KINO

BUCHUNGSSYSTEM

Im Rahmen der Vorlesung “Fallstudie”

Binh, Cliff, Jannis, Leon, Tobias und Rebekka

INHALTSVERZEICHNIS

PROJEKT
ARCHITEKTUR
1.0

In diesem Kapitel wird die Anordnung der Systemkomponenten sowie ihre Beziehungen zueinander in hierarchischer Reihenfolge beschrieben.

FALLSTUDIE – HERR TIELSCH

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

PROJEKT-

1.1 AUFBAU

Das Projekt wurde mit dem Spring Initalizr erstellt und ist ein nach der MVC- Architektur aufgebautes, auf Spring Boot basierendes Maven-Projekt mit Java.

Das Model beinhaltet in der vorliegenden Implementierung die verschiedenen Entitäten, Models und Repositories, der Controller beinhaltet alle Endpunkte und Services. Das View entspricht dem FrontEnd, das in React implementiert wurde und nicht direkt in Spring eingebunden ist, sondern über REST kommuniziert. Als Grundlage diente unser im zweiten Semester erstelltes Klassendiagramm.

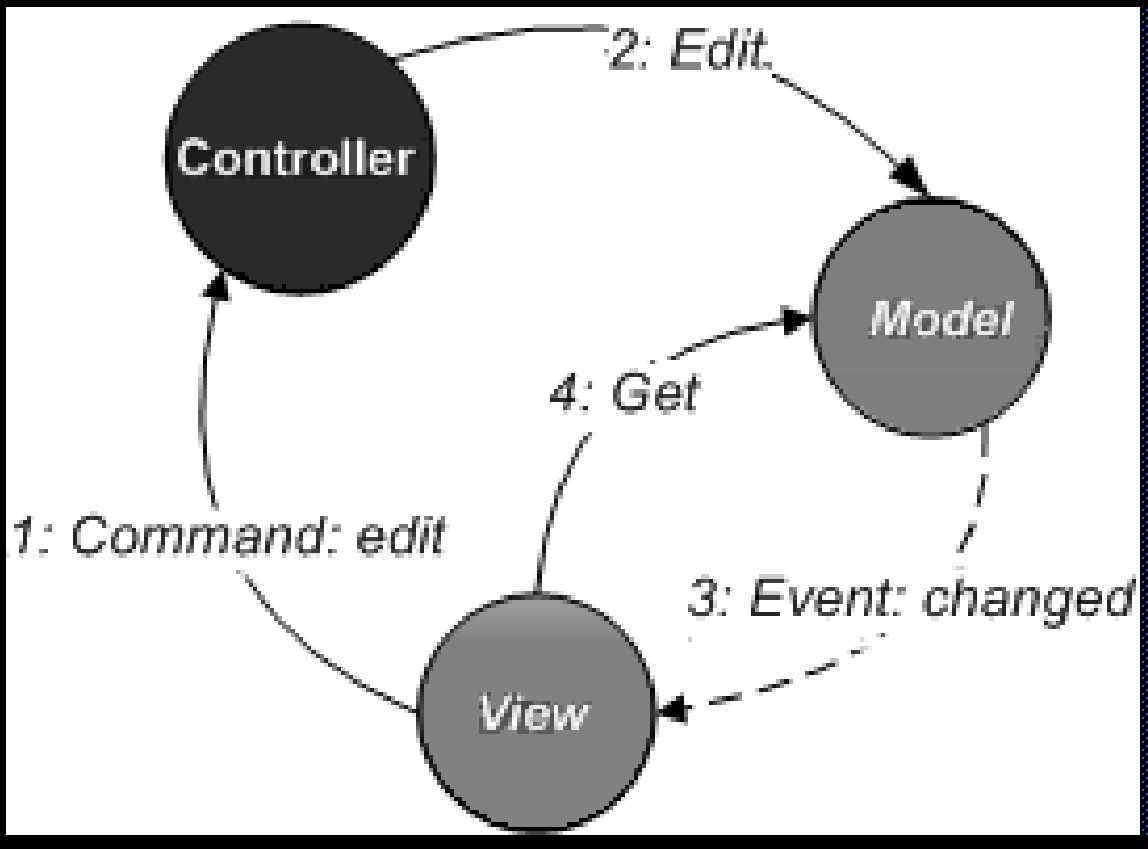
Die hieraus resultierenden Entitäten wurden in mehrere Modelle überführt, um Daten in Spring speichern zu können.

Um für Spring kenntlich zu machen, dass es sich um JPA-Entitäten handelt, sind die Klassen jeweils mit der Annotation @Entity versehen. Jede Klasse enthält eine Annotation, die mit @Id annotiert ist. Sie kennzeichnet die unique Id. Alle Klassen enthalten mit der Datenbank übereinstimmende private Attribute, die über Getter- und Settermethoden abgefragt und manipuliert werden können.

