

Kino-Verwaltung

Jan Malchert, Marek Hohmann, Bernhard Koll

Kurs: Java EE

Prof. Dr. Alexander Auch

26.06.2019

Inhalt

[Einleitung 1](#_Toc11939816)

[Projektteam 1](#_Toc11939817)

[Beitrag zum Projekt 1](#_Toc11939818)

[Projektplan 2](#_Toc11939819)

[Implementierung 3](#_Toc11939820)

[Frontend 4](#_Toc11939821)

[Backend 4](#_Toc11939822)

[Authentifizierung 5](#_Toc11939823)

[UML 6](#_Toc11939824)

[Persistenz 6](#_Toc11939825)

[Entity-Klassen 7](#_Toc11939826)

[Use-Cases 8](#_Toc11939827)

[Guest 8](#_Toc11939828)

[Customer 9](#_Toc11939829)

[Moderator 9](#_Toc11939830)

[Admin 10](#_Toc11939831)

[API-Dokumentation 11](#_Toc11939832)

[Setup 12](#_Toc11939833)

[Build 12](#_Toc11939834)

[Deployment 12](#_Toc11939835)

Abbildungsverzeichnis

[Abbildung 1 Beispielhafte Darstellung der Nutzung des Entity-Control-Boundary Patterns 4](#_Toc12199037)

[Abbildung 2 UML Abbildung von Security 5](file:///C:\Users\bko\Dropbox\NaJa%20Bernherd%20Kram\Semester%206\JEE\Dokumentation.docx#_Toc12199038)

[Abbildung 3 UML Abbildung der Controls 6](#_Toc12199039)

[Abbildung 4 UML Abbildung der Repositorys mit Specifications 6](#_Toc12199040)

[Abbildung 5 Übersicht der Entity-Klassen 7](#_Toc12199041)

[Abbildung 6 Use-Cases der Guest-Rolle 8](#_Toc12199042)

[Abbildung 7 Use-Cases der Customer-Rolle 9](#_Toc12199043)

[Abbildung 8 Use-Cases der Moderator-Rolle 9](#_Toc12199044)

[Abbildung 9 Use-Cases der Admin-Rolle 10](#_Toc12199045)

# Einleitung

Die Kino-Software bietet eine komfortable Möglichkeit, um die Verwaltung eines Kinobetriebes zu erleichtern. Dies umfasst die Erstellung der jeweiligen Kinosäle mit entsprechenden Sitzplätzen, die angebotene Auswahl an Filmen und deren Vorstellungen in den Kinosälen sowie die Möglichkeit der Reservierung von Plätzen durch Kunden.

## Projektteam

Das Java EE Projekt Kino wurde von den folgenden Studenten aus dem Kurs INF16B der DHBW Mosbach entwickelt und umgesetzt:

* Jan Malchert
* Marek Hohmann
* Bernhard Koll

## Beitrag zum Projekt

Eine Auflistung der Beiträge im Projekt:

|  |  |
| --- | --- |
| Name |  |
| Jan Malchert | **api, control, security, success, repository, util, kino-frontend** |
| Marek Hohmann | **entity, api, control, util, Persistenz** |
| Bernhard Koll | **api, control, security, success, role, Bugfixing, Angular-Integration, Dokumentation** |

## Projektplan

Für das Kick-off-Meeting wurden verschiedene Ideen eingebracht zur Realisierung des Projektes. Hierbei wurde einstimmig beschlossen, eine Kino-Verwaltung umzusetzen.

Im ersten Schritt wurde überlegt, welche Entitäten das Projekt gebraucht werden. Dazu wurden mögliche Use-Cases generiert, wie zum Beispiel: Ein Kunde möchte eine Reservierung für einen Film durchführen. Im nächsten Schritt wurde der Projektrahmen für die Präsentations-, Business-Logik- und Persistenz-Schicht definiert. Es wurde festgelegt für die Präsentationsschicht das Typescript-basierte Framework Angular 6 zu benutzen. Anhand des Programmierparadigmas REST soll Angular mit dem Backend kommunizieren und der Datenaustausch über das Datenformat JSON durchführen. Für die Business-Logik soll eine klare Trennung herrschen, indem das Entity-Control-Boundary-Pattern benutzt wird. Zuletzt soll für die Persistenz-Schicht das Repository-Pattern angewandt werden, damit unter anderem auf einfacher Weise die Persistenz getestet werden kann.

