

„LibraNova“: Full-Stack-Bibliotheksmanagementapp für die Ausleihe von Büchern

”LibraNova“ – A Full-Stack Library Management System for Lending Books

Bearbeiter: Mohammed Salih Mezraoui

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr.-Ing. Georg Schneider

Trier, den 19.08.2025

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Konzeption und Umsetzung von *LibraNova*, einer modernen, webbasierten Full-Stack-Anwendung zur digitalen Verwaltung einer Bibliothek sowie zur effizienten Organisation von Buchausleihen. Ziel des Projekts ist es, sowohl Nutzer:innen als auch Administrator:innen eine intuitive, leistungsfähige und sichere Plattform bereitzustellen, die den gesamten Ausleihprozess digital unterstützt.

Die Anwendung wurde mit **Spring Boot** im Backend und **React** mit Type-Script im Frontend realisiert. Nutzer:innen können Bücher recherchieren, deren Verfügbarkeit in Echtzeit prüfen sowie Rezensionen lesen und verfassen. Administrator:innen erhalten Werkzeuge zur Verwaltung des Buchbestands und zur Kommunikation mit den Nutzer:innen.

Zur Absicherung sensibler Funktionen werden moderne Authentifizierungs- und Autorisierungsverfahren eingesetzt, darunter **JWT** und **OAuth2** via Okta. Für kostenpflichtige Bibliotheksdienste ist die **Stripe API** eingebunden, um sichere Zahlungen zu ermöglichen. Die persistente Datenhaltung erfolgt über die **MySQL-Datenbank**.

Die Anwendung ist in **deutscher und englischer Sprache** verfügbar und legt großen Wert auf **Barrierefreiheit**, um einen breiten Nutzerzugang zu gewährleisten. Im Rahmen der Entwicklung wurde besonderer Wert auf eine responsive Benutzeroberfläche, ein konsistentes UX-Design sowie die Qualitätssicherung durch automatisierte Tests mit **JUnit** und **Mockito** gelegt. *LibraNova* demonstriert, wie durch den Einsatz moderner Webtechnologien eine skalierbare und benutzerfreundliche Lösung für das Bibliotheksmanagement realisiert werden kann.

Abstract

This thesis presents *LibraNova*, a modern, web-based full-stack application designed to facilitate the digital management of a library and the efficient organization of book lending. The project aims to provide both users and administrators with an intuitive, performant, and secure platform that fully supports the lending process digitally.

The application is built using **Spring Boot** for the backend and **React** with TypeScript for the frontend. Users can search for books, check their real-time availability, and read or write reviews. Administrators are equipped with tools to manage the book inventory and communicate with users.

To secure sensitive functions, modern authentication and authorization techniques such as **JWT** and **OAuth2** via Okta are employed. For paid library services, the **Stripe API** ensures secure payment processing. Persistent data storage is handled via the **MySQL database**.

The application supports both **English and German languages** and emphasizes **accessibility** to ensure broad user access. Special attention was given to a responsive user interface, consistent UX design, and quality assurance through automated testing using **JUnit** and **Mockito**. *LibraNova* demonstrates how modern web technologies can be leveraged to create a scalable and user-friendly solution for library management.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Ziele der Arbeit	1
2	Verwandte Arbeiten	3
2.1	Einführung	3
2.2	Existierende Systeme	3
2.2.1	Thalia Website	3
2.2.2	Stadtbücherei Trier	4
2.3	Beitrag von LibraNova	4
2.4	Fazit	5
3	Grundlagen	6
3.1	Back-End Technologien	6
3.1.1	Java	6
3.1.2	Spring Boot	6
3.1.3	Hibernate	7
3.1.4	REST-API	8
3.1.5	Spring Security	8

Inhaltsverzeichnis	v
3.1.6 HTTPS und SSL/TLS	8
3.1.7 Stripe API	9
3.2 Front-End Technologien	9
3.2.1 HTML/CSS	9
3.2.2 Bootstrap	10
3.2.3 TypeScript	10
3.2.4 React	10
3.2.5 i18next	11
3.3 Datenbanktechnologien	11
3.3.1 MySQL	11
3.3.2 MySQL-Workbench und SQL	11
3.4 Authentifizierungs- und Autorisierungsprotokolle	12
3.4.1 Okta	12
3.4.2 JWT	12
3.4.3 OAuth2	12
3.4.4 OpenID Connect	13
3.5 Version Control mit Git und GitHub	13
3.6 Testen und Qualitätssicherung	13
3.6.1 Unit-Tests mit JUnit 5	13
3.6.2 Mocking mit Mockito	14
3.6.3 API-Tests mit Postman	14
3.7 Modellierung mit UML	14
4 Konzept	15
4.1 Blockbild der Architektur	15

4.2 UML-Diagramme zur Konzeption	16
4.2.1 Klassendiagramm	16
4.2.2 Sequenzdiagramme	18
4.3 Anforderungen an das System	20
4.3.1 Funktionale Anforderungen	20
4.3.2 Nicht-funktionale Anforderungen	21
4.3.3 Technische Anforderungen	22
4.4 Wichtige Algorithmen (Pseudocode)	22
4.4.1 Buchausleihe	22
4.4.2 Verlängerung der Buchausleihe	23
5 Realisierung	24
5.1 Backend-Architektur	24
5.1.1 Projektinitialisierung und Startklasse	24
5.1.2 Modulare Paketstruktur des Backends	26
5.1.3 Teststrategie und Testintegration	28
5.2 Frontend-Struktur	29
5.2.1 Einstiegspunkt der React-Anwendung	29
5.2.2 React-Projektstruktur	31
5.2.3 Internationalisierung mit i18next	33
5.3 Datenbankmodell und Persistenz	36
5.3.1 Datenbankschema	36
5.3.2 Datenpersistenz mit Spring Data und Hibernate	38
6 Implementierung	41
6.1 Ausleihprozess eines Buches	41

