HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Fakultät

Ingenieurwissenschaften und Informatik

Schriftliche Ausarbeitung zum Thema:

Entwicklung eines Theaterplaners unter Anwendung des Domain-Driven-Designs mit Quarkus REST-APIs, Webcrawling und Serverseitigem Rendering

im Rahmen des Moduls
Software-Architektur – Konzepte und Anwendungen,
des Studiengangs Informatik-Medieninformatik

Autor:	Jonas Nelson
Matr.-Nr.:	860359
E-Mail:	jonas.nelson@hs-osnab-rueck.de
Themensteller:	Prof. Dr. Rainer Roosmann

Abgabedatum: 24.07.2023

Inhaltsverzeichnis

Abbildungsverzeichnis

Die Abbildungen eines Anhangs werden nicht im Abbildungsverzeichnis aufgeführt. Dieses gilt analog für das Tabellenverzeichnis.

Tabellenverzeichnis

Source-Code Verzeichnis

Abkürzungsverzeichnis

CDI	Context and Dependency Injection for the Java EE Plattform
ECB	Entity-Controller-Boundary Pattern
Java EE	Java Enterprise Edition, in der Version 7
SWA	Software-Architektur
IDE	Integrierte Entwicklungsumgebung

1 Einleitung

1.1 Vorstellung des Themas

Theater ist eine faszinierende und kulturell bereichernde Kunstform, die Emotionen weckt und uns zum Nachdenken anregt. Als leidenschaftlicher Theatergänger musste ich allerdings feststellen, dass die Website, des Osnabrücker Theaters auf der ich Informationen über zukünftige Theaterstücke und Aufführungen finde, oft unübersichtlich und bezüglich der Planung von Besuchen relativ unkomfortabel ist. Da es keinen anderen Theaterplaner gibt, der meine Anforderungen erfüllt und das eigenständige Planen sehr viel Zeit in Anspruch nimmt, entstand die Idee, eine eigene App zu entwickeln, die mir und anderen Theaterfreunden dabei hilft, unsere Theaterbesuche zu planen und die Informationen schnell und in einem übersichtlichen Rahmen zu finden und zu filtern.

So soll die App den Nutzern erleichtern, kommende Veranstaltungen zu entdecken, Aufführungstermine einzusehen und Theaterstücke nach individuellen Interessen zu filtern. Hierbei ist zu erwähnen, dass es ermöglicht wird, Veranstaltungen und einzelne Aufführungen mit einem Status zu versehen, sowie sich Lieblingsveranstaltungen als Favorit zu markieren, um sie sich immer in Erinnerung zu halten. Weiterhin soll es ermöglicht werden, sich Kommentare zu setzen, sowie eine schnelle Übersicht über die Veranstaltungsorte und Theaterkassen zu erlangen. Durch die intuitive Benutzeroberfläche soll die Planung von Theaterbesuchen zu einer deutlich angenehmeren Erfahrung werden.

1.2 Ziel der Ausarbeitung

Das Ziel dieser Arbeit besteht darin, eine Softwarearchitektonische Lösung für den Theaterplaner zu entwickeln, der unter anderem das Domain-Driven-Design als architektonisches Paradigma nutzt und Level 3 des Richardson Maturity Models anstrebt. Dabei sollen REST-APIs mittels Quarkus als Technologieplattform genutzt werden, um die notwendige Interaktion mit anderen Systemen zu ermöglichen. Auch wird Web-Crawling verwendet, um die Daten von der Theaterwebsite Osnabrück zu analysieren und in eine Datenbank einzupflegen, welche als Grundlage für die geplante App dienen. Server Side Rendering mit Quarkus Qute soll hierbei für eine angemessene Web-App sorgen, welche den Theaterplaner im Browser zugänglich macht.

Hierbei werden die, in der im Anhang angefügten Muss- Wunsch- und Abgrenzungs- Kriterien der Projektaufgabe aufgegriffen und implementiert.

Es ist wichtig anzumerken, dass der Fokus dieser Arbeit schwerpunktmäßig auf der technischen Umsetzung dieser Applikation liegt, sodass die Herausforderungen bei der Anwendung des Domain-Driven Designs sowie der Integration von Quarkus REST-APIs, Web-Crawling und Serverseitigem Rendering im Fokus liegen. Die User Experience, welche durch Mockups bereitgestellt wurde, wird als bereits als gegeben angenommen und steht nicht im Zentrum der Untersuchungen.

1.3 Aufbau der Hausarbeit

Zunächst werden die theoretischen Grundlagen und Technologien beleuchtet, um eine Basis für die Entwicklung der Applikation zu schaffen.

In dem Kapitel Anwendung erfolgt eine detaillierte Betrachtung der Projektplanung und des Vorgehens bei der Entwicklung des Theaterplaners. So werden zunächst die einzelnen Phasen der Projektplanung unter Anwendung des C4-Models betrachtet, woraufhin anschließend die Implementation einiger wichtiger Teile näher beleuchtet wird.

Im darauf anschließenden Kapitel werden die Ergebnisse unter Bezugnahme der zuvor erwähnten Kriterien validiert. Dadurch erhalten wir eine fundierte Bewertung der entwickelten Softwarelösung.

Alle Ergebnisse der Hausarbeit werden dann im abschließenden Teil zusammengefasst. Weiterhin gibt es ein kurzes selbstreflektierendes Fazit, sowie einen kurzen Ausblick auf die zukünftigen Perspektiven des Theaterplaner-Projekts, welcher erläutert, wie es mit diesem Projekt nach der Abgabe weitergeht.

2 Darstellung der Grundlagen

Gemäß der Einleitung werden in diesem Kapitel nun die wesentlichen Grundlagen, die zur Ausarbeitung des Projekts dienen, dargestellt. So wird zunächst detaillierter auf das Theater und die Schnittstelle zum Theater Osnabrück, sowie auf die für dieses Projekt verwendeten Tools und Technologien eingegangen.

2.1 Theater

Das Theater ist als eine darstellende Kunst zu definieren, wie durch [@DU] erläutert wird. In anderen Definitionen von [@FR] wird es als Bezeichnung für eine szenische Darstellung eines inneren und äußeren Geschehens zwischen dem Publikum und den Darstellern beschrieben.

So ist das Theater eine Kunstform, die im Laufe der Geschichte und auch heute noch eine große Rolle spielt. [SIM, 2007]

Wie die [@NOZ1] erläutert, ist das Theater Osnabrück ein Theater welches fünf Sparten bedient: Musiktheater, Schauspiel, Tanz, Konzert, sowie das OSKAR.

Das Theater selbst bietet keine öffentlich bekannte Schnittstelle zu ihrer Datenbank an.

Da dieses Projekt in erster Linie eine Entwicklung für persönliche Zwecke ist, und in gewisser Weise auch einen Proof of Concept darstellt, wird das Theater für dieses Projekt nicht zur Bereitstellung einer Schnittstelle angefragt.

Daher wird sich mithilfe eines Webcrawler, welches eine Technologie bezeichnet, die das Internet durchsucht, wie [@SM1] beschreibt an den öffentlich zugänglichen Daten des Theater Osnabrück bedient.

