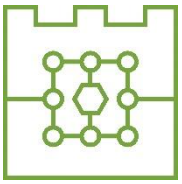
**Politechnika Krakowska**
**im. Tadeusza Kościuszki**
Wydział Informatyki i Telekomunikacji



**Kamil Syktus**
Numer albumu: 139843

# Aplikacja blogowa z wykorzystaniem architektury API oraz niezależnych interfejsów dla postujących i czytelników

# Blog application using API architecture and independent interfaces for posting and readers

**Praca inżynierska**
**na kierunku Informatyka**

Praca wykonana pod kierunkiem:
**dr Radosław Kycia**

**Kraków, 2024**

# Spis treści

# 1. Wstęp

## 1.1 Wprowadzenie

W dzisiejszych czasach powszechnego dostępu do internetu szybka wymiana informacji oraz dzielenie się nią z innymi odgrywa ważną rolę w naszym codziennym życiu. Jedną z form takiej komunikacji są blogi, których autorzy skupiają się na danej tematyce i przyciągają czytelników, zainteresowanych omawianym tematem. Aktualnie można zauważyć, że blogi są prowadzone nie tylko przez pojedyncze osoby, ale także przez firmy czy specjalistów pragnących zbudować swoją własną markę i zdobyć szerokie grono klientów. „W tak dynamicznie zmieniającym się środowisku biznesowym, tworzy się wiele nowych nisz rynkowych, dlatego warto zastanowić się nad założeniem własnego e-pamiętnika, który nie tylko będzie miejscem wymiany doświadczeń i prezentowania własnych zainteresowań, ale także bardzo skutecznym modelem biznesowym (…).” [1]. Jednocześnie dynamiczny rozwój technologiczny oraz zmieniające się preferencje użytkowników stawia przed tą branżą spore wyzwania. Jednym z takich wyzwań jest wprowadzenie bardziej innowacyjnych i intuicyjnych rozwiązań, które pozwolą aplikacjom blogowym zbliżyć się funkcjonalnością do mediów społecznościowych, np. poprzez wprowadzenie czatu umożliwiającego lepszą interakcję między użytkownikami. Twórcom blogów z pewnością zależy na prostej aplikacji, a jednocześnie mającej wiele zaawansowanych funkcji edytowania oraz zarządzania treścią. Dodatkowo ważne również jest zapewnienie odpowiedniej autentyczności, gdyż aktualnie bardzo łatwo oszukać czytelnika nieprawdziwymi informacjami szerząc tym samym dezinformację. To wszystko przemawia za stworzeniem aplikacji łączącej cechy zarówno standardowej aplikacji blogowej, jak i aplikacji typu social media.

## 1.2 Porównanie podobnych rozwiązań

Przeglądając najpopularniejsze narzędzia do blogowania dostępne w internecie można natknąć się na takie platformy jak Medium, HubPages czy Dev.to. Każda z nich wyróżnia się własną filozofią i funkcjonalnościami, przyciągając zarówno indywidualnych twórców, jak i grupy odbiorców zainteresowanych konkretnymi tematami. Poniżej zostały przedstawione i przeanalizowane wymienione wyżej platformy, aby pokazać ich możliwości, ograniczenia, a także wskazać kluczowe różnice względem aplikacji będącej przedmiotem tej pracy.

### 1.2.1 Medium

Medium [2] to platforma blogowa, która powstała z myślą o dzieleniu się treściami wysokiej jakości, które mają poprawiać zrozumienie świata poprzez siłę tworzenia takich tekstów. Aplikacja ta stawia na prostotę i estetykę, zawiera minimalisty edytor oraz schludny, nowoczesny wygląd postów. Skierowana jest zarówno do profesjonalnych autorów, jak i do amatorów pragnących dzielić się swoją wiedzą z szerokim gronem odbiorców. Sama aplikacja ma obecnie ponad milion użytkowników z wykupioną licencją, a miesięcznie odwiedza ją ponad 100 milionów osób.

Zalety tej aplikacji to m.in.:

- prosty i intuicyjny interfejs,
- wysoka jakość treści i dbałość o ich wiarygodność,
- możliwość zarabiania na tworzeniu własnych postów.

Do wad można zaliczyć:

- widoczność treści uzależniona od algorytmów rekomendacji,
- duża ilość postów, do których dostęp uzyskany jest dopiero po wykupieniu płatnej licencji,
- możliwość tworzenia własnych postów również wymaga licencji,
- brak bardziej rozbudowanego systemu filtracji postów.

### 1.2.2 HubPages

HubPages [3] to platforma dająca możliwość dzielenia się własnymi doświadczeniami, wiedzą i poradnikami w bardzo szerokim zakresie kategorii. Aplikacja ta przyciąga wielu użytkowników z różnych dziedzin i zainteresowań głównie ze względu braku skupienia się na pojedynczej tematyce, a bardziej rozdzieleniu aplikacji na pojedyncze kategorie. Prócz wielu ciekawych artykułów, użytkowników będących autorami postów przyciąga także możliwość monetyzacji dzięki reklamom wyświetlanym na ich postach. Autorzy aplikacji przekonują o miesięcznej liczbie odwiedzających w wysokości 35 milionów osób.

Do mocnych stron tej aplikacji należą m.in.:

- szeroka gama kategorii, co pozwala na łatwe grupowanie treści,
- możliwość zarabiania na tworzeniu własnych postów,
- kontrolowanie treści dodawanych na platformie.

Wadami są zdecydowanie:

- mniej intuicyjny interfejs oraz przestarzały edytor postów,
- brak interakcji między użytkownikami, komentowanie jest dostępne tylko dla wybranych postów, co sprawia że platforma jest bardziej statyczna.

### 1.2.3 Dev.to

Ostatnią z omawianych aplikacji jest serwis Dev.to [4]. Aplikacja ta skupia na sobie środowisko programistów i ma na celu pomagać w nauce, wspierać wspólne rozwiązywanie problemów oraz dzielić się wiedzą. W odróżnieniu od poprzednich dwóch aplikacji, dev.to dotyczy tylko jednej grupy odbiorców, tym samym postawiono na brak kategorii, a danie możliwości dodawania tagów, pozwalających na łatwiejsze grupowanie, bądź znalezienie postów.

Zalety aplikacji Dev.to:

- skupienie się na jednej niszy – programowaniu i technologii, co pozwala na łatwe nawiązywanie kontaktów między osobami z tymi samymi zainteresowaniami,
- prosty i przejrzysty interfejs publikacji oraz wyświetlania postów,
- kod źródłowy aplikacji dostępny dla wszystkich zachęca do tworzenia własnych, podobnych aplikacji.

Do minusów można zaliczyć:

- brak bardziej rozbudowanej możliwości filtracji postów,
- ograniczona możliwość dotarcia do większej ilości odbiorców ze względu na trudność w tworzeniu unikalnych treści w tak zamkniętym środowisku

### 1.2.4 Podsumowanie omówionych rozwiązań

Podsumowując, każda z wyżej omówionych aplikacji ma swoje wady i zalety, a poprzez swoje funkcjonalności odpowiada na potrzeby różnych twórców treści i czytelników. *Medium* wyróżnia się swoim nowoczesnym wyglądem i intuicyjnym edytorem, skupiając się na promowaniu wysokiej jakości postów, ale tracąc jednocześnie na wymaganiu wykupienia członkostwa do ich dużej ilości. *HubPages*, z kolei, skupia się bardziej na prostych, praktycznych treściach, dając możliwość zarabiania, dzieląc się swoimi doświadczeniami w aplikacji. Niestety, zmniejszona możliwość wymiany swoich wrażeń między użytkownikami oraz nieco prymitywny edytor zmniejsza przyjemność z korzystania z tej aplikacji, która zdaje się być bardziej statyczną. *Dev.to* to platforma

skupiona przede wszystkim na programistach i entuzjastach technologii, co daje poczucie zaangażowanej społeczności, jednak zmniejsza możliwość dotarcia do szerszego grona odbiorców, spoza tej niszy.

Aplikacja będąca przedmiotem tej pracy czerpie inspiracje z najlepszych cech powyższych platform, jednocześnie odpowiada na luki między nimi poprzez rozszerzenie o dodatkowe, brakujące elementy, które zostaną omówione w kolejnych rozdziałach.

# 2. Cel, zakres i metodyka pracy

## 2.1 Cel

Celem tej pracy jest zaprojektowanie oraz implementacja aplikacji blogowej połączonej z systemem czatu dla użytkowników działającym w czasie rzeczywistym. Projekt ma za zadanie nie tylko zawrzeć standardowe funkcje bloga, takie jak tworzenie i zarządzanie postami, kategoriami i komentarzami, ale również wprowadzenie nowych funkcjonalności, które zwiększyły możliwość interakcji między użytkownikami systemu poprzez czat oparty o działanie protokołu komunikacyjnego WebSocket.

## 2.2 Zakres

W kolejnych rozdziałach szczegółowo zostaną omówione poszczególne etapy tworzenia projektu. W rozdziale trzecim opisane zostaną kluczowe koncepcje teoretyczne, takie jak architektura aplikacji wraz z użytymi technologiami oraz wzorcami projektowymi. W rozdziale czwartym przedstawiona zostanie analiza wymagań, a także założeń, które aplikacja ma spełniać. Następnie, w piątym rozdziale przedstawiony będzie proces projektowania całego systemu. Kolejno, w rozdziale szóstym zawarty będzie proces implementacji, w tym szczegółowe omówienie struktury części backendowej i frontendowej, a także etapu testowania aplikacji. Na koniec podsumowana zostanie całość wykonanych czynności, a także możliwe dalsze ścieżki rozwoju aplikacji.

## 2.3 Metodyka

Do realizacji projektu zastosowano model kaskadowy, który zakłada tworzenie każdego z etapów oprogramowania w konkretnej kolejności, jeden po drugim. Przed przejściem do kolejnej fazy, poprzednia musi zostać zakończona, co pozwala na dopracowanie każdego z elementów systemu. Poszczególne etapy, na które składa się omawiany projekt to:

- analiza wymagań – szczegółowo zidentyfikowane zostały wymagania funkcjonalne i niefunkcjonalne aplikacji dzięki czemu łatwiej było określić cele i podstawy do dalszej pracy,

- projektowanie – stworzono projekt systemu, zawierający opis przypadków użycia, struktury bazy danych oraz sposobów zapewnienia odpowiedniego poziomu bezpieczeństwa aplikacji,
- implementacja – na podstawie stworzonego projektu zaimplementowano aplikację,
- testowanie – po zakończeniu fazy implementacji przeprowadzono testy, które zweryfikowały poprawność działania aplikacji,
- dokumentacja – ostatni etap obejmował sporządzenie dokumentacji zawierającej opis wszystkich elementów systemu.

# Część teoretyczna

# 3. Architektura aplikacji i technologie

Aplikacja opiera się o architekturę typu klient-serwer, gdzie rolę klienta pełni frontend, a rolę serwera – backend. Część frontendowa odpowiada za interfejs użytkownika, czyli to, co widzą użytkownicy aplikacji oraz to co mogą wykonywać. Natomiast część backendowa przetwarza te żądania, zapewnia autoryzację i autentykację oraz komunikuje się z bazą danych zwracając odpowiednie dane. Całość tworzy spójny system, który zapewnia wydajność i łatwość w rozwijaniu. Warto przyjrzeć się każdej z poszczególnych warstw bardziej szczegółowo, aby lepiej zrozumieć sposób działania tej aplikacji, a także poznać technologie wykorzystane do ich implementacji oraz testowania.

## 3.1 Warstwy aplikacji

### 3.1.1 Warstwa prezentacji (frontend)

Warstwa prezentacji odpowiada za dynamiczne generowanie treści dla użytkownika i zapewnia interfejs umożliwiający wykonywanie wszystkich działań określonych w wymaganiach funkcjonalnych aplikacji. Do implementacji tej części wykorzystano framework Angular [5], który jest obecnie jednym z najpopularniejszych narzędzi do budowy aplikacji typu SPA (ang. Single Page Application). Angular zapewnia aplikacji:

- intuicyjność interfejsu – dzięki użyciu Angular Material, który dostarcza gotowe komponenty, takie jak przyciski, listy, czy menu, pozwalając na osiągnięcie jak najlepszych wrażeń w trakcie korzystania z aplikacji,
- dynamiczność – nie ma potrzeby przeładowywania całej strony, ponieważ dane są automatycznie odświeżane,
- zarządzanie komunikacją z backendem – dzięki żądaniom HTTP dane z serwera są pobierane i obsługiwane w serwisach, co poprawia czytelność kodu.

Interfejs użytkownika był tworzony z wykorzystaniem gotowych komponentów biblioteki Angular Material oraz wielu komponentów stworzonych od podstaw w oparciu o SASS, będący preprocesorem CSS, dzięki czemu tworzenie styli zostało lepiej uporządkowane i szybciej utworzone.

Testy tej części zrealizowane zostały dzięki wbudowanym w framework Angular narzędziom:

- Jasmine [6] – framework używany do testowania funkcji i komponentów,

- Karma [7] – narzędzie, które uruchamia utworzone testy w przeglądarce zapewniając intuicyjne raporty w czasie rzeczywistym.

### 3.1.2 Warstwa logiki biznesowej (backend)

Backend, oparty na frameworku ASP.NET Core [8], jest używany jako serwer, to tutaj decydowane jest jaki użytkownik ma dostęp do czego i jak dane są przetwarzane. Komunikacja z frontendem opiera się o działanie RESTful API, które jest bardzo popularnym stylem architektonicznym, pozwalającym na efektywną komunikację z klientem. Wykorzystano również interfejs ASP.NET Core Identity, który pomaga w obsługiwaniu logowania, rejestracji i wszelkiej obsługi użytkowników, jak np. szyfrowanie ich haseł, czy obsługa ról w systemie. Nieco osobną częścią jest czat, w którym komunikacja, zaimplementowana za pomocą biblioteki SignalR [9], pozwala na wysyłanie wiadomości w czasie rzeczywistym za pomocą protokołu WebSocket. Dzięki SignalR użytkownicy mogą łatwo i wygodnie komunikować się ze sobą bez konieczności odświeżania strony czatu.

