Duale Hochschule
Baden-Württemberg Mannheim
Coblitzallee 1 – 9
68163 Mannheim


# Dokumentation

unseres Kinoticket-Reservierungssystem

„*Kinovation*“

**Bearbeitet von Gruppe 1:**

Anna Khristolyubova,

Babett Müller,

Frederick Orschiedt,

Jost-Tomke Müller

Marcel Mildenberger,

Mathis Neunzig und

Nathalie Möck

Kurs: WWI 2020 SE B

**Unter Aufsicht von:**

Georg Tielsch

**Im Modul:**

Methoden der Wirtschaftsinformatik – Fallstudie

**Abgabedatum:** 07.02.2022

# Inhaltsverzeichnis

# 1. Einleitung

## 1.1 Motivation

Das in der folgenden Dokumentation beschriebene Projekt entstand aus der Aufgabenstellung des Moduls „Methoden der Wirtschaftsinformatik – Fallstudie“ heraus ein Kinoticket-Reservierungssystem auf Basis unserer Vorarbeit aus dem vorherigen Semester zu entwickeln.

Auf der einen Seite war eine Motivation – natürlich – die Benotung dieses Projekts und die damit verbundene Endnote für das Moduls dieses Semesters. Jedoch war das nur ein Teilaspekt von weiteren:

Wir sahen dieses Gesamtprojekt auch als eine Lernchance für uns an, uns in verschiedenen Sprachen, Systemen und Techniken weiterzubilden. Uns war es dadurch möglich unser Wissen in verschiedenen Teilbereichen weiter zu vertiefen oder gar neu zu generieren. So haben wie neue Programmiersparen und Fähigkeiten in Bezug auf Entwicklungstool erlernt. Des Weiteren bat dieses Projekt uns die Möglichkeit uns in der Kunst des Projektmanagement zu versuchen. Dazu gehört nicht nur das Arbeiten im größeren Team unter dem SCRUM-Konzept, sondern eben auch die Absprache untereinander sowie die gemeinschaftliche Problemlösung miteinander. Im Gegensatz dazu haben wir aber auch einen gewissen Ansporn hinsichtlich des Konkurrenzkampfes innerhalb des Kurses gesehen, denn jeder möchte schließlich die bestmögliche Leistung abgeben, um sein System gut darstellen zu können. Auf Grund der jugend- und alltagstauglichen Thematik eines Kinos fiel es uns sehr einfach und mental in die Problematik sowie die verschiedenen Use-Cases hineinzuversetzen, um somit ein möglichst realitätsnahes System zu entwickeln. Dementsprechend war es auch eine komplett neue Erfahrung ein gesamtes System von Grund auf mit allen möglichen Kleinigkeiten und Entwicklungsschritte aufzuziehen, wie eben bei einem Full-Stack-Development. So waren wir in der Lage alles von ersten Prototypen bis hin zum Bug-Fixing mitzuverfolgen, was für die meisten von uns eine ganz neue Erfahrung darstellte.

## 1.2 Grundlegendes zur Arbeitsweise

Unser SCRUM-Team mit Nathalie Möck als Product Owner und Babett Müller als Scrum Master traf sich jeden Montag um 16 Uhr zu unserem wöchentlichen Stand-Ups, in welchen wir Probleme, die weiteren Vorgehensweisen, o.Ä. besprochen haben.

Dabei beschäftigten sich Jost-Tomke Müller und Mathis Neunzig mit dem Backend, wobei Anna Khristolyubova, Babett Müller, Frederick Orschiedt, Marcel Mildenberger und Nathalie Möck sich um Belange des Frontends kümmerten. Je nach Kapazität und Auslastung übernahmen wir aber auch dynamisch Aufgaben aus anderen Teilbereichen.

# 2. Umsetzung des Projektes

## 2.1 Zielbestimmungen

Aus dem Pflichtenheft gingen für das Projekte diverse Ziele hervor. Diese unterteilten wir in Muss-Kriterien, also Kriterien, die zu einer erfolgreichen Projekterfüllung erfüllt werden müssen und Kann-Kriterien, welche zusätzlich dem Projekt ein höhere Qualität und Funktionalität bringen würde. Während des Entwicklungsprozesses wurde jedoch ersichtlich, dass nicht alle Kriterien mit der begrenzten Zeit und Ressourcen erfüllt werden können. Im Folgenden wird erläutert, welche Ziele wir wie umsetzen konnten und welche nicht.

### 2.1.1 Muss-Kriterien

An unser System stellten wir diverse Muss-Kriterien, welche wir alle Erfolgreich umsetzen konnten. Im Folgenden eine kurze Erläuterung der Ziele und wie wir diese umsetzten:

- **Ziel:** leicht verständliche und übersichtliche Oberfläche für Kunden und Mitarbeiter. Mitarbeiter sollen die Möglichkeit haben, Tickets zu überprüfen.
  **Umsetzung:** Während des Entwicklungsprozesses achteten wir darauf, eine leicht bedienbare Oberfläche für Kunden zu entwickeln. Durch 5 verschiedene Unterseiten kann der Endbenutzer sich leicht über die Menübar zurechtfinden. Es wurden überflüssige Schritte gespart und der Buchungsprozess insofern vereinfacht, dass man vom groben ins feine geht (Film => Vorstellungen => Sitzplan => Snacks => Buchung). Bei einer erfolgreichen Buchung erhält man ein Ticket mit QR-Code per E-Mail zugesendet, welches der Mitarbeiter leicht mit einem QR-Code Scanner überprüfen kann.

- **Ziel:** Mitarbeiter und Kunden unterschiedliche Log-in-Bedingungen und Oberflächen zur Verfügung stellen.
  **Umsetzung:** Hier sind wir auf eine zufriedenstellenden Kompromiss gekommen. Sowohl Kunden als auch Mitarbeiter verwenden für den Log-in die gleiche Oberfläche. Ein Mitarbeiter jedoch, welcher eine andere Rolle (Bsp.: Admin) hat, kann auf diverse Inhalte der Website zugreifen (Bsp.: Admin Seite), welche anderen Benutzern verborgen bleibt.

- **Ziel:** schnellen Überblick über laufende und in der näheren Zukunft startenden Filme ermöglichen.
  **Umsetzung:** Der Benutzer kann sich einen Überblick aller aktuell laufenden Filme auf der Programmseite verschaffen und die Filme nach gewissen Kriterien filtern. Er kann

aber auch den Chatbot verwenden und Theo Ticket Fragen stellen (Bsp.: „Zeig mir alle Filme!“). Über in der näheren Zukunft startenden Filme kann sich der Benutzer auf der „News & Event“ Seite informieren.

- **Ziel:** Zahlung per Kreditkarte ermöglichen und Tickets digital zu erhalten, am besten mit QR-Code.
  **Umsetzung:** Der Benutzer hat die Möglichkeit beim Check-Out zwischen drei gängigen Bezahlungsmöglichkeiten zu wählen (Kreditkarte, PayPal und Klarna). Bei einer erfolgreichen Buchung wird dem Benutzer an seine E-Mail die Tickets mit dem dazugehörigen QR-Code im PDF-Format für einen reibungslosen Check-in gesendet.

- **Ziel:** Benutzern Informationen über die Auslastung und Beschaffenheit der Säle ermöglichen.
  **Umsetzung:** Wenn der Benutzer sich für eine Vorstellung entscheidet, wird ihm der Sitzplan für den dazugehörigen Kinosaal angezeigt. Dabei kann er leicht anhand der Symbole und Farben erkennen, um welche Art von Sitz es sich handelt, ob dieser bereits belegt ist oder noch frei verfügbar ist. Die Auslastung lässt sich an den bereits gebuchten Plätzen erkennen.

Als Fazit lässt sich hier also ziehen, dass alle Muss-Kriterien genauso umgesetzt wurden, wenn nicht sogar besser als geplant. Dadurch konnten wir mit unserem System den Anforderungen des Pflichtenheftes gerecht werden.

### 2.1.2 Kann-Kriterien

Während des Planungsprozess stellten wir auch diverse Kann-Kriterien auf, welche Vorteilhaft wären, um die Qualität unseres Systems zu verbessern, jedoch nicht zwingend notwendig sind, um die Anforderungen zu erfüllen. Mögliche Ziele und deren Umsetzungen:

- **Ziel:** Basierend auf bisherigen Buchungen Vorschläge machen („Das könnte ihnen auch gefallen“).
  **Umsetzung:** Dieses Feature konnte nicht in der Implementierung unseres Systems umgesetzt werden.

- **Ziel:** Großkunden Säle anzeigen, die zu dessen Forderungen passen
  **Umsetzung:** Für Großkunden gibt es keine besondere Implementierung. Sie müssen genauso wie normale Nutzer sich eine Vorstellung raussuchen und überprüfen, ob genug bzw. die gewünschten Plätze vorhanden sind. Großbestellungen sind möglich.

- **Ziel:** Benutzern die Möglichkeit geben, Buchungen online zu stornieren
  **Umsetzung:** Unter der Profil-Seite können Benutzer ihre Buchungen mit dem dazugehörigen QR-Code einsehen und sich Details zu der Buchung ansehen sowie diese stornieren.

- **Ziel:** Treuepunkte für Kunden
  **Umsetzung:** Dieses Feature konnte nicht in der Implementierung unseres Systems umgesetzt werden.

- **Ziel:** Doppelte Buchungen erkennen und verhindern
  **Umsetzung:** In der Webanwendung können mehrere Clients den gleichen Sitz zum Kaufen auswählen und damit bis zum Check-Out gelangen. Wir nehmen keine „Blockierung“ vor, wenn der Benutzern bereits im Sitzplan den Platz auswählt. Unser Interesse liegt in der Gewinnmaximierung, weshalb Benutzer den gewünschten Platz erhalten sollen, wenn Sie auch dafür zahlen. Sollten zwei Benutzer jedoch gleichzeitig einen Platz kaufen wollen, so wird im Backend über Synchronisation (Semaphore) dies verhindert, und nur einer erhält die erfolgreiche Buchung.

- **Ziel:** Einlasspersonal soll die Position der Sitze ermitteln können, um Kunden eine präzisere Wegbeschreibung geben zu können.
  **Umsetzung:** Das Einlasspersonal kann die QR-Codes der Tickets scannen, um so Details über deren Buchung zu erhalten.

- **Ziel:** Statistiken
  **Umsetzung:** Wir sammeln aktuell keine Statistiken über durchgeführte Buchungen etc.

- **Ziel:** Sonderaktionen
  **Umsetzung:** Wir bieten diverse Aktionen auf unsere „News & Events“ Seite für Kunden an.

## 2.2 Einsatz

### 2.2.1 Anwendungsbereiche

Das System bietet die Möglichkeit den gesamten Prozess des Ticket-Erwerbs abzubilden und automatisch zu managen. Dies ist sowohl für Kunden als auch für das Personal im Kino möglich. Jedoch nur mit der Einschränkung, dass jegliche Transaktionen online stattfinden

müssen. Demnach ist zum jetzigen Zeitpunkt keine Überprüfung oder Ausgabe der Tickets vor Ort möglich.

### 2.2.2 Zielgruppen

Die Zielgruppe des Kinoticket-Reservierungssystem bezieht sich auf 2 Hauptgruppen:

- Kunden, welche ein oder mehrere Tickets bestellen können, ohne eine Art „besonderer Unterstützung oder Hilfestellung“ zu benötigen und
- dem Personal des Kinos bzw. der Kinokette und somit des gesamten Verwaltungssystems.

Das Personal besteht in der Realität wiederum aus dem Einlasspersonal, den Kassierer sowie den System- bzw. Filmadministrator.

In unserer Version des System sind bis jetzt nur die Rolle des Kunden sowie des Administrators vorgesehen. Letztere sind für die Verwaltung der einzelnen Kinofilme sowie ihrer Vorstellung berechtigt und benötigt. Des Weiteren können sie neue Events und News anlegen und diese auf der Webseite verwalten.

Im Kino selbst ist es für das Einlasspersonal möglich, den durch die Buchung generierten QR-Code zu scannen und die Käufer des Kinotickets somit für die Vorstellung „einzuchecken“. Dies gewährleistet besonders in Corona-Zeiten einen sicheren Umgang mit den Kinobesuchern und verbessert auf Grund der Zeitersparnis auch die Erfahrung der Besucher durch das schnelle und effektive Einchecken (z.B. mittels Handy).

### 2.2.3 Betriebsbedingungen

Das System ist mit einer Internetverbindung von jedem Ort aus bedienbar. Dies liegt vor allem an unserer guten Verfügbarkeit der Datenbank und des Servers durch externe Provider. Im Moment wurde nur eine Desktop-Website realisiert. Diese ist zwar auch mobil nutzbar, besitzt aber noch nicht die entsprechende, optische Skalierung.

