# TEAMS

Dokumentacja projektowa PZSP2

Wersja 1

06.11.2025r.

Semestr 25z

Zespół nr 1 w składzie:

Damian D'Souza Kamil Marszałek Michał Suski Michał Szwejk

Mentor zespołu: dr Marcin Szlenk
Właściciel tematu: prof. dr hab. Robert Nowak

# Spis treści

# 1 Wprowadzenie

## 1.1 Cel projektu

Celem projektu jest wykorzystanie API aplikacji MS Teams, aby umożliwić tworzenie własnych rozwiązań.

## 1.2 Wstępna wizja projektu

Najważniejszą częścią projektu jest biblioteka umożliwiająca komunikowanie się z API. Biblioteka będzie umożliwiała wykonanie konkretnych requestów zapewniając przy tym odpowiedni poziom abstrakcji. Użytkownik będzie mógł z niej skorzystać w celu automatyzacji procesów. W celu demonstracji możliwości biblioteki powstanie aplikacja (command-line interface lub terminal-user interface). Projekt zostanie zaimplementowany w języku GO.

# 2 Metodologia wytwarzania

W pierwszej kolejności zaimplementowana zostanie biblioteka wraz z testami jednostkowymi i odpowiednią dokumentacją. Kiedy pokrycie testami będzie wystarczające zostanie rozpoczęty proces wytwarzania drugiego modułu systemu – aplikacji terminalowej demonstrujące jej działanie.

W procesie wytwarzania oprogramowania zostaną wykorzystane mechanizmy serwisu GitHub: GitHub Actions (pipeline), Issues.

# 3 Analiza wymagań

## 3.1 Wymagania użytkownika i biznesowe

Użytkownik wykorzystując bibliotekę lub aplikację terminalową będzie mógł:

- Automatycznie utworzyć kanały dla zespołów projektowych definiując listę z podziałem na grupy.

- Ustawić automatyczne wysłanie wiadomości na kanale o konkretnej porze (np. w celu przypomnienia o spotkaniu).

- Automatycznie wysłać parametryzowane wiadomości prywatne do wielu wskazanych użytkowników.

- Automatycznie wysłać parametryzowane wiadomości na wiele wskazanych kanałów.

- Przejrzeć wszystkie wiadomości otrzymane na kanałach zespołów we wskazanym oknie czasowym.

( Lista funkcjonalności z dużym prawdopodobieństwem rozrośnie się w czasie implementacji projektu. )

## 3.2 Wymagania funkcjonalne i niefunkcjonalne

W celu obsługi wymagań użytkownika biblioteka zapewnia dostęp do następujących funkcji:

1. Zarządzanie zespołami teams.
2. Zarządzanie kanałami zespołu.
3. Zarządzanie członkami zespołu.
4. Pobieranie i wysyłanie wiadomości na kanale.
5. Zarządzanie chatami.
6. Pobieranie wiadomości nieodczytanych.

( Lista obsłużonych funkcjonalności może się rozrosnąć )

## 3.3 Przypadki użycia

Jedynym aktorem jest Użytkownik.

# Przypadki biznesowe

**PB1. Wysłanie wiadomości parametryzowanych.**

Scenariusz główny:

1. Użytkownik definiuje szablon wiadomości i parametry.
2. Użytkownik adresuje wiadomość.
3. Użytkownik zleca wysłanie wiadomości.
4. System wysyła wiadomości do podanych adresatów.

Scenariusz alternatywny – Adresat nie istnieje

1-3. Jak w scenariuszu głównym.

4. System powiadamia o nieistniejącym adresacie.

**PB2. Utworzenie kanałów dla zespołów projektowych.**

Scenariusz główny:

1. Użytkownik wskazuje zespół, gdzie utworzyć kanały.
2. Użytkownik podaje sparametryzowaną nazwę kanałów i listę członków.
3. Użytkownik zleca utworzenie kanałów.
4. System tworzy kanały dla zdefiniowanych zespołów.
5. System dodaje do utworzonych kanałów członków według listy.

**PB3. Odczytanie nowych wiadomości.**

Scenariusz główny:

1. Użytkownik wskazuje okno czasowe.
2. System wyświetla listę nieodczytanych wiadomości.

Scenariusz alternatywny – brak nieodczytanych wiadomości w wskazanym oknie czasowym:

1. Jak w scenariuszu głównym.
2. System powiadamia o braku nieodczytanych wiadomości.

**PB4. Wysłanie wiadomości według ustalonego harmonogramu.**

Scenariusz główny:

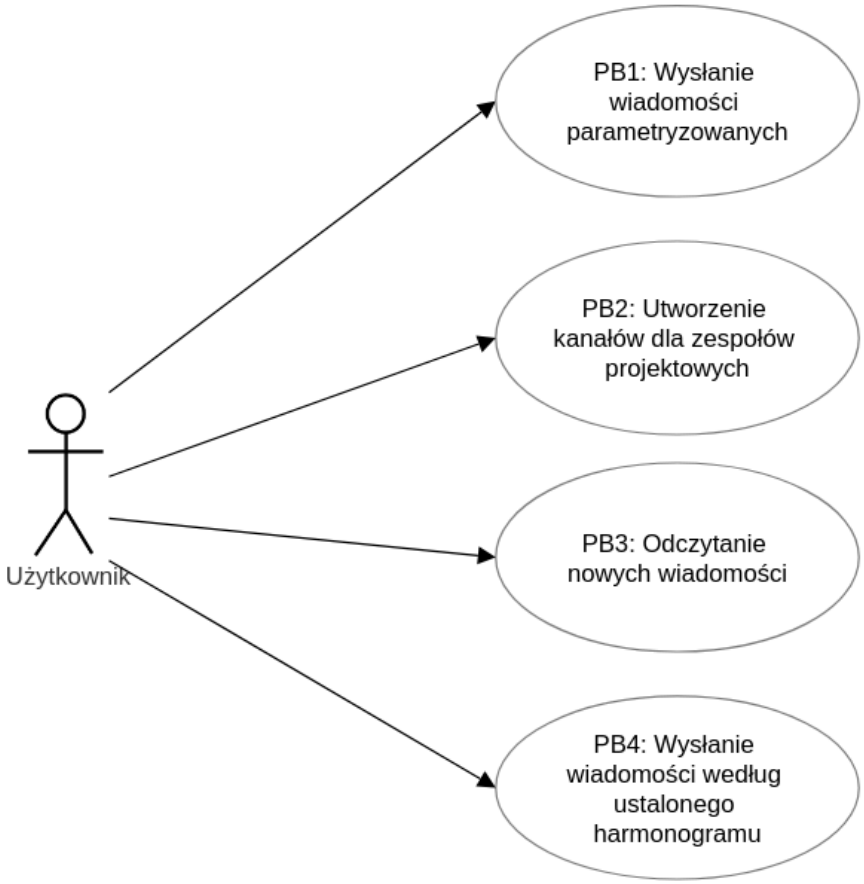
1. Użytkownik definiuje wiadomości i ich harmonogram.

2. Użytkownik adresuje wiadomości.
3. Użytkownik zleca wysłanie wiadomości.
4. System wysyła wiadomości zgodnie ze wskazanym harmonogramem.

Scenariusz  alternatywny – Adresat nie istnieje

1-3. Jak w scenariuszu głównym.

4. System powiadamia o nieistniejącym adresacie.




# Przypadki systemowe

**FU1. Logowanie użytkownika.**

Funkcjonalność potrzebna do wszystkich PB.

Scenariusz główny:

1. System za pośrednictwem przeglądarki wyświetla okno logowania.

2. Użytkownik loguje się na platformę Teams.
3. System sprawdza, czy operacja logowania zakończyła się sukcesem.
4. System zamyka okno logowania i zapamiętuje token użytkownika.

Scenariusz alternatywny 1 – nieudane logowanie:

1-3. Jak w scenariuszu głównym.

4. Powrót do kroku 1 scenariusza głównego.

Scenariusz alternatywny 2 – użytkownik był już zalogowany:

1. System wczytuje zapamiętany token użytkownika.

**FU2. Wysyłanie wiadomości.**

Wspiera PB1 i PB4. Korzysta z FU1 i FU3.

Scenariusz główny:

1. Użytkownik zleca wysłanie wiadomości.
2. System ustala treść wiadomości, adresata, porę wysłania wiadomości odczytując je z szablonu.
3. System wysyła request POST do MS Graph API.
4. System potwierdza wysłanie wiadomości.

Scenariusz alternatywny – nieudana próba wysłania wiadomości:

1-3. Jak w scenariuszu głównym.

4. System powiadamia o nieudanej próbie wysłania wiadomości.

**FU3. Korzystanie z szablonów.**

Wspiera PB1, PB2 i PB4.

Scenariusz główny:

1. System określa, w jaki sposób zdefiniować dane do funkcjonalności.
2. Użytkownik wypełnia dane.
3. System odczytuje wypełniony szablon.
4. System powiadamia o poprawnym odczytaniu szablonu.

Scenariusz alternatywny – szablon niepoprawnie wypełniony:

1-3. Jak w scenariuszu głównym.

4. System powiadamia o nieudanym odczycie szablonu.

**FU4. Pobieranie wiadomości przychodzących.**

Wspiera PB3. Korzysta z FU1.

Scenariusz główny:

1. Użytkownik zleca pobranie wiadomości przychodzących.

2. System odczytuje specyfikację listy wiadomości, którą ma pobrać.
3. System pobiera wiadomości przychodzące na Teams odpytując MS Graph API przez request GET.
4. System potwierdza poprawne pobranie listy wiadomości.

**FU5. Tworzenie i zarządzanie kanałami zespołów.**

Wspiera PB2. Korzysta z FU1 i FU3.

Scenariusz główny:

1. Użytkownik wypełnia szablon korzystając z FU3.
2. System ustala, jakie kanały trzeba utworzyć.
3. System wysyła request POST do MS Graph API w celu utworzenia kanału.
4. System potwierdza utworzenie kanału.

Scenariusz alternatywny – kanał o tej nazwie już istnieje:

1-4. Jak w scenariuszu głównym.
5. System powiadamia o duplikacie nazwy kanału.

