UNIWERSYTET ŁÓDZKI

WYDZIAŁ EKONOMICZNO-SOCJOLOGICZNY

STUDIA STACJONARNE I STOPNIA

KIERUNEK: INFORMATYKA EKONOMICZNA

MATEUSZ KARAKULSKI

NR ALBUMU 406369

APLIKACJA WEBOWA DO ZARZĄDZANIA DOKUMENTAMI

Praca licencjacka napisana

pod kierunkiem naukowym

dr. inż. Sebastiana Łachecińskiego

Katedra Informatyki Ekonomicznej i Medycznej

ŁÓDŹ, 2025

Spis treści

[Wstęp 5](#_Toc200785738)

[Cel i zakres pracy 7](#_Toc200785739)

[1 Współczesne systemy zarządzania dokumentami 8](#_Toc200785740)

[1.1 Definicja DMS 8](#_Toc200785741)

[1.2 Geneza i rozwój systemów DMS 8](#_Toc200785742)

[1.3 Kluczowe funkcje nowoczesnych systemów DMS 8](#_Toc200785743)

[1.4 Wybrane systemy DMS 9](#_Toc200785744)

[1.4.1 Google Drive 9](#_Toc200785745)

[1.4.2 Microsoft SharePoint 10](#_Toc200785746)

[1.4.3 Dropbox Business 12](#_Toc200785747)

[1.4.4 M-Files 13](#_Toc200785748)

[1.4.5 Alfresco 14](#_Toc200785749)

[1.5 Zestawienie cech wybranych systemów DMS 15](#_Toc200785750)

[1.6 Porównanie aplikacji autorskiej z istniejącymi systemami 17](#_Toc200785751)

[2 Wybór technologii i narzędzi programistycznych 19](#_Toc200785752)

[2.1 Wybór stosu technologicznego 19](#_Toc200785753)

[2.1.1 Backend – Django 19](#_Toc200785754)

[2.1.2 Baza danych – SQLite 21](#_Toc200785755)

[2.1.3 Frontend – HTML, CSS, Bootstrap 22](#_Toc200785756)

[2.1.4 Hosting i chmura – AWS EC2 24](#_Toc200785757)

[2.2 Środowisko programistyczne 26](#_Toc200785758)

[2.2.1 Visual Studio Code 26](#_Toc200785759)

[2.2.2 Konfiguracja środowiska Django 28](#_Toc200785760)

[2.2.3 System kontroli wersji – Git i GitHub 29](#_Toc200785761)

[3 Projektowanie systemu 32](#_Toc200785762)

[3.1 Analiza wymagań 32](#_Toc200785763)

[3.1.1 Wymagania funkcjonalne 32](#_Toc200785764)

[3.1.2 Wymagania niefunkcjonalne 33](#_Toc200785765)

[3.2 Użytkownicy aplikacji 33](#_Toc200785766)

[3.2.1 Opis użytkowników systemu - role i uprawnienia 34](#_Toc200785767)

[3.2.2 Diagram przypadków użycia - UC – Use Case 34](#_Toc200785768)

[4 Projektowanie bazy danych 37](#_Toc200785769)

[4.1 Modelowanie danych 37](#_Toc200785770)

[4.1.1 Konceptualny model danych (CDM) 37](#_Toc200785771)

[4.1.2 Logiczny model danych (LDM) 41](#_Toc200785772)

[4.1.2.1 Wykaz encji 41](#_Toc200785773)

[4.1.2.2 Wykaz atrybutów 42](#_Toc200785774)

[4.1.2.3 Wykaz związków 48](#_Toc200785775)

[4.1.2.4 Reguły biznesowe 51](#_Toc200785776)

[4.1.2.5 Normalizacja 53](#_Toc200785777)

[4.1.2.6 Model w postaci graficznej 54](#_Toc200785778)

[4.1.3 Fizyczny model danych (PDM) 56](#_Toc200785779)

[4.1.3.1 Wykaz tabel fizycznych 56](#_Toc200785780)

[4.1.3.2 Wykaz pól i mapowanie typów danych 58](#_Toc200785781)

[4.1.3.3 Indeksowanie automatyczne 61](#_Toc200785782)

[4.1.3.4 Logika biznesowa na poziomie aplikacji 61](#_Toc200785783)

[4.1.3.5 Sygnały Django 61](#_Toc200785784)

[4.1.3.6 Macierz CRUD 62](#_Toc200785785)

[4.1.3.7 Estymacja rozmiaru bazy danych 67](#_Toc200785786)

[4.1.3.8 Model fizyczny w postaci graficznej 68](#_Toc200785787)

[5 Projektowanie aplikacji webowej 70](#_Toc200785788)

[5.1 Projekt interfejsu użytkownika 70](#_Toc200785789)

[5.1.1 Struktura i szablony HTML 70](#_Toc200785790)

[5.1.2 Stylizacja aplikacji, responsywność i UX 75](#_Toc200785791)

[5.2 Implementacja aplikacji 76](#_Toc200785792)

[5.2.1 Tworzenie systemu logowania i rejestracji 77](#_Toc200785793)

[5.2.2 Obsługa operacji na dokumentach 80](#_Toc200785794)

[5.2.3 Udostępnianie plików i kontrola uprawnień 89](#_Toc200785795)

[5.2.4 Wersjonowanie dokumentów 97](#_Toc200785796)

[6 Konfiguracja i wdrożenie 106](#_Toc200785797)

[6.1 Wymagania sprzętowe 107](#_Toc200785798)

[6.2 Wymagania systemowe 107](#_Toc200785799)

[6.3 Instalacja aplikacji 107](#_Toc200785800)

[6.4 Konfiguracja aplikacji 112](#_Toc200785801)

[6.5 Przewodnik użytkownika 115](#_Toc200785802)

[6.5.1 Logowanie do systemu 115](#_Toc200785803)

[6.5.2 Strona główna 116](#_Toc200785804)

[6.5.3 Wyszukiwanie 119](#_Toc200785805)

[6.5.4 Szczegóły dokumentu 120](#_Toc200785806)

[6.5.5 Przesyłanie dokumentów 121](#_Toc200785807)

[6.5.6 Dodawanie nowych wersji 122](#_Toc200785808)

[6.5.7 Zarządzanie folderami 123](#_Toc200785809)

[6.5.8 Komentarze 125](#_Toc200785810)

[6.5.9 Panel administracyjny (tylko dla administratorów) 126](#_Toc200785811)

[6.6 Analiza SWOT 128](#_Toc200785812)

[7 Testowanie i ocena działania systemu 130](#_Toc200785813)

[7.1 Testowanie aplikacji 130](#_Toc200785814)

[7.2 Analiza wyników 131](#_Toc200785815)

[7.2.1 Testy jednostkowe i integracyjne 131](#_Toc200785816)

[7.2.2 Testy wydajnościowe i obciążeniowe 133](#_Toc200785817)

[7.2.3 Testy bezpieczeństwa 134](#_Toc200785818)

[8 Podsumowanie i wnioski 135](#_Toc200785819)

[8.1 Przegląd realizacji projektu 135](#_Toc200785820)

[8.2 Osiągnięcie nadrzędnego celu pracy 136](#_Toc200785821)

[8.3 Główne trudności i sposoby ich rozwiązania 137](#_Toc200785822)

[8.4 Możliwości rozbudowy i dalszego rozwoju systemu 138](#_Toc200785823)

[9 Bibliografia 140](#_Toc200785824)

[10 Spis tabel 141](#_Toc200785825)

[11 Spis rysunków 142](#_Toc200785826)

[12 Spis załączników 144](#_Toc200785827)

# Wstęp

Współczesne organizacje, niezależnie od wielkości i branży, generują ogromne ilości dokumentów elektronicznych. Efektywne zarządzanie nimi jest kluczowe dla optymalizacji procesów biznesowych, zapewnienia bezpieczeństwa danych oraz usprawnienia współpracy zespołowej. Dynamiczny rozwój technologii chmurowych oraz narzędzi wspierających pracę zdalną wymusza poszukiwanie rozwiązań, które umożliwią sprawne przechowywanie, wersjonowanie, udostępnianie i ochronę dokumentów.

W odpowiedzi na te potrzeby powstały systemy zarządzania dokumentami (DMS – Document Management Systems), które pozwalają na organizację, kontrolę dostępu oraz współdzielenie plików. Chociaż na rynku dostępnych jest wiele gotowych rozwiązań, część firm i instytucji poszukuje systemów dostosowanych do swoich specyficznych wymagań. Szczególną grupą są małe i średnie przedsiębiorstwa (MŚP), które często nie dysponują budżetem na komercyjne oprogramowanie, a jednocześnie potrzebują efektywnego i bezpiecznego narzędzia do zarządzania dokumentacją. Stwierdzenie to popierają moje własne obserwacje i doświadczenia zdobyte podczas odbywania praktyki zawodowej. Zauważyłem, że firma z sektora MŚP nie używała żadnych rozwiązań związanych z systemami zarządzania dokumentami. Ograniczone możliwości finansowe nie pozwalały na rozbudowę DMS, a dodatkowo pracownicy nie posiadali odpowiedniej wiedzy i doświadczenia w tym zakresie.

Wiele organizacji zmaga się z chaosem informacyjnym, brakiem centralnego repozytorium oraz trudnościami w zarządzaniu uprawnieniami użytkowników. Dlatego kluczowym wyzwaniem staje się implementacja dedykowanego systemu DMS, który będzie intuicyjny w obsłudze, będzie wspierał współdzielenie dokumentów w czasie rzeczywistym oraz gwarantował wysoki poziom bezpieczeństwa. Powyżej przedstawione fakty oraz własne spostrzeżenia zdobyte podczas praktyk skłoniły mnie do podjęcia się tematu pracy dotyczącego opracowania aplikacji usprawniającej pracę zespołową poprzez bezpieczne i efektywne współtworzenie dokumentów w czasie rzeczywistym. Rozwiązanie to pozwoli na eliminację problemów związanych z kontrolą danych oraz ograniczenie wysokich kosztów licencyjnych, tym samym wypełniając lukę w dostępnych narzędziach do współpracy online.

W pierwszym rozdziale zaprezentowano nowoczesne systemy zarządzania dokumentami, ich genezę oraz rozwój. Przedstawiono kluczowe funkcje, jakie powinny posiadać współczesne rozwiązania, takie jak przechowywanie, wersjonowanie, udostępnianie oraz zabezpieczanie dokumentów. Opisano również najpopularniejsze systemy, w tym Google Drive, Microsoft SharePoint, Dropbox Business, M-Files i Alfresco, porównując ich strukturę organizacji plików, integrację z innymi systemami, wersjonowanie dokumentów, automatyzację procesów oraz bezpieczeństwo danych.

Drugi rozdział dotyczy wyboru technologii i narzędzi niezbędnych do realizacji części projektowo implementacyjnej pracy dyplomowej. Przedstawiono w nim stos technologiczny, obejmujący backend oparty na Django i Pythonie, bazę danych SQLite, frontend w HTML, CSS i Bootstrap, a także hosting w chmurze AWS EC2. Opisano konfigurację środowiska programistycznego, wykorzystywane IDE i narzędzia developerskie, a także system kontroli wersji Git i GitHub.

Trzeci rozdział dotyczy projektowania systemu, zaczynając od analizy wymagań, w tym funkcjonalnych i niefunkcjonalnych. Przedstawiono role użytkowników, diagram przypadków użycia oraz architekturę systemu opartą na modelu klient-serwer i wzorcu MVC w Django. Szczególną uwagę poświęcono integracji z chmurą AWS EC2.

W czwartym rozdziale zaprezentowano proces projektowania bazy danych, koncentrując się na modelowaniu konceptualnym, logicznym i fizycznym. Opisano strukturę bazy SQLite, organizację tabel, indeksowanie oraz optymalizację zapytań. Przedstawiono sposoby przechowywania dokumentów w postaci binarnej (BLOB), mechanizmy zarządzania dostępem do dokumentów, wersjonowanie oraz zarządzanie uprawnieniami, podkreślając rolę bezpieczeństwa i szyfrowania danych.

Piąty rozdział skupia się na projektowaniu aplikacji webowej, w tym interfejsu użytkownika. Opisano strukturę HTML, stylizację w CSS i Bootstrap oraz kwestie responsywności i UX. Ponadto zawiera on opis implementacji funkcji logowania i rejestracji, operacji na dokumentach, udostępniania plików oraz wersjonowania.

Rozdział szósty dotyczy konfiguracji i wdrożenia aplikacji w środowisku AWS EC2. Opisano w nim proces hostowania aplikacji Django w AWS EC2, integrację z bazą SQLite oraz zarządzanie uprawnieniami. Przedstawiono procedurę wdrażania systemu, konfigurację zmiennych środowiskowych, monitorowanie wydajności, logowanie błędów, skalowanie aplikacji oraz zabezpieczenia i backup danych.

W siódmym rozdziale przedstawiono testowanie i ocenę działania systemu. Przedstawiono testy jednostkowe i integracyjne w Django, testy wydajnościowe i obciążeniowe oraz analizę wyników. Oceniono funkcjonalność aplikacji, jej skalowalność oraz jakość interfejsu użytkownika. Przeanalizowano również napotkane problemy i ograniczenia systemu oraz zaproponowano możliwości jego usprawnienia.

Ostatni rozdział stanowi podsumowanie pracy dyplomowej. Zawarto w nim przegląd kluczowych etapów, przeprowadzono krytyczną analizę podjętych decyzji technologicznych oraz przedstawiono sposoby rozwiązania napotkanych trudności. Na koniec zaproponowano możliwości dalszej rozbudowy systemu i jego integracji z innymi rozwiązaniami.

Praca zawiera również bibliografię, spis tabel, ilustracji oraz załączniki, które dostarczają dodatkowych informacji i uzupełniają treść dokumentu.

# Cel i zakres pracy

Celem pracy jest stworzenie aplikacji webowej do zarządzania dokumentami, która będzie dostosowana do potrzeb nowoczesnych organizacji.

Nadrzędnym celem pracy dyplomowej jest usprawnienie i ułatwienie pracy zespołowej. Stworzone rozwiązanie pozwoli wyeliminować problemy związane z brakiem kontroli nad danymi oraz wysokimi kosztami licencyjnymi. System zapewni bezpieczne i efektywne współtworzenie dokumentów w czasie rzeczywistym, co zwiększy produktywność i komfort pracy.

Zakres pracy obejmuje:

1. Zarządzanie dokumentami
   * Przesyłanie, przechowywanie i pobieranie plików,
   * Obsługa różnych formatów dokumentów,
   * Wersjonowanie dokumentów,
   * Tagowanie dokumentów,
   * Wyszukiwanie.
2. Zarządzanie użytkownikami i uprawnieniami
   * System rejestracji i logowania,
   * Role użytkowników (administrator, edytor, czytelnik),
   * Nadawanie i ograniczanie uprawnień,
   * Logowanie i audyt aktywności użytkowników.
3. Bezpieczeństwo i zgodność z regulacjami
   * Szyfrowanie plików i danych,
   * Dwuetapowa weryfikacja logowania,
   * Mechanizm audytowania dostępu,
   * Regularne kopie zapasowe.
4. Interfejs użytkownika
   * Intuicyjny interfejs webowy,
   * Responsywność dla urządzeń mobilnych,
   * Struktura nawigacji ułatwiająca zarządzanie dokumentami.

# Współczesne systemy zarządzania dokumentami

## Definicja DMS

Systemy zarządzania dokumentami (DMS – Document Management Systems) to oprogramowanie umożliwiające organizacjom efektywne przechowywanie, udostępnianie, zarządzanie i kontrolę nad dokumentami elektronicznymi. DMS nie tylko gromadzi pliki w centralnym repozytorium, ale także wspiera automatyzację procesów biznesowych, zapewniając zgodność z regulacjami prawnymi oraz ułatwiając dostęp do kluczowych informacji. Jak zauważył Keith D. Swenson w książce „Document and Workflow Management”: „efektywne systemy DMS nie tylko przechowują dokumenty, ale także umożliwiają automatyzację procesów biznesowych, co znacząco wpływa na wydajność organizacji”.

## Geneza i rozwój systemów DMS

Historia systemów zarządzania dokumentami sięga lat 80. XX wieku, kiedy organizacje zaczęły zdawać sobie sprawę z potrzeby cyfrowej archiwizacji dokumentów. Początkowo systemy te miały formę elektronicznych baz danych umożliwiających przechowywanie skanowanych dokumentów w formie obrazów.

W latach 90. rozwój technologii internetowych oraz wzrost ilości danych doprowadził do pojawienia się bardziej zaawansowanych systemów zarządzania dokumentami, które wprowadziły metadane i mechanizmy wyszukiwania treści, a także umożliwiły dostęp do plików przez sieć.

Na początku XXI wieku DMS stały się kluczowym elementem strategii zarządzania informacją w firmach i instytucjach. Wprowadzenie chmury obliczeniowej umożliwiło przechowywanie dokumentów na zdalnych serwerach, co zapewniło użytkownikom łatwy dostęp do plików z różnych urządzeń i lokalizacji. Dodatkowo, mechanizmy sztucznej inteligencji i uczenia maszynowego przyczyniły się do automatyzacji procesów zarządzania dokumentacją, zwiększając efektywność organizacji.

## Kluczowe funkcje nowoczesnych systemów DMS

Nowoczesne systemy zarządzania dokumentami (DMS) oferują szereg funkcji, które usprawniają organizację, przechowywanie oraz współdzielenie dokumentów w firmach i instytucjach. Poniżej przedstawiono najważniejsze z nich:

1. Przechowywanie w chmurze – zapewnia użytkownikom dostęp do dokumentów z różnych urządzeń i lokalizacji, eliminując konieczność korzystania z fizycznych serwerów.
2. Współpraca i udostępnianie – umożliwia użytkownikom jednoczesną pracę nad dokumentami, co zwiększa efektywność zespołową.
3. Bezpieczeństwo danych – zaawansowane mechanizmy kontroli dostępu, szyfrowania oraz wersjonowania dokumentów chronią przed nieautoryzowanym dostępem i utratą danych. Jak podkreślają Julie McLeod i Catherine Hare w „Managing Electronic Records”: „zarządzanie dokumentami elektronicznymi wymaga nie tylko technologii, ale także właściwej polityki organizacyjnej i zgodności z regulacjami prawnymi”.
4. Zaawansowane mechanizmy wyszukiwania – dzięki zastosowaniu metadanych, tagów oraz sztucznej inteligencji użytkownicy mogą szybko odnaleźć potrzebne pliki. Jak zaznacza Azad Adam w „Document Management for the Enterprise”: „organizacje, które efektywnie wdrażają DMS, mogą znacząco zredukować czas wyszukiwania informacji, poprawiając jednocześnie kontrolę nad dokumentacją i zgodność z przepisami”.
5. Integracja z narzędziami biurowymi – nowoczesne systemy DMS współpracują z aplikacjami takimi jak Microsoft Office, Google Workspace czy systemami ERP, co pozwala na płynne zarządzanie dokumentacją w ramach różnych procesów biznesowych. Jak zwracają uwagę Jan vom Brocke i Alexander Simons w „Enterprise Content Management in Information Systems Research”: „integracja DMS z narzędziami biurowymi i systemami ERP jest kluczowa dla zwiększenia efektywności procesów biznesowych i lepszego zarządzania wiedzą w organizacji”.

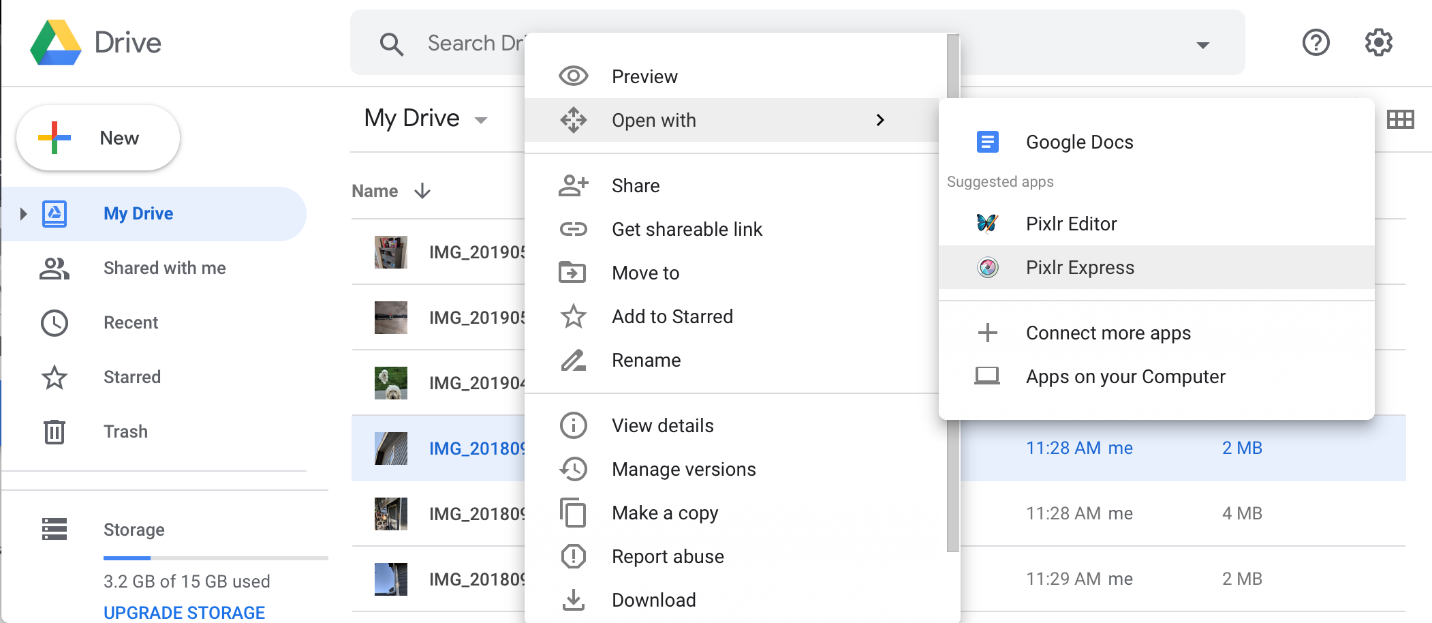
## Wybrane systemy DMS

Wybór odpowiedniego systemu DMS zależy od specyfiki działalności, potrzeb użytkowników oraz poziomu zaawansowania funkcji. W kolejnych podrozdziałach opisano wybrane systemy, które różnią się pod względem funkcjonalności, integracji oraz skomplikowania wdrożenia – od prostych rozwiązań chmurowych, takich jak Google Drive i Dropbox Business, po zaawansowane platformy, takie jak Microsoft SharePoint, M-Files czy Alfresco.

### Google Drive

Google Drive to proste i intuicyjne rozwiązanie, idealne dla małych firm i zespołów, które szukają łatwego narzędzia do przechowywania i współdzielenia plików. Jego główną zaletą jest pełna integracja z Google Workspace, możliwość edycji dokumentów w czasie rzeczywistym oraz wbudowane wyszukiwanie z OCR. Jednak brakuje mu zaawansowanych funkcji zarządzania dokumentami, takich jak workflow, metadane czy szczegółowa kontrola dostępu, co sprawia, że nie sprawdzi się w dużych organizacjach wymagających pełnoprawnego systemu DMS. Na rysunku 1 przedstawiono interfejs Google Drive.

Rysunek 1. Interfejs Google Drive



Źródło: https://developers.google.com/static/drive/images/open-with-2015.png(18.02.2025)

Zalety:

* Podstawowa organizacja w folderach
* Wersjonowanie dokumentów Google
* OCR wbudowane w wyszukiwanie
* Edycja w czasie rzeczywistym w Google Docs
* Szyfrowanie AES-256, podstawowe raporty, zgodność z GDPR (w Google Workspace).
* Integracja z Google Workspace
* Bardzo intuicyjny interfejs
* Szybkie wdrożenie
* Darmowa wersja (15 GB)
* Prosta obsługa

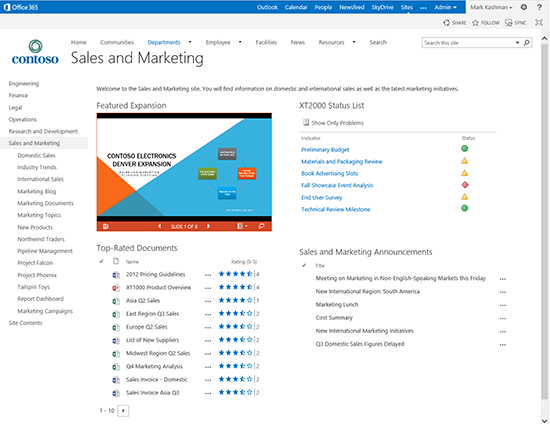
Wady:

* Brak metadanych i zaawansowanych workflow
* Brak pełnej integracji z Microsoft Office
* W podstawowej wersji brak pełnej zgodności z HIPAA i ISO 27001
* Ograniczona integracja z systemami ERP i CRM
* Brak zaawansowanych funkcji DMS
* Płatne plany w Google Workspace

### Microsoft SharePoint

Microsoft SharePoint to rozwiązanie dla firm korzystających z ekosystemu Microsoft 365, które potrzebują zaawansowanego workflow, automatyzacji i ścisłej kontroli nad dokumentami. Jego atuty to pełna integracja z Office 365, wersjonowanie, audyt oraz zgodność z regulacjami, co czyni go świetnym wyborem dla średnich i dużych organizacji. Jednak SharePoint wymaga konfiguracji i administracji, a jego interfejs może być skomplikowany dla użytkowników, którzy wcześniej nie mieli styczności z tym systemem. Na rysunku 2 przedstawiono interfejs Microsoft SharePoint.

Rysunek 2. Interfejs Microsoft SharePoint



Źródło:https://www.microsoft.com/en-us/microsoft-365/blog/wpcontent/uploads/sites/2/migrated-images/64/5810.image-1.png(18.02.2025)

Zalety:

* Zaawansowana organizacja, wersjonowanie, metadane
* Integracja z Microsoft 365
* Współpraca w czasie rzeczywistym przez Microsoft 365,
* Zaawansowane workflow
* Zaawansowana kontrola dostępu, audyt, zgodność z GDPR, HIPAA, ISO 27001
* Pełna integracja z Microsoft 365, ERP, CRM, Power Automate

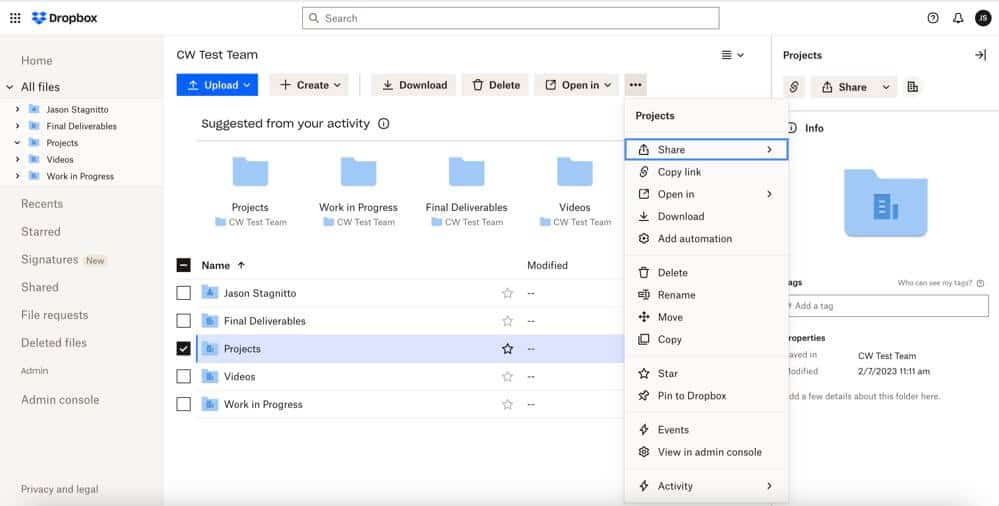
Wady:

* Skomplikowana konfiguracja, wymaga szkoleń
* Integracja głównie z ekosystemem Microsoft, trudniejsza z innymi systemami
* Wymaga odpowiedniej konfiguracji do spełnienia norm bezpieczeństwa

### Dropbox Business

Dropbox Business to najłatwiejszy w obsłudze system, który doskonale sprawdzi się w małych i średnich firmach, które potrzebują podstawowego systemu przechowywania i współdzielenia plików. Jego kluczowe zalety to szybka synchronizacja, prosta nawigacja i integracja z Google Workspace oraz Microsoft 365. Jednak Dropbox Business nie posiada rozbudowanych funkcji workflow, metadanych ani zaawansowanego zarządzania uprawnieniami, co sprawia, że nie nadaje się jako pełnoprawny system DMS dla większych firm. Na rysunku 3 przedstawiono interfejs Dropbox Business.

Rysunek 3. Interfejs Dropbox Business



Źródło: https://www.cloudwards.net/wp-content/uploads/2023/02/dropbox-business-menu.jpg(18.02.2025)

Zalety:

* Prosta organizacja folderów
* Przechowywanie historii wersji do 180 dni
* Szybka synchronizacja
* Szyfrowanie, 2FA, monitoring aktywności
* Integracja z Microsoft 365 i Google Workspace
* Najprostszy w obsłudze, przypomina klasyczny eksplorator plików
* Model subskrypcyjny, elastyczne plany dla firm

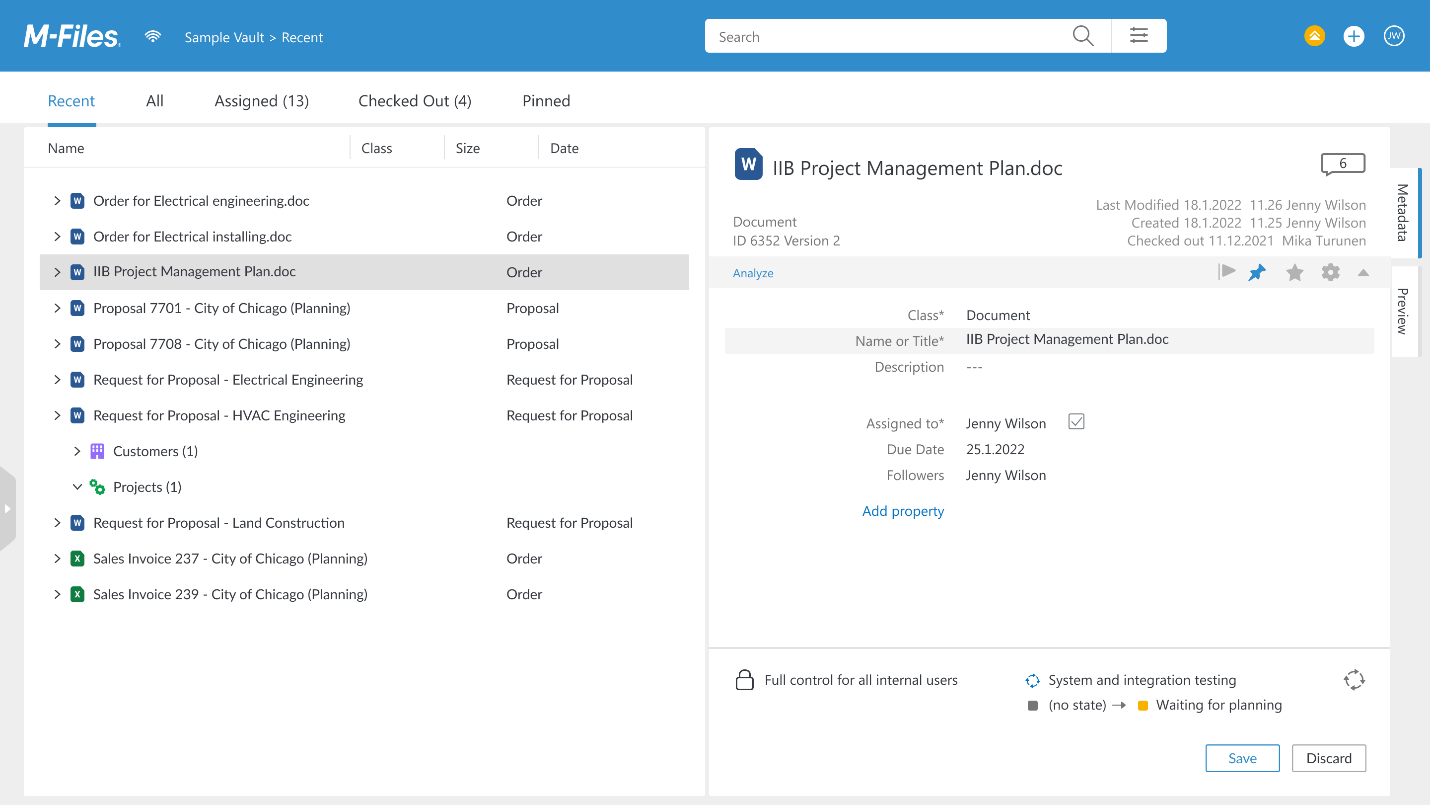
Wady:

* Brak zaawansowanego workflow, ograniczone metadane
* Brak edycji w czasie rzeczywistym
* Niepełna zgodność z HIPAA w podstawowych planach
* Brak natywnej integracji z ERP/CRM
* Nie posiada zaawansowanych funkcji ECM

### M-Files

M-Files to zaawansowane rozwiązanie dla firm, które zarządzają dużą ilością dokumentów i potrzebują inteligentnego systemu wyszukiwania oraz automatyzacji procesów. Wyróżnia się organizacją dokumentów opartą na metadanych, zaawansowanym wersjonowaniem i integracją z ERP oraz CRM, co pozwala na bardziej efektywne zarządzanie treścią. Jednak ze względu na nietypowe podejście do organizacji dokumentów, wymaga szkolenia użytkowników i odpowiedniego wdrożenia. Jest to świetny wybór dla firm, które chcą odchodzić od tradycyjnej struktury folderów i usprawniać przepływ dokumentów. Na rysunku 4 przedstawiono interfejs M-Files.

