Projekt: SportApp

Projekt-Portfolio

im Modul

DLMCSPSE01\_D

„Projekt: Software Engineering“

Studiengang Informatik (Master 120 ECTS)

IU Internationale Hochschule

Bearbeiter: Jan Sauerland Betreuender Dozent: Prof. Dr. Markus Kleffmann  
Matrikelnummer: 10230630  
Datum: 07.09.2024

Inhaltsverzeichnis

[Inhaltsverzeichnis I](#_Toc176613673)

[Abbildungsverzeichnis III](#_Toc176613674)

[Tabellenverzeichnis IV](#_Toc176613675)

[1. Projektplan 1](#_Toc176613676)

[1.1 Ziele, Umfang und angestrebtes Ergebnis 1](#_Toc176613677)

[1.2 Zielgruppe 1](#_Toc176613678)

[1.3 Projektrisiken und Gegenmaßnahmen 1](#_Toc176613679)

[1.4 Zeitplan und Meilensteine 2](#_Toc176613680)

[2. Anforderungsdokument 5](#_Toc176613681)

[2.1 Management Summary 5](#_Toc176613682)

[2.2 Systemumfang und Kontext 5](#_Toc176613683)

[2.3 Funktionale Anforderungen 5](#_Toc176613684)

[2.4 Nicht-funktionale Anforderungen 7](#_Toc176613685)

[2.5 Glossar 7](#_Toc176613686)

[3. Projektdokumentation 9](#_Toc176613687)

[3.1 Softwareprozess 9](#_Toc176613688)

[3.2 Genutzte Technologien und Tools 10](#_Toc176613689)

[3.3 Anwendungsstruktur 11](#_Toc176613690)

[3.4 Technische Schuld 15](#_Toc176613691)

[3.5 Benutzeranleitung 16](#_Toc176613692)

[4. Testprotokoll 17](#_Toc176613693)

[4.1 Testumgebung und Testvorbereitung 17](#_Toc176613694)

[4.2 Testfälle 17](#_Toc176613695)

[4.2.1 Testfall 1: Applikationsstart und Anlage der Datenbank 17](#_Toc176613696)

[4.2.2 Testfall 2: Anlage einer Übung mit Speichern 17](#_Toc176613697)

[4.2.3 Testfall 3: Anlage einer Übung mit Abbrechen 18](#_Toc176613698)

[4.2.4 Testfall 4: Suchen einer angelegten Übung 18](#_Toc176613699)

[4.2.5 Testfall 5: Anzeigen einer gesuchten Übung 18](#_Toc176613700)

[4.2.6 Testfall 6: Bearbeiten einer gesuchten Übung 19](#_Toc176613701)

[4.2.7 Testfall 7: Speichern der Änderungen auf der Datenbank 20](#_Toc176613702)

[4.2.8 Testfall 8: Verwerfen der Änderungen und erneutes Laden der Datenbank 20](#_Toc176613703)

[4.2.9 Testfall 9: Anlegen und Verknüpfen eines Objektes 20](#_Toc176613704)

[4.2.10 Testfall 10: Exportieren eines Objektes 21](#_Toc176613705)

[4.3 Testergebnisse 21](#_Toc176613706)

[4.4 Aufräumarbeiten 22](#_Toc176613707)

[5. Abstract 23](#_Toc176613708)

[Verzeichnis der Anhänge 24](#_Toc176613709)

[Anhang 25](#_Toc176613710)

Abbildungsverzeichnis

Abbildung 1: Use-Case Diagramm "Sporttrainer" 7

Abbildung 2: UML-Diagramm des ursprünglich geplanten data-Moduls 11

Abbildung 3: UML-Diagramm der umgesetzten Klassenstruktur ohne Methoden 13

Abbildung 4: ER-Diagramm der Datenbankstruktur 14

Abbildung 5: UML-Aktivitätsdiagramm "Export eines Objekts" 15

Tabellenverzeichnis

Tabelle 1: Projektrisiken und Gegenmaßnahmen 1

Tabelle 2: Phase 1 „Konzeption" 2

Tabelle 3: Phase 2 "Erarbeitung & Reflexion" 2

Tabelle 4: Phase 3 "Finalisierung" 3

Tabelle 5: Meilensteine des Projekts 4

Tabelle 6: Glossar 7

# Projektplan

## 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.

## 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.

## 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 |

## 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 |

# Anforderungsdokument

## 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 ohne zusätzliche 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 Bilddatei exportiert werden können mit allen darin enthaltenen Einheiten und Übungen.