Das System besitzt eine Verfügbarkeit („Uptime“) von 99,99%. Sollte es jedoch zu Problemen auf Seiten des Hosts kommen (Microsoft Azure oder Vercel), so kann eine Verfügbarkeit der Webseite nicht gewährleistet werden. Die Ursache dafür liegt dann jedoch beim Anbieter der Plattform und nicht in der Programmierung des Systems. Demnach ist die Webseite so ausgelegt, dass sie 24 Stunden an 7 Tagen die Woche erreichbar ist. Für Wartungen bzw. Updates ist der Zeitraum zwischen 4:00 Uhr bis 4:30 Uhr festgelegt, da zu dieser Zeit der geringste (fiktive) Kundenverkehr stattfindet. Wie ursprünglich geplant, ist dafür jedoch keine Back-Up-Webseite verfügbar, da diese aus Zeitgründen noch nicht implementiert wurde. Das heißt, der komplette Kunden- und Zahlungsverkehr ist in diesem Zeitraum entsprechend nicht möglich. Auf Grund des gewählten Zeitraums sollte dies jedoch kein Problem darstellen.

Eine ständige Beobachtung des Systems wurde nicht implementiert, da diese für obsolet in Bezug auf ein Kinoticket-Reservierungssystem gehalten wurde. Demnach war sie von Anfang an als „nicht benötigt“ eingestuft und auch nicht vorgesehen.

## 2.3 Umgebung

### 2.3.1 Backend

Architektur

Für dieses Projekt wurde die Entscheidung getroffen eine Monolithische Architektur zu verwenden. Dies hat den Grund, dass für diese Anwendung wichtig ist alle unbedingt notwendigen Teile an einem Ort zu haben. Nach genauer Prüfung wurde entschieden, dass der Kern der Anwendung, welcher die Buchung von Tickets und die Verarbeitung sämtlicher Funktionen darstellt, keinen Platz zur Aufteilung bietet. Lediglich intern wurde eine Aufteilung in einzelne Regionen vorgenommen.

Als Framework für diese Software wurde Java Spring Boot gewählt, da dieses einige wichtige Funktionen für einen Webservice von Haus aus mitbringt und sich sehr leicht erweitern lässt. Zudem ist dies ein Framework, welches sehr schnell startet, sollte es gerade nicht laufen. Außerdem benutzt dies Java, womit die Programmierer dieses Projektes gut umgehen können und sich nicht erst einarbeiten müssen.

Ein Teil der Anwendung wurde allerdings ausgegliedert, dabei handelt es sich um den virtuellen Assistenten „Theo-Ticket“, da dieser keine essenziellen Funktionen für das System umsetzt, wurde entschieden diesen in einen externen Server auszugliedern und über HTTP Calls mit dem Hauptserver zu kommunizieren.

**Interne Aufteilung**

Um eine gute Wartbarkeit gewährleisten zu können wurde die Software intern in Funktionsabhängige Teile gespalten. Diese Teile sind zumeist auf die einzelnen Tabellen bezogen, also dass jede Tabelle der Datenbank in der Software einen eigenen Controller und Repository besitzt. Bei Komponenten, welche für mehrere Funktionen benutzt werden, wurde allerdings entschieden Services einzuführen, welche von mehreren Controllern aufrufbar sind. Die wichtigsten Beispiele sind hier der Seat, Show und E-Mail-Service, zudem auch der Semaphore Vault Service. Um dies zu erklären, wird kurz auf zwei Services eingegangen.

**Seat Service**

Der Seat Service wir an allen Stellen verwendet, an welchen mit einem Sitz interagiert wird, was beim Erstellen neuer Filme, dem Buchen eines Platzes, der Stornierens und vielem mehr getan wird. Um hier nicht überall den gleichen Code schreiben zu müssen wurden diese

zentralen Funktionen für die Sitze unterzubringen, wie das Blocken, Erstellen, Löschen, etc. eines Sitzes.

**Semaphore Vault Service**

Der Semaphore Vault Service ist der für die Konsistenz der Daten wichtigste Service. Dieser regelt den Zugriff auf die kritischen Daten dieses Systems, vor allem dem Zugriff auf Sitze um diese zu Blocken. Die Funktionsweise dieses Services ist es für jede Show ein Semaphore zu erstellen und in einer internen Struktur zu speichern und allen weiteren Anfragen auf diese Show zu geben, damit unter keinen Umständen eine Doppelbuchung im System landen kann.

Abhängigkeiten

Dieses Projekt integriert einige zusätzliche Spring Boot Komponenten, wie Spring Boot Starter Data JPA, JDBC, Security, Web. Auch Spring Boot externe Komponenten werden von diesem Projekt verwendet. Der Grund für all diese Abhängigkeiten wird nun Stück für Stück erklärt werden.

**Spring Boot Starter Data JPA**

Diese Abhängigkeit ermöglich es die Datenbank sehr leicht zu manipulieren, da JPA die Möglichkeit gibt für alle Tabellen Repositories zu erstellen. Wenn solch eines erstellt wird, werden automatisch einige Methoden zum Manipulieren oder Durchsuchen der Daten erstellt, wie Filtern nach alles Attributen, Erstellen von neuen Tupeln, löschen von Tupeln, etc. Selbst sollte man eigene Methoden benötigen, so kann man diese im Interface des Repositories hinzufügen und das meist nur mit einer Deklaration der Methode, der schlimmste Fall ist, wenn man selbst eine Query schreiben muss. Alles in allem macht diese Abhängigkeit das Leben des Backend-Entwicklers einfacherer!

**Spring Boot Starter JDBC & MariaDB JDBC**

Diese Abhängigkeit ist absolut notwendig, da diese den Treiber zur Verbindung mit der Datenbank beinhaltet. Dies stellt auch den Grund dar, weswegen wir diese Komponente aufgenommen haben.

**Spring Boot Starter Security**

Diese Komponente ermöglich es auf einfachste Weise den Zugriff auf die Applikation einzuschränken. Für dieses Projekt wurde diese genutzt, da nicht jeder Server unbegrenzten Zugriff auf diese Applikation haben soll, sondern nur diese die für den Betrieb des Frontend unablässig sind, was das Frontend ist, Theo-Ticket und die Entwicklungsumgebungen, sowie localhost. Dies ermöglicht diese Komponente durch den Aufruf einer einzelnen Methode.

**Spring Boot Starter Web**

Diese Komponente ermöglich dem Backend erst über die meist genutzte Schnittstellt zu kommunizieren, nämlich der HTTP REST Calls. Dies sind die bekannten Methoden des Internets, wie „GET“, „PUT“, „DELETE“, etc. Um diese ohne weiteren Aufwand verwenden zu können, wurde diese Komponente aufgenommen, womit die Aufrufe sehr einfach umgesetzt werden können.

**Spring Boot Session**

Wie der Name der Komponente andeutet ermöglich diese das Erstellen einer Session für einen Nutzer. Genauer können hiermit Cookies beim Benutzer gespeichert werden, welche die Informationen des Benutzers speichern und damit auch seine Login Informationen, damit der Benutzer beim Wechsel der Seite eingeloggt bleibt.

**Spring Boot Starter Mail**

Diese Komponente ermöglicht es diesem Projekt auf sehr einfache Art und Weise Emails zu versenden. Dies wird hier dafür eingesetzt dem Benutzer bei der Registrierung, wenn dieser sein Passwort vergessen hat und wenn dieser ein Ticket gebucht hat eine Mail zu schicken.

**JUnit**

Diese sehr bekannte Komponente wird auch in diesem Projekt verwendet. Hier wird es zum Testen aller Komponenten des Backend eingesetzt.

**Google ZXing**

Diese Komponente wird verwendet, um verschiedenste Codes erstellen zu können. Zu diesen gehören auch die bekanntesten Codes, wie Barcodes und QR-Codes. Für dieses Projekt wurde diese Komponente gewählt, da sie das einfache Erstellen eines QR-Codes über die UUID einer Buchung ermöglicht. Da dies nur unter schwierigeren Bedingungen möglich gewesen wurde und ZXing dies enorm vereinfacht, wurde beschlossen einfach diese Komponente zu integrieren.

**Ralf Stuckert PDF Box-Layout & Apache PDF Box**

Diese beiden Komponenten werden ebenfalls für die Mails verwendet, hier allerdings nicht um diese zu versenden, sondern um die Anlagen zu generieren – genauer gesagt, um die PDFs zu formatieren. Was ohne diese Komponenten ein riesiger Aufwand wäre, wird mit dieser Komponente um einiges leichter und erlaubt es der Anwendung in wenigen Schritten PDFs zu erstellen, welche dann in Mails an den Benutzer verschickt werden.

Datenbank

**Technische Grundlagen**

Als Datenbank wurde für diese Applikation MariaDB verwendet. Diese Entscheidung wurde gefällt, da diese Datenbank die kostenfreie Version der MySQL ist. Weitere Gründe für die Wahl dieser Datenbank sind, dass für diese bereits Vorkenntnisse vorhanden sind, was die Benutzung erleichtert. Zudem stellt Spring Boot für diese einen JDBC Treiber zur Verfügung. Da für diese Anwendung keine Aspekte von sozialen Medien oder Big Data geplant sind, wurde hier auf die Benutzung einer NoSQL Datenbank, speziell einer Graphen-Datenbank verzichtet und eine einfache relationale Datenbank gewählt. Dazu bietet diese Datenbank die für dieses Projekt benötigte Geschwindigkeit und kann genug Daten speichern.

Zum Start des Projekt wurde demnach zuerst ein vorläufiges Datenbankmodell [siehe Anhang 1] erstellt, welches das vorläufige Grundgerüst unseres Systems darstellte. Dieses hat sich dann im Laufe des Projekt erweitert und wurde beständig an die neuen Funktionalitäten angepasst.

### 2.3.2 Frontend

TypeSkript

Dieses Projekt wurde in TypeScript geschrieben, einer Programmiersprache, die als Entwicklungswerkzeug für Webanwendungen positioniert ist und die Möglichkeiten von JavaScript erweitert. In unserem Fall wurde TypeScript aus den folgenden Gründen vorgezogen:

- Viele Probleme in JavaScript entstehen durch die dynamische Typisierung. In TypeScript ist die Typisierung statisch, was dazu beiträgt, viele der auftretenden Probleme zu beseitigen.
- Das wiederum macht das Projekt schneller: TypeScript trägt dazu bei, die Zeit für die Identifizierung und Behebung von Fehlern zu verkürzen, die in einer dynamischen JavaScript-Umgebung oft schwer zu erkennen sind. Fehler, die bei der Codeänderung gemacht werden, sind sofort sichtbar, nicht erst zur Laufzeit.
- Außerdem ähnelt der TypeScript-Code dem JS-Code, was ein schnelleres Erlernen von TypeScript ermöglicht, was angesichts der knappen Fristen für dieses Projekt von entscheidender Bedeutung ist.

React

Die JavaScript-Bibliothek React ist als leistungsstarkes Werkzeug für die Erstellung dynamischer und interaktiver Benutzeroberflächen bekannt. Mit React lassen sich Schnittstellen aus einzelnen Komponenten aufbauen, die leicht zu verwalten sind. Dies erleichtert die Strukturierung und Wiederverwendung von Code und wird langfristig Vorteile

bringen. Außerdem ist eine React-Komponente einfacher zu erstellen, weil sie JSX verwendet, eine optionale JavaScript-Syntaxerweiterung, die es ermöglicht, HTML mit JavaScript zu kombinieren (in unserem Fall TSX und Typeskript).

Das nächste Argument, das für React spricht, ist das Konzept des virtuellen DOM (Document Object Model). Damit ist die direkte Arbeit mit dem DOM nicht mehr nötig, stattdessen wird eine leichtgewichtige Kopie des DOMs verwendet. Die Änderungen können an der Kopie vorgenommen und dann auf das echte DOM übertragen werden. Zu diesem Zeitpunkt wird der DOM-Baum mit seiner virtuellen Kopie verglichen, der Unterschied wird erkannt und eine Neugestaltung dessen, was sich geändert hat, ausgelöst. Dadurch wird die Geschwindigkeit Ihrer Anwendung nicht beeinträchtigt.

Es ist erwähnenswert, dass React von Facebook entwickelt wurde und einer großen Gemeinschaft von Entwicklern unterstützt wird. Daher ist es gründlich getestet, wird regelmäßig aktualisiert und gewartet, und der Übergang zu neuen Versionen erfolgt so reibungslos wie möglich.

