Projekt: SportApp

Projekt-Portfolio

im Modul

DLMCSPSE01_D

„Projekt: Software Engineering“

Studiengang Informatik (Master 120 ECTS)

IU Internationale Hochschule

Bearbeiter: Jan Sauerland

Matrikelnummer: 10230630

Datum: 28.10.2024

Betreuender Dozent: Prof. Dr. Markus Kleffmann

INTERNATIONALE HOCHSCHULE

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Projektplan

1.1 Ziele, Umfang und angestrebtes Ergebnis

Ziel dieses Projektes ist es, eine Verwaltungssoftware für Sporttrainer zu erstellen, mit der Trainingseinheiten vorbereitet und geplant werden können.

Die Verwaltungssoftware enthält eine Datenbank, in der einzelne Trainingsübungen gespeichert werden können, um dann zu einer Trainingseinheit zusammengestellt werden zu können. Eine Trainingseinheit umfasst also mehrere thematisch oder sequenziell zusammenhängende Trainingsübungen. Mehrere Trainingseinheiten sollen dann zu einem Trainingsplan zusammengestellt werden können, wodurch man einen ganzen Trainingstag beschreiben kann. Die Trainingseinheiten bzw. -pläne sollen in der Datenbank gespeichert und auch in einer allgemeinen Form, z.B. als PDF, exportiert werden können, um diese mit anderen Personen zu teilen bzw. auch mobil verfügbar zu machen. Die einzelnen Übungen sollen ggf. auch mit Bildern oder Videos angereichert werden können, um diese detaillierter zu beschreiben. Außerdem sollen die Übungen anhand verschiedener Merkmale kategorisiert werden können, damit einfacher nach ähnlichen Übungen gesucht werden kann.

Die Verwaltungssoftware soll schließlich für den Sporttrainer eine Sammlung von bewährten Übungen und Trainingseinheiten sein, die immer wieder verwendet und neu zu Trainingsplänen kombiniert werden können.

1.2 Zielgruppe

Die Haupt-Zielgruppe des Projekts sind Sporttrainer, die mithilfe von Software ihre Trainingseinheiten organisieren möchten und Trainingspläne mit anderen Personen, z.B. Co-Trainern, teilen möchten.

Darüber hinaus könnte die Software auch für Sportlehrer relevant sein, die ihre Unterrichtsstunden darüber organisieren möchten.

1.3 Projektrisiken und Gegenmaßnahmen

Mögliche Risiken und zu ergreifende Gegenmaßnahmen sind in folgender Tabelle dargestellt.

Tabelle 1: Projektrisiken und Gegenmaßnahmen

Risiko	Gegenmaßnahme
Datenbank wird durch Video-Upload zu groß; Performanceverlust	Einbettung von (Youtube-) Links anstatt Upload in Datenbank
Anbindung des MySQL Datenbankserver verursacht Probleme	Umstellung auf einfache Anbindung einer SQL-Datenbankdatei über SQLite
Frontend mit PyQt verursacht Probleme; Geplante Funktionen sind nicht umsetzbar	Ausweichen auf Bibliothek „Tkinter“

Export des Plans als PDF ist nicht mit vertretbarem Aufwand möglich	Export als TXT- oder JPG-Datei
Export aller Inhalte eines Plans zu aufwändig oder technisch nicht möglich	Reduzierung der Inhalte des Exports bzw. Vereinfachung der Darstellung
Obligatorische Aufgabenpakete konnten am Ende von Sprint 4 nicht abgearbeitet werden	Durchführung eines weiteren Sprints und Verschiebung des Abgabetermins für Phase 2 / Reduzierung der Aufgabenpakete um nicht zwingend notwendige Features
Optionale Aufgabenpakete werden neu als obligatorisch eingestuft nach Phase 2	Durchführung eines 5. Sprints in Phase 3

1.4 Zeitplan und Meilensteine

Der Zeitplan und die Arbeitspakete des Projekts inkl. geschätztem Aufwand werden je nach Phase unterteilt in den folgenden Tabellen dargestellt.

Tabelle 2: Phase 1 „Konzeption"

Datum von	Datum bis	Beschreibung	Aufwand
02.07.2024	18.07.2024	Ausarbeitung Projektidee	4 h
		Sammlung von Anforderungen Erstellung Use-Case-Diagramm	6 h
		Aufstellung Projektplan	4 h
		Ausarbeitung Anwendungsstruktur Erstellung UML-Diagramme	6 h
18.07.2024		**Abgabe Phase 1**	

Tabelle 3: Phase 2 "Erarbeitung & Reflexion"

Datum von	Datum bis	Beschreibung	Aufwand
29.07.2024	03.08.2024	Einarbeitung des Feedbacks ins Konzept (Projektplan, Anforderungsdokument, Projektdokumentation)	8 h
		Aufsetzen des GitHub-Repository Einrichten der Python-Codebasis	4 h
		Sprint Planning - Vorbereitung des 1. Sprint	2 h
05.08.2024	11.08.2024	**Sprint 1: Datenbankverwaltung** SQLite-Datenbankschemata aufbauen Anwendungslogik der Datenbank implementieren Interfaces & Konnektoren implementieren	20 h
		Sprint Planning – Vorbereitung des 2. Sprints	1 h

12.08.2024	18.08.2024	**Sprint 2: Frontend-Basisentwicklung** Benutzeroberflächen entwerfen & implementieren	20 h
		Sprint Planning – Vorbereitung des 3. Sprints	1 h
19.08.2024	25.08.2024	**Sprint 3: Erweiterte Funktionen Frontend** Konsistente Benutzerführung implementieren Integration Datenbank in Frontend	20 h
		Sprint Planning – Vorbereitung des 4. Sprints	1 h
26.08.2024	01.09.2024	**Sprint 4: Dateiexport und optionale Features** Export der Trainingspläne implementieren Aufräumen der bisherigen Codebasis Optionale Features implementieren	20 h
02.09.2024	08.09.2024	Überarbeitung Projektplan, Anforderungsdokument & Projektdokumentation nach der Implementierung	12 h
		Erstellung Testprotokoll	4 h
		Verteilung der Anwendung implementieren	8 h
		Anwendung anhand Testprotokoll testen	2 h
09.09.2024		**Abgabe Phase 2**	

Tabelle 4: Phase 3 "Finalisierung"

Datum von	Datum bis	Beschreibung	Aufwand
30.09.2024	06.10.2024	Einarbeitung des Feedbacks in Konzept & Implementierung	12 h
07.10.2024	13.10.2024	Finalisierung der Anwendung	20 h
14.10.2024	20.10.2024	Erstellung Benutzeranleitung	12 h
		Erstellung Abstract	8 h
21.10.2024	27.10.2024	Finalisierung der Dokumente	20 h
28.10.2024		**Abgabe Phase 3**	

Die Meilensteine des Projekts werden unter folgenden Voraussetzungen erreicht:

Tabelle 5: Meilensteine des Projekts

Meilenstein	Voraussetzungen	Erwartetes Datum
Konzept abgeschlossen	1. Konzept erarbeitet 2. Feedback für Phase 1 eingeholt 3. Feedback in Konzept eingearbeitet	03.08.2024
Datenbankanbindung	Sprint 1 durchgeführt	11.08.2024
Frontendimplementierung	Sprint 2 durchgeführt	18.08.2024
Minimum Viable Product	Sprints 1-4 durchgeführt	01.09.2024
Erster funktionierender Build	Anwendungsverteilung implementiert, Tests durchgeführt	08.09.2024
Feature-Complete	1. Alle Sprints durchgeführt 2. Feedback von Phase 2 in Anwendung eingearbeitet 3. Anwendung finalisiert	13.10.2024

2. Anforderungsdokument

2.1 Management Summary

Das Ziel des Projekts ist es, eine Verwaltungssoftware für Sporttrainer zu erstellen, mit welcher Übungen, Einheiten und ganze Pläne verwaltet und exportiert werden können. Die Software wird in Python programmiert mit einer SQLite Datenbank im Hintergrund. Der User soll sich intuitiv im Programm zurechtfinden und mit möglichst wenig zusätzlicher Dokumentation alle Funktionen ausführen können. Einen Nutzen wird es für den User vor allem dann geben, wenn die Software regelmäßig genutzt wird und die Datenbank somit viele verschiedene Übungen bzw. Einheiten enthält, die dann wiederverwendet werden können. Hierbei hilft eine Suchfunktion zum Wiederfinden erstellter Übungen bzw. Einheiten. Bei der Bedienung sollen Datenbankänderungen erst nach expliziter Bestätigung durchgeführt werden. Außerdem sollen dem User auftretende Fehler direkt gemeldet werden, ohne die Software abstürzen zu lassen oder unbenutzbar zu machen. Eine Userverwaltung ist nicht geplant, da keine sensiblen Daten gespeichert werden. Ein erstellter Plan soll als PDF-Datei exportiert werden können mit allen darin enthaltenen Einheiten und Übungen.

2.2 Systemumfang und Kontext

Das System besteht aus einer Frontend-Anwendung auf Basis von Python mit einer dahinterliegenden SQLite-Datenbank.

Das Frontend soll optisch ansprechend und übersichtlich sein. Alle Funktionen sind leicht zu erreichen und die Navigation ist selbsterklärend und intuitiv. Änderungen an Feldern im Frontend sollen nicht direkt auf der Datenbank gespeichert werden, sondern erst nach dem Betätigen eines „Speichern“-Buttons. Bei der Erstellung einer Trainingseinheit sollen Übungen anhand von Schlüsselwörtern oder bestimmten Filtern gesucht und ausgewählt werden können. Gleiches gilt für die Suche nach Einheiten bei Erstellung eines Trainingsplans. Das Frontend soll eine Kalenderfunktion bieten, die eine Übersicht über die terminierten Trainingspläne bietet. Zusätzlich sollen die angelegten Daten in einer übersichtlichen Baumstruktur direkt auf dem Startbildschirm dargestellt werden, um dem User die Suche nach relevanten Daten zu vereinfachen.

Ein Trainingsplan kann mehrere Trainingseinheiten enthalten, welche wiederum mehrere Trainingsübungen enthalten können. Übungen und Einheiten können jeweils einer eindeutigen Kategorie zugeordnet werden. Zu Übungen können Ressourcen angegeben werden, damit diese auf Einheits- bzw. Plan-Ebene aggregiert dargestellt werden können. Ressourcen sind die in einer Übung verwendeten Materialien oder Geräte und stellen ein eigenes Datenobjekt dar. Jeder Plan kann mehreren Kalendertagen zugeordnet werden.

Es gibt keine Userverwaltung, da keine sensiblen Daten in der Datenbank gespeichert werden.