**FU6. Zarządzanie członkami kanałów.**

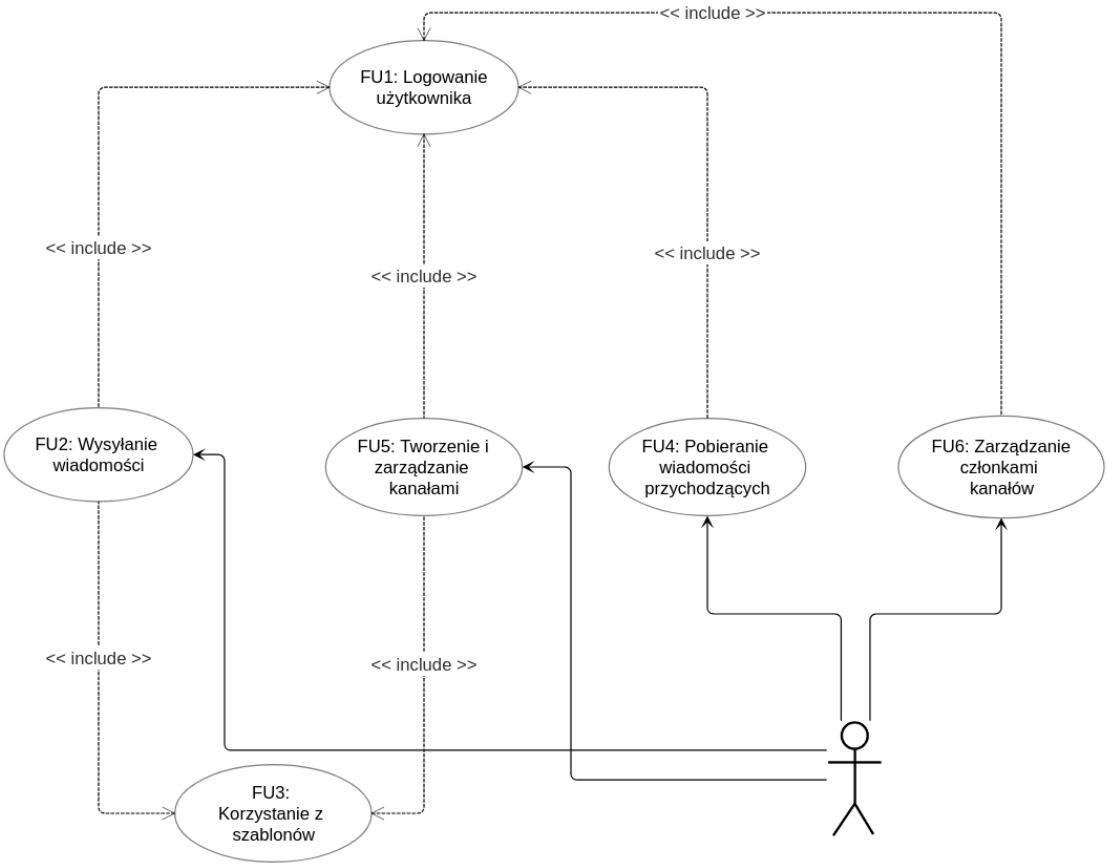
Wspiera PB2. Korzysta z FU1.

Scenariusz główny:

1. Użytkownik wskazuje kanał.
2. Użytkownik wskazuje listę członków do dodania.
3. System wysyła odpowiednie zapytania do MS Graph API.
4. System potwierdza poprawną aktualizację członków kanału.

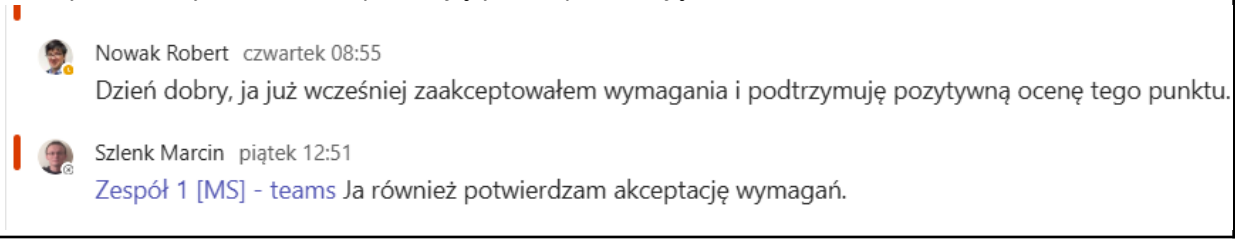
Scenariusz alternatywny – użytkownik nie istnieje:

1-3. Jak w scenariuszu głównym.
4. System powiadamia o błędnych kontach.




## 3.4 Potwierdzenie zgodności wymagań

*Należy wkleić wycinek ekranu pokazujący korespondencję z Mentorem i Właścicielem tematu.*




# 4 Definicja architektury

**perspektywa logiczna/funkcjonalna**

## Główne bloki funkcjonalne to:

1. **Główna biblioteka (Go):** Odpowiada za całą komunikację z MS Teams, walidację, cache i bezpieczeństwo.
2. **Interfejs użytkownika TUI / CLI (Go):** Warstwa interakcji z użytkownikiem, odpowiedzialna za wyświetlanie tabel, kolorów i pobieranie komend z klawiatury.
3. **Binding (Python):** Cienka warstwa, która tylko przekazuje żądania do biblioteki głównej i formatuje odpowiedzi.

**[plan struktury systemu – model komponentów, modułów, serwisów**

# Biblioteka:

1. Na zewnątrz udostępniany jest główny obiekt klient agregujący odpowiednie serwisy do obsługi zespołów, kanałów i czatów. Możliwe jest też utworzenie serwisów pojedynczo w przypadku, kiedy użytkownik nie chce korzystać ze wszystkich funkcjonalności.
2. Serwisy odpowiedzialne są za konwersję typów wygodnych do użytkowania, na takie, które są zrozumiałe dla API MS Teams. (np. nazwę zespołu / email użytkownika na ich ID).
3. Kolejna warstwa operacji odpowiada za obsługę bardziej złożonych zapytań, odczytywanie i przetwarzanie odpowiedzi od Microsoftu i obsługę otrzymanych błędów.
4. Na tę warstwę można nałożyć dekorator, który dołącza obsługę cache'a. Wtedy we wskazanym przez użytkownika pliku zapisywane są mapy (np. nazwa zespołu - ID) w celu szybszego rozwiązywania podanych referencji oraz redukcji ilości requestów do API MS Teams.
5. Najgłębsza warstwa - API odpowiada za obsługę pojedynczych zapytań, ich wysyłanie oraz odbieranie odpowiedzi.

