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

Inhaltsverzeichnis

Abbildungsverzeichnis

Diagrammverzeichnis

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.

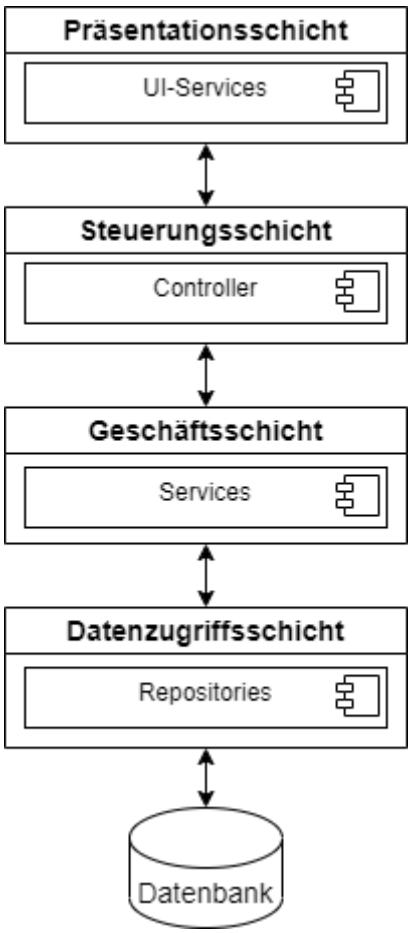

Diagramm 1: Gesamtarchitektur der Anwendung [1]

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 E/R-Modell dargestellt. Auf diesem Modell basiert auch das Komponentendiagramm, da die jeweiligen Klassen durch die Frameworks automatisch in die Datenbank überführt werden.

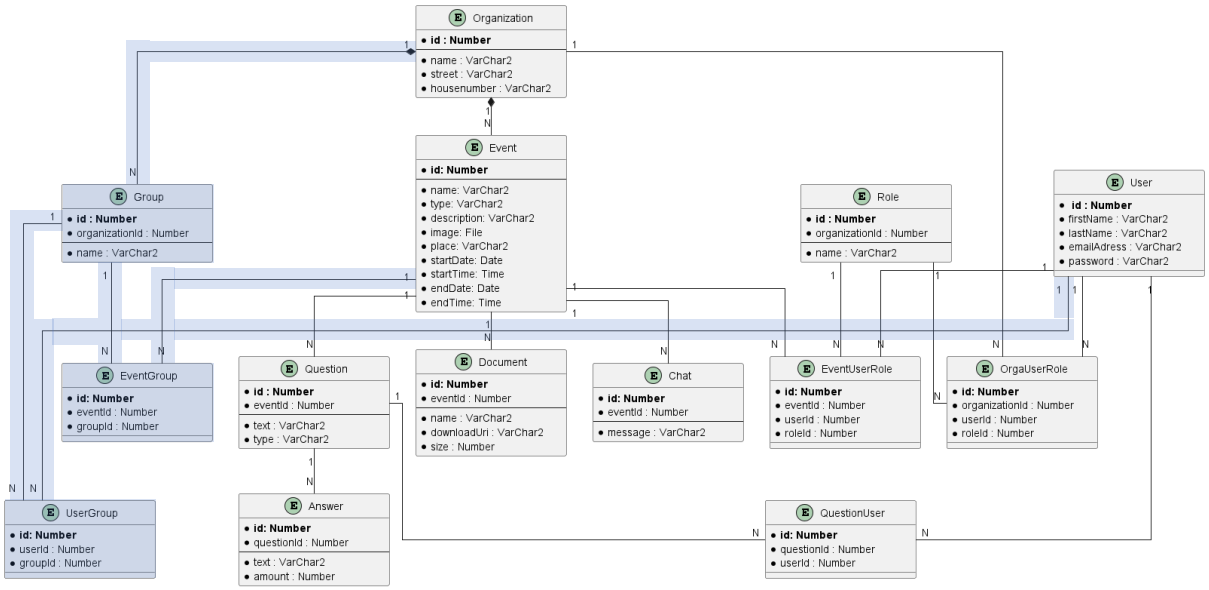

Diagramm 2: E/R-Modell der Datenbank [2]

Dieses E/R-Modell (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. EventUserRole). Da für das Backend Java Spring Boot eingesetzt wird, bildet dieses E/R-Modell 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.

Es ist außerdem zu beachten, dass es ein Wunsch von dem Kunden war, Benutzer gruppieren zu können. Zusätzlich gab der Kunde jedoch den Hinweis, dass dies nur optional und somit nicht notwendig sei. Deshalb wird sich zunächst auf die Elemente konzentriert, die nicht blau gefärbt sind. Sollte nach Absprache mit dem Kunden dieser doch den Wunsch haben, dass Gruppen eingebaut werden, oder allgemein noch genügend Zeit dafür ist, werden die blau gefärbten Elemente hinzugefügt. Dadurch wäre es dann möglich, ganze Gruppen von Benutzern zu Events einzuladen.

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.