2.2 Organisatorische Tools

Zur Organisation und Planung des Projekts werden zum einen kreative Tools verwendet, aber auch planerische Tools.

Zu den kreativeren Tools zählt zum einen Figma, welches u.a. ein Prototyping-Tool ist, das für die Entwicklung eines User Interface verwendet wird. Weiterhin kommt Miro für die Erstellung von Mindmaps und Notizen verwendet wird

Neben den zuvor genannten kreativeren Tools werden zur Planung Diagramme mithilfe von lucidCharts entworfen. Dabei wird nach dem C4-Modell gearbeitet, welches dazu dient, die Architektur von Softwaresystemen auf verschiedenen Abstraktionsebenen, gemäß [@C4M] zu beschreiben.

Weiterhin kommt Trello mit seinen Funktionalitäten bzgl. Kanban-Boards zum Einsatz, aber auch git in Kombination mit gitlab zur Versionierung.

2.3 Technische Tools, Technologien und Frameworks

Visual Studio Code wird zum Schreiben von html- und css-Code verwendet, da Visual Studio Code den Anwender mit zahlreichen Tools, wie einen Live-Server zum Visualisieren des Codes unterstützt

Als Entwicklungsumgebung wird Intellij von Jetbrains verwendet. Als Buildtool kommt Apache Maven zum Einsatz. Zur Programmierung mit Java wird das OpenJDK in der Version 17 herangezogen.

Als Full-Stack-Framework wird Quarkus in der Version 2.17 verwendet, welches ein leichgewichtiges Framework ist, mit dem Java speziell für Containerplattformen optimiert wird. [@RH1]

Zur Implementierung der Funktionalitäten werden zusätzlich zur Verwendung von Quarkus verschiedenste Libraries mithilfe von Maven eingebunden, um alle für diese App wichtigen und notwendigen Funktionalitäten zu implementieren.

Zunächst wird RESTEasy Classic zur Implementierung der Restendpoints verwendet. Weiterhin wird JSON-B herangezogen, um die Serialisierung von JSON-B zu vereinfachen und so einfach mithilfe von DTOs Daten in JSON umzuwandeln und zuückzuwandeln

Auch werden verschiedenste Funktionalitäten der Jakarta EE verwendet. Darunter fällt JPA zur Datenbankanbindung, JTA zum Transaktionsmanagement und Bean Validation als Java-Spezifikation zur Datenvalidierung.

Neben diesen Spezifikationen wird als Relationale Datenbank das leistungsstarke Open Source Datenbankmanagementsystem PostgreSQL benutzt. Zusätzlich wird das ebenfalls leistungsstarke und benutzerfreundliche ORM-Framework Panache herangezogen, da es eine modernisierte und vereinfachte Möglichkeit bietet mit Datenbanken zu interagieren. Panache ist für Quarkus optimiert und bietet einige Vorteile gegenüber der klassischen Anbindung per JPA. Mit Panache gehen allerdings auch einige Nachteile einher, da es nur eingeschränkt anpassbar ist und es nur wenige Ressourcen gibt, da Panache im Vergleich zu anderen ORM-Frameworks eine kleinere Entwickler-Community hat und es so auch nicht unbedingt langlebig unterstützt wird. Da dieses Projekt aber auf eine schnelllebige Theaterwebsite gestützt ist, ist diese Potentielle Kurzlebigkeit zu vernachlässigen.

Um die Daten der Website des theater-osnabrueck.de zu crawlen wird die Java Library eingebunden. Diese implementiert die WHATWG HTML5 specification und bietet eine komfortable Schnittstelle, um die Daten der Website zu extrahieren, zu manipulieren und in gängige Javaobjekte zu speichern. [@JS]

Zum Exportieren von Kalenderdaten wird die Java-Library iCal4J in der Version 3.2.11 per Maven eingebunden. Diese Bibliothek ermöglicht ein einfaches Lesen und Schreiben von Kalenderdaten und bietet für diese Anwendung die Möglichkeit diese Kalenderdateien als MIME-Type `text/calendar` zu exportieren. [@IC4J]

Das Framework Eclipse MicroProfile mit seinen Metriken, Fault Tolerance und OpenAPI-Definitionen wird für verschiedene Funktionalitäten verwendet. Der Bereich der Metriken dient zur Erfassung und überwachung verschiedenster Leistungsdaten, wie die Anzahl der Anfragen, die Verarbeitungszeit und Fehlerhäufigkeit, FaultTolerance ist dafür da, die Robustheit der Anwendung zu erhöhen mithilfe von verschiedensten Fehlertoleranzstrategien.

Die OpenAPI Definitionen dienen dazu die Verfügbaren Endpunkte zu beschreiben und Anfrageparameter, Antwortcodes und Ressourcenpfade der API zu Dokumentieren.

Zur Authentifizierung und Autorisierung wird für dieses Projekt zunächst die Basic- Authentifikation in Form der Form-Authentifizierung von Quarkus hinzugezogen, welche auf die Security-Annotationen von Jakarta EE zurückgreift.
Die entwickelte API wird mithilfe von Unit-Tests und Black-Box Tests über Swagger UI getestet.

Die Realisierung des grafischen Web-Frontends geschieht unter Verwendung des Server-Side-Rendering Templating-Engine Qute und wird in Teilen von dem JavaScriptFramework Unpoly erweitert, um einzelne Teile des Dokuments nach Bedarf serverseitig zu aktualisieren, um nicht immer das komplette Dokument neu zu laden.

Letzten Endes wird die Applikation in Form des Entity-Control-Boundary-Patterns und des Domain Driven Designs entwickelt und verwendet grundlegend die SOLID-Pattern.

3 Anwendung

In dem folgenden Kapitel wird die Umsetzung der in den vorigen Abschnitten beschrieben Grundlagen beschrieben. Dazu wird zunächst auf die Projektplanung eingegangen, woraufhin mit der konkreten Implementierung fortgefahren wird. Dabei werden wichtige Aspekte und Herausforderungen beleuchtet, die während des Entwicklungsprozesses aufgetreten sind.

3.1 Projektplanung

Gemäß dem C4-Modell wird zunächst ein Systemkontextdiagramm entworfen, um das theater-os-System in einem gesamtheitlichen Kontext zu betrachten. So wird zunächst betrachtet, welche Aktoren von außen auf das System wirken. Neben dem Theater Osnabrück, welches die Daten für das theater-os-System stellt, liegt ein großer Fokus auf den Personen, die dieses System nutzen können. Dazu werden zunächst Personas erstellt und betrachtet, was diese Personas für Ziele mithilfe des Theater-Os Systems lösen möchten.

