

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I  
I N A U K I N F O R M A C Y J N Y C H



# Praca dyplomowa inżynierska

na kierunku Informatyka i Systemy Informacyjne

System zarządzania projektami z możliwością komunikacji

**Maksim Makaranka**

Numer albumu 308826

**Vladyslav Shestakov**

Numer albumu 308904

promotor

dr inż. Janusz Rafałko

WARSZAWA 2023



## Streszczenie

### System zarządzania projektami z możliwością komunikacji

Niniejsza praca inżynierska zawiera projekt aplikacji webowej do zarządzania projektami z możliwością tworzenia zespołów użytkowników. Zarządzanie projektami pozwala na szereg wymaganych funkcjonalności, np.: tworzenie zadań i podzadań, przypisanie do nich użytkowników, ustalanie terminów poszczególnych zadań. Funkcjonalności te są dostępne za pomocą specjalnej tablicy kanban, która pokazuje statusy zadań. Dostępny jest również kalendarz związany z projektem, który wyświetla informacje o statusie projektu w czasie.

System zespołowy pozwala na tworzenie elastycznej hierarchii użytkowników, dopasowanej do konkretnego zespołu. Pozwala na tworzenie ról o różnych uprawnieniach związanych z funkcjami zespołów, menedżera projektów i czatów.

Aplikacja posiada komunikator z możliwością tworzenia czatów służących do koordynowania pracy. Czat może być tworzony dla użytkowników w konkretnym zespole lub projekcie, jest ograniczony wyżej wymienionymi rolami i umożliwia wysyłanie wiadomości tekstowych i obrazków.

**Słowa kluczowe:** menedżer projektów, komunikator, praca międzyzespołowa, Team It, .NET, Blazor



## **Abstract**

### **Project management system with communication**

This engineering thesis contains a project of a web application for project management with the ability to create teams of users. Project management allows for a number of required functions, for example: creating tasks and subtasks, allocating time for completion and assigned user, setting a terms for individual tasks. These functionalities are available using a special kanban-board that demonstrates task statuses. Also there is project-related calendar that displays information about the project's status over time.

The team system allows you to create a flexible hierarchy of users, tailored to a specific team. This allows creating roles with different permissions associated with teams, project manager and chats functionalities.

The application has a messenger with the ability to create chats used to coordinate work. The chat can be created for users in a specific team or project, is limited by the aforementioned roles and allows you to send text messages and pictures.

**Keywords:** project manager, messenger, cross-team work, Team It, .NET, Blazor



## Spis treści

<b>1. Wstęp</b>	<b>11</b>
1.1. Zawartość pracy	12
1.2. Istniejące rozwiązania	12
1.3. Przykład zastosowania aplikacji	14
1.4. Podział prac	15
<b>2. Specyfikacja wymagań</b>	<b>16</b>
2.1. Diagram przypadków użycia	16
2.2. Historie użytkownika	17
2.2.1. Historie użytkownika modułu Teams	18
2.2.2. Historie użytkownika modułu Project Manager	19
2.2.3. Historie użytkownika modułu Chats	20
2.3. Opis funkcjonalności systemu	21
2.3.1. Moduł Teams	21
2.3.2. Moduł Project Manager	26
2.3.3. Moduł Chats	32
2.4. Wymagania нефункционалне	35
<b>3. Opis rozwiązania</b>	<b>36</b>
3.1. Architektura systemu	36
3.1.1. Serwer	36
3.1.2. Baza danych	36
3.1.3. Aplikacja kliencka	37
3.1.4. Komunikacja	37
3.2. Opis architektury serwera	38
3.2.1. Podstawowe klasy i struktury	38
3.2.2. Moduł Teams	40
3.2.3. Moduł Project Manager	41
3.2.4. Moduł Chats	42

3.2.5.	Obsługa zapytań API . . . . .	43
3.2.6.	Czysta architektura . . . . .	44
3.3.	Opis bazy danych . . . . .	46
3.4.	Opis aplikacji klienckiej . . . . .	48
<b>4.</b>	<b>Technologie i wdrożenie . . . . .</b>	<b>49</b>
4.1.	Użyte technologie . . . . .	49
4.1.1.	Blazor . . . . .	50
4.1.2.	SignalR . . . . .	51
4.2.	Wdrożenie systemu . . . . .	52
4.2.1.	Serwer . . . . .	53
4.2.2.	Aplikacja kliencka . . . . .	55
4.2.3.	Baza danych . . . . .	55
4.2.4.	SignalR . . . . .	57
4.3.	Instrukcja użytkowania aplikacji . . . . .	58
<b>5.</b>	<b>Analiza rozwiązania . . . . .</b>	<b>59</b>
5.1.	Automatyczne testy jednostkowe . . . . .	59
5.1.1.	Testy jednostkowe warstwy Domain . . . . .	59
5.1.2.	Testy jednostkowe warstwy Infrastructure . . . . .	60
5.2.	Automatyczne testy integracyjne . . . . .	61
5.2.1.	Testy integracyjne warstwy Application . . . . .	61
5.2.2.	Testy integracyjne warstwy Models . . . . .	61
5.3.	Automatyczne testy UI . . . . .	63
<b>6.</b>	<b>Podsumowanie . . . . .</b>	<b>65</b>
6.1.	Produkt finalny . . . . .	65
6.2.	Dalszy rozwój . . . . .	65





## 1. Wstęp

Najczęściej, korzystając z istniejących rozwiązań do zarządzania projektami, zespoły bądź firmy równolegle używają jakieś dodatkowe narzędzie do komunikacji. Na przykład, często stosowanymi komunikatorami są *Microsoft Teams*, *Discord* albo nawet *Messenger*. Ale w przypadku, jeśli cała komunikacja w grupie osób dotyczy wyłącznie projektów, nad którymi ten zespół pracuje, takie rozniesienie może być niewygodne. To doprowadziło nas do pomysłu, że posiadanie komunikatora w takim systemie może zwiększyć integrację procesów roboczych. Dodatkową zaletą takiego rozwiązania jest fakt, że możemy dopasować ten komunikator na potrzeby aplikacji integrując go z pozostałymi komponentami.

Pomysł posiadania komunikatora w systemie do zarządzania projektami doprowadził do tego, że w ramach niniejszej pracy inżynierskiej została stworzona i opisana aplikacja **Team It**. Jest to aplikacja internetowa, gdzie dowolny użytkownik ma możliwość stworzenia konta i rozpoczęcia korzystania z systemu. Oprócz funkcjonalności standardowych dla menedżera projektów, aplikacja posiada również możliwość tworzenia zespołów, co pozwala na wspólną pracę. Te zespoły za pomocą systemu ról pozwalają na elastyczne wyspecyfikowanie czynności, dostępnych różnym użytkownikom w zespole, co zwiększa zakres potencjalnych zastosowań aplikacji. W celu zwiększenia integracji międzyludzkiej, idea wsparcia której jest podstawą projektu, system został wzbogacony również o komunikator i możliwość międzyzespolowej pracy nad projektami.

Otrzymujemy aplikację dla zarządzania projektami wzbogaconą o komunikator oraz możliwość łączenia się z innymi użytkownikami w zespoły, specjalnie skonfigurowane przy pomocy systemu ról. Faktycznie, jest to menedżer projektów z elementami sieci społecznościowej. Tak traktując ten system możemy zrobić krok dalej w stronę integracji użytkowników między sobą. Można zrobić to na wiele sposobów, w swojej aplikacji realizujemy możliwość dodawania do projektu innych zespołów, niż zespół, w obrębie którego projekt został utworzony. Teoretycznie, można było by jeszcze bardziej rozwinąć się w tym kierunku, na przykład uzupełnić zespoły o pewnego rodzaju system „threadów” bądź forumów, pozwalających na inny rodzaj dyskusji, niż pozwala czat, ale w ramach tej pracy inżynierskiej skupiliśmy się na już wymienionych rzeczach.

Tak zbudowany system może znaleźć swoje zastosowanie w wielu dziedzinach. Najbardziej oczywistym przypadkiem są firmy posiadające współpracujące ze sobą jednostki. Za pomocą

systemu ról moglibyśmy stworzyć hierarchię członków zespołu, odpowiadającą stanowiskom poszczególnych użytkowników w pewnej jednostce. Posiadanie oddzielnych zespołów nie zmniejsza nam integracji procesu roboczego dzięki możliwości pracy międzyzespołowej nad projektami oraz chatów. Również teoretycznie aplikacja mogłaby wspierać firmy, część pracy których jest oddawana na outsourcing.

## 1.1. Zawartość pracy

W rozdziale 2 wymienione zostały wymagania funkcjonalne i niefunkcjonalne systemu, opisane typowe przypadki użycia systemu za pomocą diagramu przypadków użycia i historii użytkownika. Przedstawiono dokładne opisy głównych funkcjonalności aplikacji.

W rozdziale 3 przedstawiony został również opis techniczny systemu. Zostały szczegółowo opisane trzy główne jego moduły: serwer, aplikacja kliencka i baza danych. Została dokładnie opisana architektura aplikacji serwerowej realizowanej z użyciem frameworku *.NET* i technologii *ASP.NET Core*, wykryta wielowarstwowa jego natura oraz komunikacja pomiędzy poszczególnymi warstwami. Podany został diagram bazy danych wykorzystującej silnik *Microsoft SQL Server* i stworzonej z wykorzystaniem technologii mapowania relacyjnego *Entity Framework Core*. Została udokumentowana struktura aplikacji klienckiej wraz z szczegółowym wyjaśnieniem doboru technologii *Blazor* i problemów, z tym związanych.

Rozdział 4 z kolei przedstawia opis niektórych użytych technologii i wyjaśnia je dobór. Również zawiera dokumentację wdrożenia systemu i instrukcję użytkowania.

Rozdział 5 skupia się na przeanalizowaniu wynikowego rozwiązania i udowodnieniu, że jest prawidłowe.

W rozdziale 6 zostały podsumowane przemyślenia odnośnie finalnego produktu i zaproponowane pomysły na jego dalszy rozwój.

## 1.2. Istniejące rozwiązania

Elementem, wyróżniającym system spośród podobnych już istniejących rozwiązań jest możliwość elastycznej międzyzespołowej pracy nad projektami oraz komunikacji między użytkownikami i zespołami użytkowników.

Współczesne systemy są skupione na realizacji metodyk prowadzenia projektów (np. *SCRUM*) oraz wykorzystaniu narzędzi do wizualizacji pracy i czasu spędzonego w tej pracy, natomiast

## 1.2. ISTNIEJĄCE ROZWIĄZANIA

mniej uważana jest przydzielona na integrację oraz komunikację międzyludzką. Bardzo często zachodzi potrzeba wykorzystania innych narzędzi. Team It rozwiązuje to wprowadzając własny komunikator oraz możliwość integrowania zespołów o niezależnych hierarchiach ról, w celu wspólnej pracy nad projektami. To powoduje, że aplikacja jest przydatnym narzędziem do zorganizowania pracy w zespołach bądź firmach o zupełnie różnych zawodach.

Na rynku istnieje wiele różnych systemów zarządzania projektami, które mogą dostarczać wiele różnych funkcjonalności. najbardziej znane rozwiązania to:

1. *Asana*[1] - popularne internetowe narzędzie do zarządzania projektami, które umożliwia zespołom organizowanie, śledzenie i współpracę nad zadaniami i projektami. Ma przyjazny dla użytkownika interfejs i oferuje różnorodne funkcje, w tym zarządzanie zadaniami, współpracę zespołową, śledzenie czasu i raportowanie. Asana ma również aplikację mobilną i oferuje integrację z innymi narzędziami, takimi jak *Dysk Google*, *Slack* i *Zoom*.
2. *Trello*[2] - wizualne narzędzie do zarządzania projektami, które umożliwia zespołom organizowanie zadań i projektów za pomocą tablic, list i kart. Jest znany ze swojej prostoty i elastyczności, dzięki czemu jest popularnym wyborem dla zespołów różnej wielkości. *Trello* oferuje podstawowe funkcje zarządzania zadaniami, współpracy zespołowej i raportowania, a także integracje z innymi narzędziami, takimi jak *Dysk Google*, *Slack* i *Evernote*.
3. *Jira*[3] - system stworzony przez firmę Atlassian, który jest przeznaczony dla zespołów programistycznych, ale może też być używany w przypadku innych projektów. Udostępnia wiele możliwości, takich jak tablice Kanban oraz Scrum, wizualizację postępu i tworzenie różnego rodzaju raportów. Dodatkowo pozwala zintegrować swoją pracę z innymi narzędziami, na przykład *Slack*, *Confluence*, oraz *Google Drive*.

Większość podobnych rozwiązań oferuje podobny zestaw narzędzi i funkcji, więc trudno stwierdzić które narzędzie jest lepsze albo gorsze. Natomiast możemy wyodrębnić zalety, które wyróżniają stworzoną w ramach tej pracy aplikację wśród innych:

1. System ról, który jest bardzo elastyczny i pozwalający na wygodne zorganizowanie pracy zespołowej i międzyzespołowej
2. Wewnętrzny komunikator, który jest zintegrowany z wyżej wymienionym systemem ról i eliminuje potrzebę korzystania z innych systemów do komunikacji
3. Aplikacja jest "lightweight", czyli nie jest obciążona dużą ilością różnych narzędzi, co jest plusem dla nowych użytkowników, dla których zrozumienie systemu i wdrażanie do procesu roboczego zajmie znacznie mniej czasu w porównaniu do bardzo rozbudowanych systemów

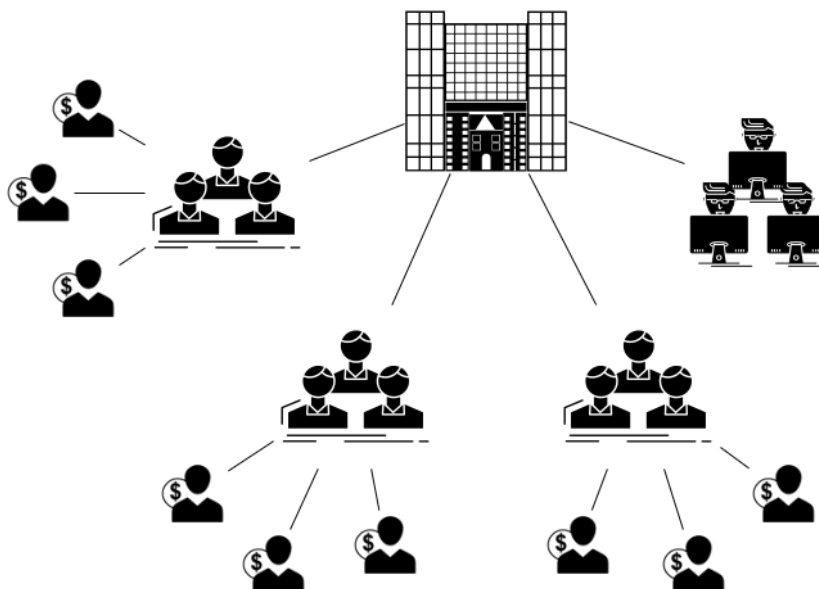
Niestety tak jak każda inna aplikacja, ma ona również wady względem konkurentów:

1. Aplikacja jest "lightweight", czyli nie oferuje wielu dodatkowych narzędzi pomocnych przy zarządzaniu projektami
2. Wbudowany komunikator nie jest dobrze rozbudowany w porównaniu do "standalone" komunikatorów i pozwala tylko na prostą wymianę powiadomieniami

Potencjalnie te wady mogą być zgładzone wraz z dalszym rozwojem aplikacji, co pozwoli na konkurowanie z podobnymi rozwiązaniami na rynku.

### 1.3. Przykład zastosowania aplikacji

Rozważmy przypadek zastosowania systemu dla firmy. Wyobraźmy sobie firmę świadczącą usługi księgowe. Załóżmy, że ta firma dla pracy z różnymi klientami posiada różne zespoły, które pracują wykorzystując wymagane narzędzie księgowe. Takie narzędzia bywają bardzo zaawansowane, więc teoretycznie firma powinna mieć specjalny zespół programistów zajmujących się wsparciem i konfiguracją tego narzędzia. Wizualizacja modelu takiej firmy została przedstawiona na rysunku 1.1.



Rysunek 1.1: Model potencjalnej firmy-konsumenta aplikacji

Przy wdrożeniu naszej aplikacji w tak umodelowanej firmie, w module **Teams** utworzylibyśmy poszczególne team'y dla zespołów księgowych, oraz team dla zespołu programistów. Team'y księgowe mieliby projekty odpowiadające różnym klientom. Teraz jeśli na którymkolwiek projekcie

## 1.4. PODZIAŁ PRAC

pojawi się problem z narzędziem, do rozwiązania którego niezbędne jest wsparcie programistów, wystarczy dodać odpowiednie zadanie opisujące problem i dodać zespół programistów do projektu, potencjalnie przypisując osobę do wykonania zadania.

Należy również powiedzieć, że tak wdrożony system również pozwoliłby na współpracę różnych zespołów księgowych. Na przykład, jeśli wybrany zespół ma wiele deadline'ów związanych z pewnym klientem, to może podzielić się tą pracą z zespołem mniej obciążonym.

### 1.4. Podział prac

Prace nad systemem zostały podzielone w następujący sposób (MM - Maksim Makaranka, VS - Vladyslav Shestakov):

1. Ogólny projekt systemu, specyfikacja wymagań - MM, VS
2. Projekt aplikacji serwerowej - MM
3. Projekt bazy danych - MM
4. Projekt aplikacji klienckiej i interfejsu użytkownika - VS
5. Implementacja serwera - MM
6. Implementacja aplikacji klienckiej - VS

Podział pracy nad niniejszym tekstem jest uciążliwy, ponieważ wszystkie rozdziały zostały napisane wspólnie, za wyjątkiem części ściśle odpowiadających aplikacji serwerowej i bazie danych (MM) oraz aplikacji klienckiej (VS).