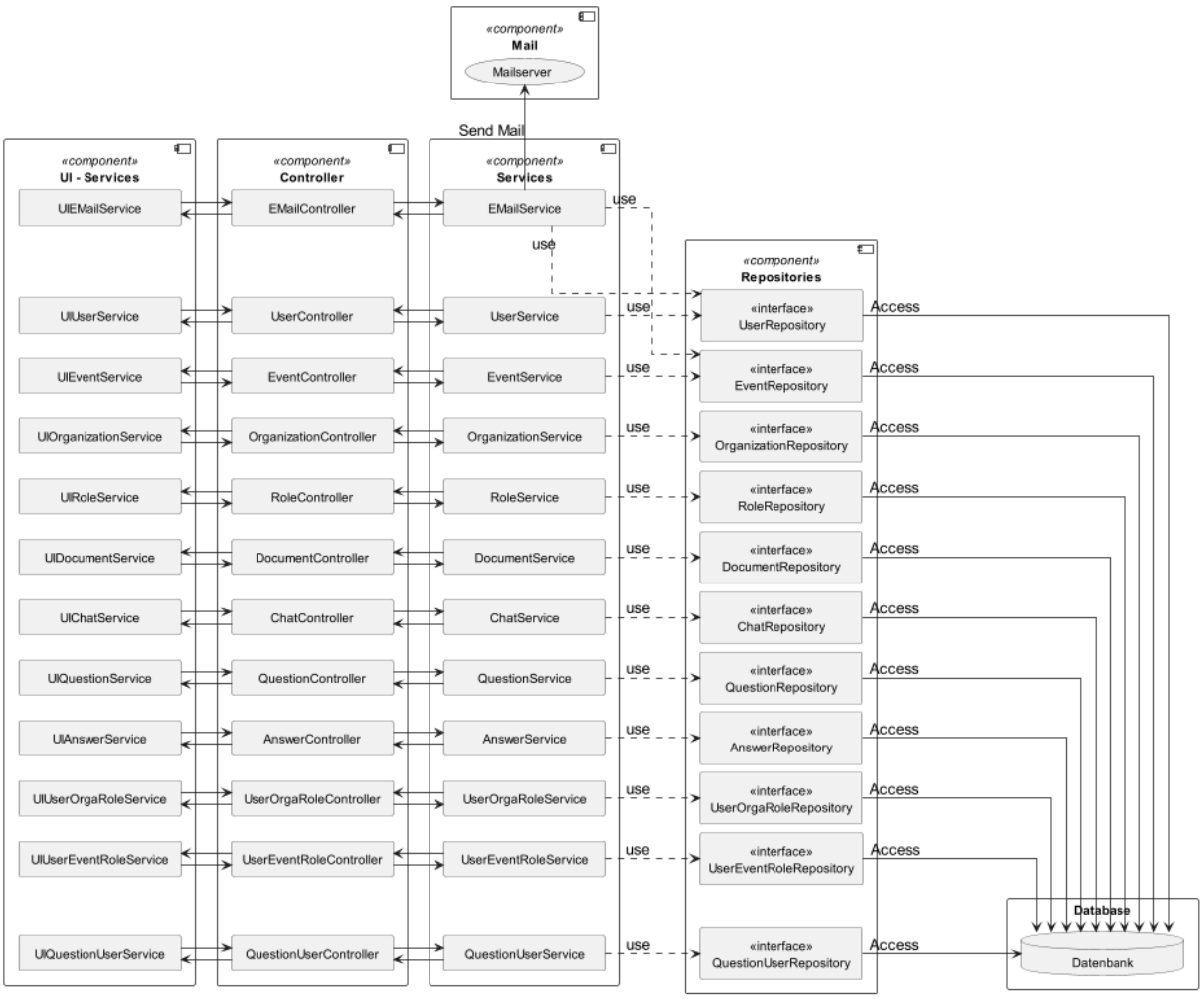

Diagramm 3: Komponentendiagramm der Anwendung [3]

Durch das Diagramm ist ersichtlich, dass zu jeder Entität aus der Datenbank ein UIService, ein REST-Controller, ein Service und ein Repository implementiert wird. Innerhalb der Komponente *Services* haben die einzelnen Services untereinander Zugriff, wenn sie ihn benötigen.

Besonderheit ist, dass zusätzlich für den E-Mail-Verkehr entsprechende Elemente eingebaut werden. Da dieser ausschließlich die Daten zu Events und Usern (wie Zeit, Ort des Events und E-Mail-Adresse des Users) benötigt, kann diese dementsprechend über die UserRepository und EventRepository auf diese Daten zugreifen.

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.

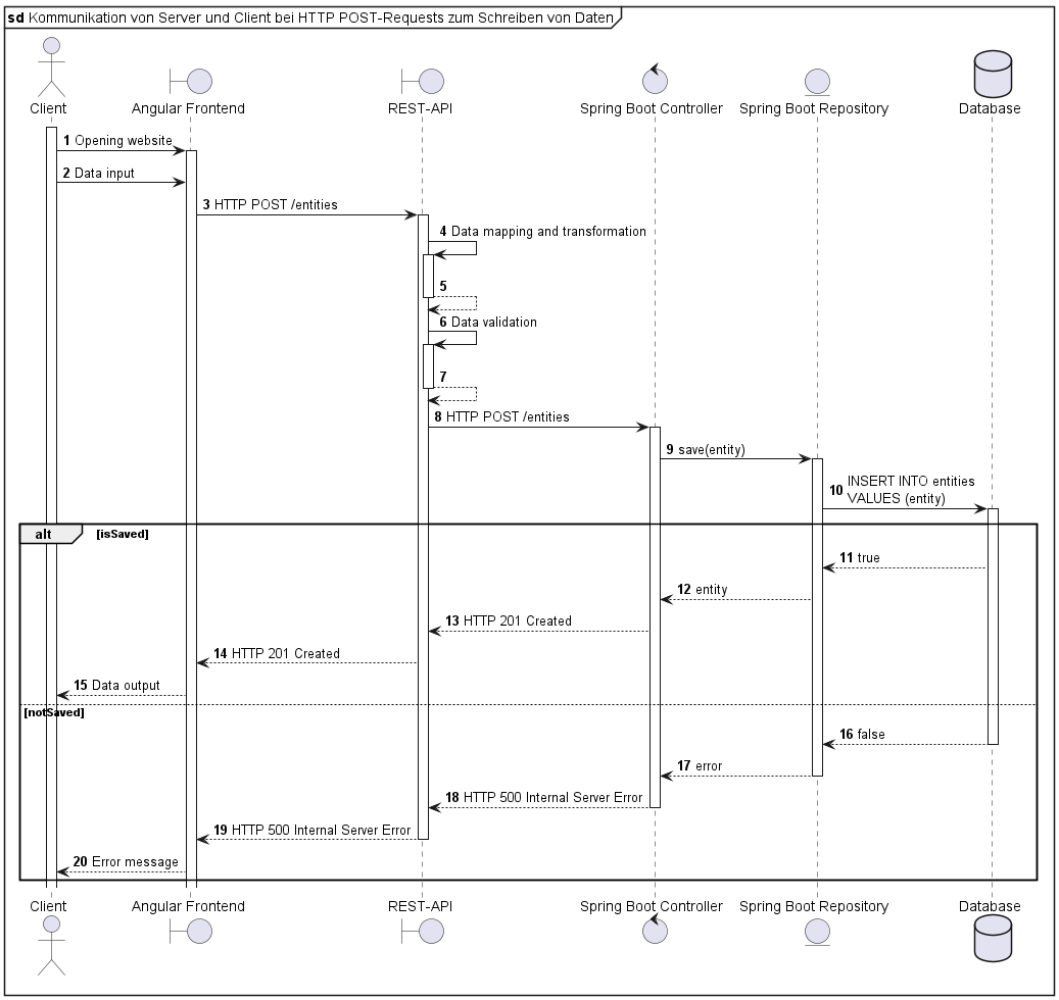

Diagramm 4: HTTP POST Request für das Speichern von Daten [4]

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 save(entity) in dem dazugehörigen Repository aus. 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.

