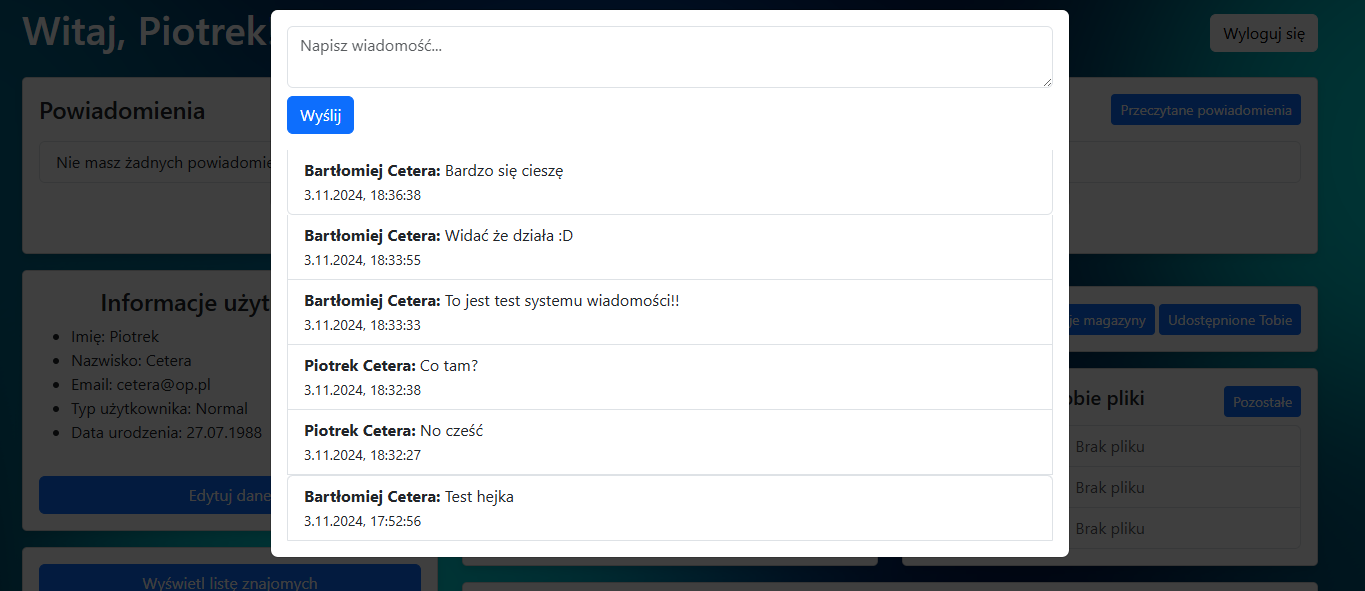
## 4.5. System powiadomień i wiadomości

System powiadomień i wiadomości w aplikacji sDrive został zaimplementowany jako element wspierający interakcje między użytkownikami oraz informowanie o istotnych zdarzeniach w systemie. W sekcji powiadomień użytkownik otrzymuje automatyczne komunikaty o zdarzeniach, takich jak nowe wiadomości, akceptacje zaproszeń od znajomych, udostępnienia plików czy aktualizacje w magazynach. Powiadomienia te są prezentowane   
w uporządkowanej formie, a użytkownik ma możliwość przeglądania zarówno nowych,   
jak i wcześniej odczytanych komunikatów. Dodatkowo w ramach tej sekcji dostępny jest moduł wiadomości, który umożliwia użytkownikom przesyłanie i odbieranie wiadomości. Każda konwersacja zawiera pełną historii. Rysunki 21 i 22 ilustrują implementację systemu powiadomień i wiadomości, przedstawiając kluczowe funkcjonalności, takie jak zarządzanie powiadomieniami oraz przesyłanie wiadomości między użytkownikami.