## 2. Specyfikacja wymagań

Z ogólnej wizji systemu wynika podział logiczny aplikacji na następujące moduły:

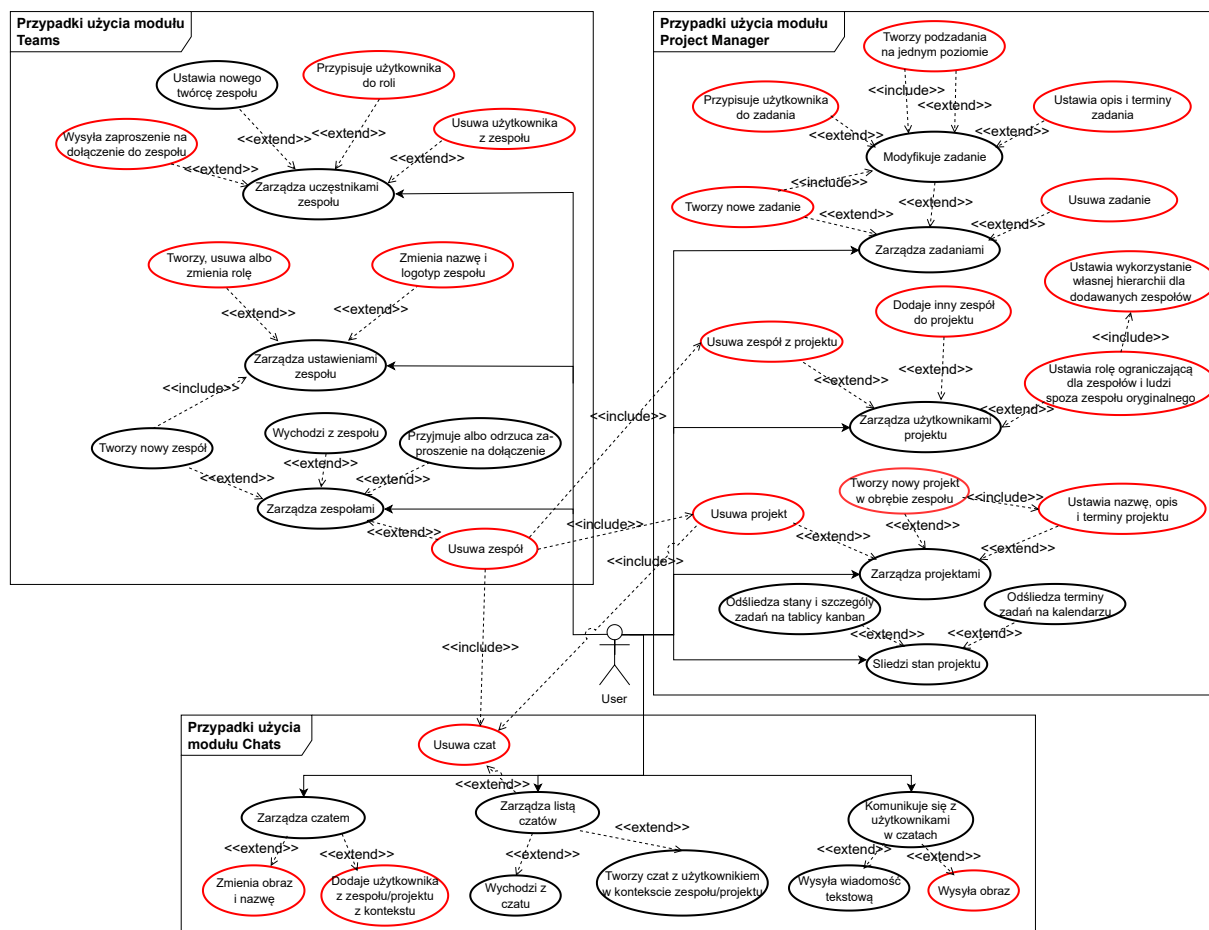
- **Project Manager** - moduł aplikacji, odpowiadający za zarządzanie pracą, w tym tworzenie, usuwanie i modyfikacja projektów i zadań, dodawanie użytkowników do projektu, przypisanie ich do zadań i inne
- **Teams** - moduł aplikacji, odpowiadający za zespoły oraz zarządzanie nimi, w tym tworzenie oraz usuwanie zespołów, zarządzanie rolami nadającymi różne uprawnienia, dodawanie nowych użytkowników i inne.
- **Chats** - moduł aplikacji, odpowiadający za komunikację pomiędzy użytkownikami współdzielonych zespołów bądź projektów

Dostęp do poszczególnych czynności w różnych modułach może wymagać odpowiednich uprawnień nadanych poprzez przypisanie do roli w zespole. Również każdy moduł połączony jest z modułem bazy danych, który obsługuje żądania odpowiadające funkcjom.

### 2.1. Diagram przypadków użycia

System posiada jednego aktora, który ma uprawnienia w modułach zależące od roli, nadanej mu w zespole, oraz od roli ograniczającej w przypadku uprawnień projektu. Na diagram przedstawionym na rysunku 2.1 czynności wymagające uprawnienia są zaznaczone na czerwono.

## 2.2. HISTORIE UŻYTKOWNIKA



Rysunek 2.1: Diagram przypadków użycia aplikacji

## 2.2. Historie użytkownika

Na podstawie powyższego diagramu formalnie opiszemy funkcjonalności za pomocą historii użytkownika. Historiom przypisujemy priorytety metodą *MoSCoW*. Taka lista stanowi wyznacznik wymagań, które spełnia aplikacja.

Każdy użytkownik aplikacji posiada dostęp do wszystkich modułów, dlatego aktorów, podanych na historiach, należy rozpatrywać z punktu spojrzenia uprawnień posiadanych w skutek nadanej roli czynności.



### 2.2.1. Historie użytkownika modułu Teams

Moduł **Teams** jest podstawowym modułem systemu, skoro de-facto jest częścią pozwalającą na kooperację z innymi użytkownikami. Do utworzonych zespołów można dodawać innych ludzi, a system ról pozwala na grupowanie tych użytkowników oraz nadawanie im uprawnień.

Członkowie zespołu w obrębie tego zespołu oraz związanych projektów i chatów będą posiadali dostęp wyłącznie do tych funkcjonalności, na które pozwala ich rola. Otrzymujemy sposób na elastyczną konfigurację tworzonych zespołów oraz dopasowanie go na konkretne potrzeby.

Historie użytkownika modułu zostały przedstawione w tabeli 2.1.

Jako ... User	Chcę...	Aby...	MoSCoW
dowolny	założyć konto	mieć profil w systemie	Must
dowolny	tworzyć zespoły	współpracować z innymi ludźmi	Must
dowolny	wychodzić z zespołów	kończyć z nimi współpracę	Must
uprawniony	tworzyć role w zespole	tworzyć hierarchię użytkowników dopasowaną do zespołu	Must
uprawniony	zmieniać uprawnienia ról	nadawać użytkownikom uprawnienia dopasowane do zespołu	Must
uprawniony	usuwać role	eliminować nieaktualne role	Must
uprawniony	dodawać zespół do projektów	współpracować z innymi ludźmi i zespołami	Must
uprawniony	usuwać zespół	eliminować zespół jeśli się rozpadnie	Must
uprawniony	wysyłać zaproszenia na dołączenie do zespołu	poszerzyć zespół o nowe osoby	Must
dowolny	akceptować lub odrzucać zaproszenia na dołączenie	nie współpracować z niepożądanymi ludźmi	Must
uprawniony	przypisywać użytkownika do roli	zmieniać nadane mu uprawnienia	Must
uprawniony	usuwać użytkownika z zespołu	kończyć z nim współpracę	Must
uprawniony	zmieniać logotyp i nazwę zespołu	dostosować zespół	Should
uprawniony	tworzyć posty na specjalnej tablicy zespołu	w taki sposób zgłaszać problemy i prosić o pomoc	Could

Tabela 2.1: Historie użytkownika modułu **Teams**

## 2.2.2. Historie użytkownika modułu Project Manager

W modułu **Project Manager** uprawnienia użytkowników są uzależnione również od roli ograniczającej i ustawień wykorzystania własnej hierarchii. Zatem pod uprawnioną osobą rozumiemy albo członka zespołu-twórcy projektu o odpowiednim uprawnieniu, albo członka zespołu zewnętrznego, przy czym wykonywana czynność powinna być dozwolona przez rolę ograniczającą oraz rolę osoby w jej zespole, jeśli ustawiono wykorzystanie własnej hierarchii. Historie użytkownika modułu zostały przedstawione w tabelach 2.2 i 2.3.

Jako ... User	Chcę...	Aby...	MoSCoW
uprawniony w zespole	stworzyć nowy projekt	zorganizować pracę	Must
uprawniony	zmienić nazwę, opis i termin ostateczny projektu	dopasować projekt oraz ustalić go cele i wymagania	Must
dowolny	widzieć tablicę kanban projektu z informacjami o zadaniach	śledzić za wykonaniem poszczególnych zadań	Must
dowolny	widzieć kalendarz projektu	widzieć terminy zadań	Could
uprawniony	dodawać wydarzenia do kalendarza projektu	zgłaszać wydarzenia ogólne	Could
uprawniony	dodawać do projektu zespoły zewnętrzne	współpracować z innymi ludźmi w obrębie projektu	Must
uprawniony	ustawić rolę ograniczającą	zarządzać uprawnieniami zewnętrznych zespołów	Must
uprawniony	ustawić wykorzystanie własnej hierarchii dla zespołów	elastyczniej zarządzać uprawnieniami	Must
uprawniony	usunąć z projektu zespół	żeby eliminować tych z którymi skończono współpracę	Must
uprawniony	stworzyć zadanie	żeby opisać pracę do zrobienia	Must
uprawniony	przypisać użytkownika do zadania	ustalić kto jest za niego odpowiedzialny	Must
uprawniony	stworzyć podzadanie	podzielić duże zadanie na mniejsze	Should
uprawniony	zmienić opis i terminy zadania	żeby ustalić do kiedy i co trzeba zrobić	Must

Tabela 2.2: Historie użytkownika modułu **Project Manager**

Jako ... User	Chcę...	Aby...	MoSCoW
uprawniony	usunąć zadanie	eliminować odwołane zadania	Must
uprawniony	usunąć projekt	skończyć pracę nad nim	Must
uprawniony	dodawać sprint'y i zmieniać ich terminy	stosować metodykę SCRUM	Could

Tabela 2.3: Historie użytkownika modułu **Project Manager** (cd.)

### 2.2.3. Historie użytkownika modułu Chats

Czaty są tworzone w kontekście zespołów albo projektów. To oznacza, że w ramach czatu zachodzą takie same uprawnienia, jak w encji kontekstowej. Również tylko członkowie tych encji mogą zostać dodane do czatu. Historie użytkownika modułu zostały przedstawione w tabeli 2.4.

Jako ... User	Chcę...	Aby...	MoSCoW
dowolny	stworzyć nowy czat w kontekście projektu albo zespołu	komunikować się z innym użytkownikiem	Must
uprawniony	zmienić obraz i nazwę czatu	dopasować czat	Must
uprawniony	dodać użytkownika z zespołu/projektu do czatu	tworzyć czaty grupowe	Must
dowolny	wysłać wiadomość tekstową	przekazywać tekst użytkownikom	Must
dowolny	wyjść z czatu	nie brać udziału w bezużytecznych konwersacjach	Must
uprawniony	wysłać obraz	przekazywać grafikę użytkownikom	Should
uprawniony	wysłać plik	przekazywać pliki użytkownikom	Could

Tabela 2.4: Historie użytkownika modułu **Chats**

### 2.3. Opis funkcjonalności systemu

Aby w większym stopniu zrozumieć mechanizmy działania systemu, opiszemy funkcjonalności przy pomocy tabel. Wyszczególniamy ich warunki, wykorzystywane dane i wyniki.

#### 2.3.1. Moduł Teams

Moduł **Teams** pozwala na szereg funkcjonalności z zakresu zarządzania zespołami, ich użytkownikami oraz ustawieniami.

#### Zarządzanie zespołami

Do zarządzania zespołami wlicza się tworzenie zespołu, usuwanie i wychodzenie z zespołu, oraz akceptacja lub odrzucenie zaproszenia na dołączenie się do zespołu. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.5, 2.6, 2.7, 2.8 oraz 2.9.

<b>Nazwa</b>	Tworzenie zespołu
<b>Dane wejściowe</b>	Nazwa i logotyp zespołu
<b>Stan początkowy</b>	Zespół nie istnieje
<b>Stan końcowy</b>	Zespół o podanej nazwie i logotypie istnieje
<b>Wynik</b>	Stworzenie zespołu, pojawienie się go na liście zespołów
<b>Skutki uboczne</b>	Nadanie twórcy zespołu roli twórcy, nadającej maksymalne uprawnienia

Tabela 2.5: Opis tworzenia zespołu

<b>Nazwa</b>	Akceptacja zaproszenia
<b>Dane wejściowe</b>	Identyfikator zaproszenia
<b>Stan początkowy</b>	Użytkownik nie jest uczestnikiem zespołu i ma otwarte zaproszenie
<b>Stan końcowy</b>	Użytkownik jest uczestnikiem zespołu
<b>Wynik</b>	Użytkownik został dodany do zespołu
<b>Skutki uboczne</b>	Nadanie użytkownikowi domyślnej roli "Guest", nadającej minimalne uprawnienia

Tabela 2.6: Opis akceptacji zaproszenia

<b>Nazwa</b>	Odrzucenie zaproszenia
<b>Dane wejściowe</b>	Identyfikator zaproszenia
<b>Stan początkowy</b>	Użytkownik nie jest uczestnikiem zespołu i ma otwarte zaproszenie
<b>Stan końcowy</b>	Użytkownik nie jest uczestnikiem i nie ma otwartego zaproszenia z tego zespołu
<b>Wynik</b>	Użytkownik odrzucił zaproszenie
<b>Skutki uboczne</b>	Brak

Tabela 2.7: Opis odrzucenia zaproszenia

<b>Nazwa</b>	Usuwanie zespołu
<b>Dane wejściowe</b>	Identyfikator zespołu
<b>Stan początkowy</b>	Zespół istnieje, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół nie istnieje
<b>Wynik</b>	Usunięcie zespołu z listy zespołów
<b>Skutki uboczne</b>	Usunięcie zespołu z projektów, do których on był dodany

Tabela 2.8: Opis usuwania zespołu

<b>Nazwa</b>	Wychodzenie z zespołu
<b>Dane wejściowe</b>	Identyfikator zespołu
<b>Stan początkowy</b>	Użytkownik jest w zespole
<b>Stan końcowy</b>	Użytkownik nie jest w zespole
<b>Wynik</b>	Usunięcie użytkownika z listy uczestników zespołu
<b>Skutki uboczne</b>	Usunięcie projektów zespołu z listy projektów użytkownika, usunięcie użytkownika ze wszystkich konwersacji związanych z tym zespołem i projektami

Tabela 2.9: Opis wychodzenia z zespołu

**Zarządzanie uczestnikami zespołu**

Do zarządzania uczestnikami zespołu wlicza się wysłanie do użytkownika zaproszenia na dołączenie się do zespołu, przypisanie mu roli oraz usuwanie użytkownika. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.10, 2.11 oraz 2.12.

<b>Nazwa</b>	Wysłanie do użytkownika zaproszenia
<b>Dane wejściowe</b>	Identyfikator użytkownika oraz identyfikator zespołu
<b>Stan początkowy</b>	Użytkownik nie jest w zespole, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Użytkownik otrzymuje zaproszenie
<b>Wynik</b>	Zaproszenie zostało wysłane
<b>Skutki uboczne</b>	Brak

Tabela 2.10: Opis zarządzania uczestnikami zespołu

<b>Nazwa</b>	Przypisanie roli do użytkownika
<b>Dane wejściowe</b>	Identyfikator użytkownika, identyfikator zespołu oraz identyfikator roli do przypisania
<b>Stan początkowy</b>	Użytkownik jest w zespole, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Użytkownik ma przypisaną żadaną rolę
<b>Wynik</b>	Nadanie uprawnień zdefiniowanych w roli
<b>Skutki uboczne</b>	Zmiana dostępnych funkcjonalności w zależności od nowych uprawnień

Tabela 2.11: Opis przypisania roli do użytkownika

<b>Nazwa</b>	Usuwanie użytkownika z zespołu
<b>Dane wejściowe</b>	Identyfikator użytkownika oraz identyfikator zespołu
<b>Stan początkowy</b>	Użytkownik jest w zespole, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Użytkownik nie jest w zespole
<b>Wynik</b>	Usunięcie użytkownika z listy użytkowników zespołu
<b>Skutki uboczne</b>	Projekty i konwersacje tego zespołu są niedostępne dla usuniętego użytkownika

Tabela 2.12: Opis usuwania użytkownika z zespołu

## Zarządzanie rolami zespołu

Do zarządzania rolami zespołu wlicza się tworzenie nowej roli, zmiana istniejącej oraz usunięcie roli. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.13, 2.14 oraz 2.15

<b>Nazwa</b>	Tworzenie nowej roli
<b>Dane wejściowe</b>	Identyfikator zespołu, nazwa roli oraz uprawnienia, które ta rola powinna nadawać
<b>Stan początkowy</b>	Brak roli o wymaganej nazwie, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół posiada rolę o wymaganych nazwie oraz uprawnieniach
<b>Wynik</b>	Dodanie roli do listy ról zespołu
<b>Skutki uboczne</b>	Brak

Tabela 2.13: Opis tworzenia nowej roli

<b>Nazwa</b>	Zmiana ustawień roli
<b>Dane wejściowe</b>	Identyfikator roli, nazwa roli oraz nowe uprawnienia
<b>Stan początkowy</b>	Rola posiada stare ustawienia, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Rola posiada nowe uprawnienia
<b>Wynik</b>	Zmiana uprawnień roli oraz czynności dostępnych jej posiadaczom
<b>Skutki uboczne</b>	Brak

Tabela 2.14: Opis zmiany ustawień roli

<b>Nazwa</b>	Usuwanie roli
<b>Dane wejściowe</b>	Identyfikator roli
<b>Stan początkowy</b>	Rola o podanym identyfikatorze istnieje, rola nie jest przypisana żadnemu użytkownikowi, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Rola o podanym identyfikatorze nie istnieje
<b>Wynik</b>	Rola została usunięta
<b>Skutki uboczne</b>	Brak

Tabela 2.15: Opis usuwania roli

### Zarządzanie ustawieniami zespołu

Jedyną funkcjonalnością dotyczącą zarządzaniem ustawieniami zespołu jest ich zmiana. Opis tej funkcjonalności został przedstawiony w tabeli 2.16.

<b>Nazwa</b>	Zmiana nazwy oraz logotypu zespołu
<b>Dane wejściowe</b>	Identyfikator zespołu, nowa nazwa oraz nowy obraz
<b>Stan początkowy</b>	Zespół posiada stare nazwę oraz logotyp, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół posiada nową nazwę oraz logotyp
<b>Wynik</b>	Ustawienia zespołu zostały zmienione
<b>Skutki uboczne</b>	Zmiana jest widoczna dla członków zespołu

Tabela 2.16: Opis zarządzania ustawieniami zespołu



### 2.3.2. Moduł Project Manager

Moduł **Project Manager** pozwala na szereg funkcjonalności z zakresu zarządzania projektami, ich uczestnikami oraz zadaniami stworzonymi w projekcie.

