Projektbegleitender Bericht

"Litera Book Catalogue"

Dokumentenversion:
1.0 – Stand: 09.05.2025

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 09.05.2025 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 Projektorganisation und Rollen

Das Projektteam setzt sich aus vier Mitgliedern zusammen:

- **Projektleitung & Koordination**: – verantwortlich für Zeitplanung, Abstimmung mit Betreuern und Berichterstattung.
- **Entwicklung Desktop-Frontend**: – zuständig für die GUI-Implementierung unter Java/Swing.
- **Backend- und Speicherlogik**: – Implementierung der JSON-Datenhaltung und Suchalgorithmen.
- **Qualitätssicherung & Dokumentation**: – verantwortlich für Testdesign, Testergebnisse und Erstellung der Benutzer- und Administratorendokumentation.

3 Anforderungen

3.1 Funktionale Anforderungen

Aus dem Pflichtenheft wurden folgende Muss-Kriterien definiert:

1. **Freitextsuche**: Der Anwender kann nach Titel, Autor, Verlag oder Genre suchen und eine Trefferliste erhält.
2. **Bibliotheksübersicht**: Darstellung der Ergebnisse als scrollbare Buch-Cards.
3. **Detailansicht**: Anzeige von Metadaten (Titel, Autor, Publisher, Genre) und Beschreibungstext.
4. **Rezensionsfunktion**: Nutzer können Bewertungen (1–5 Sterne) abgeben und Kommentare speichern.
5. **Navigation**: Zurück-Button zur Bibliotheksübersicht.

Bisher umgesetzt: Die Suche nach allen vier Kriterien ist fehlerfrei implementiert (Kapitel 2.1 bis 2.3 der Anwenderdokumentation). Die Detailansicht (Kap. 2.4) und die Rezensionsfunktion inklusive Dialogs (Kap. 2.5) wurden erfolgreich entwickelt.

3.2 Nichtfunktionale Anforderungen

- **Benutzerfreundlichkeit**: Einfache, moderne Swing-Oberfläche gemäß UI-Prototyp (siehe Anwenderdokumentation Abbildungen). Usability-Tests mit drei Testnutzern ergaben positive Rückmeldungen.
- **Portabilität**: Die Anwendung läuft auf Windows, macOS und Linux (Java 11+).

4 Technische Umsetzung & Architektur

Für das Design der Benutzeroberfläche wurde ein interaktiver Prototyp erstellt, der sämtliche Bildschirme und Navigationselemente visualisiert. Dieser Prototyp liegt als PDF vor und zeigt das Layout der Startseite, der Trefferliste und der Detailansicht mit Reviews-Bereich sowie den Dialog zum Schreiben von Rezensionen.

4.1 UI-Prototyp

Der Prototyp dokumentiert den Workflow von der Suche bis zur Detailanzeige. Er umfasst:

- Startbildschirm mit Logo, Suchfeld und Filter-Dropdown
- Bibliotheksübersicht als scrollbare Liste von Buch-Cards
- Detailansicht inklusive Metadaten, Beschreibung und Reviews
- Modal-Dialog zum Schreiben neuer Rezensionen

4.2 Klassenübersicht

Die Anwendung folgt dem MVC-Muster, implementiert über folgende zentrale Klassen und Interfaces:

- **ApplicationInterface**: Schnittstellendefinition für alle UI-zu-Logik-Operationen (Suche, Detailabruf, Reviews)
- **ApplicationWindow**: Hauptfenster mit CardLayout; lädt Start-, Listen- und Detail-Panels und startet die Swing-Anwendung
- **StartPanel**: Begrüßt den Nutzer beim Programmstart, zeigt Logo, Titel und Start-Button zum Wechsel in die Bibliotheksübersicht

- **Controller**: Vermittler, der die Operationen des ApplicationInterface an den Buchkatalog weiterleitet
- **Buchkatalog**: Geschäftslogikklasse für Suchen, Detailabruf und Review-Verwaltung; nutzt StorageService zur JSON-Datenhaltung

Modellklassen

- **Buch**: Repräsentiert Kerninformationen eines Buches (Titel, Autor, Verlag, Genres, ID, Beschreibung)
- **Autor**: Kapselt Autor-Name für Anzeige und JSON-Parsing
- **Genre**: Repräsentiert ein Genre mit Namen und Beschreibung (z. B. „Fantasy: Magische Welten…“)
- **Verlag**: Modelliert den Verlag eines Buches mit aktuellem Fokus auf den Namen
- **Rezension**: Modellklasse für Nutzerbewertungen mit Bewertung (1–5), Kommentar, Datum und Buch-ID

