THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Bachelorarbeit

Konzeption, Design und Implementierung einer Softwarelösung zur automatisierten Erstellung und Verwaltung von Prüfungen - Der ExamBuilder

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Vorgelegt dem

Fachbereich Mathematik, Naturwissenschaften und Informatik der Technischen Hochschule Mittelhessen

Danielou Mounsande Sandamoun

im Dezember 2025

Referent der Arbeit: **Prof. Dr. Steffen Vaupel**

Korreferent: **Prof. Dr. Thorsten Weyer**

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Die Erstellung und Verwaltung von Prüfungen stellt für Lehrende oft einen zeitaufwändigen und mühsamen Prozess dar. Das manuelle Kopieren von Fragen aus alten Dokumenten, die uneinheitliche Formatierung, die fehleranfällige Neunummerierung der Aufgaben und die separate Erstellung eines Lösungsblatts sind nur einige der Hürden, die die Effizienz beeinträchtigen und die Qualität der Prüfungsunterlagen gefährden können. Mit der zunehmenden Digitalisierung im Bildungsbereich wächst der Bedarf an spezialisierten Werkzeugen, die diesen Prozess gezielt optimieren. Das vorliegende Projekt, der ***ExamBuilder***[1], adressiert diese Herausforderungen, indem es eine Softwarelösung zur effizienten Konzeption, Gestaltung und Implementierung von Prüfungen bereitstellt.

1.2 Zielsetzung der Arbeit

Ziel dieser Bachelorarbeit ist die Konzeption, das Design und die Implementierung einer benutzerfreundlichen Softwarelösung namens ***ExamBuilder***. Diese Anwendung soll Lehrende dabei unterstützen, Prüfungen automatisiert zu erstellen und zu verwalten. Im Fokus stehen dabei folgende Aspekte:

- Die Bereitstellung einer intuitiven grafischen Benutzeroberfläche zur einfachen Erfassung und Bearbeitung von Fragen und Prüfungsmetadaten.
- Die Implementierung von Funktionen zur Unterstützung verschiedener Fragetypen, einschließlich Multiple-Choice-Fragen und offener Fragen, sowie die Möglichkeit zur Integration von Bildern.
- Die automatisierte Generierung von Prüfungsdokumenten im Microsoft Word (.docx) Format, inklusive der Erstellung separater Lösungsblätter.
- Die Gewährleistung der Wartbarkeit und Erweiterbarkeit der Software durch eine modulare Architektur.

Die Arbeit zielt darauf ab, einen Prototyp zu entwickeln, der die Machbarkeit und den Nutzen eines solchen Tools demonstriert.

1.3 Aufbau der Arbeit

Die vorliegende Bachelorarbeit gliedert sich wie folgt:

- **Kapitel 2: Grundlagen und verwandte Arbeiten** führt in die theoretischen und technologischen Konzepte ein, die für die Entwicklung des ExamBuilders relevant sind, und analysiert bestehende Lösungen.
- **Kapitel 3: Anforderungsanalyse** detailliert die funktionalen und nicht-funktionalen Anforderungen an die Software.
- **Kapitel 4: Konzeption und Entwurf der Softwarelösung** beschreibt die Systemarchitektur, das Datenmodell und den UI/UX-Entwurf des ExamBuilders.

[1] Der Begriff **ExamBuilder** bedeutet auf Deutsch *Prüfungsersteller*.

- **Kapitel 5: Implementierung** erläutert die technische Umsetzung der Kernkomponenten und die Realisierung des Word-Exports.
- **Kapitel 6: Evaluation** präsentiert das Testkonzept, die Durchführung der Tests und eine kritische Würdigung der Ergebnisse.
- **Kapitel 7: Zusammenfassung und Ausblick** fasst die wichtigsten Erkenntnisse zusammen, zieht ein Fazit und gibt einen Ausblick auf mögliche zukünftige Erweiterungen.

2 Grundlagen und verwandte Arbeiten

2.1 Theoretische Grundlagen (MVC-Architekturmuster, UI/UX-Design)

Das Model-View-Controller (MVC)-Architekturmuster ist ein etabliertes Designmuster in der Softwareentwicklung, das darauf abzielt, die Anwendungslogik von der Benutzeroberfläche zu trennen. [1] Diese Trennung fördert die Modularität, Wartbarkeit und Testbarkeit von Software. Im MVC-Muster werden drei Hauptkomponenten unterschieden:

- **Model:** Repräsentiert die Daten und die Geschäftslogik der Anwendung. Es ist unabhängig von der Benutzeroberfläche und benachrichtigt die Views über Änderungen in seinem Zustand. [1]
- **View:** Ist für die Darstellung der Daten des Models verantwortlich. Es visualisiert den Zustand des Models und reagiert auf Benutzereingaben, indem es diese an den Controller weiterleitet. [1]
- **Controller:** Agiert als Vermittler zwischen Model und View. Es empfängt Benutzereingaben von der View, verarbeitet diese, aktualisiert das Model entsprechend und wählt die passende View zur Darstellung aus. [1]

Die Anwendung des MVC-Musters im ExamBuilder ermöglicht eine klare Strukturierung der Anwendung, erleichtert die Entwicklung und spätere Anpassungen.

Benutzeroberflächen- (UI) und Benutzererfahrungs- (UX) Design sind entscheidend für die Akzeptanz und Effektivität von Software. UI-Design konzentriert sich auf das visuelle Erscheinungsbild und die Interaktivität der Benutzeroberfläche, während UX-Design die gesamte Erfahrung des Benutzers mit dem Produkt umfasst. [2] Ein gutes UI/UX-Design zeichnet sich durch intuitive Navigation, klare Rückmeldungen und eine ansprechende Ästhetik aus. [2] Für den ExamBuilder bedeutet dies, eine Oberfläche zu schaffen, die Lehrenden eine einfache und effiziente Erstellung von Prüfungen ermöglicht, ohne durch komplexe Bedienung abgelenkt zu werden.

2.2 Fachliche Grundlagen (Didaktik der Prüfungserstellung)

Die Erstellung von Prüfungen ist nicht nur ein technischer, sondern auch ein didaktischer Prozess. Eine gut konzipierte Prüfung dient nicht nur der Leistungsüberprüfung, sondern auch der Förderung des Lernprozesses.[3] Im Kontext des ExamBuilders sind folgende didaktische Aspekte relevant:

2.2.1 Prüfungsformate und deren didaktischer Wert

- **Offene Fragen:** Sie fördern die Fähigkeit zur freien Formulierung, Argumentation und Strukturierung von Wissen. Sie erlauben eine tiefgehende Überprüfung des Verständnisses.[4, 5]
- **Multiple-Choice-Fragen (MCQ):** Effizient in der Korrektur können sie ein breites Spektrum an Wissen abfragen. Sie erfordern jedoch eine sorgfältige Konstruktion, um reines Raten zu minimieren und tatsächlich das Verständnis zu prüfen. Distraktoren (falsche Antwortmöglichkeiten) müssen plausibel, aber eindeutig falsch sein.[4, 5]
- **Lückentextfragen:** Überprüfen spezifisches Faktenwissen oder das Verständnis von Zusammenhängen, indem Schlüsselbegriffe ergänzt werden müssen.[4, 5]

- **Richtig/Falsch-Fragen:** Schnell zu beantworten und zu korrigieren, aber anfällig für das Raten. Sie eignen sich gut zur Überprüfung von grundlegendem Wissen oder zur Identifizierung von Missverständnissen.[4, 5]

Der ExamBuilder unterstützt die Erstellung dieser gängigen Fragetypen, um Lehrenden Flexibilität in der didaktischen Gestaltung ihrer Prüfungen zu bieten.

2.2.2 Qualitätsmerkmale von Prüfungsaufgaben

- **Objektivität:** Das Ergebnis der Prüfung sollte unabhängig von der Person des Korrigierenden sein. Standardisierte Bewertungsraster und Musterlösungen tragen dazu bei.[5]
- **Reliabilität:** Eine Prüfung ist reliabel, wenn sie bei wiederholter Durchführung unter gleichen Bedingungen zu denselben Ergebnissen führt. Dies wird durch klare Aufgabenstellungen und eindeutige Bewertungskriterien gefördert.[5]
- **Validität:** Eine Prüfung ist valide, wenn sie tatsächlich das misst, was sie messen soll. Dies erfordert eine sorgfältige Abstimmung der Aufgaben auf die Lernziele und Inhalte der Lehrveranstaltung.[5]
- **Transparenz:** Die Erwartungen an die Studierenden und die Bewertungskriterien sollten klar kommuniziert werden.[5]

Durch Funktionen wie die detaillierte Musterlösungserfassung und die Möglichkeit zur präzisen Formatierung trägt der ExamBuilder indirekt zur Verbesserung dieser Qualitätsmerkmale bei.

2.2.3 Automatisierung und Individualisierung

Die Möglichkeit, Prüfungsvarianten zu generieren (z.B. durch das Mischen von Unterfragen oder das Ersetzen von Synonymen), unterstützt die Individualisierung von Prüfungen und kann dazu beitragen, Täuschungsversuchen vorzubeugen. Gleichzeitig entlastet die Automatisierung bei der Erstellung von Lösungsblättern und der Formatierung die Lehrenden erheblich, sodass sie sich stärker auf die inhaltliche Qualität der Aufgaben konzentrieren können.

2.3 Technologische Grundlagen

Der ExamBuilder basiert auf einer Reihe von Schlüsseltechnologien, die für die Entwicklung moderner Desktop-Anwendungen im Java-Ökosystem relevant sind.

2.3.1 Java und das JavaFX Framework

Java ist eine weit verbreitete, objektorientierte Programmiersprache, die für ihre Plattformunabhängigkeit und Robustheit bekannt ist. JavaFX ist ein modernes GUI-Toolkit für Java, das eine reichhaltige Bibliothek von UI-Komponenten und eine leistungsstarke Grafik-Engine bietet. Es ermöglicht die Entwicklung von plattformübergreifenden Desktop-Anwendungen mit einer ansprechenden Benutzeroberfläche. [6]

2.3.2 Apache Maven für das Build-Management

Apache Maven ist ein leistungsstarkes Build-Automatisierungstool, das auf dem Konzept eines ***Project Object Model*** (POM) basiert. Es standardisiert den Build-Prozess, verwaltet Abhängigkeiten, kompiliert Code, führt Tests durch und erstellt ausführbare Artefakte. [7] Die Verwendung von Maven im ExamBuilder gewährleistet einen konsistenten und reproduzierbaren Build-Prozess.

2.3.3 Apache POI für den Microsoft Word Dokumenten-Export

Apache POI[1] ist eine Open-Source-Bibliothek, die es ermöglicht, Microsoft Office-Formate (wie Word, Excel und PowerPoint) mit Java-Programmen zu lesen und zu schreiben. [8] Im ExamBuilder wird Apache POI verwendet, um die generierten Prüfungen und Lösungsblätter im .docx-Format zu exportieren. Dies bietet eine hohe Kompatibilität mit gängigen Textverarbeitungsprogrammen.

2.3.4 Jsoup für die HTML-Verarbeitung

Jsoup ist eine Open-Source-Java-Bibliothek, die entwickelt wurde, um mit realem HTML zu arbeiten. Sie bietet eine sehr bequeme API zum Extrahieren und Manipulieren von Daten, die die besten DOM-, CSS- und jQuery-ähnlichen Methoden nutzt. [9] Im ExamBuilder wird Jsoup eingesetzt, um den vom ‘HTMLEditor‘ generierten HTML-Code zu parsen. Die Bibliothek wandelt den HTML-String in eine DOM-Struktur um, die dann rekursiv durchlaufen werden kann. Dies ermöglicht eine gezielte Umwandlung von HTML-Tags (z.B. ‘<b>‘, ‘<i>‘) in entsprechende, formatierte ‘XWPFRun‘- oder ‘XWPFParagraph‘-Objekte von Apache POI, was für die Konvertierung von formatiertem Text in das Word-Dokument entscheidend ist.

2.3.5 Alternative Technologien und Begründung der Wahl

Bei der Auswahl der Technologien für den ExamBuilder wurden verschiedene Optionen evaluiert, um die bestmögliche Balance zwischen Entwicklungsaufwand, Funktionalität und Wartbarkeit zu finden.

Alternative UI-Frameworks

Neben JavaFX wurden auch andere Java-basierte UI-Frameworks wie **Swing** und **AWT** in Betracht gezogen. Diese sind zwar etabliert und bieten eine hohe Kompatibilität, gelten jedoch im Vergleich zu JavaFX als weniger modern und flexibel in der Gestaltung ansprechender Benutzeroberflächen. JavaFX wurde aufgrund seiner umfangreichen Bibliothek an UI-Komponenten, der Unterstützung für FXML zur deklarativen UI-Gestaltung und seiner leistungsstarken Grafik-Engine bevorzugt, die eine moderne und reaktionsfähige Benutzererfahrung ermöglicht.

Alternative Dokumentengenerierung

Ursprünglich wurde auch die Generierung von PDF-Dokumenten in Erwägung gezogen, wofür Bibliotheken wie **iText** oder **Apache PDFBox** zur Verfügung stehen. Diese Anforderung wurde jedoch im Projektverlauf zugunsten einer Fokussierung auf den Word-Export (.docx) fallen gelassen, da Microsoft Word das primäre Arbeitswerkzeug der Zielgruppe ist. Für den Word-Export wurde Apache POI gewählt, da es eine umfassende und robuste API zur direkten Manipulation von .docx-Dateien bietet, was eine präzise Formatierung und Inhaltskontrolle ermöglicht. Alternativen wie die direkte

[1] Das Akronym **POI** steht inoffiziell für "Poor Obfuscation Implementation", ein Name, der sich humorvoll auf die Komplexität der zugrunde liegenden Microsoft-Formate bezieht.

Generierung von WordML (Office Open XML) wurden aufgrund der höheren Komplexität und des damit verbundenen Entwicklungsaufwands verworfen.

Andere Anwendungsplattformen

Obwohl der ExamBuilder als Java-Desktop-Anwendung konzipiert wurde, existieren auch plattformübergreifende Frameworks wie **Electron** (basierend auf Webtechnologien) oder native Ansätze mit Sprachen wie **C#** (für Windows mit WPF/WinForms) oder **Python** (mit PyQt/PySide). Die Entscheidung für Java und JavaFX basierte auf der vorhandenen Expertise im Entwicklungsteam, der starken Plattformunabhängigkeit von Java und der guten Integration in das bestehende Java-Ökosystem, was die Nutzung von Tools wie Maven und Bibliotheken wie Apache POI vereinfacht.

2.4 Analyse bestehender Lösungen

Bevor mit der Konzeption des ExamBuilders begonnen wurde, erfolgte eine Analyse bestehender Lösungsansätze zur Prüfungserstellung. Ziel war es, Stärken und Schwächen aktueller Methoden zu identifizieren und daraus die Notwendigkeit sowie die spezifischen Anforderungen für den ExamBuilder abzuleiten.

Der traditionelle Ansatz, den viele Lehrende verfolgen, basiert auf der manuellen Erstellung von Prüfungen mit Standard-Textverarbeitungsprogrammen wie Microsoft Word. Dieser Prozess bietet zwar maximale Flexibilität in der Gestaltung, ist jedoch äußerst zeitaufwändig, fehleranfällig bei der Formatierung und erschwert die systematische Wiederverwendung und Verwaltung von Fragen.

Am anderen Ende des Spektrums existieren umfassende digitale Prüfungsplattformen wie ***Exam.net*** oder ***eXaminer***. Diese Systeme sind primär für die Durchführung von Online- oder sicheren Offline-Prüfungen konzipiert und bieten oft komplexe Funktionalitäten, die über das Ziel der reinen Dokumentenerstellung hinausgehen. Ebenso gibt es breite Schulverwaltungs-Software wie ***ESDi***, in denen die Prüfungserstellung nur eine untergeordnete Funktion darstellt. Solche Lösungen sind für den spezifischen Anwendungsfall, schnell und unkompliziert eine druckfertige Prüfung im .docx-Format zu generieren, oft überdimensioniert und erfordern eine hohe Einarbeitungszeit.

Der ExamBuilder positioniert sich bewusst in der Lücke zwischen diesen beiden Extremen. Er ist kein System zur Durchführung digitaler Prüfungen, sondern ein spezialisiertes Desktop-Werkzeug, das den etablierten, dokumentenbasierten Arbeitsablauf von Lehrenden gezielt optimiert. Die Analyse hat gezeigt, dass bestehende Lösungen entweder zu rudimentär (manuelle Word-Erstellung) oder zu komplex und überdimensioniert (umfassende Prüfungsplattformen) für den spezifischen Bedarf der effizienten Erstellung druckfertiger Prüfungsdokumente sind. Der ExamBuilder schließt diese Nische, indem er eine benutzerfreundliche Oberfläche mit leistungsstarken Automatisierungsfunktionen für die Erstellung, Verwaltung und den Export von Prüfungsdokumenten in das universelle .docx-Format kombiniert. Dies ermöglicht Lehrenden eine erhebliche Zeitersparnis und Qualitätssteigerung bei der Erstellung traditioneller, papierbasierter Prüfungen, ohne die Notwendigkeit einer aufwendigen Einarbeitung in komplexe Systeme. Seine Notwendigkeit ergibt sich somit aus dem ungedeckten Bedarf an einem fokussierten, effizienten und leicht bedienbaren Werkzeug für die dokumentenbasierte Prüfungserstellung.

2.5 Zielszenarien: Zwei Repräsentative Klausuren

Um die Herausforderungen der manuellen Prüfungserstellung und die Lösungsansätze des ExamBuilders greifbar zu machen, werden in dieser Arbeit zwei repräsentative Beispielklausuren als durchgängige Zielszenarien verwendet. Diese sollen ein breites Spektrum an Anwendungsfällen abdecken.

Das erste Zielszenario, eine Klausur aus dem Fach *Projektmanagement* (siehe Anhang 7.4), dient dazu, die grundlegende Notwendigkeit und die Funktionsweise der Kernfeatures des ExamBuilders zu kontextualisieren. Diese Klausur ist so konzipiert, dass sie eine typische Mischung aus verschiedenen Anforderungsarten enthält:

- **Struktur und Hierarchie:** Die Klausur besteht aus mehreren Hauptaufgaben, die wiederum in mehrere, teils verschachtelte Teilaufgaben untergliedert sind.
- **Inhaltstypen:** Sie enthält nicht nur reinen Text, sondern auch Aufzählungslisten und Bilder.
- **Layout-Anforderungen:** Bestimmte Hauptaufgaben sollen zur besseren Übersichtlichkeit auf einer neuen Seite beginnen.