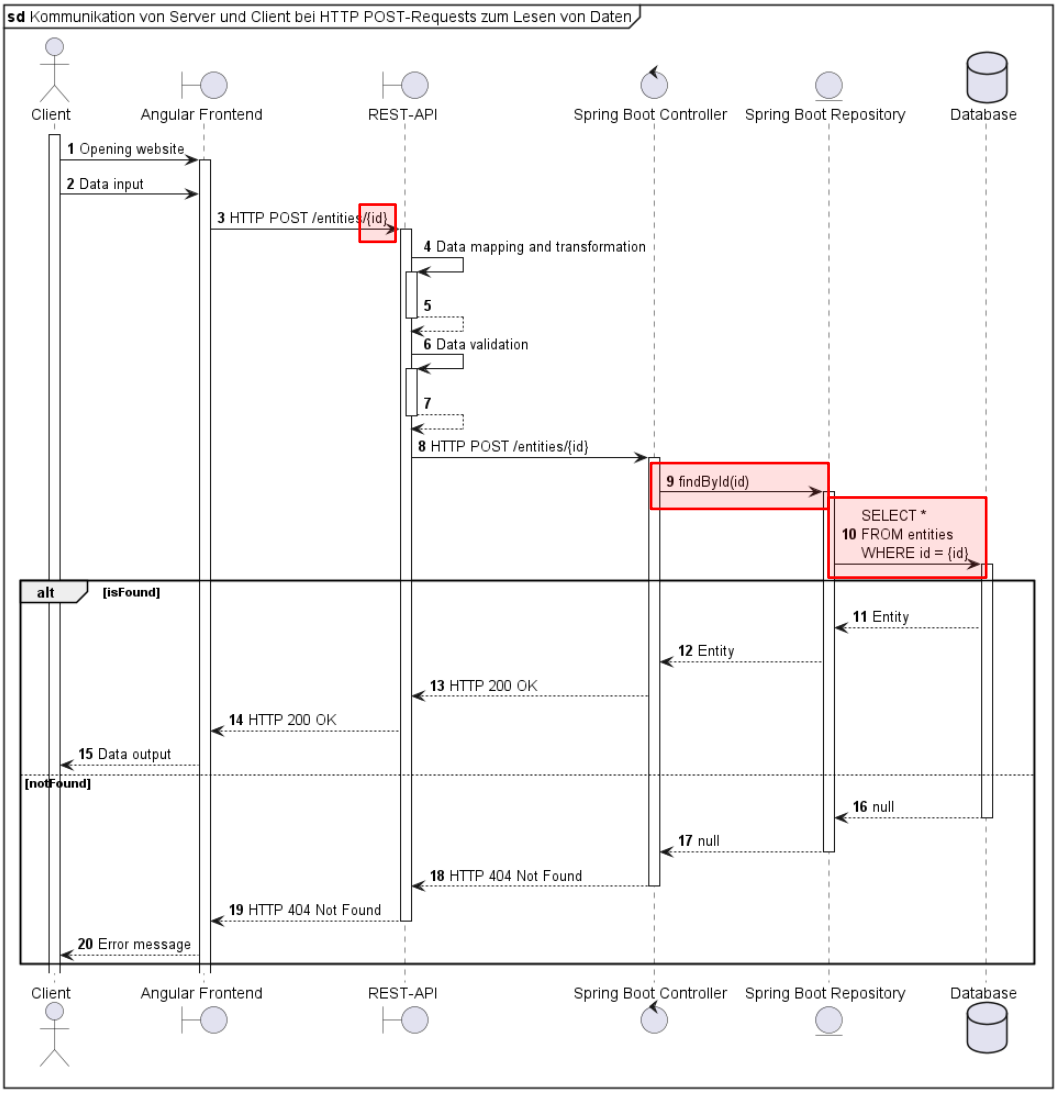

Diagramm 5: HTTP POST Request für das Laden von Daten [5]

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 findById 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.

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.

In den folgenden Unterpunkten werden Zustandsdiagramme der einzelnen Ansichten dargestellt. Die Zustände beschreiben dabei eine Ansicht. 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.

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.

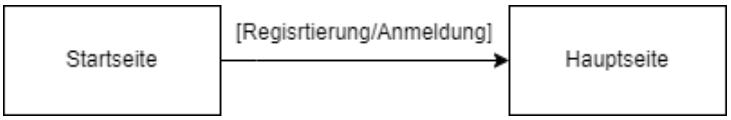

Diagramm 6: Zustandsdiagramm - Startseite

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.

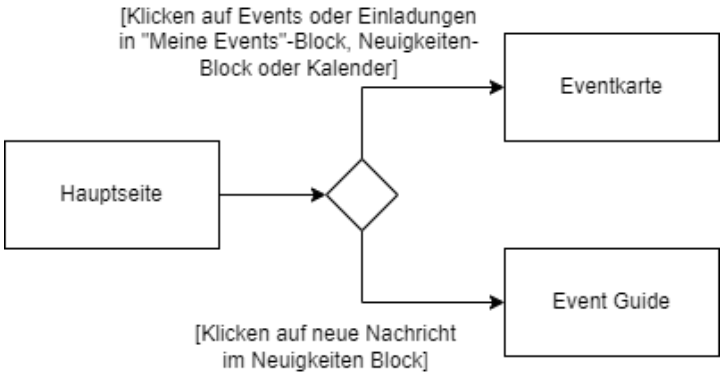

Diagramm 7: Zustandsdiagramm - Hauptseite

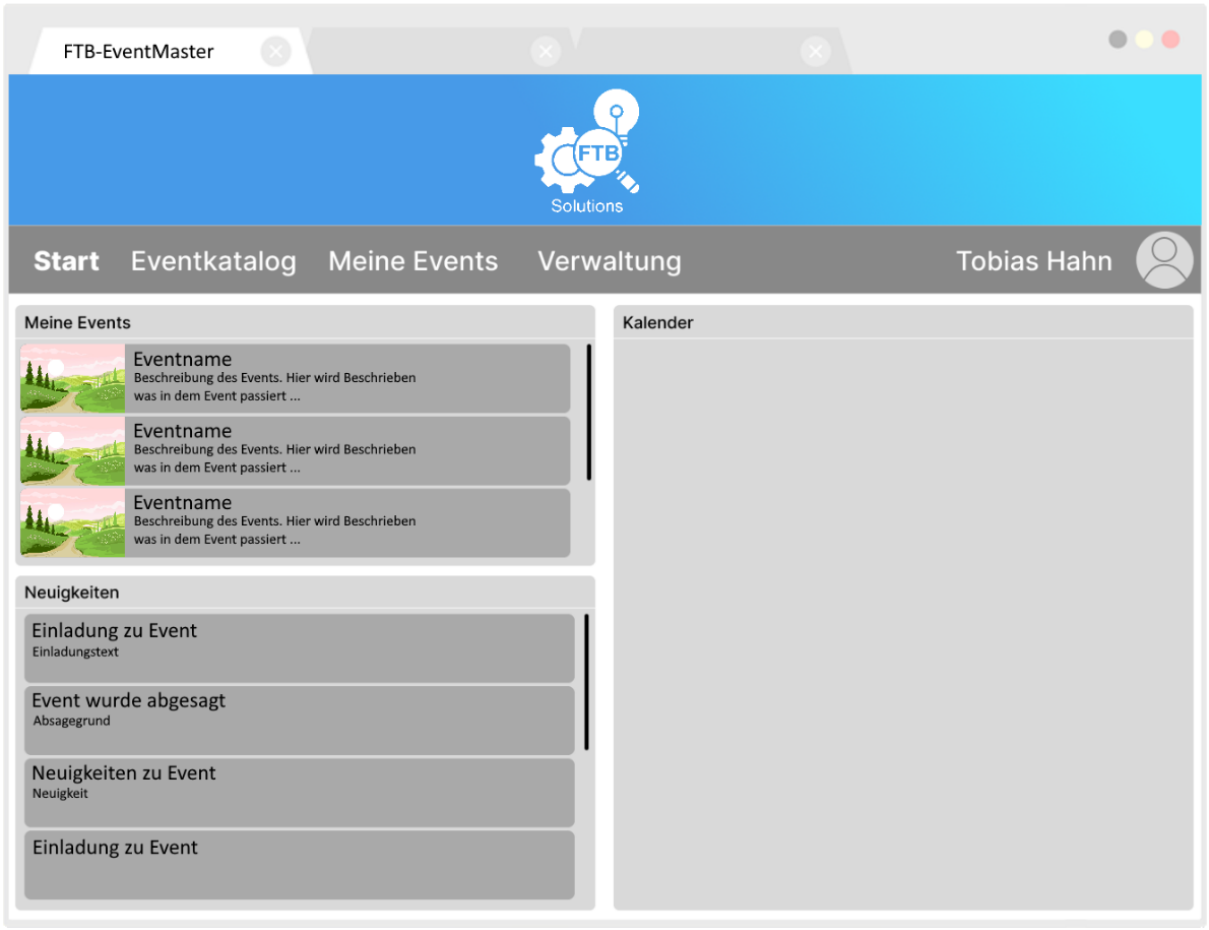

Abbildung 1: GUI-Konzept – Hauptseite [6]

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.

