**Temat projektu: Komunikator Internetowy**

Skład grupy: Andrzej Świderski, Bartosz Prusaczyk, Tomasz Milianowicz, Bartosz Giełdon, Maksymilian Głowacki

Opis projektu:

Aplikacja na komputery osobiste i urządzenia mobilne z systemem android przeznaczona do komunikacji z innymi użytkownikami.

1. **Założenia funkcjonalne:**

- Użytkownik ma możliwość założenia konta,

- wyszukiwanie użytkowników po nazwie konta (nick),

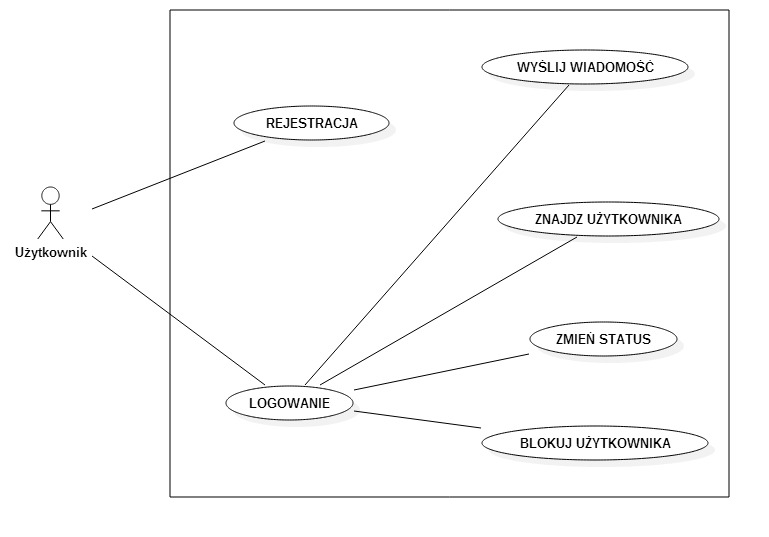
- wysyłanie wiadomości tekstowych,

- wyświetlanie statusu użytkownika (np. dostępny, zajęty, niedostępny),

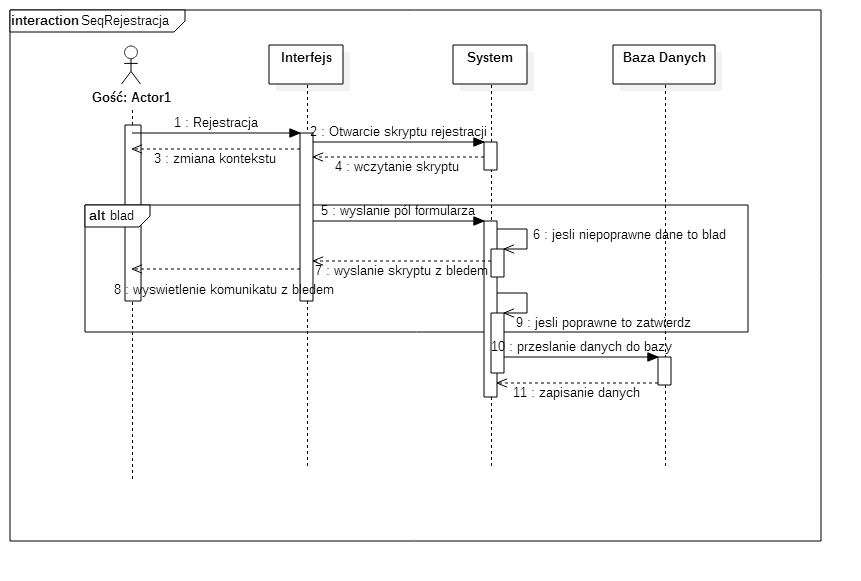
- wyświetlanie daty i godziny wysłanych wiadomości,