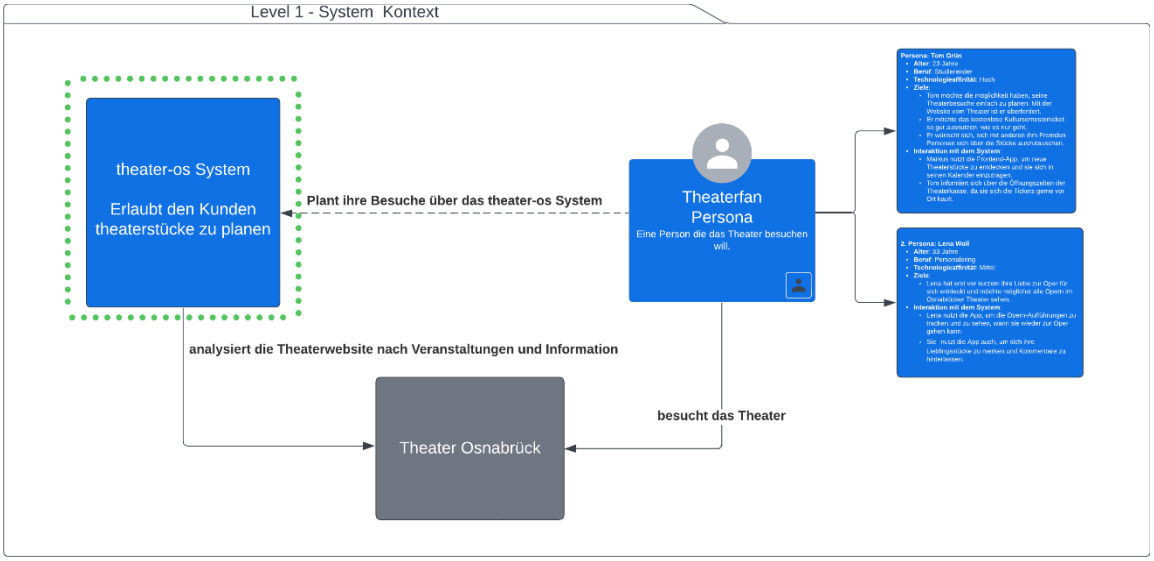

Abbildung 1: System Kontext - Level 1

In einem nächsten Schritt wird das Container Diagramm herangezogen. Hierbei wechselt der Schwerpunkt auf das grün umrandete theater-os System. Dabei sind die wichtigsten Container des Systems herauszustellen. Dies sind konkret die Datenbank, die API-Applikation selbst, die mit Quarkus Qute zu entwickelnde Webapplikation und das Web-Applikation Frontend, welches die mobile First Strategie einsetzt.

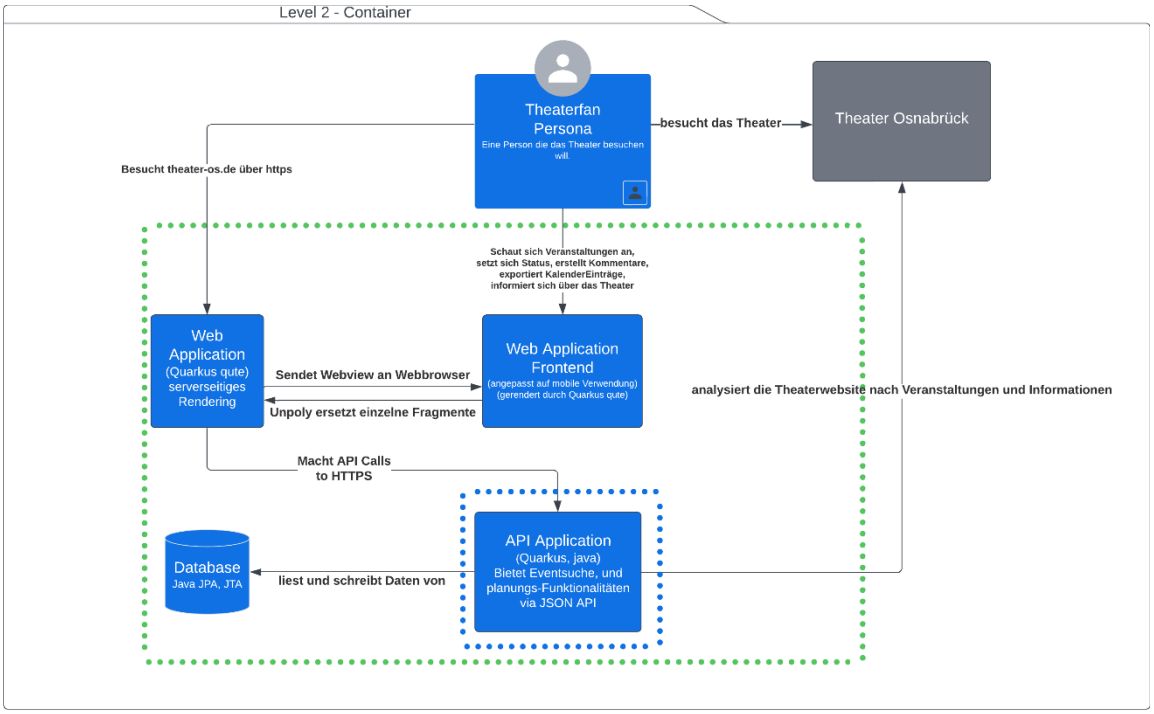

Abbildung 2: Container - Level 2

Im dritten Schritt kommt das Komponentendiagramm, welches sich auf die eigentlichen Komponenten innerhalb der Container stützt. Hier wird die API-Applikation, zu erkennen an der blauen Umrandung, im primären betrachtet. Dabei stellen sich gemäß des Domain Driven Designs für dieses Projekt sechs verschiedene fachlich getrennte Module heraus. Dazu zählen die Module Security, Usermanagement, eventmanagement, commentmanagement, theaterinfo und dataprovider.

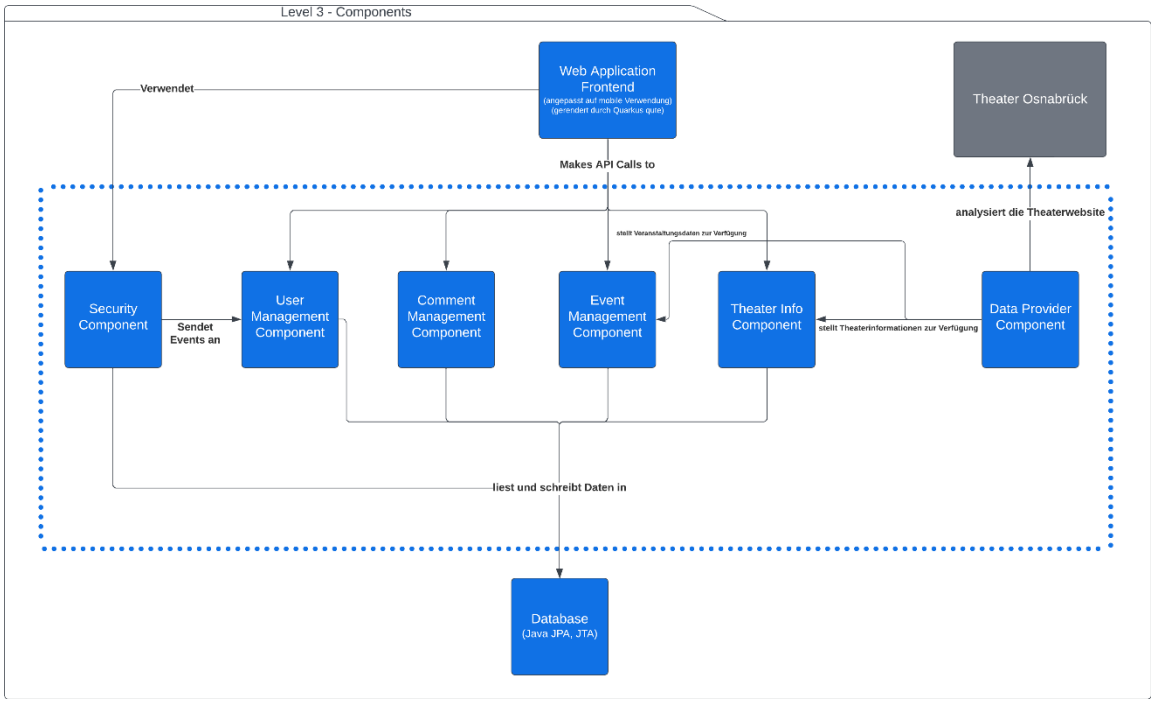

Abbildung 3: Components - Level 3

Das Modul **Usermanagement** kümmert sich um die Verwaltung von Nutzern, und ihren Objekten, wie die ids und status der gespeicherten Veranstaltungen, und die ids der Kommentare.

