Projektbegleitender Bericht

"Litera Book Catalog"

Dokumentenversion:
2.0 – Stand: 03.06.2025

Inhalt

1. Einleitung

Der vorliegende projektbegleitende Bericht gibt einen umfassenden Überblick über den Entwicklungsstand und die Fortschritte des Softwareprojekts *„Litera Book Catalogue“*. Ziel des Projekts ist die Entwicklung einer Desktop-Anwendung zur Verwaltung, Suche und Bewertung von Büchern auf Grundlage lokal gespeicherter JSON-Daten. Der Bericht deckt den Zeitraum vom Projektbeginn bis zum 06.06.2025 ab und richtet sich an Betreuende, Projektbeteiligte sowie relevante Stakeholder. Er enthält Informationen zu den organisatorischen Rahmenbedingungen, den funktionalen und technischen Anforderungen, der konkreten Umsetzung, durchgeführten Tests, identifizierten Herausforderungen sowie einem Ausblick auf abschließende Schritte.

2. Team und Rollen

Das Projektteam bestand aus vier Mitgliedern:
• Marija Ratnikova
• Lena Heberlein
• Rama Alshaer
• Ranim Khallouf

Marija Ratnikova übernahm die dynamische Verteilung der Rollen im Team. Jede Person war sowohl in die Analyse als auch in die Umsetzung der Anwendung sowie in die Erstellung der Dokumentation eingebunden. Dadurch konnten die Aufgaben gleichmäßig verteilt und alle Deadlines rechtzeitig eingehalten werden.

3. Entwicklungsphasen

	Phasen	Begin	Ende
1	Auswahl geeigneter Werkzeuge für Analyse, Entwicklung und Dokumentation	24.03.25	03.04.25
2	Anforderungsanalyse und Entwurf der Anwendungsarchitektur	04.04.25	22.04.25
3	Entwicklung und Testen der Anwendung	23.04.25	06.05.25
4	Verfassen der Projektdokumentation	05.05.25	09.05.25
5	Erstellung der JAR-Datei	09.05.25	09.05.25
6	Umsetzung neuer Anforderungen	23.05.25	05.06.25

3.1 Auswahl geeigneter Werkzeuge

Zu Beginn des Projekts analysierte das Team verschiedene Werkzeuge im Hinblick auf ihre Eignung für die jeweiligen Projektanforderungen. Für jede Kategorie – darunter Prototyping, IDE, Testautomatisierung, Dokumentation, Obfuskation, Versionskontrolle, UML-Modellierung und Build-Systeme – wurden mehrere Alternativen anhand definierter Kriterien (wie Benutzerfreundlichkeit, Funktionsumfang, Integration, Preis und Teamerfahrung) verglichen.

Auf dieser Basis wurden folgende Tools ausgewählt:

- **Figma** für Prototyping, aufgrund seiner Benutzerfreundlichkeit, Funktionsvielfalt und kollaborativen Möglichkeiten.

- **Visual Studio Code** als Entwicklungsumgebung, da es leichtgewichtig, erweiterbar, intuitiv und kostenlos ist.
- **JUnit** für Testautomatisierung wegen seiner einfachen Integration, klaren Syntax und breiten Community.
- **Javadoc** für die Projektdokumentation, da es vollständig in Java integriert ist und ohne großen Aufwand genutzt werden kann.
- **ProGuard** für die Obfuskation, weil es eine starke Verschleierung bei guter Usability bietet.
- **Git** als Versionsverwaltungstool, dank seiner Flexibilität, Verbreitung und vorhandenen Teamkompetenz.
- **PlantUML** für UML-Diagramme, da es eine einfache textbasierte Modellierung mit Codegenerierung ermöglicht.
- **Maven** als Build-Tool, aufgrund seiner strukturierten Projektorganisation und guten IDE-Integration.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Erstellung der Vergleichstabelle	2
	Analyse und Vergleich von Tools für das Prototyping der Benutzerschnittstelle	3
	Ausarbeitung und Formatierung der Dokumentation	2
Lena Heberlein	Analyse und Vergleich von Kollaborationstools sowie Versionsverwaltungssystemen	5
Rama Alshaer	Analyse und Vergleich von UML-Tools mit Codegenerierung, Build-Tools und Versionsverwaltungssystemen	3
Ranim Khallouf	Analyse und Vergleich von Dokumentationstools sowie Obfuskatoren	2

3.2 Anforderungsanalyse und Entwurf der Anwendungsarchitektur

In dieser Phase analysierte jedes Teammitglied die Anforderungen an die Anwendung und arbeitete an einem spezifischen Typ von UML-Diagramm. Ziel war es, eine geeignete Anwendungsarchitektur zu entwickeln und eine benutzerfreundliche Oberfläche zu gestalten.