#### Zarządzanie projektami

Do zarządzania projektami wlicza się tworzenie, wyjście i usunięcie projektu. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.17, 2.18 oraz 2.19.

<b>Nazwa</b>	Tworzenie projektu
<b>Dane wejściowe</b>	Nazwa, opis, logotyp, data początku i termin ostateczny, nazwa roli ograniczającej wraz z jej uprawnieniami, identyfikator zespołu założyciela, polityka ograniczeń
<b>Stan początkowy</b>	Brak projektu posiadającego wymagane cechy
<b>Stan końcowy</b>	Projekt posiadający wymagane cechy istnieje
<b>Wynik</b>	Projekt z wymaganymi cechami został stworzony
<b>Skutki uboczne</b>	Pojawienie się projektu na liście projektów wszystkich członków zespołu

Tabela 2.17: Opis tworzenia projektu

<b>Nazwa</b>	Wyjście z projektu
<b>Dane wejściowe</b>	Identyfikator projektu i zespołu
<b>Stan początkowy</b>	Projekt jest dostępny dla uczestników zespołu, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół nie jest uczestnikiem danego projektu
<b>Wynik</b>	Zespół wyszedł z projektu
<b>Skutki uboczne</b>	Wszystkie dane związane z danym zespołem zostają zmienione na domyślne

Tabela 2.18: Opis wyjścia z projektu

### 2.3. OPIS FUNKCJONALNOŚCI SYSTEMU

<b>Nazwa</b>	Usunięcie projektu
<b>Dane wejściowe</b>	Identyfikator projektu
<b>Stan początkowy</b>	Projekt jest dostępny dla uczestników, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Projekt o podanym identyfikatorze nie istnieje
<b>Wynik</b>	Projekt został usunięty
<b>Skutki uboczne</b>	Wszystkie dane powiązane z projektem zostały usunięte

Tabela 2.19: Opis usunięcia projektu

#### Zarządzanie ustawieniami projektu

Jedyną funkcjonalnością dotyczącą zarządzaniem ustawieniami projektu jest ich zmiana. Opis tej funkcjonalności został przedstawiony w tabeli 2.20.

<b>Nazwa</b>	Zmiana ustawień projektu: nazwy, opisu, logotyp, daty początku i terminu ostatecznego, nazwy i uprawnień roli ograniczającej oraz polityki ograniczeń
<b>Dane wejściowe</b>	Zaktualizowane ustawienia projektu, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan początkowy</b>	Projekt nie posiada wymaganych cech
<b>Stan końcowy</b>	Projekt posiada wymagane cechy
<b>Wynik</b>	Ustawienia projektu zostały zmienione
<b>Skutki uboczne</b>	Zmiany są widoczne dla wszystkich uczestników projektu

Tabela 2.20: Opis zarządzania ustawieniami projektu

#### Zarządzanie uczestnikami projektu

Do zarządzania uczestnikami projektu wlicza się dodanie i usunięcie zespołu do/z projektu oraz wyjście zespołu użytkownika z projektu. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.21, 2.22 oraz 2.23.

<b>Nazwa</b>	Dodanie zespołu do projektu
<b>Dane wejściowe</b>	Identyfikator zespołu i projektu
<b>Stan początkowy</b>	Zespół nie jest uczestnikiem projektu, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół jest uczestnikiem projektu
<b>Wynik</b>	Zespół został dodany do projektu i jest dostępny dla wszystkich uczestników zespołu
<b>Skutki uboczne</b>	Uprawnienia uczestników zespołu w obrębie danego projektu zostają ustawione zgodnie z rolą ograniczającą i polityką ograniczeń

Tabela 2.21: Opis dodania zespołu do projektu

<b>Nazwa</b>	Usunięcie zespołu z projektu
<b>Dane wejściowe</b>	Identyfikator zespołu i projektu
<b>Stan początkowy</b>	Zespół jest uczestnikiem projektu, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół nie jest uczestnikiem w projekcie
<b>Wynik</b>	Zespół został usunięty z projektu, który teraz nie jest dostępny dla uczestników zespołu
<b>Skutki uboczne</b>	Wszystkie dane związane z zespołem usuniętym zostają zmienione na domyślne

Tabela 2.22: Opis usunięcia zespołu z projektu

<b>Nazwa</b>	Wyjście zespołu z projektu
<b>Dane wejściowe</b>	Identyfikator zespołu i projektu
<b>Stan początkowy</b>	Zespół jest uczestnikiem projektu, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zespół nie jest uczestnikiem w projekcie
<b>Wynik</b>	Zespół został usunięty z projektu, który teraz nie jest dostępny dla uczestników zespołu
<b>Skutki uboczne</b>	Wszystkie dane związane z zespołem usuniętym zostają zmienione na domyślne

Tabela 2.23: Opis wyjścia zespołu z projektu

**Zarządzanie zadaniami**

Do zarządzania zadaniami wlicza się tworzenie, usuwanie oraz edycja zadania oraz podzadań. Opis tych funkcjonalności zostały przedstawione w tabelach 2.24, 2.25, 2.26, 2.27, 2.28, 2.29, 2.30 oraz 2.31.

<b>Nazwa</b>	Tworzenie zadania
<b>Dane wejściowe</b>	Identyfikator projektu, nazwa, opis, przypisany użytkownik, data początku i termin ostateczny zadania
<b>Stan początkowy</b>	Zadanie o podanych cechach nie istnieje, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zadanie o podanych cechach istnieje
<b>Wynik</b>	Zadanie zostało stworzone
<b>Skutki uboczne</b>	Zadanie jest widoczne w tablicy kanban oraz kalendarzu przy odpowiednim trybie wyświetlania zadań

Tabela 2.24: Opis tworzenia zadania

<b>Nazwa</b>	Zmiana ustawień zadania: nazwa, opis, przypisany użytkownik, data początku i termin ostateczny
<b>Dane wejściowe</b>	Identyfikator zadania i nowe ustawienia
<b>Stan początkowy</b>	Zadanie posiada początkowe ustawienia, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zadanie posiada zmienione ustawienia
<b>Wynik</b>	Ustawienia zadania zostały zmienione i są widoczne dla innych użytkowników
<b>Skutki uboczne</b>	Zmiany daty początkowej i terminu ostatecznego są widoczne w kalendarzu przy odpowiednim trybie wyświetlania zadań

Tabela 2.25: Opis zmiany ustawień zadania

<b>Nazwa</b>	Zmiana statusu zadania
<b>Dane wejściowe</b>	Identyfikator zadania i nowy status
<b>Stan początkowy</b>	Zadanie posiada początkowy status, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zadanie posiada zmieniony status
<b>Wynik</b>	Status zadania został zmieniony
<b>Skutki uboczne</b>	Zmiana statusu jest widoczna na tablicy kanban przy odpowiednim trybie wyświetlania zadań

Tabela 2.26: Opis zmiany statusu zadania

<b>Nazwa</b>	Usunięcie zadania
<b>Dane wejściowe</b>	Identyfikator zadania
<b>Stan początkowy</b>	Zadanie o podanym identyfikatorze istnieje, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Zadanie o podanym identyfikatorze nie istnieje
<b>Wynik</b>	Zadanie zostało usunięte i nie jest widoczne dla użytkowników
<b>Skutki uboczne</b>	Jeśli zadanie miało podzadania, zostały one również usunięte

Tabela 2.27: Opis usunięcia zadania

<b>Nazwa</b>	Tworzenie podzadania
<b>Dane wejściowe</b>	Identyfikator projektu i zadania głównego, nazwa, opis, przypisany użytkownik, data początku i termin ostateczny podzadania
<b>Stan początkowy</b>	Podzadanie nie istnieje, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Podzadanie o podanych cechach istnieje
<b>Wynik</b>	Podzadanie zostało stworzone
<b>Skutki uboczne</b>	Zadanie jest widoczne w panelu edycji zadania głównego oraz w tablicy kanban i kalendarzu przy odpowiednim trybie wyświetlania zadań

Tabela 2.28: Opis tworzenia podzadania

## 2.3. OPIS FUNKCJONALNOŚCI SYSTEMU

<b>Nazwa</b>	Zmiana ustawień podzadania: nazwa, opis, przypisany użytkownik, data początku i termin ostateczny
<b>Dane wejściowe</b>	Identyfikator podzadania i nowe ustawienia
<b>Stan początkowy</b>	Podzadanie posiada początkowe ustawienia, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Podzadanie posiada zmienione ustawienia
<b>Wynik</b>	Ustawienia podzadania zostały zmienione
<b>Skutki uboczne</b>	Zmiany daty początkowej i terminu ostatecznego są widoczne w kalendarzu przy odpowiednim trybie wyświetlania zadań

Tabela 2.29: Opis zmiany ustawień podzadania

<b>Nazwa</b>	Zmiana statusu podzadania
<b>Dane wejściowe</b>	Identyfikator zadania i nowy status
<b>Stan początkowy</b>	Podzadanie posiada początkowy status, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Podzadanie posiada zmieniony status
<b>Wynik</b>	Status podzadania został zmieniony
<b>Skutki uboczne</b>	Zmiana statusu jest widoczna na tablicy kanban przy odpowiednim trybie wyświetlania zadań

Tabela 2.30: Opis wyjścia z projektu

<b>Nazwa</b>	Usunięcie podzadania
<b>Dane wejściowe</b>	Identyfikator podzadania
<b>Stan początkowy</b>	Podzadanie o podanym identyfikatorze istnieje, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Podzadanie o podanym identyfikatorze nie istnieje
<b>Wynik</b>	Podzadanie zostało usunięte i nie jest widoczne dla uczestników
<b>Skutki uboczne</b>	Brak

Tabela 2.31: Opis usunięcia podzadania

### 2.3.3. Moduł Chats

Moduł **Chats** jest modulem służącym do komunikacji między użytkownikami w obrębie zespołu lub projektu. Czyli użytkownicy, którzy nie mają wspólnych projektów/zespołów nie mogą komunikować między sobą. Uczestnicy chatu mogą wysyłać zwykły tekst lub obrazki, dodawać nowych użytkowników i zmieniać ustawienia chatu.

#### Zarządzanie listą czatów

Do zarządzania listą czatów wlicza się utworzenie i usunięcie czatu. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.32 oraz 2.33.

<b>Nazwa</b>	Tworzenie czatu
<b>Dane wejściowe</b>	Nazwa czatu, identyfikator drugiej osoby oraz kontekst (identyfikator zespołu lub projektu)
<b>Stan początkowy</b>	Czat nie istnieje
<b>Stan końcowy</b>	Czat istnieje
<b>Wynik</b>	Użytkownik widzi nowy czat na swojej liście czatów
<b>Skutki uboczne</b>	Inny uczestnik czatu też widzi czat na swojej liście

Tabela 2.32: Opis tworzenia czatu

<b>Nazwa</b>	Wyjście z czatu
<b>Dane wejściowe</b>	Identyfikator czatu
<b>Stan początkowy</b>	Użytkownik jest uczestnikiem czatu
<b>Stan końcowy</b>	Użytkownik nie jest uczestnikiem czatu
<b>Wynik</b>	Użytkownik wyszedł z chatu i nie widzi go na swojej liście chatów
<b>Skutki uboczne</b>	Inni uczestnicy czatu widzą wiadomość o wyjściu użytkownika z czatu

Tabela 2.33: Opis wyjścia z czatu

#### Zarządzanie czatem

Do zarządzania czatem wlicza się zmiana obrazu i nazwy czatu, dodanie i usunięcie użytkownika oraz wyjście z chatu. Opisy tych funkcjonalności zostały przedstawione w tabelach 2.34, 2.35, 2.36 oraz 2.37.

## 2.3. OPIS FUNKCJONALNOŚCI SYSTEMU

<b>Nazwa</b>	Zmiana obrazu i nazwy czatu
<b>Dane wejściowe</b>	Identyfikator czatu, obraz i nazwa czatu
<b>Stan początkowy</b>	Chat posiada początkowe obraz i nazwę, wykonawca czynności posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Chat posiada nowe obraz i nazwę
<b>Wynik</b>	Zmiany są widoczne dla uczestników chatu
<b>Skutki uboczne</b>	Użytkownicy widzą wiadomość o zmianie obrazu i nazwy chatu

Tabela 2.34: Opis zmiany obrazu i nazwy czatu

<b>Nazwa</b>	Dodanie użytkownika do czatu
<b>Dane wejściowe</b>	Identyfikator czatu i użytkownika
<b>Stan początkowy</b>	Użytkownik nie jest uczestnikiem czatu, ale jest uczestnikiem odpowiedniego zespołu lub projektu, użytkownik dodający posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Użytkownik jest uczestnikiem czatu
<b>Wynik</b>	Użytkownik dodany widzi czat na swojej liście czatów oraz zawartą w nim konwersację
<b>Skutki uboczne</b>	Inni uczestnicy czatu widzą wiadomość o dodaniu nowego użytkownika

Tabela 2.35: Opis dodania użytkownika do czatu

<b>Nazwa</b>	Usunięcie użytkownika z czatu
<b>Dane wejściowe</b>	Identyfikator czatu i użytkownika
<b>Stan początkowy</b>	Użytkownik jest uczestnikiem czatu, użytkownik usuwający posiada niezbędne uprawnienia
<b>Stan końcowy</b>	Użytkownik nie jest uczestnikiem czatu
<b>Wynik</b>	Użytkownik usunięty nie widzi czat na swojej liście czatów
<b>Skutki uboczne</b>	Inni uczestnicy czatu widzą wiadomość o usunięciu użytkownika

Tabela 2.36: Opis usunięcia użytkownika z czatu



<b>Nazwa</b>	Wyjście z czatu
<b>Dane wejściowe</b>	Identyfikator czatu i użytkownika
<b>Stan początkowy</b>	Użytkownik jest uczestnikiem czatu
<b>Stan końcowy</b>	Użytkownik nie jest uczestnikiem czatu
<b>Wynik</b>	Użytkownik wyszedł z czatu i nie widzi chat na swojej liście chatów
<b>Skutki uboczne</b>	Inni uczestnicy czatu widzą wiadomość o wyjściu użytkownika

Tabela 2.37: Opis wyjścia z czatu

## Komunikacja

Jedynym elementem komunikacji jest wysłanie wiadomości tekstowej wraz z opcjonalnym obrazkiem. Opis tej funkcjonalności został przedstawiony w tabeli 2.38.

<b>Nazwa</b>	Wysłanie wiadomości
<b>Dane wejściowe</b>	Identyfikator czatu i wiadomość
<b>Stan początkowy</b>	Wiadomość jest wpisana w odpowiednie miejsce i (opcjonalnie) obrazek jest załadowany
<b>Stan końcowy</b>	Wiadomość widoczna dla uczestników chatu
<b>Wynik</b>	Wiadomość została wysłana
<b>Skutki uboczne</b>	Inni uczestnicy chatu otrzymują powiadomienie o nowej wiadomości, o ile znajdują się na stronie chatów

Tabela 2.38: Opis wysłania wiadomości

### 2.4. Wymagania niefunkcjonalne

Zbiór tych wymagań opisuje, jakie wymagania wobec systemu mają być spełnione, oprócz wymagań funkcjonalnych. Głównie skupiają się na sposobie pracy, wydajności, bezpieczeństwie i innych podobnych rzeczach.

Stworzona aplikacja jest aplikacją internetową, dlatego raczej niema żadnych wymagań dotyczących wydajności. Jedynym, czego potrzebuje użytkownik do korzystania z systemu, to jest połączenie internetowe oraz przeglądarka. Zaleca się, żeby była to przeglądarka wspierająca protokół *WebSocket*, ponieważ usługa *SignalR* z niego korzysta, ale użycie starszych wersji przeglądarek też nie powinno stanowić większego problemu, ponieważ *SignalR* wspiera również starsze transporty (np. *EventSource*).

Również gorąco zaleca się stosowanie myszki, ponieważ to znacznie upraszcza korzystanie z niektórych elementów systemu (np. z tablicy kanban).

## 3. Opis rozwiązania

### 3.1. Architektura systemu

Stworzona aplikacja webowa składa się z 3 części izolujących fundamentalne strefy odpowiedzialności systemu - aplikacji klienckiej, serwera udostępniającego API oraz bazy danych. Taki podział pozwala na uniezależnienie logiki biznesowej systemu od aplikacji klienckiej oraz wyodrębnienie warstwy dostępu do danych. Dzięki temu zachodzi możliwość modułowej pracy nad systemem oraz niezależnego rozwoju poszczególnych części przez członków zespołu.

#### 3.1.1. Serwer

Przy tworzeniu serwera wykorzystana platforma *ASP.NET Core*[7]. Pozwala na tworzenie aplikacji internetowych z ujednoliconym scenariuszem na potrzeby tworzenia interfejsu użytkownika i interfejsów API. Platforma również udostępnia szerokie możliwości testowania systemu, mechanizm wstrzykiwania zależności oraz łatwą integracją innych technologii *.NET*.

Serwer budowany jest zgodnie z zasadami czystej architektury[15] oraz *DDD*[21], wyodrębnione są warstwy obsługi przepływów procesów biznesowych (**Application**), logiki biznesowej (**Domain**) oraz dostępu do modułów zewnętrznych, w tym bazy danych (**Infrastructure**). Również wyodrębniona jest warstwa prezentacji (**WebUI**), która zawiera aplikację kliencką, oraz warstwa **Models**, definiująca *DTO-objekty* i służąca pewnego rodzaju kontraktem pomiędzy frontend'em a backend'em. Warstwa **Application** realizuje wzorzec *CQRS*[18] (*ang. Command and Query Responsibility Segregation*) za pomocą wzorca projektowego mediator, implementowanego przy użyciu biblioteki *MediatR*[22].

Szczegółowy opis architektury serwera podany jest w rozdziale 2.

#### 3.1.2. Baza danych

Baza danych jest realizowana przy użyciu silnika *Microsoft SQL Server* oraz *Entity Framework Core*[10] - technologii mapowania relacyjnego rozpracowanej przez korporację Microsoft i pozwalającej na tworzenie bazy danych i komunikację z nią na podstawie zdefiniowanych klas

### 3.1. ARCHITEKTURA SYSTEMU

obiektów. Wykorzystanie tej technologii sprowadza komunikację z bazą danych do zarządzania obiektami *C#*.

#### 3.1.3. Aplikacja kliencka

Aplikacja kliencka jest stworzona za pomocą framework'u *Blazor*[4]. Pozwala na tworzenie aplikacji webowych i natywnych z użyciem *C#*, *HTML* i *CSS*. Został użyty model hostowania *WebAssembly*, w którym komunikacja z serwerem odbywa się tylko gdy jest potrzeba w pobieraniu danych, ale o tym będzie powiedziano więcej w rozdziale 4.1.1. Aplikacja będzie podzielona na dwie części: serwisy (*Services*) do komunikacji z serwerem i pozostała część służąca do budowania widoku.

