# Temat projektu:

System sprzedaży biletów na różne typy wydarzeń.

**Przedmiot:**

Technologie Aplikacji Webowych II

**Skład grupy:**

Marcin Król

**Nr indeksu:**

34300

**Prowadzący:**

mgr inż. Dariusz Piwko

# 1. Diagramy przypadków użycia.

Na diagramach przypadków użycia aplikacji webowej systemu sprzedaży biletów wyróżniono trzy główne moduły:

- **moduł użytkownika zalogowanego (MUZ)**
- **moduł użytkownika niezalogowanego (MUN)**
- **moduł organizatora (MO)**

Moduł użytkownika niezalogowanego (MUN) reprezentuje interakcje zachodzące między użytkownikami niezalogowanymi, a systemem (rys. 1.1.).

Moduł użytkownika zalogowanego (MUZ) przedstawia interakcje między użytkownikami zalogowanymi, systemem bankowości oraz systemem sprzedaży biletów (rys. 1.2.).

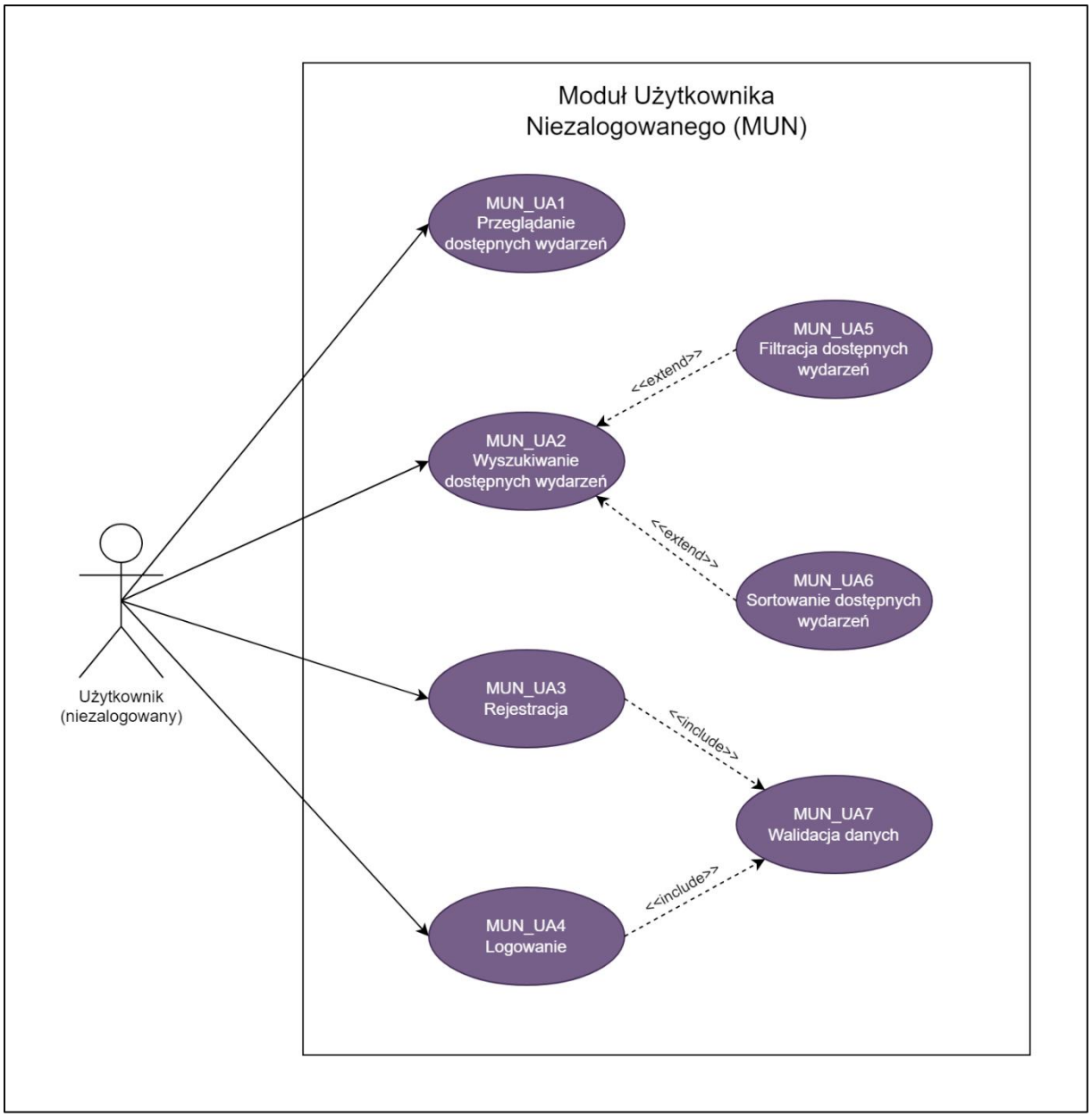
Moduł organizatora (MO) reprezentuje interakcje pomiędzy organizatorami, a systemem (rys. 1.3.).