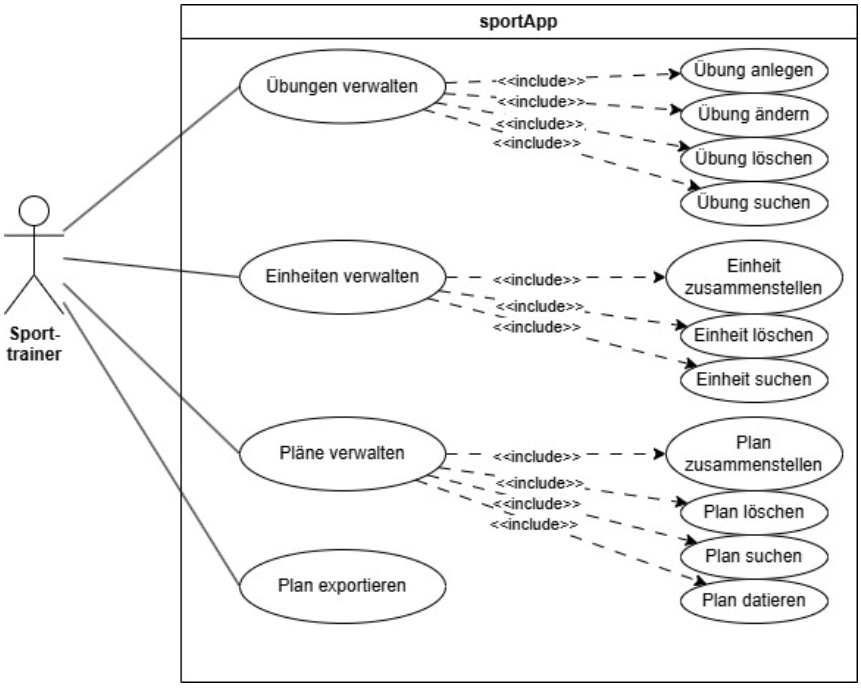
2.3 Funktionale Anforderungen

Folgende User Stories werden für die Anwendung als funktionale Anforderungen definiert:
(obligatorische Anforderungen sind mit [+] gekennzeichnet, optionale mit [-])

1. Als Trainer möchte ich einen Trainingsplan für einen Trainingstag erstellen, damit ich organisiert die Spieler trainieren kann. [+]
2. Als Trainer möchte ich einen erstellten Trainingsplan exportieren, um ihn mit anderen Trainern oder den Spielern teilen zu können. [+]
3. Als Trainer möchte ich die Übungen und Einheiten mit detaillierten Beschreibungen versehen, damit die Spieler auf dieser Basis die Übungen und Einheiten auch selbstständig ausführen können. [+]
4. Als Trainer möchte ich die erstellten Übungen und Einheiten bzw. Teile eines Trainingsplans für die Erstellung eines neuen Trainingsplans wiederverwenden. [+]
5. Als Trainer möchte ich im Trainingsplan auf einen Blick sehen können, welche Ressourcen für den Trainingstag insgesamt benötigt werden. [-]
6. Als Trainer möchte ich Pläne, Einheiten und Übungen mit Kategorien versehen können, um darüber später verwandte Übungen bzw. Übungen zu ähnlichen Themenbereichen finden zu können. [-]
7. Als Trainer möchte ich bereits angelegte Übungen, Einheiten und Pläne im Nachhinein verändern können, um die Objekte an neue Erkenntnisse anzupassen und die Qualität der Datenbasis sicherzustellen. [+]
8. Als Trainer möchte ich bereits angelegte Übungen, Einheiten und Pläne löschen können, um die Qualität der Datenbasis sicherzustellen. [-]
9. Als Trainer möchte ich einen erstellten Trainingsplan auf ein Datum terminieren können, um auch später noch zu wissen, wann bereits welcher Trainingsplan ausgeführt wurde oder wann die Ausführung eines Trainingsplans vorgesehen ist. [-]

Auf der Grundlage dieser User Stories können die in Abbildung 1 gezeigten Anwendungsfälle für den Akteur „Sporttrainer“ abgeleitet werden.

Abbildung 1: Use-Case Diagramm "Sporttrainer"



2.4 Nicht-funktionale Anforderungen

Folgende nicht-funktionale Anforderungen werden an die Software gestellt:
(obligatorische Anforderungen sind mit [+] gekennzeichnet, optionale mit [-])

- Datenbankveränderungen erst nach expliziter Bestätigung eines Popups oder durch das Drücken eines „Speichern“-Buttons [+]
- Prozesse mit hoher Laufzeit sollen abgebrochen werden können [-]
- Auftretende Fehler sollen die Anwendung nicht beenden bzw. abstürzen lassen, sondern eine Fehlermeldung hervorrufen [+]
- Selbsterklärung der Bedienelemente durch sprechende Beschreibungen, Tooltips oder Quick-Infos [+]
- Verhinderung von SQL Injection Angriffen durch Eingabevalidierung aller Felder [+]

2.5 Glossar

Tabelle 6: Glossar

Begriff	**Erläuterung**
Ressource	Material bzw. Gerät, das für eine Übung benötigt wird
Trainingsübung	Kurze, in sich geschlossene und zeitlich begrenzte Anweisung zur Ausführung von Tätigkeiten
Trainingseinheit	Zusammenschluss mehrerer thematisch zusammenhängender Trainingsübungen
Trainingsplan	Zusammenschluss mehrerer Trainingseinheiten zu einem Plan, der einen kompletten Trainingstag umfasst

3. Projektdokumentation

Das GitHub-Repository der Anwendung kann unter folgendem Link gefunden werden:
https://github.com/UltimateCuB3R/DLMCSPSE01_D

Die finale Anwendung wird als ZIP-Datei ausgeliefert, die im GitHub-Repository als „sportApp_v1.2.zip“ gefunden werden kann. In der ZIP-Datei liegen eine ausführbare Datei „sportApp.exe“ und ein Ordner „_internal“. Die Anwendung ist für die Ausführung unter Windows 10/11 gedacht und benötigt keine vorherige Installation einer Python-Laufzeitumgebung. Der Ordner „_internal“ muss zwingend auf der gleichen Ebene wie die ausführbare Anwendung liegen. Die Datenbankdatei liegt im Ordner „_internal\data\“, im Auslieferungszustand ist die Datei „main.db“ nicht vorhanden, da sie beim ersten Anwendungsstart automatisch erzeugt wird. Stattdessen wird eine Datei „main_prefilled.db“ mitgeliefert, die bei Bedarf als „main.db“ kopiert und genutzt werden kann. Hierin sind Beispieldaten enthalten, die zum initialen Verständnis der Anwendung beitragen können.

3.1 Softwareprozess

Die Erarbeitungsphase soll nach dem agilen Projektmodell Scrum durchgeführt werden. Es werden 4 Sprints angesetzt, die jeweils eine Woche lang dauern und sich sukzessive auf einen bestimmten Bereich der Implementierung fokussieren. Am Ende eines jeden Sprints soll eine funktionierende Anwendung stehen, die mindestens die als obligatorisch definierten Anforderungen des Sprints erfüllt. Während einem Sprint ist die Behebung von Bugs der Implementierung von neuen Features vorzuziehen.

Da keine weiteren Personen bei diesem Projekt involviert sind, werden die drei Rollen aus Scrum (Product Owner, Scrum Master & Entwickler) von derselben Person ausgeführt. Zu Beginn jedes Sprints werden die Anforderungen bzw. Aufgabenpakete in einem Sprint Backlog festgehalten und sowohl als obligatorisch bzw. optional markiert, als auch mit einer Priorität versehen. Bei Abarbeitung eines Aufgabenpaketes wird dies im Sprint Backlog vermerkt, damit mit das nächste Aufgabenpaket bearbeitet werden kann. Sofern ein Bug auftritt, der nicht am gleichen Tag noch behoben werden kann, muss dieser im Sprint Backlog vermerkt werden. An jedem Tag eines Sprints wird zu Beginn ein kurzes maximal 10-minütiges Daily Scrum Meeting veranstaltet, in dem die schon bearbeiteten Aufgabenpakete und die daraus eventuell resultierten Bugs angeschaut werden, um die zu bearbeitenden Aufgaben für den Tag zu definieren. Am Ende eines Sprints wird ein Sprint Review Meeting abgehalten, in dem die erledigten und offenen Aufgabenpakete gesichtet werden, um diese ggf. im nächsten Sprint abzuarbeiten oder auf einen späteren Sprint bzw. die Finalisierungsphase zu verschieben.

Am Ende jedes Tages sollen die gemachten Änderungen in das Repository übertragen werden, wobei auch alle automatisierten Tests, also Unit- und ggf. Integrationstests, ausgeführt werden sollen. Pro Sprint wird ein neuer Branch in GitHub angelegt, mit dem Ziel, diesen Branch am Ende eines Sprints erfolgreich in den main-Branch mergen zu können. Das erfolgreiche Mergen in den

main-Branch soll am Ende jedes Sprints gegenüber der Implementierung optionaler Features priorisiert werden.
Die in Scrum übliche „Definition of Done“ wird innerhalb dieses Projekts im Allgemeinen wie folgt definiert: „Erledigt ist, was funktioniert und aus sauberem Code besteht.“

Für die Projektumsetzung wird Scrum verwendet, um flexibel während eines Sprints auf auftretende Herausforderungen eingehen zu können, ohne Termine verschieben oder obligatorische Anforderungen fallen lassen zu müssen. Durch die Fokussierung auf funktionierenden Code wird außerdem Aufwand zur Fehlerbehebung in späteren Phasen vermieden, wie es bei anderen Projektmodellen, wie z.B. dem sequenziellen Wasserfallmodell, üblich ist. Bei Scrum stehen die Kundenanforderungen, also im Falle dieses Projekts die definierten Use Cases und User Stories, im Vordergrund und definieren, was die fertige Anwendung bieten soll. Ein weiterer Vorteil der Nutzung von Scrum ist die Nutzung des Product Backlogs, in dem alle Anforderungen enthalten sind, die für das finale Produkt erfüllt sein sollen. Die Items aus dem Product Backlog werden dann in das jeweilige Sprint Backlog übernommen, wodurch stets eine Übersicht darüber besteht, welche Themen noch offen sind und was im Projekt schon erreicht wurde.