FALLSTUDIE – HERR TIELSCH

MVC-PATTERN

1.2



BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

MVC-PATTERN

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

FALLSTUDIE – HERR TIELSCH

Das MVC-Pattern ist ein Konzept, welches sich sowohl als Architektur-, als auch als Entwurfsmuster eignet und einen flexiblen Programmentwurf ermöglicht, der spätere Änderungen und Erweiterungen erleichtert. Dazu wird die Applikation in drei Komponenten unterteilt, welche durch das Datenmodell (nachfolgend Model (engl.)), die Präsentation (nachfolgend View (engl.)) und die Programmsteuerung (nachfolgend Controller (engl.)) repräsentiert werden. Rückgaben verarbeitet.

Das Model kann aus mehreren Klassen bestehen, wobei jede Klasse eine elementare Einheit innerhalb der verwendeten Datenstruktur darstellt. Das View soll als grafische Benutzeroberfläche die im Model vorhandenen Daten darstellen und Benutzerinteraktionen an den Controller weitergeben. Eine direkte Verbindung zwischen dem Model und dem View existiert nicht. Der Controller fungiert als Bindeglied zwischen Model und View und steuert diese, indem er durch das View über Benutzerinteraktionen informiert wird und diese nach einem Aufwertungsprozess durch entsprechende Datenänderungen oder Rückgaben verarbeitet.

CONTROLLER

Um die verschiedenen Abfragen über Schnittstellen zur Verfügung stellen zu können, werden verschiedene Controller benötigt. Die implementierten Controller-Klassen enthalten alle folgende Annotationen:

1.3

- **@RestController**

Definiert den Endpunkt als Rest-Endpunkt.

- **@RequestMapping**

Definiert die subUri, unter der der entsprechende Controller erreicht wird.

- **@CrossOrigin**

Ermöglicht es, den Service ohne CORS-Fehler zu erreichen.

- **@Autowired**

Ermöglicht es, ein Bean per Dependency-Injection erst bei Bedarf zu verwenden.

- **@PutMapping**

Nimmt eine HTTP-Put-Anfrage an, die einen Request-Body mit JSON Nutzlast enthält

- **@GetMapping**

signalisiert, das die unterstehende Methode, über ein HTTP-Get aufgerufen werden soll.

BACKEND
INFRASTRUKTUR
2.0

Um backendseitig die Anfragen des Kinoticketreservierungssystems mittels einer Logik zu verarbeiten und Daten zwischen einer Datenbank und dem Frontend weitergeben zu können, haben wir uns für die Verwendung von Java Spring Boot entschieden.

FALLSTUDIE – HERR TIELSCH

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

2.1

Nahezu alle vorstellbaren Applikationen lassen sich mithilfe der in Spring Boot einbindbaren Erweiterungen und der von Drittanbietern inkludierbaren Bibliotheken umsetzen.

Spring Boot bietet einen hohen Funktionsumfang, welcher vor allem aus den Kernkonzepten von Spring Boot basiert.

Der Inversion of Control (IoC) Container instanziiert sogenannte Spring Beans, das sind Java Objekte, die anhand einer vom Entwickler geschriebenen Bean Definition erzeugt und verwaltet werden. Diese Bean Definition kann in Form von Annotationen oder als xml Konfiguration bereitgestellt werden, in unserem Projekt finden sich viele solcher Annotationen, wie etwa @Service oder @Repository. Über diese Annotationen weiß der IoC Container, wie er die darunterstehenden Klassen zu deuten hat.

Ein weiteres Kernkonzept in Spring Boot stellt die Dependency Injection (Abhängigkeitsinjektion) dar. Hierbei verwaltet Spring für uns die Abhängigkeiten eines Objekts zur Laufzeit. Über die Annotation @Autowired (die sich auch häufig in unserem Backend-Quellcode findet) wird angezeigt, dass dieses Bean die Injektion eines anderen Beans benötigt. Spring setzt dann die Referenz in diesem Bean auf das entsprechende benötigte Bean aus dem IoC.

Mittels Java Spring Boot haben wir also einen REST-Service aufgebaut, der die Schnittstelle zum Frontend darstellt. Dieser ist über unseren Tomcat-Server zu erreichen: http://5.45.107.109:4000/api über entsprechende Pfade (wie /moviedata) lassen sich dann bereitgestellte Daten abrufen, bzw. HTTP-Methoden wie POST, PUT oder GET aufrufen. Im Backend-Code werden diese Pfade über die Annotation @RequestMapping() angefordert und über @PostMapping(), @PutMapping() oder @GetMapping() entsprechend den REST-API Abfragen zugeordnet. @RestController teilt Spring mit, dass die zurückgegebenen Daten der Methoden direkt in eine Antwort auf eine solche Anfrage geschrieben werden sollen.