Als zweites, technisch fokussiertes Zielszenario wird eine Probeklausur aus dem Fach *Praktische Informatik 1* herangezogen (siehe Anhang 7.6). Diese Klausur zeichnet sich durch einen hohen Anteil an formatiertem Quellcode (HTML, TypeScript), textuellen Diagrammen und spezifischen informatischen Fragestellungen aus. Sie dient dazu, die Fähigkeiten des ExamBuilders im Umgang mit technisch anspruchsvollen Inhalten und präzisen Layout-Anforderungen (z.B. für Codeblöcke und seitenfüllende Aufgaben) zu demonstrieren.

Diese Kombination aus zwei Szenarien verdeutlicht die Komplexität, die bei der Erstellung unterschiedlicher Prüfungstypen entsteht. Die manuelle Handhabung dieser Aspekte in einem Textverarbeitungsprogramm ist zeitaufwendig und fehleranfällig. Der ExamBuilder zielt darauf ab, genau diesen Prozess zu strukturieren und zu automatisieren. In den folgenden Kapiteln wird immer wieder auf diese Zielszenarien Bezug genommen, um die Implementierung und Evaluation der Software zu veranschaulichen.

3 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die Softwarelösung *ExamBuilder* detailliert. Es wird zwischen funktionalen Anforderungen, die beschreiben, was das System tun soll, und nicht-funktionalen Anforderungen, die die Qualitätsmerkmale des Systems definieren, unterschieden.[10]

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen definieren die konkreten Funktionen und Fähigkeiten, die der ExamBuilder dem Benutzer zur Verfügung stellen muss [10].

3.1.1 FA1.0 Anlegen eines Projektes

Die Anwendung muss es dem Benutzer ermöglichen, ein neues, leeres Prüfungsprojekt zu initialisieren. Dies beinhaltet die Erstellung einer neuen Instanz des 'Exam'-Objekts mit Standardwerten und die Vorbereitung der Benutzeroberfläche für die Eingabe neuer Prüfungsdaten. **Beispiel:** Der Benutzer klickt auf 'Neues Projekt' und sieht eine leere Oberfläche mit Standardmetadaten.

3.1.2 FA1.1 Speichern eines Projektes

Die Anwendung muss es dem Benutzer ermöglichen, ein vollständiges Prüfungsprojekt zu jedem Zeitpunkt zu sichern. Dies wird durch eine Speicherfunktion realisiert, die den gesamten Zustand der Prüfung, einschließlich aller Metadaten, Fragen, Bilder und Konfigurationen, in einer einzigen Datei im JSON-Format serialisiert. **Beispiel:** Nach Eingabe von Metadaten und Fragen speichert der Benutzer das Projekt als 'MeineErstePruefung.json'.

3.1.3 FA1.2 Laden eines Projektes

Ein entsprechender Lade-Dialog muss es erlauben, eine zuvor gespeicherte Projektdatei wieder in die Anwendung zu importieren, um die Arbeit nahtlos fortzusetzen oder Prüfungen zu archivieren. **Beispiel:** Der Benutzer lädt 'MeineErstePruefung.json' und sieht alle zuvor eingegebenen Daten wiederhergestellt.

3.1.4 FA2 Erstellung und Verwaltung von Fragen

Das Kernstück der Anwendung ist die hierarchische Verwaltung von Fragen. Das System muss die Erstellung von Hauptfragen und beliebig tief verschachtelten Unterfragen unterstützen. Eine tabellarische Baumansicht (TreeTableView)[1] dient zur Visualisierung dieser Hierarchie. Der Benutzer kann Fragen hinzufügen, ihre Position in der Hierarchie bearbeiten und sie wieder löschen. Die Auswahl einer Frage in der Baumansicht lädt deren Details in einen Bearbeitungsbereich. **Beispiel:** Der Benutzer fügt eine Hauptfrage und zwei Unterfragen hinzu, verschiebt eine Unterfrage und löscht eine andere.

[1] Ein *TreeTableView* ist eine UI-Komponente, die eine Baumstruktur (zur Darstellung von Hierarchien) mit den Spalten einer Tabelle kombiniert. Sie eignet sich ideal zur Anzeige von hierarchischen Daten wie Fragen und Unterfragen mit ihren jeweiligen Eigenschaften (z.B. Titel, Punkte).

3.1.5 FA3 Erfassung von Musterlösungen

Für jede erstellte Frage und Unterfrage muss ein separates Feld zur Verfügung stehen, in das eine detaillierte Musterlösung eingegeben werden kann. Diese Information wird mit der jeweiligen Frage gespeichert und dient ausschließlich als Grundlage für die spätere Generierung eines separaten Lösungsdokuments. **Beispiel:** Für eine offene Frage wird eine detaillierte Musterlösung eingegeben und gespeichert.

3.1.6 FA4 Verwaltung von Prüfungsmetadaten

Um ein standardisiertes Deckblatt für die Prüfung zu generieren, muss die Software Eingabefelder für wesentliche Metadaten bereitstellen. Dazu gehören der Titel der Prüfung, der Name der Hochschule oder Universität, der zuständige Fachbereich, die Modulbezeichnung sowie das betreffende Semester. Diese Daten werden zentral im Prüfungsprojekt gespeichert. **Beispiel:** Der Benutzer gibt Titel, Hochschule, Fachbereich, Modul und Semester ein, die auf dem Deckblatt erscheinen sollen.

3.1.7 FA5 Rich-Text-Formatierung für Fragen

Der Text einer Frage muss über einen integrierten Rich-Text-Editor (HTMLEditor) formatiert werden können. Dies ermöglicht grundlegende Stiloptionen wie Fett- und Kursivschrift, Unterstreichungen, Listen und farbliche Hervorhebungen. Die so erstellten HTML-formatierten Inhalte müssen sowohl in der Anwendung korrekt dargestellt als auch beim Export in das Word-Dokument interpretiert und umgewandelt werden. **Beispiel:** Der Benutzer formatiert einen Teil des Fragetextes fett und kursiv und fügt eine Aufzählungsliste hinzu.

3.1.8 FA6 Integration von Bildern in Fragen

Benutzer müssen die Möglichkeit haben, eine Bilddatei (z.B. PNG, JPG) von ihrem lokalen System auszuwählen und zu einer Frage hinzuzufügen. Das Bild wird in der Benutzeroberfläche als Vorschau angezeigt und intern als Base64-String kodiert direkt im Prüfungsprojekt gespeichert. Beim Export wird das Bild dekodiert und in das Zieldokument eingebettet. **Beispiel:** Der Benutzer fügt ein PNG-Bild zu einer Frage hinzu, das in der Vorschau und im Export sichtbar ist.

3.1.9 FA7 Konfiguration von Prüfungshinweisen

Eine dedizierte Dialogbox muss zur Verfügung stehen, um die Rahmenbedingungen der Prüfung zu konfigurieren. Dies umfasst ein Textfeld für allgemeine Anweisungen, eine Reihe von Checkboxen für standardmäßige Hilfsmittel (z.B. Taschenrechner, Formelsammlung) sowie ein Eingabefeld für weitere, frei definierbare Hilfsmittel. Zusätzlich muss die Bearbeitungszeit der Prüfung in Minuten einstellbar sein. **Beispiel:** Der Benutzer wählt 'Taschenrechner' und 'Formelsammlung' als Hilfsmittel aus und setzt die Bearbeitungszeit auf 120 Minuten.

3.1.10 FA8 Manuelle Layout-Steuerung

Um dem Lehrenden eine präzise Kontrolle über das finale Erscheinungsbild des Dokuments zu geben, müssen folgende Funktionen zur manuellen Layout-Steuerung implementiert werden:

3.1.10.1 FA8.1 Seitenumbruch-Steuerung

Der Benutzer muss die Möglichkeit haben, vor jeder Frage oder Unterfrage einen manuellen Seitenumbruch zu erzwingen. Dies wird durch eine dedizierte Checkbox in der Fragen-Tabelle realisiert und stellt sicher, dass logisch zusammengehörige Blöcke nicht durch einen automatischen Seitenumbruch getrennt werden. **Beispiel:** Der Benutzer markiert eine Hauptfrage, um sie auf einer neuen Seite beginnen zu lassen, und eine Unterfrage, um einen Seitenumbruch mit Fortsetzungsmeldung zu erzwingen.

3.1.10.2 FA8.2 Selektive Text-Justierung

Für längere Fragentitel muss eine Option zur Verfügung stehen, um den Text als Blocksatz zu formatieren. Die Aktivierung dieser Funktion, ebenfalls über eine Checkbox in der Fragen-Tabelle, ermöglicht eine saubere, ästhetisch ansprechende Darstellung von mehrzeiligen Fragestellungen. **Beispiel:** Der Benutzer aktiviert Blocksatz für einen langen Fragentitel, der daraufhin im Word-Export als Blocksatz formatiert wird.

3.1.11 FA9 Generierung von Prüfungsvarianten

Die Anwendung soll eine Funktion bieten, um eine alternative Version einer Prüfung zu erstellen. Diese Funktion kombiniert zwei Modifikationen: Erstens wird der Titel und Text jeder Frage durch einen einfachen Thesaurus-basierten Algorithmus analysiert, um einzelne Wörter durch Synonyme zu ersetzen. Zweitens wird die Reihenfolge von Unterfragen innerhalb einer Hauptfrage zufällig gemischt. Um jedoch ein vom Autor bewusst gestaltetes Seitenlayout nicht zu zerstören, wird dieser Mischvorgang für eine Gruppe von Unterfragen übersprungen, falls eine dieser Fragen eine manuelle Seitenumbruch-Markierung aufweist. **Beispiel:** Der Benutzer generiert eine Variante, bei der Synonyme im Fragetext ersetzt und Unterfragen gemischt werden, außer bei Fragen mit Seitenumbruch.

3.1.12 FA10 Selektiver Export von Fragen

Für den Export muss der Benutzer die Flexibilität haben, nur eine Teilmenge der erstellten Fragen auszuwählen. Dies wird durch eine Checkbox in jeder Zeile der Fragen-Tabelle realisiert. Vor dem Export prüft das System, ob eine Auswahl getroffen wurde. Ist dies nicht der Fall, wird der Benutzer gefragt, ob stattdessen alle Fragen exportiert werden sollen. **Beispiel:** Der Benutzer wählt nur 3 von 5 Fragen aus und exportiert diese in ein Word-Dokument.

3.1.13 FA11 Export als Word-Dokument

Die primäre Exportfunktion der Anwendung ist die Generierung eines professionell formatierten Microsoft Word-Dokuments (.docx). Dieser Export muss ein standardisiertes Deckblatt mit den zuvor erfassten Metadaten, eine Bewertungstabelle, die formatierten Fragen und, je nach Fragentyp, eine angemessene Anzahl von leeren Antwortzeilen enthalten. **Beispiel:** Ein Projekt mit Deckblatt, Metadaten und formatierten Fragen wird erfolgreich als .docx-Datei exportiert.

3.1.14 FA12 Generierung eines separaten Lösungsblatts

Zusätzlich zur Prüfungsversion für die Studierenden muss das System in der Lage sein, ein separates Lösungsdokument zu generieren. Dieses Dokument soll im Aufbau identisch zur Prüfung sein, jedoch anstelle der leeren Antwortzeilen die zuvor erfassten Musterlösungen enthalten, um eine schnelle und einfache Korrektur zu ermöglichen. **Beispiel:** Ein Lösungsblatt wird generiert, das alle Musterlösungen anstelle der leeren Antwortzeilen enthält.

3.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Qualitätsmerkmale und Randbedingungen, unter denen das System seine Funktionen erbringen muss[10].

3.2.1 Benutzerfreundlichkeit (Usability)

Die Anwendung richtet sich an Lehrende, die nicht zwangsläufig über tiefgreifende technische Kenntnisse verfügen. Daher ist eine hohe Benutzerfreundlichkeit von entscheidender Bedeutung. Das System muss eine intuitive und selbsterklärende grafische Benutzeroberfläche (GUI) bieten. Dies wird durch ein klares, in logische Bereiche unterteiltes Layout, den Einsatz von Tooltips zur Erklärung von Funktionen und eine konsistente Bedienlogik erreicht. Komplexe Aktionen wie die Konfiguration der Prüfungshinweise werden in dedizierte, leicht verständliche Dialogfenster ausgelagert, um den Benutzer schrittweise durch den Prozess zu führen.

3.2.2 Wartbarkeit (Maintainability)

Die Software muss so strukturiert sein, dass sie in Zukunft leicht korrigiert, angepasst oder erweitert werden kann. Um dies zu gewährleisten, wird die Anwendung nach dem Model-View-Controller (MVC) Architekturmuster entwickelt. Diese klare Trennung zwischen Datenmodell (Model), Benutzeroberfläche (View) und Anwendungslogik (Controller) reduziert die Komplexität und ermöglicht es Entwicklern, Änderungen an einer Komponente vorzunehmen, ohne unerwartete Nebenwirkungen in den anderen zu verursachen. Die Organisation des Quellcodes in logische Pakete (z.B. ‘model‘, ‘controller‘, ‘service‘) trägt ebenfalls zur Übersichtlichkeit und damit zur Wartbarkeit bei.

3.2.3 Plattformunabhängigkeit (Platform Independence)

Der ExamBuilder soll auf den gängigsten Betriebssystemen (Windows, macOS, Linux) lauffähig sein, um eine maximale Reichweite unter den Lehrenden zu gewährleisten. Diese Anforderung wird durch die Wahl der zugrundeliegenden Technologie erfüllt. Die Implementierung basiert vollständig auf der Programmiersprache Java und dem GUI-Toolkit JavaFX. Dank der Java Virtual Machine (JVM), auf der die Anwendung ausgeführt wird, ist der kompilierte Code portabel und kann ohne betriebssystemspezifische Anpassungen auf jeder Plattform ausgeführt werden, die eine kompatible Java-Laufzeitumgebung bereitstellt.

4 Konzeption und Entwurf der Softwarelösung

In diesem Kapitel werden die konzeptionellen und gestalterischen Entscheidungen beschrieben, die der Implementierung des ExamBuilders zugrunde liegen. Es werden die gewählte Systemarchitektur, das entworfene Datenmodell und die Gestaltung der Benutzeroberfläche erläutert.

4.1 Systemarchitektur nach dem MVC-Pattern

Für die Anwendungsarchitektur des ExamBuilders wurde das etablierte Model-View-Controller (MVC) Entwurfsmuster gewählt. Dieses Muster fördert eine saubere Trennung der Verantwortlichkeiten zwischen der Datenhaltung (Model), der Darstellung (View) und der Anwendungslogik (Controller), was die Wartbarkeit und Testbarkeit des Systems maßgeblich verbessert. [1] Wie in Abbildung 4.1 dargestellt, gliedert sich die konkrete Umsetzung im Projekt in drei Kernkomponenten:

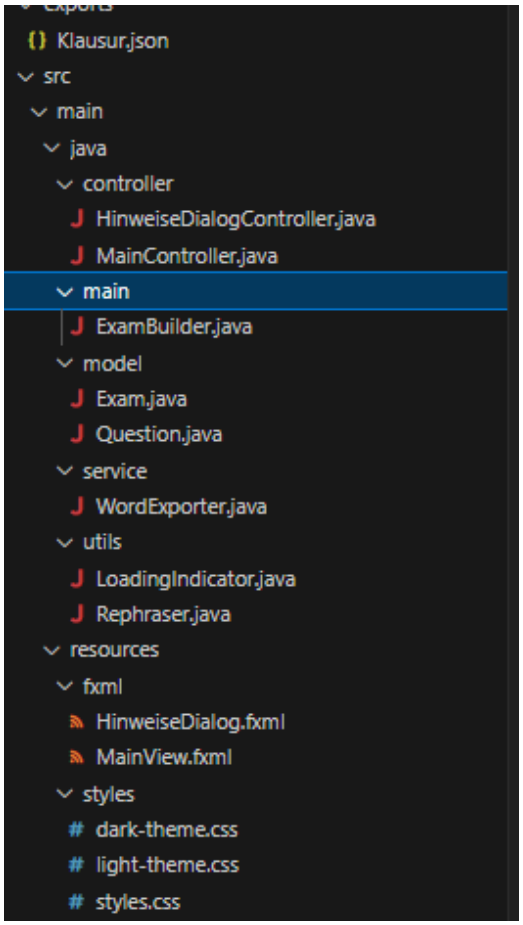


Abbildung 4.1: Architektur des ExamBuilders nach dem MVC-Muster.

- **Model:** Die Klassen im ‘model‘-Paket, namentlich ‘Exam‘ und ‘Question‘, repräsentieren die Datenstrukturen der Anwendung. Sie enthalten die reinen Anwendungsdaten sowie die Geschäftslogik, die direkt auf diesen Daten operiert (z.B. die Berechnung der Gesamtpunktzahl in der ‘Exam‘-Klasse).
- **View:** Die Darstellung der Benutzeroberfläche wird durch die FXML-Dateien im ‘resources/fxml‘-Verzeichnis definiert (‘MainView.fxml‘, ‘HinweiseDialog.fxml‘). Diese deklarativen XML-Dateien beschreiben die Struktur und das Layout der UI-Komponenten und sind somit für das Erscheinungsbild der Anwendung verantwortlich.
- **Controller:** Die Klassen im ‘controller‘-Paket (‘MainController‘, ‘HinweiseDialogController‘) fungieren als Bindeglied zwischen Model und View. Sie nehmen Benutzereingaben aus der View entgegen, verarbeiten diese, aktualisieren das Model und sorgen im Gegenzug dafür, dass Änderungen im Model in der View korrekt dargestellt werden.

Zusätzlich wurde ein ‘service‘-Paket implementiert, um komplexe, wiederverwendbare Geschäftslogik auszulagern. Ein zentrales Beispiel hierfür ist der ‘WordExporter‘, der die gesamte Logik für die Erstellung der .docx-Dateien kapselt. Dieser Service nutzt die Apache POI Bibliothek, um die Interaktion mit dem externen Word-Dokumentformat zu ermöglichen. Durch diese Auslagerung bleibt der Controller

schlank und fokussiert auf seine primäre Aufgabe der UI-Koordination, während die spezifische Dokumentengenerierungslogik im Service-Layer gekapselt ist.

4.1.1 Die Klasse Exam

Die Klasse ‘Exam‘ ist das Wurzelobjekt des Datenmodells (siehe Abbildung 4.2). Sie kapselt alle globalen Informationen einer Prüfung. Dazu gehören die in der Anforderungsanalyse definierten Metadaten wie ‘title‘, ‘hochschule‘, ‘fachbereich‘, ‘module‘ und ‘semester‘. Darüber hinaus speichert sie die konfigurierten Prüfungshinweise (‘allgemeineHinweise‘), die Bearbeitungszeit (‘bearbeitungszeit‘) und eine Liste der erlaubten Hilfsmittel (‘hilfsmittel‘). Das wichtigste Attribut ist eine Liste von ‘Question‘-Objekten, die den inhaltlichen Kern der Prüfung bilden.

