**PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA  
W NOWYM SĄCZU**

**INSTYTUT TECHNICZNY**

**PRACA DYPLOMOWA**

**SYSTEM WSPOMAGANIA ZARZĄDZANIEM ORGANIZACJĄ PROOBRONNĄ**

**Autor: Karol Chryczyk  
Kierunek: Informatyka stosowana  
Nr albumu: 27349**

**Promotor: dr hab. Witold Przygoda**

**NOWY SĄCZ 2021**

**Spis treści**

[1. Wstęp 4](#_Toc63168925)

[1.1. Wprowadzenie 4](#_Toc63168926)

[1.2. Geneza 6](#_Toc63168927)

[2. Cel i zakres pracy 7](#_Toc63168928)

[3. Technologie 8](#_Toc63168929)

[3.1. .NET 8](#_Toc63168933)

[3.2. C# 8](#_Toc63168934)

[3.3. ASP .NET Core 8](#_Toc63168935)

[3.4. WPF 9](#_Toc63168936)

[3.5. Xamarin 9](#_Toc63168937)

[3.6. TypeScript 9](#_Toc63168938)

[3.7. Angular 9](#_Toc63168939)

[4. Projekt aplikacji 10](#_Toc63168940)

[4.1. Wymagania funkcjonalne 10](#_Toc63168942)

[4.2. Wymagania niefunkcjonalne 10](#_Toc63168943)

[4.3. Komunikacja komponentów projektu 10](#_Toc63168944)

[4.4. Rest API 11](#_Toc63168945)

[4.4.1. Baza Danych 14](#_Toc63168946)

[4.4.2. Struktura projektu REST API 16](#_Toc63168947)

[4.4.3. Dostęp do zasobów 16](#_Toc63168948)

[4.5. Aplikacja WEB 17](#_Toc63168949)

[4.5.1. Diagram przypadków użycia dla użytkownika 18](#_Toc63168950)

[4.5.2. Diagram przypadków użycia dla administratora 19](#_Toc63168951)

[4.5.3. Dostęp do panelu administracyjnego i obsługa błędów 20](#_Toc63168952)

[4.6. Aplikacja desktopowa 20](#_Toc63168953)

[4.6.1. Model MVVM 21](#_Toc63168954)

[4.6.2. Diagram przypadków użycia 22](#_Toc63168955)

[4.6.3. Struktura aplikacji 22](#_Toc63168956)

[4.6.4. Edytor taktyczny i budowa znaków 23](#_Toc63168957)

[4.6.5. Obsługa edytora taktycznego 26](#_Toc63168958)

[4.6.6. Zasada działania edytora 28](#_Toc63168959)

[4.7. Aplikacja mobilna 30](#_Toc63168960)

[4.7.1. Diagram przypadków użycia aplikacji mobilnej 31](#_Toc63168961)

[4.7.2. Struktura aplikacji mobilnej 31](#_Toc63168962)

[4.7.3. Wiązanie strony z modelem widoku 32](#_Toc63168963)

[5. Podsumowanie 35](#_Toc63168964)

[5.1. Podsumowanie pracy 35](#_Toc63168966)

[5.2. Plany rozwoju systemu 35](#_Toc63168967)

[Spis rysunków 36](#_Toc63168968)

[Spis tabel 37](#_Toc63168969)

[Bibliografia 38](#_Toc63168970)

## Wstęp

### Wprowadzenie

Organizacje proobronne to stowarzyszenia i związki, które skupiają się na tematyce wychowania patriotycznego, oraz tzw. „przysposobienia wojskowego” (Paszyn, Kordowski i Zalewski, 2016). Duża część jednostek paramilitarnych wywodzi się ze Strzelca założonego w 1910r. przez marszałka Józefa Piłsudzkiego. Strzelec gromadził pół miliona członków, a do jego głównych zadań należało szkolenie społeczeństwa z zakresu wojskowości i patriotyczne wychowanie w dopiero co odzyskanym kraju. Wiele organizacji do dzisiaj kontynuuje tradycje zapoczątkowane przez marszałka Piłsudskiego.

Organizacje proobronne stanowią ważny element polskiego systemu obronności. Członkowie organizacji mogą stanowić zaplecze Wojsk Obrony Terytorialnej, oraz innych służb mundurowych. Stowarzyszenia proobronnę są również wykorzystywane w zarządzaniu kryzysowym do pomocy w likwidacjach skutków klęsk żywiołowych. Wiele takich organizacji można spotkać na obchodach uroczystości państwowych, gdzie często wystawiane są pododdziały honorowe oraz na różnego rodzaju akcjach charytatywnych, w które organizacje często się włączają.

Głównym celem członków organizacji proobronnej jest szkolenie w zakresie wojskowości. Zazwyczaj pierwszym jego etapem jest podobnie jak w WP szkolenie unitarne, które zawiera podstawową tematykę z różnych dziedzin wojskowości. Następnie odbywają się szkolenia z różnych tematów od podstawowych jak BLOS, taktykę czarną, zieloną, szkolenia strzeleckie i ratownictwa, po specjalistyczne kursy jak np. kurs spadochroniarzy, płetwonurków, czy ratowników pola walki. Duża część szkoleń odbywa się we współpracy z wojskowymi centrami szkoleń, jednostkami wojskowymi i innymi organizacjami jak np. GOPR.

Jednostki proobronne przyciągają do siebie dużo chętnych przez możliwość zdobycia dodatkowych punktów do rekrutacji w jednostkach mundurowych, chęcią rozwijania zainteresowań, czy możliwością spędzenia czasu aktywnie. Dużą liczbę różnych organizacji, federacji i stowarzyszeń, rozlokowaną na terenie Polski, jak to przedstawia Rysunek 1. Przez brak dostępnych na rynku rozwiązań wspomagających zarządzanie takimi jednostkami ciężko określić liczbę aktywnych członków wchodzących w skład jednostek paramilitarnych.





Rysunek 1. Oddziały organizacji paramilitarnych w Polsce

*Źródło:* (Soloch, Żurawski i Dryblak, 2015, str. 26)

### Geneza

Genezą do zaprojektowania i stworzenia takiej aplikacji były zainteresowania autora w zakresie obronności kraju oraz doświadczenia nabyte przez prowadzenie plutonu organizacji proobronnej, która nie posiadała żadnego systemu informatycznego. Rozmowy autora z innymi członkami różnych organizacji proobronnych wskazywały, że na taki system jest zapotrzebowanie, gdyż pozwoli na łatwiejszą kontrolę stanu osobowego jednostki, łatwiejszy dostęp do kalendarza zajęć i ocen. Dodatkowo obserwując oprogramowanie w Wojsku Polskim, które nie jest dostępne poza jednostkami wojskowymi, autor chciał stworzyć alternatywę, którą będą mogły używać jednostki paramilitarne do planowania ćwiczeń taktycznych, oraz poznania grafiki operacyjnej.

## Cel i zakres pracy

Celem pracy jest zaprojektowanie i implementacja pakietu aplikacji na różne platformy (Web, Windows 10, Android) wspomagających zarządzanie organizacją proobronną. Każda z aplikacji ma umożliwić użytkownikowi logowanie, podgląd swoich danych, ocen, pododdziału, oraz zajęć. Aplikacja webowa ma dodatkowo pozwolić użytkownikowi na rejestrację, zmianę swoich danych, oraz hasła, a administratorowi zarządzanie ocenami, zajęciami, użytkownikami, stopniami i pododdziałami. Aplikacja desktopowa rozbudowana będzie o edytor sytuacji taktycznych ułatwiająca planowanie zajęć taktycznych, oraz podstawową naukę symboli używanych w grafice operacyjnej.

Zakres pracy obejmuje stworzenie REST API w ASP .NET Core z wykorzystaniem języka C#, które za pomocą zapytań do *endpointów* przy użyciu protokołu HTTP ma umożliwić autoryzację, autentykację oraz dostęp do zasobów bazy danych. Odpowiedzi do klienta będą zwracane w formacie JSON. Baza danych utworzona będzie w podejściu *Code-First* z użyciem EntityFramework. Aplikacja desktopowa ma zostać napisana w języku C# z wykorzystaniem WPF i darmowego rozszerzenia ModernWPF który zagwarantuje nowoczesny styl, oraz reagowanie na ustawiony motyw systemu Windows 10. Aplikacja na urządzenia mobilne z systemem Android będzie wykorzystywać platformę Xamarin, a aplikacja Web napisana będzie przy użyciu *frameworka* Angular.

W pierwszym rozdziale zostaną zaprezentowane technologie zastosowane do stworzenia aplikacji. W kolejnej części przedstawiony zostanie projekt systemu, a w podsumowaniu plany rozwojowe aplikacji.

## Technologie



### .NET

.NET jest darmową platformą programistyczną, a od wersji Core również otwarto-źródłową. Umożliwia ona pisanie większości rodzajów aplikacji webowych, chmurowych, desktopowych, mobilnych czy konsolowych, a nawet gier i aplikacji wykorzystujących uczenie maszynowe. .NET pozwala na programowanie w językach: C#, F# oraz Visual Basic. Najnowsza wersja platformy to .NET 5.0 która wprowadza obsługę nowych wersji języków, poprawia wydajność i wiele więcej (Wprowadzenie do .NET, 2020).