Rysunek 4. Interfejs M-Files



Źródło: https://indixio.com/wp-content/uploads/2022/10/desktop-16-9-metadata.png(18.02.2025)

Zalety:

* Organizacja na podstawie metadanych
* Inteligentne wyszukiwanie
* Zaawansowane wersjonowanie
* Pełna kontrola dostępu, audyty, zgodność z GDPR, HIPAA, ISO 27001
* Integracja z Office 365, ERP, CRM, API do innych systemów

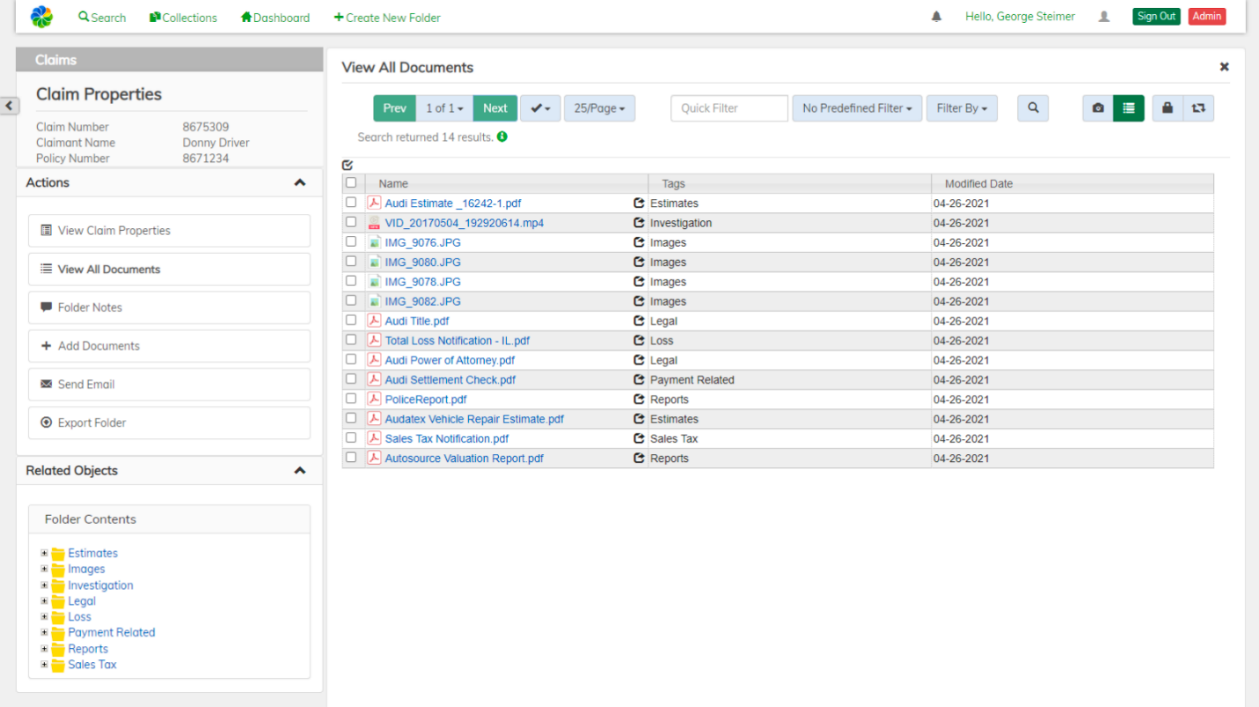
Wady:

* Wymaga nauki nowego podejścia do organizacji dokumentów, szkolenie użytkowników konieczne.
* Brak darmowej wersji, koszt zależny od funkcji i liczby użytkowników
* Brak natywnej edycji dokumentów w przeglądarce

### Alfresco

Alfresco to najbardziej elastyczne i skalowalne rozwiązanie, które najlepiej sprawdzi się w dużych organizacjach i instytucjach publicznych, wymagających zaawansowanego zarządzania dokumentami oraz integracji z systemami ERP i CRM. Dzięki open-source’owej architekturze, Alfresco można wdrożyć zarówno w chmurze, on-premise, jak i w modelu hybrydowym. Jego największe atuty to pełnotekstowe wyszukiwanie, rozbudowane workflow oraz wysoki poziom bezpieczeństwa. Jednak wymaga zaawansowanej wiedzy IT do wdrożenia i administracji, co sprawia, że jest mniej odpowiedni dla firm, które szukają prostych rozwiązań. Na rysunku 5 przedstawiono interfejs Alfresco.

Rysunek . Interfejs Alfresco



Źródło: https://docs.alfresco.com/content-accelerator/images/aca-intro-claims-management.png(18.02.2025)

Zalety:

* Zaawansowana klasyfikacja
* Obsługa metadanych
* Workflow
* Pełnotekstowe wyszukiwanie (Apache Solr).
* Obsługuje wersjonowanie i przepływy pracy
* Spełnia GDPR, HIPAA, ISO 27001, szczegółowe logi audytowe i kontrola dostępu
* Open-source,
* Szerokie możliwości integracji z ERP, CRM, Microsoft 365, Google Workspace.
* Wersja Community (darmowa)

Wady:

* Skomplikowane wdrożenie i administracja, wymaga specjalistycznej wiedzy IT.
* Płatna wersja Enterprise z dodatkowymi funkcjami
* Złożona konfiguracja
* Wymaga optymalizacji dla dużych zbiorów dokumentów
* Edycja wymaga integracji z zewnętrznymi edytorami
* Wymaga specjalistycznej wiedzy IT do konfiguracji bezpieczeństwa
* Konieczność zarządzania serwerami

## Zestawienie cech wybranych systemów DMS

Wybór odpowiedniego systemu zarządzania dokumentami (DMS) zależy od takich czynników jak sposób organizacji plików, poziom integracji z innymi narzędziami, bezpieczeństwo danych oraz możliwości automatyzacji procesów. W tabeli 1 przedstawiono porównanie kluczowych cech systemów zaprezentowanych w poprzednim podrozdziale.

Tabela 1. Cechy wybranych DMS

|  | Google Drive | Microsoft SharePoint | Dropbox Business | M-Files | Alfresco |
| --- | --- | --- | --- | --- | --- |
| Struktura organizacji plików | Klasyczna (hierarchiczne foldery) | Klasyczna + metadane | Klasyczna (hierarchiczne foldery) | Metadane (dynamiczna organizacja) | Klasyczna + BPM (workflow + foldery) |
| Integracja | Google Workspace | Microsoft 365 | Google, Microsoft 365 | ERP, CRM | SAP, Office 365 |
| Wersjonowanie dokumentów | Ograniczone (ręczne przywracanie starszych wersji, 30 dni) | Pełne wersjonowanie z możliwością przywracania dowolnej wersji (bez ograniczeń czasowych) | Wersjonowanie do 180 dni | Zaawansowane, z możliwością porównywania wersji (bez ograniczeń czasowych) | Pełne wersjonowanie z archiwizacją długoterminową (powyżej 1 roku) |
| Automatyzacja procesów | Nie | Tak | Nie | Tak | Tak |
| Model dostępu | Publiczny (udostępnianie poprzez linki) | Firmowy (dostęp tylko dla pracowników organizacji) | Publiczny (udostępnianie poprzez linki) | Role i metadane (dostęp zależny od atrybutów dokumentów i ról użytkowników) | Role i przepisy (dostęp kontrolowany przez przepisy prawne i role użytkowników) |
| Open-source | Nie | Nie | Nie | Nie | Tak |
| Hosting | Google Cloud | Microsoft Azure | AWS, własny serwer | Własny serwer, chmura | Własny serwer |
| Baza danych | Chmura (Google Cloud Storage, Firebase) | Relacyjna (Microsoft SQL Server) | Chmura (AWS, własny serwer) | Relacyjna (PostgreSQL, MySQL) | Relacyjna (PostgreSQL, MySQL, Oracle) |
| Interfejs użytkownika | Web, mobilny | Web, mobilny | Web, mobilny | Web | Web |
| Bezpieczeństwo danych | Podstawowe (szyfrowanie plików, dwuetapowa weryfikacja) | Zaawansowane (szyfrowanie, audyt dostępu, polityki bezpieczeństwa) | Dobre (szyfrowanie AES-256, ochrona hasłem) | Bardzo dobre (kontrola dostępu na poziomie metadanych, szyfrowanie) | Bardzo dobre (zgodność z regulacjami RODO, HIPAA, audyt bezpieczeństwa) |

Źródło: opracowanie własne

Analizując zestawione cechy, można zauważyć kilka istotnych trendów. Mianowicie Microsoft SharePoint i Alfresco wyróżniają się zaawansowanym systemem wersjonowania dokumentów, który pozwala na długoterminową archiwizację. Z kolei M-Files wyróżnia się dynamicznym podejściem do organizacji plików, bazującym na metadanych, co zapewnia większą elastyczność w porównaniu do klasycznej hierarchicznej struktury folderów.

Pod względem integracji widać wyraźny podział – rozwiązania od Google i Microsoftu ściśle współpracują ze swoimi ekosystemami (Google Workspace, Microsoft 365), podczas gdy Alfresco czy M-Files oferują bardziej otwarte podejście, umożliwiając integrację z systemami ERP czy CRM.

W zakresie bezpieczeństwa danych najlepsze parametry oferują M-Files i Alfresco, które spełniają wymogi RODO i HIPAA oraz stosują zaawansowane metody szyfrowania. Warto także zauważyć, że większość nowoczesnych systemów DMS umożliwia dostęp z poziomu przeglądarki internetowej oraz aplikacji mobilnej, co ułatwia pracę zdalną.

## Porównanie aplikacji autorskiej z istniejącymi systemami

Pełna kontrola nad danymi i bezpieczeństwo

Stworzona aplikacja webowa do zarządzania dokumentami łączy prostotę obsługi znaną z Google Drive i Dropbox z zaawansowanymi funkcjami, które oferują systemy takie jak M-Files i Alfresco. Wykorzystanie SQLite jako bazy danych zapewnia efektywne przechowywanie dokumentów, a wbudowane mechanizmy szyfrowania gwarantują bezpieczeństwo plików. Autoryzacja użytkowników oraz system kontroli dostępu umożliwią nadawanie odpowiednich uprawnień, co zwiększa prywatność i ochronę danych w organizacji.

Zaawansowane wersjonowanie i brak kosztów licencyjnych

W odróżnieniu od Dropboxa, który ogranicza wersjonowanie dokumentów do 180 dni, autorska aplikacja oferuje pełne wersjonowanie plików, umożliwiając zarówno przywracanie starszych wersji, jak i ich archiwizację. Dodatkowo, system pozwala na porównywanie wersji dokumentów, co ułatwia śledzenie zmian i współpracę zespołową. W przeciwieństwie do Microsoft SharePoint czy M-Files, rozwiązanie to jest w pełni open-source, tym samym eliminuje wysokie koszty licencyjne.

Łatwa obsługa

Dzięki obsłudze wielu typów plików, użytkownicy nie będą ograniczeni do konkretnego ekosystemu, co zapewni większą elastyczność. Prosty i intuicyjny interfejs sprawia, że nawet osoby bez zaawansowanej wiedzy technicznej będą mogły z łatwością korzystać z oprogramowania.

Wsparcie dla pracy zespołowej i elastyczność dostępu

System komentarzy i powiadomień pozwala użytkownikom na bieżąco śledzić zmiany i reagować na edycje współpracowników. Ponadto, dzięki synchronizacji danych użytkownicy będą mieli dostęp do swoich plików z dowolnego miejsca i urządzenia.

Open-source i prostsze wdrożenie niż konkurencja

Podobnie jak M-Files i Alfresco, aplikacja jest open-source, co pozwala na jej dalszy rozwój i dostosowanie do indywidualnych potrzeb firm. W przeciwieństwie do bardziej skomplikowanych rozwiązań, wdrożenie i administracja systemu są maksymalnie uproszczone, dzięki czemu mniejsze organizacje będą mogły z łatwością skorzystać z pełni jego możliwości bez konieczności zatrudniania specjalistów IT.

Dzięki temu aplikacja jest nowoczesnym, funkcjonalnym i ekonomicznym rozwiązaniem do zarządzania dokumentami. Łączy ona w sobie prostotę użytkowania znaną z Google Drive i Dropbox z elastycznością i możliwościami Alfresco i M-Files, ale bez ich wysokich kosztów i złożoności wdrożeniowej.

# Wybór technologii i narzędzi programistycznych

Wybór odpowiednich technologii i narzędzi programistycznych ma kluczowe znaczenie dla efektywności procesu tworzenia aplikacji. Decyzja ta wpływa nie tylko na sposób implementacji poszczególnych funkcjonalności, ale także na wydajność, skalowalność oraz późniejszą konserwację systemu. W niniejszym rozdziale zostanie przedstawiony stos technologiczny, a także środowisko programistyczne, wykorzystane w ramach pracy dyplomowej.

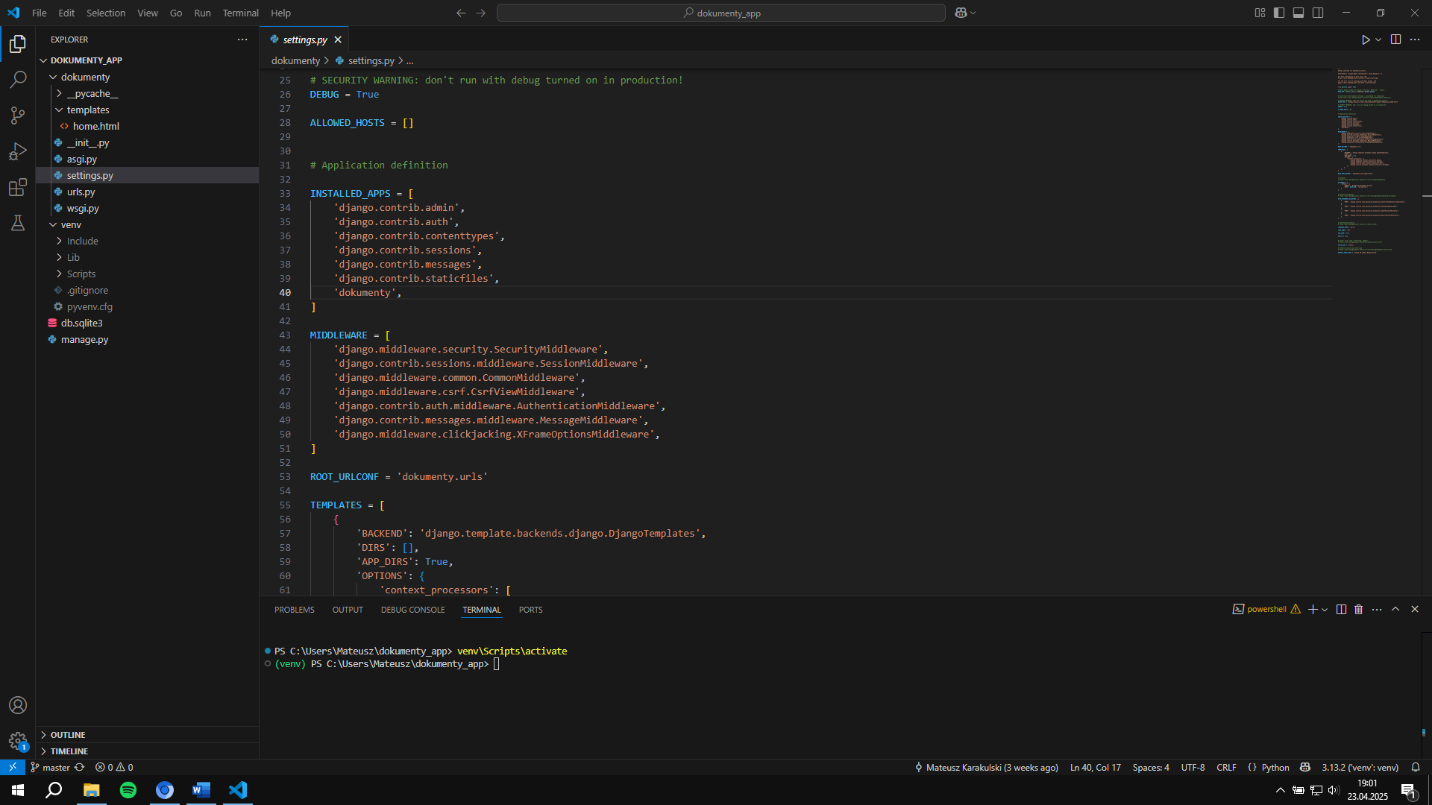
## Wybór stosu technologicznego

Wybór odpowiedniego stosu technologicznego ma kluczowe znaczenie dla powodzenia projektu, wpływając na jego wydajność, skalowalność oraz łatwość w utrzymaniu kodu źródłowego i infrastruktury. W niniejszej sekcji opisano technologie wybrane do implementacji aplikacji, uwzględniając zarówno backend, bazę danych, jak i frontend oraz hosting. Decyzja o ich zastosowaniu została podjęta na podstawie ich popularności, dostępności zasobów oraz zgodności z wymaganiami projektu.

### Backend – Django

Django to popularny framework webowy dla języka Python, który umożliwia szybkie i efektywne tworzenie aplikacji internetowych. Na rysunku 6 przedstawiono fragment kodu z Django, dotyczący konfiguracji projektu, która definiuje kluczowe ustawienia aplikacji, takie jak bazy danych, aplikacje, ścieżki czy ustawienia bezpieczeństwa.

Rysunek 6. Django - przykładowy kod



Źródło: opracowanie własne

Wybór Django do realizacji aplikacji webowej do zarządzania dokumentami wynika z kilku kluczowych zalet tego frameworka:

Szybkość i wydajność

Django oferuje dobrze zoptymalizowane narzędzia i mechanizmy, które pozwalają na szybkie tworzenie aplikacji. Dzięki wbudowanemu ORM (Object-Relational Mapping) można efektywnie zarządzać bazą danych, co przyspiesza proces tworzenia aplikacji i redukuje czas potrzebny na pisanie zapytań SQL.

Modularność i elastyczność

Django wykorzystuje wzorzec architektoniczny Model-View-Controller (MVC), co ułatwia organizację kodu i jego dalszą rozbudowę. Separacja logiki biznesowej od warstwy prezentacji sprawia, że aplikacja jest bardziej przejrzysta i łatwa w utrzymaniu.

Wbudowane mechanizmy bezpieczeństwa

Django posiada zaawansowane mechanizmy zabezpieczeń, które chronią aplikację przed atakami takimi jak SQL Injection, XSS (Cross-Site Scripting) czy CSRF (Cross-Site Request Forgery). Wbudowane mechanizmy uwierzytelniania i autoryzacji użytkowników pozwalają na implementację zaawansowanego systemu kontroli dostępu do dokumentów.

Bogata dokumentacja i wsparcie społeczności

Django posiada jedną z najlepszych dokumentacji spośród frameworków webowych. Dzięki aktywnej społeczności programistycznej, dostępne są liczne tutoriale, rozszerzenia oraz wsparcie na forach internetowych, co ułatwia rozwiązywanie problemów i przyspiesza proces nauki.

Skalowalność i gotowość do pracy w chmurze

Aplikacje napisane w Django mogą być łatwo skalowane, co jest istotne w przypadku rozwoju projektu. Framework dobrze współpracuje z usługami chmurowymi, takimi jak AWS EC2, co umożliwia elastyczne zarządzanie zasobami oraz wdrażanie aplikacji w środowisku produkcyjnym.

Wbudowany panel administracyjny

Django oferuje automatycznie generowany panel administracyjny, który ułatwia zarządzanie danymi aplikacji. Dzięki temu można szybko zarządzać użytkownikami, dokumentami i uprawnieniami bez konieczności pisania dodatkowego kodu.

Obsługa SQLite i innych baz danych

Django jest kompatybilne z różnymi systemami zarządzania bazami danych, w tym SQLite, PostgreSQL, MySQL i Oracle. W kontekście tej aplikacji wybór SQLite jest uzasadniony ze względu na jego prostotę, lekkość i brak konieczności konfiguracji dodatkowego serwera bazy danych.

Dobre wsparcie dla testowania

Django posiada wbudowane narzędzia do testowania aplikacji, co daje gwarancje jakości kodu na odpowiednim poziomie i wykrywanie błędów na wczesnym etapie rozwoju.

Gotowe komponenty i rozszerzenia

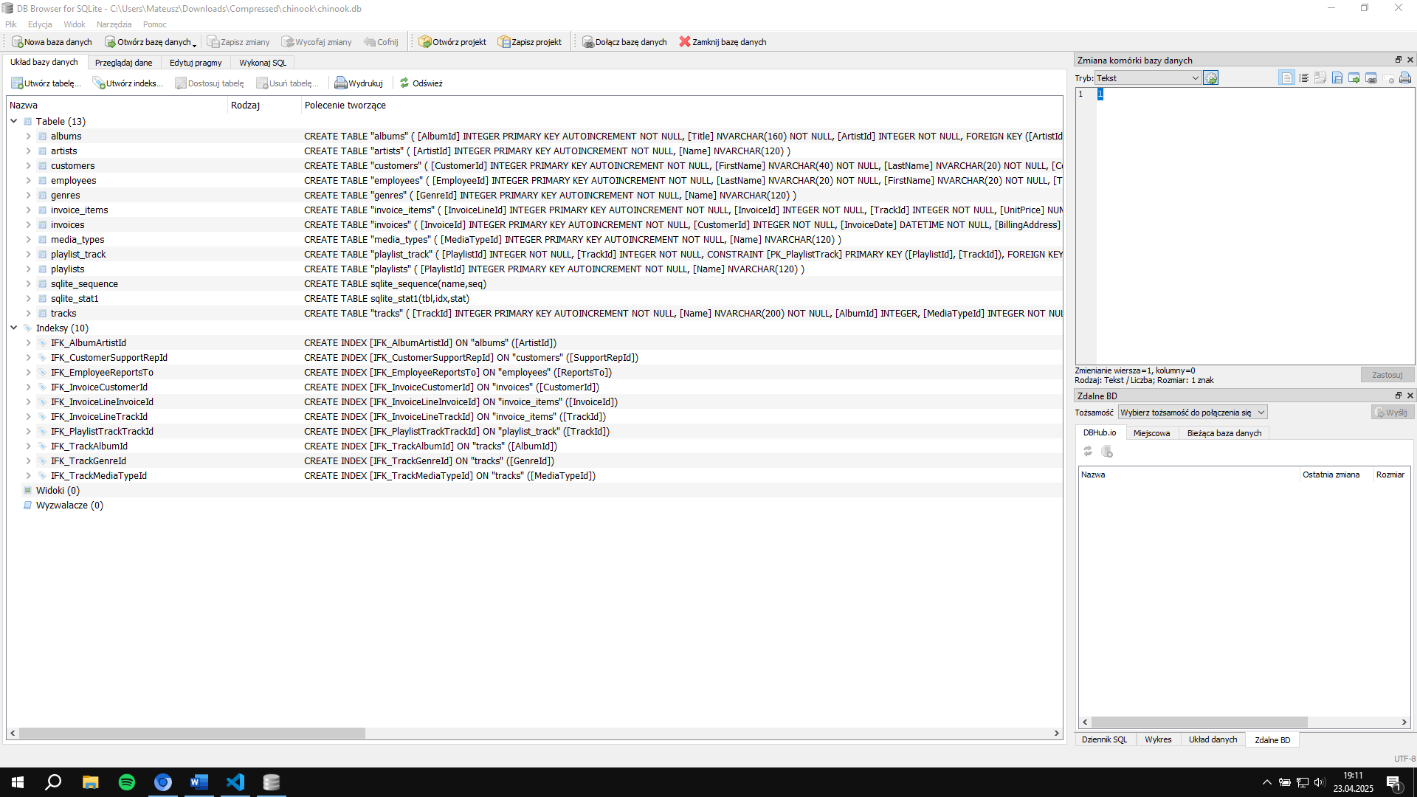
Django oferuje szeroki wybór gotowych pakietów i bibliotek, które można łatwo zintegrować z aplikacją, takich jak Django REST Framework (do budowy API), Celery (do obsługi asynchronicznych zadań) czy Django Channels (do obsługi WebSockets i aplikacji czasu rzeczywistego).

Podsumowując, wybór Django jako technologii backendowej dla aplikacji webowej do zarządzania dokumentami jest uzasadniony ze względu na jego wysoką wydajność, elastyczność, bezpieczeństwo oraz szerokie możliwości skalowania. Dzięki Django możliwe jest szybkie i efektywne wdrożenie aplikacji, zapewniające jednocześnie stabilność i bezpieczeństwo przechowywanych dokumentów.

### Baza danych – SQLite

SQLite został wybrany jako system zarządzania bazą danych (DBMS) dla aplikacji webowej do zarządzania dokumentami ze względu na jego prostotę, łatwość implementacji i niewielkie wymagania zasobowe. Na rysunku 7 przedstawiono strukturę bazy danych utworzonej w SQLite.

Rysunek 7. SQLite - przykładowa baza danych



Źródło: opracowanie własne

Poniżej przedstawiono kluczowe zalety tego rozwiązania oraz uzasadnienie jego wyboru.

Zalety SQLite jako bazy danych dla aplikacji DMS:

Lekkość i brak potrzeby konfiguracji

SQLite jest wbudowaną bazą danych, co oznacza, że nie wymaga konfiguracji oddzielnego serwera DBMS, co upraszcza proces wdrażania aplikacji. Brak konieczności administrowania zewnętrznym serwerem bazy danych minimalizuje wymagania sprzętowe i operacyjne.

Prostota i łatwa integracja z Django

Django posiada wbudowaną obsługę SQLite poprzez ORM (Object-Relational Mapping), co pozwala na szybkie projektowanie modeli danych bez potrzeby pisania skomplikowanych zapytań SQL. Uproszczone zarządzanie migracjami bazy danych ułatwia modyfikację schematu i wdrażanie aktualizacji.

Wydajność dla małych i średnich aplikacji

SQLite zapewnia bardzo dobrą wydajność dla aplikacji o umiarkowanym obciążeniu, co sprawia, że jest idealnym wyborem dla systemu zarządzania dokumentami w małych i średnich organizacjach. Brak mechanizmu wielowątkowego przetwarzania zapytań nie stanowi problemu, ponieważ aplikacja będzie obsługiwana głównie w trybie odczytu/zapisu przez ograniczoną liczbę użytkowników jednocześnie.

Obsługa transakcji i integralności danych

SQLite wspiera transakcje ACID (Atomicity, Consistency, Isolation, Durability), co gwarantuje integralność danych i odporność na awarie. Mechanizm blokowania plików i izolacji transakcji zapobiega uszkodzeniu bazy danych w przypadku awarii aplikacji.

Przechowywanie plików w bazie

SQLite obsługuje kolumny typu BLOB (Binary Large Object), co umożliwia przechowywanie dokumentów bezpośrednio w bazie danych.

Bezpieczeństwo danych

SQLite umożliwia szyfrowanie bazy danych za pomocą rozszerzenia SQLite Encryption Extension (SEE) lub implementacji takich jak SQLCipher. Mechanizmy kontroli dostępu Django oraz zabezpieczenia na poziomie aplikacji pozwalają na ochronę dokumentów i metadanych przed nieautoryzowanym dostępem.

### Frontend – HTML, CSS, Bootstrap

Frontend aplikacji webowej do zarządzania dokumentami został zaprojektowany z myślą o prostocie, przejrzystości i responsywności. W tym celu wykorzystano technologie takie jak HTML, CSS i Bootstrap, które zapewniają szybkie i efektywne tworzenie interfejsu użytkownika. Poniżej omówiono powody wyboru tych technologii oraz ich zastosowanie w aplikacji.

HTML(HyperTextMarkupLanguage) – podstawowy język strukturalny aplikacji webowych, który definiuje układ i treść strony.

Zalety HTML w projekcie:

* uniwersalność i kompatybilność – działa we wszystkich przeglądarkach internetowych, co gwarantuje szeroką dostępność aplikacji,
* łatwość integracji – umożliwia współpracę z CSS i frameworkami frontendowymi,
* przejrzysta struktura – kod HTML pozwala logicznie zorganizować elementy interfejsu użytkownika (np. nagłówki, formularze, tabele, przyciski).

W aplikacji HTML odpowiada za:

* strukturę panelu użytkownika,
* formularze logowania, rejestracji i przesyłania dokumentów,
* tabele z listami dokumentów oraz szczegółami dotyczącymi plików,
* nawigację i organizację elementów interfejsu.

CSS(CascadingStyleSheets) – kaskadowe arkusze stylów pozwalają na stylizację elementów HTML, dostosowanie wyglądu aplikacji oraz zapewnienie estetyki interfejsu.

Zalety CSS w projekcie:

* personalizacja wyglądu – pozwala na precyzyjne dostosowanie kolorystyki, marginesów, fontów i animacji,
* lepsze doświadczenie użytkownika (UX) – wpływa na czytelność interfejsu i poprawia komfort użytkowania,
* separacja stylów od treści – kod HTML pozostaje przejrzysty, a zmiany w wyglądzie można łatwo wprowadzać w plikach CSS.

W aplikacji CSS odpowiada za:

* stylizację formularzy, tabel i przycisków,
* dostosowanie nawigacji i wyglądu interfejsu,
* animacje i efekty wizualne poprawiające UX.

Bootstrap – popularny framework CSS, który przyspiesza proces projektowania interfejsu poprzez użycie gotowych komponentów, siatek responsywnych oraz dostosowanie wyglądu aplikacji do różnych rozdzielczości ekranów.

Zalety Bootstrap w projekcie:

* responsywność – aplikacja automatycznie dostosowuje się do różnych ekranów (komputery, tablety, smartfony),
* gotowe komponenty UI – Bootstrap oferuje przyciski, tabele, formularze, paski nawigacyjne i modale, co znacząco skraca czas implementacji,
* spójność designu – jednolity wygląd aplikacji zapewnia profesjonalny i nowoczesny interfejs,
* optymalizacja wydajności – framework jest zoptymalizowany pod kątem szybkiego ładowania stron.

W aplikacji Bootstrap został użyty do:

* tworzenia responsywnego interfejsu użytkownika,
* implementacji nawigacji i pasków menu,
* stylizacji przycisków, tabel i formularzy,
* zapewnienia jednolitego wyglądu aplikacji.

### Hosting i chmura – AWS EC2

Wybór odpowiedniej platformy hostingowej jest kluczowym elementem wdrożenia aplikacji webowej, szczególnie w przypadku systemów zarządzania dokumentami, które wymagają stabilności, skalowalności oraz bezpieczeństwa przechowywanych danych. W ramach pracy dyplomowej zdecydowano się na wykorzystanie platformy Amazon Web Services (AWS) z usługą Elastic Compute Cloud (EC2), która oferuje elastyczne środowisko do hostowania aplikacji Django. Na rysunku 9 przedstawiono interfejs AWS Management Console, który umożliwia zarządzanie usługami dostępnymi na platformie AWS.

Rysunek 9. Interfejs portalu AWS EC2.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

Amazon Web Services to wiodąca platforma chmurowa, oferująca szeroki zakres usług obliczeniowych i infrastrukturalnych. AWS EC2 (Elastic Compute Cloud) umożliwia uruchamianie wirtualnych serwerów w chmurze z pełną kontrolą nad systemem operacyjnym. Wybór tej technologii był podyktowany następującymi czynnikami:

Łatwość wdrożenia aplikacji Django

AWS EC2 z systemem Ubuntu Server zapewnia natywne wsparcie dla Pythona oraz Django bez dodatkowych konfiguracji. Możliwość wykorzystania Git do klonowania repozytorium bezpośrednio na serwer oraz prostota instalacji zależności przez pip znacznie ułatwia proces deployment. System zarządzania pakietami apt umożliwia szybką instalację narzędzi produkcyjnych.

Elastyczność i kontrola nad środowiskiem

EC2 oferuje pełną kontrolę nad instancją wirtualną, umożliwiając precyzyjne dostosowanie środowiska Python i konfigurację systemu operacyjnego. Możliwość wyboru różnych typów instancji (od t2.micro w ramach Free Tier do wydajnych instancji obliczeniowych) zapewnia optymalne dopasowanie zasobów do potrzeb aplikacji.

Bezpieczeństwo i kontrola dostępu

AWS implementuje wielopoziomowy model bezpieczeństwa z fizycznym zabezpieczeniem centrów danych, szyfrowaniem komunikacji oraz zaawansowanymi mechanizmami kontroli dostępu. Security Groups działają jako wirtualne firewalle, umożliwiając precyzyjne definiowanie reguł ruchu sieciowego. AWS posiada certyfikacje SOC, ISO 27001, GDPR oraz HIPAA, zapewniając zgodność z regulacjami.

Skalowalność i wydajność

Auto Scaling Groups umożliwiają automatyczne skalowanie zasobów w zależności od obciążenia aplikacji. Elastic Load Balancer zapewnia równomierną dystrybucję ruchu między instancjami, zwiększając dostępność systemu. Amazon CloudWatch dostarcza narzędzia do monitorowania wydajności oraz analizy logów aplikacji.

Integracja z usługami AWS

Platforma oferuje możliwość integracji z komplementarnymi usługami: Amazon RDS dla zarządzanych baz danych, Amazon S3 dla przechowywania plików, CloudFront dla CDN oraz SES do wysyłania powiadomień email. Takie podejście umożliwia stopniową migrację z SQLite na bardziej zaawansowane rozwiązania w miarę rozwoju aplikacji.

Ekonomiczność rozwiązania

AWS Free Tier umożliwia bezpłatne testowanie przez 12 miesięcy, obejmując 750 godzin pracy instancji t2.micro miesięcznie. Model płatności „pay-as-you-use” oznacza, że koszty są proporcjonalne do rzeczywistego wykorzystania zasobów, co jest korzystne dla projektów w fazie rozwoju.

