Projektbegleitender Bericht

"Litera Book Catalogue"

Dokumentenversion:
2.0 – Stand: 03.06.2025

Inhalt

1. Einleitung

Der vorliegende projektbegleitende Bericht dokumentiert den aktuellen Stand sowie die Fortschritte des Softwareprojekts „Litera Book Catalogue“. Ziel des Projekts ist die Entwicklung einer Desktopanwendung zur Katalogisierung, Suche und Bewertung von Büchern auf Basis lokaler JSON-Daten. Dieser Bericht bezieht sich auf den Zeitraum vom Projektstart bis zum 06.06.2025 und richtet sich an Betreuer, Projektteam und Stakeholder. Er beschreibt organisatorische Rahmenbedingungen, Anforderungen, technische Umsetzung, Testaktivitäten sowie Risiken und gibt einen Ausblick auf die weiteren 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.

Diese fundierte Auswahl stellte sicher, dass das Projektteam effizient, koordiniert und zielgerichtet arbeiten konnte.

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:

- **Marija Ratnikova** – Erstellung des Anwendungsfalldiagramms, Konzeption der Gesamtarchitektur sowie Evaluation und Korrektur des Endergebnisses
- **Lena Heberlein** – Erstellung des Klassendiagramms
- **Rama Alshaer** – Erstellung des Sequenzdiagramms
- **Ranim Khallouf** – Erstellung des Aktivitätsdiagramms

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 und mit Hilfe von Javadoc dokumentiert.

Aufgabenzuteilung:

- **Marija Ratnikova** – Erstellung von Wireframes, Entwicklung des Prototyps in Figma, Implementierung der Benutzeroberfläche mit Java Swing, Generierung der JavaDoc
- **Lena Heberlein** – Entwicklung des StorageService-Moduls, Erstellung der ersten Version der pom.xml
- **Rama Alshaer** – Implementierung der Logikklasse Buchkatalog sowie Entwicklung des Controllers
- **Ranim Khallouf** – Überarbeitung der Klassen Buch, Autor, Verlag, Genre sowie Unterstützung bei der UI-Entwicklung

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.

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

Aufgabenzuteilung:

- **Marija Ratnikova** – Überarbeitung und Erweiterung der Benutzerdokumentation, Korrektur der Administratordokumentation
- **Lena Heberlein -** Erstellung der ersten Version der Benutzerdokumentation und des projektbegleitenden Berichts
- **Rama Alshaer** – Erstellung der Entwicklerdokumentation

- **Ranim Khallouf** – Erstellung der Administratordokumentation

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.

Trotz erfolgreicher Generierung der Datei buchkatalog-1.0.0-all-obf.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.

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.

Aufgabenverteilung:

- **Marija Ratnikova** – Erstellung neuer UI-Klassen zur Anzeige ähnlicher Bücher, Überarbeitung bestehender Klassen zur Darstellung von Covern, erneute Obfuskation und Erstellung der neuen JAR-Datei, sowie Überarbeitung des *Projektbegleitenden Berichts*
- **Lena Heberlein** – Aktualisierung aller Dokumentationen und UML-Diagramme
- **Rama Alshaer** – Implementierung der Methode zur Ermittlung ähnlicher Bücher, Obfuskation und Generierung des neuen JAR
- **Ranim Khallouf** – Ergänzung der Buchcover im Projekt, Pflege der JSON-Daten sowie Erstellung der About-Klasse

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.

- **Koordination der parallelen Arbeitsschritte:**
  Die parallele Entwicklung der unterschiedlichen Anwendungsschichten (Speicherverwaltung, Logik, Benutzeroberfläche) führte zeitweise zu Synchronisationsproblemen. Durch regelmäßige Meetings und klare Rollenzuweisungen konnten diese Probleme jedoch effektiv gelöst werden.

- **Obfuskation:**
  Der Einsatz von ProGuard für die Obfuskation stellte sich als herausfordernd heraus. Vor allem die Konfiguration, um alle benötigten Klassen zu erhalten und dennoch den Code ausreichend zu verschleiern, war komplexer als ursprünglich angenommen.

5. Abschließende Bewertung

Das Projekt „Litera Book Catalogue“ wurde trotz einiger Herausforderungen erfolgreich umgesetzt. Die enge Zusammenarbeit im Team, die regelmäßige Kommunikation und die klare Aufgabenverteilung ermöglichten eine rechtzeitige und qualitativ hochwertige Fertigstellung der Anwendung.

Die entwickelten UML-Diagramme boten eine solide Grundlage für die spätere Implementierung und erleichterten die systematische Umsetzung. Besonders hervorzuheben ist die schnelle Reaktionsfähigkeit des Teams auf zusätzliche Kundenanforderungen, wodurch die Kundenzufriedenheit erheblich gesteigert wurde.

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.