Wird eine auf Spring Boot basierende Applikation gestartet, Beans und Einstellungen dynamisch geladen und auf den Applikationskontext angewendet, ohne Quellcode generieren zu müssen oder vorhandene Dateien zu ändern. Hierbei kann eine Bean mehrfach verwendet werden, es muss jedoch nur einmal instanziiert werden.

FALLSTUDIE – HERR TIELSCH

MAVEN

Dieses Projekt wurde über den Spring initializr als Maven Projekt angelegt

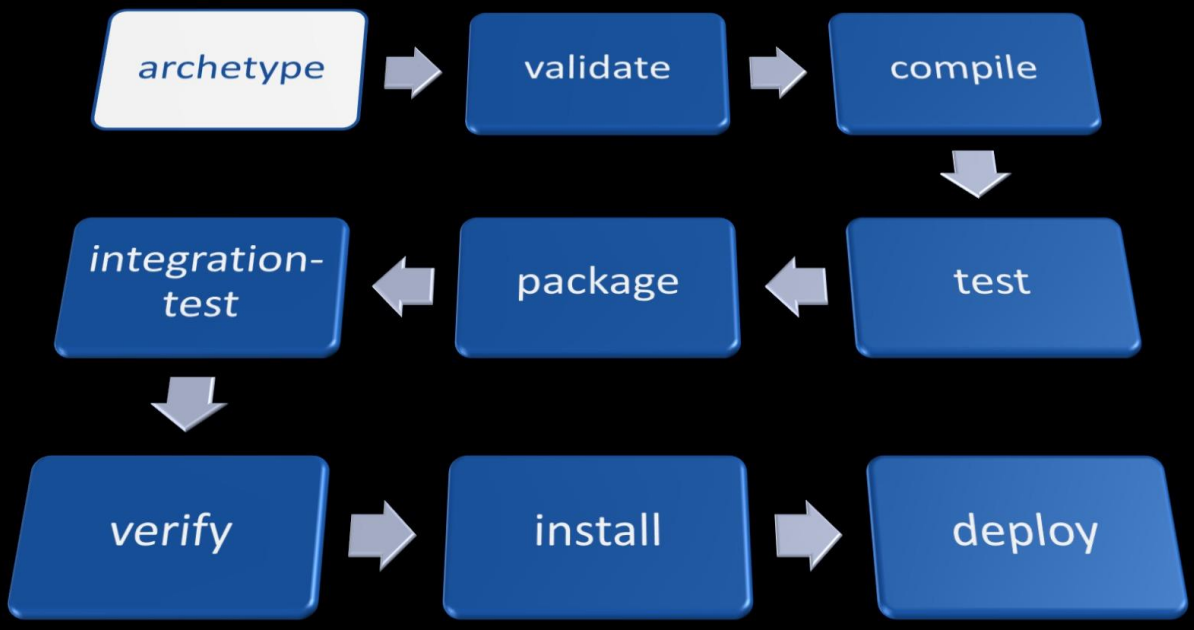
2.2

Maven, ist ein jiddisches Wort für Wissensspeicher und begann als Versuch, die Bauprozesse im Turbinenprojekt von Jakarta zu vereinfachen. Es gab mehrere Projekte, jedes mit seinen eigenen Ant-Build-Dateien, die alle leicht unterschiedlich waren. JARs wurden in CVS eingecheckt. Hieraus wurde eine Standardmethode zum Erstellen von Projekten welche eine klare Definition des Projektinhalts, eine einfache Möglichkeit zum Veröffentlichen von Projektinformationen und eine Möglichkeit, JARs über mehrere Projekte hinweg zu teilen mit sich brachte. Maven ist eines der beliebtesten und am weitesten Verbreiteten Java Build-Tools.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="h
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>Kinoticketreservierungssystem</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>Kinoticketreservierungssystem</name>
    <description>Kinoticketreservierungssystem</description>
```

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM



In diesem Projekt wurde sich gegen Alternativen wie Gradle oder Ant entschieden, da Maven als das Stabilste Tool gilt und weltweit am weitesten verbreitet ist. Die Konfiguration von Maven erfolgt über die Pom.xml, hier werden verwendete Plugins und Dependencies eingebunden, aber auch Versionen etwa von Java festgelegt.

Maven hat einen Lifecycle mit 8 Phasen (siehe Abbildung), in welchen der Code validiert wird, kompiliert, Tests durchgeführt werden, das Projekt in ein Package gepackt wird und installiert und deployed wird. In der Pom.xml wurden die Einstellungen vorgenommen, welche unter anderem die File packaged, sodass diese dann auf dem Tomcat Server deployed wird und läuft. Maven bietet außerdem eine sehr angenehme automatische Teststruktur.

Eine Kernfunktion in Maven sind Dependencies (Abhängigkeiten). Über die Pom.xml können Dependencies in Maven eingebunden und verwaltet werden. Für jede Dependency muss eine Group-ID und eine Artifact-ID angegeben werden, darüber wird Maven mitgeteilt, welche Abhängigkeitsdaten für das Projekt erforderlich sind. Desweiteren können Feinheiten wie die zu verwendende Version, der Scope oder auch Ausschlüsse spezifiziert werden.

Für unser Projekt wurden eine Reihe von Dependencies eingebunden. Standardmäßig hat der Spring Initilizr einige essenzielle Abhängigkeiten direkt von Beginn an inkludiert, etwa "Spring Boot Starter Web" und "Spring Boot Starter Tomcat", um das direkte starten auf einem lokalen Tomcat Server aus der Entwicklungsumgebung heraus zu ermöglichen. Weiterhin waren zusätzliche Abhängigkeiten für die Implementierung unseres Reservierungssystems erforderlich, so etwa die Einbindung der Azure Cosmos Datenbank, das Erstellen von PDFs für die Tickets, JUnit für automatisiertes Testing, oder JaCoCo für die Auswertung der Testabdeckung.

Zu Beginn unserer Pom.xml steht eine Zeile mit:

```
<packaging>war</packaging>
```

Diese Zeile bewirkt, dass das Projekt beim Ausführen des Lifecycle-Goals "Package" zu einem .war-File verpackt wird. Dieser Schritt ist wichtig, da das gebuildete Projekt anschließend auf einem Tomcat-Server deployed werden soll.

JENKINS 2.3

Ein wichtiger Knotenpunkt in der Infrastruktur unseres Kinoticketreservierungssystems ist der Jankins-Server. Wir setzen Jenkins als CI/CD-Tool ein, sowohl für das Frontend, als auch für das Backend. Für Front- und Backend existieren jeweils eigene GitHub-Repositories:
https://github.com/juengeja/KinoBackend
https://github.com/juengeja/KinoFrontend

Unser Jenkins ist nicht öffentlich lesbar, das Login ist zu erreichen unter:

http://5.45.107.109:8080/

Mit den Anmeldedaten:
User: tielschgreg
Passwort: 6bSVfyCh6BWuq3k
Erhält man Zugang zu Jenkins.

In Jenkins gibt es zwei Projekte: jenkins-react-app und KinoBackend. Diese sind jeweils über eine Verlinkung und einen GitHub-Hook mit dem zugehörigen Repository verbunden. Sobald etwas neues in eines der beiden Repositories gepusht wird, nimmt Jenkins davon Notiz und löst für Front- bzw. Backend den Build-Vorgang aus.

Für das Frontend-Projekt bedeutet das: In einem Jenkinsfile (enthalten im GitHub-Repo) befindet sich die Definition einer Pipeline. Diese wird ausgeführt, das React-Projekt wird gebuildet und über Nginx deployed. Die Änderungen sind nach Erfolgreichem Durchlaufen der Pipeline auf unserer Kinowebsite zu bestaunen:

http://5.45.107.109:3000/

Für das Backend-Projekt hat ein Push in das zugehörige Repository ähnliche Konsequenzen. Auch hier wird ein Build ausgelöst, der CI/CD-Ablauf ist jedoch direkt in Jenkins definiert. Es wird der Maven-Lifecycle "Package" ausgeführt, das Projekt wird in einem .war-File verpackt, dieses wird anschließend auf einem Tomcat Server deployed. Die deployede Version läuft anschließend und kann vom Frontend angesprochen werden.

Der hierfür erforderliche Tomcat Server ist ebenfalls auf unserem Debian-Server installiert und läuft Standardmäßig auf Port 4000. Unter dem Pfad /api mit entsprechenden unterpfaden wird das Kinoprojekt deployed, hier kann dann mittels HTTP-Anfragen kommuniziert werden.

Das Frontend wird über die Pipeline auf einem NGINX-Server deployed, dieser wartet auf Bereitstellung von Dateien und gibt diese dann auf Port 3000 wider.

TESTING 2.4

Eine Besonderheit des Backend Projekts in Jenkins ist, das als Post-Build Schritt ein Report über die Testabdeckung erstellt wird. Sobald der main Branch gebuildet wird, wird mittels JaCoCo ein Bericht der Testabdeckung generiert, dieser wird daraufhin in Jenkins angezeigt. Dazu muss man lediglich den letzten Build des Projekts anzeigen, daraufhin auf "Testabdeckung" klicken und man erhält eine detaillierte Zusammenfassung der Backend-Tests.

JENKINS

Insgesamt ist eine Testabdeckung von 60% Statement-Coverage erreicht, nicht alle Klassen und Methoden waren auch sinnvoll zu testen.

Overall Coverage Summary

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
all classes	61% M: 1409 C: 2163	40% M: 96 C: 64	58% M: 165 C: 232	64% M: 324 C: 567	70% M: 95 C: 222	97% M: 1 C: 35

Coverage Breakdown by Package

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
com.example.kinoticketreservierungssystem	M: 5 C: 3 38%	M: 0 C: 0 100%	M: 1 C: 1 50%	M: 2 C: 1 33%	M: 1 C: 1 50%	M: 0 C: 1 100%
com.example.kinoticketreservierungssystem.blSupport	M: 72 C: 166 70%	M: 4 C: 4 50%	M: 12 C: 30 71%	M: 26 C: 61 70%	M: 8 C: 30 79%	M: 0 C: 3 100%