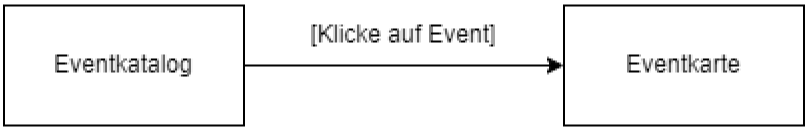

Diagramm 8: Zustandsdiagramm - Eventkatalog

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.

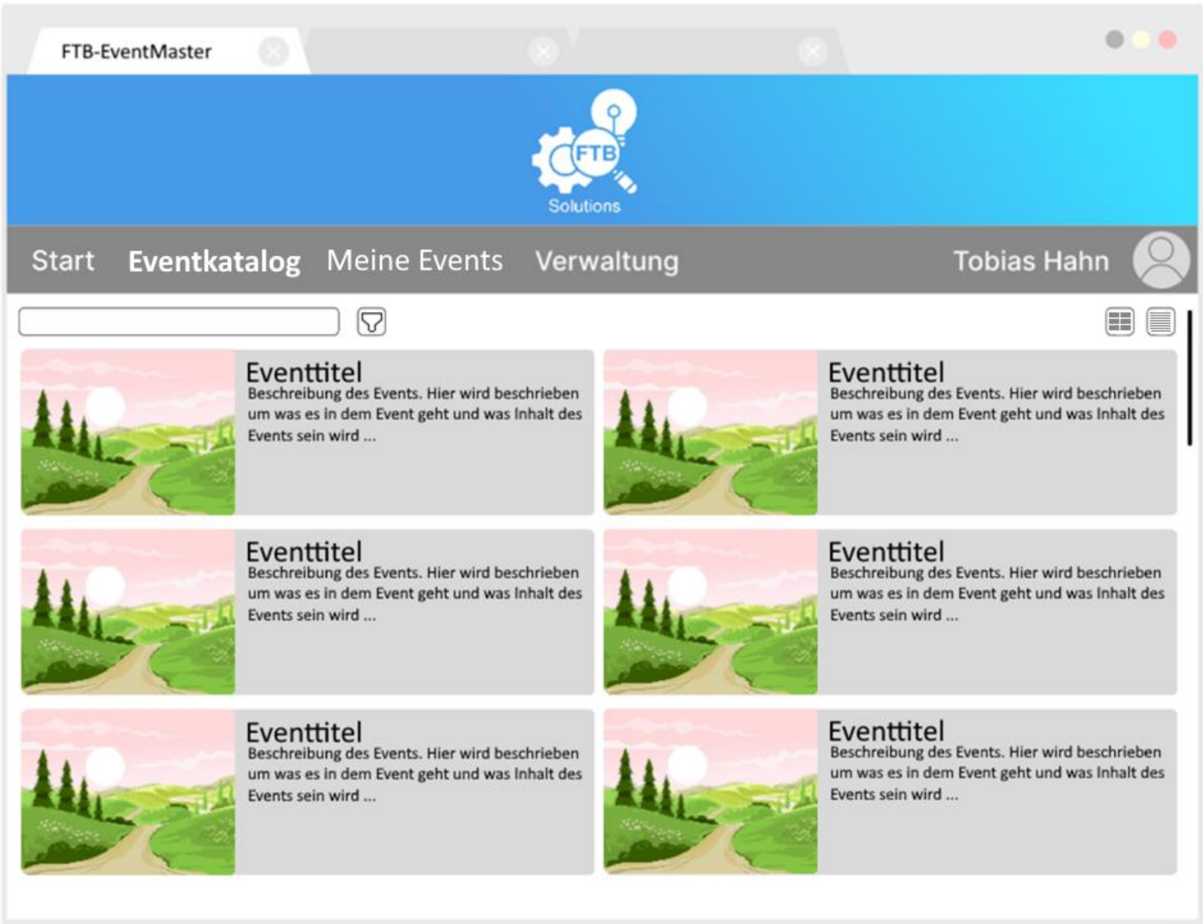

Abbildung 2: GUI-Konzept – Eventkatalog [7]

<u>Meine Events</u>

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.

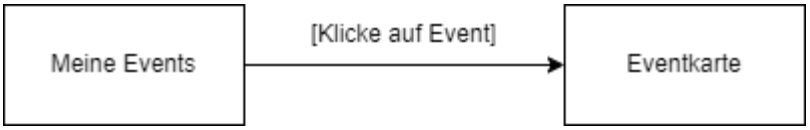

Diagramm 9: Zustandsdiagramm - Meine Events

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.

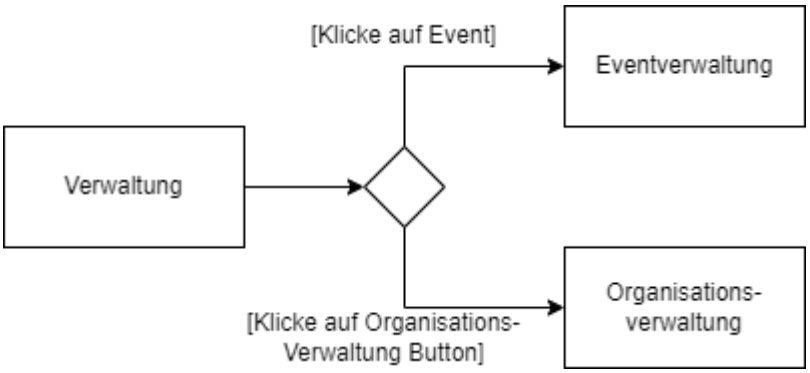

Diagramm 10: Zustandsdiagramm - Verwaltung

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.

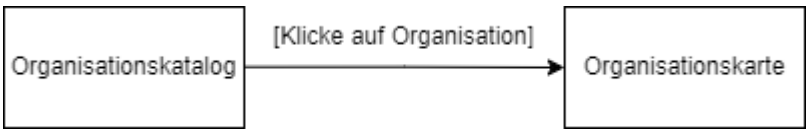

Diagramm 11: Zustandsdiagramm - Organisationskatalog

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.

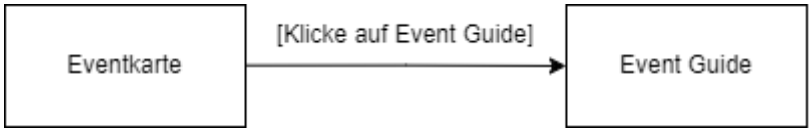

Diagramm 12: Zustandsdiagramm - Eventkarte

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. Controller, Services und DataAccess

An dem Komponentendiagramm (Diagramm 3) ist bereits zu erkennen, dass durch die einzelnen Komponenten auch horizontale Pakete festgelegt werden können. Beispielsweise können *UserController*, *UserService* und *UserRepository* zu einem Paket *User-Management* zusammengefasst werden. Dieses Paket wäre dann für die Verwaltung von Usern verantwortlich. Die Controller erhalten dabei die Daten über HTTP Requests durch die UIServices.