4.1.2 Die Klasse Question

Die Klasse ‘Question‘ modelliert eine einzelne Aufgabe. Jede Frage besitzt einen ‘title‘, einen ‘text‘ (der HTML-formatiert sein kann), eine definierte Punktzahl (‘points‘) und einen Typ (‘type‘, z.B. Öffene Frage"). Ein optionales Attribut ‘imageBase64‘ kann ein Bild als Base64-kodierten String aufnehmen. Zur Korrektur wird zudem eine ‘musterloesung‘ gespeichert. Neu hinzugekommen sind die Attribute ‘startOnNewPage‘ (Boolean), das festlegt, ob die Frage auf einer neuen Seite beginnen soll, und ‘justify‘ (Boolean), das die Blocksatzformatierung des Fragetitels steuert.

Das entscheidende Merkmal der ‘Question‘-Klasse ist ihre Fähigkeit zur hierarchischen Strukturierung: Jedes ‘Question‘-Objekt enthält eine weitere Liste von ‘Question‘-Objekten, ‘subQuestions‘. Dies ermöglicht die Abbildung von Hauptfragen mit beliebig tief verschachtelten Teilaufgaben. Die Logik zur Punktevergabe ist so implementiert, dass eine Hauptfrage automatisch die Summe der Punkte ihrer Unterfragen als eigene Punktzahl annimmt, was die konsistente Punktevergabe sicherstellt.

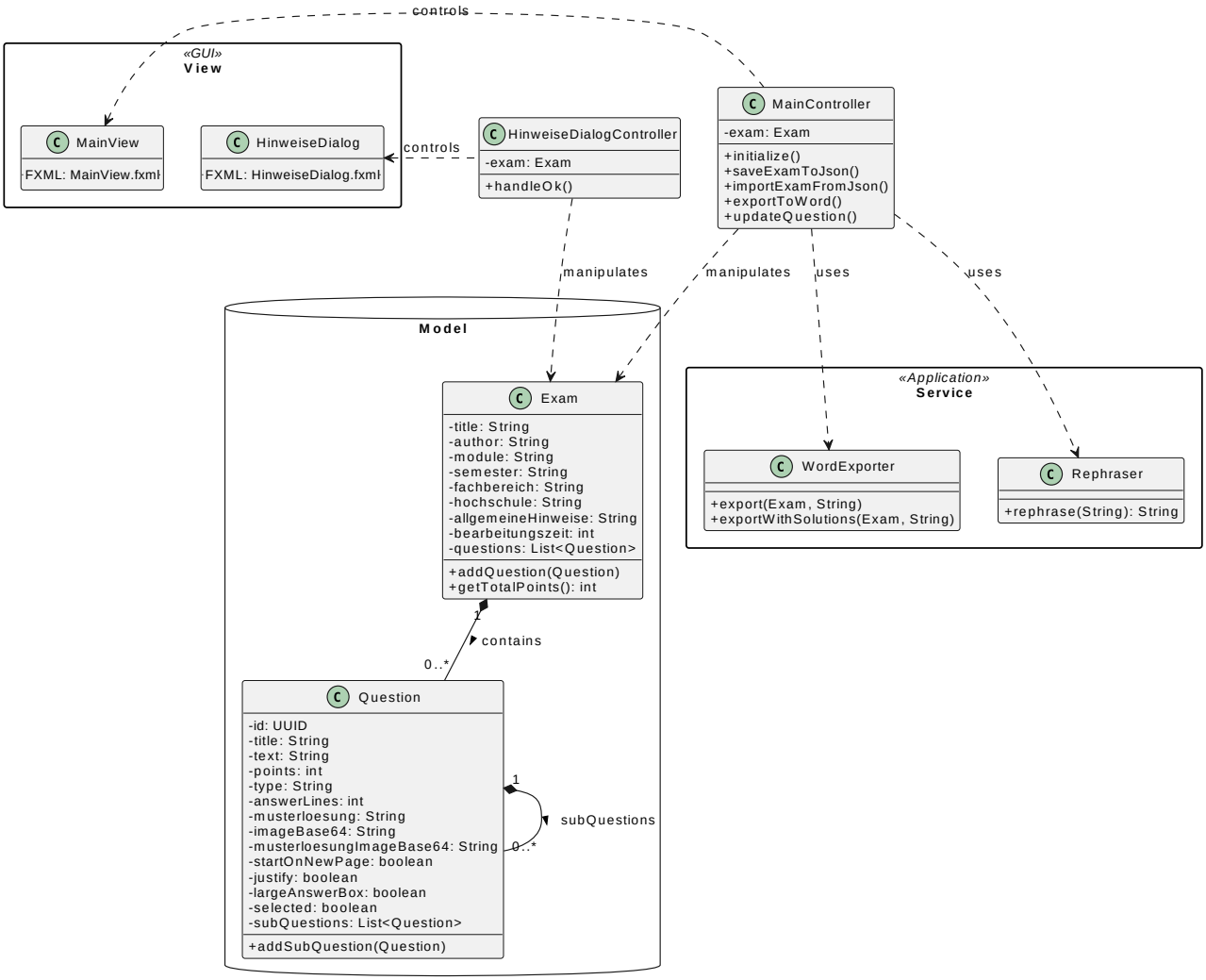


Abbildung 4.2: UML-Klassendiagramm des Datenmodells.

4.2 UI/UX-Entwurf der grafischen Benutzeroberfläche

Der Entwurf der Benutzeroberfläche zielt darauf ab, den Prozess der Prüfungserstellung so intuitiv und effizient wie möglich zu gestalten. Das Hauptfenster der Anwendung ist in drei logische Bereiche aufgeteilt, die einen klaren Arbeitsablauf von links nach rechts und von oben nach unten unterstützen (siehe Abbildung 4.4).

- **Oberer Bereich:** Hier werden die globalen Metadaten der Prüfung (Titel, Modul etc.) erfasst. Ein zentraler Button ermöglicht den Zugriff auf den Dialog zur Konfiguration der Prüfungshinweise.
- **Zentraler Bereich:** Das Kernstück der Ansicht ist die ‘TreeTableView‘, die alle Fragen und Unterfragen in ihrer hierarchischen Struktur anzeigt. Sie gibt dem Benutzer einen ständigen Überblick über den Gesamtumfang und die Gliederung der Prüfung.
- **Rechter Bereich:** Dieser Bereich ist dem Editieren gewidmet. Er enthält alle Formularfelder zur Bearbeitung der Details einer ausgewählten Frage (Titel, Text, Punkte, Bild etc.). Er ist nur aktiv, wenn eine Frage bearbeitet oder neu erstellt wird.

4.2.1 Dialog für Prüfungshinweise

Um die Hauptansicht übersichtlich zu halten und den Benutzer nicht mit zu vielen Optionen zu überfordern, wurde die Konfiguration der allgemeinen Prüfungshinweise in eine separate, modale Dialogbox ausgelagert (siehe Abbildung 4.3). Dieser Dialog bündelt alle zugehörigen Einstellungen an einem Ort: ein Textfeld für allgemeine Anweisungen, eine Reihe von Checkboxen für häufig genutzte Hilfsmittel und ein Spinner zur Einstellung der Bearbeitungszeit. Ein besonderes Merkmal zur Steigerung der Benutzerfreundlichkeit ist die integrierte Live-Vorschau. Jede Änderung, die der Benutzer an den Einstellungen vornimmt, wird sofort in einem Vorschaufeld aktualisiert. Dies gibt dem Benutzer direktes visuelles Feedback und ermöglicht ihm, das exakte Erscheinungsbild des Hinweistextes zu kontrollieren, bevor er die Änderungen übernimmt.

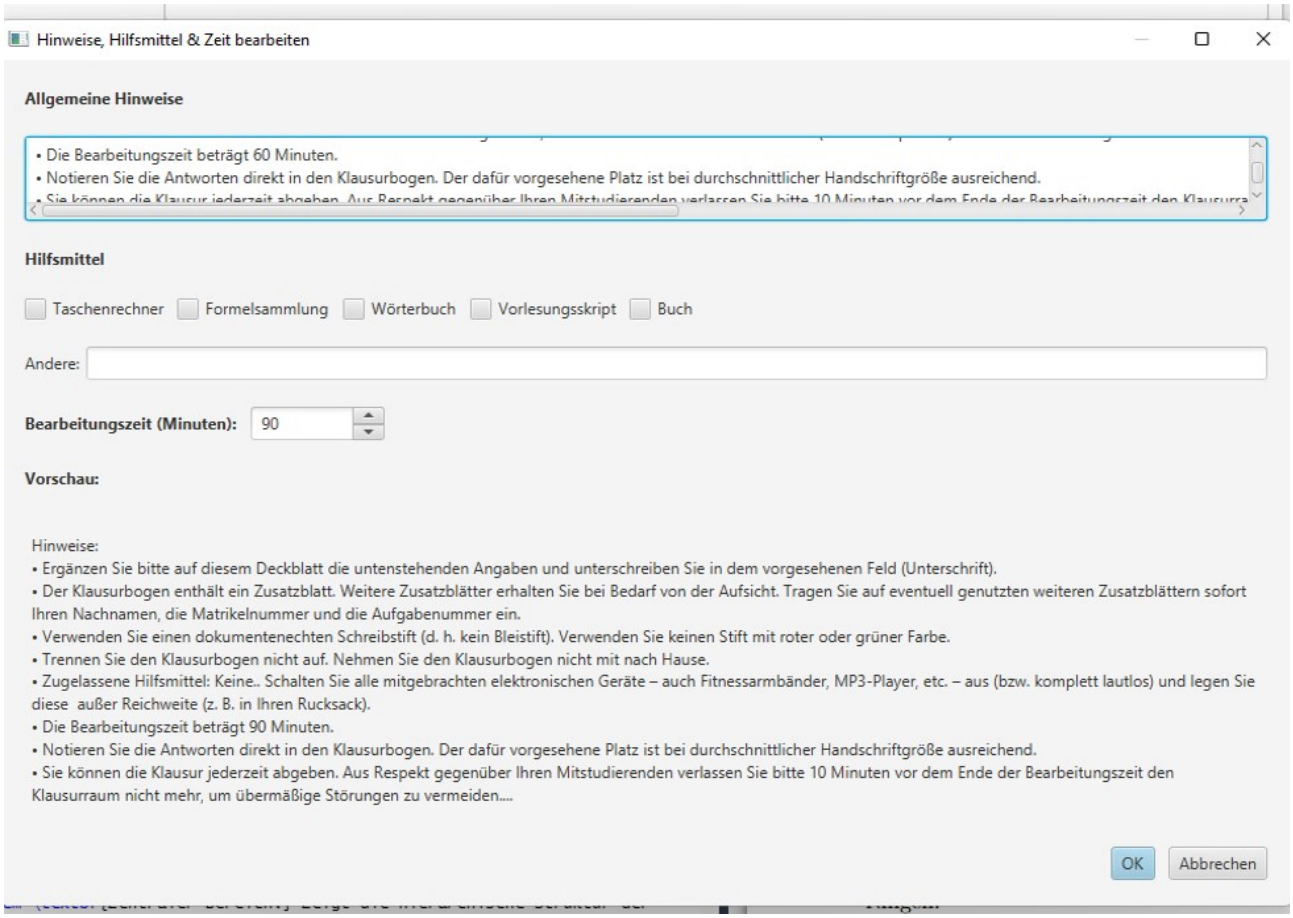


Abbildung 4.3: Dialog zur Konfiguration der Prüfungshinweise.

4.2.2 Wireframes der Hauptansichten

Die folgende Abbildung zeigt eine schematische Darstellung (Wireframe) der Hauptansicht, um die Anordnung der UI-Elemente zu visualisieren:

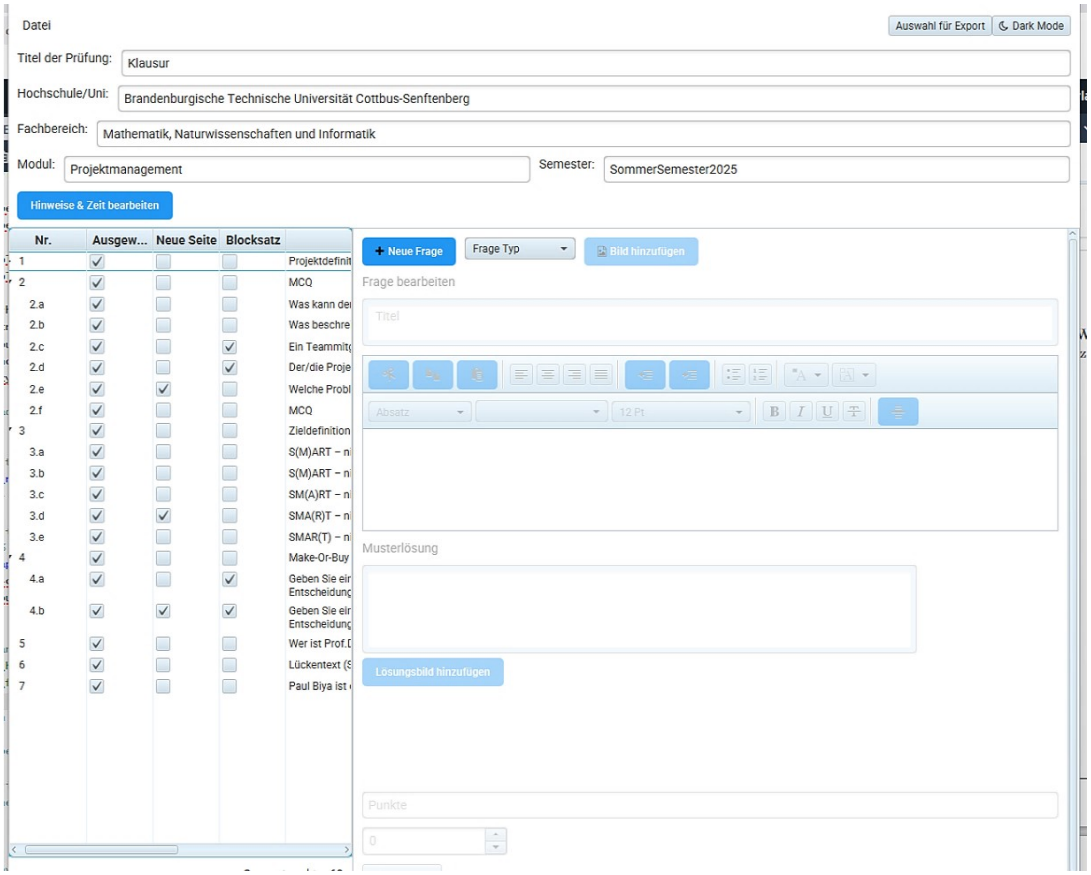


Abbildung 4.4: Wireframe der Hauptansicht des ExamBuilders.

5 Implementierung

5.1 Allgemeine Implementierungsdetails

Dieses Kapitel beschreibt die technische Umsetzung des ExamBuilders. Zunächst werden die allgemeinen Implementierungsdetails der Kernkomponenten (Model, View, Controller) erläutert, die als Grundlage für die Realisierung der einzelnen funktionalen Anforderungen dienen.

5.1.1 Umsetzung der Kernkomponenten (Model, View, Controller)

In diesem Abschnitt wird die technische Umsetzung der Kernkomponenten des ExamBuilders gemäß dem im vorherigen Kapitel beschriebenen MVC-Architekturmuster erläutert.

5.1.1.1 Implementierung des Models

Das Datenmodell wurde als Plain Old Java Objects (POJOs)[1] in den Klassen ‘Exam‘ und ‘Question‘ umgesetzt. Diese Klassen enthalten primär private Attribute zur Kapselung der Daten sowie öffentliche Getter- und Setter-Methoden für den kontrollierten Zugriff. Eine Besonderheit ist die Verwendung von JavaFX Properties, wie zum Beispiel die ‘BooleanProperty‘ für das ‘selected‘-Attribut, ‘startOnNewPage‘ und ‘justify‘ in der ‘Question‘-Klasse. Dieser Ansatz ermöglicht es der Benutzeroberfläche, sich direkt an Datenänderungen im Model zu binden (Data Binding), was zu einer reaktiven und konsistenten UI führt. Die hierarchische Natur von Fragen wird durch eine ‘List<Question>‘ innerhalb der ‘Question‘-Klasse selbst realisiert, was eine rekursive und flexibel tiefe Verschachtelung von Aufgaben erlaubt.

5.1.1.2 Implementierung der View

Die Benutzeroberfläche wurde deklarativ mit FXML, einer XML-basierten Sprache von JavaFX, erstellt. Die FXML-Dateien (‘MainView.fxml‘, ‘HinweiseDialog.fxml‘) definieren die statische Struktur, das Layout und die Komponenten der einzelnen Ansichten. Dieser Ansatz hat den entscheidenden Vorteil, dass die Gestaltung der Oberfläche vollständig von der Anwendungslogik getrennt ist. Änderungen am Design erfordern somit keine Eingriffe in den Java-Code, was die Entwicklung und Wartung erheblich vereinfacht.

5.1.1.3 Implementierung des Controllers

Die Controller-Klassen, insbesondere der ‘MainController‘, bilden das Herz der Anwendungslogik. Sie sind über die ‘fx:controller‘-Anweisung in der FXML-Datei mit ihrer jeweiligen Ansicht verknüpft. Die ‘@FXML‘-Annotation wird im Java-Code verwendet, um die in der FXML-Datei definierten UI-Komponenten (z.B. ‘TextField‘, ‘TreeTableView‘) direkt in den Controller zu injizieren und sie programmatisch manipulierbar zu machen. Wie in Abbildung 5.1 dargestellt, werden die Felder der Klasse, die UI-Komponenten repräsentieren, mit ‘@FXML‘ annotiert. Die ‘initialize()‘-Methode dient als primärer Einstiegspunkt nach dem Laden der FXML-Datei, um Event-Handler zu registrieren, die

[1] Ein *Plain Old Java Object* (POJO) ist ein einfaches Java-Objekt, das keine speziellen Framework-Abhängigkeiten oder komplexen Schnittstellen implementiert und primär der reinen Datenhaltung dient. [11]

Tabelle zu konfigurieren und den initialen Zustand der Anwendung herzustellen. Der ‘MainController‘ hält die zentrale Instanz des ‘Exam‘-Objekts und orchestriert alle Interaktionen zwischen dem Datenmodell und der Benutzeroberfläche.

```
1  public class MainController {
2
3      @FXML
4      private TreeTableView<Question> questionsTable;
5
6      @FXML
7      private HTMLEditor questionTextField;
8
9      @FXML
10     private Button newQuestionButton;
11
12     @FXML
13     private void exportToWord() {
14         // ... Logik für den Export
15     }
16
17     // ... weitere Methoden und Felder
18 }
```

Abbildung 5.1: Beispiel für die Injektion von FXML-Elementen im Controller.