Ten framework został wybrany do tworzenia aplikacji klienckiej zamiast *Flutter*[5] z kilku powodów:

- Łatwiejsza komunikacja z serwerem
- Spójny język programowania
- Lepsze przystosowanie do tworzenia aplikacji webowych

#### 3.1.4. Komunikacja

Komunikacja jest realizowana odpowiednio ze stylem architektonicznym *REST*[19] za pomocą domyślnych narzędzi dostępnych w *.NET* w następujący sposób:

- Aplikacja kliencka wysyła zapytanie *HTTP* do serwera na odpowiedni endpoint API
- Serwer przetwarza zapytanie
- Serwer wysyła odpowiedź na zapytanie klienta

Dane są przesyłane w *DTO-objektach*, opisanych w wyodrębnionej warstwie **Models**.

Niektóre funkcjonalności wymagają komunikacji "w czasie rzeczywistym", na co nie pozwala protokół *HTTP*. Na przykład komunikator powinien wyświetlać nowe wiadomości bez potrzeby odświeżania strony. Z tego powodu została również wykorzystana usługa *SignalR*[13]. Udostępnia prosty interfejs API do tworzenia zdalnych wywołań procedur serwer-klient wywołujących funkcje Języka JavaScript w przeglądarkach klienta (i innych platformach klienckich) z kodu platformy *.NET* po stronie serwera. *SignalR* to abstrakcja niektórych transportów, które są wymagane do wykonywania pracy w czasie rzeczywistym między klientem a serwerem. W naszym systemie komunikacja odbywa się przy użyciu protokołu *WebSocket*.

### 3.2. Opis architektury serwera

#### 3.2.1. Podstawowe klasy i struktury

**Picture** - klasa odpowiadająca za obrazki, nadawane zespołom, użytkownikom, chatom. Posiada pola:

<b>Picture</b>
+ Id: long
+ ImagePath: string

**PermissionEnum** - typ wyliczeniowy predefiniujący uprawnienia nadawane użytkownikom za pomocą ról. Wartości odpowiadające różnym uprawnieniom są pogrupowane po modułu, którego te uprawnienia dotyczą. Predefiniowane wartości:

Wartość	Opis uprawnienia
TEAM_EDIT	Zmiana nazwy i obrazu zespołu
TEAM_DELETE	Usunięcie zespołu
TEAM_ADD_USER	Dodanie użytkownika do zespołu
TEAM_KICK_USER	Usunięcie użytkownika z zespołu
TEAM_ASSIGN_ROLE	Przypisanie użytkownika do roli
TEAM_MANAGE_ROLE	Dodanie, zmiana albo modyfikacja roli
TEAM_ADD_TO_PROJECT	Dodanie zespołu do projektów innych zespołów
TEAM_LEAVE_PROJECT	Wyjście całego zespołu z projektu
TEAM_DELETE_PROJECT	Usuwanie projektów stworzonych przez zespół
PM_EDIT	Zmiana nazwy, opisu i deadline'u projektu
PM_ADD_TEAM	Dodanie innych zespołów do projektu
PM_KICK_TEAM	Usunięcie zespołu razem z jego członkami z projektu
PM_ADD_USERS	Dodanie użytkownika bądź zespołu do projektu
PM_SET_LIMIT_ROLE	Ustawienie roli ograniczającej i wykorzystania własnej hierarchii
PM_CREATE_TASK	Tworzenie zadań
PM_ASSIGN_TASK	Przypisanie użytkownika do zadania
PM_CREATE_SUBTASK	Tworzenie podzadań
PM_EDIT_TASK	Zmiana opisu i deadline'u zadania
PM_DELETE_TASK	Usunięcie zadania

### 3.2. OPIS ARCHITEKTURY SERWERA

CHAT_EDIT	Zmiana obrazu i nazwy chatu
CHAT_DELETE	Usunięcie chatu
CHAT_ADD_USER	Dodanie użytkownika z zespołu/projektu do chatu
CHAT_SEND_IMAGE	Wysyłanie obrazów

**Permission** - klasa odpowiadająca za predefiniowane uprawnienia nadawane użytkownikom za pomocą ról. Klasa zawiera pola:

Permission
+ Id: PermissionEnum
+ Name: string

**Role** - klasa odpowiadająca za role tworzone i nadawane użytkownikom, zawiera pola:

Role
+ Id: long
+ Name: string
+ Permissions: List<Permission>

**User** - klasa odpowiadająca za dane użytkownika, która zawiera pola:

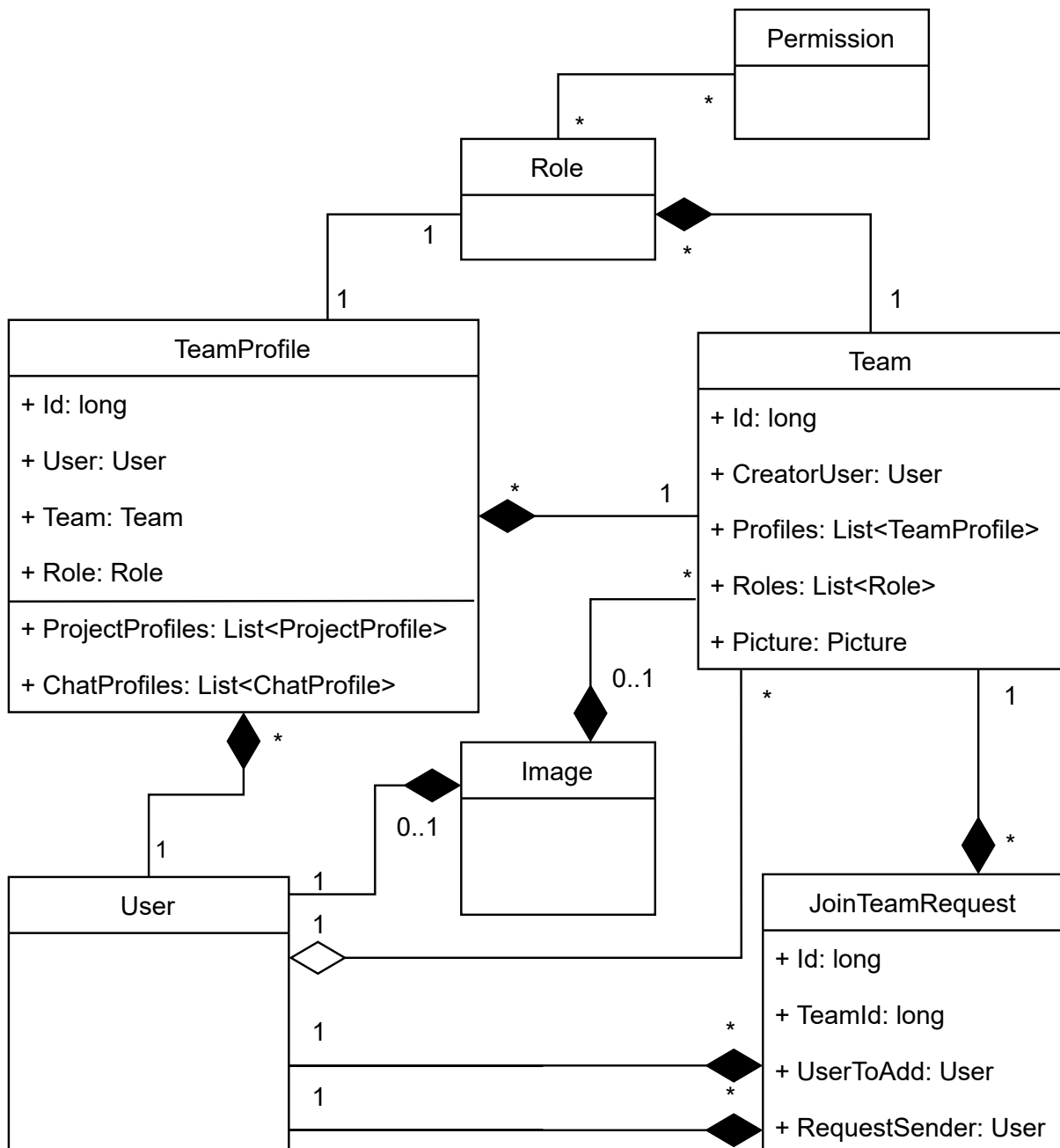
User
+ Id: string
+ UserName: string
+ Name: string
+ Surname: string
+ Image: Picture?

**TaskStateEnum** - Typ wyliczeniowy predefiniujący stany nadawane zadaniom w trakcie pracy nad nimi. Predefiniowane wartości:

Wartość	Opis stanu
BACKLOG	Stan nadawany zadaniu po stworzeniu
TODO	Zadanie jest do zrobienia
IN_PROGRESS	Zadanie jest w trakcie realizacji
REVIEW	Zadanie jest zrobione i oczekuje na weryfikację
DONE	Zadanie jest zrobione i zweryfikowane

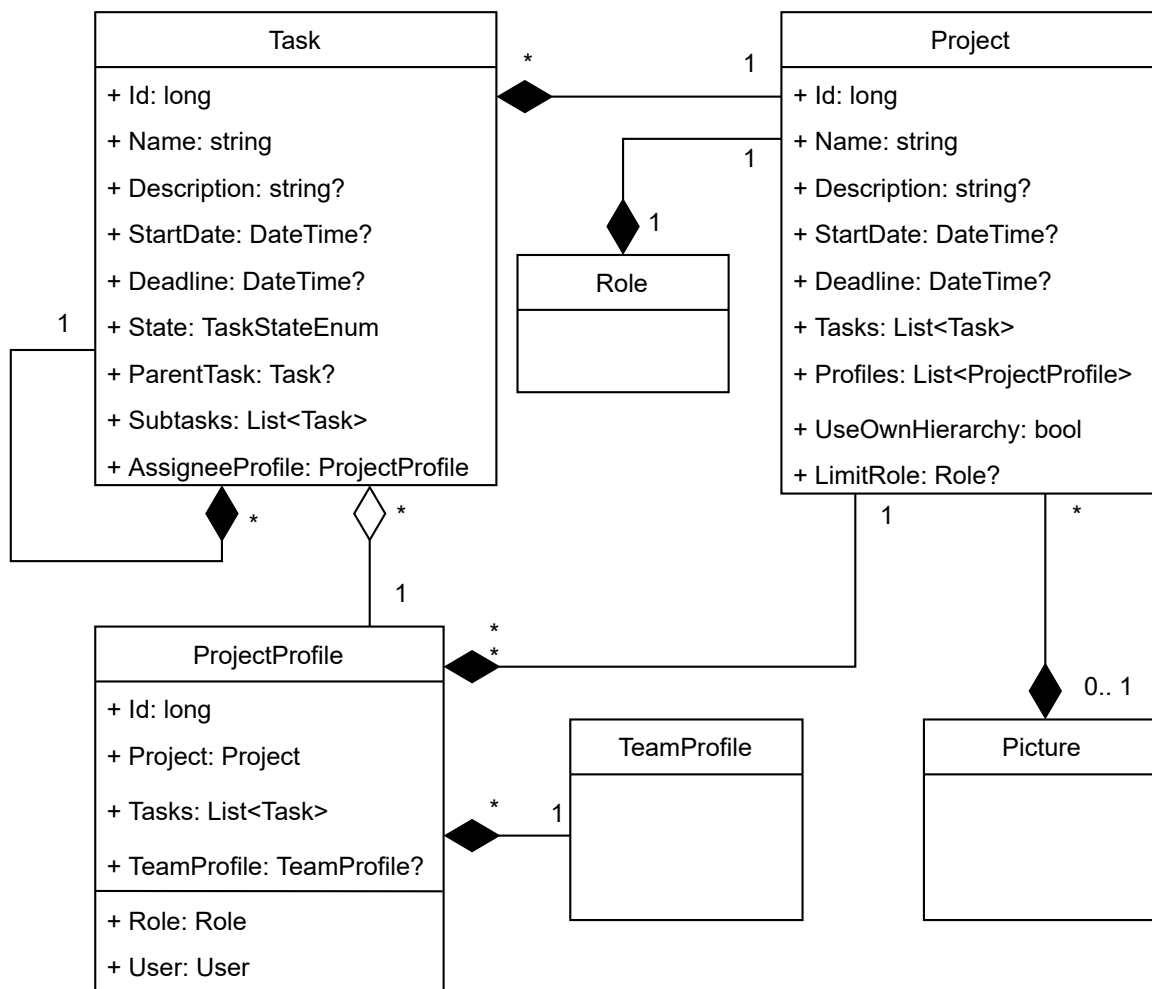
## 3.2.2. Moduł Teams

Poniżej na rysunku 3.1 jest przedstawiony diagram modułu **Teams** pozwalającego na szereg funkcjonalności z zakresu zarządzania zespołami, ich użytkownikami oraz ustawieniami. Również na diagramie zostały przedstawione relacje między klasami opisanymi w dziale 3.2.1.

Rysunek 3.1: Diagram klas modułu **Teams**

## 3.2.3. Moduł Project Manager

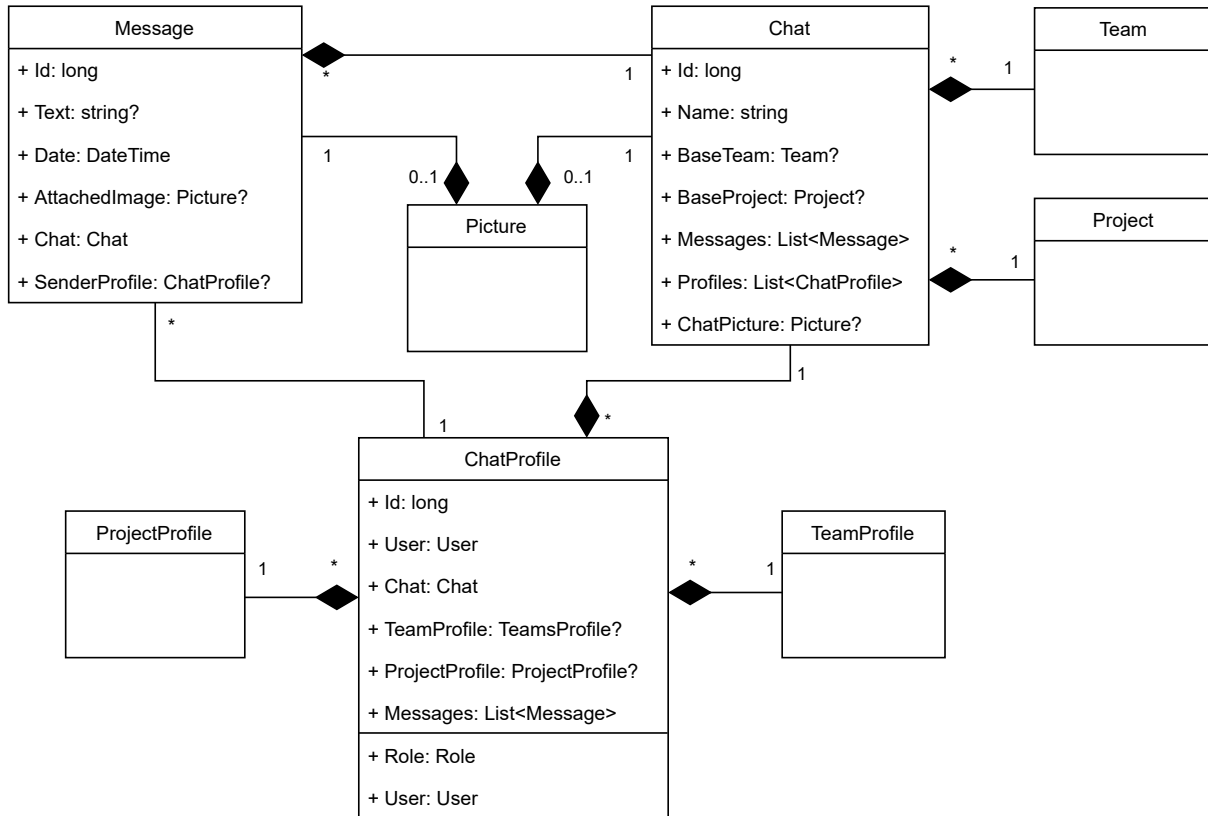
Poniżej na rysunku 3.2 jest przedstawiony diagram klas modułu **Project Manager** pozwalającego na szereg funkcjonalności z zakresu zarządzania projektami, ich uczestnikami oraz zadaniami stworzonymi w projekcie.

Rysunek 3.2: Diagram klas modułu **Project Manager**



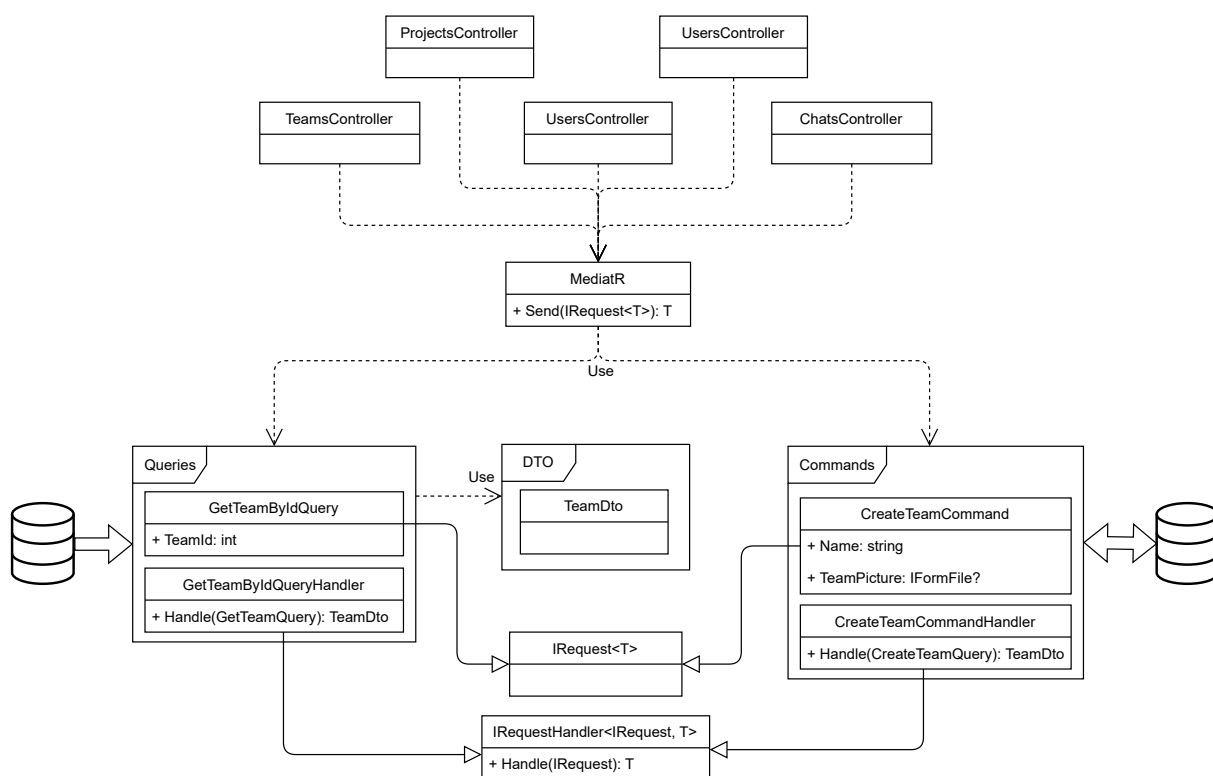
## 3.2.4. Moduł Chats

Poniżej na rysunku 3.3 jest przedstawiony diagram klas modułu **Chats** pozwalającego na szereg funkcjonalności z zakresu komunikacji oraz zarządzania chatami i ich uczestnikami.