Rysunek 21. System powiadomień i wiadomości - podgląd powiadomienia

[źródło: Opracowanie własne]



Rysunek 22. System powiadomień i wiadomości – konwersacja

[źródło: Opracowanie własne]

Fragment kodu 12 przedstawia fragment kodu, który zawiera trzy funkcje obsługujące powiadomienia w systemie. Pierwsza z nich, getNotification, umożliwia pobranie szczegółowych informacji o konkretnym powiadomieniu na podstawie jego identyfikatora (notificationId). Wykonywane jest zapytanie do bazy danych, aby uzyskać dane takie jak tytuł, treść, nadawca, identyfikator notyfikacji oraz typ powiadomienia. Jeśli powiadomienie istnieje, zwracane są jego szczegóły w odpowiedzi JSON. W przeciwnym wypadku funkcja zwraca komunikat o błędzie. Druga funkcja, markNotificationAsRead, służy do oznaczania konkretnego powiadomienia jako przeczytane. Na podstawie identyfikatora powiadomienia (notificationId) wykonywane jest zapytanie aktualizujące w bazie danych pole status\_notifications, zmieniając jego wartość na „read”. W przypadku powodzenia funkcja zwraca komunikat o sukcesie, a w razie błędów zwraca odpowiedni komunikat o błędzie. Trzecia funkcja, getReadNotifications, pozwala na pobranie listy wszystkich przeczytanych powiadomień dla danego użytkownika. Na podstawie identyfikatora użytkownika (safeId) system wyszukuje w bazie danych powiadomienia oznaczone jako „read” i zwraca ich listę   
w odpowiedzi JSON, zawierając szczegóły takie jak tytuł, treść i data. W razie błędów funkcja zwraca komunikat o błędzie z kodem statusu 500. Kod ten umożliwia zarządzanie powiadomieniami w sposób przejrzysty i efektywny, a dzięki zastosowaniu odpowiednich zapytań SQL zapewnia spójność danych w systemie.

const getNotification = (*req*, *res*) => {

    const notificationId = *req*.params.id;

    dbLogins.query(

        'SELECT head\_notifications, msg\_notifications, dispatcher\_notifications, noteid\_notifications, type\_notifications FROM notifications WHERE id\_notifications = ?',

        [notificationId],

        (*err*, *result*) => {

            if (*err*) {

                return *res*.json({ status: 'error', error: 'Błąd pobierania powiadomienia' });

            }

            if (*result*.length === 0) {

                return *res*.json({ status: 'error', error: 'Powiadomienie nie istnieje' });

            }

*res*.json({ status: 'success', notification: *result*[0] });

        }    );};

const markNotificationAsRead = (*req*, *res*) => {

    const notificationId = *req*.params.id;

    dbLogins.query(

        'UPDATE notifications SET status\_notifications = "read" WHERE id\_notifications = ?',

        [notificationId],

        (*err*, *result*) => {

            if (*err*) {

                return *res*.json({ status: 'error', error: 'Błąd podczas oznaczania powiadomienia jako przeczytane' });

            }

            return *res*.json({ status: 'success', success: 'Powiadomienie zostało oznaczone jako przeczytane' });

        }    );};

const getReadNotifications = (*req*, *res*) => {

    const safeId = *req*.user.safeid\_users;

    dbLogins.query(

        'SELECT id\_notifications, head\_notifications, msg\_notifications, date\_notifications FROM notifications WHERE user\_notifications = ? AND status\_notifications = "read"',

        [safeId],

        (*err*, *results*) => {

            if (*err*) {

                console.error("Błąd podczas pobierania przeczytanych powiadomień:", *err*);

                return *res*.status(500).json({ status: "error", error: "Błąd podczas pobierania przeczytanych powiadomień" });

            }

*res*.json({ status: "success", notifications: *results* });

        }

Fragment kodu 12. System powiadomień i wiadomości – powiadomienia

[źródło: Opracowanie własne]