W systemie wyróżniono czterech głównych aktorów:

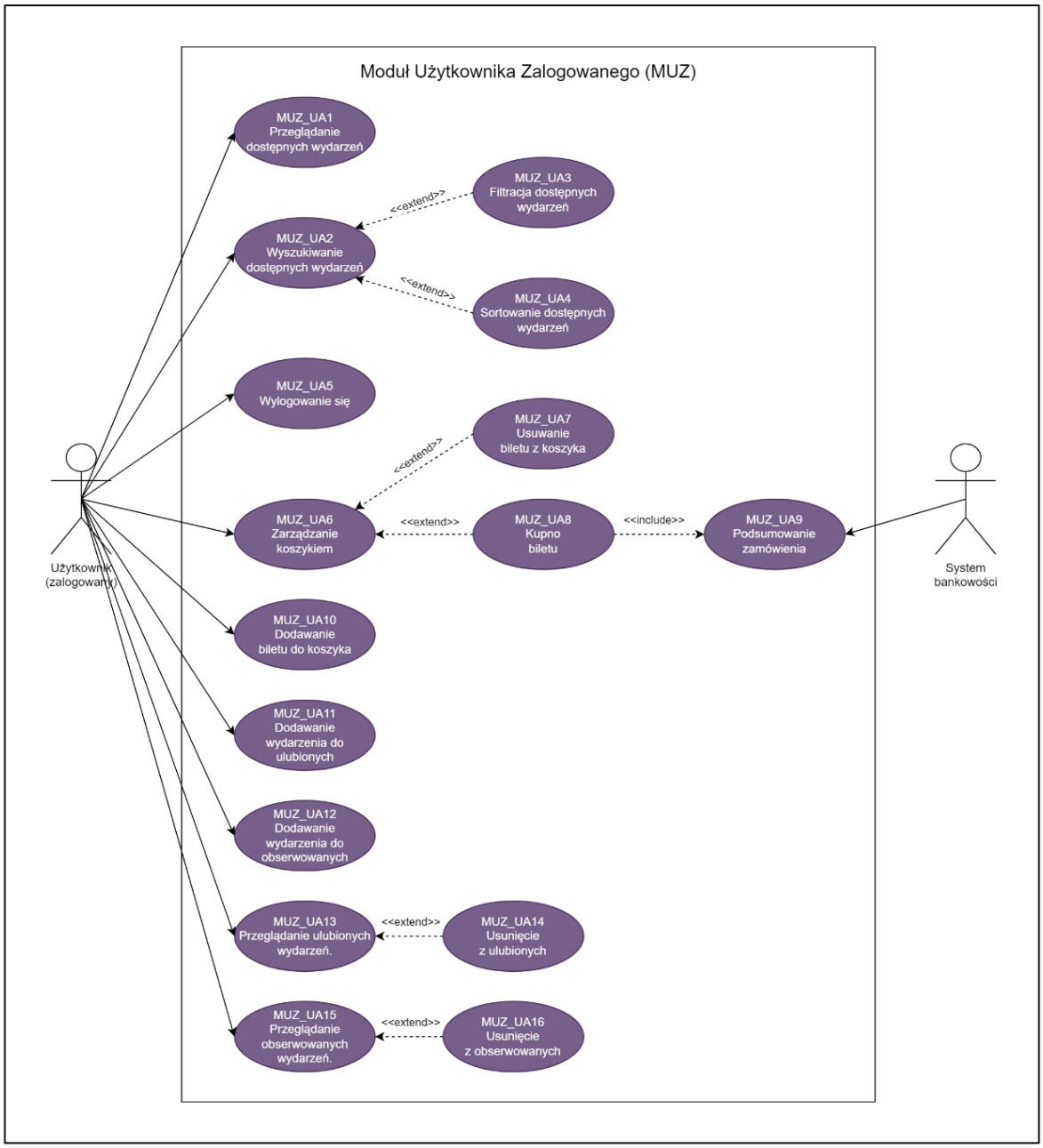
- Użytkownik niezalogowany - to domyślnie klient, który odwiedza aplikację webową bądź mobilną po raz pierwszy. Użytkownik niezalogowany posiada możliwość przeglądania i wyszukiwania dostępnych wydarzeń, może utworzyć nowe konto w systemie wypełniając poprawnie odpowiedni formularz dostępny z pozycji interfejsu użytkownika. Użytkownik niezalogowany może, wypełniając poprawnie formularz logowania, zalogować się do systemu, tym samym zyskując dostęp do nowych funkcjonalności systemu.
- Użytkownik zalogowany - to klient posiadający już konto w systemie. Użytkownik zalogowany posiada dostęp do podstawowych funkcji systemu tj. wyszukiwanie i przeglądanie dostępnych wydarzeń, ale również do funkcjonalności tj. dodawanie wydarzeń do ulubionych, obserwowanych czy koszyka, a także zakup biletów.
- Organizator - to firma lub organizacja zewnętrzna, która pomyślnie przeszła etap rejestracji konta organizatora dostępny z poziomu aplikacji webowej oraz pomyślnie zalogowała się do systemu danymi konta organizatora. Organizator po zalogowaniu zyskuje dostęp do specjalnego panelu administracyjnego w którym

