Cinema Management Software

Informationsinfrastrukturen Unit 2_Q1.2025

Jojann Alberts, Dennis Buchholz, Christopher Cirak, Raha Senemar

GitHub Repository: https://github.com/dlbuchholz/cinema-management-software

Inhaltsverzeichnis

Einleitung

Projektbeschreibung und besondere Anforderungen

Im Rahmen dieses Projekts wurde eine vollfunktionsfähige Webanwendung für ein Kino entwickelt. Die Plattform ermöglicht es Kunden, Filme auszuwählen sowie **Tickets für bestimmte Vorführungen zu reservieren, zu buchen oder zu stornieren**. Zusätzlich bietet das System Kino-Besitzern eine eigene Verwaltungsoberfläche, über die sie zentrale Aspekte des Kinos bearbeiten können, darunter **das Anlegen von Kinosälen**, die **Zuweisung von Sitzplätzen**, sowie das **Erstellen und Verwalten von Filmen und Vorführungen**.

Besondere technische Anforderungen

Ein zentrales Ziel dieses Projekts war es, ein **ereignisgesteuertes System mithilfe von RabbitMQ** zu implementieren. Dafür wurde eine lose Kopplung zwischen dem Frontend, dem Anwendungsteil (ApplicationService) und dem Persistenzteil (PersistenceService) geschaffen. Alle Befehle, wie das Erstellen eines Tickets oder das Anlegen eines Nutzers, werden über **RabbitMQ-Queues** verarbeitet. Um dies strukturiert umzusetzen, wurde zusätzlich das **Command Pattern** verwendet, welches eine klare Trennung von Befehlslogik und deren Ausführung ermöglicht.

Ein weiteres Alleinstellungsmerkmal war die Integration von zwei unterschiedlichen Datenbanktypen:

- Eine **relationale SQL-Datenbank** (MariaDB/MySQL) für transaktionale Daten (z. B. Kunden, Tickets, Kinosäle)
- Eine **Graphdatenbank** zur Speicherung und Analyse von Statistiken und Beziehungsdaten (z. B. Zusammenhänge zwischen Kundenverhalten und Filmen)

Technologiestack

Backend:

- Java mit **Spring Boot** zur schnellen Entwicklung von REST-Schnittstellen
- **Jackson** zur Verarbeitung von JSON-Daten
- **JWT** (jsonwebtoken) zur sicheren Verwaltung von Benutzertokens
- **RabbitMQ** als Messaging-Broker
- **JPA/Hibernate** zur Datenpersistierung

Frontend:

- **React** mit **TypeScript** für eine moderne, komponentenbasierte Benutze- Oberfläche
- Kommunikation mit dem Backend über **REST-APIs**

