

FILIA
UNIwersytetu w Białymstoku
w WILNIE
INFORMATYKA



Filia **UwB**
w Wilnie

Gabriel Rafał Ingielewicz
Nr albumu: 89428

SOPS – SYSTEM OBSŁUGI
PRAKTYK STUDENCKICH

*Praca dyplomowa napisana
pod kierunkiem
dr. Mariusza Żynela*

Wilno 2026

Spis treści

Wstęp	1
1 Wprowadzenie	2
2 Narzędzia oraz środowisko pracy	3
2.1 Technologie backendowe	3
2.1.1 ASP.NET	3
2.1.2 Entity Framework	3
2.1.3 MS SQL Server	4
2.2 Technologie frontendowe	4
2.2.1 React	4
2.2.2 TypeScript	4
2.3 Dodatkowe biblioteki i narzędzia wspomagające	4
2.3.1 TanStack Query	4
2.3.2 i18n	4
2.3.3 Git	5
3 Cel i zakres projektu	6
3.1 Cel projektu	6
3.2 Użytkownicy systemu	6
3.2.1 Firma	6
3.2.2 Student	7
3.2.3 Opiekun praktyk	7
3.2.4 Dziekan	7
3.2.5 Administrator	7
3.3 Analiza istniejących rozwiązań	8
3.3.1 Obecny sposób realizacji praktyk	8
3.3.2 Dostępne rozwiązania	8
3.3.3 Wnioski	8
4 Wymagania systemu	9
4.1 Wymagania funkcjonalne	9
4.2 Wymagania нефункционалне	10

5	Projektowanie systemu	11
5.1	Architektura systemu	11
5.2	Struktura bazy danych	11
5.3	Projekt backendu	14
5.4	Projekt interfejsu użytkownika	15
5.5	Diagram przypadków użycia i BPMN	15
6	Implementacja	18
6.1	Autoryzacja	18
6.2	Przechowywanie dokumentów	20
6.3	Praktyki studenckie	23
6.4	Oferty praktyk	27
6.5	Podania na oferty praktyk	29
6.6	Bezpieczeństwo	33
7	Testowanie	35
7.1	Hostowanie aplikacji	35
7.2	Scenariusze testowe	36
8	Propozycje rozwoju	38
9	Podsumowanie	40
	Spis rysunków	41
	Bibliografia	42

Wstęp

Praktyki mają na celu umożliwienie studentowi zdobycia profesjonalnego doświadczenia oraz poznanie, jak w rzeczywistości funkcjonuje firma, w tym jak wykonywane zadania wpływają na jej działanie i doświadczenie klientów oraz poczucie odpowiedzialności za powierzone obowiązki. Często podczas praktyk da się spotkać specyficzne problemy, które nie istnieją poza pracą, które należy zrozumieć i nauczyć się rozwiązywać. Ponadto praktyki pozwalają doświadczyć pracę w grupie, gdzie każdemu są przydzielone konkretne zadania i terminy ich wykonania, tu wymaga się też komunikacja z drużyną, bo często trzeba rozbudowywać wykonaną pracę innych członków zespołu lub z niej korzystać.

Praktyki występują na większości kierunków studiów, jednak w wielu miejscach zarządzanie praktykami wciąż jest realizowane ręcznie, mimo że żyjemy w czasach, w których prawie wszystko da się załatwić online.

Realizowany projekt jest próbą utworzenia rozwiązania, które umożliwia cyfryzację procesów zarządzania praktykami. System obsługi praktyk studenckich (SOPS) ma na celu ułatwienie zarządzania praktykami nie tylko administracji, ale także studentom. Jego zadaniem jest zaprezentowanie, w jaki rzeczywisty sposób taki system mógłby wpłynąć na zarządzanie praktykami.

W ramach systemu studenci mają możliwość przesyłania dokumentacji online. Znacznie upraszcza to cały proces, ponieważ student nie musi planować dnia w sposób pozwalający na osobiste przyjście do uczelni i złożenie dokumentów. Zamiast tego może on po prostu przesłać dokumenty do systemu i oczekiwać odpowiedzi administracji. Dodatkowo administracja również na tym korzysta, ponieważ może łatwiej weryfikować poprawność dokumentów studenta. W rezultacie rozwiązanie to powinno zmniejszyć problemy z rozpoczynaniem praktyk zbyt późno, zwłaszcza gdy przyczyną są dokumenty praktyk.

Opracowany program ma na celu nie tylko wspomagać stronę administracyjną, ale także pomagać studentom w znajdowaniu praktyk. System ma funkcję utworzenia konta firmowego, które pozwala na publikowanie ofert praktyk oraz przeglądanie podań studentów. Jeśli jednak firma nie chce zakładać konta, a chce jednorazowo dodać ofertę, administrator systemu może wprowadzić ją w imieniu firmy, bez potrzeby tworzenia konta.

Rozdział 1

Wprowadzenie

Procedura praktyk studenckich rozpoczyna się od samodzielnego znalezienia miejsca praktyk przez studenta. W tym celu najczęściej wykorzystywane są internetowe portale z ofertami pracy, takie jak LinkedIn lub lokalne serwisy ogłoszeniowe. Część studentów korzysta też z kontaktów osobistych co pozwala na odbycie praktyk poprzez znajomych pracujących w danej branży.

Zdarzają się jednak sytuacje, w których studentowi nie udaje się samodzielnie znaleźć miejsca praktyk. W takim przypadku powinien on skontaktować się z opiekunem praktyk w celu uzyskania wsparcia. W praktyce jednak studenci nie zawsze informują opiekuna o problemach w odpowiednim czasie. W konsekwencji problem bywa zgłaszany zbyt późno bądź w ogóle nie jest zgłaszany, co może utrudnić jego rozwiązanie przed upływem terminu, a w skrajnych przypadkach doprowadzić do niezaliczenia przedmiotu.

Po znalezieniu odpowiedniego miejsca student zobowiązany jest do zgłoszenia go opiekunowi praktyk oraz wypełnienia deklaracji planowanej praktyki zawodowej, a następnie oczekuje na jej weryfikację. Opiekun dokonuje oceny, czy zakres obowiązków w wybranej instytucji jest zgodny z profilem kierunku studiów. Dopiero po uzyskaniu pozytywnej decyzji student może przejść do kolejnego etapu.

Następnym krokiem jest przygotowanie umowy oraz uzyskanie niezbędnych podpisów. Student wypełnia odpowiednie formularze, uzyskuje podpis kierownika instytucji przyjmującej, a następnie składa komplet dokumentów w dziekanacie. Po ich zatwierdzeniu i podpisaniu przez dziekana, praktyka zostaje formalnie rozpoczęta, zgodnie z terminami wskazanymi w dokumentach.

W trakcie realizacji praktyk student zobowiązany jest do prowadzenia dziennika praktyk, w którym dokumentuje wykonywane zadania. Zaleca się uzupełnianie wpisów na bieżąco, co najmniej raz w tygodniu, aby zapewnić rzetelność dokumentacji. W praktyce jednak zdarzają się sytuacje, w których dziennik wypełniany jest dopiero pod koniec praktyki.

Po zakończeniu praktyk student składa w dziekanacie dziennik praktyk. W wyznaczonym terminie odbywa się również indywidualne spotkanie z opiekunem praktyk, którego celem jest weryfikacja przebiegu praktyki oraz zaliczenie przedmiotu.

Rozdział 2

Narzędzia oraz środowisko pracy

2.1 Technologie backendowe

2.1.1 ASP.NET

ASP.NET jest otwartoźródłowym¹ frameworkiem opracowanym przez Microsoft, przeznaczonym do tworzenia aplikacji webowych oraz serwisów API².

W projekcie framework został wykorzystany do stworzenia warstwy backendowej³ w postaci API obsługującego komunikację z frontendem oraz bazą danych.

2.1.2 Entity Framework

Entity Framework jest narzędziem typu ORM⁴ działającym w oparciu o bibliotekę ADO.NET⁵.

Entity Framework tłumaczy zapytania tworzone przy użyciu LINQ⁶ na zapytania SQL wykonywane przez bazę danych. Pozwala to uprościć kod dostępu do danych oraz zwiększyć jego czytelność.

¹ang. open source – oprogramowanie którego kod źródłowy jest publicznie dostępny i modyfikowany przez społeczność.

²Application Programming Interface – interfejs umożliwiający komunikację między systemami.

³ang. backend – część systemu odpowiedzialna za logikę oraz komunikację z bazą danych

⁴Object-Relational Mapping – technika mapowania obiektów języka programowania na struktury relacyjnej bazy danych.

⁵ADO.NET – biblioteka platformy .NET służąca do komunikacji z bazą danych

⁶Language Integrated Query – mechanizm umożliwiający tworzenie skomplikowanych zapytań w składni języka C#.

2.1.3 MS SQL Server

Microsoft SQL Server jest serwerem zarządzania relacyjną bazą danych opracowanym przez Microsoft. W projekcie to oprogramowanie zostanie wykorzystane jako główna baza danych. Wybór tej technologii w tym kontekście jest oczywisty i wynika z zapewnienia kompatybilności z platformą .NET oraz dobrej integracji z Entity Framework.

2.2 Technologie frontendowe

2.2.1 React

React to otwartoźródłowa biblioteka do budowy interfejsów użytkownika (ang. frontend) opracowana przez Meta⁷. Umożliwia tworzenie aplikacji typu SPA⁸.

React opiera się na architekturze komponentowej, czyli interfejs budowany jest z elementów które zawierają strukturę kodu HTML razem z logiką napisaną w JavaScript.

2.2.2 TypeScript

TypeScript jest rozszerzeniem języka JavaScript opracowanym przez Microsoft. Wprowadza statyczne typowanie oraz dodatkowe mechanizmy kontroli typów, co eliminuje błędy związane z niepoprawnym użyciem danych.

2.3 Dodatkowe biblioteki i narzędzia wspomagające

2.3.1 TanStack Query

TanStack Query jest biblioteką dla React, która wspomaga komunikację API poprzez obsługę pobierania danych (fetching), ich buforowania (caching) oraz synchronizacji stanu. To masowo zmniejsza ilość pisanego kodu poprzez te funkcje.