Davon ist das Modul **Security** strikt getrennt und kümmert sich um die Authentifizierung und Autorisierung von Anwendern, kommuniziert nur zur Erstellung von Usern über lose gekoppelte Events mit dem userdata-Modul und ermöglicht so eine einfache Austauschbarkeit und Wartbarkeit.

Das **eventmanagement** verwaltet die Veranstaltungen. Das **commentmanagement** verwaltet die Kommentare losgelöst von den anderen Modulen, um Abhängigkeiten zu verringern. **theaterinfo** stellt Informationen über das Theater zur Verfügung, wie die Öffnungszeiten der Theaterkasse. Letztendlich stellt die **DataProvider**-Komponente die Schnittstelle zur Theaterwebsite und liefert die Daten für das eventmanagement und das theaterinfo-modul.

Aus diesem dritten Schritt heraus wird im nächsten Schritt das UML-Klassendiagramm entwickelt. Dieses befindet sich in leicht vereinfachter Form im Anhang.

Gemäß des ecb-Patterns ist eine hierarchische Struktur von oben nach unten zu erkennen, die eine klare Trennung der Zuständigkeiten fördert. So bestehen die einzelnen Module von oben nach unten jeweils aus einem boundary- control- entity und einem zusätzlichen Gateway-Paket, welches sich um die Kommunikation mit der Datenbank kümmert. Aufgrund dessen, dass einige (Data-Transfer-)Objekte global verwendet werden, kommt noch ein **shared**-Package hinzu, um Codeduplikationen entgegenzuwirken. Hierbei ist zu beachten, dass ein ResponseWrapper gebildet wird, dessen Ziel es ist die JSON-Api-Spezifikationen v1.1, gemäß [@JNI] in einem für dieses Projekt angemessen Rahmen zu erfüllen.

Aufgrund der Komplexität und des zeitlichen Aufwands der anderen Module wird die detailliertere Planung und Implementation der Module commentmanagement und theaterinfo in diesem Projektbericht nicht weiter aufgeführt, sodass diese in den nachfolgenden Beschreibungen und Abbildungen nicht weiter auftauchen.

Weiterhin werden die in der Projektaufgabe beschriebenen REST-APIs entsprechend der zuvor ermittelten Modulaufteilung deutlich erweitert und angepasst, um zum einen der Webapplikation eine Schnittstelle zu ermöglichen und weiterhin den Anforderungen des Domain Driven Designs gerecht zu werden. Eine tabellarische Darstellung der REST-APIs inklusive einer kurzen Beschreibung befindet sich auf der folgenden Seite:

Tabelle 1: Mobile Ressourcen

mobile/events	GET	Ruft gefilterte und paginierte Veranstaltungen auf
mobile/events/{eventId}	GET	Ruft eine Veranstaltung anhand seiner ID via PathParam für auf
mobile/login	GET	Ruft die Login-Seite auf.
mobile/performances	GET	Ruft gefilterte und paginierte Aufführungen auf.
mobile/register	GET	Ruft die Registrierungsseite auf.
mobile/register	POST	Registriert einen neuen Benutzer.
mobile/user/usere-vents/{eventId}	GET	Ruft das UserEvent für eine Veranstaltung anhand der Veranstaltungs-ID via PathParam ab.
mobile/user/usere-vents/{eventId}	POST	Aktualisiert das UserEvent für eine Veranstaltung anhand der Veranstaltungs-ID via PathParam

Tabelle 2: API Ressourcen

Pfad	Methode	Beschreibung
api/admin/events/{eventId}	PUT	Aktualisiert ein Ereignis anhand der Ereignis-ID via PathParam.

	DELETE	Löscht ein Ereignis anhand der Ereignis-ID via PathParam.
api/events	GET	Ruft gefilterte und paginierte Ereignisse ab.
api/events/{eventId}	GET	Ruft ein Ereignis anhand der Ereignis-ID via PathParam ab.
api/events/{eventId}/performances	GET	Ruft Aufführungen für ein Ereignis anhand der Ereignis-ID via PathParam ab.
api/performances	GET	Ruft gefilterte und paginierte Aufführungen ab.
api/performances/{performanceId}	GET	Ruft eine Aufführung anhand der Aufführungs-ID via PathParam ab.
api/performances/{performanceId}/calendar	GET	Ruft den Kalendereintrag als Text/Calendar für eine Aufführung anhand der Aufführungs-ID via PathParam ab.
api/user/userevents	GET	Ruft alle UserEvents für einen Benutzer ab.
	POST	Erstellt ein neues UserEvent.
api/user/userevents/{userEventId}	GET	Ruft ein UserEvent anhand der UserEvent-ID via PathParam ab.
	PUT	Aktualisiert ein UserEvent.
	DELETE	Löscht ein UserEvent.
	PATCH	Ändert entweder isFavorite oder EventState für ein UserEvent.
api/user/userperformance	GET	Ruft alle UserPerformances für einen Benutzer ab.
	POST	Erstellt eine neue UserPerformance.
api/user/userperformance/{id}	GET	Ruft alle UserPerformances für einen Benutzer ab.
	PUT	Aktualisiert eine UserPerformance für einen Benutzer.
	DELETE	Löscht eine UserPerformance für einen Benutzer.
crawler/local	GET	Crawlt Ereignisse von einer Website und aktualisiert die Datenbank.
crawler/web	GET	Crawlt Ereignisse von einer Website und aktualisiert die Datenbank.

3.2 Implementation

In diesem Kapitel werden nun einige wichtige Aspekte und Herausforderungen näher beleuchtet. Dabei wird zunächst auf allgemeingültigere Punkte eingegangen, woraufhin projektspezifischere Punkte genauer besprochen werden.

3.2.1 Konfiguration, OpenAPI und Logging

In der application.properties sind einige quarkus-spezifische Konfigurationen hinterlegt. So werden die Pfade für http-requests festgelegt.

Weiterhin werden einige OpenAPI-Definitionen gesetzt und die Database Konfiguration vorgenommen.

