# Sprawozdanie - System Zarządzania Biletami Ticketmaster

## 1. Opis Aplikacji

### 1.1 Wprowadzenie

Ticketmaster to webowa aplikacja do zarządzania wydarzeniami i sprzedaży biletów, zbudowana w oparciu o framework Spring Boot. System umożliwia organizatorom tworzenie wydarzeń, klientom zakup biletów, a administratorom zarządzanie całą platformą.

**Główne cechy systemu:** - System wieloużytkownikowy z trzema rolami (Admin, Organizer, Client) - Pełny cykl zarządzania wydarzeniami od tworzenia do sprzedaży biletów - Interaktywny system wyboru miejsc z mapą wizualną - Responsywny interfejs użytkownika dostosowany do urządzeń mobilnych - Bezpieczna autoryzacja i uwierzytelnianie z Spring Security - Konteneryzacja z Docker dla łatwego wdrożenia - REST API dla funkcji czasu rzeczywistego

### 1.2 Architektura Aplikacji

Aplikacja została zbudowana w architekturze MVC (Model-View-Controller) z wykorzystaniem następujących technologii:

**Backend:** - **Spring Boot 3.5.0** - główny framework aplikacyjny - **Spring Security** - autoryzacja i uwierzytelnianie  
- **Spring Data JPA** - warstwa dostępu do danych - **Hibernate** - ORM (Object-Relational Mapping) - **MySQL 8.0** - baza danych - **Bean Validation (JSR-303)** - walidacja danych - **Maven 3.9.9** - zarządzanie zależnościami

**Frontend:** - **Thymeleaf** - silnik szablonów HTML po stronie serwera - **Bootstrap 5.3** - framework CSS z responsywnym designem - **Bootstrap Icons** - kompletny zestaw ikon - **JavaScript (ES6+)** - interaktywność po stronie klienta - **Chart.js** - wykresy i wizualizacje (importowane przez CDN)

**Narzędzia budowania:** - **Maven 3.9.9** - zarządzanie zależnościami i budowanie projektu - **Docker** - konteneryzacja aplikacji - **Docker Compose** - orkiestracja kontenerów