Do realizacji testów na backendzie użyto frameworku xUnit [10], który jest jednym z najpopularniejszych narzędzi do testowania kodu w .NET oraz bibliotekę Moq [11], która pozwala na tworzenie tzw. „mocków". Symulują one działanie wybranych zależności aplikacji, dzięki czemu testy są odseparowane od używanej bazy danych i logiki biznesowej.

### 3.1.3 Warstwa przechowywania danych (baza danych)

Warstwę przechowywania danych można uznać za fundament aplikacji. Tutaj przechowywane są wszystkie dane użytkowników i aplikacji. W projekcie wybrano bazę danych PostgreSQL [12], która została połączona z aplikacją za pomocą biblioteki Entity Framework Core [13], która jest maperem relacyjno-obiektowym (ORM, ang. Object–relational mapping). „Zastosowanie technologii ORM, gwarantuje implementację podstawowych operacji, takich jak: dodawanie, odczyt, modyfikacja, usuwanie nazywanymi operacjami typu CRUD (ang. create, read, update, delete)." [14]. Dzięki takiemu podejściu operowanie na bazie danych może odbywać się w sposób obiektowy, gdyż wszelkie zapytania SQL są generowane na podstawie kodu napisanego w języku C#.

## 3.2 Wzorce projektowe

Wzorce projektowe są sprawdzonymi rozwiązaniami problemów, które powszechnie powtarzają się w procesie projektowania oprogramowania. Pomagają one programistom tworzyć kod w szybszy sposób, a co za tym idzie, kod ten staje się prostszy i nadaje się do wielokrotnego użytku. Książka „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku" autorstwa „Gangu Czworga" (Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides) jest uznawana za klasyczne dzieło w tej dziedzinie, opisujące 23 wzorce, które mogą być stosowane w różnych sytuacjach w programowaniu obiektowym. Autorzy tej książki idealnie podsumowali dlaczego warto używać wzorców projektowych: „Pomaga to projektantom wielokrotnie używać udanych projektów przez oparcie nowych rozwiązań na wcześniejszych doświadczeniach. Projektant znający takie wzorce może po natrafieniu na problem natychmiast je wykorzystać, zamiast wymyślać od nowa." [15].

### 3.2.1 Wzorzec repozytorium

W warstwie backendowej zastosowano wzorzec repozytorium [16], który pozwala na oddzielenie logiki biznesowej od logiki dostępu do danych, zwiększając czytelność
i ułatwiając przyszłe modyfikacje kodu. Dzięki temu serwer operuje na serwisach, które zarządzają zapytaniami do bazy danych. Każda encja, jak na przykład post, czy komentarz posiada swój odpowiadający serwis, który zawiera metody obsługujące metody na danej encji.

### 3.2.2 Wzorzec wstrzykiwania zależności

Kolejnym kluczowym, wykorzystywanym wzorcem jest wzorzec wstrzykiwania zależności [17], który umożliwia dynamiczne przekazywanie zależności, takich jak interfejsy, konfiguracje itd. do poszczególnych komponentów aplikacji w momencie ich tworzenia. Dzięki takiemu podejściu kod staje się bardziej modularny oraz łatwiejszy w testowaniu, ponieważ możemy w łatwy sposób zastąpić prawdziwe zależności ich odpowiednikami używanymi do testów.

### 3.2.3 Wzorzec singleton

Singleton [15] jest wzorcem projektowym, zapewniającym, że dana klasa ma dokładnie jedną instancję w całej aplikacji. W omawianej aplikacji, wzorzec ten jest używany wraz z biblioteką SignalR do poprawnej obsługi połączeń z klientami (użytkownikami) czatu w czasie rzeczywistym. Dzięki zastosowaniu singletona każde połączenie jest zarządzane przez pojedynczą instancję, co pozwala na efektywną obsługę wiadomości w czacie.

## 3.3 Bezpieczeństwo aplikacji

Bezpieczeństwo aplikacji jest jednym z najważniejszych elementów każdego systemu, tym bardziej w aplikacji internetowej, która przechowuje dane użytkowników i udziela dostępu do różnych zasobów. Omawiana aplikacja zawiera kilka mechanizmów, które mają na celu ochronę zarówno użytkowników, jak i samej aplikacji. Poniżej szczegółowo opisano działanie uwierzytelniania, autoryzacji oraz innych środków mających zapewnić tę ochronę.

### 3.3.1 Uwierzytelnianie

Uwierzytelnianie (autentykacja) jest procesem, który służy do weryfikacji tożsamości użytkownika i upewnienia się, że jest on tym za kogo się podaje. W projekcie działanie tego procesu opiera się na użyciu ASP.NET Identity, które oferuje już gotowe i wysoce bezpieczne mechanizmy zarządzania użytkownikami wraz z ich danymi.

W trakcie rejestracji użytkownika, jego hasło jest zapisywane w bazie danych w postaci zaszyfrowanej. Do szyfrowania stosowany jest algorytm PBKDF2, który zapewnia dodatkowe bezpieczeństwo przed atakami typu brute force, ponieważ wykorzystuje wielokrotne iteracje funkcji mieszającej. Do każdego hasła, przed jego zaszyfrowaniem dodawana jest tzw. sól, która jest losowym ciągiem znaków, zapewniającym, że dwa takie same hasła będą w bazie danych inaczej zaszyfrowane, co dodatkowo utrudnia ataki słownikowe. Następnie, kiedy użytkownik loguje się do aplikacji (podając adres e-mail i hasło), system znajduje danego użytkownika za pomocą wprowadzonego adresu e-mail, potem proces szyfrowania hasła, opisany wyżej, jest powtarzany i wprowadzone hasło jest porównywane z tym zapisanym w bazie danych. Jeżeli wszystko się zgadza, system generuje token autoryzacyjny (JWT, ang. JSON Web

Token) i przekazuje go klientowi. Następnie ten token wykorzystywany jest do obsługi autoryzacji żądań.

### 3.3.2 Autoryzacja

Autoryzacja polega na określaniu do jakich zasobów aplikacji użytkownik ma dostęp oraz udzielanie tego dostępu. W celu wsparcia tego procesu system zawiera trzy role użytkowników, opisanych w rozdziale 4.1.

Każde żądanie HTTP wysyłane do serwera zawiera unikalny token JWT przypisywany przy każdym logowaniu do aplikacji. Na podstawie tego tokenu i zawartych w nim informacji takich jak przypisana rola, serwer decyduje, czy dostęp powinien zostać przydzielony czy zablokowany. Należy także wspomnieć, że token ten ma odpowiedni czas ważności oraz specjalny podpis, które również weryfikuje serwer. Do zalet autoryzacji opartej na tokenach JWT należy:

- redukcja obciążenia serwera – ze względu na bezstanowość tokenów JWT, serwer nie musi przechowywać informacji o sesji użytkownika,
- wysokie bezpieczeństwo – token jest zaszyfrowany za pomocą algorytmu HMAC SHA256 oraz podpisany cyfrowo.

### 3.3.3 Pozostałe środki bezpieczeństwa

Prócz odpowiedniej autentykacji i autoryzacji aplikacja wprowadza także inne metody zapewniające dodatkowe bezpieczeństwo.

Pierwszą z nich jest odpowiednia walidacja danych wejściowych. Aplikacja zawiera wiele różnych formularzy, które są weryfikowane zarówno po stronie frontendu, jak i backendu. Walidacja po stronie klienta poprawia doświadczenia użytkownika, wyświetlając odpowiednie komunikaty w trakcie uzupełniania pól, a po stronie serwera zapewnia, że dane wejściowe są rzeczywiście poprawne.

Kolejną z takich metod jest wymuszenie odpowiednio silnych haseł użytkowników. System wymaga, aby hasło użytkownika posiadało co najmniej dziesięć znaków, w tym jedną i dużą literę, cyfrę oraz znak specjalny. „Hasła krótsze niż 10 znaków są zwykle uważane za zbyt słabe.” [18].

Dodatkowo warto także wspomnieć, że w frameworku Angular automatycznie, w żądaniach HTTP, są dodawane tokeny CSRF, które chronią aplikację przed nieautoryzowanym wykonaniem jakiejś akcji przez nieodpowiednie osoby.

# Część praktyczna

# 4. Analiza wymagań aplikacji

Etap wytwarzania oprogramowania zwykle zaczyna się od analizy, która ma za zadanie poznać i opisać określone przez użytkownika wymagania, powiązanie ich oraz na tej podstawie dokładne opisanie co system ma robić. „Zawartość specyfikacji wymagań powinna dokładnie określać, co oprogramowanie ma robić, nie powinna jednak narzucać, jak ma być zbudowane.” [19]. Następujący rozdział szczegółowo omawia wymagania aplikacji oraz założenia techniczne, które zostały przyjęte podczas projektowania systemu, a także przedstawienie ról użytkowników dostępnych w nim.

## 4.1 Charakterystyka użytkowników aplikacji

Użytkownikami omawianego w tej pracy systemu są osoby, które korzystają z blogów w celu czytania lub publikowania treści, a także wzajemnej interakcji poprzez dostępny dla nich czat lub komentarze. W tym celu, aby zapewnić płynne i bezpieczne poruszanie się po aplikacji wyodrębnione zostały następujące grupy użytkowników:

- użytkownik standardowy (rola przypisana w systemie - User) to osoba, która ma dostęp do przeglądania postów, może je komentować oraz czatować z innymi użytkownikami, ale nie może dodawać własnych treści,
- użytkownik publikujący (rola przypisana w systemie – PostingUser) rozszerza użytkownika standardowego o możliwość publikacji własnych postów oraz zarządzania nimi, kluczowe jest zapewnienie odpowiedniej wygody w tych działaniach oraz możliwość kategoryzowania postów,
- administrator (rola przypisana w systemie – Admin) jest osobą mającą dostęp do wszystkich funkcjonalności aplikacji, odpowiada za jej nadzór.

Powyższy podział zapewnia odpowiednie poziomy dostępu, dostosowując się do potrzeb każdego typu użytkownika.

## 4.2 Wymagania funkcjonalne

Wymagania funkcjonalne to zestaw kluczowych funkcji, które powinna spełniać aplikacja, aby realizować zamierzone cele. Zgodnie z tym, na etapie analizy zostały wyznaczone odpowiednie funkcjonalności podzielone według typów użytkowników opisanych w poprzednim podrozdziale.
Dla użytkowników standardowych:

- proces rejestracji i logowania,
- przeglądanie wszystkich postów,
- filtrowanie postów (po tytule, po kategorii, względem daty utworzenia),
- możliwość wybrania ilości postów wyświetlanych na jednej stronie (10, 20, 50)
- możliwość przeczytania wybranego postu,
- wyrażanie swojej opinii o poście poprzez możliwość jego komentowania,
- rozpoczęcie czatu z innymi użytkownikami systemu,
- zmienianie swojego hasła,
- zmienianie swojego zdjęcia profilowego.

Dla użytkowników piszących posty:

- wszystkie funkcjonalności standardowego użytkownika,
- tworzenie nowych postów,
- zarządzanie swoimi postami poprzez możliwość edytowania lub usuwania.

Dla administratora:

- możliwość zarządzania kategoriami (tworzenie, usuwanie),
- możliwość zarządzania użytkownikami (modyfikowanie roli w systemie, usuwanie konta),
- możliwość moderowania treści udostępnianych w aplikacji (usuwanie i edycja postów, usuwanie komentarzy).

## 4.3 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają cechy, które pozwalają aplikacji działać w wydajny i wygodny sposób. Mówiąc inaczej, określają one pewne granice w jakich powinny działać funkcjonalności systemu.

Wymagania niefunkcjonalne omawianej aplikacji:

- zapewnienie wydajności – czat w czasie rzeczywistym oraz obsługa zapytań do bazy danych powinna działać płynnie, bez większych opóźnień,
- zapewnienie bezpieczeństwa danych – hasła użytkowników powinny być przechowywane w postaci zahashowanej,
- autoryzacja - tokeny JWT mają odpowiadać za bezpieczne przyznawanie dostępu do określonych zasobów i chronić przed nieautoryzowanym dostępem,

- schludny design i responsywność – interfejs użytkownika musi być przystosowany do działaniach na różnych typach urządzeń,
- skalowalność – aplikacja powinna być zaimplementowany w sposób, który umożliwi bezproblemowe dodawanie nowych funkcjonalności bez konieczności znacznych zmian w istniejącym już kodzie.