Rysunek 3.3: Diagram klas modułu **Chats**

## 3.2.5. Obsługa zapytań API

Obsługa zapytań API implementowana jest z wykorzystaniem wzorca projektowego mediator, realizowanego za pomocą biblioteki *MediatR*[22]. Po trafieniu zapytania na kontroler, w zależności od rodzaju zapytania tworzy się obiekt *Query* albo *Command*, przekazywany dalej poprzez warstwę pośredniczącą do odpowiedniego *Handler'a*. Poniżej na rysunku 3.4 jest przedstawiony przykładowy diagram działania mediatora dla zapytań **GetTeamById** oraz **CreateTeam**. Również pokazane są przepływy danych pomiędzy bazą a serwerem w ramach zapytań.



Rysunek 3.4: Diagram obsługi zapytań API

### 3.2.6. Czysta architektura

Czysta architektura jest filozofią projektowania oprogramowania, główną ideą której jest podział systemu na oddzielne warstwy. Ważnym celem czystej architektury jest zapewnienie programistom sposobu organizowania kodu w taki sposób, aby zawierał logikę biznesową, ale oddzielał ją od mechanizmu dostarczania. Główną zasadą czystej architektury jest to, że kod na warstwach wewnętrznych nie może mieć żadnej wiedzy o kodzie na warstwach zewnętrznych. Zmienne, funkcje i klasy (dowolne byty), które istnieją w warstwach zewnętrznych, nie mogą być wymienione na poziomach bardziej wewnętrznych. Czysta architektura została stworzona przez Roberta Martina, opisana w książce *"Clean architecture"*[15] oraz promowana w jego blogu *Uncle Bob*.

Podobnie jak inne filozofie projektowania oprogramowania, czysta architektura stara się zapewnić opłacalną metodologię, która ułatwia tworzenie wysokiej jakości kodu, który będzie łatwiejszy do zmiany i będzie mieć mniej zależności. Wizualnie poziomy czystej architektury są zorganizowane w nieokreśloną liczbę warstw.

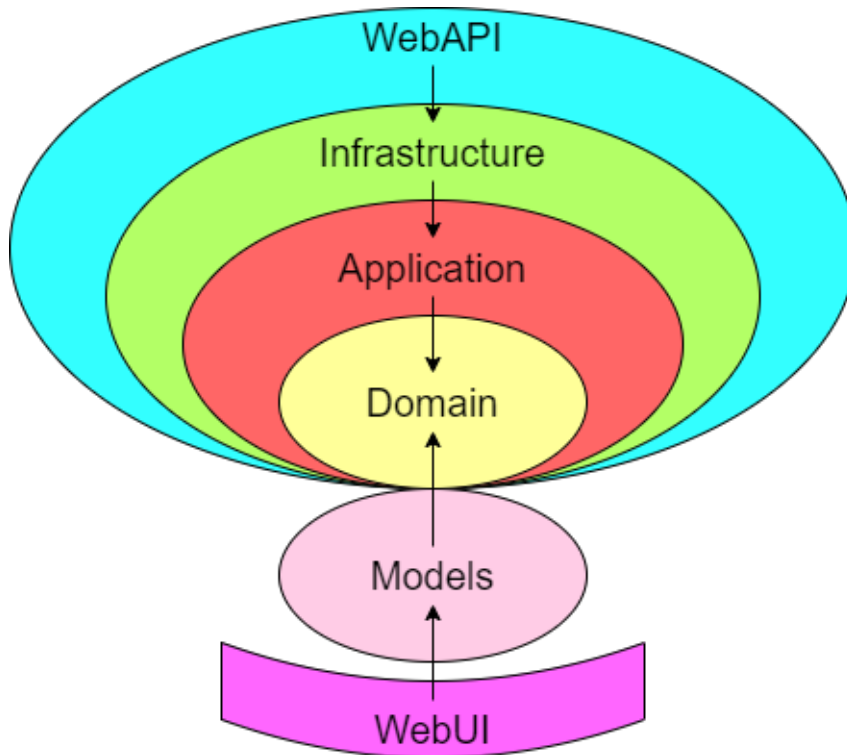
Żeby rozpatrzyć realizację czystej architektury w naszym systemie należy najpierw spojrzeć na strukturę rozwiązania. Wyodrębnione zostały 6 warstw:

- **Domain** – warstwa zawierająca definicje encji domenowych oraz logikę biznesową, im towarzyszącą. Ta warstwa nie jest zależna od pozostałych, stanowi jądro systemu. Nazwa warstwy została wybrana ze względu na paradygmaty *DDD*, ale w sumie nie możemy rozważać to podejście zbyt blisko w naszej aplikacji, ponieważ brakuje nam procesów i założeń biznesowych opisujących potrzeby biznesu, wszystkie założenia zostały umodelowane przez zespół programistów, zatem traci się sens użycia *DDD*, który polega na ujednoliceniu języku biznesu i development'u.
- **Application** – warstwa obsługi przepływów procesów biznesowych, zawierająca handlersy zapytań API oraz kontrakty serwisów pracujących z bazą oraz tożsamością użytkownika. Tutaj znajduje się logika dotycząca obsługi operacji zdefiniowanych przez biznes.
- **Infrastructure** – warstwa zawierająca konfiguracje dostępu do bazy danych oraz implementacje serwisów, kontrakty na które zostały zdefiniowane w warstwie **Application**. Są to serwisy "zewnętrzne", nie implementujące logiki biznesowej.
- **Models** – warstwa definiująca kontrakty pomiędzy aplikacją kliencką a serwerem oraz *DTO-objekty*. Ta warstwa jest uzależniona od warstwy **Domain** w celu zastosowania *Auto-Mappera*[11]. Dzięki tej warstwie mamy spójność zapytań wysyłanych z aplikacji klienckiej do API i ich obsługi po stronie serwera.

### 3.2. OPIS ARCHITEKTURY SERWERA

- **WebAPI** – warstwa zawierająca kod kontrolerów z definicjami punktów końcowych API.
- **WebUI** – warstwa aplikacji klienckiej, zawiera jej kod.

Na rysunku 3.5 poniżej widać zależności pomiędzy warstwami.

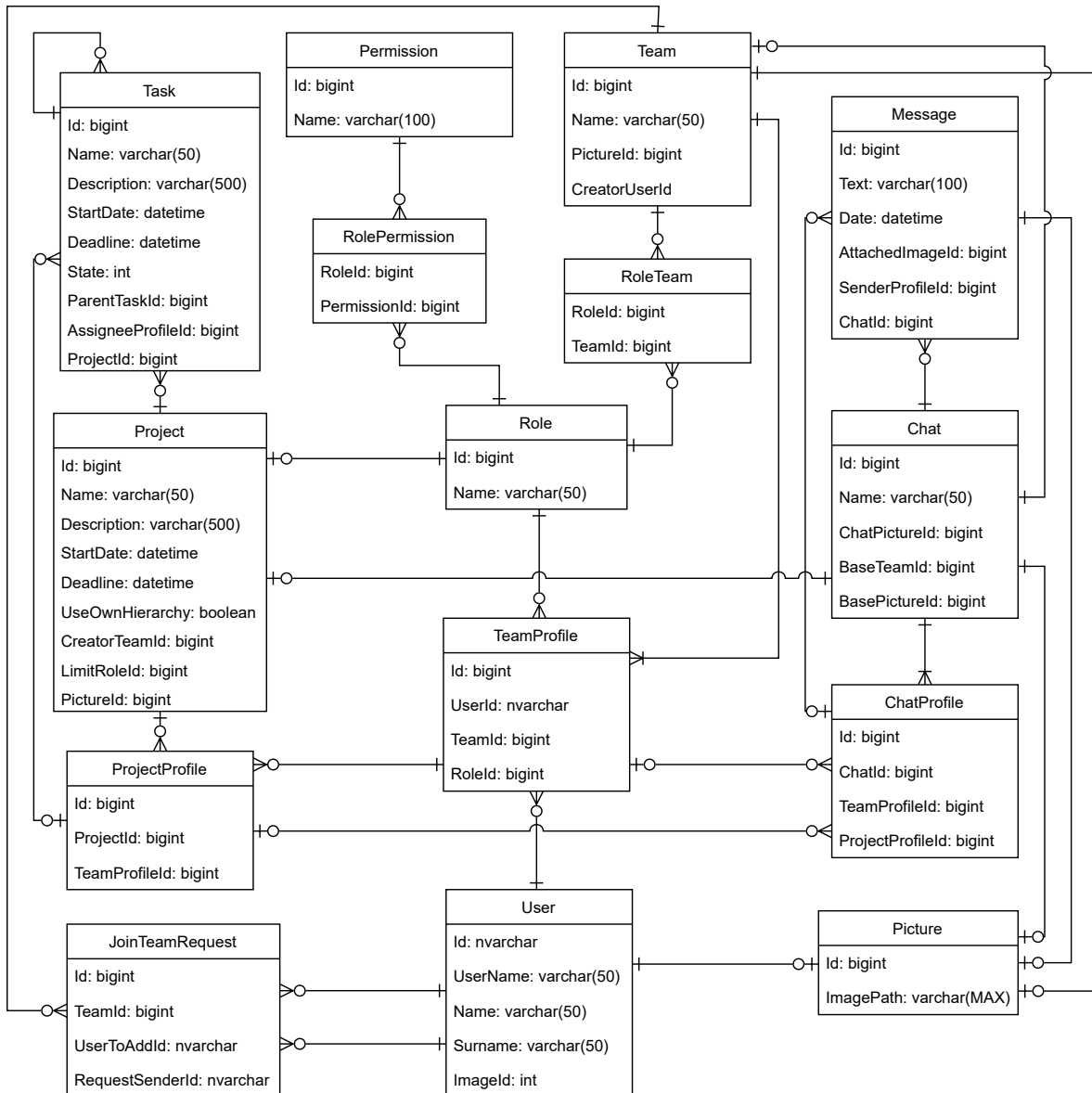


Rysunek 3.5: Podział systemu na warstwy

Stosowanie *AutoMapper* doprowadza nas do potrzeby uzależnienia warstwy **Models** od warstwy **Domain**, co nie jest dobrą praktyką, ponieważ chcielibyśmy jak najbardziej uniezależnić warstwę kontraktów. Również taka zależność doprowadza nas do ryzyko niepożądanego użycia encji domenowych w *DTO-objektach* podczas dalszego rozwoju systemu. Żeby to rozwiązać, należałoby przenieść mapowania do warstwy **Domain**. Ale to uzależniłoby warstwę **Domain** od **Models**, lecz tego namierzamy uniknąć. Lepszym rozwiązaniem byłoby wyodrębnienie oddzielnej warstwy mapperów, wtedy warstwy **Domain** i **Models** byłyby zupełnie uniezależnione. Taki refactor stanowiłby obowiązkowy wymóg podczas dalszego rozwoju systemu.

### 3.3. Opis bazy danych

Poniżej na rysunku 3.6 jest przedstawiony schemat bazy danych. Wykorzystujemy relacyjną bazę danych *Microsoft SQL Server*. Jest realizowana przy pomocy *Entity Framework Core* - technologii mapowania relacyjnego, która pozwala utworzyć bazę danych na podstawie stworzonych klas, które zostały opisane w rozdziale 3.2. Takie podejście w dokumentacji Microsoft nazywa się *Code First*.



Rysunek 3.6: Diagram bazy danych

### 3.3. OPIS BAZY DANYCH

Struktura bazy danych jest dosyć oczywista co do części odpowiadających modułom biznesowym aplikacji (**Teams**, **Project Manager**, **Chats**), natomiast należy zwrócić uwagę na tabele **TeamProfile**, **ProjectProfile** oraz **ChatProfile**, będące reprezentacją użytkownika w instancjach zespołu, projektu bądź czatu. Jest to de-facto struktura tabel reprezentujących dosyć skomplikowaną relację wiele-do-wielu pomiędzy encjami aplikacji a użytkownikiem.

Również należy wspomnieć, jak wygląda konfiguracja bazy ze strony serwera. W warstwie **Infrastructure** na tą potrzebę została wydzielona podwarstwa **Persistence**. Zawiera implementację kontekstu danych oraz klasy konfiguracyjne, opisujące tabele bazy danych i relacje pomiędzy tabelami.

### 3.4. Opis aplikacji klienckiej

Aplikacji klienta jest podzielona na 2 logiczne części:

1. **Services** - serwisy odpowiadające głównie za komunikację z serwerem (z wyjątkiem jednego). Zawiera następujące serwisy:
  - **LocalStorageService** - serwis odpowiadający za zapisywanie tokenu uwierzytelniającego w pamięci przeglądarki, co pozwala nie tylko nie wprowadzać danych do logowania podczas każdego zapytania do serwera, ale też na zamykanie okna przeglądarki bez potrzeby zalogowania po ponownym otwarciu. Zostały tutaj wykorzystane odpowiednie funkcje *JavaScript*, które są wywoływane z poziomu *C#*[6].
  - **HttpService** - serwis odpowiadający za stworzenie zapytania do serwera z odpowiednią zawartością, wysłanie i obsługę odpowiedzi. W przypadku gdy jest wysyłany obrazek, dane zostają przekonwertowane do formatu *form-data*, co jest wymogiem serwera. Poza tym zawiera w sobie adres serwera, który jest używany podczas ładowania obrazków.
  - **UserService** - serwis służący do zalogowania, wylogowania i rejestracji użytkownika. Podczas tych działań używa **LocalStorageService** do zapisu, odczytu lub usunięcia tokenu. Ponadto ma w sobie obiekt zalogowanego użytkownika, który jest często używany.
  - **TeamService**, **ProjectService**, **ChatService**, **PermissionService** - serwisy służące do komunikacji z endpoint'ami API odpowiednich modułów. Nie zawierają żadnej logiki oprócz odpowiednich wywołań funkcji **HttpService**.
2. **Pages** - makiety stron, które są wypełniane danymi otrzymanymi od serwera i pokazywane użytkownikowi. Używane są tutaj **Helpers** - klasy pomocnicze, które eliminują niektóre przypadki duplikowania kodu i dodają użyteczne funkcje.

## 4. Technologie i wdrożenie

### 4.1. Użyte technologie

Wszystkie wykorzystane technologie już zostały wspomniane wcześniej podczas opisu architektury systemu, ale żeby usystematyzować wymienimy je jeszcze raz:

- *ASP.NET Core*[7] - framework do tworzenia aplikacji internetowych stworzony przez korporację Microsoft
- *Blazor*[4] - technologia do tworzenia aplikacji webowych w języku *C#*
- *Blazorise*[8] - biblioteka udostępniająca wiele komponentów i narzędzi do stylowania aplikacji napisanych w technologii *Blazor*
- *Blazor ApexCharts*[9] - wrapper do biblioteki *Apex Charts*, która umożliwia tworzenie różnego rodzaju wykresów
- *Entity Framework Core*[10] - technologia mapowania relacyjnego (ang. ORM - object-relational mapping)
- *Microsoft SQL Server* - silnik relacyjnej bazy danych
- *MediatR* - biblioteka *C#* pozwalająca na łatwą implementację wzorca mediator
- *AutoMapper*[11] - biblioteka *C#* pozwalająca na łatwe przekopiowanie wartości pól obiektów obiektów
- *SignalR*[13] - usługa pozwalająca na ustalenie komunikacji "w czasie rzeczywistym"
- *Azure* - portal oferujący usługi chmurowe, został użyty do wdrożenia aplikacji[20]
- *GitHub Actions* - narzędzie do automatyzacji przepływów tworzenia oprogramowania z poziomu repozytorium GitHub, został użyty do wdrożenia
- *Git* - system kontrolowania wersji oprogramowania



Większość technologii zostały wybrane z powodu dobrej ich znajomości przez autorów tej pracy, natomiast chcielibyśmy bardziej wyjaśnić dobór technologii aplikacji klienckiej oraz opowiedzieć o usłudze użytej do realizacji komunikacji "w czasie rzeczywistym".

#### 4.1.1. Blazor

Jest to technologia opracowana przez firmę Microsoft pozwalająca na tworzenie aplikacji webowych w języku C# z użyciem *HTML*. Największą dla nas zaletą tej technologii jest to, że łatwo można uzgodnić model komunikacji między aplikacją kliencką a serwerem, który również jest napisany w języku C#. Oprócz tego, nie zabrania korzystania z *JavaScript*, umożliwiając wywoływanie wcześniej zdefiniowanych funkcji.

Dla *Blazor* istnieje 2 modele hostowania aplikacji[12]:

- *Server* – to jest zwykły model, podobny do aplikacji napisanych na przykład w *PHP*. Aplikacja jest przechowywana na serwerze, a poszczególne strony są wysyłane do klienta na żądanie.
- *WebAssembly* - inny model hostowania, różniący się od powyższego tym, że po wejściu użytkownika na stronę cała skompilowana aplikacja jest pobierana na maszynę kliencką.

Różnicą między tymi trybami łatwo widać podczas interakcji użytkownika z interfejsem, na przykład kliknięcie w przycisk. W modelu *Server* to spowoduje wysłanie zapytania do serwera z informacją o tym zdarzeniu, które będzie przetworzone przez serwer i zwrócenie wyniku do klienta oraz ewentualne ponowne renderowanie poszczególnych komponentów, np. pola dla wpisywania danych. Jest to wygodne w przypadkach, kiedy maszyna klienta jest bardzo słaba, ale wymaga ciągłego połączenia z serwerem, ponieważ zniknięcie połączenia uniemożliwia jakąkolwiek interakcję.

