# Project Ticketmaster - System sprzedaży biletów

## 1. Opis projektu i założenia projektowe

### Cel projektu

Aplikacja **Ticketmaster** to kompleksowy system do sprzedaży biletów na wydarzenia, umożliwiający organizatorom tworzenie i zarządzanie wydarzeniami, a klientom przeglądanie i kupowanie biletów.

### Założenia projektowe

#### Funkcjonalność główna:

* **Zarządzanie wydarzeniami** - tworzenie, edycja, zatwierdzanie wydarzeń
* **System sprzedaży biletów** - rezerwacja, kupno, anulowanie biletów
* **Zarządzanie użytkownikami** - rejestracja, logowanie, role systemowe
* **System płatności** - obsługa płatności za bilety
* **Panel administracyjny** - zarządzanie całym systemem

#### Technologie:

* **Backend**: Spring Boot 3.5.0, Spring Security, Spring Data JPA
* **Frontend**: Thymeleaf, Bootstrap 5.3, JavaScript
* **Baza danych**: MySQL 8.0
* **Konteneryzacja**: Docker, Docker Compose

## 2. Role aplikacyjne i funkcjonalności

### A. ADMIN (Administrator)

**Uprawnienia**: Pełny dostęp do systemu

**Funkcjonalności**: - Zarządzanie użytkownikami (przeglądanie, usuwanie) - Zatwierdzanie/odrzucanie wydarzeń organizatorów - Usuwanie wydarzeń z systemu - Przeglądanie statystyk systemu - Moderacja treści - Zarządzanie rolami użytkowników

### B. ORGANIZER (Organizator wydarzeń)

**Uprawnienia**: Tworzenie i zarządzanie własnymi wydarzeniami

**Funkcjonalności**: - Tworzenie nowych wydarzeń - Edycja własnych wydarzeń (przed zatwierdzeniem) - Przeglądanie sprzedanych biletów na swoje wydarzenia - Dashboard z statystykami sprzedaży - Zarządzanie informacjami o wydarzeniu - Przeglądanie danych klientów kupujących bilety

### C. CLIENT (Klient)

**Uprawnienia**: Przeglądanie i kupowanie biletów

**Funkcjonalności**: - Przeglądanie dostępnych wydarzeń - Filtrowanie wydarzeń (kategoria, data, cena) - Wyszukiwanie wydarzeń - Kupowanie biletów - Rezerwacja biletów - Zarządzanie własnymi biletami - Anulowanie biletów - Przeglądanie historii zakupów

## 3. Przypadki użycia (Use Cases)

### Grupa: ADMINISTRATOR

#### UC-01: Zatwierdzanie wydarzeń

**Aktor**: Administrator **Opis**: Administrator przegląda i zatwierdza wydarzenia utworzone przez organizatorów **Warunki wstępne**: Administrator jest zalogowany **Przepływ główny**: 1. Administrator przechodzi do sekcji “Pending Events” 2. Przegląda szczegóły wydarzenia 3. Zatwierdza lub odrzuca wydarzenie 4. System aktualizuje status wydarzenia 5. Organizator otrzymuje powiadomienie o decyzji

#### UC-02: Zarządzanie użytkownikami

**Aktor**: Administrator **Opis**: Administrator zarządza kontami użytkowników w systemie **Warunki wstępne**: Administrator jest zalogowany **Przepływ główny**: 1. Administrator przechodzi do sekcji “Users” 2. Przegląda listę użytkowników 3. Może usunąć konto użytkownika 4. System aktualizuje bazę danych

### Grupa: ORGANIZER

#### UC-03: Tworzenie wydarzenia

**Aktor**: Organizator **Opis**: Organizator tworzy nowe wydarzenie w systemie **Warunki wstępne**: Organizator jest zalogowany **Przepływ główny**: 1. Organizator przechodzi do formularza tworzenia wydarzenia 2. Wypełnia dane wydarzenia (nazwa, data, lokalizacja, cena, opis) 3. Wybiera kategorię wydarzenia 4. Określa liczbę dostępnych miejsc 5. Zatwierdza formularz 6. System zapisuje wydarzenie ze statusem “PENDING\_APPROVAL”

#### UC-04: Przeglądanie sprzedanych biletów

**Aktor**: Organizator **Opis**: Organizator przegląda bilety sprzedane na swoje wydarzenie **Warunki wstępne**: Organizator jest zalogowany, ma utworzone wydarzenia **Przepływ główny**: 1. Organizator wybiera wydarzenie z listy 2. Przechodzi do sekcji “Tickets” 3. Przegląda listę sprzedanych biletów 4. Może filtrować bilety według statusu 5. Przegląda dane klientów