Service-Klassen

- **StorageService**: Verantwortlich für Laden und Speichern von Buch- und Rezensionsdaten in JSON-Dateien (books_short.json, reviews.json) mit Jackson

4.3 Datenhaltung & JSON-Struktur

Alle Buch- und Review-Daten liegen in lokalen JSON-Dateien:

- **books_short.json** enthält Buchdatensätze mit Feldern wie bookId, title, author, publisher, genres und description.
- **reviews.json** speichert alle Rezensionen mit bookId, bewertung, kommentar und datum.

Die Klasse StorageService nutzt Jacksons ObjectMapper, um beide Dateien bidirektional in Java-Objekte zu überführen und neue Rezensionen dauerhaft zu speichern.

4.4 UML-Diagramme

Zur Visualisierung der Systemstruktur und Abläufe wurden folgende UML-Diagramme erstellt:

4.4.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm zeigt die zentralen Akteure (Nutzer) und Anwendungsfälle: Suche nach Büchern, Anzeige von Buch-Details und Schreiben

von Rezensionen. Es bildet die funktionalen Anforderungen aus dem Pflichtenheft ab.

4.4.2 Klassendiagramm

Im Klassendiagramm sind die wichtigsten Klassen und deren Beziehungen dargestellt. Dazu gehören Modellklassen (Buch, Autor, Genre, Rezension), Service-Klassen (StorageService), Controller und die GUI-Komponenten (StartPanel, BookListPanel, BookDetails).

4.4.3 Aktivitätsdiagramm

Das Aktivitätsdiagramm beschreibt den Ablauf des Suchvorgangs: Eingabe des Suchbegriffs, Aufruf der Suchmethode im Controller, Ladung der Daten durch StorageService und Ausspielen der Ergebnisse in der Bibliotheksübersicht.

4.4.4 Sequenzdiagramm

Im Sequenzdiagramm wird die Interaktion zwischen GUI, Controller und StorageService während einer Such-Operation sequenziell dargestellt. Es verdeutlicht das Nachrichtenprotokoll und die Aufrufreihenfolge.

5 Test und Qualitätssicherung Test und Qualitätssicherung

Die Qualitätssicherung umfasst umfangreiche Unit- und Integrationstests, um die korrekte Funktionalität aller Kernkomponenten sicherzustellen.

5.1 Unit-Tests

- **BuchTest**: Validiert die Setter- und Getter-Methoden der Modellklasse Buch. Der Test legt einen Buch-Datensatz mit Buch-ID, Titel, Autor, Verlag, Genres und Beschreibung an und überprüft, dass alle Felder korrekt ausgelesen werden. Dies gewährleistet die Datenintegrität der Buchobjekte.
- **RezensionTest**: Testet die Klasse Rezension auf korrekte Umsetzung von toString() sowie die Funktionalität der Setter- und Getter-Methoden. Der Test stellt sicher, dass Bewertungen, Kommentare und Datumsangaben korrekt zugewiesen und nicht null sind.
- **StorageServiceTest**: Simuliert das Laden von Büchern aus einer JSON-Datei. Der Test erstellt eine temporäre books_short.json mit einem einzelnen Buch-Eintrag und verifiziert anschließend, dass StorageService.ladeBuecher() die erwartete Liste mit korrekten Buch-IDs und Titeln zurückgibt.

5.2 Integrationstests

Zur Prüfung des Zusammenspiels der Komponenten werden automatisierte Integrationstests eingesetzt. Diese führen vollständige Suchläufe durch, öffnen Detailansichten und prüfen, ob die korrekten Daten angezeigt werden. Fehlerfälle wie leere Suchanfragen werden ebenfalls abgedeckt.

5.3 Usability-Tests

Drei Testnutzer führten typische Anwendungsfälle (Suche, Detailanzeige, Rezension schreiben) durch. Das gesammelte Feedback floss in UI-Optimierungen ein, darunter größere Buttons und höhere Kontraste.

6 Entwicklungsumgebung & Toolauswahl

Im Folgenden werden die im Projekt verwendeten Tools und Entwicklungsumgebungen vorgestellt und die Auswahl begründet. Die Bewertung basiert auf Kriterien wie Benutzerfreundlichkeit, Funktionsumfang, Community-Support und Team-Erfahrung.