Hierbei wird die Datenbank auf postgresql festgelegt, sowie die Generation der Datenbank auf drop-and-create gesetzt. Bezüglich der authentifizierung per form-auth werden einige Einstellungen gesetzt um die form-auth mit `quarkus.http.auth.form.enabled=true` zu aktivieren, die landingpages zu definieren, die Namen der Formularfelder für die Authentifizierung zu definieren und einen Namen für den authentification-cookie zu setzen.

Abschließend werden die logs konfiguriert, die beispielsweise mit `Log.info("Dies ist ein Log")` in den jeweiligen Klassen aufgerufen werden.

3.2.2 Transaktionalität

Um eine konsistente Transaktionalität zu gewährleisten, wird sich über JTA an den @Transactional-Annotationen bedient.

Hierbei wird auf boundary-Ebene immer eine neue Transaktion mit der Annotation `@Transactional(Transactional.TxType.REQUIRES_NEW)` gestartet um sicherzustellen, dass keine bereits bestehende Transaktion verwendet wird und für jeden Datenbankzugriff eine neue Transaktion gestartet wird.

Auf Ebene der Repositories wird die auf der Boundary-Ebene erstellte Transaktion mit der Annotation
`@Transactional(Transactional.TxType.MANDATORY)`
betreten um sicherzustellen, dass auf Boundary-Ebene eine Transaktion erzeugt wird. Andernfalls wird eine InvalidTransactionException geworfen und die Transaktion zurückgesetzt.

3.2.3 FaultTolerance

Um die Gesamtstabilität der Anwendung zu gewährleisten, werden verschiedene Strategien zur Fehlertoleranz angewendet. Darunter fällt das Retrying, also das Wiederholen von Operationen, Timeout (Zeitüberschreitung), Fallback, also ein Ersatzverhalten, falls eine Methode nicht innerhalb des Timeout Rahmens ohne Probleme abgeschlossen wird und ein Circuit-Breaker, der eine Art Leitungsschutzschalter-Mechanismus konfiguriert. Dieser überwacht die Anzahl der fehlerhaften Aufrufe einer Methode und entscheidet nach den gegebenen Kriterien, ob die Methode weiterhin aufgerufen oder auf eine Fallback-Methode umgeleitetet werden

muss um eine Überlastung oder Ausfall einer externen Resource zu vermeiden. In diesem Projekt wurde dieses Verhalten umgesetzt, indem jeder http-Request mit den folgenden Annotationen versehen wurde:

```
@Retry
@Timeout(5000)
@Fallback(fallbackMethod = "getEventsFallback")
@CircuitBreaker(requestVolumeThreshold = 4, failureRatio = 0.75, delay = 10000)
```

Snippet 1: Fault Tolerance

Dabei gibt @Retry an, wie häufig eine Fehlgeschlagene Methode wiederholt werden darf, wobei “3” der Standardwert ist, weshalb er nicht explizit gesetzt wird. @Timeout gibt an, dass die Methode innerhalb von 5 Sekunden abgeschlossen werden sollte, bevor die anderen Mechanismen greifen. @Fallback, gibt die Methode an, die im Falle eines Fehlverhaltens aufgerufen wird. Der @CircuitBreaker spezifiziert mit seinen Parametern, wie viele Aufrufe überwacht werden sollen, failureRatio den Prozentsatz fehlerhafter Aufrufe, und delay definiert die Zeitspanne, nachdem der CircuitBreaker nach einer Auslösung erneut geprüft wird.

Dabei ist zu beachten, dass dies relativ gängige Werte sind, man für den Production-State diese Werte aber gegebenenfalls genauer bestimmen und noch einmal anpassen sollte.

3.2.4 JSON-API-Spezifikation

Gemäß der json-API-Spezifikationen wird ein ResponseWrapper implementiert, der im folgenden Beispiel aus der EventResourceAPI gefüllt wird.

So existiert auf dem TopLevel entweder ein Array von errors, in diesem Fall beispielsweise, wenn die EventCollection leer ist, oder die primary Data, weiterhin kann auf dem TopLevel ein Links-Object mit in diesem Fall Paginierung befüllt werden. Data besteht hier aus einem Array von Resource-Objekten, die je einen String type und id enthalten, sowie das OutgoingEventDTOApi, als attributes, sowie ein selflink als linksDTO und ein relationshipObjekt, welches in diesem Fall die Performances dieser Relation sind. Zurückgegeben wird dann in diesem Beispiel mit einer pageSize von 1 folgendes Element:

```
Collection<Event> events = eventOperations.getEvents(queryParametersDTO);
        ResponseWrapperDTO<Object> responseWrapperDTO = new ResponseWrapperDTO<>();
        if(events.isEmpty()) {
            responseWrapperDTO.errors = new ArrayList<>();
            responseWrapperDTO.errors.add(new ErrorDTO("404", "EVENTS:1","No events found", "Couldn't find any
events with the given filters"));
            return Response.status(Response.Status.NOT_FOUND).entity(responseWrapperDTO).build();
        }

        responseWrapperDTO.data = events.stream()
                .map( play -> {
                    String id = String.valueOf(play.id);
                    String type = "play";

                    LinksDTO linksDTO = linkBuilder.createSelfLink(EventResourceApi.class, uriInfo, id);
                    OutgoingEventDTOApi outgoingEventDTOApi = OutgoingEventDTOApi.Converter.toDTO(play);
                    RelationshipDTO<Object> relationshipDTO = linkBuilder.addRelationship(EventResourceApi.class,
uriInfo, play.id, "performances");
                    return new ResourceObjectDTO<>(id, type, outgoingEventDTOApi, relationshipDTO, linksDTO);
                })
                .toList();
        long maxSize = eventOperations.getEventsCount(queryParametersDTO);
        responseWrapperDTO.links = linkBuilder.createPaginationLinks(EventResourceApi.class, uriInfo, pageNumber,
pageSize, maxSize);
```

Snippet 2: Building Response Wrapper

Snippet 3: JSON-Response einer Page mit einem Element

3.2.5 Bean Validation, Authentification und Events in der Nutzerverwaltung

Das Security-Management ist lose mit dem Usermanagement gekoppelt, um bei Bedarf in der zukünftigen Planung dieses mit einer sicheren Authentifizierung auszutauschen.

Hier wurde konkret die form-authentification genutzt, die ähnlich wie die Basic Authentification mit Jakarta Persistence und der Annotation @UserDefinition als PanacheEntity implementiert wird. Hierbei wird ein User mit einem username, einem password, welches mit BcyrptUtil gehashed wird, und einer Role, je als String in der Datenbank persistiert.

Mit der Startup-Klasse werden zum Start von Quarkus zu Testzwecken alle Nutzer aus dem System gelöscht und ein Nutzer “user” mit gleichnamigem Passwort und gleichnamiger Role, sowie ein “admin” persistiert.

Die Registrierung ist über /mobile/register aufrufbar und mit einem Klick auf den Button “Registrieren” wird der Nutzer der Datenbank hinzugefügt und ein registerEvent vom Typ `javax.enterprise.event.Event<String>` gefired, welches vom usermanagement in der Boundary mit der Annotation `@Observes String username` observed wird, woraufhin der user im usermanagement ebenfalls angelegt wird.

Nach erfolgter Registrierung wird auf die Seite `/mobile/login` weitergeleitet, woraufhin man nach Bestätigung des Loginbuttons per formauth über quarkusinterne Mechanismen per Post-Request auf dem Pfad `“j_security_check”` angemeldet wird.