- blokowanie użytkowników,

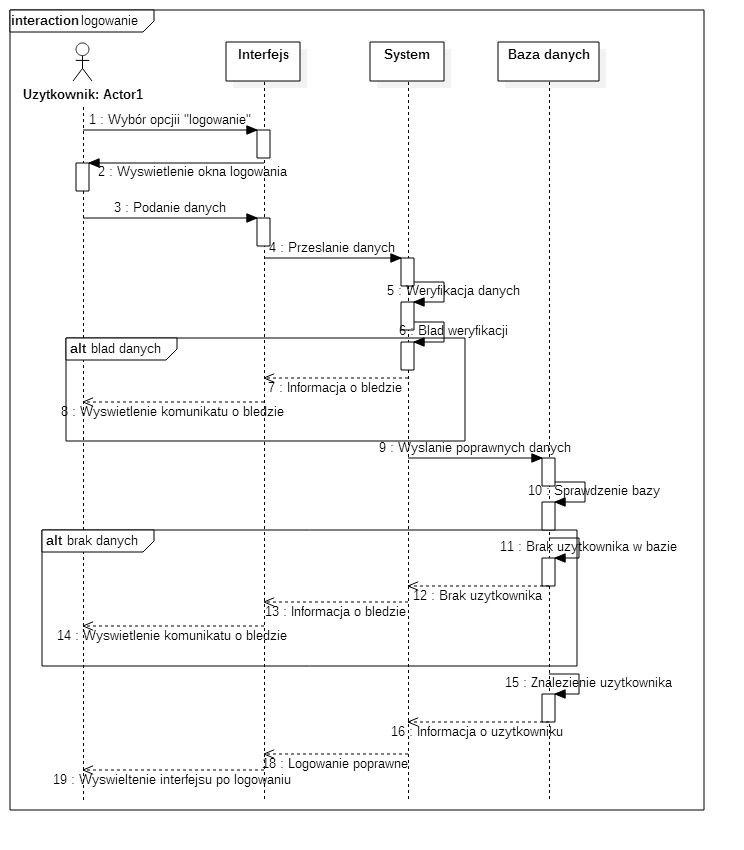
1. **Założenia niefunkcjonalne:**
2. Użyteczność – Widok wiadomości będzie wyświetlany od góry w dół (na dole będą znajdować się najnowsze wiadomości), wybór użytkownika z wyświetloną nazwą i statusem dostępności.
3. Niezawodność - Czas awarii nie może być dłuższy niż 3 godziny. Czas wysyłania wiadomości nie może być dłuższy niż 3 minuty.
4. Wydajność - Czas wysyłania wiadomości nie może być dłuższy niż 3 minuty. Czas zmiany statusu nie może być dłuższy niż 5 minut. Czas logowania do aplikacji nie może trwać dłużej niż 3 minuty.
5. Wspieralność – Aplikacja na telefon będzie obsługiwana przez system Android KitKat 4.4 lub nowszy, a aplikacja na komputery osobiste będzie wspierana i obsługiwana przez system Windows 7 lub nowszy.
6. **Diagram przypadków użycia:**