### Grupa: CLIENT

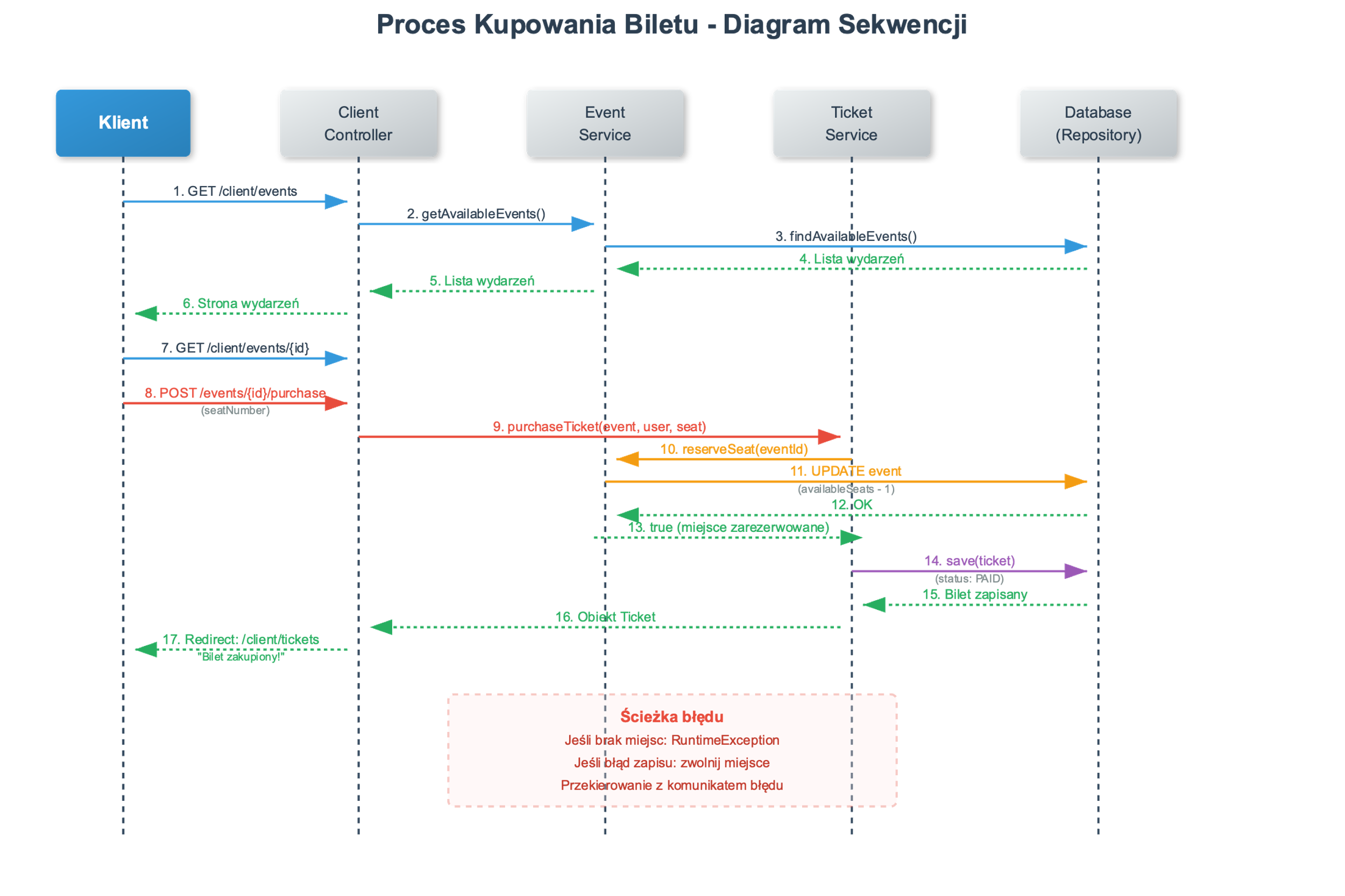
#### UC-05: Kupowanie biletu

**Aktor**: Klient **Opis**: Klient kupuje bilet na wybrane wydarzenie **Warunki wstępne**: Klient jest zalogowany, wydarzenie jest dostępne **Przepływ główny**: 1. Klient przegląda listę dostępnych wydarzeń 2. Wybiera szczegóły wydarzenia 3. Wybiera miejsce (opcjonalnie) 4. Potwierdza zakup 5. System przetwarza płatność 6. System generuje bilet ze statusem “PAID” 7. Zmniejsza liczbę dostępnych miejsc