Weiterhin werden Username und Password mithilfe der BeanValidation wie folgt validiert, sodass invalide Anfragen mithilfe des Hibernate Validators abgefangen werden.

```
@Valid @Size(min=3, max=25)
@Pattern(regexp = "^[a-zA-z ]*$")
@FormParam("username") String username,
@Size(min=8, max=20)
@Pattern(regexp = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=!])(?=\\S+$).{8,}$", message = "the password has to
contain at least one uppercase letter, one lowercase letter, one number and one special character")
@FormParam("password") String password
```

Snippet 4: Bean Validation

3.2.6 User-Management

Das Usermanagement kümmert sich primär um das Setzen und Abfragen von Event- und Performance-Status, sowie um das Favorisieren von Veranstaltungen.

Hierbei wird bei Aufrufen eines POST-Requests der UserEventResourceMobile, je nachdem ob der EventState oder isFavorite auf eine eventId gesetzt ist, das jeweilige Element geupdatet und auf die Detailansicht weitergeleitet.

```
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Path("{eventId}")
    public Response updateEventStatebyEventIdOfUser(@PathParam("eventId") long eventId, @FormParam("isFavorite")
boolean isFavorite, @FormParam("eventState") EventState eventState, @Context SecurityContext securityContext){
        String username;
        int active=3;
        if(securityContext== null || securityContext.getUserPrincipal() == null){
            return Response.seeOther(URI.create("/mobile/login")).build();
        }
        username = securityContext.getUserPrincipal().getName();
        if(eventState!= null)
            userDataOperations.updateEventStatebyEventIdOfUser(eventId, eventState, username);
        else
            userDataOperations.updateIsFavoritebyEventIdOfUser(eventId, isFavorite, username);
        return Response.seeOther(URI.create("/mobile/events/"+eventId)).build();

    }
```

Snippet 5: Update von UserEvent

3.2.7 Event-Management

Das Eventmanagement bietet wie in dem UML-Klassendiagramm ersichtlich Ressourcen für die Abfrage von Events an. Hierbei sind die Resourcen für die Optimierung auf html-Seiten zu denen für die Rückgabe von json zu unterscheiden.

Unter den json-Resourcen gibt es auch eine AdminResource, die das Ändern und Löschen von Veranstaltungen in der Rolle Admin erlaubt.

Die json-Resourcen richten sich dabei nach den der json-api-Spezifikation.

Die Mobile Ressourcen enthalten für ihre Funktionalitäten angepasste DTOs, sodass möglichst wenig overhead generiert wird.

Ein wichtiger Aspekt ist die Generierung von Calendarfiles des Mime-typ text/calendar. Hierbei wird die library iCal4J verwendet, um eine Kalenderdatei des Typs .ics zu generieren. Dabei werden die Daten title location und date verwendet, um einen Kalender mit dem Kalenderevent zu generieren.

Daraufhin wird, wie im folgenden codeSnippet zu sehen ist, der Header der Response angepasst, sodass, die Kalenderdatei den Titel der Veranstaltung als Dateinamen annimmt.

```
Performance performance = optionalPerformance.get();
        String eventName = performance.getEvent().getTitle();
        Location eventLocation =new Location(performance.getEvent().getLocation());
        Date date = new Date(performance.getDatetime().toEpochSecond(java.time.ZoneOffset.UTC));
        VEvent vEvent = new VEvent(date, eventName);
        vEvent.getProperties().add(eventLocation);
        calenderFile.getComponents().add(vEvent);
        String fileName = eventName.replaceAll("\\s+", "_") + ".ics";
        String calendarContent = calenderFile.toString();
        return Response.ok(calendarContent)
                .header("Content-Disposition", "attachment; filename=\"" + fileName + "\"")
                .build();
```

Snippet 6: Erstellung eines CalendarEvents

3.2.8 Data-Provider

Das Package Data-Provider bildet die Schnittstelle zur Theaterwebsite. Hierbei wird in der Resource CrawlerResource die Kalenderseite des Theater Osnabrück mithilfe der jsoup-library abgerufen.

Daraufhin werden die heruntergeladenen Daten im CrawlerService, in dem durch eine Liste an Kalenderelementen iteriert wird verarbeitet, sodass alle wichtigen Elemente aus dem Kalender gespeichert und hinterher mit einem Aufruf des AddEventsCatalog mithilfe von CalendarElementDTOs in der Datenbank persistiert werden. An die Resource wird daraufhin ein Set an stids: String zurückgegeben, worüber sich die einzelnen Stücke auf der Theaterwebsite identifizieren.

Anschließend werden für die einzelnen Stücke die Detailinformationen ebenfalls abgerufen. Hierbei werden auch die Bilder der Website abgerufen und lokal im /media-Ordner gespeichert, und per Angabe des Pfades, über die Methoden des Service ebenfalls an den Catalog persistiert. Nach Abschluss dieser Funktionen, wird eine Meldung zurückgeliefert, wie viele Events aktualisiert wurden.

3.2.9 Qute

Qute wird verwendet, um die html-Seiten serverseitig zu rendern. Dabei wird ein Template im Ordner main/resources/templates als .html-Seite erstellt und per @Inject Template {templatename} in eine Resource injiziert. In der Resource wird dann eine TemplateInstance erstellt und dieser werden daraufhin die benötigten Daten übergeben:

```
TemplateInstance templateInstance = stuecke.data("events", outgoingEventDTOMobiles, "queryParameters",
queryParametersDTO, "active", active, "showMore", hasNextPage);
String html = templateInstance.render();
return Response.ok().entity(html).build();
```

Snippet 7: Rückgabe eines Qute Templates

So werden, wie im codesnippet, dem Template eine Liste an outgoingEventDTOMobiles übergeben, welche im Template dann über die bezeichnung “events” referenziert werden. Weiterhin wird dem Template noch ein active-Objekt übergeben, welches angibt, welches Icon in der Button-Navigation-Bar aktiv ist, sowie ein boolean showMore, der angibt, ob es eine folgende Seite gibt.

Html-seitig gibt es neben dem gerade erwähnten Template stuecke, noch ein base.html-Template, welches dafür sorgt, dass Elemente, die auf mehreren Seiten verwendet werden, nicht dupliziert werden müssen, sondern einfach über die eingebunde base.html-Seite in das bestehende Template injiziert werden. So wird unter anderem einmalig eine BottomNavigationBar in base.html erstellt, die über alle weiteren Templates dann verwendet werden kann. Weiterhin werden über die base.html alle benötigten stylesheets und javascript-Dokumente eingebunden.

In der stuecke.html-Seite werden dann Ausdrücke mit geschweiften Klammern gesetzt, die auf die übergebenen Objekte referenzieren.

