Pflichtenheft

UrVent
A2
20.10.2019
Endversion

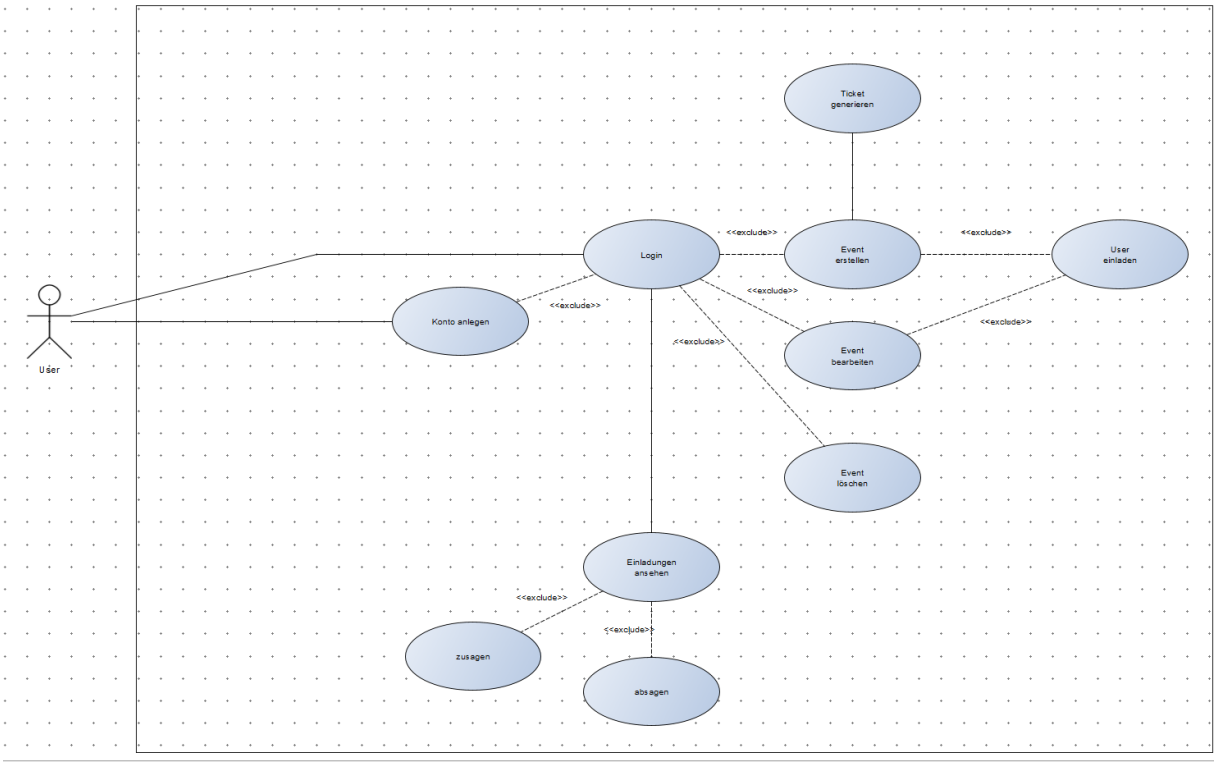
Wichtige Hinweise:

- ***Die in diesem Dokument aufgeführten Beschreibungen in Kursivschrift (außer einigen Abschnittüberschriften) sind beispielhaft und erläuternd und müssen aus dem fertiggestellten Bericht entfernt werden.***
- ***Kennzeichnen Sie welcher Abschnitt dieses Berichts von welchem Teammitglied erstellt wurde.***

1 Aufgabenstellung und Zielbestimmung

Der Anwendungsfall des Projekts stellt sich wie folgt dar: Als Veranstalter eines Events ist es möglich in „UrVent“ ein Event zu erstellen, um so Werbung für eine Veranstaltung zu machen. Außerdem kann man als Veranstalter die Teilnehmeranzahl einsehen, um so eine Übersicht zur potenziellen Besucheranzahl zu bekommen.
Als „Endkunde“ kann man Veranstaltungen in der Nähe einsehen und auf diese zugreifen. Dort ist es dann möglich eine Beschreibung der Veranstaltung auszurufen und an einer Veranstaltung teilzunehmen bzw. zuzusagen. Außerdem sind der Ticketerwerb und das Beschauen von Bildern der vergangenen Veranstaltungen möglich.

2 Anwendungsfall-Diagramm



3 Funktionale Anforderungen

- Als User möchte ich mit meiner E-Mail-Adresse und einem Passwort ein Konto anlegen können, um Zugang zu „UrVent“ zu erhalten.