może dodawać, edytować usuwać bądź analizować utworzone przez siebie wydarzenia.

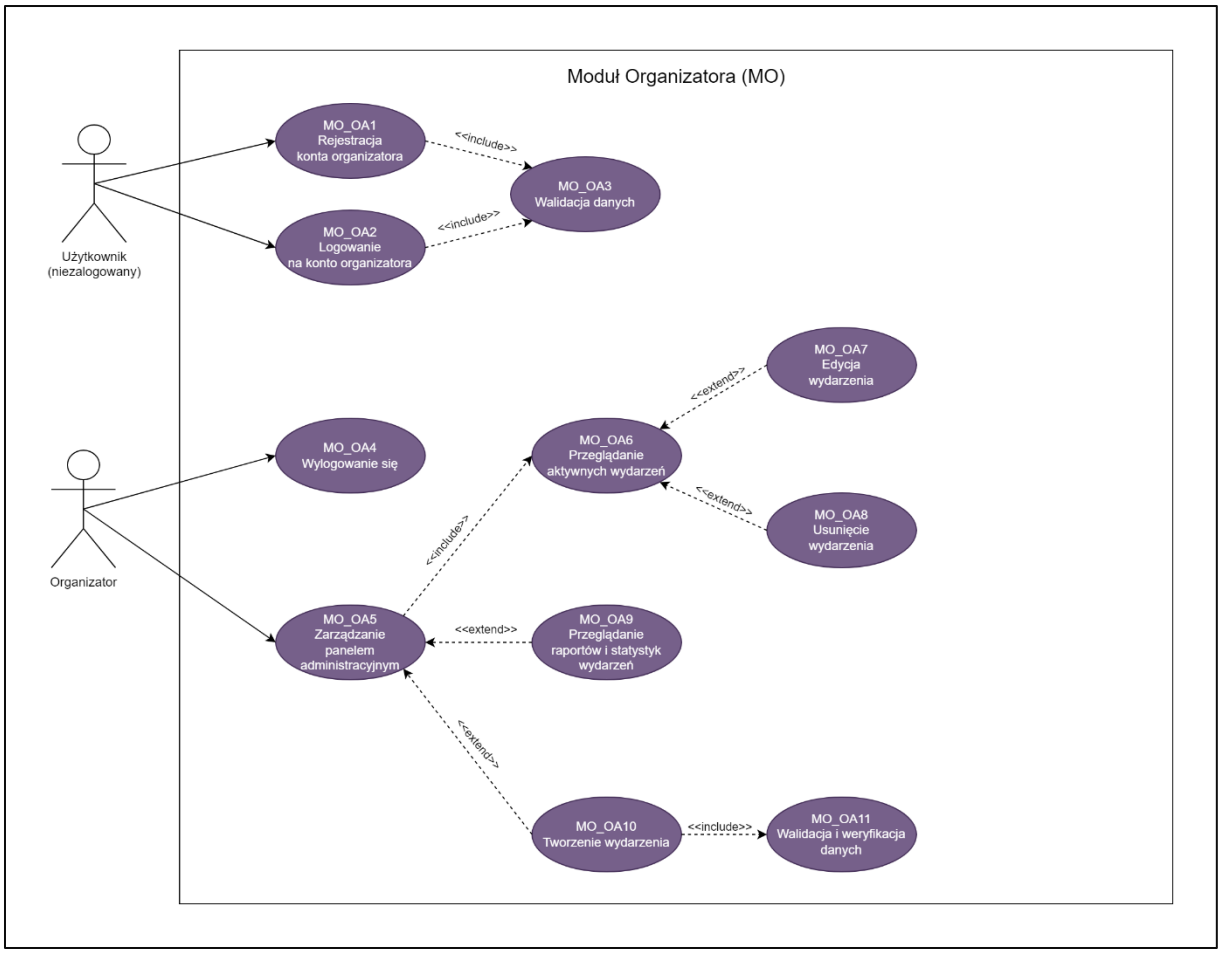
- System bankowości - to firma lub organizacja zewnętrzna realizująca płatność użytkowników zalogowanych w systemie. System bankowości dodatkowo powinien przesyłać informację zwrotną potwierdzającą bądź odrzucającą płatność systemowi w celu finalizacji bądź odrzuceniu zamówienia po stronie. Jako, że system bankowości jest aktorem zewnętrznym (terminator), projekt nie implementuje funkcjonalności systemu bankowości.