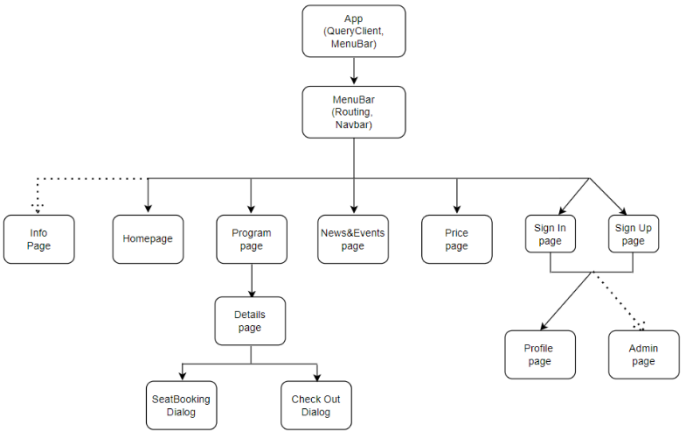
Struktur des Projekts

Die Gruppierung der Dateien nach Dateityp wurde gewählt, um die Dateien in Projekte zu strukturieren. Zu diesem Zweck wurden die folgenden Verzeichnisse definiert:

1) **Komponenten:** dies ist im Wesentlichen unsere Bibliothek benutzerdefinierter Komponenten. Das Verzeichnis enthält außerdem zwei Unterabschnitte:
   - **Dialoge:** Dieses Verzeichnis enthält alle im Projekt verwendeten Dialogfenster.
   - **Layouts:** Hier befinden sich die Komponenten, die allen Seiten einer Anwendung gemeinsam sind. Insbesondere die Navigationsleiste und die Ladeanimation werden hier platziert.
2) **Seiten:** hier werden alle Seiten eingerichtet - und zwar so, dass sie angezeigt werden können.
3) **Config:** hier werden Daten über die verwendete API und die Farbpalette gespeichert.

Frontend-Architektur

Der gesamte Code befindet sich in der App. Routing und die Navigationsleiste sind in der Menubar implementiert. Danach folgen die fertig gestalteten Seiten. Jede besteht aus einer Reihe von Komponenten, die das Markup und ggf. die zugehörige Logik enthalten.



Zusätzlich zu den Seiten, die allen Nutzern der Website zur Verfügung stehen, wurden auch die Admin-Page und die Info-Page eingerichtet, die nur für Kinomitarbeiter mit den entsprechenden Zugriffsrechten zugänglich sind.

Dependencies

**React query**

Zum Abrufen, Synchronisieren, Aktualisieren und Caching serverseitiger Daten wurde die React Query Library verwendet. Zu den Vorteilen gehören beispielsweise: automatische Caching, automatische Datenaktualisierung. Der Entwickler muss nur angeben, wo die Daten gespeichert werden sollen, und die Aktualisierung und Invalidierung der Daten erfolgt im Hintergrund ohne zusätzlichen Code oder Konfiguration.

Darüber hinaus reduziert diese Bibliothek den Umfang des Codes erheblich (z.B. entfällt die Notwendigkeit von useState- und useEffect-Hooks: sie werden durch einige Zeilen ersetzt). Das Abrufen von Daten über verfügbare Filme aus dem Backend erfordert zum Beispiel nur ein paar Zeilen:

```
const apiUrlMovies = `${APIUrl.apiUrl}/movie/getAll`;

const moviesData = useQuery("Movies", () =>
  fetch(apiUrlMovies).then((res) => res.json())
);
```

Sollte die Anwendung in Zukunft weiterentwickelt werden, wird es mit React Query außerdem recht einfach sein, eine Paginierung einzurichten, um nur bestimmte Daten auf dem Bildschirm anzuzeigen (z.B. im Bereich „News & Events“ oder in den Rezensionen auf der Filmdetailseite).

**Material UI (MUI)**

MUI ist eine einfache und anpassungsfähige React-Komponentenbibliothek, die auf Googles Material Design basiert. Material Design bietet viele vorgefertigte Komponenten wie Buttons, Dropdown-Menüs, Switches, Toobars, usw. Die Bibliothek verfügt über eine umfangreiche Dokumentation und bietet die notwendigen Informationen zur Verwendung der einzelnen Komponenten, so dass selbst mit wenig Designerfahrung schnell saubere und attraktive mobile oder Webanwendungen erstellt werden können. Gleichzeitig kann Material UI mit Styling-Mitteln wie einfachem CSS kombiniert werden.

Im Gegensatz zu Bootstrap sind für Material Design keine JavaScript-Frameworks oder -Bibliotheken für die Entwicklung von Websites oder Anwendungen erforderlich. Außerdem ist MUI gut mit verschiedenen Browsern kompatibel.

**React Router**

Routing ist ein wichtiger Bestandteil jeder soliden Webanwendung. Vor allem ermöglicht das Routing dem Benutzer zu sehen, wo er sich zu einem bestimmten Zeitpunkt in der Anwendung befindet. Es ist auch wichtig, dass man sich durch Historie navigieren kann.

React selbst unterstützt kein Routing, aber es gibt leistungsfähige Bibliotheken mit einem ganzen Arsenal an Funktionen. In dieses Projekt wurde die React-Router-Bibliothek integriert. Diese Bibliothek ist beliebt, einfach zu benutzen und hat eine gute Dokumentation.

React Router reduziert den Arbeitsaufwand, der mit der manuellen Konfiguration von Routen für alle Seiten in der Anwendung verbunden ist, indem es uns drei Hauptkomponenten zur Verfügung stellt, die uns helfen, das Routing zu vereinfachen - Route, Link und BrowserRouter.

### 2.3.3 Hosting

Das Backend wird auf Microsoft Azure gehostet. Microsoft Azure bietet Studentinnen und Studenten einen kostenlosen Account mit Startguthaben und Zugang zu vielen kostenlosen Services, die für die Entwicklung eines Backends benötigt werden. Azure stellt Webseiten bereit, auf denen das Backend deployed werden kann, sowie die nötigen Automatisierungsmechanismen, damit dies auch automatisch und einfach passieren kann. Zusätzlich stellt Azure auch Datenbanken zur Verfügung, von denen jedoch hier kein Gebrauch gemacht wird. Der Chatbot TheoTicket wird wie das Backend auf Microsoft Azure gehostet, da die gleiche Technologie wie im Backend genutzt wird.

Die Datenbank wird auf einem privaten Server in einem Frankfurter Rechenzentrum betrieben, mit welchem schon vor Entwicklung dieser Applikation zusammengearbeitet wurde. Dadurch sind bereits Anbieter, Techniker, Statistiken und Gebäude bekannt. Intern werden alle Daten gespiegelt gesichert, damit der Fall eines Datenverlustes sehr unwahrscheinlich wird. Um einen parallelen Betrieb der Entwicklungsumgebung und Produktionsumgebung ermöglichen zu können, wird eine Produktive und eine Entwicklungsdatenbank gehostet.

Im Frontend wird durch einen anderen Einsatz von Technologien auf Microsoft Azure verzichtet, da das Hosten dort mit den Frontendtechnologien, die auf der Webseite genutzt werden, langsamer und störanfälliger ist, als es beim Backend der Fall ist. Deshalb wird das Frontend auf Vercel gehostet, welcher alle Anforderungen erfüllt. Auf Vercel kann das Frontend geladen und gehostet werden, nachdem auch hier automatische Prozesse zum deployen ablaufen, damit dies nicht von Hand passieren muss. Im Gegensatz zu Azure hat Vercel jedoch nicht so viele Überwachungs- und Konfigurationsfunktionen, jedoch sind alle Deployments stabil und funktionieren reibungslos.

### 2.3.4 Architekturmodelle und -diagramme

Entity-Relationship-Modell

Bevor das Projekt vom Backend heraufgezogen wurde, benötigten wir ein ER-Modell, nach welchem wir die Datenbank aufbauen und strukturieren konnten, sodass alle benötigten Daten im Frontend zur Anzeige und im Backend zur Aufbereitung verschiedenster Objekte vorhanden sind. Dementsprechend musste auch bereits die Beziehung der Entitäten untereinander definiert werden, sodass man dem Benutzer die gewollten Daten über entsprechende API-Calls anzeigen konnte.

Das ER-Modell hat sich vom Anfang natürlich im Laufe des Projekts verändert. Im Anhang sind sowohl der erste Entwurf sowie das endgültige Modell zu finden [siehe Anhang 2]. Alle neuen Teilbereiche genaustens zu erklären, wurde vermutlich aber den Rahmen der Dokumentation sprengen, weshalb wir uns nur auf die folgenden Kernpunkte beschränkt haben.

Auffällig ist vor Allem, dass im endgültigen Modell ein großer Fokus auf den Benutzer und seine Beziehungen mit den verschiedenen Entitäten gelegt wurde. Dies liegt an der vermehrten Verwendung von Cookies, um den Kunden die bestmögliche User Experience liefern zu können und somit den Ticketkauf bequem zu gestalten.

Weiterhin sind Entitäten wie „Neuigkeiten“, „Events“ oder aber auch die komplette Thematik der dynamischen Kinosälen mit „Blueprints“ für die einzelnen Sitze hinzugekommen, die den Aufbau unserer Webseite wesentlich dynamischer gestalten. So kann man individuell neue Kinosäle z.B. hinzufügen.

Hervorzuheben ist, dass wir die Entitäten „Tickets“ und „Buchungen“ voneinander separiert haben, um den Prozess des Kaufens sowie Reservieren von Sitzen in Vorstellungen besser verwalten zu können. Auch das Hinzufügen von Snacks für die Vorstellung ließ sich durch die getrennte Betrachtung dieser zwei Objekte leichter realisieren. So werden Snacks und die Blockierung von Sitzen über die Buchung der Vorstellung realisiert. Während Der Preis für einen Film bzw. einen bestimmten Saal oder Sitz über das Ticket geregelt werden, welches in Verknüpfung zur Vorstellung dann zusammengeführt wird. Das Gesamtkonstrukt kann dann einem Benutzer zugehordnet werden. Dies haben wir so realisiert, da ein Benutzer mehrere Tickets buchen kann, es sich dabei dann jedoch nur um eine einzige Buchung handelt.

Sequenzdiagramm

Das Sequenzdiagramm des Projektes [siehe Anhang 3] stellt den grundlegenden Ablauf dar der Umgesetzt wurde, um ein Ticket auf unserer Plattform zu erwerben. Dabei geht die erste Aktion vom Nutzer aus, welcher allgemeine Informationen über die verfügbaren Filme im Kino erhalten möchte. Dafür kann er auf unsere Programmübersicht Seite über den entsprechenden Knopf im Menü steuern, da dies eine grundlegende Funktion der Anwendung

ist soll sie dort ohne Probleme zu finden sein. Dort wird ihm die Überblicks Ansichten der verschiedenen Filme angezeigt, um sicherzustellen der Nutzer hat eine breites Angebot an Informationen und kann frei wählen. Von dort möchte der Nutzer mehr Informationen zu einem bestimmten Film erhalten. Dies wurde bei uns über zwei Wege umgesetzt: Zum einen kann der Nutzer normal über die bereits angezeigte Programmübersicht des Kinos einen Film auswählen, welcher dann über einen eigenen Vorstellungsplan verfügt mit entsprechenden Zeiten, aus denen der Nutzer wählen kann. Dies wurde umgesetzt, um dem Nutzer eine Anwendung repräsentieren zu können die einen guten End-to-End-Flow vorweist und unterwegs möglichst keine Abstecher benötig. Zum anderen besteht die Option den Chatbot von überall aus auf der Website nach Informationen zu einem bestimmten Film zu Fragen, dazu muss lediglich der Name des Films angegeben werden, damit eine Verlinkung zur Informationsseite des Films zurückgegeben wird. Hier wurde eine Variante für Nutzer geschaffen die bereits mit konkreten Absichten Tickets für einen bestimmten Film zu kaufen die Website besuchen und sich so einige Klicks durch das Menü sparen können.

Nachdem der Besucher nun einen Film und eine Vorstellung des Films ausgewählt hat, möchte er den Preis seines Kinobesuches erfahren. Dies geht zum einen automatisch über das Auswählen von Tickets auf dem generierten Sitzplan der entsprechenden Show. Dies entspricht auch dem Weg, den das Sequenzdiagramm geht und ist ebenfalls im Interesse des durchlaufenden Ablaufes einer Buchung implementiert worden. Anderweitig kann ein Nutzer jederzeit auf den separaten Menüpunkt zur Preisübersicht des Kinos steuern, dies erschien uns sinnvoll, da vor allem Rabatte aktuell noch nicht online einpflegbar sind und so der Nutzer erfahren kann, ob sich eine Vorort-Buchung als lohnend herausstellt.