# Aplikacja (TUI / CLI)

1. Umożliwia interakcję w trybie tekstowym i interaktywnym
2. Tryb CLI pozwala na szybkie użycie w skryptach i automatyzacji
3. Tryb TUI upraszcza nawigację po komendach
4. Oba tryby korzystają z tej samej logiki i biblioteki

# Binding w Pythonie

Biblioteka jest dostępna również w Pythonie, gdzie jej obsługa jest analogiczna jak w Go.

- **Most w Go**

5. Obudowuje główną bibliotekę w obsługę komunikacji przez stdin/stdout za pomocą formatu json.

- **Główna część w Pythonie**

1. Biblioteka uruchamia w ramach działania skompilowany do wersji binarnej proces z biblioteką w Go z dodatkowym mostem i komunikuje się z nim za pomocą stdin/stdout.

2. Główny klient odpowiada za inicjalizację tego zdefiniowanego w Go oraz za opakowanie komunikatów w odpowiedni format.
3. Serwisy zespołów / kanałów/ chatów pozwalają na wygodne wywoływanie funkcji z głównej biblioteki.

**Zastosowane szablony architektoniczne:**

CLI z interfejsem TUI zostało zaprojektowane w oparciu o wzorzec Model–View–Update (MVU).

Model odpowiada za stan aplikacji, View za renderowanie interfejsu w terminalu, a Update obsługuje zdarzenia i aktualizacje stanu. Taki podział upraszcza logikę, zwiększa czytelność kodu i ułatwia dalszy rozwój narzędzia.

**oraz perspektywa sprzętowa (tzw. deployment): systemy operacyjne,**

**Perspektywa sprzętowa (deployment):**

1. **Systemy operacyjne:** Windows / Linux
2. **Deployment:**

- Bibliotekę Go można zaimportować wewnątrz kodu Go podając link do repozytorium.
- Aplikację CLI można zainstalować za pomocą polecenia go install
- Binding w Pythonie można zainstalować za pomocą narzędzia **pip**.

**Serwer aplikacyjny:**

Brak serwera, proces jest lokalny u użytkownika.

**System bazy danych i inne mechanizmy utrwalania danych:**

Cache i zarządzanie tokenem uwierzytelniania, szerzej opisane w kolejnym punkcie dokumentacji.

**System raportowania:**

Biblioteka zwraca odpowiednie informacje o błędach wewnątrz kodu jako odpowiedź wywołania funkcji.

**Mechanizmy zarządzania:**

Zalecaną formą korzystania z biblioteki jest korzystanie z pliku .env zapewniającego odpowiednie parametry dla działania aplikacji. Przykładowy plik oraz jego obsługa znajduje się w repozytorium projektu. Użytkownik sam odpowiada za podanie pliku, w którym cache będzie zapisany - może go też w każdym momencie wyczyścić.

**Mechanizmy bezpieczeństwa:**

Bezpieczne przechowywanie tokenu, brak wrażliwych danych w cache, szyfrowana komunikacja sieciowa z API MS Teams. Sekcja szerzej opisana w punkcie **10** dokumentacji.

# 5 Dane trwałe

Aplikacja nie przechowuje danych trwałych, nie jest to wymagane do prawidłowego działania systemu, ponieważ ciało zapytań (nazwa zespołu, nazwa kanału, wiadomość etc.) do API Microsoft Graph jest

uzupełniane przez użytkownika wywołującego zapytanie. Wobec tego nie została przygotowana relacyjna baza danych.

Jedynymi danymi, które są przechowywane w systemie są związane z warstwą cachującą. W systemie został opracowany prosty system pamięci podręcznej zrealizowanej za pośrednictwem pliku JSON, która utrwala niezbędne mapowania wymagane do poprawnego działania systemu (np. *TeamRef -> UUID)*. Takie rozwiązanie umożliwia wydajne i szybkie rozwiązywanie referencji, ponieważ nie jest wymagane każdorazowe wykonywanie zapytania do API Microsoft Graphu, aby dowiedzieć się jaki identyfikator ma dany obiekt. Cache uzupełniany jest podczas wywołań różnych requestów, np. wywołując metodę *ListChannels* w pamięci podręcznej pojawią się wpisy dotyczące mapowań nazw kanałów na unikalne identyfikatory. Czyszczony jest za każdym razem gdy okaże się, że dany wpis jest niepoprawny - skupiamy się na przede wszystkim na poprawności danych.

# 6 Specyfikacja analityczna i projektowa

PZSP-Teams
Utworzyliśmy organizację na githubie i tam utworzyliśmy osobne repozytoria na:
- biblioteka w Go,
- binding biblioteki w Pythonie,
- narzędzie CLI.

Rozdzieliliśmy kod na kilka repozytoriów, ponieważ użytkownik raczej będzie chciał wykorzystać albo Go albo Pythona. Narzędzie CLI też jest osobnym bytem, który wprowadza dużo zależności, które nie pasowałyby semantycznie do biblioteki. Podział kodu na kilka repozytoriów pozwolił nam na lepsze korzystanie z potoków. (dokładniejszy opis potoków dostępny jest tutaj**)**

Do realizacji projektu wykorzystaliśmy język Go do utworzenia biblioteki oraz narzędzia CLI. Miał to być jedyny język używany w projekcie. Natomiast na prośbę prowadzącego biblioteka umożliwiająca łatwe korzystanie z API Microsoft Graph została również udostępniona w Pythonie.

