

# MinISpace - Aplikacja do śledzenia aktywności życia studenckiego

Sebastian Prokop, Bartosz Olszewski, Maciej Pluta,  
Piotr Rowicki, Adrian Rudź

11 stycznia 2024

## Spis treści

<b>1</b>	<b>Opis Projektu - cel</b>	<b>3</b>
<b>2</b>	<b>Funkcjonalności systemu</b>	<b>4</b>
2.1	Student	4
2.1.1	wydarzenia	4
2.1.2	Administracja	4
2.1.3	Przyjaciele	4
2.1.4	Historie użytkownika	4
2.2	Organizator wydarzenia	5
2.2.1	Tworzenie i edytowanie wydarzenia	5
2.2.2	Komunikacja z uczestnikami	5
2.2.3	Monitorowanie frekwencji	5
2.2.4	Analiza danych	5
2.2.5	Administracja	5
2.2.6	Historie organizatora wydarzenia	5
2.3	Znajomy	6
2.3.1	Dodatkowy filtr wydarzeń	6
2.3.2	Nowa możliwość zapraszania	6
2.3.3	Szersze informacje na temat wydarzenia	6
2.3.4	Historie znajomego	7
2.4	Administrator systemu	7
2.4.1	Nadzór użytkowników	7
2.4.2	Nadzór systemu	7
2.4.3	Analiza systemu	7
2.4.4	Historie administratora	7
<b>3</b>	<b>Przypadki użycia</b>	<b>9</b>
3.1	Wydarzenia	9
3.2	Administracja systemem	10
3.3	Znajomi	11
<b>4</b>	<b>Diagram klas</b>	<b>12</b>
<b>5</b>	<b>Struktura Systemu</b>	<b>13</b>
5.1	Diagramy aktywności	13
5.1.1	przeglądanie wydarzeń	13
5.1.2	Activity diagram: dodawanie posta	14
5.1.3	Activity diagram: rozpatrywanie problemu użytkownika	15
5.1.4	Activity diagram: tworzenie wydarzenia	17
5.1.5	Activity diagram: zapisywanie na wydarzenia	20
5.2	Diagramy stanów	21
5.2.1	State Diagram: wydarzenie	21
5.2.2	State diagram: post	22

5.2.3	State diagram: problem użytkownika . . . . .	23
<b>6</b>	<b>Komunikacja Systemu</b>	<b>24</b>
6.1	Diagramy sekwencji . . . . .	24
6.1.1	Zaproszenie znajomego . . . . .	24
6.1.2	Dodawanie wydarzenia . . . . .	25
6.1.3	Edycja zawartości . . . . .	27
6.1.4	Przeglądanie zgłoszeń . . . . .	28
6.1.5	Aktualizacja konta . . . . .	29
6.1.6	Wysyłanie powiadomień o nadchodzących wydarzeniach . . . . .	30
6.1.7	Przeglądanie wydarzeń . . . . .	31
6.1.8	Zapisanie na wydarzenie . . . . .	32
6.1.9	Interakcja z wydarzeniem . . . . .	33
6.1.10	Dodawanie posta . . . . .	34
6.1.11	Statystyki wydarzenia . . . . .	35
6.2	Schematy komunikacji W RAML . . . . .	36

# 1 Opis Projektu - cel

MiNISpace to aplikacja stworzona do śledzenia aktywności życia studenckiego. Umożliwia ona studentom przeglądanie, zapisywanie się na wydarzenia, tworzenie własnych aktywności oraz interakcję z innymi studentami. Aplikacja skupia się na kulturalnych, dydaktycznych i społecznych wydarzeniach dostępnych dla studentów.

## Ogólna koncepcja systemu:

### 1. Rejestracja i Konta Użytkowników:

- System umożliwia studentom rejestrację i tworzenie kont użytkowników.

### 2. Przeglądanie Wydarzeń:

- Użytkownicy mogą przeglądać dostępne wydarzenia z podziałem na kategorie, daty, lokalizacje itp.
- System zapewnia filtrowanie i wyszukiwanie, aby ułatwić znalezienie interesujących aktywności.

### 3. Zapisywanie się na Wydarzenia:

- Studenci mają możliwość zapisywania się na wybrane wydarzenia.
- Organizatorzy wydarzeń mogą zarządzać dostępnością zapisów, na przykład limitem miejsc.

### 4. Powiadomienia i Interakcje:

- Użytkownicy otrzymują powiadomienia o nadchodzących wydarzeniach, zaproszeniach od znajomych oraz interakcjach z ich aktywnościami.
- Istnieje możliwość komentowania i oceniania wydarzeń, oraz postów z nimi związanych.

### 5. Organizacja Wydarzeń:

- Organizatorzy mogą tworzyć nowe wydarzenia i dodawać do nich posty.
- Organizatorzy mają dostęp do statystyk i list uczestników.
- Organizatorzy mogą powiadamiać mailowo osoby deklarujące chęć udziału w wydarzeniu o jego szczegółach.

### 6. System Znajomych:

- Użytkownicy mogą nawiązywać znajomości - wysyłać zaproszenia do znajomych. Przy wydarzeniach można zobaczyć kto z listy znajomych zamierza wziąć w nim udział.

### 7. Bezpieczeństwo i Moderacja:

- System zawiera mechanizmy bezpieczeństwa, umożliwiające zgłaszanie problemów, blokowanie użytkowników oraz moderację treści.
- Za moderację odpowiedzialni są administratorzy systemu, którzy rozpatrują zgłoszenia.

System MiNISpace ma na celu ułatwienie organizacji wydarzeń studenckich różnej skali, poprzez usprawnienie sposobu ogłaszania wydarzeń i ich promocji.

## 2 Funkcjonalności systemu


### 2.1 Student

Głównym odbiorcą aplikacji jest właśnie student. Z tego powodu zakres działań jaki może podjąć jest szeroki. Podzielić go można na 3 kluczowe elementy: Interakcja z wydarzeniami, z administracją oraz z innymi użytkownikami.

#### 2.1.1 wydarzenia

Interakcja z wydarzeniami obejmuje możliwość przeglądania tych dostępnych oraz zapisywać się na nie, znając liczbę zapisanych użytkowników. Dostępna jest również możliwość komentowania i reagowania na interesujące go wydarzenia przed i po ich odbyciu, aby kolejno wyrazić zainteresowanie przed i opinie po tymże wydarzeniu. Dodatkowo ma on możliwość otrzymywania powiadomień o zbliżających się dla niego wydarzeniach. Schemat tych funkcjonalności można zobaczyć na diagramie użycia na rysunku 1.

#### 2.1.2 Administracja

Student ma możliwość komunikacji z administracją przez aplikację w dwóch przypadkach: aby zgłosić błąd aplikacji, lub aby zgłosić innego użytkownika jeżeli ten nie przestrzega regulaminu. Schemat tychże funkcjonalności dostępny jest na diagramie 2 

#### 2.1.3 Przyjaciele

aplikacja umożliwia Studentowi zapraszanie innych użytkowników do znajomych. Więcej informacji o interakcji z nimi znajduje się w sekcji Przyjaciel


#### 2.1.4 Historie użytkownika

Poniżej przedstawione zostały historie użytkownika dotyczące wszystkich wyżej wymienionych funkcjonalności

US1: Jako student chcę przeglądać dostępne wydarzenia, aby znaleźć interesujące mnie aktywności. Acceptance criteria checklist

- (a) Jakie wydarzenia są dla mnie widoczne ?
- (b) Jakie informacje o wydarzeniu są dla mnie widoczne ?

Non Functional Requirements ? 

-  (a) ile wydarzeń jestem w stanie zobaczyć w jednej sesji ?
- (b) Czy mogę odświeżyć wyświetlane się wydarzenia ?
- (c) Czy mam pewność że wydarzenie jest dla mnie dostępne ?
- (d) Czy tematyka wydarzenia jest jasno przekazana ?

US2: Jako student chcę mieć możliwość zapisania się na wydarzenie, aby w nim uczestniczyć. Acceptance criteria checklist:

- (a) Czy mogę zapisać się na dowolnie wybrane wydarzenie?
- (b) Czy mogę zmienić wybrane wydarzenie?
- (c) Czy mogę zobaczyć szczegóły wydarzenia przed zapisaniem się?

Non Functional Requirements:

- (a) Czy najbliższe wydarzenia są wyróżnione?
- (b) Czy zapisanie się na wydarzenie wymaga więcej niż 1 kroku?
- (c) Czy inni studenci widzą moje zainteresowanie wydarzeniem?

US3: Jako student chcę otrzymywać powiadomienia o nadchodzących wydarzeniach, aby ich nie przegapić.

US4: Jako student chcę reagować na wydarzenia przed ich rozpoczęciem, aby wyrazić swoje zainteresowanie.

- US5: Jako student chcę oceniać wydarzenia po ich odbyciu, aby podzielić się swoją opinią.
- US6: Jako student chcę komentować wydarzenia, aby zachęcić innych użytkowników do dyskusji.
- US7: Jako student chcę widzieć liczbę osób zapisanych na wydarzenie, aby znać jego skalę.
- US8: Jako student chcę zapraszać innych użytkowników do znajomych, aby dzielić się z nimi swoją aktywnością.
- US9: Jako student chcę mieć możliwość zgłaszania napotkanych błędów, aby powiadomić o nich administratora.
- US10: Jako student chcę mieć możliwość zgłaszania użytkowników niestosujących się do regulaminu, aby powiadomić o nich jednocześnie organizatora wydarzenia oraz administratora systemu.

## **2.2 Organizator wydarzenia**

Organizator wydarzenia jak sama nazwa wskazuje odgrywa kluczową rolę w naszym systemie, jednak zakres jego działań jest znacznie szerszy niż mogło by się wydawać.

### **2.2.1 Tworzenie i edytowanie wydarzenia**

Organizator jest odpowiedzialny za utworzenie nowego wydarzenia w systemie. Musi dostarczyć wszystkie niezbędne informacje, takie jak nazwa, data, godzina, miejsce, opis, zdjęcia i inne szczegóły związane z imprezą. Ma za zadanie dbać o kompletność i dokładność informacji dotyczących wydarzenia. Może aktualizować szczegóły oraz dodawać nowe informacje.

### **2.2.2 Komunikacja z uczestnikami**

Organizator utrzymuje kontakt z uczestnikami poprzez system, dostarczając im ważnych informacji dotyczących wydarzenia. Komunikacja może odbywać się za pomocą wiadomości e-mail lub powiadomień wewnętrznych (postów).

### **2.2.3 Monitorowanie frekwencji**

System umożliwia organizatorowi śledzenie liczby zarejestrowanych uczestników oraz monitorowanie frekwencji w czasie rzeczywistym. Dzięki temu organizator może dostosować plan imprezy, jeśli zajdzie taka potrzeba.

### **2.2.4 Analiza danych**

Organizator ma dostęp do danych dotyczących rejestracji, frekwencji i innych statystyk związanych z wydarzeniem. Po zakończeniu imprezy organizator może zbierać opinie uczestników, oceny i komentarze. Analiza tych danych pozwala na ocenę sukcesu wydarzenia oraz dostarcza informacji zwrotnej na przyszłość.

### **2.2.5 Administracja**

Organizator podobnie jak student może komunikować się z administracją w dwóch przypadkach. Aby zgłosić błąd związany z funkcjonalnością systemu lub zgłosić uczestnika swojego wydarzenia.

### **2.2.6 Historie organizatora wydarzenia**

US1: Jako organizator wydarzenia chcę mieć możliwość dodania nowego wydarzenia, aby promować aktywności skierowane do studentów.

Acceptance criteria checklist:

- (a) Czy mogę dodać współorganizatorów wydarzenia ?
- (b) Czy mogę zapisać fragment formularza tworzenia wydarzenia, aby móc później do niego wrócić ?
- (c) Czy mogę edytować istniejące wydarzenie ?
- (d) Czy mogę usunąć istniejące wydarzenie ?

Non Functional Requirements:

- (a) Tworzenie wydarzenia powinno być responsywne, nawet przy obsłudze dużej liczby użytkowników i wydarzeń.
- (b) System powinien obsługiwać rosnącą liczbę wydarzeń i użytkowników bez istotnych problemów z wydajnością, zapewniając płynne doświadczenia użytkowników.
- (c) Wyświetlane są przyjazne dla użytkownika komunikaty błędów dotyczące walidacji pól podczas tworzenia wydarzenia, z sugestiami dotyczącymi poprawy.

US2: Jako organizator wydarzenia chcę mieć dostęp do statystyk uczestnictwa w moim wydarzeniu, aby ocenić jego popularność.

US3: Jako organizator wydarzenia chcę mieć dostęp do listy uczestników, aby mieć możliwość jej modyfikacji.

US4: Jako organizator wydarzenia chcę mieć możliwość określania grupy docelowej wydarzenia, aby dotarło do odpowiednich odbiorców.

US5: Jako organizator wydarzenia chcę mieć możliwość udostępnienia informacji o wydarzeniu w mediach społecznościowych, aby trafiło do większej grupy odbiorców.

US6: Jako organizator wydarzenia chcę mieć wgląd w feedback (opinie) od uczestników, aby poprawić jakość organizowanych wydarzeń.

US7: Jako organizator wydarzenia chcę mieć możliwość dodawania postów z nim związanych, aby móc przekazać niezbędne informacje.

Acceptance criteria checklist:

- (a) Czy mogę edytować i aktualizować istniejące posty związane z wydarzeniem?
- (b) Czy mogę określić datę i czas publikacji każdego posta?
- (c) Czy mogę załączać do posta linki, grafiki i filmy?

Non Functional Requirements:

- (a) Czy mogę wyświetlić podgląd postu, żeby wiedzieć jak będzie wyglądał po publikacji?
- (b) Czy podczas pisania posta podkreślane są literówki i błędy?
- (c) Czy mogę zapisać wersję roboczą posta, żeby dokończyć go i opublikować później?

US8: Jako organizator wydarzenia chcę mieć możliwość wysyłania wiadomości e-mail do uczestników, aby móc przysyłać im niezbędne do uczestnictwa spersonalizowane wejściówki.

## 2.3 Znajomy

Znajomy jest tak na prawdę studentem, natomiast został wydzielony w opisie systemu jako oddzielny aktor dla wyodrębnienia funkcjonalności. Można powiedzieć, że Student (użytkownik) zyskuje rolę znajomego w momencie dodania pierwszej osoby do listy znajomych. Status znajomego rozszerza możliwości jakie Student ma na samym początku tuż po rejestracji.

### 2.3.1 Dodatkowy filtr wydarzeń

Oprócz standardowych filtrów przeglądania wydarzeń, Znajomy może wyświetlić listę wydarzeń, którymi są zainteresowani jego znajomi (przynajmniej jeden z nich).

