Designentwurf

HOR-TINF2021

Gruppe

Tobias Hahn
Lars Holweger
Fabian Unger
Timo Zink
Frank Sadoine
Fabian Eilber

Betreuer

Prof. Dr. Phil. Antonius van Hoof

Dokumentverlauf

Version	Beschreibung/Änderung	Autor	Datum
0.1	Verfassen von Kapitel 1 und 2	Fabian Unger	05.04.2023
0.2	Verfassen von Kapitel 4.1.	Tobias Hahn	05.04.2023
0.3	Hinzufügen der Architektur, Technologien, Frameworks, Komponenten und Kommunikation	Fabian Unger	06.04.2023
0.4	Hinzufügen der Datenbankstruktur	Lars Holweger	06.04.2023
0.5	Überarbeitung des Kapitels 4.1.	Tobias Hahn	07.04.2023
0.6	Hinzufügen der einzelnen Packages in 4.2. mit passenden Erklärungen	Fabian Eilber	08.04.2023
0.7	Verfeinerung einzelner Erläuterungen und Text und Qualitätsüberprüfung	Fabian Unger	09.04.2023
1.0	Hinzufügen der Dateiverweise	Fabian Unger	09.04.2023
2.0	Änderungsvorschläge vom 11.04. eingebaut	Fabian Unger	12.04.2023
3.0	Änderungen zu den Anforderungen aus dem Designreview	Fabian Unger, Lars Holweger, Fabian Eilber	16.04.2023

Inhaltsverzeichnis

Dokumentverlauf 1
Inhaltsverzeichnis 1
Abbildungsverzeichnis 2
Diagrammverzeichnis 2
1. Allgemeines 4
1.1. Designprozess 4
2. Produktübersicht 4
3. Grundsätzliche Struktur- und Entwurfsentscheidungen 5
3.1. Design-Architektur 5
3.2. Technologien und Frameworks 6
3.3. Datenbankstruktur 8
3.4. Überblick über Komponenten 8
3.5. Kommunikation zwischen Client und Server 11
4. Struktur- und Entwurfsentscheidungen zu Komponenten 14
4.1. User-Interface 14
4.2. Services 21

4.3. Kommunikation zwischen den Services 23
4.4. Controller 25
4.4.1 User Controller 25
4.4.2 Atendee Controller 26
4.4.3 Tutor Controller 26
4.4.4 Organizer Controller 27
4.4.5 Admin Controller 28
4.4.6 System Admin Controller 28
Dateiverweise 29

Abbildungsverzeichnis

Abbildung 1: GUI-Konzept – Hauptseite [6] 16
Abbildung 2: GUI-Konzept – Eventkatalog [7] 17
Abbildung 3: GUI-Konzept – Eventkarte [8] 19
Abbildung 4: GUI-Konzept – Event Guide [9] 20

Diagrammverzeichnis

Diagramm 1: Gesamtarchitektur der Anwendung [1] 5
Diagramm 2: Relationales Modell der Datenbank [2] 8
Diagramm 3: Komponentendiagramm der Anwendung [3] 9
Diagramm 4: HTTP POST Request für das Speichern von Daten [4] 11
Diagramm 5: HTTP POST Request für das Laden von Daten [5] 12
Diagramm 6: Aktivitätsdiagramm zur RESTful-API [11] 13
Diagramm 7: Zustandsdiagramm zur gesamten Anwendung [12] 14
Diagramm 8: Zustandsdiagramm - Startseite 15
Diagramm 9: Zustandsdiagramm - Hauptseite 15
Diagramm 10: Zustandsdiagramm - Eventkatalog 16
Diagramm 11: Zustandsdiagramm - Meine Events 17
Diagramm 12: Zustandsdiagramm - Verwaltung 18
Diagramm 13: Zustandsdiagramm - Organisationskatalog 18
Diagramm 14: Zustandsdiagramm - Eventkarte 19
Diagramm 15: Services-Komponente [10] 22
Diagramm 16: Sequenzdiagramm über die für einen User verfügbaren Events [13] 23
Diagramm 17: Sequenzdiagramm zum Anmelden zu einem Event [14] 24
Diagramm 18: Sequenzdiagramm zum Hinzufügen eines Users zu einer Gruppe [15] 24
Diagramm 19: Klasse UserController [16] 25
Diagramm 20: Klasse AttendeeController [17] 26

Diagramm 21: Klasse TutorController [18] 26
Diagramm 22: Klasse OrganizerController [19] 27
Diagramm 23: Klasse AdminController [20] 28
Diagramm 24: Klasse SystemAdminController [21] 28

1. Allgemeines

Mit Hilfe des Designentwurfs soll eine genau definierte Vorstellung der Architektur, Struktur und des Verhaltens der zu entwickelnden Anwendung geschaffen werden. Der Designentwurf dient als Grundlage für die Implementierung der Anwendung. Dies soll dabei helfen, potenzielle Probleme und Schwachstellen frühzeitig zu erkennen und zu lösen. Darüber hinaus sind durch die Erstellung des Designentwurfs die Struktur und das Verhalten der Anwendung von Anfang an klar definiert. Dies hilft zum einen bei der Implementierung als Team, da so alle Entwickler denselben Vorgaben folgen und bereits definiert ist, wie deren Komponente entwickelt und in die Anwendung eingebunden werden soll. Zum anderen unterstützt dies auch das Verständnis zwischen den Entwicklern und anderen Beteiligten, da durch den Designentwurf alle das gleiche Verständnis dafür haben, wie die Anwendung aussehen und funktionieren soll. Der Designentwurf hilft jedoch nicht nur während der Entwicklung, sondern auch nach der Fertigstellung der Anwendung. Da durch den Designentwurf der Aufbau der Anwendung klar definiert ist, lassen sich mögliche Fehler schneller finden und beheben, sodass die Anwendung dadurch auch einfacher zu warten sein wird.

Vorgaben für den Designentwurf sind die Gliederung dieses Dokuments, sowie die Erstellung von UML-Diagrammen für eine statische und dynamische Sicht auf die Anwendung. Unterstützt werden die Diagramme durch schriftliche Erläuterungen.

1.1. Designprozess

Der Designprozess der Anwendung hat bereits mit der Definition der Anforderungen begonnen. Auf Grundlage dieser Anforderungen wird nun der Designentwurf entwickelt. Hierzu gehört die Festlegung der Architektur, Technologien, Frameworks, sowie die Interaktionen zwischen den verschiedenen Komponenten der Anwendung. Des Weiteren werden die einzelnen Komponenten selbst mit deren Inhalt und Beziehung untereinander entwickelt. Anschließend werden eine Bewertung und Überprüfung durch Kunden und Betreuer erfolgen. Dabei gilt es sicherzustellen, dass die Designentscheidungen den Anforderungen entsprechen und bei möglichen Problemen, diese zu beheben.

2. Produktübersicht

Die zu entwickelnde Anwendung ist ein Online-Event-Management-System. In diesem System können Mitglieder einer Organisation Events erstellen und planen. Events werden dabei verschiedene Daten, wie beispielsweise Name, Ort, Zeit zugewiesen. Zur Planung gehört unter anderem die Teilnehmerverwaltung. Die Teilnehmerverwaltung ermöglicht das Einladen von Personen zu dem Event, Nachrichten an diese zu schicken, Dateien zu teilen, eine Anwesenheitsliste während dem Event zu führen und vieles mehr. Alle Funktionalitäten der Anwendung lassen sich im Pflichtenheft des Projekts nachlesen.

Zu den Zielgruppen gehören unter anderem Organisationen mit deren Mitarbeitern, die Seminare oder andere Veranstaltungen für ihre Mitarbeiter planen. Die Veranstaltungen können dann über die Anwendung vorbereitet, geplant und Mitarbeiter eingeladen werden können und diese dann weitere Informationen zum Event abrufen können.

3. Grundsätzliche Struktur- und Entwurfsentscheidungen

In diesem Kapitel werden die Struktur- und Entwurfsentscheidungen zur gesamten Anwendung erläutert. Dazu gehören die Architektur der Anwendung, die zugrunde liegenden Technologien und Frameworks, sowie die Interaktion verschiedener Komponenten. Dabei werden einzelne Erläuterungen grafisch durch UML-Diagramme unterstützt.

3.1. Design-Architektur

Die Design-Architektur soll eine Darstellung der Gesamtarchitektur widerspiegeln. Diese setzt sich aus den Schichten Präsentations-, Steuerungs-, Geschäfts- und Datenbankzugriffsschicht in Verbindung mit der Datenbank zusammen. In den Schichten selbst werden Komponenten dargestellt, welche in den jeweiligen Schichten agieren. Im Folgenden Diagramm 1 wird die Architektur dargestellt.

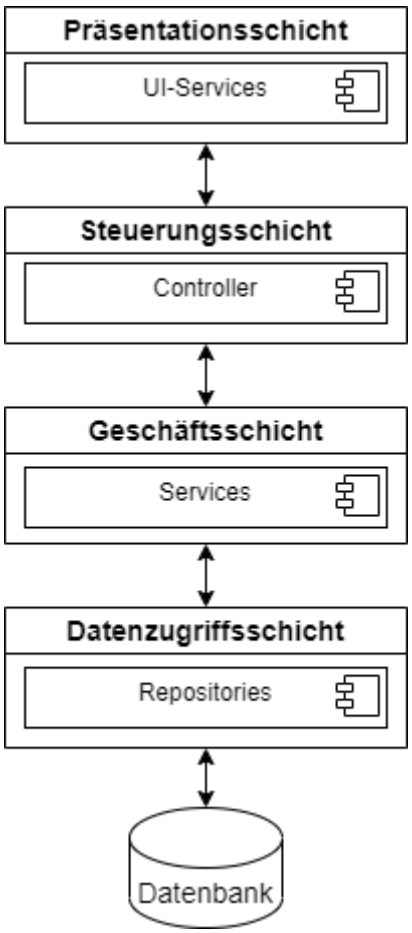

Diagramm 1: Gesamtarchitektur der Anwendung [1]

Die ***Präsentationsschicht*** beinhaltet die Komponenten, die mit Angular erstellt werden. Sie beinhalten zum einen die grafische Darstellung der Anwendung selbst und der Daten, die sie durch die Steuerungsschicht erhalten. Zum anderen sind sie auch für die Verarbeitung von Benutzeranfragen und -eingaben verantwortlich. Diese werden über HTTP POST Requests an die Steuerungsschicht gesendet.

Die **Steuerungsschicht** dient als Schnittstelle zwischen der Präsentationsschicht und der Geschäftsschicht. Sie empfängt die HTTP Requests der Präsentationsschicht, verarbeitet diese und leitet die darin enthaltenen Daten an die Geschäftsschicht weiter. Andersrum erhält sie Daten aus der Geschäftsschicht und verarbeitet diese zu HTTP Requests, um die Daten an die Präsentationsschicht weiterzuleiten.

Die **Geschäftsschicht** enthält die Geschäftslogik der Anwendung. Dabei ist sie die Schicht, die für die letztendliche Verarbeitung und Manipulation von Daten zuständig ist. In ihr sind die verschiedenen Services mit unterschiedlichen Aufgabenbereichen enthalten. Zu diesen gehören unter anderem Services zur Verwaltung von Events, Usern, Organisationen uvm. Dadurch sollen die Controller möglichst so frei von Logik gehalten werden, dass diese die Anfragen der Präsentationsschicht lediglich an die Services der Geschäftsschicht weiterleiten. Dies macht nicht nur die Entwicklung einfacher sondern auch die Anpassbarkeit.

Die **Datenzugriffsschicht** stellt den Zugriff auf die **Datenbank** bereit. Sie beinhaltet Repositories, die Funktionen zur Ausführung von Operationen zu der Datenbank bereitstellen. Zu diesen Operationen gehören die CRUD-Operationen. CRUD ist die Abkürzung für Create, Read, Update und Delete. Das bedeutet, dass dadurch bereits alle notwendigen Operationen zur Erstellung, zum Lesen, zum Aktualisieren und zum Löschen von Datenbankeinträgen bereitgestellt werden.

