Dokumentation Kinoreservierungssystem Gruppe 5

Motivation und Anforderungen

Im Rahmen des Moduls „Fallstudie“ war es unsere Aufgabe ein Kinoreservierungssystem zu entwickeln, welches diverse Anforderungen erfüllt, die in Form von User Stories im vorherigen Semester definiert wurden.

Die User Stories stammen von den einzelnen Personas, die das Kinoreservierungssystem nutzen wollen. Aus den wichtigsten User Stories, die für das Funktionieren des Systems notwendig sind haben sich folgende grundlegende Anforderungen ergeben:

Als Administrator muss man neue Filme anlegen können, die auch Vorstellungen beinhalten, welche von Kunden besucht werden können.

Mitarbeiter und Kunden müssen Filme suchen können, um die Filme schneller zu finden. Zudem sollen Plätze für bestimmte Vorstellungen reserviert und auch wieder storniert werden können.

Nach diesen grundlegenden Anforderungen gibt es noch weitere User Stories, die bei Erfüllung der hieraus resultierenden Anforderungen zu einem kompletten Kinoreservierungssystem führen, dass den Anforderungen aller Personas genügt.

Kunden sollen sehen können, welche Filme als nächstes laufen und alle für sie wichtigen Informationen zum Film angezeigt bekommen. Zudem sollen z.B. Studenten einen ermäßigten Tarif erhalten können und bei erfolgreicher Buchung auch eine Bestätigungsmail bekommen. Um die Daten der Kunden zu speichern, soll es auch eine Funktion geben, sich ein Konto zu erstellen.

Basierend auf diesen und weiteren Anforderungen entstehen verschiedene Funktionen, die im Kinoreservierungssystem implementiert werden müssen. Dabei muss auch darauf geachtet werden, dass das System einwandfrei funktioniert und z.B. eine Doppelbuchung verhindert wird. Falls alle User Stories in Anforderungen übersetzt wurden und diese dann auch als Funktionen entwickelt und in das System eingebaut wurden, so ergibt sich am Ende ein komplett funktionstüchtiges Kinoreservierungssystem.

Datenmodell

Anmerkung: Die Attribute FSK des Films, Role des Users und SeatState des Sitzes sind als Enumeration umgesetzt.

Im Zentrum des Datenmodells steht das Event. Ein Event bezeichnet dabei die einmalige Ausführung eines Films. Ein Event hat daher einen Ausführungszeitpunkt.

Ein Event spielt ein Film (Movie) ab. In diesem spielen mehrere Schauspieler (Actor) mit. In der Entität ActsIn wird gespeichert welcher Schauspieler in welchem Film spielt und wie sein Charakter heißt. Außerdem hat ein Film einen oder mehrere Produzenten, Direktoren und Genres. Für das Filmposter gibt es zwei Speichervarianten. Entweder liegt das Bild auf einer externen DB, in diesem Fall wird ein Link gespeichert, oder das Bild wird als Byte-Array in der Entität ImageData gespeichert.

Ein Event wird in einem Kinosaal (CinemaHall) abgespielt. Eine Saal gehört dabei zu einem Kino (Cinema). Zu einem Saal wird ein Sitzplan(Seatingplan), bestehend aus Sitzblaupausen (Seatblueprints) gespeichert. Eine Blaupause ist dabei eine Vorlage für einen späteren Sitz. Eine Blaupause hat einen Sitz-Typ (SeatType), über welchen sich später der Preis bestimmt.

Beim Erstellen eines Events werden aus den Blaupausen tatsächliche Sitze (Seat) erstellt. Sitz wird dabei immer einem Ticket zugeordnet. In der Entität Sitz wird gespeichert ob ein Sitz frei, reserviert oder gebucht ist. Für den Fall einer Reservierung wird ein Ablaufzeitpunkt für die Reservierung gespeichert. Wird ein Ticket gebucht, so wird eine Buchung (Booking) erstellt, welche dem Ticket zugeordnet wird. Einer Buchung ist immer der jeweilige User zugeordnet. 1

TECHNOLOGIE-STACK

Java Springboot with Maven

In unserem Backend benutzen wir Java mit dem Framework Spring Boot, sowie Maven für einfaches Dependency Management.