2.3.2 i18n

Biblioteka i18n pomaga w umiędzynarodowieniu aplikacji poprzez dostosowaniu jej do obsługi wielu języków. Obejmuje w szczególności mechanizm lokalizacji, takie jak tłumaczenie interfejsu użytkownika.

⁷Meta – korporacja technologiczna zarządzająca między innymi Facebook, Instagram, WhatsApp, Messenger.

⁸Single Page Application – aplikacja w której zawartość aktualizowana jest dynamicznie bez przeładowywania strony.

W tej pracy, biblioteka ta, została wykorzystana do utworzenia interfejsu w języku polskim oraz angielskim.

2.3.3 Git

Git to najczęściej wykorzystywany systemem kontroli wersji (VCS⁹). Umożliwia śledzenie zmian w kodzie, przywracanie wcześniejszych wersji oraz równoległą pracę wielu osób nad projektem.

W tej pracy był wykorzystany do zarządzania kodem źródłowym oraz wersjami, a także do pracy z różnych komputerów bez kłopotów z synchronizacją wersji projektu.

⁹Version Control System – system umożliwiający zarządzanie wersjami kodu oraz kolaborację pomiędzy wieloma programistami.

Rozdział 3

Cel i zakres projektu

3.1 Cel projektu

Celem projektu jest opracowanie zamkniętego systemu z interfejsem webowym, cyfryzującego procesy związane z praktykami studenckimi, które dotychczas wykonywane są ręcznie. System ma uprościć studentom zrealizowanie praktyk poprzez cyfryzację procesów takich jak przesyłanie dokumentów oraz wypełnianie dziennika praktyk.

Jednocześnie system też wspomaga pracę pracowników uniwersytetu, między innymi poprzez eliminację konieczności osobistej weryfikacji dokumentów, co jest możliwe przez sprawdzanie dokumentacji w systemie zdalnie. Dodatkowo umożliwia na żywo monitorowanie postępów studentów.

System nie ma na celu pełnej cyfryzacji całego procesu praktyk, lecz skupia się na kluczowych elementach, które można i warto usprawnić. System został opracowany na podstawie regulaminu uczelni, jednak nie oznacza to, że nie może być wykorzystany poza nią.

3.2 Użytkownicy systemu

System posiada różne typy użytkowników. Każdy z nich korzysta z aplikacji w różny sposób, oraz potrzebuje inne uprawnienia. Jako że system jest zamknięty, nie ma otwartej rejestracji, konta użytkowników tworzone są przez administratorów, którzy jednocześnie przypisują użytkownikom odpowiednie role.

System obsługuje następujące typy użytkowników: firma, opiekun praktyk, dziekan, administrator.

3.2.1 Firma

Użytkownik typu *firma* może publikować oferty praktyk w systemie oraz akceptować lub odrzucać studentów ubiegających się o przyjęcie na praktykę.

Posiada dostęp tylko do własnych ofert i nie ma wglądu do danych dotyczących praktyk ani innych użytkowników. Wyjątkiem są podstawowe informacje o studentach, którzy złożyli podanie na daną ofertę.

3.2.2 Student

Użytkownik typu *student* posiada konto, które umożliwia mu dostęp do przypisanych praktyk oraz ofert publikowanych przez firmy. Może ubiegać się o przyjęcie na praktykę poprzez składanie podania do ofert. W podaniu ma możliwość przesłania swojego CV. Po rozpoczęciu okresu realizacji praktyki student uzyskuje dostęp do jej szczegółów oraz do funkcjonalności związanych z nią, takich jak przesyłanie dokumentów oraz wypełnianie dziennika praktyk. Proces przebiegu praktyk jest zgodny z opisem przedstawionym w rozdziale 1.

3.2.3 Opiekun praktyk

Użytkownik typu *opiekun praktyk* ma dostęp do informacji o przebiegu praktyk przypisanych mu studentów. Do jego obowiązków należy weryfikacja oraz akceptacja lub odrzucenie deklaracji praktyk składanych przez studentów. Oprócz tego może też przeglądać oferty praktyk, ale nie ma możliwości wysyłania podania.

3.2.4 Dziekan

Użytkownik typu *dziekan* posiada dostęp do informacji o praktykach wszystkich studentów w systemie. Jego zadaniem jest akceptacja lub odrzucenie umów związanych z realizacją praktyk. Podobnie jak opiekun praktyk, może on przeglądać oferty praktyk, ale bez możliwości wysyłania podań.

3.2.5 Administrator

Użytkownik typu *administrator* posiada najszerszy zakres uprawnień w systemie. Odpowiada za zarządzanie danymi oraz weryfikację ich poprawności. Administrator ma dostęp do wszystkich zasobów systemu, z wyjątkiem procesów związanych z akceptacją lub odrzuceniem dokumentów praktyk. Może on zarządzać użytkownikami oraz praktykami, w tym dodawać, edytować i usuwać dane. Dodatkowo administrator może wykonywać operacje w imieniu innych użytkowników. Może on publikować oferty praktyk jako firma lub przysyłać dokumenty praktyk jako student.

3.3 Analiza istniejących rozwiązań

3.3.1 Obecny sposób realizacji praktyk

Obecnie proces organizacji i realizacji praktyk studenckich odbywa się w sposób manualny, bez wsparcia dedykowanego systemu informatycznego. Obejmuje on między innymi ręczne dostarczanie dokumentów, ich wypełnianie, oraz osobistą weryfikację przez pracowników uczelni.

Takie podejście jest czasochłonne, podatne na opóźnienia i utrudnia, a nawet uniemożliwia monitorowanie postępów studentów, co wpływa negatywnie na cały proces.

3.3.2 Dostępne rozwiązania

Na rynku dostępne są komercyjne systemy wspierające zarządzanie praktykami studenckimi, które oferują szeroki zakres funkcjonalności. Często są to bardzo rozbudowane rozwiązania, zawierające wiele zbędnych funkcji z perspektywy uczelni, przy jednoczesnym braku wsparcia dla bardziej specyficznych wymagań. Dodatkowo ich wdrożenie wiąże się z wysokimi kosztami licencyjnymi.

Alternatywą są systemy tworzone wewnętrznie przez inne uczelnie. Takie rozwiązania są zazwyczaj dostosowane do indywidualnych potrzeb uczelni, co może utrudnić ich wykorzystanie w innym środowisku bez modyfikacji. Ponadto ich użycie wymagałoby współpracy między uczelniami lub udostępnienia systemu.

3.3.3 Wnioski

Na podstawie przeprowadzonej analizy można stwierdzić, że obecne rozwiązanie nie odpowiada w pełni potrzebom uczelni. Proces manualny może prowadzić do opóźnień, natomiast dostępne systemy komercyjne i wewnętrzne mogą być kosztowne lub niedostosowane do specyficznych wymagań.

W związku z tą analizą, uzasadnione jest opracowanie własnego systemu, który jest dostosowany do konkretnych potrzeb uczelni

Rozdział 4

Wymagania systemu

4.1 Wymagania funkcjonalne

1. Autoryzacja

- Użytkownik może zalogować się do systemu
- Administrator może zarządzać kontami użytkowników

2. Obsługa praktyk

- Student może przysyłać i usuwać dokumenty związane z praktyką
- Student może wypełniać dziennik praktyk
- Opiekun praktyk i dziekan mają dostęp do informacji o praktykach przypisanych studentów
- Opiekun praktyk i dziekan mogą zaakceptować lub odrzucić dokumenty
- Administrator może zarządzać praktykami
- Administrator może wykonywać operacje w imieniu studenta, w tym przesyłanie dokumentów

3. Oferty praktyk

- Firma może zarządzać ofertami praktyk
- Firma może rozpatrywać podania przesłane przez studentów
- Student może przeglądać oferty praktyk oraz składać podania
- Opiekun i dziekan mogą przeglądać oferty praktyk
- Administrator może zarządzać ofertami w imieniu firmy
- Administrator może rozpatrywać podania w imieniu firmy

4.2 Wymagania niefunkcjonalne

Bezpieczeństwo : System zapewnia autoryzację opartą na rolach, co oznacza, że bez odpowiedniej roli użytkownik nie będzie mógł wykonać poszczególnych operacji. Zabezpieczenie jest w interfejsie użytkownika, ale też w backendzie.

Wydajność : System powinien zapewniać odpowiednią wydajność poprzez optymalizację zapytań do bazy danych oraz minimalizację operacji wykonanych podczas przetwarzania danych.

Rozdział 5

Projektowanie systemu

5.1 Architektura systemu

System korzysta z architektury klient-serwer. Serwer odpowiada za przetwarzanie logiki biznesowej oraz komunikację z bazą danych, natomiast klientem jest przeglądarka internetowa, która pobiera dane i prezentuje je użytkownikowi jako interfejs webowy.

Komunikacja pomiędzy klientem a serwerem odbywa się za pośrednictwem API, z wykorzystaniem protokołu HTTP oraz formatu JSON do wymiany danych. Zastosowanie otwartego API pozwala na integrację z systemami zewnętrznymi, jak i automatyzację operacji za pomocą skryptów, niezależnie od interfejsu użytkownika.

Backend został podzielony na warstwy złożone z kontrolerów i serwisów, gdzie serwisy stanowią logikę biznesową systemu razem z dostępem do danych.

Frontend jest zbudowany z komponentów, co upraszcza wielokrotne korzystanie z tych samych elementów interfejsu.