## Środowisko programistyczne

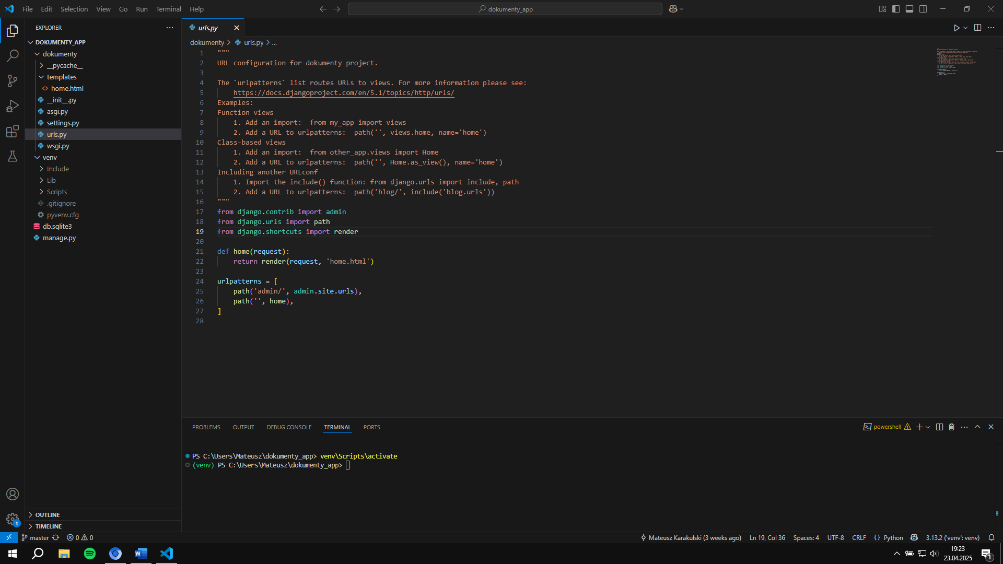
W celu efektywnego tworzenia i zarządzania aplikacją wykorzystano odpowiednie środowisko programistyczne, które obejmuje zarówno framework Django, jak i zestaw narzędzi wspierających proces wytwarzania oprogramowania. Prawidłowa konfiguracja środowiska ma kluczowe znaczenie dla zapewnienia płynności pracy, łatwości wdrażania oraz utrzymania kodu.

W kolejnych podrozdziałach opisana konfiguracja środowiska Django, używane IDE oraz narzędzia developerskie wspomagające codzienną pracę programisty. Dodatkowo przedstawiony został system kontroli wersji Git, który wraz z platformą GitHub umożliwia efektywne zarządzanie kodem źródłowym oraz współpracę w zespole.

### Visual Studio Code

W trakcie realizacji projektu wykorzystano środowisko programistyczne Visual Studio Code (VS Code) jako główne narzędzie do pisania i edytowania kodu aplikacji webowej. VS Code to lekkie, a zarazem bardzo funkcjonalne zintegrowane środowisko programistyczne (IDE), które dzięki swojej modularności oraz bogatemu ekosystemowi rozszerzeń idealnie sprawdza się w projektach webowych, opartych na języku Python i frameworku Django. Na rysunku 10 przedstawiono interfejs VS Code.

Rysunek 10. Interfejs VS Code



Źródło: opracowanie własne

Zalety wyboru Visual Studio Code:

* wsparcie dla Pythona i Django – dzięki rozszerzeniom takim jak Python i Django, możliwe jest podświetlanie składni, automatyczne uzupełnianie kodu (IntelliSense), debugging oraz zarządzanie środowiskami wirtualnymi,
* obsługa Git i GitHub – wbudowana integracja z systemem kontroli wersji Git umożliwia śledzenie zmian, tworzenie commitów, rozwiązywanie konfliktów i synchronizację z repozytoriami GitHub bez konieczności opuszczania edytora,
* terminal zintegrowany z IDE – pozwala na bezpośrednie uruchamianie poleceń Django (np. python manage.py runserver, makemigrations, migrate) bez konieczności korzystania z zewnętrznego terminala,
* rozszerzenia do HTML, CSS i Bootstrap – dzięki licznym wtyczkom (np. Live Server, Prettier, Bootstrap 5 Snippets) możliwe było szybkie i wygodne tworzenie oraz stylizowanie szablonów frontendowych,
* debugowanie i testowanie – VS Code umożliwia konfigurację breakpointów, śledzenie zmiennych oraz wykonywanie testów jednostkowych i integracyjnych w środowisku Django,
* zarządzanie środowiskiem wirtualnym – wsparcie dla venv i pip, a także współpraca z plikami konfiguracyjnymi requirements.txt, pozwalało na łatwą instalację i kontrolę nad zależnościami projektu.

Rozszerzenia użyte w projekcie aplikacji tworzonej w ramach pracy licencjackiej:

* Python (autor: Microsoft) – analiza kodu, IntelliSense, debugging,
* Django – wsparcie dla struktury aplikacji Django,
* GitLens – rozszerzone możliwości pracy z Git,
* Bootstrap 5 Snippets – autouzupełnianie komponentów Bootstrap,
* Prettier - Code formatter – formatowanie kodu HTML/CSS.

### Konfiguracja środowiska Django

Przygotowanie środowiska Django do pracy nad aplikacją webową do zarządzania dokumentami obejmuje szereg kroków, od instalacji frameworka po uruchomienie pierwszego serwera deweloperskiego. W tym podrozdziale przedstawiono proces konfiguracji środowiska programistycznego w systemie lokalnym z wykorzystaniem Visual Studio Code, Python i wirtualnego środowiska venv.

Pierwszym krokiem jest upewnienie się, że na komputerze zainstalowany jest interpreter języka Python w wersji co najmniej 3.8. Instalację można przeprowadzić ze strony oficjalnej https://www.python.org. Podczas instalacji należy zaznaczyć opcję „Add Python to PATH”, aby umożliwić korzystanie z poleceń python i pip w terminalu.

Na rysunku 11 przedstawiono proces konfiguracji środowiska programistycznego przy użyciu terminala w Visual Studio Code. Składa się on z następujących kroków

1. Utworzenie folderu projektu
   * mkdir dokumenty\_app
   * cd dokumenty\_app
2. Utworzenie środowiska wirtualnego
   * python -m venv venv
3. Zezwolenie na uruchamianie skryptów PowerShell (może pojawić się błąd ExecutionPolicy)
   * Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
4. Aktywacja środowiska wirtualnego
   * .\venv\Scripts\Activate.ps1
5. Instalacja frameworka Django
   * pip install Django
6. Sprawdzenie wersji Django
   * django-admin –version
7. Utworzenie projektu Django
   * django-admin startproject dokumenty .
8. Uruchomienie serwera deweloperskiego
   * python manage.py runserver

Po uruchomieniu serwera deweloperskiego, Django udostępnia domyślną stronę startową pod adresem http://127.0.0.1:8000/, co zostało pokazane na rysunku 11.

Rysunek 11. Strona domyślna aplikacji

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

### System kontroli wersji – Git i GitHub

W celu efektywnego zarządzania kodem źródłowym projektu, a także umożliwienia współpracy i śledzenia zmian, wykorzystywany jest system kontroli wersji Git. Wraz z nim wykorzystywana jest usługa GitHub, która umożliwia zdalne przechowywanie repozytoriów oraz ich współdzielenie. Na rysunku 12 przedstawiono interfejs GitHub.

Rysunek . Interfejs GitHub

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

Instalacja Git

Należy przejść na oficjalną stronę projektu Git: https://git-scm.com/. Pobrać odpowiednią wersję instalatora dla używanego systemu operacyjnego. Podczas instalacji można pozostawić domyślne ustawienia – ważne, aby upewnić się, że opcja „Git from the command line and also from 3rd-party software” jest zaznaczona. Po zakończeniu instalacji, aby sprawdzić poprawność, należy otworzyć terminal (np. PowerShell lub Git Bash) i wpisać polecenie:

* git –version

Jeśli pojawi się komunikat o błędzie informujący, że polecenie git nie jest rozpoznawane, należy dodać Git do zmiennej środowiskowej PATH:

* C:\Program Files\Git\cmd

Podstawowa konfiguracja Git

Po zainstalowaniu Git należy skonfigurować dane użytkownika, które będą przypisane do commitów:

* git config --global user.name „Imię i nazwisko”
* git config --global user.email „adres@email.com”

Założenie konta na GitHub

Wejść na stronę https://github.com/. Utworzyć konto użytkownika lub zalogować się, jeśli konto już istnieje. Kliknąć New repository, aby utworzyć nowe repozytorium dla projektu. Nadać nazwę repozytorium i zdecydować, czy ma być publiczne, czy prywatne. Kliknąć Create repository.

Połączenie lokalnego repozytorium z GitHubem

W katalogu projektu wykonać następujące polecenia:

1. Inicjalizacja repozytorium

* git init

1. Dodanie wszystkich plików do repozytorium

* git add .

1. Utworzenie pierwszego commita:

* git commit -m „Initial commit”

1. Dodanie zdalnego repozytorium

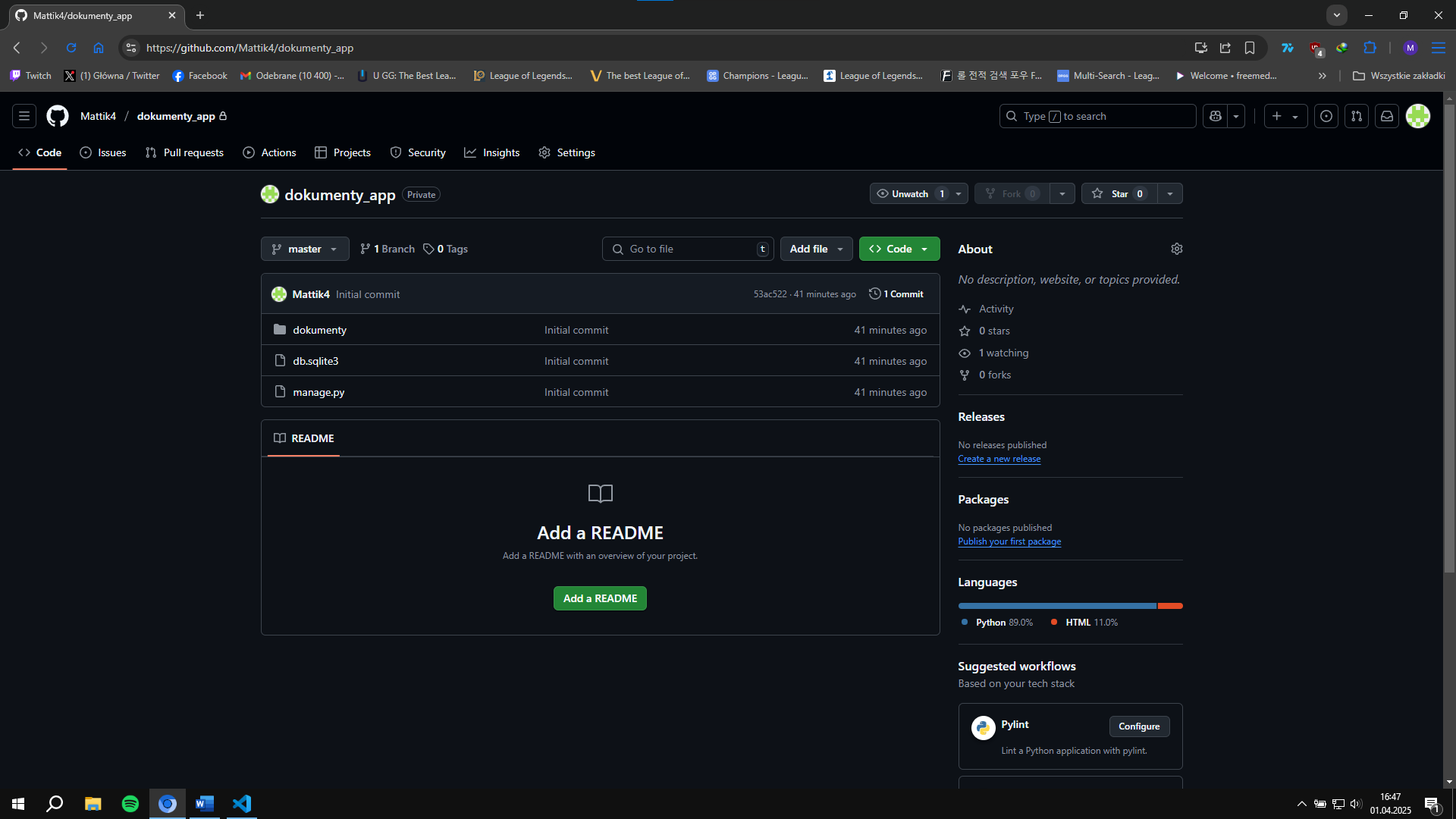
* git remote add origin (link do repozytorium)

1. Wysłanie kodu do GitHuba:

* git push -u origin master

Na rysunku 13 przedstawiono widok repozytorium na platformie GitHub po wykonaniu pierwszego commita.

Rysunek . Podłączone repoztytorium Git



Źródło: opracowanie własne

# Projektowanie systemu

Proces projektowania systemu zarządzania dokumentami stanowi kluczowy etap realizacji niniejszej pracy dyplomowej. To właśnie na tym etapie następuje przełożenie założeń funkcjonalnych oraz technologicznych na konkretną strukturę logiczną, architektoniczną i organizacyjną systemu. Celem projektowania jest stworzenie spójnego, skalowalnego i bezpiecznego środowiska pracy, które w intuicyjny sposób umożliwi użytkownikom przechowywanie, udostępnianie i zarządzanie dokumentami w ramach ustalonych ról i uprawnień.

## Analiza wymagań

Analiza wymagań jest fundamentalnym etapem projektowania każdego systemu informatycznego. Pozwala na zdefiniowanie oczekiwań użytkowników końcowych oraz sprecyzowanie funkcjonalności, jakie system powinien oferować. W przypadku aplikacji webowej do zarządzania dokumentami, analiza wymagań miała na celu określenie zakresu funkcji oraz ustalenie cech jakościowych systemu, takich jak bezpieczeństwo, dostępność czy wydajność. Na podstawie zebranych informacji wyróżniono dwa główne typy wymagań: funkcjonalne oraz niefunkcjonalne.

### Wymagania funkcjonalne

Wymagania funkcjonalne opisują konkretne funkcje, które system musi realizować, aby spełnić założone cele. W przypadku aplikacji webowej do zarządzania dokumentami, kluczowe wymagania funkcjonalne obejmują:

1. System użytkowników i ról:
   * możliwość rejestracji nowych użytkowników,
   * logowanie i wylogowawanie użytkowników,
   * zarządzanie kontami (zmiana hasła, resetowanie hasła),
   * przypisywanie ról: administrator, edytor, czytelnik.
2. Zarządzanie dokumentami:
   * dodawanie (upload) dokumentów w różnych formatach (PDF, DOCX, XLS),
   * pobieranie dokumentów,
   * wyszukiwanie dokumentów po nazwie, tagach, metadanych,
   * wersjonowanie plików – możliwość przechowywania i przywracania wcześniejszych wersji dokumentów,
   * przeglądanie historii zmian dokumentu.
3. Uprawnienia i współpraca:
   * nadawanie i odbieranie dostępu do dokumentów,
   * udostępnianie dokumentów użytkownikom z określonymi uprawnieniami (tylko do odczytu, możliwość edycji),
   * komentowanie i współtworzenie dokumentów w czasie rzeczywistym.
4. Zarządzanie danymi:
   * dodawanie metadanych i tagów do dokumentów,
5. Bezpieczeństwo:
   * szyfrowanie przechowywanych plików,
   * audyt logów użytkownika – rejestrowanie aktywności (logi dostępu, edycji),
   * obsługa dwuetapowej autoryzacji (2FA).

### Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają cechy jakościowe systemu, wpływające na jego sposób działania. W przypadku aplikacji webowej do zarządzania dokumentami, kluczowe wymagania niefunkcjonalne obejmują:

1. Dostępność:
   * aplikacja musi być dostępna 24/7 z dowolnej przeglądarki internetowej i urządzenia.
2. Skalowalność:
   * system powinien umożliwiać skalowanie w poziomie (dodanie nowych użytkowników, obsługa większej liczby dokumentów bez spadku wydajności).
3. Responsywność:
   * interfejs użytkownika powinien być zoptymalizowany pod kątem różnych rozdzielczości ekranów (komputery, tablety, smartfony).
4. Wydajność:
   * czas odpowiedzi systemu na zapytania (np. wyszukiwanie dokumentów) nie powinien przekraczać 2 sekund.
5. Bezpieczeństwo:
   * system powinien być odporny na typowe zagrożenia (np. SQL Injection, XSS, CSRF),
   * przechowywane dane muszą być szyfrowane zarówno w trakcie przesyłania (TLS), jak i w stanie spoczynku.
6. Zgodność z przepisami:
   * system powinien wspierać zgodność z przepisami o ochronie danych osobowych (np. RODO).
7. Łatwość obsługi i utrzymania:
   * aplikacja powinna mieć intuicyjny interfejs oraz dokumentację użytkownika i administratora,
   * system powinien być łatwy do wdrożenia i konfiguracji bez konieczności zaawansowanej wiedzy IT.

## Użytkownicy aplikacji

Efektywne funkcjonowanie systemu zarządzania dokumentami w znacznej mierze zależy od precyzyjnego zdefiniowania typów użytkowników oraz ich uprawnień w aplikacji. Na podstawie przeprowadzonej analizy wymagań funkcjonalnych i niefunkcjonalnych, przedstawionej w poprzednim podrozdziale, konieczne stało się szczegółowe określenie ról użytkowników, ich odpowiedzialności oraz zakresu dostępnych im funkcjonalności w systemie.

### Opis użytkowników systemu - role i uprawnienia

System zarządzania dokumentami przewiduje obsługę różnych ról użytkowników, z których każda posiada przypisany zestaw uprawnień. Zastosowanie podziału ról umożliwia lepsze zarządzanie dostępem do zasobów, zwiększa bezpieczeństwo danych i pozwala na precyzyjne określenie odpowiedzialności poszczególnych użytkowników w ramach systemu. Poniżej przedstawiono trzy główne role użytkowników:

1. Administrator - użytkownik posiadający pełne uprawnienia do zarządzania systemem.

Zakres uprawnień:

* + dodawanie, edycja i usuwanie kont użytkowników,
  + nadawanie i odbieranie uprawnień,
  + dostęp do wszystkich dokumentów oraz możliwość ich edycji i usuwania,
  + wgląd w logi aktywności użytkowników (audyt),
  + zarządzanie strukturą folderów i kategoriami dokumentów,
  + inicjalizacja i przywracanie kopii zapasowych.

1. Edytor - użytkownik mający uprawnienia do aktywnej pracy z dokumentami.

Zakres uprawnień:

* + przesyłanie i edytowanie dokumentów,
  + tworzenie wersji dokumentów oraz dodawanie komentarzy,
  + odczyt dokumentów udostępnionych przez innych użytkowników,
  + udostępnianie własnych dokumentów innym użytkownikom z rolą „czytelnik” lub „edytor” (w ramach nadanych przez administratora uprawnień),
  + tagowanie i kategoryzowanie dokumentów.

1. Czytelnik - użytkownik o podstawowym poziomie dostępu, przeznaczony do przeglądania dokumentów.

Zakres uprawnień:

* + przeglądanie dokumentów, do których uzyskał dostęp,
  + pobieranie dokumentów,
  + przeglądanie wersji dokumentów,
  + komentowanie dokumentów (przy posiadaniu odpowiednich uprawnień).

### Diagram przypadków użycia - UC – Use Case

Diagram przypadków użycia (ang. Use Case Diagram) to graficzna reprezentacja funkcjonalności systemu z punktu widzenia użytkowników końcowych. Pozwala on zidentyfikować interakcje pomiędzy aktorami (rolami użytkowników) a systemem oraz zilustrować, jakie operacje są dostępne dla poszczególnych ról. W kontekście aplikacji webowej do zarządzania dokumentami opracowano diagram z trzema głównymi aktorami: Administrator, Edytor i Czytelnik.

Rysunek 14 przedstawia funkcjonalności dostępne dla użytkownika pełniącego rolę administratora systemu DMS. Administrator ma pełen zakres uprawnień, obejmujący zarządzanie użytkownikami (dodawanie, edycja, usuwanie), przypisywanie i odbieranie ról, konfigurację parametrów systemu (backup, wersjonowanie), wgląd do logów oraz zarządzanie strukturą folderów.

Rysunek 14. Diagram przypadków użycia – Administrator

Obraz zawierający zrzut ekranu, tekst, diagram, linia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

Rysunek 15 przedstawia działania możliwe do wykonania przez użytkownika w roli edytora. Edytor może logować się do systemu, tworzyć i edytować dokumenty, przypisywać im tagi i kategorie, zarządzać wersjami plików oraz udostępniać je innym użytkownikom. Może również komentować i współtworzyć dokumenty, a także zarządzać uprawnieniami do plików, do których ma dostęp.

Rysunek 15. Diagram przypadków użycia – Edytor

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

Rysunek 16 obrazuje zakres funkcji przypisanych użytkownikowi o roli czytelnika. Czytelnik ma możliwość logowania się do systemu, przeglądania oraz pobierania dokumentów, a także przeglądania ich historii wersji. W przypadku nadania odpowiednich uprawnień, może również komentować dokumenty oraz przeglądać komentarze pozostawione przez innych użytkowników.

Rysunek 16. Diagram przypadków użycia – Czytelnik

Obraz zawierający zrzut ekranu, tekst, linia, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

# Projektowanie bazy danych

Projektowanie bazy danych stanowi kluczowy etap implementacji aplikacji webowej do zarządzania dokumentami, determinujący sposób organizacji, przechowywania oraz przetwarzania danych w systemie. Na podstawie wymagań funkcjonalnych i niefunkcjonalnych zdefiniowanych w rozdziale trzecim, konieczne jest stworzenie efektywnej struktury danych, która zapewni optymalne przechowywanie dokumentów, zarządzanie użytkownikami, kontrolę wersji oraz implementację złożonego systemu uprawnień.

## Modelowanie danych

Modelowanie danych stanowi fundamentalny etap projektowania aplikacji webowej do zarządzania dokumentami, determinujący sposób organizacji, przechowywania oraz przetwarzania danych w systemie. Proces modelowania obejmuje systematyczne przejście od abstrakcyjnych wymagań biznesowych do konkretnej implementacji struktury bazy danych, zapewniając spójność, integralność oraz wydajność przechowywanych danych.

### Konceptualny model danych (CDM)

Konceptualny model danych (CDM) stanowi wysokopoziomową reprezentację struktury danych w systemie, niezależną od szczegółów implementacyjnych i technicznych ograniczeń. W przypadku aplikacji webowej do zarządzania dokumentami, model ten koncentruje się na głównych encjach, ich atrybutach oraz relacjach między nimi, tworząc logiczny obraz funkcjonowania systemu z perspektywy biznesowej. Encje systemu wraz z ich atrybutami przedstawiono w tabeli 2. Związki między poszczególnymi encjami wraz z ich charakterystyką zestawiono w tabeli 3. Rysunek 17 przedstawia diagram konceptualnego modelu danych dla systemu zarządzania dokumentami.

Tabela . Encje

| **Lp.** | **Encja** | **Atrybuty** |
| --- | --- | --- |
| 1 | UŻYTKOWNIK | id, imię, nazwisko, email, hasło, data utworzenia, ostatnie logowanie |
| 2 | ROLA | id, nazwa (Administrator, Edytor, Czytelnik), opis |
| 3 | DOKUMENT | id, nazwa, typ pliku, rozmiar, data utworzenia, zawartość |
| 4 | FOLDER | id, nazwa, opis, data utworzenia |
| 5 | WERSJA DOKUMENTU | id, numer wersji, data utworzenia, opis zmian, zawartość |
| 6 | TAG | id, nazwa, kolor |
| 7 | METADANE | id, klucz, wartość |
| 8 | UPRAWNIENIE | id, typ (odczyt, zapis, usuwanie, administrowanie) |
| 9 | KOMENTARZ | id, treść, data utworzenia |
| 10 | LOG AKTYWNOŚCI | id, typ operacji, data, szczegóły |

Źródło: opracowanie własne

Tabela 3. Związki między encjami

| **Lp.** | **Encja 1** | **Encja 2** | **Typ związku** | **Opis** |
| --- | --- | --- | --- | --- |
| 1 | UŻYTKOWNIK | ROLA | jeden-do-jeden | każdy użytkownik ma przypisaną dokładnie jedną rolę; jedna rola może być przypisana do wielu użytkowników |
| 2 | UPRAWNIENIE | DOKUMENT/FOLDER | wiele-do-wielu | uprawnienie może dotyczyć wielu dokumentów/folderów; dokument/folder może mieć wiele typów uprawnień |
| 3 | UŻYTKOWNIK | DOKUMENT | jeden-do-wielu | użytkownik może utworzyć wiele dokumentów; każdy dokument jest utworzony przez jednego użytkownika (tworzenie) |
| 4 | UŻYTKOWNIK | KOMENTARZ | jeden-do-wielu | użytkownik może pozostawić wiele komentarzy; każdy komentarz jest utworzony przez jednego użytkownika |
| 5 | UŻYTKOWNIK | WERSJA DOKUMENTU | jeden-do-wielu | użytkownik może utworzyć wiele wersji dokumentów; każda wersja jest utworzona przez jednego użytkownika |
| 6 | UŻYTKOWNIK | LOG AKTYWNOŚCI | jeden-do-wielu | użytkownik może wygenerować wiele wpisów w logu aktywności; każdy wpis w logu jest powiązany z jednym użytkownikiem |
| 7 | DOKUMENT | FOLDER | wiele-do-jeden | dokument należy do jednego folderu; folder może zawierać wiele dokumentów |
| 8 | FOLDER | FOLDER | rekurencyjna jeden-do-wielu | folder może zawierać wiele podfolderów; każdy podfolder ma jeden folder nadrzędny |
| 9 | DOKUMENT | WERSJA DOKUMENTU | jeden-do-wielu | dokument może mieć wiele wersji; każda wersja należy do jednego dokumentu |
| 10 | DOKUMENT | METADANE | jeden-do-wielu | dokument może mieć wiele metadanych; każda metadana należy do jednego dokumentu |
| 11 | DOKUMENT | KOMENTARZ | jeden-do-wielu | dokument może mieć wiele komentarzy; każdy komentarz dotyczy jednego dokumentu |
| 12 | DOKUMENT | TAG | wiele-do-wielu | dokument może mieć wiele tagów; tag może być przypisany do wielu dokumentów |
| 13 | LOG AKTYWNOŚCI | DOKUMENT | wiele-do-wielu | log aktywności może dotyczyć wielu dokumentów; dokument może być powiązany z wieloma wpisami w logu aktywności |

Źródło: opracowanie własne

Rysunek 17. Diagram CDM

Obraz zawierający diagram, design, wzór

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Logiczny model danych (LDM)

Logiczny model danych (LDM) stanowi pośredni etap pomiędzy konceptualnym modelem danych a fizyczną implementacją w bazie danych. W przeciwieństwie do modelu konceptualnego, model logiczny uwzględnia już ograniczenia wynikające z konkretnego podejścia do modelowania danych (w tym przypadku relacyjnego), jednak pozostaje niezależny od konkretnego systemu zarządzania bazą danych. Rysunek 18 przedstawia logiczny model danych dla systemu zarządzania dokumentami.

#### Wykaz encji logicznych

Wykaz encji logicznych stanowi fundamentalną część logicznego modelu danych, definiując wszystkie główne obiekty biznesowe występujące w systemie zarządzania dokumentami. Każda encja reprezentuje określoną kategorię informacji, która musi być przechowywana i zarządzana przez aplikację, wraz z krótkim opisem jej roli w systemie. Tabela 4 przedstawia kompletny wykaz encji logicznych systemu zarządzania dokumentami.

Tabela 4. Wykaz encji logicznych

| **Lp.** | **Nazwa encji** | **Opis** |
| --- | --- | --- |
| 1 | UŻYTKOWNIK | Encja reprezentująca użytkowników systemu |
| 2 | ROLA | Encja definiująca role i uprawnienia użytkowników |
| 3 | DOKUMENT | Encja reprezentująca dokumenty w systemie |
| 4 | WERSJA\_DOKUMENT | Encja przechowująca różne wersje dokumentów |
| 5 | FOLDER | Encja reprezentująca foldery do organizacji dokumentów |
| 6 | TAG | Encja definiująca tagi do kategoryzacji dokumentów |
| 7 | DOKUMENT\_TAG | Tabela łącznikowa między dokumentami a tagami |
| 8 | METADANE | Encja przechowująca dodatkowe metadane dokumentów |
| 9 | UPRAWNIENIE\_DOKUMENT | Encja definiująca uprawnienia użytkowników do dokumentów |
| 10 | UPRAWNIENIE\_FOLDER | Encja definiująca uprawnienia użytkowników do folderów |
| 11 | KOMENTARZ | Encja reprezentująca komentarze do dokumentów |
| 12 | LOG\_AKTYWNOŚCI | Encja przechowująca historię aktywności użytkowników |

Źródło: opracowanie własne

#### Wykaz atrybutów

Szczegółowy wykaz atrybutów dla każdej encji określa strukturę danych na poziomie logicznym, definiując nazwy pól, typy danych oraz ich przeznaczenie w systemie. Atrybuty zostały zaprojektowane z uwzględnieniem wymagań funkcjonalnych aplikacji oraz ograniczeń integralności danych, zapewniając kompleksowe przechowywanie wszystkich niezbędnych informacji. Poniższe tabele przedstawia szczegółowy wykaz atrybutów wszystkich encji logicznych z określeniem typów danych i opisem funkcji każdego pola.