Im Folgenden werden diese Pakete aufgelistet und genauer dargestellt.

EventManagement

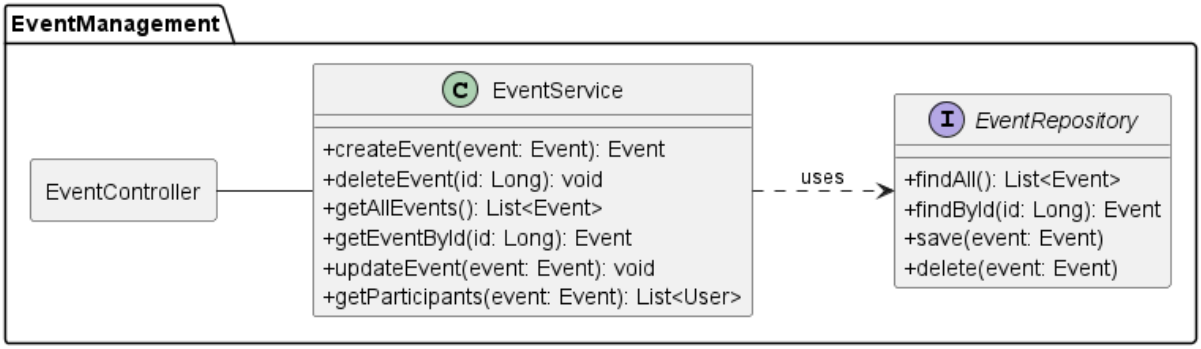

Diagramm 13: Paket "EventManagement" [10]

Die Events werden von dem Paket *EventManagement* verarbeitet. Dabei ist der EventService grundlegend für die Erstellung, die Bearbeitung und das Löschen von Events zuständig. Die zu verarbeitenden Daten werden über HTTP Request an den Controller gesendet, der auf Basis des API-Aufrufes die entsprechende Funktion im EventService aufruft. Die Methode getParticipants fasst die Teilnehmer, die über eine Gruppe (EventGroup) teilnehmen und die sich selbst angemeldet haben (UserEventRole) zusammen.

Im Folgenden wird ein Sequenzdiagramm (Diagramm 14) für das Szenario *Organisator erstellt ein Event* dargestellt. In diesem soll kenntlich gemacht werden, wie die Komponenten dieses Paketes kommunizieren.

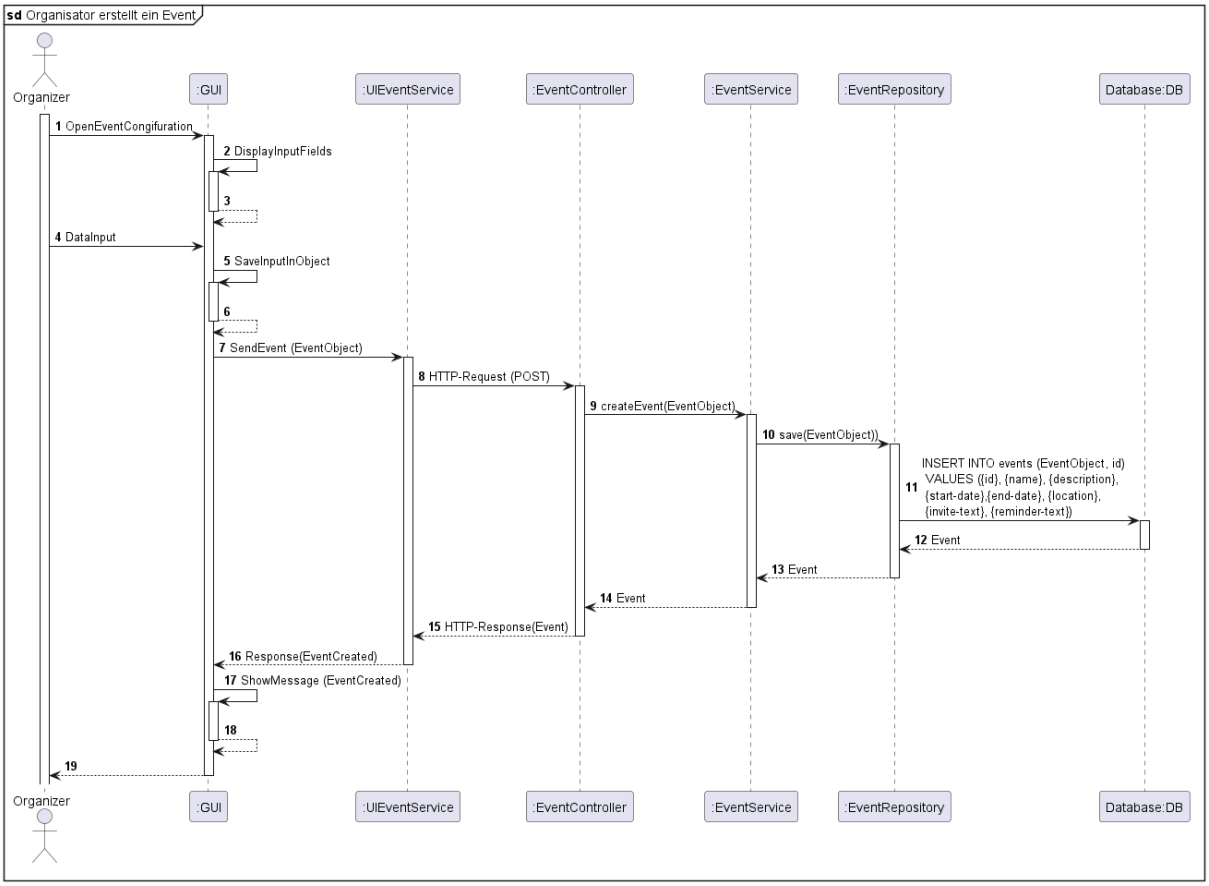

Diagramm 14: Sequenzdiagramm zum Szenario „Organisator erstellt Event“ [11]

Gemäß den getroffenen Designentscheidungen (Diagramm 13) benötigt der EventController ein Event Objekt. Dieses wird vom Frontend erzeugt, indem der Organistor in der Eventerstellung seine Konfiguration tätigt und diese übermittelt. Nachdem das Event-Objekt von der REST-API entgegengenommen und validiert wurde, wird das Event-Objekt mit dem Aufruf der Methode des EventService *createEvent(Event)* an den EventService weitergegeben. Die Implementierung des EventServices sorgt in diesem Szenario dazu, dass die Methode *save(Event)* des EventRepositories aufgerufen wird. Dieses Repository persistiert das EventObjekt und seine Attribute in einer Event-Tabelle mittels SQL-Befehle in der Datenbank.