5.2 Implementierung der Funktionalen Anforderungen

In diesem Abschnitt wird die Umsetzung der einzelnen funktionalen Anforderungen (FA) detailliert beschrieben, wobei auf die entsprechenden Abschnitte in Kapitel 3 verwiesen wird.

5.2.1 FA1 Projekt- und Datenverwaltung

Die Implementierung der Projekt- und Datenverwaltung umfasst die Initialisierung, das Speichern und Laden von Prüfungsprojekten.

5.2.1.1 FA1.0 Anlegen eines Projektes (siehe Anforderung 3.1.1)

Das Anlegen eines neuen Projektes wird durch die Methode ‘newQuestion()‘ im ‘MainController‘ initiiert, die den aktuellen Bearbeitungszustand zurücksetzt und eine leere ‘Exam‘-Instanz vorbereitet.

5.2.1.2 FA1.1 Speichern eines Projektes (siehe Anforderung 3.1.2)

Die Persistenz der erstellten Prüfungen wird durch die JSON-Serialisierung mittels der Java-Bibliothek *Jackson*(‘com.fasterxml.jackson.databind‘) realisiert. [12] Die Methode ‘saveExamToJson()‘ im ‘MainController‘ ist für diesen Prozess verantwortlich. Sie konvertiert das ‘Exam‘-Objekt in einen formatierten JSON-String und speichert diesen in einer vom Benutzer gewählten Datei. Abbildung 5.2 zeigt eine vereinfachte Darstellung dieser Methode.

```java
@FXML
private void saveExamToJson() {
    updateExamMetadata(); // Stellt sicher, dass Metadaten aktuell sind
    try {
        ObjectMapper mapper = new ObjectMapper();
        mapper.enable(SerializationFeature.INDENT_OUTPUT); // Für lesbare JSON-Dateien

        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save Exam as JSON");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("JSON Files", "*.json"));

        File file = fileChooser.showSaveDialog(mainPane.getScene().getWindow());
        if (file != null) {
            mapper.writeValue(file, exam); // Serialisierung des 'exam'-Objekts
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Abbildung 5.2: Speichern des Exam-Objekts als JSON-Datei mit der Jackson-Bibliothek.

5.2.1.3 FA1.2 Laden eines Projektes (siehe Anforderung 3.1.3)

Der Ladevorgang wird durch die Methode `importExamFromJson()` im `MainController` abgewickelt. Wie in Abbildung 5.3 dargestellt, wird zunächst ein `FileChooser` geöffnet, der dem Benutzer die Auswahl einer `.json`-Datei ermöglicht.

Nach der Auswahl wird eine Instanz des `ObjectMapper` aus der Jackson-Bibliothek verwendet, um die Datei zu lesen. Die Methode `readValue()` deserialisiert den JSON-Inhalt direkt in eine neue `Exam`-Objektinstanz. Eine wichtige Konfiguration ist hierbei `DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES = false`. Diese Einstellung stellt die Abwärtskompatibilität sicher, da sie verhindert, dass der Import fehlschlägt, wenn die JSON-Datei Felder enthält, die im aktuellen `Exam`-Datenmodell nicht mehr existieren.

Nach dem erfolgreichen Laden wird die Methode `updateUIFromExam()` aufgerufen, um die Benutzeroberfläche mit den Daten der importierten Prüfung zu aktualisieren.

```java
@FXML
private void importExamFromJson() {
    try {
        // 1. Datei-Auswahldialog für den Benutzer öffnen
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Exam JSON File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("JSON Files", "*.json"));
        Stage stage = (Stage) mainPane.getScene().getWindow();
        File file = fileChooser.showOpenDialog(stage);

        if (file != null) {
            // 2. Jackson ObjectMapper für die Deserialisierung vorbereiten
            ObjectMapper mapper = new ObjectMapper();
            // Stellt sicher, dass der Import nicht fehlschlägt, wenn neue Felder im JSON fehlen
            mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

            // 3. JSON-Datei in ein Exam-Objekt umwandeln
            exam = mapper.readValue(file, Exam.class);

            // 4. Benutzeroberfläche mit den neuen Daten aktualisieren
            updateUIFromExam();
        }
    } catch (IOException e) {
        e.printStackTrace(); // Fehlerbehandlung für den Fall, dass  Datei nicht gelesen werden kann
    }
}
```

Abbildung 5.3: Implementierung des JSON-Imports. Der Code zeigt, wie eine Prüfungsdatei ausgewählt, mit der Jackson-Bibliothek geparst und das resultierende Objekt zur Aktualisierung der Benutzeroberfläche verwendet wird.

5.2.2 FA2 Erstellung und Verwaltung von Fragen (siehe Anforderung 3.1.4)

Die Implementierung der hierarchischen Fragenverwaltung erfolgt hauptsächlich über die ‘TreeTableView‘ in der ‘MainView.fxml‘ und die zugehörigen Methoden im ‘MainController‘. Das Hinzufügen, Bearbeiten und Löschen von Fragen wird durch die Methoden ‘addQuestion()‘, ‘editQuestion()‘, ‘updateQuestionAndReturnSuccess()‘ und ‘deleteQuestion()‘ gesteuert. Die hierarchische Struktur wird durch die rekursive Methode ‘populateSubQuestions()‘ und die ‘Question‘-Klasse selbst (mit ihrer ‘List<Question> subQuestions‘) abgebildet.

Die Methode ‘addQuestion()‘, gezeigt in Abbildung 5.4, ist zentral für das Hinzufügen neuer Fragen. Sie prüft zunächst, ob eine gültige Punktzahl eingegeben wurde. Anschließend wird die Hilfsmethode ‘createQuestionFromInput()‘ aufgerufen, um ein neues ‘Question‘-Objekt aus den Eingabefeldern der Benutzeroberfläche zu erstellen. Die Methode unterscheidet, ob eine Hauptfrage oder eine Unterfrage hinzugefügt werden soll, indem sie die Variable ‘parentForSubQuestion‘ prüft. Je nachdem wird die neue Frage entweder der Hauptfragenliste des ‘Exam‘-Objekts oder der Sub-Fragenliste der Elternfrage hinzugefügt. Abschließend wird die ‘TreeTableView‘ aktualisiert, die Eingabefelder zurückgesetzt und der Bearbeitungsmodus beendet.

```java
@FXML
private void addQuestion() {
    // Validierung der Punkteeingabe
    if (isPointsInvalid()) {
        return;
    }

    // Erstellt ein neues Question-Objekt aus den UI-Eingabefeldern
    Question newQuestion = createQuestionFromInput();

    // Prüft, ob eine Sub-Frage oder eine Hauptfrage hinzugefügt wird
    if (parentForSubQuestion != null) {
        // Fügt die neue Frage zur Liste der Sub-Fragen der Elternfrage hinzu
        parentForSubQuestion.getValue().addSubQuestion(newQuestion);
        parentForSubQuestion = null; // Kontext zurücksetzen
    } else {
        // Fügt die neue Frage zur Liste der Hauptfragen des Examens hinzu
        exam.addQuestion(newQuestion);
    }

    // Aktualisiert die TreeTableView, um die neue Frage anzuzeigen
    refreshTreeTableView();
    // Leert die Eingabefelder für die nächste Eingabe
    clearQuestionFields();
    // Deaktiviert den Bearbeitungsmodus
    setEditMode(false);
}
```

Abbildung 5.4: Implementierung des Hinzufügens von Fragen. Die Methode unterscheidet zwischen Haupt- und Unterfragen und aktualisiert anschließend die Benutzeroberfläche.

5.2.3 FA3 Erfassung von Musterlösungen (siehe Anforderung 3.1.5)

Die Erfassung von Musterlösungen wird durch das ‘musterloesungField‘ (TextArea) und ‘musterloesungImageView‘ (ImageView) in der Benutzeroberfläche ermöglicht. Die Daten werden direkt im ‘Question‘-Objekt gespeichert und bei Bedarf vom ‘WordExporter‘ für das Lösungsblatt verwendet. Abbildung 5.5 zeigt die beiden relevanten Code-Ausschnitte. Beim Speichern einer Frage in der Methode ‘updateQuestionAndReturnSuccess‘ wird der textuelle Inhalt aus dem ‘musterloesungField‘ direkt in das ‘Question‘-Objekt übernommen. Für Bild-Musterlösungen liest die Methode ‘addSolutionImage‘ die ausgewählte Bilddatei, kodiert sie in einen Base64-String und speichert diesen im ‘musterloesungImageBase64‘-Attribut des ‘Question‘-Objekts.

```java
// Ausschnitt aus der Methode updateQuestionAndReturnSuccess()
// Speichert den Text aus dem Textfeld in das Question-Objekt.
Question questionToUpdate = selectedItem.getValue();
questionToUpdate.setMusterloesung(musterloesungField.getText());

// Ausschnitt aus der Methode addSolutionImage()
// Wird aufgerufen, wenn der Benutzer ein Lösungsbild hinzufügt.
if (selectedFile != null) {
    // Liest die Bilddatei als Byte-Array.
    byte[] fileContent = Files.readAllBytes(selectedFile.toPath());
    // Kodiert die Bytes in einen Base64-String zur einfachen Speicherung in JSON.
    String base64String = Base64.getEncoder().encodeToString(fileContent);
    // Speichert den Base64-String im Question-Objekt.
    questionToUpdate.setMusterloesungImageBase64(base64String);
}
```

Abbildung 5.5: Speichern der textuellen und bildlichen Musterlösung. Der Code zeigt, wie sowohl Text- als auch Bilddaten für die Musterlösung im ‘Question‘-Objekt persistiert werden.

5.2.4 FA4 Verwaltung von Prüfungsmetadaten (siehe Anforderung 3.1.6)

Die Prüfungsmetadaten (Titel, Hochschule, Fachbereich, Modul, Semester) werden über entsprechende ‘TextField‘-Komponenten in der ‘MainView.fxml‘ erfasst und im ‘Exam‘-Objekt gespeichert. Die Methode ‘updateExamMetadata()‘ im ‘MainController‘ synchronisiert die UI-Eingaben mit dem Datenmodell.

5.2.5 FA5 Rich-Text-Formatierung für Fragen (siehe Anforderung 3.1.7)

Die Rich-Text-Formatierung für Fragentexte wird durch die Integration des ‘HTMLEditor‘-Komponente von JavaFX in der ‘MainView.fxml‘ realisiert. Der ‘HTMLEditor‘ ermöglicht es dem Benutzer, Texte mit grundlegenden Stiloptionen wie Fett, Kursiv, Unterstrichen, Listen und Farben zu formatieren. Die Inhalte werden intern als HTML-Strings im ‘text‘-Attribut des ‘Question‘-Objekts gespeichert. Abbildung 5.6 zeigt die deklarative Einbindung der Komponente in der FXML-Datei. Der ‘fx:id="questionTextField„ verknüpft die FXML-Komponente mit dem entsprechenden Feld im ‘MainController‘, wodurch ein programmatischer Zugriff auf den HTML-Inhalt ermöglicht wird. Beim Export in das Word-Dokument werden diese HTML-Strings durch den ‘WordExporter‘ interpretiert und in entsprechende WordML-Formatierungen umgewandelt (siehe Abschnitt 5.2.13).

5.2.5.1 Formatierung von Code-Blöcken

Zusätzlich zu den Standardformatierungen wurde eine Funktion zur expliziten Kennzeichnung von Code-Fragmenten implementiert, um deren Lesbarkeit in der exportierten Prüfung zu verbessern.

In der FXML-Datei `MainView.fxml` wurde ein "CodeButton zur UI hinzugefügt. Im `MainController` wurde dieser Button mit einer Aktion verknüpft, die den im `HTMLEditor` markierten Text mithilfe eines JavaScript-Befehls in `<code>`-Tags einschließt.

Der `WordExporter` wurde erweitert, um diese Tags zu verarbeiten. Wenn ein `<code>`- oder `<pre>`-Tag erkannt wird, wird die Schriftart für diesen Textabschnitt auf `Courier New` gesetzt. Für `<pre>`-Blöcke wird zusätzlich der gesamte Absatz mit einem hellgrauen Hintergrund versehen, um den Code-Block

visuell hervorzuheben. Dies wird durch eine neue Hilfsmethode `setParagraphShading` realisiert, die direkt auf das OOXML-Schema von Apache POI zugreift.

```
<!-- In MainView.fxml -->
<VBox fx:id="editPane" spacing="10">
    <!-- ... andere UI-Komponenten ... -->

    <!-- Deklarative Einbindung des HTMLEditor -->
    <!-- Die fx:id ermöglicht die Verknüpfung mit dem Controller -->
    <HTMLEditor fx:id="questionTextField" prefHeight="200.0" />

    <!-- ... andere UI-Komponenten ... -->
</VBox>
```

Abbildung 5.6: Deklarative Einbindung der ‘HTMLEditor‘-Komponente in der ‘MainView.fxml‘. Diese Komponente stellt die grafische Benutzeroberfläche für die Textformatierung bereit.

5.2.6 FA6 Integration von Bildern in Fragen (siehe Anforderung 3.1.8)

Die Integration von Bildern in Fragen wird durch die ‘addImage()‘-Methode im ‘MainController‘ ermöglicht. Wie in Abbildung 5.7 zu sehen ist, öffnet diese Methode einen ‘FileChooser‘, damit der Benutzer eine Bilddatei auswählen kann. Wenn eine Datei ausgewählt wird, wird ihr Inhalt in ein Byte-Array gelesen und anschließend mit der ‘Base64‘-Klasse von Apache Commons Codec in einen Base64-String kodiert. Dieser String wird dann im ‘imageBase64‘-Attribut des aktuell ausgewählten ‘Question‘-Objekts gespeichert. Gleichzeitig wird das Bild zur sofortigen visuellen Rückmeldung in einer ‘ImageView‘-Komponente in der Benutzeroberfläche angezeigt. Dieser Ansatz, das Bild als Base64-String direkt im JSON-Dokument zu speichern, stellt sicher, dass die gesamte Prüfung in einer einzigen, portablen Datei gekapselt ist.

```java
@FXML
private void addImage() {
    // Öffnet einen Dialog zur Auswahl einer Bilddatei.
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Bild auswählen");
    File selectedFile = fileChooser.showOpenDialog(mainPane.getScene().getWindow());

    if (selectedFile != null) {
        try {
            // Liest die Datei als Byte-Array.
            byte[] fileContent = Files.readAllBytes(selectedFile.toPath());
            // Kodiert die Bytes in einen Base64-String.
            String base64String = Base64.getEncoder().encodeToString(fileContent);

            // Speichert den String im aktuell ausgewählten Question-Objekt.
            TreeItem<Question> selectedItem = questionsTable.getSelectionModel().getSelectedItem();
            if (selectedItem != null) {
                selectedItem.getValue().setImageBase64(base64String);
            }
            // Zeigt das Bild in der Benutzeroberfläche an.
            questionImageView.setImage(new Image(new ByteArrayInputStream(fileContent)));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Abbildung 5.7: Implementierung des Bild-Uploads. Der Code zeigt das Einlesen, die Base64-Kodierung und die Speicherung der Bilddaten im Datenmodell sowie die Anzeige in der UI.

5.2.7 FA7 Konfiguration von Prüfungshinweisen (siehe Anforderung 3.1.9)

Die Konfiguration der Prüfungshinweise, Hilfsmittel und Bearbeitungszeit wird über einen separaten modalen Dialog (‘HinweiseDialog.fxml‘, ‘HinweiseDialogController‘) realisiert. Der ‘HinweiseDialogController‘ sammelt die Benutzereingaben (allgemeine Hinweise, ausgewählte Checkboxen für Hilfsmittel, Bearbeitungszeit) und aktualisiert das ‘Exam‘-Objekt. Eine Live-Vorschau im Dialog zeigt dem Benutzer das finale Erscheinungsbild der Hinweise. Die Kommunikation zwischen dem Hauptfenster und dem Dialog wird durch die Übergabe des zentralen ‘Exam‘-Objekts realisiert, wie in Abbildung 5.8 dargestellt. Die Methode ‘openHinweiseDialog‘ im ‘MainController‘ ruft vor dem Anzeigen des Dialogs die ‘setData‘-Methode des ‘HinweiseDialogController‘ auf und übergibt die aktuelle ‘Exam‘-Instanz. Dadurch kann der Dialog die vorhandenen Daten anzeigen und bei Änderungen direkt im zentralen Datenmodell aktualisieren.

```java
// Im MainController: Methode zum Öffnen des Dialogs
@FXML
private void openHinweiseDialog() {
    if (hinweiseDialogStage != null) {
        // Übergibt das zentrale Exam-Objekt an den Dialog-Controller.
        hinweiseDialogController.setData(this.exam);
        // Zeigt den Dialog und wartet, bis er geschlossen wird.
        hinweiseDialogStage.showAndWait();
    }
}

// Im HinweiseDialogController: Methode zum Empfangen der Daten
public void setData(Exam exam) {
    this.exam = exam;
    // Befüllt die UI-Elemente des Dialogs mit den Daten aus dem Exam-Objekt.
    // z.B. hinweiseTextArea.setText(exam.getAllgemeineHinweise());
}
```

Abbildung 5.8: Datenübergabe an den modalen Dialog. Der ‘MainController‘ übergibt das ‘Exam‘-Objekt an den ‘HinweiseDialogController‘, um die Konsistenz der Daten zu gewährleisten.

5.2.8 FA8 Manuelle Layout-Steuerung

Die manuelle Layout-Steuerung im Word-Export wird durch spezifische Attribute in der ‘Question‘-Klasse (‘startOnNewPage‘, ‘justify‘) und deren Verarbeitung im ‘WordExporter‘ implementiert.