Für die Implementierung wurden weiterhin Richtlinien festgelegt:

* Benutzung von Interfaces
* Separation of Concerns
* Klare Trennung der Schichten
* Statische Methoden, wenn diese pur sind
* Exception Handling über Problems (RFC7807)
* KISS

Für die Kollaboration wurde auch beschlossen, Github als Versionskontrollsystem zu benutzen und entsprechend soll mit dem Workflow „Gitflow“ versioniert werden. Zusätzlich werden serverseitige Hooks mit Travis CI eingesetzt, um ein Continuous Pipeline zu erstellen. Der Fokus der Pipeline ist das Testen und Entdeckung mögliche Probleme der Applikation.

|  |  |
| --- | --- |
| Aufgaben | Status |
| **Planung** |  |
| Kick-off-Meeting | **Abgeschlossen** |
| Workshop Clean Code / Pattern | **Abgeschlossen** |
| **Initialisierung** |  |
| Anforderung Projektrahmen | **Abgeschlossen** |
| Anforderung Entities | **Abgeschlossen** |
| Aufsetzen des Arche-Typen (Java / SQL / Maven) | **Abgeschlossen** |
| **Entwicklung** |  |
| Entities | **Abgeschlossen** |
| Rollenkonzept | **Abgeschlossen** |
| Login (SecurityContext / JWT) | **Abgeschlossen** |
| API (Control ) | **Abgeschlossen** |
| Business Logik (Control) | **Abgeschlossen** |
| Frontend (Angular / Integration in Projekt ) | **Abgeschlossen** |
| Bugfixing | **Abgeschlossen** |
| Dokumentation | **Abgeschlossen** |
| **Betrieb** |  |
| Systemtest | **Abgeschlossen** |
| **Abgabe** |  |

Tabelle 1 Aufgabenabschnitte

In Tabelle 1 sind die von den Teammitgliedern besprochenen Aufgabenabschnitte übersichtlich aufgeführt.

# Implementierung

Das Projekt wurde anhand der 3-Schicht-Architektur Präsentation, Geschäftslogik und Persistenz entwickelt.

## Frontend

Das Frontend wurde mit Angular implementiert. Für die Datenübertragung zwischen dem Frontend und Server wurde im Backend eine RESTful API genutzt und Daten werden anhand von JSON-Objekten kommuniziert.

## Backend

Das Backend wurde in Java EE implementiert. Dabei wurde als Anwendungsserver Wildfly verwendet. Die angebotenen Webservices der RESTful API wurden mithilfe des Frameworks Jersey bzw. JAX-RS entwickelt. Die Geschäftslogik wird durch das Entity-Control-Boundary-Pattern abgebildet. Ein Use-Case wird wie folgt auf die einzelnen Komponenten des Patterns aufgeteilt: Boundary agiert als Schnittstelle zwischen einem User und der Applikation. Die Control ist das Steuerungsobjekt zwischen Boundary und der Entity und bildet die Funktionalität der Business-Logik (Validierung und Mapping zwischen Transfer- Datenbankobjekten) ab. Die Entity repräsentieren die Informationsobjekte aus dem Entity-Modell.

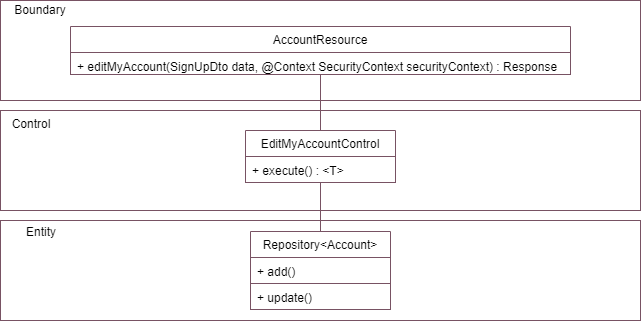


Abbildung 1 Beispielhafte Darstellung der Nutzung des Entity-Control-Boundary Patterns