### 2.3.2 Nowa możliwość zapraszania

Student wybierający się na wydarzenie może przesłać zaproszenie do wybranych z pośród swoich znajomych. Otrzymują oni powiadomienie przekazujące kto zaprasza ich na jakie wydarzenie, wraz z linkiem do wydarzenia.

### 2.3.3 Szersze informacje na temat wydarzenia

Znajomy w szczegółach wydarzenia oprócz liczby zainteresowanych osób widzi również imienną listę jego znajomych, którzy biorą udział w tym wydarzeniu (domyślnie nie da się wyświetlić listy użytkowników zainteresowanych wydarzeniem).

### 2.3.4 Historie znajomego

- US1: Jako znajomy chcę móc przeglądać wydarzenia, w których uczestniczą moi znajomi, aby uczestniczyć w podobnych wydarzeniach co oni.
- US2: Jako znajomy chcę otrzymywać powiadomienia o aktywnościach moich znajomych, aby być na bieżąco z ich planami.
- US3: Jako znajomy chcę wysyłać moim znajomym zaproszenia do wydarzeń, aby zachęcić ich do uczestnictwa.
- US4: Jako znajomy chcę móc potwierdzić lub odrzucić zaproszenia na wydarzenia od znajomych, aby dać im informację zwrotną.
- US5: Jako znajomy chcę widzieć listę moich znajomych zapisanych na wydarzenie, aby dowiedzieć się, kto z nich się na nim pojawi.

## 2.4 Administrator systemu

Administrator systemu pełni bardzo ważną funkcję w aplikacji. Ma on przede wszystkim za zadanie nadzorować wszystkich użytkowników jak również działanie aplikacji.

Kontrola nad zachowaniem osób korzystających z serwisu jest elementem nieodłącznym od poprawnego, przejrzystego oraz dogodnego działania systemu. Począwszy od usuwania treści niezgodnych z powszechnymi zasadami kultury, etyki, moralności oraz regulaminu, kończąc na poprawnym i wydajnym utrzymaniu działania aplikacji.

Kolejną nieodzowną funkcjonalnością administratora jest zarządzanie użytkownikami - jego danymi oraz rozwiązywanie potencjalnych problemów napotkanych przez korzystających z aplikacji czy też nadawaniu odpowiednich uprawnień.

Następną rzeczą, nad którą administrator ma kontrolę to wprowadzanie łatek czy też aktualizacji. Ponadto, ma on również dostęp do danych analitycznych systemu.

### 2.4.1 Nadzór użytkowników

Interakcja użytkowników z aplikacją obejmuje opisywanie własnych doświadczeń czy też opinii. W związku z tym, aby panował porządek, administrator musi mieć możliwość kontroli nad zawartością umieszczaną w serwisie. Ponadto, musi mieć on również możliwość zarządzania kontem użytkowników, tak aby móc nadawać odpowiednie uprawnienia, czy też pomóc odzyskać konto.

### 2.4.2 Nadzór systemu

Administrator ma dostęp do konfiguracji środowiska/aplikacji, w celu wprowadzania zmian lub aktualizacji serwisu, na potrzeby poprawnego działania.

Co więcej, ma on dostęp do powiadomień o nieprawidłowym działaniu systemu, aby móc szybko reagować na ewentualne awarie lub im zapobiegać.

### 2.4.3 Analiza systemu

Aplikacja umożliwia administratorowi dostęp do danych analitycznych systemu dotyczących aktywności użytkowników, tak aby mógł on oceniać wydajność aplikacji.

### 2.4.4 Historie administratora

- US1: Jako administrator systemu chcę mieć możliwość usuwania nieodpowiednich treści, aby utrzymywać porządek na portalu.
- US2: Jako administrator systemu chcę mieć dostęp do panelu zarządzania użytkownikami, aby rozwiązywać problemy z ich kontami.
- Acceptance criteria checklist:

- (a) Czy mogę wysłać użytkownikowi kod resetujący hasło?
- (b) Czy mogę usunąć konto użytkownika, gdy podejrzewana jest jego kradzież?
- (c) Czy mogę sprawdzić aktywność użytkownika na jego koncie?

Non Functional Requirements:

- (a) Czy wiadomość z kodem dociera do użytkownika w ciągu minuty?
- (b) Czy usunięte konta są dla mnie widoczne?
- (c) Czy usunięte konta są zaznaczone na czerwono?
- (d) Czy aktywność użytkownika obejmuje cały okres od założenia konta?

US3: Jako administrator systemu chcę mieć dostęp do panelu zarządzania użytkownikami, aby nadawać im uprawnienia.

US4: Jako administrator systemu chcę być powiadamiany o nieprawidłowościach w działaniu systemu, aby szybko reagować na ewentualne awarie.

US5: Jako administrator systemu chcę mieć dostęp do danych analitycznych dotyczących aktywności użytkowników, aby ocenić wydajność aplikacji.

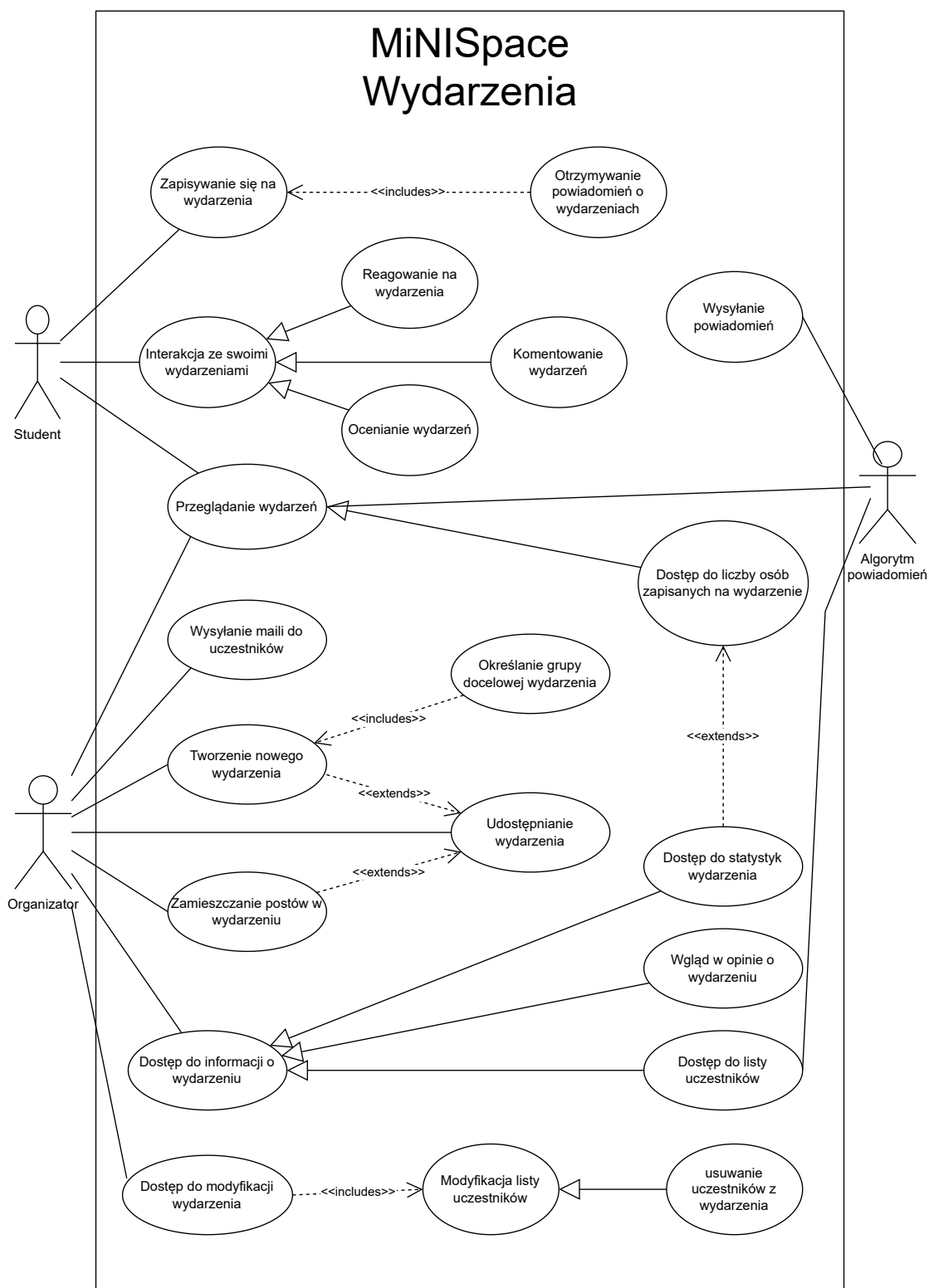
US6: Jako administrator systemu chcę mieć możliwość aktualizacji aplikacji, aby wprowadzać nowe funkcjonalności i poprawki.

US7: Jako administrator systemu chcę mieć możliwość odbierania dostępu użytkownikom do portalu, aby utrzymywać porządek na portalu.



### 3 Przypadki użycia

#### 3.1 Wydarzenia



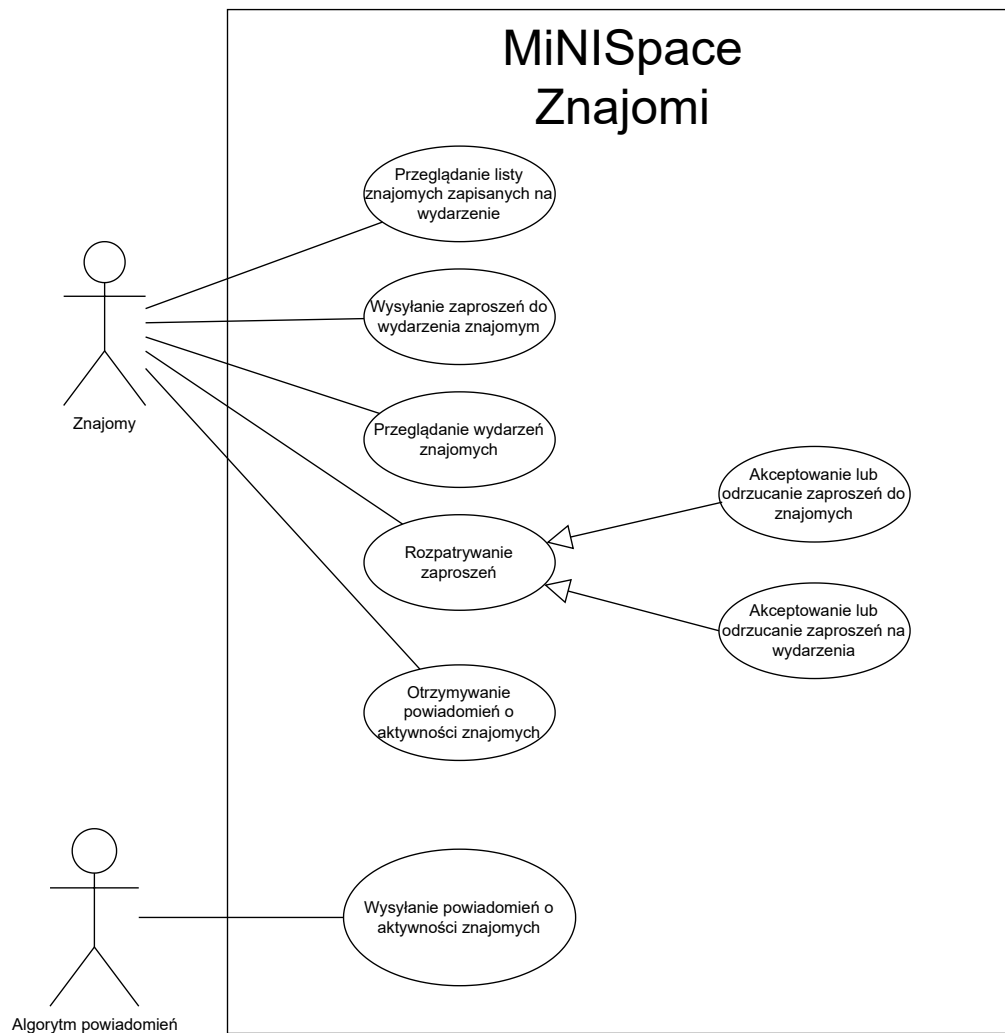
Rysunek 1: Use Case Diagram: Wydarzenia

### 3.2 Administracja systemem



Rysunek 2: Use Case Diagram: Administracja

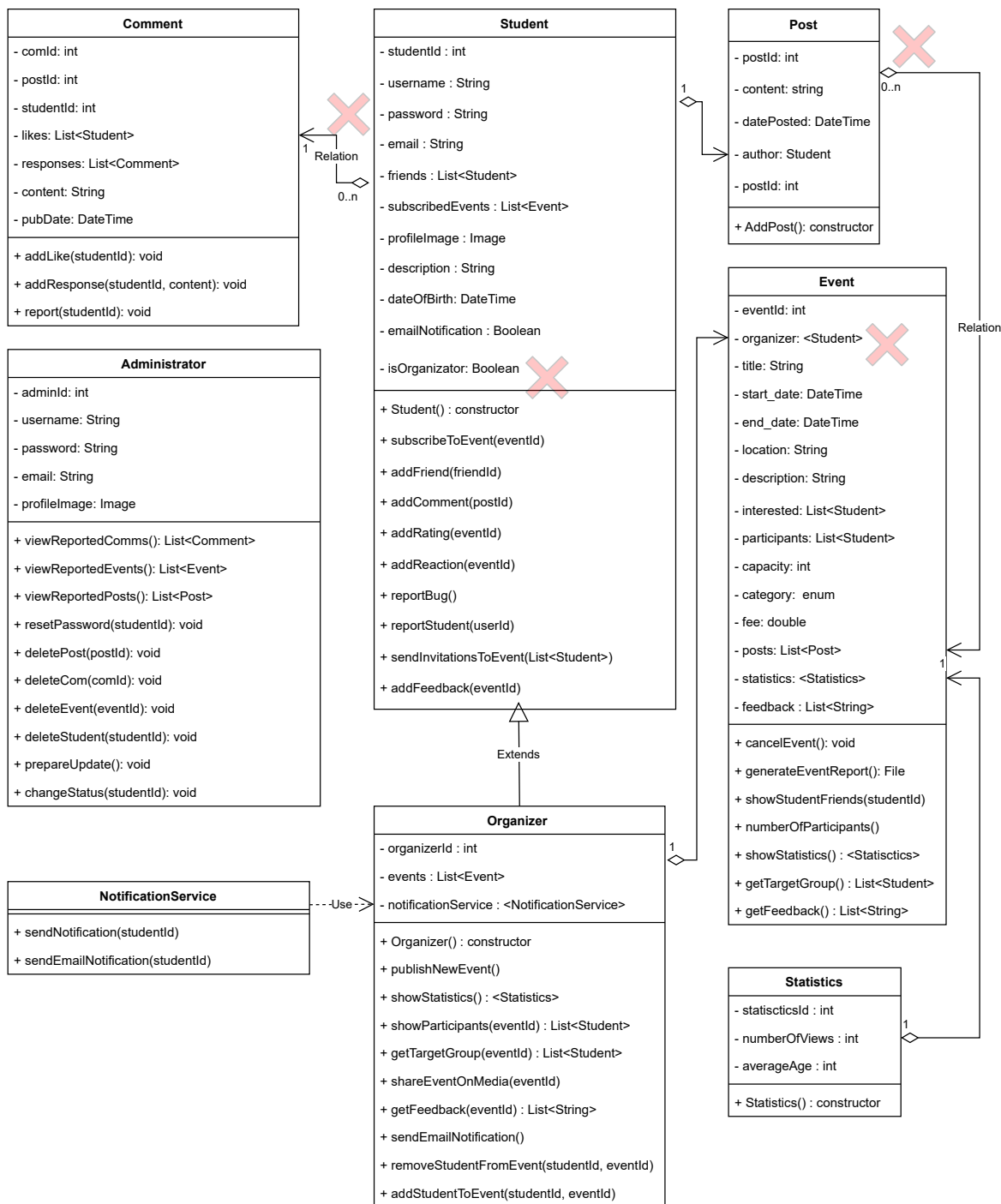
### 3.3 Znajomi



Rysunek 3: Use Case Diagram: Znajomi

## 4 Diagram klas

Poniższy diagram przedstawia schematy klas, z których korzysta, aby spełnić opisane wcześniej funkcjonalności