3.2 Genutzte Technologien und Tools

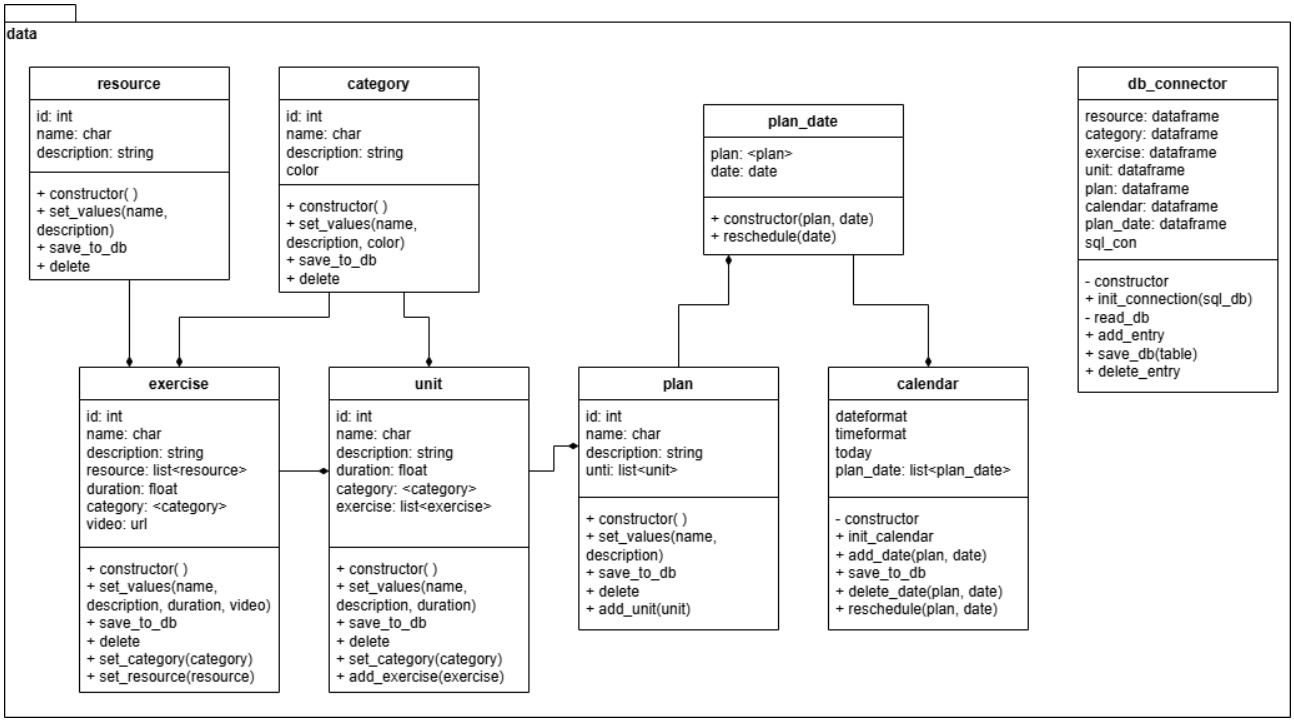
Die Anwendung soll in der Programmiersprache Python geschrieben werden und als Datenbank eine SQLite-DB nutzen. Die ursprünglich geplante Verwendung von MySQL als Datenbanksystem wurde verworfen, da diese einen großen Aufwand in der Verwaltung bedeutet hätte ohne entsprechende Vorteile zu bieten. So würden zum einen die Stärken von MySQL, Authentifizierung und Multi-User, in der geplanten Anwendung nicht zum Tragen kommen, da sie nur von einem User ohne Authentifizierung genutzt werden soll. Stattdessen wird nun die Datenbankanbindung mittels SQLite realisiert, einer leichtgewichtigen Datenbank-Engine, die direkt in der Standardauslieferung von Python verfügbar ist und zudem auch in viele andere Bibliotheken integriert ist. In Python werden die Daten größtenteils mittels der Bibliothek pandas organisiert. Für die Entwicklung der Benutzeroberfläche kommt die Bibliothek PyQt zum Einsatz, da diese einen einfachen WYSIWYG-Editor für den Entwurf der Oberflächen bietet. Für die Umsetzung der HTML-Ansicht eines Trainingsplans kommt zum einen die WebEngine von PyQt zum Einsatz, zum anderen die Bibliothek Jinja2. Dies ermöglicht, einfache HTML-Templates zu erstellen, die über Jinja2 mit den dynamischen Inhalten gefüllt werden und dann über die WebEngine im Frontend dargestellt werden können. Die WebEngine bietet zusätzlich noch den Vorteil, dass direkt aus der dargestellten HTML-Seite heraus eine PDF-Datei erstellt werden kann.
Python wird genutzt, da dies eine benutzerfreundlichere Entwicklungsoberfläche bietet als z.B. Java. Außerdem bietet Python eine modernere Art der Programmierung und ein besseres Handling von Abhängigkeiten z.B. über Conda, womit auch bestehende Bibliotheken einfacher eingebunden werden können.

Zur Realisierung der Datenbankverbindung in Python wurde das Singleton-Entwurfsmuster in der Klasse DatabaseConnector des Moduls app\data.py eingesetzt, damit zu jedem Zeitpunkt sichergestellt ist, dass nur eine Datenbank existiert und angesprochen wird.
Zudem ist der komplette Code nach dem MVC-Pattern aufgeteilt in Datenlogik (data.py), Businesslogik (control.py) und Visualisierung (view.py). Dies bietet den Vorteil, dass die Verantwortlichkeiten klar abgegrenzt sind und im Python-Code die projektspezifischen Bibliotheken nur in den Modulen importiert werden müssen, in denen es notwendig ist.
Als Entwicklungsoberfläche wird die PyCharm Community Edition genutzt, worüber direkt ein GitHub-Repository angesprochen und verwaltet werden kann.
Bei der Programmierung wurde darauf geachtet, so wenig wie möglich hart zu codieren und stattdessen so viel wie möglich allgemein zu halten und zu dynamisieren. Aus diesem Grund wurden zwei XML-Dateien app\data\db_def.xml und app\view\gui_def.xml erstellt, die durch einen XML-Parser eingelesen und weiterverarbeitet werden. Die db_def.xml beinhaltet hierbei die Definition der Datenbanktabellen und ihren Abhängigkeiten und die gui_defxml die Definition der Felder der GUI-Widgets und welchen Typ diese haben. Außerdem wurde eine Übersetzung der technischen Namen für Tabellen und Felder über die Datei app\view\dictionary_de.txt realisiert, sodass die App später ggf. auch mehrsprachig gestaltet werden kann. Damit konnte einiges an Aufwand gespart werden und der Code im Allgemeinen wartungsfreundlicher gestaltet werden.

3.3 Anwendungsstruktur

Ursprünglich war im Konzept geplant, die einzelnen Objekttypen im data-Modul mit separaten Klassen zu programmieren. Dies hätte bedeutet, dass jedes Datenobjekt (z.B. eine Übung „exercise“) eine eigene Instanz der Klasse gewesen wäre. Mehrere Übungen wären in einem Dictionary gespeichert worden. Aus diesem Dictionary heraus würden die Inhalte in einen Dataframe geladen werden, welcher dann schlussendlich auf die SQL-Datenbank gespeichert worden wäre.
Ein entsprechendes UML-Diagramm dieses Konzepts ist in Abbildung 2 zu sehen.

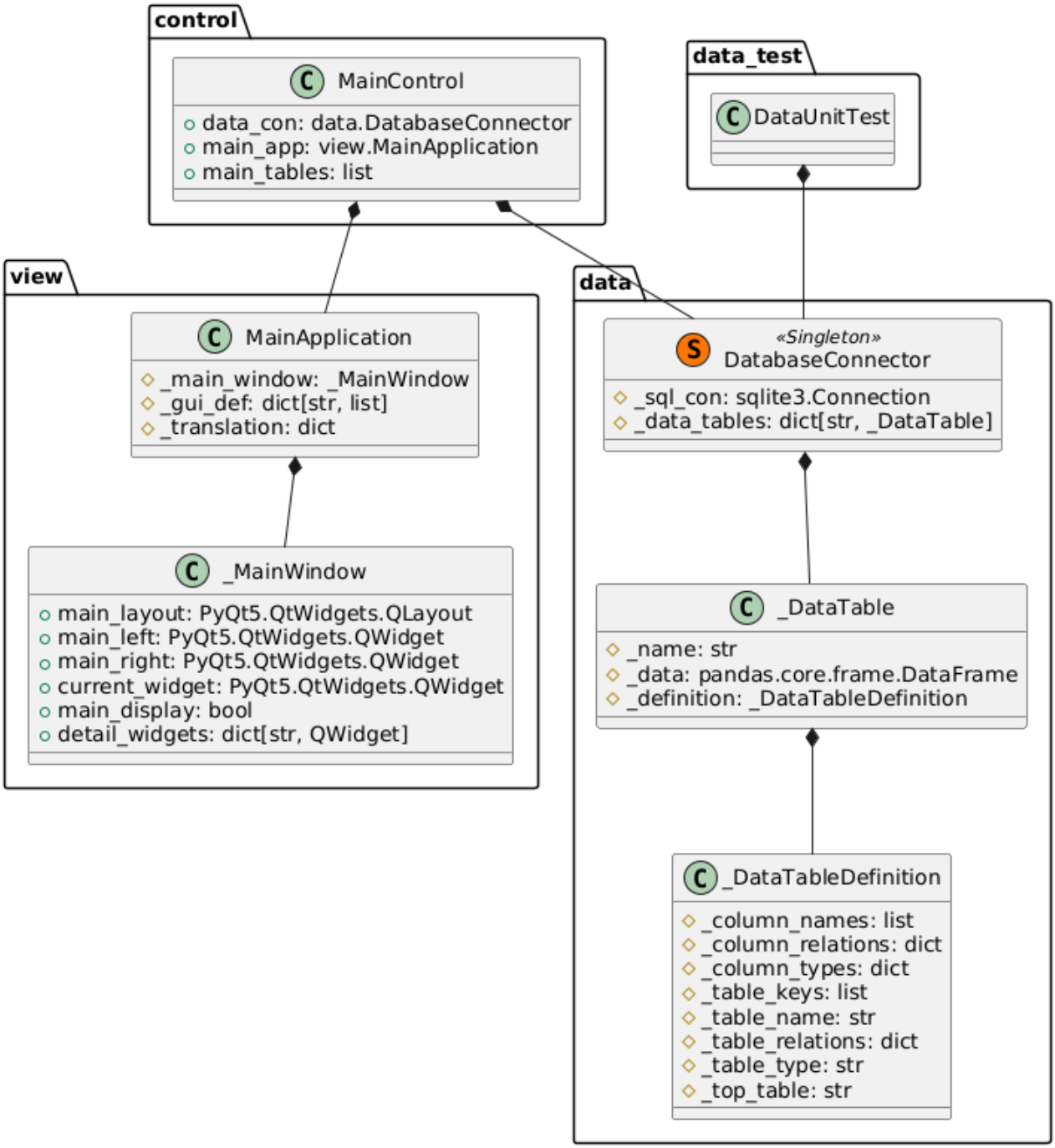
Abbildung 2: UML-Diagramm des ursprünglich geplanten data-Moduls



Dieses Vorhaben wurde im Implementierungsprozess jedoch recht schnell in Sprint 1 verworfen, da hiermit viele Dinge hätten mehrfach programmiert oder sogar hart codiert werden müssen. Um den Code möglichst dynamisch und schlank zu gestalten, wurde im data-Modul darauf verzichtet, jede einzelne Tabelle auszuprogrammieren, stattdessen wurde mithilfe der bereits beschriebenen db_def.xml ein Weg gefunden, die Tabellen dynamisch zur Laufzeit zu erstellen und zu definieren. Hierin werden auch Beziehungen und sogar Hierarchien zwischen den unterschiedlichen Tabellen definiert, auf die unter anderem über die Business-Logik des control-Moduls zurückgegriffen wird. Damit muss bei Änderungen nicht an vielen unterschiedlichen Stellen die Programmierung geändert werden, sondern alle Tabellen sind Instanzen der gleichen Klasse und können somit auf die gleiche Weise verarbeitet werden. Dies ermöglicht zudem, dass die Export-Funktion nicht nur wie ursprünglich angefordert dem Plan-Objekt zur Verfügung steht, sondern auch alle anderen Objekte bei Bedarf exportiert werden können. Darüber hinaus kann auf diese Weise die starke pandas-Bibliothek voll genutzt werden, um die Daten in einem DataFrame zu verwalten, der direkt in eine SQL-Tabelle geschrieben werden kann. Somit entfällt die Implementierung einiger Logiken, die das ursprüngliche Konzept mit sich gebracht hätte.

In der folgenden Abbildung 3 ist ein UML-Diagramm der umgesetzten Klassenstruktur zu sehen, aus Platzgründen jedoch ohne Methoden. Man kann sehen, dass mit der neuen Struktur viel weniger Klassen notwendig sind und mehr Funktionen unter zentralen Klassen zusammengefasst werden konnten.

