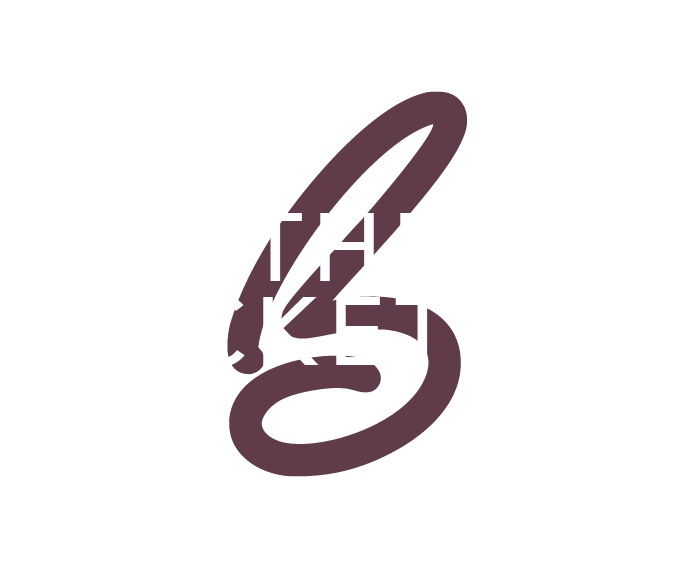
The Bucketlist

Software Entwicklung 2, Sommersemester 2021

<https://gitlab.mi.hdm-stuttgart.de/st093/bucketlist>





**Pia Schilling,** ps149@hdm-stuttgart.de

**Sara Tietze,** st093@hdm-stuttgart.de

**Merve Özdemir,** mo064@hdm-stuttgart.de

1. Kurzbeschreibung des Projekts

Unser Projekt ist, wie der Name schon verrät eine **Bucketlist** (zu Deutsch: Wunschliste).

Wenn man das Programm startet, kann man zuallererst eine **Eventliste** mit Ablaufdatum erstellen. Diese wird dann im nächsten Schritt mit **Events** gefüllt (z.B. Spanisch lernen, nach Mexico reisen, …) die man ebenfalls selbst anlegt. Man kann hier Titel und Kategorie auswählen. Wobei diese beiden Pflichtangaben sind.

Es gibt 13 verschiedene **Kategorien**: Skills, Travel, Culture, Shopping, Lifegoals, Culinary, Education, Sport, Hobby, Family, Relationship, Friends und No Category speziell für Events, die zu keiner der anderen Kategorien zugeordnet werden können.

Sobald man ein Event abgeschlossen hat, kann man dieses dementsprechend markieren. Es kann eine **abschließende Beschreibung** und ein **Bild** hinzugefügt werden.

In einer eigenen Scene kann man sich die **Events je Kategorie**, als eigene Liste anzeigen lassen.