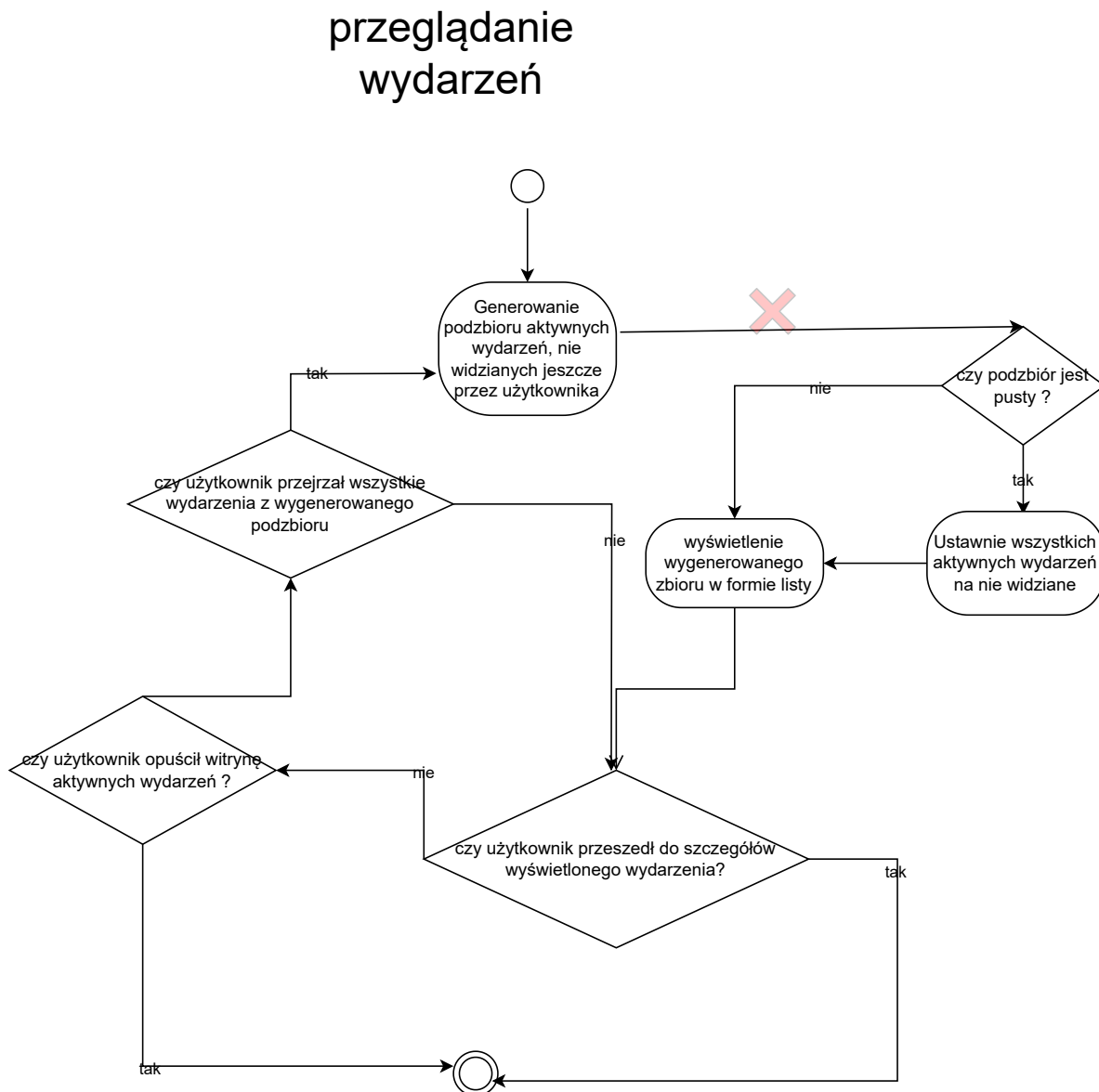
## 5 Struktura Systemu

Poniższe diagramy stanów i aktywności opisują wygląd i działanie komponentów systemu. Zawierają one informacje jak system będzie postrzegał obiekty, oraz warunki aby to postrzeganie uległo zmianie.

### 5.1 Diagramy aktywności

#### 5.1.1 przeglądanie wydarzeń

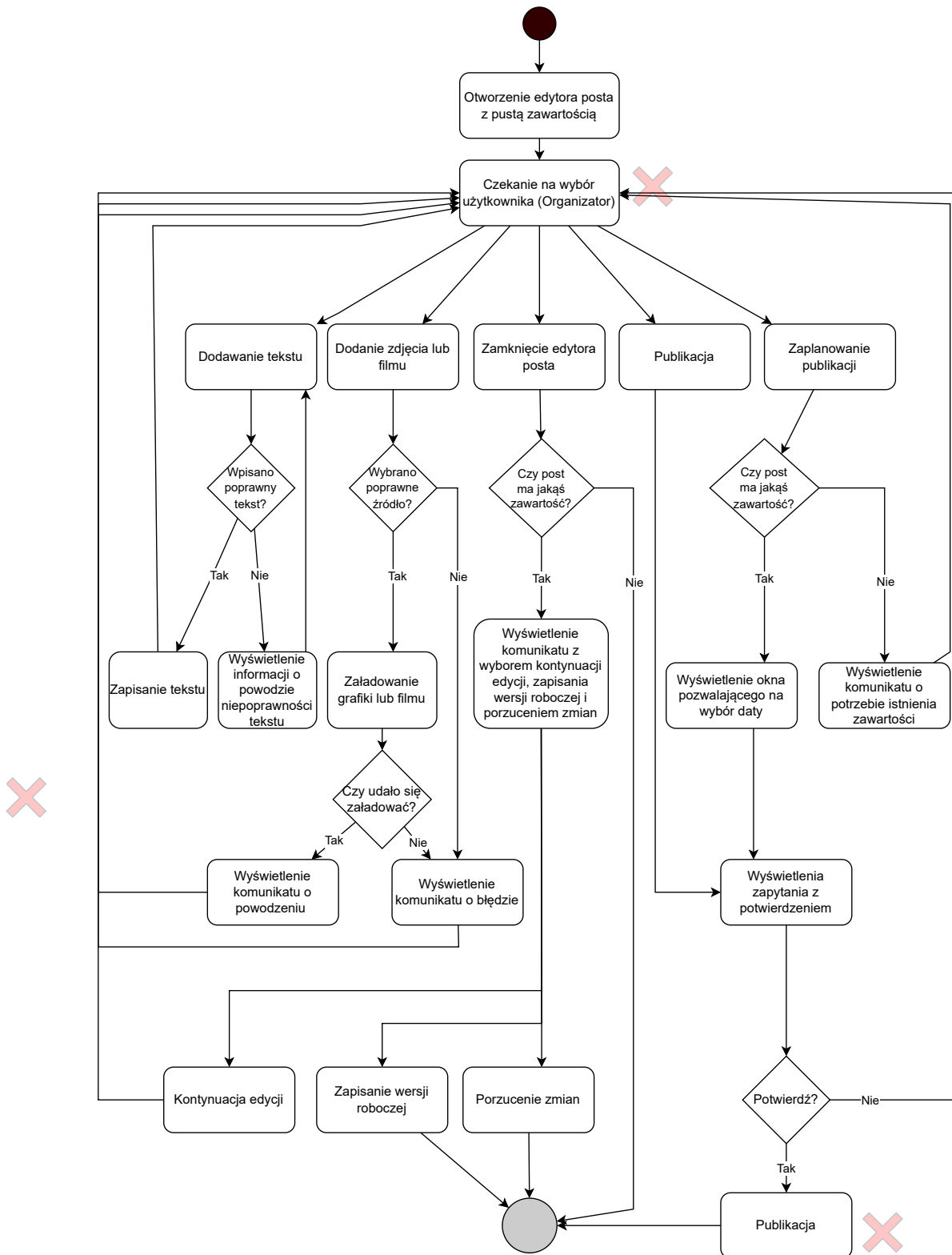
Poniższy diagram przedstawia aktywności jakie zostaną wykonane, kiedy Student będzie przeglądał wydarzenia. System generował będzie zbiór wydarzeń których użytkownik jeszcze nie widział. Po przejrzeniu całego zbioru, generacja powtórzy się, aż do momentu przejrzenia wszystkich, wtedy zbiór przejrzanych wydarzeń stanie się zbiorem pustym, i procedura się powtórzy. Przeglądanie może zakończyć się kiedy student opuści witrynę zamykając ją, lub przechodząc do szczegółów wybranego wydarzenia.



Rysunek 4: Activity diagram: Przeglądanie wydarzeń

### 5.1.2 Activity diagram: dodawanie posta

Podczas dodawania posta pojawia się okno, w którym można wpisywać treść, oraz dodawać różnego rodzaju załączniki. Wszystko odbywa się w swego rodzaju pętli, co widać po strukturze poniższego diagramu.



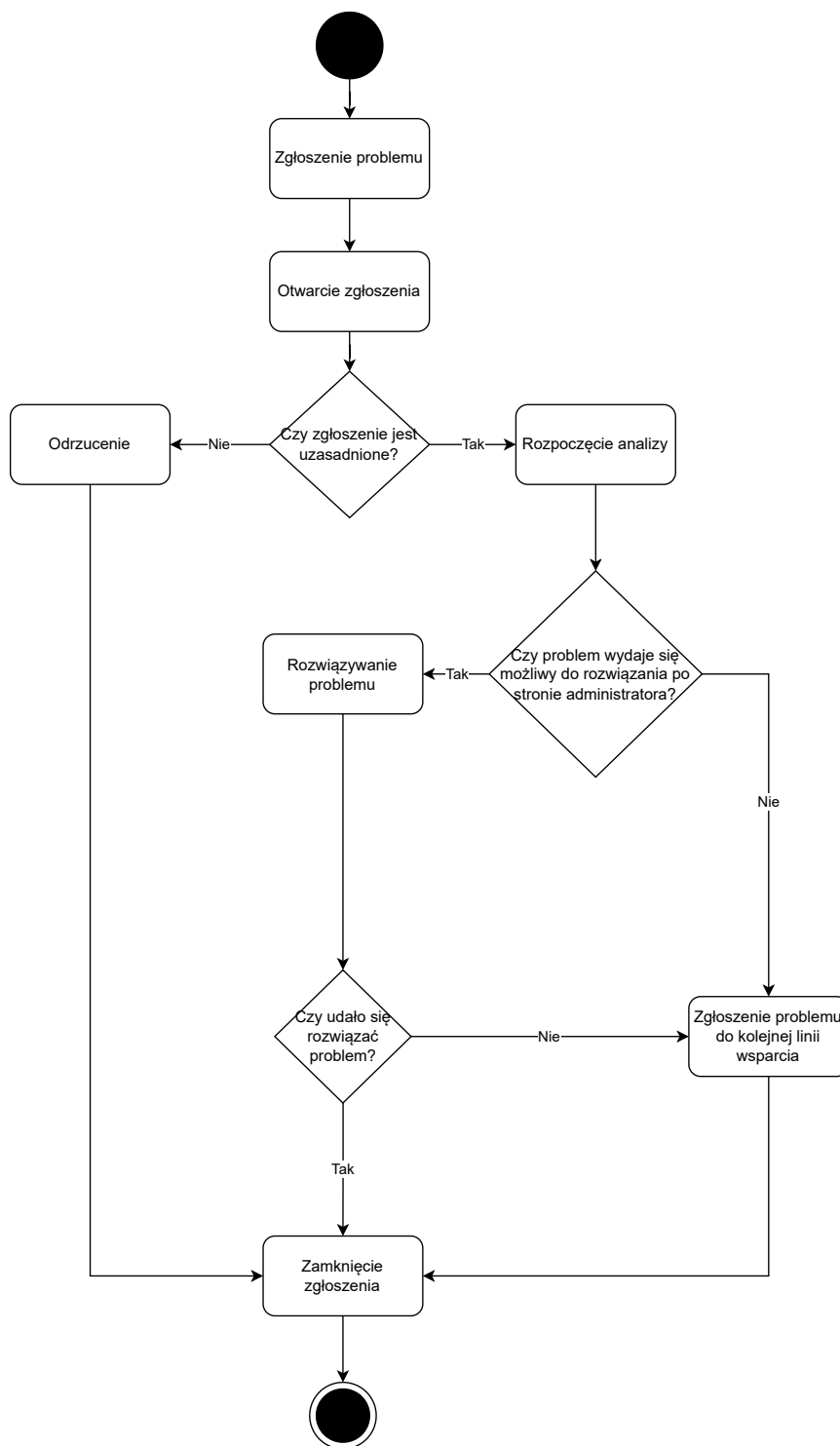
Rysunek 5: Activity diagram: Dodawanie posta

### 5.1.3 Activity diagram: rozpatrywanie problemu użytkownika

Jednym z zadań administratora jest rozpatrywanie problemów zgłoszonych przez użytkowników. Na początek student lub organizator tworzy zgłoszenie. Administrator otwiera je i decyduje, czytając opis, czy jest ono uzasadnione. Jeśli uzna zgłoszenie za niuzasadnione, odrzuca je. W przeciwnym wypadku podejmuje się analizy. Jeżeli jego zdaniem opisany problem nie jest możliwy do rozwiązania po jego stronie (np. jest to błąd, który powstał na poziomie kodu), zgłasza go do kolejnej linii wsparcia, czyli programistów. Jeśli jednak stwierdza, że jest w stanie samemu rozwiązać problem, to oznacza go, jako rozwiązywany. W przypadku sukcesu, problem zostaje rozwiązany i zgłoszenie jest zamknięte. Jeżeli mu się nie uda, podobnie jak wcześniej, problem jest zgłaszany do kolejnej linii wsparcia.