## 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.

## Funktionale Anforderungen

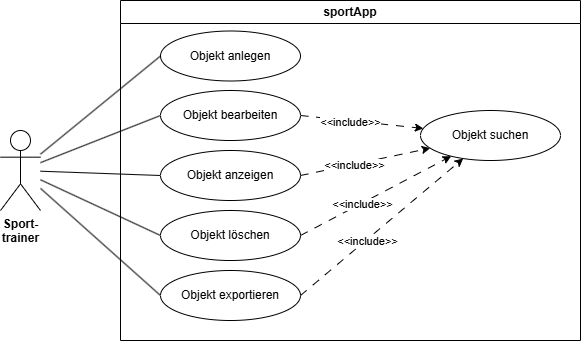
Folgende User Stories werden für die Anwendung als funktionale Anforderungen definiert:

(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 allgemeine Anwendungsfälle für den Akteur „Sporttrainer“ abgeleitet werden. Hierbei wurden die unterschiedlichen Objekte nicht einzeln genannt, da der Use Case für jedes Objekt (Übung, Einheit, Plan) ungeachtet der Objektart gleich aussieht.

Abbildung 1: Use-Case Diagramm "Sporttrainer"



## 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 [+]

## 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 Trainingstag umfassenden Plan |

# Projektdokumentation

Das GitHub-Repository der Anwendung kann unter folgendem Link gefunden werden:

<https://github.com/UltimateCuB3R/DLMCSPSE01_D>

Die Anwendung wird als ZIP-Datei ausgeliefert, die im GitHub-Repository als „sportApp\_v1.0.zip“ gefunden werden kann. In der ZIP-Datei liegen eine ausführbare Datei 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. 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.

## 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.

## 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.

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.

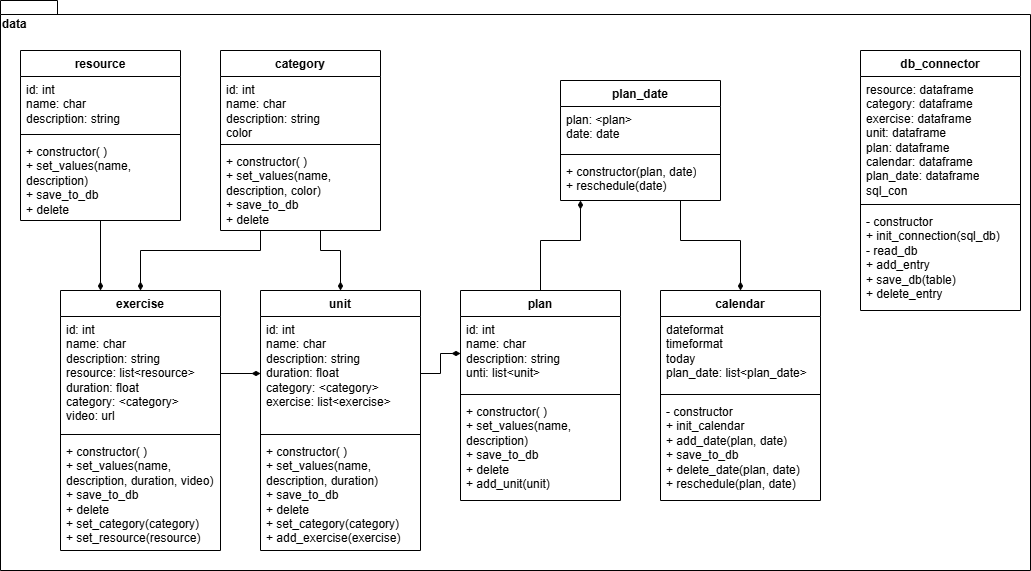
Bei der Programmierung wurde darauf geachtet, so wenig wie möglich hart zu codieren und stattdessen so viel wie möglich allgemein zu halten und zu dynamisieren. Aus diesem Grund wurden zwei XML-Dateien app\data\db\_def.xml und app\view\gui\_def.xml erstellt, die durch einen XML-Parser eingelesen und 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. Damit konnte einiges an Aufwand gespart werden und der Code im Allgemeinen wartungsfreundlicher gestaltet werden.

## 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 verworfen, da hiermit viele Dinge hätten mehrfach programmiert oder sogar hart codiert werden müssen. Um den Code möglichst dynamisch 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

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

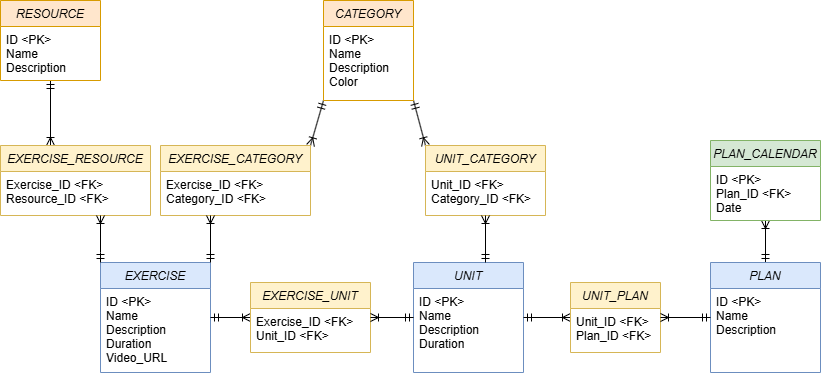
Automatisch generierte Beschreibung

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 sehen.