Im Sequenzdiagramm möchte der Nutzer als nächstes zusätzliche Items seinem Warenkorb hinzufügen, die den Kinobesuch versüßen. Dafür wurde in unserer Anwendung die Snack Kategorie implementiert, in welche der Nutzer zusammen mit dem Ticket direkt für Popcorn, Getränke und andere Verpflegung bezahlen kann, um diese vor Ort nur noch abholen zu müssen. Auf die Auswahl der Tickets online und das Hinzufügen möglicher Extras folgt für den Nutzer nun der Weg zur Reservierung und Bezahlung der entsprechenden Tickets. Dafür wählt der Nutzer im Sequenzdiagramm die Bezahlen-Option (pay) und auch unsere Website ermöglicht dies dem Besucher über einen einfachen Knopfdruck. Durch Eingabe der Nutzerdaten und das damit verbundene einwählen in ein Nutzerkonto geht der Actor sowohl theoretisch als auch praktisch zum nächsten Schritt in diesem Abwicklungsprozess über. Die Verifizierung der Nutzerdaten ist an dieser Stelle notwendig, da bereits hinterlegte Daten wie Zahlungsmittel und ähnliches, dem Nutzer erst zur Verfügung stehen, wenn er sich entsprechend angemeldet hat. Der Nutzer möchte am Ende seiner Buchung eine Quittung, sowie ein reales Ticket in Form eines QR-Codes erhalten. Beides geschieht bei unserer Anwendung automatisch durch das Abschließen des Bezahlvorganges. Dieser kann im finalen

Produkt sowohl durch PayPal als auch unter Eingabe herkömmlicher Bankinformationen geschehen, wodurch die Anwendung die aktuell herrschenden Vorlieben der meisten Kunden abdecken dürfte. Nach Bestätigung der Zahlung erhält der Nutzer, auf die von ihm in seinem Benutzerkonto hinterlegte E-Mail eine Nachricht, die die beiden Dokumente enthält. Hier lag der Fokus neben der Erfüllung des Sequenzdiagramms einmal auf Sicherheit und zum anderen auf Beständigkeit der Daten. Der Nutzer kann so weder sein Ticket auf einem öffentlichen Computer “vergessen” noch kann er es durch das Schließen eines Fensters verlieren. Im Anschluss kehrt der Nutzer sowohl im Sequenzdiagramm als auch in der Realität in die Ausgangssituation zurück nun im Besitz von den entsprechenden Tickets und kann den Prozess jederzeit von neuem beginnen. Die Ausgangssituation wieder herzustellen ist für die dauerhafte Nutzbarkeit dieser Website unabdingbar, da kaum jemand nur ein einziges Mal in Kino geht.

Klassendiagramm

An dieser Stelle sollen besonders Veränderungen zum ursprünglich vorgestellten Klassendiagramm aus dem zweiten Semester [siehe Anhang 4] betrachtet werden.

Insgesamt konnten während der Erstellung des eigentlichen Projektes viele Klassen, die im ursprünglichen Diagramm vorhanden waren, weggelassen werden, da sie sich bei genauerer Betrachtung als überflüssig herausgestellt haben. Von ursprünglich 30 Klassen sind nur noch 19 vorhanden, wobei einige der aktuell existierenden zu Beginn nicht geplant waren. Eine Kürzung konnte in der Kategorie der Bistroartikel durchgeführt werden, welche im Original sowohl aus einer Menüübersicht als auch den einzelnen Artikeln selbst bestand. Im finalen Produkt reicht eine Klasse Snack welche in Form einer Liste der Klasse Booking mitgegeben wird. Dies stellte sich insgesamt als unkomplizierter, für den Aufbau übersichtlicher und daher wünschenswerter in der Umsetzung heraus.

Ebenfalls eingekürzt werden konnte im Bereich der Endprodukte einer Buchung, die der Nutzer erhalten sollte. Im Vorschlag aus dem zweiten Semester wurden sowohl ein QR-Code als auch eine Quittung als separate Klassen in kompositiver Beziehung zum Ticket erstellt. In der Realumsetzung konnte beides in Form eines Attributes QR-Code in einer Buchung eingefügt werden. Auch an dieser Stelle vereinfacht sich dadurch die Handhabung in der Datenbank und der Weitergabe der Daten an den Nutzer der Anwendung.

Stark verändert wurde auch der Aufbau und Ablauf hinter dem Erstellen und Nutzen von anpassbaren Kinosaalplänen. Dabei ist die Strukturierung der Beziehungen zwischen dem Kinosaal, den Sitzen desselbigen und den Shows, die diesen Nutzen am meisten beeinflusst worden. In der ursprünglichen Modellierung hat jeder Kinosaal über einen separaten Sitzplan verfügt, dieser wurde im finalen Produkt ersetzt durch eine Liste von sogenannten Seat-Blue-Print Objekten. Diese Seat-Blue-Print Objekte existieren daher für jeden erstellten Kinosaal

(auch CinemaRoom). Wenn nun eine Show für einen Film angelegt wird, wird dort ein entsprechender Kinosaal ausgewählt. Dadurch wird der Show auch die Liste mit den Seat-Blue-Print Objekten übergeben, woraus an dieser Stelle der tatsächliche Sitzplan für das Frontend generiert wird. Im gleichen Zuge hat sich die Art wie Sitze voneinander unterschieden werden gewandelt. Statt pro Kategorie eine einzelne Klasse anzulegen konnte der Prozess in Form eines Attributes der Klasse Seat vereinfacht werden. In Folge dieser Umstrukturierung sind von den zuerst benötigten acht verschiedenen Klassen nur noch drei essenziell, um diese anpassbaren Kinosäle zu realisieren.

Eine weitere Veränderung ist bei der Hinterlegung der Preise für entsprechende Produkte zu finden. Diese standen ursprünglich in einer Assoziativen Verbindung zu Vorstellungen, wurden aber im überarbeiteten Modell in Verbindung mit dem Ticket, sowie dem SeatBluePrint gebracht. Dies geschah da der Preis für ein Ticket sich aus diesen BluePrints generiert und dann dem Ticket mitgegeben werden muss.

Zuletzt konnten die Beziehungen rund um den Benutzer vereinfacht werden, da davon abgesehen wurde einen Gastzugang zum jetzigen Zeitpunkt zu realisieren. Stattdessen ist die Anmeldung Voraussetzung zum Buchen eines Tickets. Dadurch ist auch hier das Klassendiagramm dünner geworden.

## 2.4 Daten

Die zu speichernden Daten sind differenziert in die folgenden Bereiche:

- Vorstellungsdaten, die Informationen über den Film, den Start- und Endzeitpunkt, die Tickets, dem Kino sowie Kinosaal und den getätigten Buchungen enthalten.
- Filmdaten, welche Informationen über den Name, die Dauer, die Bewertungen und Rezensionen sowie nähere Informationen zum Film enthalten. Für unseren Test haben wir einige wenige Filmdaten mit ca. 10 – 20 Filmen in der Datenbank hinterlegt.
- Kinodaten, die Informationen über die Räume, die Sitzpläne und dem Standort enthalten. Hierfür besitzen wir nur wenige Datensätze, da von unserer Kinoreihe nur wenige Kinos in ein paar Orten existieren.
- Kundendaten, welche langfristig gespeichert werden, bis das entsprechende Kundenkonto entfernt wird. Diese Daten sollten eine Kundenummer, den Name des Kunden, die gekauften Tickets und die Kreditkarteninformationen enthalten.

Ein wie im Pflichtenheften erwähntes hohes Datenvolumen ist in der Speicherung der Daten auf Grund von Redundanzfreiheit nicht eingetreten. Sowohl der Speicher als auch die Datenbank sind dynamisch erweiterbar sein, sodass es auf Dauer zu keinem Datenverlust kommt. Da es sich hierbei nur um ein Testsystem handelt und wir natürlich kein reales eigenes Kino eröffnen, entsprechen die geplanten Datensätze des Pflichtenhefts natürlich nicht der

Realität. Zum Testen wurden nur ein Bruchteil der vorher eingeplanten Daten hinterlegt, jedoch sind mehr Datensätze durchaus möglich.

Eine automatische Löschung von Daten nach einer gewissen Zeit wurde nicht implementiert. Das heißt, dass zurzeit alle in der Datenbank gesammelten bzw. eingetragene Datensätze für immer dort existieren und die Datenbank somit über die Zeit immer weiterwachsen wird. Dies liegt vor allem daran, dass es sich hierbei um kein „echtes“ Kinosystem handelt.

## 2.5 User Interface

Ziel war es, wie bereits in den Muss-Kriterien erläutert, ein User Interface zu entwickeln, welches sich gut und intuitiv nutzen lässt, um damit eine hohe Usability zu erzielen. Zur Entwicklung entschieden wir uns dabei für „React“ und verwendeten die React UI Library von „Mui“ als Basis für unser Design. Um eine hohe Usability zu erzielen, orientierten wir uns am Grundsatz der Usability:

Produkte oder Anwendungen weisen eine hohe Usability für den Nutzer auf, wenn sie von den vorgesehenen Benutzern einfach erlernt und effizient verwendet werden können und diese damit ihre beabsichtigten Ziele und Aufgaben zufriedenstellen ausführen können.
Kriterien, die sich daraus für unser System ergaben und für uns wegweisend sein sollten, waren:

1. Leichte Erlernbarkeit

Die leichte Erlernbarkeit der Anwendung (auch Intuition) ist insofern wichtig, da sie zu einer höheren Conversion Rate (Kauf) führt. Der Benutzer soll sich nicht „verlaufen“ und dabei frustriert werden. Wir versuchten also, den Prozess vom Besuch unserer Website bis hin zum Kauf eines Tickets intuitiv und mit wenigen Klicks zu ermöglichen, um eine hohe Conversion Rate zu erzielen.

2. Hohe Effizienz

Die Effizienz sollte sich für den Benutzer vor allem in dem einfachen und schnellen Buchungsprozess eines Tickets wiederfinden. Durch den Chatbot sollen Fragen des Nutzers möglichst präzise beantwortet werden und Vorschläge zu Vorstellungen und Filmen gemacht werden. Aber vor allem war uns auch die Effizienz der Mitarbeiter wichtig. Unter der Admin Seite können diese einfach Filme, Sitzpläne, Vorstellungen und diverse andere Strukturen pflegen und für das System anlegen.

3. Gute Einprägsamkeit

Einprägsamkeit ist insofern wichtig, dass bei erneutem Besuch der Anwendung der Benutzer sich schneller und einfacherer zurechtfinden. Um dies zu gewährleisten, setzten wir darauf, dass der Buchungsprozess für egal welchen Film identisch ist, um Konsistenz zu bewahren.

4. Wenige / nicht schwerwiegende Fehler

Fehler frustrieren Anwender. Anwendungsfehler wie Bugs haben wir durch eigenständiges Testen identifiziert und behoben. Aber auch Fehler durch den Benutzer können diesen Frustrieren (Bsp.: Findet sein Ziel nicht). Durch den einfachen und immer gleichbleibenden Buchungsprozess versuchen wir den Benutzer gezielt durch die Anwendung zu leiten, sodass er wenige bis keine Fehler (verklicken oder etwas falsch verstehen) beim Buchungsprozess erlebt.

5. Zufriedenheit während und nach der Nutzung

Die Zufriedenheit versuchen wir durch den bereits beschriebenen einfach gehalten Buchungsprozess zu ermöglichen, ein ansprechendes UI und durch schnelle Antwortzeiten des Servers sowie einer hohen Uptime.

6. Kurze Ladezeiten

Lange Ladezeiten frustrieren Benutzer, weshalb wir versuchten, diese zu verringern und die Anwendung zu optimieren. So wenig Calls an das Backend wie möglich, aber so viel wie nötig. Als Hosting für unsere API setzen wir dabei auf Azure. Azure ermöglicht uns, schnelle API-Calls an das Backend durchzuführen und so Ladezeiten sehr gering zu halten. Fürs Frontend Hosting setzen wir auf den Gratis-Hosting-Dienst „Vercel“, welcher unseren Anforderungen bisher gerecht wird.

Um zu testen, inwiefern und mit welcher Qualität wir diese Kriterien erfüllen, genügt eine eigene Einschätzung nicht, dient aber als Ansatz. Zukünftig könnte man schwächen und Optimierungsmöglichkeiten durch Usability Tests mit Endbenutzern identifizieren und so das System iterativ optimieren.

## 2.6 Qualität

### 2.6.1 Produktqualität

Ein Fokus bei der Entwicklung der Applikation zum Buchen von Kinotickets war die Qualität des Produktes selber, wie sie ein Endnutzer wahrnehmen wird. Der Endnutzer will eine einfache und intuitive Webseite besuchen, die visuell und auch von ihren Funktionen her viel zu bieten hat. Das Design der Webseite wurde mit Bedacht gewählt und angepasst, um einen objektiv schönen Anblick zu ermöglichen. Das Design ist auf der ganzen Seite konsistent und

angenehm anzuschauen. Durch hohe Kontraste und Akzente bei den einzelnen Elementen auf der Seite ist es auch sinneseingeschränkten Benutzern der Webseite gestattet, diese zu ihrem vollen Ausmaß zu nutzen. Eine komplett barrierefreie Version ist für die Zukunft geplant, sodass dann auch zum Beispiel die Kompatibilität mit Hilfsmitteln wie Screenreadern optimiert wird. Dadurch, dass Menschen, die visuell eingeschränkt sind, nicht die Hauptzielgruppe eines Kinos sind, wurde dem Aspekt der Qualität bei der Entwicklung eine geringere Priorität gegeben.