## 4.6. System enkrypcji i degrupcji danych

System enkrypcji i dekrypcji danych w aplikacji sDrive został zaimplementowany   
w oparciu o algorytm AES, który odgrywa kluczową rolę w zapewnieniu poufności   
i bezpieczeństwa przechowywanych informacji. AES działa jako szyfr blokowy, dzieląc dane na bloki o stałym rozmiarze, które są następnie szyfrowane w ramach serii matematycznych operacji. Proces ten obejmuje wielokrotne przekształcenia danych, takich jak substytucja bajtów, przesunięcie wierszy w macierzy, mieszanie kolumn oraz łączenie danych z kluczem szyfrowania. W aplikacji sDrive algorytm AES znajduje zastosowanie zarówno podczas logowania, jak i wylogowywania użytkownika. W trakcie logowania dane są odszyfrowywane, aby umożliwić ich przetwarzanie w systemie. Z kolei przy wylogowywaniu wszystkie dane   
są ponownie szyfrowane z użyciem nowo wygenerowanego klucza AES. Dynamiczna wymiana klucza za każdym razem, gdy użytkownik opuszcza system, znacząco zwiększa bezpieczeństwo, minimalizując ryzyko wykorzystania potencjalnie przechwyconych informacji. Integralnym elementem systemu bezpieczeństwa jest także wykorzystanie algorytmu bcryptjs, który odpowiada za ochronę danych uwierzytelniających użytkownika. Dzięki wielokrotnemu haszowaniu haseł i zastosowaniu unikalnej soli, bcryptjs skutecznie utrudnia złamanie zabezpieczeń nawet w przypadku kompromitacji bazy danych.

Fragment kodu 13 ilustruje proces enkrypcji danych w systemie sDrive, wykorzystując algorytm AES-256 w trybie CTR. W pierwszej kolejności generowany jest klucz szyfrowania za pomocą funkcji generateKey, która wykorzystuje metodę crypto.randomBytes do stworzenia losowego ciągu bajtów o długości 32, zgodnie z wymaganiami dla klucza AES-256. Funkcja encryptDirectory odpowiada za rekurencyjne szyfrowanie wszystkich plików w podanym katalogu. Na początku sprawdzane jest, czy wskazany katalog istnieje. Następnie lista plików w katalogu jest odczytywana za pomocą fs.readdirSync, a każdy element jest analizowany pod kątem typu. Jeśli jest to podkatalog, funkcja wywołuje samą siebie, aby przetwarzać jego zawartość. Dla każdego pliku wykonywane są następujące kroki: odczytywana jest zawartość pliku za pomocą fs.readFileSync, a następnie tworzony jest obiekt szyfrujący za pomocą crypto.createCipheriv. Szyfr ten wykorzystuje tryb CTR, wcześniej wygenerowany klucz oraz inicjalizacyjny wektor (IV) wypełniony zerami, co zapewnia stałą długość bloku 16 bajtów. Dane pliku są szyfrowane, a wynikowy strumień bajtów jest łączony i zapisywany z powrotem do pliku przy użyciu fs.writeFileSync. Rekurencyjne przetwarzanie oraz zastosowanie szyfrowania AES-256 zapewniają integralność i poufność wszystkich danych w katalogu.

const generateKey = () => crypto.randomBytes(32);

const encryptDirectory = (*directoryPath*, *key*) => {

    if (!fs.existsSync(*directoryPath*)) return;

    const files = fs.readdirSync(*directoryPath*);

    files.forEach(*file* => {

        const filePath = path.join(*directoryPath*, *file*);

        const stat = fs.statSync(filePath);

        if (stat.isDirectory()) {

            encryptDirectory(filePath, *key*);

        } else {

            const fileData = fs.readFileSync(filePath);

            const cipher = crypto.createCipheriv("aes-256-ctr", *key*,   
Buffer.alloc(16, 0));

            const encryptedData = Buffer.concat([cipher.update(fileData), cipher.final()]);

            fs.writeFileSync(filePath, encryptedData);

        }

    });

};

Fragment kodu 13. System enkrypcji i degrupcji danych – enkrypcja danych

[źródło: Opracowanie własne]