UserManagement

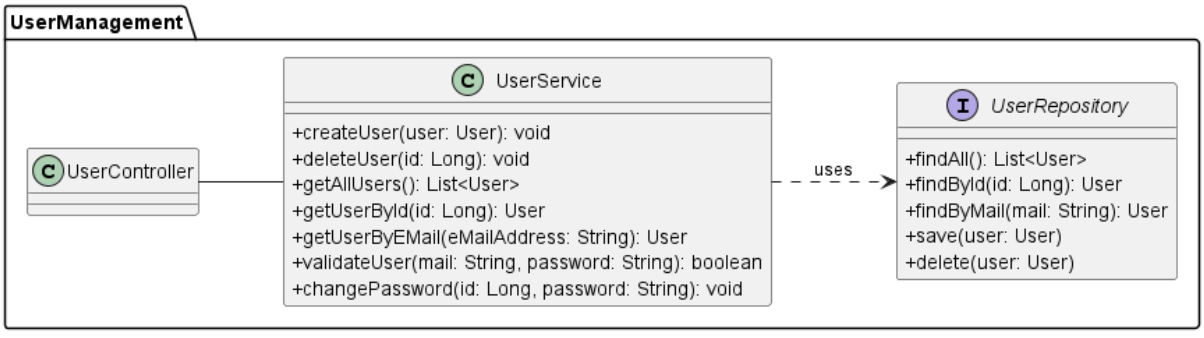

Diagramm 15: Paket "UserManagement" [12]

Die User werden in dem Paket *UserManagement* verarbeitet. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *UserController* gesendet und auf Basis des API-Aufrufes der entsprechenden Funktion im *UserService* zugeteilt. Dieser Service erlaubt es neben dem Erstellen, dem Bearbeiten und Löschen eines Users auch weitere Aktionen, wie die Authentifizierung eines Users.

Im Folgenden wird ein Sequenzdiagramm (Diagramm 16) für das Szenario User meldet sich an dargestellt. In diesem soll kenntlich gemacht werden, wie die Komponenten dieses Paketes kommunizieren.

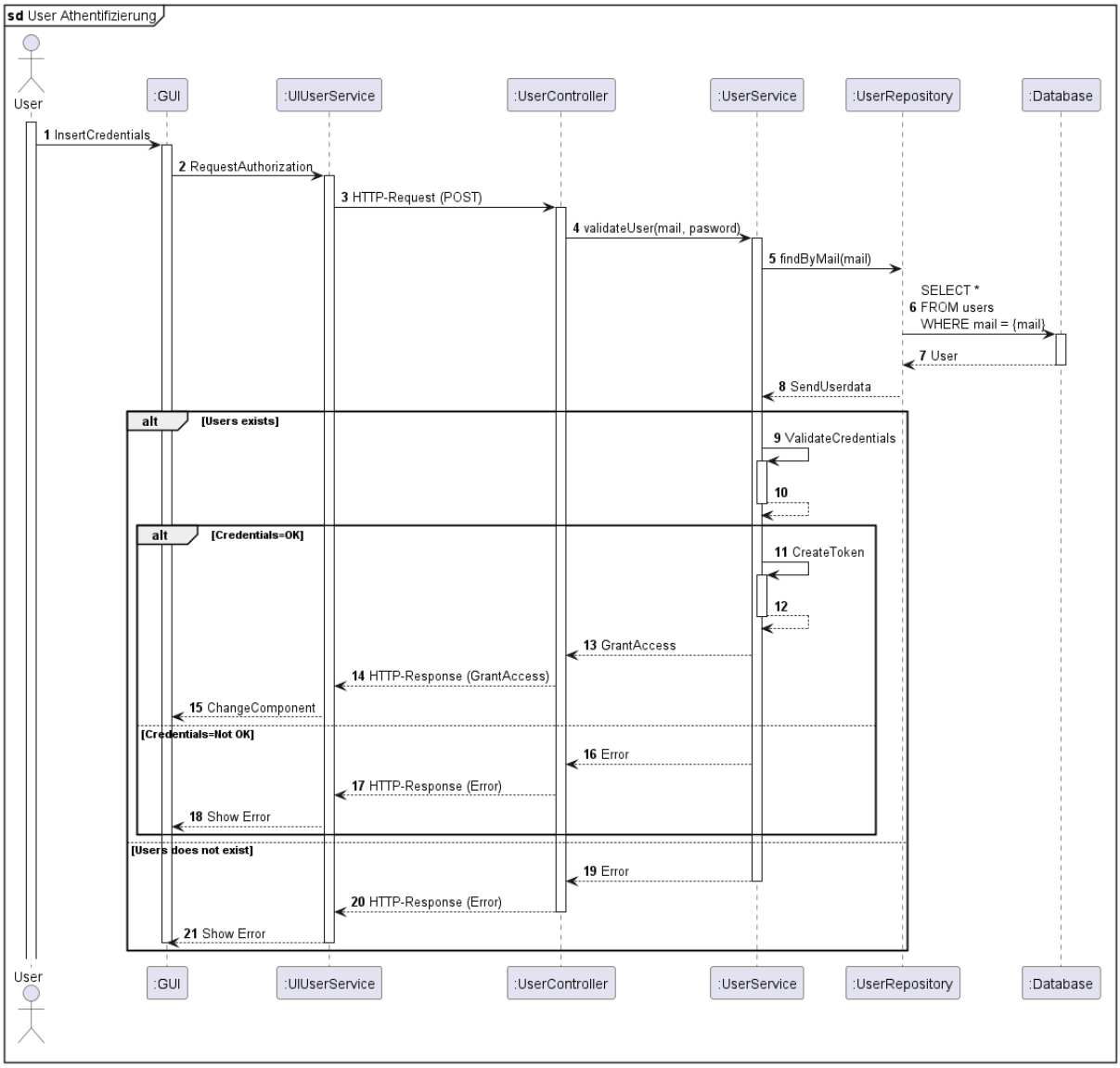

Diagramm 16: Sequenzdiagramm zum Szenario „User meldet sich an“ [13]

Meldet sich ein Nutzer in der Anwendung an, werden seine Anmeldedaten mit einem HTTP Post Request an den *UserController* gesendet. Nachdem die Daten vom *UserController* entgegengenommen und validiert wurden, werden diese weiter an den *UserService* gesendet. Die Implementierung im *UserService* sorgt dann dafür, dass die die Validierung der Mail und des Passworts durchgeführt wird. Dazu wird das Repository mit *findByMail(mail)* aufgefordert

in der Datenbank mittels SQL einen Eintrag zu finden. Sollte ein User mit dieser Mail existieren, wird nach erfolgreicher Validierung der Mail und des Passworts ein Token erstellt, welcher einen User eindeutig identifiziert und seine weiteren Anfragen autorisiert.