### 2.6.2 Technische Qualität

CI / CD

Die Applikation, sowie das Frontend als auch das Backend und alle weiteren dazugehörigen Services wurden unter den Konzepten der Continuous Integration (CI) und Continous Deployment (CD) entwickelt. Um CI und CD zu nutzen, wurde GitHub als Tool für die interne Kollaboration genutzt. Die

CI besteht daraus, dass jedes Mal, wenn ein Feature oder eine größere Änderung in den momentanen Basis-Code geschrieben wird, getestet wird, ob das Programm immer noch alle Funktionen erfüllt, alle verfügbaren Tests erfolgreich sind und ob sich der Code überhaupt aufbauen kann. Passiert dies nicht, muss das Feature noch überarbeitet werden und ist nicht bereit in den Haupt-Code aufgenommen zu werden. Der ganze CI Prozess ist in yml-Dateien hinterlegt und wird bei bestimmten Aktionen auf GitHub automatisch ausgeführt. Eine technisch qualitativ hohe Applikation zeugt von gutem Code und Sicherheit und Stabilität der Applikation, was alles mit den Tests und Checks von CI getestet und verifiziert wird. Nur sauberer, guter und funktionierender Code wird in der Applikation und auf der Webseite genutzt.

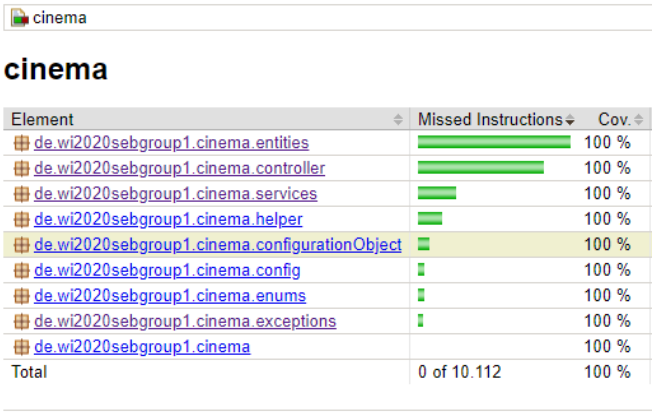
CD bedeutet, dass funktionierende Versionen der Applikation und der Webseite immer wieder hochgeladen und über eine URL zur Verfügung gestellt werden. Wenn alle CI-Tests und -Checks erfolgreich waren und ein Feature oder Funktionalitäten nun zur Verfügung stehen, wird automatisch von GitHub ein Deployment angestoßen. Dies besteht im Normalfall daraus, die Applikation zu bauen und auf einer bereitgestellten Plattform zu deployen. Auch dies trägt zur Qualität bei, da durch das ständige Deployen alle Zugang zu der Applikation haben und sie testen können, damit Bugs und Fehler gefunden werden können, die nicht in dem finalen Release sein sollten.

Code Tests

Ein wichtiger Bestandteil von qualitativ hochwertigen Applikationen ist ein gut getesteter Code. Jede Funktion einer Applikation muss an sich funktionieren und darf auch nicht durch neue Änderungen oder anderen Funktionen in ihrer Essenz blockiert oder behindert werden. Um

durchgehend im Rahmen von Continuous Integration sicherzustellen, dass jede Funktion im Code auch nach Änderungen noch weiterhin so funktioniert, wie sie es soll, werden Tests geschrieben. Diese Tests testen einmal alle Funktionen und Methoden an und für sich, ob sie – unabhängig von allen äußeren Einflüssen – im Grundsatz funktionieren. Zusätzlich zu diesen sogenannten Unit Tests werden Integrationstests ausgeführt, die das Zusammenspiel von verschiedenen Funktionen und Methoden untersuchen. Bei allen Tests wird eine Erwartung an die Funktion bzw. an das Zusammenspiel von Funktionen definiert, die erfüllt werden muss, damit die Applikation alle CI-Checks absolviert. Schlägt ein Test fehl kann das ein Anzeichen sein, dass die neu eingepflegte Funktion den Code an anderer Stelle einschränkt, was nicht gewollt und auch nicht gewünscht ist.

Die Effektivität und das Ausmaß der Tests kann durch die Code Coverage dargestellt werden. Die Code Coverage gibt an, wie viele Zeilen des Codes und wie viele verschiedene Abarbeitungswege im Code getestet werden. Eine hohe Code Coverage sagt an, dass die Chance Code von schlechter technischer Qualität herzustellen, deutlich geringer ist, als es bei einer niedrigen Code Coverage der Fall ist. Somit geht mit einer höheren Code Coverage auch eine höhere Qualität einher. Bei Applikation zum Buchen von Kinotickets wurde eine 100% Code Coverage angestrebt, die jede Zeile des Codes an jeder Stelle testet. Dies garantiert zwar noch keine absolute Ausfallsicherheit von Funktionen, ist aber wie bereits beschrieben ein sehr wichtiger Teil einer qualitativ hochwertigen Applikation.


cinema

**cinema**

| Element | Missed Instructions | Cov. |
|---|---|---|
| de.wi2020sebgroup1.cinema.entities | | 100 % |
| de.wi2020sebgroup1.cinema.controller | | 100 % |
| de.wi2020sebgroup1.cinema.services | | 100 % |
| de.wi2020sebgroup1.cinema.helper | | 100 % |
| de.wi2020sebgroup1.cinema.configurationObject | | 100 % |
| de.wi2020sebgroup1.cinema.config | | 100 % |
| de.wi2020sebgroup1.cinema.enums | | 100 % |
| de.wi2020sebgroup1.cinema.exceptions | | 100 % |
| de.wi2020sebgroup1.cinema | | 100 % |
| Total | 0 of 10.112 | 100 % |

Code Reviews

Ein weiterer Teil, der zu einer hohen Qualität beiträgt, sind Code Reviews, die durchgeführt werden. Auch wenn durch CI und CD alles automatisiert ist und diese Automatisierungen Sicherheiten geben, dass der Code und dadurch auch das Produkt eine hohe Qualität hat, ist es immer noch wichtig, mit eigenen Augen den Code anzuschauen. Auch wenn alle Checks und Tests im Rahmen von CI bestanden sind, muss jedes Mal noch eine nicht an der Entwicklung der Funktion beteiligten Person den Code selber anschauen und bewerten. Fallen Unstimmigkeiten auf oder ist der Code an einigen Stellen zu kompliziert, ressourceneinnehmend oder nicht menschenlesbar, werden Änderungen beantragt. Somit werden nur Features und Funktionen in den Haupt-Code integriert, die nicht nur technisch gut, sondern auch optisch und durch menschliche Augen als qualitativ hochwertig abgesegnet sind.

## 2.7 Ergänzungen

TheoTicket

TheoTicket ist ein vom Backend unabhängiger Microservice, der die Funktionalitäten eines Chatbots besitzt. TheoTicket kann vom Endnutzer benutzt werden, um schnell Informationen zu finden, die auf der Webseite einige Klicks benötigen. Vor allem als Suchassistent ist TheoTicket gut zu gebrauchen, da er auf verschiedene Arten von Anfragen verschiedene Antworten hat. Dabei ist es egal, ob die Nachricht Rechtschreibfehler enthält oder umschrieben formuliert werden. Der Chatbot analysiert die Anfrage, gleicht sie mit der im Backend benutzten Datenbank ab und formuliert auf dessen Basis eine Antwort, die so gut wie möglich die Anfrage beantwortet. TheoTicket lernt nicht mit, sondern ist als statische Suchhilfe geplant und implementiert worden. Aus dem Grund sind keine alltäglichen Unterhaltungen möglich.

# 3. Stärken, Risiken und Kompromisse

Insgesamt konnten wir die geforderte Leistung ein funktionsfähiges Interface zu erstellen, über das ein Ticket für eine Vorstellung im Kino gebucht werden kann, erbringe, wodurch wir das grundlegendste Ziel dieses Projektes fertigstellen konnten. Zum Schluss unserer Bearbeitungszeit wurde nun rückblickend betrachtet, welche der von uns zusätzlich gesetzten Ziele wir mit besonderer Bravour absolvieren konnten, an welchen Stellen wir aktuell noch Probleme in der Anwendung sehen und zu welchen Zeitpunkten in der Entwicklung wir zwar ein Ergebnis erzielt haben, dieses aber von den ursprünglichen Plänen abweichen musste, wir also Kompromisse eingegangen sind.

Einige der für uns erfreulichen Ergebnisse befinden sich bereits auf einer strukturellen Ebene um das Projekt herum und haben uns die Arbeit an dem Selbigen stark vereinfacht. Dazu gehört zum einen unsere Arbeit über das gesamte Projekt verteilt mit JIRA, die uns ermöglicht hat ein gutes Zeitmanagement beizubehalten und gleichzeitig nie den Überblick über noch fehlende Aufgaben zu verlieren. Dadurch haben wir die meisten unserer zu Anfang gesetzten Ziele auf Programmier-Ebene im Auge behalten können und sie schlussendlich in unser Projekt integrieren können. Auch in diese Kategorie der Stärken unseres Projektes fällt die Arbeit über GitHub, die uns die starke Parallelisierung von Arbeitsschritten, aufgeteilt auf verschiedene Gruppenmitglieder, ermöglicht hat. Da bereits zu Beginn der Arbeitsphase eine wohldurchdachte Struktur mit verschiedenen Repositories für verschiedene Teile des Projektes erstellt wurde, konnten wir durchgehend ohne größere Konflikte innerhalb der Codebase arbeiten und unsere Zeit effektiv für andere Aufgaben als das Beheben von Konflikten nutzen. Insgesamt können also unsere allgemeine Strukturierung und zielorientiertes Handeln als Stärke unsere Projektes gesehen werden.

Ein weiterer positiver Aspekt kann bei uns im Backend gefunden werden, welches aufgrund seiner Umsetzung mit Java Spring Boot auch für die Zukunft gut nutzbar wäre, da es dem Nutzer eine Hochskalierung auf einen größeren Umfang an Daten einfach ermöglicht. Dadurch ist unsere Anwendung nicht nur auf einige Testfälle beschränkt, sondern könnte auch in realen Situationen genutzt werden. Ein besonders hervorzuhebender Teil dieser Umsetzung, der erfolgreich implementiert werden konnte sind die durch Spring Boot bereitgestellten JPA Repositories, welche die Arbeit mit Daten in Bezug auf APIs stark vereinfachen und unterstützen. Durch ihre Nutzung konnten viele bereits bereitgestellte Methoden mit in das Projekt aufgenommen werden, um dieses zu unterstützen und das Bereitstellen der Daten für das Frontend zu optimieren. Ergänzt wird die Stärke des Backends durch die Test-Coverage, welche sämtliche implementierte Funktionen desselbigen abdeckt und somit die vorgegebenen Werte übertrifft und als Vorteil dieser Anwendung betrachtet werden kann.

Als nächstes werden einige Ergebnisse konzeptioneller Natur unseres Projektes präsentiert die außerhalb des grundsätzlichen Rahmens liegen. Neben dem normalen Buchungsprozess stehen in unserer Anwendung dem Nutzer noch weitere Ansichten, in Anlehnung an ein reguläres Kino zur Verfügung, die eine realitätsnahe Erfahrung besser simulieren als die ursprünglich geforderte Perspektive. Dazu gehören eine separate Events und News Page des Kinos, die Übersicht über die Kosten für einen Kinobesuch, sowie eine vollständige Bereitstellung aller gesetzlich relevanten Informationen, die im Rahmen einer Website aktuell gefordert sind. Letzteres deckt ein Impressum, die in Deutschland geforderten AGBs und verpflichtenden Cookies einer öffentlich verfügbaren Seite im Internet ab. Dadurch nimmt die Website insgesamt einen seriöseren Gesamteindruck an, der von uns positiv wahrgenommen wird.

Auch einige funktionale Features über den Erwartungen des ursprünglichen Projektes sind in diese Anwendung mit eingeflossen und können daher als Stärken betrachtet werden. Dazu gehört zum einen die Möglichkeit des Nutzers sich ein eigenes Konto auf der Website anzulegen über welches die Tickets gebucht werden können. Dies ist durch die Bereitstellung entsprechender Funktionen sowohl des Frontends als auch des Backends umgesetzt worden. Neben normalen Tätigkeiten, wie dem Hinterlegen von Kundendaten kann der Kunde zusätzlich Reviews zu Filmen verfassen, die sich andere Kunden angucken können. Ein weiterer positiver Aspekt der Anwendung ist der Chatbot der dem Nutzer jederzeit innerhalb der Anwendung zu Verfügung steht und einfache Fragen beantworten kann, wie beispielsweise die Informationen zu einem bestimmten Film bereitzustellen.

Des Weiteren hervorzuheben ist sind die Funktionalitäten auf Seiten der Administration unsere Anwendung. Zum einen besteht die Möglichkeit Filme mit entsprechenden Vorstellungen anzulegen, zu bearbeiten und wieder zu löschen. All dies geschieht in einer separaten Ansicht, die für den normalen Nutzer verborgen bleibt. Zum anderen können individuelle Saalpläne erstellt werden spezifisch angepasst auf die gegebenen Bedingungen eines Kinos. Dabei können Neben Anzahl von Reihen und Sitzen pro Reihe auch verschiedene Preiskategorien angelegt werden die entsprechend für den Benutzer im Frontend visualisiert werden. Dieses Feature macht unsere Anwendung besonders realistisch, da Sitzpläne für Kinos keine starren Arrays sind und über dieses Konzept realitätsnah für jede Vorstellung verändert werden können und dies direkt aus der Anwendung, ohne separat auf die Datenbank zugreifen zu müssen.