### C#

C# jest obiektowym językiem programowania wysokiego poziomu, stworzonym dla firmy Microsoft, a jego pierwsze wydanie pojawiło się w lipcu 2000 r. Język ten, podobnie jak C, C++, Java, czy JavaScript, ma składnię opartą o nawiasy klamrowe, co umożliwia programistom na łatwe przeniesienie się na niego. Ponadto został on wyposażony w obsługę typów generycznych, typy *nullowalne*, wyrażenia lambda i dużo innych przydatnych funkcji ułatwiających pisanie aplikacji w tym języku. LINQ (Language-Integrated Query) umożliwia wykonywanie zapytań na obiektach co pozwala programiście na łatwą i szybką pracę z dużą ilością obiektów. Aplikacje napisane w C# działają na platformie .NET, a kod źródłowy kompilowany jest do kodu pośredniego (intermediate language), który spełnia specyfikację CLI (Common Language Infrastructure). Do uruchomienia takiej aplikacji system musi być wyposażony w środowisko uruchomieniowe CLR (Common Language Runtime). Najnowszą wersją języka C# jest wersja 9.0, która wymaga do działania platformy .NET 5 (C#, 2020).

### ASP .NET Core

ASP .NET jest platformą do tworzenia aplikacji webowych, serwisów i *backend*-u aplikacji mobilnych. Wersja Core jest otwarto-źródłowym rozwinięciem ASP .NET 4. W porównaniu do wersji opartej na .NET Framework działa na wielu systemach. Charakteryzuje się lekkością i lepszą wydajnością. ASP .NET Core umożliwia programowanie aplikacji w oparciu o wzorzec MVC (Model-View-Controller) lub Razor Pages, czyli model programowania oparty na stronach. Framework wyposażony jest również w funkcję automatycznego łączenia modeli z zapytania HTTP do parametrów metod, oraz umożliwia ich walidację po stronie serwera jak i klienta (ASP .NET, 2020).

### WPF

Windows Presentation Foundation (WPF) jest *frameworkiem* interfejsu użytkownika dla desktopowych aplikacji z systemem Windows. Stanowi część platformy .NET. Umożliwia łatwe tworzenie widoków z pomocą języka XAML i edytora WYSIWYG, który pozwoli na tworzenie wyglądu aplikacji bez znajomości programowania. Dużo wbudowanych kontrolek, oraz spora ilość pakietów rozbudowujących ich bazę lub zmieniająca wygląd pozwala na dopasowanie widoków do potrzeb tworzonej aplikacji. XAML wyposażony jest w funkcję *Hot Reload*, która pozwala na edycję wyglądu w czasie działania programu bez konieczności kompilacji kodu. Ważnym atutem tego *frameworka* jest mechanizm wiązania danych, który umożliwia łączenie widocznych na ekranie informacji ze źródłami danych (WPF, 2020).

### Xamarin

Xamarin to otwarto-źródłowa platforma służąca do tworzenia aplikacji na urządzenia mobilne i komputery z systemem Windows z użyciem .NET. Xamarin działa jako abstrakcyjna warstwa, która komunikuje się z platformą, na którą jest dedykowana. Podejście takie zapewnia możliwość wykorzystania około 90% współdzielonego kodu aplikacji na wszystkie platformy, zachowując jednocześnie natywną szybkość działania i wygląd na każdej z nich. Aplikacje kompilowane są do dedykowanych rozszerzeń platform jak .apk dla Androida, czy .IPA dla iOS (Xamarin, 2021).

### TypeScript

TypeScript jest językiem programowania stworzonym przez firmę Microsoft. Motywacją do stworzenia takiego języka była chęć pisania aplikacji webowych w JavaScript z wykorzystaniem tradycyjnego podejścia obiektowego. Język ten oferuje wszystkie możliwości JavaScriptdodając do tego system silnego typowania. Dzięki temu TypeScript potrafi wskazywać niechciane zachowania w kodzie programu, minimalizując tym samym szansę na powstanie błędów (TypeScript, 2021).

### Angular

Angular to platforma do tworzenia aplikacji webowych typu *single-page* przy użyciu HTML i języka TypeScript. Struktura aplikacji Angular oparta jest na komponentach, które są dynamicznymi widokami i dostosowywane są w zależności od danych i logiki programu. Komponenty organizowane są w moduły gromadzące kod w bloki funkcjonalne. Do działania komponenty wykorzystują serwisy, które zapewniają określone funkcjonalności. Serwisy są wstrzykiwane do komponentów jako zależności, co pozwala zachować czysty, modularny i wydajny kod (Angular, 2021).

## Projekt aplikacji



### Wymagania funkcjonalne

Przed aplikacjami stawiane są następujące wymagania funkcjonalne:

* Możliwość rejestracji na aplikacji webowej.
* Możliwość logowania na każdej platformie.
* Możliwość przeglądania ocen użytkownika na każdej platformie.
* Możliwość podglądu zajęć użytkownika na każdej platformie.
* Możliwość podglądu własnego pododdziału i jego członków.
* Możliwość tworzenia sytuacji taktycznych na platformie desktopowej.
* Możliwość edycji swoich danych na aplikacji webowej.
* Panel administracyjny na aplikacji webowej.

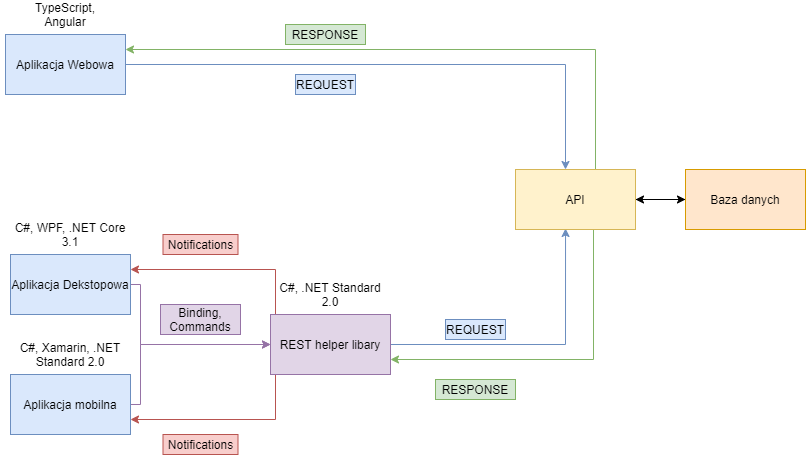
### Wymagania niefunkcjonalne

Przed aplikacjami stawiane są następujące wymagania niefunkcjonalne:

* Dopasowanie stylu interfejsu do ustawionego motywu systemowego na platformie mobilnej i desktopowej.
* Responsywna aplikacja webowa.
* Prosty i łatwy w użytkowaniu interfejs użytkownika.

### Komunikacja komponentów projektu

Żeby otrzymać informację z bazy danych aplikacje klienckie muszą komunikować się pośrednio, lub bezpośrednio z API. Wykorzystując fakt, że aplikacje na systemy Windows i urządzenia mobilne korzystają z platformy .NET, można oddzielić od projektu warstwę pośrednią w formie biblioteki zajmującej się komunikacją między API, a widokami aplikacji klienckich (Rysunek 2).



Rysunek 2 Schemat komunikacji

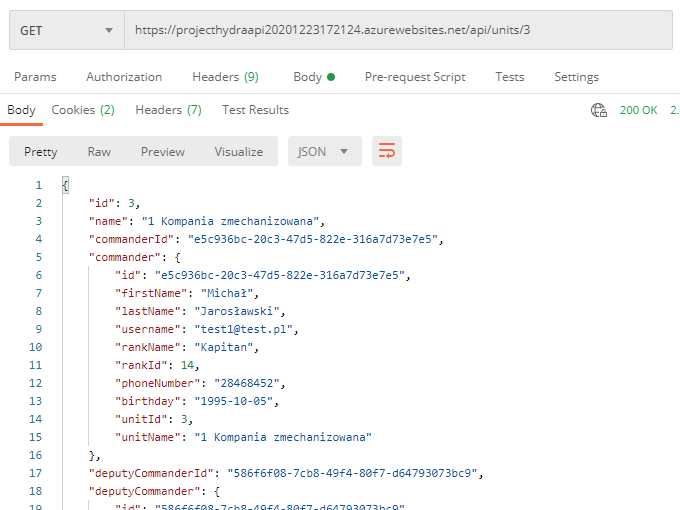
Źródło: Opracowanie własne

### Rest API

**REST (Representational State Transfer) API** jako architektura w tworzeniu serwisów webowych powstał w 2000 r. Zaproponował ją Roy Fielding. Nie ona jest związana z żadnym protokołem, lecz powszechnym stało się stosowanie jej z wykorzystaniem protokołu HTTP. Podstawowymi założeniami REST jest używanie architektury Client-Server (implementacja serwera powinna być całkowicie niezależna od aplikacji klienckich). Połączenia powinny być bezstanowe (stateless), co oznacza, że każde z nich powinno być niezależne, a API przy ich obsłudze bazować tylko na danych dostarczonych wraz z połączeniem. Zapytanie HTTP powinno składać się z:

* Request method – jaka operacja ma zostać wykonana. Podstawowe metody zapytań to:
  + GET – pobieranie danych.
  + PUT – edycja danych.
  + POST – utworzenie nowych danych.
  + DELETE – usunięcie danych.
  + PATCH – częściowa edycja danych.
* Nagłówek (header) – zawiera dodatkowe informacje dotyczące żądania.
* Ścieżka (resource path) – URI do danego zasobu.
* Body – dodatkowe informacje np. do edycji/dodania zasobu.

Odpowiedzi od serwera mają strukturę podobną do zapytania, lecz zamiast używanej metody odpowiedź zawiera kod odpowiedzi (Response Code). Przykładem takiego kodu może być 404 (nie znaleziono), lub 200 (OK) (Block i in., 2016).