- Als User möchte ich mich mit meiner E-Mail-Adresse und meinem Passwort anmelden können.

- Als User möchte ich eine Veranstaltung anlegen können, um eine Veranstaltung zu planen und zu veranstalten. Hierbei soll automatisch ein Ticket mit QR-Code generiert werden. Dabei kann ich sowohl private als auch öffentliche Veranstaltungen erstellen.

- Als User möchte ich von mir erstellte Veranstaltungen bearbeiten können, um Änderungen vorzunehmen und um ggf. Informationen hinzufügen zu können.

- Als User möchte ich von mir erstellte Veranstaltungen löschen können, um Veranstaltungen wieder abzusagen.

- Als User möchte ich alle Veranstaltungen einsehen können, zu denen ich eingeladen worden bin, um mich über diese Einladungen und Veranstaltungen zu informieren.

- Als User möchte ich auf eine Einladung mit einer Zu- oder Absage reagieren können, um den Veranstalter zu informieren.

- Als User möchte ich andere User zu einer von mir erstellten Veranstaltung einladen können.

4 Qualitätsanforderungen

Qualitätsziele anhand einer Tabelle bestimmen, wie unten angeführt:

Systemqualität	Sehr gut	Gut	Normal	Nicht relevant
Funktionalität		X		
Zuverlässigkeit	X			
Benutzbarkeit	X			
Effizienz		X		
Wartbarkeit		X		
Portabilität			X	

Tabelle 1: Qualitätsanforderungen

5 Abnahmekriterien

- Alle Screens sind zu implementieren
- Funktionalitäten zuverlässig und einfach zu bedienen
- JSON-Standard wird eingehalten
- Alle Use-Cases sind zu implementieren
- Views, ViewModels & Models sind zu kapseln
- Klassen, Methoden, usw. sind nach JavaDoc-Standard zu kommentieren
- Kritische Funktionen sind über Unit-Tests zu testen
    - Client-API-Verbindung
    - JSON-String zu JSON-Object und umgekehrt
    - EventHandlers
    - ViewModel-Funktionen
- Clean Code
- Allman/BSD - Coding Style
- Einhaltung und Funktionalität auf der vorgegebenen Java- & NetBeans-Version gegeben

6 Übersicht der Beziehungen zwischen den Klassen

Klassendiagramm aufgeteilt zur besseren Lesbarkeit (hochauflösende Version separat):

6.1 Allgemeines

Die Architektur von UrVent orientiert sich lose am MVVM-Pattern, das heißt es soll eine möglichst lose Kopplung zwischen Model, View & ViewModel erreicht werden, was die Testbarkeit von einzelnen Modulen stark vereinfacht, eine starke Modularisierung sowie effektive Datenkapselung ermöglicht.

Das ViewModel stellt den zentralen Bestandteil der Architektur dar. Im Grunde genommen lässt sich sagen, dass ein ViewModel einen einzelnen Screen des Programms repräsentiert und soll die Präsentationslogik implementieren sowie die für das jeweilige ViewModel relevanten Models als Feld/Member enthalten. Außerdem soll das ViewModel die für sich relevanten EventHandler (je nach Quelle auch Actions genannt) erzeugen.

Views werden durch. fxml Dokumente des JavaFX Frameworks implementiert und sollen durch Bidirektionale Datenbindung an das entsprechende ViewModel gebunden werden. [1]

Models stellen „dumme“ Objekte aus der Datenbank dar, d.h. sie enthalten selbst nur minimale Funktionalität, wie getters und Konstruktor und sollen aus einem JSONObject [2] erzeugt werden, welches wiederum aus einem String, der dem JSON Standard entspricht und von der API geliefert wird, erzeugt wird.

Zusammenfassend lässt sich die Beziehung zwischen den Klassen wie folgt beschreiben:

View↔ViewModel↔Model

6.2 Model

Um einzelne Objekte, die in der Datenbank abgelegt sind wie z.B. einen User, als Java-Objekte zu modellieren, sollen alle derartigen Klassen von einer gemeinsamen Parent-Klasse erben (Model), die allgemeine Methoden und Felder der einzelnen Modellklassen zur Verfügung stellt. Ein Model soll dabei Datenbank-Objekte 1:1 abbilden, d.h. jede Model Klasse hat jede Spalte in der entsprechenden Tabelle als Feld/Member. Jedes Model ist von einem ViewModel zu erzeugen und soll in diesem referenziert werden.

Die von Model erbenden Klassen sollen den Default-Konstruktor überschreiben und es ermöglichen ein Objekt ohne weitere Umwandlung aus einem JSON-Objekt zu erzeugen, um eine simple und standardisierte „Sprache“ zwischen API und Client zu ermöglichen und um das Projekt möglichst modular aufzubauen.