Abbildung 1 zeigt ein Beispiel eines Use-Cases, welcher die Daten eines Accounts anhand der ID aktualisiert. Hierbei nimmt die Boundary AccountResource in der Methode editMyAccount() durch die HTTP-Anfragemethode PUT ein Data Transfer Object (DTO) an, welches die zu aktualisierenden Informationen des Accounts enthält. Mithilfe des SecurityContext kann der Token, der den User authentifiziert, extrahiert und an die Control EditMyAccountControl übergeben werden. Im nächsten Schritt wird in der Control Funktion der extrahierte Token überprüft und die ID des Accounts extrahiert. Anhand der ID wird die Account-Entity via Account Repository aus der Datenbank geladen und durch die Informationen des DTOs aktualisiert. Die Entity bildet das POJO (Plain Old Java Object) aus der Persistenz ab und nutzt die implementierten JPA Funktionalitäten durch die Bereitstellung eines Entity-Managers und deren Methoden wie (z.B. add,query, remove) zur Kommunikation mit der Datenbank.

### Authentifizierung

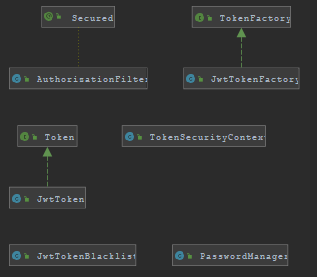


Abbildung UML Abbildung von Security

Für die Authentifizierung und Nutzung von Tokens für Sessions wird die Bibliothek JWT (JSON Web Token) verwendet (siehe Abbildung 2). Die Klasse JWTFactory im package security erstellt, validiert und invalidert einen JWT Token. Der JWT Token enthält die ID des Users, die Haltbarkeit des Tokens und den Token-String. Insgesamt wird dieser mit SHA-512 kodiert und in einem Base64-Format als String generiert. Falls ein User sich ausloggt oder der User sein Account löscht, wird der valide Token kurzzeitig in die JwtTokenBlacklist hinzugefügt, sodass dieser für die Dauer der Validität nicht mehr benutzt werden kann.

### UML

Das Projekt wurde mit Jave EE entwickelt. Nachfolgend sind die Klassendiagramme der Packages control und repository wiederzufinden.

Hierbei zeigt sich, dass control eine abstrakte ManagingControl Klasse besitzt (siehe Abbildung 3), die dafür sorgt, dass die Persistenz-Verbindung zur Datenbank automatisiert geschlossen wird.

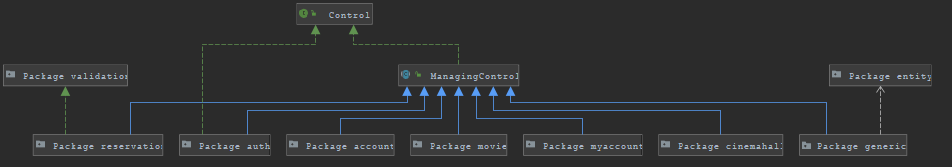


Abbildung UML Abbildung der Controls

In Abbildung 4 in der repository wird dargestellt, wie die Persistenz-Schicht das Repository-, Factory- und Specification-Pattern benutzt wird.

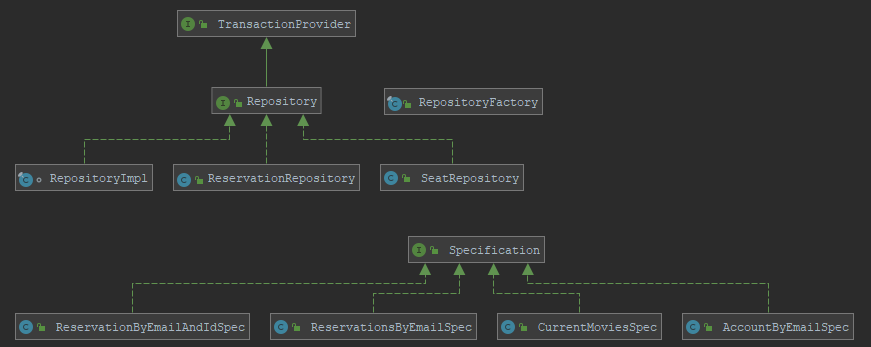


Abbildung UML Abbildung der Repositorys mit Specifications

## Persistenz

Für die Persistenz wurde MySQL mit MariaDB verwendet.