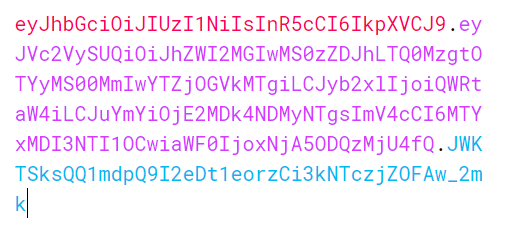
Rysunek 3 Przykład zapytania i odpowiedzi API w programie Postman

Źródło: Zrzut ekranu z programu Postman

Rest API ma umożliwić autoryzację i dostęp do zasobów z bazy danych. W tym samym projekcie ma zostać również utworzona baza danych z wykorzystaniem Entity Framework Core.

**Entity Framework Core** – nowoczesny mapper obiektów do i z bazy danych dla platformy .NET. Obsługuje zapytania LINQ i migracje, oraz współpracuje z różnymi bazami danych. W wersji Core działający również na wielu platformach (Entity Framework, 2020).

**JSON Web Token** – to otwarty standard (RFC 7519) oferujący prostą i bezpieczną wymianę informacji z użyciem obiektów JSON. Informacja może być zaufana i zweryfikowana, ponieważ szyfrowana jest sekretnym kluczem z użyciem algorytmu HMAC lub parą kluczy prywatnych/publicznych RSA, lub ECDSA. JWT może zostać użyty do wymiany informacji, lecz zazwyczaj używany jest do autoryzacji. Token składa się z nagłówka (header), zawartości (payload), oraz sygnatury (signature). Każda z części tokenu oddzielona jest ‘.’. Przykładowy token JWT wygląda następująco:



Rysunek 4 Token JWT

Źródło: Zrzut ekranu ze strony jwt.io. Token wygenerowany przez własne API

Nagłówek tokenu zawiera zazwyczaj informacje jaki algorytm został użyty do szyfrowania, oraz typ informacji (w tym przypadku JWT).



Rysunek 5 Zdekodowany nagłówek JWT

Źródło: Zrzut ekranu ze strony jwt.io

Zawartość tokenu to dane jakie zawiera klucz. Można je wykorzystać do przechowania informacji o użytkowniku w celu jego łatwiejszej identyfikacji, daty ważności klucza i innych danych.



Rysunek 6 Zdekodowany payload JWT

Źródło: Zrzut ekranu ze strony jwt.io

Ostatnim elementem JWT jest sygnatura która pozwoli na weryfikacje czy token nie został w żaden sposób zmodyfikowany (JWT, 2020).

#### Baza Danych

Baza danych przechowuje informacje, oraz umożliwia ich odczyt, zapis i edycję. Bazy danych są integralną częścią większości aplikacji. Najczęściej stosowanym typem są relacyjne bazy danych w której dane przechowane są w tabelach składających się z kolumn i wierszy (rekordów). Relacje umożliwiają powiązanie ze sobą danych z różnych lub tych samych tabel.

Do projektu zostanie użyta baza danych tworzona i rozwijana przez firmę Microsoft MSSQL. Umożliwi to łatwą integracje z API, które jest realizowane w technologii rozwijanej przez tę samą korporacje, oraz ułatwi wdrożenie na platformę chmurową Azure.

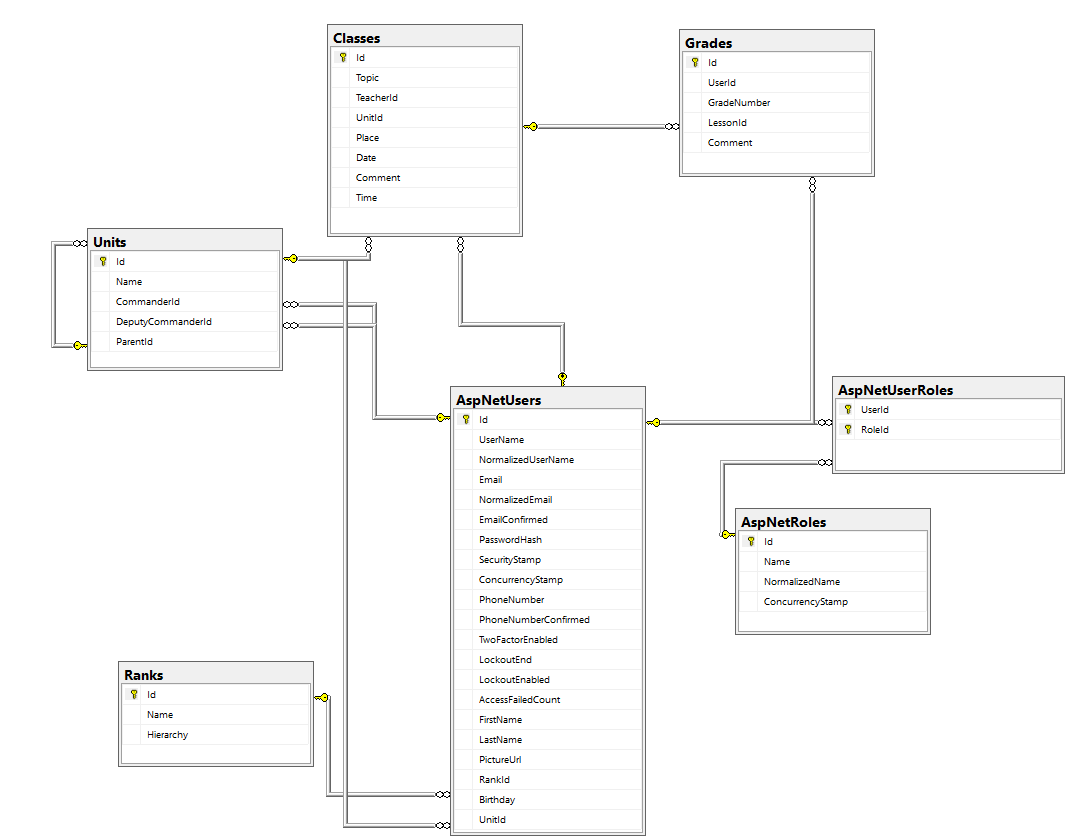
Poszczególne rekordy identyfikowane są za pomocą unikalnego identyfikatora ID który w przypadku tego projektu przyjmuje liczbę naturalną lub GUID.

**GUID** to globalnie unikatowy identyfikator używany głównie w systemie Windows. Użycie GUID w bazie danych umożliwia łatwe połączenie kilku baz danych o tej samej strukturze, ponieważ prawdopodobieństwo wystąpienia dwóch tych samych kluczy jest bardzo niewielkie.



Rysunek 7 Przykładowy GUID

Źródło: GUID wygenerowany przez generator www.guidgenerator.com



Rysunek 8 Struktura bazy danych

Źródło: Zrzut ekranu z programu MSQLSMS

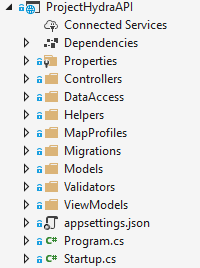
Schemat powyżej (Rysunek 8) obrazuje tabele i relacje zachodzące między nimi w tworzonym projekcie.

Tabele wchodzące w skład bazy danych:

* AspNetUsers – tabela przechowująca dane dotyczące użytkowników, ich jednostkę, stopień, oraz zahaszowane hasła.
* AspNetRoles – tabela przechowująca role dostępne w aplikacji.
* AspNetUserRoles – tabela potrzebna do relacji wiele do wielu między użytkownikami, a rolami.
* Ranks – tabela przechowująca informacje o stopniach. Każdy stopień przypisany ma również numer w hierarchii co pozwala na sortowanie użytkowników w danej jednostce.
* Untis – tabela przechowująca informacje o pododdziałach. Zawiera nazwę Id dowódcy i jego zastępcy, oraz opcjonalny pododdział pod który podlega.
* Classes – tabela przechowująca informacje o zajęciach. Zawiera informacje dla kogo są organizowane zajęcia, ich datę, czas trwania, kto będzie je prowadził, oraz opcjonalny komentarz.
* Grades – tabela przechowująca dane o ocenach użytkowników.

#### Struktura projektu REST API

* Controllers – klasy odpowiedzialne za obsługę zapytań
* DataAccess – klasy zawierające logikę dostępu do danych
* Helpers – pomocnicze klasy zawierające nazwy
* MapProfiles – profile mapowania modeli do i z view-modeli
* Migrations – pliki migracyjne bazy danych
* Models – klasy zawierające strukturę danych
* Validators – klasy zawierające obsługę walidacji modeli
* ViewModels – klasy zawierające strukturę modeli odpowiedzialną za udostępnianie danych z i do widoku
* Program i Startup – pliki do konfiguracji i uruchamiania API



Rysunek 9 Struktura API

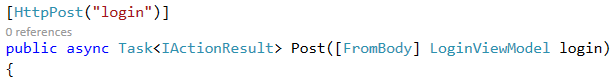
Źródło: Zrzut ekranu z programu Visual Studio 2019

#### Dostęp do zasobów

Bezpieczeństwo zasobów to istotna kwestia, którą należy uwzględnić przy projektowaniu aplikacji. Należy rozdzielić metody dostępne dla użytkownika zalogowanego, administratora, oraz gościa. Edycja danych inne niż swoje to domena administratora, użytkownik nie powinien również edytować danych innego użytkownika lub swoich ocen, a gość nie powinien mieć dostępu do poufnych informacji użytkownika. Żeby to osiągnąć zostanie użyta autoryzacja oparta na rolach. ASP .NET Core umożliwia w łatwy sposób kontrolowanie dostępu do metody lub całej klasy przez określenie atrybutu *Authorize*.

