# Projekt Dokumentation: Event-Ticketing-Plattform API

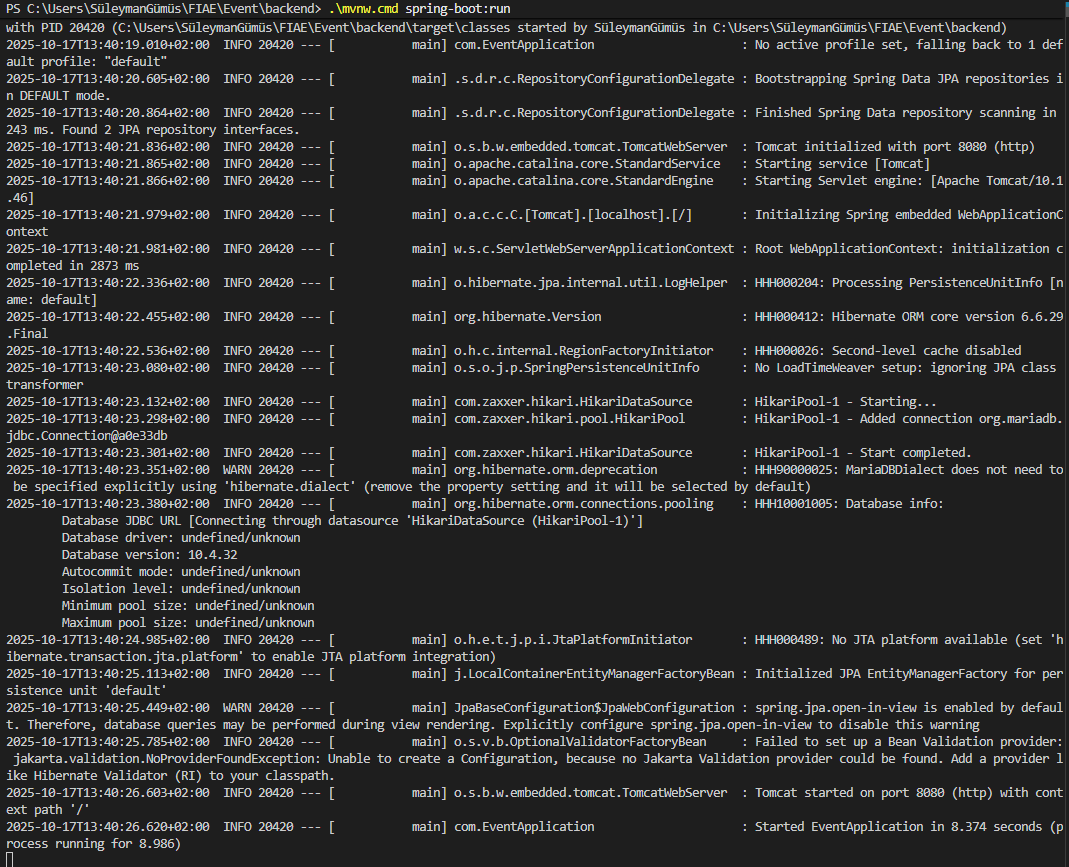
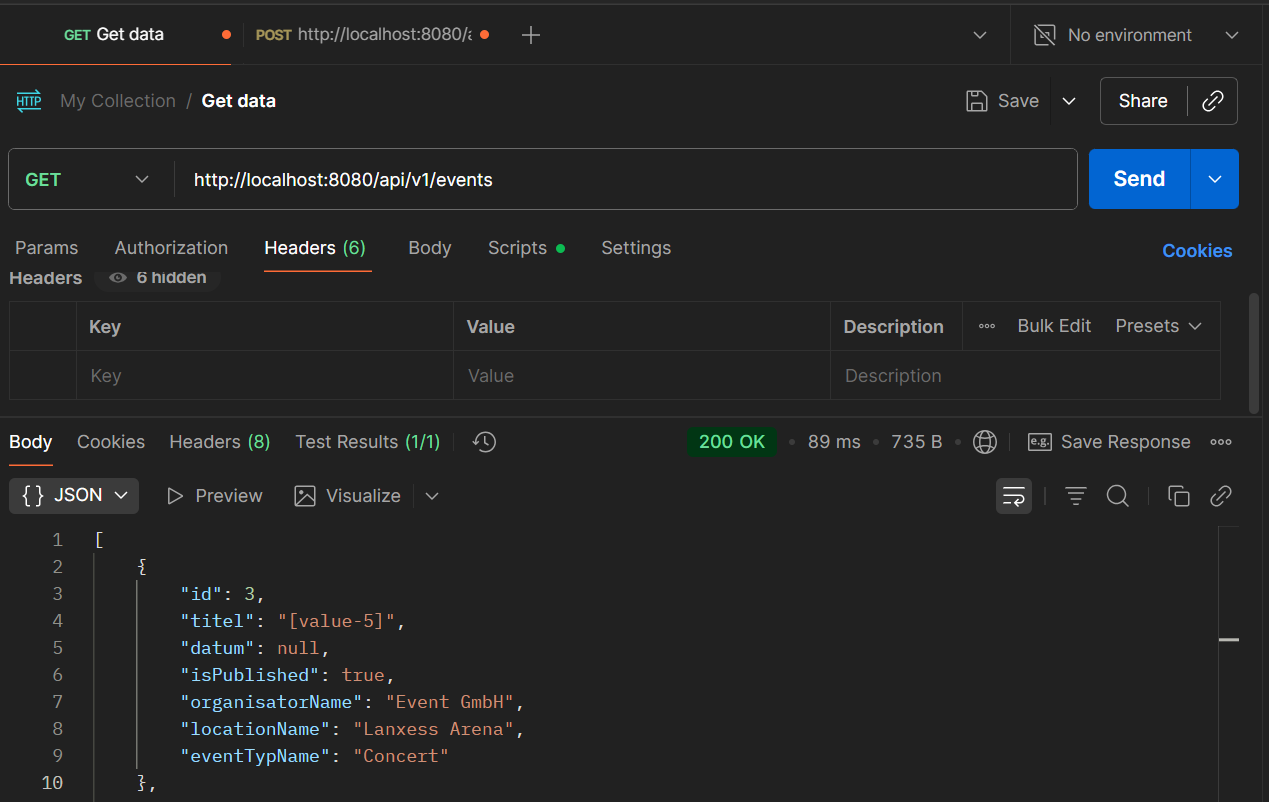
## 1. Einleitung und Kernziel