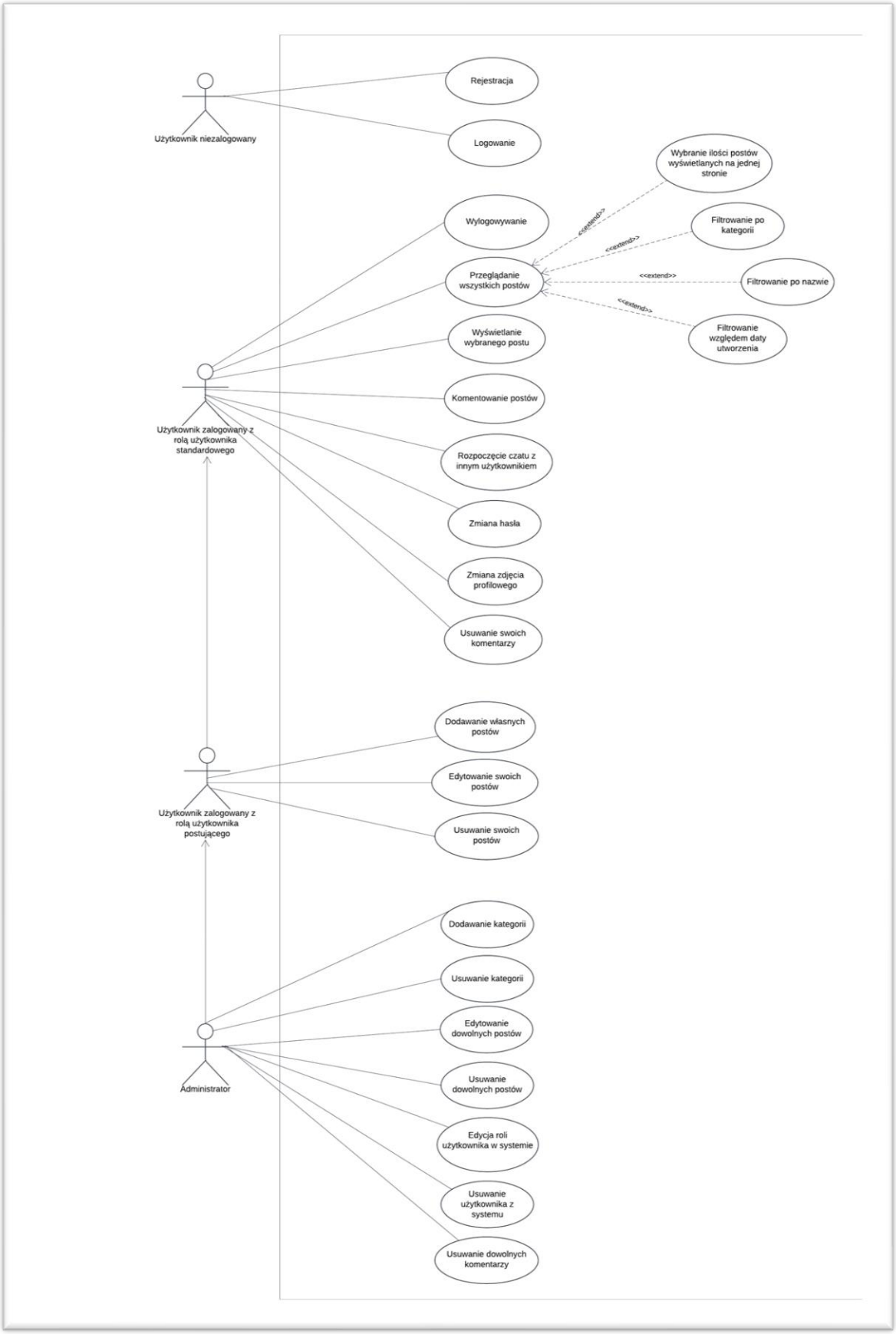
# 5. Projekt systemu

Po zakończeniu etapu analizy wymagań można przejść do fazy projektowania całej aplikacji. W tym celu, w następującym rozdziale zostaną wyodrębnione i opisane kluczowe elementy dla projektu aplikacji:

- przypadki użycia – najpierw w formie diagramu, a następnie w sposób opisowy, aby dokładnie pokazać możliwe interakcje użytkowników z systemem, będą one uwzględniały różnych aktorów, którymi są poszczególne role użytkowników aplikacji,
- opis bazy danych – obejmujący diagram ERD, relacje pomiędzy tabelami oraz strukturę przechowywanych danych,

## 5.1 Przypadki użycia

Na rys. 5.1 przedstawiono diagram przypadków użycia, który pozwala wskazać wszystkie sposoby używania aplikacji przez użytkowników.




Rys. 5.1. Diagram przypadków użycia (źródło: Opracowanie własne)

### 5.1.1 Opisy przypadków użycia

1. Rejestracja użytkownika

   **Aktor:** Użytkownik niezarejestrowany

   **Ścieżka główna:**

   1.1. Aktor wybiera stronę rejestracji.

   1.2. System wyświetla odpowiedni formularz z wymaganymi polami: adres e-mail, nazwa użytkownika, hasło oraz potwierdzenie hasła.

   1.3. Aktor wypełnia wymagane pola formularza.

   1.4. Jednocześnie w trakcie wypełniania formularza trwa walidacja wprowadzanych danych (format adresu e-mail, unikalna nazwa użytkownika, zgodność obu haseł).

   1.5. Aktor zatwierdza formularz klikając przycisk „Sign up”.

   1.6. Jeśli wprowadzone dane były prawidłowe system tworzy nowe konto użytkownika, wprowadzając jego dane do bazy danych, szyfrując wprowadzone hasło oraz przypisując mu rolę użytkownika standardowego (User).

   1.7. Wyświetlany jest odpowiedni komunikat o udanej rejestracji i następuje przekierowanie użytkownika na stronę logowania.

   **Ścieżka alternatywna:**

   1.4.1. Jeśli aktor podał nieprawidłowe dane, system wyświetla odpowiedni komunikat walidacyjny przypisany do wypełnianego pola.

   1.5.1. Wprowadzone dane były poprawne, ale e-mail lub nazwa użytkownika istniały w bazie danych, system wyświetla odpowiedni błąd i następuje powrót do punktu 1.2.

2. Logowanie użytkownika

   **Aktor:** Użytkownik zarejestrowany

   **Ścieżka główna:**

   2.1. Aktor otwiera stronę logowania.

   2.2. System wyświetla odpowiedni formularz z wymaganymi polami: adres e-mail i hasło.

   2.3. Aktor wypełnia wymagane pola formularza.

2.4. Jednocześnie w trakcie wypełniania formularza trwa walidacja wprowadzanych danych.

2.5. Aktor zatwierdza formularz klikając przycisk „Log in”.

2.6. System weryfikuje przesłane dane, sprawdzając czy podany e-mail istnieje w bazie danych, a jeśli tak to sprawdza poprawność hasła.

2.7. Po udanej weryfikacji, system loguje użytkownika, ustawia token JWT i przekierowuje na stronę główną.

**Ścieżka alternatywna:**

2.6.1. Jeśli aktor wprowadził nieprawidłowe informacje, system wyświetla odpowiedni błąd i następuje powrót do punktu 1.2.

3. Wylogowanie użytkownika

**Aktor:** Użytkownik zalogowany

**Ścieżka główna:**

3.2. Aktor klika na przycisk wylogowania.

3.3. System usuwa token JWT przypisany przy logowaniu i przekierowuje na stronę logowania.

4. Przeglądanie strony głównej

**Aktor:** Użytkownik zalogowany

**Ścieżka główna:**

4.2. Aktor otwiera stronę główną aplikacji poprzez poprawne zalogowanie się lub kliknięcie na lewym menu przycisku „Home”.

4.3. System wyświetla trzy losowane, dostępne posty określane jako wyróżnione. Każdy post zawiera tytuł, kategorię, zdjęcie główne, nazwę użytkownika autora wraz z jego zdjęciem, datę utworzenia oraz średni czas potrzebny do przeczytania.

4.4. Aktor może wybrać dowolny post, klikając gdziekolwiek na kafelek postu.

4.5. System przekierowuje aktora na stronę z szczegółami postu.

5. Przeglądanie wszystkich postów

**Aktor:** Użytkownik zalogowany

**Ścieżka główna:**

5.2. Aktor otwiera stronę z wszystkimi postami za pomocą przycisku „Dashboard” znajdującego się na lewym panelu.

5.3. System wyświetla listę dostępnych postów. Każdy z nich zawiera tytuł, kategorię, zdjęcie główne, nazwę użytkownika autora, datę utworzenia, średni czas potrzebny do przeczytania oraz liczbę komentarzy.

5.4. Aktor może filtrować posty według tytułu, kategorii lub daty utworzenia (od najnowszych lub od najstarszych). Dodatkowo system zawiera paginację, dzięki której aktora może wybrać wyświetlaną ilość postów na pojedynczej stronie, przechodzić na kolejne strony i wracać. Domyślnie system wyświetla 10 postów na stronę (dostępne opcje: 5, 10, 20, 50).

5.5. Aktor może wybrać dowolny post, klikając gdziekolwiek na kafelek postu.

5.6. System przekierowuje aktora na stronę z szczegółami postu.

6. Dodawanie postu

**Aktor:** Zalogowany użytkownik z rolą użytkownika postującego lub administrator

**Ścieżka główna:**

6.2. Aktor przechodzi do zakładki tworzenia postu poprzez kliknięcie przycisku „Creator” na lewym panelu.

6.3. System wyświetla formularz z polami: tytuł, kategoria, treść, zdjęcie główne.

6.4. Aktor wypełnia wszystkie pola formularza.

6.5. Jednocześnie w trakcie wypełniania formularza trwa walidacja wprowadzanych danych.

6.6. Aktor zatwierdza formularz klikając przycisk „Create”.

6.7. System weryfikuje przesłane dane i w przypadku poprawności zapisuje post w bazie danych.

6.8. System wyświetla komunikat o udanym dodaniu postu.

**Ścieżka alternatywna:**

6.6.1. W przypadku błędnych danych przesłanych do serwera, system wyświetla komunikat o błędzie i następuje powrót do punktu 6.2.

7. Dodawanie komentarzy

**Aktor:** Zalogowany użytkownik

**Ścieżka główna:**

7.2. Aktor otwiera stronę wybranego postu.

7.3. System wyświetla sekcję komentarzy poniżej całej treści postu.

7.4. Aktor wypełnia pole z treścią komentarza i klika przycisk „Add” lub przycisk „Enter” na klawiaturze.

7.5. System waliduje treść komentarza, a po udanej weryfikacji dodaje komentarz do bazy danych.

7.6. System wyświetla komunikat o poprawnym dodaniu komentarza.

7.7. System automatycznie aktualizuje sekcję komentarzy i wyświetla je wraz z nowym komentarzem.

**Ścieżka alternatywna:**

7.4.1. W przypadku, gdy pole z treścią było puste lub wystąpił błąd, system wyświetla odpowiedni komunikat z błędem.

8. Usuwanie komentarzy

**Aktor:** Zalogowany użytkownik

**Ścieżka główna:**

8.2. Aktor otwiera stronę wybranego postu.

8.3. System wyświetla sekcję komentarzy poniżej całej treści postu.

8.4. Aktor będący autorem komentarza ma dostępny, przy każdym dodanym przez siebie komentarzu, przycisk do usuwania go. Aktor klika ten przycisk.

8.5. System wyświetla okno z potwierdzeniem usunięcia komentarza.

8.6. Aktor wciska przycisk „Delete”.

8.7. System usuwa komentarz z bazy danych i aktualizuje widok, dodatkowo informuje o udanym usunięciu komentarza.

**Ścieżka alternatywna:**

8.5.1. Aktor wciska przycisk „Cancel”, a system zamyka okno potwierdzenia usunięcia i następuje powrót do punktu 8.2.

8.6.1. W przypadku jakiegokolwiek błędu, system wyświetla odpowiedni komunikat o niepowodzeniu.

9. Zmienianie zdjęcia profilowego użytkownika

**Aktor:** Zalogowany użytkownik

**Ścieżka główna:**

9.2. Aktor otwiera stronę swojego profilu poprzez wciśnięcie przycisku „Profile” na lewym panelu.

9.3. System wyświetla dane użytkownika, w tym jego aktualne zdjęcie profilowe i nazwę użytkownika.

9.4. Użytkownik klika na swoje aktualne zdjęcie profilowe.

9.5. System wyświetla okno wyboru pliku z urządzenia.

9.6. Aktor wybiera plik będący obrazem i zatwierdza swój wybór.

9.7. System sprawdza wybrany czy plik ma odpowiedni format oraz czy nie przekracza maksymalnego rozmiaru.

9.8. System zapisuje plik na serwerze, aktualizuje ścieżkę do pliku w bazie danych i wyświetla nowe zdjęcie na stronie profilu.

9.9. System wyświetla komunikat o pomyślnej aktualizacji zdjęcia.

**Ścieżka alternatywna:**

9.6.1. W przypadku błędu lub nieprawidłowego pliku system wyświetla odpowiedni komunikat z błędem i następuje powrót do punktu 8.4.

10. Zmienianie hasła użytkownika

**Aktor:** Zalogowany użytkownik

**Ścieżka główna:**

10.2. Aktor otwiera stronę swojego profilu poprzez wciśnięcie przycisku „Profile” na lewym panelu.

10.3. System wyświetla formularz (aktualne hasło, nowe hasło oraz potwierdzenie nowego hasła) zmiany hasła pod jego danymi.

10.4. Aktor wypełnia formularz.

10.5. Jednocześnie w trakcie wypełniania formularza trwa walidacja wypełnianych pól.

10.6. Aktor zatwierdza formularz poprzez wciśnięcie przycisku „Change password”.

10.7. System weryfikuje poprawność aktualnego hasła oraz zgodność nowego hasła z jego potwierdzeniem.

10.8. System aktualizuje hasło użytkownika w bazie danych jednocześnie je szyfrując.

10.9. System informuje aktora o pomyślnej zmianie hasła.

**Ścieżka alternatywna:**

10.4.1. W przypadku błędnej walidacji system wyświetla odpowiedni komunikat.

10.6.1. W przypadku niepoprawności przesłanych danych system wyświetla odpowiedni komunikat błędu i następuje powrót do punktu 9.2.

11. Zarządzanie swoimi postami

**Aktor:** Zalogowany użytkownik z rolą użytkownika postującego lub administrator

**Ścieżka podstawowa:**

11.2. Aktor otwiera stronę swojego profilu poprzez wciśnięcie przycisku „Profile" na lewym panelu, a następnie przechodzi na zakładkę „Posts".

11.3. System wyświetla posty autorstwa aktora w formie tabeli zawierającej: identyfikator postu, tytuł, kategorię, datę utworzenia. Dodatkowo w ostatniej kolumnie znajdują się trzy przyciski w formie ikon, za pomocą których użytkownik może:

11.3.1. przejść do edycji postu

a) system wyświetla dokładnie taki sam formularz jak w przypadku tworzenia nowego postu, ale tym razem jest on wypełniony danymi, które dany post zawierał
b) aktor edytuje dane pole, a następnie klika przycisk „Update"
c) system weryfikuje przesłane dane i w przypadku poprawności aktualizuje post w bazie danych
d) system, w przypadku sukcesu wyświetla odpowiedni komunikat o udanej edycji, bądź błędnej edycji postu w przypadku błędu

11.3.2. przejść do strony szczegółów postu

a) system przekierowuje aktora na stronę z szczegółami postu

11.3.3. usunąć post

a) system wyświetla użytkownikowi okno z potwierdzeniem, że nastąpi usunięcie postu
b) użytkownik ma możliwość kliknięcia przycisku „Cancel" lub „Delete"
c) w przypadku potwierdzenia usunięcia postu poprzez naciśnięcie przycisku „Delete" system usuwa post z bazy danych i aktualizuje widok lub w przypadku anulowanie poprzez przycisk „Cancel" system zamyka okno i następuje powrót do punktu 10.2.