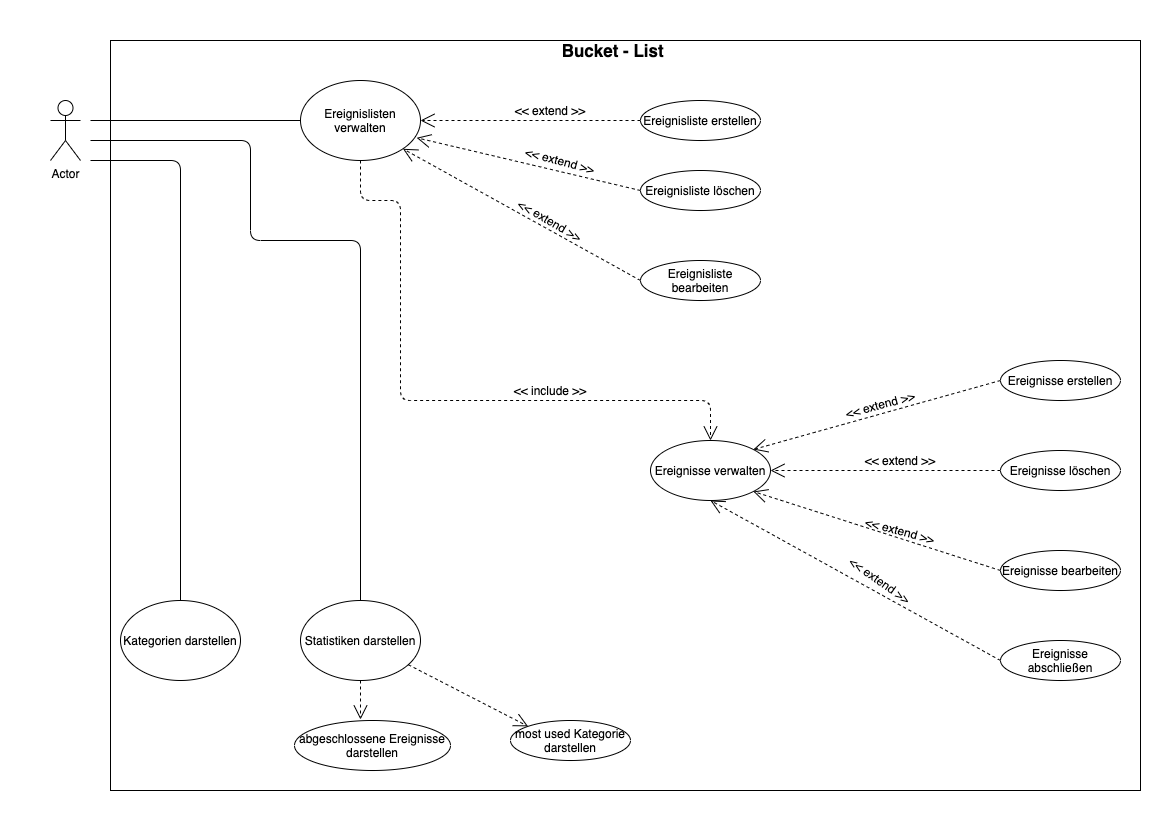
Außerdem kann man separat **Statistiken** aufrufen. Hier wird gezeigt wie viele Events angelegt oder abgeschlossen wurden, die Anzahl der angelegten Eventlisten, der Fortschritt jeder einzelnen Liste und wie viele Tage Zeit man noch hat die Liste abzuschließen. Zudem kann man sich in einer übersichtlichen **PieChart** anzeigen lassen, wie viele Events in jeder Kategorie erstellt wurden.