3.2. Technologien und Frameworks

Aus der Design-Architektur geht hervor, dass sowohl für die Benutzeroberfläche als auch für die Logik und Datenhaltung geeignete Technologien und Frameworks gewählt werden sollten. Deshalb werden im Folgenden für die jeweiligen Schichten geeignete Technologien und Frameworks mit deren Einsatz bei der Entwicklung der Anwendung erläutert.

Angular

Angular ist ein Framework zur Entwicklung von clientseitigen Webanwendungen mit TypeScript und JavaScript. Vorteile die für die Verwendung von Angular für die clientseitige Webentwicklung sind:

Aufgrund der Vorteile von Angular (Produktivität, Modularität, Routing, Datenbindung uvm.) und den bereits vorhandenen Kenntnissen im Team dazu, entscheiden wir uns für Angular. Des Weiteren muss die Implementierung in recht kurzer Zeit durchgeführt werden. Deshalb und da mehrere Entwickler gleichzeitig an der Implementierung arbeiten werden, entscheiden wir uns aufgrund der Vorteile von Angular und einigen Kenntnissen im Team für dieses Framework im Frontend.

RESTful-API

Die REST-API macht den Austausch von Informationen zwischen Benutzeroberfläche und Datenbank möglich. Besonders im Zeitalter von Computer und mobilen Endgeräten ist eine Benutzung von REST-APIs notwendig. Bei einer REST-API handelt es sich um eine Maschine-Maschine-Kommunikation. REST-APIs ermöglichen einen einheitlichen Informationsaustausch durch das Zusammenbringen von Systemen und Geräten. Die Informationen werden mithilfe

von HTTP-Requests ausgetauscht. Ein solcher HTTP-Request setzt sich zusammen aus einem definierten Endpoint und entsprechenden Parametern. Daraus lässt sich schließen, dass in der Anwendung für jeden Anfrage ein Endpoint definiert werden muss.

Da durch die RESTful-API Client und Server unabhängig voneinander sind, können diese auch unabhängig voneinander entwickelt und geändert werden. Somit können in der Gruppe sich Entwickler rein auf das Backend oder Frontend konzentrieren. Aus diesen Gründen wird für die Entwicklung dieser Anwendung die RESTful-API verwendet.

Spring Boot

Spring Boot ist ein Framework, das auf dem Spring Framework aufbaut. Es ist für die Entwicklung von Java-Anwendungen entwickelt worden und erleichtert die Erstellung von RESTful Web Services. Spring Boot bietet dabei eine Reihe von vorgefertigten Bibliotheken, Tools und Konventionen, die es Entwicklern ermögliche, sich auf die Geschäftslogik der Anwendung konzentrieren zu können. Zu den Tools von Spring Boot zählt unter anderem auch die Unterstützung verschiedener Datenbanken, einschließlich relationaler Datenbanken.

Durch Spring Boot wird die Erstellung der RESTful-API der Anwendung deutlich vereinfacht, da Annotationen verwendet werden können, um Endpunkte für HTTP Requests anzugeben. Da für die Anwendung die RESTful-API verwendet wird, wird für das Backend Spring Boot verwendet.

MySQL

MySQL ist eine Open-Source Relationale Datenbankverwaltungssystemsoftware, die eine zuverlässige Lösung für die Speicherung, Abfrage und Verwaltung von Daten bietet.

Da für diese Anwendung eine Datenbank von essenzieller Bedeutung ist, bietet es sich also an Spring Boot in Verbindung mit MySQL zu verwenden. Dadurch kann Spring Boot die Daten für MySQL bereitstellen, die dann von MySQL in einer Datenbank gespeichert werden. Die Java-Datentypen werden dabei durch Spring Boot in die entsprechenden MySQL-Datentypen übertragen.

3.3. Datenbankstruktur

Um die der Anwendung zugrunde liegenden Datenbank leichter verstehen zu können, wird im Folgenden ein relationales Datenbankmodell dargestellt. Auf diesem Modell basiert auch das Komponentendiagramm, da die jeweiligen Klassen durch die Frameworks automatisch in die Datenbank überführt werden.

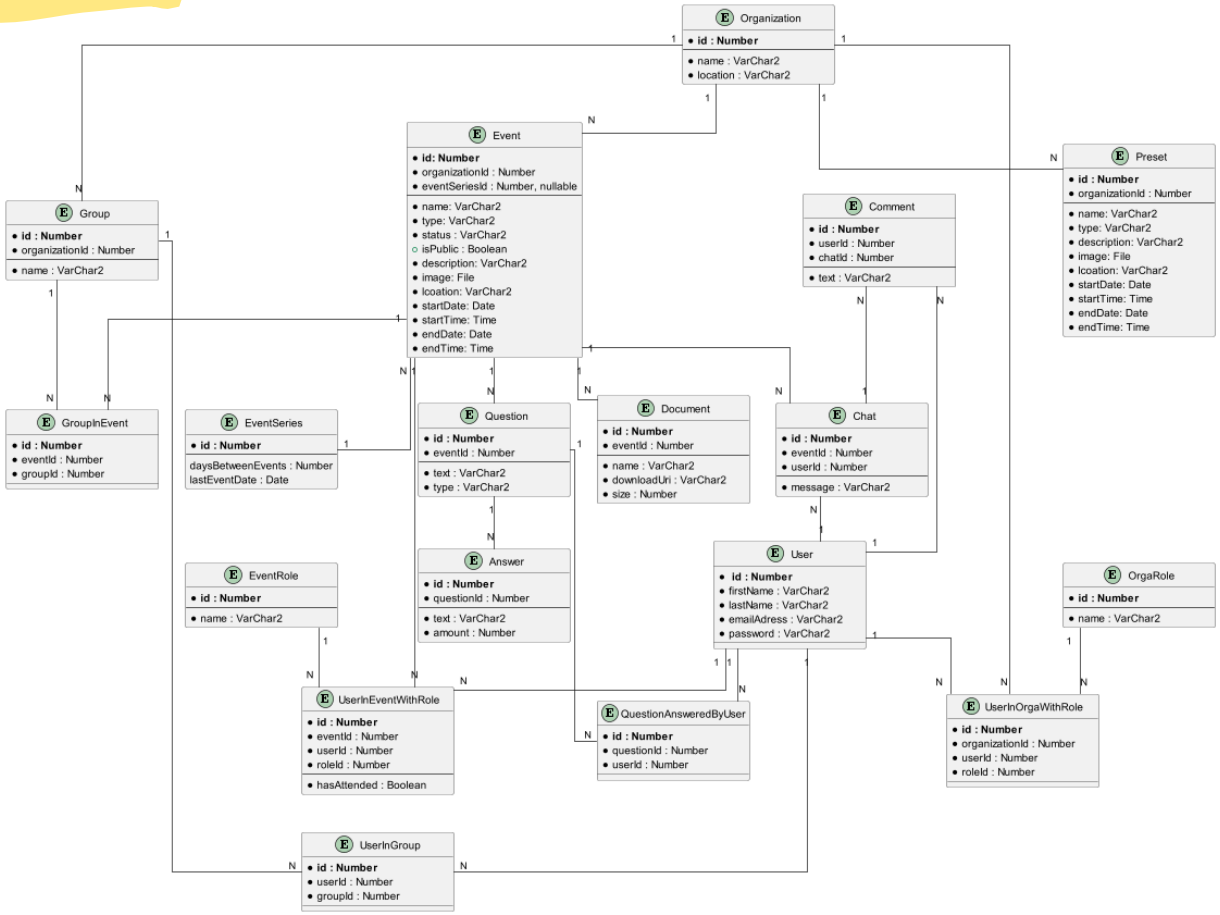

Diagramm 2: Relationales Modell der Datenbank [2]

Dieses relationale Datenbankmodell (Diagramm 2) bildet die Grundlage für die gesamte Anwendung. Da normalerweise viele N:M-Beziehungen zwischen verschiedenen Entitäten existieren (bspw. mehrere User können unterschiedlichen Events mit unterschiedlichen Rollen beitreten), haben wir uns dafür entschieden, bei solchen Beziehungen, weitere Entitäten dazwischen einzubauen (bspw. UserInEventWithRole). Da für das Backend Java Spring Boot eingesetzt wird, bildet dieses Datenbankmodell auch eine wichtige Grundlage für die Struktur und Beziehungen der Klassen. Dies bedeutet, dass diese Entitäten so als Klassen in das Backend eingebaut und mit Getter- und Setter-Methoden implementiert werden. Java Spring Boot wandelt dann die entsprechend gekennzeichnete Klassen in das Datenbankmodell um.

3.4. Überblick über Komponenten

Das folgende Komponentendiagramm (Diagramm 3) stellt den Aufbau der Anwendung unter Berücksichtigung der Gesamtarchitektur und der eingesetzten Technologien und Frameworks dar.

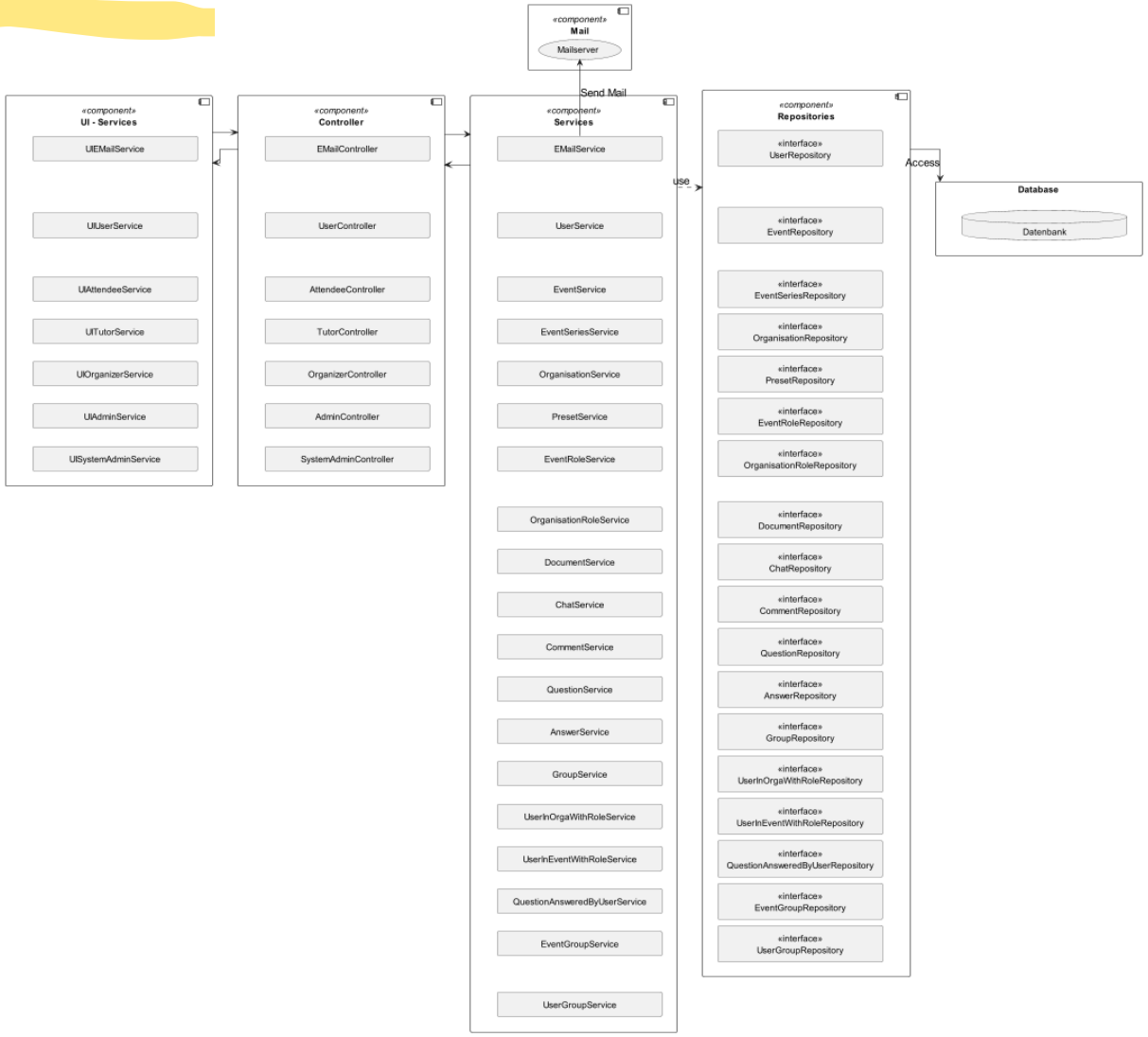

Diagramm 3: Komponentendiagramm der Anwendung [3]

Durch das Diagramm ist ersichtlich, dass zu jeder Entität aus der Datenbank ein UIService, ein REST-Controller, ein Service und ein Repository implementiert wird. Um das Diagramm übersichtlich zu halten, werden die Assoziationen innerhalb der Komponente Services nicht dargestellt.