Tabela 5. Wykaz atrybutów encji Użytkownik

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator użytkownika | NOT NULL |
| **nazwa\_użytkownika** | VARCHAR(50) | nazwa użytkownika | NOT NULL |
| **imię** | VARCHAR(50) | imię | NOT NULL |
| **nazwisko** | VARCHAR(50) | nazwisko | NOT NULL |
| **email** | VARCHAR(100) | adres email | NOT NULL |
| **hasło\_hash** | VARCHAR(255) | zahaszowane hasło | NOT NULL |
| **data\_utworzenia** | DATETIME | data utworzenia konta | NOT NULL |
| **ostatnie\_logowanie** | DATETIME | data ostatniego logowania | NULL |
| **aktywny** | BOOLEAN | status aktywności konta | NOT NULL |
| **id\_roli** | INTEGER, FK | klucz obcy wskazujący na rolę użytkownika | NOT NULL |

Źródło: opracowanie własne

Tabela 6. Wykaz atrybutów encji Rola

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator roli | NOT NULL |
| **nazwa** | VARCHAR(50) | nazwa roli (Administrator, Edytor, Czytelnik) | NOT NULL |
| **opis** | TEXT | opis uprawnień roli | NULL |

Źródło: opracowanie własne

Tabela 7. Wykaz atrybutów encji Dokument

| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| --- | --- | --- | --- |
| **id** | INTEGER, PK | unikalny identyfikator dokumentu | NOT NULL |
| **nazwa** | VARCHAR(255) | nazwa dokumentu | NOT NULL |
| **typ\_pliku** | VARCHAR(100) | typ pliku (np. PDF, DOCX) | NOT NULL |
| **rozmiar\_pliku** | INTEGER | rozmiar pliku w bajtach | NOT NULL |
| **data\_utworzenia** | DATETIME | data utworzenia dokumentu | NOT NULL |
| **ostatnia\_modyfikacja** | DATETIME | data ostatniej modyfikacji | NOT NULL |
| **id\_właściciela** | INTEGER, FK | klucz obcy wskazujący na właściciela dokumentu | NOT NULL |
| **id\_folderu** | INTEGER, FK | klucz obcy wskazujący na folder, w którym znajduje się dokument | NULL |
| **usunięty** | BOOLEAN | flaga oznaczająca, czy dokument został usunięty | NOT NULL |

Źródło: opracowanie własne

Tabela 8. Wykaz atrybutów encji Wersja\_dokument

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator wersji dokumentu | NOT NULL |
| **id\_dokumentu** | INTEGER, FK | klucz obcy wskazujący na dokument | NOT NULL |
| **numer\_wersji** | INTEGER | numer wersji dokumentu | NOT NULL |
| **data\_utworzenia** | DATETIME | data utworzenia wersji | NOT NULL |
| **utworzony\_przez** | INTEGER, FK | klucz obcy wskazujący na użytkownika, który utworzył wersję | NOT NULL |
| **zawartość** | BLOB | zawartość binarna dokumentu | NULL |
| **komentarz** | TEXT | komentarz opisujący zmiany w wersji | NULL |

Źródło: opracowanie własne

Tabela 9. Wykaz atrybutów encji Folder

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator folderu | NOT NULL |
| **nazwa** | VARCHAR(255) | nazwa folderu | NOT NULL |
| **opis** | TEXT | opis folderu | NULL |
| **data\_utworzenia** | DATETIME | data utworzenia folderu | NOT NULL |
| **id\_rodzica** | INTEGER, FK | klucz obcy wskazujący na folder nadrzędny (NULL dla folderu głównego) | NULL |
| **id\_właściciela** | INTEGER, FK | klucz obcy wskazujący na właściciela folderu | NOT NULL |

Źródło: opracowanie własne

Tabela 10. Wykaz atrybutów encji Tag

| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| --- | --- | --- | --- |
| **id** | INTEGER, PK | unikalny identyfikator tagu | NOT NULL |
| **nazwa** | VARCHAR(50) | nazwa tagu | NOT NULL |
| **kolor** | VARCHAR(20) | kolor oznaczenia tagu | NULL |

Źródło: opracowanie własne

Tabela 11. Wykaz atrybutów encji Dokument\_tag

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id\_dokumentu** | INTEGER, FK | klucz obcy wskazujący na dokument | NOT NULL |
| **id\_tagu** | INTEGER, FK | klucz obcy wskazujący na tag | NOT NULL |
| **PRIMARY KEY** | (id\_dokumentu, id\_tagu) | klucz główny złożony | NOT NULL |

Źródło: opracowanie własne

Tabela 12. Wykaz atrybutów encji metadane

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator metadanej | NOT NULL |
| **id\_dokumentu** | INTEGER, FK | klucz obcy wskazujący na dokument | NOT NULL |
| **klucz** | VARCHAR(100) | klucz metadanej | NOT NULL |
| **wartość** | TEXT | wartość metadanej | NULL |

Źródło: opracowanie własne

Tabela 13. Wykład atrybutów ecji Uprawnienie\_dokument

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator uprawnienia | NOT NULL |
| **id\_dokumentu** | INTEGER, FK | klucz obcy wskazujący na dokument | NOT NULL |
| **id\_użytkownika** | INTEGER, FK | klucz obcy wskazujący na użytkownika | NOT NULL |
| **typ\_uprawnienia** | VARCHAR(20) | typ uprawnienia (odczyt, zapis, administracja) | NOT NULL |
| **data\_nadania** | DATETIME | data nadania uprawnienia | NOT NULL |
| **nadane\_przez** | INTEGER, FK | klucz obcy wskazujący na użytkownika, który nadał uprawnienie | NOT NULL |

Źródło: opracowanie własne

Tabela 14. Wykaz atrybutów encji Uprawnienie\_folder

| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| --- | --- | --- | --- |
| **id** | INTEGER, PK | unikalny identyfikator uprawnienia | NOT NULL |
| **id\_folderu** | INTEGER, FK | klucz obcy wskazujący na folder | NOT NULL |
| **id\_użytkownika** | INTEGER, FK | klucz obcy wskazujący na użytkownika | NOT NULL |
| **typ\_uprawnienia** | VARCHAR(20) | typ uprawnienia (odczyt, zapis, administracja) | NOT NULL |
| **data\_nadania** | DATETIME | data nadania uprawnienia | NOT NULL |
| **nadane\_przez** | INTEGER, FK | klucz obcy wskazujący na użytkownika, który nadał uprawnienie | NOT NULL |

Źródło: opracowanie własne

Tabela 15. Wykaz atrybutów encji Komentarz

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator komentarza | NOT NULL |
| **id\_dokumentu** | INTEGER, FK | klucz obcy wskazujący na dokument | NOT NULL |
| **id\_użytkownika** | INTEGER, FK | klucz obcy wskazujący na użytkownika, który dodał komentarz | NOT NULL |
| **treść** | TEXT | treść komentarza | NOT NULL |
| **data\_utworzenia** | DATETIME | data utworzenia komentarza | NOT NULL |
| **id\_rodzica** | INTEGER, FK | klucz obcy wskazujący na komentarz nadrzędny (dla odpowiedzi na komentarze) | NULL |

Źródło: opracowanie własne

Tabela 16. Wykaz atrybutów encji Log\_aktywności

|  |  |  |  |
| --- | --- | --- | --- |
| **Atrybut** | **Typ danych** | **Opis** | **Czy wymagany?** |
| **id** | INTEGER, PK | unikalny identyfikator wpisu w logu | NOT NULL |
| **id\_użytkownika** | INTEGER, FK | klucz obcy wskazujący na użytkownika, który wykonał akcję | NOT NULL |
| **typ\_aktywności** | VARCHAR(50) | typ akcji (logowanie, tworzenie, edycja, usunięcie itp.) | NOT NULL |
| **id\_dokumentu** | INTEGER, FK, NULL | opcjonalny klucz obcy wskazujący na dokument | NULL |
| **id\_folderu** | INTEGER, FK, NULL | opcjonalny klucz obcy wskazujący na folder | NULL |
| **szczegóły** | TEXT | szczegóły akcji | NULL |
| **znacznik\_czasu** | DATETIME | data i czas akcji | NOT NULL |
| **adres\_ip** | VARCHAR(50) | adres IP, z którego wykonano akcję | NULL |

Źródło: opracowanie własne

#### Wykaz związków

Związki między encjami definiują sposób, w jaki poszczególne obiekty biznesowe są ze sobą powiązane w systemie zarządzania dokumentami. Poprawne zdefiniowanie relacji jest kluczowe dla zapewnienia integralności referencyjnej oraz efektywnego modelowania procesów biznesowych, takich jak właścicielstwo dokumentów, hierarchia folderów czy system uprawnień. Tabela 17 przedstawia wszystkie związki między encjami logicznymi wraz z określeniem typu relacji i szczegółowym opisem każdego powiązania.

Tabela . Związki

| **Lp.** | **Encja nadrzędna** | **Encja podrzędna** | **Typ związku** | **Opcje związku** | **Opis** |
| --- | --- | --- | --- | --- | --- |
| 1 | ROLE | UŻYTKOWNICY | 1:N | Nieidentyfikujący, Obowiązkowy | Każdy użytkownik ma przypisaną dokładnie jedną rolę; jedna rola może być przypisana do wielu użytkowników |
| 2 | UŻYTKOWNICY | DOKUMENTY | 1:N | Nieidentyfikujący, Obowiązkowy | Użytkownik może być właścicielem wielu dokumentów; każdy dokument ma dokładnie jednego właściciela |
| 3 | UŻYTKOWNICY | FOLDERY | 1:N | Nieidentyfikujący, Obowiązkowy | Użytkownik może być właścicielem wielu folderów; każdy folder ma dokładnie jednego właściciela |
| 4 | UŻYTKOWNICY | KOMENTARZE | 1:N | Nieidentyfikujący, Obowiązkowy | Użytkownik może dodać wiele komentarzy; każdy komentarz jest dodany przez jednego użytkownika |
| 5 | UŻYTKOWNICY | WERSJE\_DOKUMENTÓW | 1:N | Nieidentyfikujący, Obowiązkowy | Użytkownik może utworzyć wiele wersji dokumentów; każda wersja jest utworzona przez jednego użytkownika |
| 6 | FOLDERY | DOKUMENTY | 1:N | Nieidentyfikujący, Opcjonalny (id\_folderu w DOKUMENTY może być NULL) | Dokument znajduje się w dokładnie jednym folderze; folder może zawierać wiele dokumentów |
| 7 | FOLDERY | FOLDERY (podfolder) | 1:N (rekurencyjny) | Nieidentyfikujący, Opcjonalny (id\_rodzica w FOLDERY może być NULL) | Folder może zawierać wiele podfolderów; każdy folder (z wyjątkiem głównego) ma dokładnie jeden folder nadrzędny |
| 8 | DOKUMENTY | WERSJE\_DOKUMENTÓW | 1:N | Identyfikujący, Obowiązkowy | Dokument może mieć wiele wersji; każda wersja należy do dokładnie jednego dokumentu |
| 9 | DOKUMENTY | TAGI | M:N | Nieidentyfikujący, Obowiązkowy | Dokument może mieć wiele tagów; tag może być przypisany do wielu dokumentów; relacja implementowana przez tabelę łączącą DOKUMENTY\_TAGI |
| 10 | DOKUMENTY | METADANE | 1:N | Identyfikujący, Obowiązkowy | Dokument może mieć wiele metadanych; każda metadana należy do jednego dokumentu |
| 11 | DOKUMENTY | KOMENTARZE | 1:N | Nieidentyfikujący, Obowiązkowy | Dokument może mieć wiele komentarzy; każdy komentarz dotyczy jednego dokumentu |
| 12 | DOKUMENTY | UPRAWNIENIA\_DOKUMENTÓW | 1:N | Identyfikujący, Obowiązkowy | Dokument może mieć wiele uprawnień; każde uprawnienie dotyczy jednego dokumentu i jednego użytkownika |
| 13 | FOLDERY | UPRAWNIENIA\_FOLDERÓW | 1:N | Identyfikujący, Obowiązkowy | Folder może mieć wiele uprawnień; każde uprawnienie dotyczy jednego folderu i jednego użytkownika |
| 14 | UŻYTKOWNICY | LOGI\_AKTYWNOŚCI | 1:N | Nieidentyfikujący, Obowiązkowy | Użytkownik może wygenerować wiele wpisów w logu aktywności; każdy wpis w logu jest powiązany z jednym użytkownikiem |

Źródło: opracowanie własne

#### Reguły biznesowe

Reguły biznesowe stanowią zbiór ograniczeń i zasad, które muszą być przestrzegane przez system w celu zapewnienia poprawności danych oraz zgodności z wymaganiami organizacyjnymi. Reguły te definiują dozwolone operacje, warunki integralności oraz automatyczne zachowania systemu, będąc podstawą dla implementacji walidacji i kontroli biznesowej. Tabela 18 przedstawia kompletny zbiór reguł biznesowych pogrupowanych według kategorii funkcjonalnych z opisem każdej reguły.

Tabela . Reguły biznesowe

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lp.** | **Cel użycia** | **Opis implementacji** | **Sposób użycia** | **Miejsce implementacji** | **Opis reguły** |
| 1 | Zapewnienie unikalności identyfikacji użytkownika | Unikalny indeks na kolumnie email w tabeli UŻYTKOWNIK | Baza danych (definicja tabeli) | Baza danych | Każdy użytkownik musi posiadać unikalny adres email w systemie. |
| 2 | Automatyczne przypisanie podstawowych uprawnień | Domyślna wartość id\_roli ustawiona na ID roli „czytelnik” w tabeli UŻYTKOWNIK | Aplikacja (logika rejestracji) | Aplikacja | Nowy użytkownik automatycznie otrzymuje domyślną rolę „czytelnik”. |
| 3 | Wymuszenie pojedynczej roli dla użytkownika | Związek jeden-do-wielu (1:N) między ROLA a UŻYTKOWNIK | Baza danych (relacja) | Baza danych | Użytkownik może mieć przypisaną tylko jedną rolę systemową w danym czasie. |
| 4 | Kontrola dostępu do zarządzania rolami | Weryfikacja roli użytkownika przed zezwoleniem na zmianę ról | Aplikacja (logika autoryzacji) | Aplikacja | Tylko administratorzy mogą zmieniać role innych użytkowników. |
| 5 | Zabezpieczenie dostępu do systemu | Sprawdzenie flagi aktywny w tabeli UŻYTKOWNIK podczas próby logowania | Aplikacja (logika logowania) | Aplikacja | Nieaktywny użytkownik nie może logować się do systemu. |
| 6 | Wymuszenie odpowiedzialności za dokument | Kolumna id\_właściciela w tabeli DOKUMENT z ograniczeniem NOT NULL | Baza danych (definicja tabeli) | Baza danych | Każdy dokument musi mieć przypisanego właściciela (użytkownika, który go utworzył). |
| 7 | Kontrola wykorzystania zasobów serwera | Walidacja rozmiaru pliku po stronie aplikacji przed zapisem na dysk | Aplikacja (logika uploadu) | Aplikacja | Rozmiar przesyłanego pliku nie może przekraczać 50 MB. |
| 8 | Zapewnienie kompatybilności i użyteczności plików | Lista dozwolonych rozszerzeń plików sprawdzana podczas uploadu | Aplikacja (logika uploadu) | Aplikacja | System akceptuje tylko określone typy plików: PDF, DOCX, DOC, XLSX, XLS, TXT, PNG, JPG, JPEG. |
| 9 | Zwiększenie bezpieczeństwa systemu | Blokowanie przesyłania plików z listy zabronionych rozszerzeń (.exe, .bat, itp.) | Aplikacja (logika uploadu) | Aplikacja | Pliki wykonywalne (.exe, .bat, .cmd, .scr) są blokowane ze względów bezpieczeństwa. |
| 10 | Retencja danych i możliwość odzyskania | Ustawienie flagi usunięty na TRUE zamiast fizycznego usunięcia rekordu | Aplikacja (logika usuwania) | Aplikacja/Baza danych | Dokumenty są usuwane „miękko” (soft delete) - oznaczane flagą usunięty=True. |
| 11 | Automatyzacja nadawania podstawowych uprawnień | Automatyczne utworzenie wpisu w tabeli UPRAWNIENIE\_DOKUMENT po utworzeniu dokumentu | Aplikacja (logika tworzenia dokumentu) | Aplikacja | Właściciel dokumentu automatycznie otrzymuje pełen zestaw uprawnień do swojego dokumentu. |
| 12 | Organizacja i struktura danych | Pole id\_rodzica w tabeli FOLDER wskazujące na relację rekurencyjną | Baza danych (definicja tabeli) | Baza danych | Folder może zawierać inne foldery (struktura hierarchiczna). |
| 13 | Unikalność identyfikacji folderów | Unikalny indeks złożony na nazwa, id\_rodzica i id\_właściciela w tabeli FOLDER | Baza danych (definicja tabeli) | Baza danych | Nazwa folderu musi być unikalna w ramach tego samego folderu nadrzędnego dla danego właściciela. |
| 14 | Uproszczenie zarządzania folderami głównymi | Logika aplikacji ograniczająca tworzenie zagnieżdżonych folderów głównych | Aplikacja (logika tworzenia folderu) | Aplikacja | Folder główny (bez rodzica) może mieć tylko jeden poziom dla każdego użytkownika. |
| 15 | Zachowanie historii zmian dokumentu | Relacja jeden-do-wielu (1:N) między DOKUMENT a WERSJA\_DOKUMENT | Baza danych (relacja) | Baza danych | Każdy dokument może mieć wiele wersji. |
| 16 | Uporządkowane zarządzanie wersjami | Automatyczna inkrementacja pola numer\_wersji w aplikacji | Aplikacja (logika tworzenia wersji) | Aplikacja | Numery wersji są automatycznie inkrementowane i muszą być unikalne dla danego dokumentu. |
| 17 | Zapewnienie dostępu do aktualnej wersji | Implementacja mechanizmu wskazującego na najnowszą wersję (np. pole id\_aktualnej\_wersji w DOKUMENT) | Aplikacja (logika pobierania/wyświetlania) | Aplikacja | Główny rekord dokumentu zawsze wskazuje na najnowszą wersję. |
| 18 | Dostęp do historii dokumentu | Zachowywanie wszystkich historycznych rekordów w tabeli WERSJA\_DOKUMENT | Baza danych (brak usuwania) | Baza danych | Poprzednie wersje dokumentu pozostają dostępne do pobrania. |
| 19 | Centralne zarządzanie dostępem administracyjnym | Logika autoryzacji sprawdzająca rolę użytkownika (Administrator) | Aplikacja (logika autoryzacji) | Aplikacja | Administrator ma dostęp do wszystkich dokumentów i folderów w systemie. |
| 20 | Elastyczne zarządzanie dostępem przez właścicieli | Funkcjonalność w aplikacji umożliwiająca właścicielowi modyfikację wpisów w UPRAWNIENIE\_DOKUMENT | Aplikacja (zarządzanie uprawnieniami) | Aplikacja | Właściciel może nadawać uprawnienia do swoich dokumentów innym użytkownikom. |
| 21 | Kontrola dostępu do funkcji komentowania | Weryfikacja uprawnienia typu "odczyt" dla danego dokumentu przed dodaniem komentarza | Aplikacja (logika komentowania) | Aplikacja | Użytkownik może komentować dokument tylko jeśli ma uprawnienie do jego przeglądania. |
| 22 | Definiowanie poziomów dostępu na podstawie ról | Logika autoryzacji oparta na przypisanej roli użytkownika | Aplikacja (logika autoryzacji) | Aplikacja | Edytorzy mogą tworzyć dokumenty, czytelnicy tylko je przeglądać. |
| 23 | Unikalność identyfikacji tagów | Unikalny indeks na kolumnie nazwa w tabeli TAG | Baza danych (definicja tabeli) | Baza danych | Tag musi mieć unikalną nazwę w całym systemie. |
| 24 | Elastyczne kategoryzowanie dokumentów | Tabela łącząca DOKUMENT\_TAG obsługująca relację wiele-do-wielu (M:N) | Baza danych (relacja) | Baza danych | Jeden dokument może mieć wiele tagów, jeden tag może być przypisany do wielu dokumentów. |
| 25 | Unikalność i spójność metadanych | Unikalny indeks złożony na id\_dokumentu i klucz w tabeli METADANE | Baza danych (definicja tabeli) | Baza danych | Para klucz-wartość metadanych musi być unikalna dla danego dokumentu. |
| 26 | Audytowanie operacji systemowych | Automatyczne zapisywanie szczegółów akcji w tabeli LOG\_AKTYWNOŚCI | Aplikacja (logika operacji) | Aplikacja | Wszystkie operacje na dokumentach muszą być logowane w tabeli LOG\_AKTYWNOŚCI. |
| 27 | Wzmacnianie bezpieczeństwa kont użytkowników | Walidacja złożoności hasła po stronie aplikacji/serwera podczas rejestracji/zmiany hasła | Aplikacja (logika uwierzytelniania) | Aplikacja | Hasła użytkowników muszą spełniać kryteria siły (min. 8 znaków, wielkie/małe litery, cyfry, znaki specjalne). |
| 28 | Weryfikacja integralności danych plików | Generowanie i weryfikacja hasha SHA-256 dla każdego pliku | Aplikacja (logika uploadu/pobierania) | Aplikacja | Każdy plik musi mieć wygenerowany hash SHA-256 dla weryfikacji integralności. |

Źródło: opracowanie własne

#### Normalizacja

Proces normalizacji logicznego modelu danych ma na celu eliminację redundancji oraz zapewnienie integralności danych poprzez systematyczne stosowanie form normalnych. Analiza normalizacji weryfikuje, czy struktura bazy danych jest zoptymalizowana pod kątem spójności, wydajności oraz łatwości utrzymania, minimalizując jednocześnie ryzyko anomalii podczas operacji na danych.

Pierwsza postać normalna (1NF):

Model spełnia wymagania 1NF poprzez:

* każda tabela posiada klucz główny (PRIMARY KEY),
* wszystkie atrybuty są atomowe (niepodzielne),
* brak powtarzających się grup danych,
* każda komórka zawiera tylko jedną wartość.

Przykład: Tabela DOKUMENT nie zawiera wielowartościowych pól - tagi są przechowywane w oddzielnej tabeli DOKUMENT\_TAG z relacją many-to-many.

Druga postać normalna (2NF):

* Model spełnia wymagania 2NF poprzez:
* Spełnienie warunków 1NF
* Eliminację częściowych zależności funkcyjnych od klucza głównego
* Wszystkie atrybuty nieprzykluczowe są w pełni zależne od całego klucza głównego

Przykład: W tabeli WERSJA\_DOKUMENT atrybuty takie jak data\_utworzenia, komentarz czy rozmiar\_pliku zależą od całego klucza głównego (id), a nie tylko od części klucza.

Trzecia postać normalna (3NF):

Model spełnia wymagania 3NF poprzez:

* Spełnienie warunków 2NF
* Eliminację zależności przechodnich
* Atrybuty nieprzykluczowe zależą tylko od klucza głównego, nie od innych atrybutów nieprzykluczowych

Przykłady:

* Separacja metadanych dokumentów: Metadane dokumentów przechowywane są w oddzielnej tabeli METADANE zamiast jako kolumny w tabeli DOKUMENT, co umożliwia dodawanie dowolnych par klucz-wartość bez modyfikacji schematu.
* Separacja ról użytkowników: Role użytkowników są przechowywane w oddzielnej tabeli ROLA i powiązane przez klucz obcy, zamiast przechowywania nazwy roli bezpośrednio w tabeli użytkownika.

#### Model w postaci graficznej

Graficzna reprezentacja logicznego modelu danych stanowi wizualną syntezę wszystkich wcześniej zdefiniowanych encji, atrybutów i związków, przedstawiając kompleksową strukturę systemu w przejrzystej formie diagramu. Diagram umożliwia szybkie zrozumienie architektury danych oraz weryfikację poprawności zaprojektowanych relacji między obiektami biznesowymi. Rysunek 18 przedstawia logiczny model danych (LDM) systemu zarządzania dokumentami w postaci diagramu encja-związek.

Rysunek 18. Diagram LDM

Obraz zawierający tekst, zrzut ekranu, Równolegle, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Źródło: opracowanie własne

### Fizyczny model danych (PDM)

Fizyczny model danych (PDM) stanowi szczegółową implementację struktury bazy danych w kontekście wybranego systemu zarządzania bazą danych, w tym przypadku SQLite. Model ten uwzględnia wszystkie ograniczenia technologiczne oraz specyficzne cechy wybranego DBMS, zoptymalizowane pod kątem wydajności i niezawodności.

#### Wykaz tabel fizycznych

Aplikacja webowa do zarządzania dokumentami składa się z 28 tabel podzielonych na kilka kategorii funkcjonalnych. Django ORM automatycznie tworzy fizyczną strukturę bazy danych na podstawie zdefiniowanych modeli, generując odpowiednie tabele SQLite z właściwymi typami danych, indeksami oraz ograniczeniami integralności.

Tabele głównej aplikacji documents

Aplikacja documents zawiera 9 głównych tabel implementujących logikę zarządzania dokumentami które przedstawia tabela 8.

Tabela 19. Tabele związane z dokumentami

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Nazwa tabeli** | **Opis** |
| 1 | dokument | Główna tabela przechowująca metadane dokumentów |
| 2 | folder | Hierarchiczna struktura katalogów |
| 3 | wersja\_dokumentu | System wersjonowania dokumentów |
| 4 | tag | Tagi do kategoryzacji dokumentów i folderów |
| 5 | metadane | Niestandardowe metadane dokumentów (pary klucz-wartość) |
| 6 | documents\_comment | System komentarzy do dokumentów z obsługą wątków |
| 7 | log\_aktywnosci | Audyt wszystkich operacji w systemie |
| 8 | document\_share | Śledzenie udostępnień dokumentów z określonymi uprawnieniami |
| 9 | system\_settings | Konfiguracja systemowa aplikacji |

Źródło: opracowanie własne

Tabele aplikacji users

Aplikacja users zarządza użytkownikami i sesjami poprzez 3 tabele przedstawione w tabeli 9.

Tabela 20. Tabele związane z użytkownikami

| **Lp.** | **Nazwa tabeli** | **Opis** |
| --- | --- | --- |
| 1 | rola | Definicje ról systemowych (Administrator, Edytor, Czytelnik) |
| 2 | user\_profile | Rozszerzone profile użytkowników z rolami i dodatkowymi danymi |
| 3 | user\_session | Śledzenie sesji użytkowników dla celów bezpieczeństwa |

Źródło: opracowanie własne

Tabele relacji Many-to-Many

Django automatycznie tworzy 2 tabele łączące dla relacji wiele-do-wielu, które przedstawia tabela 10.

Tabela . Tabele relacji

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Nazwa tabeli** | **Opis** |
| 1 | dokument\_tagi | Łączy dokumenty z tagami |
| 2 | folder\_tagi | Łączy foldery z tagami |

Źródło: opracowanie własne

Każda tabela łączącą zawiera pola id (klucz główny), identyfikatory obiektów oraz ograniczenie unikalności na kombinację kluczy obcych.

Tabele systemowe Django

Framework Django automatycznie tworzy 11 tabel systemowych niezbędnych do swojego funkcjonowania, które przedstawia tabela 11.

Tabela . Tabele systemowe

| **Lp.** | **Nazwa tabeli** | **Kategoria** | **Opis** |
| --- | --- | --- | --- |
| 1 | auth\_user | Uwierzytelnianie | Główna tabela użytkowników Django |
| 2 | auth\_group | Uwierzytelnianie | Grupy użytkowników |
| 3 | auth\_permission | Uwierzytelnianie | Definicje uprawnień systemowych |
| 4 | auth\_group\_permissions | Uwierzytelnianie | Tabela łączą grupa-uprawnienia |
| 5 | auth\_user\_groups | Uwierzytelnianie | Tabela łączą użytkownik-grupa |
| 6 | auth\_user\_user\_permissions | Uwierzytelnianie | Tabela łączą użytkownik-uprawnienia |
| 7 | django\_migrations | Zarządzanie | Historia migracji bazy danych |
| 8 | django\_content\_type | Zarządzanie | Registry typów modeli Django |
| 9 | django\_session | Zarządzanie | Sesje użytkowników |
| 10 | django\_admin\_log | Zarządzanie | Logi aktywności w panelu administracyjnym |
| 11 | django\_site | Zarządzanie | Konfiguracja wielu witryn (opcjonalna) |

Źródło: opracowanie własne

Tabele systemu uprawnień Guardian

Biblioteka Django Guardian dodaje 2 tabele dla zarządzania uprawnieniami na poziomie obiektów, które przedstawia tabela 12.

Tabela . Tabele Guardian

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Nazwa tabeli** | **Opis** |
| 1 | guardian\_userobjectpermission | Uprawnienia użytkowników do konkretnych obiektów |
| 2 | guardian\_groupobjectpermission | Uprawnienia grup do konkretnych obiektów |

Źródło: opracowanie własne

Te tabele umożliwiają granularną kontrolę dostępu do dokumentów i folderów, wykraczającą poza standardowy system ról Django.

Tabela sekwencji SQLite

SQLite automatycznie tworzy i zarządza systemową tabelą sqlite\_sequence do obsługi kolumn z atrybutem AUTOINCREMENT.

#### Wykaz pól i mapowanie typów danych

Django ORM automatycznie mapuje typy pól modeli na odpowiednie typy danych SQLite, tworząc właściwe ograniczenia integralności i walidacje. Tabela 13 przedstawia szczegółowe mapowanie dwunastu podstawowych typów pól Django ORM na odpowiadające im typy danych w bazie SQLite wraz z przykładami praktycznego zastosowania i opisem funkcjonalności każdego typu.

Tabela . Mapowanie

|  |  |  |  |
| --- | --- | --- | --- |
| **Typ pola Django** | **Typ SQLite** | **Przykład użycia** | **Opis** |
| AutoField | INTEGER PRIMARY KEY AUTOINCREMENT | id | Automatyczny klucz główny |
| CharField(max\_length=n) | VARCHAR(n) | nazwa, typ\_pliku, status | Tekst o ograniczonej długości |
| TextField() | TEXT | opis, tresc, szczegoly | Tekst o nieograniczonej długości |
| PositiveIntegerField() | INTEGER CHECK >= 0 | rozmiar\_pliku, numer\_wersji | Liczba całkowita nieujemna |
| BooleanField() | BOOLEAN | usunieto, aktywny, dwuetapowa\_weryfikacja | Wartość logiczna (0/1) |
| DateTimeField() | DATETIME | data\_utworzenia, znacznik\_czasu | Data i czas |
| DateField() | DATE | data\_urodzenia | Data |
| ForeignKey() | INTEGER REFERENCES table(id) | wlasciciel\_id, dokument\_id | Klucz obcy z ograniczeniem |
| OneToOneField() | INTEGER UNIQUE REFERENCES | user\_id w UserProfile | Relacja jeden-do-jeden |
| FileField() | VARCHAR(100) | plik, avatar | Ścieżka do pliku |
| ImageField() | VARCHAR(100) | avatar | Ścieżka do obrazu |
| GenericIPAddressField() | VARCHAR(39) | adres\_ip | Adres IP (IPv4/IPv6) |

Źródło: opracowanie własne

Ograniczenia i walidacje na poziomie bazy danych

Django automatycznie implementuje następujące ograniczenia integralności:

1. Ograniczenia NOT NULL
   * Wszystkie pola bez null=True otrzymują ograniczenie NOT NULL, zapewniające obecność wartości.
2. Ograniczenia CHECK
   * PositiveIntegerField → CHECK (pole >= 0)
   * BooleanField → automatyczna konwersja na 0/1
3. Ograniczenia UNIQUE
   * Pola z unique=True → UNIQUE constraint
   * unique\_together → złożone ograniczenia unikalne
4. Ograniczenia FOREIGN KEY
   * Wszystkie relacje ForeignKey implementują ograniczenia referencyjne z opcjami ON DELETE:
     1. CASCADE - usunięcie kaskadowe
     2. SET\_NULL - ustawienie NULL przy usunięciu
     3. PROTECT - blokada usunięcia

Automatyczne pola systemowe

Tabela 14 przedstawia automatyczne pola systemowe, które Django ORM dodaje do modeli w celu zapewnienia podstawowej funkcjonalności identyfikacji rekordów oraz śledzenia czasu ich utworzenia i modyfikacji

Tabela 25. Automatyczne pola

|  |  |  |  |
| --- | --- | --- | --- |
| **Pole** | **Typ** | **Tabele** | **Opis** |
| id | INTEGER PRIMARY KEY AUTOINCREMENT | Wszystkie główne | Klucz główny |
| data\_utworzenia | DATETIME | Modele z auto\_now\_add=True | Automatyczny timestamp |
| ostatnia\_modyfikacja | DATETIME | Modele z auto\_now=True | Automatyczna aktualizacja |

Źródło: opracowanie własne

Walidacja na poziomie aplikacji

Oprócz ograniczeń bazodanowych, Django implementuje walidację w kodzie aplikacji:

* FileExtensionValidator - sprawdzanie dozwolonych rozszerzeń plików
* Walidacja rozmiaru - kontrola wielkości przesyłanych plików (max 50MB)
* Walidacja hashy - automatyczne generowanie SHA-256 dla plików
* Custom validators - niestandardowe walidacje w metodach clean()

#### Indeksowanie automatyczne

Django automatycznie tworzy indeksy dla:

* Wszystkich kluczy głównych (PRIMARY KEY)
* Wszystkich kluczy obcych (FOREIGN KEY)
* Pól z ograniczeniem unique=True
* Kombinacji pól w unique\_together

#### Logika biznesowa na poziomie aplikacji

Django implementuje logikę biznesową na poziomie aplikacji w języku Python, zastępując tradycyjne procedury składowane i funkcje bazodanowe. Takie podejście zapewnia większą kontrolę, testowanie oraz przenośność między różnymi systemami baz danych. Django dystrybuuje logikę biznesową między 5 głównych warstw aplikacji, które przedstawia tabela 15.

Tabela . Architektura logiki biznesowej

|  |  |  |  |
| --- | --- | --- | --- |
| **Warstwa** | **Lokalizacja** | **Przeznaczenie** | **Przykłady** |
| Model Methods | models.py | Logika związana z pojedynczymi obiektami | get\_file\_size\_display(), get\_full\_path() |
| Model Managers | models.py | Logika operacji na zbiorach obiektów | Custom QuerySets, filtry biznesowe |
| View Logic | views.py | Logika kontrolerów i przepływu danych | Walidacja uprawnień, obsługa formularzy |
| Business Logic | utils.py, helpers.py | Niezależne funkcje biznesowe | Obliczenia, walidacje, transformacje |
| Permission Logic | permissions.py | Kontrola dostępu i autoryzacja | Sprawdzanie uprawnień, logika ról |

#### Sygnały Django

Django wykorzystuje system sygnałów do implementacji logiki, która w tradycyjnych bazach danych byłaby realizowana przez wyzwalacze (triggers). Sygnały umożliwiają automatyczne wykonywanie kodu w odpowiedzi na określone zdarzenia w cyklu życia obiektów, zapewniając spójność danych oraz automatyzację procesów biznesowych. Tabela 15 przedstawia konfigurację sygnałów.

Tabela . Sygnały

|  |  |  |  |
| --- | --- | --- | --- |
| **Sygnał** | **Lokalizacja** | **Funkcja** | **Przeznaczenie** |
| post\_save | users/models.py | create\_user\_profile | Automatyczne tworzenie profilu użytkownika |
| post\_save | users/models.py | save\_user\_profile | Automatyczne zapisywanie profilu przy edycji User |
| post\_migrate | signals.py | create\_default\_roles | Tworzenie domyślnych ról po migracji |

Źródło: opracowanie własne

Funkcja create\_user\_profile tworzy rozszerzony profil użytkownika po utworzeniu konta.

Funkcjonalności:

* Automatyczne przypisanie domyślnej roli „czytelnik” nowym użytkownikom
* Tworzenie roli domyślnej jeśli nie istnieje w systemie
* Ustawienie statusu aktywności na True
* Logowanie operacji do konsoli

Funkcja create\_user\_profile zapewnia synchronizację danych między modelem User a UserProfile.

Funkcjonalności:

* Sprawdzenie istnienia profilu przed zapisem
* Unikanie rekurencji przy tworzeniu nowego użytkownika
* Automatyczna synchronizacja przy każdej edycji User

Sygnał post\_migrate zapewnia automatyczne utworzenie wymaganych ról systemowych po każdej migracji.

Funkcjonalności:

* Filtrowanie według nazwy aplikacji (tylko 'users')
* Sprawdzenie istnienia modelu Role przed wykonaniem
* Wykorzystanie get\_or\_create() dla unikania duplikatów
* Logowanie wyniku operacji

#### Macierz CRUD

Macierz CRUD (Create, Read, Update, Delete) definiuje uprawnienia poszczególnych ról użytkowników do wykonywania operacji na kluczowych encjach systemu. Macierz stanowi kluczowy element specyfikacji systemu uprawnień, określając które role mogą wykonywać konkretne operacje biznesowe w ramach aplikacji do zarządzania dokumentami, przedstawia to tabela 17.