OrganizationManagement

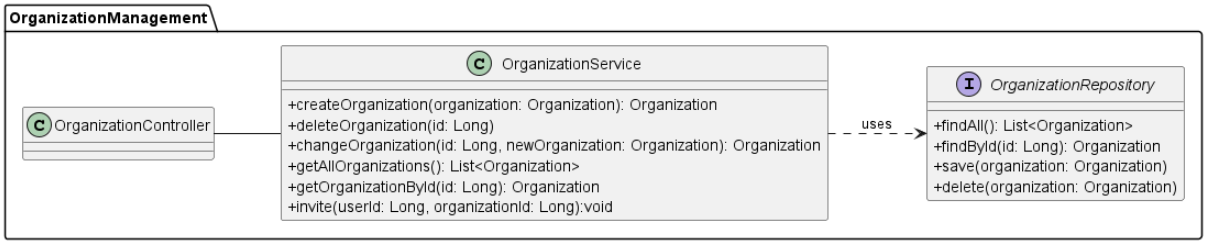

Diagramm 17: Paket "OrganizationManagement" [14]

Die Organisationen werden in dem Paket *OrganizationManagement* verarbeitet. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *OrganizationController* gesendet und auf Basis des API-Aufrufes der entsprechenden Funktion im *OrganizationService* zugeteilt. Dieser Service erlaubt neben dem Erstellen, Bearbeiten und Löschen einer Organisation auch weitere Aktionen, wie das Abrufen weiterer Daten einer oder mehrer Organisation/-en. Zudem wird auch hier die Möglichkeit geboten, dass Benutzer zu einer Organisation eingeladen werden, ohne dass sie sofort zur Organisation hinzugefügt werden. Das letztendliche Hinzufügen erfolgt in dem folgenden Paket *RoleManagement*.

RoleManagement

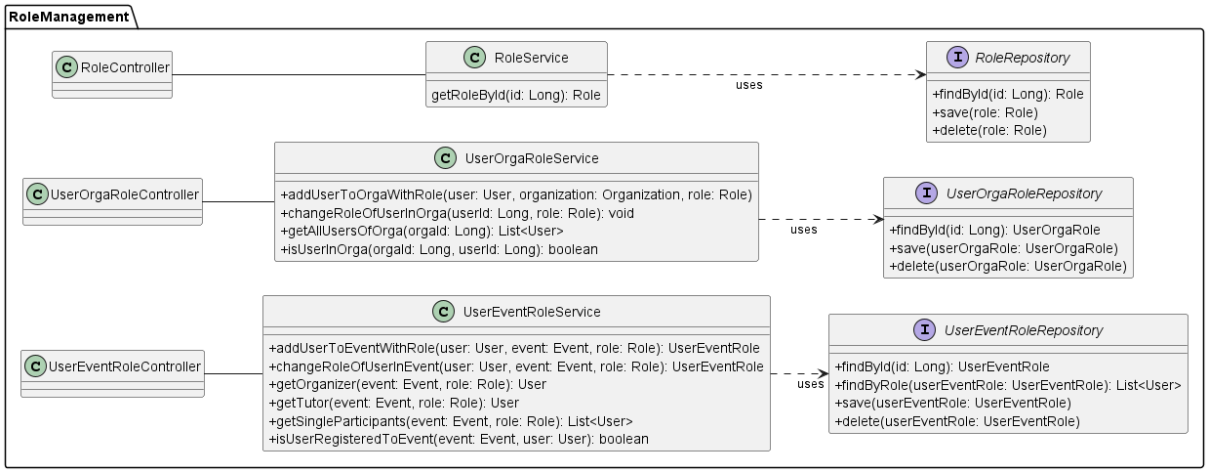

Diagramm 18: Paket "RoleManagement" [15]

Das Paket *RoleManagement* enthält eine umfassendere Funktionalität als die bereits aufgezeigten Pakete. In diesem Paket werden alle Vorgänge, die mit Rollen arbeiten zusammengefasst. Dabei handelt es sich um Rollen, Rollen eines Users in einer Organisation und Rollen eines Users in einem Event. Im Folgenden werden die jeweiligen Teilkomponente kurz erklärt.

Role:

RoleController, -Service und -Repository dienen im Allgemeinen lediglich dazu, eine Rolle eines Users durch seine UserId in Erfahrung zu bringen. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *RoleController* gesendet. Dieser ruft die einzige Funktion im *RoleService* auf und gibt die dem User zugehörige Rolle zurück.

UserOrgaRole:

UserOrgaRoleController, -Service und -Repository dienen dazu, User zu einer Organisation mit einer bestimmten Rolle zuzuweisen. Die Rolle eines Users in einer Organisation kann bei Bedarf geändert werden. Darüber hinaus läuft auch hierüber die Abfrage, ob ein User in einer Organisation ist. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *UserOrgaRoleController* gesendet. Dieser ruft auf Basis des API-Aufrufes die entsprechende Funktion des *UserOrgaRoleService* auf.

UserEventRole:

UserEventRoleController, -Service und -Repository dienen dazu, einen User zu einem Event mit einer bestimmten Rolle zuzuweisen. Vom Prinzip her, läuft dies also ähnlich zu UserOrgaRole ab, nur dass nun mit Events statt Organisationen gearbeitet wird. Es ist also auch hier möglich, abzufragen, wer einem Event zugewiesen ist, bzw. bei welchen Events der Benutzer angemeldet ist. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *UserEventRoleController* gesendet. Dieser ruft auf Basis des API-Aufrufes die entsprechende Funktion des *UserEventRoleService* auf.

InformationDistribution

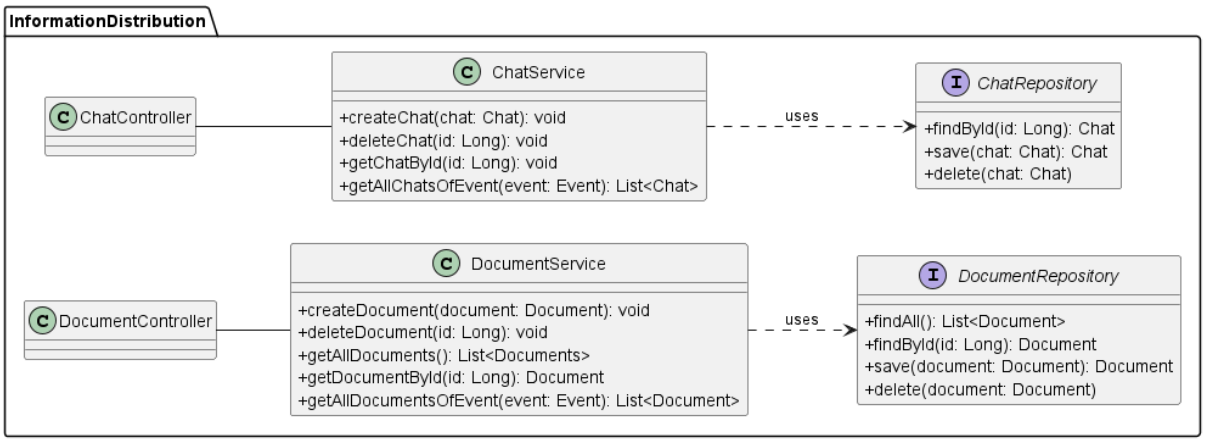

Diagramm 19: Paket "InformationDistribution" [16]

Die Informationen, die den Usern während eines Events zur Verfügung gestellt werden, werden in dem Paket *InformationDistribution* verarbeitet. Dazu zählen das Speichern und Abrufen von Chatnachrichten, sowie von Dateien. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *Chat*- bzw. *DocumentController* gesendet und auf Basis des API-Aufrufes der entsprechenden Funktion im *Chat*- bzw. *DocumentController* zugeteilt.