Betrachtet man die SOLID-Prinzipien, werden durch diesen Aufbau der Anwendung folgende Prinzipien erfüllt:

- **Single Responsibility**: Laut diesem Prinzip sollten Klassen jeweils nur eine Aufgabe bzw. Verantwortung haben. Dadurch werden Klassen erstens nicht sehr groß und zweitens wird dadurch die Weiterentwicklung und Änderung der Klassen vereinfacht. Da wir für jede Entität einen UIService, einen Controller, einen Service und ein Repository anlegen werden, ist dadurch gewährt, dass die einzelnen Klassen tatsächlich nur für deren zugrundeliegende Entität verantwortlich sind. Dadurch werden Events beispielsweise nicht in einer größeren Klasse erstellt, sondern haben hierfür ihren eigenen EventService.

- **Interface segregation**: Dieses Prinzip beschreibt, dass kleine und spezifizierte Interfaces großen einzelnen Interfaces vorzuziehen sind. Aus diesem Grund haben wir uns dafür entschieden, dass jede Entität ihr eigenes Repository erhält. Dadurch arbeiten nicht mehrere Elemente, wie User und Events mit demselben Repository, sondern haben für den Zugriff zur Datenbank ihr eigenes Repository.
- **Dependency inversion**: Hierbei handelt es sich um ein Prinzip, bei dem zwischen zwei Klassen ein Interface eingebaut wird, um die Kopplung zwischen diesen auszuweichen. Dadurch können die Klassen unabhängig voneinander bearbeitet und geändert werden. In unserem Fall befindet sich zwischen den Services und der Datenbank eine Datenzugriffsschicht, sodass die Services nicht direkt auf die Datenbank zugreifen, sondern zunächst über ein Interface. Dadurch sind die Methoden, wie man auf die Datenbank zugreift, immer gleich benannt und aufgebaut. Die Services können dementsprechend einfacher geändert werden.

3.5. Kommunikation zwischen Client und Server

Um die allgemeine Kommunikation zwischen Client und Server zu verdeutlichen, wird im Folgenden ein Sequenzdiagramm dargestellt. Dieses Sequenzdiagramm besteht nicht aus verschiedenen Klassen, sondern aus den zuvor genannten Schichten und Frameworks. Da es sich hierbei zunächst um einen allgemeinen Überblick über die Kommunikation handelt, werden noch keine genauen Klassen- und Methodennamen verwendet. Genauere Abläufe und Funktionen werden in folgenden Kapiteln erläutert.

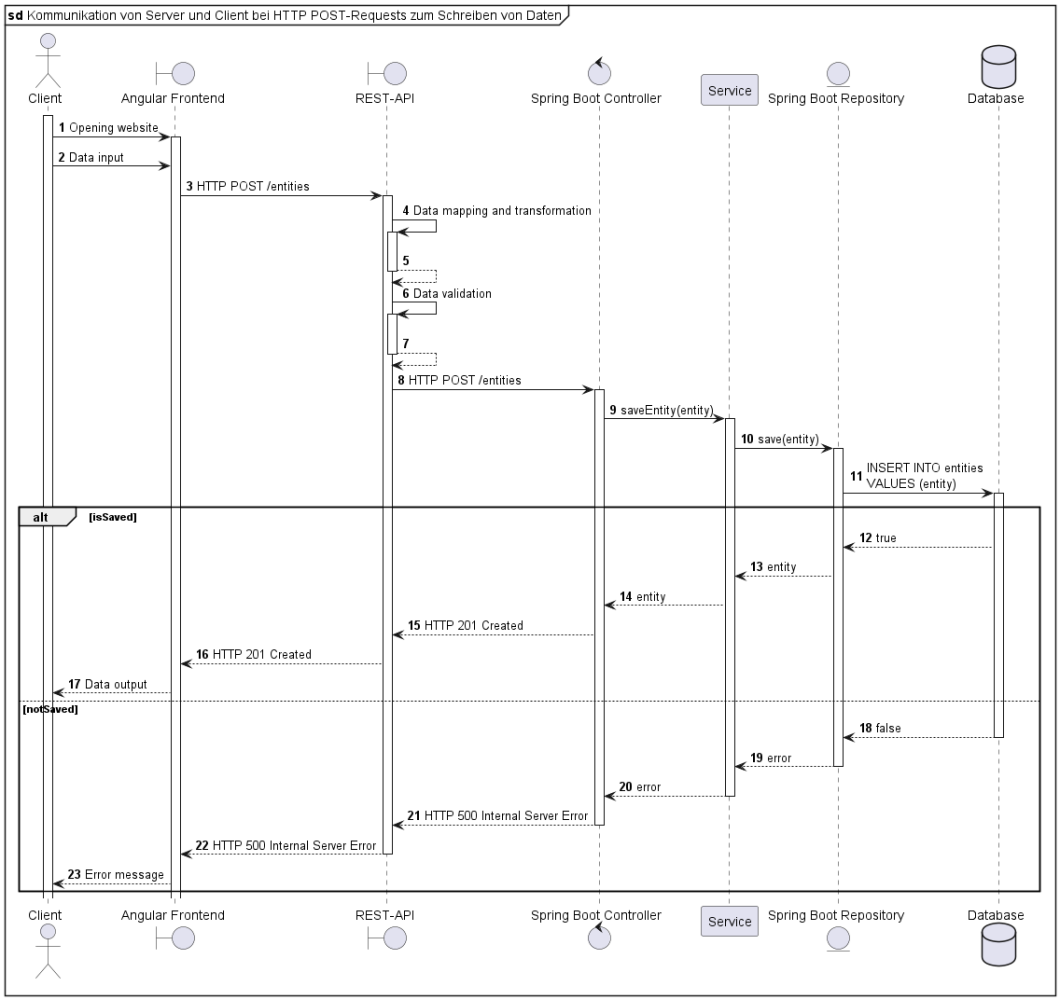

Diagramm 4: HTTP POST Request für das Speichern von Daten [4]

In Diagramm 4 ist der Ablauf einer HTTP POST Request zu sehen. Wir werden diese Request sowohl für das Speichern als auch das Abrufen von Daten verwenden, da POST-Request einen deutlich höheren Datenschutz bieten als beispielsweise GET Requests. Bei GET Requests werden die Daten über die URL gespeichert, sodass sie nicht nur für jeden einsehbar sind, sondern auch im Browserverlauf gespeichert werden und zudem eine geringere Kapazität bereitstellen. In *Diagramm 4* wird zunächst der Ablauf dargestellt, wie Daten gespeichert werden. Der Client öffnet zunächst die Website, sodass das Frontend geladen wird. Dann gibt er die Daten ein, die gespeichert werden sollen. In diesem Diagramm handelt es sich allgemein um ein Tupel einer Entität, das der Client wünscht, zu speichern. In der tatsächlichen Anwendung kann dies beispielsweise ein Event sein. Die Anfrage des Clients wird dann über

eine HTTP POST Request mit einer URL an die RESTful-API weitergeleitet. Bei der URL handelt es sich um den Endpunkt, in dem sich die Daten befinden. Die RESTful-API führt dann eine Datenmapping- und Transformationsoperation durch, um sicherzustellen, dass die Daten im richtigen Format vorliegen. Anschließend wird eine Datenvalidierung durchgeführt, um die Gültigkeit der Daten festzustellen. Wenn die Daten gültig sind, leitet die RESTful-API die Request an den Spring Boot Controller weiter. Die Controller werden von den Entwicklern selbst implementiert. In diesem Fall führt dieser die Methode saveEntity(entity) in dem dazugehörigen entsprechenden Service aus, der dann die save(entity) Methode im Repository ausführt. Das Repository speichert dann über eine SQL-Anfrage das Tupel in der Datenbank und an den Controller einen Boolean zurück, ob die Operation erfolgreich verlaufen ist. Abhängig des Erfolgs, erhält der Client eine Nachricht, ob die Operation erfolgreich war oder fehlgeschlagen ist.

Im Folgenden Diagramm 5 wird ein weiteres Sequenzdiagramm zur Kommunikation zwischen Client und Server dargestellt. Bei diesem Diagramm wird kein Speichern von Daten durchgeführt, sondern das Lesen von Daten.

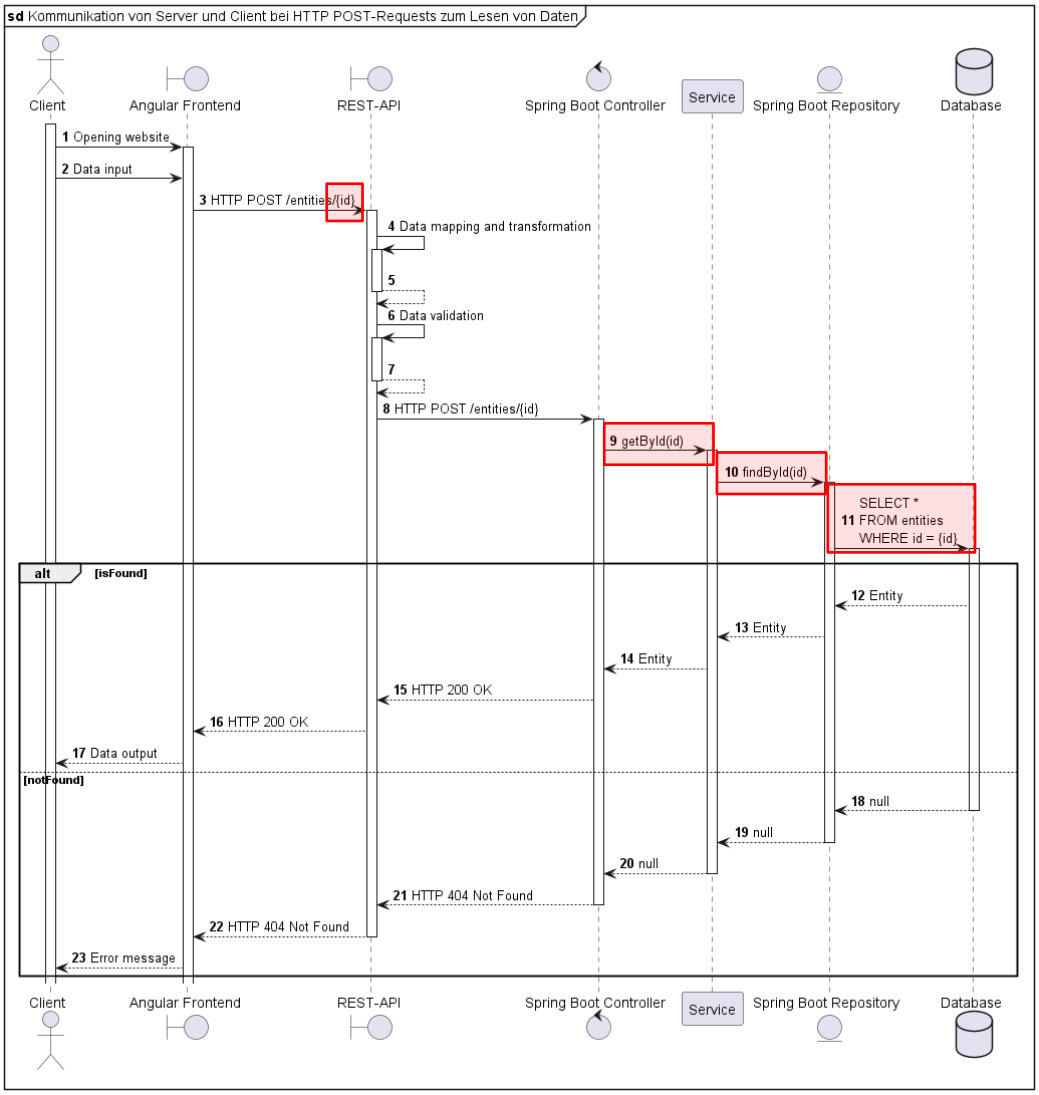

Diagramm 5: HTTP POST Request für das Laden von Daten [5]