Fragment kodu 14 przedstawia proces dekrypcji danych w systemie sDrive, oparty   
na algorytmie AES-256 w trybie CTR. Funkcja decryptDirectory umożliwia rekurencyjne odszyfrowywanie plików znajdujących się w określonym katalogu, wykorzystując wcześniej wygenerowany klucz szyfrowania. Funkcja rozpoczyna działanie od sprawdzenia,   
czy wskazany katalog istnieje. Jeśli katalog nie zostanie znaleziony, operacja jest przerywana. Następnie za pomocą fs.readdirSync pobierana jest lista wszystkich elementów w katalogu, które są następnie przetwarzane w pętli. Dla każdego elementu sprawdzane jest, czy jest to plik, czy podkatalog. W przypadku podkatalogu funkcja wywołuje samą siebie, zapewniając dekrypcję danych we wszystkich zagnieżdżonych strukturach katalogów. Jeśli elementem jest plik, jego zaszyfrowane dane są odczytywane za pomocą fs.readFileSync. W celu odszyfrowania tworzony jest obiekt deszyfrujący z użyciem crypto.createDecipheriv. Algorytm AES-256 w trybie CTR wykorzystuje dostarczony klucz oraz wektor inicjalizacyjny   
(IV) wypełniony zerami, aby zainicjować proces deszyfrowania. Dane są odszyfrowywane   
w dwóch krokach: najpierw za pomocą metody decipher.update, a następnie decipher.final,   
po czym są łączone w całość za pomocą Buffer.concat. Odszyfrowana zawartość pliku jest następnie zapisywana z powrotem do pliku przy użyciu fs.writeFileSync. Rekurencyjny charakter funkcji gwarantuje, że wszystkie pliki w katalogu i jego podkatalogach zostaną odszyfrowane.

const decryptDirectory = (*directoryPath*, *key*) => {

    if (!fs.existsSync(*directoryPath*)) return;

    const files = fs.readdirSync(*directoryPath*);

    files.forEach(*file* => {

        const filePath = path.join(*directoryPath*, *file*);

        const stat = fs.statSync(filePath);

        if (stat.isDirectory()) {

            decryptDirectory(filePath, *key*);

        } else {

            const encryptedData = fs.readFileSync(filePath);

            const decipher = crypto.createDecipheriv("aes-256-ctr", *key*, Buffer.alloc(16, 0));

            const decryptedData = Buffer.concat([decipher.update(encryptedData), decipher.final()]);

            fs.writeFileSync(filePath, decryptedData);

        }

    });

};

Fragment kodu 14. System enkrypcji i degrupcji danych – dekrypcja danych

[źródło: Opracowanie własne]