12. Czat w czasie rzeczywistym

**Aktor:** Zalogowany użytkownik

**Ścieżka główna:**

12.2. Aktor otwiera stronę czatu poprzez wciśnięcie przycisku „Chat” na lewym panelu.

12.3. System wyświetla listę dostępnych czatów.

12.4. Aktor wybiera czat z wyświetlonej listy lub rozpoczyna nową poprzez wyszukanie innego użytkownika.

12.5. System obok listy wyświetla okno rozmowy wraz z polem tekstowych i przyciskiem „Send”.

12.6. Aktor wypełnia pole tekstowe, a następnie wysyła wiadomość poprzez wciśnięcie przycisku „Send” lub przycisku „Enter” na klawiaturze.

12.7. Użytkownik będący odbiorcą wiadomości i mający włączoną tę samą konwersację otrzymuje wiadomość w czasie rzeczywistym, bez konieczności odświeżania strony.

12.8. System zapisuje każdą wysłaną wiadomość w bazie danych.

13. Zarządzanie wszystkimi użytkownikami

**Aktor:** Zalogowany użytkownik z rolą administratora

**Ścieżka główna:**

13.2. Aktor otwiera stronę do zarządzania użytkownikami poprzez wciśnięcie przycisku „Admin Panel” na lewym panelu.

13.3. System wyświetla zakładkę „Users” z wszystkimi użytkownikami istniejącymi w bazie danych w formie tabeli zawierającej: id użytkownika, nazwę użytkownika, adres e-mail. Dodatkowo w ostatniej kolumnie znajdują się dwa przyciski w formie ikon, za pomocą których aktor może:

13.3.1. zmienić rolę użytkownikowi w systemie:

a) system wyświetla okno z polem nazwy użytkownika, którego aktor wybrał oraz pole rozwijane z wyborem roli dostępnej w systemie

b) aktor wybiera jedną z trzech dostępnych ról oraz klika przycisk „Update”

c) system aktualizuje rolę użytkownika w bazie danych

13.3.2. usunąć konto użytkownika:

a) system wyświetla okno z potwierdzeniem, że nastąpi usunięcie użytkownika
b) aktor ma możliwość kliknięcia przycisku „Cancel” lub „Delete”
c) w przypadku potwierdzenia usunięcia użytkownika poprzez naciśnięcie przycisku „Delete” system usuwa użytkownika z bazy danych wraz z wszystkimi powiązanymi z nim postami i komentarzami i aktualizuje widok lub w przypadku anulowania poprzez przycisk „Cancel” system zamyka okno i następuje powrót do punktu 13.2.

14. Zarządzanie wszystkimi postami

**Aktor:** Zalogowany użytkownik z rolą administratora

**Ścieżka główna:**

14.2. Aktor otwiera stronę do zarządzania postami poprzez wciśnięcie przycisku „Admin Panel” na lewym panelu, a następnie przejście na zakładkę „Posts”.

14.3. System wyświetla wszystkie posty w formie tabeli zawierającej: id postu, nazwę użytkownika autora, tytuł, kategorię, datę utworzenia. Dodatkowo w ostatniej kolumnie znajdują się trzy przyciski w formie ikon, za pomocą których aktor może:

14.3.1. przejść do edycji postu

a) system wyświetla dokładnie taki sam formularz jak w przypadku tworzenia nowego postu, ale tym razem jest on wypełniony danymi, które dany post zawierał
b) aktor edytuje dane pole, a następnie klika przycisk „Update”
c) system weryfikuje przesłane dane i w przypadku poprawności aktualizuje post w bazie danych
d) system, w przypadku sukcesu wyświetla odpowiedni komunikat o udanej edycji, bądź błędnej edycji postu w przypadku błędu

14.3.2. przejść do strony szczegółów postu

a) system przekierowuje aktora na stronę z szczegółami postu

14.3.3. usunąć post

a) system wyświetla aktorowi okno z potwierdzeniem, że nastąpi usunięcie postu
b) aktor ma możliwość kliknięcia przycisku „Cancel” lub „Delete”

c) w przypadku potwierdzenia usunięcia postu poprzez naciśnięcie przycisku „Delete” system usuwa post z bazy danych i aktualizuje widok lub w przypadku anulowania poprzez przycisk „Cancel” system zamyka okno i następuje powrót do punktu 14.2.

15. Zarządzanie wszystkimi kategoriami

**Aktor:** Zalogowany użytkownik z rolą administratora

**Ścieżka główna:**

15.2. Aktor otwiera stronę do zarządzania kategoriami poprzez wciśnięcie przycisku „Admin Panel” na lewym panelu, a następnie przejście na zakładkę „Categories”.

15.3. System wyświetla wszystkie kategorie w formie tabeli zawierającej: identyfikator kategorii, nazwę kategorii. Dodatkowo w ostatniej kolumnie znajduje się przycisk w formie ikony, za pomocą której aktor może:

15.3.1. usunąć kategorię

a) system wyświetla aktorowi okno z potwierdzeniem, że nastąpi usunięcie kategorii
b) aktor ma możliwość kliknięcia przycisku „Cancel” lub „Delete”
c) w przypadku potwierdzenia usunięcia kategorii poprzez naciśnięcie przycisku „Delete” system usuwa kategorię z bazy danych i aktualizuje widok lub w przypadku anulowania poprzez przycisk „Cancel” system zamyka okno i następuje powrót do punktu 15.2.

16. Dodawanie nowej kategorii

**Aktor:** Zalogowany użytkownik z rolą administratora

**Ścieżka główna:**

16.2. Aktor otwiera stronę do zarządzania kategoriami poprzez wciśnięcie przycisku „Admin Panel” na lewym panelu, a następnie przejście na zakładkę „Categories”.

16.3. Aktor klika przycisk „Add category” znajdujący się nad tabelą z wszystkimi kategoriami.

16.4. System wyświetla aktorowi okno z polem do wpisania nazwy kategorii.

16.5. Aktor wpisuje nazwę kategorii, a jednocześnie w trakcie wpisywania nazwy trwa walidacja wprowadzanych danych.

16.6. Aktor zatwierdza dodanie nowej kategorii poprzez wciśnięcie przycisku „Add”.

16.7. System weryfikuje poprawność przesłanych danych oraz dodaje kategorię do bazy danych, następnie wyświetla odpowiedni komunikat o sukcesie.

**Ścieżka alternatywna:**

16.5.1. W przypadku niepoprawnych danych system wyświetla odpowiedni komunikat błędu i następuje powrót do punktu 16.3.

## 5.2 Opis bazy danych

Kolejnym krokiem tworzenia aplikacji było zaprojektowanie odpowiedniej bazy danych i jej utworzenie. Poniżej szczegółowo zostały opisane poszczególne tabele oraz relacje między nimi.

### 5.2.1 Tabele w bazie danych

Jak wspomniano wcześniej, baza danych zawiera zarówno tabele domyślne, dostarczane przez interfejs ASP.NET Identity (opisane jako pierwsze trzy), jak i tabele zdefiniowane samodzielnie na potrzeby aplikacji. Poniżej znajduje się opis każdej z nich.

1. Tabela AspNetUsers

   **Opis:** To tabela przechowująca użytkowników aplikacji.

   **Kolumny:**

   - Id (typ: text) – zawiera unikalny identyfikator użytkownika i jest kluczem głównym
   - UserName (typ: varchar) – zawiera nazwę użytkownika
   - Email (typ: varchar) – zawiera adres e-mail użytkownika
   - Password (typ: text) – zawiera hasło użytkownika w postaci zaszyfrowanej
   - AvatarPath (typ: text) – zawiera unikalną ścieżkę do zdjęcia profilowego użytkownika

2. Tabela AspNetRoles

   **Opis:** To tabela przechowująca role dostępne w aplikacji

   **Kolumny:**

- Id (typ: text) – zawiera unikalny identyfikator roli i jest kluczem głównym
- Name (typ: text) – zawiera nazwę roli

3. Tabela AspNetUserRoles

   **Opis:** To tabela łączeniowa między użytkownikami, a rolami.

   **Kolumny:**

   - UserId (typ: text) – zawiera identyfikator użytkownika i jest kluczem obcym
   - RoleId (typ: text) – zawiera identyfikator roli i jest kluczem obcym

4. Tabela Posts

   **Opis:** To tabela przechowująca posty tworzone przez użytkowników

   **Kolumny:**

   - Id (typ: integer) – zawiera identyfikator postu i jest kluczem głównym
   - Title (typ: text) – zawiera tytuł postu
   - Content (typ: text) – zawiera treść postu
   - CreatedAt (typ: timestamp with time zone) – zawiera dokładny czas utworzenia postu
   - UpdatedAt (typ: timestamp with time zone) – zawiera dokładny czas aktualizacji postu
   - AuthorId (typ: text) – zawiera identyfikator autora posta i jest kluczem obcym
   - CategoryId (typ: integer) – zawiera identyfikator kategorii posta i jest kluczem obcym
   - ImagePath (typ: text) – zawiera unikalną ścieżkę do zdjęcia posta
   - TimeToRead (typ: integer) – zawiera średni czas, potrzebny do przeczytania posta

5. Tabela Categories

   **Opis:** To tabela przechowująca dostępne kategorie, które pomagają w filtrowaniu i oznaczaniu postów

   **Kolumny:**

- Id (typ: integer) – zawiera unikalny identyfikator kategorii i jest kluczem głównym
- Name (typ: text) – zawiera nazwę kategorii

6. Tabela Comments

   **Opis:** To tabela przechowująca komentarze dodawane do postów przez użytkowników

   **Kolumny:**

   - Id (typ: integer) – zawiera unikalny identyfikator komentarza i jest kluczem głównym
   - Content (typ: text) – zawiera treść komentarza
   - PostId (typ: integer) – zawiera identyfikator posta, do którego został dodany komentarz i jest kluczem obcym
   - AuthorId (typ: text) – zawiera identyfikator użytkownika, który dodał komentarz i jest kluczem obcym

7. Tabela Chats

   **Opis:** To tabela przechowująca dane dotyczące czatów pomiędzy użytkownikami

   **Kolumny:**

   - Id (typ: uuid) – zawiera identyfikator czatu i jest kluczem głównym
   - Participant1Id (typ: text) – zawiera identyfikator pierwszego użytkownika przypisanego do czatu i jest kluczem obcym
   - Participant2Id (typ: text) – zawiera identyfikator drugiego użytkownika przypisanego do czatu i jest kluczem obcym
   - CreatedAt (typ: timestamp with time zone) – zawiera dokładną datę utworzenia czatu

8. Tabela ChatsMessages

   **Opis:** To tabela przechowująca wiadomości wysłane w konkretnym czacie

   **Kolumny:**

   - Id (typ: uuid) – zawiera unikalny identyfikator wiadomości i jest kluczem głównym

- ChatId (typ: uuid) – zawiera identyfikator czatu, w którym wiadomość została wysłana i jest kluczem obcym
- SenderId (typ: text) – zawiera identyfikator nadawcy wiadomości i jest kluczem obcym
- MessageText (typ: text) – zawiera treść wiadomości
- SentAt (typ: timestamp with timezone) – zawiera dokładny czas wysłania wiadomości

### 5.2.2 Relacje pomiędzy tabelami

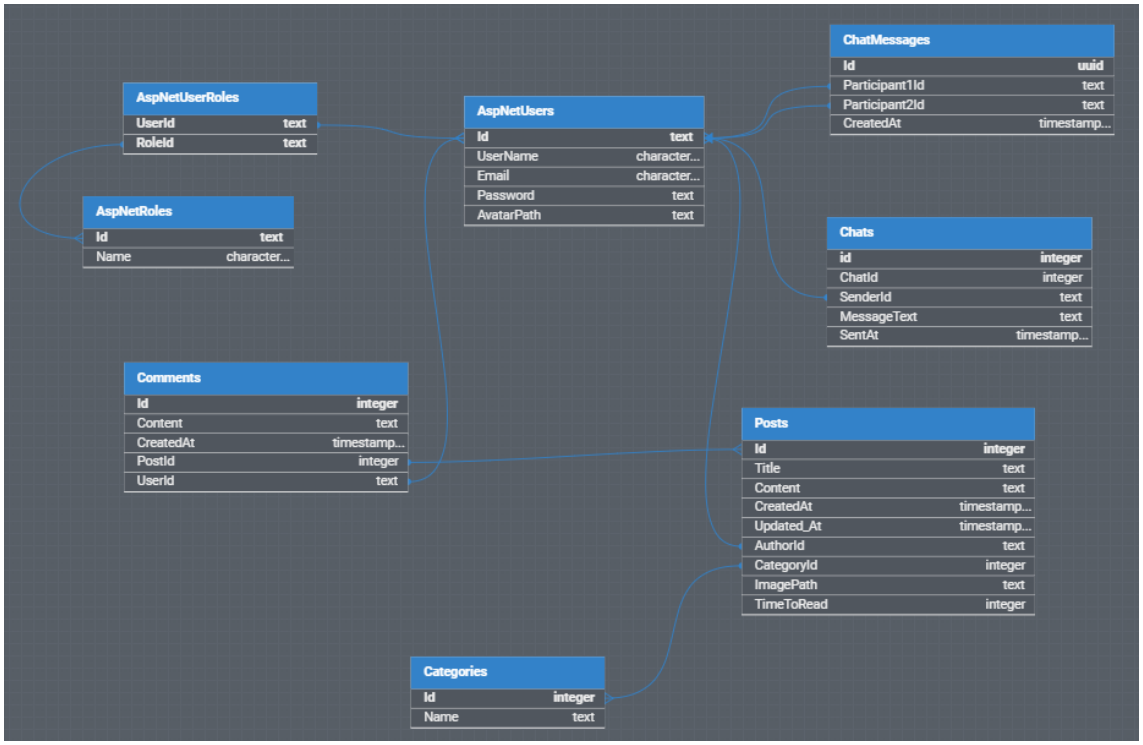
W celu zapewnienia odpowiednich wymagań aplikacji, relacje między poszczególnymi tabelami zostały zaprojektowane w następujący sposób:

1. Relacja jeden-do-wielu:
   - Tabela AspNetUsers z tabelą Posts, gdyż jeden użytkownik może tworzyć wiele postów, ale post może mieć tylko jednego autora
   - Tabela AspNetUsers z tabelą Comments, gdyż jeden użytkownik może tworzyć wiele komentarzy, ale komentarz może mieć tylko jednego autora
   - Tabela Posts z tabelą Comments, gdyż jeden post może zawierać wiele komentarzy, ale komentarz musi być przypisany do jednego posta
   - Tabela Categories z tabelą Posts, gdyż jedna kategoria może być przypisana do wielu postów, ale post musi mieć jedną kategorię
2. Relacja wiele-do-wielu:
   - Tabela AspNetUser z tabelą AspNetRoles za pośrednictwem tabeli AspNetUserRoles, gdyż użytkownicy mogą mieć przypisaną więcej niż jedną rolę
3. Relacja jeden-do-jeden:
   - Tabela Chats z tabelą AspNetUsers, gdyż każdy czat ma dokładnie dwóch uczestników

### 5.2.3 Diagram ERD bazy danych

W celu zwizualizowania omawianej bazy danych i relacji, które zawiera przygotowany został diagram ERD, przedstawiony na rys. 5.2. Pozwala on na wizualizacje związków pomiędzy poszczególnymi encjami, dzięki czemu łatwiej jest

zrozumieć całą strukturę. Dodatkowo, stanowi podstawową dokumentację zaprojektowanego systemu bazy danych.