Anwendungsfälle

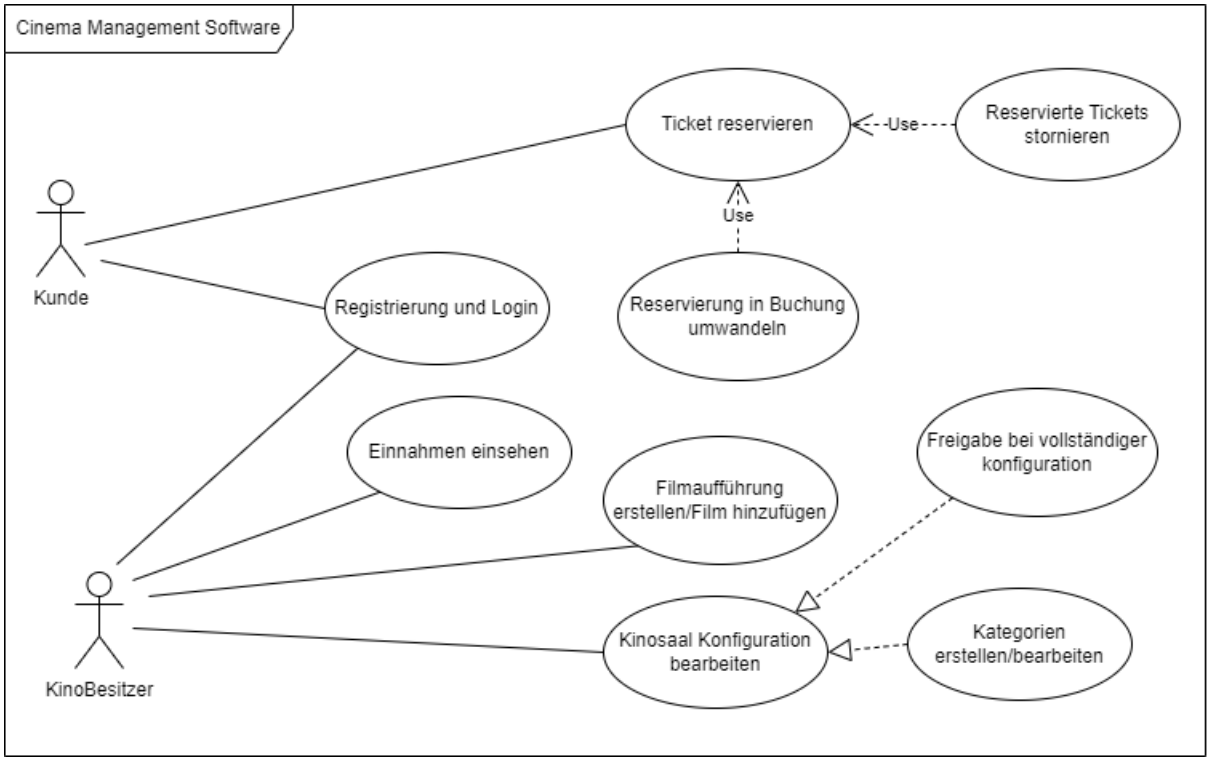

Abb. Anwendungsfalldiagramm

Das System zur Verwaltung von Kinobuchungen und -reservierungen stellt eine Vielzahl an Funktionen bereit, die den gesamten Ablauf eines Kinobesuchs digital abbilden. Im Folgenden werden die zentralen Anwendungsfälle des Systems beschrieben.

Ein Kunde kann sich zunächst im System registrieren, indem er seine persönlichen Daten wie Name, E-Mail-Adresse, Passwort und Telefonnummer angibt. Anschließend hat der Kunde die Möglichkeit, sich mit seinen Zugangsdaten im System anzumelden und die verschiedenen Funktionen zu nutzen.

Nach erfolgreicher Anmeldung kann der Kunde verfügbare Filme und Vorführungen einsehen. Hierbei werden Details wie Filmtitel, Beschreibung, Genre sowie Vorführungszeiten und verfügbare Sitzplätze angezeigt. Der Kunde kann eine bestimmte Vorführung auswählen und einen oder mehrere Sitzplätze reservieren. Dabei werden ihm die unterschiedlichen Sitzplatzkategorien „Parkett“, „Loge“ und „Loge Service“ angeboten, die sich im Komfort und Preis unterscheiden.

Reservierungen können vom Kunden jederzeit storniert werden, solange sie nicht in eine Buchung umgewandelt wurden.

Alternativ kann der Kunde seine Reservierung auch direkt in eine Buchung umwandeln. Darüber hinaus besteht die Möglichkeit, Tickets ohne vorherige Reservierung direkt zu buchen, sofern für die gewünschte Vorführung noch Plätze verfügbar sind.

Für den Kinobetreiber bietet das System ebenfalls spezifische Anwendungsfälle. So kann der Betreiber neue Kinosäle anlegen und konfigurieren, indem er die Anzahl und Anordnung der

Sitzreihen sowie die Sitzplatzkategorien festlegt. Nach Abschluss der Konfiguration kann der Saal für den Ticketverkauf freigegeben werden.

Zudem kann der Betreiber neue Filme in das System einpflegen und für diese Filme Vorführungen in den verschiedenen Kinosälen planen. Während des Betriebs kann der Kinobetreiber jederzeit eine Übersicht über die aktuellen Buchungen und Reservierungen abrufen und sich die daraus resultierenden Einnahmen anzeigen lassen.

Durch die Kombination dieser Anwendungsfälle wird ein vollständiger digitaler Ablauf für den Ticketverkauf im Kino ermöglicht – von der Konfiguration der Sitzplätze bis zur Buchung und Stornierung durch den Kunden.