### 1.3 Struktura Projektu

ticketmaster/  
├── src/  
│ ├── main/  
│ │ ├── java/com/example/ticketmaster/  
│ │ │ ├── config/ # Konfiguracja aplikacji  
│ │ │ │ ├── SecurityConfig.java # Konfiguracja Spring Security  
│ │ │ │ └── DbInitializer.java # Inicjalizacja danych startowych  
│ │ │ ├── controller/ # Kontrolery REST i MVC  
│ │ │ │ ├── AdminController.java # Panel administratora  
│ │ │ │ ├── AuthController.java # Autoryzacja i rejestracja  
│ │ │ │ ├── ClientController.java # Panel klienta  
│ │ │ │ ├── CustomErrorController.java # Obsługa błędów  
│ │ │ │ ├── HomeController.java # Strona główna  
│ │ │ │ ├── OrganizerController.java # Panel organizatora  
│ │ │ │ └── SeatApiController.java # REST API dla miejsc  
│ │ │ ├── dto/ # Data Transfer Objects  
│ │ │ │ ├── CreateEventDto.java # DTO do tworzenia wydarzeń  
│ │ │ │ ├── EventDto.java # DTO wyświetlania wydarzeń  
│ │ │ │ ├── TicketDto.java # DTO biletów  
│ │ │ │ ├── UserDto.java # DTO użytkowników  
│ │ │ │ └── UserRegistrationDto.java # DTO rejestracji  
│ │ │ ├── entity/ # Encje JPA (modele danych)  
│ │ │ │ ├── Event.java # Wydarzenia  
│ │ │ │ ├── Role.java # Role użytkowników  
│ │ │ │ ├── Ticket.java # Bilety  
│ │ │ │ └── User.java # Użytkownicy  
│ │ │ ├── mapper/ # Konwertery Entity ↔ DTO  
│ │ │ │ ├── EventMapper.java  
│ │ │ │ ├── TicketMapper.java  
│ │ │ │ └── UserMapper.java  
│ │ │ ├── repository/ # Repozytoria Spring Data JPA  
│ │ │ │ ├── EventRepository.java  
│ │ │ │ ├── RoleRepository.java  
│ │ │ │ ├── TicketRepository.java  
│ │ │ │ └── UserRepository.java  
│ │ │ └── service/ # Logika biznesowa  
│ │ │ ├── EventService.java # Zarządzanie wydarzeniami  
│ │ │ ├── TicketService.java # Zarządzanie biletami  
│ │ │ └── UserService.java # Zarządzanie użytkownikami  
│ │ └── resources/  
│ │ ├── static/ # Zasoby statyczne  
│ │ │ ├── css/  
│ │ │ │ └── styles.css # Style aplikacji  
│ │ │ └── js/  
│ │ │ └── script.js # JavaScript funkcjonalności  
│ │ ├── templates/ # Szablony Thymeleaf  
│ │ │ ├── admin/ # Szablony panelu administratora  
│ │ │ │ ├── dashboard.html # Dashboard admina  
│ │ │ │ ├── events.html # Zarządzanie wydarzeniami  
│ │ │ │ └── users.html # Zarządzanie użytkownikami  
│ │ │ ├── auth/ # Szablony autoryzacji  
│ │ │ │ ├── login.html # Strona logowania  
│ │ │ │ └── register.html # Strona rejestracji  
│ │ │ ├── client/ # Szablony panelu klienta  
│ │ │ │ ├── dashboard.html # Dashboard klienta  
│ │ │ │ ├── events.html # Przeglądanie wydarzeń  
│ │ │ │ ├── event-details.html # Szczegóły wydarzenia  
│ │ │ │ └── tickets.html # Zarządzanie biletami  
│ │ │ ├── organizer/ # Szablony panelu organizatora  
│ │ │ │ ├── dashboard.html # Dashboard organizatora  
│ │ │ │ ├── events.html # Lista wydarzeń  
│ │ │ │ ├── create-event.html # Tworzenie wydarzenia  
│ │ │ │ ├── edit-event.html # Edycja wydarzenia  
│ │ │ │ └── event-tickets.html # Bilety wydarzenia  
│ │ │ ├── fragments/ # Komponenty wielokrotnego użytku  
│ │ │ │ └── layout.html # Layout i fragmenty  
│ │ │ ├── error/ # Strony błędów  
│ │ │ │ └── error.html # Uniwersalna strona błędów  
│ │ │ └── index.html # Strona główna (publiczna)  
│ │ └── application.properties # Konfiguracja aplikacji  
│ └── test/ # Testy jednostkowe  
│ └── java/com/example/ticketmaster/  
│ └── TicketmasterApplicationTests.java  
├── .mvn/wrapper/ # Maven Wrapper  
├── target/ # Skompilowane pliki (generowane)  
├── pom.xml # Konfiguracja Maven  
├── Dockerfile # Definicja obrazu Docker  
├── docker-compose.yml # Orkiestracja kontenerów  
├── .gitignore # Wykluczenia Git  
├── .gitattributes # Atrybuty Git  
├── mvnw # Maven Wrapper (Linux/macOS)  
├── mvnw.cmd # Maven Wrapper (Windows)  
└── README.md # Dokumentacja projektu

## 2. Funkcjonalności Systemu

### 2.1 Role Użytkowników

System implementuje trzy role użytkowników z ścisłym rozgraniczeniem uprawnień:

#### 1. ADMIN - Administrator systemu

**Uprawnienia:** - Zarządzanie wszystkimi użytkownikami (przeglądanie, usuwanie) - Zatwierdzanie/odrzucanie wydarzeń organizatorów - Podgląd wszystkich wydarzeń i biletów w systemie - Dostęp do panelu administracyjnego z statystykami systemu - Usuwanie wydarzeń (wszystkich, nie tylko własnych) - Pełny dostęp do wszystkich funkcji systemu

**Główne endpointy:** - /admin/dashboard - Panel z ogólnymi statystykami - /admin/users - Zarządzanie użytkownikami - /admin/events - Zarządzanie wszystkimi wydarzeniami - /admin/events/{id}/approve - Zatwierdzanie wydarzeń - /admin/events/{id}/reject - Odrzucanie wydarzeń - /admin/events/{id}/delete - Usuwanie wydarzeń - /admin/users/{id}/delete - Usuwanie użytkowników

#### 2. ORGANIZER - Organizator wydarzeń

**Uprawnienia:** - Tworzenie nowych wydarzeń (wymagają zatwierdzenia przez admin) - Edycja własnych wydarzeń (ograniczona po zatwierdzeniu) - Podgląd sprzedanych biletów dla swoich wydarzeń - Dostęp do statystyk sprzedaży własnych wydarzeń - Zarządzanie tylko własnymi wydarzeniami