Rys. 5.2. Diagram ERD bazy danych (źródło: Opracowanie własne)

## 5.3 Podsumowanie

Całość opisanego procesu projektowania aplikacji stanowi solidną podstawę, na której opiera się cała budowa systemu. Dzięki temu, kolejny proces, jakim jest etap implementacji, może przebiegać w uporządkowany sposób, zgodny z określonymi wymaganiami.

# 6. Proces implementacji

Mając udokumentowany proces projektowania aplikacji, można przejść do kolejnego, kluczowego etapu realizacji projektu jakim jest jego implementacja. W niniejszym rozdziale zostaną omówione najważniejsze etapy tworzenia systemu, realizującego wszystkie założone wymagania. Przedstawiona zostanie zarówno implementacja backendu, jak i frontendu, wraz z zrzutami ekranu w pełni działającej aplikacji.

## 6.1 Struktura backendu

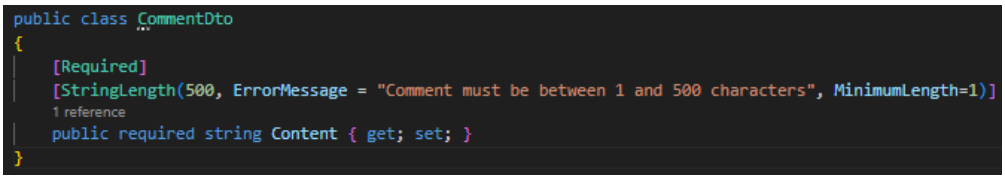
Implementację backendu oparto na architekturze trójwarstwowej, dzięki czemu można oddzielić warstwę logiki biznesowej od warstwy prezentacji danych. Cała struktura opiera się o działanie:

- modeli,
- kontrolerów,
- serwisów.

Modele zawierają definicje danych encji oraz dodatkowe klasy DTO (Data Transfer Object), które są używane do wymiany informacji między serwerem, a klientem. Definicja encji umożliwia odpowiednie utworzenie jej w bazie danych za pomocą frameworku Entity Framework Core, natomiast używanie DTO sprowadza się do prostszego obiektu zawierającego tylko potrzebne informacje, przesyłane między backendem, a frontendem. Na rysunkach 6.1 i 6.2 przedstawiono kolejno przykładowy model encji komentarza oraz odpowiadającej klasie DTO. Jak można zauważyć, klasa CommentDTO zawiera dodatkowo odpowiednie adnotacje, dzięki którym przesłany od klienta komentarz może być następnie zweryfikowany po stronie serwera w kontrolerze.

```
public class Comment
{
    1 reference
    public int Id { get; set; }
    1 reference
    public required string Content { get; set; }
    1 reference
    public DateTime CreatedAt { get; set; }

    3 references
    public int PostId { get; set; }
    1 reference
    public required Post Post { get; set; }
    1 reference
    public required ApplicationUser User { get; set; }
}
```

Rys. 6.1. Klasa encji komentarza (źródło: Opracowanie własne)




Rys. 6.2. Klasa DTO komentarza (źródło: Opracowanie własne)

Kontrolery odpowiadają za obsługę wszystkich żądań HTTP płynących z frontendu. Zawierają one odpowiednie adnotacje dotyczące autoryzacji, które ograniczają dostęp do wybranych operacji w zależności od roli użytkownika w systemie. Zwracają odpowiednie statusy w przypadku sukcesu lub błędu, co pozwala na poprawne obsłużenie takiej odpowiedzi po stronie klienta. Każda z operacji w kontrolerze ma zdefiniowaną odpowiednią trasę, za pomocą której są wykonywane żądania HTTP. Przykładowo, kontroler komentarzy zawiera następujące trasy:

- GET /api/Comment - żądanie zwracające wszystkie komentarze dla wybranego postu,
- POST /api/Comment/{postId} – żądanie umożliwiające utworzenie komentarza dotyczącego danego postu,
- DELETE /api/Comment/{commentId} – żądanie umożliwiające usunięcie wybranego komentarza.

Serwisy zawierają logikę biznesową aplikacji, oddzielając kontrolery od bezpośredniego dostępu do danych. Obsługują one odpowiednie akcje, których żąda kontroler do wykonania jakiejś operacji CRUD (tworzenie, odczytywanie, aktualizowanie, usuwanie) jednocześnie operując na bazie danych. Opisane działanie opiera się na działaniu wzorca projektowego Repozytorium, w którym oddzielana jest warstwa logiki od warstwy dostępu do danych.

Dodatkowo, w celu automatycznego zarządzania cyklem życia obiektów i poprawienia modularności kodu używany jest wzorzec wstrzykiwania zależności. Żeby serwis mógł być używany w kontrolerze, najpierw należy go dodać do wbudowanego w framework ASP.NET Core kontenera DI (Dependency Injection). Odbywa się to w pliku Program.cs, gdzie zarejestrowane są wszystkie używane serwisy, które następnie można wstrzykiwać do kontrolerów za pomocą

ich konstruktorów. Na rys. 6.3 przedstawiono zarejestrowane w systemie serwisy wraz z ich interfejsami. Metoda AddScoped<T>() oznacza, że powstanie jedna instancja serwisu na każde żądanie HTTP.

```
builder.Services.AddScoped<TokenService>();
builder.Services.AddScoped<IPostService, PostService>();
builder.Services.AddScoped<ICommentService, CommentService>();
builder.Services.AddScoped<ICategoryService, CategoryService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IChatService, ChatService>();
```

Rys. 6.3. Rejestrowanie serwisów w kontenerze DI (źródło: Opracowanie własne)

Przykładowo, serwis kategorii przedstawiono na rys. 6.4, następnie poniżej, na rys. 6.5 przedstawiono kontroler kategorii, w którym ten serwis jest wstrzykiwany.

```
public class CategoryService : ICategoryService
{
    9 references
    private readonly BlogDbContext _context;

    0 references
    public CategoryService(BlogDbContext context)
    {
        _context = context;
    }

    2 references
    public async Task<IEnumerable<Category>> GetAllCategoriesAsync()
    {
        return await _context.Categories.ToListAsync();
    }
    2 references
    public async Task<Category> AddCategoryAsync(CategoryDto categoryDto)
    {
        var category = new Category
        {
            Name = categoryDto.Name
        };

        _context.Categories.Add(category);
        await _context.SaveChangesAsync();
        return category;
    }
    2 references
    public async Task<bool> UpdateCategoryAsync(int id, CategoryDto categoryDto)
    {
        var category = await _context.Categories.FindAsync(id);
        if (category == null) return false;

        category.Name = categoryDto.Name;
        await _context.SaveChangesAsync();
        return true;
    }
    2 references
    public async Task<bool> DeleteCategoryAsync(int id)
    {
        var category = await _context.Categories.FindAsync(id);
        if (category == null) return false;

        _context.Categories.Remove(category);
        await _context.SaveChangesAsync();
        return true;
    }
}
```

Rys. 6.4. Serwis kategorii (źródło: Opracowanie własne)

```
[ApiController]
[Route("api/[controller]")]
1 reference
public class CategoryController : ControllerBase
{
    5 references
    private readonly ICategoryService _categoryService;

    0 references
    public CategoryController(ICategoryService categoryService)
    {
        _categoryService = categoryService;
    }

    [Authorize]
    [HttpGet]
    1 reference
    public async Task<IActionResult> GetAllCategories()
    {
        var categories = await _categoryService.GetAllCategoriesAsync();
        return Ok(categories);
    }

    [Authorize(Roles = "Admin")]
    [HttpPost]
    0 references
    public async Task<IActionResult> CreateCategory([FromBody] CategoryDto categoryDto)
    {
        if(!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var category = await _categoryService.AddCategoryAsync(categoryDto);
        return CreatedAtAction(nameof(GetAllCategories), new { id = category.Id }, category);
    }

    [Authorize(Roles = "Admin")]
    [HttpPut("{categoryId}")]
    0 references
    public async Task<IActionResult> UpdateCategory(int categoryId, [FromBody] CategoryDto categoryDto)
    {
        var result = await _categoryService.UpdateCategoryAsync(categoryId, categoryDto);
        if(!result) return NotFound();
        return NoContent();
    }

    [Authorize(Roles = "Admin")]
    [HttpDelete("{categoryId}")]
    0 references
    public async Task<IActionResult> DeleteCategory(int categoryId)
    {
        var result = await _categoryService.DeleteCategoryAsync(categoryId);
        if(!result) return NotFound();
        return NoContent();
    }
}
```

Rys. 6.5. Kontroler kategorii (źródło: Opracowanie własne)

## 6.2 Struktura frontendu

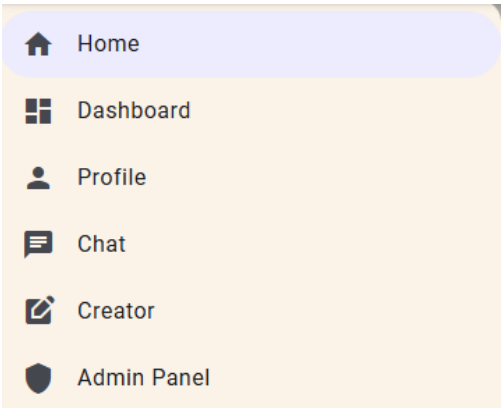
Frontend opiera się głównie na tworzeniu komponentów przy użyciu frameworka Angular. Jednakże, w celu zapewnienia wysokiej jakości kodu stworzone zostały także odpowiednie modele, które odpowiadają modelom z serwera i umożliwiają wyświetlanie odpowiednich danych w aplikacji. Stworzone zostały także serwisy, które obsługują wszystkie żądania HTTP wysyłane do serwera. W tym celu w każdym z serwisów, za pomocą wstrzykiwania zależności jest dodawany HttpClient umożliwiający komunikację klient-serwer. Na rys. 6.6 przedstawiono przykładowy kod serwisu odpowiadającego za kategorie postów.

```
export class CategoriesService {

  constructor(private httpClient: HttpClient) { }

  getAllCategories(): Observable<Category[]> {
    return this.httpClient.get<Category[]>(`${environment.apiUrl}/Category`);
  }

  addCategory(category: Category): Observable<Category> {
    return this.httpClient.post<Category>(`${environment.apiUrl}/Category`, category);
  }

  deleteCategory(categoryId: number): Observable<void> {
    return this.httpClient.delete<void>(`${environment.apiUrl}/Category/${categoryId}`);
  }
}
```

Rys. 6.6. Klasa serwisu kategorii po stronie klienta (źródło: Opracowanie własne)

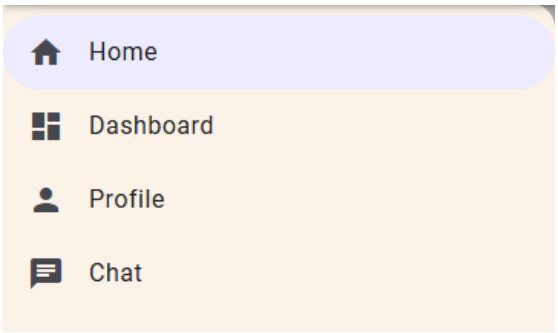
### 6.2.1 Bezpieczeństwo po stronie klienta

W celu zapewnienia odpowiednich wymagań funkcjonalnych, token JWT jest przechowywany w pamięci podręcznej, co pozwala na późniejszy dostęp do zasobów serwera, gdyż ten token, za pomocą interceptora będzie dodawany do każdego żądania wysyłanego do serwera. Gdy użytkownik loguje się do aplikacji, backend po udanym uwierzytelnieniu zwraca ten token, a frontend odczytuje z niego odpowiednie dane, takie jak rola użytkownika w systemie. Dzięki temu przez cały czas trwania ważności tokenu, aplikacja wie z jakim typem użytkownika ma do czynienia, co pozwala na wyświetlanie odpowiednich treści dla konkretnego z nich. Przykładem tego zachowania jest komponent lewego panelu menu, którego wygląd i zachowanie jest zależne od roli użytkownika. Każdy z dostępnych na tym menu przycisków powoduje przekierowanie na konkretny widok aplikacji, dzieje się to za pośrednictwem zdefiniowanych tras, zawierających dany komponent oraz ścieżkę URL. Każda z tych ścieżek jest dodatkowo chroniona za pośrednictwem tzw. guardów, co pozwala na kontrolę tego co może zobaczyć użytkownik po stronie klienta. Na rysunkach 6.7, 6.8 i 6.9 przedstawiono ten komponent w trzech wariantach, które uzależnione są od konkretnego typu użytkownika.