Die Main-Methode befindet sich in der Klasse **Main.**

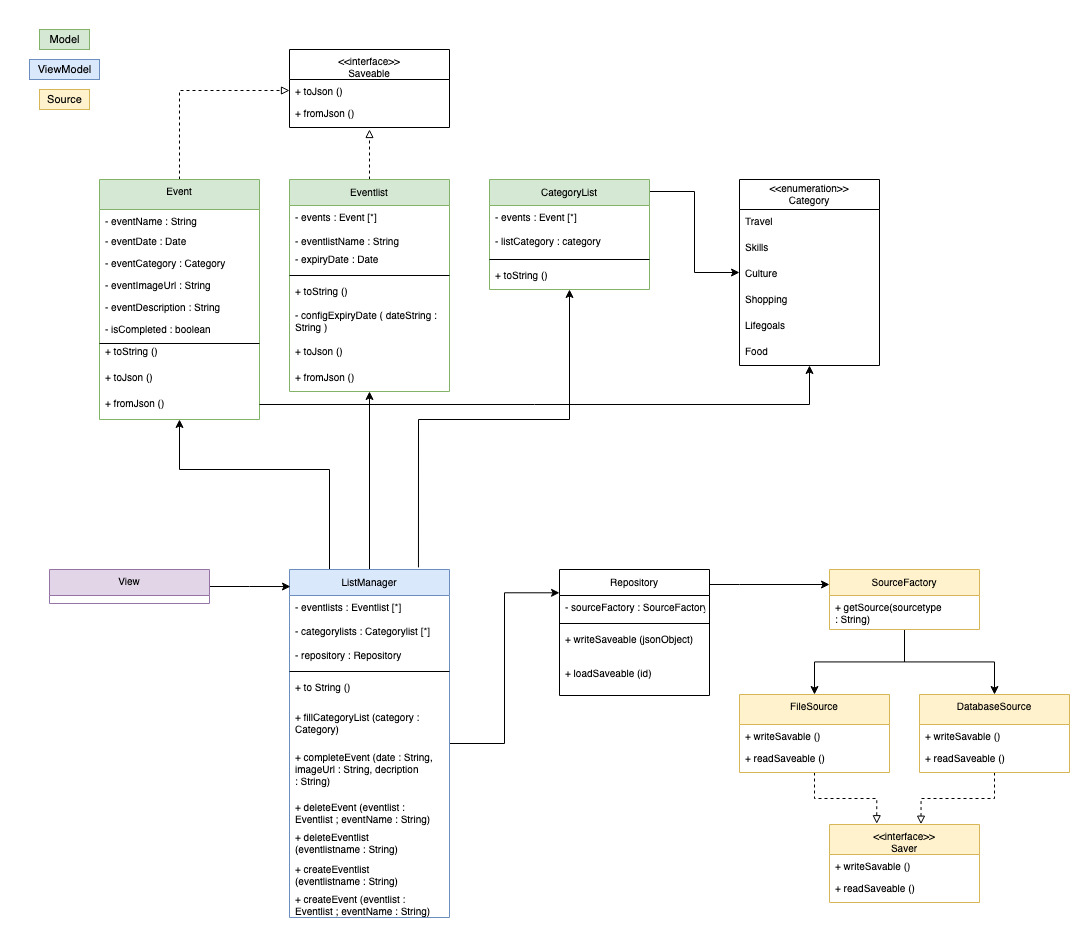
1. UML

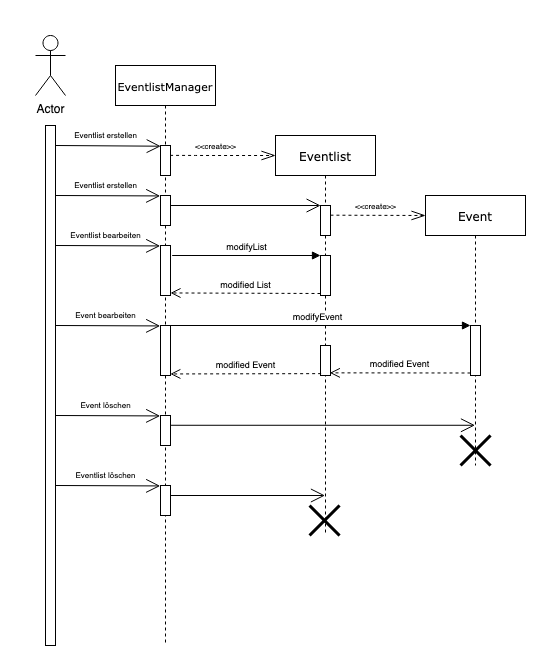
Die Diagramme befinden sich im Git-Repository.

**Use Case Diagram**

****

**Class Diagram**

****

**Sequence Diagram**

1. Stellungnahmen

Architektur

**Singelton Access:**

**Interface:** Wir haben mehrere Interfaces erstellt, zum Beispiel das Listener Interface, das dafür sorgt, dass alle Controller Klassen darüber informiert werden, wenn ein Model-Zustand sich ändert oder das Narrator Interface, welches dafür sorgt, dass alle Klassen, die dieses implementiert haben von GUI Klassen observiert werden können.