W modelu *WebAssembly* nie ma potrzeby wysłać zapytanie do serwera po każdym zdarzeniu, bo cała aplikacja jest na maszynie klienckiej i wszelkie interakcje mogą być przetwarzane lokalnie. Jest to wygodne w przypadkach, kiedy mamy słabe albo niestabilne połączenie z serwerem, ponieważ w tym przypadku komunikacja z serwerem jest używana tylko do pobierania lub zapisu danych. Pozwala to na korzystanie z aplikacji nawet gdy stracimy połączenie z serwerem, przy założeniu, że nie będą wysyłane zapytania na pobranie/zapis danych do serwera. Z drugiej strony taki model hostowania spowalnia działanie aplikacji, bo nie każdy użytkownik ma komputer, który ma wystarczająco mocy obliczeniowej.

Żaden z tych modeli nie ma absolutnej przewagi nad drugim, więc wybraliśmy model *WebAssembly* ze względu na egzotyczność i możliwość uruchomienia całej aplikacji lokalnie.

### 4.1.2. SignalR

Jest to biblioteka *ASP.NET*, która pozwala na komunikację serwer-klient w czasie rzeczywistym. Powodem do wykorzystania tej biblioteki jest niemożliwość takiego sposobu komunikacji z wykorzystaniem protokołu *HTTP*. Usługa *SignalR* może służyć do implementacji dowolnej funkcjonalności, która potrzebuje wysłania wiadomości od serwera do klientów "w czasie rzeczywistym". Czat jest często używany jako przykład, ale ogólnie prawie każdą funkcję da się ulepszyć za pomocą *SignalR*. *SignalR* ma wbudowane następujące techniki transportowe[13]:

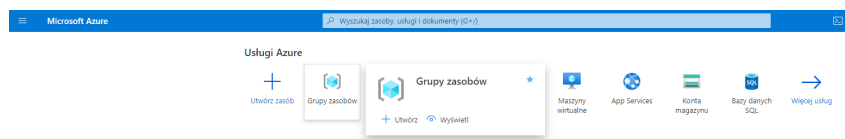
- *WebSockets* - wspierane przez niektóre przeglądarki
- *Server Sent Events* - wspiera notyfikację w stylu "push"
- *Long polling* - otwieramy połączenie i utrzymujemy je otwarte zmuszając klienta (przeglądarkę) do czekania udając, że dane połączenie rzeczywiście trwa długo

SignalR wybiera najlepszy typ transportu, które są wspierane zarówno przez klienta jak i przez serwera, chociaż można wymusić określony rodzaj transportu. W naszej aplikacji korzystamy z tak zwanych "grup", które są identyfikowane przez unikalne identyfikatory. To jest grupa użytkowników, którzy mają połączenie z serwerem. W przypadku czatów działa to następująco: po wejściu na stronę jest nawiązywane stałe połączenie z serwerem i wysyłane są kolejno zdarzenia o zapisaniu się do wszystkich "grup" odpowiadającym czatom, do których użytkownik ma dostęp. Podczas wysłania jednym z użytkowników wiadomości w chacie, rozsyłane jest zdarzenie do wszystkich użytkowników zapisanych do odpowiedniej grupy. To zdarzenie jest w odpowiedni sposób obsługiwane przez aplikację kliencką. W ten sposób jest realizowana komunikacja w czasie rzeczywistym.

## 4.2. Wdrożenie systemu

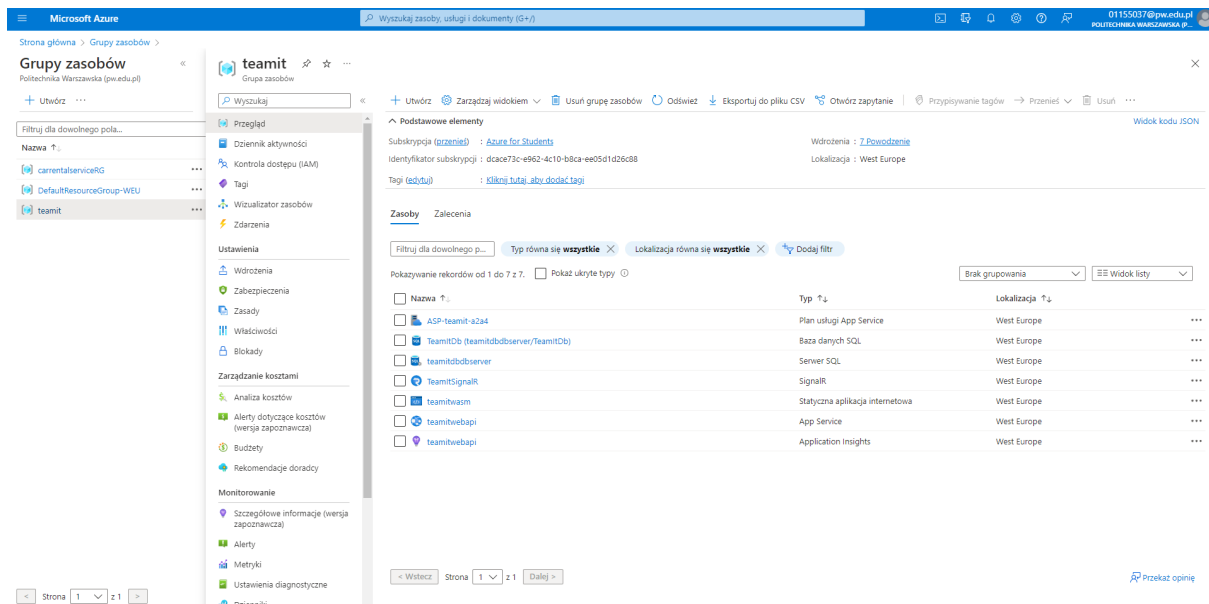
System został wdrożony przy użyciu narzędzi udostępnionych przez portal *Azure*[20]. Subskrypcja studencka udostępniona przez korporację Microsoft proponuje wszystkie niezbędne zasoby i narzędzia, a same te narzędzia są mocno zintegrowane ze środowiskiem programistycznym *Visual Studio* oraz repozytorium kodu *GitHub*, stosowanymi przez nas do tworzenia i przechowywania systemu.

W celu wdrożenia systemu musimy wdrożyć poszczególne jego moduły oraz podłączyć wymagane serwisy. Wszystkie te komponenty powinny należeć do jednej grupy zasobów, którą możemy łatwo utworzyć na stronie głównej *Azure*, jak na rysunku 4.1.



Rysunek 4.1: Grupy zasobów na stronie głównej

Wewnątrz utworzonej grupy będziemy dodawać potrzebne zasoby. Na rysunku 4.2 został podany widok grupy po dodaniu wszystkich niezbędnych komponentów, opisy wdrożenia poszczególnych zasobów są podane w kolejnych rozdziałach.



Rysunek 4.2: Grupa zasobów po dodaniu wszystkich komponentów

## 4.2. WDROŻENIE SYSTEMU

### 4.2.1. Serwer

W celu wdrożenia serwera należy utworzyć zasób aplikacji internetowej *App Service*, który znajdziemy na zakładce "Marketplace". W zakładce "Podstawowe" zaznaczamy grupę zasobów, typ środowiska uruchomieniowego, nazwę domeny oraz region, jak pokazano na rysunku 4.3.

Microsoft Azure

Strona główna > Marketplace > Aplikacja internetowa >

### Utwórz aplikację internetową

Podstawowe Wdrożenie Sieć Monitorowanie Tagi Przeglądanie + tworzenie

Funkcja App Service Web Apps umożliwia szybkie tworzenie, wdrażanie i skalowanie aplikacji internetowych, aplikacji mobilnych i aplikacji interfejsu API klasy korporacyjnej uruchamianych na dowolnej platformie. Spełniaj rygorystyczne wymagania dotyczące wydajności, skalowalności, zabezpieczeń i zgodności, używając w pełni zarządzanej platformy do konserwacji infrastruktury. [Dowiedz się więcej](#)

**Szczegóły projektu**

Wybierz subskrypcję, aby zarządzać wdrożonymi zasobami i kosztami. Użyj grup zasobów jak folderów, aby organizować wszystkie Twoje zasoby i zarządzać nimi.

Subskrypcja \* Azure for Students

Grupa zasobów \* teamit

[Utwórz nowy](#)

**Szczegóły wystąpienia**

Potrzebujesz bazy danych? [Wypróbuj nowe środowisko Internet i baza danych.](#)

Nazwa \* teamitwebapi

Nazwa aplikacji teamitwebapi jest niedostępna .azurewebsites.net

Publikuj \* ☒ Kod ☐ Kontener Docker ☐ Statyczna aplikacja internetowa

Stos środowiska uruchomieniowego \* .NET 7 (STS)

System operacyjny \* ☐ Linux ☒ Windows

Region \* West Europe

Nie możesz znaleźć planu usługi App Service? Spróbuj użyć innego regionu lub wybierz środowisko App Service Environment.

**Plany cenowe**

Od warstwy cenowej planu usługi App Service zależy lokalizacja, funkcje, koszty i zasoby obliczeniowe skojarzone z aplikacją. [Dowiedz się więcej](#)

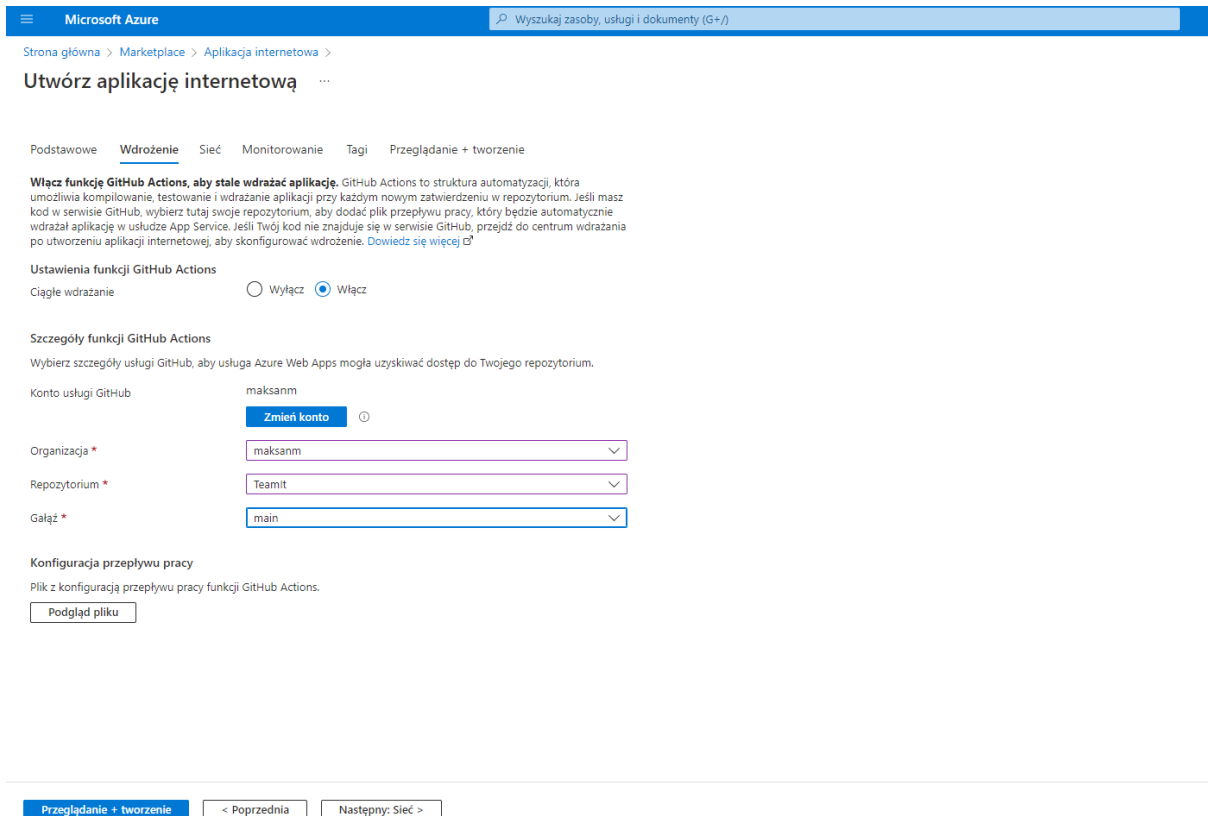
Plan systemu Windows (West Europe) \* ASP-teamit-a2a4 (F1)

[Utwórz nową](#)

[Przeglądanie + tworzenie](#) < Poprzednia Następný: Wdrożenie >

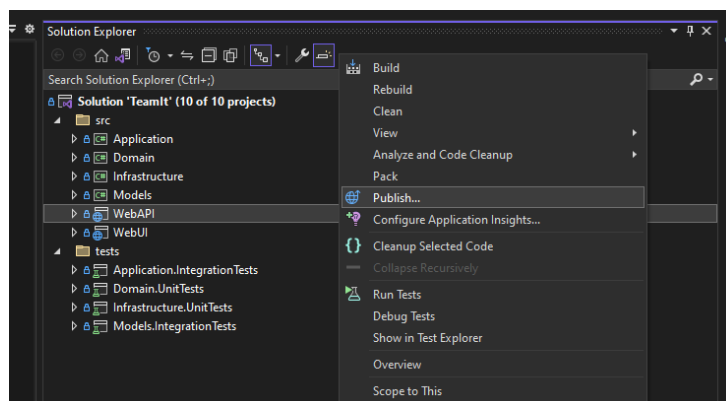
Rysunek 4.3: Podstawowa konfiguracja serwera

W celu umożliwienia pobrania przez portal *Azure* kodu przechowanego na *GitHub* oraz wygenerowania plików *.yml* definiujących zachowanie pipeline'ów puszczanych w narzędziu *GitHub Actions*, należy podłączyć odpowiednie konto *GitHub* zaznaczając wszystkie wymagane dostępy, oraz wybrać repozytorium zawierające kod systemu i gałąź, jak pokazano na rysunku 4.4. Resztę konfiguracji zostawiamy bez zmian. W repozytorium automatycznie wygeneruje się plik *.yml*, który będzie uruchamiać pipeline wdrażający serwer w narzędziu *GitHub Actions* za każdą zmianą kodu.



Rysunek 4.4: Konfiguracja wdrożenia systemu

Teraz musimy opublikować naszą aplikację do utworzonego zasobu. W tym celu w środowisku *Visual Studio* wybieramy menu "Publish" projektu **WebAPI**, jak pokazano na rysunku 4.5.

Rysunek 4.5: Menu "Publish" projektu **WebAPI**

Następnie wybieramy *Azure* i logujemy się do konta Microsoft, jeśli jest taka potrzeba. Szczegółowy opis wybrania utworzonego przez nas zasobu jest ominięty z powodu jego trywialności.

## 4.2. WDROŻENIE SYSTEMU

### 4.2.2. Aplikacja kliencka

W celu wdrożenia aplikacji klienckiej należy utworzyć zasób aplikacji statycznej *App Service Static Web App*, który tak samo jak wcześniej znajdziemy na zakładce "Marketplace". W zakładce "Podstawowe" tak samo jak dla serwera zaznaczamy grupę zasobów, nazwę aplikacji oraz region. Następnie należy wybrać odpowiedni repozytorium i gałąź oraz podać ściekę do aplikacji klienckiej, jak pokazano na rysunku 4.6.

Microsoft Azure

Strona główna > Marketplace > Statyczna aplikacja sieci Web >

### Utwórz statyczną aplikację sieci Web

Region dla interfejsu API usługi Azure Functions i środowisk przejściowych \*

West Europe

Szczegóły wdrożenia

Zróżło

☒ GitHub ☐ Azure DevOps ☐ Inne

Konto usługi GitHub

maksanm

Zmień konto ⓘ

❗ Jeśli nie możesz znaleźć organizacji lub repozytorium, może być konieczne włączenie dodatkowych uprawnień w serwisie GitHub.

Organizacja \*

maksanm

Repozytorium \*

Teamit

Rozgałęzienie \*

main

Szczegóły kompilacji

Wprowadź wartości, aby utworzyć plik przepływu pracy akcji usługi GitHub na potrzeby kompilowania i zwalniania. Możesz później zmodyfikować plik przepływu pracy w repozytorium GitHub.

Ustawienia wstępne kompilowania

Blazor

❗ Te pola będą odzwierciedlać domyślną strukturę projektu typu aplikacji. Zmień wartości tak, aby pasowały do aplikacji.

Lokalizacja aplikacji \* ⓘ

Teamit/src/WebUI ✓

Lokalizacja interfejsu API ⓘ

Api

Lokalizacja wyjściowa ⓘ

wwwroot

Podgląd pliku przepływu pracy

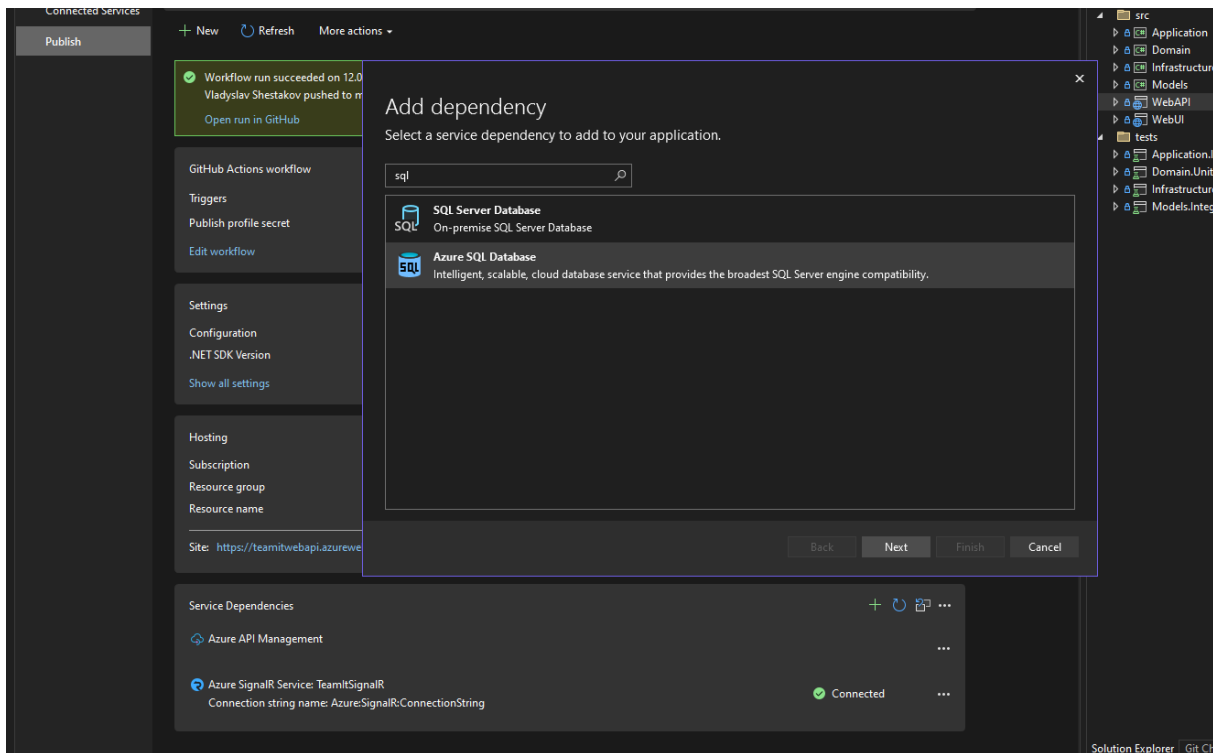
Przeglądanie + tworzenie < Poprzednia Następny: Tagi >

Rysunek 4.6: Podstawowa konfiguracja zasobu aplikacji statycznej

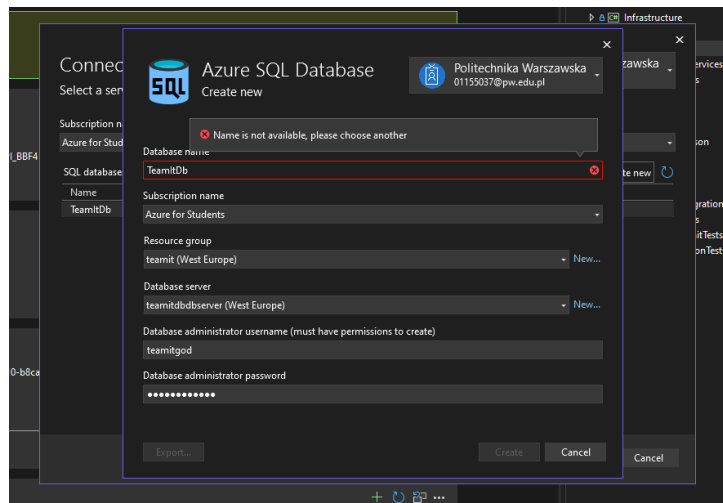
W repozytorium automatycznie wygeneruje się plik `.yml`, który będzie uruchamiać pipeline wdrażający aplikację kliencką w narzędziu *GitHub Actions* za każdą zmianą kodu.