*Rys. 1.1. Diagram przypadków użycia MUN.*




*Rys. 1.2. Diagram przypadków użycia MUZ.*




*Rys. 1.3. Diagram przypadków użycia MO.*

## 2. Opis Architektury.

System składa się z dwóch nadrzędnych warstw: aplikacji serwerowej i aplikacji webowej.

**Aplikacja Webowa:** Aplikacja kliencka została zrealizowana przy użyciu frameworka Angular. Projekt podzielony jest na szereg modułów i komponentów w celu zwiększenia czytelności i skalowalności kodu.

- **/components** (Komponenty): W tym module znajdują się różne komponenty aplikacji, które są odpowiedzialne za wyświetlanie i zarządzanie różnymi częściami interfejsu użytkownika.
- **/directives** (Dyrektywy): W tym module znajdują się dyrektywy do manipulowania elementami DOM oraz dodawania specjalnych funkcjonalności do komponentów.
- **/pipes** (Pipe'y): W tym module znajdują się pipe'y pozwalające na przetwarzanie i formatowanie danych przed ich wyświetleniem na interfejsie użytkownika.
- **/services** (Serwisy): W tym module znajdują się serwisy Angulara, które odpowiadają za komunikację z serwerem, przetwarzanie danych oraz zarządzanie logiką aplikacji.
- **/interfaces** (Interfejsy): Tutaj zdefiniowane są różne interfejsy, które pomagają w typowaniu danych oraz utrzymaniu spójności w aplikacji.
- **/assets/img** (Zasoby / Grafiki): W tym module umieszczone są pliki graficzne, takie jak logo czy inne obrazy wykorzystywane w aplikacji.

**Aplikacja Serwerowa:** Aplikacja serwerowa oparta jest na technologiach Node.js i Express.js. Architektura skupia się na realizacji logiki biznesowej, komunikacji z bazą danych MongoDB oraz obsłudze zapytań HTTP.

- **/business** (Logika Biznesowa): W tym module zawarta jest główna logika aplikacji, odpowiedzialna za przetwarzanie danych oraz implementację funkcjonalności biznesowych.