com.example.kinoticketreservierungssystem.config	M: 13 C: 55 81%	M: 0 C: 0 100%	M: 4 C: 13 76%	M: 5 C: 19 79%	M: 4 C: 13 76%	M: 0 C: 3 100%
com.example.kinoticketreservierungssystem.controller	M: 130 C: 18 12%	M: 2 C: 0 0%	M: 11 C: 6 35%	M: 19 C: 6 24%	M: 10 C: 6 38%	M: 0 C: 6 100%
com.example.kinoticketreservierungssystem.entity	M: 566 C: 900 61%	M: 64 C: 30 32%	M: 104 C: 138 57%	M: 153 C: 309 67%	M: 58 C: 137 70%	M: 1 C: 10 91%
com.example.kinoticketreservierungssystem.repository	M: 0 C: 0 100%	M: 0 C: 0 100%	M: 0 C: 0 100%	M: 0 C: 0 100%	M: 0 C: 0 100%	M: 0 C: 0 100%
com.example.kinoticketreservierungssystem.service	M: 364 C: 1018 74%	M: 26 C: 30 54%	M: 23 C: 43 65%	M: 76 C: 170 69%	M: 4 C: 34 89%	M: 0 C: 11 100%
com.example.kinoticketreservierungssystem.tempController	M: 259 C: 3 1%	M: 0 C: 0 100%	M: 10 C: 1 9%	M: 43 C: 1 2%	M: 10 C: 1 9%	M: 0 C: 1 100%

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

DATENBANK KONZEPT 3.0

In diesem Projekt wurde mit Azure gearbeitet. Eine Azure Cosmos DB Datenbank ist ein skalierbarer nicht-relationaler Datenbankdienst, welcher für die Cloud entwickelt wurde. Sie stützt automatisierte, KI-gestützte Funktionen, und bietet ein serverloses Computing- und Hyperscale-Speicheroptionen an, mit welcher die Ressourcen automatisch nach Bedarf skalier werden können. Der Datenbankdienst bietet schlüsselfertige, globale Verteilungen, Multi-Master-Replikationen, automatische Skalierung und Lese-/Schreiblatenz im einstelligen Millisekunden Bereich an. In diesem Projekt wurde Azure über eine integrierte API Schnittstelle verwendet.

AZURE DATABASE

FALLSTUDIE – HERR TIELSCH

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

DATENBANK KLASSEN 3.1

Der Zugriff auf die Datenbank wird in folgenden Klassen geregelt:

Klasse Cosmos DB:

Ruft Datenbank Zugangsdaten(schlüssel, URI etc) aus der ressource application.properties auf

Klasse ConfigDatasource:

Regelt die Verbindung und nimmt dafür die aus CosmosProperties geflieferten Variablen und erstellt einen Client zur Anbindung

DATENBANK ANBINDUNG 3.2

Entitäten in der Datenbank werden mit der Annotation @Container gekennzeichnet. Container sind Dokumentansammlungen und enthalten die JSON Dokumente, welche auch "Items“ gennant werden. Dabei sind die Container Namen einzigartig.
Jedes Item in einem Container hat somit eine Unique ID und einen PartitionKey. Welches Attribut als Unique ID und PartitionKey genommen wird, wird in den Entitätsklassen geregelt und entsprechend mit @Id und @PartitionKey gekennzeichnet.

Ein PartitionKey teilt Items in unterschiedliche physische Partitionen auf. Die Items werden also je nach PartitionKey in Gruppen unterteilt, wobei pro Gruppe ein eigener physischer Rechen und Speicherplatz existiert. Entsprechend werden dadurch die Rechen und Speicherleistung bzw Abfrage und Einfügegeschwindigkeit der Daten gleichmäßig auf die Rechenzentren aufgeteilt. PartitionKey sollte als Attribut möglichst final sein. Sollte entsprechend auch nicht verändert werden, genau wie die ID. Es ist zudem ratsam ein Attribut zu nehmen, welches die Items in dem Container in gleichmäßige Gruppen aufteilt damit die einzelnen Partitionen auch entsprechend gleichmäßig groß sind. Ein typisches Beispiel wäre das "Genre".

Ebenfalls hat auch jeder Container einen einzigartigen Namen. Die Container werden zusammnen in einer Datenbank "dhbw-kino-free-tier" gespeichert. Für diese Datenbank ist ein fester Maximal Durchsatz am Anfang festgelegt worden.