Większość aktywności na diagramie związanych jest z przypisaniem do problemu odpowiedniego statusu, co umożliwia wygodniejsze zarządzanie istniejącymi zgłoszeniami i zapobiega pomyłkom, takim jak przypadkowe pominięcie zgłoszenia. Aktywność "Zamknięcie zgłoszenia" oznacza zapalenie flagi, która informuje, czy zgłoszenie jest aktualne, czy historyczne (nie należy już go rozpatrywać). W momencie zamknięcia przypisany zostaje dodatkowy status (odrzucony, rozwiązany, przesłany dalej). Daje to możliwość przejrzania historii wszystkich zgłoszeń, ale jednocześnie w prosty sposób możliwe jest przejrzanie tylko tych aktualnych.

W momencie przesłania problemu do kolejnej linii wsparcia również następuje zamknięcie zgłoszenia. Chodzi tutaj o zamknięcie zgłoszenia z perspektywy administratora, a ten nie ma już nic więcej do zrobienia z danym zgłoszeniem. Odpowiednia wiadomość zostaje wysłana do zespołu programistów, którzy dalej zajmują się problemem.



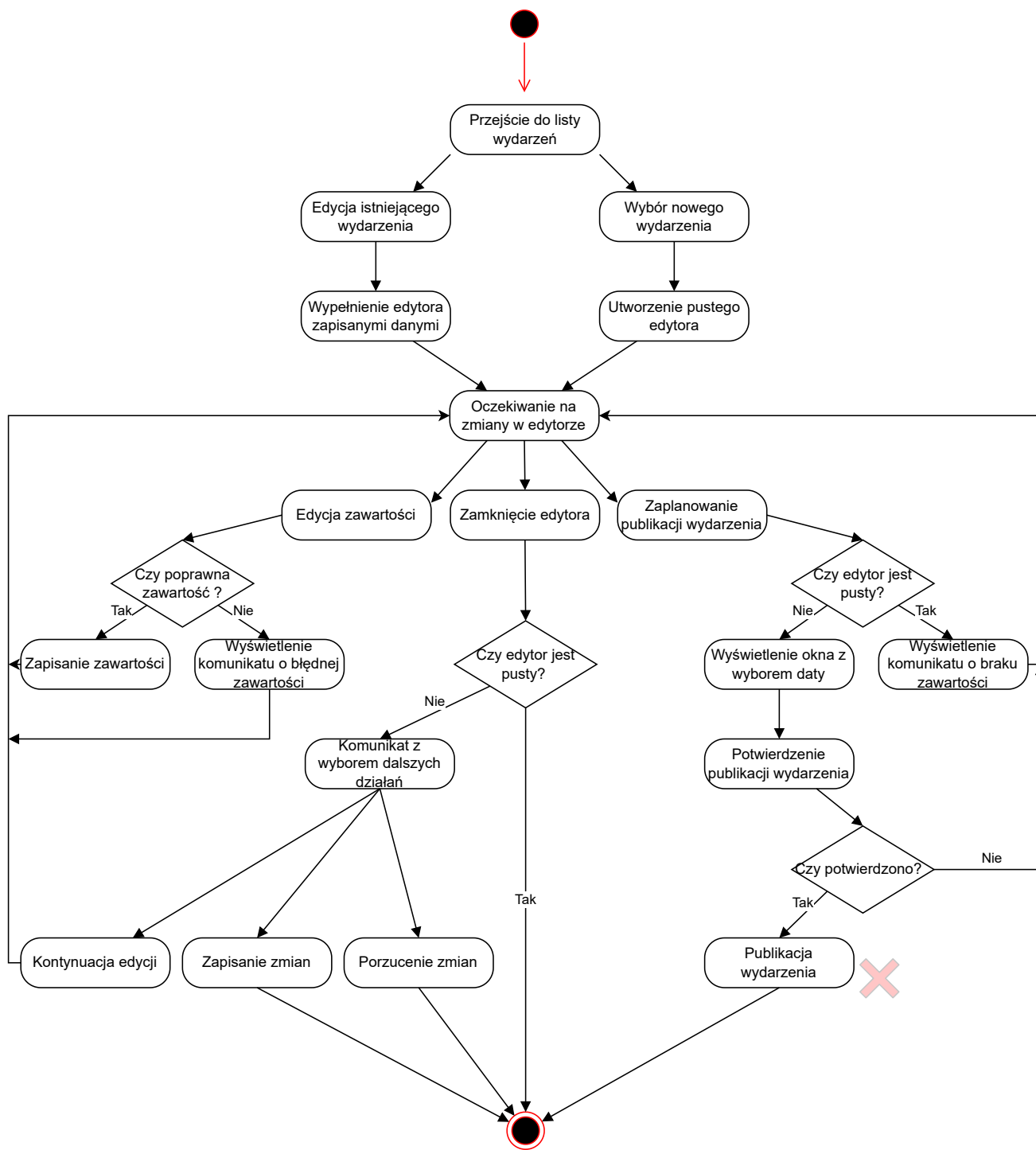
Rysunek 6: Activity diagram: rozpatrywanie problemu użytkownika



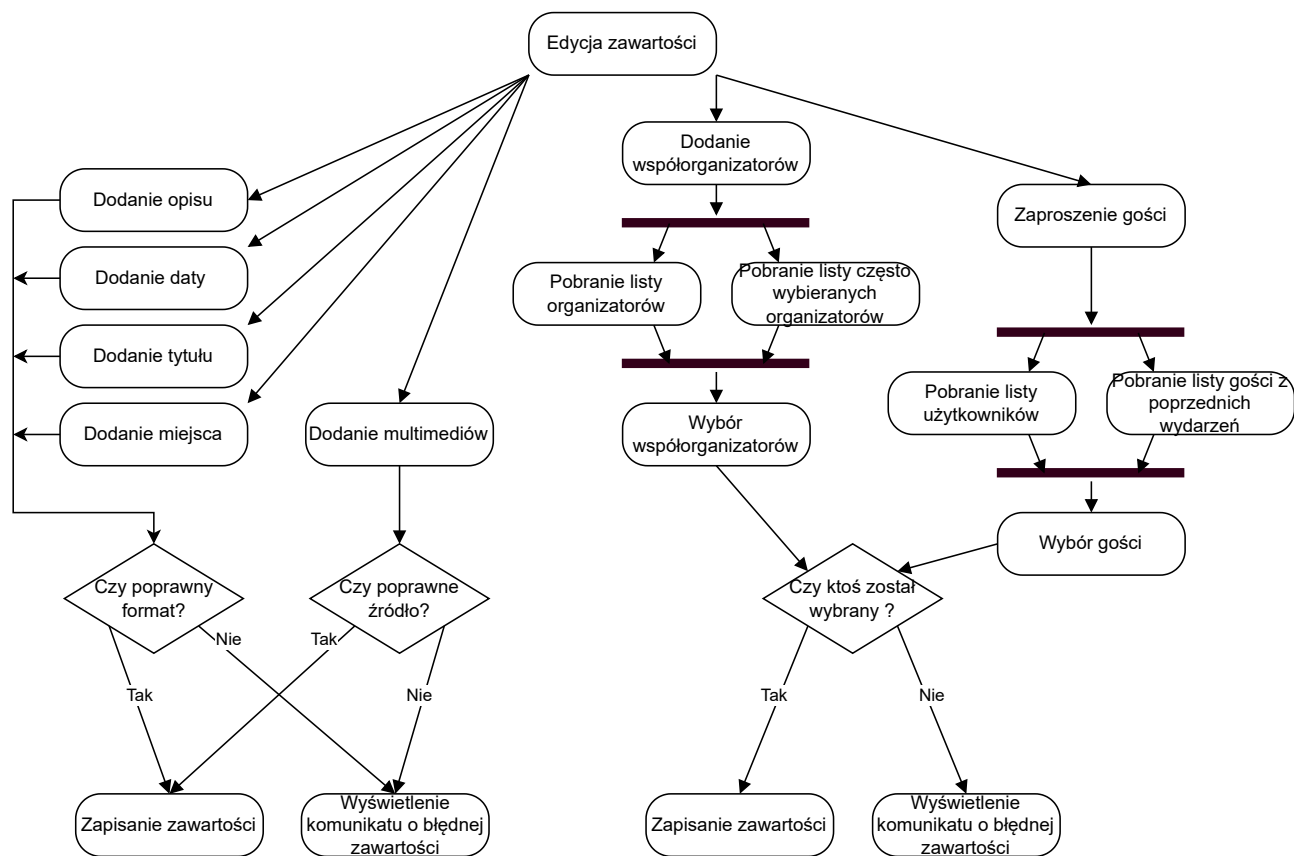
#### 5.1.4 Activity diagram: tworzenie wydarzenia

Poniższy diagram przedstawia sekwencję kroków, które organizator podejmuje, aby utworzyć nowe wydarzenie lub wprowadzić zmiany w już istniejącym. Proces ten obejmuje zarówno etapy tworzenia, jak i modyfikacji wydarzenia, uwzględniając interakcje z systemem oraz pomiędzy różnymi komponentami platformy społecznościowej. Dodatkowo system umożliwia funkcjonalność zapisania wydarzenia w formie kopii roboczej, aby w każdej chwili móc powrócić do procesu jego tworzenia bez konieczności wprowadzania danych od nowa. Ważnym elementem procesu tworzenia wydarzenia jest jego publikacja, system umożliwia publikację natychmiastową lub zaplanowaną.

(Aby diagram był bardziej czytelny, elementy modyfikacji zawartości na diagramie głównym został rozwinięty w postaci drugiego diagramu aktywności, który dotyczy jedynie tej funkcjonalności, stanowi on spójną część diagramu głównego.)



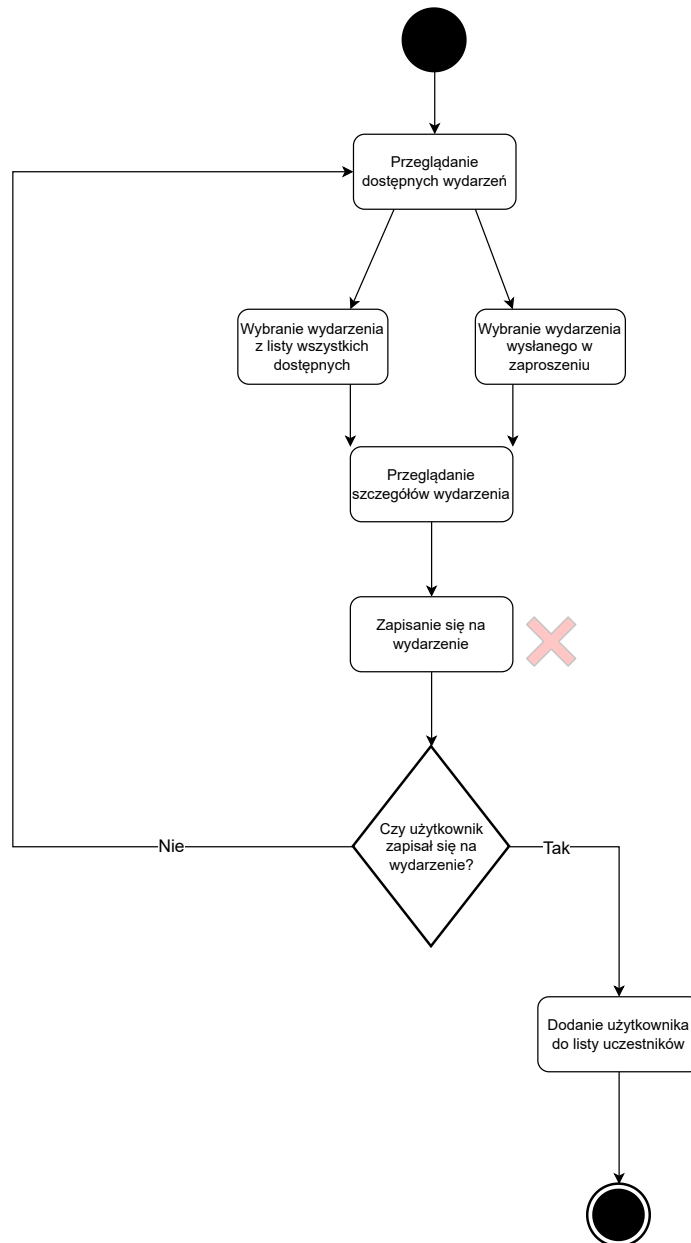
Rysunek 7: Activity diagram: tworzenie wydarzenia



Rysunek 8: Activity diagram: Rozszerzenie edycji zawartości z diagramu "tworzenie wydarzeń"

### 5.1.5 Activity diagram: zapisywanie na wydarzenia

Poniższy diagram przedstawia kolejne aktywności w celu zapisania się na wydarzenie. Użytkownik podczas przeglądania dostępnych dla niego wydarzeń, tzn. takich, do których spełnia wymagania (np. wiekowe lub grupowe - czy jest studentem 1. roku), ma możliwość wybrania takiego eventu. Kolejną możliwością jest wybranie wydarzenia, na które otrzymał zaproszenie. Po wybraniu eventu, użytkownik przechodzi do przeglądania jego szczegółów, gdzie może zobaczyć pełne informacje. Następnie ma możliwość zapisania się na imprezę. W przypadku, gdy się nie zapisał, wraca do przeglądania dostępnych wydarzeń. Natomiast w przeciwnym przypadku, zostaje dodany do listy uczestników.