Abbildung 3: UML-Diagramm der umgesetzten Klassenstruktur ohne Methoden

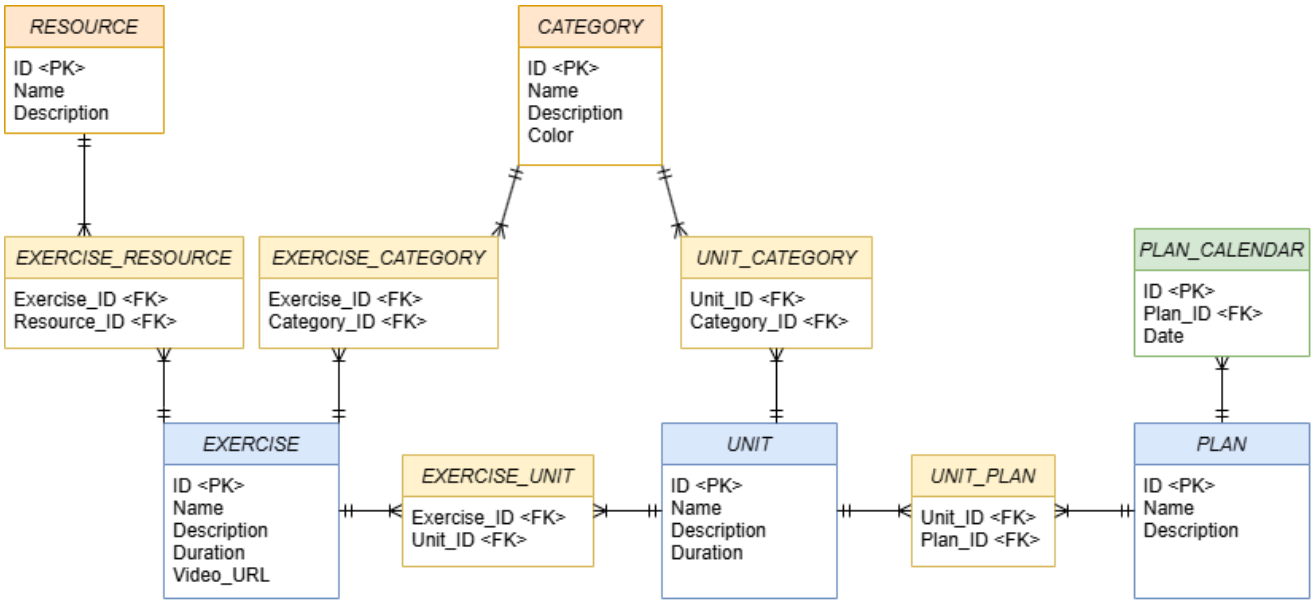


Die Klassen _DataTable und _DataTableDefinition wurden als protected markiert, da es nicht möglich sein soll, diese von außerhalb des data-Moduls zu instanziieren. Gleiches gilt für die Klasse _MainWindow des view-Moduls.

Ein UML-Klassendiagramm des data-Moduls mitsamt den implementierten Methoden ist aus Platzgründen in Anhang 1 zu finden.

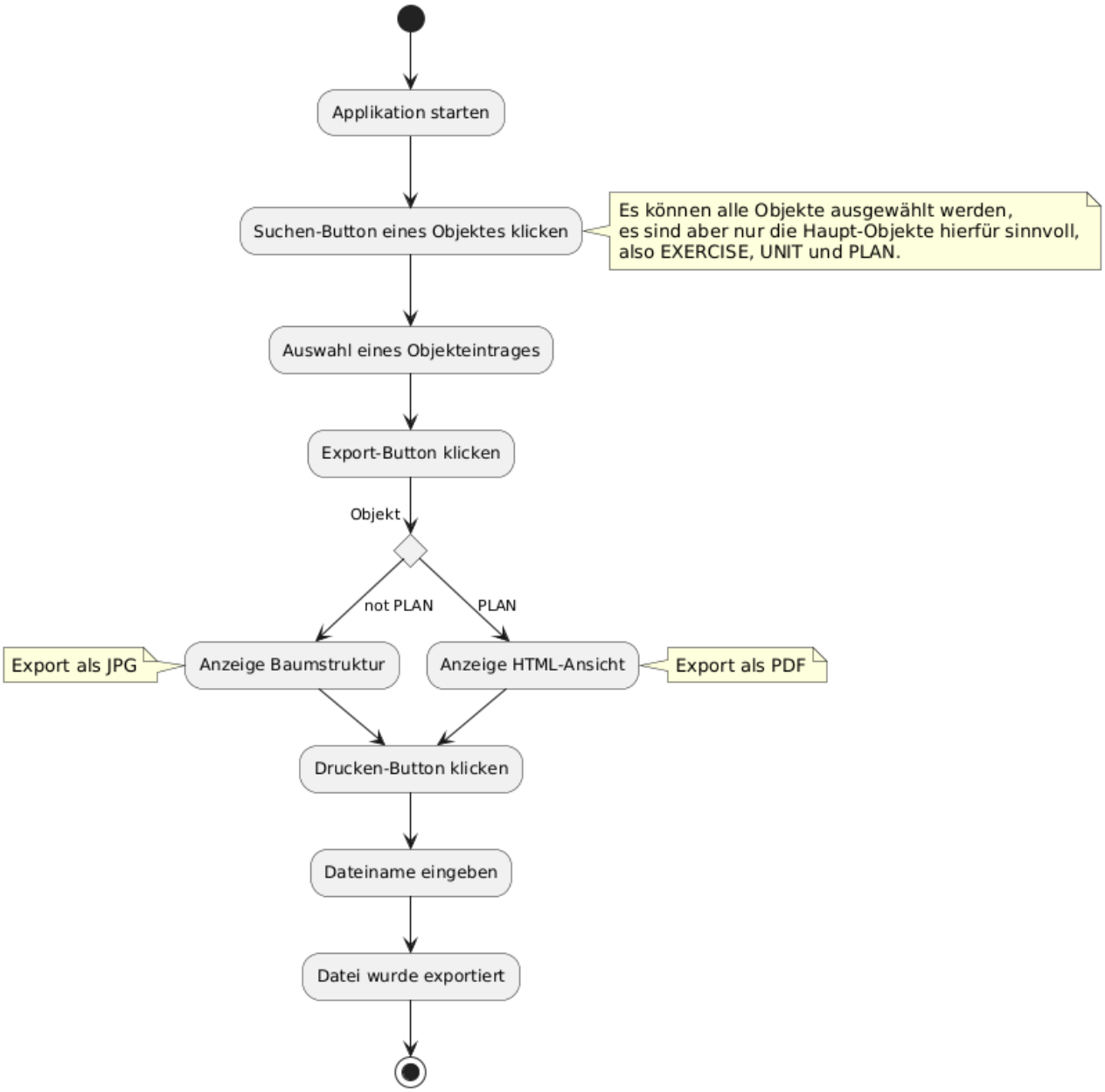
Für die Datenbankstruktur wurde ein ER-Diagramm erstellt, um die Zusammenhänge zu visualisieren. Dieses ist in der folgenden Abbildung 4 zu sehen und zeigt die unterschiedlichen Tabellen. Die Haupttabellen sind blau, die Beziehungstabellen gelb und die untergeordneten Tabellen orange markiert. Zusätzlich gibt es noch eine grün markierte Tabelle, die eben sowohl eine untergeordnete Tabelle als auch eine Beziehungstabelle ist. Alle Felder mit „ID“ im Namen sind von Typ „INTEGER“, die sonstigen Felder sind von Typ „TEXT“.

Abbildung 4: ER-Diagramm der Datenbankstruktur



Zur Veranschaulichung der implementierten Prozesse ist in Abbildung 5 ein Aktivitätsdiagramm des Prozesses „Export eines Objekts“ zu finden.

Abbildung 5: UML-Aktivitätsdiagramm "Export eines Objekts"



Der Export eines Objektes war ursprünglich als PDF geplant, damit die Datei einfach geteilt und gelesen werden kann und zudem Links direkt geöffnet werden können. Aufgrund technischer Schwierigkeiten mit PyQt5 und dem Drucken von Widgets gegen Ende des Sprint 4 wurde jedoch auf die Erstellung einer Bilddatei ausgewichen, die einfach das aktuell angezeigte Widget als Screenshot speichert. Diese technische Hürde konnte in der Finalisierungsphase jedoch über eine WebEngine in PyQt und die Bibliothek Jinja2, welche HTML-Templates ermöglicht, noch überwunden werden. Ein Plan wird nun immer als PDF exportiert, alle anderen Objekte werden als JPG exportiert.

3.4 Technische Schuld

Im aktuellen Stand der Implementierung wurden folgende Anforderungen noch nicht umgesetzt:

- Terminierung eines Trainingsplans auf ein bestimmtes Datum
    - Grund: Zeitmangel
- Anzeige der Terminierungen von Trainingsplänen in einem Kalender-Widget auf dem Startbildschirm
    - Grund: Zeitmangel
- Konsolidierte/Aggregierte Darstellung der Ressourcen unterhalb eines Trainingsplans
    - Grund: technische Schwierigkeiten

Diese als optional definierten Anforderungen konnten aus den genannten Gründen im Projektverlauf nicht umgesetzt werden.

3.5 Benutzeranleitung

3.5.1 Objekte in der Anwendung

In der Anwendung gibt es fünf verschiedene Objekte: Plan, Einheit, Übung, Kategorie und Ressource.

1. Eine Ressource ist ein Material bzw. Gerät, das für eine Übung benötigt wird und kann ausschließlich mit Übungen verknüpft werden.
2. Eine Kategorie ist eine Möglichkeit, ähnliche Übungen und Einheiten zu gruppieren, um später einfacher danach suchen zu können.
3. Eine Übung ist eine kurze, in sich geschlossene und zeitlich begrenzte Anweisung zur Ausführung von Tätigkeiten und kann mit Einheiten, Ressourcen und Kategorien verknüpft werden.
4. Eine Einheit ist ein Zusammenschluss mehrerer thematisch zusammenhängender Übungen und kann mit Plänen, Übungen und Kategorien verknüpft werden.
5. Ein Plan ist ein Zusammenschluss mehrerer Einheiten, um einen kompletten Trainingstag zu umfassen. Ein Plan kann nur mit Übungen verknüpft werden.

3.5.2 Startbildschirm

Der Startbildschirm ist der Bildschirm, der sich direkt nach dem Start der Anwendung öffnet. Von hier aus kann man fast alle Funktionen der Anwendung ausführen. Der Startbildschirm ist aufgeteilt in einen oberen, einen linken und einen rechten Bereich. Siehe hierzu Abbildung 6.

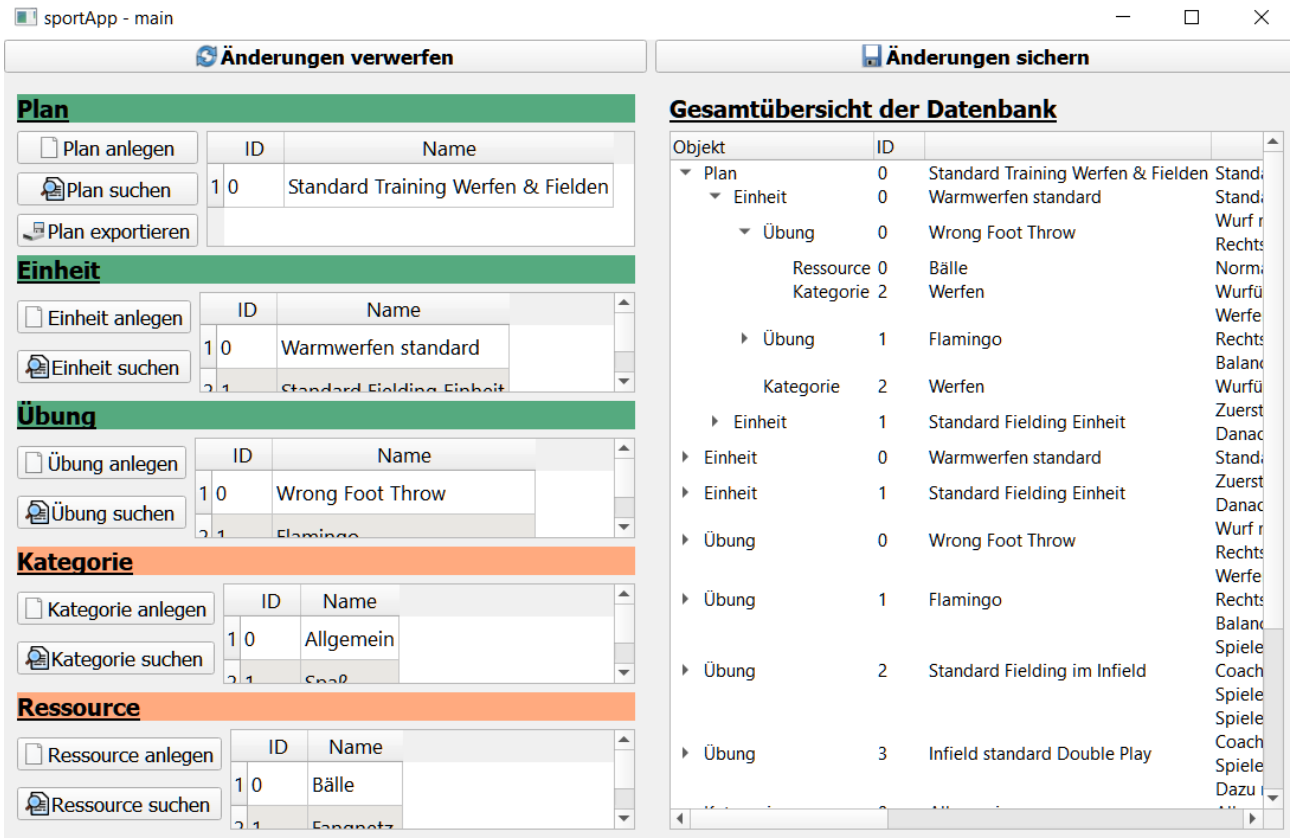
Im oberen Bereich befinden sich die Buttons, um Änderungen an den Daten zu verwerfen, also auf den zuletzt in der Datenbank gesicherten Stand zurückzusetzen, oder zu sichern, also auf die Datenbank zu speichern.