**Główne endpointy:** - /organizer/dashboard - Panel organizatora ze statystykami - /organizer/events - Lista własnych wydarzeń - /organizer/events/create - Tworzenie nowego wydarzenia - /organizer/events/{id}/edit - Edycja wydarzenia - /organizer/events/{id}/tickets - Podgląd sprzedanych biletów

#### 3. CLIENT - Klient

**Uprawnienia:** - Przeglądanie zatwierdzonych i dostępnych wydarzeń - Zakup biletów z wyborem konkretnych miejsc - Rezerwacja biletów (z późniejszą możliwością opłacenia) - Zarządzanie własnymi biletami (przeglądanie, anulowanie) - Dostęp tylko do publicznych treści

**Główne endpointy:** - /client/dashboard - Panel klienta z podsumowaniem biletów - /client/events - Katalog dostępnych wydarzeń - /client/events/{id} - Szczegóły wydarzenia z możliwością zakupu - /client/events/{id}/purchase - Zakup biletu - /client/events/{id}/reserve - Rezerwacja biletu - /client/tickets - Zarządzanie własnymi biletami - /client/tickets/{id}/pay - Opłacenie zarezerwowanego biletu - /client/tickets/{id}/cancel - Anulowanie biletu

### 2.2 Główne Moduły

#### Moduł Uwierzytelniania (AuthController)

**Funkcjonalności:** - **Rejestracja użytkowników** (GET/POST /register) - Walidacja danych (username, email, hasło, dane osobowe) - Wybór roli podczas rejestracji (Client/Organizer) - Automatyczne hashowanie haseł BCrypt - Sprawdzanie unikalności username i email

* **Logowanie** (GET /login)
  + Integracja z Spring Security
  + Wyświetlanie kont demonstracyjnych
  + Przekierowanie na odpowiedni dashboard po zalogowaniu
* **Zarządzanie sesjami**
  + Automatyczne przekierowanie (GET /dashboard)
  + Bezpieczne wylogowanie z czyszczeniem sesji

#### Moduł Wydarzeń

**Zarządzanie cyklem życia wydarzenia:** 1. **Tworzenie** (Organizer) → **Moderacja** (Admin) → **Publikacja** → **Sprzedaż**

**Statusy wydarzeń:** - PENDING\_APPROVAL - oczekuje na zatwierdzenie przez administratora - APPROVED - zatwierdzone, dostępne do sprzedaży biletów - REJECTED - odrzucone przez administratora - CANCELLED - anulowane - COMPLETED - zakończone

**Kategorie wydarzeń:** - CONCERT - Koncerty muzyczne - THEATRE - Spektakle teatralne - CINEMA - Seanse kinowe - SPORT - Wydarzenia sportowe - CONFERENCE - Konferencje i seminaria - OTHER - Inne wydarzenia

#### Moduł Biletów

**Funkcjonalności zakupu:** - **Natychmiastowy zakup** - bezpośrednie przejście do statusu PAID - **Rezerwacja** - tymczasowe zablokowanie miejsca ze statusem RESERVED

**Statusy biletów:** - RESERVED - zarezerwowane, oczekują na opłacenie - PAID - opłacone i potwierdzone - CANCELLED - anulowane (miejsce zostaje zwolnione) - USED - wykorzystane (zeskanowane na wejściu)

**System zarządzania miejscami:** - Automatyczne zarządzanie dostępnością miejsc - Sprawdzanie w czasie rzeczywistym przez REST API - Zabezpieczenie przed podwójną rezerwacją tego samego miejsca

### 2.3 System Wyboru Miejsc

**Interaktywna mapa miejsc:** - Dynamiczne generowanie układu miejsc na podstawie całkowitej liczby - Automatyczny podział na rzędy i kolumny - Wizualne oznaczenie dostępnych/zajętych/wybranych miejsc - Integracja z REST API do sprawdzania dostępności

**REST API dla miejsc (SeatApiController):** - GET /api/events/{eventId}/occupied-seats - Lista zajętych miejsc - POST /api/events/{eventId}/check-seat - Sprawdzenie dostępności konkretnego miejsca

**Zabezpieczenia:** - Walidacja dostępności przed finalizacją zakupu - Sprawdzanie w czasie rzeczywistym (inne osoby mogą kupować równocześnie) - Automatyczne odświeżanie mapy w przypadku konfliktu