**Factory:** Ist ein Hilfsmittel, um in unserem Fall die Datenquelle zu bestimmen. Die Methode getSource liefert die passende Quelle Filesource oder Databasesource zurück.

**Enum:** Unsere Enums beinhalten die verschiedenen Event-Kategorien, die man zu jedem Event zuordnen muss und die Sourcetype der Daten. Dadurch wird eine eindeutige Wahl gewährleistet.

**Vererbung:** Alle Klassen im Unterordner Customnodes von View/Controller erben von der Klasse Anchorpane.

**Ordnerstruktur:**

Im Ordner de.hdmstuttgart.mi.bucketlist befinden sich neben der Main-Klasse, noch sechs Unterordner.

**Exceptions:** beinhaltet unsere eigenen Exceptionklassen

**Model:** beinhaltet das Enum Category, das Interface Narrator und andere Klassen die, die Attribute von Events und Eventlisten beinhalten

**ModelController:** beinhaltet die Controller Klassen, die die Eventlisten und Events steuern

**Persistence:** beinhaltet Klassen und Interfaces zur persistenten Datenspeicherung

**Threads:** beinhaltet unsere Thread Klasse

**View:** beinhaltet die Controller Klassen, die die fxml Dateien steuern und das Interface Listener

Clean Code

In den Klassen wird nirgends ein public member verwendet.

Tests

Wir haben im Ordner (test > java > de.hdmstuttgart.mi.bucketlist) mit **JUnit Tests** **Testklassen** erstellt, die die wichtigsten Methoden aus den Klassen auf ihre Funktion testen. Es gibt für eine bessere Übersicht auch hier Unterordner (Model, ModelController, Persistence).

GUI (JavaFX)

Die GUI haben wir mit dem **SceneBuilder** erstellt. Die jeweiligen fxml Dateien befinden sich im Ordner (scr > main > resources > fxml). Außerdem befinden sich die zugehörigen Controller Klassen im Ordner (scr > main > java > de.hdmstuttgart.mi.bucketlist > View).

Logging/Exceptions

**log.debug** >> Allgemeine Informationen wie z.B. der Start einer Methode

**log.error** >> Wenn z.B. kein passendes Objekt gefunden wird

**log.info** >> Informationen über den Erfolg einer Funktion

**Exception:** Wir haben einmal die **ElementAlreadyExistsException**, die von der Klasse Eventlist ausgeführt und aufgerufen wird, wenn z.B. der Titel einer Eventliste bereits vergeben ist. Und wir haben die **EmptyDirectoryException**, die von der Klasse FileSource ausgeführt und aufgerufen wird, wenn ein Verzeichnis leer ist oder nicht existiert.

UML

Wir haben mit einem **Use Case Diagramm** angefangen. Dadurch wurde uns bewusst, welche grundlegenden Funktionen wir für unsere Idee brauchen. Diese haben wir dann im Nachhinein im **Klassendiagramm** ausgearbeitet und erweitert. Außerdem haben wir ein **Sequence Diagramm** erstellt mit möglichen User Szenarien, die uns nochmal gezeigt haben, welche Funktionen eventuell noch fehlen könnten. Als wir fast fertig mit unserer Software waren, haben wir die Diagramme nochmal überarbeitet.

Threads

**PersistenceRunnable** erstellt Threads, die überprüfen, ob die persistente Datenspeicherung lauffähig ist und funktioniert.

Streams

Factories

Unser Programm besitzt im Ordner (scr > main > java > de.hdmstuttgart.mi.bucketlist > Persistance) eine Klasse namens **SourceFactory**. In dieser befindet sich die Methode getSource, die überprüft welchen **Sourcetype** eine Datei hat und diesen ausgibt.

1. Bewertungsbogen
2. Profiling Analyse

Stattdessen Nachdenkzettel vom Team gemeinsam ausgefüllt im Git-Repository.