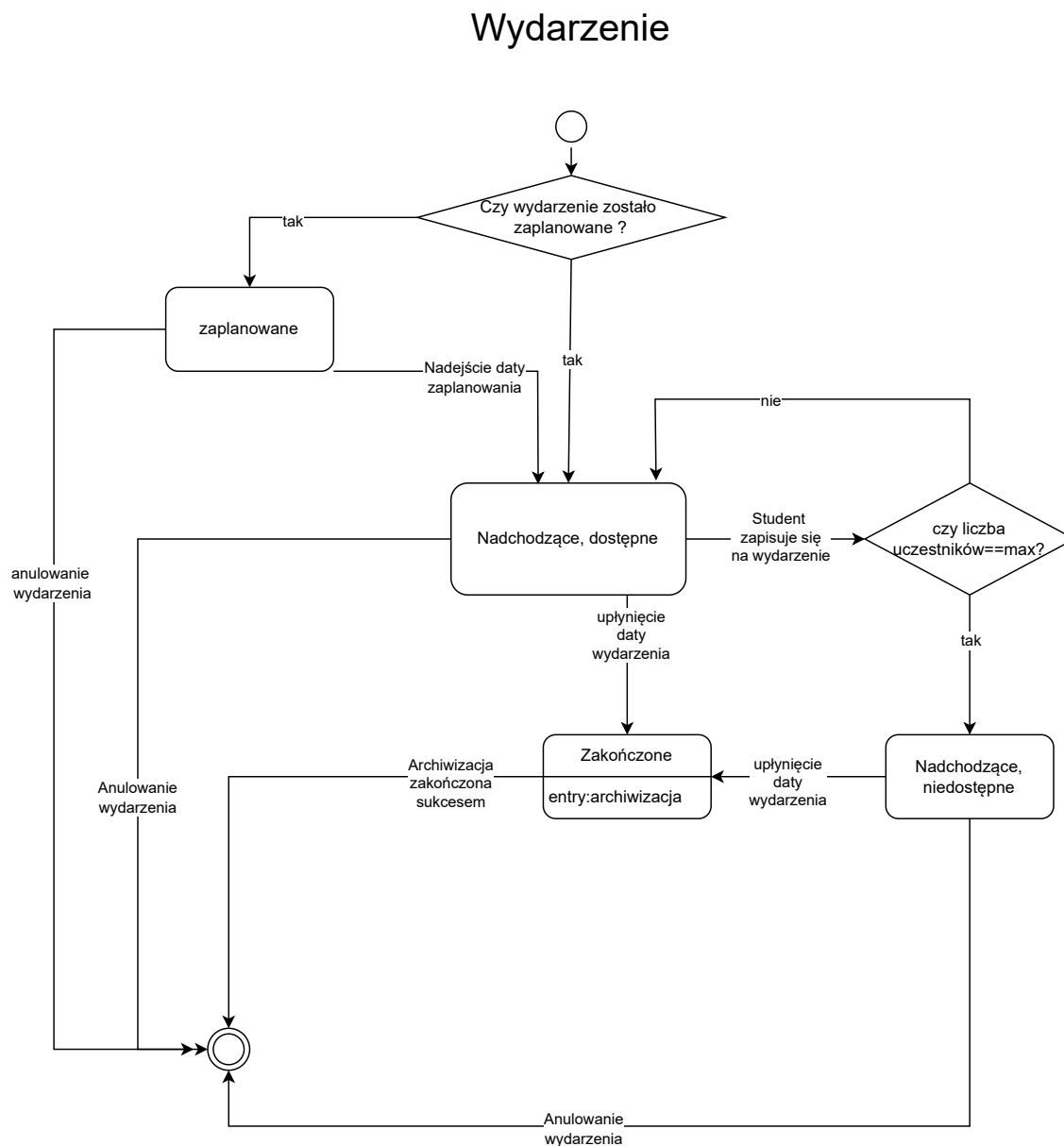


Rysunek 9: Activity diagram: zapisywanie na wydarzenia

## 5.2 Diagramy stanów

### 5.2.1 State Diagram: wydarzenie

Poniższy diagram przedstawia w jakich stanach wydarzenie będzie postrzegane przez system. Organizator wydarzenia będzie mógł zaplanować jego publikację lub wykonać to od razu. Następnie to nadchodzące wydarzenie będzie dostępne do zapisu, dopóki nie zapisze się maksymalna ilość osób (jeżeli taka dla wydarzenia istnieje) wtedy stanie się nie dostępne. Niezależnie od dostępności, po zakończeniu, wydarzenie zostanie poddane archiwizacji, aby pozwolić na analizę biznesową jego statystyk.



Rysunek 10: State Diagram: wydarzenie

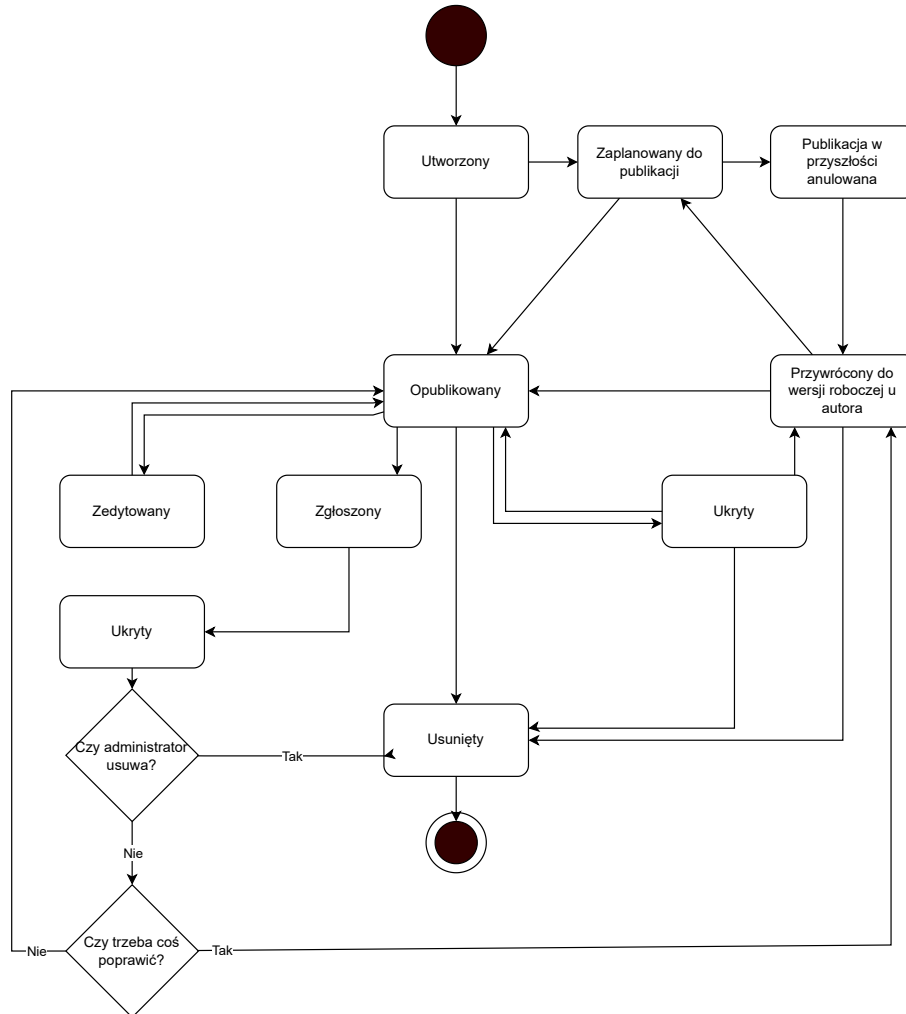
### 5.2.2 State diagram: post

Post po utworzeniu może być bezpośrednio opublikowany, lub może zostać przygotowany do publikacji w wyznaczonej dacie. Taką publikację można anulować zanim nastąpi.

Pomocny dla autora posta jest stan "Ukryty". W tym stanie post jest widoczny wyłącznie dla autora. Po przywróceniu posta nie są tracone komentarze i polubienia.

Post można w dowolnym momencie edytować.

Kiedy post zostanie zgłoszony, przechodzi do stanu "Ukryty" i czeka na decyzję administratora. Administrator może zgłosić autorowi posta konieczność naniesienia poprawek. Przenosi go wtedy do wersji roboczej u autora. Kiedy autor naniesie potrzebne poprawki może taki post znów opublikować.



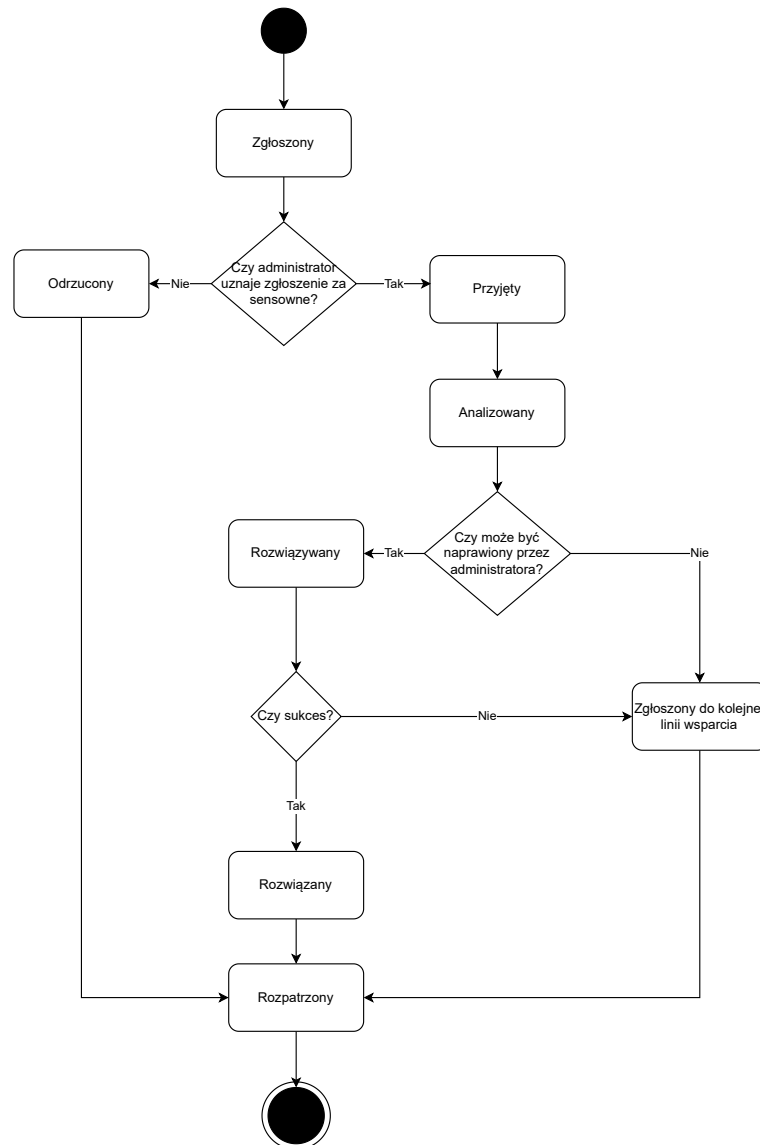
Rysunek 11: State diagram: post

### 5.2.3 State diagram: problem użytkownika

Każdy użytkownik systemu może zgłosić problem. Rolą administratora jest jego analiza i próba rozwiązania. Na początku problem może od razu zostać odrzucony. Jeśli jednak administrator uzna go za stosowny, to problem zostaje oznaczony jako przyjęty, a po rozpoczęciu analizy - analizowany. Po wstępnej analizie problem zostaje zgłoszony do kolejnej linii wsparcia lub jest rozwiązywany bezpośrednio przez administratora. Po rozwiązaniu problemu lub zgłoszeniu do kolejnej linii wsparcia, jest on uznany za rozpatrzony.

Wymienione stany są związane przede wszystkim z flagami, które przypisane są do konkretnego problemu. Służą one głównie do łatwiejszej organizacji zgłoszeń i zapobieganiu przypadkowemu pominięcia niektórych z nich. Dzięki temu administrator może je wygodniej filtrować i nimi zarządzać.

Programiści wiedzą, którymi problemami powinni się zająć, ponieważ są one oznaczone jako zgłoszone do kolejnej linii wsparcia. W ten sposób automatycznie przeglądają tylko te, które są dla nich istotne.



Rysunek 12: State diagram: problem użytkownika

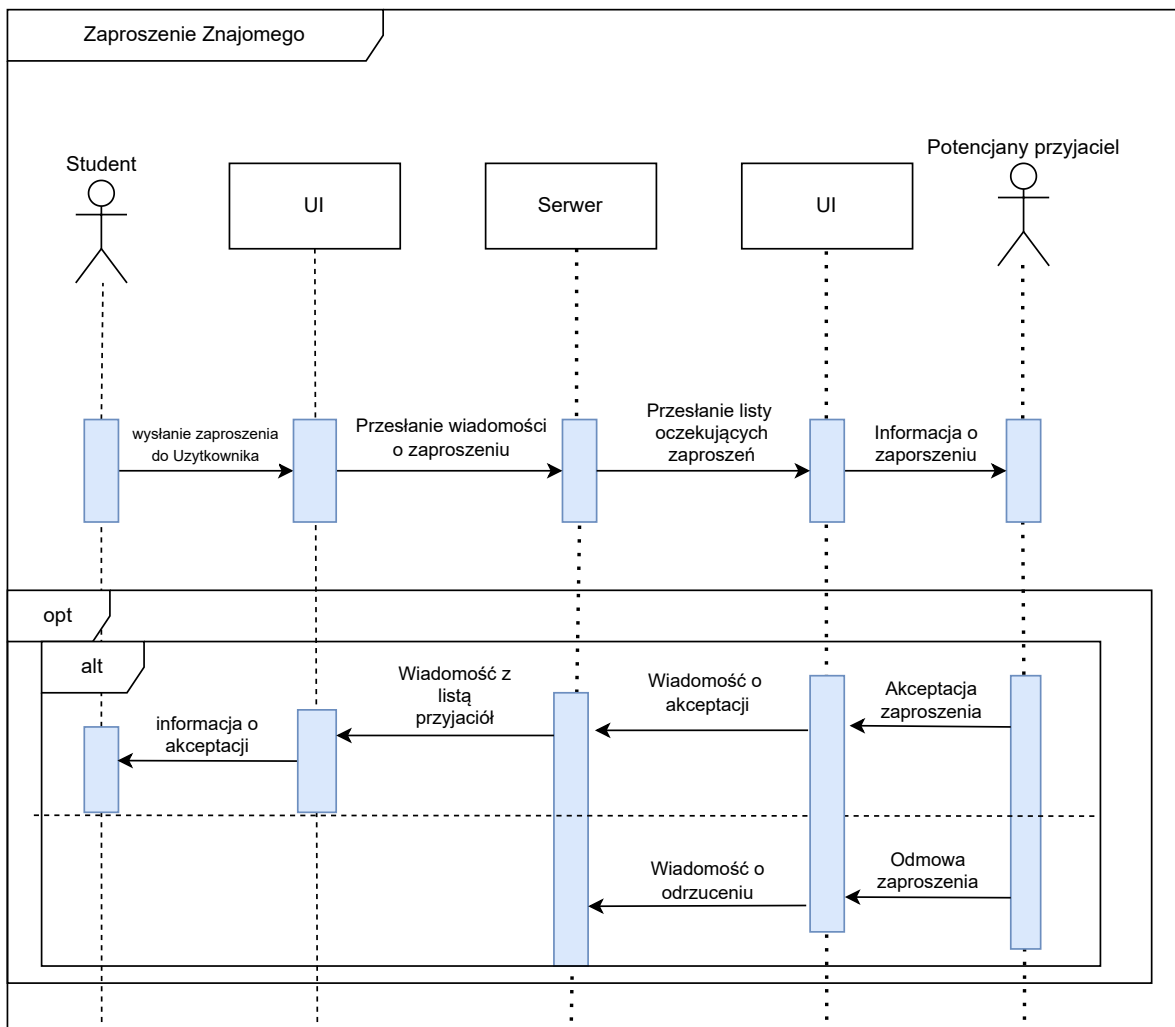
## 6 Komunikacja Systemu

Kluczowym aspektem systemu jest komunikacja między komponentami i aktorami, co wynika z jego natury. Znacząca część funkcjonalności Aktorów wymaga interakcji bezpośrednio z innym aktorem, lub danymi które ten aktor wprowadził do systemu. Szczegóły tej komunikacji pokazane zostały na diagramach sekwencji oraz w na schematach RAML

### 6.1 Diagramy sekwencji

#### 6.1.1 Zaproszenie znajomego

Wysyłanie zaproszenia i jego otrzymanie odbywa się po przez pośrednio serwer. Zapraszający wysyła wiadomość o chęci dodania Zaproszonego do znajomych. Serwer następnie wyśle informacje do Zapraszającego który może przyjęcie zignorować albo rozpatrzyć. w zależności od tego czy zaakceptuje czy odrzuci zaproszenie, odpowiednia informacja zostanie przekazana do serwera. W przypadku akceptacji odpowiednia wiadomość zostanie wysłana do zapraszającego. Informacje od i do serwera aktorzy wysyłają po przez interfejs użytkownika. Szczegółowe informacje o zapytaniach do i od serwera można znaleźć w schemacie RAML.