Dieses Dokument beschreibt die Architektur, Technologieauswahl und die wichtigsten technischen Entscheidungen der entwickelten Event-Ticketing-Plattform API. Ziel ist es, ein Verständnis für die Struktur, Funktion und Integrität des Projekts zu vermitteln.

|  |  |
| --- | --- |
| Produkt | Event-Ticketing-System zur Verwaltung von Theater, Konzerten und Seminaren. |
| Kernziel | Aufbau einer sicheren, skalierbaren REST API mit Fokus auf die Datenintegrität im Verkaufsprozess. |
| Status | Backend läuft stabil (Java/Spring Boot), Kommunikation mit Frontend hergestellt.  Server läuft Stabil Tomcat läuft auf <http://localhost:8080>.  Frontend Sichtbarkeit hergestellt, die Seite läd lokal. Der CORS-Filter erlaubtAnfragen von Port 5500 an das Backend.  Datenstruktur ist fehlerfrei, alle Entitäten (Models) sind korrekt definiert. |

Ein Bild, das Text, Screenshot, Multimedia-Software, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.



## 2. Technologie-Stack (Das Fundament des Projekts)

|  |  |  |
| --- | --- | --- |
| **Komponente** | **Technologie** | **Erklärung** |
| Die Basis Sprache | Java 21 (LTS) | Java ist die Basis-Sprache des Backends. Wir haben diese Sprache gewählt, weil sie der Industriestandard für **geschäftskritische, hochskalierbare Systeme** ist (z.B. Banken, große Plattformen). LTS = Long-Term Support. |
| Der Bauleiter (Anwendungs-Motor) | Spring Boot 3 | Seine Hauptaufgabe ist es, die Architektur **vollautomatisch zusammenzusetzen**, damit sich die Entwickler auf die Geschäftslogik konzentrieren können. Es verbindet die drei logischen Schichten: „Controller (Eingang), Service (Logik), Repository (Datenbank)“. |
| Das Lagerhaus (die Daten) | MariaDB 10.x (ACID) | Der Ort, an dem die Daten dauerhaft und sicher gespeichert werden (die Datenbank). **ACID** ist eine Qualitätsgarantie: Es stellt sicher, dass Ticketverkäufe immer **zuverlässig und konsistent** ablaufen (keine Überbuchung). |
| Der Übersetzer (ORM)  || | JPA/ Hibernate | Der "Übersetzer" zwischen der **Java-Sprache** (Objekten) und der **Datenbank-Sprache** (SQL-Tabellen). Wir können Code schreiben, ohne komplexe SQL-Befehle selbst tippen zu müssen.  Spezifikationen: „die formalen Regeln, wie die Java-Objekte mit der Datenbank zusammenarbeiten sollen. |
| Der Bauwerkzeug | Maven | Das automatische Werkzeug, das alle Bibliotheken verwaltet, den Code baut und die Anwendung startet. |
| Tooling | CORS-Filter | Kommunikation zwischen Frontend (:5500) und Backend (:8080) |
| Frontend Logik | HTML | Der Rahmen und Inhalt: Definiert die Struktur und den Inhalt der Webseite (Überschriften, Bilder, Formulare). Es ist das "Skelett" der Anwendung, das festlegt, wo die Event-Titel und die Ticket-Auswahlfelder platziert sind. |
| || | CSS | **Das Aussehen:** Definiert das visuelle Design, die Farben, Schriftarten und das Layout (z.B. wie die Event-Karten angeordnet sind oder wie die Ticket-Auswahl formatiert ist). |