Im Allgemeinen sind die Abläufe recht ähnlich zum Speichern von Daten. Sie unterscheiden sich zunächst darin, dass bei der HTTP-Request nun auch ein eindeutiges Attribut (bspw. ID als Primärschlüssel) mitübergeben wird. Darüber hinaus wird beim Controller die Methode getById(id) ausgeführt, sodass nun das Repository keinen INSERT INTO – Befehl ausführt, sondern den SELECT – Befehl, um Daten von der Datenbank abzurufen. Falls dies erfolgreich durchgeführt worden ist, liefert die Datenbank das angeforderte Tupel zurück. Befindet sich in der Datenbank kein Tupel, das die Bedingung erfüllt, liefert die Datenbank null zurück. Abhängig von der Rückgabe der Datenbank werden unterschiedliche Daten an das Frontend weitergeleitet, sodass der Client entweder die angeforderten Daten oder eine Error message erhält.

Um zu verdeutlichen, was die RESTful-API tut, wird im Folgenden Diagramm 6 ein Aktivitätsdiagramm dargestellt, bei dem die API eine POST-Request erhält und verarbeitet.

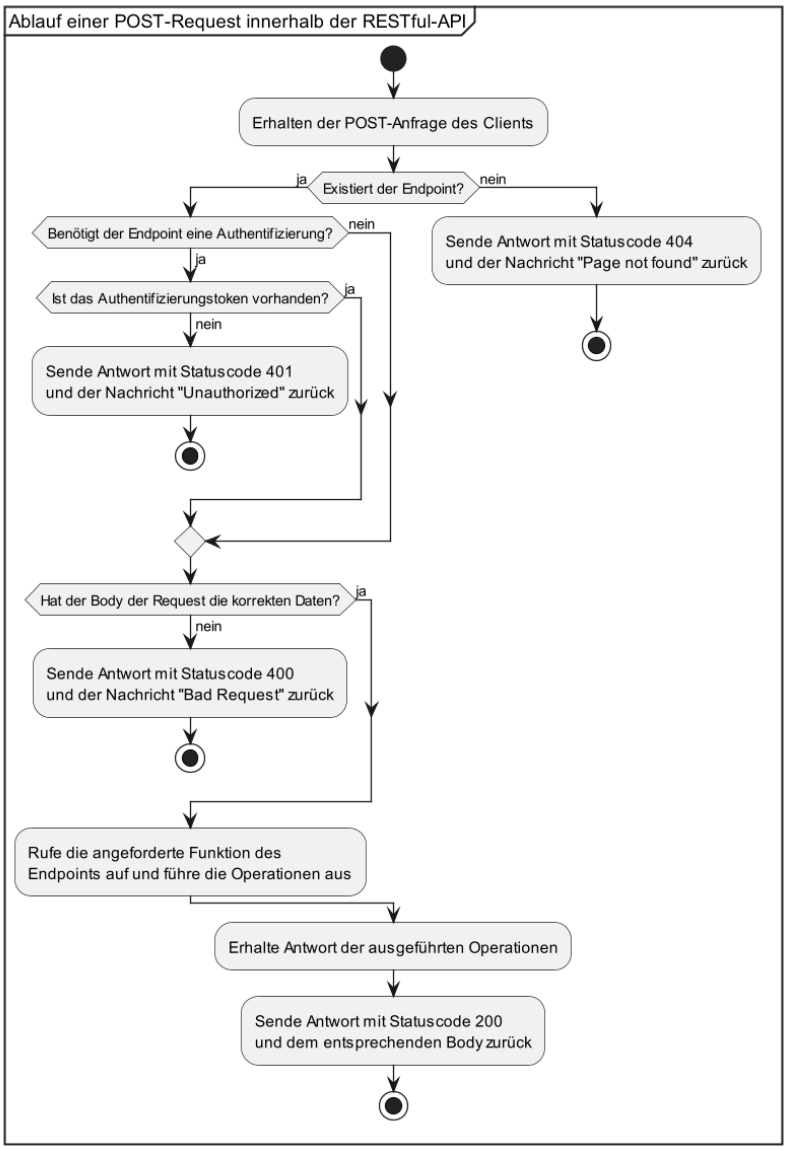

Diagramm 6: Aktivitätsdiagramm zur RESTful-API [11]

4. Struktur- und Entwurfsentscheidungen zu Komponenten

Inhalt dieses Kapitels sind die Struktur- und Entwurfsentscheidungen einzelner Komponenten. Dabei wird zunächst auf die Komponente UI eingegangen. Anschließend wird die Komponente Controller erläutert, bevor die Komponenten Services und DataAccess erläutert werden.

4.1. User-Interface

Die Benutzeroberfläche wird aus verschiedenen Ansichten bestehen, die sich teilweise abhängig von den verschiedenen Rollen voneinander unterscheiden. Durch diese Ansichten kann mit der Maus navigiert werden. Zudem ist eine Tastatur für Eingaben notwendig.

Da die Anforderung besteht, die Software kompatibel mit anderen Plattformen wie Smartphones, Tablets etc. zu machen, wird auch eine leicht optimierte Ansicht für Mobilgeräte entworfen.

Da die einzelnen UIServices lediglich dafür verantwortlich sind, die Daten und Aufrufe an das Backend weiterzuleiten, wird in diesem Kapitel darauf eingegangen, für welche Abläufe und Darstellungen der Benutzeroberfläche wir uns entschieden haben.

Im Folgenden Diagramm 7 wird die gesamte Darstellung in einem Zustandsdiagramm zusammengefasst. Für eine bessere Übersicht wurden die Bedingungen für die Übergänge weggelassen. Diese werden in späteren Diagrammen genauer erläutert. Da von jeder Seite aus auf die vorherige Seite, „Hauptseite“, „Eventkatalog“, „Meine Events“, die Profilansicht, die Organisationsauswahl und je nach Rolle die Seite „Verwaltung“ navigiert werden kann, werden diese Zustandswechsel nicht modelliert und als vorausgesetzt betrachtet.

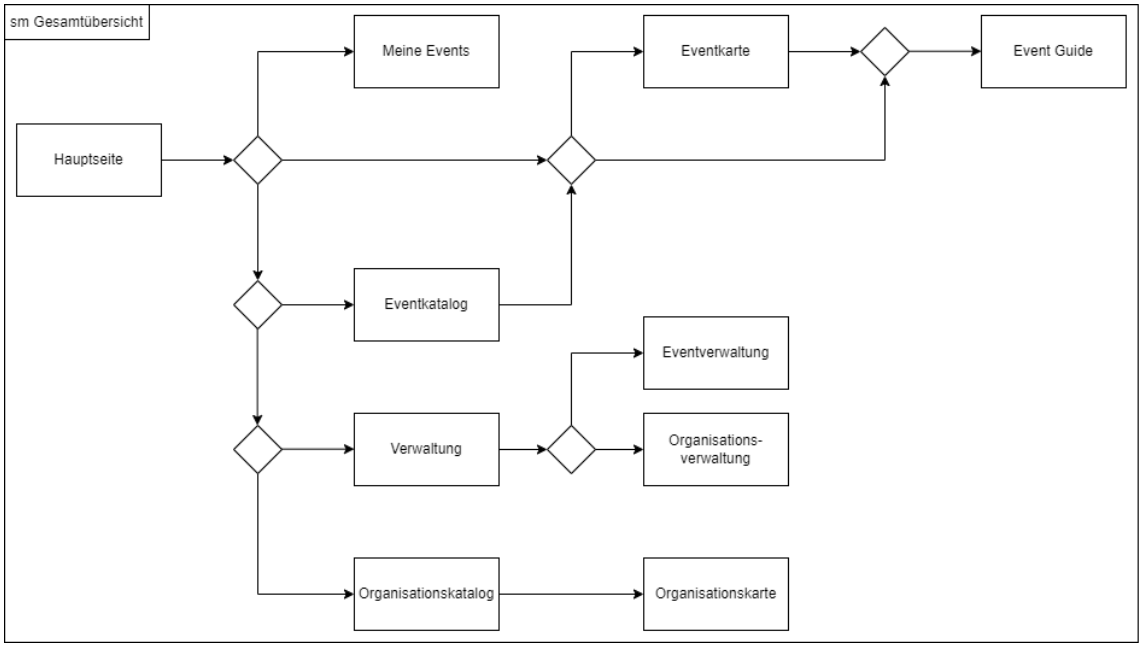

Diagramm 7: Zustandsdiagramm zur gesamten Anwendung [12]

In den folgenden Unterpunkten werden Zustandsdiagramme der einzelnen Ansichten dargestellt. Die Zustände beschreiben dabei eine Ansicht. Auch hier gilt die Voraussetzung bezüglich der Navigation auf vorherige Seiten.

Startseite

Die Startseite ist die Landingpage für jeden Benutzer, der auf den Eventmaster über den Browser zugreift. Auf dieser Startseite hat der Benutzer die Möglichkeit sich zu registrieren und einen Account zu erstellen oder sich anzumelden, sofern er schon einen Account hat.

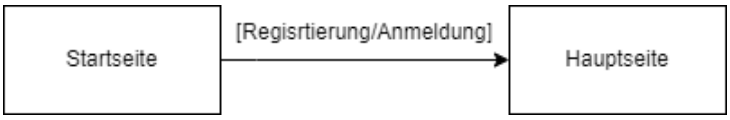

Diagramm 8: Zustandsdiagramm - Startseite

Hauptseite

Die Hauptseite ist die erste Seite in der richtigen Anwendung. Sie dient als eine Art Infoboard auf der alle wichtigen Informationen übersichtlich dargestellt werden. Im Wesentlichen besteht die Hauptseite aus 3 Teilen: „Meine Events“, „Neuigkeiten“ und „Kalender“. Eine konzeptionelle Darstellung der Hauptseite ist in Abbildung 1 dargestellt und zeigt die einzelnen Komponenten.