Im linken Bereich befinden sich für jedes der fünf pflegbaren Objekte (Plan, Einheit, Übung, Kategorie, Ressource) ein eigener Bereich mit Buttons und einer Tabelle der existierenden Einträge. Dabei hat jedes Objekt einen Button für die Anlage und einen weiteren für die Suche. Beim Plan ist

zusätzlich noch ein Button für den Export verfügbar - um diese Funktion auszuführen, muss in der Tabelle rechts daneben ein Eintrag ausgewählt worden sein. In jeder Tabelle des Startbildschirms ist es möglich, einen Eintrag per Doppelklick direkt im Anzeigen-Modus aufzurufen.

Im rechten Bereich befindet sich eine Gesamtübersicht der Datenbank als Baumstruktur. Hier kann zu jedem in der Anwendung gespeicherten Objekt direkt auf einen Blick nachvollzogen werden, zu welchen anderen Objekten dieses verknüpft ist und mit welchen Werten das Objekt gepflegt ist. Jede Ebene kann aufgeklappt werden, um die verknüpften Objekteinträge anzuzeigen, wie beispielhaft in Abbildung 6 zu sehen. Diese Baumstruktur aktualisiert sich nach jeder Änderung an den Daten der Anwendung.

Abbildung 6: Startbildschirm der Anwendung



3.5.3 Suchbildschirm

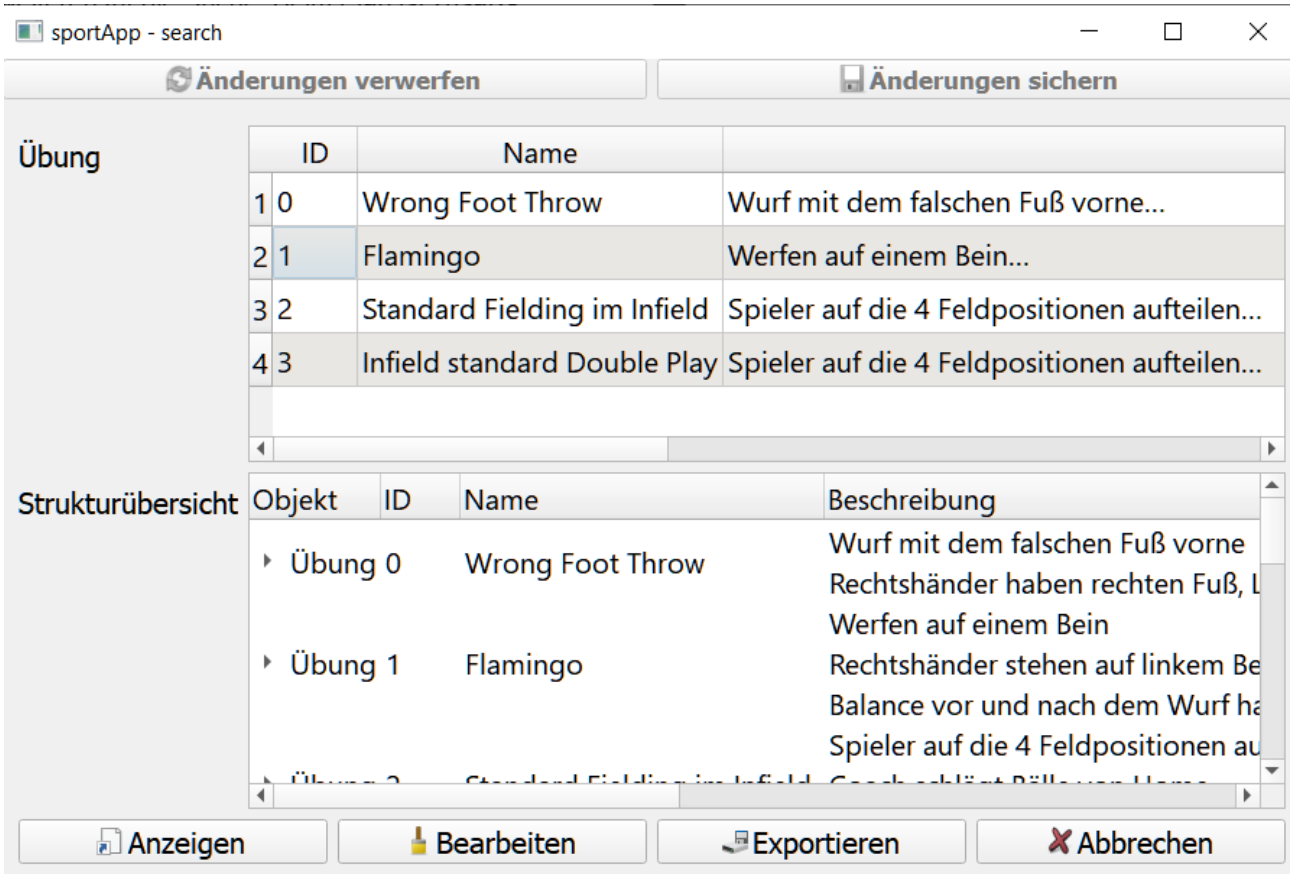
Bei einem Klick auf einen der „Suchen“-Buttons auf dem Startbildschirm öffnet sich der Suchbildschirm für das jeweilige Objekt. Siehe hierzu Abbildung 7.

Auf dem Suchbildschirm wird oben links der Name des ausgewählten Objekts und rechts daneben werden alle Einträge dieses Objektes in Tabellenform angezeigt. In dieser Tabelle sind alle gepflegten Daten zum Objekt ersichtlich. Zusätzlich können unterhalb der Tabelle in der Strukturübersicht die existierenden Verknüpfungen der Einträge in einer Baumstruktur nachvollzogen werden.

Vom Suchbildschirm aus kann man einen Eintrag der Tabelle selektieren und danach einen der Buttons „Anzeigen“, „Bearbeiten“ oder „Exportieren“ klicken, um die entsprechende Funktion auszuführen. Alternativ kann auch per Doppelklick auf einen Tabelleneintrag direkt in die Anzeige des

entsprechenden Eintrags gesprungen werden. Über den Button „Abbrechen“ wird wieder zurück auf den Startbildschirm gesprungen.

Abbildung 7: Suchbildschirm Übung



3.5.4 Detailbildschirm eines Objekteintrags

Nach Ausführung einer der Funktionen „Anlegen“, „Anzeigen“ oder „Bearbeiten“, bzw. Ausführung eines Doppelklicks auf einen Tabelleneintrag im Start- oder Suchbildschirm, wird auf den Detailbildschirm des ausgewählten Objekteintrags gesprungen. Der Detailbildschirm ist in Abbildung 8 zu sehen.

Auf dem Detailbildschirm sind alle Daten des ausgewählten Eintrags zu sehen, sofern ein existierender Eintrag ausgewählt wurde. Im Beispiel der Übung sind die Felder „Name“, „Beschreibung“, „Dauer“ und „Video URL“ zu sehen, zusätzlich können noch Kategorien und Ressourcen verknüpft werden. Vom Detailbildschirm aus sind die Funktionen „Speichern“ und „Löschen“ verfügbar je nachdem, ob der Bildschirm im Anlage-, Anzeige- oder Bearbeitungsmodus geöffnet wurde. In jedem Modus kann über den Button „Abbrechen“ wieder zurück auf den Startbildschirm gesprungen werden.

Im Anlagemodus sind alle Datenfelder leer, außerdem ist nur die Funktion „Speichern“ verfügbar.

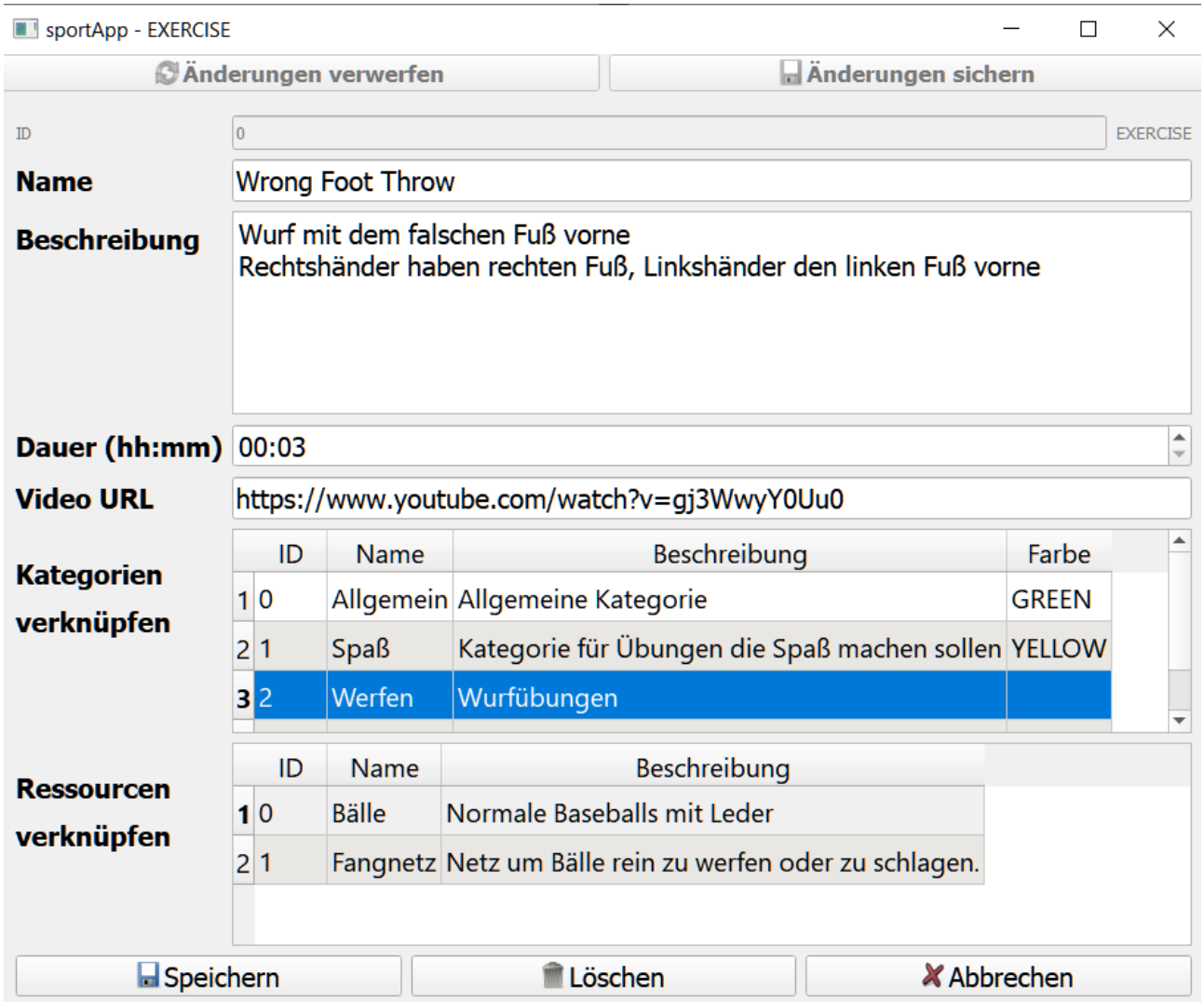
Im Anzeigemodus sind alle Datenfelder mit den entsprechenden Daten gefüllt, aber nicht bearbeitbar. Die existierenden Verknüpfungen werden in den Tabellen angezeigt und können nicht verändert werden. Es ist nur die Funktion „Löschen“ verfügbar.