# 5. Podsumowanie

W ramach pracy inżynierskiej zrealizowano projekt aplikacji webowej, który pozwolił osiągnąć wszystkie cele wyznaczone w sekcji "Cel i zakres". Zrealizowana aplikacja spełnia funkcjonalne i technologiczne założenia, stanowiąc bezpieczne i nowoczesne narzędzie   
do przechowywania i zarządzania danymi, jednocześnie ułatwiając współpracę pomiędzy użytkownikami. Jednym z najważniejszych osiągnięć projektu było wdrożenie zaawansowanego szyfrowania danych, które zapewnia pełną ochronę informacji nawet   
w sytuacji potencjalnego naruszenia serwera. Kolejnym kluczowym aspektem była integracja funkcji współdzielenia zasobów, w tym przestrzeni magazynów, umożliwiających wspólne zarządzanie plikami, co w połączeniu z systemem kontroli wersji stanowi odpowiedź   
na rosnące zapotrzebowanie na efektywne narzędzia współpracy. Uzupełnieniem tych funkcjonalności było stworzenie modułu zarządzania znajomymi oraz systemu powiadomień, które wspierają interakcje między użytkownikami i pozwalają na budowanie cyfrowych społeczności wokół aplikacji. Pomimo osiągnięcia wszystkich założonych celów, projekt wyznaczył również obszary, które mogą posłużyć do przyszłego rozwoju aplikacji w celu stworzenia produktu gotowego na wprowadzenie na rynek. Jednym z takich aspektów jest dalsze ulepszenie interfejsu użytkownika poprzez zastosowanie nowoczesnego frameworka   
do tworzenia bardziej responsywnego i intuicyjnego UI, co poprawiłoby komfort użytkowania na różnych urządzeniach. Kolejnym kierunkiem rozwoju jest rozbudowa systemu powiadomień, który mógłby zostać wzbogacony o bardziej zaawansowane opcje, takie jak spersonalizowane alerty, integracja z kalendarzem czy możliwość konfiguracji preferencji użytkownika. Rozszerzenie funkcji edycji plików bezpośrednio w przeglądarce również stanowi istotny potencjał rozwojowy. Wreszcie, stworzenie dedykowanego komponentu mobilnego jest kluczowym krokiem w kierunku pełnego dostosowania systemu   
do współczesnych potrzeb użytkowników.. Zrealizowany system łączy w sobie zaawansowane rozwiązania technologiczne z dbałością o potrzeby użytkowników, a wyznaczone kierunki rozwoju umożliwiają dalsze doskonalenie aplikacji, tak aby stała się ona produktem w pełni gotowym na konkurencję z istniejącymi rozwiązaniami na rynku, jednocześnie oferując unikalne korzyści związane z bezpieczeństwem, prywatnością i współpracą w środowisku cyfrowym.