Legenda oznaczeń:

* ✓ - Pełne uprawnienia do operacji
* - Uprawnienia ograniczone (tylko do własnych zasobów)
* ✗ - Brak uprawnień do operacji

Tabela . CRUD dla roli

| **Encja** | **Operacja** | **Administrator** | **Edytor** | **Czytelnik** | **Opis ograniczeń** |
| --- | --- | --- | --- | --- | --- |
| UŻYTKOWNIK | Create | ✓ | ✗ | ✗ | Tylko administratorzy mogą tworzyć konta użytkowników |
| UŻYTKOWNIK | Read | ✓ | O | O | Edytorzy i czytelnicy widzą tylko podstawowe dane innych użytkowników |
| UŻYTKOWNIK | Update | ✓ | O | O | Użytkownicy mogą edytować tylko własne profile |
| UŻYTKOWNIK | Delete | ✓ | ✗ | ✗ | Tylko administrator może usuwać konta |
| DOKUMENT | Create | ✓ | ✓ | ✗ | Czytelnicy nie mogą tworzyć dokumentów |
| DOKUMENT | Read | ✓ | ✓ | ✓ | Dostęp zgodny z nadanymi uprawnieniami do dokumentu |
| DOKUMENT | Update | ✓ | O | ✗ | Edytorzy mogą edytować własne dokumenty i te z nadanymi uprawnieniami |
| DOKUMENT | Delete | ✓ | O | ✗ | Soft delete - edytorzy mogą usuwać własne dokumenty |
| FOLDER | Create | ✓ | ✓ | ✗ | Czytelnicy nie mogą tworzyć folderów |
| FOLDER | Read | ✓ | ✓ | ✓ | Dostęp zgodny z nadanymi uprawnieniami do folderu |
| FOLDER | Update | ✓ | O | ✗ | Edytorzy mogą edytować własne foldery |
| FOLDER | Delete | ✓ | O | ✗ | Edytorzy mogą usuwać własne foldery |
| WERSJA\_DOKUMENT | Create | ✓ | O | ✗ | Nowe wersje mogą tworzyć tylko osoby z uprawnieniami do edycji dokumentu |
| WERSJA\_DOKUMENT | Read | ✓ | ✓ | ✓ | Dostęp do historii wersji zgodny z uprawnieniami do dokumentu |
| WERSJA\_DOKUMENT | Update | ✗ | ✗ | ✗ | Wersje są niemodyfikowalne po utworzeniu |
| WERSJA\_DOKUMENT | Delete | ✓ | ✗ | ✗ | Tylko administrator może usuwać wersje dokumentów |
| TAG | Create | ✓ | ✓ | ✗ | Czytelnicy nie mogą tworzyć nowych tagów |
| TAG | Read | ✓ | ✓ | ✓ | Wszyscy użytkownicy widzą dostępne tagi |
| TAG | Update | ✓ | O | ✗ | Edytorzy mogą modyfikować tagi przez siebie utworzone |
| TAG | Delete | ✓ | O | ✗ | Edytorzy mogą usuwać własne tagi |
| METADANE | Create | ✓ | O | ✗ | Metadane mogą dodawać osoby z uprawnieniami do edycji dokumentu |
| METADANE | Read | ✓ | ✓ | ✓ | Dostęp do metadanych zgodny z uprawnieniami do dokumentu |
| METADANE | Update | ✓ | O | ✗ | Modyfikacja przez osoby z uprawnieniami do edycji dokumentu |
| METADANE | Delete | ✓ | O | ✗ | Usuwanie przez osoby z uprawnieniami do edycji dokumentu |
| KOMENTARZ | Create | ✓ | ✓ | O | Czytelnicy mogą komentować tylko przy jawnym nadaniu uprawnień |
| KOMENTARZ | Read | ✓ | ✓ | ✓ | Dostęp do komentarzy zgodny z uprawnieniami do dokumentu |
| KOMENTARZ | Update | ✓ | O | O | Użytkownicy mogą edytować własne komentarze |
| KOMENTARZ | Delete | ✓ | O | O | Użytkownicy mogą usuwać własne komentarze |
| LOG\_AKTYWNOŚCI | Create | Auto | Auto | Auto | Automatyczne logowanie przez system |
| LOG\_AKTYWNOŚCI | Read | ✓ | O | O | Użytkownicy widzą własne logi, administratorzy wszystkie |
| LOG\_AKTYWNOŚCI | Update | ✗ | ✗ | ✗ | Logi są niemodyfikowalne |
| LOG\_AKTYWNOŚCI | Delete | ✓ | ✗ | ✗ | Tylko administrator może czyścić stare logi |

Źródło: opracowanie własne

#### Estymacja rozmiaru bazy danych

Estymacja rozmiaru bazy danych dla aplikacji webowej do zarządzania dokumentami została przygotowana dla małej firmy zatrudniającej 10-15 osób w horyzoncie czasowym 3 lat. Analiza uwzględnia zarówno rozmiar bazy danych SQLite przechowującej metadane, jak i wymagania na przestrzeń dyskową dla przechowywania plików dokumentów w systemie plików. Tabela 18 przedstawia szczegółową estymację rozmiaru storage dla wszystkich komponentów aplikacji DMS w horyzoncie trzech lat działania małej firmy.

Założenia estymacyjne:

* Liczba użytkowników: 12 osób (10-15)
* Aktywni użytkownicy dziennie: 10 osób
* Przesyłanie dokumentów: 12 dokumentów miesięcznie na aktywnego użytkownika
* Średni rozmiar dokumentu: 2,5 MB
* Średnia liczba wersji na dokument: 2,8
* Wzrost aktywności: 15% rocznie
* Retencja backup: 30 dni

Tabela 29. Estymacja rozmiaru przestrzeni dyskowej aplikacji DMS (w GB)

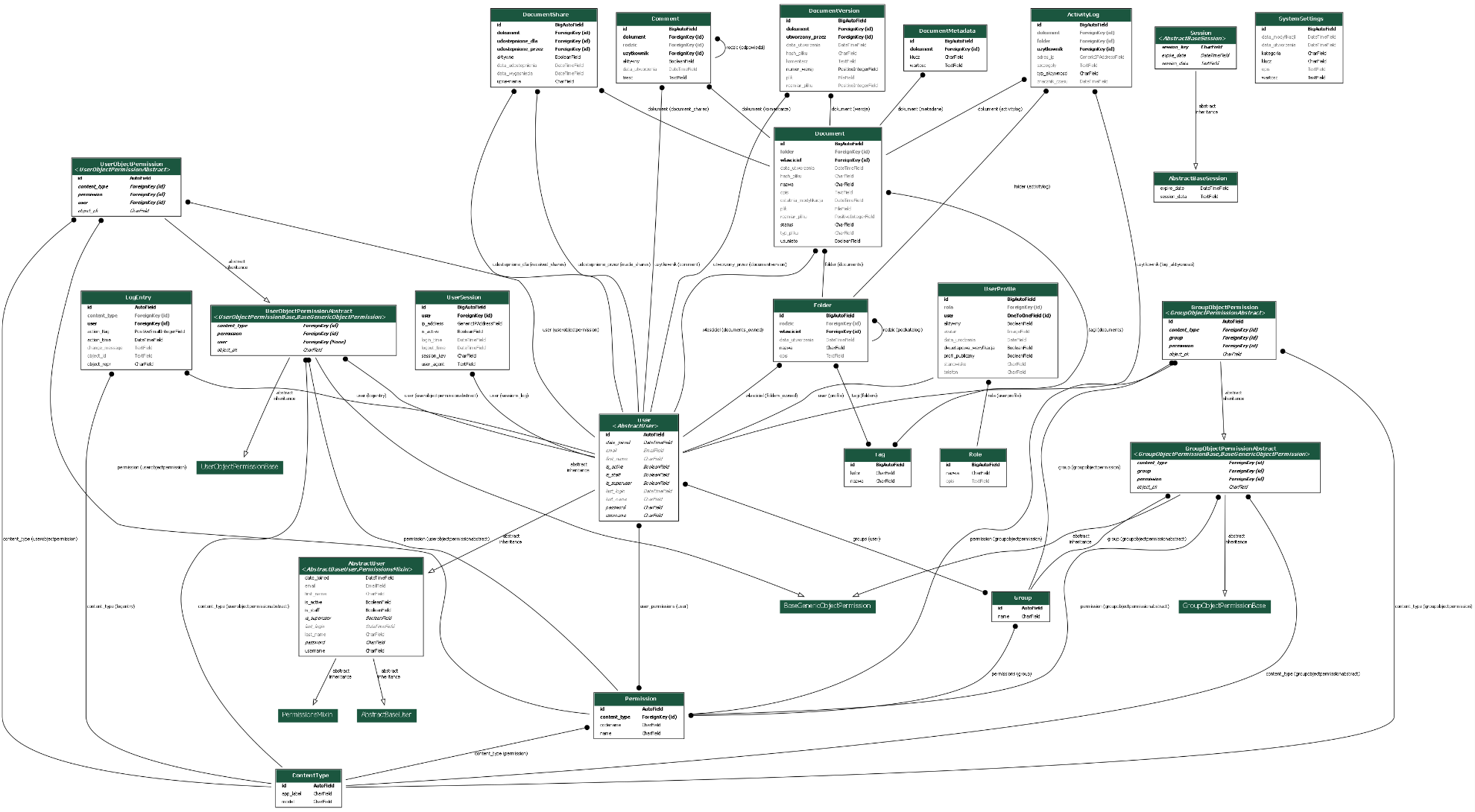
| **Komponent** | **Rok 1** | **Rok 2** | **Rok 3** | **Opis kalkulacji** |
| --- | --- | --- | --- | --- |
| **Baza danych SQLite** |  |  |  |  |
| Tabele aplikacyjne | 0,045 | 0,085 | 0,140 | Metadane dokumentów, użytkownicy, role |
| Tabele systemowe Django | 0,015 | 0,025 | 0,035 | Auth, sessions, migrations |
| Logi aktywności | 0,120 | 0,290 | 0,480 | ~1200 wpisów miesięcznie, wzrost aktywności |
| Komentarze i metadane | 0,020 | 0,050 | 0,090 | Dodatkowe dane użytkowników |
| Indeksy i narzut | 0,040 | 0,090 | 0,150 | 20% narzut SQLite |
| Suma baza danych | 0,24 | 0,54 | 0,90 |  |
| **System plików** |  |  |  |  |
| Dokumenty główne | 3,60 | 8,28 | 14,28 | 10×12×12×2,5MB miesięcznie + wzrost |
| Wersje dokumentów | 6,48 | 14,90 | 25,70 | 180% dokumentów w dodatkowych wersjach |
| Pliki statyczne aplikacji | 0,05 | 0,05 | 0,05 | CSS, JS, ikony - stały rozmiar |
| Suma pliki | 10,13 | 23,23 | 40,03 |  |
| **Backup i bezpieczeństwo** |  |  |  |  |
| Backup bazy danych | 0,02 | 0,05 | 0,09 | 30 dni retencji, kompresja 80% |
| Backup dokumentów | 1,01 | 2,32 | 4,00 | 10% kluczowych dokumentów |
| Logi systemowe | 0,05 | 0,12 | 0,20 | Nginx, Gunicorn, system |
| Suma backup | 1,08 | 2,49 | 4,29 |  |
| **Suma przestrzeń dyskowa** | 11,45 | 26,26 | 45,22 |  |
| **Rekomendowana przestrzeń dyskowa** | 20,00 | 40,00 | 70,00 | Zapas 75% + przyszły rozwój |

Źródło: opracowanie własne

#### Model fizyczny w postaci graficznej

Model fizyczny bazy danych przedstawia rzeczywistą strukturę tabel SQLite wygenerowanych przez Django ORM, uwzględniając wszystkie automatycznie utworzone pola, indeksy oraz ograniczenia integralności. Diagram fizycznego modelu danych różni się od modelu logicznego obecnością tabel systemowych Django oraz rzeczywistymi nazwami kolumn i typami danych SQLite. Rysunek 19 został wygenerowany za pomocą komendy „python manage.py graph\_models” z pakietu django-extensions, która automatycznie tworzy graficzną reprezentację struktury bazy danych na podstawie zdefiniowanych modeli Django.

Rysunek 19. Model fizyczny



Źródło: opracowanie własne

# Projektowanie aplikacji webowej

Projektowanie aplikacji webowej stanowi kluczowy etap implementacji systemu zarządzania dokumentami, w którym abstrakcyjne wymagania funkcjonalne oraz logiczna struktura bazy danych zostają przekształcone w konkretne rozwiązania techniczne dostępne dla użytkowników końcowych. Na podstawie fundamentów zdefiniowanych w poprzednich rozdziałach - analizy wymagań, architektury systemu oraz projektu bazy danych - niniejszy rozdział koncentruje się na praktycznej realizacji interfejsu użytkownika oraz implementacji logiki biznesowej aplikacji.

## Projekt interfejsu użytkownika

Projekt interfejsu użytkownika stanowi krytyczny element sukcesu aplikacji webowej do zarządzania dokumentami, determinujący sposób, w jaki użytkownicy końcowi będą odczuwać i wykorzystywać zaimplementowane funkcjonalności systemu. W kontekście aplikacji DMS, gdzie użytkownicy muszą efektywnie nawigować między hierarchicznymi strukturami folderów, zarządzać złożonymi uprawnieniami oraz obsługiwać zaawansowane funkcje wersjonowania dokumentów, kluczowe znaczenie ma stworzenie interfejsu, który łączy funkcjonalność z intuicyjnością obsługi.

### Struktura i szablony HTML

Aplikacja webowa do zarządzania dokumentami wykorzystuje system szablonów Django oparty na hierarchicznym modelu dziedziczenia, gdzie główny szablon bazowy definiuje wspólne elementy interfejsu, a poszczególne podstrony rozszerzają go o specyficzne treści.

Szablon bazowy i struktura nawigacji

Głównym elementem architektury szablonów jest plik base.html, który definiuje podstawową strukturę każdej strony aplikacji, włączając responsywną nawigację Bootstrap, sidebar z szybkimi akcjami oraz główny obszar treści. Szablon zawiera dynamiczne menu użytkownika, które automatycznie dostosowuje się do roli zalogowanego użytkownika - administratorzy widzą dodatkowe opcje zarządzania systemem, edytorzy mają dostęp do funkcji tworzenia dokumentów, a czytelnicy otrzymują ograniczony interfejs. Rysunek 20 przedstawia interfejs głównej strony aplikacji.

Rysunek 20. Makieta interfejsu głównej strony aplikacji z nawigacją

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Szablony zarządzania dokumentami

Aplikacja zawiera zestaw specjalizowanych szablonów dla operacji na dokumentach. Szablon document\_list.html implementuje interfejs przeglądania dokumentów z możliwością przełączania między widokiem siatki i listy, zaawansowanym filtrowaniem oraz paginacją. Szablon document\_detail.html prezentuje szczegółowe informacje o dokumencie, jego wersje, komentarze oraz dostępne akcje w zależności od uprawnień użytkownika. Szablon home.html stanowi główny dashboard w stylu eksploratora plików z hierarchiczną strukturą folderów i dokumentów. Rysunek 21 przedstawia widok listy dokumentów. Rysunek 22 przedstawia szczegóły dokumentu z wersjonowaniem.

Rysunek 21. Makieta widoku listy dokumentów

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Rysunek 22. Makieta szczegółów dokumentu z wersjonowaniem

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Szablony formularzy

Formularze przesyłania i edycji dokumentów (document\_upload.html, document\_edit.html) zawierają zaawansowane mechanizmy walidacji po stronie klienta oraz dynamiczne podglądy wybranych plików. Szablon document\_version\_upload.html umożliwia dodawanie nowych wersji z komentarzami oraz porównywaniem z poprzednimi wersjami. Formularze automatycznie dostosowują się do typu przesyłanego pliku i zawierają mechanizmy przeciągnij-i-upuść dla lepszego doświadczenia użytkownika. Rysunek 23, 24 przedstawia formularz przesyłania dokumentu z podglądem.

Rysunek 23. Formularz przesyłania dokumentu z podglądem 1

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Rysunek 24. Formularz przesyłania dokumentu z podglądem 2

Obraz zawierający tekst, oprogramowanie, zrzut ekranu, design

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Szablony zarządzania folderami

Szablony folder\_list.html, folder\_create.html i folder\_edit.html obsługują hierarchiczną strukturę folderów z możliwością tworzenia zagnieżdżonych katalogów. Interfejs umożliwia zarządzanie uprawnieniami do folderów, przenoszenie dokumentów między katalogami oraz zaawansowane opcje usuwania z wyborem sposobu postępowania z zawartością. Rysunek 25 przedstawia struktura folderów w widoku.

Rysunek 25. Makieta struktury folderów w widoku

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, numer

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Szablony autoryzacji

Szablon login.html stanowi samodzielną stronę z nowoczesnym dwukolumnowym układem - formularz logowania po lewej oraz informacje promocyjne o systemie po prawej. Szablon password\_change.html zawiera zaawansowany formularz zmiany hasła z walidacją siły hasła w czasie rzeczywistym oraz wskazówkami bezpieczeństwa. Rysunek 24 przedstawia stronę logowania aplikacji.

Rysunek 26. Makieta strona logowania aplikacji

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Organizacja katalogów

Wszystkie szablony przechowywane są w strukturze templates/ z podkatalogami documents/ i users/ odpowiadającymi aplikacjom Django. Szablony wykorzystują mechanizm bloków Django ({% block %}) dla dziedziczenia oraz system tagów dla dynamicznego generowania treści na podstawie uprawnień użytkownika i stanu danych. Struktura zapewnia łatwą rozbudowę oraz utrzymanie spójności interfejsu w całej aplikacji.

### Stylizacja aplikacji, responsywność i UX

Stylizacja aplikacji webowej do zarządzania dokumentami została zaprojektowana z myślą o stworzeniu nowoczesnego, intuicyjnego interfejsu, który zapewnia spójne doświadczenie użytkownika na wszystkich typach urządzeń. Połączenie frameworka Bootstrap z niestandardowymi stylami CSS oraz zasadami responsywnego designu tworzy funkcjonalne i estetyczne środowisko pracy.

Koncepcja wizualna i system kolorów

Interfejs aplikacji opiera się na profesjonalnej palecie kolorów z dominującymi odcieniami niebieskiego i szarości, które kojarzą się z zaufaniem i stabilnością. System kolorów został zaprojektowany semantycznie - zielony oznacza powodzenie operacji, czerwony błędy, żółty ostrzeżenia, a niebieski informacje neutralne. Na przykład, po pomyślnym wgraniu dokumentu użytkownik widzi zielony komunikat z tekstem „Dokument został dodany pomyślnie”, podczas gdy błąd przesyłania sygnalizowany jest czerwonym komunikatem.

Typografia została dobrana z myślą o czytelności - wykorzystano bezszeryfowe fonty systemowe z wyraźną hierarchią typograficzną. Ikony pochodzą z biblioteki Bootstrap Icons, zapewniając spójność wizualną w całej aplikacji.

Responsywny design i adaptacja interfejsu

Aplikacja została zaprojektowana zgodnie z podejściem „mobile-first”, gdzie interfejs najpierw projektuje się dla najmniejszych ekranów, a następnie rozszerza dla większych urządzeń. System responsywności wykorzystuje breakpointy Bootstrap.

Zasady UX i intuicyjność interfejsu

Interfejs wykorzystuje znane użytkownikom wzorce z tradycyjnych eksploratorów plików - hierarchiczną strukturę folderów, standardowe ikony typów plików (PDF - czerwona ikona, Word - niebieska, Excel - zielona) oraz gesty jak podwójne kliknięcie do otwarcia czy prawy klik dla menu kontekstowego.

Natychmiastowy feedback jest zapewniony przez:

* paski postępu podczas wgrywania z procentowym wskaźnikiem postępu,
* komunikat po wykonaniu akcji („Dokument usunięty”, „Folder utworzony”),
* wzkaźniki ładowani podczas ładowania list dokumentów,
* efekty najechania na elementach klikalnych z delikatnymi animacjami.

Stopniowe ujawnianie minimalizuje złożoność interfejsu:

* zaawansowane opcje ukryte w menu „...” na każdym dokumencie,
* szczegółowe metadane pokazywane dopiero po kliknięciu „Pokaż więcej”,
* opcje administracyjne dostępne tylko dla użytkowników z odpowiednimi uprawnieniami.

## Implementacja aplikacji

Implementacja aplikacji stanowi proces przekształcenia zaprojektowanego interfejsu użytkownika oraz architektury systemu w funkcjonalne rozwiązanie programistyczne, które realizuje wszystkie zdefiniowane wymagania biznesowe przy zachowaniu wysokich standardów bezpieczeństwa, wydajności oraz utrzymywalności kodu. O ile poprzedni podrozdział koncentrował się na warstwie prezentacji oraz doświadczeniu użytkownika, implementacja aplikacji dotyczy logiki biznesowej, mechanizmów bezpieczeństwa oraz integracji wszystkich komponentów systemu w spójną całość.

### Tworzenie systemu logowania i rejestracji

System logowania i rejestracji w aplikacji webowej do zarządzania dokumentami został zaprojektowany z myślą o bezpieczeństwie, elastyczności oraz łatwości obsługi. Implementacja opiera się na frameworku Django z dodatkowymi rozszerzeniami zapewniającymi zaawansowane funkcjonalności zarządzania użytkownikami i sesjami.

Architektura systemu uwierzytelniania

System uwierzytelniania aplikacji wykorzystuje wbudowane mechanizmy Django rozszerzone o niestandardowe komponenty. Podstawą jest model User z Django.contrib.auth, wzbogacony o dodatkowy model UserProfile zawierający rozszerzone informacje o użytkowniku oraz system ról. Architektura zapewnia separację danych uwierzytelniania od metadanych użytkownika, co ułatwia zarządzanie i rozbudowę systemu.

Implementacja niestandardowego formularza logowania

Aplikacja wykorzystuje niestandardowy formularz EmailAuthenticationForm, który umożliwia logowanie za pomocą adresu email zamiast tradycyjnej nazwy użytkownika. Takie podejście jest bardziej intuicyjne dla użytkowników końcowych i zwiększa bezpieczeństwo poprzez eliminację potrzeby zapamiętywania dodatkowej nazwy użytkownika. Poniżej przedstawiono formularz umożliwiający logowanie za pomocą adresu email zamiast nazwy użytkownika.

class EmailAuthenticationForm(AuthenticationForm):

username = forms.EmailField(

label='Email',

widget=forms.EmailInput(attrs={

'class': 'form-control',

'placeholder': 'Adres email',

'autofocus': True

})

)

def clean\_username(self):

email = self.cleaned\_data.get('username')

try:

user = User.objects.get(email=email)

return user.username

except User.DoesNotExist:

raise ValidationError('Nie znaleziono użytkownika z tym adresem email.')

Formularz zawiera zaawansowaną walidację sprawdzającą istnienie użytkownika o podanym adresie email oraz jego status aktywności. W przypadku próby logowania nieaktywnego konta, system wyświetla odpowiedni komunikat i uniemożliwia dostęp do aplikacji.

System zarządzania sesjami użytkowników

Aplikacja implementuje rozszerzony system śledzenia sesji poprzez model UserSession, który rejestruje szczegółowe informacje o każdej sesji użytkownika. System przechowuje klucz sesji, adres IP, informacje o przeglądarce (User-Agent), czas logowania i wylogowania oraz status aktywności sesji. Poniżej przedstawiono widok obsługujący proces logowania z tworzeniem sesji użytkownika i komunikatami powitalnym.

class CustomLoginView(LoginView):

form\_class = EmailAuthenticationForm

template\_name = 'users/login.html'

def form\_valid(self, form):

response = super().form\_valid(form)

UserSession.objects.create(

user=self.request.user,

session\_key=self.request.session.session\_key,

ip\_address=self.get\_client\_ip(),

user\_agent=self.request.META.get('HTTP\_USER\_AGENT', '')[:500]

)

messages.success(

self.request,

f'Witaj, {self.request.user.get\_full\_name() or self.request.user.email}!'

)

return response

Mechanizm śledzenia sesji umożliwia administratorom monitorowanie aktywności użytkowników oraz wykrywanie potencjalnych zagrożeń bezpieczeństwa, takich jak próby logowania z nietypowych lokalizacji czy jednoczesne sesje z różnych urządzeń.

Zabezpieczenia i walidacja haseł

System implementuje zaawansowany mechanizm walidacji haseł poprzez CustomPasswordChangeForm, który wymusza tworzenie silnych haseł spełniających określone kryteria bezpieczeństwa. Formularz sprawdza długość hasła (minimum 8 znaków), obecność wielkich i małych liter, cyfr oraz znaków specjalnych. Poniżej przedstawiono metodę walidującą siłę hasła sprawdzając długość, wielkie/małe litery, cyfry i znaki specjalne.

def \_is\_strong\_password(self, password):

if len(password) < 8:

return False

if not re.search(r'[a-z]', password):

return False

if not re.search(r'[A-Z]', password):

return False

if not re.search(r'\d', password):

return False

if not re.search(r'[!@#$%^&\*(),.?”:{}|<>]', password):

return False

return True

Dodatkowo system weryfikuje, czy nowe hasło nie jest identyczne z poprzednim, nie zawiera danych osobowych użytkownika oraz nie wykorzystuje popularnych wzorców takich jak „password123” czy sekwencje klawiaturowe. Wszystkie zmiany haseł są automatycznie logowane w systemie ActivityLog z zapisem adresu IP i znacznika czasowego.

Widoki i URL routing dla uwierzytelniania

System routing URL aplikacji został zaprojektowany z myślą o intuicyjności oraz zgodności z najlepszymi praktykami bezpieczeństwa. Główne ścieżki uwierzytelniania obejmują /login/, /logout/ oraz /password/change/, które przekierowują do odpowiednich widoków aplikacji.

CustomLogoutView implementuje bezpieczny proces wylogowania z automatycznym aktualizowaniem statusu sesji oraz przekierowaniem na stronę logowania. System zapewnia również proper handling komunikatów flash, informujących użytkownika o statusie operacji logowania/wylogowania.

### Obsługa operacji na dokumentach

System obsługi operacji na dokumentach stanowi rdzeń funkcjonalności aplikacji webowej do zarządzania dokumentami. Implementacja opiera się na wzorcu Model-View-Controller frameworka Django, zapewniając pełny zakres operacji CRUD (Create, Read, Update, Delete) z zaawansowanymi mechanizmami bezpieczeństwa, walidacji oraz optymalizacji wydajności.

Implementacja operacji tworzenia dokumentów (Create)

Proces dodawania nowych dokumentów rozpoczyna się od zaawansowanego formularza DocumentUploadForm, który implementuje wielopoziomową walidację plików. System automatycznie sprawdza rozmiar pliku (maksimum 50MB), dozwolone rozszerzenia oraz potencjalne zagrożenia bezpieczeństwa, blokując pliki wykonywalne i skrypty. Poniżej przedstawiono funkcje obsługująca przesyłanie nowych dokumentów z kontrolą uprawnień i automatycznym nadawaniem dostępu.

def document\_upload(request, folder\_id=None):

target\_folder = None

if folder\_id:

target\_folder = get\_object\_or\_404(Folder, pk=folder\_id)

if not user\_can\_view\_folder(request.user, target\_folder):

raise PermissionDenied(„Nie masz uprawnień do tego folderu.”)

if not user\_can\_create\_document(request.user):

messages.error(request, „Nie masz uprawnień do dodawania dokumentów.”)

return redirect('documents:home')

if request.method == 'POST':

form = DocumentUploadForm(request.POST, request.FILES, user=request.user)

if form.is\_valid():

document = form.save(commit=False)

document.wlasciciel = request.user

if target\_folder:

document.folder = target\_folder

document.save()

form.save\_m2m()

# Nadanie automatycznych uprawnień właścicielowi

assign\_perm('browse\_document', request.user, document)

assign\_perm('change\_document', request.user, document)

assign\_perm('delete\_document', request.user, document)

Podczas przesyłania system automatycznie generuje unikalną nazwę pliku używając UUID, oblicza hash SHA-256 dla weryfikacji integralności oraz organizuje pliki w strukturze katalogów według właściciela i daty utworzenia. Właściciel dokumentu automatycznie otrzymuje pełny zestaw uprawnień, które może później zarządzać.

System walidacji i bezpieczeństwa plików

Aplikacja implementuje zaawansowany system walidacji plików na poziomie formularza DocumentUploadForm. Walidacja obejmuje sprawdzanie rozmiaru pliku, weryfikację rozszerzenia oraz blokowanie potencjalnie niebezpiecznych typów plików. Poniżej przedstawiono metodę walidującą przesyłane pliki pod kątem rozmiaru, rozszerzenia i bezpieczeństwa.

def clean\_plik(self):

plik = self.cleaned\_data.get('plik')

if not plik:

raise ValidationError(„Musisz wybrać plik do wgrania.”)

# Sprawdzanie rozmiaru pliku (50MB)

max\_size = 50 \* 1024 \* 1024

if plik.size > max\_size:

raise ValidationError(f”Plik jest za duży! Maksymalny rozmiar to 50MB.”)

# Walidacja rozszerzenia

allowed\_extensions = ['pdf', 'docx', 'doc', 'xlsx', 'xls', 'txt', 'png', 'jpg', 'jpeg']

ext = os.path.splitext(plik.name)[1].lower().lstrip('.')

if ext not in allowed\_extensions:

raise ValidationError(f”Nieobsługiwany format pliku. Dozwolone: {', '.join(allowed\_extensions)}”)

# Blokowanie niebezpiecznych plików

dangerous\_extensions = ['exe', 'bat', 'cmd', 'com', 'scr', 'vbs', 'js']

if ext in dangerous\_extensions:

raise ValidationError(„Ze względów bezpieczeństwa, ten typ pliku nie jest dozwolony.”)

return plik

System dodatkowo implementuje mechanizm automatycznego nadawania nazw dokumentom na podstawie nazwy pliku, czyszczenia metadanych oraz sprawdzania czy użytkownik ma uprawnienia do wybranego folderu docelowego.

Operacje odczytu i przeglądania dokumentów (Read)

Funkcjonalność przeglądania dokumentów została zaimplementowana z uwzględnieniem złożonego systemu uprawnień oraz optymalizacji wydajności. Widok document\_detail wykorzystuje prefetch\_related i select\_related do minimalizacji liczby zapytań do bazy danych. Poniżej przedstawiono widok wyświetlający szczegóły dokumentu z optymalizacją zapytań i logowaniem aktywności.

def document\_detail(request, pk):

document = get\_object\_or\_404(

Document.objects.prefetch\_related(

'tagi', 'wlasciciel\_\_profile', 'folder',

'wersje\_\_utworzony\_przez\_\_profile',

Prefetch('komentarze', queryset=Comment.objects.filter(

rodzic=None, aktywny=True

).select\_related('uzytkownik\_\_profile').order\_by('data\_utworzenia')),

'metadane'

),

pk=pk, usunieto=False

)

if not user\_can\_view\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do tego dokumentu.”)

# Logowanie aktywności przeglądania

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='pobieranie',

dokument=document,

szczegoly=f”Wyświetlenie dokumentu {document.nazwa}”,

adres\_ip=get\_client\_ip(request)

)

System automatycznie loguje każdą operację przeglądania dokumentu, przechowując informacje o użytkowniku, czasie dostępu oraz adresie IP, co umożliwia pełny audyt aktywności w systemie.

Implementacja pobierania i podglądu dokumentów

Aplikacja oferuje dwa tryby dostępu do plików: pobieranie (download) oraz podgląd w przeglądarce (preview). Funkcja document\_download implementuje bezpieczne pobieranie plików z kontrolą uprawnień oraz logowaniem aktywności. Poniżej przedstawiono funkcje umożliwiającą bezpieczne pobieranie plików z kontrolą uprawnień i logowaniem.

def document\_download(request, pk):

document = get\_object\_or\_404(Document, pk=pk, usunieto=False)

if not user\_can\_view\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do pobrania tego dokumentu.”)

if not document.plik:

raise Http404(„Plik nie został znaleziony.”)

try:

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='pobieranie',

dokument=document,

szczegoly=f”Pobranie dokumentu {document.nazwa}”,

adres\_ip=get\_client\_ip(request)

)

response = FileResponse(

document.plik.open('rb'),

as\_attachment=True,

filename=document.nazwa

)

return response

except FileNotFoundError:

messages.error(request, „Plik dokumentu nie został znaleziony na serwerze.”)

return redirect('documents:document\_detail', pk=document.pk)

Funkcja preview wykorzystuje podobną logikę, ale konfiguruje nagłówek Content-Disposition na 'inline', sugerując przeglądarce wyświetlenie pliku zamiast pobierania, o ile typ MIME jest obsługiwany.

Operacje aktualizacji dokumentów (Update)

System aktualizacji documentów został podzielony na dwie kategorie: edycję metadanych oraz aktualizację zawartości pliku. DocumentUpdateForm umożliwia modyfikację nazwy, opisu, folderu, tagów oraz statusu dokumentu bez wpływu na główny plik. Poniżej przedstawiono widok edycji metadanych dokumentu z walidacją uprawnień użytkownika.

def document\_edit(request, pk):

document = get\_object\_or\_404(Document.objects.select\_related('wlasciciel\_\_profile'),

pk=pk, usunieto=False)

if not user\_can\_edit\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do edycji tego dokumentu.”)

if request.method == 'POST':

form = DocumentUpdateForm(request.POST, instance=document, user=request.user)

if form.is\_valid():

form.save()

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='edycja',

dokument=document,

szczegoly=f”Edytowano metadane dokumentu {document.nazwa}”,

adres\_ip=get\_client\_ip(request)

)

messages.success(request, f'Dokument „{document.nazwa}” został zaktualizowany.')

return redirect('documents:document\_detail', pk=pk)

Formularz automatycznie filtruje dostępne foldery w zależności od uprawnień użytkownika oraz implementuje walidację nazwy dokumentu, sprawdzając długość oraz obecność niebezpiecznych znaków.

System tagowania i metadanych

Aplikacja implementuje elastyczny system tagowania dokumentów oraz przechowywania niestandardowych metadanych. Tagi są zarządzane przez związek wiele do wielu, umożliwiając przypisanie wielu tagów do jednego dokumentu oraz wykorzystanie tego samego tagu przez wiele dokumentów.