- Entität
    - Für jede Entität unseres Datenmodells existiert eine jeweilige Klasse im Projekt (entities package), welche die Grundlage für unsere Persistenzeinheit (JPA) darstellt. Sie enthält alle Attribute, die in der Datenbank persistiert werden.
- Repository
    - Für jede Entität existiert ein Repository, welches die Schnittstelle zu unserer Datenabnk ist. Hierfür erweitern wir das Interface JpaRepository, durch welches wir Queries durch korrekte Semantik im Methodennamen erzeugen können.
- Service
    - Für jede Entität existiert ein Service, welches die Logik der betreffenden Entität beinhaltet. Lediglich der Service einer Entität hat Zugriff auf das dazugehörige Repository, andere Entitäten müssen den Service der fremden Entität als Schnittstelle benutzen.
- Controller
    - Sollte eine Entität an das Frontend exposed werden, so geschieht dies über einen Controller. Hier werden die Endpunkte definiert, über welche das Frontend Daten anfragen kann. Die Antworten an das Frontend werden in einer ResponseEntity gewrapped, welche zusätzliche Informationen über den Erfolg des Requests an das Frontend weitergibt.

Wir setzen ebenfalls auf Spring Security um unsere Endpunkte zu schützen, beispielsweise unsere Admin-Page vor regulären Usern und Gästen. Ebenso schützen wir unser Backend durch nicht autorisierte Requests mithilfe einer Cors-Policy.

MySQL

Für unsere Datenbank setzen wir auf eine klassische MySQL-Datenbank, welche mittels JpaRepository mit unserem Backend verbunden ist.