Im Ergebnis entstanden vier UML-Diagramme:
– Klassendiagramm
– Anwendungsfalldiagramm
– Sequenzdiagramm
– Aktivitätsdiagramm

Basierend auf der Analyse wurde eine modulare Architektur formuliert. Der Code ist dabei in mehrere logische Schichten unterteilt:
– ein Speichermanagement-Modul (StorageService),

– eine zentrale Logikschicht mit den Hauptfunktionen (Buchkatalog),
– ein Controller zur Koordination der Abläufe,
– sowie eine grafische Benutzeroberfläche mit verschiedenen Panels.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Erstellung des Anwendungsfalldiagramms	3
	Konzeption der Gesamtarchitektur sowie Evaluation und Korrektur des Endergebnisses	6
Lena Heberlein	Erstellung des Klassendiagramms	7
Rama Alshaer	Erstellung des Sequenzdiagramms	8
Ranim Khallouf	Erstellung des Aktivitätsdiagramms	7

3.3 Entwicklung und Testen der Anwendung

In dieser Phase begann die eigentliche Entwicklung der Anwendung. Zu Beginn wurden mithilfe des zuvor erstellten Klassendiagramms die zentralen Klassen generiert. Anschließend wurden Klassen wie Buch, Autor, Verlag, Genre usw. strukturiert und mit passenden Getter- und Setter-Methoden ergänzt.

Danach erfolgte die parallele Arbeit an verschiedenen Modulen:
– an der **Datenhaltungsschicht** (StorageService), die für das Laden und Speichern von Daten aus der JSON-Datei zuständig ist,
– an der **Logikschicht** (Buchkatalog), die die Hauptfunktionen zur Verarbeitung der Daten enthält,
– und an der **Entwicklung des Benutzerinterfaces**, das zunächst mit Wireframes konzipiert und anschließend in Figma prototypisch visualisiert wurde.

Auf Basis dieser Architektur wurde der **Controller** implementiert, der zwischen der Logik- und UI-Schicht vermittelt. Im weiteren Verlauf wurden zusätzliche Interface-Klassen erstellt, um die verschiedenen Bereiche der grafischen Benutzeroberfläche klar zu trennen und eine bessere Nutzererfahrung zu gewährleisten.

Abschließend wurde der Code überarbeitet, umfassend kommentiert.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Erstellung von Wireframes	3
	Entwicklung des Prototyps in Figma	6
	Implementierung der Benutzeroberfläche mit Java Swing	20
	Anpassung des Codes an die Mozilla Java Code Conventions	1
	Testierung verschiedener Funktionalitäten mit JUnit	2
	Aktualisierung des Aktivitätsdiagramms und des Klassendiagramms	1
	Anpassung der Projektstruktur	1
Lena Heberlein	Entwicklung des Storage Service-Moduls	7
	Codegenerierung aus dem Klassendiagramm	5

	Mitarbeit an der Klasse Buchkatalog	6
Rama Alshaer	Implementierung der Logikklasse Buchkatalog	5
	Entwicklung des Controllers	3
Ranim Khallouf	Überarbeitung der Klassen(Buch, Autor, Genre, Verlag, Rezenzion und andere) nach der Codegenerierung	9
	Unterstützung bei der UI-Entwicklung	10

3.4 Verfassen der Projektdokumentation

In dieser Phase wurde die Projektdokumentation erstellt und weiterentwickelt. Dazu gehören drei Hauptteile:
– die **Benutzerdokumentation**, um Endanwendern die Nutzung der Anwendung zu erleichtern,
– die **Administratordokumentation**, mit Hinweisen zur Installation, Wartung und Weiterentwicklung,
– sowie die **Entwicklerdokumentation**, die den technischen Aufbau und die Architektur der Software detailliert beschreibt.

Zur besseren Wartbarkeit und Verständlichkeit des Quellcodes wurde für alle Hauptklassen und zentralen Methoden eine ausführliche JavaDoc-Dokumentation erstellt. Diese Dokumentation wurde automatisch generiert und als HTML exportiert, um den Zugriff auf Klassendetails, Methodenbeschreibungen und Parametererklärungen zu erleichtern.