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"

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

## 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 im Kalender-Widget auf dem Startbildschirm
  + Grund: Zeitmangel
* Löschen von Objekten
  + Grund: Zeitmangel
* Konsolidierte Darstellung der Ressourcen unterhalb eines Trainingsplans
  + Grund: technische Schwierigkeiten
* Selbsterklärung der Bedienelemente durch Tooltips oder Quickinfos
  + Grund: Zeitmangel

Diese als optional definierten Anforderungen konnten aus den genannten Gründen in Phase 2 noch nicht umgesetzt werden. Falls in Phase 3 noch ausreichend Kapazität vorhanden ist, werden diese evtl. noch umgesetzt.

## Benutzeranleitung

(erst für Phase 3 relevant)

# Testprotokoll

## 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/>.

## Testfälle

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

### 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 Hauptbildschirm mit der Auswahlmöglichkeit der anzulegenden Tabellen, verschiedenen Buttons, einem Platzhalter für die Übersicht der angelegten Objekte (Baumstruktur) und eine Kalenderansicht.

**Tatsächliches Ergebnis:** OK - Datei main.db wird unter erwartetem Pfad angelegt und enthält die Datenbankdefinitionen für alle im Startbildschirm auswählbaren Tabellen und deren Beziehungstabellen.

### Testfall 2: Anlage einer Übung mit Speichern

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

**Ablauf:** In der Combobox wird das Objekt „EXERCISE“ ausgewählt, danach wird auf den Button „Anlegen“ gedrückt. 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 Hauptbildschirm zurückgelangt.

**Erwartetes Ergebnis:** Alle Felder sind bearbeitbar mit Ausnahme des Feldes „ID“. Die eingegebenen Daten wurden zwischengespeichert und sind direkt auf dem Hauptbildschirm rechts in der Baumstruktur ersichtlich. Es erscheint keine Fehlermeldung. Beim erneuten Aufruf des Anlagebildschirms von EXERCISE sind die eingegebenen Daten nicht mehr vorhanden.

**Tatsächliches Ergebnis:** OK - Nach Auswahl von EXERCISE und Klick auf „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 in der Baumstruktur ersichtlich. Beim erneuten Klick auf „Anlegen“ ist der Bildschirm wieder leer.

### Testfall 3: Anlage einer Übung mit Abbrechen

**Vorbedingungen:** Die Applikation ist gestartet

**Ablauf:** In der Combobox wird das Objekt „EXERCISE“ ausgewählt, danach wird auf den Button „Anlegen“ gedrückt. 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 Hauptbildschirm zurückgelangt.

**Erwartetes Ergebnis:** Die eingegebenen Daten wurden nicht gespeichert und sind nicht in der Baumstruktur rechts auf dem Hauptbildschirm ersichtlich. Es erscheint keine Fehlermeldung. Beim erneuten Aufruf des Anlagebildschirms von EXERCISE sind die eingegebenen Daten nicht mehr vorhanden.

**Tatsächliches Ergebnis:** OK - Nach Auswahl von EXERCISE und Klick auf „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 zu sehen.

### Testfall 4: Suchen einer angelegten Übung

**Vorbedingungen:** Testfall 2 wurde erfolgreich ausgeführt oder main\_prefilled.db wurde geladen.

**Ablauf:** In der Combobox wird das Objekt „EXERCISE“ ausgewählt, danach wird auf den Button „Suchen“ gedrückt. 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 der ausgewählten Objektebene („EXERCISE“) angezeigt.

**Tatsächliches Ergebnis:** OK - Nach dem Klick auf „Suchen“ wird die angelegte Übung angezeigt und kann ausgewählt werden. Die Baumstruktur wird unten angezeigt.

### Testfall 5: Anzeigen 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 „Anzeigen“ Buttons wird die ausgewählte Übung angezeigt. Nach Überprüfung der Inhalte kann durch Drücken des „Abbrechen“ Buttons wieder auf den Hauptbildschirm zurückgekehrt 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. Über „Abbrechen“ wird wieder auf den Hauptbildschirm zurückgekehrt. Die Daten in der Baumstruktur rechts auf dem Hauptbildschirm 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:** Teilweise 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 sind nicht bearbeitbar und „Speichern“ ist nicht möglich. Der Test, ob eine Fehlermeldung erscheint, wenn keine Zeile markiert wurde, konnte nicht ausgeführt werden, da beim erneuten Aufrufen des Suchbildschirms weiterhin die Zeile markiert ist und die Markierung nicht aufgehoben werden kann. Wenn die Applikation neu gestartet ist, man noch nie eine Zeile im Suchbildschirm markiert hat und dann auf „Anzeigen“ oder „Bearbeiten“ klickt, erscheint eine Fehlermeldung, dass genau eine Zeile markiert werden soll.