Unsere Website stellt zu dem E-Mail-Benachrichtigungen zur Verfügung, um den Nutzer über seine Tätigkeiten auf der Website zu informieren. Dazu gehört z.B. die Versendung eines gekauftes Tickets inklusive QR-Code. Auch dieses Feature erhöht den Realitätsbezug unserer Anwendung ungemein. Zusammenfassend verfügt das Projekt über

eine Vielzahl von wohlüberlegten Add-Ons neben den Voraussetzungen, um ein einfaches Ticket zu buchen.

Trotz der Erfolge existieren auch einige Baustellen, die in der aktuellen Iteration noch nicht perfektioniert werden konnten. Diese stellen die Risiken unserer Anwendung dar. Besonders problematisch ist aktuell die Abspeicherung von Passwörtern die Zugang zu Nutzerkonten sowie Datenbankdaten ermöglichen. Da wir unsere Anwendung möglichst kostengünstig im Betrieb halten wollen haben wir keine Möglichkeit gefunden diese Daten anderweitig abzuspeichern, wodurch sie aktuell auf dem öffentlich zugänglichen Repository unseres Projektes zu finden sind. Diese Manko müsste für einen möglichen Go-Live zwingend behoben werden, da es sich zum einen um persönliche Nutzerdaten handelt, die dem Datenschutz unterliegen und zum anderen Daten, die die Integrität unserer Anwendung schützen sollen. Beide Formen der Daten können genutzt werden, um Schaden anzurichten, entweder für eine Privatperson, die auf unserer Plattform agiert oder für den Betreiber der Website, da durch die Datenbank Passwörter Änderungen an internen Tabellen durchgeführt werden könnten.

Neben den finalen Stärken und Schwächen/Risiken unserer Anwendung gab es auch einige Stellen auf dem Weg an denen unerwartete Hindernisse aufgetreten sind, die zu spontanen Planänderungen geführt haben. Ein solcher Kompromiss ist im Bereich des Testens zu finden. Dort wurden zum reibungslosen Ablauf des Programms und um unerwartete Abstürze zu vermeiden verschiedenste Exceptions eingebaut die zu großen Teilen von externen Quellen bereitgestellt wurden. Dadurch fehlt an einigen Stellen die Möglichkeit die Exceptions ordnungsgemäß mit Hilfe von Unit- oder Integration-Tests zu testen. Da wir an dieser Stelle die Wahl hatten die Exceptions nicht zu verwenden oder uns auf die angegebenen Funktionalität dieser zu verlassen, wurde für das wohl des Endergebnisses vorläufig die zweite Möglichkeit bevorzugt.

# 4. Fazit

Zusammenfassend ist zu sagen, dass das Projekt ein Erfolg ist und funktioniert. Die meisten aller Kriterien, auf welche sich innerhalb der Gruppe geeinigt wurde, konnten gut umgesetzt werden. Alle nicht implementierten Features sind entweder aus Zeit- oder anderen Gründen aus der Entwicklung entfernt worden. Zu Beginn des Projektes sprachen wir intern Ziele ab, wie z.B. die Weiterentwicklung der Programmierkenntnisse einzelner Teilnehmer. Auch diese persönlichen Ziele wurden im Laufe der Entwicklung erreicht. Abhängig von den Aufgaben im Team haben die Teilnehmer Programmiersprachen und Technologien wie React, TypeScript, Java Spring Boot und Datenbanktechnik erlernt und vertieft.

Während der Entwicklung hatten wir jedoch einige Rückschläge einzustecken. Dazu gehörte zum Beispiel die Umstellung auf einen anderen Host im Frontend, da durch Azure immer mehr Probleme auftraten. Durch solche Rückschläge und Stolpersteine waren wir aber in der Lage von diesen zu lernen und können sie nun für weitere Projekte nutzen. Insgesamt hat jeder im Team gelernt, wie es ist, ein Programmierprojekt von Planung bis zur Fertigstellung zu begleiten. Zusätzlich stand natürlich auch der Spaß im Hintergrund. Durch die dynamische Atmosphäre und viele Hilfestellungen, die sich die Teilnehmer gegenseitig gaben, war es möglich, dass alle Spaß an der Entwicklung und der Durchführung dieses Projektes hatten.

Insgesamt schlossen wir also am Montag unseren letzten Sprint mit einem für uns alle zufriedenstellenden Ergebnis ab. Auch, wenn nicht immer alles perfekt lief, so war das Projekt alles in allem sehr lehrreich für jeden von uns und festigte unsere Fähigkeit im Team zu arbeiten und effektiv, ehrgeizig und entschlossen auf ein großes Ziel gemeinsam hinzuarbeiten.

# 5. Ausblick und weitere Informationen

## 5.1 Erweiterungsmöglichkeiten

Es gibt aus unserem Team noch einige Ideen für Features, welche in der aktuellen Version noch nicht implementiert sind. Es handelt sich hierbei um Funktionen, die für den ersten Release noch nicht so wichtig sind, aber durchaus ihre Daseinsberechtigung für die Verbesserung der Website hätten. Unser Team denkt also, dass diese Funktionen in Zukunft noch implementiert werden könnten. Diese werden in diesem Kapitel vorgestellt.

Mehrsprachigkeit:

Momentan ist unsere Website nur auf Deutsch. Unsere Teammitglieder haben allerdings auch Kenntnisse in einigen anderen Sprachen, welche auf der Website angewendet werden könnten. Das könnte so aussehen, dass man auf der Homepage in einem Dropdown-Menü seine Sprache auswählen kann und diese dann auf sämtliche Seiten angewendet wird.

Umbuchung von Tickets:

Aktuell kann man Tickets nur kaufen und diese danach nicht mehr verändern. Möchte man sein Ticket nur verändern ist es etwas umständlicher erst zu stornieren und sich danach ein Neues zu kaufen. In Zukunft kann die Website so überarbeitet werden, dass man in seiner Ticket Ansicht dieses verändern kann, also zum Beispiel Sitzplätze tauschen oder eine andere Vorstellung auswählen, sofern die Plätze noch nicht alle vergeben sind.

Besondere Tickets:

Auf unserer Event Seite ist es möglich Informationen zu anstehenden Events zu bekommen. Tickets für diese kann man allerdings noch nicht erwerben. Denkbar wäre daher auf der Event Seite direkt einen Button für den Ticketkauf zu haben, welcher zu einer gesonderten Funktion für den Kauf spezieller Tickets führt.

Barrierefreiheit:

Um sehbehinderte Menschen bei der Nutzung der Website zu unterstützen ist die Integration eines Screenreaders denkbar. Bei der ersten Nutzung der Website würde dann gefragt werden, ob man das möchte. Geantwortet würde mit einem Tastendruck.

Bewertung von Reviews:

Im Moment kann man auf der Detailseite über einen Film die Bewertung des Films einsehen. In Zukunft könnte es, um diese Bewertung besser nachvollziehen zu können, Reviews geben, welche von anderen Nutzern bewertet werden können, ob diese hilfreich und angemessen

waren. So könnte man die Reviews danach sortieren und hätte garantiert, dass man objektive Reviews zu dem Film sieht.

Integration von weiteren Finanzdienstleistern:

Abgesehen von den bereits integrierten Funktionen zur Zahlung eines Tickets könnten weitere verbreitete Finanzdienstleister wie Google Pay oder PayPal integriert werden, um auch hier die Benutzerfreundlichkeit zu erhöhen.

Weitere Benutzerrollen:

Aktuell gibt es zwei Benutzerrollen für Benutzer der Website: „User“, also einen ganz normalen Kunden, welcher die Website nicht gestalten kann und „Admin“, also einen Mitarbeiter des Kinos, welcher die Website verändern kann. In Zukunft könnte die Rolle des Administrators mehr differenziert werden, da nicht jeder Mitarbeiter des Kinos alle Berechtigungen für die Website benötigt. Denkbar wären Rollen für Programmgestalter, Eventgestalter und Verwaltungsmitarbeiter, welche weniger Rechte als die Rolle „Admin“ haben. Auch auf Seite der Kunden könnte mehr differenziert werden mit zum Beispiel einer separaten Rolle für Großkunden.

Verbesserung des Chatbots:

Der Chatbot ist eine gute Funktion, um die User Experience auf der Website zu verbessern, doch auch dieser kann immer noch weiter verbessert werden. So könnte er ansprechendere Formulierungen verwenden und noch mehr Funktionen haben, um dem Kunden zu helfen oder Arbeit abzunehmen, womit der Chatbot auch ein Faktor im Bereich Barrierefreiheit ist.

Verbesserung der E-Mails:

Auch die E-Mails, die bei Erwerb des Tickets an den Kunden versendet werden, könnten noch deutlich ansprechbarer gestaltet werden, sodass man mehr das Gefühl bekommt, ein Mensch hätte diese geschrieben.

Farbthemen:

Um generell die Personalisierung und somit die User Experience zu erhöhen ist das Einführen verschiedener Farbthemen denkbar. Diese würden auf der Homepage per Dropdown-Menü auswählbar sein. Auch in Bezug auf die vorher bereits erwähnte Barrierefreiheit hätte diese Funktion Vorteile in Form eines Farbthemas für Farbenblinde.

Reservierung mit vor Ort Zahlung:

Im Moment ist eine Sitzplatzreservierung nur möglich, wenn man auch direkt online das Ticket dazu kauft. Daher ist es denkbar abseits davon eine reine Reservierung zu implementieren. Das tatsächliche Ticket würde man dann erst vor Ort im Kino bekommen und bezahlen.

## 5.2 Weiterführende Quellen

In diesem Unterkapitel werden einige Links bereitgestellt werden, um sich über die einzelnen Funktionalitäten und verwendeten Werkezuge des Systems genauer informieren zu können.

Backend:

MariaDB: https://mariadb.org/

SpringBoot: https://spring.io/projects/spring-boot

Azure: https://azure.microsoft.com/de-de/

Frontend:

React: https://reactjs.org/

React-Query: https://react-query.tanstack.com/overview

React-Router: https://v5.reactrouter.com/

MUI: https://mui.com/

TypeScript: https://www.typescriptlang.org/

Vercel: https://vercel.com/

GitHub:

Frontend: https://github.com/DrBackmischung/Kino-Frontend

Backend: https://github.com/DrBackmischung/Kino-Backend

TheoTicket: https://github.com/DrBackmischung/Kino-TheoTicket

Dokumentation: https://github.com/DrBackmischung/Kino-Dokumentation

Email-Templates: https://github.com/DrBackmischung/Kino-Email

JIRA:

https://dhbw-ma-wwi2020seb.atlassian.net/jira/software/c/projects/KINO/boards/1

Azure:

Backend-Dev: http://kinovation.azurewebsites.net/

Frontend-Dev: https://kino-frontend.vercel.app/

# 6. Metadokument

Kapitelzuteilung:

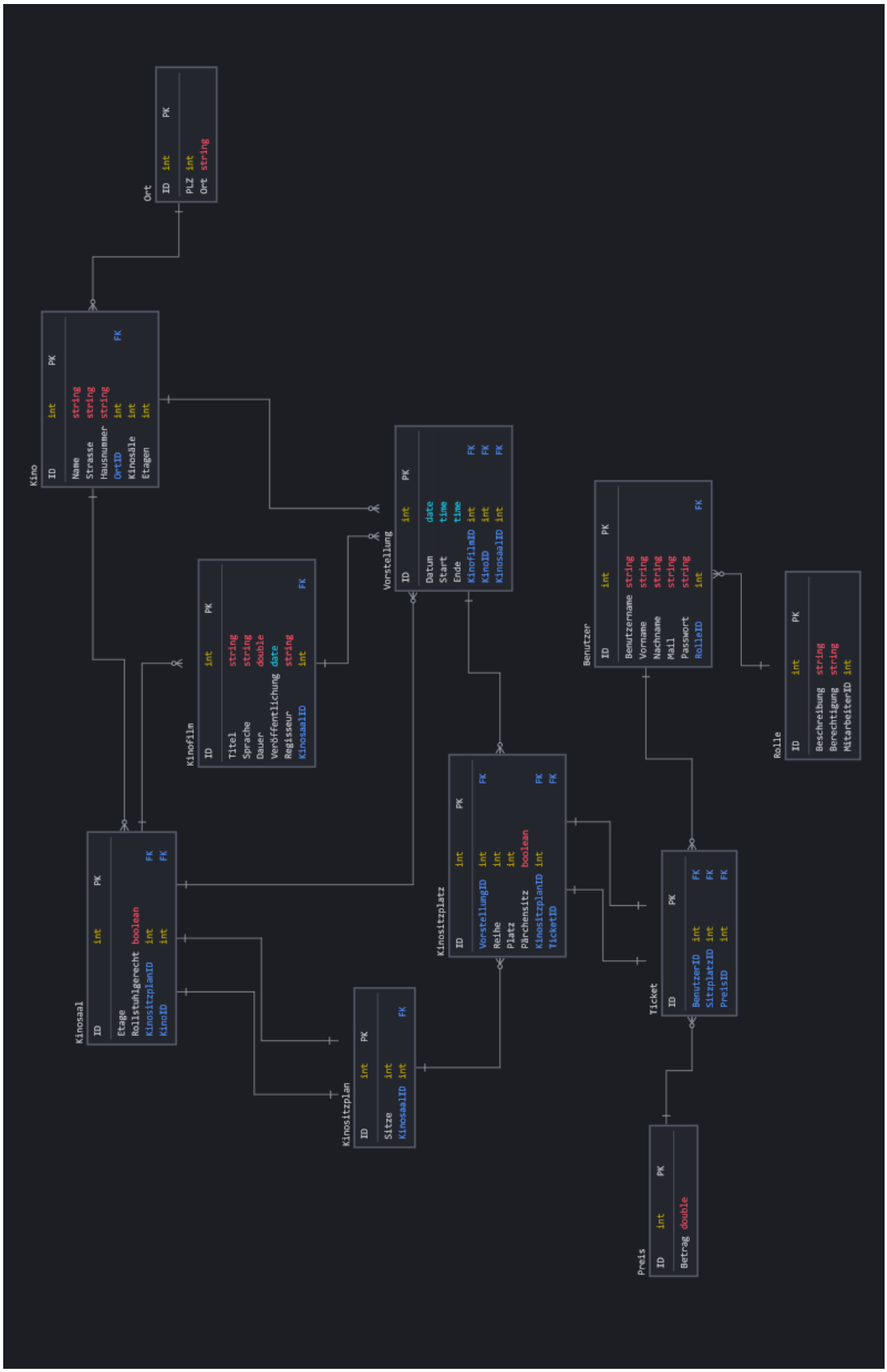
1. **Einleitung** (zusammen)
2. **Umsetzung des Projektes**
    - 2.1. Zielbestimmungen (Marcel Mildenberger)
    - 2.2. Einsatz (Babett Müller)
    - 2.3. Umgebung
        - 2.3.1. Backend (Jost-Tomke Müller)
        - 2.3.2. Frontend (Anna Khristolyubova)
        - 2.3.3. Hosting (Jost-Tomke Müller, Mathis Neunzig)
        - 2.3.4. Architekturmodelle und -diagramme
            - 2.3.4.1. Entity-Relationship-Modell (Babett Müller)
            - 2.3.4.2. Sequenzdiagramm (Nathalie Möck)
            - 2.3.4.3. Klassendiagramm (Nathalie Möck)
    - 2.4. Daten (Babett Müller)
    - 2.5. User Interface (Marcel Mildenberger)
    - 2.6. Qualität (Mathis Neunzig)
    - 2.7. Ergänzungen (Mathis Neunzig)
3. **Stärken, Risiken und Kompromisse** (Nathalie Möck)
4. **Fazit** (zusammen)
5. **Ausblick und weitere Informationen**
    - 5.1. Erweiterungsmöglichkeiten (Frederick Orschiedt)
    - 5.2. Weiterführende Quellen (Babett Müller)

Anmerkung:

Unser Team hat einheitlich entschieden niemand zu bevorzugen oder „schlechter“ dastehen zu lassen. Wir haben als Team dieses System auf die Beine gestellt und die Arbeiten so verteilt, wie jeder sie in der Lage war zu bearbeiten. Problem wurden gemeinsam angegangen und gemeinschaftlich gelöst. Jeder hat seinen entsprechenden Teil zu diesem Projekt beigetragen und sollte es doch mal kleinere Unstimmigkeiten untereinander gegeben haben, so wurden diese offen angesprochen und geklärt. Daher verzichten wir auf eine entsprechende Angabe zum Anteil jedes Teammitglieds an diesem Projekt.