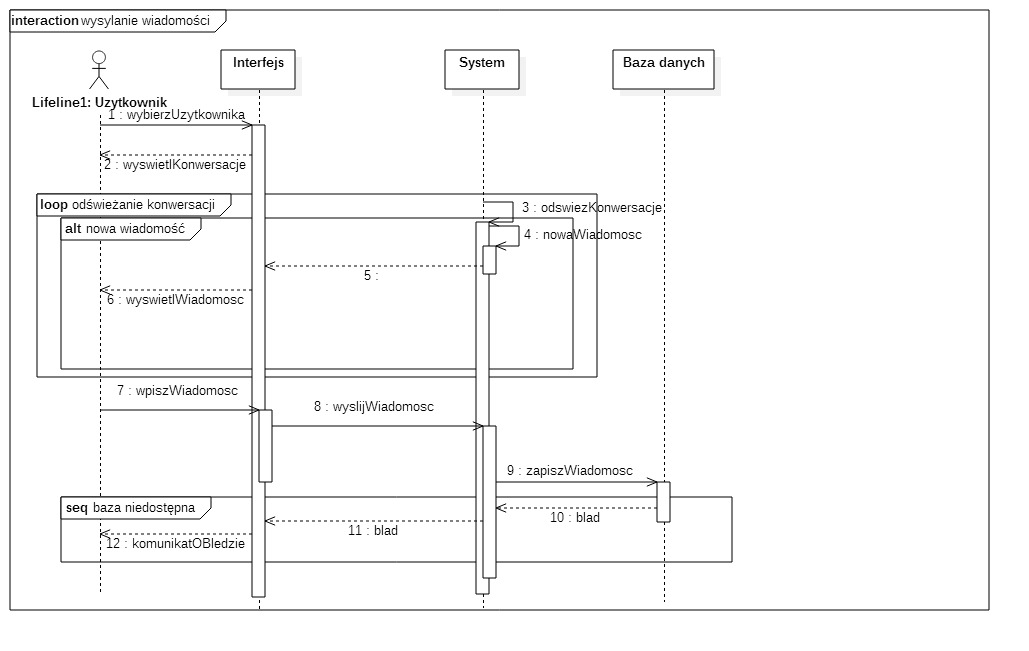
1. **Diagramy sekwencji:**
2. Rejestracja