Entsprechend des Designs aus der Projektaufgabe werden dann die einzelnen Seiten via html und css erstellt und die variablen Elemente über Referenzierungen, wie in folgendem Code-Snippet zu sehen, angegeben.

```
{#for event in events}
<a class="card" href="/mobile/events/{event.id}">
    <div class="image" style="background-image: url('{#if event.thumbnailPath}{event.thumbnailPath}{#else}/media
/sg221017196.jpeg{/if}');"></div><!--todo use title attribute for alt-text-->
    <div class="basic-info">
        <div>
            <h2>{event.title}</h2>
            <span id="chip"
                  {#if event.eventState}
                { class="{event.eventState.cssClass} " > {event.eventState.getLabel}  {#else} class = "none-
state">Nicht gesetzt {/if}</span>
        </div>
        <h3>{#if event.kind}{event.kind}{/if}</h3>
        {#if event.nextPerformance}
        <h3>{#if event.nextPerformance.performanceType}
            {event.nextPerformance.performanceType}
            {#else} als nächstes {/if} am
            {event.nextPerformance.startDate}
            {#if event.nextPerformance.hasTime} um {event.nextPerformance.startTime} {/if}
            {/if}
        </h3>
    </div>
</a>
{#else}
        <h2>Keine Stücke gefunden</h2>
{/for}
```

Snippet 8: Qute Template mit Ausdrücken

3.2.10 Unpoly

Wie in den Grundlagen erwähnt kommt weiterhin noch unpoly zum Einsatz um einzelne Elemente auszutauschen ohne das komplette Dokument nachzuladen.

Dies wird als javascript-framework eingebunden und über wenige Annotationen eingesetzt.

So gibt es beispielsweise auf der stuecke-Seite unten einen Button, der die nächste Seite lädt. Dies geschieht über die folgende Form mit der Annotation up-target=”main”, sodass der main-Tag des html Dokuments mit den nachgeladenen Inhalten ersetzt wird. `<form class="showMore" action="/mobile/events" method="GET" up-target="main">` Auch bietet unpoly die Möglichkeit einzelne Fragmente als modals und popups darzustellen. Dies wird eingesetzt, um das “Filtern”, “Suchen” und “Status setzen”, oberhalb der eigentlichen View zu ermöglichen, wie in der folgenden Abbildung visualisiert ist.

Dies wird mit folgenden Tags ermöglicht.

```
<a href="/filters.html" up-layer="new" up-mode="modal">
```

Die statische html-Seite startet dann einen GET-Requests mit den ausgewählten, als Buttons visualisierten Checkboxen an `/mobile/events` und lässt so wieder den Inhalt aktualisieren.

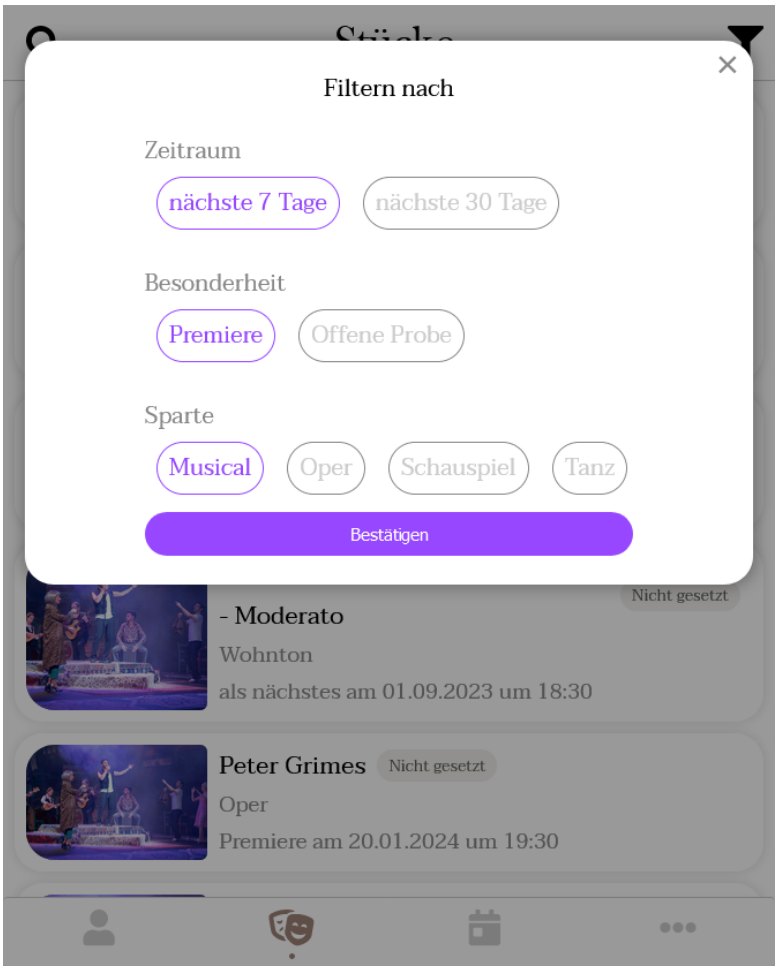

Abbildung 4: Filterfunktionalität

4 Validierung

Nachdem im vorigen Kapitel die Projektplanung und Entwicklung ausführlich beschrieben wurde, werden die Ergebnisse in diesem Kapitel unter Bezugnahme der aus der Projektaufgabe hervorgehenden Kriterien abschließend validiert, dazu werden zunächst alle Musskriterien noch einmal tabellarisch aufgeführt, mit einem entsprechenden Haken angekreuzt und mit einem Kommentar selbstkritisch reflektiert.

Musskriterien
Als (nicht-) registrierter Internetnutzer kann ich mir eine einzelne Veranstaltung anzeigen lassen ✓ ist efüllt.
Als (nicht-) registrierter Internetnutzer kann ich mir Aufführungszeiten eines ausgewählten Stücks anzeigen lassen ✓ ist erfüllt
Als nicht-registrierter Internetnutzer kann ich einen Account erstellen ✓ via forms
Als registrierter Internetnutzer kann ich mich am System anmelden ✓ via form-auth
Als angemeldeter Nutzer kann ich Veranstaltungen für mich speichern ✓ über das Herz-Icon
Als angemeldeter Nutzer kann ich für Veranstaltungen einen persönlichen Status setzen ✓ auf der Detail-seite.
Als angemeldeter Nutzer kann ich Aufführungszeiten für mich speichern. ✓ via api
Als angemeldeter Nutzer kann ich Veranstaltungen einen nur für mich sichtbaren Kommentar setzen. ✗ ist nicht implementiert.
Als angemeldeter Nutzer kann ich eine Übersicht über meine gespeicherten Theaterstücke sehen ✗ ist nicht implementiert.
Als angemeldeter Nutzer kann ich eine Übersicht über meine gespeicherten Aufführungszeiten sehen ✗ ist nicht implementiert.
Als (nicht-) registrierter Internetnutzer kann ich eine Übersicht über Veranstaltungen sehen ✓ ist erfüllt.
Als (nicht-) registrierter Internetnutzer kann ich eine Übersicht über Spielzeiten zukünftiger Theaterstücke sehen. ✓ ist erfüllt.

