Entwicklerdokumentation: Litera Buchkatalog

Bearbeitet von:
Ranim Khallouf
Marija Ratnikova
Rama Alshaer
Lena Heberlein

Stand: 03.06.2025

1. Einleitung

Diese Entwicklerdokumentation beschreibt den Aufbau und die technischen Details der Litera Buchkatalog-Anwendung. Die Anwendung ermöglicht das Durchsuchen eines Buchkatalogs, das Anzeigen von Buchdetails und das Hinzufügen sowie Anzeigen von Rezensionen.

2. Systemübersicht und Architektur

Die Anwendung ist in mehrere Schichten unterteilt:

Benutzeroberfläche (GUI): Realisiert mit Java Swing. Verantwortlich für die Darstellung der Daten und die Interaktion mit dem Benutzer.

Steuerungsschicht (Controller & Interface): Dient als Bindeglied zwischen der GUI und der Geschäftslogik. Das ApplicationInterface (ApplicationInterface.java) definiert die Operationen, die von der GUI aufgerufen werden können, und der Controller (Controller.java) implementiert dieses Interface.

Geschäftslogik (Buchkatalog): Enthält die Kernlogik der Anwendung, wie Suchfunktionen, Filterung und die Verwaltung von Buch- und Rezensionsdaten (Buchkatalog.java).

Datenhaltung (StorageService & JSON): Verantwortlich für das Laden und Speichern der Buch- und Rezensionsdaten aus bzw. in JSON-Dateien.

3. Komponenten im Detail

3.1. Benutzeroberfläche (GUI)

Die GUI ist mit Java Swing implementiert und besteht aus mehreren Panels, die in einem ApplicationWindow mittels CardLayout verwaltet werden.

ApplicationWindow.java: Das Hauptfenster der Anwendung. Initialisiert das CardLayout und die verschiedenen Panels (StartPanel, BookListPanel, BookDetails). Enthält die main-Methode zum Starten der Anwendung, welche den StorageService und den Controller instanziiert und an das Fenster übergibt.

StartPanel.java: Der Willkommensbildschirm. Zeigt ein Logo (images/Logo Weiss.png), den Titel der Anwendung ("Litera"), einen "Start"-Button, der zur Bücherliste (BookListPanel) navigiert und einen „About"-Button, der zu den grundlegenden Informationen über Litera (AboutPanel) führt.

BookListPanel.java: Dieses Panel ist für die Anzeige einer Liste von Büchern zuständig. Es verfügt über einen Kopfbereich mit einem Logo (images/LogoText.png), einem Dropdown-Menü (JComboBox) zur Auswahl der Suchkategorie (Titel, Autor, Verlag, Genre), einem Texteingabefeld (JTextField) für den Suchbegriff und einem "Search"-Button. Bei einer Suche (ausgelöst durch Button-Klick oder Enter im Suchfeld) oder initial werden die Ergebnisse (Buchtitel und Autor) in einer scrollbaren Liste (JScrollPane mit JPanel) angezeigt. Jedes Buchelement in der Liste ist klickbar (Maus-Cursor ändert sich zu Hand) und führt bei Auswahl zur BookDetails-Ansicht des entsprechenden Buches. Die Interaktion zur Abfrage der Bücher erfolgt über den Controller (ctrl.buchsuche(), ctrl.sucheNachAutor(), etc.). Wenn keine Bücher gefunden werden, wird "Nothing found" angezeigt.

BookDetails.java: Zeigt detaillierte Informationen zu einem ausgewählten Buch an, einschließlich Titel, Cover, Autor, Verlag, Genres und Beschreibung. Es lädt und zeigt auch Rezensionen für das aktuelle Buch an und bietet einen "Write review"-Button. Außerdem wird auch eine Auswahl von ähnlichen Büchern angezeigt. Ein "←"-Button ermöglicht die Rückkehr zur BookListPane. Rezensionen werden in einem scrollbaren Bereich angezeigt. Das Hinzufügen einer neuen Rezension erfolgt über ein JOptionPane-Dialogfenster, das nach Bewertung (1-5) und Kommentar fragt. Die Interaktion erfolgt über den Controller (ctrl.showRezensionen(), ctrl.reviewHinzufuegen()).

SimilarBooks.java: Dieses Panel wird in der **BookDetails-Ansicht** eingebettet und zeigt ganz unten eine Liste von Büchern an, die dem aktuell angezeigten Titel ähnlich sind. Beim Initialisieren ruft es über den Controller (ctrl.aehnlich(bookId)) eine Liste von **Buch**-Objekten ab, deren Genres mindestens zu sieben übereinstimmen. Für jedes gefundene Buch erstellt das Panel eine klickbare Komponente (ein JLabel mit Cover-Thumbnail und Buchtitel), die in einem JScrollPane mit FlowLayout angeordnet wird. Ein Klick auf eine Buch-Card löst denselben Navigationsmechanismus aus wie in der **BookListPanel**, sodass über den Controller bzw. das ApplicationWindow die Detailansicht des ausgewählten ähnlichen Buches geöffnet wird. Falls keine ähnlichen Bücher gefunden werden, zeigt das Panel stattdessen eine entsprechende Kurzmeldung („Keine ähnlichen Bücher gefunden") an.