Zusätzlich wurde eine erste Version des **projektbegleitenden Berichts** verfasst, die den gesamten Entwicklungsprozess reflektiert.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Überarbeitung und Erweiterung der Benutzerdokumentation	1
	Überschreiben der Administratordokumentation	2
	Anpassung der Entwicklerdokumentation	1
	Aktualisierung des Pflichtenhefts	2
	Erstellung der JavaDocs	1
Lena Heberlein	Erstellung der ersten Version der Benutzerdokumentation	3
	Erstellung des projektbegleitenden Berichts	4
Rama Alshaer	Erstellung der Entwicklerdokumentation	4
Ranim Khallouf	Erstellung der Administratordokumentation	4

3.5 Erstellung der JAR-Datei und Obfuskation

Die Erstellung der ausführbaren JAR-Datei sowie die anschließende Obfuskation wurde vollständig von **Marija Ratnikova** übernommen. Mithilfe des Maven Shade Plugins wurde zunächst ein sogenanntes Fat-JAR erzeugt, das alle Abhängigkeiten der Anwendung enthält. Anschließend erfolgte die Obfuskation mit dem ProGuard-Plugin, um den Quellcode zu verschleiern und die Anwendung zu schützen.

Alle Projektdokumente, das Java-Projekt, der Prototyp sowie die zugehörigen Daten wurden vollständig zusammengetragen, strukturiert und korrekt für die Abgabe vorbereitet.

Trotz erfolgreicher Generierung der Datei buchkatalog-1.0.0-all.jar trat bei der Ausführung ein Fehler auf, da wichtige Ressourcen wie die JSON-Datei (books_short.json) und die Bilder im Unterordner images nicht korrekt eingebunden waren. Diese lagen außerhalb des Classpaths und konnten somit zur Laufzeit nicht geladen werden, was dazu führte, dass die Anwendung keine Bücher anzeigen konnte.

Dieser Fehler wurde im Anschluss analysiert und behoben, indem die Dateien korrekt in den Ressourcenordner src/main/resources verschoben und im Build-Prozess berücksichtigt wurden.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Erstellung der ausführbaren JAR-Datei mit dem Maven Shade Plugin (Erzeugung eines Fat-JARs inklusive aller Abhängigkeiten)	1
	Durchführung der Obfuskation mit dem ProGuard-Maven-Plugin	2
	Aufbereitung und Strukturierung des GitHub-Repositories zur Abgabe (inklusive Finalisierung aller relevanten Dateien und Ordner)	1
Lena Heberlein	Hilfestellung bei der Prüfung und Überarbeitung der Projektdokumente.	2

3.6 Umsetzung neuer Anforderungen

Auf Basis des Kundenfeedbacks wurden neue funktionale Anforderungen formuliert und in die Anwendung integriert. Dazu gehörten insbesondere:

- Anzeige von Buchcovern
- Vorschläge ähnlicher Bücher basierend auf Genre
- Einführung einer About-Funktion im Startbildschirm, um Informationen über das Unternehmen darzustellen

Zur Umsetzung dieser Anforderungen wurden neue Benutzeroberflächenklassen erstellt sowie bestehende Klassen wie Buch und Buchkatalog erweitert. Außerdem wurde im Interface eine Galerie integriert, in der ähnliche Bücher horizontal angezeigt werden. Beim Anklicken eines Buches aus dieser Galerie wird die Detailansicht des jeweiligen Buches geöffnet.

Zusätzlich wurden der JavaDoc erneut generiert, die JAR-Datei neu erstellt, eine Obfuskation durchgeführt sowie die gesamte Dokumentation aktualisiert.

Aufgabenzuteilung:

Name	Erledigte Aufgaben	Stunden
Marija Ratnikova	Erstellung neuer UI-Klassen zur Anzeige ähnlicher Bücher und Erstellung der About-Klasse	6
	Überarbeitung bestehender Klassen zur Darstellung von Covern	1
	Umstrukturierung des Projekts und Anpassung der Pfade im Code, um eine korrekte Funktion der Anwendung sicherzustellen	4

	Erneute Obfuskation und Erstellung der neuen JAR-Datei	5
	Überarbeitung der Administratorendokumentation und Aktualisierung des Pflichtenhefts	2
	Den projektbegleitenden Bericht neu verfassen	5
Lena Heberlein	Aktualisierung aller Dokumentationen und UML-Diagramme	5
	Erstellung des projektbegleitenden Berichts	3
Rama Alshaer	Implementierung der Methode zur Ermittlung ähnlicher Bücher	5
	Test für die Funktion, was sucht ähnliche Bücher	2
	Generierung von Java Doc	2
	Obfuskation und Generierung des neuen JAR	8
Ranim Khallouf	Ergänzung der Buchcover im Projekt	3
	Pflege der JSON-Daten	5
	Erstellung der About-Button und About Window	2

3.7 Gesamtzeitaufwand

Die Entwicklung der Anwendung *Litera Book Catalog* einschließlich aller Dokumentationen und der vollständigen Implementierung erforderte insgesamt ca. 200 Stunden Arbeitsaufwand.