5.2 Struktura bazy danych

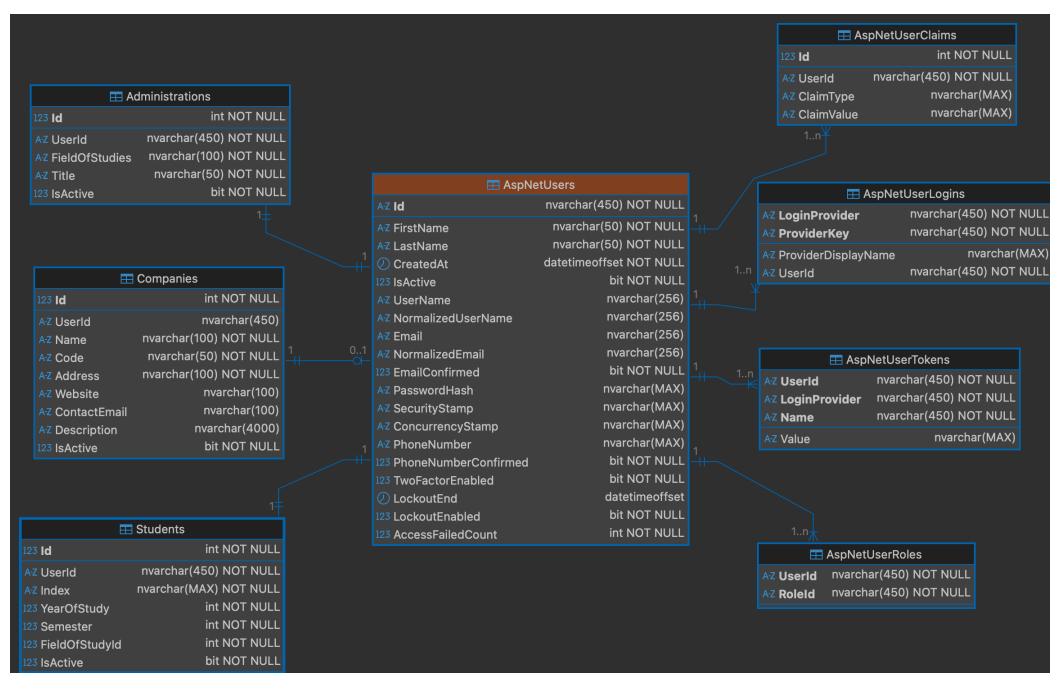
Baza danych składa się z 20 tabel. Ze względu na charakter przechowywanych danych oraz potrzeby zapewnienia ich spójności, tabele są ze sobą powiązane pośrednio lub bezpośrednio. Struktura da się rozdzielić na trzy główne obszary: użytkownicy, praktyki oraz oferty praktyk.

Tabela **AspNetUsers**, przedstawiona na rysunku 5.1, została utworzona przez framework, a następnie rozszerzona o dodatkowe pola. Najważniejszymi polami są: Email, FirstName, LastName, IsActive oraz PasswordHash. Ponadto tablica ta powiązana jest z dodatkowymi tablicami dostarczonymi przez mechanizm frameworka Identity¹ który pozwala na poprawne działanie uwierzytelnienia i autoryzacji.

¹W ASP.NET Identity to wbudowany mechanizm autoryzacji i zarządzania użytkownikami.

Do najważniejszych tablic, z punktu widzenia autoryzacji, należy tablica **AspNetUserRoles** stanowiąca kluczową część zarządzania uprawnieniami w systemie. Oprócz tego istnieją też tabele związane z tokenami, które upraszczają integrację z systemami uwierzytelnienia, takimi jak OAuth.

Profile użytkowników stanowią rozszerzenie tej tablicy o dodatkowe informacje dotyczące użytkownika. Przyjęto założenie, że użytkownik posiadający określoną rolę powinien też mieć przypisany odpowiedni profil. Brak takiego powiązania może prowadzić do błędów w systemie.



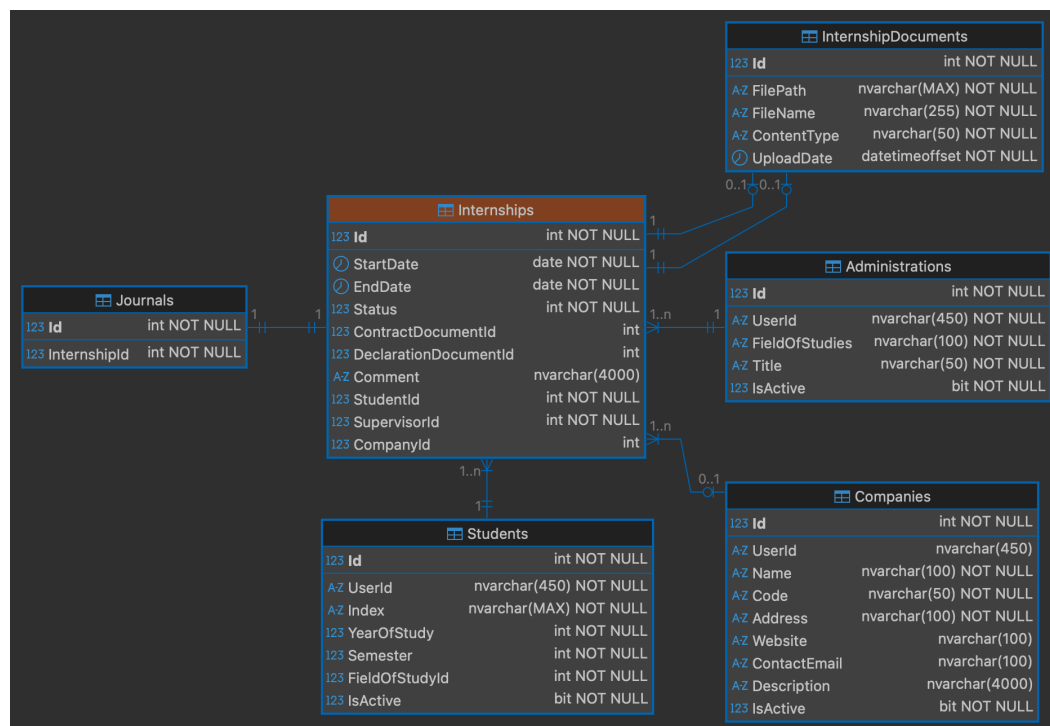
Rysunek 5.1: Diagram tabeli **AspNetUsers** oraz jej powiązania²

Kolejnym istotnym elementem jest tablica **Internships** przedstawiona na rysunku 5.2. Zawiera ona podstawowe informacje dotyczące praktyki, takie jak: jej stan, komentarz (np. w przypadku odrzucenia dokumentów) oraz klucze obce łączące ją z pozostałymi elementami systemu.

Jak pokazano na rysunku 5.2, tabela ma dwa powiązania z tabelą dokumentów praktyk. Umożliwia to przechowywanie różnych typów dokumentów, w tym wypadku deklaracji praktyki oraz umowy.

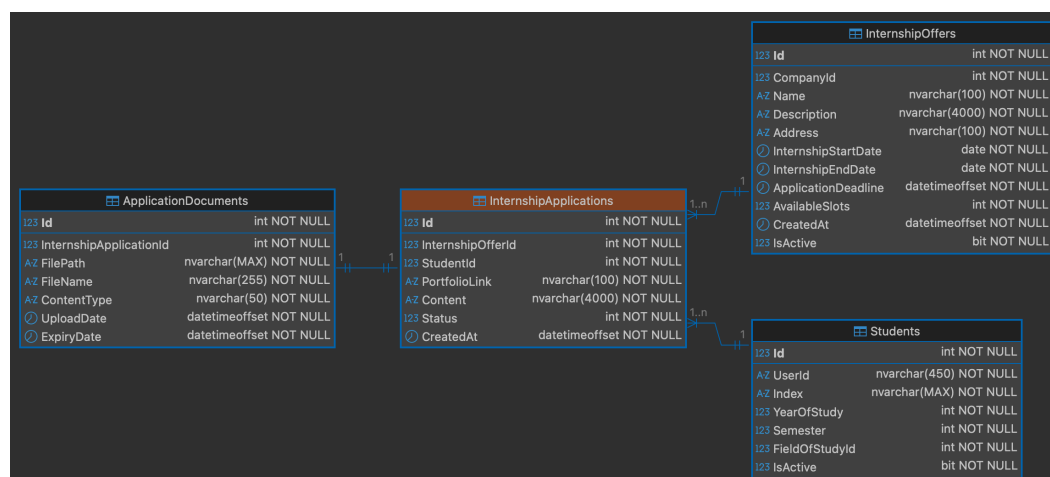
Dodatkowo tablica zawiera klucze obce odnoszące się do studenta, opiekuna, oraz opcjonalnie do firmy. Z praktyką powiązana jest też tabela **Journals**, pełniąca funkcję dziennika praktyk. Tabela **Journals** jest dalej rozszerzona o **JournalEntry**, gdzie przechowuje się wpisy dla konkretnego dnia. Relacja ta nie jest widoczna na rysunku 5.2 przedstawiającym tabelę praktyk, jednak została uwzględniona na rysunku 5.4, prezentującym pełną strukturę bazy danych.

²Wszystkie zamieszczone w pracy grafiki zostały wykonane samodzielnie.



Rysunek 5.2: Diagram tabeli Internships oraz jej powiązania

Tablica **InternshipApplications** przedstawiona na rysunku 5.3 umożliwia studentom składanie podań do ofert praktyk. Zawiera klucze obce do oferty praktyki, profilu studenta oraz dokumentu CV, a także informacje dodatkowe, takie jak: link do portfolio, wiadomość od kandydata, stan podania oraz datę utworzenia.