### 2.4 Moduł Zarządzania Błędami

**CustomErrorController:** - Centralna obsługa błędów HTTP (404, 500, 403, itp.) - Przyjazne komunikaty błędów dla użytkowników - Przekierowanie na odpowiednie strony błędów - Logowanie błędów dla administratorów

## 3. Model Danych

### 3.1 Szczegółowy Schemat Bazy Danych

-- Tabela użytkowników  
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 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Tabela ról  
CREATE TABLE roles (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(50) UNIQUE NOT NULL,  
 description VARCHAR(255)  
);  
  
-- Tabela łącząca użytkowników z rolami  
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 wydarzeń  
CREATE TABLE events (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 event\_date TIMESTAMP NOT NULL,  
 location VARCHAR(255) NOT NULL,  
 price DECIMAL(10,2) NOT NULL,  
 available\_seats INT NOT NULL,  
 total\_seats INT NOT NULL,  
 status VARCHAR(50) DEFAULT 'PENDING\_APPROVAL',  
 category VARCHAR(50) NOT NULL,  
 organizer\_id BIGINT NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 FOREIGN KEY (organizer\_id) REFERENCES users(id) ON DELETE CASCADE  
);  
  
-- Tabela biletów  
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),  
 seat\_number VARCHAR(50),  
 status VARCHAR(50) DEFAULT 'RESERVED',  
 purchase\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 FOREIGN KEY (event\_id) REFERENCES events(id) ON DELETE CASCADE,  
 FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE  
);

### 3.2 Mapowanie Encji JPA

**Kluczowe relacje:** - **User ↔ Role**: Relacja Many-to-Many (jeden użytkownik może mieć wiele ról) - **User ↔ Event**: Relacja One-to-Many (organizer może mieć wiele wydarzeń) - **User ↔ Ticket**: Relacja One-to-Many (użytkownik może mieć wiele biletów) - **Event ↔ Ticket**: Relacja One-to-Many (wydarzenie ma wiele biletów)

**Automatyczne zarządzanie:** - Timestamps (created\_at, updated\_at) z adnotacjami JPA - Generowanie UUID dla numerów biletów - Kaskadowe usuwanie powiązanych rekordów

### 3.3 Repozytoria i Zapytania

**EventRepository - zapytania niestandardowe:** - findByStatus() - wydarzenia według statusu - findByOrganizer() - wydarzenia konkretnego organizatora - findAvailableEvents() - dostępne wydarzenia (zatwierdzone + przyszłe + wolne miejsca) - findByStatusAndEventDateAfter() - filtrowanie według statusu i daty

**TicketRepository - zapytania niestandardowe:** - findByUser() - bilety konkretnego użytkownika - findByEvent() - bilety konkretnego wydarzenia - findByUserOrderByPurchaseDateDesc() - bilety użytkownika sortowane według daty zakupu

## 4. Frontend i Interfejs Użytkownika

### 4.1 Architektura Frontend

**Technologie:** - **Thymeleaf 3.0** - silnik szablonów po stronie serwera - **Bootstrap 5.3** - responsywny framework CSS - **Bootstrap Icons** - ikony - **Vanilla JavaScript ES6+** - bez dodatkowych frameworków - **Chart.js** - wykresy dla dashboardów (importowane przez CDN)

### 4.2 System Szablonów

**Fragmenty wielokrotnego użytku (fragments/layout.html):** - head(title) - sekcja HEAD z meta tagami i zasobami CSS/JS - navbar - responsywna nawigacja z menu użytkownika - sidebar-admin - boczne menu dla administratora - sidebar-organizer - boczne menu dla organizatora - sidebar-client - boczne menu dla klienta - footer - stopka z zasobami JavaScript

**Responsywny layout:** - Flexbox layout dla dashboardów - Zwijane sidebar na urządzeniach mobilnych - Bootstrap breakpoints dla różnych rozmiarów ekranów

### 4.3 Funkcjonalności JavaScript

**Podstawowe funkcje (script.js):** - Automatyczne ukrywanie alertów po 5 sekundach - Inicjalizacja tooltipów Bootstrap - Walidacja formularzy po stronie klienta - Potwierdzenia przed usunięciem elementów - Filtrowanie i wyszukiwanie w czasie rzeczywistym