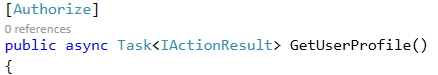
Autoryzacja oparta na rolach w ASP umożliwia nam skonfigurowanie dostępu na 3 sposoby:

* Brak atrybutu – dostęp będzie miał każdy



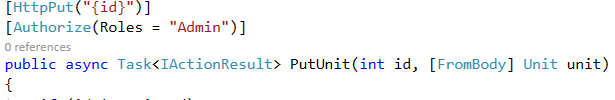
Rysunek 10 Przykład dostępu do metody dla każdego

* Pusty atrybut *[Authorize]* – dostęp dla zalogowanego użytkownika



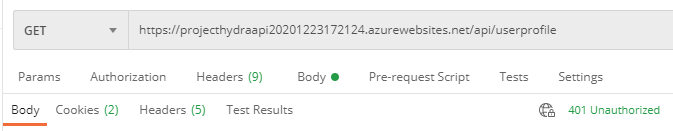
Rysunek 11 Przykład dostępu do metody dla zalogowanego użytkownika

* Określenie ról w atrybucie – dostęp dla określonych w atrybucie ról



Rysunek 12 Przykład dostępu do metody dla administratora

Zalogowany użytkownik i jego rola weryfikowane są poprzez klucz JWT. Niepoprawny klucz lub jego wygaśnięcie spowoduje odrzucenie zapytania i zwrócenie błędu jak na przykładzie poniżej (Rysunek 13).

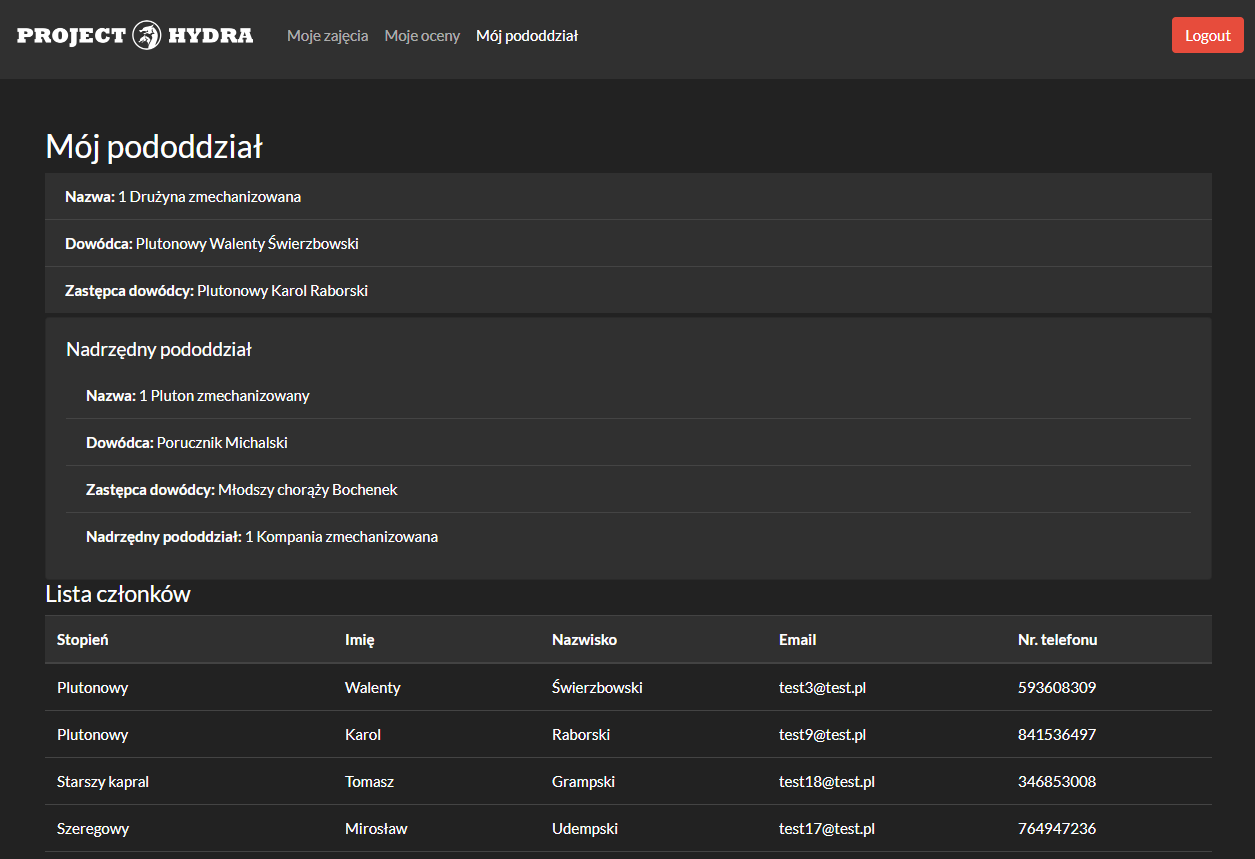


Rysunek 13 Przykład odrzuconego zapytania

Źródło: Zrzut ekranu z programu Postman

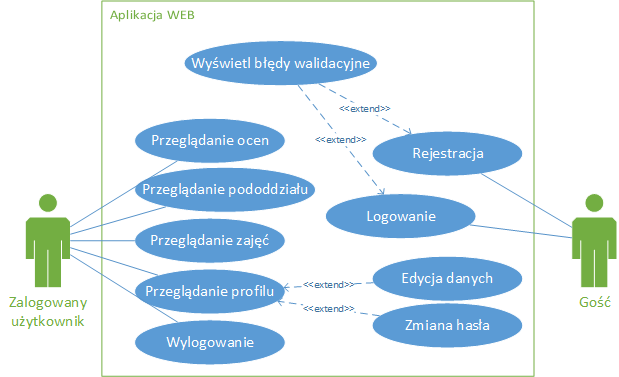
### Aplikacja WEB

Aplikacja webowa będzie jedyną możliwością utworzenia nowego konta w systemie (nie licząc ręcznego utworzenia konta przez administratora) oraz jedyną aplikacją zawierającą panel administracyjny. Rozwiązanie takie zapewni administratorom dostęp do panelu z każdego urządzenia, a użytkownikom możliwość sprawdzenia zajęć i ocen bez konieczności instalacji dodatkowych aplikacji.



Rysunek 14 Aplikacja webowa

#### Diagram przypadków użycia dla użytkownika

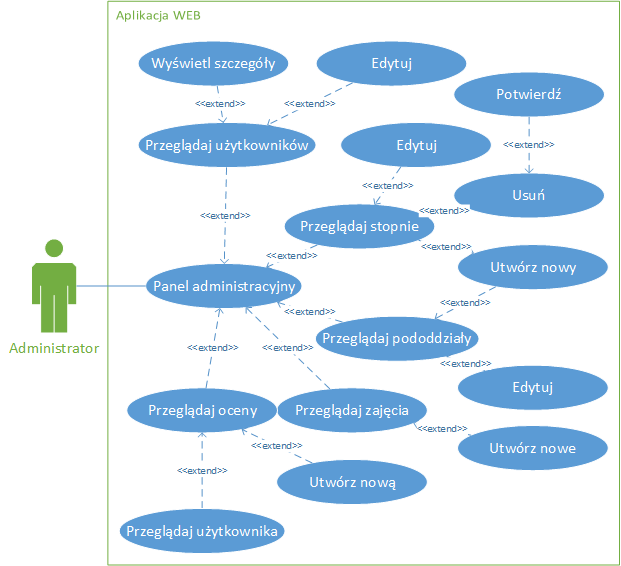


Rysunek 15 Diagram przypadków użycia dla użytkownika aplikacji webowej

Źródło: Opracowanie własne

Schemat (Rysunek 15) pokazuje uproszczony model użycia aplikacji. Niezalogowany użytkownik ma dostęp jedynie do logowania i rejestracji, a uwierzytelniona osoba ma możliwość przeglądania danych które ją dotyczą, oraz edycję niektórych swoich informacji jak np. nr telefonu.

#### Diagram przypadków użycia dla administratora



Rysunek 16 Diagram przypadków użycia dla administratora aplikacji webowej

Źródło: Opracowanie własne

Administrator aplikacji posiada wszystkie możliwości zwykłego użytkownika poszerzone o możliwość zarządzania danymi w bazie danych z poziomu panelu administracyjnego.