5.2.8.1 FA8.1 Seitenumbruch-Steuerung (siehe Anforderung 3.1.10.1)

Die ‘startOnNewPage‘-Eigenschaft einer Frage, die über eine Checkbox in der ‘TreeTableView‘ gesetzt werden kann, wird vom ‘WordExporter‘ interpretiert. Ist diese Eigenschaft für eine Hauptfrage aktiviert, wird vor dieser ein expliziter Seitenumbruch eingefügt. Für Unterfragen, die auf einer neuen Seite beginnen sollen, wird zusätzlich eine Fortsetzungsmeldung auf der vorherigen Seite platziert, um den Lesefluss zu gewährleisten. Abbildung 5.9 illustriert diesen Prozess innerhalb der rekursiven ‘writeQuestion‘-Methode. Bevor eine Unterfrage geschrieben wird, die auf einer neuen Seite beginnen soll, wird ein Hinweistext am Ende der aktuellen Seite eingefügt und anschließend ein harter Seitenumbruch (‘setPageBreak(true)‘) erzwungen.

```
// Innerhalb der rekursiven Methode writeQuestion()
if (subQuestion.isStartOnNewPage()) {
    // Fügt einen Hinweistext am Ende der aktuellen Seite ein.
    XWPFParagraph continueMessage = document.createParagraph();
    continueMessage.setAlignment(ParagraphAlignment.RIGHT);
    XWPFRun continueRun = continueMessage.createRun();
    continueRun.setText("Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.");
    continueRun.setItalic(true);

    // Erzwingt einen expliziten Seitenumbruch.
    document.createParagraph().setPageBreak(true);
}
// Danach wird die Sub-Frage auf die neue Seite geschrieben.
writeQuestion(document, subQuestion, ...);
```

Abbildung 5.9: Implementierung des manuellen Seitenumbruchs für Unterfragen. Es wird ein Hinweistext eingefügt und anschließend ein Seitenumbruch erzwungen.

5.2.8.2 FA8.2 Selektive Text-Justierung (siehe Anforderung 3.1.10.2)

Die ‘justify‘-Eigenschaft, ebenfalls über eine Checkbox in der ‘TreeTableView‘ steuerbar, beeinflusst die Ausrichtung des Fragentitels im Word-Dokument. Ist sie aktiviert, wird der entsprechende Absatz im ‘WordExporter‘ mit Blocksatz formatiert. Wie in Abbildung 5.10 gezeigt, wird vor dem Schreiben des Fragentitels die ‘isJustify()‘-Eigenschaft des ‘Question‘-Objekts geprüft. Wenn diese ‘true‘ ist, wird die Ausrichtung des ‘XWPFParagraph‘-Objekts auf ‘ParagraphAlignment.BOTH‘ gesetzt, was im Word-Dokument zu Blocksatz führt.

```
// Innerhalb der Methode writeQuestion()
XWPFParagraph questionTitle = document.createParagraph();

// Prüft, ob die Blocksatz-Option für diese Frage aktiviert ist.
if (question.isJustify()) {
    // Setzt die Absatzausrichtung auf Blocksatz.
    questionTitle.setAlignment(ParagraphAlignment.BOTH);
}
// ... weiterer Code zum Schreiben des Titels
```

Abbildung 5.10: Anwenden von Blocksatz auf den Fragentitel. Die ‘isJustify‘-Eigenschaft der Frage steuert die Absatzausrichtung im exportierten Dokument.

5.2.9 FA9 Generierung von Prüfungsvarianten (siehe Anforderung 3.1.11)

Die Generierung von Prüfungsvarianten wird durch die Methode `exportVariedVersion()` im `MainController` initiiert. Diese Funktion kombiniert zwei Haupttechniken: die automatische Text-Reformulierung und das Mischen von Unterfragen. Der Gesamtprozess der Variantenerzeugung ist in Abbildung 5.11 dargestellt.

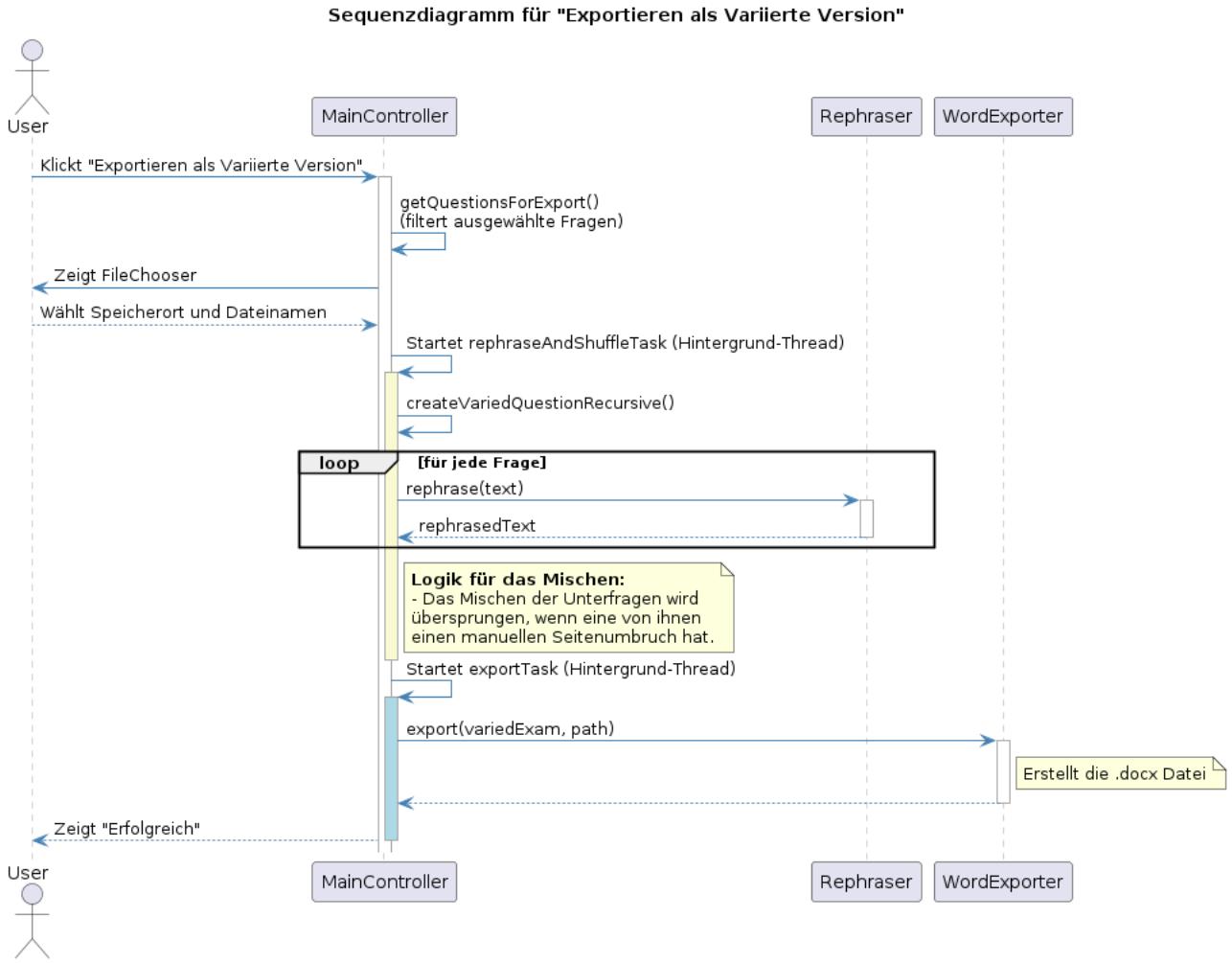


Abbildung 5.11: Sequenzdiagramm für den Export von Prüfungsvarianten. Es illustriert den Prozess der Text-Reformulierung sowie des Mischens von Unterfragen.

5.2.9.1 Text-Reformulierung mit dem Rephraser

Die Logik zur Abwandlung von Texten ist in der Utility-Klasse ‘Rephraser‘ gekapselt. Sie lädt einen Thesaurus (‘openthesaurus.txt‘) und ersetzt eine begrenzte Anzahl von Wörtern in den Frage-Titeln und -Texten durch Synonyme, wobei die Groß- und Kleinschreibung beibehalten wird.

5.2.9.2 Mischen der Fragenreihenfolge

Die rekursive Methode ‘createVariedQuestionRecursive()‘ im ‘MainController‘ mischt die Reihenfolge der Unterfragen. Dieser Mischvorgang wird jedoch übersprungen, falls eine der Unterfragen eine manuelle Seitenumbruch-Markierung aufweist, um die vom Autor beabsichtigte Layout-Struktur zu erhalten. Der gesamte Prozess wird asynchron in einem Hintergrund-Task ausgeführt, um die UI-Reaktionsfähigkeit zu gewährleisten.

5.2.10 FA10 Selektiver Export von Fragen (siehe Anforderung 3.1.12)

Der selektive Export von Fragen wird durch einen modalen Workflow ermöglicht, der über den ‘selectionModeButton‘ im ‘MainController‘ aktiviert wird. Im Auswahlmodus wird die ‘selectedColumn‘ in der ‘TreeTableView‘ editierbar, sodass der Benutzer über Checkboxen festlegen kann, welche Fragen exportiert werden sollen. Vor dem eigentlichen Export verarbeitet die Methode ‘filterSelected()‘ diese Auswahl. Wie in Abbildung 5.12 gezeigt, durchläuft diese Methode rekursiv die Liste der Fragen und ihrer Unterfragen. Für jede Frage wird geprüft, ob die ‘isSelected‘-Eigenschaft ‘true‘ ist. Wenn ja, wird eine tiefe Kopie der Frage erstellt und der gefilterten Liste hinzugefügt. Dieser rekursive Ansatz stellt sicher, dass die hierarchische Struktur auch in der gefilterten Auswahl korrekt erhalten bleibt.

```
1  private List<Question> filterSelected(List<Question> questions) {
2      List<Question> selected = new ArrayList<>();
3      // Iteriert durch die übergebene Liste von Fragen.
4      for (Question q : questions) {
5          // Prüft, ob die Frage ausgewählt wurde.
6          if (q.isSelected()) {
7              // Erstellt eine tiefe Kopie, um das Original nicht zu verändern.
8              Question copy = new Question(q);
9              // Rekursiver Aufruf für die Unterfragen.
10             if (q.getSubQuestions() != null && !q.getSubQuestions().isEmpty()) {
11                 copy.setSubQuestions(filterSelected(q.getSubQuestions()));
12             }
13             // Fügt die kopierte und gefilterte Frage zur Ergebnisliste hinzu.
14             selected.add(copy);
15         }
16     }
17     return selected;
18 }
```

Abbildung 5.12: Rekursives Filtern der für den Export ausgewählten Fragen. Die Methode erstellt eine neue Liste, die nur die ausgewählten Fragen und deren ausgewählte Unterfragen enthält.

5.2.11 FA11 Export als Word-Dokument (siehe Anforderung 3.1.13)

Die primäre Exportfunktion ist in der Service-Klasse ‘WordExporter‘ gekapselt. Die Methode ‘export()‘ generiert ein professionell formatiertes Microsoft Word-Dokument (.docx). Dies umfasst die Erstellung eines standardisierten Deckblatts (‘createCoverPage()‘), die Darstellung der Fragen (‘writeQuestion()‘) und die Seitennummerierung (‘createPageNumbering()‘). Wie in Abbildung 5.13 gezeigt, wird der Exportprozess in der Methode ‘exportToWord()‘ des ‘MainController‘ angestoßen. Zuerst werden die Metadaten aktualisiert und die zu exportierenden Fragen gefiltert. Anschließend wird ein ‘FileChooser‘ zur Auswahl des Speicherorts geöffnet. Der eigentliche Export, der potenziell zeitaufwendig ist, wird in einen ‘javafx.concurrent.Task‘ gekapselt und in einem separaten Thread ausgeführt. Dies verhindert das Einfrieren der Benutzeroberfläche und gewährleistet die UI-Reaktionsfähigkeit.

```
@FXML
private void exportToWord() {
    // Stellt sicher, dass alle Metadaten auf dem neuesten Stand sind.
    updateExamMetadata();
    // Filtert die Fragen basierend auf der Benutzerauswahl.
    List<Question> questionsToExport = getQuestionsForExport();
    if (questionsToExport == null) return;

    // Erstellt ein temporäres Exam-Objekt für den Export.
    Exam examToExport = new Exam(exam);
    examToExport.setQuestions(questionsToExport);

    // Öffnet einen Dialog zum Speichern der Datei.
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Exam as Word Document");
    File file = fileChooser.showSaveDialog(mainPane.getScene().getWindow());
    if (file != null) {
        // Kapselt den langwierigen Exportvorgang in einem Task.
        Task<Void> exportTask = new Task<>() {
            @Override
            protected Void call() throws Exception {
                // Ruft die statische Exportmethode der WordExporter-Klasse auf.
                WordExporter.export(examToExport, file.getAbsolutePath());
                return null;
            }
        };
        // Startet den Task in einem neuen Thread.
        new Thread(exportTask).start();
    }
}
```

Abbildung 5.13: Starten des asynchronen Word-Export-Prozesses. Der Export wird in einen Hintergrund-Thread ausgelagert, um die Benutzeroberfläche reaktionsfähig zu halten.

5.2.12 FA12 Generierung eines separaten Lösungsblatts (siehe Anforderung 3.1.14)

Die Generierung eines separaten Lösungsblatts erfolgt ebenfalls über den ‘WordExporter‘. Die Methode ‘exportAnswerKey()‘ im ‘MainController‘, dargestellt in Abbildung 5.14, orchestriert diesen

Prozess. Ähnlich wie beim normalen Export werden die Daten vorbereitet und der Prozess in einem Hintergrund-Thread ausgeführt. Der entscheidende Unterschied ist der Aufruf der Methode ‘WordExporter.exportWithSolutions()‘. Diese Methode ruft intern dieselbe Logik wie der normale Export auf, setzt jedoch den Parameter ‘withSolutions‘ auf ‘true‘. Dieser boolesche Wert dient als Schalter innerhalb der ‘WordExporter‘-Klasse, um anstelle von leeren Antwortzeilen die hinterlegten Musterlösungen in das Dokument einzufügen.

```
1  @FXML
2  private void exportAnswerKey() {
3      // Vorbereitung der Daten, identisch zum normalen Export.
4      updateExamMetadata();
5      List<Question> questionsToExport = getQuestionsForExport();
6      if (questionsToExport == null) return;
7
8      Exam examToExport = new Exam(exam);
9      examToExport.setQuestions(questionsToExport);
10
11     // Öffnet einen Dialog zum Speichern der Lösungsdatei.
12     FileChooser fileChooser = new FileChooser();
13     fileChooser.setTitle("Save Answer Key as Word Document");
14     File file = fileChooser.showSaveDialog(mainPane.getScene().getWindow());
15     if (file != null) {
16         // Kapselung in einem Task für die asynchrone Ausführung.
17         Task<Void> exportTask = new Task<>() {
18             @Override
19             protected Void call() throws Exception {
20                 // Aufruf der spezifischen Methode für den Export mit Lösungen.
21                 WordExporter.exportWithSolutions(examToExport, file.getAbsolutePath());
22                 return null;
23             }
24         };
25         new Thread(exportTask).start();
26     }
27 }
```

Abbildung 5.14: Starten des asynchronen Exports des Lösungsblatts. Der Prozess ist fast identisch zum normalen Export, ruft aber eine andere Methode im ‘WordExporter‘ auf.

Bei der Entwicklung des ExamBuilders traten mehrere technische Herausforderungen auf, für die spezifische Lösungsstrategien entwickelt werden mussten.

5.2.13 Konvertierung von HTML nach WordML

Eine der größten Herausforderungen war die Übertragung der im ‘HTMLEditor‘ erstellten Textformatierungen in das Word-Dokument. Der Editor liefert den Inhalt als HTML-String, während Apache POI ein eigenes, XML-basiertes Format (WordML) erwartet. Eine direkte Konvertierung wird von der Bibliothek nicht unterstützt.

Lösung: Anstatt auf externe und potenziell schwergewichtige Konvertierungsbibliotheken zurückzugreifen, wurde eine leichtgewichtige, manuelle Konvertierung implementiert. Unter Verwendung der Jsoup-Bibliothek (siehe Abschnitt 2.3.4) wird der HTML-String in eine DOM-Struktur geparst.

Anschließend wird dieser DOM-Baum rekursiv durchlaufen. Für jeden HTML-Tag (z.B. ‘<b>‘, ‘<i>‘, ‘<p>‘) wird ein entsprechendes, formatiertes ‘XWPFRun‘- oder ‘XWPFParagraph‘-Objekt in Apache POI erzeugt. Dieser Ansatz bietet eine granulare Kontrolle über die Umwandlung und hält die Anzahl der externen Abhängigkeiten gering.

5.2.14 Sicherstellung der UI-Reaktionsfähigkeit

Operationen wie der Word-Export oder die Generierung einer Prüfungsvariante können je nach Umfang der Prüfung mehrere Sekunden in Anspruch nehmen. Würden diese Prozesse auf dem JavaFX Application Thread ausgeführt, würde die gesamte Benutzeroberfläche für die Dauer der Operation einfrieren, was zu einer schlechten Benutzererfahrung führt.

Lösung: Um die Reaktionsfähigkeit der UI zu gewährleisten, wurde die JavaFX Concurrency API eingesetzt. Jede langlaufende Operation wird in ein ‘javafx.concurrent.Task‘-Objekt[2] gekapselt und auf einem separaten Hintergrund-Thread ausgeführt. Um dem Benutzer währenddessen Feedback zu geben, wird ein modaler ‘LoadingIndicator‘ angezeigt, der den Fortschritt signalisiert und die UI für weitere Eingaben sperrt. Sobald der Task abgeschlossen ist, wird der Indikator wieder ausgeblendet.

5.2.15 Zustandsverwaltung der hierarchischen Daten

Die Verwaltung der hierarchischen Fragenstruktur in der ‘TreeTableView‘ stellte eine weitere Herausforderung dar. Das Hinzufügen, Löschen oder Aktualisieren von verschachtelten Elementen kann schnell zu Inkonsistenzen zwischen dem Datenmodell und der angezeigten Ansicht führen, wenn versucht wird, die UI-Komponente direkt zu manipulieren.

Lösung: Es wurde das Prinzip einer ***Single Source of Truth***[3] verfolgt. Anstatt einzelne UI-Elemente zu modifizieren, wird bei jeder Änderung direkt das zentrale Datenmodell (das ‘Exam‘-Objekt mit seinen Listen) aktualisiert. Unmittelbar danach wird die Methode ‘refreshTreeTableView()‘ aufgerufen. Diese Methode löscht den gesamten Inhalt der Tabelle und baut sie aus dem nun aktuellen Datenmodell vollständig neu auf. Dieser Ansatz ist zwar auf den ersten Blick weniger performant, erweist sich in der Praxis jedoch als wesentlich robuster und weniger fehleranfällig, da die Konsistenz zwischen Daten und Darstellung jederzeit gewährleistet ist.

5.2.16 Steuerung von Seitenumbrüchen im Word-Export

Eine der zentralen Anforderungen an die Benutzerfreundlichkeit war die Gewährleistung eines sauberen und professionellen Layouts in der exportierten Word-Datei. Ein Problem der initialen Implementierung war, dass Aufgabenblöcke an ungünstigen Stellen von automatischen Seitenumbrüchen zerschnitten wurden, was die Lesbarkeit der Prüfung beeinträchtigte.