UML-Fachklassendiagramm

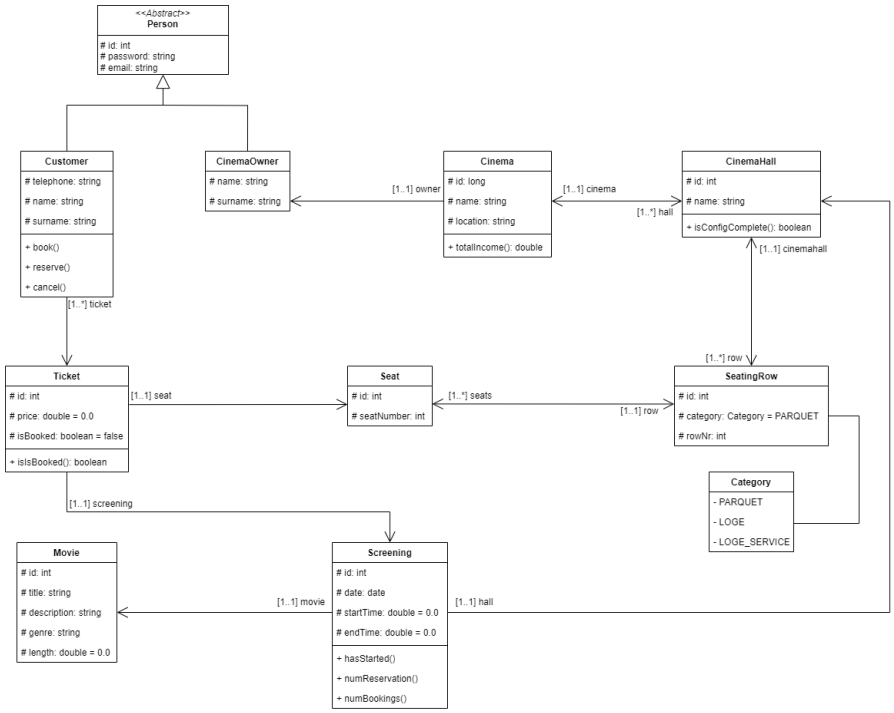

Abb. UML-Fachklassendiagramm

Wenn der Kinobesitzer einen Film schauen möchte, muss dieser sich ein Kunden Konto erstellen. Der Kunde kann Tickets buchen, in denen der Sitzplatz, die Sitzreihe, die Kategorie, die Vorführung und der Kinosaal zu finden sind. Ein Film kann in mehreren Vorführungen, an verschiedenen Tagen und in Kinosälen gespielt werden.

Technische Dokumentation – Frontend

Das Frontend der Cinema Management Software wurde mit dem Framework React entwickelt. Ziel war es, eine benutzerfreundliche und übersichtliche Oberfläche bereitzustellen, die die Interaktion mit dem System intuitiv gestaltet. Das Frontend bildet die Schnittstelle zwischen den Endbenutzern (Kunden und Kinobetreiber) und den Backend-Services.

Die React-Anwendung ist als Single-Page-Application (SPA) konzipiert und kommuniziert über REST API-Endpunkte mit dem ApplicationService im Backend. Für die Gestaltung der Benutzeroberfläche kommen moderne UI-Bibliotheken zum Einsatz, wodurch eine ansprechende und konsistente Optik gewährleistet wird.

Die Anwendung ist in verschiedene Komponenten unterteilt, die die Hauptfunktionen abbilden:

- Registrierung und Login: Hier können sich neue Kunden registrieren und bestehende Nutzer anmelden. Die Authentifizierung erfolgt durch den AuthenticationService, der dem Frontend bei erfolgreichem Login ein JWT-Token bereitstellt.
- Filmliste und Vorführungen: Im öffentlichen Bereich der Anwendung können alle registrierten Nutzer verfügbare Filme und deren Vorführungen einsehen. Die Daten werden über die API vom ApplicationService geladen und im Frontend in übersichtlicher Form dargestellt.
- Sitzplatzreservierung und Buchung: Kunden können im Frontend die gewünschten Sitzplätze für eine bestimmte Vorführung auswählen und diese entweder reservieren oder direkt buchen. Hierbei wird die Sitzplatzkonfiguration des jeweiligen Kinosaals berücksichtigt, sodass der Nutzer eine visuelle Darstellung der Sitzreihen und Kategorien erhält.
- Reservierungs- und Buchungsübersicht: Im Kundenbereich können Nutzer ihre bestehenden Reservierungen und Buchungen einsehen und ggf. stornieren oder umbuchen.
- Administration: Für Kinobetreiber existiert ein gesonderter Bereich, in dem Kinos, Kinosäle, Sitzplatzkonfigurationen sowie Filme und Vorführungen verwaltet werden können.

Die Kommunikation zwischen Frontend und Backend erfolgt ausschließlich über HTTP-Requests im JSON-Format. Eingehende Daten werden im Frontend validiert, um fehlerhafte Anfragen zu vermeiden und eine bessere User Experience zu gewährleisten.

Die Architektur der Anwendung orientiert sich am Model-View-Controller (MVC)-Prinzip, wobei React die View-Schicht bildet. Zustandsverwaltung erfolgt mithilfe von React Context API und lokalen States. Dadurch wird eine klare Trennung von Logik und Darstellung erreicht, was die Wartbarkeit und Erweiterbarkeit der Anwendung vereinfacht.

Technische Architektur des Backends

Das Backend der Cinema Management Software ist in Java unter Verwendung von Maven entwickelt und als servicebasiertes System konzipiert, bei dem verschiedene Dienste über eine Messaging-Infrastruktur (RabbitMQ) miteinander kommunizieren. Die Architektur ermöglicht eine lose Kopplung zwischen Komponenten und eine einfache Erweiterbarkeit.