AboutPanel.java: Dieses Panel zeigt grundlegende Informationen über die Firma Litera. Oben befindet sich ein Logo (JLabel mit ImageIcon aus images/LogoText.png) sowie eine Überschrift („Über Litera"). Darunter ist ein mehrzeiliges Textfeld (JTextArea oder JEditorPane in einem JScrollPane), in dem die Historie von Litera, Mission und Kontaktdaten angezeigt werden. Ganz unten gibt es einen „Zurück"-Button (JButton), der per Klick über das CardLayout wieder ins **StartPanel** wechselt. Es gibt keine Datenbank- oder Controller-Interaktion, da das AboutPanel nur statische Texte und Bilder darstellt.

Die Navigation zwischen den Panels erfolgt durch Aufrufe von cl.show(parent, "panelName").

3.2. Steuerung & Schnittstelle

ApplicationInterface.java: Definiert das Interface, das die GUI zur Interaktion mit der Anwendungslogik verwendet. Es entkoppelt die GUI von der konkreten Implementierung der Geschäftslogik.

- Methoden: buchsuche(String q), sucheNachAutor(String q), sucheNachGenre(String q), sucheNachVerlag(String q), buchdetails(String id), showRezensionen(String id), reviewHinzufuegen(String id, Rezension r).

Controller.java: Implementiert das ApplicationInterface. Dient als Fassade zur Buchkatalog-Klasse. Erhält Anfragen von der GUI und leitet sie an den Buchkatalog weiter. Im Konstruktor wird ein StorageService erwartet, der an den Buchkatalog weitergegeben wird.

3.3. Geschäftslogik

Buchkatalog.java: Enthält die Kernlogik der Anwendung.

- Verwaltet eine interne Liste von Buch-Objekten, die beim Start über den StorageService geladen wird.
- Bietet Methoden zur Suche und Filterung von Büchern nach Titel, Autor, Genre und Verlag. Suchanfragen sind nicht case-sensitiv und leere Suchanfragen bei Titel geben alle Bücher zurück, bei anderen Feldern eine leere Liste.
- Ermöglicht das Abrufen von Details zu einem spezifischen Buch anhand seiner ID (buchDetails(String id)).
- Verwaltet das Hinzufügen (reviewHinzufuegen(String id, Rezension rez)) und Anzeigen (showRezensionen(String id)) von Rezensionen. Rezensionen werden über den StorageService geladen und gespeichert.
- Findet ähnliche Bücher anhand des Genres (aehnlich(Buch buch)).

3.4. Datenhaltung

StorageService.java: Verantwortlich für das Lesen und Schreiben von Daten aus/in JSON-Dateien.

- Verwendet die Jackson-Bibliothek (ObjectMapper) für die Konvertierung zwischen Java-Objekten und JSON.
- ladeBuecher(): Lädt Bücher aus books_short.json. Gibt eine leere ArrayList zurück, falls die Datei nicht existiert oder ein Lesefehler auftritt (Fehler wird auf printStackTrace() ausgegeben).
- ladeRezensionen(): Lädt Rezensionen aus reviews.json. Gibt eine leere ArrayList zurück, wenn die Datei fehlt oder nicht lesbar ist (Fehler wird auf printStackTrace() ausgegeben).
- speichereRezension(Rezension r): Lädt bestehende Rezensionen, fügt die neue hinzu und schreibt die gesamte Liste (formatiert mit writerWithDefaultPrettyPrinter()) zurück in reviews.json. Fehler beim Speichern werden via printStackTrace() ausgegeben.
- Dateipfade (BOOKS_FILE = "books_short.json", REVIEWS_FILE = "reviews.json") sind als private statische finale Konstanten definiert.

Modellklassen:

- **Buch.java**: Repräsentiert ein Buch mit Attributen wie bookId, title, author, publisher, genres (Liste von Strings), description und cover. Enthält Getter-Methoden für alle Felder.
- **Rezension.java**: Repräsentiert eine Rezension mit bookId (zur Zuordnung zum Buch), bewertung (int), kommentar (String) und datum (java.util.Date). Enthält Getter und Setter für alle Felder.
- **Autor.java**: Einfache Klasse mit name und biografie (Strings) und zugehörigen Gettern/Settern. Wird im aktuellen Projektfluss nicht direkt zur Speicherung/Ladung von Buchdaten verwendet (Autor ist String in Buch).
- **Verlag.java**: Einfache Klasse mit name (String) und zugehörigen Gettern/Settern. Wird im aktuellen Projektfluss nicht direkt zur Speicherung/Ladung von Buchdaten verwendet (Verlag ist String in Buch).
- **Genre.java**: Einfache Klasse mit name (String) und zugehörigen Gettern/Settern. Wird im aktuellen Projektfluss nicht direkt zur Speicherung/Ladung von Buchdaten verwendet (Genres sind Liste von Strings in Buch).