Herausforderung und erster Lösungsansatz: Die naheliegende Anforderung war, einen Hinweis wie „Die Aufgabe wird auf der nächsten Seite fortgesetzt“ einzufügen, wenn ein solcher Umbruch auftritt. Wie bereits im Abschnitt über technische Herausforderungen diskutiert, ist dies mit der verwendeten Bibliothek Apache POI nicht zuverlässig umsetzbar, da diese keine Kenntnis über das finale Rendering des Dokuments hat.

Ein alternativer, technisch eleganter Ansatz wurde daraufhin implementiert: Jeder Aufgabenblock wurde in eine unsichtbare, einzeilige Tabelle gekapselt, für die die Word-Eigenschaft „Zeilenumbruch auf der Seite nicht zulassen“ (‘cantSplit‘) aktiviert wurde. Die Hypothese war, dass Word einen

[2] Ein ‘javafx.concurrent.Task‘ ist eine Klasse aus der JavaFX-Bibliothek, die es ermöglicht, langwierige Operationen in einem separaten Hintergrund-Thread auszuführen, um die Benutzeroberfläche (UI) reaktionsfähig zu halten. [13]

[3] Das *Single Source of Truth* (SSoT)-Prinzip ist ein Konzept in der Softwareentwicklung, bei dem sichergestellt wird, dass jede Information an genau einem autoritativen Ort gespeichert und von dort aus referenziert wird, um Inkonsistenzen zu vermeiden. [14]

Aufgabenblock, der nicht mehr vollständig auf eine Seite passt, automatisch auf die nächste Seite verschieben würde. In der Praxis führte dieser Ansatz jedoch zu neuen Layout-Problemen. Insbesondere bei sehr langen Aufgabenblöcken, die eine ganze Seite überschreiten, führte diese Einstellung zu unerwünschtem Verhalten und teilweise zu großen, leeren Flächen im Dokument, was ebenfalls die Layout-Anforderungen verletzte.

Finale Lösungsentscheidung: Angesichts der unvorhersehbaren Ergebnisse des ‘cantSplit‘-Ansatzes wurde eine Rückkehr zu einer einfacheren, aber robusteren und direkteren Methode beschlossen. Die finale Lösung kombiniert zwei gezielte Anweisungen an Word:

1. Ein expliziter Seitenumbruch (‘setPageBreak(true)‘) wird nach jedem Hauptaufgabenblock eingefügt. Dies stellt eine klare und verlässliche Trennung zwischen den einzelnen Übungen sicher.
2. Für die Absätze, die den Titel einer Aufgabe enthalten, werden die Eigenschaften ‘keepLines‘ und ‘keepNext‘ gesetzt. Die ‘keepLines‘-Eigenschaft weist Word an, den Titelabsatz nicht selbst durch einen Seitenumbruch zu zerteilen. Die ‘keepNext‘-Eigenschaft sorgt dafür, dass der Titelabsatz immer auf derselben Seite wie der unmittelbar folgende Absatz (der Aufgabeninhalt) bleibt.

Diese Kombination verhindert die häufigsten und störendsten Layout-Fehler – verwaiste Titel am Ende einer Seite und das Zerschneiden von Titeln – ohne die Komplexität und die unvorhersehbaren Nebeneffekte der Tabellen-basierten Lösung einzuführen. Dieser pragmatische Ansatz erwies sich als der zuverlässigsten Weg, um die Anforderung nach einem stimmigen Layout zu erfüllen.

5.3 Qualitätssicherung und Testen

Um die Stabilität und korrekte Funktionsweise der Anwendung sicherzustellen, wurde eine zweistufige Teststrategie verfolgt, die aus automatisierten Unit-Tests für die Backend-Logik und manuellen Tests für die Benutzeroberfläche bestand.

5.3.1 Unit-Tests mit JUnit 5

Für die Kernlogik der Anwendung wurden Unit-Tests mit dem Framework *JUnit 5* geschrieben.[15] Der Fokus lag dabei auf der Validierung von Klassen, deren Verhalten isoliert und ohne eine laufende grafische Oberfläche überprüft werden konnte. Beispielsweise wurden für die Klasse ‘Rephraser‘ Tests entwickelt, um sicherzustellen, dass der Algorithmus zur Synonymersetzung korrekt funktioniert. Ebenso wurde die Geschäftslogik der Datenmodelle, wie die rekursive Punkteberechnung in der ‘Question‘-Klasse, durch Unit-Tests abgesichert. Dieser Ansatz ermöglichte es, Fehler in der Kernlogik frühzeitig im Entwicklungsprozess zu identifizieren und zu beheben. Ein exemplarisches Beispiel für einen solchen Testfall ist in Abbildung 5.15 aufgeführt.

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class RephraserTest {

    @Test
    void testRephraseSentenceWithKnownSynonym() {
        // Annahme: Das Wort "wichtig" ist im Thesaurus und hat Synonyme
        String originalSatz = "Dies ist ein wichtiger Satz.";

        // Die rephrase-Methode wird aufgerufen
        String rephrasedSatz = Rephraser.rephrase(originalSatz);

        // Der Test prüft, ob der Satz verändert wurde,
        // da ein Austausch stattgefunden haben sollte.
        assertNotEquals(originalSatz, rephrasedSatz,
            "Der Satz sollte umformuliert worden sein.");

        // Man könnte weiter prüfen, ob die Grundstruktur erhalten blieb
        assertTrue(rephrasedSatz.startsWith("Dies ist ein"));
        assertTrue(rephrasedSatz.endsWith(" Satz."));
    }
}
```

Abbildung 5.15: Beispiel eines Unit-Tests für die Rephraser-Klasse mit JUnit 5.

5.3.2 Manuelle UI-Tests

Die Tests der grafischen Benutzeroberfläche und des Zusammenspiels der Komponenten erfolgten manuell. Aufgrund der hohen Komplexität und des Aufwands, der mit der Einrichtung von automatisierten UI-Test-Frameworks für JavaFX verbunden ist, wurde dieser pragmatische Ansatz gewählt. Die manuellen Tests folgten einem systematischen Vorgehen, bei dem alle in der Anforderungsanalyse definierten Anwendungsfälle durchgespielt wurden. Dies umfasste den gesamten Workflow von der Erstellung einer neuen Prüfung, dem Hinzufügen und Bearbeiten von Fragen, dem Speichern und Laden des Projekts bis hin zum finalen Export der Word-Dokumente. Jedes exportierte Dokument wurde anschließend auf korrekte Formatierung und Vollständigkeit überprüft.

6 Evaluation

In diesem Kapitel wird die Evaluation des entwickelten ExamBuilder-Prototyps beschrieben. Ziel der Evaluation ist es, die erstellte Software anhand der zuvor definierten Anforderungen zu überprüfen und ihre Qualität kritisch zu bewerten.

6.1 Testkonzept und Durchführung

Um eine umfassende Qualitätssicherung zu gewährleisten, wurde eine zweistufige Teststrategie angewendet, die sich aus automatisierten Unit-Tests für die isolierte Geschäftslogik und manuellen, szenariobasierten Systemtests für die Gesamtfunktionalität zusammensetzt.

6.1.1 Unit-Tests

Die Kernlogik der Anwendung wurde durch Unit-Tests mit dem Framework *JUnit 5* abgesichert. Diese Tests dienten dazu, die korrekte Funktionsweise einzelner Methoden und Klassen unabhängig von der grafischen Benutzeroberfläche zu verifizieren. Schwerpunkte der Unit-Tests waren unter anderem:

- Die Validierung der ‘Rephraser‘-Klasse, um die korrekte Anwendung von Synonymen sicherzustellen.
- Die Überprüfung der Geschäftslogik im Datenmodell, insbesondere die rekursive Berechnung der Gesamtpunktzahl einer Frage mit Unterfragen in der ‘Question‘-Klasse.

6.1.2 Manuelle Systemtests

Die Überprüfung des Gesamtsystems aus Benutzersicht erfolgte durch manuelle Tests. Hierfür wurden konkrete Anwenderszenarien definiert, die den gesamten Lebenszyklus der Prüfungs-erstellung abdecken. Die wichtigsten Testszenarien waren:

Szenario A: Neuerstellung einer Prüfung
: Dieser Testfall umfasste die Erstellung einer kompletten Prüfung von Grund auf, basierend auf dem in Abschnitt 2.5 eingeführten und im Anhang (siehe Anhang 7.4) detailliert ausgeführten Zielszenario. Der Prozess beinhaltete die Eingabe der Metadaten, die Erstellung der definierten Haupt- und Unterfragen (inklusive der Frage mit Bild), die Konfiguration der Prüfungshinweise und den abschließenden Export als .docx-Datei sowie als separates Lösungsblatt.

Szenario B: Laden und Bearbeiten
: Es wurde geprüft, ob ein zuvor gespeichertes Prüfungsprojekt im JSON-Format fehlerfrei geladen werden kann. Anschließend wurden Änderungen vorgenommen (z.B. eine Frage bearbeitet, eine andere gelöscht) und das Projekt erneut gespeichert, um die Persistenz der Änderungen zu verifizieren.

Szenario C: Evaluierung der Export-Varianten
: In diesem Szenario wurden die verschiedenen Exportfunktionen gezielt getestet. Dies umfasste den selektiven Export von nur ausgewählten Fragen sowie die Generierung einer variierten Prüfungsversion, bei der die korrekte Anwendung der Reformulierung und der Mischung von Unterfragen im Zieldokument überprüft wurde.

Zusätzlich zu den definierten Szenarien wurde der ExamBuilder einem Praxistest unterzogen, bei dem mehrere Altklausuren von verschiedenen Professoren der Technischen Hochschule Mittelhessen (THM) importiert und verarbeitet wurden. Das Ziel war es, die Anwendung mit realen, oft komplex und uneinheitlich formatierten Daten zu konfrontieren. Die Ergebnisse waren äußerst positiv: Die Anwendung konnte die Altklausuren erfolgreich importieren und in das strukturierte Format des ExamBuilders überführen. Obwohl in einigen Fällen kleinere manuelle Anpassungen an der Formatierung notwendig waren, war der Zeitgewinn im Vergleich zur vollständig manuellen Neuerstellung erheblich. Insbesondere die automatische Neunummerierung, die konsistente Anwendung von Layout-Vorlagen und die einfache Generierung des Lösungsblatts wurden als wesentliche Vorteile identifiziert, die den Arbeitsaufwand drastisch reduzieren.

Während dieser Tests wurden auch kleinere Limitationen des Word-Exports identifiziert. So wird zwar die Struktur von Code-Blöcken durch eine Monospace-Schriftart und einen schattierten Hintergrund korrekt dargestellt, die farbliche Syntaxhervorhebung aus dem Editor wird jedoch nicht in das Word-Dokument übertragen. Des Weiteren werden Bilder automatisch skaliert, um auf die Seite zu passen, was gelegentlich zu einer Größe führt, die für den optimalen Lesefluss eine schnelle, manuelle Anpassung im Zieldokument erfordert. Dieser manuelle Schritt dauert jedoch nur wenige Sekunden und wurde als unerheblich für den Gesamtnutzen bewertet.

6.2 Abgleich der Ergebnisse mit den definierten Anforderungen

Die abschließende Verifizierung des Prototyps erfolgte durch einen systematischen Abgleich der implementierten Funktionalitäten mit dem in Kapitel 3 definierten Anforderungskatalog. Die Ergebnisse bestätigen, dass der Prototyp die Anforderungen in vollem Umfang erfüllt. Die Resultate für die funktionalen und nicht-funktionalen Anforderungen werden in den folgenden Tabellen separat dargestellt.

Nicht-funktionale Anforderung	**Status**	**Bemerkung**
Benutzerfreundlichkeit	Größtenteils erfüllt	Informelle Tests zeigten eine hohe intuitive Bedienbarkeit. Das Konzept des separaten Auswahlmodus wurde als positiv bewertet.
Wartbarkeit	Erfüllt	Die konsequente Anwendung des MVC-Musters und die saubere Code-Struktur gewährleisten eine gute Wartbarkeit.
Plattformunabhängigkeit	Erfüllt	Durch die Wahl von Java und JavaFX ist die Anwendung auf allen gängigen Desktop-Betriebssystemen lauffähig.

Tabelle 6.1: Abgleich der nicht-funktionalen Anforderungen.

Funktionale Anforderung	**Status**	**Bemerkung / Validierung**
Projektverwaltung (Speichern/Laden)	Vollständig erfüllt	Validiert durch Testszenario B. Die Serialisierung via JSON funktioniert zuverlässig.
Fragenverwaltung (CRUD)	Vollständig erfüllt	Validiert durch Testszenario A und B. Alle Operationen sind implementiert.
Hierarchische Unterfragen	Vollständig erfüllt	Die rekursive Struktur des Datenmodells und die Darstellung im 'TreeTableView' setzen dies um.
Metadaten-Verwaltung	Vollständig erfüllt	Alle Felder sind im UI vorhanden und werden korrekt gespeichert und exportiert.
Rich-Text-Formatierung	Vollständig erfüllt	Der 'HTMLEditor' und die Konvertierungslogik im 'WordExporter' erfüllen die Anforderung.
Bildintegration	Vollständig erfüllt	Bilder können hinzugefügt, gespeichert und exportiert werden.
Konfiguration der Hinweise	Vollständig erfüllt	Der dedizierte Dialog erfüllt alle Aspekte dieser Anforderung.
Manuelle Seitenumbruch-Steuerung	Vollständig erfüllt	Die Funktion wurde implementiert und ermöglicht eine präzise Layout-Kontrolle.
Selektive Text-Justierung	Vollständig erfüllt	Die Blocksatz-Checkbox für Fragentitel wurde erfolgreich umgesetzt.
Prüfungsvariation	Vollständig erfüllt	Reformulierung und bedingtes Mischen funktionieren wie spezifiziert.
Selektiver Export	Vollständig erfüllt	Der Auswahlmodus und die Filterlogik wurden erfolgreich getestet.
Word-Export (Prüfung & Lsg.)	Vollständig erfüllt	Alle Exportszenarien erzeugten korrekt formatierte .docx-Dokumente.

Tabelle 6.2: Abgleich der funktionalen Anforderungen.

Die Evaluation bestätigt, dass alle definierten funktionalen Kernanforderungen im Prototyp erfolgreich umgesetzt wurden. Die nicht-funktionalen Anforderungen wurden ebenfalls erfüllt, wobei die Benutzerfreundlichkeit als subjektives Kriterium stets weiteres Verbesserungspotenzial bietet.

Zusammenfassend lässt sich festhalten, dass der ExamBuilder ein erfolgreicher Prototyp ist, der die Machbarkeit und den Nutzen des Konzepts eindrucksvoll belegt. Er bietet eine solide Grundlage für ein potenzielles Open-Source-Projekt, das den Arbeitsalltag von Lehrenden erheblich erleichtern könnte. Abschließend werden die Ergebnisse des Projekts kritisch gewürdigt, um die Stärken und Limitationen der entwickelten Lösung zu diskutieren und einen Ausblick auf mögliche Weiterentwicklungen zu geben.

6.2.1 Stärken des Projekts

Der entwickelte Prototyp des ExamBuilders stellt eine erfolgreiche Umsetzung der initialen Zielsetzung dar. Die wesentlichen Stärken der Anwendung sind:

- **Vollständiger Workflow:** Die Anwendung deckt den gesamten Prozess von der Erstellung einzelner Fragen über die Zusammenstellung einer Prüfung bis hin zum Export druckfertiger .docx-Dokumente ab. Sie löst damit das in der Problemstellung beschriebene Kernproblem.
- **Flexibilität und Wiederverwendbarkeit:** Durch die hierarchische Fragenstruktur und das Speichern der Projekte im JSON-Format wird eine hohe Modularität und Wiederverwendbarkeit von Inhalten gefördert. Lehrende können leicht neue Prüfungen aus bestehenden Fragenpools zusammenstellen.
- **Gezielter Funktionsumfang:** Im Gegensatz zu überladenen Lernmanagementsystemen konzentriert sich der ExamBuilder auf eine einzige, klar definierte Aufgabe. Dies führt zu einer schlanken Anwendung mit einer steilen Lernkurve für die Zielgruppe.
- **Innovative Ansätze:** Funktionen wie die automatische Generierung von Prüfungsvarianten gehen über eine reine Verwaltungssoftware hinaus und bieten einen echten didaktischen Mehrwert.

6.2.2 Limitationen und Ausblick

Trotz der erfolgreichen Umsetzung der Kernanforderungen weist der Prototyp in seiner jetzigen Form einige Limitationen auf, die als Ansatzpunkte für zukünftige Erweiterungen dienen können:

- **Einfacher Rephraser-Algorithmus:** Der zur Textvariation eingesetzte Algorithmus basiert auf einem einfachen Thesaurus und kann keinen semantischen Kontext verstehen. Dies kann gelegentlich zu stilistisch unpassenden Synonymen führen. Zukünftige Versionen könnten auf fortgeschrittenere NLP-Modelle (Natural Language Processing) zurückgreifen.
- **Manuelle UI-Tests:** Wie im Testkonzept beschrieben, wurde die Benutzeroberfläche ausschließlich manuell getestet. Für eine produktive Anwendung mit kontinuierlicher Weiterentwicklung wäre die Einrichtung einer automatisierten UI-Testsuite (z.B. mit *TestFX*) sinnvoll, um die Stabilität bei Änderungen langfristig zu sichern.
- **Fehlende Kollaborationsfunktionen:** Die Anwendung ist als Einzelplatzlösung konzipiert. Eine zukünftige Ausbaustufe könnte eine Anbindung an eine zentrale Datenbank beinhalten, um die gemeinsame Nutzung und Verwaltung von Fragenpools durch mehrere Lehrende zu ermöglichen.
- **Begrenzte Fragetypen:** Eine weitere Limitation ist die auf die gängigsten Anwendungsfälle fokussierte Auswahl an Fragetypen. Während mit offenen Fragen, Multiple-Choice, Lückentext und Richtig/Falsch-Fragen eine solide Basis geschaffen wurde, die einen Großteil der Prüfungsanforderungen abdeckt, fehlen spezialisierte Typen wie beispielsweise Zuordnungsaufgaben (Drag-and-Drop) oder interaktive Diagramme. Das modulare Design der Anwendung, insbesondere die Abstraktion der ‘Question‘-Klasse, bietet jedoch eine robuste Grundlage, um das System in Zukunft um solche spezifischen Fragetypen zu erweitern.

Zusammenfassend lässt sich festhalten, dass der ExamBuilder ein erfolgreicher Prototyp ist, der die Machbarkeit und den Nutzen des Konzepts eindrucksvoll belegt. Er bietet eine solide Grundlage für ein potenzielles Open-Source-Projekt, das den Arbeitsalltag von Lehrenden erheblich erleichtern könnte.

7 Zusammenfassung und Ausblick

Dieses letzte Kapitel fasst die zentralen Ergebnisse der vorliegenden Arbeit zusammen, zieht ein abschließendes Fazit und gibt einen Ausblick auf mögliche zukünftige Weiterentwicklungen des ExamBuilder-Projekts.

7.1 Zusammenfassung der Ergebnisse

Das Ziel dieser Bachelorarbeit war die Konzeption und prototypische Implementierung einer Softwarelösung zur Effizienzsteigerung bei der Erstellung von Prüfungsdokumenten. Ausgehend von der Problemstellung des zeitaufwändigen und fehleranfälligen manuellen Prozesses wurden zunächst die funktionalen und nicht-funktionalen Anforderungen an ein solches System detailliert.

Auf Basis dieser Analyse wurde der *ExamBuilder* als Java-basierte Desktop-Anwendung konzipiert und entwickelt. Die Umsetzung erfolgte nach dem etablierten Model-View-Controller-Architekturmuster, um eine saubere Trennung der Verantwortlichkeiten und eine hohe Wartbarkeit zu gewährleisten. Das entwickelte System ermöglicht den gesamten Workflow von der Erfassung der Prüfungsmetadaten über die Erstellung und hierarchische Verwaltung von Fragen bis hin zum Export druckfertiger Prüfungs- und Lösungsdokumente im .docx-Format. Zu den Kernfunktionen gehören die Persistenz von Projekten durch JSON-Serialisierung, die Unterstützung von Rich-Text und Bildern in Fragen sowie innovative Funktionen wie die automatische Generierung von Prüfungsvarianten durch Text-Reformulierung und eine intelligente, bedingte Mischung von Aufgaben, sowie eine granulare Layout-Steuerung, die es dem Lehrenden erlaubt, durch manuelle Seitenumbrüche und selektiven Blocksatz das finale Erscheinungsbild des Dokuments präzise zu gestalten.

Die abschließende Evaluation hat gezeigt, dass der entwickelte Prototyp alle wesentlichen Anforderungen erfüllt und die im Vorfeld identifizierten Probleme adressiert. Die Anwendung stellt eine robuste und benutzerfreundliche Lösung dar, die den Prozess der Prüfungserstellung nachweislich vereinfacht.

7.2 Fazit

Das Projekt *ExamBuilder* hat erfolgreich demonstriert, dass es möglich ist, eine spezialisierte, aber dennoch flexible Software zu entwickeln, die eine signifikante Lücke zwischen komplexen Lernmanagementsystemen und der rein manuellen Dokumentenerstellung schließt. Der Prototyp belegt die Machbarkeit des Konzepts und liefert eine wertvolle, direkt einsetzbare Grundlage für ein Werkzeug, das den Arbeitsalltag von Lehrenden spürbar erleichtern kann. Die in dieser Arbeit getroffenen Design- und Implementierungsentscheidungen haben zu einer stabilen und erweiterbaren Anwendung geführt, die das initial formulierte Ziel vollständig erreicht hat.

7.3 Mögliche zukünftige Erweiterungen

Aufbauend auf dem aktuellen Prototyp sind zahlreiche Erweiterungen denkbar, um den Funktionsumfang und die Anwendbarkeit des ExamBuilders weiter zu erhöhen:

- **Intelligentere Textvariation:** Der aktuelle ‘Rephraser‘-Algorithmus könnte durch den Einsatz von fortgeschrittenen NLP-Modellen (Natural Language Processing) ersetzt werden, um semantisch passendere und qualitativ hochwertigere Textvariationen zu erzeugen.
- **Kollaborative Funktionen:** Eine der größten potenziellen Erweiterungen wäre die Umstellung von einer lokalen JSON-Speicherung auf eine zentrale Datenbank. Dies würde die Tür für Multi-User-Funktionen öffnen, sodass mehrere Lehrende gemeinsam an einem zentralen Fragenpool arbeiten und Prüfungen teilen können.
- **Erweiterung der Fragetypen:** Obwohl die Anwendung mit offenen Fragen, Multiple-Choice, Lückentext und Richtig/Falsch-Fragen bereits eine breite Palette an gängigen Aufgabenformaten unterstützt, liegt ein potenzielles Erweiterungsfeld in der Integration hochspezialisierter Fragetypen. Denkbar wären hier interaktive Formate wie Zuordnungsaufgaben (Drag-and-Drop) oder beschriftbare Diagramme, deren Implementierung auf der bestehenden modularen Architektur der ‘Question‘-Klasse aufbauen könnte.
- **Automatisierte UI-Tests:** Um die langfristige Stabilität des Projekts bei zukünftigen Änderungen zu sichern, wäre die Implementierung einer umfassenden Suite für automatisierte UI-Tests mit Frameworks wie *TestFX* ein wichtiger nächster Schritt.

Anhang

Dieses Kapitel enthält zwei vollständig mit der ExamBuilder-Anwendung generierte Klausuren. Sie dienen als praktische Beispiele, um die vielfältigen Exportfunktionalitäten und die Qualität der Ausgabeformate des ExamBuilders zu demonstrieren.

7.4 Beispielklausur: Projektmanagement

Die nachfolgende Klausur wurde vollständig mit der ExamBuilder-Anwendung generiert. Sie dient als reales Beispiel für die Exportfunktionalität und die Ausgabeformate.

Brandenburgische Technische Universität Cottbus-Senftenberg | Mathematik, Naturwissenschaften und Informatik

Klausur - Projektmanagement | SommerSemester2025

Bitte lesen Sie die folgenden Hinweise aufmerksam durch!

Hinweise:

• Ergänzen Sie bitte auf diesem Deckblatt die untenstehenden Angaben und unterschreiben Sie. Der Klausurbogen enthält ein Zusatzblatt; weitere erhalten Sie bei Bedarf von der Aufsicht. Tragen Sie auf allen Zusatzblättern sofort Ihren Nachnamen, Matrikelnummer und die Aufgabenummer ein.

• Verwenden Sie einen dokumentenechten Schreibstift (d. h. kein Bleistift). Verwenden Sie keinen Stift mit roter oder grüner Farbe.

• Trennen Sie den Klausurbogen nicht auf und nehmen Sie ihn nicht mit nach Hause. Notieren Sie die Antworten direkt in den Klausurbogen; der dafür vorgesehene Platz ist bei durchschnittlicher Handschriftgröße ausreichend.

• Zugelassene Hilfsmittel: Wörterbuch.. Schalten Sie alle mitgebrachten elektronischen Geräte – auch Fitnessarmbänder, MP3-Player, etc. – aus (bzw. komplett lautlos) und legen Sie diese außer Reichweite (z. B. in Ihren Rucksack).

• Die Bearbeitungszeit beträgt 90 Minuten. Sie können die Klausur jederzeit abgeben, jedoch bitten wir Sie aus Respekt gegenüber Ihren Mitstudierenden, den Raum in den letzten 10 Minuten vor dem Ende der Bearbeitungszeit nicht mehr zu verlassen, um übermäßige Störungen zu vermeiden.

Viel Erfolg!

Abschnitt: Von dem/der Studierenden auszufüllen

Name	
Vorname	
Matrikelnummer	
Unterschrift	

Abschnitt: Von dem/der Prüfenden auszufüllen

A1	A2	A3	A4	A5	A6	A7	Gesamt
15	16	10	12	3	4	12	72

Matrikelnummer: Name:

1. Projektdefinition und Projektrisiken (3 + 3 + 5 + 4 = 15 Punkte)

Geben Sie hier den Aufgabentext oder weitere Anweisungen ein.

a. Geben Sie an, welche Eigenschaften ein Projekt unter Anderem hat. (3 Punkte)

b. Geben Sie ein Beispiel (fiktiv) für ein konkretes Projekt, dass die zuvor genannten Eigenschaften hat. (3 Punkte)

c. Nennen Sie fünf Projektrisiken. (5 Punkte)

d. Was beschreibt die Prinzipal-Agenten-Theorie? (4 Punkte)

Matrikelnummer: Name:

2. MCQ (2 + 2 + 4 + 2 + 2 + 2 + 2 = 16 Punkte)

a. Was kann der Begleitausschuss frühzeitig erkennen? (2 Punkte)

☐ A) Akzeptanzprobleme
☐ B) Finanzierungsprobleme
☐ C) Technische Schwierigkeiten
☐ D) Ressourcenengpässe

b. Was beschreibt die Prinzipal-Agent-Theorie? (2 Punkte)

☐ A) Das mögliche Verhalten von Vertragspartnern in Hierarchien
☐ B) Das Kommunikationsmanagement in einem Projekt
☐ C) Die technischen Risiken eines Projekts
☐ D) keine der vorgenannten Antworten

c. Ein Teammitglied möchte sein Versagen in einer Teamsitzung mit dem Projektleiter verbergen und nimmt sich vor „nicht zu kommunizieren". Welche Aussage diesbezüglich ist nichtzutreffend? (4 Punkte)

☐ A) Die abweichende Darstellung einer wahren Situation ist bei einer Informationsasymmetrie zwischen Teammitglied und der Projektleitung gut möglich.
☐ B) Das ansonsten zuverlässige Teammitglied vermeidet die Kommunikation und bleibt der Sitzung einfach fern.
☐ C) Bei einem gleichen Wissensstand ist eine abweichende Darstellung einer wahren Situation in der Regel nicht gut möglich.
☐ D) keine der vorgenannten Antworten.

d. Der/die Projektleiter/in fragt Sie nach dem verspätetem betreten des Besprechungsraumes: „Wissen Sie eigentlich wieviel Uhr es ist? (im Raum hängt gut sichtbar eine Uhr). Wie verhalten Sie sich unter Anwendung des Kommunikationsmodells nach Schulz von Thun (4 Ohren)? (2 Punkte)

☐ A) Der Sender bitte um eine Sachinformation, ich antworte mit „JA".
☐ B) Der Sender vertraut mir, da die Uhr im Raum falsch gehen könnte, ich antworte mit der korrekten Uhrzeit.
☐ C) Der Sender appelliert an meine Pünktlichkeit. Ich entschuldige mich für die Verspätung.
☐ D) Der Sender offenbart dadurch, dass er eine neue Uhr braucht. Ich schenke ihm bei nächster Gelegenheit eine neue Uhr.

Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.

Matrikelnummer: Name:

e. Welche Probleme können bzgl. des Erlebensfeldes in der Kommunikation auftreten? (2 Punkte)

☐ A) Die Wahrnehmung eines wahrnehmbaren Sachverhalts kann bei Empfänger eingeschränkt sein.
☐ B) Bei einer perfekten Wahrnehmung eines wahrnehmbaren Sachverhalts kann die mangelnde Kognition (im Sinne der geistigen Verarbeitungsfähigkeit) dazu führen, dass eine falsche Reaktion (Verhalten Kommunikation) erfolgt.
☐ C) Bei einer perfekten Wahrnehmung eines wahrnehmbaren Sachverhalts und der vollständigen Kognition (im Sinne der geistigen Verarbeitungsfähigkeit) kann es passieren, dass das Ergebnis des Denkprozesses falsch kommuniziert wird.
☐ D) Alle der vorgenannten Antworten

f. Wofür wird ein uint32_t* oft in Embedded-Systemen verwendet? (2 Punkte)

☐ A. Für Strings
☐ B. Für das Lesen von Memory-Mapped-Registern
☐ C. Für Zufallszahlenerzeugung
☐ D. Für dynamische Interruptpriorisierung

g. Was passiert bei der Operation *ptr++? (2 Punkte)

☐ A. Der Pointer wird zuerst erhöht, dann dereferenziert.
☐ B. Die Variable wird zweimal gelesen.
☐ C. Der Inhalt der aktuellen Adresse wird gelesen, danach wird der Pointer erhöht.
☐ D. Es handelt sich um einen Compilerfehler.

Matrikelnummer: Name:

3. Zieldefinitionen und Make-Or-Buy (2 + 2 + 2 + 2 + 2 = 10 Punkte)

Definieren sie jeweils ein Projektziel, dem nur eines der SMART-Kriterien fehlt. Erläutern Sie zudem, warum das Kriterium nicht erfüllt ist.

a. S(M)ART – nicht messbar (2 Punkte)

b. S(M)ART – nicht messbar (2 Punkte)

c. SM(A)RT – nicht attraktiv (2 Punkte)

Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.

d. SMA(R)T – nicht realisierbar (2 Punkte)

e. SMAR(T) – nicht terminiert (2 Punkte)

Matrikelnummer: Name:

4. Make-Or-Buy (6 + 6 = 12 Punkte)

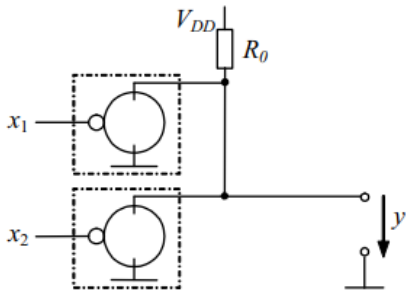


Bild 4-11 Zwei Gatter mit Open-Collector-Ausgängen, verschaltet zu einem virtuellen Gatter.

a. Geben Sie ein Beispiel für eine Make-Or-Buy-Entscheidung, die nach Anwendung des Entscheidungsprozesses eine nachvollziehbare Make-Entscheidung ergeben wird. Begründen Sie kurz (6 Punkte)

b. Geben Sie ein Beispiel für eine Make-Or-Buy-Entscheidung, die nach Anwendung des Entscheidungsprozesses eine nachvollziehbare Buy-Entscheidung ergeben wird. Begründen Sie kurz (6 Punkte)

Matrikelnummer: Name:

5. Wer ist Prof.Dr Steffen Vaupel? (3 Punkte)