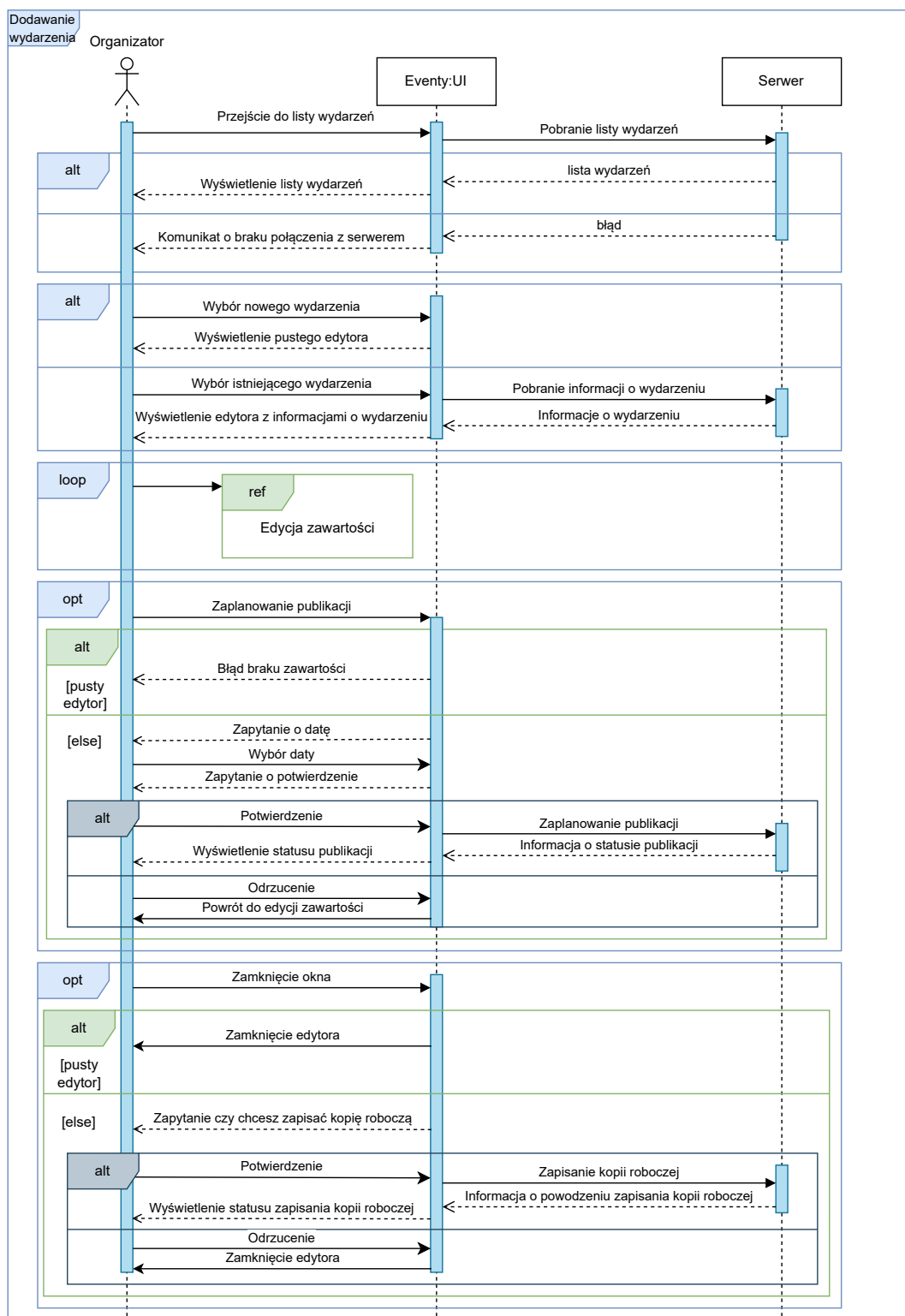


Rysunek 13: Sequence diagram: zaproszenie znajomego



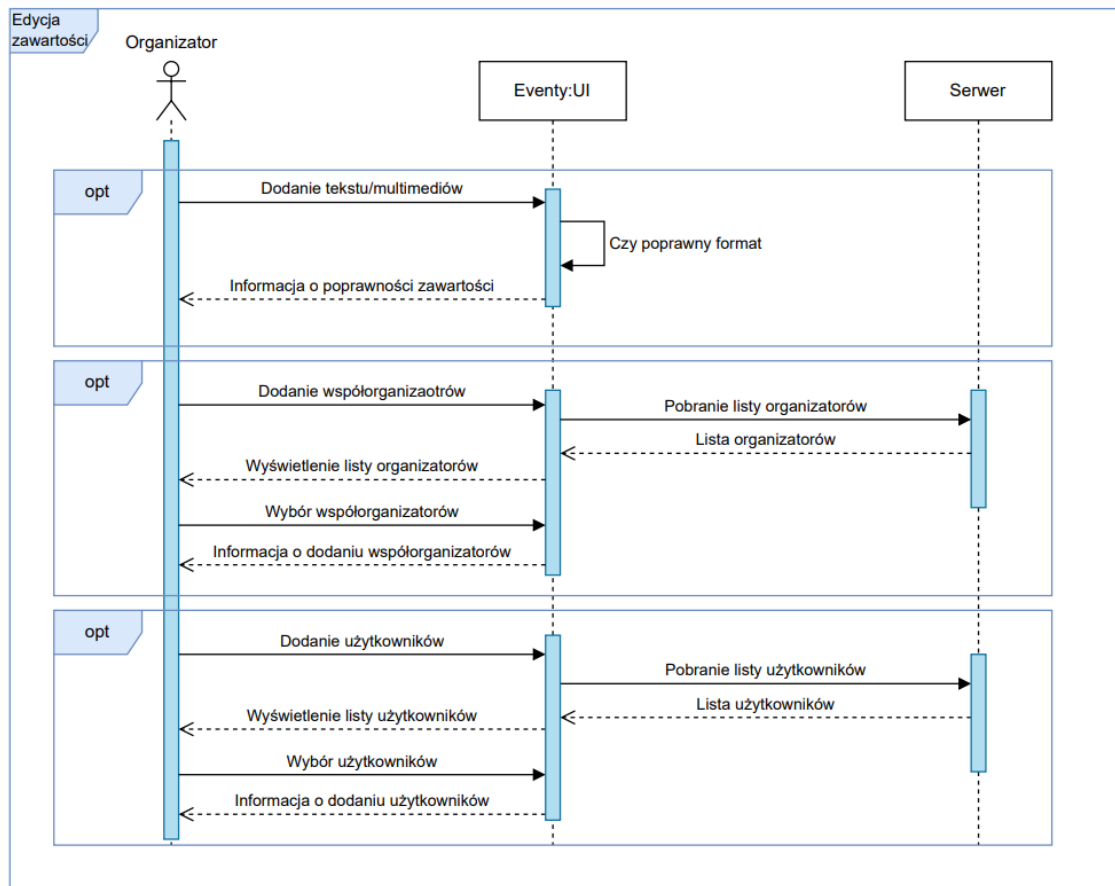
### 6.1.2 Dodawanie wydarzenia

- Organizator może zarówno dodać nowe jak i edytować już istniejące wydarzenie. Po odpowiedniej interakcji z interfejsem komunikuje się on z serwerem aby pobrać listę aktywnych wydarzeń. Po otwarciu edytora wydarzenia organizator komunikuje się z interfejsem, przekazując mu niezbędne informacje o wydarzeniu, takie jak nazwa, data, godzina, miejsce, opis itp.
- Po wprowadzeniu danych, interfejs użytkownika przesyła zapytanie o dodanie wydarzenia do serwera. W tym momencie następuje komunikacja między UI a serwerem, gdzie UI przekazuje dane o wydarzeniu, a serwer odpowiednio je przetwarza. Serwer może sprawdzić poprawność danych, dostępność miejsca, a także przypisać identyfikator unikalny dla nowego wydarzenia.
- Po pomyślnym dodaniu wydarzenia, serwer przesyła potwierdzenie do interfejsu użytkownika. UI aktualizuje widok, prezentując organizatorowi informację o powodzeniu operacji. Organizator otrzymuje potwierdzenie dodania wydarzenia i może kontynuować korzystanie z funkcji interfejsu użytkownika.
- W przypadku wystąpienia błędów, na przykład z powodu niepoprawnych danych, serwer przesyła odpowiedź z informacją o błędzie do interfejsu użytkownika. UI wyświetla odpowiednie komunikaty o błędach, informując organizatora o konieczności poprawienia wprowadzonych danych.



Rysunek 14: Sequence diagram: dodawanie wydarzenia

### 6.1.3 Edycja zawartości

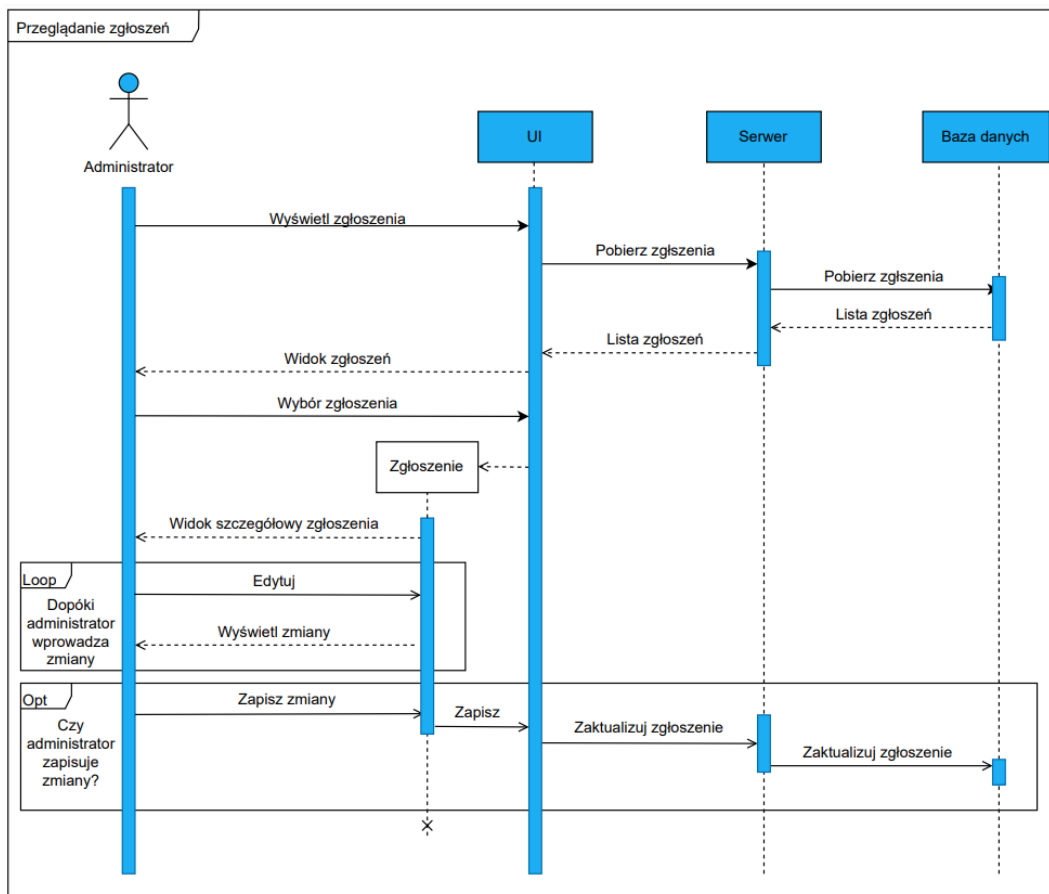


Rysunek 15: Sequence diagram: edycja zawartości

#### 6.1.4 Przeglądanie zgłoszeń

Administrator ma możliwość przeglądania zgłoszenia użytkowników. Po otwarciu zakładki ze zgłoszeniami do serwera wysyłane jest zapytanie o dostępne zgłoszenia (czyli takie, które nie zostały jeszcze odrzucone lub rozpatrzone). Serwer pobiera je z bazy danych i zwraca ich listę, która wyświetlana jest administratorowi.

Administrator wybiera zgłoszenie, po czym wyświetla się widok szczegółów i edycji. Może on wprowadzić zmiany, takie jak odrzucenie lub rozpatrywanie zgłoszenia. Po zamknięciu zgłoszenia, jeśli administrator chce zapisać zmiany, do serwera wysyłane jest zapytanie zawierające zgłoszenie ze zmienionymi danymi. Następnie dane te są aktualizowane w bazie danych.