Alle Backend-Komponenten basieren auf dem Spring Boot Framework, das eine schnelle und produktionsreife Entwicklung von Java-basierten Services ermöglicht. Spring Boot stellt dabei Konventionen, Autokonfigurationen und einen eingebetteten Webserver (Apache Tomcat) bereit, wodurch die Konfiguration stark vereinfacht wird.

Komponentendiagramm

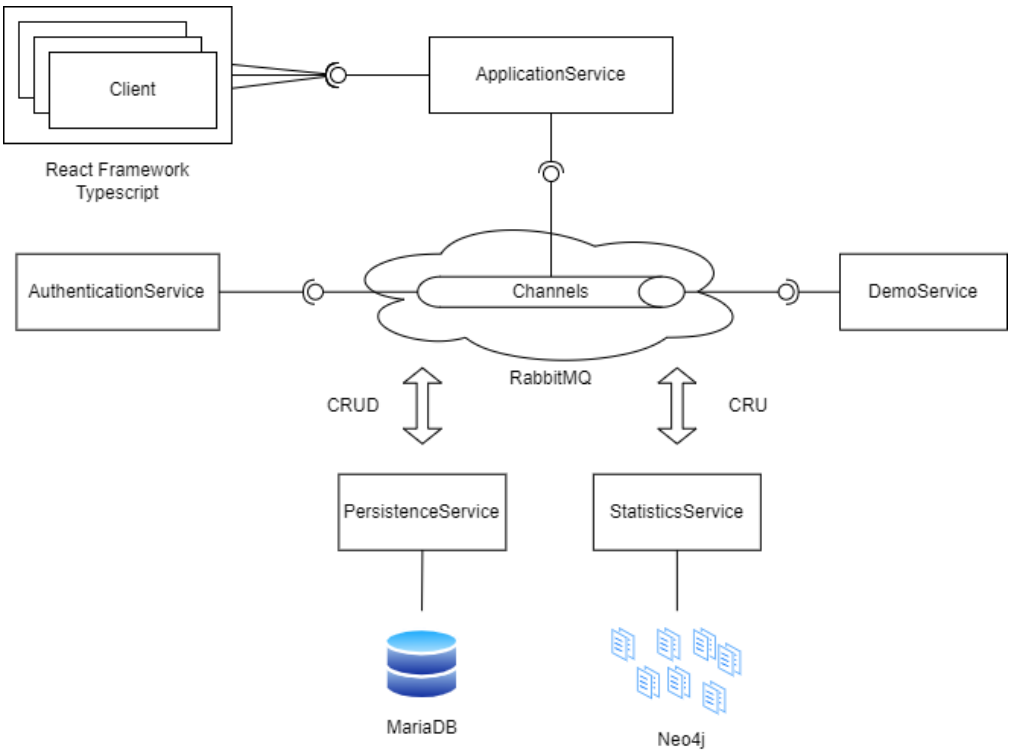

Abb. Komponentendiagramm

- Der AuthenticationService ist ein dedizierter Service, der für die Verwaltung von Benutzerauthentifizierung und -autorisierung verantwortlich ist. Dieser Service bietet folgende Funktionalitäten:
    - Benutzerregistrierung: Erstellen eines neuen Benutzerkontos mit zugehörigen Daten.
    - Benutzerauthentifizierung: Validierung von Anmeldeinformationen und Generierung von Token (z.B. JWT), die für geschützte Ressourcen erforderlich sind.
    - Tokenvalidierung: Überprüfung der Gültigkeit von Tokens bei eingehenden Anfragen an geschützte Endpunkte.
- Der ApplicationService ist der zentrale Dienst, der die Geschäftslogik der Anwendung umsetzt. Die Hauptaufgaben umfassen:
    - Verwaltung die Filmen, Vorstellungen und Buchungen
    - Bereitstellung von REST API-Endpunkten für das Frontend, um verschiedene Änderungen an den Stammdaten durchzuführen; Diese Endpunkte werden durch Controller-Klassen wie MovieController bereitgestellt, die HTTP-Anfragen entgegennehmen, verarbeiten und an andere Dienste weiterleiten.
    - Die Controller interagieren über RabbitMQ mit anderen Diensten, indem sie Nachrichten an spezialisierte Queues senden, wie z.B movie.create (Erstellen eines Films), movie.fetch (Abrufen eines Films), movie.update (Aktualisieren eines Films),movie.delete (Löschen eines Films), movie.search
- Der DemoService ist ein unterstützender Dienst, dessen Hauptaufgabe darin besteht, die Datenbank für Präsentationszwecke mit Testdaten zu füllen.