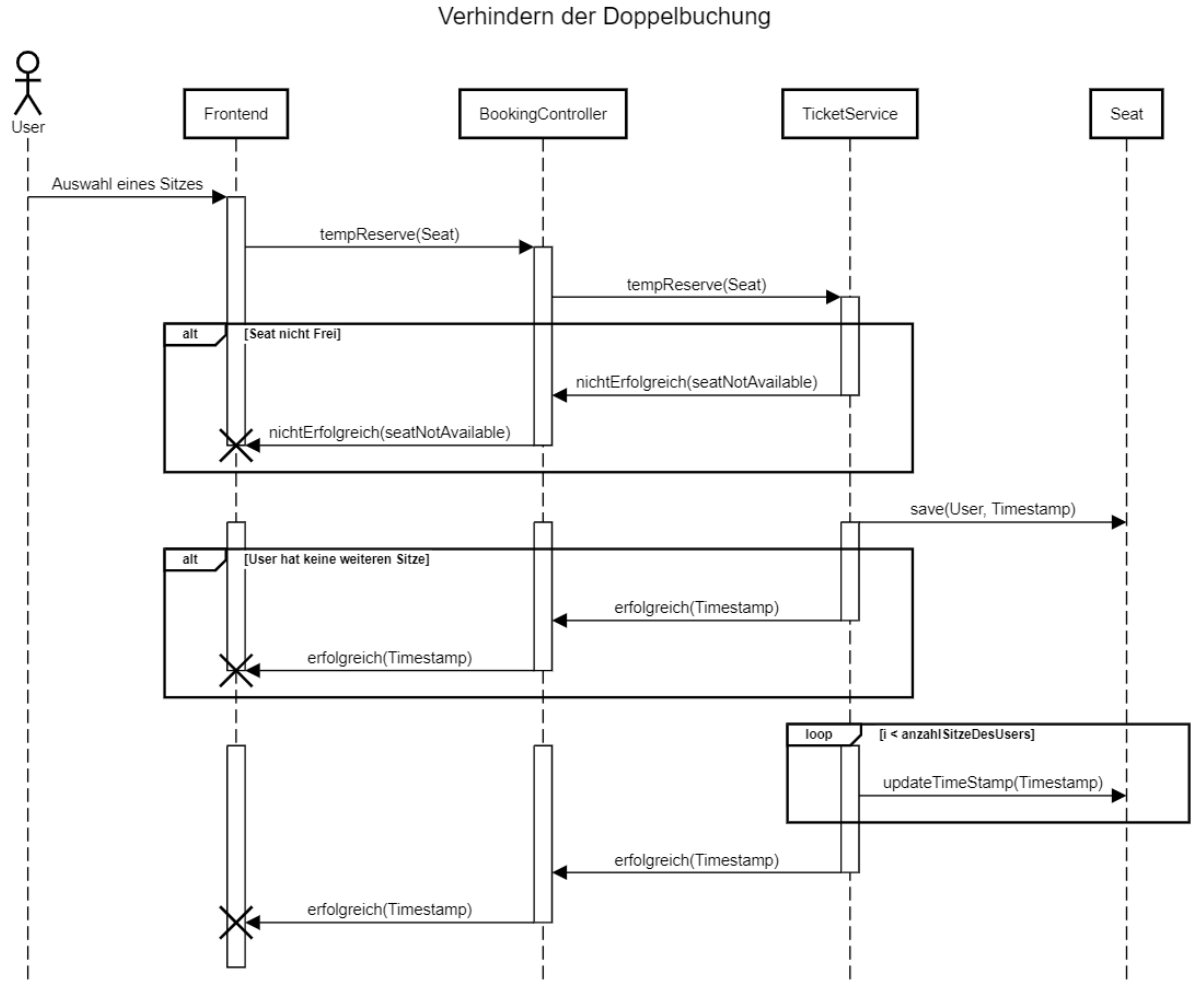
Hosting auf Azure

- App services
    - Unser Backend, sowie unser Frontend werden als App Services über Azure gehosted. Das Frontend ist erreichbar unter (Home - Microsoft Azure (wwi21seb-group5cinema.azurewebsites.net)).
- MySQL-Instanz
    - Unsere MySQL-Datenbank läuft als einzelne Instanz auf Azure.
- GitHub-Actions Pipelines
    - Sowohl im Frontend- als auch im Backend-Repository gibt es eine CD.yml Datei, welche bei Pushes auf den Master-branch das Projekt via Maven baut und auf Azure deployed.

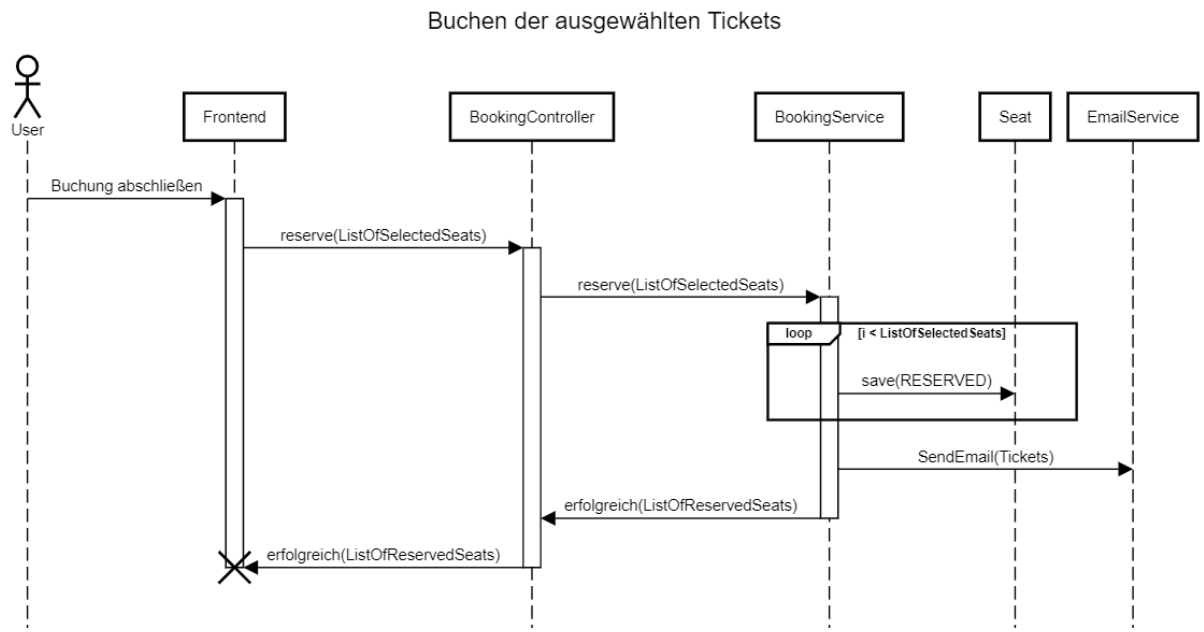
BUCHUNGSPROZESS

Doppelbuchung

Das verhindern der Doppelbuchung erfolgt durch Timestamps, welche in der Seat Entität gespeichert werden. Zusätzlich wird noch die UserID des buchenden Users abgelegt. Ein Seat hat außerdem einen Status, welcher die Werte „FREE, TEMP_RESERVED und RESERVED“ annehmen kann. Fügt ein User einen Seat zu seinem Warenkorb hinzu, dann wird ein Timestamp auf 15 Minuten „in die Zukunft“ gesetzt, vorausgesetzt dieser war zum Zeitpunkt der Auswahl auf „FREE“. Fügt der User einen weiteren Sitz zu seinem Warenkorb hinzu, werden alle Sitze, welche er bereits im Warenkorb hat, auf den neuen Timestamp aktualisiert. Bei jedem Aufruf auf das entsprechende Event wird geprüft ob einer der Timestamps, der Seats, welche auf „TEMP_RESERVED“ sind, abgelaufen ist. Ist dies der Fall wird der Seat wieder auf „FREE“ gesetzt und ist demnach wieder buchbar. Folgendes Sequenzdiagramm stellt diesen Prozess dar:



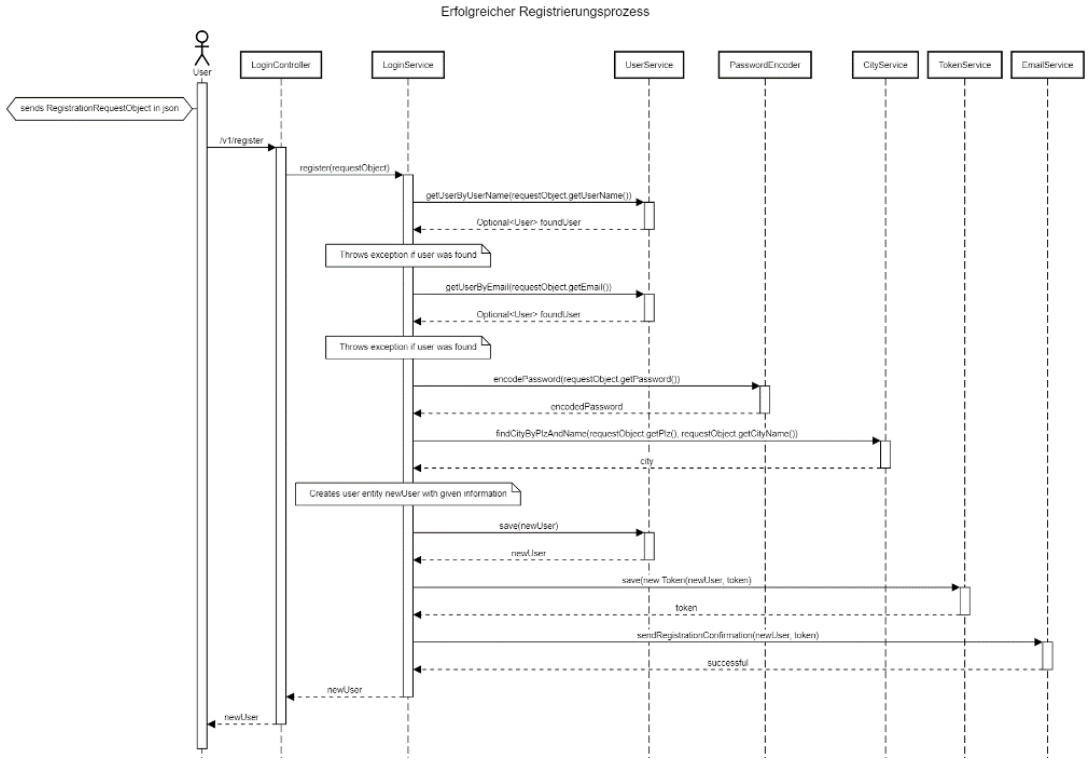
Möchte besagter User die Seats im Warenkorb nun Buchen, dann werden alle Seats im Warenkorb auf „RESERVED“ gesetzt. Hier ist keine weitere Prüfung nötig, da der User nur ursprünglich freie Seats zu seinem Warenkorb hinzufügen kann. Verbringt der User die vollen 15 Minuten in der Sitzauswahl, fordert ihn ein Pop-up dazu auf, auf die Eventauswahl zurückzukehren. Dieser Teil des Buchungsprozess ist in dem folgenden Sequenzdiagramm dargestellt:



REGISTRIERUNGS- UND LOGINPROZESS

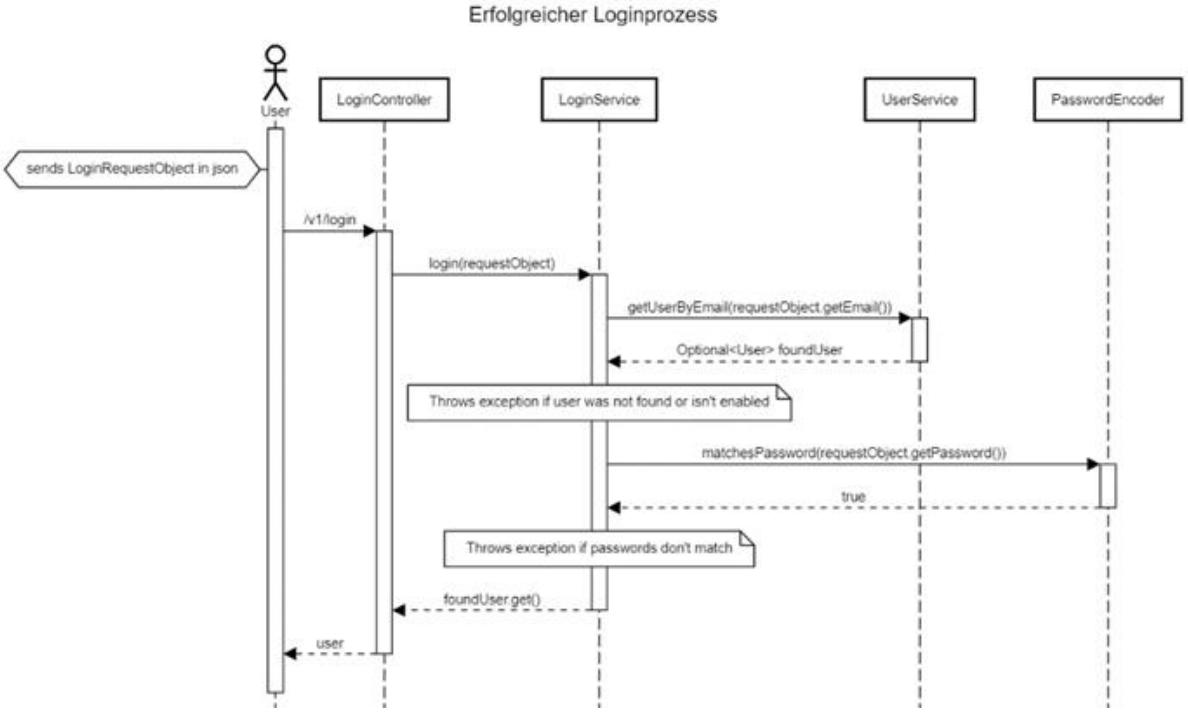
Registrierung

- UserDetailsService
    - Unser Registrierungsprozess läuft über die UserDetails-Funktion von Spring Boot. Dies bietet uns die Funktionalität von Authoritäten, bzw. Rollen, sowie eine Funktion, welche User zwingt ihre E-Mail zu bestätigen.
    - Es gibt die Rollen “USER” und “ADMIN”.
    - Bei der Registrierung wird ein Token generiert und der User erhält eine E-Mail, mit welcher er sich verifizieren muss, bevor er sich anmelden kann.
    - Spring Security greift bei Requests auf den CurrentUserDetailsService zurück und erhält die Authoritäten des Users zur weiteren Verarbeitung.
- Ablauf



Login

- User können sich mit ihrer E-Mail-Adresse und ihrem Passwort einloggen.



ENTWICKLUNGSPROZESS

Um eine gewisse Codequalität zu gewährleisten wurde im Entwicklungsprozess mit Featurebranches und darauffolgenden Pull-Requests über Git gearbeitet. Ein direkter Merge auf den Master-Branch ist unmöglich. Der Pull-Request kann erst auf den Master-Branch gemerged werden, soweit ein Rewiew durch einen anderen Entwickler erfolgte und die Code-Coverage nach den vorgenommenen Änderungen bei mindestens 80% liegt.

SETUP DES PROJEKTS

1. Backend
    a. In einer IDE freier Wahl GitHub-Repository clonen ‘git clone https://github.com/wwi21seb-group5cinema/Cinema-Backend.git’
    b. Notwendige Dependencies herunterladen, ‘mvn clean -U install’
    c. Eine MySQL-Datenbank auf Port 3306 aufsetzen mit vorhandenem Schema: cinema
    d. Automatisch erstellte Runtime-Configuration um profile=dev erweitern, dies initialisiert die vorherig erstellte MySQL-Datenbank
    e. Runtime Configuration starten
2. Frontend
    a. In einer IDE freier Wahl GitHub-Repository clonen ‘git clone https://github.com/wwi21seb-group5cinema/Cinema-Frontend.git’
    b. Notwendige Dependencies herunterladen ‘npm install’
    c. Frontend lokal starten ‘npm start’, läuft auf localhost:3000
    d. Automatisch angelegter Admin-User:
        i. E-Mail. admin@cineverse.de
        ii. Passwort: admin123