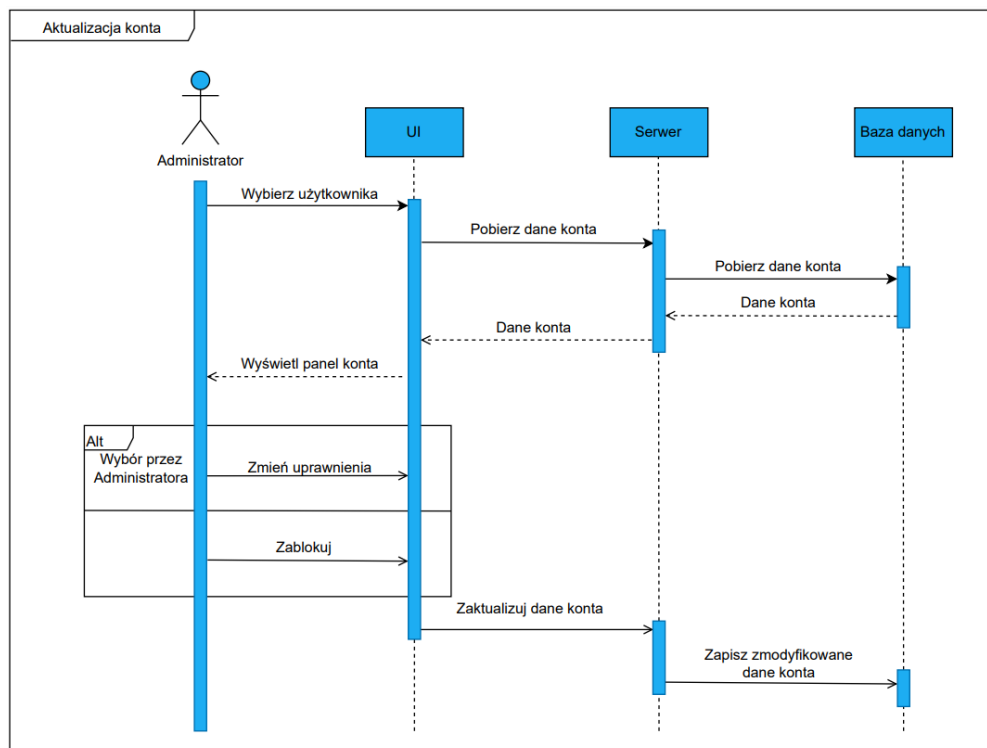


Rysunek 16: Sequence diagram: przeglądanie zgłoszeń

### 6.1.5 Aktualizacja konta

Do zadań administratora należy również aktualizacja kont użytkowników. Jego rolą jest nadawanie odpowiednich uprawnień lub blokowanie osób, które nie przestrzegają regulaminu.

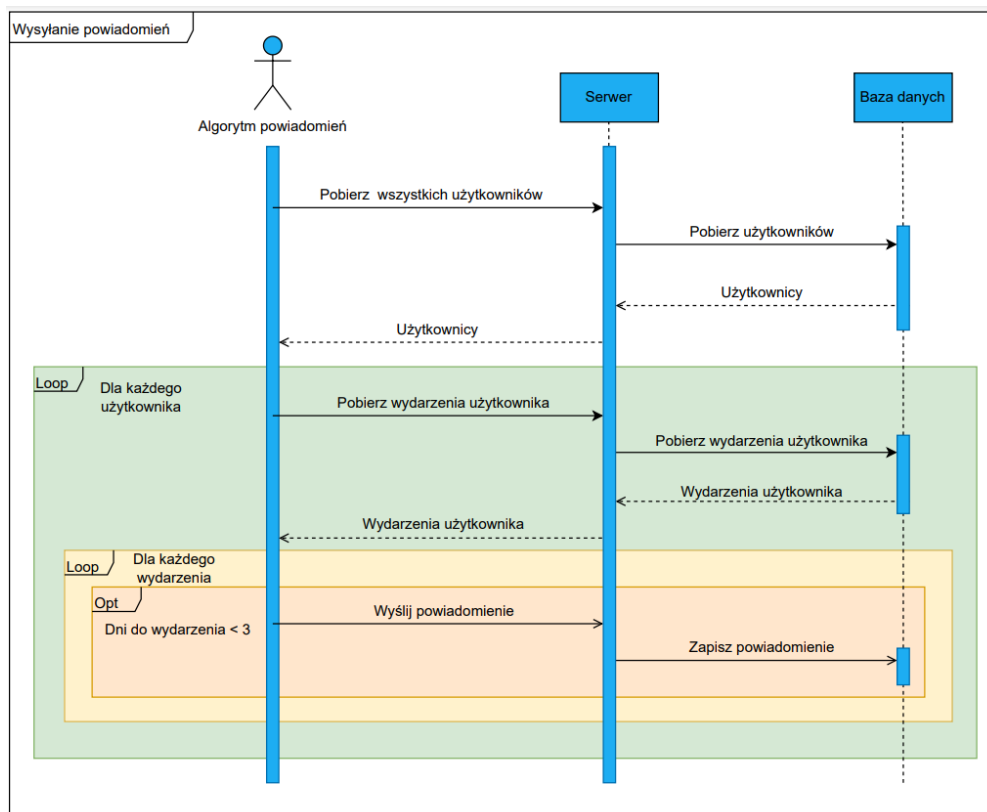
Administrator wybiera użytkownika. Może to zrobić z różnych miejsc w aplikacji. Do serwera wysyłane jest zapytanie o szczegółowe dane konta, które następnie są pobierane z bazy danych i wyświetlane w formie panelu konta administratorowi. Administrator może zmienić uprawnienia użytkownika (możliwość dodawania wydarzeń) lub zablokować jego konto, jeśli naruszył on regulamin. Po wykonaniu jednej z wymienionych czynności do serwera wysyłane jest zapytanie, które aktualizuje dane konta.



Rysunek 17: Sequence diagram: Aktualizacja konta

### 6.1.6 Wysyłanie powiadomień o nadchodzących wydarzeniach

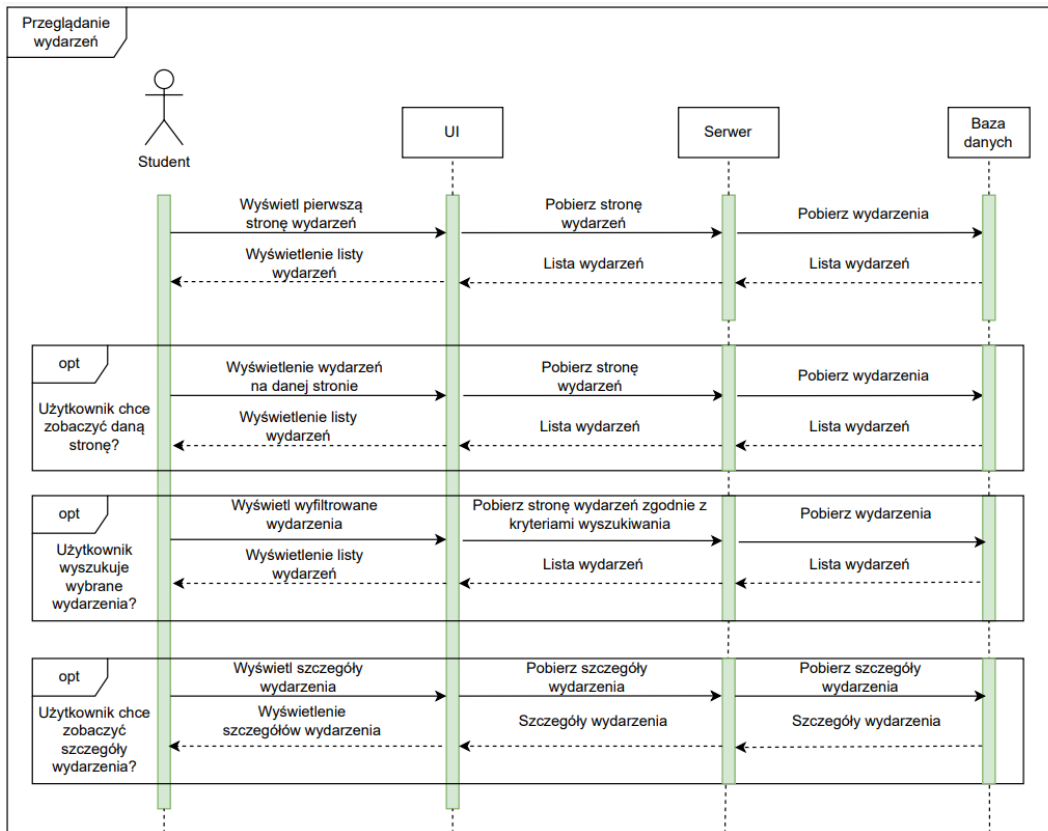
Jeśli zbliża się wydarzenie, na które zapisany jest użytkownik, dostaje on powiadomienie. Algorytm powiadomień uruchamiany jest co pewien czas (np. co godzinę). Pobiera on wtedy każdego aktywnego użytkownika. Przegląda wszystkich pobranych użytkowników i pobiera z bazy danych wydarzenia, na które są zapisani. Jeśli znajdzie wydarzenie, którego termin jest za mniej niż 3 dni, zapisuje powiadomienie przypisane do każdego użytkownika. Powiadomienia te są automatycznie odczytywane i wyświetlane użytkownikowi, gdy loguje się on na platformie. Umożliwia to przypomnienie o nadchodzących wydarzeniach.



Rysunek 18: Sequence diagram: Wysyłanie powiadomień

### 6.1.7 Przeglądanie wydarzeń

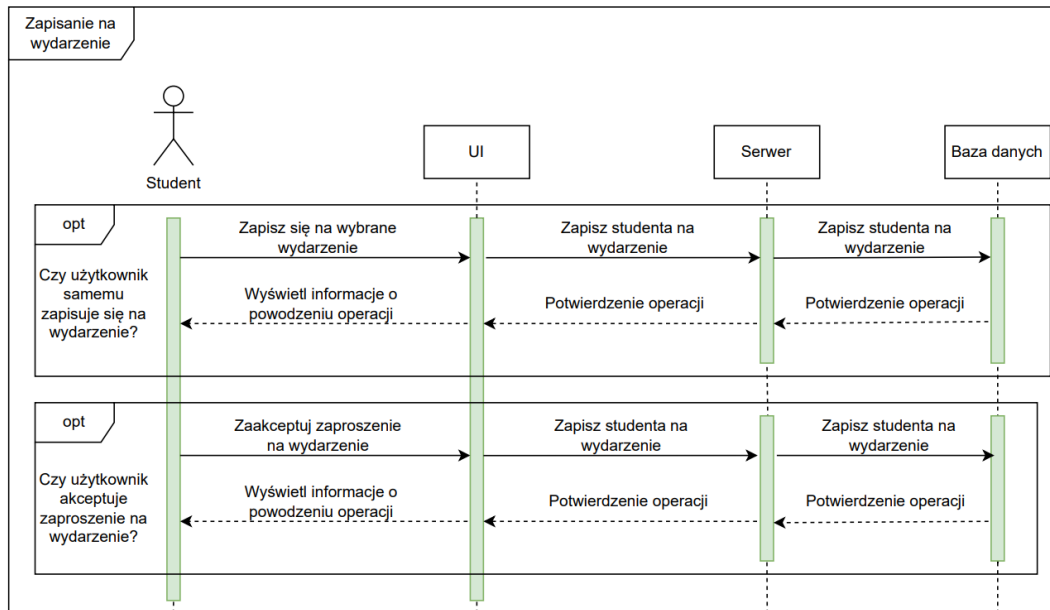
Użytkownik ma możliwość przeglądania dostępnych dla niego wydarzeń, tj. takich, do których spełnia wymagania lub otrzymał zaproszenie. Po wyświetleniu widoku przeglądania wydarzeń, widzi określoną liczbę najbliższych wydarzeń na pierwszej stronie. Ponadto, użytkownik ma możliwość przeglądania kolejnych stron z następnymi wydarzeniami. Kolejną rzeczą jest filtrowanie eventów na podstawie odpowiednich kryteriów wyszukiwania, które są możliwe do wybrania, tj. data rozpoczęcia, cena wejścia, liczba zapisanych znajomych, lokalizacja, sortowanie według zadanego kryterium. Następną możliwą ścieżką w tym widoku, jest przeglądanie szczegółów danego wydarzenia, gdzie użytkownik może zobaczyć bogatą liczbę dodatkowych informacji, takich jak dokładny opis, godzinę rozpoczęcia/zakończenia, adres miejsca odbywania się wydarzenia, znajomych, którzy również biorą udział.



Rysunek 19: Sequence diagram: Przeglądanie wydarzeń

### 6.1.8 Zapisanie na wydarzenie

Student, aby zapisać się na wydarzenie ma dwie możliwości. Pierwszą z nich jest dobrowolne wybranie wydarzenia z widoku przeglądania wydarzeń i wzięcie w nim udziału. Kolejną możliwością jest zaakceptowanie otrzymanego zaproszenia na wydarzenie. Po próbie zapisania się na event, użytkownik otrzymuje potwierdzenie o powodzeniu operacji.