Azure CosmosDB rechnet dabei nach sogenannten RU also "Request Units" ab. Komplexere Datenabfragen verlangen entsprechend mehr Request Units. Einfache findByID Abfragen kosten ca. 0,5-2 Request Units. Komplexere Queries hingegen können um die 2-8 RU kosten.

Der in diesem Projekt festgelegte Durchsatz beläuft sich auf 1000 RU/s. 1000 Request Units pro Sekunde können durch die Datenbank maximal bearbeitet werden. Wenn mehr Abfragen stattfinden erhöht dies die Request Zeit, da jedoch der Durchsatz auf dieses lLmit festgelegt wurde kann dieses auch nicht überschritten werden.

DATENBANK ABFRAGEN 3.3

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

Auf der Datenbank werden über CRUD Operationen Abfragen gehandlet. Diese sind im CosmosRepository Interface definiert, welches wiederum eine Implementierung des CRUDrepository Interfaces ist und CRUD Operationen speziell an CosmosDB angepasst regelt. Jede Entität, bedeutet, dass jeder Container eine eigene Implementierung des CosmosRepository Interface hat.

-Hierin werden die jeweiligen CRUD Operationen auf den jeweiligen Container zugeschnitten

Zu den wichtigsten CRUD Operationen gehören:

findBy: Diese Operation kann mit den jeweiligen Attributen nach denen man einen Container durchsuchen möchte, erweitert werden. Ein Beispiel wäre „findByUsername“

queries: Diese sind mit @Query Annotationen gekennzeichnet und ermöglichen eigene SQL Abfragen auf die Cosmos Datenbank. Diese können angewendet werden, weil die SQL API von Cosmso DB benutzt wird. Dabei werden @Query Abfragen ermöglicht und die findBy Abfragen in Queries umgewandelt. Dies kostet zwar mehr Request Units ist jedoch um einiges schneller ist und kann noch weiter angepasst werden. Beispielsweise können sich mit findAllByPriceGreaterThan(double 3.5) alle Items in dem Container ausgeben lassen, welche mehr kosten als 3,5.

save(): Hiermit werden Objekte aus dem Code, sofern sie einer Entität entsprechen, die als Container in der Datenbank aufgelistet ist, in dieser gespeichert. Gleiche Einträge werden dabei von einem neuen Eintrag überschrieben. Da die Kombination aus @Id und @PartitionKey der Entitäten Unique ist, werden diese Attribute in der **equals** und **hashcode methode** einer jeden Entität verglichen. Nur beides zusammen ist somit unique, weil beispielsweise die gleiche @Id von Items im selben Container genutzt werden kann, wenn beide einen unterschiedlichen @PartitionKey haben.

Methoden in den @Service Klassen und @Controller Klassen rufen diese CRUD Operationen auf, welche in den Repository interfaces genauer definiert wurden.

FRONTEND

4.0

Das folgende Kapital befasst sich mit dem FrontEnd und der eigesetzten Coding Sprache „React“. React ist eine JavaScript Bibliothek, welche nützliche Komponenten zum Erstellen von Webseiten zur Verfügung stellt. React wurde von Facebook entwickelt und als Open-Source-Projekt veröffentlicht.

FALLSTUDIE – HERR TIELSCH

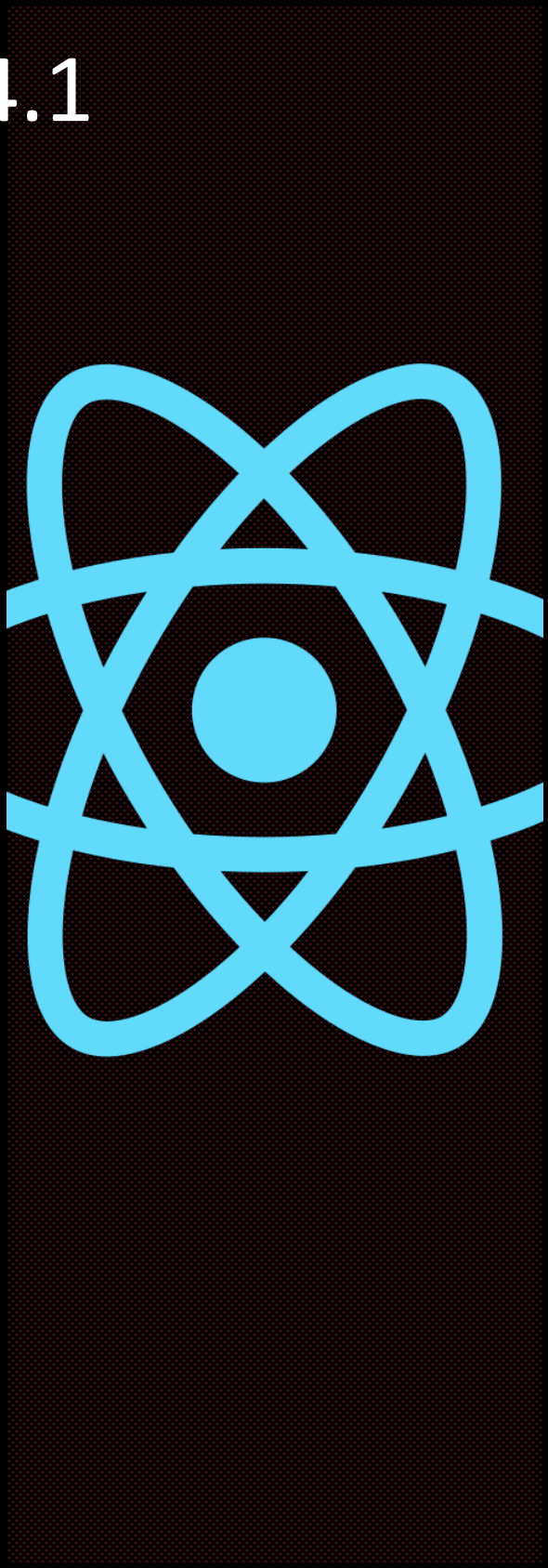
BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

KINO BUCHUNGSSYTSEM

REACT

4.1

Ein Vorteil von React ist der unidirektionale Datenfluss, der die Kommunikation zwischen Komponenten vereinfacht. Somit ist eine Art der Vererbung realisierbar. Dies nutzen wir beim Zugriff auf die Filmliste. In Kombination mit den “States” von React sind Standardprobleme wie die Aktualisierung einer Liste entsprechend einer Sucheingabe effizient umsetzbar (siehe Filter). Die Strukturierung in verschiedene Komponenten erleichtert die Entwicklungsarbeit durch die klare Trennung in einzelne Aufgabenbereiche.