### 4.2.3. Baza danych

W celu wdrożenia bazy danych należy utworzyć zasób bazy danych SQL. Ta baza powinna zostać umieszczona na serwerze SQL, więc musimy utworzyć odpowiedni zasób również dla tego serwera. Żeby to zrobić, w menu "Publish" projektu **WebAPI** w sekcji "Service Dependencies" dodajemy zależność do bazy danych *Azure Sql Database*, jak pokazano na rysunku 4.7.

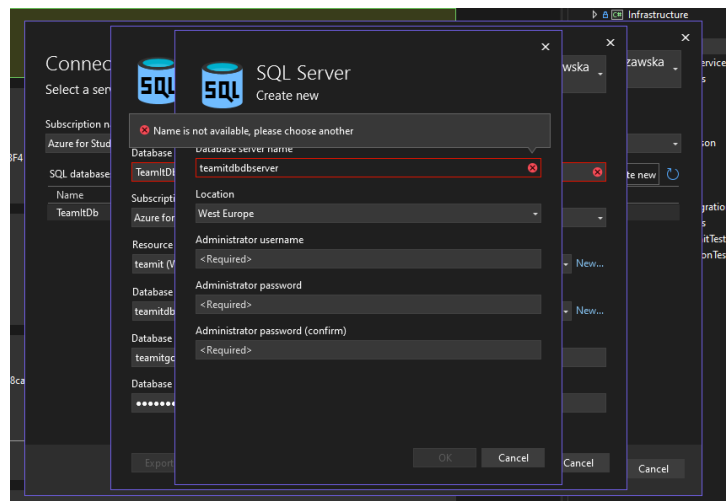
Rysunek 4.7: Menu wyboru bazy danych *Azure Sql Database*

Dalej należy utworzyć bazę danych i wprowadzić wszystkie wymagane dane, jak pokazano na rysunku 4.8.

Rysunek 4.8: Tworzenie bazy danych *SQL*

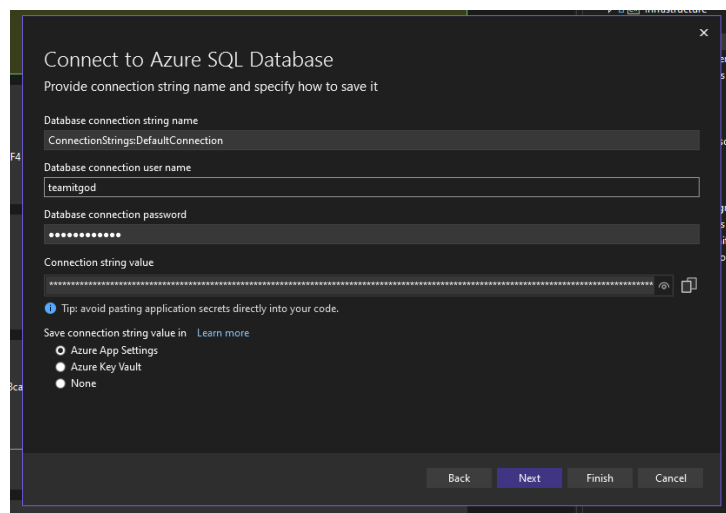
Żeby utworzyć serwer *SQL*, należy nacisnąć przycisk "New..." koło odpowiedniego pola i wprowadzić dane do konfiguracji serwera, jak pokazano na rysunku 4.9.

## 4.2. WDROŻENIE SYSTEMU



Rysunek 4.9: Tworzenie serwera *SQL*

Teraz zostaje nam skonfigurować połączenie do utworzonej bazy, jak pokazano na rysunku 4.10.



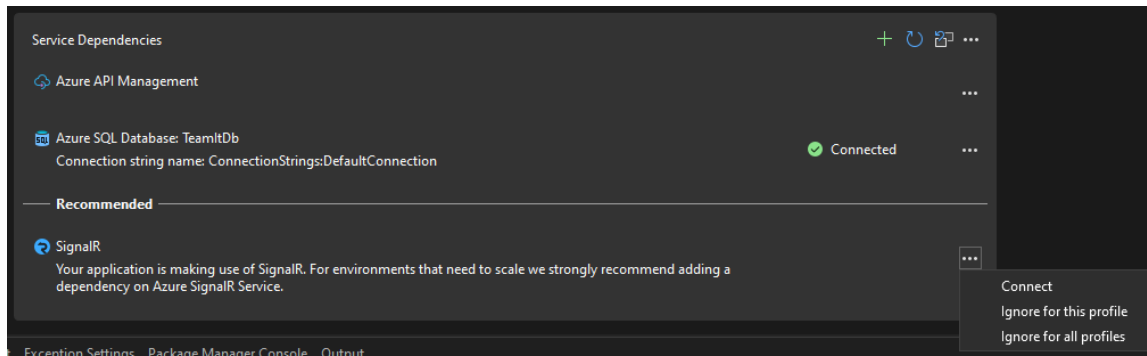
Rysunek 4.10: Konfiguracja połączenia do bazy

### 4.2.4. SignalR

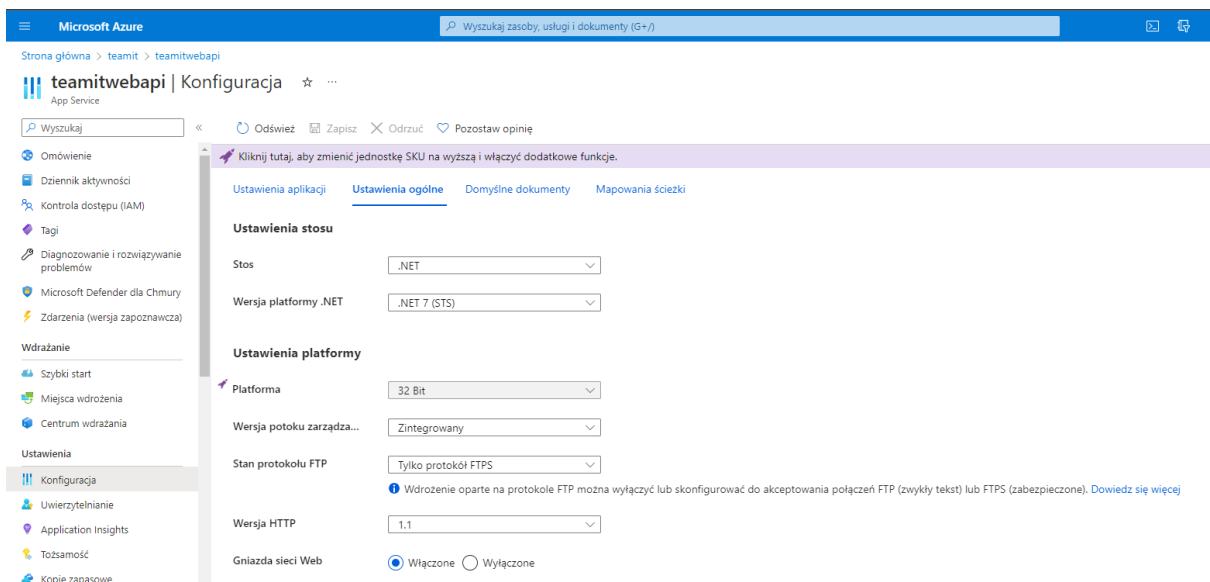
Żeby wdrożona aplikacja wspierała komunikację w czasie rzeczywistym musimy umożliwić wykorzystanie usługi *SignalR* przez serwer. W tym celu należy podjąć kilka działań:

1. W Visual Studio w menu "Publish" w sekcji "Service Dependencies" po podłączeniu usługi pojawi się pole *Azure SignalR Service*. Należy ją podłączyć, po skonfigurowaniu służba zostanie widoczna na portale *Azure*, jak pokazano na rysunku 4.11.



Rysunek 4.11: Pole *SignalR* w menu "Publish" projektu **WebAPI**

2. W ustawieniach serwera (zasób *App Service*) trzeba włączyć obsługę protokołu *WebSocket*, jak pokazano na rysunku 4.12.

Rysunek 4.12: Menu włączenia gniazd sieci *Web*

### 4.3. Instrukcja użytkowania aplikacji

Ze względu na swoją objętość, instrukcja użytkowania aplikacji przedstawiona jest w Załączniku 1.

## 5. Analiza rozwiązania

Niniejszy rozdział przedstawia opis testów przeprowadzonych w celu udowodnienia poprawnej realizacji systemu. Niektóre testy zostały napisane w celu zapewnienia poprawnego działania komponentów oraz możliwości jego kontroli z czasem, inne powstały jako rozwiązanie niektórych problemów, zaistniałych podczas implementacji jako skutek wybranej architektury.

### 5.1. Automatyczne testy jednostkowe

Ze względu na to, że warstwa **Application** głównie odpowiada za integrację całego systemu i jest pokryta przez testy integracyjne, testy jednostkowe skupiają się na logice zawartej w warstwach **Domain** oraz **Infrastructure**. Najważniejszą rolą tych testów jest udowodnienie, że logika dotycząca uprawnień użytkowników i ról została zaimplementowana poprawnie.

#### 5.1.1. Testy jednostkowe warstwy Domain

Do testów jednostkowych logiki zawartej w warstwie **Domain** wykorzystano bibliotekę *NUnit*. Ta warstwa zawiera istotną logikę obliczenia uprawnień użytkowników w obrębie zespołu, projektu bądź chatu. W celu przetestowania tej logiki powstały testy właściwości **Role** klas profilowych modułów **Project Manager** oraz **Chats**, natomiast testy ww. właściwości klasy profilowej modułu **Teams** zostały ominięte z powodu jej trywialności. Przykład testu jednostkowego klasy **ProjectProfile** został pokazany na rysunku 5.1

```
[Test]
0 references
public void ShouldHasPermssionUsingOwnHierarchy_SetByTeamRoleAndLimitRole()
{
    _project.UseOwnHierarchy = true;
    _project.CreatorTeam = _otherCreatorTeam;
    var hasAddTeamPermission = _projectProfile
        .Role
        .Permissions
        .Any(p => p.Id == PermissionEnum.PM_ADD_TEAM);
    Assert.IsTrue(hasAddTeamPermission);
}
```

Rysunek 5.1: Przykład testu jednostkowego klasy **ProjectProfile**

### 5.1.2. Testy jednostkowe warstwy Infrastructure

Do testów jednostkowych warstwy **Infrastructure** wykorzystano biblioteki *NUnit* oraz *Moq*[17]. Biblioteka *Moq* pozwala na symulację działania wybranych komponentów systemu bez ich inicjalizacji. W szczególności możemy wyróżnić następujące testy:

- **PermissionValidatorTests** - testy serwisu sprawdzającego uprawnienia użytkowników w obrębie modułów. Jest używany przez warstwę **Application**, wykorzystuje wymienione wyżej właściwości **Role** klas profilowych. Wykorzystuje Mock-obiekt serwisu **IdentityService** do symulacji obecnie zalogowanego użytkownika. Przykład testu został pokazany na rysunku 5.2

```
[Test]
public void ShouldNotThrowException_UserIsTeamCreator()
{
    _identityServiceMock
        .Setup(service => service.GetCurrentUserTeamProfileAsync(It.IsAny<long>()))
        .ReturnsAsync(_creatorUserTeamProfile);
    Assert.DoesNotThrowAsync(() =>
        _permissionValidator
            .ValidateTeamPermission(0, PermissionEnum.TEAM_CHANGE_ROLE));
}
```

Rysunek 5.2: Przykład testu jednostkowego klasy **PermissionValidator**

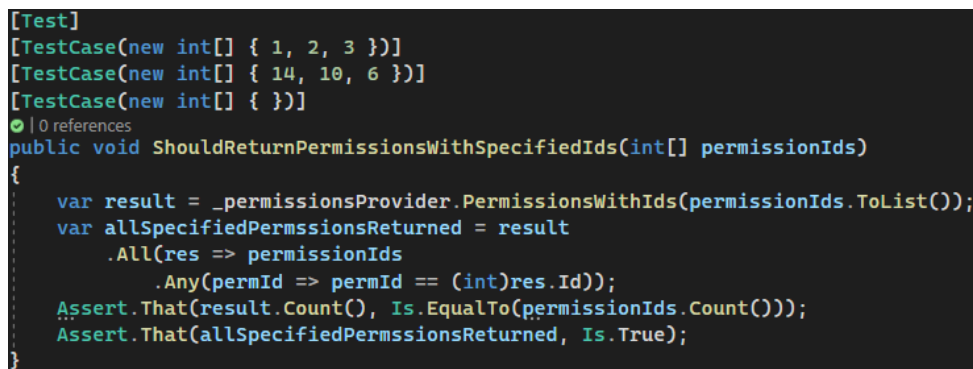
- **IdentityServiceTests** - testy serwisu odpowiadającego za logowanie, rejestrację oraz wszelkie interakcje z obecnie zalogowanym użytkownikiem. Zostały przetestowane metody zwracające profile bieżącego użytkownika w poszczególnych modułach. W tym celu został zmockowany sam **IdentityService**, żeby zasymulować zwracanie obecnie zalogowanego użytkownika. Nie tworzy to problemów, bo biblioteka *Moq* pod spodem dziedziczy po klasach mockowanych, więc testując Mock-obiekt testowanej klasy de-facto testujemy te same bloki kodu. Przykład testu został pokazany na rysunku 5.3

```
[Test]
public void ShouldReturnTeamProfile()
{
    var returnedProfile = _identityServiceMock
        .Object
        .GetCurrentUserTeamProfileAsync(_team.Id)
        .Result;
    Assert.That(returnedProfile.User.Id, Is.EqualTo(_currentUser.Id));
}
```

Rysunek 5.3: Przykład testu jednostkowego klasy **IdentityService**

## 5.2. AUTOMATYCZNE TESTY INTEGRACYJNE

- **PermissionsProviderTests** - testy serwisu zwracającego uprawnienia według ich **Id** bądź predefiniowane zbiory uprawnień (np. Team Creator). Wykorzystuje zmockowany kontekst bazy danych. Przykład testu został pokazany na rysunku 5.4



```
[Test]
[TestCase(new int[] { 1, 2, 3 })]
[TestCase(new int[] { 14, 10, 6 })]
[TestCase(new int[] { })]
public void ShouldReturnPermissionsWithSpecifiedIds(int[] permissionIds)
{
    var result = _permissionsProvider.PermissionsWithIds(permissionIds.ToList());
    var allSpecifiedPermissionsReturned = result
        .All(res => permissionIds
            .Any(permId => permId == (int)res.Id));
    Assert.That(result.Count(), Is.EqualTo(permissionIds.Count()));
    Assert.That(allSpecifiedPermissionsReturned, Is.True);
}
```

Rysunek 5.4: Przykład testu jednostkowego klasy **PermissionsProvider**

### 5.2. Automatyczne testy integracyjne

Do testów integracyjnych używana jest biblioteka *NUnit* oraz standardowe narzędzia udostępniane przez framework *ASP.NET Core*.