Rys. 6.7. Wygląd lewego panelu menu dla użytkownika z rolą administratora (źródło: Opracowanie własne)


Rys. 6.8. Wygląd lewego panelu menu dla użytkownika z rolą postującego (źródło: Opracowanie własne)




Rys. 6.9. Wygląd lewego panelu menu dla zwykłego użytkownika (źródło: Opracowanie własne)

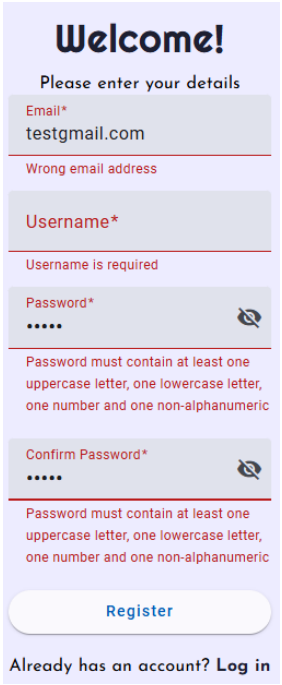
### 6.2.2 Walidacja danych po stronie klienta

W aplikacji dużą rolę odgrywają formularze, których uzupełnienie jest konieczne do wykonywania wielu założeń funkcjonalnych takich jak:

- logowanie,

- rejestracja,
- zmiana hasła,
- dodawanie i edycja postu.

W celu zapewnienia jak najbardziej intuicyjnego interfejsu użytkownika, każde uzupełniane pole ma zdefiniowane odpowiednie do swoich zadań walidatory sprawdzające poprawność wprowadzanych danych. Dzięki temu, już na etapie uzupełniania formularza użytkownik wie jakie kryteria musi spełnić żeby wszystko się zgadzało. Każdy z takich formularzy został zaimplementowany przy użyciu modułu reactive forms dostępnego w Angularze. Przykładowy formularz z komunikatami walidacyjnymi został przedstawiony na rys. 6.10 i dotyczy komponentu rejestracji użytkownika.




Rys. 6.10. Formularz rejestracji z komunikatami walidacyjnymi (źródło: Opracowanie własne)

## 6.3 Testowanie aplikacji

W celu zapewnienia odpowiedniej stabilności oraz działania zgodnego z wszystkimi założeniami projektowymi, przeprowadzono testy jednostkowe zarówno po stronie serwera, jak i po stronie klienta.

Zastosowanie testów jednostkowych pozwoliło na podzielenie funkcjonalności systemu na pojedyncze, mniejsze zachowania i przetestować ich działanie jako działającej jednostki. Testy po stronie serwera koncentrowały się głównie na sprawdzeniu

logiki biznesowej oraz poprawności działania wszystkich kontrolerów. Natomiast testy części klienta skupiły się głównie na komponentach oraz serwisach odgrywających kluczową rolę w obsłudze interfejsu użytkownika. Na rys. 6.11 przedstawiono przykładowy test na serwerze, który sprawdza czy metoda kontrolera kategorii - GetAllCategories zwraca wynik zawierający listę kategorii z jednym elementem oraz odpowiedni status.

```
[Fact]
0 references
public async Task GetAllCategories_ReturnsOkResult_WithListOfCategories()
{
    // Arrange
    var categories = new List<Category> { new Category { Id = 1, Name = "Test" } };
    _mockCategoryService.Setup(service => service.GetAllCategoriesAsync()).ReturnsAsync(categories);

    // Act
    var result = await _controller.GetAllCategories();

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnCategories = Assert.IsType<List<Category>>(okResult.Value);
    Assert.Single(returnCategories);
}
```

Rys. 6.11. Przykładowy test metody GetAllCategories, zwracającej wszystkie kategorie (źródło: Opracowanie własne)

Natomiast na rys. 6.12 został pokazany przykładowy fragment testów komponentu profilu użytkownika na frontendzie, który sprawdza czy dane użytkownika są odpowiednio wczytywane z serwisów.

```
  fixture = TestBed.createComponent(ProfileComponent);
  component = fixture.componentInstance;
  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
  userService = TestBed.inject(UserService) as jasmine.SpyObj<UserService>;
  postsService = TestBed.inject(PostsService) as jasmine.SpyObj<PostsService>;

  authService.getUserId.and.returnValue('123');
  authService.getUserEmail.and.returnValue('test@example.com');
  authService.getUsername.and.returnValue('testuser');
  userService.getUserAvatar.and.returnValue(of({ imagePath: 'avatar.png' }));
  postsService.getPostsByUserId.and.returnValue(of([]));

  fixture.detectChanges();
});

it('should initialize user data', () => {
  expect(component.userEmail).toBe('test@example.com');
  expect(component.username).toBe('testuser');
  expect(component.changePasswordForm.get('email')?.value).toBe('test@example.com');
});
```

Rys. 6.12. Przykładowy test komponentu profilu użytkownika (źródło: Opracowanie własne)

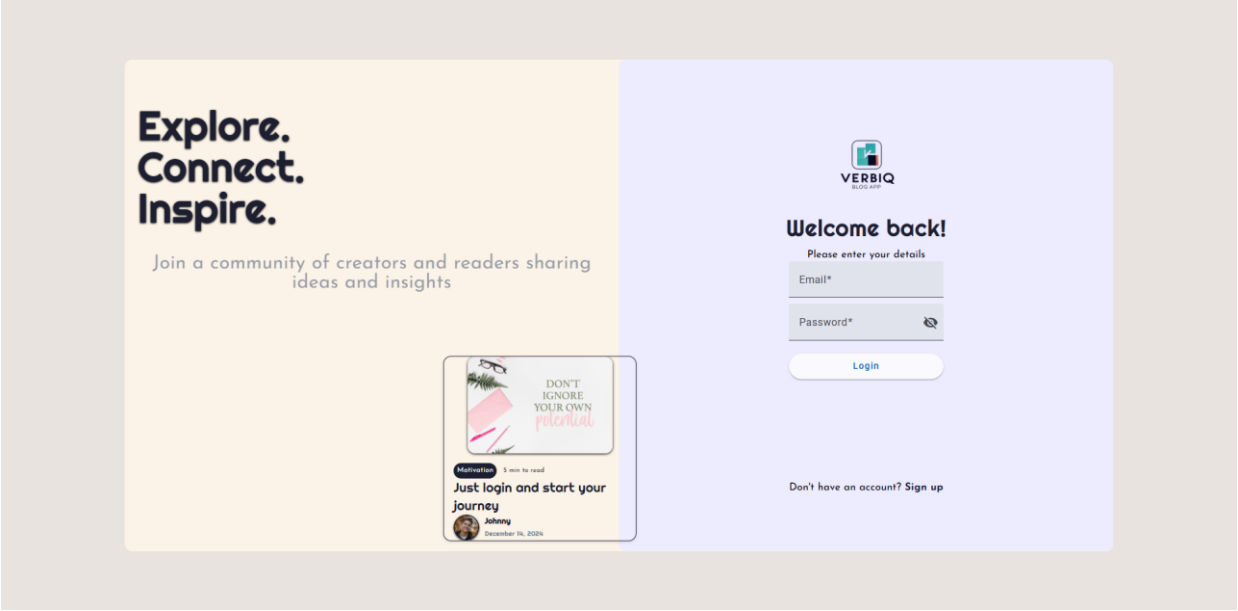
Do korzyści z zastosowanych testów jednostkowych można zaliczyć:

- znaczne poprawienie stabilności systemu – testy zapewniły szybkie wykrycie i naprawienie błędów w logice aplikacji,
- łatwiejsza rozbudowa – dzięki istniejącym testom łatwiej wprowadzać nowe funkcjonalności bez ryzyka uszkodzenia już istniejących, działających modułów.

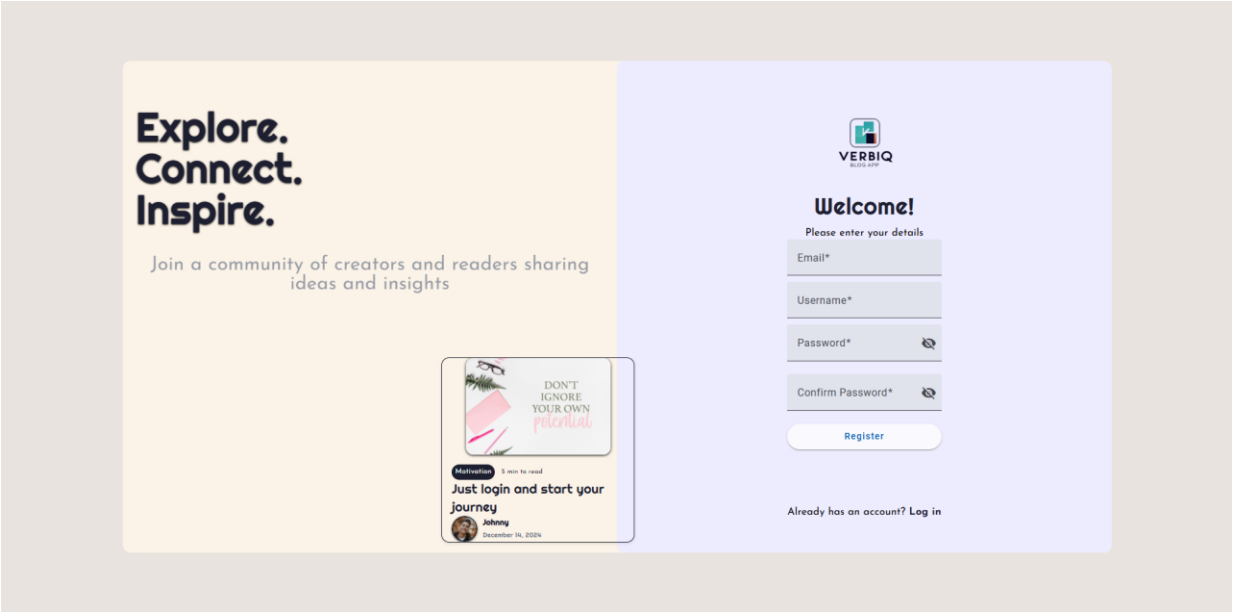
Podsumowując, w trakcie testów aplikacji nie zostały wykryte poważne błędy, które mogłyby znacznie wpłynąć na działanie systemu. Te, które znaleziono dotyczyły głównie nieścisłości w implementacji niektórych funkcji, ale zostały szybko rozwiązane. Mimo to, odpowiednie przetestowanie aplikacji było bardzo ważnym etapem jej rozwoju, gdyż umożliwiło osiągnięcie wysokiej jakości końcowego oprogramowania.

## 6.4 Interfejs użytkownika

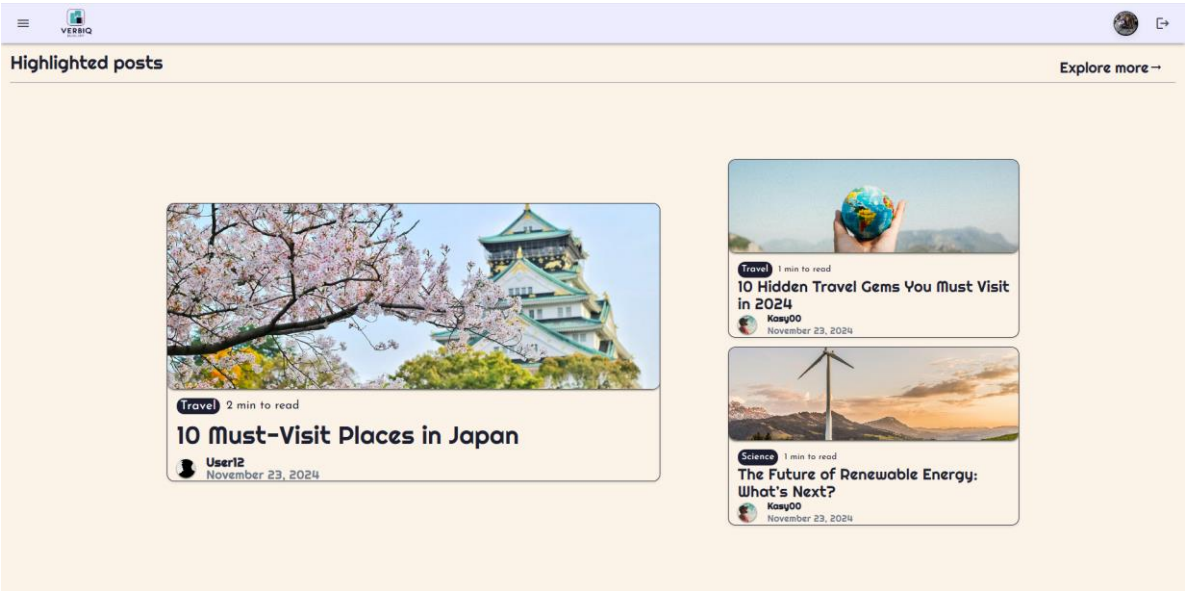
Poniżej, na rysunkach 6.13-6.26 zaprezentowano wszystkie widoki interfejsu użytkownika, by zobrazować działanie aplikacji oraz pokazać przejrzystość interfejsu.