#### Dostęp do panelu administracyjnego i obsługa błędów

Dostęp do zasobów będzie zaimplementowany po stronie API, jednak by zwiększyć bezpieczeństwo i uniemożliwić użytkownikom przeglądanie panelu administracyjnego, w aplikacji webowej zostanie dodana obsługa błędów i przechwytywanie zapytań. Widoczność przycisku panelu administracyjnego będzie zależna od roli jaka przechowywana jest w tokenie JWT. Jeśli jednak użytkownik wpisze ręcznie adres URI panelu administracyjnego zostanie zwrócony następujący komunikat:

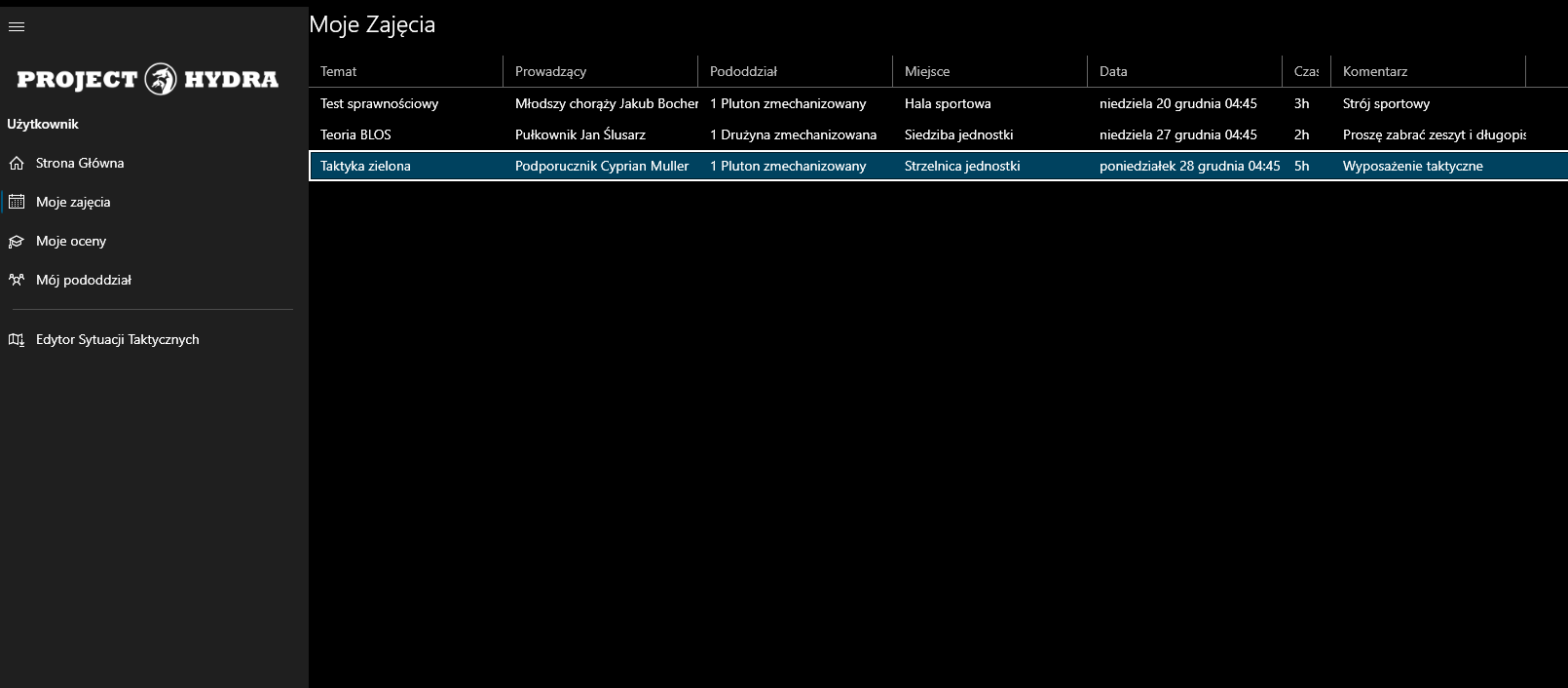


Rysunek 17 Komunikat błędu aplikacji webowej

Wyświetlanie takiego komunikatu jest zasługą *HttpInterceptor* przez którego przechodzą wszystkie zapytania z i do aplikacji. Jeśli w pamięci jest token JWT to klasa przechwytująca dołącza go do nagłówka przy zapytaniach. Przy odbieraniu odpowiedzi sprawdza jaki jest kod odpowiedzi i w przypadku kodu błędu zwraca odpowiednią stronę z błędem. Jeśli zwrócony w odpowiedzi kod oznacza poprawne wykonanie akcji pozwala na dostęp do danej strony.

### Aplikacja desktopowa

Aplikacja na urządzenia z systemem Windows ma się cechować przede wszystkim graficznym edytorem sytuacji taktycznych, lecz także będzie posiadała możliwość przeglądania zajęć, ocen i pododdziału. Aplikacja zostanie zrealizowana z wykorzystaniem architektury MVVM z *ViewModelami* wydzielonymi do osobnego projektu jak przedstawia to(Rysunek 2).



Rysunek 18 Aplikacja desktopowa

#### Model MVVM

*Model-view-viewmodel* to wzorzec architektoniczny oprogramowania, który ma za zadanie rozdzielenie poszczególnych warstw aplikacji w celu podziału odpowiedzialności i zredukowania zależności między nimi. Wzorzec został zapoczątkowany przez architektów oprogramowania z firmy Microsoft, a mianowicie Ken Cooper’a i Ted Peters’a w celu uproszczenia programowania sterowanego zdarzeniami. Wzorzec MVVM dzielimy na 3 warstwy:

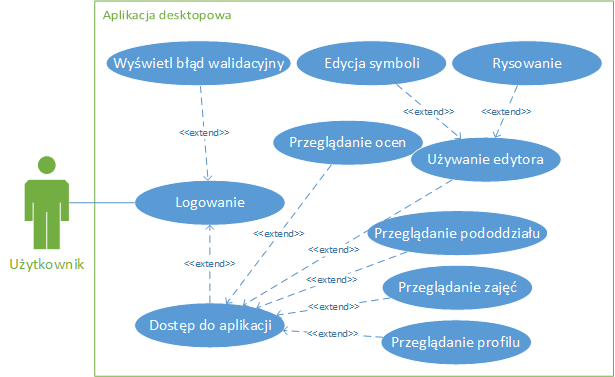
* Warstwa modelu – warstwa ta zajmuje się reprezentacją danych. Może komunikować się z bazą danych, pobierać dane z pliku, bądź internetowego *API*. Komunikuję się z *view-modelem* za pomocą komend i *data binding*.
* Warstwa widoku – część zajmująca się komunikacją z użytkownikiem. Posiada interfejs użytkownika (*UI*). Dane do wyświetlania otrzymuje od warstwy modelu widoku.
* Warstwa modelu widoku – łącznik widoku i modelu. Do zadań tej warstwy należy udostępnianie danych do widoku, oraz pobieranie i aktualizowanie informacji w modelu (Matulewski, 2016).



Rysunek 19 Wzorzec MVVM

Źródło: Opracowanie własne

#### Diagram przypadków użycia



Rysunek 20 Diagram przypadków użycia aplikacji desktopowej

Źródło: Opracowanie własne

Diagram przedstawiony powyżej (Rysunek 20) pokazuje możliwe przypadki użycia aplikacji. Użytkownik po włączeniu aplikacji może się jedynie zalogować. W aplikacji desktopowej nie ma przewidzianej możliwości rejestracji. Po poprawnym zalogowaniu użytkownik zyskuje dostęp do przeglądania informacji i używania edytora taktycznego.

#### Struktura aplikacji

• Images – logo i inne grafiki.

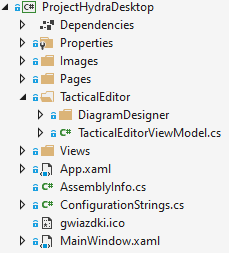
• Pages – strony reprezentujące wybraną zakładkę.

• TacticalEditor – foldery i klasy odpowiedzialne za edytor taktyczny.

• Views – katalog zawierający okna aplikacji.

• ConfigurationStrings.cs –klasa zawierający adres API.

• MainWindow.xaml – główne okno aplikacji dostępne po zalogowaniu.