Biblioteka korzysta głównie z biblioteki Microsoftu do budowania i wysyłania zapytań:
- Microsoft Graph SDK

Narzędzie CLI wykorzystuje biblioteki:
- framework do tworzenia TUI - bubbletea.
- ułatwiającą tworzenie interfejsów command-line - cobra
- logger - charm-logger

Do realizacji ciągłej integracji używamy GitHub Actions.

**Diagram klas biblioteki:**

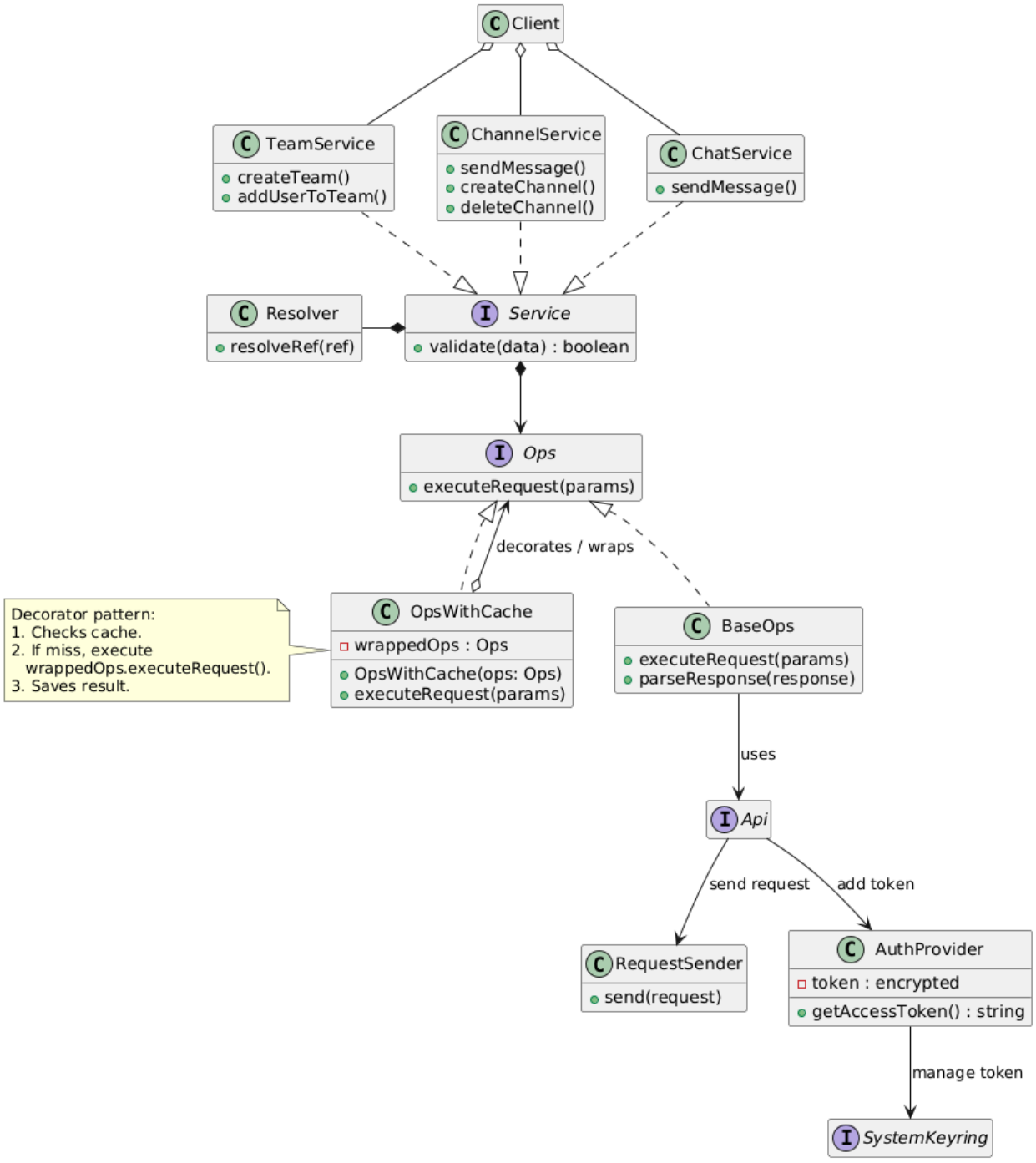


**Diagram klas CLI:**

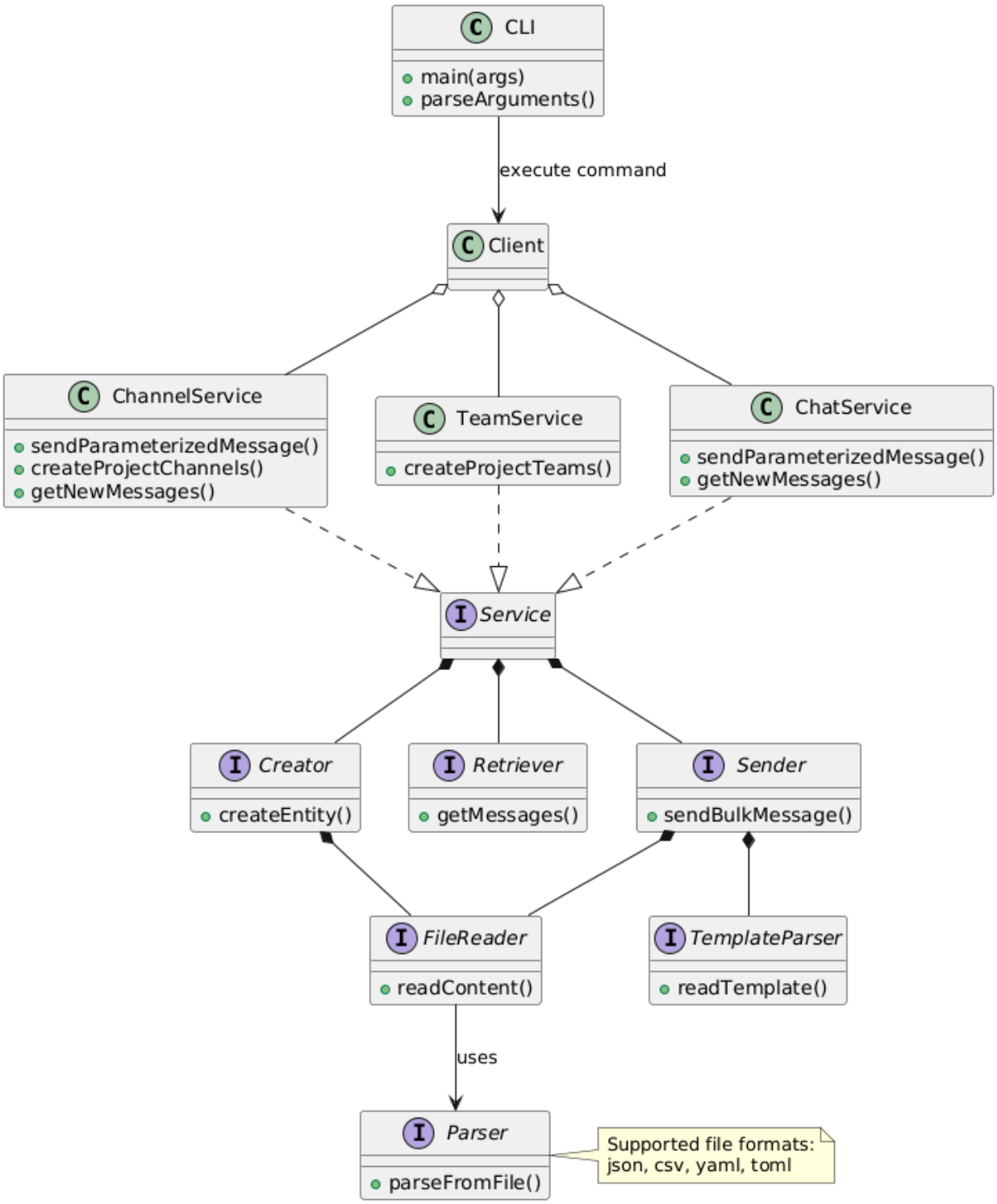
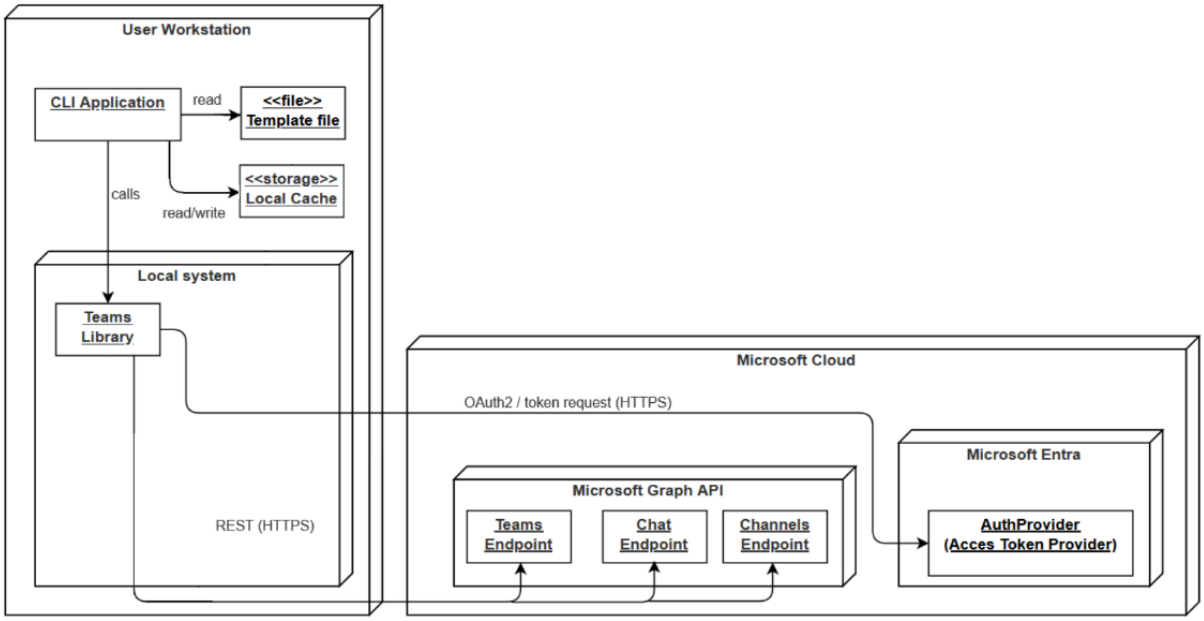


**Diagram wdrożenia:**




**Statystyki: liczba plików, linie kodu, liczba testów jednostkowych**

| | Biblioteka | CLI | Python Binding |
|---|---|---|---|
| **Liczba plików** | **139** | **181** | **65** |
| **Liczba linii kodu** | **18325** | **15117** | **6323** |
| **Liczba testów** | **971** | **450** | **44 (integracyjne)** |

# 7 Projekt standardu interfejsu użytkownika

Interfejs użytkownika w projekcie ogranicza się do TUI. W związku z jego prostotą i ograniczeniami nie był on projektowany żadnym specjalizowanym narzędziem.