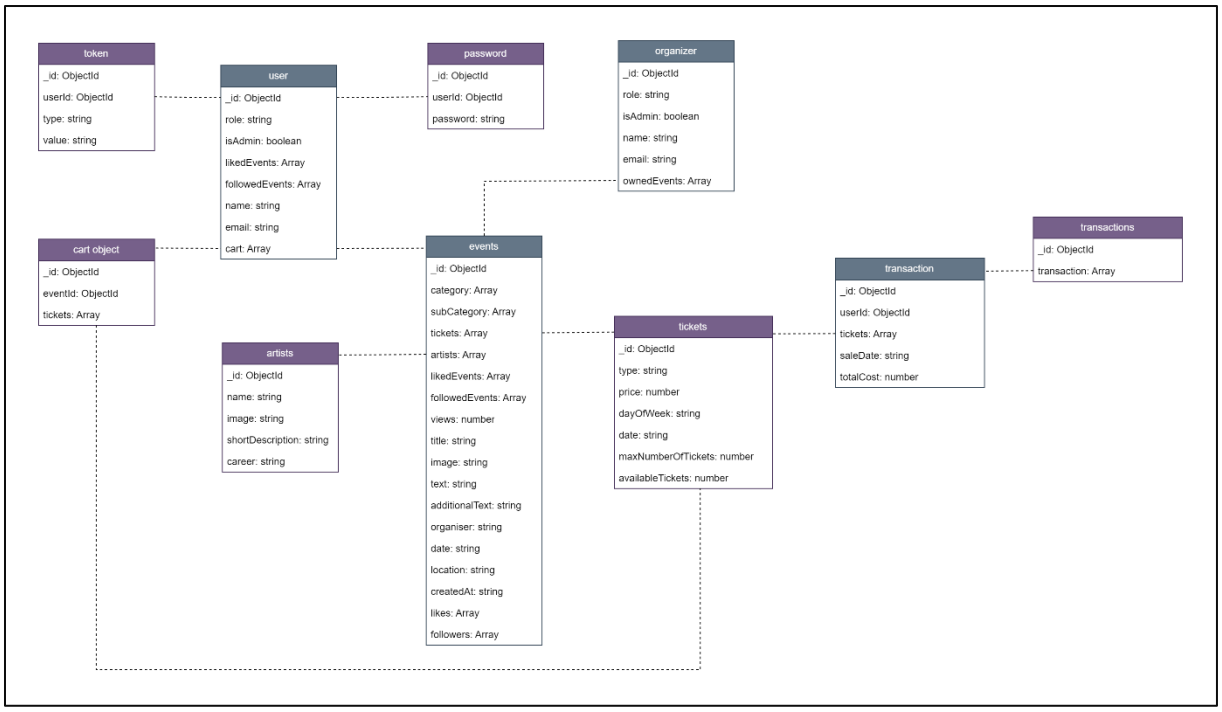
- **/DAO:** Ten moduł obsługuje interakcje z bazą danych MongoDB. Zapewnia dostęp do danych, umożliwiając odczyt, zapis oraz modyfikację informacji w bazie.
- **/middleware:** Moduł zawiera komponenty pośredniczące, które przetwarzają żądania HTTP przed ich przekazaniem do obsługi w warstwie REST.
- **/REST**: W tej części znajdują się endpointy API, które komunikują się z aplikacją klienta. Dzięki nim możliwa jest wymiana danych między serwerem, a aplikacją kliencką.
- **/service:** Ten moduł zawiera specjalizowane usługi, takie jak momentWrapper oraz mongoConverter, które wspierają różne aspekty aplikacji, takie jak manipulacja datami czy konwersja danych do formatu zrozumiałego dla bazy danych.
- **/utils:** Tutaj umieszczone są różnorodne narzędzia, takie jak konfiguracja Swaggera, style dla interfejsu Swagger UI oraz schematy dla Swagger.io.

**Dodatkowe założenia:**

- Nazwy zmiennych, funkcji i komentarze pisane są w języku angielskim.
- Nazwy zmiennych i funkcji pisane są w standardzie notacji camelCase.

# 3. Wybór bazy danych (diagram ERD).

W projekcie zdecydowano się na wykorzystanie bazy danych MongoDB. Na przedstawionym na rysunku 3.1. diagramie ERD wyróżniono kolekcje i dokumenty znajdujące się w bazie danych. Jako, że baza MongoDB jest bazą nierelacyjną, połączenia między kolekcjami symbolizują jedynie referencje lub zagnieżdżenia dokumentów, które umożliwiają powiązanie danych między sobą.




*Rys. 3.1. Schemat ERD bazy danych.*