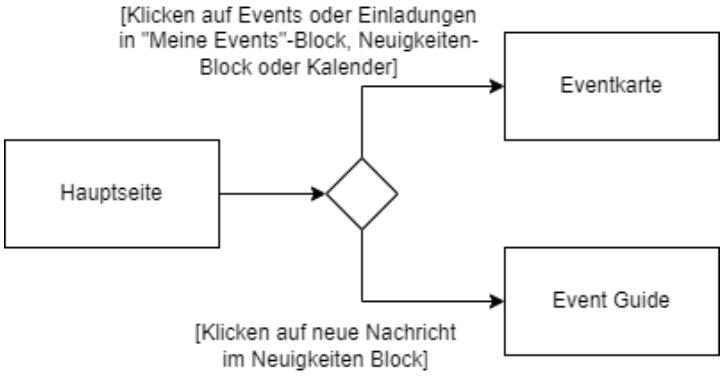

Diagramm 9: Zustandsdiagramm - Hauptseite

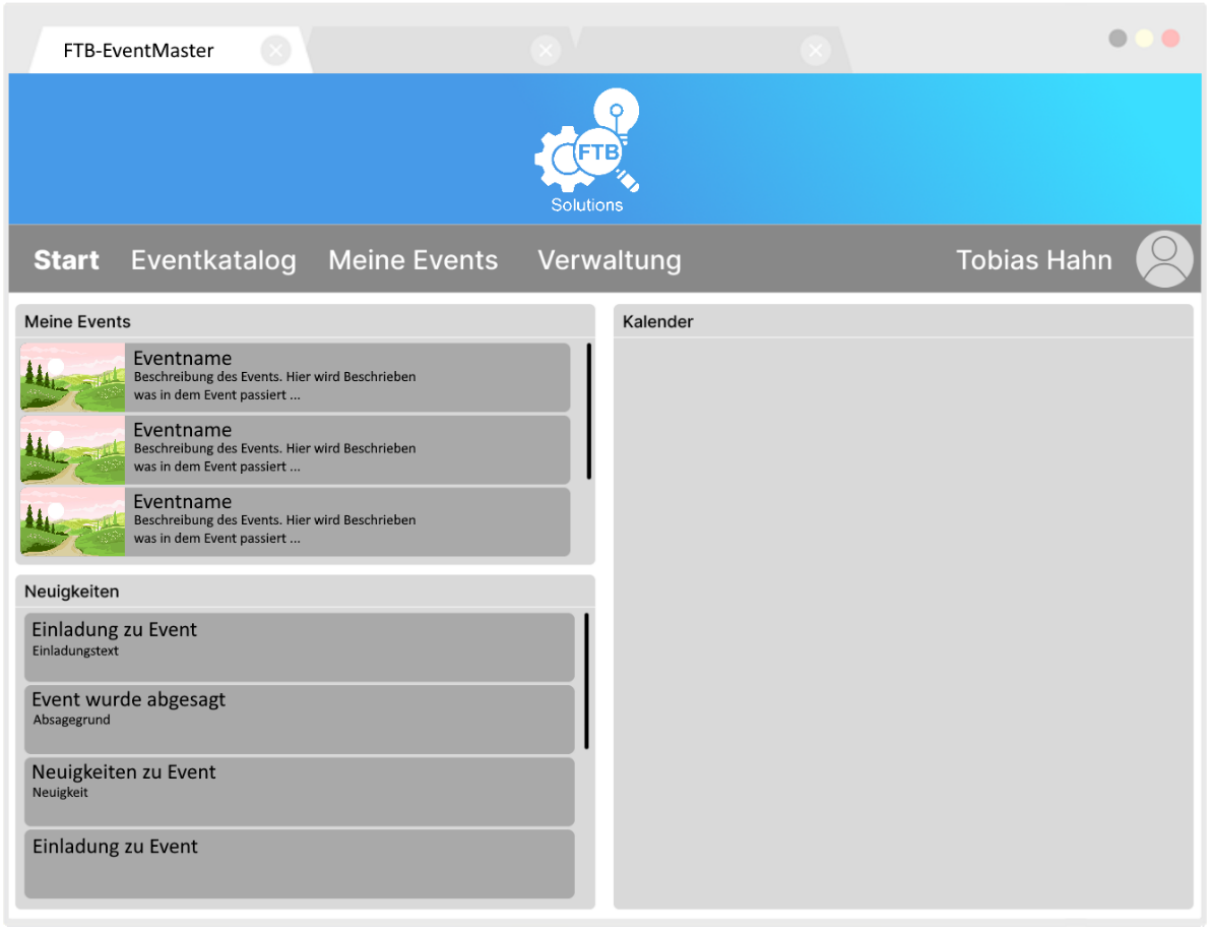

Abbildung 1: GUI-Konzept – Hauptseite [6]

Eventkatalog

Der Eventkatalog ist die Ansicht, in der man nach Events suchen kann und Events, zu denen man Eingeladen wurde, jedoch noch nicht bestätigt hat, angezeigt werden. Dabei gibt es eine Such- und Filterfunktion. Gefiltert werden kann nach verschiedenen Kriterien wie Organisation, Art, Datum, etc. Zudem kann, bei Klicken auf ein angezeigtes Event, auf die Ansicht der Eventkarte gewechselt werden.

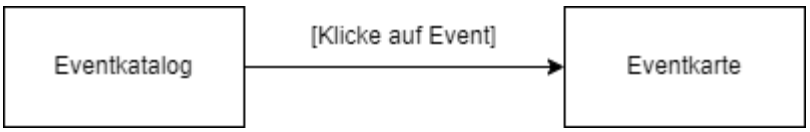

Diagramm 10: Zustandsdiagramm - Eventkatalog

Es soll auch die Möglichkeit geben den Anzeige Modus der Ansicht zu ändern. Dabei gibt es einmal die Ansicht als „Mini“-Eventkarten. Bei diesem Anzeige Modus werden die Events mit einem Bild und einer kurzen Beschreibung angezeigt. Bei dem zweiten Anzeige Modus soll eine rein Tabellarische Ansicht dargestellt werden. Diese beinhalten nur die wichtigsten Informationen wie Name, Zeitraum und Veranstaltungsort.

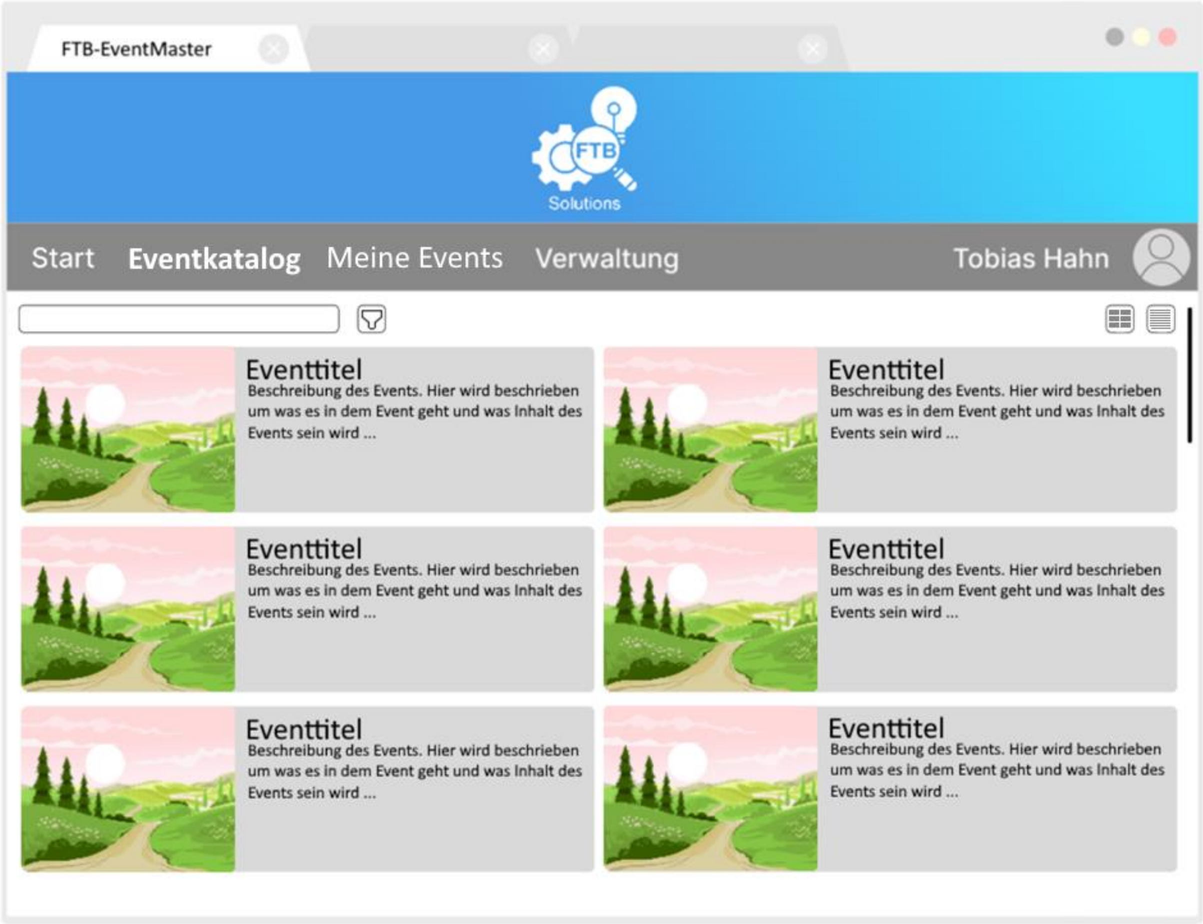

Abbildung 2: GUI-Konzept – Eventkatalog [7]

Meine Events

Die Ansicht „meine Events“ ist gleich aufgebaut von der Benutzeroberfläche wie der Eventkatalog. Der Unterschied hierbei liegt darin, dass hier nur Events angezeigt werden, zu denen man angemeldet ist. Man kommt hierüber auch auf die Eventkarten der angezeigten Events.

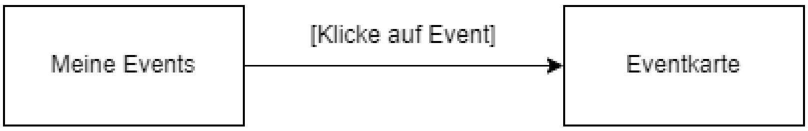

Diagramm 11: Zustandsdiagramm - Meine Events

Verwaltung

Die Ansicht „Verwaltung“ ähnelt von der Struktur ebenfalls sehr stark der Ansicht „Eventkatalog“ (siehe Abbildung 2). Wenn man in der Verwaltungsansicht jedoch auf ein Event klickt, dann öffnet sich nicht die Eventkarte, sondern die Ansicht „Eventverwaltung“. In dieser Ansicht kann nun das ausgewählte Event bearbeitet und verwaltet werden.

Dazu gilt es bei der Verwaltung zwischen zwei Ansichten zu unterscheiden. Der Organisator- und der Administrator Ansicht. Der grundsätzliche Aufbau ist gleich, die Admin-Ansicht ist jedoch um ein paar wenige Funktionen erweitert.

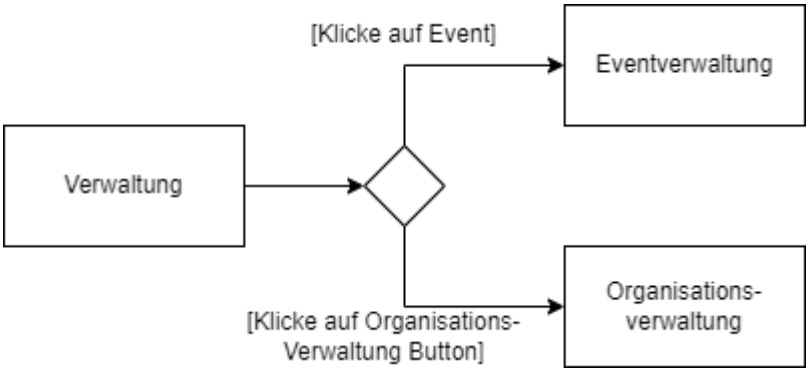

Diagramm 12: Zustandsdiagramm - Verwaltung

Die verwaltbaren Events des Administrators sind im Vergleich zum Organisator nicht nur die selbsterstellten Events, sondern beinhalten alle Events innerhalb der Organisation. Zudem kann nur ein Administrator in die Ansicht „Organisationsverwaltung“ wechseln. Ein Organisator sieht diese Option in seiner Ansicht nicht.