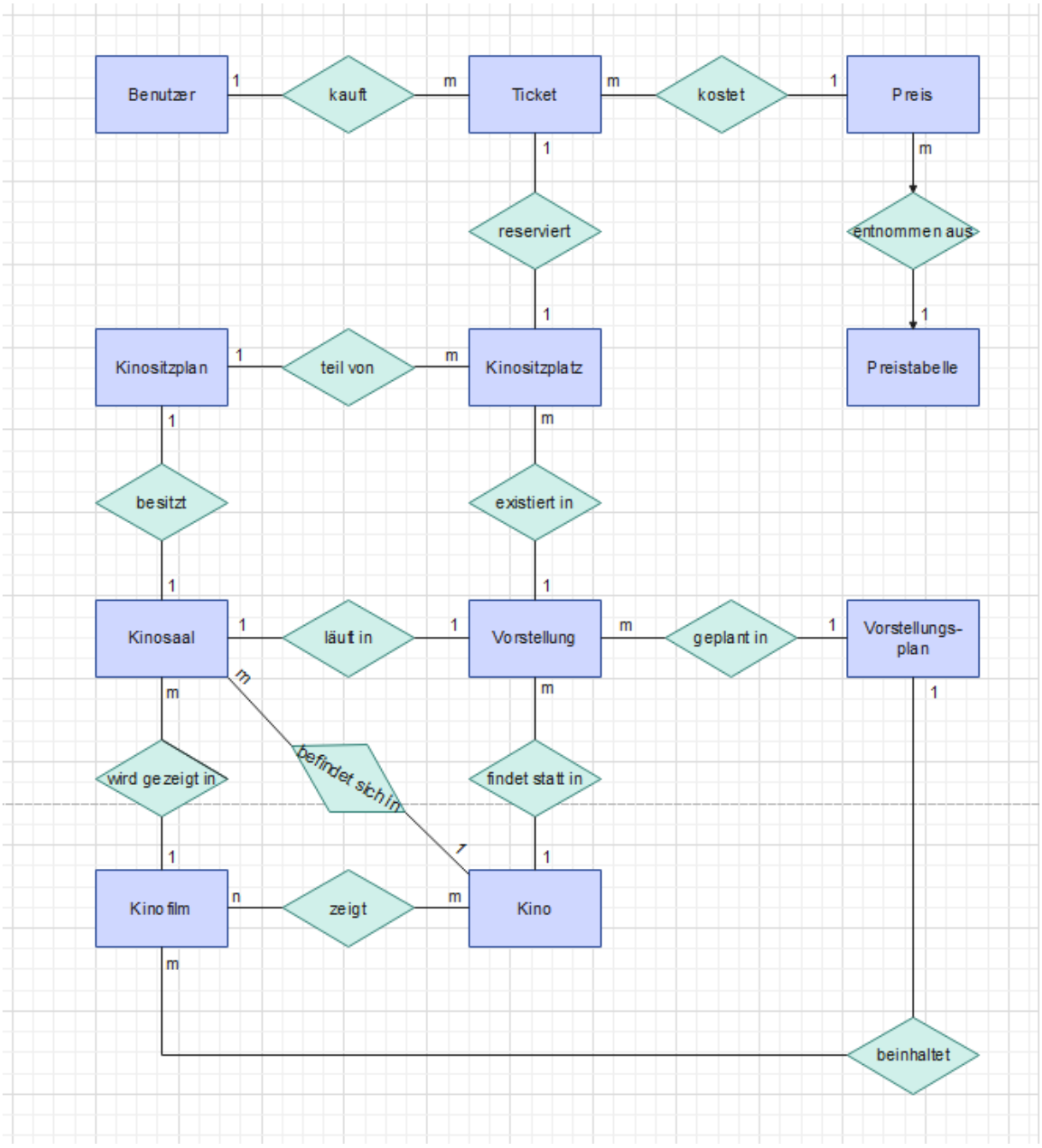
# 7. Anhang

## Anhang 1: Erster Entwurf des Datenbankmodells

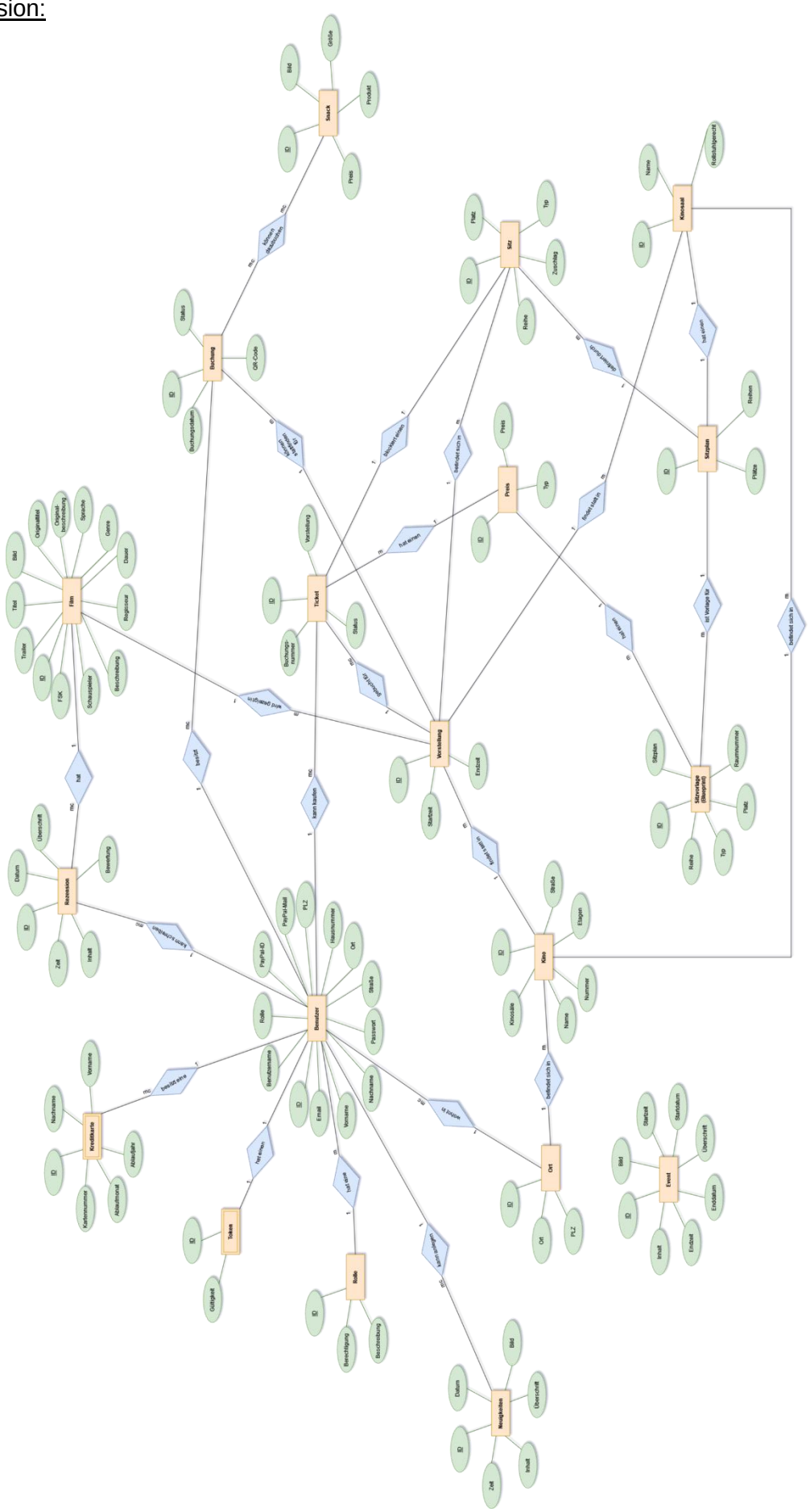


# Anhang 2: Entity-Relationship-Modell

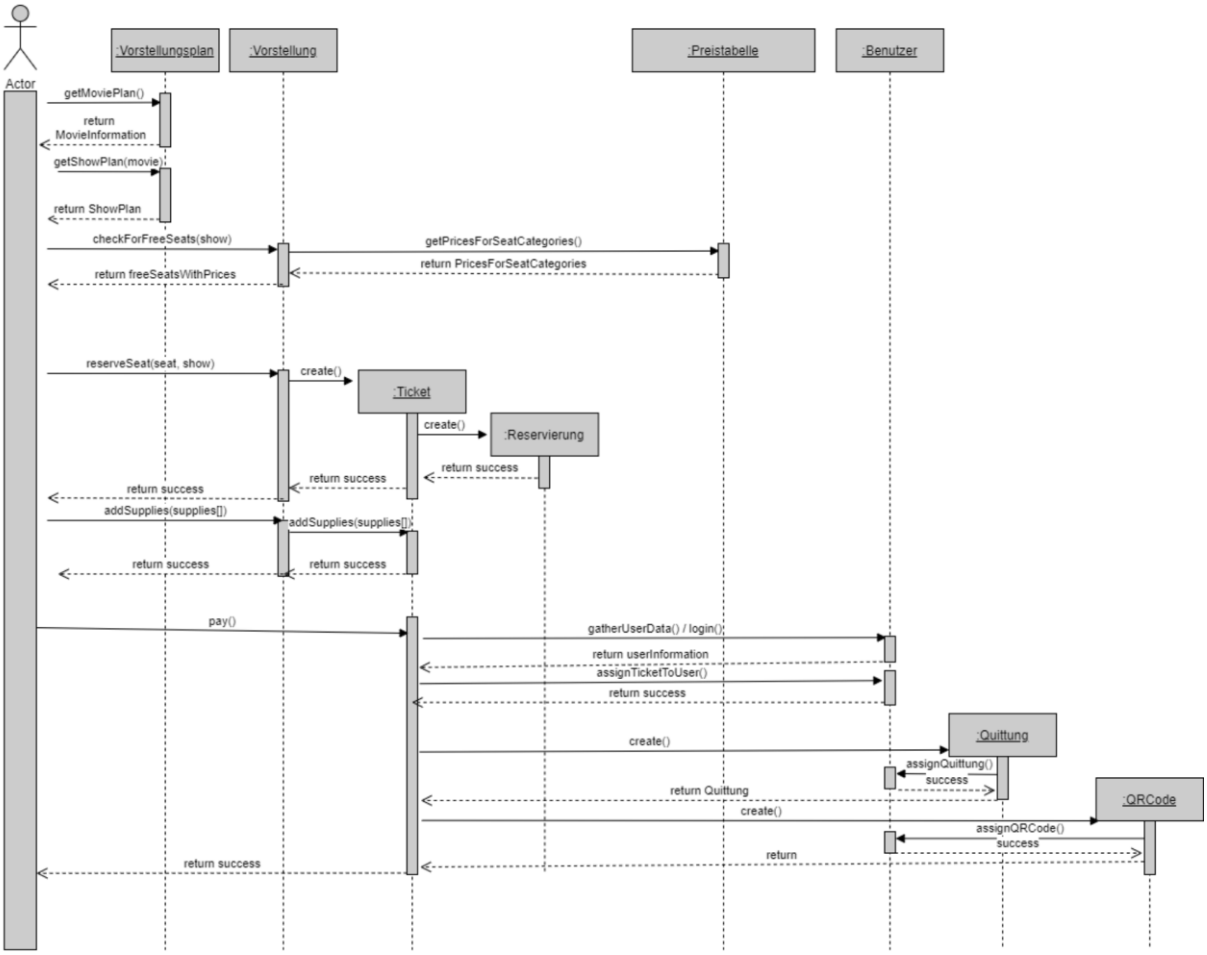
Erste Version



Endversion:

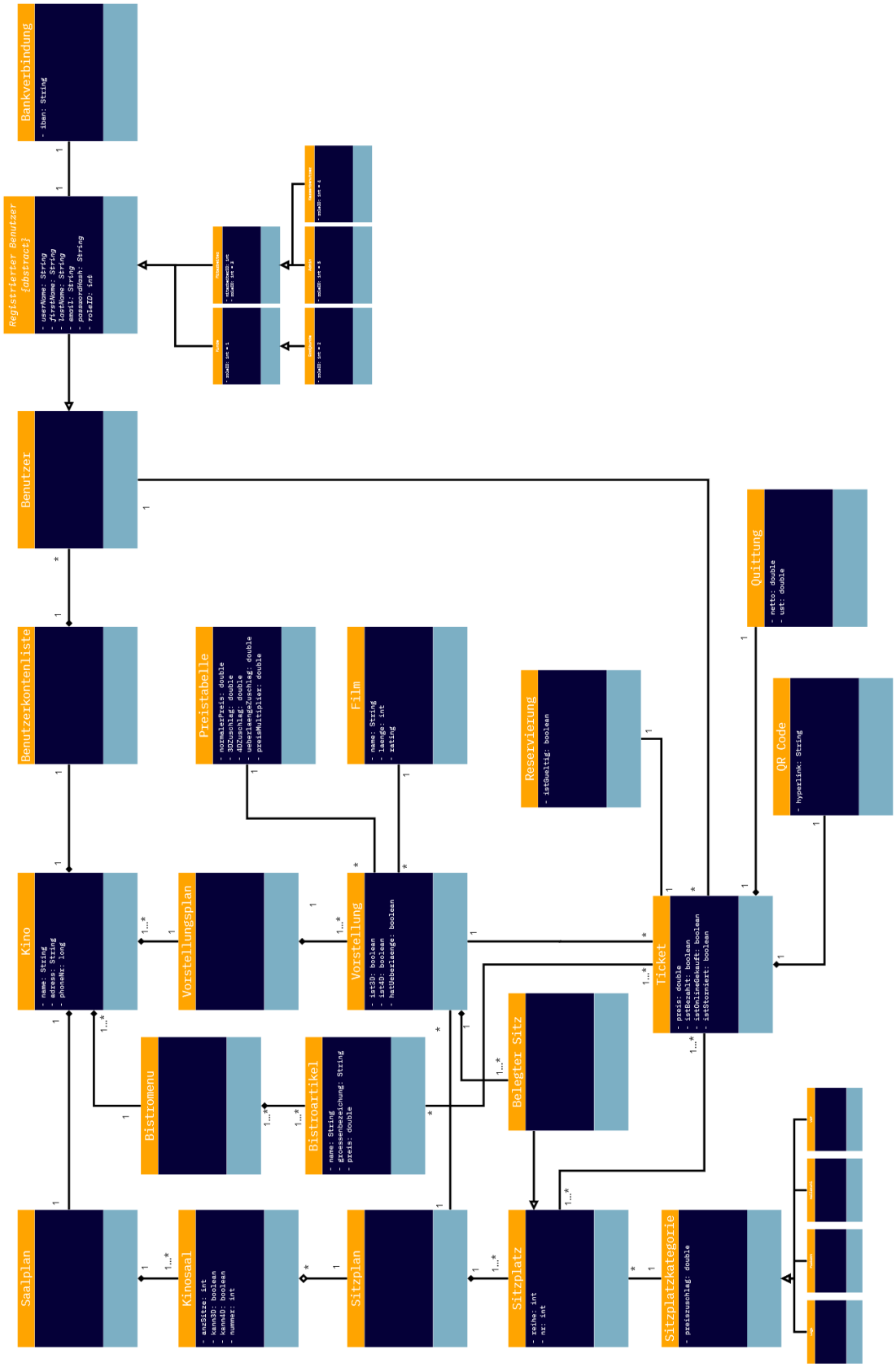


# Anhang 3: Sequenzdiagramm

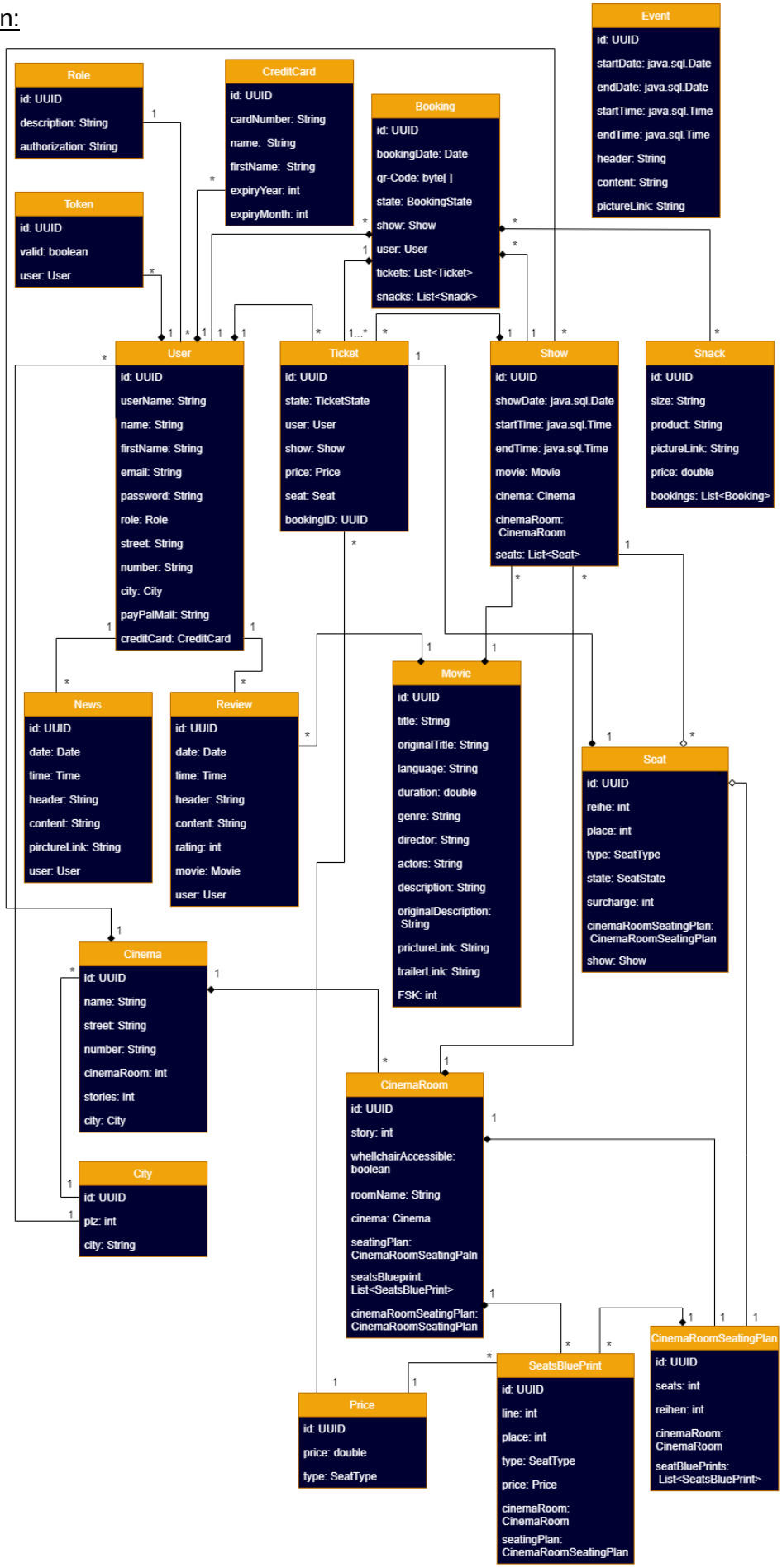


# Anhang 4: Klassendiagramm

Erste Version:



Endversion:


Role
id: UUID
description: String
authorization: String
CreditCard
id: UUID
cardNumber: String
name: String
firstName: String
expiryYear: int
expiryMonth: int
Booking
id: UUID
bookingDate: Date
qr-Code: byte[ ]
state: BookingState
show: Show
user: User
tickets: List<Ticket>
snacks: List<Snack>
Event
id: UUID
startDate: java.sql.Date
endDate: java.sql.Date
startTime: java.sql.Time
endTime: java.sql.Time
header: String
content: String
pictureLink: String
Token
id: UUID
valid: boolean
user: User
User
id: UUID
userName: String
name: String
firstName: String
email: String
password: String
role: Role
street: String
number: String
city: City
payPalMail: String
creditCard: CreditCard
Ticket
id: UUID
state: TicketState
user: User
show: Show
price: Price
seat: Seat
bookingID: UUID
Show
id: UUID
showDate: java.sql.Date
startTime: java.sql.Time
endTime: java.sql.Time
movie: Movie
cinema: Cinema
cinemaRoom: CinemaRoom
seats: List<Seat>
Snack
id: UUID
size: String
product: String
pictureLink: String
price: double
bookings: List<Booking>
News
id: UUID
date: Date
time: Time
header: String
content: String
pircture Link: String
user: User
Review
id: UUID
date: Date
time: Time
header: String
content: String
rating: int
movie: Movie
user: User
Movie
id: UUID
title: String
originalTitle: String
language: String
duration: double
genre: String
director: String
actors: String
description: String
originalDescription: String
prictureLink: String
trailerLink: String
FSK: int
Seat
id: UUID
reihe: int
place: int
type: SeatType
state: SeatState
surcharge: int
cinemaRoomSeatingPlan: CinemaRoomSeatingPlan
show: Show
Cinema
id: UUID
name: String
street: String
number: String
cinemaRoom: int
stories: int
city: City
City
id: UUID
plz: int
city: String
CinemaRoom
id: UUID
story: int
whellchairAccessible: boolean
roomName: String
cinema: Cinema
seatingPlan: CinemaRoomSeatingPaln
seatsBlueprint: List<SeatsBluePrint>
cinemaRoomSeatingPlan: CinemaRoomSeatingPlan
CinemaRoomSeatingPlan
id: UUID
seats: int
reihen: int
cinemaRoom: CinemaRoom
seatBluePrints: List<SeatsBluePrint>
SeatsBluePrint
id: UUID
line: int
place: int
type: SeatType
price: Price
cinemaRoom: CinemaRoom
seatingPlan: CinemaRoomSeatingPlan
Price
id: UUID
price: double
type: SeatType