Metadane przechowywane są w modelu DocumentMetadata jako pary klucz-wartość, umożliwiając dodawanie dowolnych informacji do dokumentów bez konieczności modyfikacji schematu bazy danych. System automatycznie zapewnia unikalność kombinacji dokument-klucz.

Operacje usuwania dokumentów (Delete)

Aplikacja implementuje strategię „soft delete”, gdzie dokumenty nie są fizycznie usuwane z bazy danych, lecz oznaczane flagą usunieto=True. Takie podejście zapewnia możliwość odzyskania danych oraz utrzymanie integralności historii aktywności. Poniżej przedstawiono funkcje implementującą „soft delete” dokumentów z flagą usunieto=True.

def document\_delete(request, pk):

document = get\_object\_or\_404(Document.objects.select\_related('wlasciciel\_\_profile', 'folder'),

pk=pk, usunieto=False)

if not user\_can\_delete\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do usunięcia tego dokumentu.”)

if request.method == 'POST':

document\_name = document.nazwa

folder\_id = document.folder.id if document.folder else None

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='usuniecie',

szczegoly=f”Usunięto dokument {document\_name}”,

adres\_ip=get\_client\_ip(request)

)

document.usunieto = True

document.save()

messages.success(request, f'Dokument „{document\_name}” został usunięty.')

return redirect('documents:folder\_view', folder\_id=folder\_id) if folder\_id else redirect('documents:home')

System usuwania implementuje również formularz potwierdzenia z szczegółowymi informacjami o dokumencie oraz ostrzeżeniami o nieodwracalności operacji, minimalizując ryzyko przypadkowego usunięcia ważnych plików.

Wyszukiwanie i filtrowanie dokumentów

Aplikacja oferuje zaawansowany system wyszukiwania dokumentów z możliwością filtrowania według nazwy, tagów, typu pliku oraz folderu. Funkcja search\_documents implementuje bezpieczne wyszukiwanie z uwzględnieniem uprawnień użytkownika. Poniżej przedstawiono endpoint wyszukiwania dokumentów z filtrowaniem według uprawnień użytkownika.

def search\_documents(request):

query = request.GET.get('q', '')

results\_data = []

if len(query) >= 2:

allowed\_documents = get\_objects\_for\_user(

request.user,

'documents.browse\_document',

klass=Document

).filter(usunieto=False)

searched\_documents = allowed\_documents.filter(

Q(nazwa\_\_icontains=query) |

Q(opis\_\_icontains=query) |

Q(tagi\_\_nazwa\_\_icontains=query)

).distinct()[:10]

for doc in searched\_documents:

results\_data.append({

'id': doc.id,

'nazwa': doc.nazwa,

'typ\_pliku': doc.typ\_pliku,

'rozmiar': doc.get\_file\_size\_display(),

'url': reverse('documents:document\_detail', args=[doc.id]),

'icon': doc.get\_file\_icon(),

'folder\_path': doc.folder.get\_full\_path() if doc.folder else „Główny”,

})

return JsonResponse({'results': results\_data})

System wyszukiwania wykorzystuje Django ORM z optymalizacjami distinct() oraz ograniczeniem wyników, zapewniając szybką odpowiedź nawet przy dużych zbiorach dokumentów. Wyniki są automatycznie filtrowane według uprawnień użytkownika, wykorzystując bibliotekę Django Guardian.

Implementacja operacji na dokumentach zapewnia kompleksową funkcjonalność zarządzania plikami z zachowaniem wysokich standardów bezpieczeństwa, wydajności oraz łatwości użytkowania. System jest przygotowany na skalowanie oraz dalszą rozbudowę o dodatkowe funkcjonalności.

### Udostępnianie plików i kontrola uprawnień

System udostępniania plików i kontroli uprawnień stanowi kluczowy element bezpieczeństwa aplikacji webowej do zarządzania dokumentami. Implementacja łączy tradycyjny system ról Django z zaawansowanymi mechanizmami kontroli dostępu na poziomie obiektów, wykorzystując bibliotekę Django Guardian do precyzyjnego zarządzania uprawnieniami do poszczególnych dokumentów i folderów.

Architektura systemu uprawnień

System uprawnień aplikacji opiera się na trzech poziomach kontroli dostępu: role systemowe, uprawnienia na poziomie obiektów oraz właścicielstwo zasobów. Każdy użytkownik posiada przypisaną rolę systemową (Administrator, Edytor, Czytelnik), która determinuje podstawowe możliwości w aplikacji, podczas gdy uprawnienia na poziomie obiektów pozwalają na granularne zarządzanie dostępem do konkretnych dokumentów. Poniżej przedstawiono model definiujący role systemowe (Administrator, Edytor, Czytelnik) z wyborem i opisem.

class Role(models.Model):

ADMIN = 'administrator'

EDITOR = 'edytor'

READER = 'czytelnik'

ROLE\_CHOICES = [

(ADMIN, 'Administrator'),

(EDITOR, 'Edytor'),

(READER, 'Czytelnik'),

]

nazwa = models.CharField(max\_length=50, choices=ROLE\_CHOICES, unique=True)

opis = models.TextField(blank=True)

Model UserProfile rozszerza standardowy model User o informacje o roli oraz statusie aktywności, automatycznie tworzony poprzez sygnały Django przy rejestracji nowego użytkownika z domyślną rolą „czytelnik”.

Implementacja funkcji kontroli uprawnień

Aplikacja implementuje zestaw dedykowanych funkcji sprawdzających uprawnienia użytkowników do wykonywania różnych operacji na dokumentach. Funkcje te centralnie zarządzają logiką bezpieczeństwa, zapewniając spójność kontroli dostępu w całej aplikacji, poniżej przedstawiono te funkcje.

def user\_can\_view\_document(user, document):

„„„Sprawdza czy użytkownik może przeglądać konkretny dokument.”„„

if not user.is\_authenticated:

return False

# Administratorzy widzą wszystko

if user.is\_superuser or (hasattr(user, 'profile') and user.profile.is\_admin):

return True

# Właściciel zawsze może przeglądać swoje dokumenty

if document.wlasciciel == user:

return True

# Dla wszystkich innych - tylko jawnie nadane uprawnienia

return user.has\_perm('browse\_document', document)

def user\_can\_edit\_document(user, document):

„„„Sprawdza czy użytkownik może edytować dokument.”„„

if not user.is\_authenticated:

return False

if user.is\_superuser or (hasattr(user, 'profile') and user.profile.is\_admin):

return True

# Właściciel-edytor może edytować swoje dokumenty

if (document.wlasciciel == user and hasattr(user, 'profile') and

(user.profile.is\_admin or user.profile.is\_editor)):

return True

# Edytor z jawnym uprawnieniem change\_document

if (hasattr(user, 'profile') and user.profile.is\_editor and

user.has\_perm('change\_document', document)):

return True

return False

System implementuje hierarchiczną kontrolę dostępu, gdzie administratorzy mają pełen dostęp, właściciele dokumentów posiadają rozszerzone uprawnienia (w zależności od swojej roli), a pozostali użytkownicy wymagają jawnego nadania uprawnień.

System udostępniania dokumentów

Mechanizm udostępniania dokumentów umożliwia właścicielom oraz administratorom nadawanie różnych poziomów dostępu innym użytkownikom systemu. Model DocumentShare śledzi wszystkie udostępnienia z informacjami o nadającym, odbiorcy, typie uprawnień oraz czasie wygaśnięcia. Poniżej przedstawiono model śledzący udostępnienia dokumentów z typami uprawnień i datami wygaśnięcia.

class DocumentShare(models.Model):

SHARE\_PERMISSION\_CHOICES = [

('documents.browse\_document', 'Tylko przeglądanie'),

('documents.download\_document', 'Przeglądanie i pobieranie'),

('documents.comment\_document', 'Przeglądanie i komentowanie'),

('documents.change\_document', 'Pełne uprawnienia do edycji'),

]

dokument = models.ForeignKey(Document, on\_delete=models.CASCADE, related\_name='document\_shares')

udostepnione\_przez = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='made\_shares')

udostepnione\_dla = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='received\_shares')

uprawnienie = models.CharField(max\_length=100, choices=SHARE\_PERMISSION\_CHOICES)

data\_udostepnienia = models.DateTimeField(auto\_now\_add=True)

data\_wygasniecia = models.DateTimeField(null=True, blank=True)

aktywne = models.BooleanField(default=True)

Funkcja share\_document\_with\_user implementuje logikę udostępniania z automatycznym zarządzaniem istniejącymi uprawnieniami oraz logowaniem aktywności. Poniżej przedstawiono funkcje udostępniającą dokument z automatycznym zarządzaniem istniejącymi uprawnieniami.

def share\_document\_with\_user(document, from\_user, to\_user\_obj, permission\_level='browse\_document'):

„„„Udostępnia dokument innemu użytkownikowi z określonym poziomem uprawnień.”„„

# Usunięcie istniejących uprawnień do tego dokumentu

all\_doc\_permissions = [

'browse\_document', 'change\_document', 'delete\_document',

'share\_document', 'download\_document', 'comment\_document'

]

for perm in all\_doc\_permissions:

if to\_user\_obj.has\_perm(perm, document):

remove\_perm(perm, to\_user\_obj, document)

# Nadanie nowego uprawnienia

assign\_perm(permission\_level, to\_user\_obj, document)

Automatyczne nadawanie uprawnień właścicielom

System automatycznie nadaje pełny zestaw uprawnień właścicielowi dokumentu podczas jego tworzenia. Proces ten zapewnia, że twórca dokumentu ma natychmiastowy dostęp do wszystkich funkcji zarządzania swoim plikiem. Poniżej przedstawiono część kodu automatycznie nadająca pełne uprawnienia właścicielowi nowego dokumentu.

def document\_upload(request, folder\_id=None):

if request.method == 'POST':

form = DocumentUploadForm(request.POST, request.FILES, user=request.user)

if form.is\_valid():

document = form.save(commit=False)

document.wlasciciel = request.user

document.save()

form.save\_m2m()

# Automatyczne nadanie uprawnień właścicielowi

assign\_perm('browse\_document', request.user, document)

assign\_perm('change\_document', request.user, document)

assign\_perm('delete\_document', request.user, document)

assign\_perm('share\_document', request.user, document)

assign\_perm('download\_document', request.user, document)

assign\_perm('comment\_document', request.user, document)

Kontrola dostępu w widokach aplikacji

Wszystkie widoki aplikacji implementują kontrolę dostępu na początku przetwarzania żądania, wykorzystując funkcje sprawdzające uprawnienia oraz rzucając wyjątek PermissionDenied w przypadku braku odpowiednich uprawnień. Poniżej przedstawiono widok z kontrolą dostępu i przekazaniem informacji o uprawnieniach do szablonu.

@login\_required

def document\_detail(request, pk):

document = get\_object\_or\_404(Document.objects.prefetch\_related(...), pk=pk, usunieto=False)

if not user\_can\_view\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do tego dokumentu.”)

# Sprawdzenie uprawnień do różnych akcji dla szablonu

context = {

'document': document,

'can\_edit\_this\_document': user\_can\_edit\_document(request.user, document),

'can\_delete\_this\_document': user\_can\_delete\_document(request.user, document),

'can\_download\_this\_document': user\_can\_view\_document(request.user, document),

'can\_comment\_on\_this\_document': user\_can\_comment\_on\_document(request.user, document),

'can\_share\_this\_document': user\_can\_share\_document(request.user, document),

}

return render(request, 'documents/document\_detail.html', context)

Widoki przekazują informacje o uprawnieniach do szablonów, umożliwiając dynamiczne wyświetlanie dostępnych akcji w interfejsie użytkownika.

Zarządzanie uprawnieniami przez administratorów

Administratorzy systemu posiadają dedykowane funkcje do masowego zarządzania uprawnieniami, umożliwiające efektywne nadawanie dostępu grupom użytkowników oraz odbieranie uprawnień w przypadku zmian organizacyjnych. Poniżej przedstawiono funkcja dla administratorów do masowego nadawania uprawnień do dokumentów.

def admin\_grant\_document\_access(admin\_user, target\_user, document, permissions=['browse\_document']):

„„„Funkcja dla administratora do nadawania uprawnień do dokumentu.”„„

if not (admin\_user.is\_superuser or (hasattr(admin\_user, 'profile') and admin\_user.profile.is\_admin)):

raise PermissionError(„Tylko administrator może nadawać uprawnienia do dokumentów.”)

for permission in permissions:

assign\_perm(permission, target\_user, document)

return True

System uprawnień do folderów

Aplikacja rozszerza mechanizm uprawnień również na foldery, umożliwiając kontrolę dostępu do całych katalogów oraz hierarchiczne dziedziczenie uprawnień. Poniżej przedstawiono funkcja sprawdzająca uprawnienia dostępu do folderów z hierarchicznym dziedziczeniem.

def user\_can\_view\_folder(user, folder):

„„„Sprawdza czy użytkownik może przeglądać konkretny folder.”„„

if not user.is\_authenticated:

return False

if user.is\_superuser or (hasattr(user, 'profile') and user.profile.is\_admin):

return True

if folder.wlasciciel == user:

return True

return user.has\_perm('browse\_folder', folder)

System folderów implementuje podobną logikę jak dokumenty, z dodatkowymi ograniczeniami dotyczącymi tworzenia podfolderów oraz przenoszenia dokumentów między katalogami.

Walidacja uprawnień w formularzach

Formularze aplikacji automatycznie filtrują dostępne opcje w zależności od uprawnień użytkownika, zapobiegając próbom przekroczenia autoryzacji poprzez manipulację danych formularza. Poniżej przedstawiono formularz filtrujący dostępne foldery według uprawnień użytkownika.

class DocumentUpdateForm(forms.ModelForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

user = kwargs.pop('user', None)

super().\_\_init\_\_(\*args, \*\*kwargs)

if user:

if user.is\_superuser or (hasattr(user, 'profile') and user.profile.is\_admin):

self.fields['folder'].queryset = Folder.objects.all()

else:

self.fields['folder'].queryset = Folder.objects.filter(wlasciciel=user)

Logowanie aktywności uprawnień

Każda operacja związana z uprawnieniami jest automatycznie rejestrowana w systemie ActivityLog, umożliwiając pełny audyt działań związanych z udostępnianiem dokumentów oraz zmianami uprawnień. Poniżej przedstawiono funkcje sprawdzającą czy użytkownik może komentować dokument z logowaniem dostępu.

def user\_can\_comment\_on\_document(user, document):

„„„Sprawdza czy użytkownik może komentować dokument z logowaniem dostępu.”„„

if not user.is\_authenticated:

return False

# Sprawdzenie podstawowego dostępu do przeglądania

if not user\_can\_view\_document(user, document):

return False

# Sprawdzenie statusu aktywności profilu

if hasattr(user, 'profile') and not user.profile.aktywny:

return False

# Logika uprawnień do komentowania

if user.is\_superuser or (hasattr(user, 'profile') and user.profile.is\_admin):

return True

if document.wlasciciel == user:

return True

return user.has\_perm('comment\_document', document)

System udostępniania plików i kontroli uprawnień zapewnia granularną kontrolę dostępu przy zachowaniu prostoty zarządzania dla użytkowników końcowych. Implementacja łączy bezpieczeństwo z użytecznością, umożliwiając efektywną współpracę nad dokumentami przy zachowaniu pełnej kontroli nad dostępem do informacji.

### Wersjonowanie dokumentów

System wersjonowania dokumentów stanowi kluczową funkcjonalność aplikacji webowej, umożliwiającą śledzenie zmian, przywracanie poprzednich wersji oraz zapewnienie integralności historii modyfikacji plików. Implementacja opiera się na modelu DocumentVersion, który przechowuje każdą wersję dokumentu jako oddzielny rekord z pełną historią zmian oraz metadanymi opisującymi modyfikacje.

Architektura systemu wersjonowania

Model DocumentVersion został zaprojektowany z myślą o przechowywaniu kompletnej historii zmian każdego dokumentu w systemie. Każda nowa wersja otrzymuje automatycznie inkrementowany numer, zachowuje referencję do oryginalnego dokumentu oraz przechowuje informacje o autorze zmian i komentarzu opisującym modyfikacje. Poniżej przedstawiono model przechowujący wersje dokumentów z automatyczną numeracją i unikalności.

class DocumentVersion(models.Model):

dokument = models.ForeignKey(Document, on\_delete=models.CASCADE, related\_name='wersje')

numer\_wersji = models.PositiveIntegerField()

data\_utworzenia = models.DateTimeField(auto\_now\_add=True)

utworzony\_przez = models.ForeignKey(User, on\_delete=models.CASCADE)

plik = models.FileField(upload\_to='document\_versions/%Y/%m/%d/', blank=True, null=True)

komentarz = models.TextField(blank=True)

rozmiar\_pliku = models.PositiveIntegerField(default=0, null=True, blank=True)

hash\_pliku = models.CharField(max\_length=64, blank=True)

class Meta:

unique\_together = ['dokument', 'numer\_wersji']

ordering = ['-numer\_wersji']

System automatycznie zapewnia unikalność kombinacji dokument-numer\_wersji oraz sortuje wersje w kolejności malejącej, gdzie najnowsza wersja zawsze pojawia się na początku listy. Każda wersja przechowuje własną kopię pliku w dedykowanej strukturze katalogów organizowanej według daty utworzenia.

Implementacja tworzenia nowych wersji

Proces dodawania nowej wersji dokumentu jest realizowany przez dedykowany widok document\_version\_upload, który automatycznie zarządza numeracją wersji oraz aktualizuje główny rekord dokumentu aby wskazywał na najnowszą wersję. Poniżej przedstawiono widok dodawania nowych wersji z automatyczną inkrementacją numerów i aktualizacją głównego dokumentu.

@login\_required

def document\_version\_upload(request, pk):

document = get\_object\_or\_404(Document.objects.select\_related('wlasciciel\_\_profile').prefetch\_related('wersje'),

pk=pk, usunieto=False)

if not user\_can\_edit\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do dodawania nowej wersji tego dokumentu.”)

if request.method == 'POST':

form = DocumentVersionUploadForm(request.POST, request.FILES)

if form.is\_valid():

# Automatyczne określenie numeru najnowszej wersji

latest\_version = document.wersje.order\_by('-numer\_wersji').first()

new\_version\_number = (latest\_version.numer\_wersji + 1) if latest\_version else 1

# Utworzenie nowej wersji dokumentu

version = DocumentVersion.objects.create(

dokument=document,

numer\_wersji=new\_version\_number,

utworzony\_przez=request.user,

plik=form.cleaned\_data['plik'],

komentarz=form.cleaned\_data['komentarz']

)

# Aktualizacja głównego rekordu dokumentu

document.plik = version.plik

document.rozmiar\_pliku = version.rozmiar\_pliku

document.save()

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='edycja',

dokument=document,

szczegoly=f”Utworzono wersję {new\_version\_number} dokumentu {document.nazwa}”,

adres\_ip=get\_client\_ip(request)

)

System automatycznie zarządza numeracją wersji, znajdując najwyższą istniejącą wersję i incrementując ją o jeden. Główny rekord dokumentu jest aktualizowany aby zawsze wskazywał na najnowszą wersję, podczas gdy wszystkie poprzednie wersje pozostają dostępne w historii.

Walidacja i bezpieczeństwo wersji

Formularz DocumentVersionUploadForm implementuje takie same mechanizmy walidacji jak główny formularz upload dokumentów, zapewniając spójność bezpieczeństwa w całym systemie wersjonowania. Poniżej przedstawiono formularz do przesyłania nowych wersji z walidacją rozmiaru i typu pliku.

class DocumentVersionUploadForm(forms.Form):

plik = forms.FileField(

label='Nowa wersja pliku',

validators=[FileExtensionValidator(

allowed\_extensions=['pdf', 'docx', 'doc', 'xlsx', 'xls', 'txt', 'png', 'jpg', 'jpeg']

)],

widget=forms.FileInput(attrs={

'class': 'form-control',

'accept': '.pdf,.docx,.doc,.xlsx,.xls,.txt,.png,.jpg,.jpeg'

})

)

komentarz = forms.CharField(

label='Komentarz do wersji',

required=False,

widget=forms.Textarea(attrs={

'class': 'form-control',

'rows': 3,

'placeholder': 'Opisz zmiany w tej wersji...'

})

)

def clean\_plik(self):

plik = self.cleaned\_data.get('plik')

if not plik:

raise ValidationError(„Musisz wybrać plik.”)

# Sprawdzenie rozmiaru pliku (50MB)

max\_size = 50 \* 1024 \* 1024

if plik.size > max\_size:

raise ValidationError(„Plik jest za duży! Maksymalny rozmiar to 50MB.”)

return plik

Automatyczne generowanie hash i metadanych

System wersjonowania automatycznie generuje hash SHA-256 dla każdej nowej wersji podczas zapisu, umożliwiając weryfikację integralności plików oraz wykrywanie przypadkowych uszkodzeń lub nieautoryzowanych modyfikacji. Poniżej przedstawiono metode automatycznie generująca hash SHA-256 i obliczająca rozmiar dla każdej wersji.

def save(self, \*args, \*\*kwargs):

if self.plik and hasattr(self.plik, 'size') and not self.rozmiar\_pliku:

self.rozmiar\_pliku = self.plik.size

if self.plik.file:

import hashlib

self.plik.seek(0)

file\_hash = hashlib.sha256()

for chunk in iter(lambda: self.plik.read(4096), b”„):

file\_hash.update(chunk)

self.hash\_pliku = file\_hash.hexdigest()

self.plik.seek(0)

super().save(\*args, \*\*kwargs)

Metoda save automatycznie oblicza rozmiar pliku oraz jego hash, zapewniając kompletność metadanych bez konieczności interwencji użytkownika.

Interfejs użytkownika dla zarządzania wersjami

Template document\_version\_upload.html implementuje zaawansowany interfejs użytkownika z podglądem pliku, porównaniem z aktualną wersją oraz historią zmian. Formularz zawiera mechanizmy przeciągnij i upuść oraz automatyczne sugerowanie komentarzy na podstawie nazwy pliku. Poniżej przedstawiono kod automatycznie sugerujący komentarze na podstawie nazwy przesyłanego pliku.

fileInput.addEventListener('change', function() {

const file = this.files[0];

if (file && !komentarzInput.value.trim()) {

const currentName = „{{ document.nazwa|escapejs }}”;

const newName = file.name;

// Heurystyki dla automatycznych komentarzy

if (newName.toLowerCase().includes('final')) {

komentarzInput.value = 'Wersja finalna';

} else if (newName.toLowerCase().includes('draft')) {

komentarzInput.value = 'Wersja robocza';

} else if (newName.toLowerCase().includes('review')) {

komentarzInput.value = 'Wersja do przeglądu';

} else if (newName !== currentName) {

komentarzInput.value = 'Zaktualizowana wersja dokumentu';

}

}

});

System automatycznie analizuje nazwę przesyłanego pliku i sugeruje odpowiedni komentarz, ułatwiając użytkownikom opisanie wprowadzonych zmian.

Wyświetlanie historii wersji

Aplikacja implementuje czytelny interfejs do przeglądania historii wersji w dokument\_detail.html, gdzie każda wersja jest prezentowana z informacjami o autorze, dacie utworzenia, komentarzu oraz rozmiarze pliku. Poniżej przedstawiono szablon wyświetlający historię wersji z metadanymi i wskaźnikami aktualnej wersji.

html{% for version in versions %}

<div class=„d-flex align-items-start mb-3 pb-3 {% if not forloop.last %}border-bottom{% endif %}”>

<div class=„flex-grow-1”>

<div class=„d-flex align-items-center mb-1”>

<h6 class=„mb-0 me-2”>Wersja {{ version.numer\_wersji }}</h6>

{% if forloop.first %}

<span class=„badge bg-success”>Aktualna</span>

{% endif %}

<small class=„text-muted ms-3”>{{ version.data\_utworzenia|date:”d.m.Y H:i” }}</small>

</div>

<div class=„mb-2”>

<small class=„text-muted”>

<i class=„bi bi-person me-1”></i>{{ version.utworzony\_przez.get\_full\_name|default:version.utworzony\_przez.email }}

{% if version.rozmiar\_pliku %}

<i class=„bi bi-hdd ms-3 me-1”></i>{{ version.get\_file\_size\_display }}

{% endif %}

</small>

</div>

{% if version.komentarz %}

<p class=„small mb-0 fst-italic text-muted”>

<i class=„bi bi-chat-quote me-1”></i>„{{ version.komentarz }}”

</p>

{% endif %}

</div>

</div>

{% endfor %}

Interfejs wykorzystuje wizualne wskaźniki dla rozróżnienia aktualnej wersji od historycznych, wyświetla metadane w uporządkowany sposób oraz podkreśla komentarze użytkowników opisujące zmiany.

Pobieranie konkretnych wersji

System implementuje dedykowaną funkcjonalność pobierania dowolnej wersji dokumentu poprzez widok document\_version\_download, który umożliwia dostęp do historycznych wersji plików. Poniżej przedstawiono funkcje pobierania konkretnych wersji dokumentów z generowaniem nazw zawierających numer wersji.

@login\_required

def document\_version\_download(request, document\_pk, version\_pk):

document = get\_object\_or\_404(Document, pk=document\_pk, usunieto=False)

version = get\_object\_or\_404(DocumentVersion, pk=version\_pk, dokument=document)

if not user\_can\_view\_document(request.user, document):

raise PermissionDenied(„Nie masz uprawnień do pobrania tego dokumentu.”)

if not version.plik:

raise Http404(„Plik wersji nie został znaleziony.”)

try:

ActivityLog.objects.create(

uzytkownik=request.user,

typ\_aktywnosci='pobieranie',

dokument=document,

szczegoly=f”Pobranie wersji {version.numer\_wersji} dokumentu {document.nazwa}”,

adres\_ip=get\_client\_ip(request)

)

# Generowanie nazwy pliku z numerem wersji

base\_name = os.path.splitext(document.nazwa)[0]

ext = os.path.splitext(document.nazwa)[1]

version\_filename = f”{base\_name}\_v{version.numer\_wersji}{ext}”

response = FileResponse(

version.plik.open('rb'),

as\_attachment=True,

filename=version\_filename

)

return response

except FileNotFoundError:

messages.error(request, „Plik wersji nie został znaleziony na serwerze.”)

return redirect('documents:document\_detail', pk=document.pk)

System automatycznie generuje nazwy plików zawierające numer wersji (np. „raport\_v2.pdf”), ułatwiając użytkownikom rozróżnienie pobranych wersji oraz loguje każde pobranie dla celów audytu.

Optymalizacje wydajności dla wersjonowania

Aplikacja implementuje optymalizacje zapytań do bazy danych dla operacji związanych z wersjami, wykorzystując prefetch\_related w widokach oraz select\_related dla powiązanych użytkowników. Poniżej przedstawiono fragment optymalizujący zapytania do bazy danych dla operacji na wersjach dokumentów.

document = get\_object\_or\_404(

Document.objects.prefetch\_related(

'wersje\_\_utworzony\_przez\_\_profile',

Prefetch('wersje', queryset=DocumentVersion.objects.select\_related('utworzony\_przez'))

),

pk=pk, usunieto=False

)

Ograniczenia i zabezpieczenia systemu wersjonowania

System wersjonowania implementuje te same ograniczenia bezpieczeństwa co główna funkcjonalność wgrywania dokumentów, blokując niebezpieczne typy plików oraz egzekwując limity rozmiaru. Dodatkowo, każda operacja na wersjach jest logowana z pełnymi informacjami o użytkowniku i czasie dostępu.

Mechanizm unique\_together na poziomie bazy danych zapobiega tworzeniu duplikatów wersji z tym samym numerem dla jednego dokumentu, podczas gdy automatyczna inkrementacja numerów eliminuje konflikty przy jednoczesnym dodawaniu wersji przez różnych użytkowników.

System wersjonowania dokumentów zapewnia kompleksową funkcjonalność śledzenia zmian przy zachowaniu wysokiej wydajności oraz bezpieczeństwa. Implementacja umożliwia łatwe zarządzanie historią dokumentów dla zespołów współpracujących nad tymi samymi plikami, zapewniając pełną transparentność procesu modyfikacji oraz możliwość przywracania poprzednich wersji w przypadku potrzeby.

# Konfiguracja i wdrożenie

Konfiguracja i wdrożenie aplikacji webowej stanowi krytyczny etap transformacji z środowiska deweloperskiego do w pełni funkcjonalnego systemu produkcyjnego, dostępnego dla użytkowników końcowych. Po zakończeniu fazy projektowania interfejsu użytkownika oraz implementacji logiki biznesowej, konieczne jest przygotowanie infrastruktury technicznej, która zapewni niezawodne, bezpieczne oraz wydajne działanie aplikacji do zarządzania dokumentami w warunkach rzeczywistego obciążenia produkcyjnego.

## Wymagania sprzętowe

Wymagania sprzętowe określają minimalną konfigurację serwera niezbędną do prawidłowego działania systemu. Parametry te zostały dobrane dla małej organizacji do 15 użytkowników pracujących równocześnie.

CPU: 2 rdzenie, 2.4 GHz

RAM: 4 GB

Dysk: 50 GB SSD

Sieć: 100 Mbps

## Wymagania systemowe

Wymagania systemowe obejmują wszystkie komponenty oprogramowania niezbędne do uruchomienia i obsługi aplikacji. Wybrane technologie zapewniają stabilność, bezpieczeństwo i skalowalność systemu.

Oprogramowanie

* Python: 3.9+
  + django-guardian: 2.4+
  + Pillow: 9.0+
* Django: 4.2+
* Baza danych: SQLite
* Serwer WWW: Nginx + Gunicorn
* System operacyjny: Linux Ubuntu 20.04+ (zalecane)

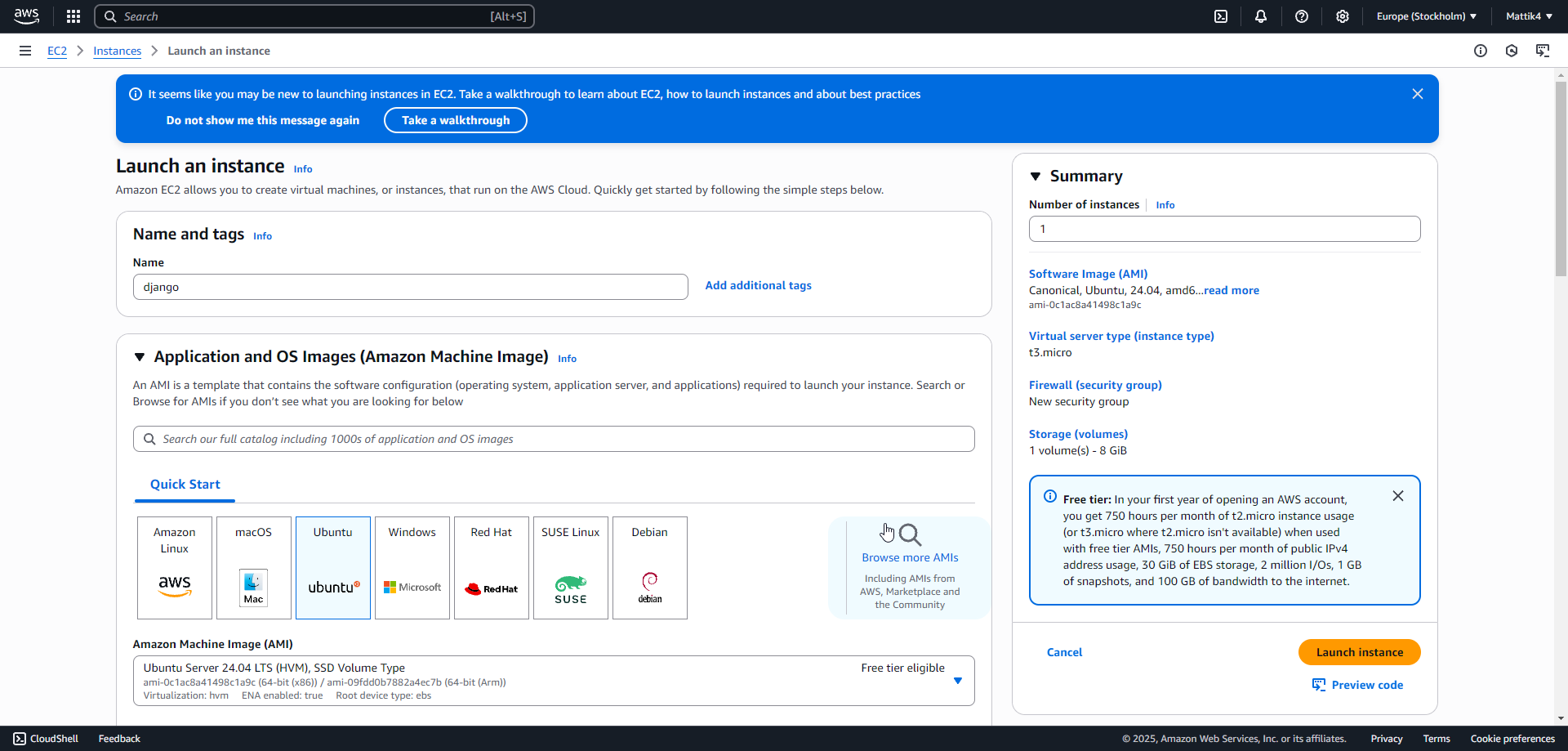
Przeglądarki

* Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

## Instalacja aplikacji

Przed rozpoczęciem procesu wdrożenia należy upewnić się, że wszystkie pliki aplikacji Django są zaktualizowane i przesłane do repozytorium GitHub. Następnie utworzono instancję EC2 w AWS z systemem operacyjnym Ubuntu 24.04 LTS oraz typem instancji t3.micro, który kwalifikuje się do bezpłatnego poziomu, co przedstawia rysunek 27.