### 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 angezeigt. Nach Bearbeitung eines oder mehrerer Felder wird durch Drücken des „Speichern“ Buttons auf den Hauptbildschirm 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 Hauptbildschirm in der Baumstruktur rechts ersichtlich. Alternativ bei Betätigung des „Abbrechen“ Buttons werden 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:** Nicht 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 sind bearbeitbar, auch die ID (kritischer Fehler!). Die geänderten Daten werden nach einem Klick auf „Speichern“ im Startbildschirm in der Baumstruktur angezeigt. Klickt man auf „Abbrechen“ werden die Daten nicht gespeichert, in der Baumstruktur sind immer noch die gleichen Daten ersichtlich.

### 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 Hauptbildschirm der Button „Änderungen auf Datenbank 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 Hauptbildschirm in der Baumstruktur rechts ersichtlich sein.

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

### 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 Hauptbildschirm der Button „Änderungen verwerfen - Datenbankinhalte neu laden“ 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 Hauptbildschirm 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.

### 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 Hauptbildschirm.

**Ablauf:** In der Combobox wird ein Objekt ausgewählt, z.B. „UNIT“. Danach wird auf den Button „Anlegen“ gedrückt. In dem daraufhin zu sehenden Bildschirm werden die Daten für das Objekt eingegeben. Zusätzlich wird in den Tabellen unterhalb der Datenfelder ein Objekt einer anderen Objektart ausgewählt, die zum aktuellen Objekt verknüpft werden soll. Es können auch mehrere Objekte gleichzeitig verknüpft werden. Sind alle Daten eingegeben, wird auf den Button „Speichern“ gedrückt, woraufhin man zum Hauptbildschirm zurückgelangt.

**Erwartetes Ergebnis:** Die eingegebenen Daten wurden analog Testfall 2 zwischengespeichert und sind in der Baumstruktur rechts auf dem Hauptbildschirm ersichtlich. Zusätzlich kann die Ebene des angelegten (Haupt-) Objektes 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.

### 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. Das gewünschte Objekt wird in der Suchtabelle ausgewählt und der Button „Exportieren“ wird gedrückt. Im Nachfolgenden Export-Bildschirm wird die aufgeklappte Baumstruktur des Objekts gezeigt. 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 Hauptbildschirm.

**Erwartetes Ergebnis:** Nach dem Drücken des „Drucken“ Buttons und der Bestätigung des Dateinamens auf dem Export-Bildschirm wird eine Bilddatei im ausgewählten Verzeichnis mit dem eingegebenen Dateinamen erzeugt. Die Bilddatei zeigt den Export-Bildschirm als der Button gedrückt wurde. Der User wird auf den Hauptbildschirm zurückgeleitet.

**Tatsächliches Ergebnis:** OK - Nach dem Klick auf „Exportieren“ aus dem Suchbildschirm heraus wird auf den Exportbildschirm gesprungen. Hier wird der angelegte Plan mit den verknüpften Einheiten und Übungen als aufgeklappte Baumstruktur 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 Bilddatei unter dem eingegebenen Speicherort ab, welche die ausgewählte Übung und ihre Inhalte zeigt.

## Testergebnisse

Die Tests konnten größtenteils fehlerfrei durchgeführt werden. Bei Testfall 5 gab es eine leichte Abweichung, da eine einmal markierte Zeile im Suchbildschirm für die restliche Programmlaufzeit erhalten bleibt und die Markierung nicht aufgehoben werden kann. Bei Testfall 6 wurde ein kritischer Fehler festgestellt, da das Feld „ID“ geändert werden kann und somit Datenbankinkonsistenzen auftreten können.

Daher lassen sich aus den Testergebnissen folgende Arbeitspakete ableiten:

* Markierung im Suchbildschirm aufheben, nachdem dieser verlassen wurde
* Feld „ID“ nicht bearbeitbar machen, wenn ein Objekt im Bearbeitungsmodus angezeigt wird

## 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 Objekte bzw. deren Bilddateien sollten im Anwendungsverzeichnis gelöscht werden.

# Abstract

(erst für Phase 3 relevant)

Verzeichnis der Anhänge

Anhang 1: UML-Diagramm des data-Moduls mit Methoden 25

Anhang

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