SurveyManagement

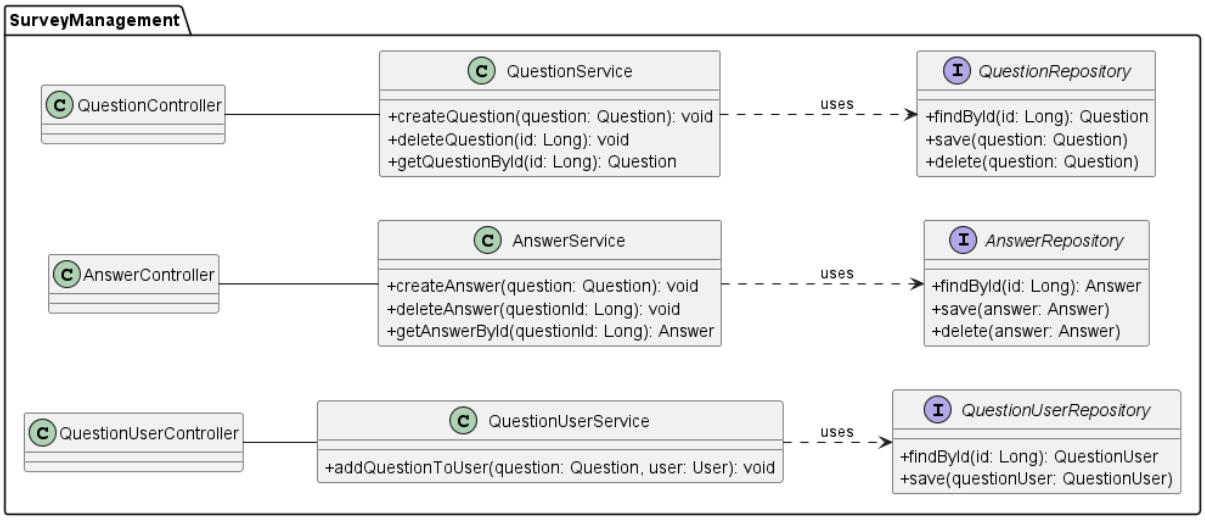

Diagramm 20: Paket "SurveyManagement" [17]

Das Paket *SurveyManagement* enthält wie das Paket *RoleManagement* eine umfassendere Funktionalität. In diesem Paket hängen die Teilkomponente zusammen und befassen sich mit den Fragebögen, die ein Organisator oder Tutor erstellen kann. Bei den Teilkomponenten handelt es sich um Question, Answer und QuestionUser. Im Folgenden werden die jeweiligen Abschnitte kurz erklärt.

Question:

QuestionController, -Service und -Repository dienen dem Anlegen einer Frage durch einen Organisator. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *QuestionController* gesendet. Dieser ruft auf Basis des API-Aufrufes die entsprechende Funktion des Services auf, wie z.B. *createQuestion* für das Erstellen einer Frage.

Answer:

AnswerController, -Service und -Repository dienen dem Anlegen einer Antwortmöglichkeit bei multiple-choice Fragen durch einen Organisator. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *AnswerController* gesendet. Dieser ruft auf Basis des API-Aufrufes die entsprechende Funktion des Services auf. Für den speziellen Fall einer multiple-choice Frage, wird hierbei die Funktion *createAnswer* aufgerufen, welche einer Question Anwortmöglichkeiten aus dem *AnswerRepository* zuweist.

QuestionUser:

QuestionUserController, -Service und -Repository dienen dem zuteilen einer Frage zu einem User (Ein User ist in diesem Fall ein Teilnehmer eines Events, für welches diese Frage erstellt wurde).

MailManagement

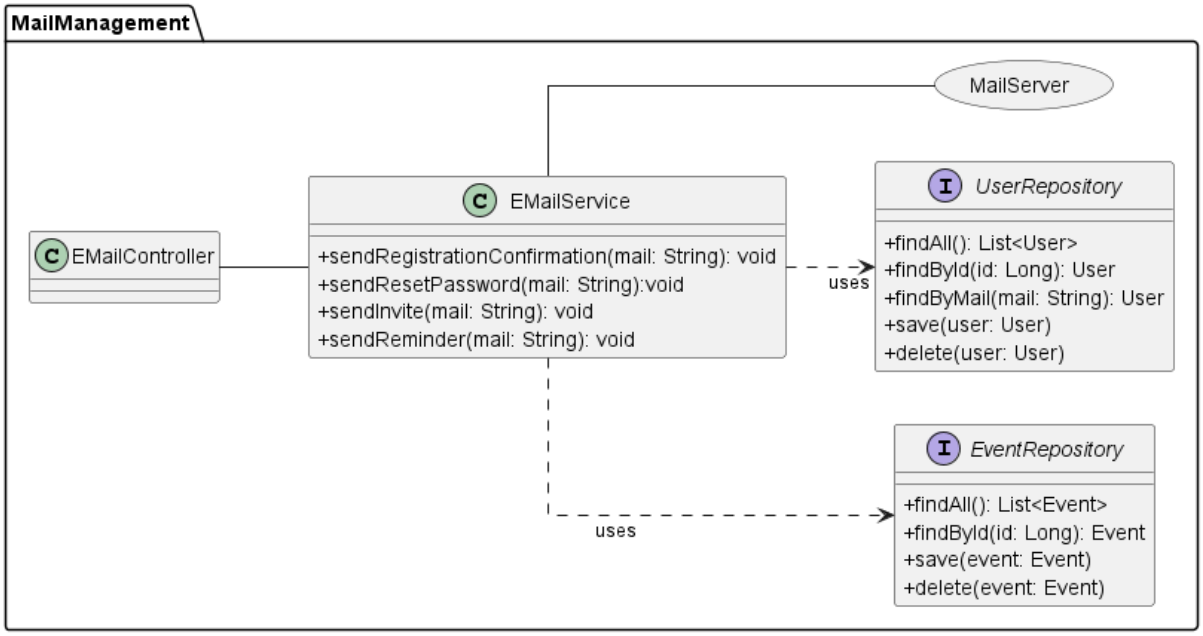

Diagramm 21: Paket "MailManagement" [18]

Der von der Anwendung ausgehende Mail-Verkehr wird in dem Paket *MailManagement* verarbeitet. Dabei werden die zu verarbeiteten Daten über HTTP Request an den *EMailController* gesendet und auf Basis des API-Aufrufes der entsprechenden Funktion im *EMailService* zugeteilt. Dieser Service erlaubt es Mails bzgl. Registrierung, Einladung (sowohl zu einer Organisation als auch zu einem Event), Erinnerungen und Passwort Rücksetzungen zu senden.

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 13	Package_EventManagement.png
[11]	Diagramm 14	Sequenzdiagramm_Event_erstellen.png
[12]	Diagramm 15	Package_UserManagement.png
[13]	Diagramm 16	Sequenzdiagramm_Userauthentifizierung.png
[14]	Diagramm 17	Package_OrganizationManagement.png
[15]	Diagramm 18	Package_RoleManagement.png
[16]	Diagramm 19	Package_InformationDistribition.png
[17]	Diagramm 20	Package_SurveyManagement.png
[18]	Diagramm 21	Package_MailManagement.png