Im Bearbeitungsmodus sind alle Datenfelder mit den entsprechenden Daten gefüllt und bearbeitbar. Auch die existierenden Verknüpfungen können bearbeitet werden. Es sind beide Funktionen „Speichern“ und „Löschen“ verfügbar.

Die Verknüpfung von anderen Objekteinträgen geschieht per einfachem Klick auf die entsprechende Zeile der Tabelle, sowohl zum Aktivieren der Verknüpfung als auch zum Deaktivieren. Alle beim Speichern markierten Einträge werden entsprechend gespeichert und alle nicht markierten Einträge werden ggf. als Verknüpfung gelöscht.

Beim Löschen eines Objekteintrags werden der Eintrag selbst sowie alle Verknüpfungen zu anderen Einträgen gelöscht. Die erfolgreiche Löschung wird mit einer Meldung in der Anwendung bestätigt.

Abbildung 8: Detailbildschirm einer Übung im Bearbeitungsmodus



3.5.5 Exportbildschirm eines Objekteintrags

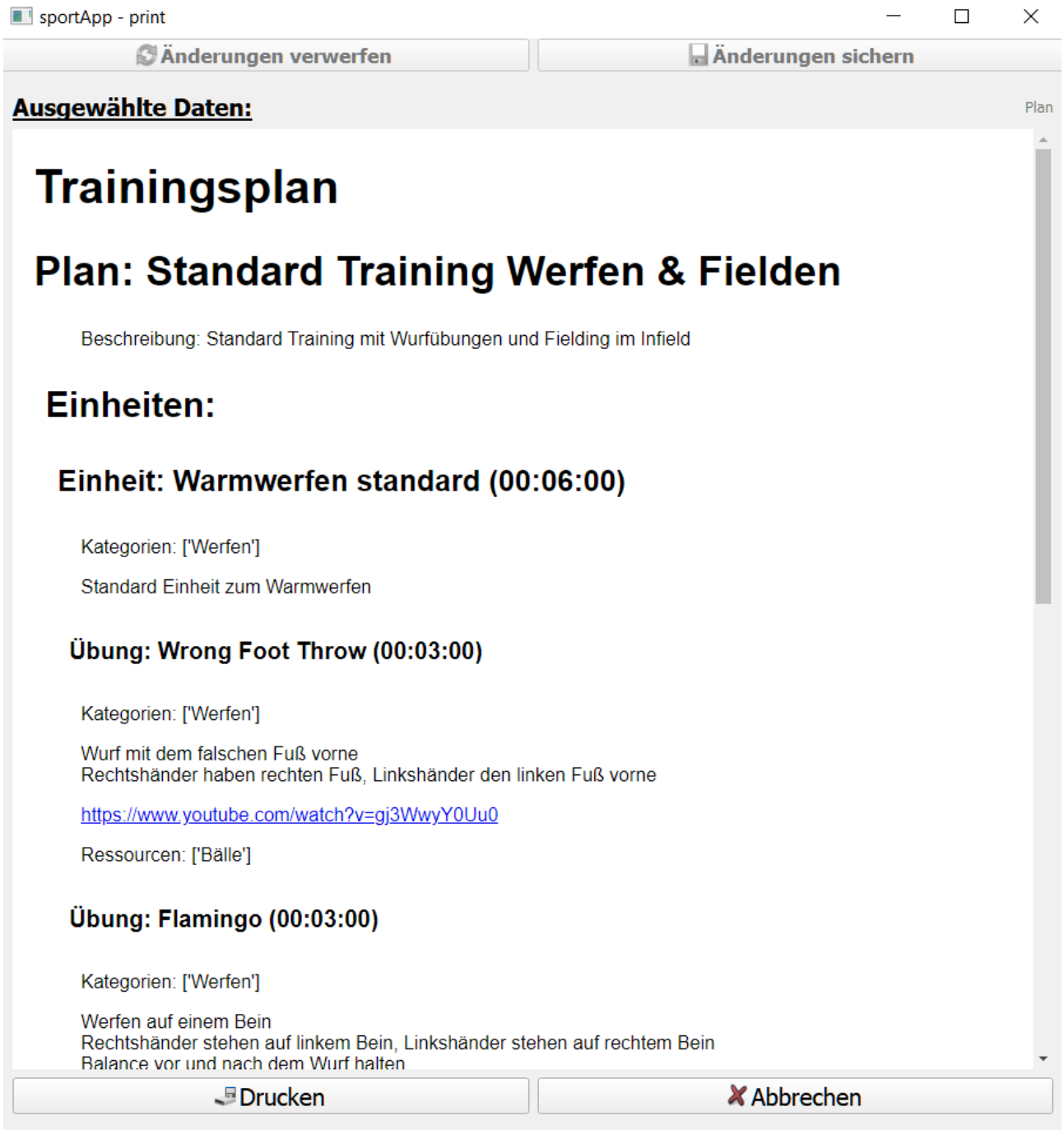
Nach Ausführung der Funktion „Exportieren“, entweder im Startbildschirm für den Plan oder im Suchbildschirm für jedes beliebige Objekt, erscheint der Exportbildschirm des jeweiligen Objekteintrages. Dieser Exportbildschirm ist beispielhaft für einen Plan in Abbildung 9 zu sehen.

Auf dem Exportbildschirm sind im Beispielfall die Daten des Plans und aller verknüpften Objekteinträge als einfache Website zu sehen. Diese Website kann durch Ausführen der „Drucken“-Funktion

als PDF gespeichert werden, wobei in einem Dialog der Dateiname und -speicherort ausgewählt werden kann.

Falls ein anderes Objekt als ein Plan exportiert wird, geschieht dies nicht als PDF, sondern als JPG-Bilddatei, die einen Screenshot der ausgeklappten Baumstruktur des ausgewählten Objekteintrages zeigt.

Abbildung 9: Exportbildschirm eines Plans



4. Testprotokoll

4.1 Testumgebung und Testvorbereitung

Als Ausgang für die meisten der nachfolgenden Tests wird eine frisch ausgelieferte Applikation ohne bestehende Datenbank vorausgesetzt.

Für einige Tests ist es aber sinnvoll, eine vorab befüllte Datenbank zu nutzen, um bei eventuell auftretenden Fehlern in den Testfällen keinen Komplettstillstand der Tests zu riskieren. Hierfür wird eine Datenbank _internal\data\main_prefilled.db mit ausgeliefert, die bei Bedarf als _internal\data\main.db kopiert und verwendet werden kann.

Zur Analyse der Datenbankdatei kann die portable Version des „SQLite Browser“ verwendet werden, die hier heruntergeladen werden kann: https://sqlitebrowser.org/dl/.

4.2 Testfälle

Generell ist es sinnvoll, die nachfolgenden Testfälle sequenziell abzuarbeiten, da diese z.T. inhaltlich aufeinander aufbauen.

4.2.1 Testfall 1: Applikationsstart und Anlage der Datenbank

Vorbedingungen: Die Datei _internal\data\main.db existiert noch nicht.

Ablauf: Die Applikation wird gestartet.

Erwartetes Ergebnis: Datei _internal\data\main.db wird angelegt und enthält alle definierten Datenbanken mit den definierten Feldern. Die Applikation wird gestartet und zeigt den Startbildschirm.

Tatsächliches Ergebnis: OK - Datei main.db wird unter erwartetem Pfad angelegt und enthält die Datenbankdefinitionen für alle definierten Tabellen und deren Beziehungstabellen. Die Anwendung zeigt den Startbildschirm.

4.2.2 Testfall 2: Anlage einer Übung mit Speichern

Vorbedingungen: Die Applikation ist gestartet, der User befindet sich auf dem Startbildschirm.

Ablauf: Im Bereich „Übung“ wird auf den Button „Übung anlegen“ geklickt. In dem daraufhin zu sehenden Bildschirm werden die Daten für die Übung eingegeben. Sind alle Daten eingegeben, wird auf den Button „Speichern“ gedrückt, woraufhin man zum Startbildschirm zurückgelangt.

Erwartetes Ergebnis: Alle Felder sind bearbeitbar mit Ausnahme des Feldes „ID“. Die eingegebenen Daten wurden zwischengespeichert und sind direkt auf dem Startbildschirm ersichtlich, einmal rechts in der Baumstruktur und einmal links im Bereich „Übung“ als Tabelleneintrag. Es erscheint keine Fehlermeldung. Beim erneuten Aufruf des Anlagebildschirms sind die eingegebenen Daten nicht mehr vorhanden.

Tatsächliches Ergebnis: OK - Nach Klick auf „Übung anlegen“ erscheint ein neuer Bildschirm. Alle Felder bis auf „ID“ sind bearbeitbar. Nach dem Klick auf „Speichern“ erscheint wieder der Startbildschirm und die eingegebenen Daten sind rechts in der Baumstruktur und links in der entsprechenden Tabelle ersichtlich. Beim erneuten Klick auf „Übung anlegen“ sind die Felder wieder leer.

4.2.3 Testfall 3: Anlage einer Übung mit Abbrechen

Vorbedingungen: Die Applikation ist gestartet

Ablauf: Im Bereich „Übung“ wird auf den Button „Übung anlegen“ geklickt. In dem daraufhin zu sehenden Bildschirm werden die Daten für die Übung eingegeben. Sind alle Daten eingegeben, wird auf den Button „Abbrechen“ gedrückt, woraufhin man zum Startbildschirm zurückgelangt.

Erwartetes Ergebnis: Die eingegebenen Daten wurden nicht gespeichert und sind nicht in der Baumstruktur rechts bzw. der Tabelle links auf dem Startbildschirm ersichtlich. Es erscheint keine Fehlermeldung. Beim erneuten Aufruf des Anlagebildschirms sind die eingegebenen Daten nicht mehr vorhanden.

Tatsächliches Ergebnis: OK - Nach Klick auf „Übung anlegen“ erscheint ein neuer Bildschirm. Nach Eingabe der Daten wird auf „Abbrechen“ geklickt und es erscheint wieder der Startbildschirm. Die eingegebenen Daten sind nicht in der Baumstruktur oder der Tabelle zu sehen.