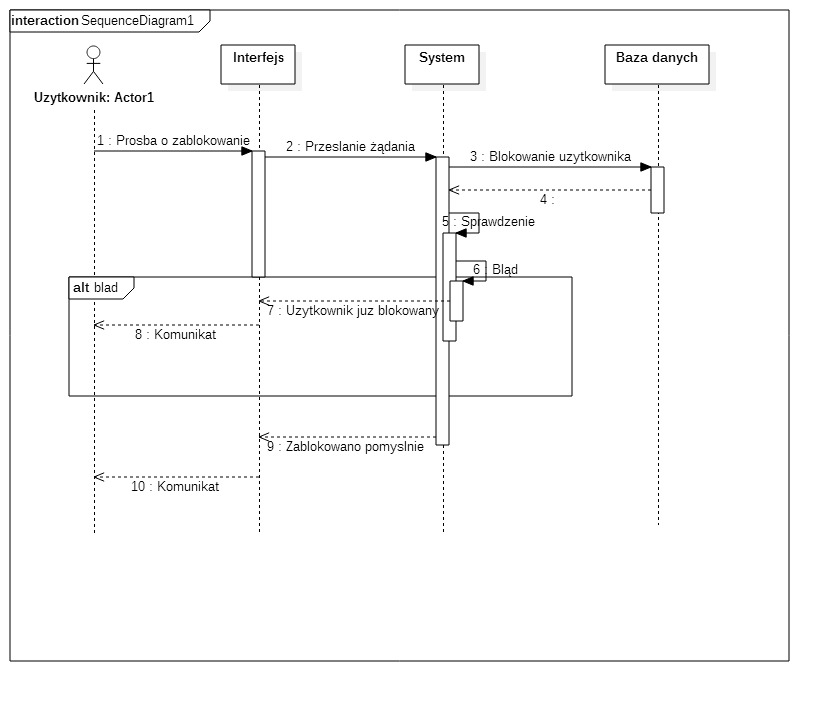
1. Logowanie



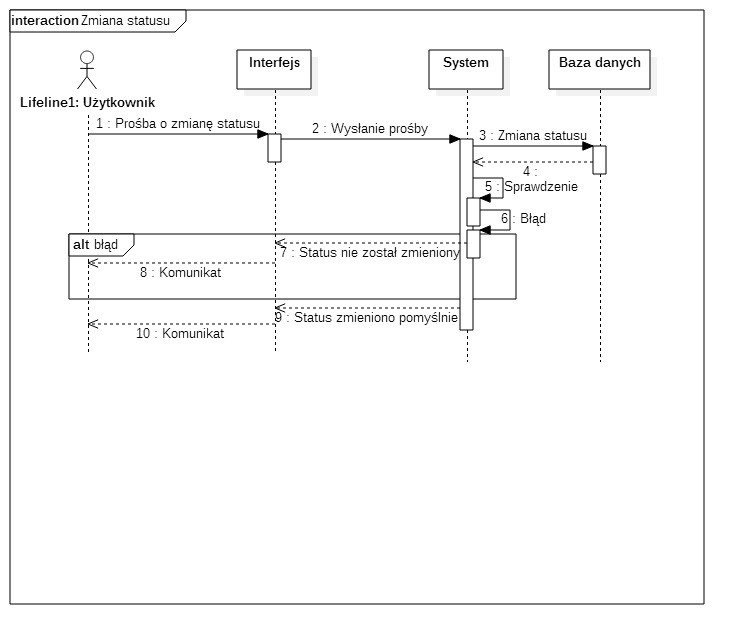
1. Wysyłanie wiadomości



1. Blokowanie użytkownika



1. Zmiana statusu



1. **Technologie i metodyka:**

Środowiska programistyczne użyte w projekcie:

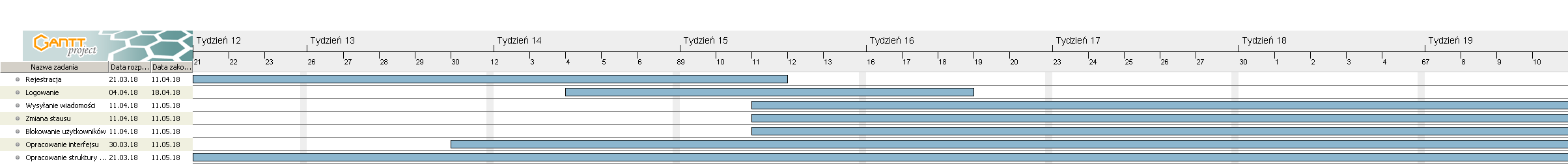
Visual studio 2015 – środowisko te posłuży nam do stworzenia aplikacji desktopowej w języku C#

Android Studio - środowisko te posłuży nam do stworzenia aplikacji desktopowej w języku JAVA

StarUML - program ten posłuży nam do stworzenia diagramów w języku UML.

GanttProject – program do wykonania harmonogramu zadań.

1. **Podział zadań i harmonogram**



Rejestracja (Andrzej Świderski) – 21.03 – 11.04

Logowanie (Maksymilian Głowacki) – 4.04 – 18.04

Wysyłanie wiadomości (Bartosz Prusaczyk) – 11.04 – 11.05

Zmiana statusu (Bartosz Giełdon) – 11.04 – 11.05

Blokowanie użytkowników (Tomasz Milianowicz) – 11.04-11.05

Opracowanie interfejsu (wszyscy) – 30.03 – 11.05

Opracowanie struktury bazy danych (wszyscy) -21.03 – 11.05

**7. Opis kodu**

1. Aplikacja desktopowa

# Klasa Konwersacja

Przestrzeń nazw: komunikator.WysylanieWiadomosci

Klasa reprezentująca konwersację pomiędzy użytkownikami.

## Pola:

*public string* ***login*** – login zalogowanego użytkownika.

*public string* ***adresat*** – login użytkownika, z którym jest prowadzona konwersacja.

*private const string* ***daneBazy*** – dane serwera MySQL.

## Konstruktory:

*public* ***Konwersacja****(string login, string adresat)* – tworzy egzemplarz klasy.

Argumenty:

***login*** – login zalogowanego użytkownika.

***adresat*** – login użytkownika, z którym prowadzona jest konwersacja.

## Metody:

*private static string* ***znajdzIdUzytkownika****(string login)* – zwraca ID w bazie danych danego użytkow­nika.

Argumenty:

***login*** – login użytkownika, którego ID ma zostać znalezione.

*private static string* ***znajdzUzytkownikaPoId****(string id)* – zwraca login użytkownika, do którego jest przypisane dane ID w bazie danych.