    - Dateninitialisierung: Automatisierte Erstellung von Filmen, Vorstellungen, Buchungen und Benutzern.
    - Datenbereinigung: Möglichkeit, Testdaten zu entfernen, um eine leere Datenbank für neue Demonstrationen vorzubereiten
    - Fair Scheduling: Sicherstellen, dass alle Filme aus den Testdaten gleichmäßig gezeigt werden
- Der PersistenceService ist das eigentliche Bindeglied zwischen der Anwendung und der Datenbank. Dieser Dienst verwendet JPA (Java Persistence API) zur Interaktion mit einer relationalen MySQL-Datenbank. Die Speicherung und Abfrage von Daten erfolgt über spezialisierte Repository-Klassen.
    - JPA-Entitäten: Daten werden durch Klassen wie MovieEntity repräsentiert, die mit Datenbanktabellen verknüpft sind.
    - CRUD-Operationen: Das Speichern, Abrufen, Aktualisieren und Löschen von Daten wird durch JPA-Methoden realisiert.
    - Benutzerdefinierte Abfragen: Zusätzlich zur Standardfunktionalität werden spezielle Abfragen ermöglicht, um z.B. nach Filmen anhand von Titeln, Genres oder Beschreibungen zu suchen.
- Der StatisticsService ist ein separater Dienst, der sich auf die Erhebung und Bereitstellung statistischer Daten spezialisiert. Die Datenverarbeitung erfolgt über Neo4J als Graphdatenbank
    - Nutzung von Events: Events wie event.movie.created oder event.booking.created werden empfangen, sobald der PersistenceService darüber informiert, dass eine relevante Operation durchgeführt wurde und daraufhin ebenso in der Graphdatenbank angelegt
    - Statistik-Anfragen werden über Queues wie statistics.revenue.screening oder statistics.top-movies empfangen.
    - Die asynchrone Verarbeitung sorgt dafür, dass Statistiken stets auf aktuellem Stand sind, ohne den Betrieb anderer Services zu behindern.
    - Vorteile von Neo4j:
        - Effiziente Speicherung und Abfrage von Beziehungsdaten zwischen Entitäten (z.B. Filme, Buchungen, Kinosäle).
        - Aggregationen und Metriken lassen sich performant berechnen.

Datenfluss & Interaktion

RabbitMQ wird als Message-Broker verwendet, um Nachrichten zwischen den Diensten zu transportieren.

- Queues: Nachrichten werden in eindeutig benannte Nachrichtenwarteschlangen für einzelne CRUD-Operationen (z.B. movie.create, movie.fetch, movie.update usw.) zwischengespeichert
- Bindings: Verknüpfen Queues mit Exchanges (wie cinemaExchange), um Nachrichten mit bestimmten Routing-Keys zuzuordnen.
- Message Routing: Jede Anfrage des Clients wird als Nachricht an RabbitMQ gesendet, der diese dann an den entsprechenden Konsumenten weiterleitet (z.B. PersistenceService).

- Serialisierung: Der Inhalt einer jeden Nachricht wird mit JSON serialisiert
- Nachrichtenverarbeitung: Der PersistenceService empfängt Nachrichten über spezialisierte Konsumenten wie MovieCommandConsumer, die eingehenden Nachrichten verarbeiten und über RabbitMQ wieder zurücksenden.