**Interaktywny system miejsc:** - Dynamiczne generowanie mapy miejsc - Komunikacja z REST API dla sprawdzania dostępności - Walidacja wyboru przed przesłaniem formularza - Responsywne dostosowanie do rozmiaru ekranu

**Funkcje wyszukiwania i filtrowania:** - Filtrowanie wydarzeń według kategorii - Wyszukiwanie według nazwy i lokalizacji - Sortowanie wyników (data, cena, nazwa) - Filtrowanie biletów według statusu

### 4.4 Style CSS

**Design System:** - Zmienne CSS dla kolorów głównych - Spójne komponenty UI (karty, przyciski, formularze) - Hover efekty i transycje - Responsive typography

**Komponenty specjalne:** - Dashboard cards z kolorową krawędzią - Stylizowane bilety z perforacją - Interaktywne przyciski miejsc - Loading states i animacje

## 5. Instalacja i Konfiguracja

### 5.1 Wymagania Systemowe

* **Java 17** lub nowsza
* **Maven 3.6+**
* **MySQL 8.0+**
* **Docker** i **Docker Compose** (opcjonalnie)

### 5.2 Instalacja z Wykorzystaniem Docker

Jest to najprostsza metoda uruchomienia aplikacji.

1. **Sklonuj repozytorium:**

git clone <repository\_url>  
cd ticketmaster

1. **Uruchom aplikację używając Docker Compose:**

docker-compose up --build

**Docker Compose automatycznie:** - Zbuduje obraz aplikacji Spring Boot - Uruchomi kontener MySQL z odpowiednimi zmiennymi środowiskowymi - Skonfiguruje sieć między kontenerami - Uruchomi aplikację na porcie 8080 - Utworzy volume dla persystencji danych MySQL

1. **Dostęp do aplikacji:**
   * Aplikacja: http://localhost:8080
   * MySQL: localhost:3306 (dla zewnętrznych połączeń)

### 5.3 Instalacja Manualna

#### Krok 1: Konfiguracja Bazy Danych

1. **Zainstaluj MySQL 8.0:**

# Ubuntu/Debian  
sudo apt-get install mysql-server  
  
# macOS (używając Homebrew)  
brew install mysql

1. **Utwórz bazę danych:**

CREATE DATABASE ticketmaster;  
CREATE USER 'ticketmaster'@'localhost' IDENTIFIED BY 'ticketmaster';  
GRANT ALL PRIVILEGES ON ticketmaster.\* TO 'ticketmaster'@'localhost';  
FLUSH PRIVILEGES;

#### Krok 2: Konfiguracja Aplikacji

**Edytuj plik** application.properties:

# Połączenie z bazą danych  
spring.datasource.url=jdbc:mysql://localhost:3306/ticketmaster?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC  
spring.datasource.username=ticketmaster  
spring.datasource.password=ticketmaster  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
  
# Konfiguracja JPA/Hibernate  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
  
# Konfiguracja Thymeleaf  
spring.thymeleaf.cache=false  
  
# Serwer  
server.port=8080  
  
# Logowanie  
logging.level.org.springframework.security=DEBUG  
logging.level.com.example.ticketmaster=DEBUG  
  
# Format daty dla Jackson  
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss  
spring.jackson.time-zone=Europe/Warsaw

#### Krok 3: Budowanie i Uruchamianie

1. **Używając Maven Wrapper (zalecane):**

# Linux/macOS  
./mvnw clean package  
./mvnw spring-boot:run  
  
# Windows  
mvnw.cmd clean package  
mvnw.cmd spring-boot:run

1. **Używając zainstalowanego Maven:**

mvn clean package  
mvn spring-boot:run

1. **Uruchamianie skompilowanego JAR:**

java -jar target/ticketmaster-0.0.1-SNAPSHOT.jar

### 5.4 Konfiguracja Docker

**Dockerfile - wieloetapowe budowanie:**

# Etap budowania  
FROM maven:3.9.9-eclipse-temurin-17 AS build  
WORKDIR /app  
COPY . .  
RUN mvn clean package -DskipTests  
  
# Etap produkcyjny  
FROM eclipse-temurin:17-jre  
WORKDIR /app  
COPY --from=build /app/target/\*.jar app.jar  
EXPOSE 8080  
ENTRYPOINT ["java", "-jar", "app.jar"]