Organisationskatalog

Der Organisationskatalog ist erreichbar durch Klicken auf den Banner. Dadurch öffnet sich der Organisationskatalog. Dieser ist ähnlich aufgebaut, wie der Eventkatalog in Abbildung 2 dargestellt, mit dem Unterschied, dass hier keine Events, sondern Organisationen aufgelistet werden. Hier werden an erster Stelle die Organisationen angezeigt, zu denen man angehörig ist. Des Weiteren gibt es die Möglichkeit in diesem Organisationskatalog nach neuen Organisationen zu suchen denen man beitreten möchte.

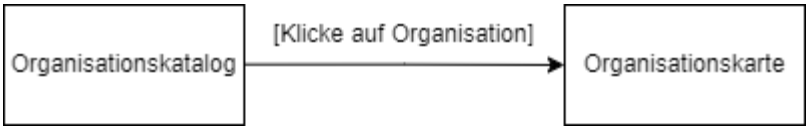

Diagramm 13: Zustandsdiagramm - Organisationskatalog

Weitere Ansichten

Zusätzlich zu den bereits dargestellten Hauptansichten, gibt es auch noch weitere Ansichten, welche nicht direkt über die Navigationsleiste erreichbar sind. Diese Ansichten sind „Eventkarte“, „Event Guide“, „Eventverwaltung“, „Organisationskarte“ und „Organisationsverwaltung“.

Eventkarte

Die Eventkarte wird dann gezeigt, wenn ein Benutzer auf ein Event klickt. Je nachdem über welche Ansicht man diese Eventkarte öffnet, unterscheidet sie sich minimal.

Abbildung 3: GUI-Konzept – Eventkarte [8]

Die Eventkarte kann man nur dann erreicht werden, wenn man Teilnehmer von dem jeweiligen Event ist.

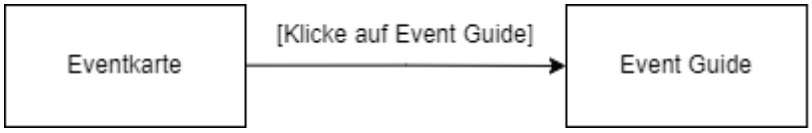

Diagramm 14: Zustandsdiagramm - Eventkarte

Event Guide

Über den Event Guide, ist es einem Teilnehmer möglich, Nachrichten von den Organisatoren und Tutoren einzusehen, an Umfragen teilzunehmen und Dateien herunterzuladen. Das Event Guide soll es den Organisatoren und Tutoren erleichtern wichtige Informationen schnell und einfach an alle Teilnehmer des Events weiterzugeben.

Abbildung 4: GUI-Konzept – Event Guide [9]

Organisationskarte

Die Organisationskarte ist ähnlich aufgebaut wie die Eventkarte in Abbildung 3. Es werden lediglich statt Eventbezogenen Daten, die Daten zu der Organisation angezeigt. Auf dieser Organisationskarte stehen alle wichtigen Informationen, die ein Benutzer wissen sollte und kann sich auch über diese Karte bei einer Organisation anmelden.

Eventverwaltung

Die Eventverwaltung ist nur für Organisatoren, Administratoren und in sehr beschränkter Form für Tutoren zugänglich. Dies Ansicht erreicht man über die Ansicht „Verwaltung“ und für Tutoren über „Meine Events“. In dieser Ansicht können Events erstellt oder bearbeitet werden. Dabei muss der Organisator alle nötigen Informationen auf der Ansicht eintragen und das System generiert daraus dann eine Eventkarte.

Organisationsverwaltung

Die Ansicht „Organisationverwaltung“ ist nur für Administratoren einsehbar. Erreicht wird diese Ansicht durch einen Button in der Ansicht „Verwaltung“, der nur für die Rollen der Administratoren sichtbar ist.

In dieser Ansicht können einige Einstellungen für die Organisation getroffen werden. Zum Beispiel kann in der Organisationverwaltung das Banner, Farbschema sowie Beschreibung gesetzt werden. Zudem hat man in dieser Ansicht eine Mitgliederverwaltung und kann Mitglieder hinzufügen, in Gruppen einteilen oder gegebenenfalls entfernen.

4.2. Services

Im Folgenden Diagramm 15 wird der Inhalt der Komponente Services detaillierter dargestellt. Die Assoziationen leiten sich aus dem Datenbankmodell ab. Da das Klassendiagramm sehr umfangreich ist, befindet sich auch zu diesem, wie zu jedem anderen Diagramm, die entsprechende Datei im *Diagramme*-Ordner. (Siehe Verweis in der Beschriftung)

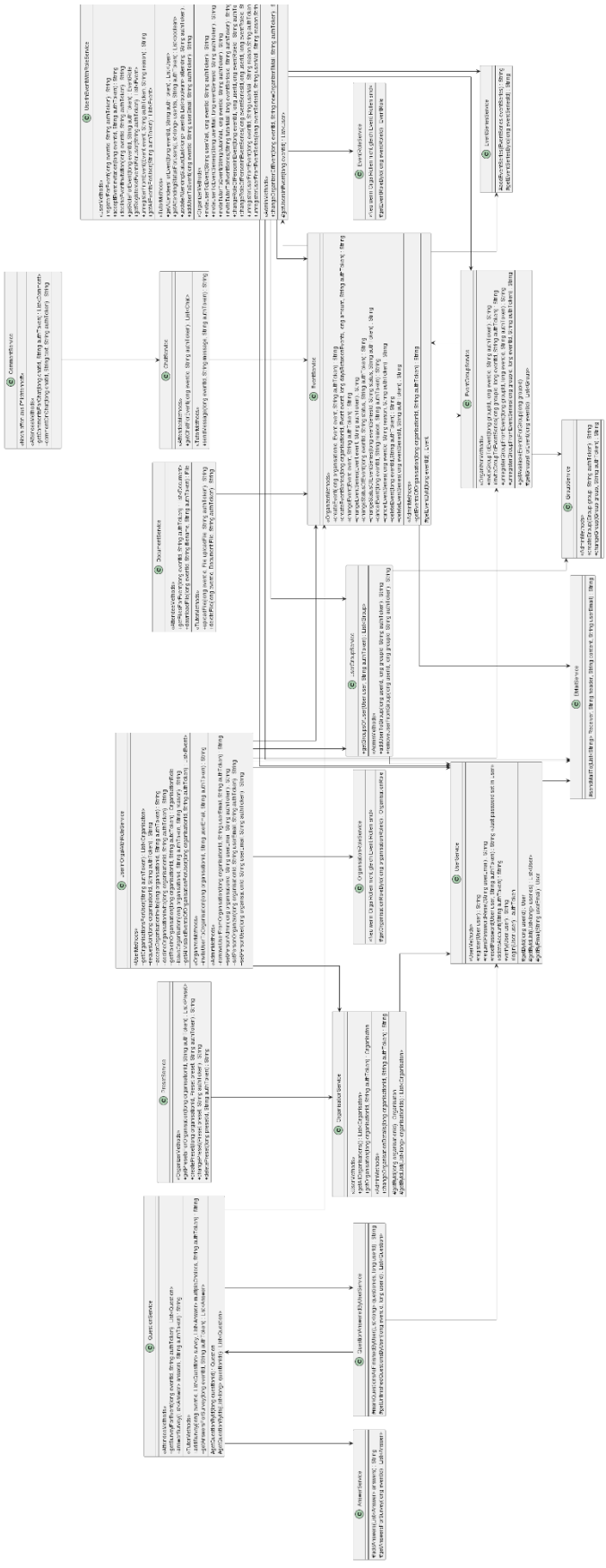

Diagramm 15: Services-Komponente [10]

4.3. Kommunikation zwischen den Services

Für einige UseCases werden mehrere Services verwendet. Dies trifft vor allem dann ein, wenn der Service zu einer Entität nur deren ID erhält und dann über den entsprechenden Service der Entität das vollständige Objekt erhält.

In den folgenden Diagrammen werden drei UseCases dargestellt, für die mehrere Services notwendig sind. Da bereits im letzten Kapitel dargestellt worden ist, wie die Services Daten in der Datenbank speichern und lesen, sowie wie Controller die HTTP Requests erhalten, werden diese Aufrufe in den folgenden Diagrammen weggelassen.

In Diagramm 16 wird der Ablauf dargestellt, wie ein User alle für ihn sichtbaren Events erhält.

Im darauffolgenden Diagramm 17 ist zu sehen, wie ein User sich selbst zu einem Event anmeldet.

Danach wird in Diagramm 18 dargestellt, wie ein User zu einer Gruppe hinzugefügt wird.

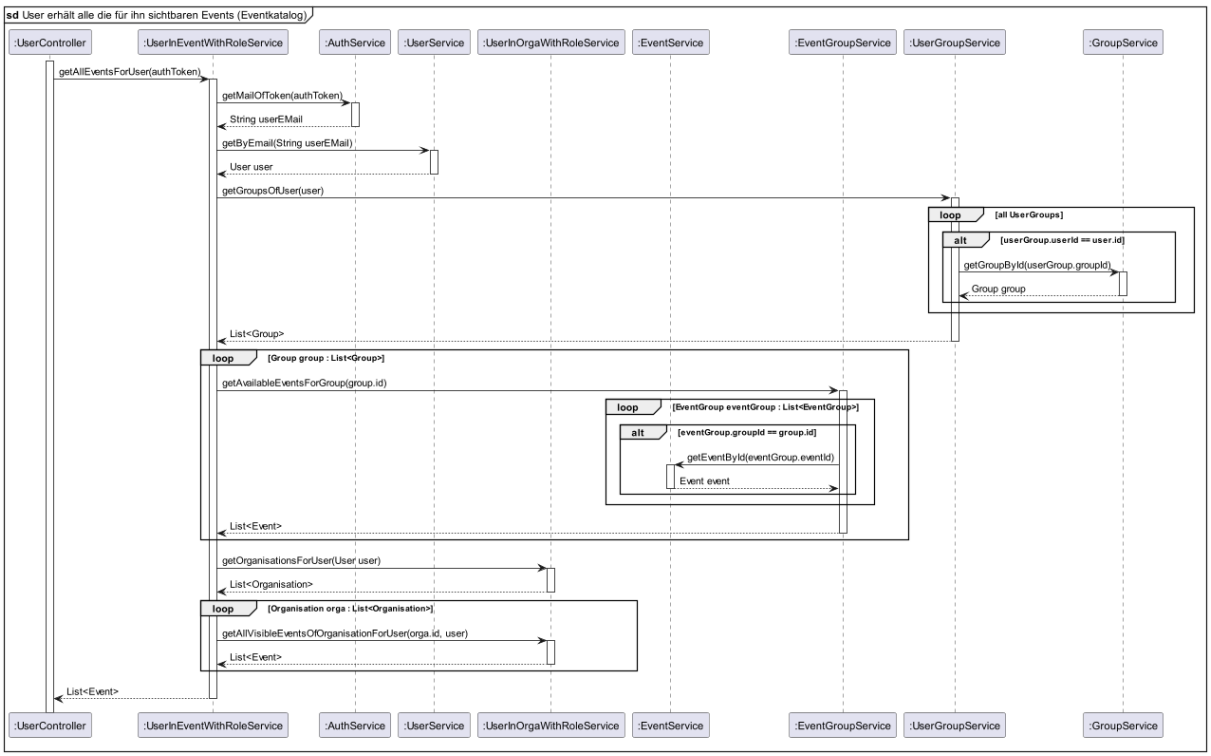

Diagramm 16: Sequenzdiagramm über die für einen User verfügbaren Events [13]