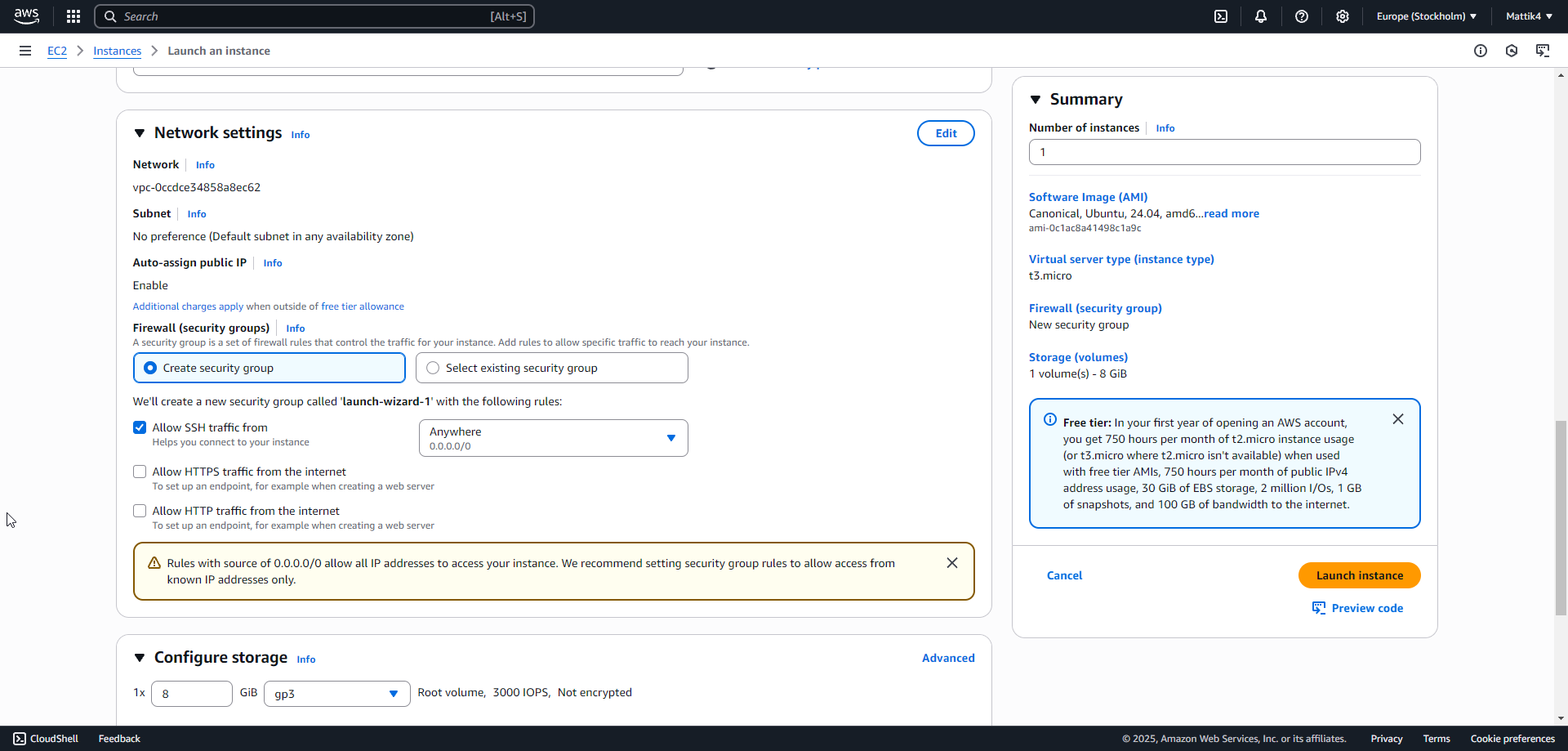
Rysunek . Tworzenie instancji



Źródło: opracowanie własne

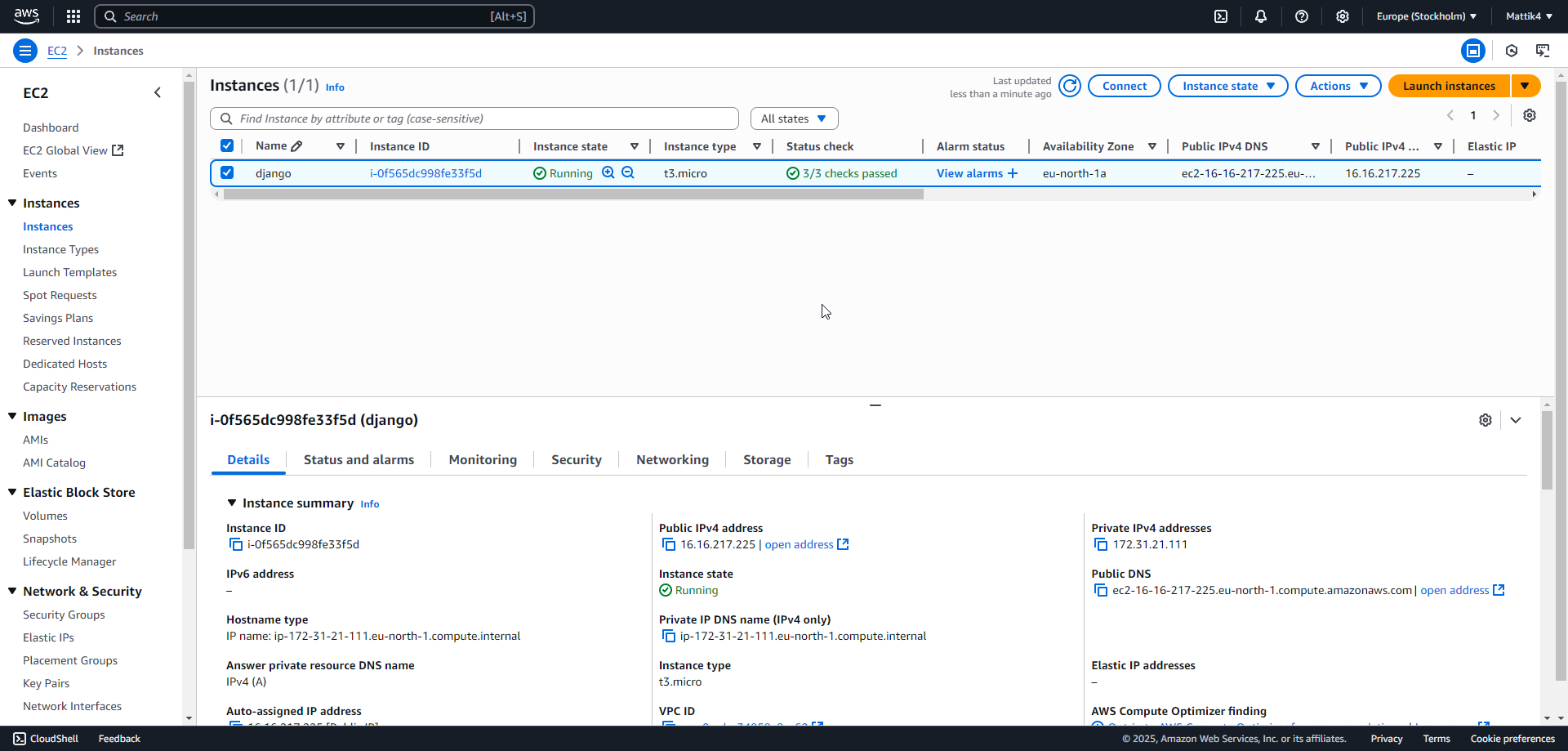
Konfiguracja sieci została ustawiona na domyślną VPC z automatycznym przypisaniem publicznego IP, storage na 8 GiB SSD, a także utworzono nową grupę Security Group z podstawowymi regułami dla SSH, co obrazuje rysunek 28. Rysunek 29 przedstawia utworzoną instancję.

Rysunek . Konfiguracja sieci



Źródło: opracowanie własne

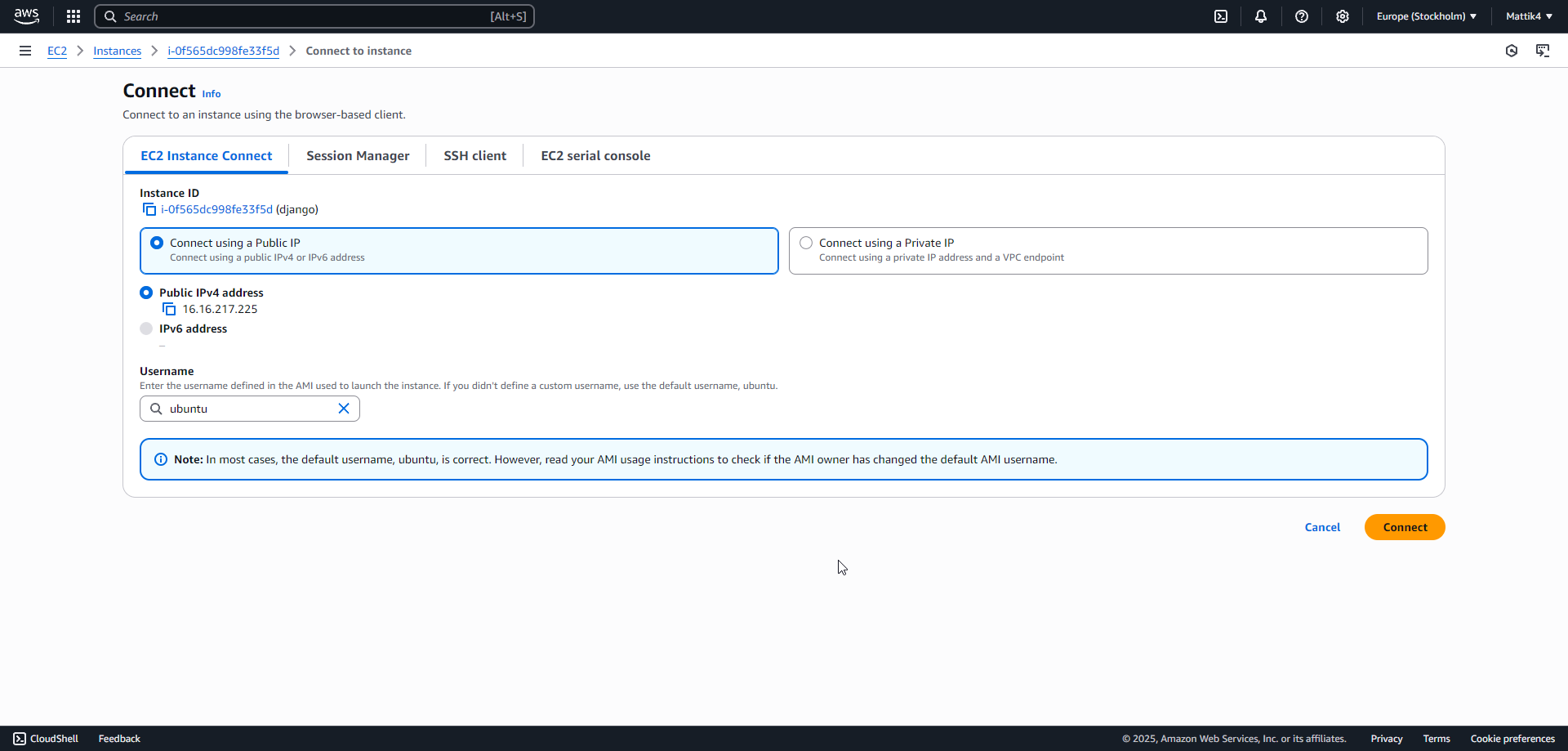
Rysunek 29. Utworzona instancja



Źródło: opracowanie własne

Połączenie z instancją realizowane jest za pomocą AWS EC2 Instance Connect przez interfejs webowy, używając domyślnego użytkownika Ubuntu, co widać na rysunku 30.

Rysunek . Połaczenie z instancją

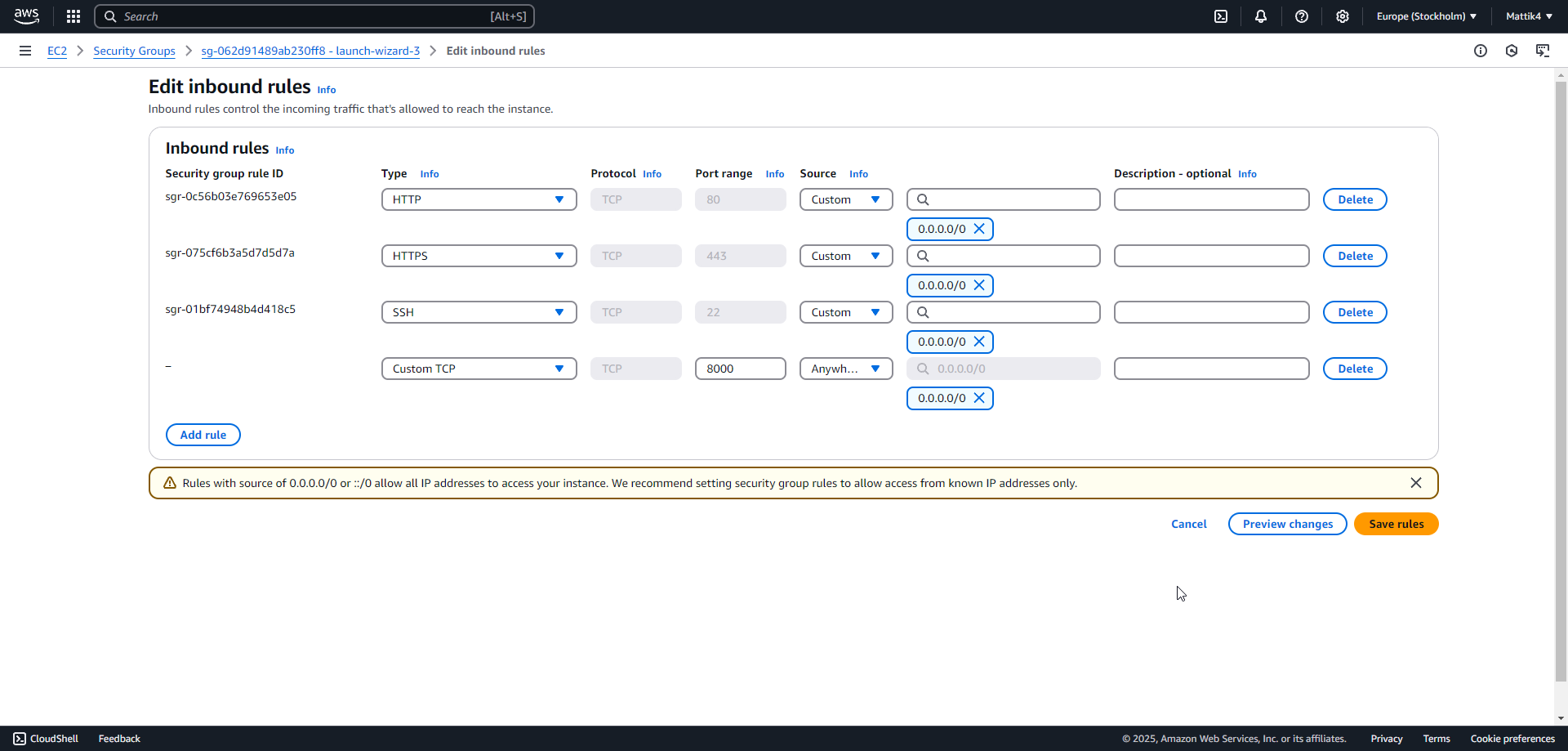


Źródło: opracowanie własne

Następnie:

1. Aktualizacja systemu
   * sudo apt-get update
2. Klonowanie repozytorium projektu
   * git clone [URL\_REPOZYTORIUM]
3. Instalacja pip dla Python 3
   * sudo apt install python3-pip -y
4. Utworzenie wirtualnego środowiska Python
   * python3 -m venv .venv
5. Aktywacja wirtualnego środowiska
   * source .venv/bin/activate
6. Instalacja zależności projektu
   * pip3 install -r requirements.txt
7. Konfiguracja dostępu sieciowego
   * Dodanie reguły w Security Group
   * Typ: Inbound Rule
   * Port: 8000
   * Źródło: 0.0.0.0/0 (dostęp z dowolnego adresu IP), prezentuje to rysunek 31.

Rysunek 31. Zmiana ustawień sieciowych



Źródło: opracowanie własne

W pliku settings.py dokonano jedynej niezbędnej zmiany, ustawiając ALLOWED\_HOSTS = ['\*'], co umożliwia dostęp do aplikacji z dowolnego adresu IP, co ukazuje rysunek 32.

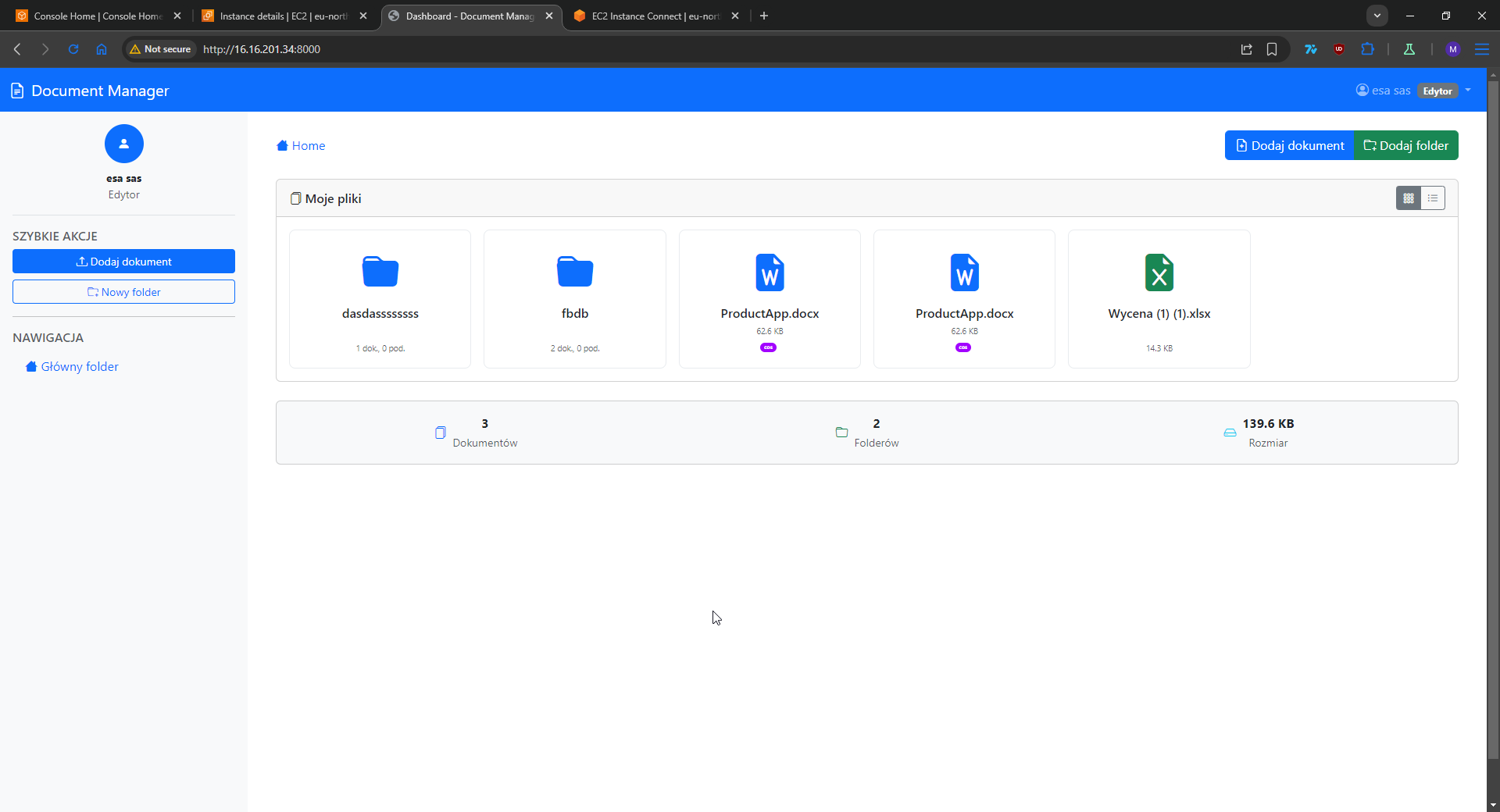
Rysunek . Zmiana settings.py



Źródło: opracowanie własne

Ostatnim krokiem było uruchomienie aplikacji za pomocą komendy python3 manage.py runserver 0.0.0.0:8000, po czym aplikacja stała się dostępna z internetu przez przeglądarkę internetową, co prezentuje rysunke 33.

Rysunek . Skończony deployment



Źródło: opracowanie własne

## Konfiguracja aplikacji

Po uruchomieniu podstawowej wersji aplikacji na AWS EC2, konieczne jest wdrożenie konfiguracji produkcyjnej obejmującej zmienne środowiskowe, serwer aplikacji Gunicorn, reverse proxy Nginx, zabezpieczenie połączeń HTTPS oraz konfigurację domeny i systemu logowania błędów.

Wszystkie przykłady w niniejszym rozdziale wykorzystują wartości demonstracyjne takie jak twoja-domena.com, twoj-email@gmail.com, [PUBLIC\_IP] itp. W rzeczywistym wdrożeniu należy zastąpić te wartości rzeczywistymi danymi odpowiednimi dla konkretnej organizacji.

Konfiguracja zmiennych środowiskowych

Pierwszym krokiem jest utworzenie pliku .env w katalogu głównym aplikacji, który zawiera wszystkie wrażliwe dane konfiguracyjne. Plik ten przechowuje kluczowe ustawienia takie jak:

* klucz tajny Django (SECRET\_KEY) - unikalny dla każdej instalacji,
* lista dozwolonych hostów (ALLOWED\_HOSTS) - zawierająca domenę organizacji,
* ustawienia email SMTP - skonfigurowane dla firmowego adresu email,
* parametry bezpieczeństwa SSL/TLS,
* ścieżki do plików statycznych i mediów.

Następnie plik settings.py Django został zmodyfikowany aby wykorzystywał te zmienne środowiskowe poprzez bibliotekę python-decouple, co umożliwia bezpieczne zarządzanie konfiguracją bez zapisywania wrażliwych danych w kodzie źródłowym.

Konfiguracja Gunicorn

Gunicorn zastępuje deweloperski serwer Django w środowisku produkcyjnym. Konfiguracja obejmuje:

* instalację i podstawową konfigurację,
* instalacja przez pip install gunicorn,
* utworzenie pliku gunicorn.conf.py z parametrami wydajności,
* konfiguracja liczby procesów roboczych (workers) na podstawie CPU,
* ustawienie timeout i limitów połączeń.

Utworzenie systemd service:

* plik /etc/systemd/system/document-manager.service,
* automatyczne uruchamianie przy starcie systemu,
* restart w przypadku awarii,
* uruchamianie jako użytkownik ubuntu dla bezpieczeństwa.

Kluczowe parametry konfiguracji Gunicorn to liczba workerów (CPU \* 2 + 1), timeout 30 sekund oraz bind na localhost:8000 dla współpracy z Nginx.

Konfiguracja Nginx

Nginx pełni rolę reverse proxy oraz serwera plików statycznych. Główne elementy konfiguracji:

Instalacja i konfiguracja podstawowa:

* Instalacja przez apt install nginx
* Utworzenie konfiguracji w /etc/nginx/sites-available/document-manager
* Przekierowanie HTTP na HTTPS
* Obsługa plików statycznych i mediów

Kluczowe funkcjonalności:

* Reverse proxy do Gunicorn (localhost:8000)
* Kompresja gzip dla lepszej wydajności
* Cache'owanie plików statycznych
* Limity upload (50MB dla dokumentów)
* Security headers (HSTS, XSS Protection, Content-Type)

Konfiguracja zawiera dwa bloki server - jeden dla HTTP (przekierowuje na HTTPS) oraz drugi dla HTTPS z pełną konfiguracją SSL.

Konfiguracja HTTPS z certyfikatem SSL

Implementacja bezpiecznych połączeń SSL/TLS wykorzystuje Let's Encrypt:

Proces konfiguracji:

* Instalacja Certbot przez snap
* Uzyskanie certyfikatu dla domeny organizacji
* Automatyczna konfiguracja odnowienia (cron job)
* Konfiguracja silnych szyfrów i protokołów TLS

Certbot automatycznie tworzy certyfikaty w /etc/letsencrypt/live/domena/ i konfiguruje automatyczne odnowienie co 12 godzin przez systemowy cron.

Konfiguracja domeny i DNS

Konfiguracja DNS wymaga utworzenia odpowiednich rekordów wskazujących na publiczny IP instancji EC2:

Wymagane rekordy DNS:

* Rekord A dla głównej domeny
* Rekord A dla subdomeny www
* Opcjonalnie rekord CNAME dla subdomen

W przypadku korzystania z AWS Route 53, konfiguracja może być wykonana przez AWS CLI lub konsolę webową. Dla innych dostawców DNS (home.pl, Cloudflare) konfiguracja odbywa się przez ich panele administracyjne.

Backup i zabezpieczenia danych

System backup obejmuje automatyczne kopie zapasowe bazy danych i plików mediów:

Komponenty systemu backup:

* Skrypt bash wykonujący kopie SQLite i plików media
* Kompresja archiwów tar.gz
* Rotacja starych backup (30 dni)
* Opcjonalna synchronizacja z AWS S3
* Logowanie operacji backup

Dodatkowe zabezpieczenia:

* Fail2ban dla ochrony przed atakami brute force
* Konfiguracja firewall (UFW)
* Regularne aktualizacje systemu
* Monitoring miejsca na dysku

Skrypt backup uruchamiany jest codziennie o 2:00 przez cron job i tworzy kopie w formacie db\_backup\_YYYYMMDD\_HHMMSS.sqlite3.

Logowanie błędów

Kompleksowy system logowania umożliwia monitorowanie aplikacji:

Konfiguracja logowania Django:

* Rotating file handlers dla logów aplikacji
* Oddzielne pliki dla błędów i informacji ogólnych
* Formatowanie logów z timestamp i szczegółami
* Automatyczna rotacja plików (15MB, 10 kopii)

Monitoring i alerting:

* Skrypt monitorujący częstość błędów
* Alerty email przy przekroczeniu progów
* Monitoring wykorzystania dysku
* Integracja z systemowym logrotate

Struktura logów:

* /home/ubuntu/document-manager/logs/django.log - logi ogólne
* /home/ubuntu/document-manager/logs/django\_error.log - tylko błędy
* /var/log/nginx/ - logi Nginx
* /home/ubuntu/document-manager/logs/gunicorn\_\*.log - logi Gunicorn

System automatycznie rotuje logi, kompresuje stare pliki i wysyła alerty email do administratora w przypadku wykrycia problemów.

## Przewodnik użytkownika

Przewodnik użytkownika opisuje sposób obsługi systemu zarządzania dokumentami dla wszystkich ról użytkowników: administratorów, edytorów oraz czytelników. System umożliwia bezpieczne przechowywanie, udostępnianie i zarządzanie dokumentami w organizacji z pełną kontrolą uprawnień oraz historią wersji.

### Logowanie do systemu

System jest dostępny przez przeglądarkę internetową pod adresem aplikacji. Obsługiwane przeglądarki to Chrome, Firefox, Safari oraz Edge w najnowszych wersjach.

Proces logowania(rysunek 34)

* Wprowadź swój adres email w polu „Adres email”
* Wpisz hasło w polu „Hasło”
* Kliknij przycisk „Zaloguj się”

System automatycznie przekieruje Cię do strony głównej po pomyślnym zalogowaniu.

Rysunek . Formularz logowania

Obraz zawierający tekst, zrzut ekranu, design

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

W przypadku problemów z logowaniem:

* Sprawdź poprawność adresu email i hasła
* Skontaktuj się z administratorem w przypadku dalszych problemów

### Strona główna

Po zalogowaniu wyświetla się główny dashboard aplikacji(rysunek 35) przypominający eksplorator plików. Interfejs składa się z:

* Paska nawigacyjnego - zawiera informacje o zalogowanym użytkowniku
* Obszaru głównego - lista dokumentów i folderów

Rysunek . Główna strona aplikacji

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Menu użytkownika dostosowuje się automatycznie do roli zalogowanego użytkownika:

* Administratorzy - pełny dostęp do wszystkich funkcji zarządzania systemem(rysunek 36)
* Edytorzy - dostęp do tworzenia i edycji dokumentów(rysunek 35)
* Czytelnicy - ograniczony interfejs do przeglądania dokumentów(rysunek 37)

Rysunek Strona główna administrator

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Rysunek 37. Strona główna czytelnik

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Widok listy dokumentów

System oferuje dwa główne widoki przeglądania dokumentów:

Widok siatki(rysunek 38)

* Dokumenty wyświetlane jako miniaturki z ikonami typu pliku
* Przydatny do szybkiego przeglądania dużej liczby dokumentów
* Podwójne kliknięcie otwiera dokument

Rysunek . Widok siatki

Obraz zawierający tekst, oprogramowanie, numer, Ikona komputerowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Widok listy(rysunek 39)

* Dokumenty w formie tabeli z szczegółowymi informacjami
* Kolumny: nazwa, typ, rozmiar, data modyfikacji, właściciel

Rysunek 39. Widok listy

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna. Źródło: opracowanie własne

Przełączanie między widokami

Użyj przycisków w prawym górnym rogu obszaru dokumentów(rysunek 40):

* Ikona siatki - przełącza na widok miniaturek
* Ikona listy - przełącza na widok tabeli

Rysunek . Ikonki do przełączani widoków

Obraz zawierający tekst, Prostokąt, zrzut ekranu, symbol

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Nawigacja w folderach(rysunek 41):

* Kliknij folder, aby go otworzyć
* Użyj „breadcrumb” (ścieżki) u góry, aby wrócić do wyższego poziomu

Rysunek . Foldery nawigacja

Obraz zawierający tekst, Czcionka, Grafika, biały

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Wyszukiwanie

Wyszukiwanie:

* Użyj pola wyszukiwania w górnej części interfejsu(rysunek 42)
* Wpisz nazwę dokumentu lub fragment nazwy
* Wyniki wyświetlą się automatycznie podczas pisania

Rysunek . Pole wyszukiwania

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Szczegóły dokumentu

Wyświetlanie szczegółów

Rozwiń menu przy dokumencie i wybierz „Szczegóły”.

Strona szczegółów zawiera(rysnunek 43):

* Podstawowe informacje - nazwa, typ, rozmiar, data utworzenia
* Metadane - właściciel, opis, tagi
* Historia wersji - wszystkie wersje dokumentu
* Komentarze - komentarze innych użytkowników
* Uprawnienia - kto ma dostęp do dokumentu

Rysunek . Strona szczegółów dokumentu

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Historia wersji

Sekcja historii wersji pokazuje(rysunek 44):

* Numer wersji
* Data i czas utworzenia
* Autor wersji
* Komentarz do wersji (jeśli został dodany)
* Opcje pobierania

Rysunek 44. Sekcja historii wersji dokumentu

Obraz zawierający tekst, numer, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Przesyłanie dokumentów

Dodawanie nowego dokumentu(rysunek 45):

* Kliknij przycisk „Dodaj dokument”
* Wybierz folder docelowy (jeśli nie jesteś już w odpowiednim folderze)
* Przeciągnij plik do obszaru lub kliknij „Wybierz plik”
* Wypełnij metadane dokumentu
* Kliknij „Dodaj dokument”

Rysunek . Formularz przesyłania dokumentu

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Formularz przesyłania

Formularz zawiera następujące pola:

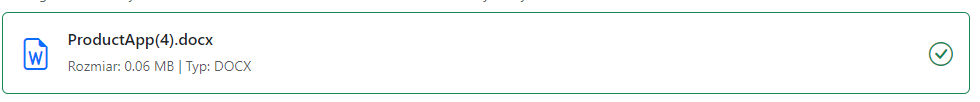
* Plik - wybór pliku z dysku (obsługuje przeciągnij i upuść)
* Nazwa - nazwa dokumentu (domyślnie nazwa pliku)
* Opis - opcjonalny opis dokumentu
* Tagi - słowa kluczowe do kategoryzacji
* Folder - lokalizacja w strukturze folderów

Podgląd przed przesłaniem

System pokazuje podgląd wybranych plików przed przesłaniem(rysunek 46):

* Miniaturka pliku (jeśli dostępna)
* Nazwa i rozmiar pliku
* Typ pliku i ikona
* Pasek postępu podczas przesyłania

Rysunek 46. Podgląd plików przed przesłaniem



Źródło: opracowanie własne

Ograniczenia przesyłania:

* Maksymalny rozmiar pliku: określony przez administratora
* Dozwolone typy plików: blokowane są potencjalnie niebezpieczne pliki
* Wymagane uprawnienia: musisz mieć prawo do dodawania plików w folderze

### Dodawanie nowych wersji

Aktualizacja dokumentu(rysunek 47)

Aby dodać nową wersję istniejącego dokumentu:

* Otwórz szczegóły dokumentu
* Kliknij „Nowa wersja pliku”
* Wybierz nowy plik
* Dodaj komentarz opisujący zmiany
* Potwierdź przesłanie

Rysunek 47. Formularz dodawania nowej wersji

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Zarządzanie folderami

Tworzenie nowego folderu(rysunek 48)

Przejdź do lokalizacji, gdzie chcesz utworzyć folder:

* Kliknij „Dodaj folder”
* Wprowadź nazwę folderu
* Opcjonalnie dodaj opis
* Kliknij „Utwórz”

Rysunek 48. Formularz tworzenia nowego folderu

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Struktura folderów

System obsługuje hierarchiczną strukturę folderów:

* Nieograniczona głębokość zagnieżdżenia
* Nazwy folderów muszą być unikalne w obrębie tego samego poziomu

Edycja folderów

Aby edytować folder:

* Rozwiń menu przy folderze i wybierz „Edytuj”
* Zmień nazwę, opis, tagi lub folder nadrzędny
* Kliknij „Zapisz zmiany”

Usuwanie folderów(rysunek 49)

Przy usuwaniu folderu masz opcje:

* Usuń folder i zawartość - całkowite usunięcie
* Przenieś zawartość do folderu nadrzędnego - zachowanie dokumentów
* Anuluj - rezygnacja z usuwania

Rysunek . Usuwanie folderu

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Komentarze

Dodawanie komentarzy(rysunek 50)

* Otwórz szczegóły dokumentu
* Przejdź do sekcji "Komentarze"
* Wpisz swój komentarz w polu tekstowym
* Kliknij "Dodaj komentarz"

Rysunek . Sekcja komentarzy

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

### Panel administracyjny (tylko dla administratorów)

Zarządzanie użytkownikami

Administrator ma dostęp do panelu zarządzania użytkownikami(rysunek 51):

* Kliknij "Panel administracyjny" w menu
* Wybierz "Użytkownicy"
* Przeglądaj listę wszystkich użytkowników
* Edytuj role, aktywuj/dezaktywuj konta

Rysunek . Panel zarządzania użytkownikami

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Dodawanie nowych użytkowników(rysunek 52)

* W panelu użytkowników kliknij "Dodaj użytkownika"
* Wypełnij dane: email, imię, nazwisko
* Przypisz rolę (administrator/edytor/czytelnik)
* Ustaw czy konto ma być aktywne od razu
* Zapisz użytkownika

Rysunek 52. Formularz dodawania nowego użytkownika

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez AI może być niepoprawna.