Die Projektstruktur, welche sich im „src Folder“ befindet, lässt sich in folgende Bereiche unterteilen:

- **Components**
- **Images**
- **Pages**
- **„allgemeine Dateien“**

Unter die allgemeinen Dateien fallen standardmäßige React-Dateien wie die index.js. Außerdem ist hier unsere App.css abgelegt, welche sämtliche Designs der Website beinhaltet. Auch die MovieContet.js befindet sich hier, welche es uns ermöglicht Daten „Global“ zur Verfügung zu stellen. In unserem Fall handelt es sich hierbei um die Filmdaten. Die gastroData.json umfasst die in unserem Kino verfügbaren Gastroangebote inkl. Preis.
Der Ordner Images beinhaltet feste, sich nicht (oder sehr selten ändernde Bilder), wie Titelbild Logo oder Banner.
Es folgt nun eine Auflistung der jeweiligen Seiten und deren Funktionen:

AddToShoppingCart: Diese Seite öffnet sich, nachdem man auf der Single Movie Seite auf buchen drückt. Hier werden vom backend, alle zu dem gewählten Film verfügbaren Events geladen und als Liste dargestellt. Wählt man ein Event aus, so öffnet sich der entsprechende Sitzplan des Events, welche ebenfalls vom Backend angefordert wird. Der Sitzplanaufbau ist Hardgecodet in der Datei und wird mit den erhaltenen Daten gefüllt. Hat man Sitze ausgewählt, kann man diese Reservierung entweder in den Warenkorb legen oder direkt zur Kasse gehen. Bei beiden Varianten werden die gewählten Sitze zurück ans Backend gegeben und dort für 15 min blockiert. Zurückgeliefert wird ein Booking JSOn, welche alle Informationen zur Reservierung beinhaltet und im weiteren vom Frontend zwischengespeichert und ergänzt wird.

Booking: Hier gelangt man hin, wenn man bei AddToShoppingCart auf direkt buchen oder im Warenkorb auf zur Kasse drückt. Hier werden nochmals die gewählten Filme mit Datum und Sitzen angezeigt (aus dem Zwischenspeicher). Wenn alle erforderlichen Daten (customerInfo) eingegeben wurde, werden diese zu der bereits erhaltenen Booking JSON hinzugefügt und diese wird wieder ans backend zurückgegeben. Als Rückgabe erhält man wieder ein Booking.Json, welches ein successful-Attribut enthält

Info: Dieser Bereich beinhaltet die Kontakt- und Anfahrtstinformationen. Zusätzlich kann der Nutzer hier mit einem Knopfdruck auf die verschiedenen Icons direkten Kontakt aufnehmen

Infektionsschutz: Dieser Bereich beinhaltet aktuelle Informationen zu der aktuellen pandemischem Lage.

Zahlungsmöglichkeiten: Hier werden alle, angebotenen Zahlungsmethoden aufgelistet. Man gelangt auf diese Seite, indem man auf der Startseite bei Services auf das Bitcoin Icon drückt.

Error: Alle Seiten werden über einen Pfad, welcher in der App.js definiert ist aufgerufen. Wird ein Pfad aufgerufen, welcher dort nicht definiert wurde, erscheint diese Fehlerpage (Err 404).

Home: Dies ist die Startseite. Sie setzt sich, wie einige andere Seiten auch, aus mehreren Komponenten (Banner, Angesagte Filme, Services und Impressionen), welche im weiteren näher erläutert werden, zusammen.

Programm: Hier wird das Programm angezeigt. Die Filmdaten werden vom Backend auf einer Domain im JSON Format bereitgestellt und können dort vom Frontend aufgegriffen werden. Mit Hilfe der movieContext.js werden die Filme geladen und können von jeder Seite aufgerufen werden. Neben der Auflistung der Filme befinden sich hier noch Filter, um die Auswahl zu begrenzen. Die Verarbeitung der Filter findet ebenfalls in der MovieContext.js statt und liefert eine neue Liste an Filmen zurück, welche dann dargestellt wird.