Sequenzdiagramm

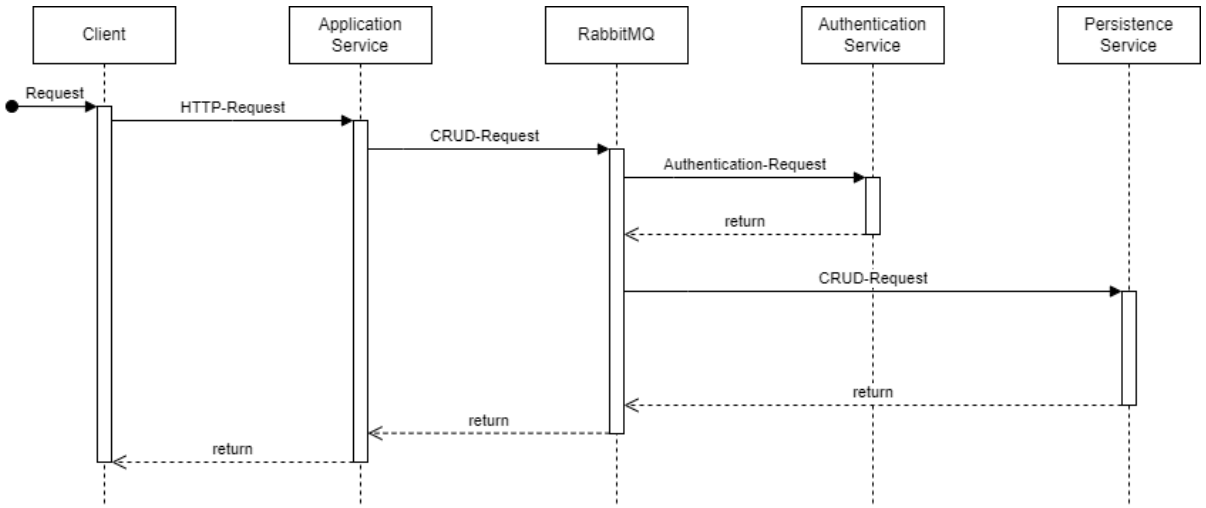

Abb. Sequenzdiagramm

1. Anfrage vom Client: Der Benutzer interagiert mit der React-Anwendung, die HTTP-Requests an den ApplicationService sendet
2. Verarbeitung im ApplicationService: Anfragen werden hier validiert. Sofern Sie erfolgreich validiert werden können, werden CRUD-bezogene Anfragen per Messager Broker an den Persistence Service weitergeleitet
    a. Sofern eine Authentifizierung erforderlich ist, wird eine Nachricht mit der Forderung einer Validierung von Token, Login-Daten oder Registierungsdaten an den AuthenticationService geschickt, um diese dort durchzuführen.
3. Der PersistenceService empfängt die Nachricht durch einen Listener (z.B. @RabbitListener Annotation) und führt die gewünschte Datenbankabfrage durch, dessen Ergebnis an den ApplicationService zurückgeschickt wird
4. Der ApplicationService sendet das Ergebnis als HTTP-Response an den Client zurück

Persistenz & Isolation

Hibernate wird als Standardimplementierung von JPA (Java Persistence API) in Spring Boot verwendet, was die Interaktion zwischen Java-Objekten und relationalen Datenbanken wie MySQL für den PersistenceService stark vereinfacht.

In der Spring Boot-Umgebung ist Hibernate nahtlos integriert. Entwickler definieren ihre Datenmodelle als einfache Java-Klassen, sogenannte Entitäten. Diese Klassen werden mit Annotationen wie @Entity, @Table, @Id und @Column versehen, um die Struktur der

Datenbanktabellen abzubilden. Ein Beispiel für eine Entität ist MovieEntity, die eine Tabelle movies in der MySQL-Datenbank repräsentiert. Sobald eine Entität definiert ist, wird sie automatisch von Hibernate verwaltet.

Hibernate übernimmt dabei mehrere Aufgaben: Es übersetzt Methodenaufrufe auf Java-Objekten automatisch in SQL-Abfragen, die mit der Datenbank kommunizieren. Wenn ein Entwickler beispielsweise eine neue MovieEntity speichert, wandelt Hibernate diese Operation in einen INSERT-Befehl um und führt diesen gegen die Datenbank aus.

Eine zentrale Komponente von Hibernate ist der SessionFactory, der als Fabrik für Session-Objekte dient. Diese Sessions repräsentieren eine Verbindung zur Datenbank und sind dafür verantwortlich, Transaktionen zu starten, Datenbankoperationen auszuführen und Transaktionen abzuschließen. In Spring Boot wird dieser Prozess jedoch durch Spring Data JPA abstrahiert, sodass Entwickler direkt mit Repository-Interfaces arbeiten können, die grundlegende CRUD-Operationen wie save(), findById(), delete() und benutzerdefinierte Abfragen bereitstellen. Hibernate bietet auch Funktionen wie Lazy Loading, Caching und Transaktionsmanagement, um die Leistung und Konsistenz der Anwendung zu optimieren.

Ein wichtiger Aspekt im Zusammenhang mit Transaktionen ist die sogenannte Isolationsebene. Diese definiert, wie unabhängig einzelne Transaktionen voneinander ablaufen, insbesondere im Hinblick auf konkurrierende Lese- und Schreibzugriffe. Hibernate unterstützt alle von SQL standardisierten Isolationsebenen:

- READ_UNCOMMITTED: Transaktionen können nicht bestätigte Änderungen anderer Transaktionen sehen (Dirty Reads).
- READ_COMMITTED: Eine Transaktion sieht nur bestätigte Änderungen anderer Transaktionen.
- REPEATABLE_READ: Daten, die innerhalb einer Transaktion gelesen wurden, ändern sich nicht während dieser Transaktion (verhindert Non-Repeatable Reads).
- SERIALIZABLE: Höchste Isolation, Transaktionen werden vollständig voneinander abgeschirmt, jedoch mit potenziellen Performance-Einbußen.