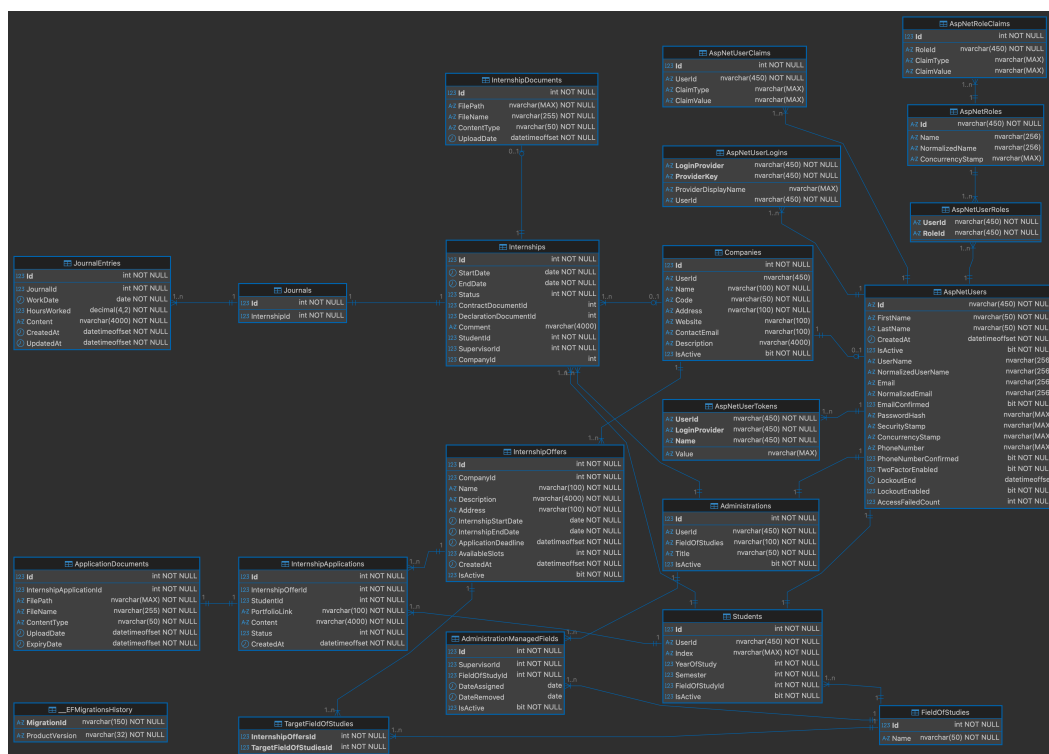


Rysunek 5.3: Diagram tabeli InternshipApplications oraz jej powiązania

Istotnym komponentem bazy jest również tabela **InternshipOffers**, któ-

rej celem jest przechowywanie informacji o dostępnych ofertach praktyk. Zawiera ona informacje ogólne o praktyce, jak i dane takie jak termin składania podań lub ilość dostępnych miejsc.

Na rysunku 5.4 przedstawiono pełną strukturę bazy danych ze wszystkimi tabelami oraz powiązaniem między nimi.



Rysunek 5.4: Struktura tabel bazy danych

5.3 Projekt backendu

Dla każdej tabeli w bazie danych utworzono osobny kontroler, którego zadaniem jest realizacja operacji CRUD. Kontrolery odpowiadają za obsługę żądań HTTP oraz zwracanie odpowiedzi do klienta. W przypadkach, gdy logika biznesowa jest złożona jest ona przeniesiona do warstwy serwisów.

Modele natomiast odwzorowują strukturę bazy danych i umożliwiają łatwe i bezpieczne typowo (ang. type-safe) operacje na obiektach z bazy danych. Do komunikacji z bazą danych wykorzystano Entity Framework, czyli ORM, który pozwala na korzystanie z danych w formie obiektów oraz eliminuje potrzebę pisania zapytań SQL.

Autoryzacja opiera się na tokenach JWT³ rozszerzonych o pole ról, co

³JWT (JSON Web Token) – standard internetowy służący do wymiany danych, ma dodatkowe opcje podpisu oraz enkrypcji [7].

pozwala na kontrolę dostępu do zasobów w zależności od uprawnień użytkownika.

Ponieważ API jest dostępne niezależnie od interfejsu użytkownika, zaimplementowano walidację danych po stronie serwera. W taki sposób backend odrzuca nieprawidłowe żądania w przypadku pominięcia weryfikacji po stronie klienta.

Dodatkowo w komunikacji wykorzystywane są obiekty DTO⁴, które umożliwiają uproszczenie struktury przesyłanych danych oraz zwiększają bezpieczeństwo komunikacji poprzez ograniczenie dostępu do specyficznych pól.

5.4 Projekt interfejsu użytkownika

Interfejs użytkownika został zaprojektowany w sposób umożliwiający zarządzanie danymi systemu. Operacje te są realizowane poprzez dedykowane widoki, ale też mniejsze formularze edycji.

System zawiera interfejsy do kluczowych obszarów, takich jak: oferty praktyk, podania na praktyki oraz zarządzanie przebiegiem praktyk.

Interfejs korzysta z mechanizmów walidacji danych, które ograniczają możliwość wprowadzenia niepoprawnych informacji oraz minimalizują liczbę błędów. Dodatkowo uwzględnia się kontrolę dostępu na poziomie interfejsu, dzięki czemu użytkownik ma dostęp do funkcji zgodnie z jego rolą w systemie.

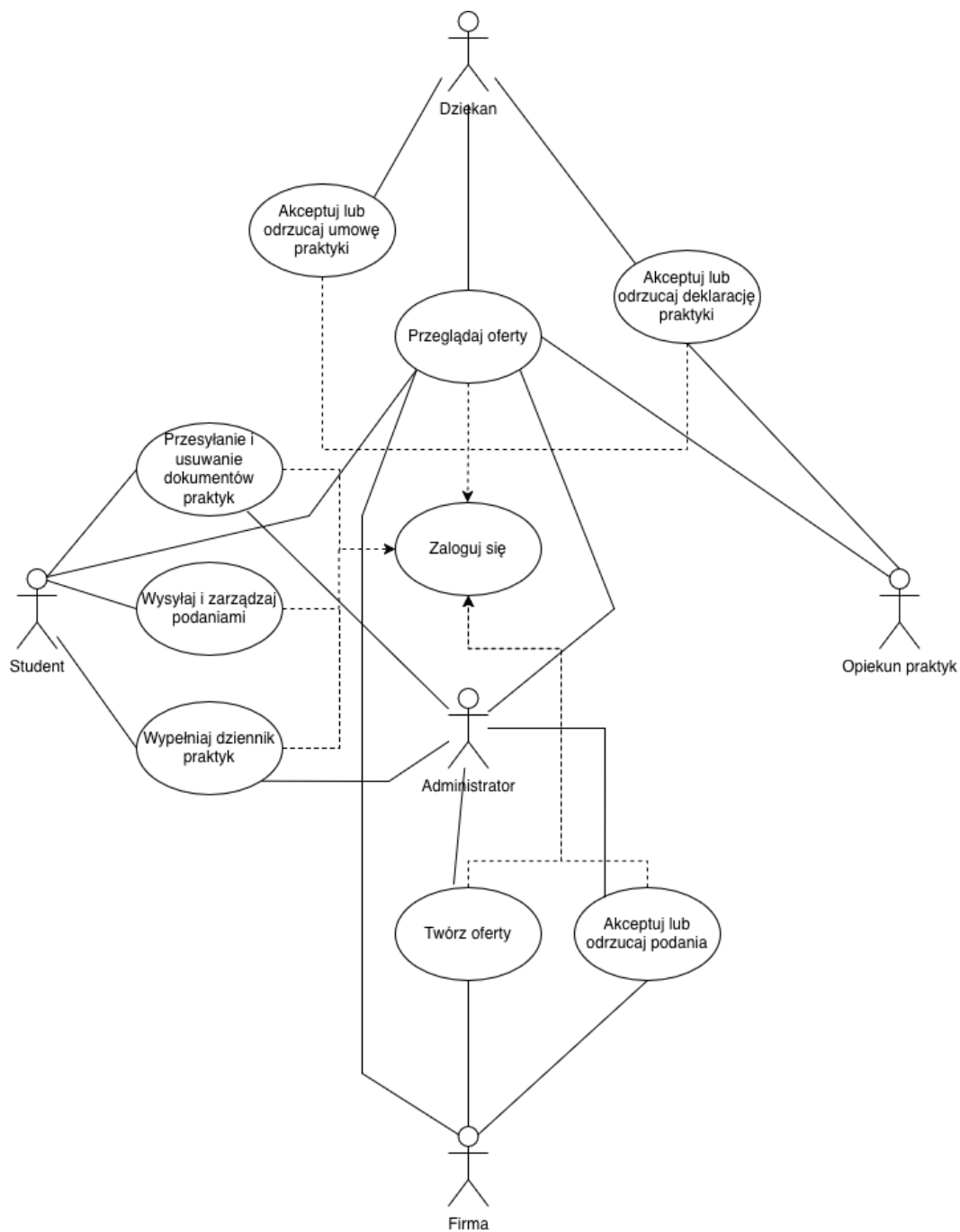
5.5 Diagram przypadków użycia i BPMN

Na rysunku 5.5 przedstawiono diagram przypadków użycia. Ma on na celu graficzne zaprezentowanie dostępnych funkcji systemu oraz przypisanych do nich ról użytkowników.

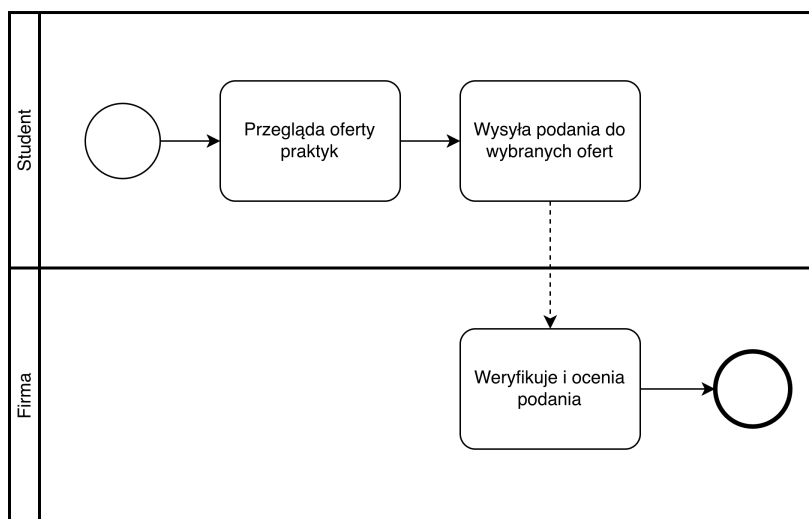
Na rysunkach 5.6 i 5.7 przedstawione są diagramy BPMN procesu szukania instytucji, w jakiej praktyka będzie realizowana, oraz sam proces realizacji praktyki. Ich celem jest pokazanie przebiegu tych procesów w systemie.

Należy zaznaczyć, że diagramy przedstawiają jedynie operacje realizowane w systemie. Czynności wykonywane poza systemem nie zostały uwzględnione, mimo że w rzeczywistości stanowią część procesu. Przykładowo, w przypadku poszukiwania praktyki, po zakończeniu etapu przedstawionego na diagramie może nastąpić komunikacja pomiędzy firmą a studentem.