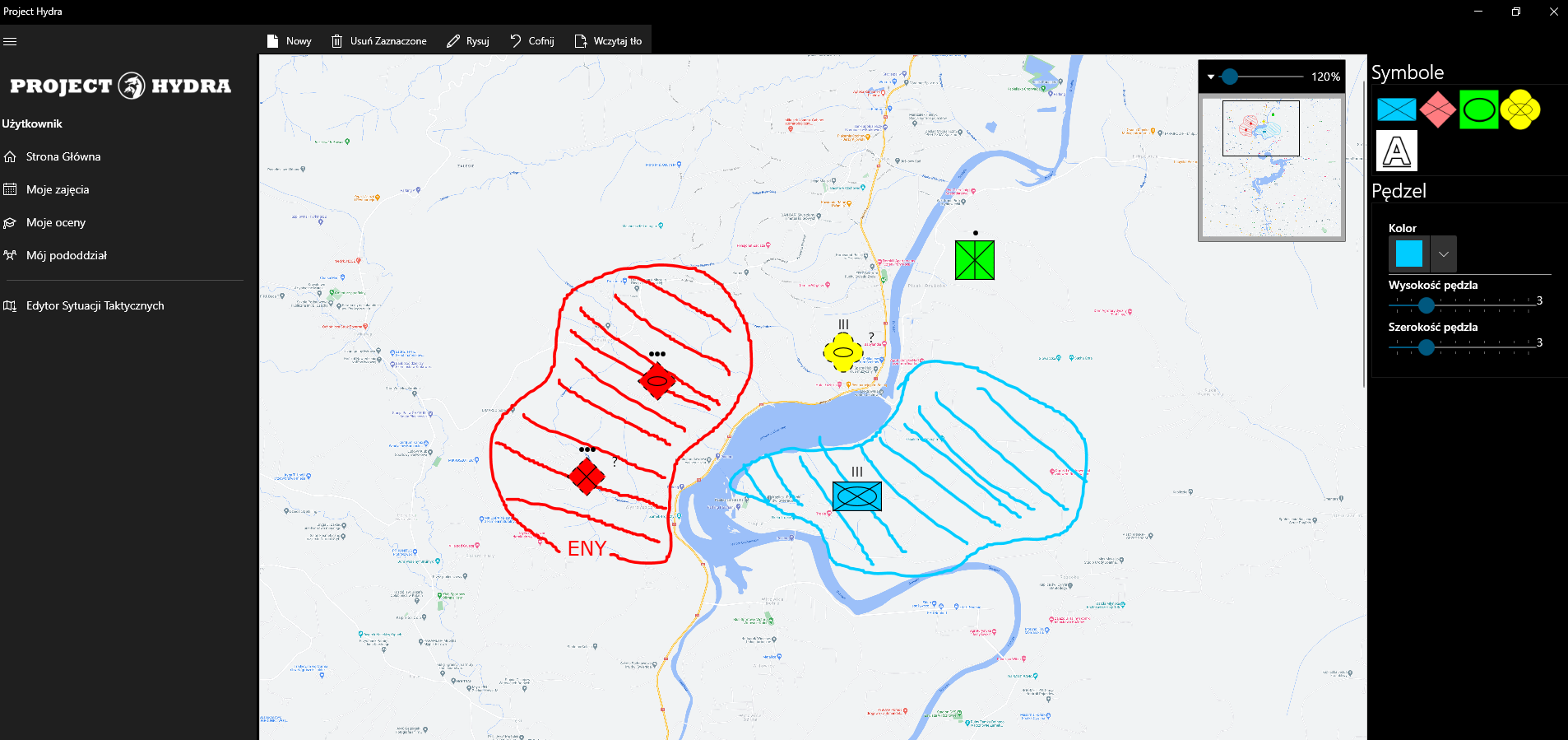


Rysunek 21 Struktura aplikacji desktopowej

Źródło: Zrzut ekranu z programu Visual Studio 2019

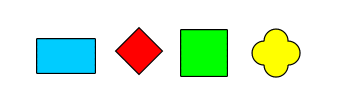
#### Edytor taktyczny i budowa znaków

Edytor taktyczny ma umożliwić użytkownikom planowanie działań taktycznych i naukę znaków taktycznych. Oprócz samej opcji dodawania znaków ma również zostać wyposażony w możliwość wczytywania plików graficznych jako tło, odręcznego rysowania z ustawieniami pędzla, oraz dodawanie tekstu. Wszystkie znaki mają być zgodne z przyjętymi normami w Wojsku Polskim.



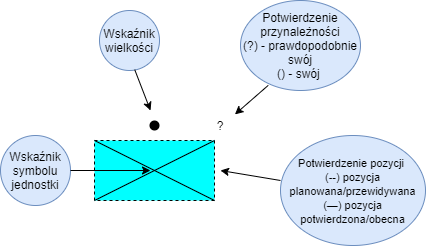
Rysunek 22 Edytor taktyczny

W pierwszej wersji oprogramowania dostępne będą jedynie symbole jednostek. Kolejne wersję mogą zostać rozbudowane o symbole obiektów i sprzętu.



Rysunek 23 Symbole przynależności jednostek

Rysunek powyżej (Rysunek 23) prezentuje rodzaje przynależności jednostek. Rozróżniając kolorystykę i kształt symbolu można zidentyfikować przynależność jako: swój, wrogi, neutralny lub sojuszniczy. Niebieski prostokąt oznacza jednostkę swoją, czerwony romb wrogą, zielony kwadrat neutralną, a żółty czterolistny ornament nieznaną (ppłk dr Wrona i in., 2005). W celu dokładniejszego zobrazowania stanu jednostki stosuje się modyfikatory. Poniżej zaprezentowany jest schemat modyfikatorów dostępnych w edytorze.



Rysunek 24 Modyfikatory symboli

Symbole jednostki na znaku służą do określenia typu danej jednostki. Pierwsza wersja edytora zawierać będzie jedynie podstawowe znaki wojsk lądowych z powodu długiego czasu wymaganego na stworzenie jednego symbolu dla każdego typu przynależności. Poniższa tabela (Tabela 1) przedstawia dostępne symbole jednostek.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nazwa typu jednostki** | **Swój** | **Przeciwnik** | **Neutralny** | **Nieznany** |
| Piechota |  |  |  |  |
| Piechota zmechanizowana |  |  |  |  |
| Piechota zmotoryzowana |  |  |  |  |
| Pancerna |  |  |  |  |

Tabela 1 Typy jednostek

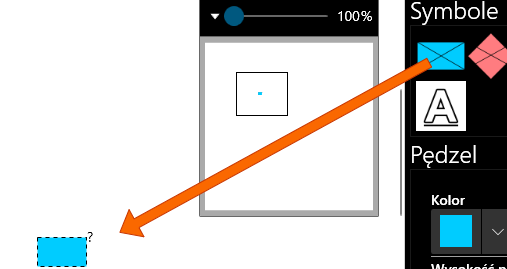
Ważnym modyfikatorem jest także wielkość jednostki. Poniżej (Tabela 2) zaprezentowane są dostępne w edytorze wskaźniki wielkości.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nazwa wskaźnika wielkości** | **Swój** | **Przeciwnik** | **Neutralny** | **Nieznany** |
| Obsługa/Sekcja |  |  |  |  |
| Drużyna |  |  |  |  |
| Pluton |  |  |  |  |
| Kompania |  |  |  |  |
| Batalion |  |  |  |  |
| Pułk |  |  |  |  |
| Brygada |  |  |  |  |
| Dywizja |  |  |  |  |
| Korpus |  |  |  |  |

Tabela 2 Wskaźniki wielkości jednostki

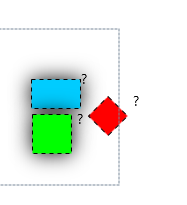
#### Obsługa edytora taktycznego

Działanie z edytorem można podzielić na dwa tryby: rysowanie i umieszczenie symboli. Kiedy tryb rysowania jest włączony nie można umieszczać symboli ani ich modyfikować. Umieszczenie symboli odbywa się przy wyłączonej opcji rysowania. W celu umieszczenia symbolu na planszy należy przeciągnąć go z menu po prawej stronie.



Rysunek 25 Umieszczenie symbolu

Przesuwanie umieszczonego symbolu odbywa się poprzez umieszczenie kurosa w okolicy znaku. Symbol kursora powinien zmienić się na strzałki należy wtedy przeciągnąć symbol w inne miejsce. Zaznaczanie wielu elementów odbywa się poprzez przytrzymanie kursora na planszy i wskazanie prostokąta w którym chcemy zaznaczyć symbole. Zaznaczone symbole powinny rzucać cień. Grupę zaznaczonych symboli można razem przesuwać oraz usuwać.



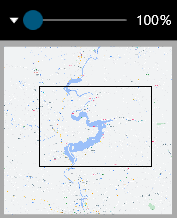
Rysunek 26 Zaznaczanie wielu symboli

Menu powyżej panelu umożliwia utworzenie nowej sytuacji taktycznej co wykasuje wszystkie symbole oraz rysunki, zaznaczone symbole można usunąć przyciskiem „Usuń Zaznaczone”. Tryb rysowania można włączyć/wyłączyć przyciskiem „Rysuj” (włączony tryb rysowania jest symbolizowany kolorem tła zależnym od ustawionego w systemie koloru akcentów). Przycisk „Cofnij” cofa ostatnio narysowany rysunek, a „Wczytaj tło” umożliwia wczytanie bitmapy na tło (zalecana wielkość bitmapy to 8000x8000px).



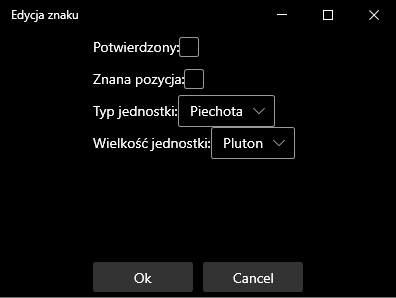
Rysunek 27 Górne menu edytora z włączoną opcją rysowania

Przybliżanie i oddalanie planszy odbywa się poprzez ruch rolką myszki w przód i w tył lub poprzez użycie suwaka znajdującego się nad podglądem planszy. Przesuwanie w inny region planszy dokonuje się poprzez przeciągnięcie prostokąta na podglądzie.



Rysunek 28 Podgląd planszy

W celu edycji znaku należy najechać na symbol. Przy kursorze powinien pojawić się wtedy znak zapytania. Okno dialogowe edycji pojawi się po podwójnym naciśnięciu lewego przycisku myszy.

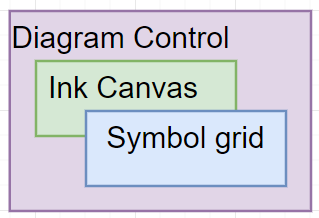


Rysunek 29 Okno dialogowe edycji znaku

#### Zasada działania edytora

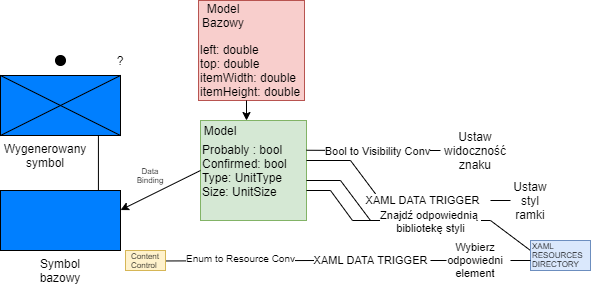
**XAML** (eXtensible Application Markup Language) – język służący do opisywania UI stosowany głównie w WPF. Jest to język oparty o XML dostosowanym do opisu złożonych interfejsów użytkownika (XAML, 2020).

Edytor podzielony jest na dwie warstwy. Warstwę dolną na która składa się kontrolka WPF *Ink Canvas*. Umożliwia ona rysowanie, oraz wczytywanie tła. Na górze znajduję się warstwa symboli, która wyświetla i przechowuję informacje o pozycji elementów. Rodzicem dwóch warstw jest *DiagramControl,* który umożliwia poruszanie się po planszy i dopasowanie jej skali.