#### UC-06: Rezerwacja biletu

**Aktor**: Klient **Opis**: Klient rezerwuje bilet z możliwością późniejszej płatności **Warunki wstępne**: Klient jest zalogowany, wydarzenie jest dostępne **Przepływ główny**: 1. Klient wybiera wydarzenie 2. Wybiera opcję “Reserve Ticket” 3. System tworzy bilet ze statusem “RESERVED” 4. Klient może później dokonać płatności lub anulować rezerwację

## 4. Diagram sekwencji - Proces kupowania biletu



## 5. Struktura bazy danych

### Tabela: users

CREATE TABLE users (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 first\_name VARCHAR(255) NOT NULL,  
 last\_name VARCHAR(255) NOT NULL,  
 enabled BOOLEAN DEFAULT TRUE,  
 created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP  
);

### Tabela: roles

CREATE TABLE roles (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(50) UNIQUE NOT NULL,  
 description VARCHAR(255)  
);

### Tabela: user\_roles (relacja wiele-do-wielu)

CREATE TABLE user\_roles (  
 user\_id BIGINT,  
 role\_id BIGINT,  
 PRIMARY KEY (user\_id, role\_id),  
 FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,  
 FOREIGN KEY (role\_id) REFERENCES roles(id) ON DELETE CASCADE  
);

### Tabela: events

CREATE TABLE events (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 event\_date DATETIME NOT NULL,  
 location VARCHAR(255) NOT NULL,  
 price DECIMAL(10,2) NOT NULL,  
 available\_seats INT NOT NULL,  
 total\_seats INT NOT NULL,  
 status ENUM('PENDING\_APPROVAL', 'APPROVED', 'REJECTED', 'CANCELLED', 'COMPLETED'),  
 category ENUM('CONCERT', 'THEATRE', 'CINEMA', 'SPORT', 'CONFERENCE', 'OTHER'),  
 organizer\_id BIGINT NOT NULL,  
 created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 FOREIGN KEY (organizer\_id) REFERENCES users(id) ON DELETE CASCADE  
);

### Tabela: tickets

CREATE TABLE tickets (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 ticket\_number VARCHAR(255) UNIQUE NOT NULL,  
 event\_id BIGINT NOT NULL,  
 user\_id BIGINT NOT NULL,  
 price DECIMAL(10,2) NOT NULL,  
 seat\_number VARCHAR(255),  
 status ENUM('RESERVED', 'PAID', 'CANCELLED', 'USED') DEFAULT 'RESERVED',  
 purchase\_date DATETIME DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (event\_id) REFERENCES events(id) ON DELETE CASCADE,  
 FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE  
);

## Ograniczenia i relacje

### Ograniczenia integralności:

1. **users.username** - UNIQUE, NOT NULL
2. **users.email** - UNIQUE, NOT NULL
3. **events.price** - > 0 (walidacja w aplikacji)
4. **events.total\_seats** - > 0 (walidacja w aplikacji)
5. **events.available\_seats** - >= 0, <= total\_seats
6. **tickets.ticket\_number** - UNIQUE, AUTO-GENERATED (UUID)

### Relacje:

1. **users (1) <-> (N) events** - jeden użytkownik może organizować wiele wydarzeń
2. **users (N) <-> (N) roles** - użytkownik może mieć wiele ról
3. **events (1) <-> (N) tickets** - jedno wydarzenie może mieć wiele biletów
4. **users (1) <-> (N) tickets** - jeden użytkownik może mieć wiele biletów

### Indeksy:

* idx\_users\_username na users(username)
* idx\_users\_email na users(email)
* idx\_events\_status na events(status)
* idx\_events\_date na events(event\_date)
* idx\_tickets\_status na tickets(status)

## Bezpieczeństwo

### Autoryzacja:

* **Spring Security** z kontrolą dostępu opartą na rolach
* **BCrypt** do hashowania haseł
* **Session-based authentication**

### Walidacja:

* **Bean Validation** (JSR-303) w encjach
* **Walidacja po stronie klienta** (JavaScript)
* **Walidacja biznesowa** w serwisach

### Transakcje:

* **@Transactional** na metodach serwisowych
* **Rollback** w przypadku błędów przy zakupie biletów
* **Atomowość** operacji rezerwacji miejsc