```
Hier ist ein Code-Beispiel inline: int x = 10;


 public class MyClass
public static void main(String[] args) {
          // Eine einfache Schleife
           for (int i = 0; i < 5; i++) {
                System.out.println("Zähler: " + i);
           }
    }
 }
Variablen können auch so deklariert werden: String name =
"Danielou";
```

Matrikelnummer: Name:

6. Lückentext: Embedded Systems & Studium (4 Punkte)

Bitte füllen Sie die folgenden Lücken mit den passenden Begriffen aus der angegebenen Wortliste aus. Achten Sie darauf, dass die eingesetzten Begriffe grammatikalisch und inhaltlich korrekt in den Satz passen.

Wortliste:
STM32 – ESP32 – Dezember – Raumfahrt – denkend

Während meines **Studiums der Ingenieurinformatik an der Technischen Hochschule Mittelhessen** habe ich mich auf **Embedded Systems, Mikrocontroller-Programmierung und Softwareintegration** spezialisiert. Ich stehe kurz vor dem **Abschluss meines Bachelorstudiums (_____ 2025)** und konnte in mehreren Projekten fundierte Praxiserfahrung mit **ARM-basierten Mikrocontrollern (_____, _____)** und **drahtlosen Schnittstellen** sammeln.

Matrikelnummer: Name:

7. In diesem Teil werden wir eine reine Simulation von Richtig-oder-Falsch-Fragen durchführen, fürs Testen von ExamBuilder. **(2 + 2 + 2 + 2 + 2 + 2 = 12 Punkte)**

a. In C wird der Ausdruck *ptr++ zuerst dereferenziert und dann der Pointer erhöht. **(2 Punkte)** ☐ Richtig ☐ Falsch

b. Ein Pointer uint8_t *p kann verwendet werden, um direkt auf ein hardware-gemapptes Register zuzugreifen. **(2 Punkte)** ☐ Richtig ☐ Falsch

c. typedef struct { ... } Sensor; definiert einen neuen Datentyp namens Sensor **(2 Punkte)** ☐ Richtig ☐ Falsch

d. while(1); blockiert das Programm in einer Endlosschleife und wird häufig in Firmware als Hauptschleife verwendet. **(2 Punkte)** ☐ Richtig ☐ Falsch

e. Beim STM32 kann das Schreiben in bestimmte Konfigurationsregister eine Wartezeit erfordern, bis die Hardware den Wert übernommen hat. **(2 Punkte)** ☐ Richtig ☐ Falsch

f. Beim STM32 kann das Schreiben in bestimmte Konfigurationsregister eine Wartezeit erfordern, bis die Hardware den Wert übernommen hat. **(2 Punkte)** ☐ Richtig ☐ Falsch

7.5.1 Begründung der Repräsentativität

Die als Anhang beigefügte Musterklausur zum Thema "Projektmanagement und Embedded Systems"(siehe 7.4) wurde sorgfältig als repräsentatives Beispiel ausgewählt, da sie die Leistungsfähigkeit und Flexibilität des ExamBuilders anhand konkreter Aufgaben und deren Struktur umfassend demonstriert:

- **Umfassende Metadaten und Hinweise:** Das Deckblatt der Klausur enthält detaillierte administrative Informationen wie den Titel "Projektdefinition und Projektrisikenßowie die Gesamtpunktzahl von 72 Punkten, aufgeschlüsselt in einzelne Aufgaben (z.B. 15 für Aufgabe 1, 16 für Aufgabe 2). Diese Elemente unterstreichen die Fähigkeit des ExamBuilders, komplexe Metadaten effizient zu verwalten und in das Exportdokument zu integrieren, um ein vollständiges und professionelles Prüfungsdokument zu erstellen.
- **Vielfältige Fragetypen und Hierarchie:** Die Klausur beinhaltet eine breite Palette an Fragetypen, die die flexible Handhabung unterschiedlicher didaktischer Anforderungen des ExamBuilders belegen:
  - **Offene Fragen:** Aufgabe 1 ("Projektdefinition und Projektrisiken") mit ihren hierarchisch strukturierten Unterfragen a) bis d) (Ëigenschaften eines Projekts", "Beispiel für ein Projekt", "Projektrisiken", "Prinzipal-Agenten-Theorie") illustriert die Möglichkeit, detaillierte, freie Antworten zu fordern und komplexere Themen abzufragen.
  - **Multiple-Choice-Fragen (MCQs):** Aufgabe 2 demonstriert die Integration von MCQs mit vier Antwortoptionen, wie in Unterfrage 2.a ("Was kann der Begleitausschuss frühzeitig erkennen?") oder 2.d ("Der/die Projektleiter/in fragt Sie..."). Die korrekte Darstellung von langen Fragestellungen und die klare Trennung der Antwortoptionen sind hier ersichtlich.
  - **Lückentextaufgaben:** Aufgabe 6 (Ëmbedded Systems & Studium") zeigt die Fähigkeit, Lückentexte für die Abfrage von spezifischem Faktenwissen einzubinden, mit klar definierten Lücken, die durch eine Wortliste zu füllen sind.
  - **Richtig/Falsch-Fragen:** Aufgabe 7 repräsentiert eine Reihe von Richtig/Falsch-Aussagen (z.B. 7.a Ïn C wird der Ausdruck *ptr++ zuerst dereferenziert..."), die jeweils einzeln bewertet werden können, was die Effizienz der Korrektur unterstützt.
- **Detaillierte Punktevergabe:** Jede Frage und Unterfrage ist mit einer spezifischen Punktzahl versehen (z.B. "3 Punkte"für Aufgabe 1.a, "2 Punkte"für Aufgabe 2.a). Die Gesamtpunktzahl für Hauptaufgaben (z.B. "3 + 3 + 5 + 4 = 15 Punkte"für Aufgabe 1 oder "2 + 2 + 4 + 2 + 2 + 2 + 2 = 16 Punkte"für Aufgabe 2) wird korrekt ausgewiesen. Dies demonstriert die präzise Punkteverwaltung und -berechnung des ExamBuilders.
- **Integration von Code-Beispielen:** Aufgabe 5 präsentiert sowohl ein eingebettetes Inline-Code-Beispiel als auch einen Java-Codeblock:
  - Inline-Code: `int x = 10;`
  - Java-Codeblock:

```java
public class MyClass {
    public static void main(String[] args) {
        System.out.println("Hallo Welt!");
    }
}
```

Diese Beispiele illustrieren die Fähigkeit des ExamBuilders, formatierte Code-Segmente nahtlos in Prüfungsfragen zu integrieren und deren Lesbarkeit zu gewährleisten.

- **Manuelle Layout-Steuerung und Lesbarkeit:** Die Klausur nutzt strategisch platzierte Seitenumbrüche, wie die Meldung "Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite."(z.B. nach Unterfrage 2.d und 3.c). Dies zeigt die Bedeutung der Layout-Steuerungsfunktionen des ExamBuilders für die Erstellung professioneller und gut lesbarer Prüfungsdokumente, die eine optimale Trennung logischer Abschnitte gewährleisten. Die klare Strukturierung der Aufgaben durch Nummerierung und Einrückung fördert ebenfalls die Lesbarkeit.
- **Eingabe von Studierendendaten und Korrekturhilfen:** Das Deckblatt der Klausur enthält klare Felder für Name, Matrikelnummer und Unterschrift der Studierenden. Zudem ist ein Abschnitt für Prüfende mit einer Bewertungstabelle und maximal erreichbaren Punkten pro Aufgabe (z.B. A1, A2...) vorgesehen. Dies unterstreicht die vollständige Funktionalität des ExamBuilders zur Generierung umfassender Prüfungsmaterialien, die sowohl organisatorischen als auch Korrekturzwecken dienen.

Zusammenfassend demonstriert diese Beispielklausur umfassend die Stärken des ExamBuilders bei der strukturierten Erfassung, Verwaltung und dem Export von komplexen, multimedialen und spezifisch formatierten Prüfungsfragen, was die Effizienz und Qualität der Prüfungserstellung erheblich verbessert.

7.6 Beispielklausur: Praktische Informatik 1

Die nachfolgende Klausur wurde ebenfalls vollständig mit der ExamBuilder-Anwendung generiert und dient als Beispiel für eine technisch anspruchsvollere Prüfung.

Technische Hochschule Mittelhessen (THM) | Mathematik, Naturwissenschaften und Informatik(MNI)

Probeklausur - Praktische Informatik 1 | SommerSemester2025

Bitte lesen Sie die folgenden Hinweise aufmerksam durch!

• Verwenden Sie bitte den vorgesehenen Platz unterhalb der Aufgaben oder – falls zusätzlicher Raum benötigt wird – die Rückseite. Machen Sie in diesem Fall bitte auf der Vorderseite einen entsprechenden Hinweis.

• Verwenden Sie einen dokumentenechten Schreibstift (kein Bleistift) und keine rote oder grüne Farbe.

• Trennen Sie den Klausurbogen nicht auf und lassen Sie die Heftung vollständig intakt.

• Legen Sie Ihren Studierendenausweis gut sichtbar neben sich auf den Tisch.

• Elektronische und nicht elektronische Hilfsmittel sind nicht zugelassen, mit Ausnahme eines selbstgeschriebenen DIN-A4-Blattes. Schalten Sie Ihr Handy und alle weiteren elektronischen Geräte vollständig aus (oder stellen Sie sie komplett lautlos) und legen Sie diese außer Reichweite.

• Die Bearbeitungszeit beträgt 90 Minuten. Sie können die Klausur jederzeit abgeben, jedoch bleiben Sie bitte am Ende der Klausur auf Ihrem Platz sitzen, damit die Aufsicht die Klausuren einsammeln kann.

• Jegliche Kommunikation mit anderen Studierenden sowie die Nutzung nicht zugelassener Materialien wird als Täuschungsversuch gewertet.

Viel Erfolg!

Abschnitt: Von dem/der Studierenden auszufüllen

Name	
Vorname	
Matrikelnummer	
Unterschrift	

Abschnitt: Von dem/der Prüfenden auszufüllen

A1	A2	A3	A4	Gesamt
26	21	27	26	100

Matrikelnummer: Name:

1. Aufgabe (6 + 5 + 7 + 8 = 26 Punkte)

a. Erklären Sie den Begriff HTML-Element und geben Sie ein vollständiges Beispiel eines HTML-Elementes an. (6 Punkte)

b. Erklären Sie, wozu das Document Object Model (DOM) im Browser dient. (5 Punkte)

c. Erstellen Sie zu dem u.a. DOM ein entsprechendes HTML-Dokument. (7 Punkte)

Beispiele:

Block-Elemente	Inline-Elemente
div	span
p	img

Matrikelnummer: Name:

d. Gegeben ist folgender Rumpf eines HTML-Dokumentes. Geben Sie CSS-Selektoren für die einzelnen Elemente mit den Texten Eins, Zwei, Drei und Vier an. (8 Punkte)

```
<body>
<div id="c">
Eins
<p>Zwei</p>
<p>Drei</p>
</div>
<p>Vier</p>
</body>
```


Matrikelnummer: Name:

2. Aufgabe (5 + 7 + 9 = 21 Punkte)

a. Gegeben folgende noch unvollständige Funktionsdefinition in TypeScript. Geben Sie für die drei Fragezeichen jeweils gültige Typen an. „Any“ wird nicht als Antwort akzeptiert. (5 Punkte)

	Typ
Fragezeichen 1 (nach a)	
Fragezeichen 2 (nach b)	
Fragezeichen 3 (vor =>)	

```
const fkt1 = (a:? ,b: ?): ? => a.toString()+b;
```

b. Betrachten Sie die beiden u.a. teilweise unvollständigen Klassendefinitionen. (7 Punkte)

```
class Fahrzeug {
   marke : string;
   baujahr : number;
   constructor(a:string, b:number) {
      ? = a;
      this.baujahr = b;
   }
}
class Auto extends Fahrzeug {
     gewicht : number;
     constructor(marke: string, baujahr: number, gewicht: number)
{
          ?
     }
}
```

Vervollständigen Sie die Konstruktoren von Fahrzeug und Auto.

Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.

Matrikelnummer: Name:

c. Schreiben Sie ein TypeScript-Programm, das die Zahlen von 1 bis n ausgibt (mittels console.log()) und KEINE Schleife benutzt. (9 Punkte)

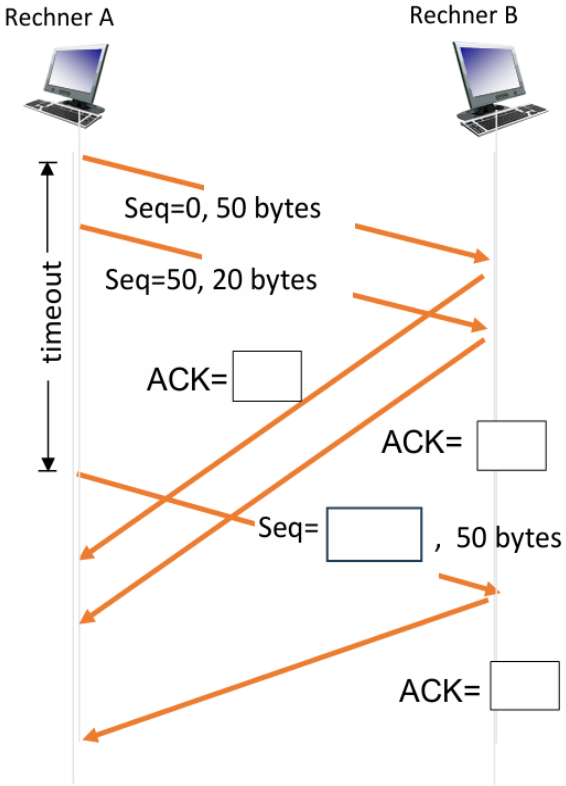
Matrikelnummer: Name:

3. Aufgabe (5 + 5 + 6 + 4 + 7 = 27 Punkte)

a. Geben Sie zu folgenden Begriffen aus dem Bereich Netzwerke an, was Ihre jeweilige Aufgabe ist. (5 Punkte)

Begriff	Aufgabe
Control Plane	
Netzwerkschicht	
Socket	
Domain Name System	
TCP	

b. Betrachten Sie den folgenden Ablauf einer Kommunikation über TCP. Schreiben Sie die korrekten Sequenz- und ACK-Nummern in die leeren Kästchen. (5 Punkte)



Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.

Matrikelnummer: Name:

c. Die Firma SuperData hat 20 TB Daten in einem Rechenzentrum in Paris gespeichert und muss dringend diese Daten in Frankfurt auswerten. Die Daten können von Paris nach Frankfurt entweder über ein 10 Gbit/s Glasfaserkabel oder auf einem Datenträger per Auto transportiert werden. Die Entfernung beträgt 500 km und das Auto fährt exakt 100 Kilometer pro Stunde. Weder bei der Verwendung des Kabels noch des Autos treten irgendwelche Verzögerungen auf. (6 Punkte)

Was ist schneller, die Verwendung des Kabels oder des Autos? Bitte ausführliche Herleitung mit korrekten Einheiten und Antwortsatz (Bitte rechnen Sie mit Faktor 1000, nicht mit 1024).

d. Erklären Sie, wieso es in einem Router zu Verzögerungen bei der Weiterleitung von Datagrammen kommen kann. (4 Punkte)

Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.

Matrikelnummer: Name:

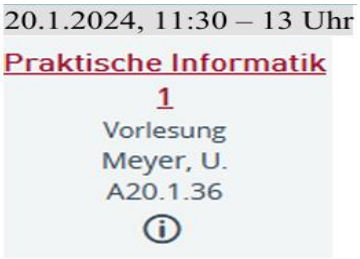
e. Berechnen Sie die CRC-Prüfsumme für D=10101011 und G=1001. (7 Punkte)

Matrikelnummer: Name:

4. Aufgabe (6 + 8 + 5 + 7 = 26 Punkte)

a. Was versteht man bei REST-Services unter „Ressource“ und „Repräsentation"? (6 Punkte)

b. Im THM-Organizer findet sich folgender Eintrag: (8 Punkte)



Stellen Sie diese Informationen als JSON-Objekt dar.

Matrikelnummer: Name:

c. Welche http-Verben benutzen Sie für folgende Zwecke, um mit dem Organizer zu interagieren: (5 Punkte)

Zweck	http-Verb
Abfragen eines Kalendereintrags	
Ändern des Dozenten eines Eintrags	
Erstellen eines neuen Eintrags	

d. Eines der sechs REST-Prinzipien ist „Caching“ – was ist der Vorteil und was der Nachteil von Caching? (7 Punkte)

7.7.1 Begründung der Repräsentativität

Diese Klausur aus der **"Praktische Informatik 1"** eignet sich hervorragend als ergänzendes, repräsentatives SZielszenario", da sie eine Vielzahl von Anforderungen abdeckt, die der ExamBuilder unterstützt:

- **Verwaltung von Metadaten und Hinweisen:** Alle Kopfdaten (Hochschule, Fachbereich, Modul, Semester, Titel, Bearbeitungszeit) sowie die umfangreichen Klausurhinweise können vollständig im ExamBuilder erfasst und auf dem Deckblatt abgebildet werden.
- **Hierarchische Fragenstruktur:** Die Klausur nutzt Haupt- und Unteraufgaben (z.B. Aufgabe 1.a, 1.b), die präzise der hierarchischen Modellierung des ExamBuilders entsprechen.
- **Rich-Text und Code-Formatierung:** Die Notwendigkeit, Code-Snippets (HTML, TypeScript, JSON) ansprechend darzustellen, wird durch den integrierten HTMLEditor und dessen dedizierte Code-Formatierungsfunktion optimal unterstützt. Dies ist für technische Prüfungen ein entscheidendes Merkmal.
- **Punktevergabe:** Die detaillierte Punktevergabe pro Unteraufgabe (z.B. $6 + 5 + 7 + 8 = 26$ Punkte) und die automatische Summation auf Hauptaufgabenebene werden vom ExamBuilder korrekt gehandhabt und auf dem Bewertungsbogen abgebildet.
- **Layout-Steuerung:** Die expliziten Hinweise für Seitenumbrüche ("Die Aufgabe folgt auf der nächsten Seite bzw. Rückseite.") können durch die SStart on New PageOption im ExamBuilder umgesetzt werden, wobei der `WordExporter` die entsprechenden Mitteilungen generiert.
- **Umgang mit speziellen Inhalten:** Aufgaben wie die zur TCP-Kommunikation (3.b), die visuelle Elemente (Kästchen für Sequenznummern) andeuten, zeigen eine Grenze der reinen Textdarstellung auf. Gleichzeitig demonstrieren sie die Stärke der **Bildintegration** des ExamBuilders, da solche Elemente problemlos als Grafik eingefügt werden könnten, um die visuelle Natur der Aufgabe zu erhalten.

Dieser Klausurentwurf ist somit ein ausgezeichnetes Beispiel, um die Leistungsfähigkeit des ExamBuilders bei der Strukturierung, Gestaltung und dem Export komplexer technischer Prüfungen zu demonstrieren und die Notwendigkeit der implementierten Features zu validieren.

Abbildungsverzeichnis

Literatur

[1] Gernot Starke. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 9. Aufl. Carl Hanser Verlag, 2020, S. 124–127. DOI: 10.3139/9783446465893 (siehe S. 3, 12).

[2] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. 2nd. Ã£Æ'Å¾Ã£â€šÂ¤Ã£Æ'Å Ã£Æ'â€œÃ¥â€¡Âº Ã§â€°Ë†, 2011. URL: https://ptgmedia.pearsoncmg.com/images/9780321683687/samplepages/0321683684.pdf (siehe S. 3).

[3] Paul Black und Dylan Wiliam. „Inside the black box: Raising standards through classroom assessment“. In: *Phi delta kappan* 92.1 (2010), S. 81–90. URL: https://greatschoolspartnership.org/wp-content/uploads/2016/11/Inside-the-Black-Box-of-Assessment-PDK-2010.pdf (siehe S. 3).

[4] Charles Secolsky und D Brian Denison. *Handbook on measurement, assessment, and evaluation in higher education*. Routledge New York, 2012. URL: https://scholar.google.com/scholar?hl=de&as_sdt=0%2C5&q=Handbook+on+Measurement%2C+Assessment%2C+and+Evaluation+in+Higher+Education&btnG= (siehe S. 3, 4).

[5] Helfried Moosbrugger und Augustin Kelava. *Testtheorie und Fragebogenkonstruktion*. 3., vollst. neu bearb., erw. u. akt. Aufl. Berlin; Heidelberg: Springer, 2020. DOI: 10.1007/978-3-662-61532-4 (siehe S. 3, 4).

[6] *JavaFX Documentation*. Oracle. URL: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm (besucht am 26. 09. 2025) (siehe S. 4).

[7] *Apache Maven Documentation*. Apache Software Foundation. URL: https://maven.apache.org/guides/ (besucht am 27. 09. 2025) (siehe S. 5).

[8] *Apache POI - The Java API for Microsoft Documents*. Apache Software Foundation. URL: https://poi.apache.org/ (besucht am 28. 09. 2025) (siehe S. 5).

[9] Jonathan Hedley. *jsoup Java HTML Parser*. https://jsoup.org/. Accessed on 2025-12-02. 2009-2025 (siehe S. 5).

[10] Bashar Nuseibeh und Steve Easterbrook. „Requirements Engineering: A Roadmap“. In: *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, S. 35–46. DOI: 10.1145/336512.336523 (siehe S. 8, 11).

[11] Martin Fowler. *POJO*. https://martinfowler.com/bliki/POJO.html. Accessed: 2025-09-28. 2003 (siehe S. 17).

[12] FasterXML. *Jackson Databind*. https://github.com/FasterXML/jackson-databind. Accessed: 2025-09-28 (siehe S. 18).

[13] Oracle. *JavaFX 19 - Task (JavaFX Concurrent)*. Accessed: 2025-09-28. 2022. URL: https://openjfx.io/javadoc/19/javafx.graphics/javafx/concurrent/Task.html (siehe S. 31).

[14] John Ladley. *Data Governance: How to Design, Deploy and Sustain an Effective Data Governance Program*. Morgan Kaufmann, 2012. URL: https://books.google.de/books?id=AkW9DwAAQBAJ (siehe S. 31).

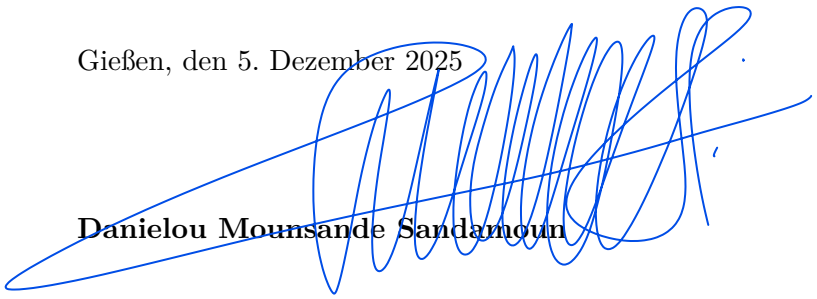
[15] JUnit Team. *JUnit 5*. https://junit.org/junit5/. Accessed: 2025-09-28 (siehe S. 32).

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne andere als die angegebenen Hilfsmittel angefertigt habe. Alle wörtlich oder sinngemäß übernommenen Stellen sind als solche kenntlich gemacht, und die Arbeit wurde weder vollständig noch auszugsweise zuvor einer anderen Prüfungsbehörde vorgelegt.

KI-basierte Werkzeuge wie DeepL, ChatGPT sowie die integrierten Korrekturhilfen von Overleaf wurden ausschließlich zur sprachlichen und stilistischen Überarbeitung genutzt. Die inhaltliche Ausarbeitung, Analyse, Strukturierung und Implementierung stammen vollständig von mir.

Gießen, den 5. Dezember 2025



Danielou Mounsande Sandamoun