Rysunek 30 Schemat warstw edytora

Każdy symbol dziedziczy po klasie bazowej podstawowe wartości potrzebne do wyświetlania i działania symboli. Następnie każdy symbol rozszerza model o swoje właściwości. Każdy symbol potrzebuje własny *DataTemplate* czyli bazowy symbol napisany w XAML. W przypadku symboli jednostek wielkość i jej typ określany jest właściwością typu wyliczeniowego który jest zamieniany przez konwerter na odpowiedni element w bibliotece symboli określonego typu, a następnie przez wykorzystanie *Data Trigger* element ten jest dodawany do symbolu przez wybór *ContentTemplate* w kontrolce. Prostsze elementy takie jak symbol „?” oznaczający prawdopodobną przynależność realizowany jest poprzez konwersję zmiennej typu *bool* (obecnej w modelu)na typ *Visibility* (wbudowany w XAML),który określa widoczność elementu. Model wiązany jest z symbolem bazowym i powiadamiany o każdej zmianie właściwości przez co nie jest wymagane ręczne odświeżanie symbolu.

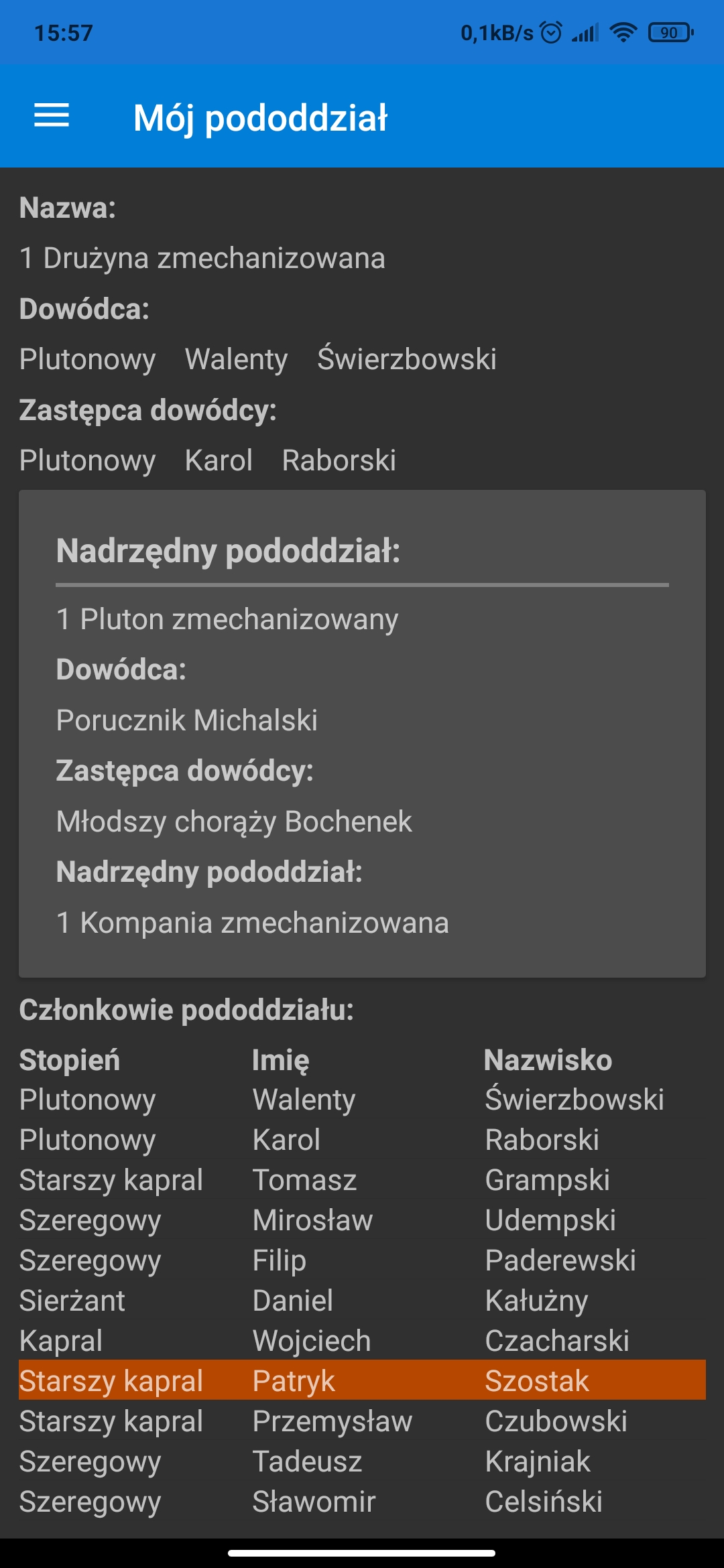


Rysunek 31 Schemat generowania symbolu

Każdy symbol generowany jest z wykorzystaniem prymitywów i ścieżek które są dostępne w WPF dzięki czemu każdy symbol wygląda dobrze niezależnie od przybliżenia, oszczędza miejsce które zajmowałyby duże ilości bitmap, oraz umożliwia ich edycję bez konieczności edycji w programie graficznym.

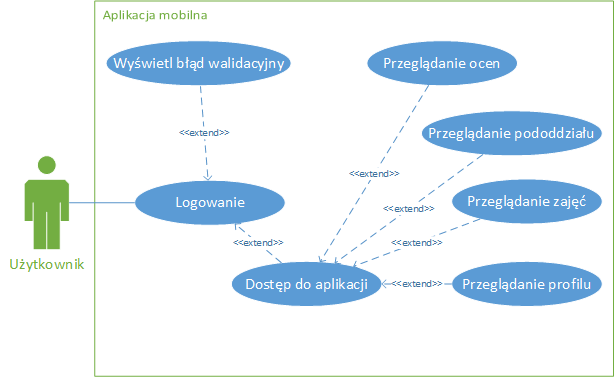
### Aplikacja mobilna

Głównym zadaniem aplikacji będzie szybki dostęp do informacji o ocenach i zajęciach z poziomu smartfonu. Z powodu braków sprzętowych aplikacja będzie dostępna jedynie na urządzenia z systemem Android, lecz kod powinien bez problemu skompilować się i działać na systemie iOS.



Rysunek 32 Aplikacja mobilna

#### Diagram przypadków użycia aplikacji mobilnej



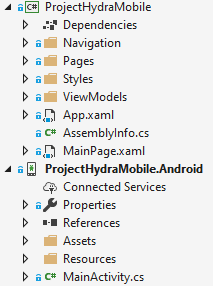
Rysunek 33 Diagram przypadków użycia aplikacji mobilnej

Aplikacja mobilna dzieli z aplikacją desktopową bibliotekę do komunikacji z *API*, co prezentuje (Rysunek 2), dlatego zasada działania i możliwości są podobne jak w przypadku aplikacji na systemy Windows (za wyjątkiem edytora, który byłby trudny w obsłudze na małym urządzeniu).

#### Struktura aplikacji mobilnej

Projekt aplikacji mobilnej dzieli się na dwie części: współdzieloną, która używana jest przez wszystkie platformy mogące działać z wykorzystaniem *frameworka* Xamarin, oraz dedykowaną konkretnej platformie. Kod zawarty w części współdzielonej posiada całą logikę programu, a w części na platformę android znajdują się m. in. grafiki oraz konfiguracja platformy.

* Navigation – strony i klasy odpowiedzialne za zakładki i boczne menu.
* Pages – strony reprezentujące wygląd i zawartość.
* Styles – pliki *css* odpowiedzialne za wygląd elementów.
* ViewModels – katalog zawierający *ViewModel* logowania.
* Assets – zestawy czcionek oraz inne dodatkowe pliki.
* Resources – logo i inne grafiki.



Rysunek 34 Struktura aplikacji mobilnej

#### Wiązanie strony z modelem widoku

Każda strona powiązana jest z modelem widoku znajdującym się w bibliotece. Na początku strony określany jest jej *ViewModel* i katalog w którym się znajduję.

            …

             xmlns:local="clr-namespace:ProjectHydraRestLibary.ViewModels;assembly=ProjectHydraRestLibary" >

    <ContentPage.BindingContext>

        <local:MyUnitViewModel />

    </ContentPage.BindingContext>

            …

Każdy model widoku implementuje interfejs *INotifyPropertyChanged* która umożliwia powiązanie go z widokiem poprzez *data binding*. Model widoku wstrzykuje wymagane zależności w konstruktorze dzięki wykorzystaniu klasy *ServiceProviderFactory*, która przechowuje referencje do dostawcy serwisów tworzonego przy uruchamianiu programu.

…

        public event PropertyChangedEventHandler PropertyChanged;

        private readonly IUnitService \_unitService;

        private readonly IAuthModel \_authModel;

        private readonly IUserService \_userService;

        public MyUnitViewModel()

        {

            \_unitService = ServiceProviderFactory.ServiceProvider

.GetRequiredService<IUnitService>();

            \_authModel = ServiceProviderFactory.ServiceProvider

.GetRequiredService<IAuthModel>();

            \_userService = ServiceProviderFactory.ServiceProvider

.GetRequiredService<IUserService>();

            \_ = LoadUserUnit();

        }

…

Następnie wywoływana jest metoda która używa wstrzykniętych serwisów do pobrania danych z *API*.

       …

 private UnitDetails unitDetails;

        private async Task LoadUserUnit()

        {

            var userDetails = await \_userService

.GetUserDetails(\_authModel.UserId);

            UnitDetails = await \_unitService

.GetUserUnit(userDetails.UnitId);

        }

…

Serwis wysyła żądanie do serwera, a następnie w przypadku poprawnego zwrócenia wyniku parsuje dane do odpowiedniego modelu i je zwraca.