### Entity-Klassen

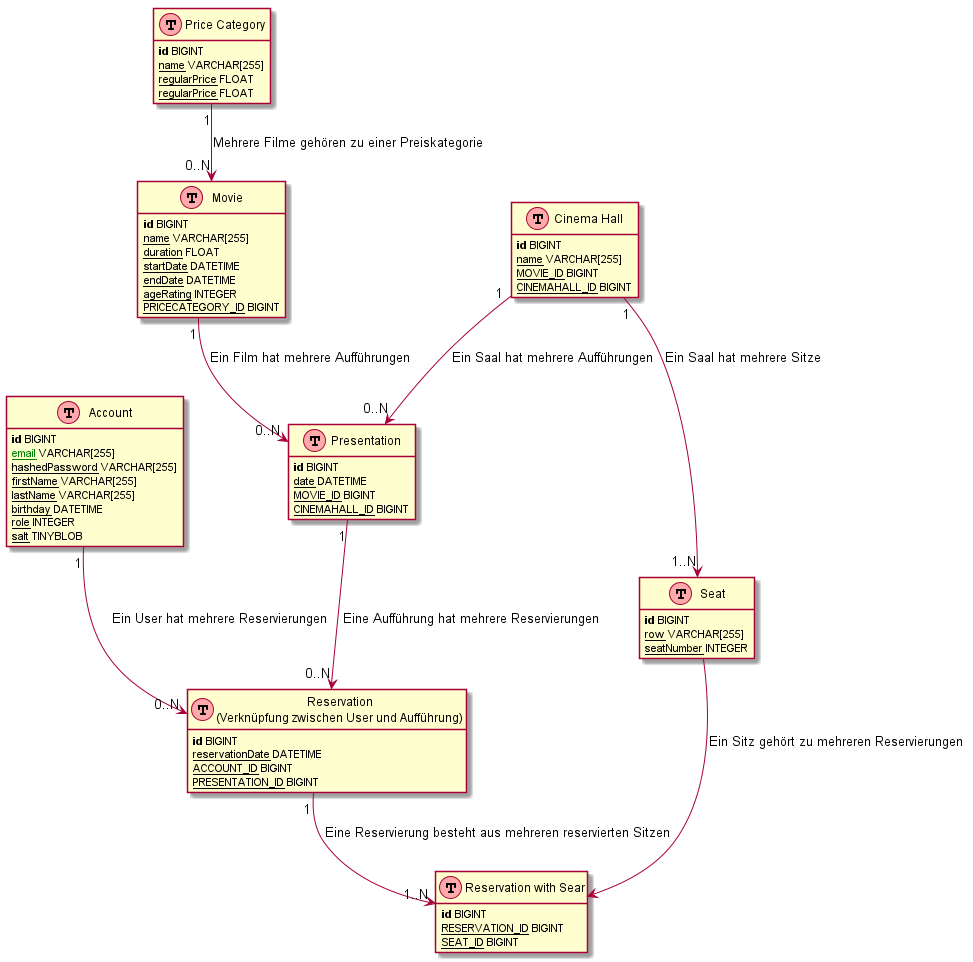


Abbildung Übersicht der Entity-Klassen

# Use-Cases

Die Anwendung besteht aus verschiedenen Use-Cases zur Nutzung der Anwendungen. Dieses wurde aus Sicht verschiedener Nutzergruppen erstellt. Die Nutzergruppen bestehen aus:

* Guest
* Customer
* Moderator
* Admin

Hierbei erbt jede Rolle die Rechte der Vorgängerrolle. Durch die Annotation @Secured und @RolesAllowed(<Rolle>) in der Boundary, kann sichergestellt werden, dass nur authentifizierte User mit der festgesetzten Rolle die Methode ausführen können. Zum Beispiel kann ein Moderator alle Aktionen durchführen, die ein Guest und ein Customer auch durchführen kann. Im Folgenden werden in Abbildung 6, Abbildung 7, Abbildung 8 und Abbildung 9 die verschiedenen Use-Cases geordnet nach den Rollen dargestellt.