4.2.4 Testfall 4: Suchen einer angelegten Übung

Vorbedingungen: Testfall 2 wurde erfolgreich ausgeführt oder main_prefilled.db wurde geladen.

Ablauf: Im Bereich „Übung“ wird auf den Button „Übung suchen“ geklickt. In dem daraufhin zu sehenden Bildschirm werden alle existierenden Übungen angezeigt, die mit einem Klick auf die Zeile markiert werden können.

Erwartetes Ergebnis: Die in Testfall 2 angelegte Übung wird im Suchbildschirm angezeigt und kann ausgewählt werden. Alternativ werden andere angelegte Übungen angezeigt und können ausgewählt werden. Im Suchbildschirm wird außerdem die Baumstruktur des ausgewählten Objekts „Übung“ angezeigt.

Tatsächliches Ergebnis: OK - Nach Klick auf „Übung suchen“ erscheint ein neuer Bildschirm. Die angelegte Übung wird in der Tabelle oben und in der Baumstruktur unten angezeigt und kann in der Tabelle ausgewählt werden.

4.2.5 Testfall 5: Anzeigen einer gesuchten Übung

Vorbedingungen: Testfall 4 wurde erfolgreich ausgeführt und der Suchbildschirm wird noch angezeigt.

Ablauf: Eine Tabelleneintrag wird ausgewählt und durch Drücken des „Anzeigen“ Buttons wird die ausgewählte Übung angezeigt. Nach Überprüfung der Inhalte kann durch Drücken des „Abbrechen“ Buttons wieder auf den Startbildschirm zurückgekehrt werden. Alternativ kann auch per Doppelklick auf einen Tabelleneintrag die entsprechende Übung angezeigt werden.

Erwartetes Ergebnis: Die ausgewählte Übung wird im Anzeigemodus angezeigt, d.h. die angezeigten Felder sind nicht bearbeitbar. Der „Speichern“ Button kann nicht betätigt werden. Der Button „Löschen“ ist verfügbar. Über „Abbrechen“ wird wieder auf den Startbildschirm zurückgekehrt. Die Daten in der Baumstruktur rechts auf dem Startbildschirm haben sich bzgl. der ausgewählten Übung

nicht geändert. Wird vom User keine Zeile ausgewählt, aber trotzdem auf „Anzeigen“ oder „Bearbeiten“ gedrückt, so erscheint eine Fehlermeldung. Falls die Übung bereits zu anderen untergeordneten Objekten verknüpft wurde, werden diese Verknüpfungen in den Tabellen im Bearbeitungsbildschirm als markierte Zeilen angezeigt. Die Zeilenmarkierungen sollen nicht verändert werden können.

Tatsächliches Ergebnis: OK - Aus dem Suchbildschirm heraus wird eine Übung ausgewählt und auf „Anzeigen“ geklickt, daraufhin erscheint ein Bildschirm, in dem die Daten der ausgewählten Übung angezeigt werden. Die Felder inkl. der Tabellen unterhalb sind nicht bearbeitbar und „Speichern“ ist nicht möglich. Der „Löschen“ Button ist verfügbar. Wenn im Suchbildschirm keine Zeile ausgewählt wurde und auf „Anzeigen“ geklickt wird, erscheint eine Fehlermeldung, dass genau eine Zeile markiert werden soll.

4.2.6 Testfall 6: Bearbeiten einer gesuchten Übung

Vorbedingungen: Testfall 4 wurde erfolgreich ausgeführt und der Suchbildschirm wird noch angezeigt.

Ablauf: Eine angezeigte Übung wird ausgewählt und durch Drücken des „Bearbeiten“ Buttons wird die ausgewählte Übung im Bearbeitungsmodus geöffnet. Nach Bearbeitung eines oder mehrerer Felder wird durch Drücken des „Speichern“ Buttons auf den Startbildschirm zurückgekehrt. Alternativ kann auch der „Abbrechen“ Button betätigt werden, um die Bearbeitung ohne Speichern der evtl. geänderten Daten zu verlassen.

Erwartetes Ergebnis: Die ausgewählte Übung wird im Bearbeitungsmodus angezeigt, d.h. alle Felder bis auf das Feld „ID“ sind bearbeitbar. Die Eingaben werden bei Betätigung des „Speichern“ Buttons zwischengespeichert und sind auf dem Startbildschirm in der Baumstruktur rechts und ggf. in der entsprechenden Tabelle links ersichtlich. Alternativ werden bei Betätigung des „Abbrechen“ Buttons die Datenänderungen verworfen und die Baumstruktur hat sich bzgl. der ausgewählten Übung nicht verändert. Falls die Übung bereits zu anderen untergeordneten Objekten verknüpft wurde, werden diese Verknüpfungen in den Tabellen im Bearbeitungsbildschirm als markierte Zeilen angezeigt. Wenn hier die Markierungen geändert werden, wird dies ebenfalls gespeichert und später in der Baumstruktur angezeigt.

Tatsächliches Ergebnis: OK - Aus dem Suchbildschirm heraus wird eine Übung ausgewählt und auf „Bearbeiten“ geklickt, daraufhin erscheint ein Bildschirm, in dem die Daten der ausgewählten Übung angezeigt werden. Alle Felder außer das Feld „ID“ sind bearbeitbar. Die geänderten Daten werden nach einem Klick auf „Speichern“ auf dem Startbildschirm in der Baumstruktur angezeigt. Im Falle einer Änderung des Namens ist dies auch in der Tabelle links ersichtlich. Klickt man auf „Abbrechen“ werden die Daten nicht gespeichert, in der Baumstruktur sind immer noch die gleichen Daten ersichtlich.

4.2.7 Testfall 7: Speichern der Änderungen auf der Datenbank

Vorbedingungen: Es wurden Änderungen an den Daten getätigt, die noch nicht auf der Datenbank gesichert sind, z.B. durch Testfall 2 oder Testfall 6.

Ablauf: Nach Durchlaufen von Testfall 2 oder Testfall 6 (Anlage oder Bearbeitung) wird auf dem Startbildschirm der Button „Änderungen sichern“ gedrückt, wodurch die bisher nur zwischengespeicherten Daten in die Datenbank geschrieben werden.

Erwartetes Ergebnis: Nachdem der Button gedrückt wurde, sollte sich das Änderungsdatum der Datenbankdatei _internal\data\main.db auf das aktuelle Datum und die aktuelle Uhrzeit geändert haben. Außerdem sollten beim nächsten Applikationsstart die gemachten Änderungen in den Objekten direkt auf dem Startbildschirm in der Baumstruktur rechts ersichtlich sein.

Tatsächliches Ergebnis: OK - Nach dem Klick auf „Änderungen sichern“ hat sich das Änderungsdatum der Datenbankdatei aktualisiert. Beim nächsten Applikationsstart waren die Daten wieder vorhanden.

4.2.8 Testfall 8: Verwerfen der Änderungen und erneutes Laden der Datenbank

Vorbedingungen: Es wurden Änderungen an den Daten getätigt, die noch nicht auf der Datenbank gesichert sind, z.B. durch Testfall 2 oder Testfall 6.

Ablauf: Nach Durchlaufen von Testfall 2 oder Testfall 6 (Anlage oder Bearbeitung einer Übung) wird auf dem Startbildschirm der Button „Änderungen verwerfen“ gedrückt, wodurch die bisher zwischengespeicherten Daten verworfen und durch die in der Datenbank gesicherten Daten ersetzt werden.

Erwartetes Ergebnis: Nachdem der Button gedrückt wurde, sollte sich die Datenbankdatei _internal\data\main.db nicht verändert haben. Außerdem sollten die Daten in der Baumstruktur rechts auf dem Startbildschirm den Datenstand widerspiegeln, der vor den gemachten Änderungen bestand. Beim Suchen sollte ein ggf. angelegtes und verworfenes Objekt nicht mehr zu finden sein.

Tatsächliches Ergebnis: OK - Nach dem Klick auf „Änderungen verwerfen…“ hat sich die Baumstruktur geändert und auf den zuletzt in der Datenbank gespeicherten Stand aktualisiert. Die in der Zwischenzeit neu eingegebene Übung wurde somit nicht gespeichert. Das Änderungsdatum der Datenbankdatei hat sich nicht aktualisiert.

4.2.9 Testfall 9: Anlegen und Verknüpfen eines Objektes

Vorbedingungen: Es existiert bereits ein Objekt, z.B. eine Übung durch die erfolgreiche Durchführung von Testfall 2. Der User befindet sich auf dem Startbildschirm.

Ablauf: Zu einem Objekt wird die Anlage gestartet, z.B. durch Klick auf „Einheit anlegen“. In dem daraufhin zu sehenden Bildschirm werden die Daten für das Objekt eingegeben. Zusätzlich wird in den Tabellen unterhalb der Datenfelder ein Eintrag eines anderen Objekts ausgewählt, der zum aktuellen Eintrag verknüpft werden soll. Es können auch mehrere Einträge gleichzeitig verknüpft werden. Sind alle Daten eingegeben, wird auf den Button „Speichern“ gedrückt, woraufhin man zum Startbildschirm zurückgelangt.

Erwartetes Ergebnis: Die eingegebenen Daten wurden analog Testfall 2 zwischengespeichert und sind in der Baumstruktur rechts auf dem Startbildschirm ersichtlich. Zusätzlich kann die Ebene des angelegten (Haupt-) Objekteintrages aufgeklappt werden, worunter man die damit verknüpften Objekte sieht. Die verknüpften Objekte sind ebenfalls im Anzeige-/Bearbeitungsbildschirm des Hauptobjektes ersichtlich und können im Bearbeitungsbildschirm auch geändert werden (siehe analog dazu Testfälle 5 und 6).

Tatsächliches Ergebnis: OK - Bei Anlage einer Einheit wurden die Daten in die Felder eingegeben und eine Übung in der Tabelle unterhalb ausgewählt. Nach dem Speichern der Einheit waren die eingegebenen Daten und die verknüpfte Übung als Unterpunkt der Einheit in der Baumstruktur auf dem Startbildschirm ersichtlich. Beim Anzeigen bzw. Bearbeiten der Einheit war die Verknüpfung ebenfalls als markierte Zeile ersichtlich.

4.2.10 Testfall 10: Exportieren eines Objektes

Vorbedingungen: Es existiert ein Objekt, das zu anderen Objekten verknüpft wurde (siehe Testfall 9). Bestenfalls soll ein Plan exportiert werden, es können aber auch alle anderen Objekte und deren Verknüpfungen exportiert werden.

Ablauf: Analog zu Testfall 4 wird nach dem zu exportierenden Objekt gesucht. Der gewünschte Objekteintrag wird in der Suchtabelle ausgewählt und der Button „Exportieren“ wird gedrückt. Alternativ kann für einen Plan der Exportbildschirm direkt vom Startbildschirm aus mit dem Button „Plan exportieren“ gestartet werden. Im nachfolgenden Export-Bildschirm wird im Falle eines Plans eine HTML-Ansicht der Planinhalte angezeigt, die später als PDF exportiert wird. Bei allen anderen Objekten wird die aufgeklappte Baumstruktur des Objekteintrages gezeigt, welche als JPG-Bilddatei exportiert wird. Danach wird der Button „Drucken“ gedrückt, woraufhin ein Dialog zur Auswahl des Dateinamens erscheint. Nach Eingabe des Dateinamens wird der Dialog bestätigt und man landet wieder auf dem Startbildschirm.