In dieser Anwendung verwenden wir READ_COMMITTED als Isolationsebene, was in den meisten Fällen ein gutes Gleichgewicht zwischen Konsistenz und Performance bietet. Diese Isolation verhindert Dirty Reads und ist die Standardeinstellung vieler relationaler Datenbanken wie MySQL.

Die Konfiguration kann über Annotationen wie @Transactional(isolation = Isolation.READ_COMMITTED) erfolgen oder in der application.properties Datei global festgelegt werden.

REST API

Die API zur Kommunikation zwischen Client (React) und Backend (im ApplicationService) verwendet ein RESTful Design, um verschiedene Ressourcen wie Benutzer, Kinos, Kinosäle, Sitzreihen, Sitzplätze, Filme und Vorführungen zu verwalten. Um auf die verschiedenen Funktionen der API zuzugreifen, werden HTTP-Methoden wie GET, POST, PUT und DELETE verwendet. Die Kommunikation zwischen Client und Server erfolgt über JSON-Daten, die sowohl für Anfragen als auch für Antworten genutzt werden.

Beispielsweise wird zur Authentifizierung der Benutzer der Endpunkt */api/users/login* verwendet. Dabei wird ein POST-Request mit einem JSON-Objekt gesendet, das die Zugangsdaten des Benutzers enthält. Ein typischer Request sieht folgendermaßen aus:

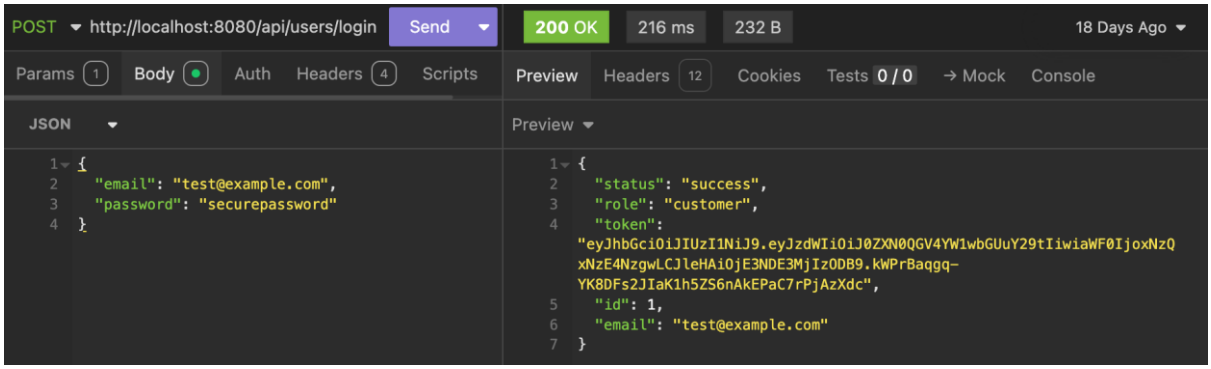

Abb. API Abfrage

Der Server verarbeitet diese Anfrage, prüft die Zugangsdaten und antwortet im Erfolgsfall mit einem JSON-Objekt, das Informationen über den Benutzer und einen sogenannten JSON Web Token (JWT) enthält. Dieser Token dient als Authentifizierungsnachweis für alle weiteren Anfragen des Benutzers.

Das Kino-Management-Modul bietet Funktionen zur Erstellung, Aktualisierung, Abruf und Löschung von Kinos. Kinos können bestimmten Besitzern zugeordnet werden, und detaillierte Informationen zu den Kinos sowie ihren verbundenen Kinohallen können abgerufen werden. Auf diese Weise wird ein zentralisiertes System zur Verwaltung aller Standorte eines Kino-Betreibers ermöglicht.

Für Kinobesitzer bietet die API-Funktionen zur Verwaltung ihrer Daten. Dies umfasst das Erstellen, Aktualisieren, Abrufen und Löschen von Kinobesitzern. Dabei werden persönliche Daten der Besitzer verwaltet und ihre Kinos mit ihnen verknüpft. Das System ermöglicht es, jedem Besitzer mehrere Kinos zuzuordnen, was besonders bei Betreibern mit mehreren Standorten von Vorteil ist.

Die Verwaltung der Kinosäle ist ein zentrales Element des Systems. Über die API können neue Kinosäle erstellt, bearbeitet, abgerufen und gelöscht werden. Diese Kinosäle sind immer mit einem bestimmten Kino verknüpft.

Innerhalb eines Kinos können Sitzreihen und Sitzplätze angelegt werden, um eine exakte Platzierung der Sitze im Saal zu ermöglichen. Dies ist wichtig, um eine genaue Buchungs- und Reservierungsverwaltung zu ermöglichen.