Felder/Members eines Models sind so zu Kapseln, dass sie die ReadOnly-Eigenschaft erfüllen, d.h. das Feld ist als private zu definieren und die entsprechende Getter-Methode als public. So sollen die vom Model gehaltenen Informationen vor Änderungen geschützt werden und die Einhaltung des Patterns forciert wird.

Zusätzlich soll es möglich sein, ein Model wieder in einen JSON-String umzuwandeln, um die Kommunikation mit der API zu ermöglichen.

6.3 View

Views werden durch. fxml und CSS Dokumente modelliert und stellen ausschließlich UI-Elemente dar. Es liegt in der Natur der Sache, dass ein View keinerlei Funktionalität enthalten kann und nur zur Definition des UI-Layouts dient.

Änderungen am View sind zur Laufzeit nur durch das jeweilige ViewModel über Bidirektionales Databinding vorzunehmen. [1]

6.4 ViewModel

Das ViewModel stellt das Herzstück des Systems dar und implementiert die Präsentationslogik, welche in seiner „initialize()"-Methode sowohl die relevanten Models als auch die relevanten EventHandler erzeugt und verbindet Model & View.

Ein ViewModel lässt sich am einfachsten als Code-Repräsentation eines „Screens" beschreiben und modelliert den Zustand der einzelnen UI-Elemente. Bidirektionales Databinding ermöglicht es Änderungen am UI nahtlos im ViewModel abzubilden, wodurch die Komplexität des Programms stark vermindert wird.

ViewModels implementieren das JavaFX-Interface „Initializable", was es ermöglicht UI-Elemente einfach zu referenzieren.

6.5 EventHandler

Klassen, die von der JavaFX-Klasse Eventhandler erben, implementieren die Funktionalität von einzelnen UI-Elementen wie z.B. einem Button. [3]

6.6 Tools

Für allgemeine Funktionalitäten wie z.B. Aufbau einer Verbindung zur API, sind „Toolklassen" nach Bedarf zu implementieren, wobei zu beachten ist, dass diese niemals instanziiert werden, d.h. sie sollen ausschließlich statische Methoden zur Verfügung stellen, wie z.B. Die Klasse HTTP.

Die Klasse HTTP stellet statische Methoden zur Verfügung, die von anderen Klassen aufgerufen werden sollen, um mit der API zu kommunizieren. Dabei sollen Requests mithilfe einer HttpURLConnection[4] an die API im JSON-Format gestellt werden und die API Objekte aus der Datenbank als JSON-String zurückgeben, aus dem dann ein JSON-Object erzeugt wird, aus welchem wiederum Objekte vom Typ Model (User, Event, usw.) erzeugt werden.

7 Grobbeschreibung der Bedienung bzw. des Ablaufs

Der Nutzer muss sich zunächst über das Registrierfenster registrieren, um sich anschließend mit diesen Daten im Anmeldebereich einloggen zu können. Daraufhin kann der Nutzer die komplette Funktionalität der Software über das Hauptfenster nutzen, indem er auf den von ihm gewünschten Bereich klickt. Dort hat der Nutzer mehrere Möglichkeiten. Er kann über einen Bereich seine Einladungen einsehen und entsprechend auf diese Veranstaltungseinladungen reagieren, indem er an- oder ablehnt. Des Weiteren hat der Nutzer die Möglichkeit eine Liste einzusehen, in der alle von ihm zugesagten Veranstaltungen aufgelistet sind.

Grundsätzlich hat der Nutzer ebenso die Möglichkeit Informationen zu einer Veranstaltung abzurufen, indem er die entsprechende Veranstaltung anklickt und nun eine Übersicht über alle relevanten Informationen über das Event erhält. Zudem hat der Nutzer die Möglichkeit eigene Events zu erstellen oder zu bearbeiten. Hierzu öffnet er das entsprechende Fenster und füllt nötige Informationen aus, um dieses Event anschließend in der Liste der eigenen Veranstaltungen einsehen zu können. In diesem Bereich kann der Nutzer jeden Event auch löschen. So ist der Nutzer durch den Homescreen in der Lage alle verfügbaren Funktionalitäten übersichtlich auszuwählen und in entsprechende Bereich zu gelangen.

8 Literatur

[1] https://www.dummies.com/programming/java/javafx-binding-properties/
[2] https://www.mkyong.com/java/json-simple-example-read-and-write-json/
[3] https://www.tutorialspoint.com/javafx/javafx_event_handling.htm
[4] https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html