        …

        private readonly IApiHelper \_apiHelper;

        public UnitService(IApiHelper apiHelper)

        {

            \_apiHelper = apiHelper;

        }

        public async Task<UnitDetails> GetUserUnit(int unitId)

        {

            var response = await \_apiHelper.GetApiClient()

.GetAsync("/api/units/" + unitId);

            if (response.IsSuccessStatusCode)

            {

                return await response.Content.ReadAsAsync<UnitDetails>();

            }

            else

            {

                throw new Exception(response.ReasonPhrase);

            }

        }

        …

Pobrane dane przypisywane są do właściwości w modelu widoku która przy ustawianiu powiadamia widok o zaktualizowanych danych.

        private UnitDetails unitDetails;

        private async Task LoadUserUnit()

        {

            var userDetails = await \_userService

.GetUserDetails(\_authModel.UserId);

            UnitDetails = await \_unitService

.GetUserUnit(userDetails.UnitId);

        }

        public UnitDetails UnitDetails

        {

            get { return unitDetails; }

            set {

                unitDetails = value;

                OnPropertyChanged("UnitDetails");

            }

        }

        protected void OnPropertyChanged(string name)

        {

            PropertyChanged?

.Invoke(this, new PropertyChangedEventArgs(name));

        }

W widoku (stronie) można użyć właściwości i jej składowych do wyświetlania informacji.

…

<Label Text="{Binding UnitDetails.Commander.RankName}" LineBreakMode="WordWrap" FontSize="Body" />

…

Podejście takie umożliwia asynchroniczne wczytywanie danych z serwera, a użycie wstrzykiwanych serwisów zapewnia możliwość używania ich w miejscach gdzie są potrzebne. Wszystkie serwisy konfigurowane są przy rozruchu aplikacji w plikach *App.cs* (dla aplikacji desktopowej i mobilnej). Użycie interfejsów pozwala na podmianę implementacji bez konieczności modyfikacji reszty programu.

## Podsumowanie



### Podsumowanie pracy

Celem pracy było zaprojektowanie i implementacja wieloplatformowego systemu zarządzania organizacją proobronną. Miałby wspierać członków różnych w organizacji w prostszym i szybszym zarządzaniu strukturami jednostki, odbywającymi się zajęciami, ocenami, jak i samymi członkami, oraz nauce znaków taktycznych i planowania zajęć.

System udało się zrealizować w podstawowej wersji. Praca na wielu aplikacjach dla wielu platform jednocześnie była wyzwaniem i pochłaniała dużo czasu, ale został osiągnięty poziom który umożliwia wdrożenie i zastosowanie w jednostkach proobronnych.

### Plany rozwoju systemu

Zastosowanie modelu wykorzystującego *REST API* i rozdzielenie aplikacji klienckich na niezależne od siebie systemy umożliwia łatwy sposób na dodanie kolejnych funkcjonalności w każdej z nich bez konieczności modyfikowania innych. W kolejnych wersjach należałoby uzupełnić bibliotekę symboli edytora o znaki sprzętu, obiektów i więcej modyfikatorów jednostek. Przeportowanie aplikacji desktopowej na nowszą platformę *UWP* umożliwiłoby również wykorzystanie dodatkowych funkcji *InkCavas* które obejmują wbudowany przybornik, oraz rozpoznawanie i poprawianie narysowanych kształtów. *API* i baza danych może zostać rozbudowana o więcej informacji jak np. składki członkowskie które można by było uregulować z wykorzystaniem płatności w aplikacji mobilnej, oraz system magazynowy do kontroli stanu magazynu i wydanych przedmiotów. Na aplikacji webowej mógłby zostać zaimplementowany system aktualności. Informacje o nowych ocenach i zajęciach mogłyby przychodzić w formie powiadomień na aplikacje mobilną.

## Spis rysunków

[Rysunek 1. Oddziały organizacji paramilitarnych w Polsce 5](#_Toc63171411)

[Rysunek 2 Schemat komunikacji 11](#_Toc63171412)

[Rysunek 3 Przykład zapytania i odpowiedzi API w programie Postman 12](#_Toc63171413)

[Rysunek 4 Token JWT 13](#_Toc63171414)

[Rysunek 5 Zdekodowany nagłówek JWT 13](#_Toc63171415)

[Rysunek 6 Zdekodowany payload JWT 13](#_Toc63171416)

[Rysunek 7 Przykładowy GUID 14](#_Toc63171417)

[Rysunek 8 Struktura bazy danych 15](#_Toc63171418)

[Rysunek 9 Struktura API 16](#_Toc63171419)

[Rysunek 10 Przykład dostępu do metody dla każdego 17](#_Toc63171420)

[Rysunek 11 Przykład dostępu do metody dla zalogowanego użytkownika 17](#_Toc63171421)

[Rysunek 12 Przykład dostępu do metody dla administratora 17](#_Toc63171422)

[Rysunek 13 Przykład odrzuconego zapytania 17](#_Toc63171423)

[Rysunek 14 Aplikacja webowa 18](#_Toc63171424)

[Rysunek 15 Diagram przypadków użycia dla użytkownika aplikacji webowej 18](#_Toc63171425)

[Rysunek 16 Diagram przypadków użycia dla administratora aplikacji webowej 19](#_Toc63171426)

[Rysunek 17 Komunikat błędu aplikacji webowej 20](#_Toc63171427)

[Rysunek 18 Aplikacja desktopowa 21](#_Toc63171428)

[Rysunek 19 Wzorzec MVVM 21](#_Toc63171429)

[Rysunek 20 Diagram przypadków użycia aplikacji desktopowej 22](#_Toc63171430)

[Rysunek 21 Struktura aplikacji desktopowej 23](#_Toc63171431)

[Rysunek 22 Edytor taktyczny 23](#_Toc63171432)

[Rysunek 23 Symbole przynależności jednostek 24](#_Toc63171433)

[Rysunek 24 Modyfikatory symboli 24](#_Toc63171434)

[Rysunek 25 Umieszczenie symbolu 26](#_Toc63171435)

[Rysunek 26 Zaznaczanie wielu symboli 27](#_Toc63171436)

[Rysunek 27 Górne menu edytora z włączoną opcją rysowania 27](#_Toc63171437)

[Rysunek 28 Podgląd planszy 27](#_Toc63171438)

[Rysunek 29 Okno dialogowe edycji znaku 28](#_Toc63171439)

[Rysunek 30 Schemat warstw edytora 29](#_Toc63171440)

[Rysunek 31 Schemat generowania symbolu 29](#_Toc63171441)

[Rysunek 32 Aplikacja mobilna 30](#_Toc63171442)

[Rysunek 33 Diagram przypadków użycia aplikacji mobilnej 31](#_Toc63171443)

[Rysunek 34 Struktura aplikacji mobilnej 32](#_Toc63171444)

## Spis tabel

[Tabela 1 Typy jednostek 25](#_Toc63171445)

[Tabela 2 Wskaźniki wielkości jednostki 26](#_Toc63171446)

## Bibliografia

Auth0. (2020). *JWT.* Pobrano 12 16, 2020 z lokalizacji JWT IO: https://jwt.io/

Block, G., Cibraro, P., Felix, P., Dierking, H. i Miller, D. (2016). *Nowoczesne API. Ewoluujące aplikacje sieciowe w technologii ASP.NET.* Helion.

Google. (2021). *Angular.* Pobrano 01 04, 2021 z lokalizacji Dokumentacja Angular: https://angular.io/guide/architecture

Matulewski, J. (2016). *MVVM i XAML w Visual Studio 2015.* Helion.

Microsoft. (2020). *ASP .NET.* Pobrano 12 3, 2020 z lokalizacji Dokumentacja Microsoft: https://docs.microsoft.com/pl-pl/aspnet/core/introduction-to-aspnet-core

Microsoft. (2020). *C#.* Pobrano 12 09, 2020 z lokalizacji https://docs.microsoft.com/pl-pl/dotnet/csharp/getting-started/

Microsoft. (2020). *Entity Framework.* Pobrano 11 24, 2020 z lokalizacji Dokumentacja Microsoft: https://docs.microsoft.com/pl-pl/ef/

Microsoft. (2020). *WPF.* Pobrano 12 09, 2020 z lokalizacji https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf

Microsoft. (2020). *Wprowadzenie do .NET.* Pobrano 11 30, 2020 z lokalizacji Dokumentacja Microsoft: https://docs.microsoft.com/pl-pl/dotnet/core/introduction

Microsoft. (2021). *TypeScript.* Pobrano 01 04, 2021 z lokalizacji https://www.typescriptlang.org/docs

Microsoft. (2021). *Xamarin.* Pobrano 01 04, 2021 z lokalizacji Dokumentacja Microsoft: https://docs.microsoft.com/pl-pl/xamarin/get-started/what-is-xamarin

Paszyn, M., Kordowski, M. i Zalewski, W. (2016). *Koncepcja Obrony Terytorialnej w Polsce.* Warszawa: Narodowe Centrum Studiów Strategicznych.

ppłk dr Wrona, A., ppłk dr Kulczycki, M., mjr dypl. Grabowski, K. i kpt. dypl. Dąbrowski, C. (2005). *Znaki Wojskowe w graficznych dokumentach dowodzenia pododdziałów i oddziałów - Poradnik.* WSOWLąd wewn.

Soloch, P., Żurawski, P. i Dryblak, Ł. (2015). *Organizacje proobronne w systemie bezpieczeństwa państwa.* Warszawa: Instytut Sobieskiego.

Wikipedia. (2020). *XAML*. Pobrano 01 09, 2021 z lokalizacji Wikipedia: https://pl.wikipedia.org/wiki/Extensible\_Application\_Markup\_Language