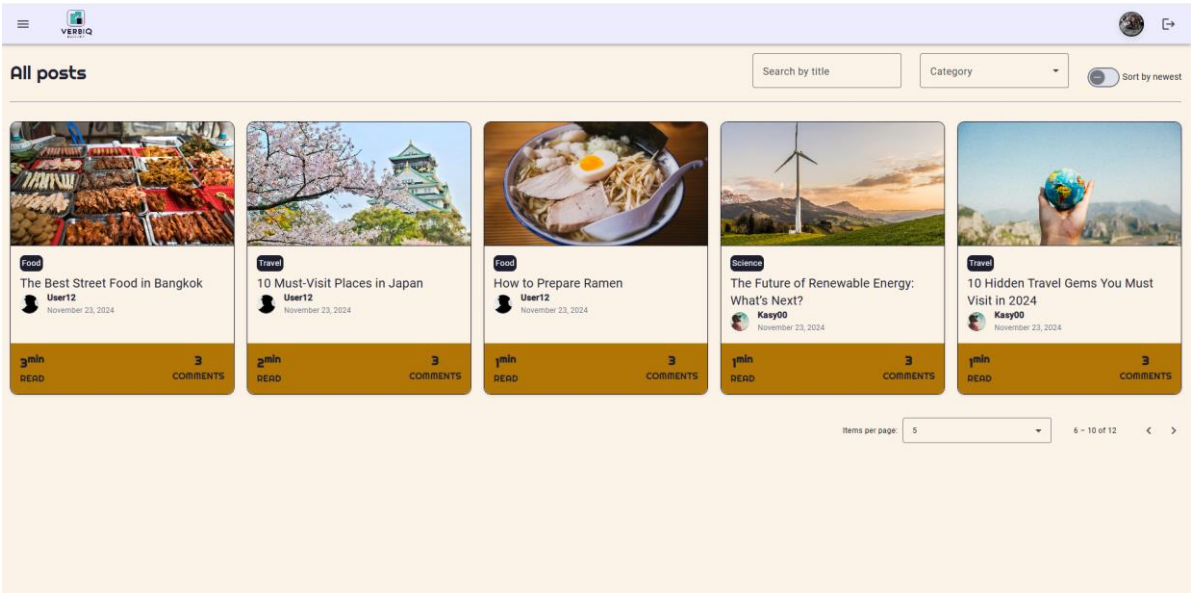

Rys. 6.13. Widok logowania (źródło: Opracowanie własne)



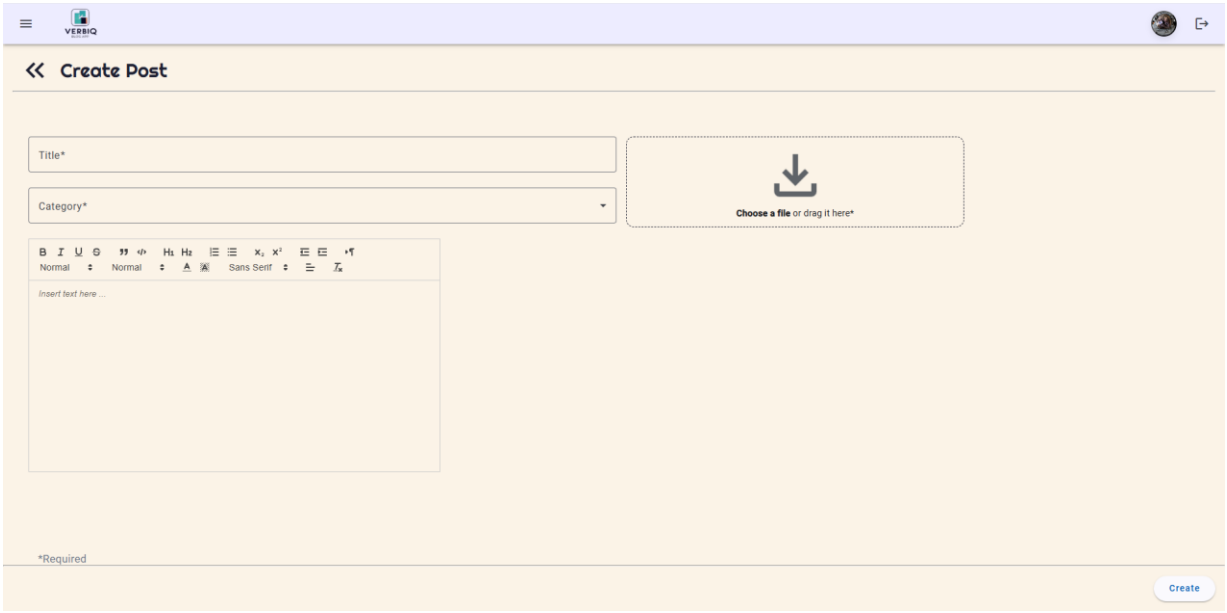

Rys. 6.14. Widok rejestracji (źródło: Opracowanie własne)




Rys. 6.15. Widok domowy, dostępny bezpośrednio po zalogowaniu (źródło: Opracowanie własne)

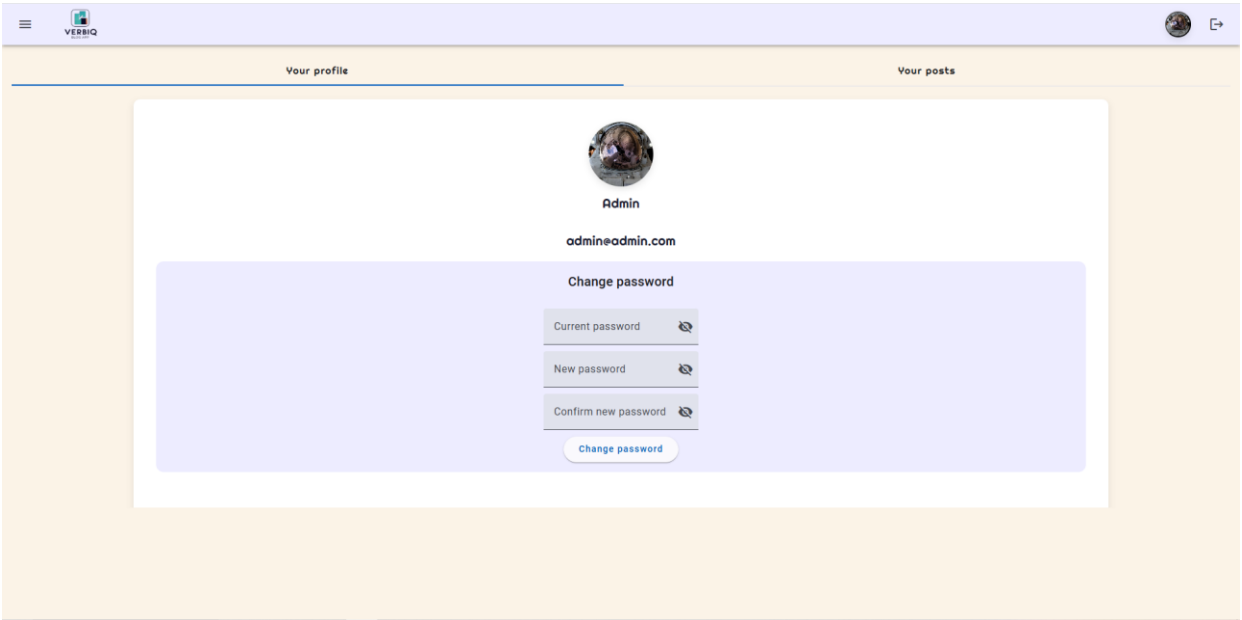



Rys. 6.16. Widok wszystkich postów wraz z dostępnymi filtrami i paginacją (źródło: Opracowanie własne)

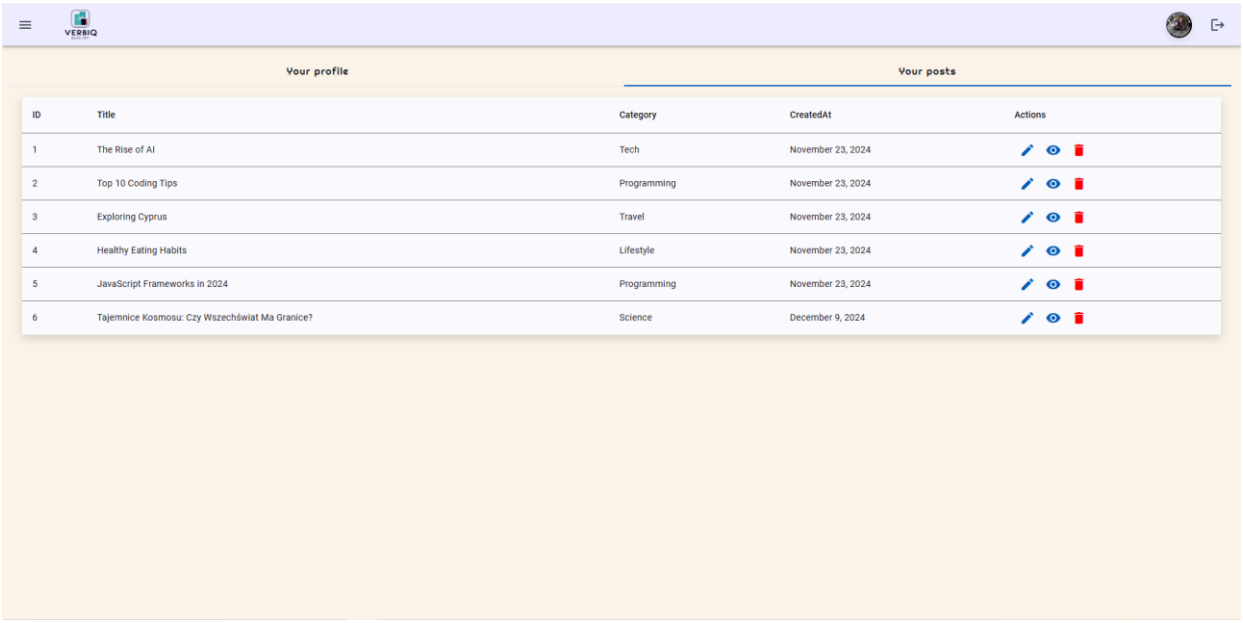



Rys. 6.17. Widok formularza tworzenia nowego postu (źródło: Opracowanie własne)

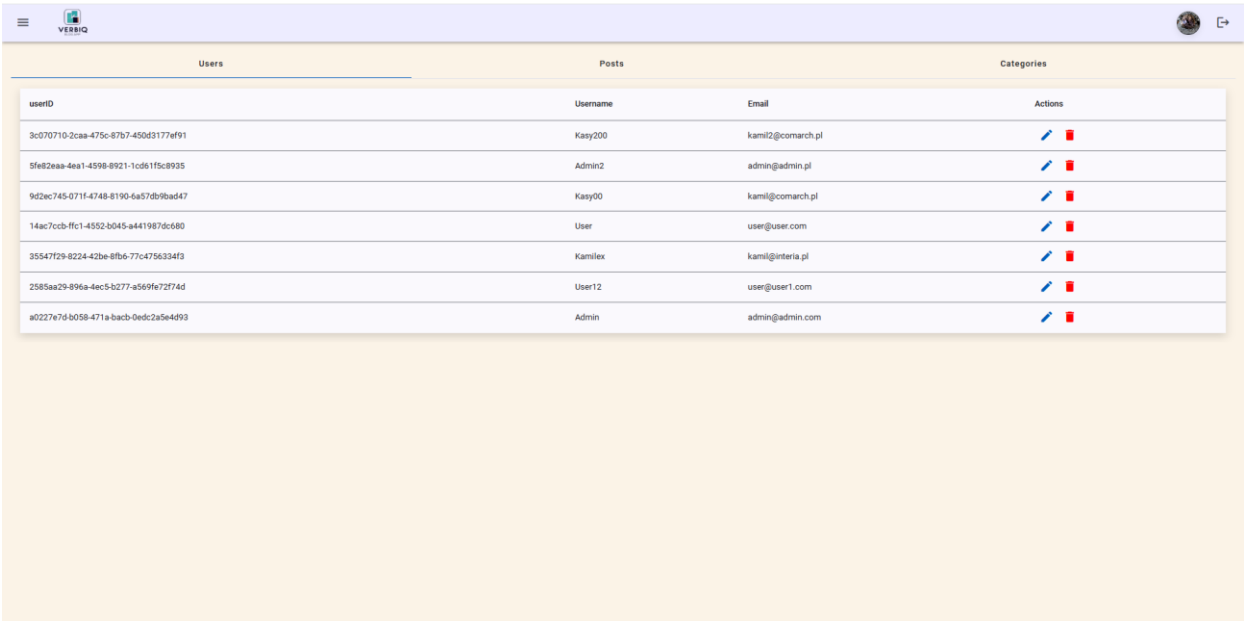

Rys. 6.18. Widok konwersacji z innym użytkownikiem (źródło: Opracowanie własne)



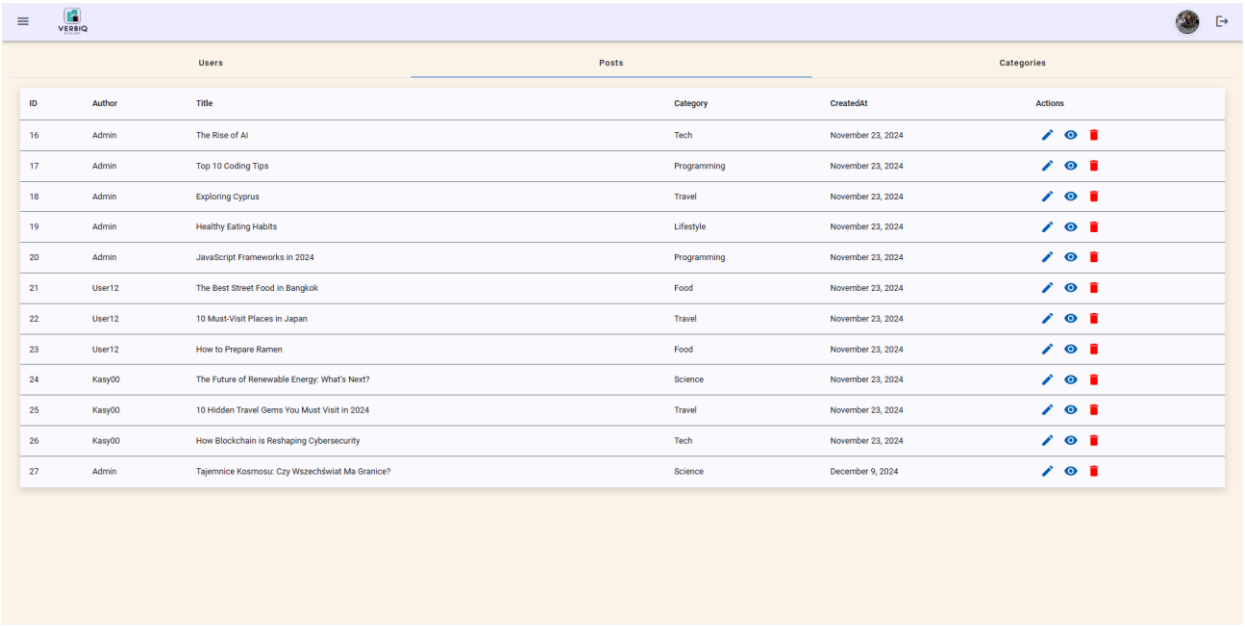

Rys. 6.19. Widok profilu użytkownika (źródło: Opracowanie własne)