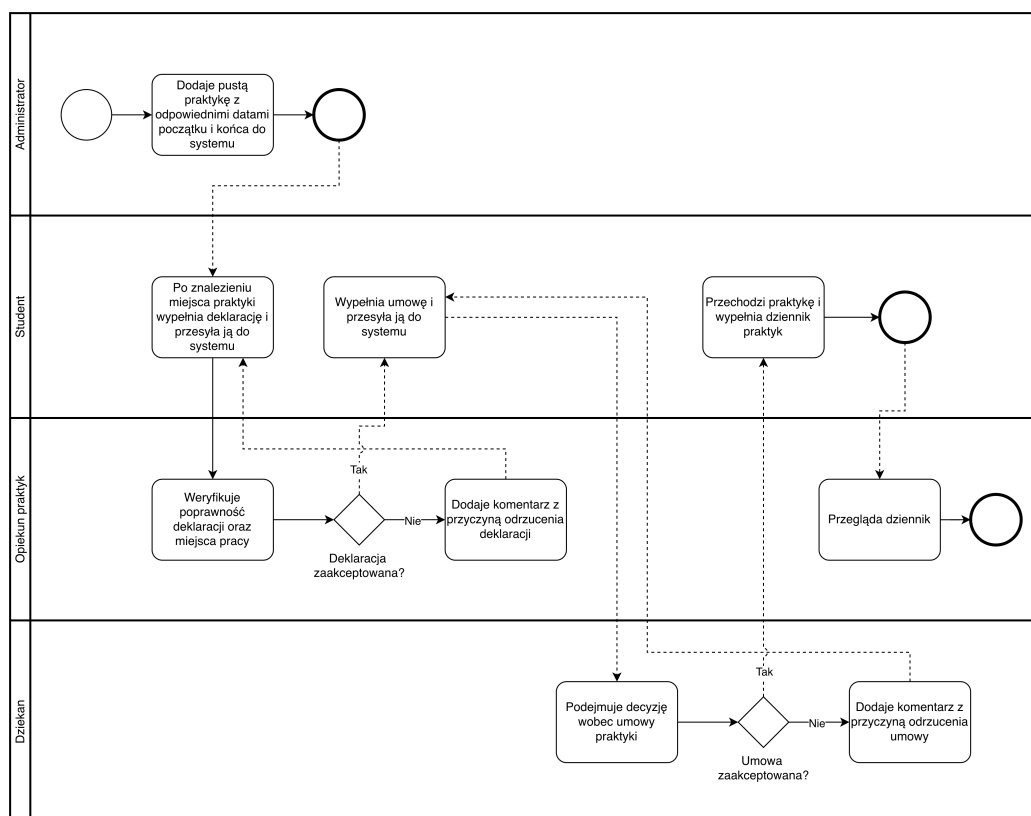
⁴Obiekt transferu danych (ang. Data transfer object) – struktura służąca do przesyłania danych. Umożliwia ograniczenie przesyłanych informacji do niezbędnego zakresu oraz uproszczenie logiki aplikacji [2].



Rysunek 5.5: Diagram przypadków użycia



Rysunek 5.6: Diagram BPMN przedstawiający proces szukania miejsca do realizacji praktyki



Rysunek 5.7: Diagram BPMN przedstawiający przebieg praktyki w systemie

Rozdział 6

Implementacja

6.1 Autoryzacja

Do implementacji autoryzacji w systemie skorzystano z tokenów JWT oraz pakietu `Microsoft.AspNetCore.Authentication.JwtBearer`, który zawiera kluczowe funkcje aby uprościć generację i walidację tokenów.

Konfiguracja autoryzacji została zdefiniowana w pliku `Program.cs`, gdzie określono parametry walidacji tokenów, takie jak wystawca, odbiorca oraz klucz podpisu. Odpowiada ona za weryfikację poprawności oraz integralności tokena.

```
var jwtSettings = builder.Configuration.GetSection("Jwt");

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateIssuerSigningKey = true,
        ValidateLifetime = true,
        ValidIssuer = jwtSettings["Issuer"],
        ValidAudience = jwtSettings["Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(jwtSettings["Key"] ?? throw new Exception())),
        RoleClaimType = ClaimTypes.Role
    };
});
```

W systemie zaimplementowano endpoint `/login`, który umożliwia uwierzytelnienie użytkownika na podstawie adresu e-mail oraz hasła. W przypadku poprawnych danych generowany jest token JWT zwracany do klienta. Gdy natomiast dane logowania są błędne, serwer zwraca odpowiedź 401, czyli nieautoryzowany dostęp (ang. `unauthorized`).

```
[HttpPost("login")]
public async Task<IActionResult> Login(LoginDto dto, [FromServices] JwtTokenService
    jwtService)
{
    var user = await _userManager.Users
        .Include(u => u.CompanyProfile)
        .FirstOrDefaultAsync(u => u.Email == dto.Email);

    if (user == null || user.IsActive == false || !await
        _userManager.CheckPasswordAsync(user, dto.Password))
    {
        return Unauthorized("Invalid credentials");
    }
    var token = jwtService.CreateToken(user);
    return Ok(new { token });
}
```

Proces generowania tokenu obejmuje dodanie dodatkowego zestawu danych (claims), które zawierają identyfikator użytkownika, adres e-mail oraz jego role. W przypadku użytkowników powiązanych z firmą dodawana jest również informacja o identyfikatorze firmy. Token podpisywany jest przy użyciu algorytmu HMAC-SHA256 oraz klucza przechowywanego w konfiguracji serwera. Token ma czas ważności wynoszący 12 godzin.

```
public async Task<string> CreateToken(User user)
{
    var jwt = _config.GetSection("Jwt");
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwt["Key"]!));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    var claims = new List<Claim>()
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Id),
        new Claim(JwtRegisteredClaimNames.Email, user.Email ?? ""),
    };

    if (user.CompanyProfile != null)
    {
        claims.Add(new Claim("companyId", user.CompanyProfile.Id.ToString()));
    }

    var roles = await _userManager.GetRolesAsync(user);
    foreach (var role in roles)
    {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }
    var token = new JwtSecurityToken(
        issuer: jwt["Issuer"],
        audience: jwt["Audience"],
        claims: claims,
        expires: DateTime.Now.AddHours(12),
        signingCredentials: creds);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Po stronie klienta zaimplementowano mechanizm obsługi autoryzacji, który komunikuje się z endpointem logowania oraz przechowuje otrzymany token JWT w pamięci przeglądarki.

Do dekodowania tokenów wykorzystano bibliotekę `jwt-decode`, co umożliwia odczytanie informacji o użytkowniku zawartej w tokenie.

```
async function login( email: string, password: string ) : Promise<{ success: boolean;
  message: string | null}>{
  const res = await fetch(`${import.meta.env.VITE_API_URL}/${endpoints.auth}/login`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({email, password})
  })

  if (!res.ok) return { success: false, message: "Invalid Credentials" };

  const data: ApiLoginResponse = await res.json();
  const token = data?.token.result

  if (!token) return { success: false, message: "Login failed: no token received" }

  localStorage.setItem(TOKEN_KEY, token)
  return { success: true, message: null }
}
```

W celu zarządzania stanem uwierzytelnienia w całej aplikacji zastosowano mechanizm kontekstu `AuthContext`. Umożliwia on dostęp do informacji o aktualnym użytkowniku oraz operacji związanych z autoryzacją, takich jak logowanie, wylogowanie, czy sprawdzenie ról użytkownika.

Dzięki temu możliwe jest centralne zarządzanie dostępem do funkcji systemu w zależności od roli użytkownika.

```
interface AuthContextType {
  user: DecodedUser | null
  isInitialized: boolean
  login: (email: string, password: string) => Promise<{ success: boolean; message: string | null }>
  logout: () => void
  isAuthenticated: () => boolean
  getToken: () => string | null
  hasRole: (searchRole: string) => boolean
}
```

System nie udostępnia publicznej rejestracji użytkowników. Konta tworzone są wyłącznie przez administratora za pośrednictwem panelu administracyjnego.

6.2 Przechowywanie dokumentów

Dokumenty przechowywane są na serwerze. W celu obsługi zapisu plików została opracowana funkcja `SaveFileAsync()`, która odpowiada za walidację oraz zapis dokumentu. Funkcja weryfikuje typ oraz rozmiar pliku, a następnie zapisuje go na serwerze z unikalnym identyfikatorem. Dodatkowo przyjmuje parametr `area`, który oznacza katalog docelowy, co pozwala na logiczne

rozdzielenie plików na podstawie ich celu.

```
private async Task<string> SaveFileAsync(IFormFile file, string area)
{
    if (!AllowedTypes.Contains(file.ContentType))
        throw new BadRequestException("File type not allowed.");

    if (file.Length > FileSizeLimit)
        throw new BadRequestException($"File cannot exceed {FileSizeLimit / 1024 / 1024} MB.");

    var extension = Path.GetExtension(file.FileName);
    var filename = $"{Guid.NewGuid()}_{extension}";
    var path = _uploadStorageHelper.GetUploadPath(area, filename);

    await using var stream = new FileStream(path, FileMode.Create);
    await file.CopyToAsync(stream);

    return path;
}
```

Funkcja ta jest wykorzystywana przez metody wyższego poziomu, jak:

- UploadInternshipDocumentAsync(),
- UploadApplicationDocumentAsync(),

które odpowiadają za zapis dokumentu oraz utworzenie odpowiedniego rekordu w bazie danych.

```
public async Task<InternshipDocument> UploadInternshipDocumentAsync(IFormFile file)
{
    var path = await SaveFileAsync(file, UploadPaths.Internships);

    return new InternshipDocument
    {
        ContentType = file.ContentType,
        FileName = file.FileName,
        FilePath = path,
        UploadDate = DateTimeOffset.UtcNow
    };
}
```

Dla poszczególnych typów dokumentów, takich jak deklaracja czy umowa, zaimplementowano dedykowane metody, które oprócz zapisu pliku realizują dodatkową logikę biznesową. Obejmuje ona weryfikację uprawnień użytkownika, sprawdzenie stanu praktyki oraz aktualizację jej statusu.