3.5. JSON Datenformat (books_short.json, reviews.json)

Der StorageService erwartet JSON-Dateien im Projektverzeichnis.

books_short.json (Beispielstruktur basierend auf bereitgestellter Datei):


```
[
 {
  "author": "Suzanne Collins",
  "description": "WINNING MEANS FAME AND FORTUNE.LOSING MEANS CERTAIN DEATH.THE HUNG
ER GAMES HAVE BEGUN. . . .",
  "genres": [
   "Young Adult",
   "Fiction",
   "Dystopia"
  ],
  "publisher": "Scholastic Press",
  "bookId": "2767052-the-hunger-games",
  "title": "The Hunger Games"
  "image": "images/75.jpg

"

 }
 // ... weitere Bucheinträge
]
```


reviews.json (Beispielstruktur basierend auf bereitgestellter Datei):


```
[
 {
  "bookId": "2767052-the-hunger-games",
```

```
    "datum": 1746485532338,
    "bewertung": 4,
    "kommentar": "Sehr Gut"
  }
  // ... weitere Rezensionen
]
```

(Datum als Millisekunden seit Epoche, was von Jackson standardmäßig für java.util.Date verwendet wird)

4. Abhängigkeiten und Build-System (pom.xml)

Das Projekt verwendet Maven als Build-System und zur Verwaltung von Abhängigkeiten. Die Konfiguration befindet sich in der pom.xml.

Java-Version: JDK 11 (gemäß maven.compiler.source und maven.compiler.target in pom.xml).

Hauptabhängigkeiten:

- **Jackson Databind (com.fasterxml.jackson.core:jackson-databind):** Version 2.13.3, für die JSON-Serialisierung und -Deserialisierung.
- **JUnit Jupiter (org.junit.jupiter:junit-jupiter-api, org.junit.jupiter:junit-jupiter-engine):** Version 5.8.2, für Unit-Tests (Scope: test).

Source Directory: In der pom.xml ist <sourceDirectory>${project.basedir}</sourceDirectory> konfiguriert.

- **Build-Plugins:**
  - maven-compiler-plugin (Version 3.8.1) zum Kompilieren des Codes.
  - maven-surefire-plugin (Version 2.22.2) zum Ausführen der Tests.

5. Ausführen

1. **Voraussetzungen:** JDK 11+ und Maven müssen installiert sein.
2. **Ausführen:** Die Anwendung wird über die main-Methode in ApplicationWindow.java gestartet.

6. Fehlerbehandlung

StorageService.java:

- **Datei nicht gefunden / JSON-Formatfehler:** IOException wird abgefangen, eine Fehlermeldung wird auf System.err via ex.printStackTrace() ausgegeben, und eine leere ArrayList wird zurückgegeben. Dies verhindert einen Absturz der Anwendung, wenn die Datendateien fehlen oder korrupt sind.
- **Schreibfehler:** IOException beim Speichern von Rezensionen wird abgefangen und eine Fehlermeldung auf System.err via ex.printStackTrace() ausgegeben.

BookDetails.java (GUI):

- Beim Hinzufügen einer Rezension wird die Bewertungseingabe (1-5) validiert. Bei ungültiger Eingabe (NumberFormatException oder Wert außerhalb des Bereichs) wird ein JOptionPane mit einer Fehlermeldung (“Rating must be 1–5”) angezeigt.

7. Testen

Eine gründliche Teststrategie ist entscheidend für die Stabilität und Zuverlässigkeit der Anwendung. Die Litera Book Catalogue Anwendung wurde mit JUnit 5 getestet. Die folgenden Testbereiche wurden dabei abgedeckt:

Mögliche Testbereiche:

- Einlesen und Schreiben von Buchdaten
  Sicherstellung, dass Bücher korrekt geladen, gespeichert und angezeigt werden.
  → *Getestet mit StorageServiceTest und Beispiel-JSON.*

- Rezensionen hinzufügen und anzeigen
  Überprüfung, ob neue Rezensionen korrekt verarbeitet und dauerhaft gespeichert werden.
  → *Testklasse RezensionTest validiert Inhalte und Zuordnung.*

- Filtern und Suchen
  Funktionalität der Suchfelder und Filteroptionen, z. B. nach Titel oder Genre.
  → *Teilweise manuell getestet in der GUI, automatisierte Tests in BuchTest.*

- GUI-Reaktionen und Ereignisverarbeitung
  Überprüfung der ActionListener und Navigationslogik (z. B. Wechsel zwischen Panels).
  → *Testbar durch manuelle Nutzung der Oberfläche.*

- Fehlerfälle
  Verhalten der Anwendung bei nicht vorhandenen oder beschädigten Daten.
  → *Beispielhafte Simulation über Testdaten mit Fehlern**.***