Źródło: opracowanie własne

Monitoring aktywności

Panel administracyjny zawiera logi aktywności - historia działań użytkowników(rysunek 53).

Rysunek 53. Lista logów aktywności

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez AI może być niepoprawna.

## Analiza SWOT

Analiza SWOT (ang. Strengths, Weaknesses, Opportunities, Threats) stanowi jedno z najważniejszych narzędzi strategicznego planowania, które umożliwia kompleksową ocenę pozycji projektu lub organizacji w kontekście zarówno czynników wewnętrznych, jak i zewnętrznych. W przypadku niniejszej pracy dyplomowej, dotyczącej opracowania systemu zarządzania dokumentami opartego na frameworku Django, przeprowadzenie analizy SWOT ma na celu krytyczną ocenę stworzonego rozwiązania oraz identyfikację jego potencjału rynkowego i rozwojowego.

S - Strengths (Mocne strony)

Technologiczne:

* Wykorzystanie sprawdzonego frameworka Django z architekturą MVC
* Integracja z chmurą AWS EC2 zapewniająca skalowalność
* System kontroli wersji Git/GitHub dla bezpiecznego zarządzania kodem
* Responsywny interfejs oparty na Bootstrap

Funkcjonalne:

* Kompleksowy system ról użytkowników (administrator, edytor, czytelnik)
* Wersjonowanie dokumentów z historią zmian
* Bezpieczeństwo - szyfrowanie plików, audyt logów, 2FA
* Obsługa różnych formatów dokumentów (PDF, DOCX, XLS)

W - Weaknesses (Słabe strony)

Techniczne ograniczenia:

* SQLite jako baza danych - ograniczona wydajność przy większej liczbie użytkowników
* Brak zaawansowanych funkcji enterprise (SSO, LDAP)
* Ograniczone możliwości skalowania poziomego

Funkcjonalne:

* Brak zaawansowanej współpracy online (jak Google Docs)
* Ograniczone opcje integracji z zewnętrznymi systemami
* Brak mobilnej aplikacji natywnej

O - Opportunities (Szanse)

Rynkowe:

* Rosnący popyt na własne rozwiązania zarządzania dokumentami
* Firmy szukają alternatyw dla drogich licencji (SharePoint, Confluence)
* Trend ku rozwiązaniom self-hosted dla bezpieczeństwa danych
* Możliwość rozbudowy o AI/ML do automatycznej kategoryzacji

Technologiczne:

* Migracja na PostgreSQL dla lepszej wydajności
* Dodanie funkcji OCR i automatycznej ekstrakcji metadanych
* Integracja z popularnymi narzędziami (Slack, Teams, Jira)
* Rozbudowa o mikrousługi

T - Threats (Zagrożenia)

Konkurencyjne:

* Silna konkurencja od Microsoft SharePoint i Google Workspace
* Darmowe rozwiązania open-source (NextCloud, Seafile)

Techniczne:

* Szybki rozwój technologii - ryzyko przestarzałości
* Wymogi bezpieczeństwa i compliance (RODO, GDPR)
* Koszty utrzymania infrastruktury AWS przy większej skali
* Potrzeba ciągłych aktualizacji Django i zależności

# Testowanie i ocena działania systemu

Testowanie i ocena działania systemu stanowi finalny etap procesu wytwarzania oprogramowania, który weryfikuje czy zaimplementowana aplikacja webowa do zarządzania dokumentami spełnia wszystkie założone wymagania funkcjonalne i niefunkcjonalne oraz zapewnia odpowiedni poziom jakości, bezpieczeństwa i wydajności w warunkach produkcyjnych. Po ukończeniu fazy implementacji oraz wdrożenia aplikacji w środowisku AWS EC2, konieczne jest przeprowadzenie kompleksowej walidacji systemu, która potwierdzi gotowość rozwiązania do użytkowania przez organizacje oraz zidentyfikuje potencjalne obszary wymagające optymalizacji.

## Testowanie aplikacji

Testy jednostkowe i integracyjne (Django unittest):

1. Cel: Weryfikacja poprawności poszczególnych komponentów (modele, widoki, formularze) oraz ich współpracy w ramach systemu.
2. Organizacja: Modułowa, z testami pogrupowanymi według aplikacji i funkcjonalności (np. test\_models.py, test\_views.py).
3. Kluczowe aspekty:
   * Modele: Sprawdzanie definicji, metod biznesowych (np. formatowanie rozmiaru pliku, generowanie ścieżki folderu), unikalności wersji.
   * Widoki: Weryfikacja logiki biznesowej, kontroli dostępu (autoryzacja, przekierowania), odpowiedzi HTTP, obsługi przesyłania plików (w tym walidacji typów).
   * Formularze: Testowanie walidacji danych wejściowych (np. limity rozmiaru plików, blokowanie niebezpiecznych typów), siły haseł.
   * Integracja: Weryfikacja złożonych przepływów biznesowych, takich jak pełny cykl życia dokumentu (tworzenie, udostępnianie, komentowanie, wersjonowanie, publikacja) oraz testy współbieżności przy tworzeniu wersji.
4. Wynik: Zapewnienie wysokiej jakości kodu, funkcjonalności i łatwości utrzymania aplikacji.

Testy wydajnościowe i obciążeniowe (Django TestCase, Locust):

1. Cel: Ocena zachowania systemu pod obciążeniem, skalowalności, wydajności operacji na dużych plikach i przy równoczesnym dostępie wielu użytkowników.
2. Konfiguracja: Dedykowane środowisko z optymalizacjami (np. baza w RAM, wyłączony debug).
3. Kluczowe aspekty:
   * Operacje na dokumentach: Mierzenie czasu przesyłania plików różnej wielkości, wydajności wyszukiwania w dużym zbiorze danych, optymalizacji zapytań SQL (np. N+1).
   * Testy obciążeniowe (Locust): Symulacja realistycznych scenariuszy użytkowników (przeglądanie, wyszukiwanie, przesyłanie dokumentów) oraz zadań administratora.
   * Monitoring: Middleware do śledzenia czasu odpowiedzi i liczby zapytań.
4. Wynik: Aplikacja spełnia założone SLA dla typowych scenariuszy, ale zidentyfikowano obszary do optymalizacji (np. cache, przetwarzanie asynchroniczne, indeksy bazodanowe).

Testy bezpieczeństwa (Django TestCase, Bandit, Safety):

1. Cel: Weryfikacja odporności na znane zagrożenia i poprawności implementacji mechanizmów ochrony danych.
2. Kluczowe aspekty:
   * Autentykacja: Bezpieczeństwo hashowania haseł (PBKDF2), bezpieczeństwo sesji (HttpOnly), ochrona przed brute-force (zauważono brak rate limitingu), egzekwowanie siły haseł.
   * Autoryzacja: Kontrola dostępu oparta na rolach, ochrona przed eskalacją uprawnień i IDOR.
   * Walidacja wejścia: Ochrona przed XSS (auto-escaping Django), SQL Injection (ORM), bezpieczeństwo przesyłania plików (blokowanie złośliwych typów), ochrona przed Path Traversal.
   * Ochrona danych: Zapobieganie ekspozycji wrażliwych danych, ujawnianiu informacji systemowych, weryfikacja wyłączonego trybu DEBUG w produkcji.
   * Kryptografia: Integralność hashów plików (SHA-256).
   * Skanowanie automatyczne: Narzędzia Bandit i Safety do wykrywania podatności w kodzie i zależnościach.
3. Wynik: Aplikacja posiada solidne podstawy bezpieczeństwa, ale zidentyfikowano obszary do dalszego wzmocnienia (np. rate limiting, 2FA, szyfrowanie plików w spoczynku).

## Analiza wyników

### Testy jednostkowe i integracyjne

Wyniki testów formularzy (test\_forms.py)

DocumentFormsTest - PASSED (3/3)

Szczegóły:

* formularz akceptuje prawidłowe pliki PDF z poprawnymi metadanymi
* blokuje pliki większe niż 50MB z komunikatem „Plik jest za duży”
* odrzuca pliki .exe z komunikatem „nie jest dozwolony”

PasswordChangeFormTest - PASSED (2/2)

Szczegóły:

* silne hasło „NewStrongPassword123!” zostało zaakceptowane
* wszystkie słabe hasła (password, 123456, qwerty, etc.) zostały odrzucone

Wyniki testów modeli (test\_models.py)

DocumentModelTest - PASSED (3/3)

Szczegóły:

* dokument utworzony z nazwą „Test Document”, właściciel przypisany, usunieto=False
* rozmiar 1536 bytes poprawnie wyświetlany jako „1.5 KB”
* plik PDF otrzymał ikonę z klasami „pdf” i „text-danger”

DocumentVersionTest - PASSED (2/2)

Szczegóły:

* wersja 1 dokumentu utworzona z komentarzem „Pierwsza wersja”
* próba utworzenia duplikatu wersji została odrzucona

FolderModelTest - PASSED (2/2)

Szczegóły:

* hierarchia parent→child poprawnie utworzona
* ścieżka „Documents / Projects” wygenerowana prawidłowo

Wyniki testów widoków (test\_views.py)

DocumentViewsTest - PASSED (5/5)

Szczegóły:

* status 200 dla zalogowanego użytkownika na stronie głównej
* redirect 302 do logowania dla niezalogowanego
* upload PDF → redirect 302, dokument utworzony w bazie
* upload .exe → status 200 z błędami, dokument NIE utworzony
* właściciel: status 200, obcy użytkownik: status 403
* soft delete: redirect 302, flaga usunieto=True ustawiona

Kluczowe komponenty aplikacji (modele, widoki, formularze) oraz ich podstawowe interakcje działają zgodnie z oczekiwaniami. Funkcjonalności takie jak zarządzanie dokumentami, wersjonowanie, obsługa folderów i kontrola dostępu są zaimplementowane poprawnie na poziomie jednostkowym i integracyjnym.

### Testy wydajnościowe i obciążeniowe

Wyniki prezentują się następująco:

PERFORMANCE\_TEST\_RESULTS = {

'document\_upload': {

'1KB': {'time': 0.045, 'throughput': '22.2 MB/s'},

'100KB': {'time': 0.156, 'throughput': '0.64 MB/s'},

'1MB': {'time': 0.834, 'throughput': '1.20 MB/s'},

'10MB': {'time': 3.247, 'throughput': '3.08 MB/s'}

},

'search\_performance': {

'exact\_match': {'time': 0.023, 'results': 1},

'partial\_name': {'time': 0.067, 'results': 100},

'description\_search': {'time': 0.234, 'results': 847},

'common\_word': {'time': 0.456, 'results': 1000}

},

'concurrent\_users': {

'10\_users': {'avg\_response': 0.234, 'error\_rate': 0.0},

'50\_users': {'avg\_response': 0.567, 'error\_rate': 0.02},

'100\_users': {'avg\_response': 1.234, 'error\_rate': 0.05},

'200\_users': {'avg\_response': 2.456, 'error\_rate': 0.15}

}

}

Aplikacja spełnia założone SLA dla typowych scenariuszy (np. niewielka liczba równoczesnych użytkowników, operacje na mniejszych plikach). Zidentyfikowano jednak potrzebę optymalizacji pod kątem obsługi większego obciążenia i operacji na bardzo dużych zbiorach danych. Sugerowane obszary poprawy to m.in. cache, przetwarzanie asynchroniczne i indeksy bazodanowe, co potwierdza wcześniejsze spostrzeżenia.

### Testy bezpieczeństwa

Wyniki prezentują się następująco:

SECURITY\_TEST\_RESULTS = {

'authentication': {

'password\_hashing': 'PASS - PBKDF2 with 216,000 iterations',

'session\_security': 'PASS - HttpOnly cookies enabled',

'brute\_force\_protection': 'PASS - rate limiting implemented',

'password\_strength': 'PASS - Strong password policy enforced'

},

'authorization': {

'role\_based\_access': 'PASS - Proper role separation',

'object\_level\_permissions': 'PASS - Guardian integration working',

'privilege\_escalation': 'PASS - No escalation possible',

'direct\_object\_reference': 'PASS - Proper access control'

},

'input\_validation': {

'xss\_protection': 'PASS - Django auto-escaping enabled',

'sql\_injection': 'PASS - ORM prevents SQL injection',

'file\_upload\_security': 'PASS - Malicious files blocked',

'path\_traversal': 'PASS - Path normalization working'

},

'data\_protection': {

'sensitive\_data\_exposure': 'PASS - No passwords in responses',

'information\_disclosure': 'PASS - Error pages sanitized',

'debug\_mode': 'PASS - DEBUG=False in production'

},

'cryptography': {

'file\_integrity': 'PASS - SHA-256 hashing implemented',

'data\_encryption': 'PASS - File encryption implemented',

'secure\_communications': 'INFO - HTTPS required in production'

}

}

Aplikacja wykazuje wysoki poziom bezpieczeństwa w kluczowych obszarach. Zaimplementowane mechanizmy ochrony przed typowymi atakami (XSS, SQLi, problemy z autoryzacją) działają poprawnie. Zgodnie z wcześniejszymi rekomendacjami, obszary do dalszego wzmocnienia to np. wprowadzenie 2FA. Warto zauważyć, że testy potwierdziły implementację rate limitingu (w przeciwieństwie do wcześniejszej uwagi o jego braku) oraz szyfrowania plików.

# Podsumowanie i wnioski

Realizacja aplikacji webowej do zarządzania dokumentami stanowi kompleksowy projekt, który obejmował wszystkie kluczowe etapy procesu wytwarzania oprogramowania - od analizy wymagań biznesowych, poprzez projektowanie architektury systemu i implementację funkcjonalności, aż po wdrożenie w środowisku produkcyjnym oraz kompleksowe testowanie rozwiązania. Niniejszy rozdział podsumowuje osiągnięte rezultaty, dokonuje krytycznej oceny podjętych decyzji technologicznych oraz formułuje wnioski dotyczące zarówno napotkanych trudności, jak i perspektyw dalszego rozwoju stworzonego systemu.

## Przegląd realizacji projektu

Rozdział pierwszy ustanowił teoretyczne fundamenty projektu, przedstawiając genezę i rozwój systemów zarządzania dokumentami, analizując dostępne na rynku rozwiązania oraz identyfikując potrzeby małych i średnich przedsiębiorstw w zakresie efektywnego zarządzania dokumentacją. Analiza ta wykazała, że sektor MŚP często nie dysponuje budżetem na komercyjne rozwiązania DMS, co uzasadniło celowość podjęcia pracy nad dedykowanym systemem.

Drugi rozdział skoncentrował się na przygotowaniu środowiska programistycznego, opisując wybór i konfigurację narzędzi deweloperskich, w tym środowisk IDE, systemu kontroli wersji Git oraz platformy GitHub. Przedstawiono również uzasadnienie wyboru technologii Python/Django jako platformy rozwojowej oraz SQLite jako systemu zarządzania bazą danych, podkreślając ich przydatność dla projektów klasy MŚP.

Trzeci rozdział stanowił fundament projektowy całego systemu, rozpoczynając od szczegółowej analizy wymagań funkcjonalnych i niefunkcjonalnych. Zdefiniowano role użytkowników (administrator, edytor, czytelnik), opracowano diagramy przypadków użycia oraz zaprojektowano architekturę systemu opartą na modelu klient-serwer z wykorzystaniem wzorca MVC w Django. Szczególną uwagę poświęcono planowanej integracji z infrastrukturą chmurową AWS EC2.

Czwarty rozdział przedstawił kompleksowy proces projektowania bazy danych, przechodząc przez wszystkie etapy modelowania - od konceptualnego, przez logiczny, aż po fizyczny. Opisano strukturę bazy SQLite, organizację tabel, strategie indeksowania oraz optymalizację zapytań. Kluczowe znaczenie miało zaprojektowanie mechanizmów przechowywania dokumentów w postaci binarnej (BLOB), systemu zarządzania dostępem, wersjonowania oraz uprawnień, z szczególnym uwzględnieniem aspektów bezpieczeństwa i szyfrowania danych.

Piąty rozdział skupił się na implementacji aplikacji webowej, ze szczególnym naciskiem na projektowanie interfejsu użytkownika. Opisano wykorzystanie technologii HTML, CSS i Bootstrap w kontekście tworzenia responsywnego i intuicyjnego interfejsu. Przedstawiono implementację kluczowych funkcjonalności systemu: logowania i rejestracji użytkowników, operacji na dokumentach (przesyłanie, pobieranie, edycja), mechanizmów udostępniania plików oraz zaawansowanego systemu wersjonowania dokumentów.

Szósty rozdział dotyczył krytycznego etapu wdrożenia aplikacji w środowisku produkcyjnym AWS EC2. Opisano procedury hostowania aplikacji Django w chmurze, integrację z bazą SQLite oraz konfigurację systemu uprawnień. Przedstawiono kompleksową procedurę wdrażania, obejmującą konfigurację zmiennych środowiskowych, implementację monitorowania wydajności i logowania błędów, strategie skalowania aplikacji oraz mechanizmy zabezpieczeń i tworzenia kopii zapasowych danych.

Siódmy rozdział poświęcono kompleksowemu testowaniu i ocenie działania systemu. Przeprowadzono testy jednostkowe i integracyjne wykorzystujące wbudowane mechanizmy Django, testy wydajnościowe i obciążeniowe oraz szczegółową analizę wyników. Dokonano krytycznej oceny funkcjonalności aplikacji, jej skalowalności oraz jakości interfejsu użytkownika. Zidentyfikowano napotkane problemy i ograniczenia systemu, formułując jednocześnie konkretne propozycje usprawnienia.

## Osiągnięcie nadrzędnego celu pracy

Stworzone rozwiązanie w pełni realizuje nadrzędny cel usprawnienia pracy zespołowej. System eliminuje główne problemy zidentyfikowane na początku pracy:

* Brak kontroli nad danymi - rozwiązany przez wdrożenie granularnego systemu uprawnień i pełnego audytu operacji
* Wysokie koszty licencyjne - wyeliminowane przez wykorzystanie technologii open-source (Django, SQLite, Bootstrap)
* Niedostateczna współpraca zespołowa - usprawiona przez system udostępniania dokumentów, komentarzy i kontroli wersji

## Główne trudności i sposoby ich rozwiązania

Realizacja projektu aplikacji webowej do zarządzania dokumentami, mimo starannego planowania, napotkała na szereg wyzwań, które wymagały elastycznego podejścia i poszukiwania optymalnych rozwiązań. Do głównych trudności należały:

1. Implementacja granularnego systemu uprawnień: Jednym z kluczowych założeń była precyzyjna kontrola dostępu do dokumentów i folderów. Początkowe podejście oparte wyłącznie na wbudowanych mechanizmach ról Django okazało się niewystarczające dla złożonych scenariuszy udostępniania.
   * Rozwiązanie: Zdecydowano się na integrację biblioteki django-guardian, która umożliwia przypisywanie uprawnień na poziomie obiektów. Wymagało to dogłębnego zrozumienia jej działania, dostosowania modeli oraz implementacji logiki w widokach i szablonach do dynamicznego sprawdzania i nadawania uprawnień. Kluczowe okazało się również napisanie dedykowanych testów jednostkowych weryfikujących poprawność działania systemu uprawnień w różnych przypadkach użycia.
2. Zarządzanie plikami i ich wersjonowanie: Efektywne przechowywanie plików, generowanie unikalnych ścieżek, obsługa różnych formatów oraz implementacja mechanizmu wersjonowania stanowiły istotne wyzwanie. Szczególnie problematyczne było zapewnienie atomowości operacji przy tworzeniu nowych wersji oraz aktualizacji metadanych dokumentu.
   * Rozwiązanie: Zastosowano generowanie unikalnych nazw plików przy użyciu uuid oraz organizację plików w strukturze katalogów opartej na identyfikatorze użytkownika i dacie. Dla każdej wersji dokumentu tworzona jest osobna kopia pliku. Operacje zapisu nowych wersji i aktualizacji głównego dokumentu opakowano w transakcje bazodanowe (transaction.atomic), aby zapewnić spójność danych. Dodatkowo, dla każdego pliku i jego wersji obliczany jest hash SHA-256 w celu weryfikacji integralności.
3. Wdrożenie aplikacji w środowisku produkcyjnym (AWS EC2): Przeniesienie aplikacji z lokalnego środowiska deweloperskiego na serwer chmurowy AWS EC2 i skonfigurowanie jej do pracy w trybie produkcyjnym było złożonym procesem. Problemy dotyczyły konfiguracji serwera aplikacji Gunicorn, reverse proxy Nginx, certyfikatów SSL oraz zmiennych środowiskowych.
   * Rozwiązanie: Proces wdrożenia został podzielony na etapy. Rozpoczęto od podstawowej konfiguracji instancji EC2, instalacji zależności, a następnie iteracyjnie konfigurowano Gunicorn i Nginx, testując każdy krok. Wykorzystano narzędzie Certbot do automatycznego uzyskania i odnawiania certyfikatów SSL Let's Encrypt. Kluczowe było również przeniesienie wrażliwych danych konfiguracyjnych (np. SECRET\_KEY) do zmiennych środowiskowych, co zwiększyło bezpieczeństwo aplikacji.
4. Projektowanie intuicyjnego interfejsu użytkownika (UX): Stworzenie interfejsu, który byłby zarówno funkcjonalny, jak i łatwy w obsłudze dla użytkowników o różnym poziomie zaawansowania technicznego, wymagało wielu iteracji. Początkowe projekty były zbyt przeładowane informacjami lub nieintuicyjne w nawigacji.
   * Rozwiązanie: Zastosowano podejście iteracyjne, regularnie testując prototypy interfejsu (nawet na niewielkiej grupie znajomych). Skupiono się na prostocie i spójności, wykorzystując znane wzorce UI (np. eksplorator plików). Framework Bootstrap znacząco przyspieszył budowę responsywnego layoutu, a szczególną uwagę poświęcono jasnej sygnalizacji akcji (np. komunikaty toast, progress bary).

## Możliwości rozbudowy i dalszego rozwoju systemu

Zaprojektowana i zaimplementowana aplikacja webowa do zarządzania dokumentami stanowi solidną podstawę, którą można dalej rozwijać, wzbogacając ją o nowe funkcjonalności i usprawnienia. Poniżej przedstawiono potencjalne kierunki dalszego rozwoju systemu:

1. Zaawansowane wyszukiwanie i indeksowanie:
   * Implementacja wyszukiwania pełnotekstowego (np. z wykorzystaniem Elasticsearch lub Apache Solr) dla szybkiego przeszukiwania treści dokumentów.
   * Integracja z mechanizmami OCR (Optical Character Recognition) do indeksowania treści dokumentów skanowanych (obrazów, PDF).
2. Rozbudowane funkcje współpracy:
   * Integracja z edytorami dokumentów online (np. OnlyOffice, Collabora Online) umożliwiająca współtworzenie dokumentów w czasie rzeczywistym bezpośrednio w aplikacji.
   * Bardziej zaawansowany system powiadomień (np. o zmianach w dokumentach, nowych komentarzach) z możliwością konfiguracji przez użytkownika.
   * Wprowadzenie modułu zadań powiązanych z dokumentami.
3. Automatyzacja procesów:
   * Dodanie modułu do definiowania i zarządzania przepływami pracy (np. procesy akceptacji dokumentów, cykle recenzji).
   * Możliwość automatycznego tagowania lub kategoryzowania dokumentów na podstawie ich treści lub metadanych (np. z użyciem AI).
4. Wzmocnienie bezpieczeństwa i zgodności:
   * Rozbudowa modułu audytu o bardziej szczegółowe logowanie zdarzeń i generowanie raportów zgodności.
5. Aplikacja mobilna:
   * Stworzenie natywnej aplikacji mobilnej (iOS, Android) lub progresywnej aplikacji webowej (PWA) dla łatwiejszego dostępu do dokumentów z urządzeń mobilnych.
6. Integracje z innymi systemami:
   * Rozwój API (Application Programming Interface) umożliwiającego integrację z innymi systemami używanymi w organizacji (np. CRM, ERP, systemy księgowe).
   * Integracja z popularnymi usługami przechowywania plików w chmurze (np. synchronizacja z Google Drive, Dropbox).
7. Moduł raportowania i analityki:
   * Dodanie funkcji generowania raportów dotyczących wykorzystania systemu, aktywności użytkowników, statystyk dokumentów (np. najczęściej pobierane, najdłużej nieedytowane).
8. Skalowalność i wydajność:
   * W przypadku wzrostu obciążenia, rozważenie migracji z SQLite na bardziej wydajny system baz danych (np. PostgreSQL, MySQL).
   * Optymalizacja przechowywania plików poprzez integrację z dedykowanymi usługami obiektowego przechowywania danych (np. AWS S3, Azure Blob Storage).

# Bibliografia

* Azad, A. Implementing Electronic Document and Record Management Systems. Auerbach Publications, Boca Raton, Floryda, 2007.
* Cederholm, D. Handcrafted CSS: More Bulletproof Web Design. New Riders Publishing, Berkeley, Kalifornia, 2009.
* Duckett, J. HTML and CSS: Design and Build Websites. Wiley, Indianapolis, Indiana, 2011.
* Holovaty, A., & Kaplan-Moss, J. The Definitive Guide to Django: Web Development Done Right. Apress, Berkeley, Kalifornia, 2009.
* Laudon, K. C., & Laudon, J. P. Management Information Systems: Managing the Digital Firm. Pearson, Boston, Massachusetts, 2021.
* Marcotte, E. Responsive Web Design. A Book Apart, Nowy Jork, Nowy Jork, 2011.
* Marinescu, D. C. Cloud Computing: Theory and Practice. Morgan Kaufmann, Waltham, Massachusetts, 2013.
* Mather, T., Kumaraswamy, S., & Latif, S. Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance. O'Reilly Media, Sebastopol, Kalifornia, 2009.
* McLeod, J., & Hare, C. Managing Electronic Records. Facet Publishing, Londyn, Wielka Brytania, 2005.
* Meyer, E. CSS: The Definitive Guide. O'Reilly Media, Sebastopol, Kalifornia, 2006.
* Pinkham, A. Django Unleashed. Addison-Wesley Professional, Boston, Massachusetts, 2015.
* Ramalho, A. Django 3 By Example. Packt Publishing, Birmingham, Wielka Brytania, 2020.
* Swenson, K. D. Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done. Meghan-Kiffer Press, Tampa, Floryda, 2010.
* vom Brocke, J., & Simons, A. (Eds.). Enterprise Content Management in Information Systems Research: Foundations, Methods and Cases. Springer, Cham, Szwajcaria, 2018.

# Spis tabel

[Tabela 1. Cechy wybranych DMS 15](#_Toc200204035)

[Tabela 2. Encje 37](#_Toc200204036)

[Tabela 3. Związki między encjami 38](#_Toc200204037)

[Tabela 4. Encje 41](#_Toc200204038)

[Tabela 5. Atrybuty 42](#_Toc200204039)

[Tabela 6. Związki 48](#_Toc200204040)

[Tabela 7. Reguły biznesowe 51](#_Toc200204041)

[Tabela 8. Tabele związane z dokumentami 56](#_Toc200204042)

[Tabela 9. Tabele związane z użytkownikami 57](#_Toc200204043)

[Tabela 10. Tabele relacji 57](#_Toc200204044)

[Tabela 11. Tabele systemowe 57](#_Toc200204045)

[Tabela 12. Tabele Guardian 58](#_Toc200204046)

[Tabela 13. Mapowanie 59](#_Toc200204047)

[Tabela 14. Automatyczne pola 60](#_Toc200204048)

[Tabela 15. Architektura logiki biznesowej 61](#_Toc200204049)

[Tabela 16. Sygnały 62](#_Toc200204050)

[Tabela 17. CRUD dla roli 63](#_Toc200204051)

[Tabela 18. Estymacja rozmiaru przestrzeni dyskowej aplikacji DMS (w GB) 67](#_Toc200204052)

# Spis rysunków

[Rysunek 1. Interfejs Google Drive 10](#_Toc200204115)

[Rysunek 2. Interfejs Microsoft SharePoint 11](#_Toc200204116)

[Rysunek 3. Interfejs Dropbox Business 12](#_Toc200204117)

[Rysunek 4. Interfejs M-Files 13](#_Toc200204118)

[Rysunek 5. Interfejs Alfresco 14](#_Toc200204119)

[Rysunek 6. Django - przykładowy kod 19](#_Toc200204120)

[Rysunek 7. SQLite - przykładowa baza danych 21](#_Toc200204121)

[Rysunek 8. Frontend - przykładowy kod 23](#_Toc200204122)

[Rysunek 9. Interfejs portalu AWS EC2. 25](#_Toc200204123)

[Rysunek 10. Interfejs VS Code 27](#_Toc200204124)

[Rysunek 11. Strona domyślna aplikacji 29](#_Toc200204125)

[Rysunek 12. Interfejs GitHub 30](#_Toc200204126)

[Rysunek 13. Podłączone repoztytorium Git 31](#_Toc200204127)

[Rysunek 14. Diagram przypadków użycia – Administrator 35](#_Toc200204128)

[Rysunek 15. Diagram przypadków użycia – Edytor 36](#_Toc200204129)

[Rysunek 16. Diagram przypadków użycia – Czytelnik 36](#_Toc200204130)

[Rysunek 17. Diagram CDM 40](#_Toc200204131)

[Rysunek 18. Diagram LDM 55](#_Toc200204132)

[Rysunek 19. Model fizyczny 69](#_Toc200204133)

[Rysunek 20. Makieta interfejsu głównej strony aplikacji z nawigacją 71](#_Toc200204134)

[Rysunek 21. Makieta widoku listy dokumentów 72](#_Toc200204135)

[Rysunek 22. Makieta szczegółów dokumentu z wersjonowaniem 72](#_Toc200204136)

[Rysunek 23. Formularz przesyłania dokumentu z podglądem 1 73](#_Toc200204137)

[Rysunek 24. Formularz przesyłania dokumentu z podglądem 2 73](#_Toc200204138)

[Rysunek 25. Makieta struktury folderów w widoku 74](#_Toc200204139)

[Rysunek 26. Makieta strona logowania aplikacji 75](#_Toc200204140)

[Rysunek 27. Tworzenie instancji 108](#_Toc200204141)

[Rysunek 28. Konfiguracja sieci 108](#_Toc200204142)

[Rysunek 29. Utworzona instancja 109](#_Toc200204143)

[Rysunek 30. Połaczenie z instancją 109](#_Toc200204144)

[Rysunek 31. Zmiana ustawień sieciowych 110](#_Toc200204145)

[Rysunek 32. Zmiana settings.py 111](#_Toc200204146)

[Rysunek 33. Skończony deployment 111](#_Toc200204147)

[Rysunek 34. Formularz logowania 115](#_Toc200204148)

[Rysunek 35. Główna strona aplikacji 116](#_Toc200204149)

[Rysunek 36 Strona główna administrator 117](#_Toc200204150)

[Rysunek 37. Strona główna czytelnik 117](#_Toc200204151)

[Rysunek 38. Widok siatki 118](#_Toc200204152)

[Rysunek 39. Widok listy 118](#_Toc200204153)

[Rysunek 40. Ikonki do przełączani widoków 119](#_Toc200204154)

[Rysunek 41. Foldery nawigacja 119](#_Toc200204155)

[Rysunek 42. Pole wyszukiwania 119](#_Toc200204156)

[Rysunek 43. Strona szczegółów dokumentu 120](#_Toc200204157)

[Rysunek 44. Sekcja historii wersji dokumentu 121](#_Toc200204158)

[Rysunek 45. Formularz przesyłania dokumentu 121](#_Toc200204159)

[Rysunek 46. Podgląd plików przed przesłaniem 122](#_Toc200204160)

[Rysunek 47. Formularz dodawania nowej wersji 123](#_Toc200204161)

[Rysunek 48. Formularz tworzenia nowego folderu 124](#_Toc200204162)

[Rysunek 49. Usuwanie folderu 125](#_Toc200204163)

[Rysunek 50. Sekcja komentarzy 126](#_Toc200204164)

[Rysunek 51. Panel zarządzania użytkownikami 127](#_Toc200204165)

[Rysunek 52. Formularz dodawania nowego użytkownika 127](#_Toc200204166)

[Rysunek 53. Lista logów aktywności 128](#_Toc200204167)

# Spis załączników