## Guest



Abbildung 6 Use-Cases der Guest-Rolle

## Customer

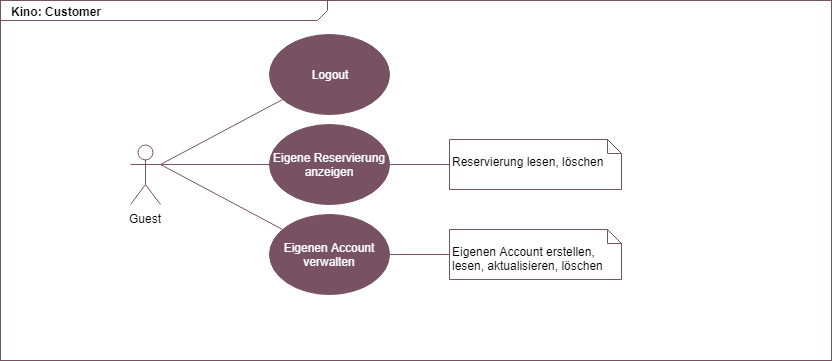


Abbildung Use-Cases der Customer-Rolle

## Moderator

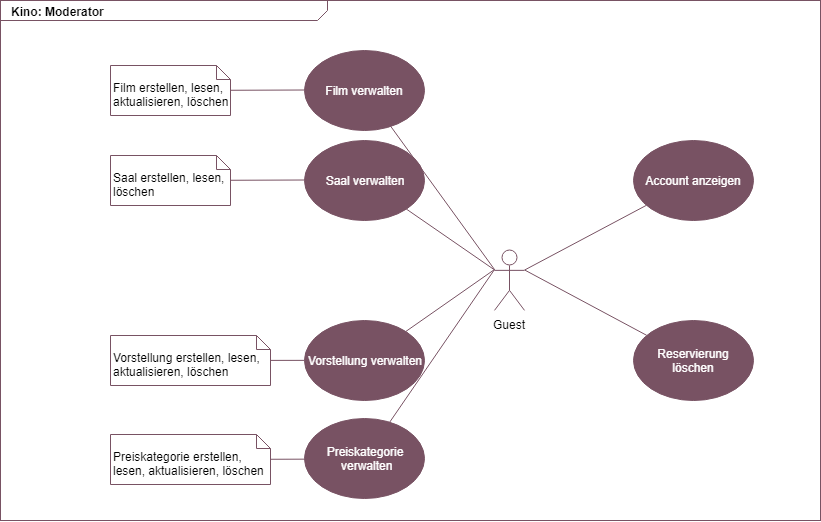


Abbildung 8 Use-Cases der Moderator-Rolle

## Admin

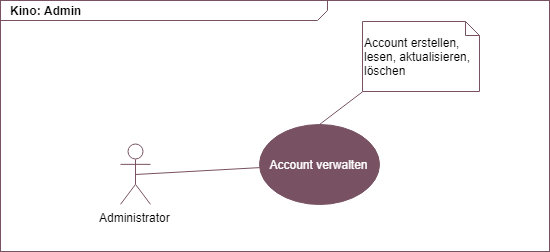


Abbildung 9 Use-Cases der Admin-Rolle

# API-Dokumentation

Hier JavaDoc einfügen

Das kann eigentlich weg oder? Keine Ahnung was man hier schreiben sollte. Haben Swagger und nochmal alles beschreiben ist redundant.

# Setup

Nachfolgende Anleitung gilt ausschließlich für localhost! Hierbei wird das Backend in JEE als auch das Frontend in Angular gebaut.

## Build

Benötigt

* Maven 3.6.0 (<https://maven.apache.org/download.cgi>)
* Node 12.0 (<https://nodejs.org/en/>)
* Java 11 (<https://jdk.java.net/11/>)

./kino>\_ %MAVEN\_ROOT% clean install

## Deployment

Benötigt

* Java (<https://jdk.java.net/11/>)
* Wildfly 16.0.0 Final (<https://wildfly.org/downloads/>)
* MySql Datenbank (<https://www.apachefriends.org/de/index.html>)
* Executable Kino.war (aus Build)

Datenbank Setup

* Nutzung der SQL Skripte in sql-scripts
* Datenbankname: kino
* User-Credential
  + User: jpa
  + Passwort: jpa

Start des Webservers

* Linux wildfly/bin/standalone.sh
* Windows wildfly\bin\standalone.bat

Deployment

* Kopieren von Kino.war nach
  + Linux widlfly/standalone/deployments
  + Windows widlfly\standalone\deployments

Dummy-Login

* Admin-Account:
  + User: [admin@account.de](mailto:admin@account.de)
  + Passwort: admin
* Moderator-Account:
  + User: [admin@account.de](mailto:admin@account.de)
  + Passwort: moderator
* Customer-Account:
  + User: [customer@account.de](mailto:customer@account.de)
  + Passwort: customer
* Customer -Account:
  + User: [customer1@account.de](mailto:customer1@account.de)
  + Passwort: customer1