**docker-compose.yml - pełna konfiguracja:**

version: '3.8'  
  
services:  
 app:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 container\_name: ticketmaster-app  
 depends\_on:  
 - mysql  
 ports:  
 - "8080:8080"  
 environment:  
 - SPRING\_DATASOURCE\_URL=jdbc:mysql://mysql:3306/ticketmaster?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC  
 - SPRING\_DATASOURCE\_USERNAME=ticketmaster  
 - SPRING\_DATASOURCE\_PASSWORD=ticketmaster  
 - SPRING\_JPA\_HIBERNATE\_DDL\_AUTO=update  
 networks:  
 - ticketmaster-network  
 restart: always  
  
 mysql:  
 image: mysql:8.0  
 container\_name: ticketmaster-mysql  
 environment:  
 - MYSQL\_DATABASE=ticketmaster  
 - MYSQL\_USER=ticketmaster  
 - MYSQL\_PASSWORD=ticketmaster  
 - MYSQL\_ROOT\_PASSWORD=root  
 ports:  
 - "3306:3306"  
 volumes:  
 - mysql-data:/var/lib/mysql  
 - ./mysql-init:/docker-entrypoint-initdb.d  
 networks:  
 - ticketmaster-network  
 restart: always  
 command: --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci  
  
networks:  
 ticketmaster-network:  
 driver: bridge  
  
volumes:  
 mysql-data:

## 6. Konfiguracja Środowiska Produkcyjnego

### 6.1 Deployment na Tomcat

1. **Modyfikacja** pom.xml **dla WAR:**

<packaging>war</packaging>  
  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 <scope>provided</scope>  
</dependency>

1. **Rozszerzenie głównej klasy aplikacji:**

@SpringBootApplication  
public class TicketmasterApplication extends SpringBootServletInitializer {  
   
 @Override  
 protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {  
 return builder.sources(TicketmasterApplication.class);  
 }  
   
 public static void main(String[] args) {  
 SpringApplication.run(TicketmasterApplication.class, args);  
 }  
}

### 6.2 Konfiguracja HTTPS

1. **Generowanie certyfikatu SSL:**

keytool -genkey -alias ticketmaster -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650

1. **Konfiguracja SSL w** application.properties:

server.port=8443  
server.ssl.key-store=classpath:keystore.p12  
server.ssl.key-store-password=yourpassword  
server.ssl.key-store-type=PKCS12  
server.ssl.key-alias=ticketmaster  
server.ssl.enabled=true

### 6.3 Profile środowiskowe

**application-prod.properties:**

# Produkcyjna baza danych  
spring.datasource.url=jdbc:mysql://prod-db:3306/ticketmaster\_prod  
spring.datasource.username=${DB\_USERNAME}  
spring.datasource.password=${DB\_PASSWORD}  
  
# Wyłączenie debug logów  
logging.level.org.springframework.security=WARN  
logging.level.com.example.ticketmaster=INFO  
  
# Cache dla Thymeleaf  
spring.thymeleaf.cache=true  
  
# Profil aktywny  
spring.profiles.active=prod

## 7. Pierwsze Uruchomienie

### 7.1 Automatyczna Inicjalizacja Danych

**DbInitializer automatycznie tworzy:** - Role systemowe (ADMIN, ORGANIZER, CLIENT) - Użytkowników demonstracyjnych z odpowiednimi rolami - Przykładowe dane startowe

### 7.2 Domyślni Użytkownicy

| Rola | Login | Hasło | Opis |
| --- | --- | --- | --- |
| Administrator | admin | admin | Pełen dostęp do systemu |
| Organizator | organizer | organizer | Może tworzyć wydarzenia |
| Klient | client | client | Może kupować bilety |

### 7.3 Weryfikacja Instalacji

1. **Otwórz przeglądarkę:** http://localhost:8080
2. **Sprawdź stronę główną** - powinna wyświetlać się bez błędów
3. **Zaloguj się** używając jednego z domyślnych kont
4. **Przetestuj funkcjonalności:**
   * Admin: przejdź do /admin/dashboard
   * Organizer: utwórz nowe wydarzenie w /organizer/events/create
   * Client: przeglądaj wydarzenia w /client/events

### 7.4 Sprawdzenie połączenia z bazą danych

**W logach aplikacji powinny pojawić się:** - Informacje o udanym połączeniu z MySQL - Automatyczne utworzenie tabel (Hibernate DDL) - Inicjalizacja ról i użytkowników - Brak błędów SQL