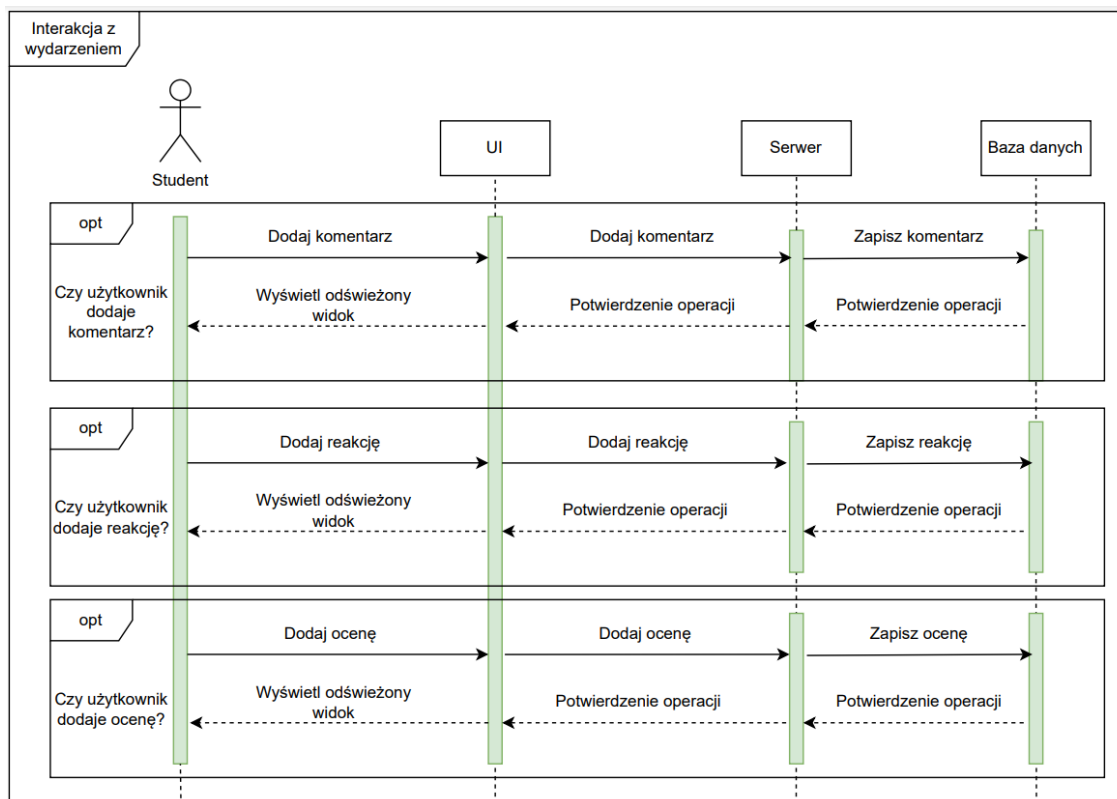


Rysunek 20: Sequence diagram: Przeglądanie wydarzeń



### 6.1.9 Interakcja z wydarzeniem

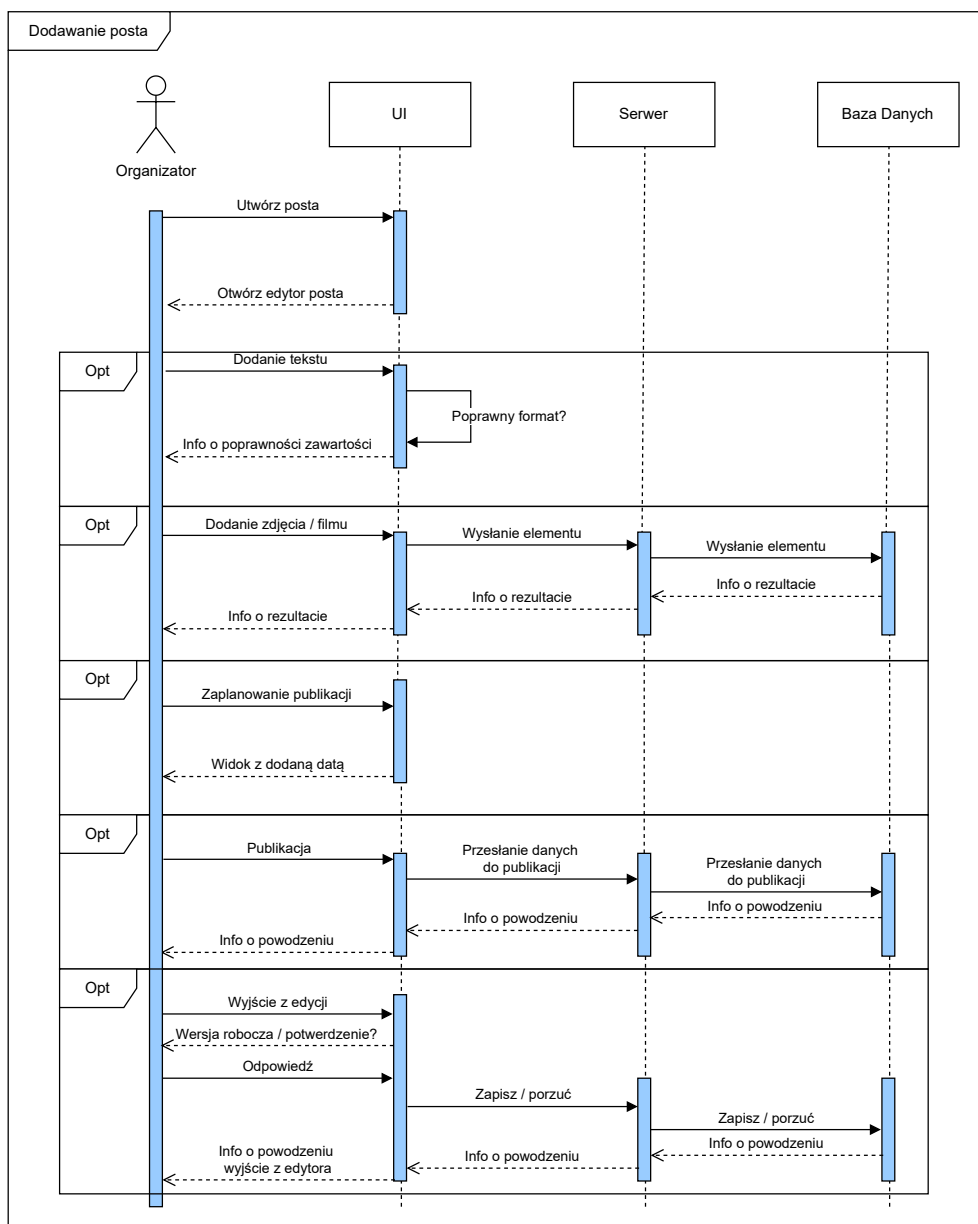
Student ma możliwość bezpośredniej interakcji z wybranym wydarzeniem. Są to: dodawanie komentarzy, reakcji oraz oceny do eventu. W przypadku danej interakcji, odpowiednia zawartość (komentarz, reakcja bądź ocena) zapisywana jest w bazie danych, następnie zwracane jest potwierdzenie wykonania operacji, a użytkownikowi odświeża się widok z uaktualnioną zawartością.



Rysunek 21: Sequence diagram: Interakcja z wydarzeniem

### 6.1.10 Dodawanie posta

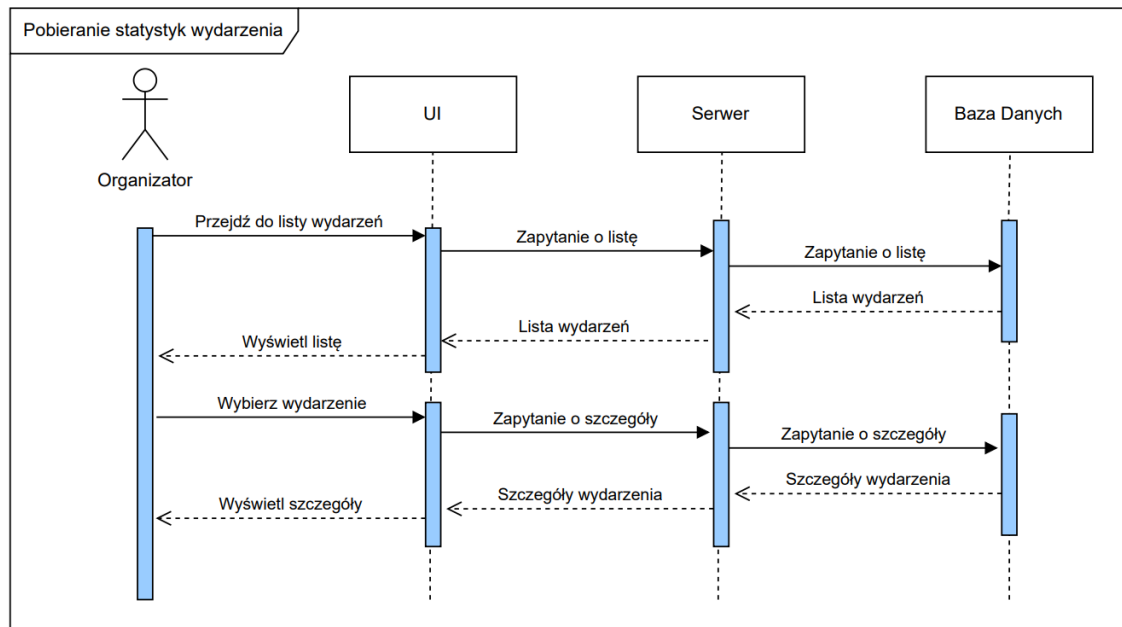
Dodawanie posta odbywa się w swojego rodzaju pętli, powracającej do momentu możliwości wyboru - co autor chciałby teraz edytować? Każdy rodzaj edycji jest opcją - na przykład nie ma obowiązku dodawania zdjęcia do posta.



Rysunek 22: Sequence diagram: Dodawanie posta

### 6.1.11 Statystyki wydarzenia

Aby wyświetlić statystyki wydarzenia, Organizator w pierwszej kolejności będzie musiał wybrać to wydarzenie z odpowiedniej kolekcji, którą dostarczy mu serwer. Dopiero po wybraniu z kolekcji żądanego elementu, serwer prześle pełną informację o szczegółach, która zostanie wyświetlona przez jego interfejs użytkownika.



Rysunek 23: Sequence diagram: Statystyki wydarzenia

## 6.2 Schematy komunikacji W RAML

Poniższy kod w języku RAML opisuje jakie Endpointy będą stosowane w komunikacji, oraz jakie dane będą przez nie przesyłane

```
##RAML 1.0
title: MiniSpace api
baseUri: https://minispace.com/api
version: 1.0
```

```
uses:
  assets: assets.lib.raml
```

```
types:
  Report:
    type: object
    properties:
      id: integer
      title: string
      type:
        enum: ["bug", "behaviour"]
      description: string
      userId: integer
```

```
ReportTitle:
  type: object
  properties:
    id: integer
    title: string
    type:
      enum: ["bug", "behaviour"]
```

```
Notification:
  type: object
  properties:
    id: integer
    userId: integer
    eventId: integer
    readByUser: bool
```

```
User:
  type: object
  properties:
    id: integer
    isBanned: bool
    canCreateEvents: bool
```

```
Event:
  type: object
  properties:
    id: integer
    hosts: User[]
    title: string
    startDate: dateTime
    endDate: dateTime
    location: string
    description: string
    participants: User[]
```

```

    capacity: integer
    fee: double

PageCriteria:
  type: object
  properties:
    currentPage: integer
    elementsPerPage: integer

FilterCriteria:
  type: object
  properties:
    fieldName: string
    direction: string # ASC OR description

SearchCriteria:
  type: object
  properties:
    pageCriteria: PageCriteria
    filterCriteria: FilterCriteria

EventSignUp:
  type: object
  properties:
    userId: number
    eventId: number

Comment:
  type: object
  properties:
    userId: number
    eventId: number
    message: string

Reaction:
  type: object
  properties:
    userId: number
    eventId: number
    reaction:
      type:
        enum: ["Hate it", "It was okay", "Wow", "Like it", "Love it"]

Rating:
  type: object
  properties:
    userId: number
    eventId: number
    rating: number

Post:
  type: object
  properties:
    hostId: number
    eventId: number
    title: string

```

```

        content: string
        date: datetime

/reports:
  get:
    queryParameters:
      access_token:
        type: string
        required: true
    description: Lists all reports
    responses:
      200:
        body:
          application/json:
            type: ReportTitle[]
  post:
    queryParameters:
      access_token:
        type: string
        required: true
    description: Create new report
    body:
      application/json:
        type: Report
  put:
    queryParameters:
      access_token:
        type: string
        required: true
    description: Update report
    body:
      application/json:
        type: Report
/{reportId}:
  get:
    description: Show report details
    responses:
      200:
        body:
          application/json:
            type: Report

/notifications:
  post:
    queryParameters:
      access_token:
        type: string
        required: true
    description: Add new notification
    body:
      application/json:
        type: Notification
  get:
    queryParameters:
      access_token:
        type: string

```

```

        required: true
description: Get user's not read notifications
headers:
  Authorization: access-token
responses:
  200:
    body:
      application/json:
        type: Notification[]

/user/{userId}:
get:
  queryParameters:
    access_token:
      type: string
      required: true
  description: Get details of the account with id {userId}
  responses:
    200:
      body:
        application/json:
          type: User
put:
  queryParameters:
    access_token:
      type: string
      required: true
  description: Update user account
  body:
    application/json:
      type: User

/event:
post:
  queryParameters:
    access_token:
      type: string
      required: true
  description: Create new event
  body:
    application/json:
      type: Event
put:
  queryParameters:
    access_token:
      type: string
      required: true
  description: Event update
  body:
    application/json:
      type: Event
/{eventId}:
get:
  queryParameters:
    access_token:

```

```

        type: string
        required: true
description: Gets event details
responses:
  200:
    body:
      application/json:
        type: Event
/comment:
post:
queryParameters:
  access_token:
    type: string
    required: true
description: Adds user's comment for an event
body:
  application/json:
    type: Comment
responses:
  200:
    descritpion: Success
get:
description: Get all comments of the event
queryParameters:
  access_token:
    type: string
    required: true
responses:
  200:
    descritpion: Success
    body:
      application/json:
        type: Comment[]
/reaction:
post:
description: Adds user's reaction for an event
queryParameters:
  access_token:
    type: string
    required: true
body:
  application/json:
    type: Reaction
responses:
  200:
    descritpion: Success
get:
description: Get all reactions of the event
queryParameters:
  access_token:
    type: string
    required: true
body:
  application/json:
    type: Reaction

```



```

    responses:
      200:
        description: Success
        body:
          application/json:
            type: Reaction[]

/rating:
  post:
    description: Adds user's rating for an event
    queryParameters:
      access_token:
        type: string
        required: true
    body:
      application/json:
        type: Rating
    responses:
      200:
        description: Success

/statistics:
  get:
    description: Get statistics for a specific event
    queryParameters:
      access_token:
        type: string
        required: true
    responses:
      200:
        body:
          application/json:
            type: EventStatistics

/events:
  post:
    description: Lists events that meets the search criteria
    queryParameters:
      access_token:
        type: string
        required: true
    body:
      application/json:
        searchCriteria: SearchCriteria
        required: true
    responses:
      200:
        body:
          application/json:
            items: Event

/sign:
  post:
    description: Signs user for a given event
    queryParameters:
      access_token:

```

```

        type: string
        required: true
    body:
        application/json:
            eventSignUp: EventSignUp
            required: true
    responses:
        200:
            description: Success

/friend:
    get:
        description: Get all user's friends
        queryParameters:
            access_token:
                type: string
                required: true
        responses:
            200:
                body:
                    application/json:
                        type: Student[]
                        example: |
                            {
                                "friends": [
                                    {
                                        "StudnetID": "1",
                                        "StudnetName": "John Doe"
                                    },
                                    {
                                        "StudnetID": "2",
                                        "StudnetName": "Jane Doe"
                                    },
                                    {
                                        "StudnetID": "3",
                                        "StudnetName": "Doe Biden"
                                    }
                                ]
                            }

/{userId}:
    post:
        description: Send friend invitation
        queryParameters:
            access_token:
                type: string
                required: true
        responses:
            200:
                description: invitation sent

/notYet:
    get:
        description: Get user's which are not friends
        queryParameters:
            access_token:
                type: string
                required: true

```

```

    responses:
      200:
        body:
          application/json:
            items: Student[]
/pending:
  post:
    description: Resolve invitation
    queryParameters:
      InvResolution:
        type: string
        enum:
          [ACC,DEC]
      InvitationId:
        type: integer
        required: true
      access_token:
        type: string
        required: true
    responses:
      200:
        description: invitatnion resolved
      404:
        description: no such inivitation
  get:
    descriptpion: Get all user's invitations
    queryParameters:
      access_token:
        type: string
        required: true
    responses:
      200:
        body:
          application/json:
            type: Student[]
            example: |
              {
                "friends": [
                  {
                    "StudnetID": "21",
                    "StudnetName": "Some Name"
                  },
                  {
                    "StudnetID": "2",
                    "StudnetName": "Jane Doe"
                  }
                ]
              }
/post:
  post:
    description: Add a new event post
    queryParameters:
      access_token:
        type: string
        required: true
    body:
      application/json:

```

```
      type: Post
    responses:
      200: Ok

/organizer:
/{organizerId}:
/events:
  get:
    description: Lists events for the organizer
    queryParameters:
      access_token:
        type: string
        required: true
    responses:
      200:
        body:
          application/json:
            type: Event[]
```