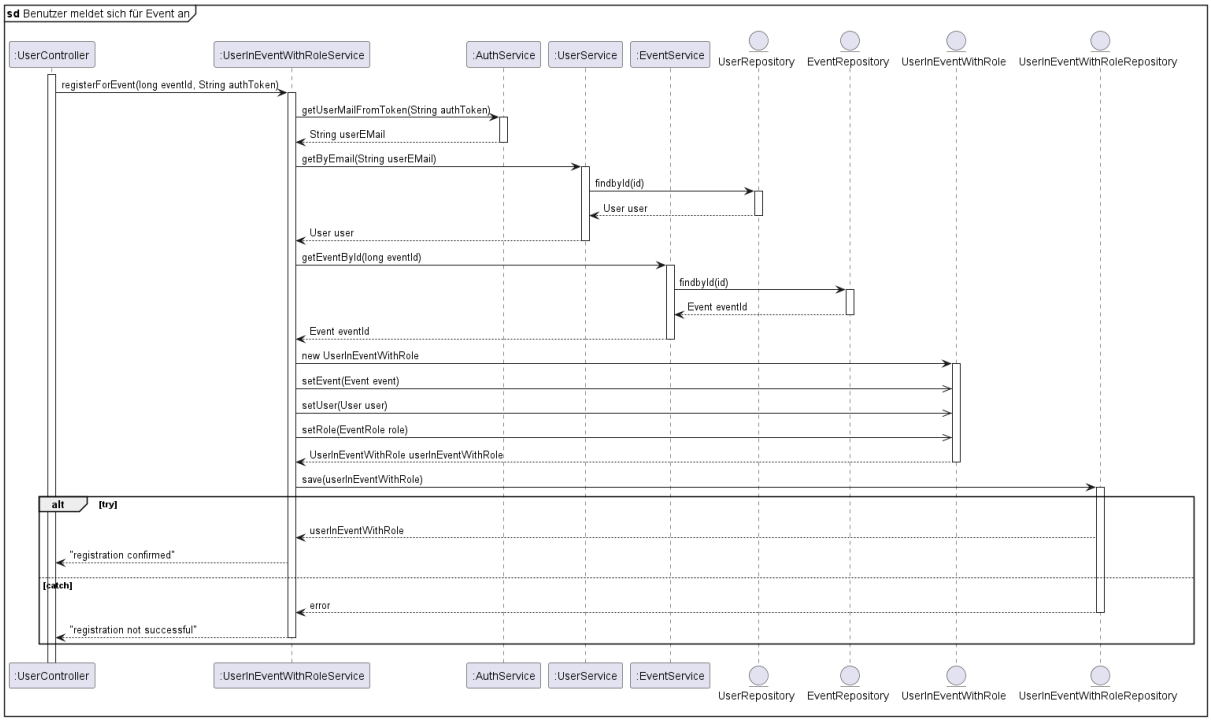

Diagramm 17: Sequenzdiagramm zum Anmelden zu einem Event [14]

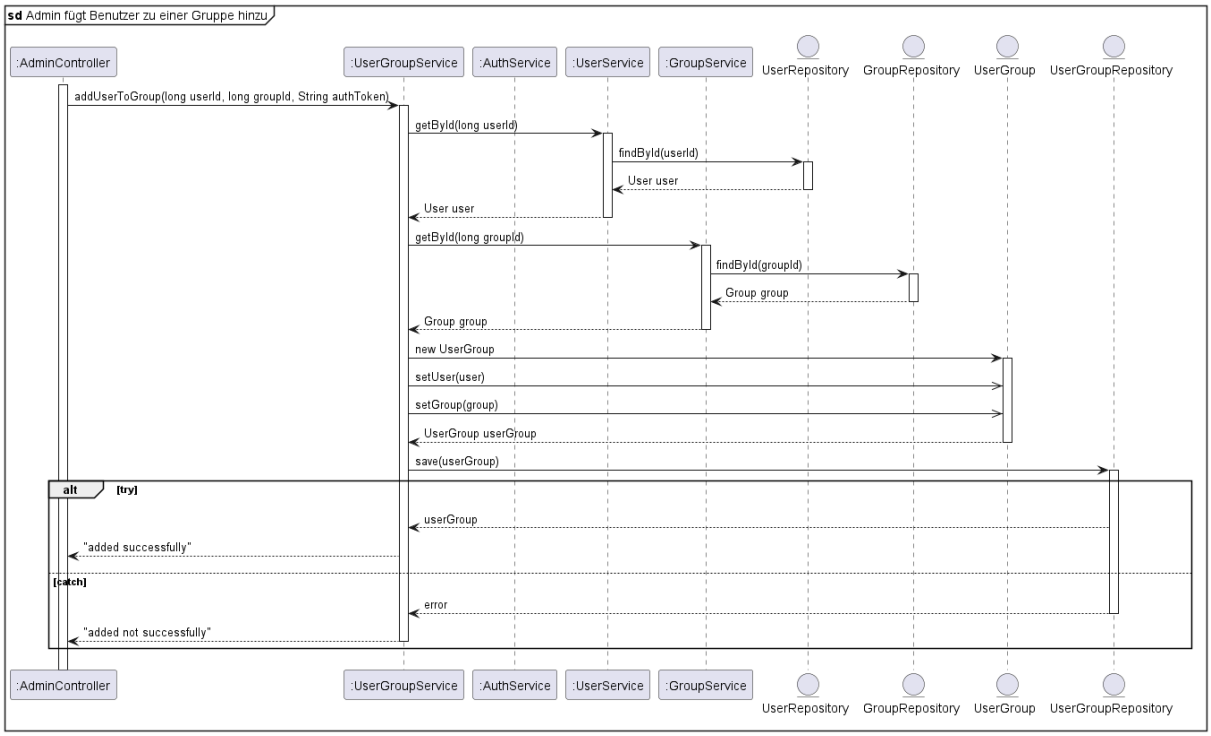

Diagramm 18: Sequenzdiagramm zum Hinzufügen eines Users zu einer Gruppe [15]

4.4. Controller

Die Controller sind für die Verbindung zwischen der Benutzeroberfläche und den Services, also der Geschäftslogik, verantwortlich. In ihnen werden benötigte Endpoints definiert und die entsprechenden Funktionen der Services aufgerufen.

4.4.1 User Controller

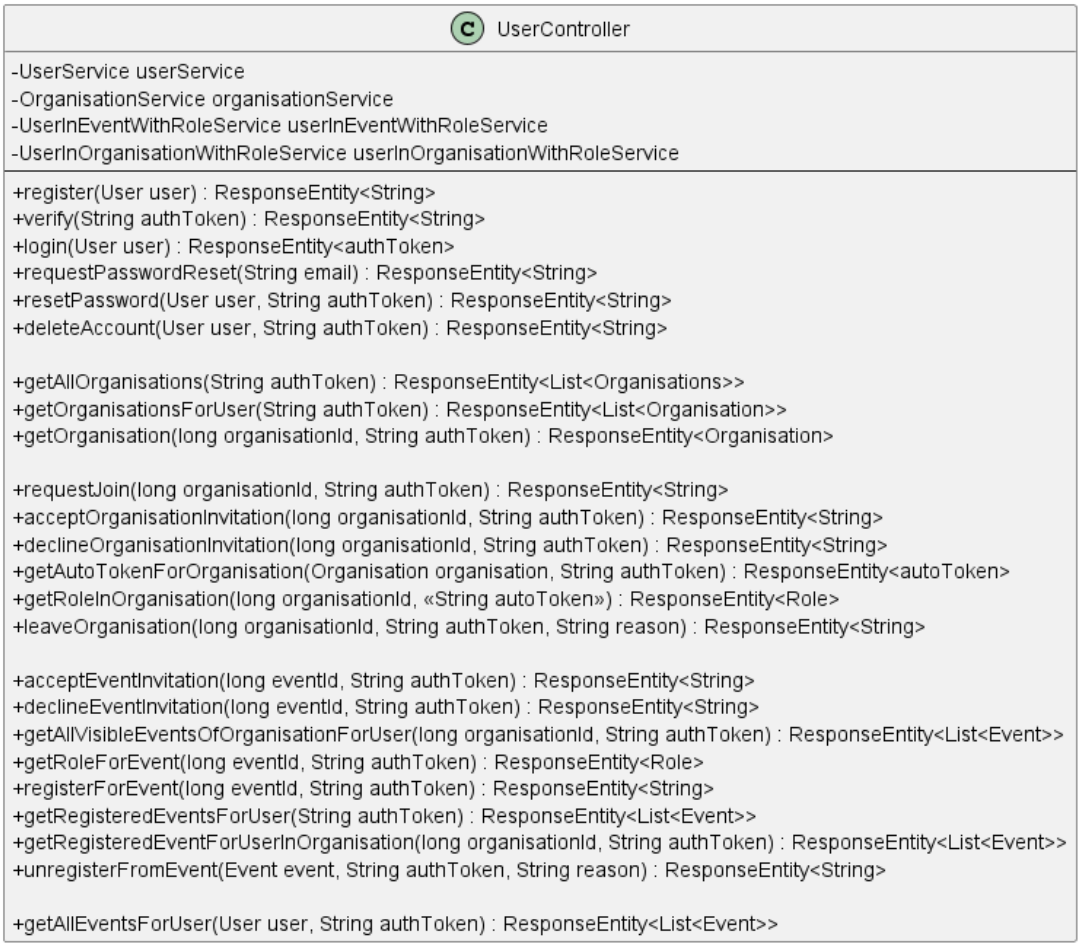

Diagramm 19: Klasse UserController [16]

Der User Controller beinhaltet alle Funktionen die einen User betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

4.4.2 Atendee Controller

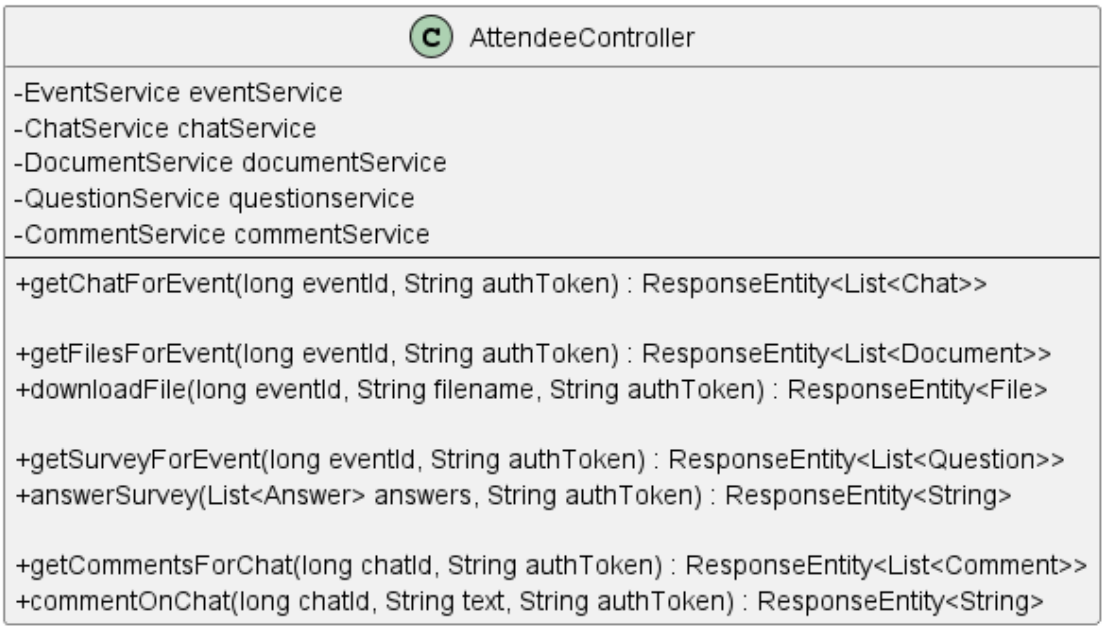

Diagramm 20: Klasse AttendeeController [17]

Der Attendee Controller beinhaltet alle Funktionen die einen Attendee(Teilnehmer) eines Events betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