Erwartetes Ergebnis: Nach dem Drücken des „Drucken“ Buttons und der Bestätigung des Dateinamens auf dem Export-Bildschirm wird eine PDF- bzw. JPG-Datei im ausgewählten Verzeichnis mit dem eingegebenen Dateinamen erzeugt. Die PDF-Datei zeigt den Inhalt der HTML-Ansicht und die JPG-Datei zeigt die Baumansicht des Export-Bildschirms als der Button gedrückt wurde. Der User wird auf den Startbildschirm zurückgeleitet.

Tatsächliches Ergebnis: OK - Nach dem Klick auf „Exportieren“ aus dem Suchbildschirm heraus wird auf den Exportbildschirm gesprungen. Gleiches gilt für die Alternative, den Button „Plan exportieren“ auf dem Startbildschirm zu klicken. Hier wird der angelegte Plan mit den verknüpften Einheiten und Übungen detailliert in einer HTML-Ansicht dargestellt. Nach dem Klick auf „Drucken“ erscheint ein Dialog zur Auswahl des Dateinamens und -speicherorts. Nach Bestätigung des Dialogs springt die Anwendung zurück auf den Startbildschirm. Es liegt eine PDF-Datei unter dem eingegebenen Speicherort ab, welche den ausgewählten Plan und seine Inhalte zeigt.

4.2.11 Testfall 11: Löschen eines Objektes

Vorbedingungen: Es existiert ein Objekteintrag, der ggf. zu anderen Einträgen verknüpft wurde (siehe Testfall 9).

Ablauf: Analog zu Testfall 5 wird ein Objekteintrag, z.B. eine Übung, angezeigt. Der Button „Löschen“ wird geklickt und die erscheinende Abfrage wird entweder mit „Yes“ bestätigt oder mit „No“ abgelehnt. Nach Bestätigung der Abfrage wird dies mit einer Meldung quittiert und man gelangt auf den Startbildschirm zurück.

Erwartetes Ergebnis: Nach dem Klick auf „Löschen“ erscheint eine Abfrage, ob man den Eintrag wirklich löschen möchte. Wenn diese bestätigt wird, erscheint eine Meldung, welche das erfolgreiche Löschen des Eintrages mit der entsprechenden ID bestätigt, außerdem gelangt man zurück auf den Startbildschirm. In diesem Fall ist nun der Objekteintrag und alle Verknüpfungen zu anderen Einträgen gelöscht und auf dem Startbildschirm nicht mehr zu finden. Wenn die Abfrage abgelehnt wird, so geschieht nichts weiter, d.h. man verbleibt auf dem Anzeigebildschirm.

Tatsächliches Ergebnis: OK - Nach dem Klick auf „Löschen“ aus dem Anzeigebildschirm einer Übung heraus und Bestätigung der Abfrage mit „Yes“ erscheint eine Meldung, dass eine ID aus der Tabelle „EXERCISE“ gelöscht wurde. Die Übung ist auf dem Startbildschirm nicht mehr zu finden, weder in der Tabelle links noch in der Baumstruktur rechts. Wenn die Abfrage mit „No“ abgelehnt wird, so verbleibt man auf dem Anzeigebildschirm und nach einem Klick auf „Abbrechen“ ist die Übung weiterhin auf dem Startbildschirm ersichtlich.

4.3 Testergebnisse

Die Tests konnten fehlerfrei durchgeführt werden. Die zuvor in Phase 2 festgestellten Fehler konnten behoben werden, zudem sind bei den neuen Testfällen 10 und 11 keine weiteren Fehler aufgetreten.

4.4 Aufräumarbeiten

Nach der Durchführung aller Testfälle kann die Datenbankdatei _internal\data\main.db entweder gelöscht oder durch eine Kopie der vorbereiteten Datenbankdatei _internal\data\main_prefilled.db ausgetauscht werden.

Eventuell exportierte Objekteinträge, also deren PDF- bzw. JPG-Dateien, sollten im Anwendungsverzeichnis gelöscht werden.

5. Abstract

Das Projekt startete mit dem Ziel, eine Verwaltungssoftware für Sporttrainer zu erstellen, mit der Trainingseinheiten vorbereitet und geplant werden können. Dieses Ziel wurde über 3 Phasen hinweg in eine Anwendung umgesetzt, die in Python programmiert wurde. Dazu wurden nach dem Scrum Softwareprozess fünf Sprints durchgeführt, die jeweils eine Woche dauerten. Im Folgenden werden einige Aspekte des Projekts noch einmal genauer beleuchtet und kritisch reflektiert.

Direkt zu Beginn des Projekts wurden einige konzeptuelle Entscheidungen getroffen, die im späteren Projektverlauf überarbeitet werden mussten. Die meisten hiervon wurden jedoch bereits von Anfang an als Projektrisiken festgehalten, sodass die Alternativlösung jeweils keinen Zusatzaufwand bedeutete. So wurde zum einen die geplante MySQL-Datenbank durch eine SQLite-Datenbank ersetzt, da keine Userverwaltung notwendig war und somit Komplexität eingespart werden konnte. Zum anderen wurde die ursprünglich geplante Klassenstruktur verallgemeinert, sodass weniger Inhalte hart codiert werden mussten. Die anfängliche Idee, Videodateien direkt in die Datenbank hochzuladen, wurde ebenfalls nicht umgesetzt und durch einen einfachen URL-Link ersetzt, da die Datenbank bzw. das Anwendungsverzeichnis zu groß geworden wäre. Der geplante Export eines Plans als PDF hatte zum Ende der Phase 2 noch Probleme bereitet, konnte in der Finalisierungsphase dann aber noch umgesetzt werden. Für Phase 2 wurde hier eine Zwischenlösung eingeführt, die lediglich einen Screenshot der Anwendung erstellt und abspeichert. Diese Zwischenlösung ist auch weiterhin für die anderen Objekte im Einsatz. Durch das Feedback nach Phase 2 konnte die Benutzerfreundlichkeit und Intuitivität der Anwendung noch einmal überarbeitet und stark verbessert werden. Hierfür wurde in Phase 2 zu wenig Zeit eingeplant, stattdessen wurde mehr auf technische Funktionalität geachtet. Dieses Projektrisiko war ursprünglich nicht eingeplant, daher musste hierfür noch eine spontane Lösung gefunden werden. Die Definition der Projektrisiken half von Anfang an dabei, schnell auf unerwartete Probleme reagieren zu können und sollte in zukünftigen Projekten weiter fortgeführt werden.

Der in Phase 1 erstellte Zeitplan war sehr grob und konnte in Phase 2 detaillierter ausgearbeitet werden. Die am Anfang der Phase definierten Termine konnten bis zur Abgabe der Phase 2 alle eingehalten werden. Die für Phase 3 definierten Aufgabenpakete wurden nicht ganz in der Reihenfolge erstellt, da zum einen noch ein fünfter Sprint durchgeführt wurde und hier z.T. ein paralleles Bearbeiten der Anwendung und der Dokumentation stattfand. Die erwarteten Termine der Meilensteine waren korrekt abgeschätzt, mit Ausnahme des Meilensteins „Feature-Complete“, welcher erst eine Woche später als geplant erreicht wurde.

Die funktionalen Anforderungen und User Stories waren gut definiert und leiteten klar durch den gesamten Entwicklungsprozess. Bei Konflikten half die Einteilung in obligatorische und optionale Anforderungen sehr dabei, die Konflikte zu lösen und die Prioritäten richtig zu setzen. Hierzu passte auch der gewählte Softwareprozess Scrum, der den Fokus auf den Kundenanforderungen und funktionierenden Code legt. Diese agilen Kerngedanken haben sich durch die gesamte Entwicklungsphase und alle fünf Sprints gezogen. Die Organisation in Sprints half dabei, die Aufgabenpakete in

umsetzbare Untereinheiten aufzuteilen und für jeden Tag des Sprints einen Plan zu haben, was umgesetzt werden soll. Aus dem definierten Ziel eines jeden Sprints ließen sich dann spontan auch weitere Aufgabenpakete ableiten, die bei der Planung des Sprints noch nicht ersichtlich waren. Die Menge an erledigter Arbeit wurde insgesamt erhöht, wobei immer darauf geachtet wurde, so wenig wie möglich unnötige Arbeit zu verrichten. Das bedeutet, dass Dinge, die „man vielleicht mal brauchen könnte“, weggelassen wurden und sich immer darauf konzentriert wurde, das Sprintziel zu erreichen.

Das geplante regelmäßige „Daily“ Meeting aus Scrum wurde in der Form durchgeführt, dass zu Beginn eines Tages die noch offenen Aufgabenpakete aufgeschrieben wurden oder neue Pakete aus dem Sprintziel abgeleitet wurden. Das „Sprint Planning“ Meeting wurde meistens erst am ersten Tag des Sprints durchgeführt, was ein paar Zeitverzögerungen für diesen Tag mit sich brachte. Das „Sprint Review“ Meeting bestand darin, die Ergebnisse des Sprints in GitHub in den main-Branch zu mergen und kurz zusammenzufassen, was in diesem Sprint erledigt wurde. Falls noch offene Aufgabenpakete bestanden, wurden diese für den folgenden Sprint vorgemerkt. Die Verwaltung des Sprint Backlogs bzw. Product Backlogs hat gut funktioniert, dadurch war zu jedem Zeitpunkt klar, welche Aufgabenpakete abgeschlossen sind und wo noch etwas zu erledigen ist.

Beim Entwurf der Anwendungsstruktur wurde konsequent darauf geachtet, das MVC-Pattern umzusetzen. Dies hat für die Datenlogik problemlos funktioniert, jedoch gab es bei der Differenzierung zwischen Businesslogik und Visualisierung einige Konflikte, welche die Anwendung komplexer macht als notwendig. Außerdem musste der Grundsatz, so wenig wie möglich hart zu programmieren, bei der Implementierung des PDF-Exports eines Plans gebrochen werden. Das Singleton-Pattern, das für die Datenbankanbindung zum Einsatz kam, spielt in der aktuellen Auslieferung der Anwendung kaum eine Rolle, da die Anwendung sowieso immer nur eine Datenbank und ein Frontend ansprechen kann. Dennoch konnte hier ein persönlicher Lerneffekt erzielt werden, wie man in Python das Singleton-Pattern umsetzen kann.

Der Anwendungscode ist insgesamt nicht unnötig aufgeblasen, jedoch eher unstrukturiert in den einzelnen Modulen. Hier wäre ein Refactoring notwendig, um die relevanten Methoden schnell und einfach zu finden, damit auch bei zukünftigen Entwicklungen keine Doppelarbeit gemacht wird. Bei zukünftigen Projekten sollte darauf geachtet werden, das Refactoring direkt beim Einfügen neuer Methoden durchzuführen, sodass der Code immer gut strukturiert ist.

Die definierten Testfälle haben gegen Ende der Entwicklungsarbeit dabei geholfen, die Anwendung in natürlichen Worten beschreiben zu können und genau zu definieren, welches Programmverhalten erwartet wird. Außerdem konnten hierdurch noch einige Bugs ausfindig gemacht und ausgemerzt werden. Dies sollte für zukünftige Projekte beibehalten und weiter ausgebaut werden.

Der Aufbau der Anwendung als ausführbare Datei wurde erst spät im Projekt berücksichtigt und hat einige Hürden bereitgehalten, die schlussendlich aber noch gemeistert werden konnten. Der persönliche Lerneffekt hieraus wird in weitere Projekte getragen werden, bei denen die Auslieferung der Anwendung direkt im Konzept und in der Projektplanung berücksichtigt werden kann.

Verzeichnis der Anhänge

Anhang

Anhang 1: UML-Diagramm des data-Moduls mit Methoden