Aufwand nach Teammitgliedern:

Name	Stunden
Marija Ratnikova	84
Lena Heberlein	47
Rama Alshaer	40
Ranim Khallouf	42

4. Schwierigkeiten während der Projektdurchführung

Während der Projektarbeit traten verschiedene Herausforderungen auf, die vom Team gemeinsam gemeistert wurden:

- **Einbindung externer Ressourcen:**
  Bei der Erstellung der JAR-Datei kam es zu Schwierigkeiten, da externe Ressourcen (wie JSON-Dateien und Bilder) zunächst nicht korrekt eingebunden wurden. Diese mussten anschließend in den src/main/resources-Ordner verschoben und entsprechend referenziert werden.

- **Java Swing-Interface:**
  Die Entwicklung eines ansprechenden und intuitiven Benutzerinterfaces mit Java Swing erwies sich als zeitaufwändiger als geplant. Insbesondere die Positionierung und dynamische Anpassung der GUI-Komponenten erforderten intensive Anpassungen und mehrfaches Testing.

- **Rollenverteilung:**
  Zu Beginn des Projekts erfolgte die Aufgabenverteilung durch gemeinsame Diskussionen, wobei sich jedes Teammitglied selbstständig Aufgaben auswählte. Dabei wurden jedoch individuelle Stärken und Vorkenntnisse nicht ausreichend berücksichtigt, was teilweise zu

unzureichenden Ergebnissen führte und eine Nachbearbeitung erforderlich machte. In der späteren Phase übernahm Marija Ratnikova die Koordination der Aufgabenverteilung. Trotz dieser zentralisierten Organisation kam es in Einzelfällen dazu, dass nicht alle Anforderungen im Voraus vollständig erfasst wurden, was dazu führte, dass bestimmte Aufgaben kurzfristig und unter Zeitdruck von einer Person allein bearbeitet werden mussten.

- **Obfuskation mit ProGuard:**
  Der Einsatz von ProGuard zur Obfuskation der Anwendung stellte sich als besonders herausfordernd heraus. Trotz intensiver Bemühungen gelang es nicht, eine funktionierende Konfiguration zu erstellen, die sowohl alle notwendigen Klassen enthielt als auch die Anwendung lauffähig ließ. Aufgrund von wiederholten Build-Fehlern und Kompatibilitätsproblemen konnte der Obfuskationsprozess letztlich nicht erfolgreich abgeschlossen werden.

5. Abschließende Bewertung

Das Projekt **„Litera Book Catalogue“** wurde – trotz einiger technischer und organisatorischer Hürden – erfolgreich abgeschlossen.

- **Zielerreichung.**
  - Alle Muss-Funktionen (Buchsuche, Detailansicht, Rezensionen, ähnliche Bücher, About-Fenster) sind implementiert und lauffähig.
  - Die Anwendung folgt einer klaren Schichten-Architektur; JSON-Persistenz, Business-Logik und Java-Swing-GUI sind sauber getrennt.
  - Eine ausführbare Fat-JAR-Datei wird automatisiert per Maven generiert; der Build-Prozess ist reproduzierbar dokumentiert.
  - Benutzerdokumentation, Administratoren- und Entwicklerdokumentation wurden erstellt, versioniert und mit JavaDoc ergänzt.
- **Teamleistung.**
  - Die Zusammenarbeit im Team verlief insgesamt strukturiert und zielorientiert. In der Anfangsphase erfolgte die Aufgabenverteilung zunächst auf freiwilliger Basis, wobei individuelle Stärken noch nicht vollständig berücksichtigt wurden. Im weiteren Projektverlauf wurde die Verteilung konkreter organisiert, was zu einer effizienteren Umsetzung beitrug.
  - Regelmäßige Absprachen, darunter wöchentliche Treffen und gemeinsame Reviews, unterstützten die Klärung offener Fragen und trugen dazu bei, den Projektumfang im vorgesehenen Rahmen zu halten.

Insgesamt erwies sich das Projekt als wertvolle Erfahrung für das gesamte Team, in der wir unsere Kenntnisse in Software-Engineering, Teamarbeit und Projektmanagement vertiefen und erfolgreich praktisch anwenden konnten. Die gewonnenen Erkenntnisse, insbesondere im Umgang mit komplexen Schnittstellen und der technischen Dokumentation, bilden eine solide Basis für zukünftige Projekte.