Ein weiterer wichtiger Bestandteil ist die Verwaltung von Filmen. Filme können erstellt, abgerufen, aktualisiert und gelöscht werden. Zu jedem Film werden Daten wie Titel, Genre, Beschreibung und Länge gespeichert. Vorführungen können anschließend für diese Filme in den vorhandenen Kinosälen geplant werden. Das System ermöglicht die Planung von Vorführungen mit Angaben zum Datum, Startzeit und Endzeit.

Die API unterstützt auch die Verwaltung von Sitzplatzkategorien wie „Parkett“, „Loge“ oder „VIP“. Diese Kategorien werden definiert und verwaltet, um unterschiedliche Preisklassen oder Komfortlevel im Buchungssystem abzubilden. Die Kategorien werden für die Platzierungssysteme der Kinosäle genutzt, um den Buchungsprozess zu optimieren.

Eine detaillierte Sitzplatzverwaltung wird durch die API ermöglicht. Sitzreihen und einzelne Sitze können in Kinosälen angelegt, abgerufen, aktualisiert und gelöscht werden. Die genaue Platzierung der Sitze ist notwendig, um ein funktionierendes Buchungssystem zu realisieren. Über spezielle Endpunkte wird auch die Buchung von Sitzplätzen für bestimmte Vorführungen ermöglicht, sowie das Abrufen und Verwalten dieser Buchungen.

Das Benutzerverwaltungssystem erlaubt es, Benutzerkonten zu erstellen, zu aktualisieren, abzurufen und zu löschen. Neben allgemeinen Benutzern gibt es spezielle Rollen wie Kinobesitzer oder Administratoren, die über erweiterte Rechte verfügen. Über die API können neue Nutzer registriert und deren Daten verwaltet werden.

Ein besonders wichtiger Bestandteil des Systems ist das Modul für Statistiken und Analysen. Die API ermöglicht es, Umsätze und Auslastungen auf verschiedenen Ebenen abzufragen. Dies umfasst einzelne Vorführungen, Filme, Kinosäle und sogar spezifische Datumsbereiche. Weiterhin können Top-Filme, Top-Vorführungen und Buchungstrends von Kunden abgerufen werden, um wertvolle Daten zur Analyse der Geschäftsleistung bereitzustellen.

Zusatzaufgabe: Andere NoSQL Datenbank —> Neo4j

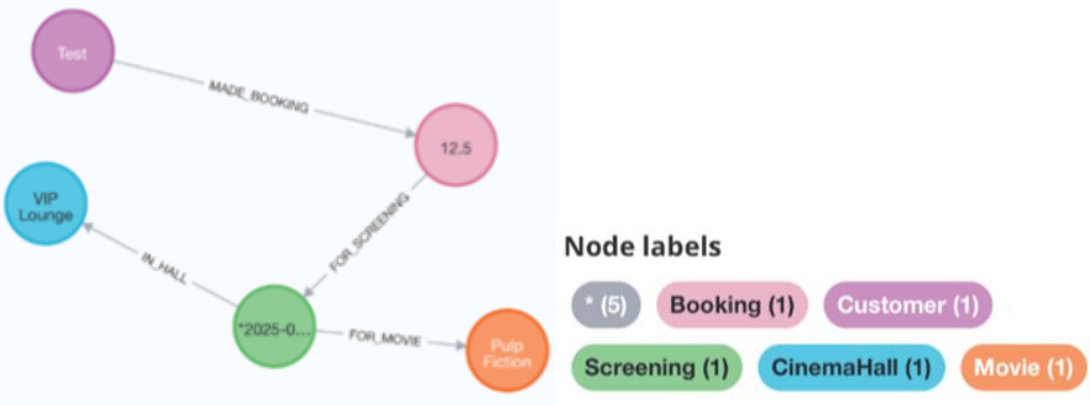

Abb. Exemplarebene (Links) und Modelebene (Rechts) Neo4j

Wir haben uns für die NoSQL Datenbank Neo4j im StatisticsService entschieden. Das ist eine Graphen Datenbank und visualisiert die Beziehungen mit Kanten und Knoten. In Neo4j werden die Kanten als Linien und die Knoten als Kreise mit Beschriftung abgebildet. Mithilfe der Abfragensprache Cypher können bestimmte Knoten gesucht werden.

In dem Beispielbild wird die Modellebene rechts angegeben und die Exemplarebene links. Hierbei wird eine bereits gebuchte Vorführung angezeigt. Der Benutzer „Test" hat eine Buchung gemacht mit der Information, dass die Vorführung im Kinosaal „VIP Lounge" stattfindet und der Film „Pulp Fiction" angeschaut wird. Insgesamt existieren fünf Pfeile bzw. Kanten zwischen den Knoten.