Argumenty:

**id** – ID w bazie danych, do którego jest przypisany szukany użytkownik.

*public string* ***wyslijWiadomosc****(string tresc)* – wysyła wiadomość do adresata. Zwraca czas serwera, w którym wiadomość została dostarczona.

Argumenty:

***tresc*** – treść wysyłanej wiadomości.

*public List<Wiadomosc>* ***wczytajWiadomosci****()* - zwraca wszystkie wiadomości, które zostały wysłane pomiędzy zalogowanym użytkownikiem i adresatem.

*public bool* ***sprawdzCzySaNoweWiadomosci****()* - zwraca **true**, jeśli zalogowany użytkownik ma nieod­czytane wiadomości, w przeciwnym przypadku **false**.

*public static bool* ***sprawdzCzySaNoweWiadomosci****(string login)* – zwraca **true**, jeśli dany użytkownik ma nieodczytane wiadomości, w przeciwnym przypadku **false**.

Argumenty:

***login*** – login użytkownika, którego wiadomości są sprawdzane.

*public List<Wiadomosc>* ***odswiezKonwersacje****()* - zwraca nieodczytane wiadomości od adresata.

*public static bool* ***znajdzUzytkownika****(string szukanyLogin)* – zwraca **true**, jeśli użytkownik o podanym loginie istnieje, w przeciwnym wypadku **false**.

Argumenty:

***szukanyLogin*** – login szukanego uzytkownika.

*public static void* ***dodajKontakt****(string login1, string login2)* – dodaje nowy kontakt.

Argumenty:

***login1***, ***login2*** – loginy użytkowników z nowego kontaktu.

*public static List<string>* ***zaladujKontakty****(string login)* – zwraca kontakty danego użytkownika.

Argumenty:

***login*** – login użytkownika, którego kontakty mają być zwrócone.

*public static void* ***usunKontakt****(string login1, string login2)* – usuwa kontakt.

***login1***, ***login2*** – loginy użytkowników z kontaktu, który ma być usunięty.

*public static Dictionary<string, int>* ***wyswietlPowiadomieniaONowychWiadomosciach****(string login)* – zwraca odwzorowanie zawierające kontakty danego użytkownika i liczbę odpowiadających im nieod­czytanych wiadomości (zwraca tylko kontakty, od których są nieodczytane wiadomości)

Argumenty:

***login*** – login użytkownika, którego liczba nieodczytanych wiadomości ma być zwrócona.

# Klasa Konwersacja.Wiadomosc

Przestrzeń nazw: komunikator.WysylanieWiadomosci

Reprezentuje wiadomość.

## Właściwości:

*public string* ***tresc*** – treść wiadomości.

*public string* ***uzytkownik*** – użytkownik, który wysłał wiadomość.

*public string* ***data*** – data i czas wysłania wiadomości.

# Klasa Konwersacja.Kontakt

Przestrzeń nazw: komunikator.WysylanieWiadomosci

Reprezentuje kontakt.

## Właściwości:

*public string* ***login*** – login użytkownika, do którego należy kontakt.

*public int* ***nieodczytaneWiadomosci*** – liczba nieodczytanych wiadomości, od danego kontaktu.

## Metody:

*public override string* ***ToString****()* - zwraca login użytkownika, do którego należy kontakt, jeśli nie ma od tego kontaktu żadnych nieodczytanych wiadomości, w przeciwnym przypadku zwraca login użyt­kownika, do którego należy kontakt, i liczbę nieodczytanych wiadomości w nawiasie.

# Klasa KonwersacjaOkno

Przestrzeń nazw: komunikator.WysylanieWiadomosci

Dziedziczy po: ***Window***

Reprezentuje okno, w którym jest wyświetlona konwersacja.

## Pola:

*private Konwersacja* ***k*** – konwersacja, która jest wyświetlana.