Wunschkriterien
Als (nicht-) registrierter Internetnutzer kann ich für die Übersicht zukünftiger Theaterstücke verschiedene Filter verwenden. ✓ Filtern nach Zeit, Titel, Sparte
Als (nicht-) registrierter Internetnutzer kann ich für die Übersicht über Spielzeiten zukünftiger Theaterstücke Filter verwenden. ✓ Filtern nach Zeit, Titel, Sparte und nach Besonderheit
Als (nicht-) registrierter Internetnutzer kann ich Theaterstücke mit anderen teilen. ✓ über den Link
Als (nicht-) registrierter Internetnutzer kann ich Spielzeiten mit anderen teilen. ✓ über den Link einer Detailseite, oder eine Kalenderdatei
Als registrierter Internetnutzer kann ich Theaterstücken einen öffentlichen Kommentar hinzufügen. ✗ Kommentarfunktion wurde nicht implementiert.
Als angemeldeter Nutzer kann ich meinen Account löschen. ✗ Wurde nicht implementiert.
Als (nicht-) registrierter Internetnutzer kann ich mir alle zuvor genannten Kriterien in einem übersichtlichen User Interface anschauen. ✓ (Fast) Alle implementierten Funktionen lassen sich visuell anzeigen.
Als (nicht-) registrierter Internetnutzer kann ich mir die Spielzeit eines Theaterstücks in einen Kalender exportieren lassen. ✓
Als (nicht-) registrierter Internetnutzer kann ich mir Informationen über die Theaterkasse anzeigen lassen. Hierzu gehören die Öffnungszeiten, die Adresse und eine Telefonnummer. ✗ theaterinfo-modul wurde nicht implementiert.

Als (nicht-) registrierter Internetnutzer kann ich die Datenbank der Theaterstücke vom System aktualisieren lassen. ✓ CrawlerResource ist derzeit nicht durch autorisierung geschützt.

Abgrenzungskriterien
Die Theaterapp soll nicht die Buchungsfunktion von Tickets abbilden können. ✓ Es gibt aber einen Link zur Buchungsseite.
Im Fokus steht nicht die Vielfalt sondern die Qualität der angebotenen Funktionalität. ✓ Einige Funktionen wurden nicht implementiert, dies liegt aber auch durch die begrenzte Zeit begründet.
Es werden zwei Schnittstellen zur Interaktion bereitgestellt, konkret: Maschine-Maschine Interaktion über eine REST-API, damit eine spätere Implementation einer Nutzeroberfläche erfolgen kann. Mensch-Maschine Interaktion über eine Web-App unter Verwendung von Quarkus Qute als SSR-Ansatz, damit Theaterfreunde den Planer Online einfach und angemessen über den Browser nutzen können. ✓ Es wurden beide Schnittstellen implementiert.

5 Zusammenfassung und Fazit

In diesem Kapitel werden nun die in den letzten Kapiteln entstandenen Ergebnisse noch einmal zusammengefasst, abschließend gibt es noch ein Fazit und einen kurzen Ausblick auf den weiteren Projektverlauf nach Abgabe der Hausarbeit.

5.1 Zusammenfassung

Die vorliegende Arbeit widmete sich der Entwicklung einer Theaterapp, die Theaterbegeisterten dabei helfen soll, ihre Theaterbesuche zu planen und Informationen zu zukünftigen Aufführungen schnell und übersichtlich zu finden.

Das Ziel der Arbeit war es, eine softwarearchitektonische Lösung zu entwickeln, welche das Domain-Driven-Design als architektonisches Paradigma nutzt. Die Technologieplattform bildeten Rest-APIs mit Quarkus um die Interaktionen mit anderen Systemen zu ermöglichen. Weiterhin wurde Web-Crawling verwendet, um Daten von der Website des Theater Osnabrück zu analysieren und in eine Datenbank einzupflegen, die als Grundlage für die geplante App diente.

Das Server-Side-Rendering mit Quarkus Qute mit Unterstützung von Unpoly sorgte für eine angemessene Web-App, die den Theaterplaner im Browser zugänglich macht.

Die Ergebnisse der Arbeit wurden unter Berücksichtigung der definierten Muss-, Wunsch und Abgrenzungskriterien selbstkritisch validiert, was zu einer fundierten Bewertung der Softwarelösung führte.

5.2 Fazit

In dieser Projektarbeit ist es gelungen einen Theaterplaner zu entwickeln, der Nutzern dabei helfen kann Theaterbesuche effizienter zu planen und relevante Informationen leichter zu finden. Die Verwendung des Domain-Driven Design ermöglichte weiterhein eine klare Strukturierung der Software und fördert die Flexibilität und Wartbarkeit.

Die Nutzung der Technologien bezüglich Transaktionalität, Fault-Tolerance, Bean-Validation, der JSON-API-Spezifikation, der losen Kopplung mithilfe von Events, sowie die Nutzung von JPA führte zu einem soliden Endergebnis, welches im Großen und Ganzen Level 3 des Richardson Maturity Model entspricht.

Es ist jedoch zu beachten, dass aufgrund der Komplexität des Projektes und einigen zunächst noch unbekannten Technologien es während der Planung und Entwicklung immer wieder zu Anpassungen und Umstrukturierungen kam, woraufhin einige Module aufgrund der kurzen Zeit, wie in der Validierung reflektiert, nicht wie geplant umgesetzt worden sind. So wurden zwar viele Wunschkriterien umgesetzt, aber nicht alle der genannten Musskriterien. Auch ist die Lose Kopplung nicht über alle Module hinweg umgesetzt worden. Im Großen und Ganzen war das Projekt aber ein großer Meilenstein für die zukünftige Weiterentwicklung.

5.3 Ausblick

Da die Entwicklung einer Softwarearchitektur ein kontinuierlicher Prozess ist, bei dem immer neue Anforderungen dazukommen, ist auch nach dem letzten Entwurf dieser Hausarbeit weitere Entwicklungsarbeit geplant.

So werden zunächst die aus der Validierung hervorgegangenen Problemstellungen weiterentwickelt und abgeschlossen.

Weiterhin wird die Authentifizierung mit einem sichereren Verfahren ausgetauscht, sodass in den kommenden Monaten ein solides Produkt auf einem Server unter der bereits reservierten Domain theater-os.de gehostet werden kann. Nach ersten Erfahrungen in der Nutzung in Verbund mit einer kürzeren Testphase mit Freunden und Kommilitonen wird angestrebt, das Theater Osnabrück bezüglich einer Kooperation anzufragen.

6 Referenzen

Web-Seiten zuletzt am 24.07.2023 abgerufen.

[@DU] Definition von Theater https://www.duden.de/rechtschreibung/Theater

[@FR] Definition von Theater, https://www.fremdwort.de/suchen/bedeutung/theater

[@Noz1] Theater Osnabrück: Nächste Spielzeit widmet sich dem Gastland Polen https://www.noz.de/lokales/osnabrueck/artikel/theater-osnabrueck-spielzeit-2324-widmet-sich-dem-gastland-polen-44735934

[@C4M] The C4 model for visualising software Architecture https://c4model.com/

[@RH1] Quarkus https://www.redhat.com/de/topics/cloud-native-apps/what-is-quarkus

[@JS] Java HTML Parser, https://jsoup.org/

[@IC4J] iCal4j, iCalendar parser and object model, https://github.com/ical4j/ical4j

Sonstige Quellen:

[Sim, 2007] Peter Simhandl, “Theatergeschichte in einem Band”, Aktualisierte Neuaufl., 3., überarb. Aufl., Henschel Verlag, 2007