# 8 Specyfikacja testów

W bibliotece i narzędziu command-line zrealizowane zostały testy jednostkowe które badają realizowane funkcjonalności. Testy są atomowe i działają w izolacji - pozostałe elementy systemu zostały zamockowane z wykorzystaniem biblioteki *gomock*.

W pythonie przeprowadzono testy integracyjne, które badają jak wszystkie funkcjonalności działają połączone w jednym zorganizowanym systemie. W przypadku tych testów jedynym problemem jest integracja z API Microsoft Graph, które jest czarną skrzynką - jego zachowania nie możemy w żaden

sposób przewidzieć. W celu zaadresowania tego problemu zewnętrzna usługa także została sztucznie spreparowana, rzeczywiste requesty nie są wysyłane.

Sytuacją wyjątkową, jaka może zaistnieć podczas interakcji z Microsoft Graph API jest niepoprawne działanie zewnętrznego serwisu, który może być w danej chwili niedostępny. W celu minimalizacji ryzyka niedostarczenia requesta zaimplementowany został niestandardowy sender, który wysyła zapytania co jakiś czas w określonych interwałach.

Główną miarą jakości testów jest miara pokrycia decyzji. Dla poszczególnych pakietów w bibliotece uzyskano następujące wyniki:

| Pakiet w bibliotece | Pokrycie decyzji |
|---|---|
| channels | 98.4% |
| config | 0.0% |
| adapter | 100.0% |
| api | 17.7% |
| auth | 0.0% |
| cacher | 87.2% |
| mentions | 100.0% |
| pepper | 100.0% |
| resolver | 87.6% |
| resources | 0.0% |
| sender | 57.3% |
| util | 100.0% |
| models | 0.0% |
| setup | 0.0% |
| search | 0.0% |
| teams | 100.0% |

Dla pakietów *models, resources, search* pokrycie wynosi 0%, ponieważ zawierają one wyłącznie definicje struktur używanych w całym projekcie. Z kolei dla pakietów takich jak *api, auth, setup* pokrycie testów

jednostkowych jest niskie, ponieważ realizują one funkcje zdefiniowane przez zewnętrzne biblioteki Microsoftu odpowiedzialne za wysyłanie requestów, autoryzację i inicjalizację klienta. Wobec tego postanowiono, że testy zostaną ograniczone do minimum, ponieważ naszym celem nie jest analiza zewnętrznych usług.

Dla pakietów w CLI uzyskano następujące wyniki:

| | |
|---|---|
| logger | 73.8% |
| utils | 100.0% |
| core | 62.4% |
| initializers | 0.0% |
| pepper | 92.9% |
| chats | 89.7% |
| file_readers | 96.7% |
| templates | 95.7% |
| channels | 94.9% |
| teams | 64.2% |
| client | 0.0% |
| handlers | 81.3% |
| formatters | 85.0% |
| config | 83.6% |
| app | 93.8% |
| cmd | 14.0% |
| tui | 16.6% |

Dla pakietów client i initializers pokrycie testami jest niskie, ponieważ zawierają one głównie inicjalizatory już przetestowanych komponentów (np. loggera) oraz elementy agregujące interfejsy.

W przypadku pakietów cmd i tui pokrycie również jest niskie, ponieważ składają się one przede wszystkim z deklaracji komend oraz warstwy UI obsługiwanej przez odpowiednie komponenty interfejsu. Sama logika komend oraz walidacja parametrów są współdzielone i znajdują się w pakiecie app

Realizacja testów akceptacyjnych:

1. wysyłanie wiadomości na kanały teams - ✓
2. pobieranie nieodczytanych wiadomości teams - ✓
3. automatyczne tworzenie kanałów dla grup projektowych - ✓

Wszystkie testy akceptacyjne zostały spełnione (ich szczegółową postać można podejrzeć na repozytorium związanym z dokumentacją).

# *9 Wirtualizacja/konteneryzacja*

W projekcie nie stosujemy konteneryzacji ani wirtualizacji, ponieważ aplikacja jest kompilowana statycznie w języku Go. Dzięki temu pliki binarne zawierają wszystkie wymagane zależności i mogą być uruchamiane bezpośrednio na docelowym systemie, bez potrzeby izolowania środowiska wykonawczego. CLI wykorzystuje bibliotekę wyłącznie jako statyczny binarny artefakt, pełni rolę warstwy wywołań.