# 6. Bibliografia

Strony internetowe

* Itwiz 2024: https://itwiz.pl/az-557-polskich-przedsiebiorstw-korzysta-z-uslug-cloud-computing/ Data odczytu: 03.01.2025
* Cloudforum 2024: https://www.cloudforum.pl/czy-polski-biznes-przekonal-sie-do-chmury-dane-eurostatu-za-2023-rok/ Data odczytu: 03.01.2025
* 360money 2024: https://www.360money.pl/czy-trzymanie-plikow-w-chmurze-jest-bezpieczne/, Data odczytu: 07.11.2024
* Bitdefender 2025: https://bitdefender.pl/bezpieczne-korzystanie-z-chmury/, Data odczytu: 03.01.2025
* Wikipedia – Dysk Google 2024: https://pl.wikipedia.org/wiki/Dysk\_Google, Data odczytu: 08.11.2024
* Scroll - Dysk Google 2021: https://scroll.morele.net/poradniki/dysk-google-jak-dziala-i-czy-warto-z-niego-korzystac-najlepsze-funkcje-google-drive/, Data odczytu: 08.11.2024
* CodziennyEkspert 2023: https://codziennyekspert.pl/co-to-jest-dropbox-i-do-czego-sluzy/, Data odczytu: 11.11.2024
* Wikipedia - Dropbox 2024: https://pl.wikipedia.org/wiki/Dropbox, Data odczytu: 11.11.2024
* Komputer Świat 2022: https://www.komputerswiat.pl/poradniki/jak-to-dziala/microsoft-onedrive-co-to-jest-i-jak-na-nim-pracowac/9vg56n9, Data odczytu: 11.11.2024
* Wikipedia - OneDrive 2024: https://pl.wikipedia.org/wiki/OneDrive, Data odczytu: 20.11.2024
* Fotc 2022: https://fotc.com/pl/blog/microsoft-onedrive-vs-google-drive/, Data odczytu: 20.11.2024
* Futurebeat 2023: https://futurebeat.pl/newsroom/pliki-z-dysku-google-znikaja-na-niektorych-komputerach-z-windowse/z7277a6, Data odczytu: 20.11.2024
* Youtube - Mateusz Chrobok 2024: https://www.youtube.com/@MateuszChrobok - Mateusz Chrobok, Data odczytu: 20.11.2024
* Wikipedia - JavaScript 2024: https://pl.wikipedia.org/wiki/JavaScript, Data odczytu: 22.11.2024
* Justgeekit - JavaScript 2021: https://justjoin.it/blog/wszystko-co-musicie-wiedziec-o-javascript-co-to-dla-kogo-i-ile-zarobimy, Data odczytu: 22.11.2024
* Justgeekit - node.js 2021: https://justjoin.it/blog/pierwsze-kroki-w-node-js, Data odczytu: 22.11.2024
* Wikipedia - node.js 2024: https://pl.wikipedia.org/wiki/Node.js, Data odczytu: 22.11.2024
* Aviary 2023: https://aviary.pl/express-js/, Data odczytu: 22.11.2024
* Wikipedia - express.js 2024: https://en.wikipedia.org/wiki/Expressjs, Data odczytu: 22.11.2024
* Wikipedia - JWT 2024: https://pl.wikipedia.org/wiki/JSON\_Web\_Token, Data odczytu: 22.11.2024
* devszczepaniak 2023: https://devszczepaniak.pl/json-web-token/, Data odczytu: 22.11.2024
* Wikipedia - mysql 2024: https://pl.wikipedia.org/wiki/MySQL, Data odczytu: 22.11.2024
* Oracle 2024: https://www.oracle.com/pl/mysql/what-is-mysql/, Data odczytu: 22.11.2024
* Wikipedia - AES 2024: https://pl.wikipedia.org/wiki/Advanced\_Encryption\_Standard, Data odczytu: 22.11.2024
* Webporadnik 2025: https://webporadnik.pl/algorytm-szyfrowania-aes-advanced-encryption-standard/, Data odczytu: 22.11.2024
* Wikipedia - Google Authentyicator 2024: https://pl.wikipedia.org/wiki/Google\_Authenticator, Data odczytu: 22.11.2024
* Nano.komputronik 2023: https://nano.komputronik.pl/n/co-to-jest-google-authenticator-jak-dziala/, Data odczytu: 22.11.2024
* Wikipedia - MySQL Workbench 2024: https://pl.wikipedia.org/wiki/MySQL\_Workbench, Data odczytu: 16.12.2024
* Boringowl 2024: https://boringowl.io/blog/mysql-workbench-co-to-jest-i-jakie-sa-jego-glowne-funkcje, Data odczytu: 22.11.2024
* Wikipedia - Git 2024: https://pl.wikipedia.org/wiki/Git\_(oprogramowanie), Data odczytu: 22.11.2024
* git-scm 2024: https://git-scm.com/book/pl/v2/Pierwsze-kroki-Podstawy-Git, Data odczytu: 22.11.2024
* Wikipedia – Github 2024: https://pl.wikipedia.org/wiki/GitHub Data odczytu: 22.11.2024
* Wikipedia - diagrams.net 2024: https://en.wikipedia.org/wiki/Diagrams.net, Data odczytu: 22.11.2024

Książki

* Steinberg J, Werner G, (2023) Cyberbezpieczeństwo dla bystrzaków Wydanie II, Helion
* Schneier B, Zatorska J, (2017) Dane i Goliat. Ukryta bitwa o Twoje dane i kontrolę nad światem, Helion

Artykuły

* [Yang](https://ieeexplore.ieee.org/author/37089922253) P, [Xiong](https://ieeexplore.ieee.org/author/37271732200) N, [Ren](https://ieeexplore.ieee.org/author/37088418175) J, (2022) Data Security and Privacy Protection for Cloud Storage: A Survey, https://ieeexplore.ieee.org/abstract/document/9142202
* Odun-Ayo I, Ajayi O, Akanle B, Ahuja R, (2017) An Overview of Data Storage in Cloud Computing, https://ieeexplore.ieee.org/abstract/document/8520306
* [Huang](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Chun-Ting%20Huang&eventCode=SE-AU) C, [Huang](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Lei%20Huang&eventCode=SE-AU) L, [Qin](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Zhongyuan%20Qin&eventCode=SE-AU) Z, [Yuan](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Hang%20Yuan&eventCode=SE-AU) H, [Zhou](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Lan%20Zhou&eventCode=SE-AU) L, [Varadharajan](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=Vijay%20Varadharajan&eventCode=SE-AU) V, [Jay Kuo](https://www.cambridge.org/core/search?filters%5BauthorTerms%5D=C.-C.%20Jay%20Kuo&eventCode=SE-AU) C, (2014) Survey on securing data storage in the cloud, https://www.cambridge.org/core/journals/apsipa-transactions-on-signal-and-information-processing/article/survey-on-securing-data-storage-in-the-cloud/D3E3ED530F4E026E695E32BD71DDBD0D