ShooppingCart: Hier wurden alle gewählten Filme, mit Eventdatum und Sitzen aufgelistet. Wurde ein Menü hinzugefügt, so wird auch diese hier aufgelistet. Die zwischen Speicherung der Daten wurde Mithilfe von React-Redux umgesetzt. Ist das Array an Reservierungen leer, wird kein Film angezeigt und ein Button zum Programm dargestellt.

SingleMovie: Hier sind nähere Details zu einem gewählten Film einsehbar. Klickt man im Programm oder auf den Startseiten auf Details bei einem Film gelangt man hier hin. Hier sind Informationen wie Regisseur, Beschreibung usw. einsehbar. Diese Daten stammen ebenfalls von der bereits erwähnten URL.

Adminpage: Um in den Adminbereich zu kommen, muss sich der Admin mit einem in der Datenbank gespeicherten Konto auf der Website anmelden. Dies kann er tun, indem er rechts oben im Navigations-Menü auf das Login Piktogram klickt. Meldet sich der Admin mit seinen Nutzerdaten korrekt an, kann er sich nun in den Adminbereich begeben. Hier kann der Admin vor allem neue Filme und deren Informationen einstellen. Ist kein Admin angemeldet, führt das Icon immer zur Login-Page, nach erfolgreicher Anmeldung führt es auf die Admin-Page. Abmelden ist über den Button am Ende der Seite möglich. Der Admin verfügt auf dieser Seite über die Möglichkeit, neue Filme anzulegen, diese im Anschluss auszuwählen und ein entspr. ShowEvent zu generieren.

Gastro: In diesem Bereich könne sich die Besucher einen Überblick über die Speisekarte im Gastro des Kinos machen und sich die Menüs anschauen. Um ein Menü online zu bestellen müssen sie jedoch erst einen Film buchen. Diesem Film kann dann entsprechend ein Menü hinzugefügt werden inklusive eines Rabatts von 10%. Wird der Warenkorb wieder geleert, so wird automatisch auch das Menü wieder entfernt.

Weitere Komponenten 4.2

Unter die Komponenten fallen einige Dateien, welche den Aufbau der Seiten erleichtern, bzw. Teile des Codes auslagern. In den Action und Reduces Ordeners befinden sich Dateien, welche dafür sorgen, dass die Speicherung der Filme im Warenkorb möglich ist. In dem Ordner PopUps befinden sich alle verfügbaren Popups, welche nach dem Reservierungs- und Buchungsprozesses aufgerufen werden. Komponenten wie Banner, Titel oder NavBar geben, wie der Name bereits sagt unser Banner bzw. einen großen Titel auf jeder Seite oder die NavBar zurück. Diese Komponenten werden so gut wie von jeder Seite aufgerufen, um ein einheitliches Erscheinungsbild darzustellen.

ContactIcons und Services beinhalten jeweils einige React-Icons mit einem Titel und einer Beschreibung. Die Servies werden von der Home-Seite und die Contact-Icons von der Info Seite aufgerufen.

Auf der Startseite werden zusätzlich noch Beliebte Filme (FeaturedMovies.js) angezeigt, welche 3 beliebte Filme darstellt. Außerdem wird hier auf die SlideShow zugegriffen, welche einige Bilder des „Kinos“ zeigt und automatisch das Bild wechselt, umgesetzt mit Hilfe von react-slideshow-Images.

Das Programm setzt sich zusammen aus ProgramContainer, sowie MovieList und MovieFilter.

BINH, CLIFF, JANNIS, LEON, REBEKKA, TOBIAS

AUFGABENBEREICHE

Dich, Quang Binh 2007710
BackEnd, Datenbank Management, Quality Management

Engelhart Leon 5527999
FrontEnd, Buchungssystem

Fitzke, Tobias 5771308
FrontEnd, Team Leader, Quality Management

Jüngert, Jannis 2079264
BackEnd, Server built

Miguez, Rebekka 6979552
Testings, Backend

Okoro, Cliford 8158562
Testings, Communication, Documentation