4.4.3 Tutor Controller

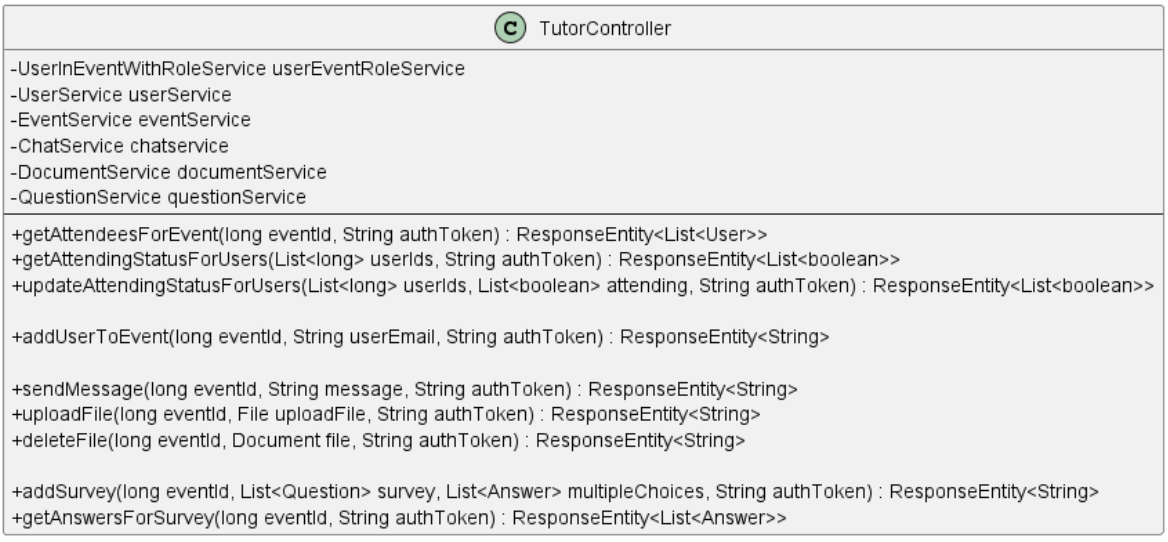

Diagramm 21: Klasse TutorController [18]

Der Tutor Controller beinhaltet alle Funktionen die einen Tutor eines Events betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

4.4.4 Organizer Controller

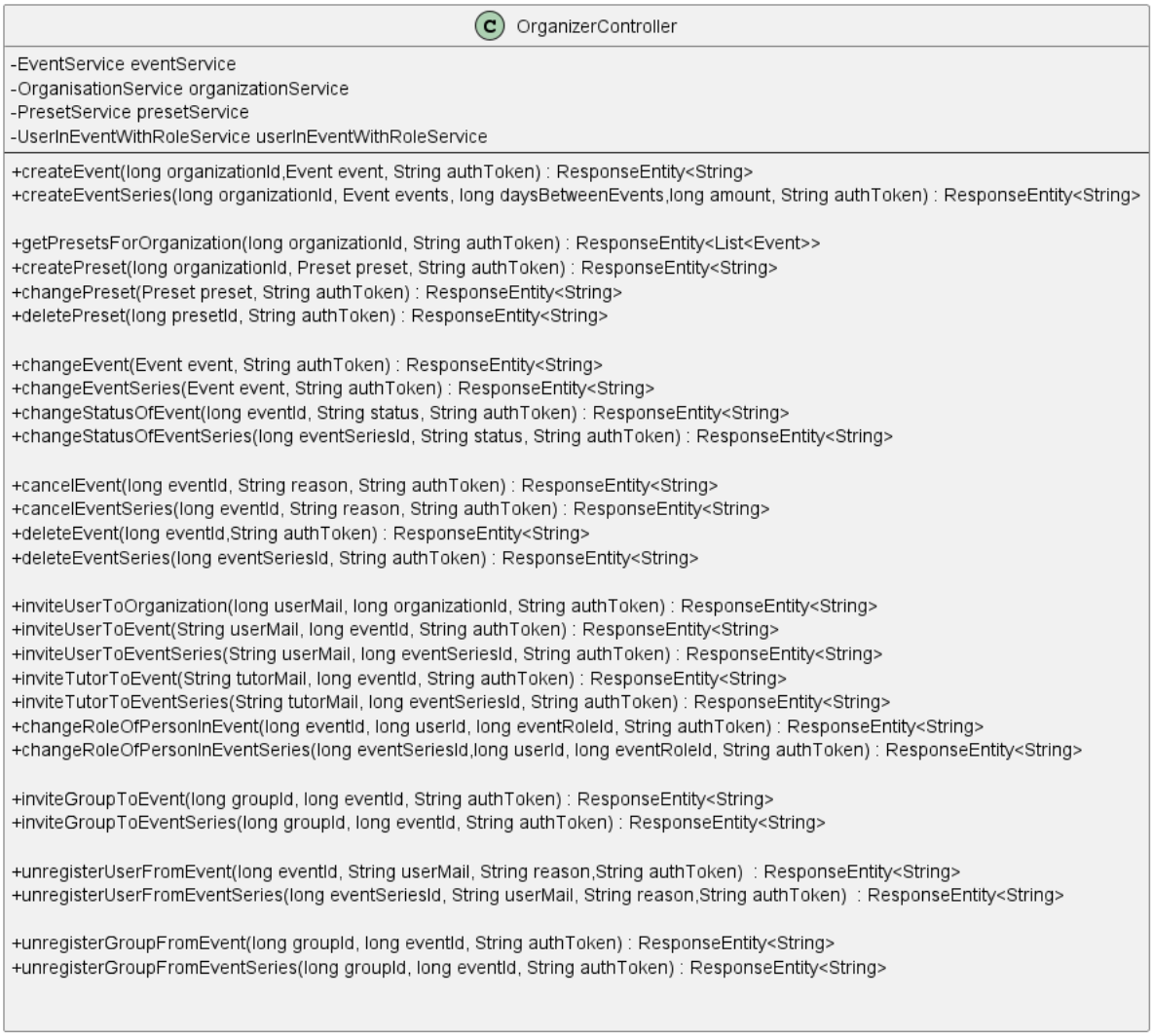

Diagramm 22: Klasse OrganizerController [19]

Der Organizer Controller beinhaltet alle Funktionen die einen Organizer eines Events betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

4.4.5 Admin Controller

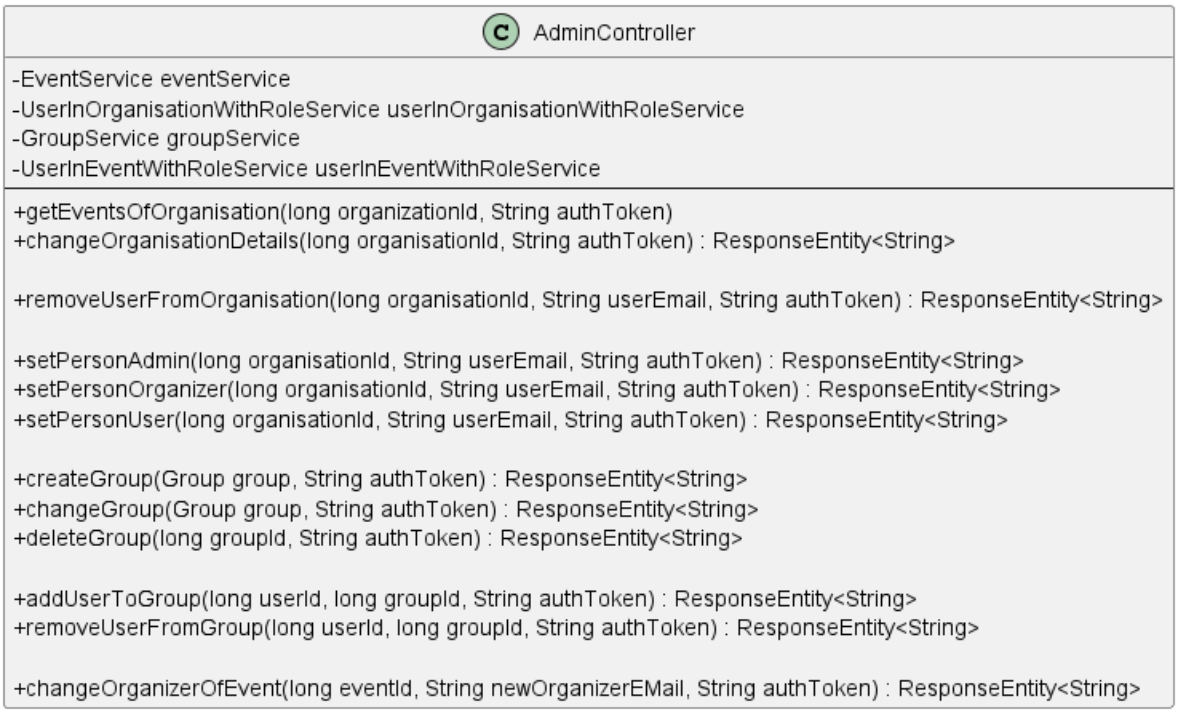

Diagramm 23: Klasse AdminController [20]

Der Admin Controller beinhaltet alle Funktionen die einen Admin eines Events betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

4.4.6 System Admin Controller

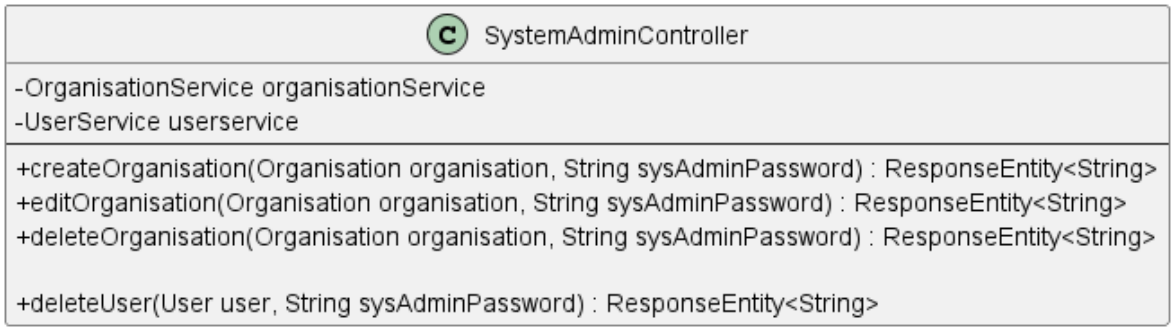

Diagramm 24: Klasse SystemAdminController [21]

Der System Admin Controller beinhaltet alle Funktionen die einen System Admin eines Events betreffen. Zu jeder dieser Funktionen wird ein Endpoint gegeben. Dieser Controller greift auf unterschiedliche Services zu, die über das Erstellen eines Objektes des jeweiligen Services genutzt werden.

Dateiverweise

Verweis	Bezeichnung	Dateiname
[1]	Diagramm 1	Architektur.png
[2]	Diagramm 2	Database.png
[3]	Diagramm 3	Komponentendiagramm_Gesamt.png
[4]	Diagramm 4	Sequenzdiagramm_POSTMapping_Speichern.png
[5]	Diagramm 5	Sequenzdiagramm_POSTMapping_Lesen.png
[6]	Abbildung 1	GUI_Konzept_Hauptseite.png
[7]	Abbildung 2	GUI_Konzept_Eventkatalog.png
[8]	Abbildung 3	GUI_Konzept_Eventkarte.png
[9]	Abbildung 4	GUI_Konzept_EventGuide.png
[10]	Diagramm 15	Klassendiagramm_Services.png
[11]	Diagramm 6	Aktivitätsdiagramm_REST.png
[12]	Diagramm 7	Zustandsdiagramme_Gesamt.png
[13]	Diagramm 16	Sequenzdiagramm_GetAvailableEventsForUser.png
[14]	Diagramm 17	Sequenzdiagramm_RegisterForEvent.png
[15]	Diagramm 18	Sequenzdiagramm_AddUserToGroup.png
[16]	Diagramm 19	Klasse_UserController.png
[17]	Diagramm 20	Klasse_AttendeeController.png
[18]	Diagramm 21	Klasse_TutorController.png
[19]	Diagramm 22	Klasse_OrganizerController.png
[20]	Diagramm 23	Klasse_AdminController.png
[21]	Diagramm 24	Klasse_SystemAdminController.png