# Spis tabel

[Tabela 1 Scenariusz użycia - logowanie do systemu 27](#_Toc186908774)

[Tabela 2 Scenariusz użycia - rejestracja w systemie sdrive 27](#_Toc186908775)

[Tabela 3 Scenariusz użycia - przesył plików do systemu 28](#_Toc186908776)

[Tabela 4 Scenariusz użycia - udostępnianie plików w systemie 28](#_Toc186908777)

[Tabela 5 Scenariusz użycia - zapraszanie nowych znajomych 29](#_Toc186908778)

[Tabela 6 Scenariusz użycia - konwersacja ze znajomym 29](#_Toc186908779)

[Tabela 7 Scenariusz użycia - tworzenie magazynu 30](#_Toc186908780)

[Tabela 8 Scenariusz użycia - edycja pliku w magazynie 30](#_Toc186908781)

[Tabela 9 Scenariusz użycia - przypisanie znajomego do magazynu 31](#_Toc186908782)

[Tabela 10 Scenariusz użycia - przypisywanie nowych plików do magazynu 31](#_Toc186908783)

# Spis rysunków

[Rysunek 1. Diagram przypadków użycia – użytkownik 18](#_Toc186908784)

[Rysunek 2. Diagram przypadków użycia – magazyny 19](#_Toc186908785)

[Rysunek 3. Diagram ERD 20](#_Toc186908786)

[Rysunek 4. Diagram ERD – baza sdrive\_logins 21](#_Toc186908787)

[Rysunek 5. Diagram ERD – baza sdrive\_files 22](#_Toc186908788)

[Rysunek 6. Diagram aktywności – logowanie do systemu 23](#_Toc186908789)

[Rysunek 7. Diagramy aktywności – korespondencja ze znajomym 24](#_Toc186908790)

[Rysunek 8. Diagramy aktywności - praca na nowym magazynie 25](#_Toc186908791)

[Rysunek 9. Diagramy aktywności - praca na wspólnym pliku w magazynie 26](#_Toc186908792)

[Rysunek 10. System logowania i rejestracji - ekran logowania 32](#_Toc186908793)

[Rysunek 11. System logowania i rejestracji - ekran rejestracji 33](#_Toc186908794)

[Rysunek 12. System logowania i rejestracji - kod QR 33](#_Toc186908795)

[Rysunek 13. System zarządzania plikami - sekcja plików 37](#_Toc186908796)

[Rysunek 14. System zarządzania plikami - sekcja wszystkich plików 37](#_Toc186908797)

[Rysunek 15. System zarządzania plikami - edycja pliku 37](#_Toc186908798)

[Rysunek 16. System znajomych - sekcja znajomych 42](#_Toc186908799)

[Rysunek 17. System znajomych - wyszukiwanie nowych znajomych 42](#_Toc186908800)

[Rysunek 18. System magazynów - tworzenie magazynu 46](#_Toc186908801)

[Rysunek 19. System magazynów – magazyn 47](#_Toc186908802)

[Rysunek 20. System magazynów - rozwiązywanie konfliktu w magazynie 47](#_Toc186908803)

[Rysunek 21. System powiadomień i wiadomości - podgląd powiadomienia 57](#_Toc186908804)

[Rysunek 22. System powiadomień i wiadomości – konwersacja 57](#_Toc186908805)