#### 5.2.1. Testy integracyjne warstwy Application

Tutaj są testowane endpoint'y API pod kątem poprawności zwracanego wyniku w zależności od kontekstu. Wszystkie testy zostały podzielone ze względu na logiczny moduł testowany - **Teams, Roles, Permissions, Projects, Tasks, Chats** oraz **Messages**. Najpierw tworzone są wszystkie niezbędne elementy systemu do przetestowania, czyli sam serwer, który będzie testowany, oraz HTTP klient do wysyłania zapytań. Baza danych jest automatycznie tworzona lokalnie za pomocą migracji podczas inicjalizacji serwera przed przetestowaniem każdego zapytania. Następnie w miarę poszczególnych testów są tworzone obiekty w bazie oraz wysyłane odpowiednie komendy bądź kwerendy. Są sprawdzane wszystkie możliwe odpowiedzi serwera oraz stan obiektów w bazie po wykonaniu zapytań. Przykład testowania endpoint'a "teams" został pokazany na rysunku 5.5

#### 5.2.2. Testy integracyjne warstwy Models

Podczas planowania architektury systemu staraliśmy się trzymać się zasad *DDD* (*ang. Domain Driven Development*)[21]. Jedną z kluczowych zasad jest wyodrębnienie logiki biznesowej oraz

```

[Test]
public async Task ShouldCreateTeam()
{
    var createTeamCommand = new CreateTeamCommand()
    {
        Name = "Test team"
    };

    var response = await _client.PostAsJsonAsync("/teams", createTeamCommand);

    var context = GetDbContext();
    Assert.IsTrue(response.IsSuccessStatusCode);
    Assert.That(context.Team.Count(), Is.EqualTo(1));
}

```

Rysunek 5.5: Przykład testu integracyjnego rozkazu **CreateTeamCommand**

encji biznesowych w oddzielnej warstwie (w naszym przypadku **Domain**). Ale posiadanie encji domenowych typu wyliczeniowego **PermissionEnum** oraz **TaskStateEnum**, wymaganych również w aplikacji klienckiej, zmusza nas do złamania ww. zasady i trzymania encji biznesowej poza domeną w warstwie kontraktów. Żeby tego uniknąć, możemy utworzyć kopie tych enum'ów dla obu warstw, co oczywiście doprowadza do złamania *DRY* (ang. *Don't Repeat Yourself*) oraz potencjalnych niespójności typów domenowego używanego przez API, oraz typu z modeli używanego przez warstwę **WebUI**. Żeby to wyrównać powstały testy integracyjne *DomainMatching-Tests*, które przy wykorzystaniu refleksji sprawdzają, czy typy wyliczeniowe sobie odpowiadają. Takim czynem rozwiązaliśmy problem nie złamawszy zasad *DDD*. Test został pokazany na rysunku 5.6

```

public void AssertEnumsEquality(Type domainEnumType, Type dtoEnumType)
{
    var domainNames = Enum.GetNames(domainEnumType);
    var dtoNames = Enum.GetNames(dtoEnumType);

    Assert.That(dtoNames.Length, Is.EqualTo(domainNames.Length),
        $"{typeof(PermissionEnumDto).Name} must have the same number of values as {typeof(PermissionEnum).Name}");
    for (int i = 0; i < domainNames.Length; i++)
        Assert.That(dtoNames[i], Is.EqualTo(domainNames[i]),
            $"{typeof(PermissionEnumDto).Name} value name should be same as corresponding {typeof(PermissionEnum).Name} value name: " +
            $"{dtoNames[i]} is not equal to {domainNames[i]}");
}

```

Rysunek 5.6: Metoda sprawdzająca równość typów wyliczeniowych

### 5.3. Automatyczne testy UI

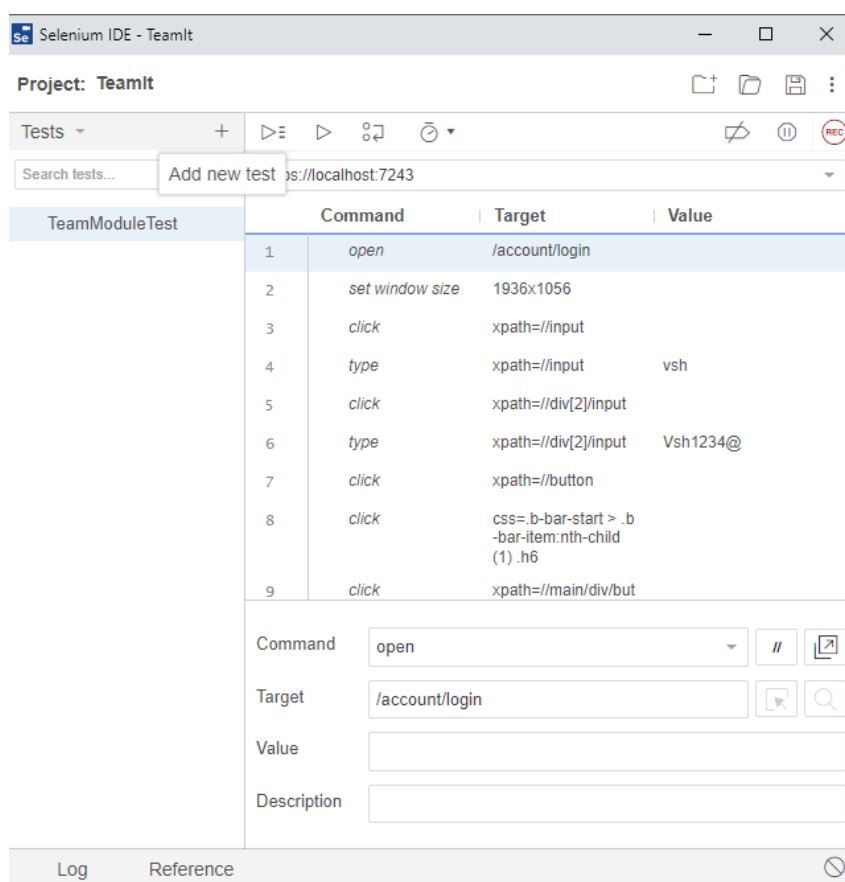
Do testów interfejsu użytkownika używana jest biblioteka *Selenium*, a dokładniej rozszerzenie *Selenium IDE*<sup>[14]</sup> dla *Google Chrome*. Pozwala ona na automatyczne przetestowanie aplikacji frontend'owych według wcześniej zdefiniowanych scenariuszy. Zaletą tej biblioteki jest to, że usuwa potrzebę ręcznego "klikania" w przyciski, wprowadzenia danych i sprawdzenia poprawności po wykonaniu jakiegokolwiek czynności. Dla każdego modułu (**Teams**, **Projects**, **Chats**) został stworzony osobny test, który sprawdza poprawność działania aplikacji ze strony końcowego użytkownika. Przykładowe działania ze scenariusza testowania modułu **Teams**:

Działanie	Oczekiwany rezultat
Stworzenie nowego zespołu	Powodzenie operacji, pojawienie się nowego zespołu na odpowiedniej liście, możliwość wejścia na jego stronę
Dodanie nowego użytkownika	Powodzenie operacji, pojawienie się użytkownika na odpowiedniej liście, możliwość zarządzania tym użytkownikiem w obrębie danego zespołu
Próba zmiany nazwy zespołu na pustą	Niepowodzenie operacji, wyświetlony odpowiedni komunikat, nazwa zespołu została niezmieniona
Wejście na stronę zespołu, w którym zalogowany użytkownik ma niepełne uprawnienia	Nie ma możliwości wykonania czynności, dla których nie mamy uprawnień

Tabela 5.1: Przetestowane akcje i oczekiwane wyniki

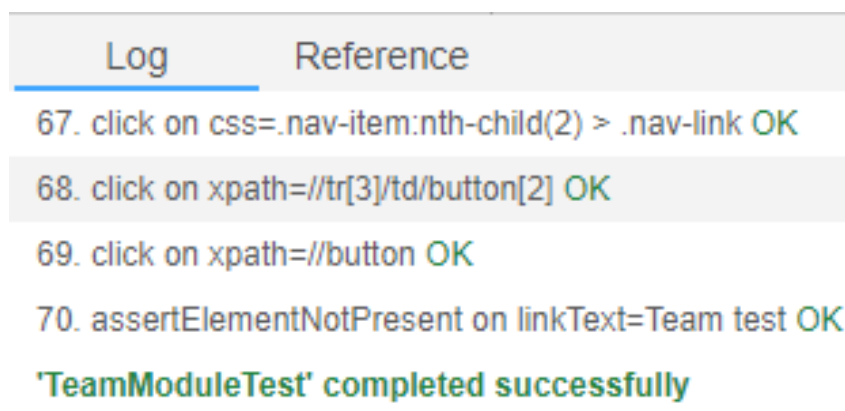
W podobny sposób testowane są wszystkie moduły aplikacji oraz proces logowania/rejestracji/wylogowania.

Tak wygląda przykładowy test dla modułu **Teams**:



Rysunek 5.7: Przykładowy test modułu **Teams**

Tak wygląda log po wykonaniu testu:



Rysunek 5.8: Przykładowy log po skończeniu testu

## 6. Podsumowanie

### 6.1. Produkt finalny

W ramach niniejszej pracy dyplomowej udało nam się stworzyć aplikację służącą do zarządzania pracą zespołową i międzyzespołową w projektach. Posiada elastyczny system ról pozwalający na precyzyjne wyspecyfikowanie czynności dostępnych dla poszczególnych użytkowników. Dodatkowo udostępnia komunikator zintegrowany z systemem eliminujący potrzebę używania zewnętrznych programów.

Z pomysłu system był skierowany na firmy posiadające współpracujące ze sobą jednostki. Za pomocą systemu ról moglibyśmy stworzyć hierarchię członków zespołu, odpowiadającą stanowiskom poszczególnych użytkowników w pewnej jednostce. Stworzona aplikacja nadaje się do takich przypadków, ponieważ posiadanie oddzielnych zespołów nie zmniejsza nam integracji procesu roboczego dzięki możliwości pracy międzyzespołowej nad projektami oraz chatów.

W trakcie projektowania systemu wykryło się również, że aplikacja teoretycznie jest podstawą do stworzenia bardziej zaawansowanego portalu społecznościowego, ponieważ już teraz posiada sporo funkcjonalności pozwalających na kooperację. To stanowi drugi potencjalny kierunek rozwoju, o co zostanie rozpowiedziane w następnym dziale.

### 6.2. Dalszy rozwój

Każdą aplikację da się ulepszyć, poprawić oraz wzbogacić o nowe funkcjonalności. W naszym przypadku w poprawie aplikacji następnymi krokami są planowane:

- Poprawa widoku aplikacji i ewentualna zmiana biblioteki do stylowania, żeby użycie było bardziej zrozumiałe i wygodne dla użytkowników
- Podobnie do komunikatora, zamienić ręczne odświeżanie danych w innych modułach na odświeżanie w czasie rzeczywistym
- Zdefiniowanie warstwy mapperów w celu uniezależnienia warstwy kontraktów w aplikacji



serwerowej

W ramach tej pracy został utworzony prototyp systemu skierowanego na integrację między-ludzką, zostały realizowane podstawowe funkcjonalności wspierające wspólną pracę i komunikację. To oznacza, że w ramach dłuższego rozwoju tego projektu pozostaje możliwość dalszego kroku w stronę sieci i portali społecznościowych, odpowiednio zaistniałej specyfice systemu. Na przykład, aplikacja może zostać wzbogacona o:

- System "thread'ów", bądź forumów, z możliwością zostawiać posty, komentarze pod nimi, głosować. To pozwoliłoby na inny rodzaj dyskusji, niż pozwala chat.
- Własną tablicę z postami dla zespołów. Uprawnieni użytkownicy zostawialiby posty w imieniu zespołu z możliwością głosowania i komentowania.
- Bardziej rozbudowany komunikator. Można kroczyć w stronę tworzenia kanałów, podobnie, jak to jest zrobione w *Telegram*.
- Strona główna zbierająca informacje ze wszystkich zespołów, czatów, forumów itp.

Mimo tego, skoro aplikacja z zasady jest menedżerem projektu, pozostaje też możliwość ruszenia w stronę bardziej zaawansowanych i rozbudowanych narzędzi do zarządzania pracą. Można to zrobić, na przykład, dodawszy więcej narzędzi do wizualizacji pracy, rozbudowawszy opisy zadań albo wdrożywszy wsparcie metodologii tworzenia oprogramowania, typu *SCRUM*.

## Bibliografia

- [1] Asana Inc., *Asana*, <https://asana.com/>, 2023 (dostęp 27-01-2023)
- [2] Atlassian, *Trello*, <https://trello.com/>, 2023 (dostęp 27-01-2023)
- [3] Atlassian, *Jira*, <https://www.atlassian.com/software/jira>, 2023 (dostęp 27-01-2023)
- [4] Microsoft, *Blazor*, <https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>, 2023 (dostęp 27-01-2023)
- [5] Google, *Flutter*, <https://flutter.dev/>, 2023 (dostęp 27-01-2023)
- [6] Microsoft, *Call JavaScript functions from .NET methods in ASP.NET Core Blazor*, <https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023)
- [7] Microsoft, *ASP.NET documentation*, <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023)
- [8] Megabit, *Blazorise Documentation*, <https://blazorise.com/docs>, 2023 (dostęp 27-01-2023)
- [9] Joakim Dangården, *Blazor-ApexCharts*, <https://github.com/apexcharts/Blazor-ApexCharts>, 2022 (dostęp 27-01-2023)
- [10] Microsoft, *Entity Framework Core*, <https://learn.microsoft.com/en-us/ef/core/>, 2021 (dostęp 27-01-2023)
- [11] Jimmy Bogard, *AutoMapper*, <https://docs.automapper.org/en/stable/>, 2017 (dostęp 27-01-2023)
- [12] Microsoft, *ASP.NET Core Blazor hosting models*, <https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023)
- [13] Microsoft, *Overview of ASP.NET Core SignalR*, <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023).

- [14] Software Freedom Conservancy (SFC), *Selenium IDE*, <https://www.selenium.dev/selenium-ide/>, 2019 (dostęp 27-01-2023).
- [15] Martin Robert, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, September 2021.
- [16] Microsoft, *ASP.NET Core Blazor*, <https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023).
- [17] Rehmann, *Moq documentation*, <https://documentation.help/Moq/> (dostęp 27-01-2023).
- [18] Microsoft, *CQRS pattern*, <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs> (dostęp 27-01-2023).
- [19] Microsoft, *RESTful web API design*, <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>, 2022 (dostęp 27-01-2023).
- [20] Microsoft, *Host and deploy ASP.NET Core Blazor WebAssembly*, <https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-7.0>, 2023 (dostęp 27-01-2023).
- [21] Laurent Grima, *An introduction to Domain-Driven Design*, <https://medium.com/inato/an-introduction-to-domain-driven-design-386754392465>, 2018 (dostęp 27-01-2023).
- [22] Microsoft, *Implement application layer using the Web API*, <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-application-layer-implementation-web-api>, 2022 (dostęp 27-01-2023).

## Wykaz symboli i skrótów

itp. i tym podobne

cd. ciąg dalszy

ang. z języka angielskiego

DTO ang. Data Transfer Object

DDD ang. Domain-Driven Design

API ang. Application Programming Interface

## Spis rysunków

1.1	Model potencjalnej firmy-konsumenta aplikacji . . . . .	14
2.1	Diagram przypadków użycia aplikacji . . . . .	17
3.1	Diagram klas modułu <b>Teams</b> . . . . .	40
3.2	Diagram klas modułu <b>Project Manager</b> . . . . .	41
3.3	Diagram klas modułu <b>Chats</b> . . . . .	42
3.4	Diagram obsługi zapytań API . . . . .	43
3.5	Podział systemu na warstwy . . . . .	45
3.6	Diagram bazy danych . . . . .	46
4.1	Grupy zasobów na stronie głównej . . . . .	52
4.2	Grupa zasobów po dodaniu wszystkich komponentów . . . . .	52
4.3	Podstawowa konfiguracja serwera . . . . .	53
4.4	Konfiguracja wdrożenia systemu . . . . .	54
4.5	Menu "Publish" projektu <b>WebAPI</b> . . . . .	54
4.6	Podstawowa konfiguracja zasobu aplikacji statycznej . . . . .	55
4.7	Menu wyboru bazy danych <i>Azure Sql Database</i> . . . . .	56
4.8	Tworzenie bazy danych <i>SQL</i> . . . . .	56
4.9	Tworzenie serwera <i>SQL</i> . . . . .	57
4.10	Konfiguracja połączenia do bazy . . . . .	57
4.11	Pole <i>SignalR</i> w menu "Publish" projektu <b>WebAPI</b> . . . . .	58
4.12	Menu włączenia gniazd sieci <i>Web</i> . . . . .	58
5.1	Przykład testu jednostkowego klasy <b>ProjectProfile</b> . . . . .	59
5.2	Przykład testu jednostkowego klasy <b>PermissionValidator</b> . . . . .	60
5.3	Przykład testu jednostkowego klasy <b>IdentityService</b> . . . . .	60
5.4	Przykład testu jednostkowego klasy <b>PermissionsProvider</b> . . . . .	61
5.5	Przykład testu integracyjnego rozkazu <b>CreateTeamCommand</b> . . . . .	62

5.6	Metoda sprawdzająca równość typów wyliczeniowych . . . . .	62
5.7	Przykładowy test modułu <b>Teams</b> . . . . .	64
5.8	Przykładowy log po skończeniu testu . . . . .	64

## Spis tabel

2.1	Historie użytkownika modułu <b>Teams</b> . . . . .	18
2.2	Historie użytkownika modułu <b>Project Manager</b> . . . . .	19
2.3	Historie użytkownika modułu <b>Project Manager</b> (cd.) . . . . .	20
2.4	Historie użytkownika modułu <b>Chats</b> . . . . .	20
2.5	Opis tworzenia zespołu . . . . .	21
2.6	Opis akceptacji zaproszenia . . . . .	21
2.7	Opis odrzucenia zaproszenia . . . . .	22
2.8	Opis usuwania zespołu . . . . .	22
2.9	Opis wychodzenia z zespołu . . . . .	22
2.10	Opis zarządzania uczestnikami zespołu . . . . .	23
2.11	Opis przypisania roli do użytkownika . . . . .	23
2.12	Opis usuwania użytkownika z zespołu . . . . .	23
2.13	Opis tworzenia nowej roli . . . . .	24
2.14	Opis zmiany ustawień roli . . . . .	24
2.15	Opis usuwania roli . . . . .	24
2.16	Opis zarządzania ustawieniami zespołu . . . . .	25
2.17	Opis tworzenia projektu . . . . .	26
2.18	Opis wyjścia z projektu . . . . .	26
2.19	Opis usunięcia projektu . . . . .	27
2.20	Opis zarządzania ustawieniami projektu . . . . .	27
2.21	Opis dodania zespołu do projektu . . . . .	28
2.22	Opis usunięcia zespołu z projektu . . . . .	28
2.23	Opis wyjścia zespołu z projektu . . . . .	28
2.24	Opis tworzenia zadania . . . . .	29
2.25	Opis zmiany ustawień zadania . . . . .	29
2.26	Opis zmiany statusu zadania . . . . .	30
2.27	Opis usunięcia zadania . . . . .	30

2.28	Opis tworzenia podzadania . . . . .	30
2.29	Opis zmiany ustawień podzadania . . . . .	31
2.30	Opis wyjścia z projektu . . . . .	31
2.31	Opis usunięcia podzadania . . . . .	31
2.32	Opis tworzenia czatu . . . . .	32
2.33	Opis wyjścia z czatu . . . . .	32
2.34	Opis zmiany obrazu i nazwy czatu . . . . .	33
2.35	Opis dodania użytkownika do czatu . . . . .	33
2.36	Opis usunięcia użytkownika z czatu . . . . .	33
2.37	Opis wyjścia z czatu . . . . .	34
2.38	Opis wysłania wiadomości . . . . .	34
5.1	Przetestowane akcje i oczekiwane wyniki . . . . .	63



## Spis załączników

1. Załącznik 1 - instrukcja użytkowania aplikacji