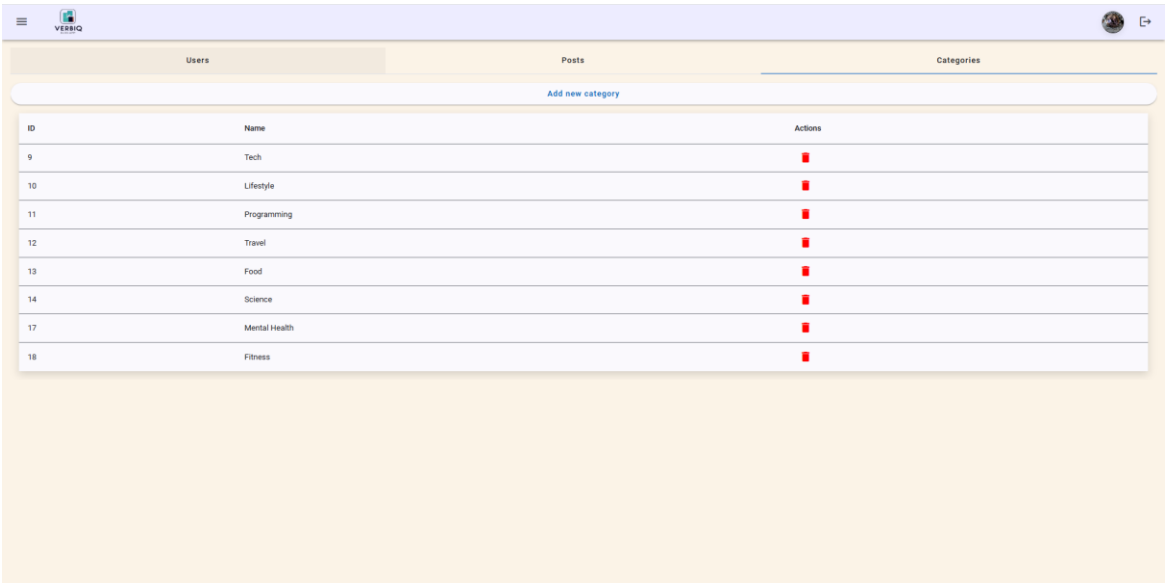

Rys. 6.20. Widok zarządzania swoimi postami (źródło: Opracowanie własne)



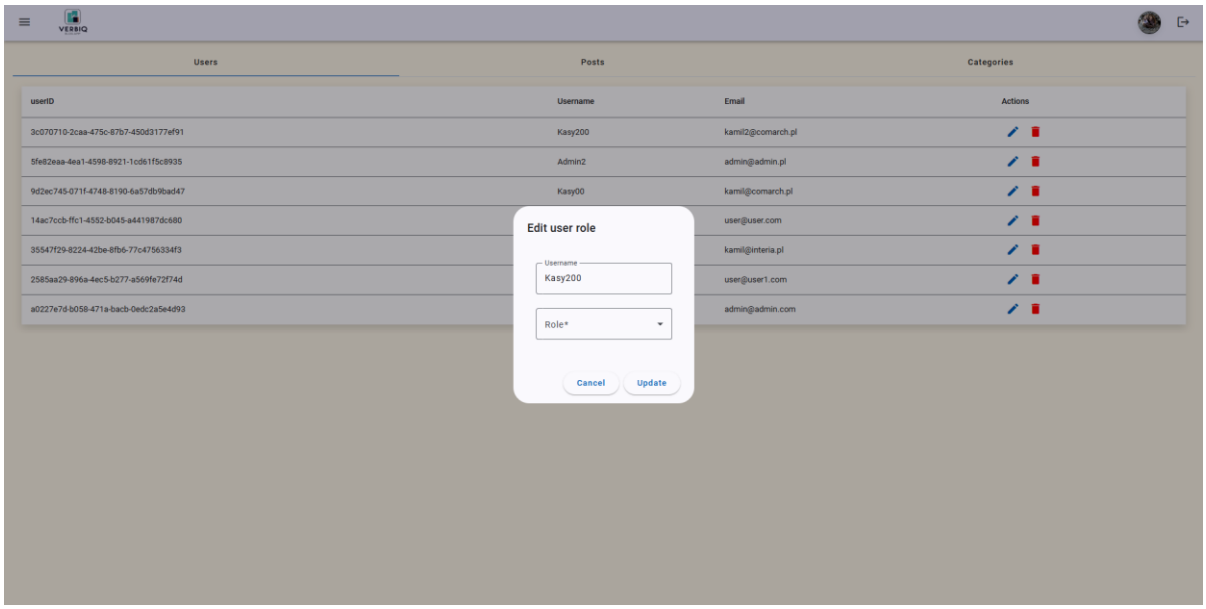

Rys. 6.21. Widok zarządzania użytkownikami w panelu administratora (źródło: Opracowanie własne)



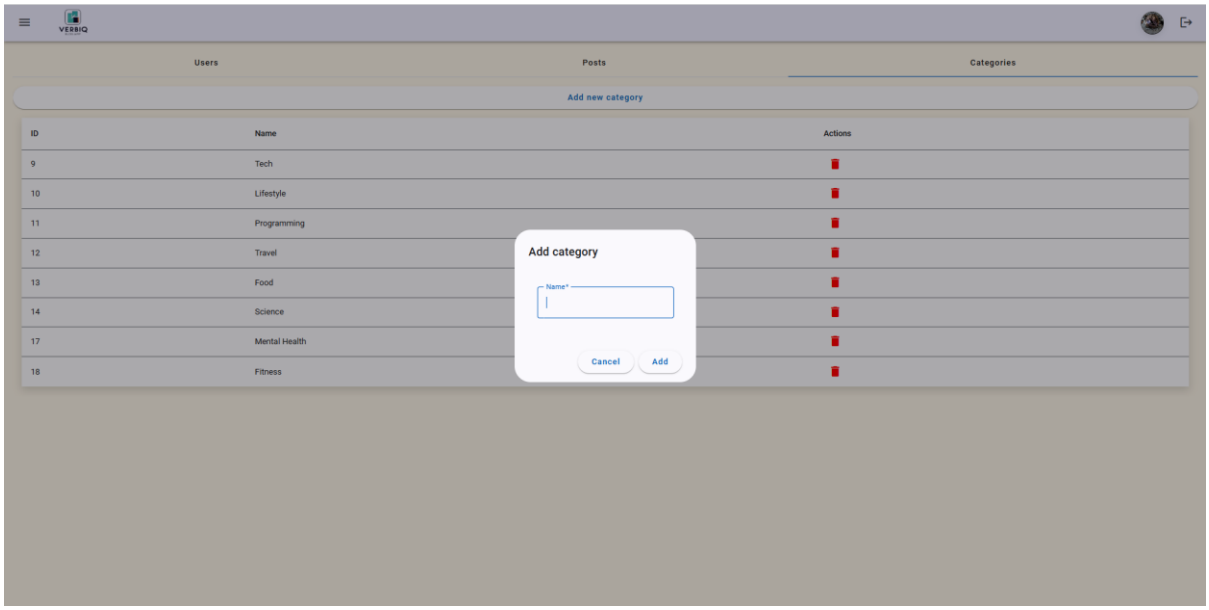

Rys. 6.22. Widok zarządzania postami w panelu administratora (źródło: Opracowanie własne)



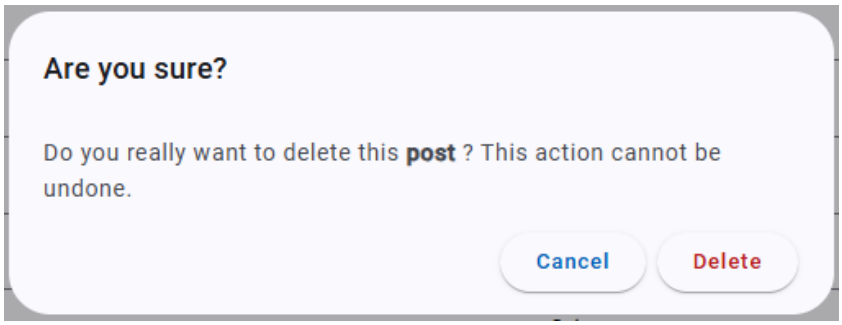

Rys. 6.23. Widok zarządzania kategoriami w panelu administratora (źródło: Opracowanie własne)




Rys. 6.24. Widok po otwarciu okna edycji roli użytkownika w panelu administratora (źródło: Opracowanie własne)




Rys. 6.25. Widok po otwarciu okna dodawania nowej kategorii w panelu administratora (źródło: Opracowanie własne)




Rys. 6.26. Przykładowe okienko pytające o potwierdzenie usunięcia postu (źródło: Opracowanie własne)

## Podsumowanie

W ramach pracy udało się zrealizować zamierzony cel, jakim było zaprojektowanie i stworzenie aplikacji blogowej zawierającej czat w czasie rzeczywistym, opartej na architekturze RESTful API. Wszystkie etapy realizacji projektu, począwszy od analizy wymagań, przez projekt bazy danych i architektury systemu, aż po implementację i na końcu testowanie, pozwoliły uzyskać efekt końcowy będący w pełni funkcjonalną aplikacją. Zastosowanie nowoczesnych technologii oraz odpowiednich wzorców projektowych sprawia, że system jest łatwy w utrzymaniu, przez co wprowadzanie nowych funkcjonalności mogłoby przebiegać w szybki sposób. Ponadto, projekt potwierdził, że odpowiednie zaprojektowanie i zaplanowanie pracy w połączeniu z sprawdzonymi narzędziami pozwala na efektywne stworzenie aplikacji, spełniającej wszystkie swoje założenia, z których najważniejsze osiągnięcia to:

- bezpieczny system logowania i rejestracji z zarządzaniem rolami użytkowników,
- poprawna obsługa postów, kategorii i komentarzy,
- funkcjonalny czat w czasie rzeczywistym,
- przejrzysty i responsywny interfejs użytkownika.

Aplikacja stanowi solidną bazę do dalszego rozwoju, który mógłby być zrealizowany poprzez dodawanie nowych, przydatnych funkcjonalności. Jedną z nich mogły by być powiadomienia, wysyłane do użytkownika w czasie rzeczywistym. Tego typu funkcjonalność dodatkowo wsparłaby interakcję użytkowników, informując ich natychmiastowo o nowych wiadomościach w czacie i komentarzach pod ich postami. Do implementacji świetnie sprawdziłaby się używana już w aplikacji biblioteka SignalR, co wpłynęłoby na dynamikę komunikacji użytkowników.

Podsumowując, praca wykonana nad opisanym projektem była nie tylko okazją do zdobycia praktycznej wiedzy z zakresu projektowania złożonych systemów, ale także do nauki i zastosowania nowoczesnych technologii i narzędzi, jednocześnie rozwiązując rzeczywisty problem.

## Bibliografia


1. Małecka M., Blogowanie – innowacyjny model biznesu XXI wieku w: Innowacje i przedsiębiorczość: Teoria i praktyka, 2014, 191-202.
2. Strona internetowa platformy Medium https://medium.com/ (data dostępu 02.01.2025 r.).
3. Strona internetowa platformy HubPages https://discover.hubpages.com/ (data dostępu 02.01.2025 r.).
4. Strona internetowa platformy Dev.to https://dev.to/ (data dostępu 02.01.2025 r.).
5. Dokumentacja techniczna narzędzia Angular https://angular.dev/overview (data dostępu 02.01.2025 r.).
6. Dokumentacja techniczna narzędzia Jasmine https://jasmine.github.io/ (data dostępu 02.01.2025 r.).
7. Dokumentacja techniczna narzędzia Karma https://karma-runner.github.io/latest/index.html (data dostępu 02.01.2025 r.).
8. Dokumentacja techniczna narzędzia ASP.NET Core https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0 (data dostępu 02.01.2025 r.).
9. Dokumentacja techniczna narzędzia SignalR https://learn.microsoft.com/en-us/aspnet/signalr/ (data dostępu 04.01.2025 r.).
10. Dokumentacja techniczna narzędzia xUnit https://xunit.net/ (data dostępu 02.01.2025 r.).
11. Dokumentacja techniczna narzędzia Moq https://github.com/devlooped/moq/wiki/Quickstart (data dostępu 02.01.2025 r.).
12. Dokumentacja techniczna narzędzia PostgreSQL https://www.postgresql.org/docs/ (data dostępu 02.01.2025 r.).
13. Dokumentacja techniczna narzędzia Entity Framework Core https://learn.microsoft.com/en-us/ef/ (data dostępu 02.01.2025 r.).
14. Oleś W., Małysiak-Mrozek B., Mrozek D., Porównanie wydajności wybranych narzędzi odwzorowania obiektowo-relacyjnego. Politechnika Śląska, Instytut Informatyki, 2013.
15. Gamma E., Helm R., Johnson R., Vlissides J., Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku. Wydawnictwo Helion, 2021.

16. Dokumentacja techniczna narzędzia .NET zawierająca opis użycia wzorca repozytorium https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design (data dostępu 02.01.2025 r.).
17. Seemann M., Dependency Injection in .NET. Manning, 2018.
18. Sönmez Turan M., Barker E., Burr W., Chen L., Recommendation for Password-Based Key Derivation Part 1: Storage Applications. NIST Special Publication 800-132, 2010.
19. Sacha K., Inżynieria Oprogramowania. Wydawnictwo Naukowe PWN, Warszawa, 2010.

## Spis rysunków