```
public async Task UploadDeclarationDocument(InternshipDocumentUploadDto dto, ClaimsPrincipal user)
{
    var internship = await EnsureStudentInternship(dto.InternshipId, user);

    if (internship.DeclarationDocument != null)
        throw new InvalidOperationException("Declaration document already uploaded");
    var declarationDocument = await documentUploadService.UploadInternshipDocumentAsync(dto.File);
    internship.DeclarationDocument = declarationDocument;
}
```

```
internship.Status = InternshipStatus.AwaitingDeclarationApproval;

await context.SaveChangesAsync();
}
```

Dostęp do przesyłania dokumentów realizowany jest poprzez endpointy API, takie jak: `/internships/declaration` oraz `/internships/contract`. Endpointy te są zabezpieczone i dostępne wyłącznie dla użytkowników posiadających odpowiednie role.

Pliki przesyłane są z wykorzystaniem formatu `FormData`¹, co pozwala na przesyłanie dokumentów wraz z dodatkowymi danymi w jednym żądaniu.

```
[Authorize(Roles = "Student, Admin")]
[HttpPost("declaration")]
public async Task<IActionResult> UploadDeclaration(
    [FromForm] InternshipDocumentUploadDto dto)
{
    try
    {
        await _service.UploadDeclarationDocument(dto, User);
        return NoContent();
    }
    catch (UnauthorizedAccessException e)
    {
        return Forbid(e.Message);
    }
    catch (KeyNotFoundException e)
    {
        return NotFound(e.Message);
    }
    catch (InvalidOperationException e)
    {
        return Conflict(e.Message);
    }
}
```

Po stronie klienta zaimplementowano mechanizm przesyłania plików z wykorzystaniem biblioteki `TanStack Query`, która obsługuje wysyłanie żądań oraz zarządzanie ich stanem.

```
const uploadDeclarationMutation = useMutation({
  mutationFn: async ({ internshipId, file }: { internshipId: number; file: File }) => {
    const formData = new FormData();
    formData.append("internshipId", internshipId.toString());
    formData.append("file", file);
    return apiFetch(endpoints.internships + "/declaration", {
      method: "POST",
      body: formData
    }, true);
  },
  onSuccess: async (_, data) => {
    await invalidateInternshipQueries(data.internshipId);
  }
});
```

¹FormData umożliwia przesyłanie danych binarnych, takich jak pliki, w żądaniach HTTP.

6.3 Praktyki studenckie

W celu umożliwienia śledzenia przebiegu praktyki w systemie wprowadzono stany, reprezentowane przez enum `InternshipStatus`. Każdy stan odpowiada określonemu etapowi realizacji praktyki.

```
public enum InternshipStatus
{
    NotStarted,
    AwaitingDeclarationApproval,
    DeclarationRejected,
    DeclarationApproved,
    AwaitingContractApproval,
    ContractRejected,
    InProgress,
    Completed
}
```

Praktyka powinna rozpoczynać się od stanu `NotStarted`. Administrator podczas tworzenia praktyki ma możliwość ustawienia innego stanu w wyjątkowych sytuacjach, jednak przy standardowym przebiegu, praktyka powinna się rozpoczynać od stanu `NotStarted`.

Przebieg praktyki opiera się na przechodzeniu pomiędzy kolejnymi stanami, które odzwierciedlają rzeczywisty proces formalny. Kluczowe są tu dokumenty, których przesłanie wpływa bezpośrednio na stan praktyki.

Deklaracja może być przesłana wyłącznie w stanie `NotStarted`. Po jej przesłaniu praktyka przechodzi w stan `AwaitingDeclarationApproval`. W przypadku odrzucenia deklaracji ustawiany jest stan `DeclarationRejected`, co oznacza konieczność poprawy dokumentu. Analogicznie wygląda mechanizm dla umowy, która może zostać przesłana tylko po zatwierdzeniu deklaracji (`DeclarationApproved`).

```
public async Task UploadDeclarationDocument(InternshipDocumentUploadDto dto,
    ClaimsPrincipal user)
{
    var internship = await EnsureStudentInternship(dto.InternshipId, user);

    if (internship.DeclarationDocument != null)
        throw new InvalidOperationException("Declaration document already uploaded");

    var declarationDocument = await
        documentUploadService.UploadInternshipDocumentAsync(dto.File);

    internship.DeclarationDocument = declarationDocument;
    internship.Status = InternshipStatus.AwaitingDeclarationApproval;

    await context.SaveChangesAsync();
}
```

Przypisany opiekun praktyk oraz dziekan mają możliwość akceptacji lub odrzucenia dokumentów, co pokazuje rysunek 6.1. W przypadku odrzucenia wymagane jest podanie komentarza, który ma na celu informowanie studenta o jego błędach.

Szczegóły praktyki ×

Status

Oczekuje na zatwierdzenie dziekana

Student

Imię i nazwisko studenta:
Emma Hughes

Numer indeksu:
89412

Kierunek studiów:
Math

Opiekun

Opiekun:
Noah Clark (PhD Computer Science)

Firma

Nazwa:
Redline Analytics

Kod:
C003

Adres:
25 Gedimino pr., Vilnius

Strona internetowa:
https://greenline.example

Email kontaktowy:
careers@greenline.example

Daty

Data rozpoczęcia:
2026-03-05

Data zakończenia:
2026-10-26

Dokumenty

Dokument deklaracji:
declaration_empty.pdf Pobierz

Dokument umowy:
contract_empty.pdf Pobierz

Zamknij Zaakceptuj umowę Odrzuć umowę

Rysunek 6.1: Weryfikacja praktyki przez opiekuna lub dziekana

Po zatwierdzeniu wymaganych dokumentów praktyka przechodzi w stan **InProgress**. Na tym etapie student uzyskuje dostęp do dziennika praktyk (rys. 6.2), gdzie może dodawać wpisy dokumentujące przebieg pracy.

Dziennik praktyk

×

Tydzień 2 (2026-03-09 - 2026-03-15)

▼

2026-03-09

Poniedziałek 4g 0m

✎

Praca nad projektem

📅 Utworzono: 05/06/2026 10:30

✎ Zaktualizowano: 05/06/2026 10:30

2026-03-10

Wtorek

+

2026-03-11

Środa

+

2026-03-12

Czwartek

+

2026-03-13

Piątek

+

2026-03-14

Sobota

+

2026-03-15

Niedziela

+

Zamknij

Wróć

Rysunek 6.2: Dziennik praktyk studenta

```
public async Task<int> CreateJournalEntryAsync(JournalEntryWriteDto dto, ClaimsPrincipal
    user)
{
    var journal = await _context.Journals.Include(journal => journal.Internship)
        .ThenInclude(internship => internship.Student)
        .SingleOrDefaultAsync(j => j.Id == dto.JournalId);

    if (journal == null)
        throw new KeyNotFoundException($"Journal with id {dto.JournalId} not found");

    AuthorizationHelper.EnsureStudentOwnership(user, journal.Internship.Student.UserId);

    var journalEntry = new JournalEntry
```

```
{  
    JournalId = dto.JournalId,  
    WorkDate = dto.Date,  
    HoursWorked = dto.HoursWorked,  
    Content = dto.Content,  
    CreatedAt = DateTimeOffset.UtcNow,  
    UpdatedAt = DateTimeOffset.UtcNow  
};  
  
await _context.JournalEntries.AddAsync(journalEntry);  
await _context.SaveChangesAsync();  
return journalEntry.Id;  
}
```

Zakończenie praktyki jest realizowane poprzez zmianę stanu na **Completed**, co może zostać wykonane przez opiekuna praktyk lub dziekana. Po przejściu do tego stanu student traci możliwość edycji dziennika, a dalsze zmiany wymagają pomocy administratora.

Warto również zaznaczyć, że opiekun praktyk posiada dostęp wyłącznie do praktyk przypisanych do kierunków, którymi zarządza. Ograniczenie to wynika z organizacji uczelni i pozwala na zachowanie odpowiedniego podziału odpowiedzialności.

Moje oferty

[Dodaj nową](#)

QA Automation Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Mobile App Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Machine Learning Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

QA Automation Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Mobile App Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Machine Learning Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

QA Automation Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Mobile App Intern
25 Gedimino pr., Vilnius
2026-07-04 - 2026-10-02

[Edytuj](#)[Usuń](#)

Rysunek 6.3: Lista ofert przypisanych do firmy

6.4 Oferty praktyk

Oferty praktyk mogą być tworzone przez administratora w imieniu firmy, jak i przez same firmy. Zarządzanie ofertami: tworzenie, edycja, usuwanie (por. rys. 6.3) zostały zrealizowane w formie standardowych operacji CRUD. Edytowanie oferty przedstawione jest na rysunku 6.4.

Aktualizuj QA Automation Intern

×

Podstawowe informacje

Nazwa oferty

QA Automation Intern

Opis

Build automated tests and help stabilize release pipelines.

Adres

25 Gedimino pr., Vilnius

Grupa docelowa

Docelowe kierunki studiów

Philosophy × Software Engineering × × | ▾

Wybierz kierunki studiów, do których skierowana jest oferta

Dostępne miejsca

4 ▾

Daty

Data rozpoczęcia praktyk

04/07/2026

Data zakończenia praktyk

02/10/2026

Termin składania aplikacji

21/06/2026, 11:11

Zamknij

Aktualizuj

Rysunek 6.4: Edycja oferty praktyki

Istotnym elementem jest sposób przeglądania ofert. Interfejs użytkownika został podzielony na dwie części: listę ofert oraz panel szczegółów wybranej oferty jak to przedstawiono na rysunku 6.5. Takie rozwiązanie pozwala na

szybkie przeglądanie wielu ofert bez konieczności przechodzenia pomiędzy oddzielnymi stronami.

Oferty praktyk

Szukaj...

Software Engineering Intern
Baltic Softworks • 12 Jasinskio g., Vilnius
2026-07-03 - 2026-10-01

Data Analyst Intern
Nordic Finance Labs • 8 Konstitucijos pr., Vilnius
2026-07-03 - 2026-10-01

QA Automation Intern
Redline Analytics • 25 Gedimino pr., Vilnius
2026-07-03 - 2026-10-01

Product Management Intern
Skywave Networks • 4 Kalvariju g., Vilnius
2026-07-03 - 2026-10-01

DevOps Intern
Amber Digital • 19 Pamenkalnio g., Vilnius
2026-07-03 - 2026-10-01

UI/UX Design Intern
Baltic Softworks • 12 Jasinskio g., Vilnius
2026-07-03 - 2026-10-01

« 1 2 »

Data Analyst Intern

Firma i lokalizacja

Firma:
Nordic Finance Labs

Adres:
8 Konstitucijos pr., Vilnius

Szczegóły praktyk

Opis:
Assist with dashboards, KPI tracking, and data validation.

Docelowe kierunki studiów:
Math Philosophy

Data rozpoczęcia:
2026-07-03

Data zakończenia:
2026-10-01

Termin składania aplikacji:
19/06/2026 11:12

Dostępne miejsca:
3

Aplikuj

Rysunek 6.5: Przegląd ofert praktyk

Po wybraniu oferty z listy, pobierane są jej szczegóły z serwera, a interfejs jest aktualizowany bez przeładowania całej strony. Podejście to zostało zrealizowane z wykorzystaniem SPA, co umożliwia dynamiczną aktualizację interfejsu oraz poprawia płynność działania aplikacji.