| || | JavaScript (JS) | **Die Intelligenz und Verbindung: Das "Gehirn" der Webseite. JavaScript ist verantwortlich für die gesamte Dynamik der Seite.**  **API-Aufrufe: Sendet die Anfragen (GET, POST) an Ihr Java/Spring Boot Backend.**  **Datenverarbeitung: Liest die JSON-Antworten des Backends (z.B. das EventDetailsDto).**  **DOM-Manipulation: Aktualisiert die Webseite in Echtzeit (z.B. zeigt die berechnete verfügbar-Zahl an, ohne die Seite neu laden zu müssen).** |

## 3. Datenstruktur und Architektur

Das System basiert auf einem relationalen Datenmodell (RDM), das die folgenden Entitäten umfasst:

* TicketKategorie: Definiert die maximale kontingentMax (wie viele Tickets maximal existieren).
* Bestellung / Bestellposition – Speichert die Menge der tatsächlich verkauften Tickets.

Ein Bild, das Diagramm, Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

### 3.1 Dynamische Verfügbarkeitsberechnung

Die verfügbare Ticketanzahl wird dynamisch berechnet:  
Verfügbarkeit = TicketKategorie.kontingentMax − SUM(Bestellposition.menge)

Umsetzung: Die @Query mit SUM(bp.menge) führt diese Berechnung direkt in der Datenbank aus.

## 3.2 API-Vertrag: Endpunkte für das Frontend (GET/ events)

Das Frontend muss die folgenden Endpunkte aufrufen, um Events zu lesen und tickets zu kaufen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Zweck** | **Methode** | **URl-Pfad** | **Erwarteter Body-Typ** |
| Übersicht | GET | /api/v1/events | List<EventDetailsDto> |
| Details & Verfügbarkeit | GET | /api/v1/events/{id} | EventDetailsDto  (mit TicketKategotieDto Liste) |

## 3.3 Kritische OrderRequestDto (Daten, die gesendet werden müssen)

Das Frontend muss diese Objekte an das Backend senden, um eine Bestellung zu erstellen:

|  |  |  |
| --- | --- | --- |
| **Feld** | **Typ** | **Beschreibung** |
| eventId | Long | ID des Events |
| ticketKatId | Long | ID der gewählten Ticketkategorie |
| quantity | Integer | Anzahl der gewünschten Tickets (min=1) |
| customerEmail | String | E-Mail des Käufers (für die Benachrichtigung) |
| bezahlArtId | Long | ID der Bezahlart |

## 4. Der Nächste Meilenstein: Transaktionssicherheit

Der wichtigste offene Punkt ist die Implementierung des sicheren Ticketverkaufs über den POST /orders-Endpunkt.

## 4.1. Die Magie von @Transactional

Die Logik des Ticketverkaufs wird in der Methode processTicketOrder() (im OrderService.java) mit der Annotation @Transactional versehen.

|  |  |
| --- | --- |
| Funktionalität | Sicherheitsgarantie (Wichtigkeit) |
| Start des Kaufs | @Transactional erstellt einen unsichtbaren, unteilbaren Block. |
| Kapazitätsprüfung | Wenn die gekaufte Menge des Kontigent übersteigt… |
| Rollback Mechanismus | …wird ein Fehler ausgelöst und die gesamte Operation automatisch rückgängig gemacht (ROLLBACK). Kein unvollständiger Verkauf wird gespeichert! |