*private Timer* ***odswiezacz*** – obiekt odpowiedzialny za odświeżanie konwersacji

## Konstruktory:

*public* ***KonwersacjaOkno****(string loginZalogowanego, string loginAdresata)* – tworzy nowe okno.

Argumenty:

***loginZalogowanego*** – login zalogowanego użytkownika.

***loginAdresata*** – login użytkownika, z którym odbywa się konwersacja.

## Metody:

*private void* ***OnOdswiezEvent****(object o)* – jeśli to konieczne, odświeża konwersację.

Argumenty:

***o*** – konwersacja, którą należy odświeżyć.

*private void* ***dodajNoweWiadomosci****()* - wyświetla nadesłane wiadomości.

## Metody powiązane ze zdarzeniami:

*private void* ***wyslijPrzycisk\_Click****(object sender, RoutedEventArgs e)* – wysyła wpisaną wiadomość, ak­tywowane po kliknięciu na przycisk do wysyłania wiadomości.

*private void* ***wiadomoscTekst\_KeyUp****(object sender, KeyEventArgs e)* - wysyła wpisaną wiadomość, aktywowane po wciśnięciu klawisza Enter.

# Klasa WyborRozmowcy

Przestrzeń nazw: komunikator.WysylanieWiadomosci

Dziedziczy po: ***Window***

Reprezentuje okno, w którym są wyświetlone kontakty.

## Pola:

*private string* ***zalogowanyUzytkownik*** – login zalogowanego użytkownika.

*private Timer* ***odswiezacz*** – obiekt odpowiadający za odświeżanie liczby nieodczytanych wiadomości.

## Konstruktory:

*public* ***WyborRozmowcy****()* - tworzy nowe okno.

## Metody:

*private void* ***OnOdswiezEvent****(object o)* – jeśli to konieczne, odświeża liczbę nieodczytanych wiado­mości.

Argumenty:

***o*** – dowolny obiekt (zalecane przekazanie wartości **null**).

*private void* ***odswiezKontaktyIWyzerujNoweWiadomosci****()* - zeruje liczbę nieodczytanych wiadomo­ści dla każdego kontaktu.

*private void* ***poinformujONowychWiadomosciach****()* - wyświetla liczbę nieodczytanych wiadomości.

## Metody powiązane ze zdarzeniami:

*private void* ***dodajUzytkownika\_Click****(object sender, RoutedEventArgs e)* – dodaje nowy kontakt, ak­tywowane po kliknięciu na przycisk dodania nowego kontaktu.

*private void* ***szukanyUzytkownik\_KeyDown****(object sender, KeyEventArgs e)* - dodaje nowy kontakt, aktywowane po wciśnięciu klawisza Enter.

*private void* ***otworz\_Click****(object sender, RoutedEventArgs e)* – otwiera okno z konwersacją, aktywo­wane po kliknięciu na przycisk otwarcia konwersacji.

*private* ***void usun\_Click(****object sender, RoutedEventArgs e)* – usuwa kontakt, aktywowane po kliknię­ciu na przycisk usunięcia kontaktu.

Aplikacja mobilna

|  |  |  |  |
| --- | --- | --- | --- |
| Numer wersji | Tytuł | Uczestnicy | Data |
| 0.1 | Opis projektu, założenia funkcjonalne i niefunkcjonalne. | Andrzej Świderski  Bartosz Prusaczyk  Tomasz Milianowicz  Bartosz Giełdon  Maksymilian Głowacki | 14.03.2018r. |
| 0.2 | Diagram przypadków użycia, diagram sekwencji. Technologia, metodyka, podział pracy, harmonogram | Andrzej Świderski  Bartosz Prusaczyk  Tomasz Milianowicz  Bartosz Giełdon  Maksymilian Głowacki | 21.03.2018r. |
| 0.3 | Dodanie opisu kodu programu – klasy Konwersacja, KonwersacjaOkno i WyborRozmowcy | Bartosz Prusaczyk | 9.04.2018r. |

Repozytorium: PWSIP\_2018\_PZ2\_GR2