```

<div className="container-fluid">
  <div className="row" style={{ height: "86vh" }}>
    <div className="col-5 d-flex flex-column" style={{ height: "100%" }}>
      <div className="overflow-y-auto">
        {offersData.items.map(offer => (
          <div
            key={offer.id}
            className="p-3 border-bottom"
            style={{ cursor: "pointer" }}
            onClick={() => setSelectedId(offer.id)}
          >
            <h6 className="mb-1">{offer.name}</h6>
            <div className="text-muted small">
              {offer.companyName} &#8226; {offer.address}
            <br/>
            {offer.internshipStartDate} - {offer.internshipEndDate}
            </div>
          </div>
        ))}
      </div>
    </div>
  </div>

```

```

    })
  </div>
  <div className="mt-auto border-top border-black border-1 border-opacity-25">
    <PaginationControls
      page={page}
      onPageChange={updatePage}
      totalPages={offersData.totalPages}
      className="p-2"/>
  </div>
</div>
{selectedId && (
  <div className="col-7" style={{overflowY: "auto", height: "100%"}}>
    <div className="d-flex flex-column gap-3">
      {detailsIsPending && (<LoadingSpinner/>)}
      {detailsIsError && <ErrorMessage/>}
      {detailsData && (
        <>
          <OfferDetailsPanel details={detailsData}/>
          {hasRole("Student") && <ApplyButton
            offerId={detailsData.id.toString()}/>}
        </>
      )}
    </div>
  </div>
)}
</div>
</div>

```

Interfejs również uwzględnia obsługę stanów pośrednich, takich jak ładowanie danych czy błędy, co zapobiega awarii interfejsu.

6.5 Podania na oferty praktyk

W celu złożenia podania student zobowiązany jest do przesłania swojego CV. Dodatkowo może dołączyć link do portfolio oraz wiadomość lub list motywacyjny, co pokazano na rysunku 6.6.

Po wysłaniu żądania serwer weryfikuje poprawność danych, w szczególności sprawdza istnienie oferty praktyki oraz profilu studenta, a także, czy podanie dla danej oferty nie zostało wcześniej złożone. W przypadku braku błędów podanie zostaje zapisane w bazie danych, a plik CV przechowywany jest na serwerze.

```

public async Task CreateAsync(InternshipApplicationWriteDto dto, ClaimsPrincipal user)
{
    var userId = user.FindFirstValue(ClaimTypes.NameIdentifier)!;

    var userProfile = await _context.Users.Include(u => u.StudentProfile)
        .FirstOrDefaultAsync(u => u.Id == userId);

    if (userProfile == null || userProfile.StudentProfile == null)
        throw new UnauthorizedAccessException("Student profile not found");

    var offer = await _context.InternshipOffers
        .FirstOrDefaultAsync(o => o.Id == dto.InternshipOfferId)
        ?? throw new KeyNotFoundException("Internship offer not found");

    if (DateTime.UtcNow > offer.ApplicationDeadline)
        throw new BadRequestException("Internship offer past expiration date");
}

```

```
var alreadyApplied = await _context.InternshipApplications
    .AnyAsync(a => a.StudentId == userProfile.StudentProfile.Id &&
        a.InternshipOfferId == dto.InternshipOfferId);

if (alreadyApplied)
    throw new InvalidOperationException("Already applied");

var applicationDocument = await
    _documentUploadService.UploadApplicationDocumentAsync(dto.ApplicationDocument);

var application = new InternshipApplication()
{
    InternshipOfferId = dto.InternshipOfferId,
    StudentId = userProfile.StudentProfile.Id,
    ApplicationDocument = applicationDocument,
    PortfolioLink = dto.PortfolioLink,
    Content = dto.Content,
    CreatedAt = DateTimeOffset.UtcNow
};

await _context.InternshipApplications.AddAsync(application);
await _context.SaveChangesAsync();
}
```

Złóż aplikację

×

Dokumenty podać

CV / Życiorys

Browse...

No file selected.

Akceptowane formaty: PDF, DOC, DOCX (maks. 10MB)

Link do portfolio

Dodatkowe informacje

List motywacyjny / Wiadomość

Zamknij

Wyślij

Rysunek 6.6: Formularz składania podania na ofertę praktyki

Po zapisaniu podania staje się ono widoczne dla firmy oraz administratora, którzy mogą podjąć decyzję o przyjęciu lub odrzuceniu (rys. 6.7). Decyzja ta wpływa na stan podania, jednak nie jest ostateczna, ponieważ istnieje możliwość jej zmiany w przypadku pojawienia się nowych okoliczności.

Scarlett Howard
X

Status podania

Oczekuje na przegląd

Informacje o ofercie

Oferta:
Mobile App Intern

Informacje o studencie

Imię i nazwisko:
Scarlett Howard
Kierunek studiów:
Math Year 2

Materiały podań

Portfolio:
<https://portfolio.example.com/user8>

Dokument podania:
[application_8.pdf](#)

Zamknij Odrzuć Akceptuj

Rysunek 6.7: Widok okna opiniowania podania przez firmę lub administratora

```

private async Task UpdateApplicationStatusAsync(int id, ClaimsPrincipal user,
    InternshipApplicationStatus status)
{
    var userId = user.FindFirstValue(ClaimTypes.NameIdentifier)!;

    var application = await _context.InternshipApplications
        .Include(a => a.InternshipOffer)
        .FirstOrDefaultAsync(a => a.Id == id)
        ?? throw new KeyNotFoundException("Internship application not found.");

    var companyProfile = await _context.Companies
        .FirstOrDefaultAsync(c => c.UserId == userId);

    var isOwner = companyProfile?.Id == application.InternshipOffer.CompanyId;
    var isAdmin = user.IsInRole("Admin");

    if (!isOwner && !isAdmin)
        throw new UnauthorizedAccessException("You do not have permission to modify status
            for this application.");

    application.Status = status;
    await _context.SaveChangesAsync();
}

```

Moje podania

[Odśwież](#)

Emma Hughes
Cloud Engineering Intern

Emma Hughes
Software Engineering Intern

Emma Hughes
Product Management Intern

« 1 »

Rysunek 6.8: Widok złożonych podań studenta

Student ma dostęp do wszystkich swoich podań wraz z ich aktualnym stanem jak na rysunku 6.8. W przypadku wykrycia błędu w przesłanych danych lub zmiany decyzji, student może wycofać podania w oknie detali, które pokazano na rysunku 6.9.

Emma Hughes ×

Status podania

Oczekuje na przegląd

Informacje o ofercie

Oferta:
Software Engineering Intern

Informacje o studencie

Imię i nazwisko:
Emma Hughes

Kierunek studiów:
Economics Year 1

Materiały podań

Portfolio:
<http://localhost:5173/offers>

Dokument podania:
[📄 document-1-2.pdf](#)

Zamknij Wycofaj

Rysunek 6.9: Widok szczegółów podania

Aby zapobiec nadużyciom, wycofanie podania możliwe jest wyłącznie przed podjęciem decyzji przez firmę lub administratora. Uniemożliwia to ponowne składanie podań w celu obejścia procesu rekrutacyjnego.

```
public async Task DeleteAsync(int id, ClaimsPrincipal user)
{
    var application = await _context.InternshipApplications
        .Include(a => a.Student)
        .Include(internshipApplication => internshipApplication.InternshipOffer)
        .FirstOrDefaultAsync(a => a.Id == id)
        ?? throw new KeyNotFoundException($"Application not found.");

    if (application.Status != InternshipApplicationStatus.AwaitingReview)
        throw new InvalidOperationException("Can't withdraw application after decision has
            been made.");

    _context.InternshipApplications.Remove(application);
    await _context.SaveChangesAsync();
}
```

6.6 Bezpieczeństwo

System jest bezpieczny na tyle na ile bezpieczne są jego komponenty. Przykładowo, w React dane wprowadzone przez użytkownika są automatycznie sanityzowane, z wyjątkiem sytuacji, gdy zostaną przekazane do parametru `_dangerouslySetInnerHTML` elementu `div`. Podobnie jest po stronie backendu, gdzie framework automatycznie sanityzuje zapytania SQL przy wyszukiwaniu według tekstu.

Zagrożeniem dla aplikacji może być jednak niepoprawne nadanie uprawnień lub jednostronna walidacja, gdzie interfejs użytkownika blokuje pewne działanie, natomiast backend już nie. Najlepszym przykładem jest rola opiekuna praktyk, gdzie każdy opiekun powinien móc widzieć i zarządzać tylko grupami swoich studentów. Jest to szczególnie ważne w kontekście weryfikacji deklaracji praktyki, gdzie opiekun nie powinien mieć możliwości zaakceptowania deklaracji studenta należącego do innej grupy. Dlatego, gdy opiekun praktyk próbuje zaakceptować deklarację, serwer weryfikuje czy ma on uprawnienia do zarządzania studentem z danej grupy.