## 8. Rozwiązywanie Problemów

### 8.1 Częste Problemy

**Problem: Błąd połączenia z bazą danych**

Solution:  
- Sprawdź czy MySQL jest uruchomiony: systemctl status mysql  
- Zweryfikuj dane dostępowe w application.properties  
- Upewnij się, że baza danych 'ticketmaster' istnieje  
- Sprawdź czy użytkownik ma odpowiednie uprawnienia

**Problem: Port 8080 jest zajęty**

Solution:  
- Zmień port w application.properties: server.port=8081  
- Lub znajdź i zatrzymaj proces: lsof -i :8080, kill -9 PID  
- Lub użyj innego portu w Docker: "8081:8080"

**Problem: Błędy kompilacji Maven**

Solution:  
- Sprawdź wersję Javy: java -version (wymagana 17+)  
- Wyczyść cache: ./mvnw clean  
- Usuń folder .m2/repository i pobierz zależności ponownie  
- Sprawdź połączenie internetowe (pobieranie zależności)

**Problem: Błędy JavaScript w przeglądarce**

Solution:  
- Sprawdź konsolę developerską (F12)  
- Upewnij się, że zasoby statyczne są dostępne  
- Sprawdź czy Bootstrap CSS i JS są załadowane  
- Wyczyść cache przeglądarki

**Problem: Błędy autoryzacji Spring Security**

Solution:  
- Sprawdź czy użytkownik ma odpowiednią rolę  
- Sprawdź mapping URL w SecurityConfig  
- Sprawdź czy sesja jest aktywna  
- Włącz debug logowanie dla security

### 8.2 Logi i Debugowanie

**Lokalizacja logów:** - Konsola podczas uruchamiania - Pliki logów (jeśli skonfigurowano): logs/spring.log

**Zwiększenie poziomu logowania:**

# Debug dla całej aplikacji  
logging.level.com.example.ticketmaster=DEBUG  
  
# Debug dla Spring Security  
logging.level.org.springframework.security=DEBUG  
  
# Debug dla Hibernate SQL  
logging.level.org.hibernate.SQL=DEBUG  
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE  
  
# Debug dla Thymeleaf  
logging.level.org.thymeleaf=DEBUG

**Monitoring aplikacji Docker:**

# Logi kontenerów  
docker-compose logs -f  
  
# Status kontenerów  
docker-compose ps  
  
# Wejście do kontenera aplikacji  
docker exec -it ticketmaster-app bash  
  
# Wejście do kontenera MySQL  
docker exec -it ticketmaster-mysql mysql -u ticketmaster -p

## 9. Bezpieczeństwo

### 9.1 Zabezpieczenia Implementowane w Aplikacji

**1. Autoryzacja i Uwierzytelnianie:** - Spring Security z BCrypt do hashowania haseł (strength: 10) - Kontrola dostępu oparta na rolach (RBAC) - Zabezpieczenie endpointów według wzorca URL i ról - Automatyczne przekierowania po zalogowaniu

**2. Walidacja Danych:** - Bean Validation (JSR-303) z adnotacjami @Valid - Walidacja po stronie serwera i klienta - Zabezpieczenie przed SQL Injection (JPA/Hibernate) - Walidacja długości, formatu i wymagalności pól

**3. Zarządzanie Sesją:** - Automatyczne wygasanie sesji po okresie nieaktywności - Zabezpieczenie ciasteczek (HttpOnly, Secure) - Usuwanie sesji po wylogowaniu - CSRF protection (Cross-Site Request Forgery)

**4. Zabezpieczenia aplikacji:** - Ukrywanie szczegółów błędów przed użytkownikami końcowymi - Logowanie bezpieczeństwa i błędów - Walidacja uprawnień na poziomie serwisu - Zabezpieczenie przed nieautoryzowanym dostępem do zasobów

### 9.2 Konfiguracja Spring Security

**SecurityConfig - kluczowe elementy:** - Mapowanie URL na role: /admin/\*\* → ROLE\_ADMIN - Konfiguracja formularza logowania z custom success handler - Logout z czyszczeniem sesji i ciasteczek - UserDetailsService implementation w UserService - Password encoder z BCrypt

### 9.3 Rekomendacje dla Środowiska Produkcyjnego

**1. Zabezpieczenia systemowe:** - Zmień domyślne hasła wszystkich kont demonstracyjnych - Używaj HTTPS ze ważnymi certyfikatami SSL - Skonfiguruj firewall (tylko porty 80, 443, 22) - Włącz automatyczne aktualizacje bezpieczeństwa