# 10 Bezpieczeństwo

Wrażliwą częścią aplikacji jest pozyskanie tokena, dzięki któremu użytkownik może wykonywać operacje w Microsoft Graph API. Logowanie możliwe jest na dwa sposoby: interaktywnie - zostanie otwarta przeglądarka i Microsoft zapewni bezpieczne logowanie lub poprzez mechanizm Device Code Flow - w konsoli zostanie wypisana jednorazowa fraza, która umożliwi logowanie nawet z systemów niemających bezpośredniego dostępu do przeglądarki, możliwe jest nawet zalogowanie na innym urządzeniu dzięki temu mechanizmowi. Po zalogowaniu aplikacja otrzymuje dwa tokeny: access - do wykonywania requestów na bieżąco oraz refresh - do opcjonalnego odświeżenia. Nasza aplikacja bezpiecznie przechowuje uzyskane tokeny w systemowym keyringu. Innym potencjalnym zagrożeniem było przechowywanie mapowanie emaili użytkowników na ich microsoftowe ID. Nie chcemy aby ktoś widział w cache maile użytkowników bezpośrednio. Aby temu przeciwdziałać, użytkownik ustawia pepper do hashowania maili. Pepper jest przechowywany bezpiecznie w keyringu.

# 11 Podręcznik użytkownika

Dla każdego z repozytoriów przygotowano osobną dokumentację, która jest dostępna pod następującymi adresami:

1. biblioteka - https://pzsp-teams.github.io/lib/
2. cli - https://pzsp-teams.github.io/teams-cli/
3. python binding - https://pzsp-teams.github.io/lib-python/

Dokumentacja jest podręcznikiem użytkownika, odpowiada na pytania co i w jaki sposób można osiągnąć korzystając z systemu.

# 12 Podręcznik administratora

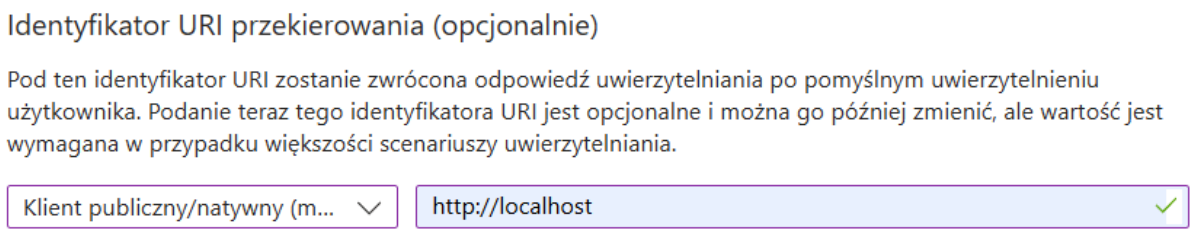
**Instalacja:**

1. **Biblioteka Go:** go get github.com/pzsp-teams/lib@latest

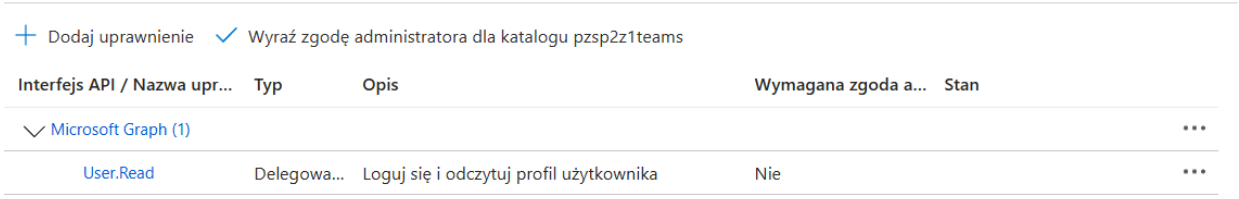
2. **CLI:** go install github.com/pzsp-teams/teams-cli@latest lub pobranie binarki z repozytorium
3. **Python binding:** pip install teams-lib-pzsp2-z1

**Rejestracja aplikacji:**

1. W serwisie Microsoft Entra należy kliknąć “Nowa rejestracja”.
2. Wprowadzić nazwę oraz konfigurację:




3. Odczytać CLIENT_ID i TENANT_ID i uzupełnić plik .env
4. Przy pierwszym użyciu użytkownik zostanie poproszony o wyrażenie zgody na dostęp do odpowiednich scope’ów podanych w .env
5. W przypadku błędów z logowaniem konieczne może być wyrażenie zgody na logowanie w Microsoft Entra aplikacji w zakładce “Uprawnienia interfejsu API”




# 13 Podsumowanie

**Krytyczna analiza osiągniętych wyników, mocne i słabe strony:**

Założenia projektu zostały w pełni osiągnięte. Wszystkie testy akceptacyjne zostały spełnione. Użytkownik jest w stanie wysyłać sparametryzowane wiadomości na wiele kanałów i chatów, wyświetlić listę nowych otrzymanych wiadomości oraz automatycznie utworzyć zespoły dla zdefiniowanych grup projektowych. Biblioteka jest dobrze udokumentowana i przetestowana. Ponadto udało się zrealizować jej binding w innym języku programowania - Pythonie.

Słabą stroną projektu jest jego mała użyteczność praktyczna. Aby korzystać z bibliotek lub TUI konieczne jest zarejestrowanie własnej aplikacji w serwisie Microsoft Entra, oraz uzyskanie dostępu do odpowiednich uprawnień dotyczących korzystania z API MS Teams. Do tego konieczne jest uzyskanie zgody administratora, co w prawdziwych organizacjach jest problematyczne.

**Możliwe kierunki rozwoju:**

- Rozszerzenie biblioteki o nowe funkcjonalności, np. zarządzanie kalendarzem, plikami zespołów, sharePointem.
- Dodanie bindingów wspierających inne języki.

# 14 Bibliografia

[Dokumentacja Microsoft Graph API]

| | |
|---|---|
| Zatwierdzam dokumentację. | ……<br>……..<br>Data i podpis Mentora |