```
if (!user.IsInRole("Dean"))
{
    var managedFieldIds = await GetManagedFieldIdsForSupervisor(userId);

    if (!managedFieldIds.Contains(internship.Student.FieldOfStudyId) ||
        internship.Supervisor.UserId != userId)
        throw new UnauthorizedAccessException("Supervisor lacks permission to accept documents
            for this field");
}
```

Podobnych zabezpieczeń jest w aplikacji wiele. Stosowane są wszędzie tam, gdzie użytkownicy danej roli powinni mieć dostęp wyłącznie do własnych da-

nych. Mechanizm ten zapobiega nieautoryzowanemu dostępowi nawet przy próbach manipulacji żadaniami HTTP.

Rozdział 7

Testowanie

7.1 Hostowanie aplikacji

Aby umożliwić publiczne testowania aplikacji i zapewnić, że działa poprawnie nawet w środowisku produkcyjnym, jest ona hostowana w chmurze. Do hostowania bazy danych, serwera plików oraz backendu wykorzystano Microsoft Azure, natomiast frontend jest hostowany na platformie DigitalOcean.

Aplikacja jest dostępna pod adresem: <https://s-ops.app/>. Ponieważ aplikacja działa na zasadzie hierarchii ról, istnieje możliwość przetestowania działania każdej z nich w rzeczywistych warunkach. Każda z ról ma co najmniej jedno konto testowe z hasłem Password123. Poniżej przedstawiono dostępne konta:

Firma (Company) :

- company1@example.com
- company2@example.com
- company3@example.com

Student (Student) :

- student1@example.com
- student2@example.com
- student3@example.com
- student4@example.com
- student5@example.com
- student6@example.com

Opiekun praktyk (Supervisor) :

- admin3@example.com

- admin4@example.com

Dziekan (Dean) :

- admin1@example.com

Administrator (Admin) :

- admin2@example.com

7.2 Scenariusze testowe

Celem testów było sprawdzenie poprawności działania systemu w zakresie kluczowych funkcji dostępnych dla poszczególnych ról użytkowników. Testy zostały przeprowadzone zarówno z poziomu interfejsu użytkownika, jak i poprzez bezpośrednie wysyłanie żądań HTTP do API, co pozwoliło zweryfikować zabezpieczenie backendu.

Firma :

- Przeglądanie ofert – użytkownik przegląda listę ofert oraz próbuje edytować ofertę innej firmy; operacja zostaje zablokowana (wynik: OK).
- Decyzja podań – użytkownik podejmuje decyzję dla podań przypisanych do własnych ofert; próba modyfikacji podań innych ofert kończy się błędem (wynik: OK).

Student :

- Wysyłanie podań – student wysyła podanie na ofertę; podanie zostaje zapisane w systemie. Próba ponownego wysłania skutkuje błędem (wynik: OK).
- Wycofanie podania – student wycofuje podanie przed podjęciem decyzji przez firmę; operacja kończy się sukcesem (wynik: OK).
- Przesyłanie dokumentów – student przesyła dokument praktyki; dokument zostaje zapisany i przypisany do praktyki (wynik: OK).
- Wypełnianie dziennika – student dodaje wpisy w trakcie trwania praktyki; po jej zakończeniu edycja zostaje zablokowana (wynik: OK).

Opiekun praktyk :

- Widoczność studentów – opiekun przegląda praktyki przypisanych studentów; brak dostępu do innych studentów (wynik: OK).

- Akceptacja deklaracji – opiekun akceptuje lub odrzuca dokument deklaracji; operacja możliwa tylko dla przypisanych praktyk (wynik: OK).

Dziekan :

- Widoczność praktyk – dziekan przegląda wszystkie praktyki w systemie (wynik: OK).
- Akceptacja dokumentów – dziekan akceptuje i odrzuca dokumenty praktyk niezależnie od przypisania (wynik: OK).

Administrator :

- Zarządzanie kontami – administrator tworzy nowe konto użytkownika; konto zostaje zapisane w systemie (wynik: OK).
- Operacje w imieniu użytkowników – administrator wykonuje akcje w imieniu innych ról (np. tworzenie praktyki, przesyłanie dokumentów); operacje kończą się sukcesem (wynik: OK).

Wszystkie przeprowadzone scenariusze testowe zakończyły się powodzeniem. Nie wykryto błędów uniemożliwiających korzystanie z systemu.

Rozdział 8

Propozycje rozwoju

Podczas realizacji projektu zidentyfikowano kilka funkcji, które mogłyby zostać rozwinięte.

Wsparcie wielu miejsc pracy – istnieją przypadki, gdy student może realizować praktykę w więcej niż jednej instytucji. Obecnie system zakłada jedynie jedno miejsce praktyki. Rozszerzenie systemu o możliwość obsługi wielu miejsc praktyk pozwoliłoby na uwzględnienie takich scenariuszy.

Integracja z Active Directory – podczas dyskusji nad projektem pojawiła się propozycja integracji aplikacji z uczelnianym systemem Active Directory. To rozwiązanie umożliwiłoby studentom i pracownikom logowanie się za pomocą istniejących kont uczelnianych. Z perspektywy administracyjnej uprościłoby to zarządzanie użytkownikami, ponieważ system mógłby automatycznie odczytywać użytkowników bez konieczności ich ręcznego dodawania. Realizacja tego rozwiązania wykracza poza zakres projektu, ponieważ wymaga dostępu do infrastruktury uczelni.

Powiadomienia – system mógłby zostać rozszerzony o funkcję automatycznych powiadomień poprzez interfejs lub pocztę e-mail. Użytkownicy byłiby informowani o zmianach stanów praktyk, nowych ofertach oraz zbliżających się terminach.

Logowanie zdarzeń – istotnym elementem byłoby wprowadzenie systemu rejestrowania działań użytkowników. W szczególności w kontekście praktyk pozwoliłoby to na śledzenie zmian oraz określenie, kto i kiedy dokonał danej operacji.

Zwiększenie bezpieczeństwa – możliwe jest wdrożenie dodatkowych mechanizmów zabezpieczających, takich jak blokowanie podejrzanych adresów IP czy czasowe blokady kont po wielokrotnych nieudanych próbach logowania.

Zarządzanie danymi w większej skali – system mógłby zostać rozbudowany o operacje masowe, takie jak przypisanie studentów do grup czy tworzenie większej liczby praktyk w jednej operacji.

Automatyzacja tworzenia praktyk – możliwe jest wprowadzenie mechanizmu tworzenia praktyk dla określonych grup studentów (np. dla konkretnego roku studiów) w ustalonym terminie, co ograniczyłoby konieczność ręcznego zarządzania tym procesem.

Rozdział 9

Podsumowanie

W ramach pracy licencjackiej opracowano prototyp systemu zarządzania praktykami studenckimi. System ten obejmuje kluczowe procesy związane z realizacją praktyk, zgodnie z przyjętym regulaminem studiów, w tym: obsługę przebiegu praktyki oraz zarządzanie ofertami praktyk. Jako aplikacja działająca w ekosystemie World Wide Web, system wyposażony został w mechanizmy zarządzania użytkownikami oraz rolami wraz z przypisanymi uprawnieniami.

Zaimplementowane rozwiązanie pozwala na cyfryzację wielu procesów, które w tradycyjnym podejściu realizowane są w sposób ręczny. Dzięki temu możliwe jest usprawnienie przepływu informacji pomiędzy studentami, firmami oraz pracownikami uczelni, a także zwiększenie kontroli nad przebiegiem praktyk.

System został zaprojektowany jako prototyp, dlatego w przypadku wdrożenia w środowisku produkcyjnym wymaga dalszego rozwoju oraz dopracowania niezawodności i zwiększenia skalowalności. Pomimo tego, już obecna wersja pozwala na ocenę korzyści wynikających z cyfryzacji procesu praktyk oraz pokazuje potencjał zastosowania tego typu rozwiązań w środowisku akademickim.

Spis rysunków

5.1	Diagram tabeli <code>AspNetUsers</code> oraz jej powiązania	12
5.2	Diagram tabeli <code>Internships</code> oraz jej powiązania	13
5.3	Diagram tabeli <code>InternshipApplications</code> oraz jej powiązania	13
5.4	Struktura tabel bazy danych	14
5.5	Diagram przypadków użycia	16
5.6	Diagram BPMN przedstawiający proces szukania miejsca do realizacji praktyki	17
5.7	Diagram BPMN przedstawiający przebieg praktyki w systemie	17
6.1	Weryfikacja praktyki przez opiekuna lub dziekana	24
6.2	Dziennik praktyk studenta	25
6.3	Lista ofert przypisanych do firmy	26
6.4	Edycja oferty praktyki	27
6.5	Przegląd ofert praktyk	28
6.6	Formularz składania podania na ofertę praktyki	30
6.7	Widok okna opiniowania podania przez firmę lub administratora	31
6.8	Widok złożonych podań studenta	32
6.9	Widok szczegółów podania	32

Bibliografia

- [1] ASP.NET Core – dokumentacja
<https://learn.microsoft.com/en-us/dotnet/core/introduction>
Dostęp na dzień: 2026.06.03
- [2] Data Transfer Object – Wikipedia
https://en.wikipedia.org/wiki/Data_transfer_object
Dostęp na dzień: 2026.04.20
- [3] Entity Framework – dokumentacja
<https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>
Dostęp na dzień: 2026.06.03
- [4] FormData – Mozilla Docs
<https://developer.mozilla.org/en-US/docs/Web/API/FormData>
Dostęp na dzień: 2026.04.28
- [5] Git – Wikipedia
<https://en.wikipedia.org/wiki/Git>
Dostęp na dzień: 2026.04.01
- [6] i18n – Introduction
<https://www.i18next.com>
Dostęp na dzień: 2026.04.01
- [7] JSON Web Token – Wikipedia
https://en.wikipedia.org/wiki/JSON_Web_Token
Dostęp na dzień: 2026.04.20
- [8] React – dokumentacja
<https://react.dev/reference/react>
Dostęp na dzień: 2026.06.03
- [9] TanStack Query – dokumentacja
<https://tanstack.com/query/latest/docs/framework/react/overview>
Dostęp na dzień: 2026.04.01

-
- [10] Zasady praktyk i dokumenty – Filia UwB
<https://wilno.uwb.edu.pl/biuro-karier-i-promocji1/praktyka-i-kariera/zasady-praktyk-i-dokumenty>
Dostęp na dzień: 2026.04.02