**2. Zabezpieczenia bazy danych:** - Użyj silnych haseł dla użytkowników MySQL - Ogranicz uprawnienia użytkownika aplikacji - Włącz SSL dla połączeń z bazą danych - Regularne backupy bazy danych

**3. Zabezpieczenia aplikacji:** - Ustaw spring.profiles.active=prod - Wyłącz debug logowanie - Skonfiguruj monitoring i alerty - Włącz audyt operacji użytkowników

**4. Zmienne środowiskowe:**

export DB\_PASSWORD=strong\_production\_password  
export JWT\_SECRET=your\_jwt\_secret\_key  
export ADMIN\_PASSWORD=strong\_admin\_password

## 10. Podsumowanie

### 10.1 Osiągnięte Cele

Aplikacja Ticketmaster to kompletny system zarządzania wydarzeniami i biletami, oferujący:

**Funkcjonalności biznesowe:** - Pełny cykl zarządzania wydarzeniami (tworzenie → moderacja → sprzedaż) - System wielorolowy z jasnym podziałem uprawnień - Interaktywny system rezerwacji miejsc - Zarządzanie biletami z różnymi statusami

**Zalety techniczne:** - Nowoczesna architektura Spring Boot z najlepszymi praktykami - Responsywny interfejs użytkownika z Bootstrap - Bezpieczna autoryzacja i uwierzytelnianie - Łatwe wdrożenie dzięki konteneryzacji Docker - REST API dla funkcji w czasie rzeczywistym - Kompletna dokumentacja i kody demonstracyjne

**Skalowalność i rozszerzalność:** - Modułowa architektura umożliwiająca łatwe dodawanie funkcji - Separation of concerns (Controller-Service-Repository) - DTO pattern dla bezpiecznej wymiany danych - Mapper pattern dla konwersji między warstwami

### 10.2 Możliwości Rozwoju

**Krótkoterminowe usprawnienia:** - Dodanie płatności online (Stripe, PayPal) - System powiadomień email/SMS - Generowanie biletów PDF z kodami QR - Mobile API (REST/GraphQL) - Raporty i analityka sprzedaży

**Długoterminowe rozszerzenia:** - Microservices architecture - Event sourcing dla audytu - Real-time notifications (WebSocket) - Machine learning dla rekomendacji - Mobile aplikacje (React Native/Flutter)

### 10.3 Gotowość Produkcyjna

System jest gotowy do wdrożenia w środowisku produkcyjnym po: - Konfiguracji produkcyjnej bazy danych - Skonfigurowaniu HTTPS i domeny - Dostosowaniu do wymagań biznesowych - Przeprowadzeniu testów bezpieczeństwa i wydajności

## 11. Źródła

### 11.1 Dokumentacja Oficjalna

* [Spring Boot Documentation](https://spring.io/projects/spring-boot)
* [Spring Security Reference](https://spring.io/projects/spring-security)
* [Spring Data JPA Reference](https://spring.io/projects/spring-data-jpa)
* [Thymeleaf Documentation](https://www.thymeleaf.org/documentation.html)
* [Bootstrap 5 Documentation](https://getbootstrap.com/docs/5.3/getting-started/introduction/)
* [Docker Documentation](https://docs.docker.com/)
* [Maven Documentation](https://maven.apache.org/guides/)
* [MySQL 8.0 Documentation](https://dev.mysql.com/doc/refman/8.0/en/)

### 11.2 Tutoriale i Kursy

* [Baeldung - Spring Boot Tutorials](https://www.baeldung.com/spring-boot)
* [Spring Guides](https://spring.io/guides)
* [Spring Security Tutorials](https://www.baeldung.com/security-spring)
* [Docker Getting Started](https://docs.docker.com/get-started/)
* [Thymeleaf + Spring Tutorial](https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html)

### 11.3 Narzędzia i Zasoby

* **IDE:** IntelliJ IDEA, Eclipse, Visual Studio Code
* **Testowanie:** Postman, curl, browser developer tools
* **Monitoring:** Spring Boot Actuator, Micrometer
* **CI/CD:** Jenkins, GitHub Actions, GitLab CI

Pełny kod źródłowy aplikacji jest dostępny w strukturze projektu opisanej w tym dokumencie.