6.1 Prototyping-Tool

Figma wurde als Prototyping-Tool ausgewählt, da es eine intuitive Benutzeroberfläche, umfangreiche Funktionen für interaktive Prototypen und Echtzeit-Kollaboration im Team bietet. Figma ist kostenlos verfügbar und die Teammitglieder sind bereits damit vertraut, wodurch kein zusätzlicher Schulungsaufwand entsteht.

6.2 IDE/Editor

Visual Studio Code wurde als Entwicklungsumgebung gewählt. Es überzeugt durch hohe Performance, ein umfangreiches Plugin-Ökosystem, benutzerfreundliche Oberfläche und starke Community-Unterstützung. VS Code ist plattformunabhängig und kostenfrei.

6.3 Test-Automatisierung

Für Unit-Tests nutzen wir **JUnit**, da es leichtgewichtig, gut dokumentiert und nahtlos in VS Code integrierbar ist. Mit JUnit lassen sich Tests schnell implementieren und ausführen.

6.4 Dokumentationstools

Zur Generierung der API-Dokumentation verwenden wir **Javadoc**, das direkt ins JDK integriert ist und eine einfache Nutzung ohne zusätzliche Tools ermöglicht. Javadoc bietet vielfältige Exportformate und ist für Java-Projekte Standard.

6.5 Obfuscator

Zur Code-Obfuskation setzen wir **ProGuard** ein. ProGuard ist Open Source, unterstützt moderne Java-Standards und bietet eine gute Balance zwischen Verschleierungskraft und Benutzerfreundlichkeit.

6.6 Code Conventions

Als Java-Stilrichtlinie verwenden wir den **Mozilla Java Style Guide**. Er bietet klare und konsistente Konventionen, eine gute Balance zwischen Striktheit und Flexibilität sowie umfassende Dokumentation.

6.7 Kollaborationstool

Die effektivste Kommunikationsform im Projekt sind **persönliche Treffen**, da sie direkte Abstimmung in Echtzeit ermöglichen, Rückfragen sofort klären und den Teamzusammenhalt stärken.

6.8 Versionsverwaltung

Für die Versionskontrolle wird **Git** verwendet. Git ist weit verbreitet, flexibel, unterstützt verteilte Workflows und ist z. B. in GitHub integriert, was das Branching und Merging erleichtert.

6.9 UML-Tools

Zur Erstellung von UML-Diagrammen kommt **PlantUML** zum Einsatz. PlantUML ist Open Source, textbasiert, leicht in Versionskontrollsysteme integrierbar und ermöglicht schnelle Anpassungen.

6.10 Build-Tools

Als Build-Management-Tool verwenden wir **Maven**, da es eine klare Projektstruktur vorgibt, umfangreiche Dokumentation bietet und in gängige IDEs integriert ist.

7 Installations- und Betriebskonzept Installations- und Betriebskonzept

Die Installation erfolgt über Auslieferung eines ausführbaren JARs:

1. Java 11+ installieren.
2. java -jar LiteraBookCatalogue.jar ausführen.

Konfigurationsdateien (z. B. Pfad zu JSON-Daten) liegen in config/. Für Windows wurde zusätzlich eine Batch-Datei erstellt (start.bat).

8 Fortschrittsübersicht

Meilenstein	Soll-Datum	Ist-Status	Kommentar
Projektsetup (Git-Repo, Maven)	15.04.2025	Abgeschlossen	Repository angelegt, POM erzeugt
Grundfunktionalität Suche	22.04.2025	Abgeschlossen	Alle vier Kriterien implementiert
GUI-Prototyp	29.04.2025	Abgeschlossen	Swing-Prototype validiert
Detailansicht & Reviews	05.05.2025	Abgeschlossen	Dialog und Anzeige getestet
Usability-Test & Optimierung	08.05.2025	Abgeschlossen	Verbesserungen eingearbeitet

9 Probleme, Risiken und Lösungsansätze

- **Problem**: JSON-Dateien mit fehlenden Feldern führten zu NullPointerExceptions.
  Lösung: Implementierung von Null-Checks und Default-Werten in StorageService.
- **Risiko**: Unvollständige Anforderungen an Reviews (Rating-Stern-Symbolik).
  Gegenmaßnahme: Standardisierte Stern-Icons gewählt und dokumentiert in Entwicklerdokumentation.
- **Problem**: Lange Ladezeiten bei großem Datenbestand (>5.000 Bücher).
  Lösung: Lazy Loading in BookListPanel implementiert.