6.1.1 Backend-Prozess	41
6.1.2 Frontend-Prozess	45
6.2 Verfahren zur Rückgabe eines Buches	47
6.2.1 Backend-Prozess	47
6.2.2 Frontend-Prozess	48
7 Beispiel	51
7.1 Startseite	51
7.1.1 Kopfzeile	51
7.1.2 Footer	52
7.1.3 Karussell und Buch-Browser	52
7.2 Seitenübersicht	53
7.2.1 Buchsuche	53
7.2.2 Buchseite	54
7.3 Bibliotheksaktivität	55
7.3.1 Ausleihen	55
7.3.2 Ausleihhistorie	56
7.4 Bibliotheksdienste	57
7.5 Bezahlungsseite	58
7.6 Admin-Bereich	59
7.6.1 Neues Buch hinzufügen	60
7.6.2 Bücher verwalten	60
7.6.3 Nachrichten	61
8 Anwendungsszenarien	62
8.1 Bildungs- und E-Learning-Plattformen	62

Inhaltsverzeichnis	VIII
8.2 Mobile Nutzung	63
8.3 Personalisierte Empfehlungen	63
8.4 Reservierungsbenachrichtigungen.....	63
9 Resümee und Ausblick	64
Literaturverzeichnis	65
Systemanforderungen	69
Quellcode	73
Glossar	76
Eigenständigkeitserklärung	78

Abbildungsverzeichnis

4.1	Blockbild der Systemarchitektur	15
4.2	Klassendiagramm der Anwendung <i>Libranova</i>	17
4.3	Sequenzdiagramm des Login-Vorgangs, angelehnt an [OT25a]	18
4.4	Zahlungsablauf mit Stripe, angelehnt an [MT25a]	19
5.1	Modularer Aufbau des Backends	27
5.2	Modularer Aufbau des Frontends	31
5.3	ER-Modell der Datenbank	37
7.1	Kopfzeile der App <i>Libranova</i>	51
7.2	Fußzeile der App <i>Libranova</i>	52
7.3	Karussell und Buch-Browser-Schaltfläche	53
7.4	Benutzeroberfläche der Suchseite	54
7.5	Benutzeroberfläche der Buchseite	55
7.6	Benutzeroberfläche der Ausleihseite	56
7.7	Benutzeroberfläche der Ausleihverlaufsseite	57
7.8	Benutzeroberfläche zum Versenden von Anfragen	58
7.9	Nachrichtenverlauf zwischen Nutzer und Bibliothek	58
7.10	Benutzeroberfläche bei ausstehenden Zahlungen	59

7.11 Benutzeroberfläche bei keinen offenen Zahlungen	59
7.12 Das Design zum Hinzufügen eines neuen Buches	60
7.13 Das Design der Buchverwaltung	61
7.14 Die Schnittstelle für die Beantwortung von Anfragen	61

Tabellenverzeichnis

A.1	Funktionale Anforderungen von LibraNova	70
A.2	Nicht-funktionale Anforderungen von LibraNova	71
A.3	Technische Anforderungen von LibraNova	72

Listings

4.1	Pseudocode für den Ausleihvorgang eines Buches	22
4.2	Pseudocode für die Verlängerung einer Buchausleihe	23
5.1	Einstiegspunkt der Spring Boot Anwendung	25
5.2	application.properties Datei	26
5.3	Projektinitialisierung mit Create React App	29
5.4	Einstiegspunkt der React-Anwendung	30
5.5	Frontend-Umgebungsvariable für Backend-Zugriff in .env-Datei	30
5.6	Initialisierung von i18next in <code>i18n.ts</code>	33
5.7	Beispielhafte Nutzung von <code>useTranslation</code>	34
5.8	Englische Übersetzung in <code>en.json</code>	35
5.9	Deutsche Übersetzung in <code>de.json</code>	35
5.10	Sprachumschalter im Header	35
5.11	JPA-Entity <code>Payment</code>	38
5.12	Payments-Repository-Schnittstelle	39
6.1	<code>CheckoutRepository.java</code>	42
6.2	<code>checkoutBook()</code> Methode im <code>BookService.java</code>	42
6.3	Überprüfung der Verfügbarkeit eines Buches	43
6.4	Prüfung auf überfällige Ausleihen	43

6.5	Erstellung einer Nullzahlung mit <code>createZeroPayment()</code>	44
6.6	Reduzierung des Buchbestands	44
6.7	Erstellung eines Ausleihdatensatzes	44
6.8	REST-Endpunkt <code>checkoutBook()</code> im <code>BookController</code>	45
6.9	Implementierung der <code>checkoutBook()</code> -Funktion in <code>CheckoutBook.tsx</code>	45
6.10	<code>returnBook()</code> Endpoint in <code>BookController.java</code>	48
6.11	<code>returnBook</code> -Button in <code>LoanDetailsModal.tsx</code>	49
6.12	<code>returnBook()</code> in <code>Loans.tsx</code>	49
B.1	Prozess des Auscheckens eines Buches im Frontend	73
B.2	Die Geschäftslogik der Rückgabe eines Buches im Backend	74

1

Einleitung

In einer zunehmend digitalen Welt wird die effiziente Verwaltung von Informationen und Ressourcen immer wichtiger. Bibliotheken – als zentrale Einrichtungen der Wissensvermittlung – stehen vor der Herausforderung, ihre Prozesse zu modernisieren und den veränderten Anforderungen der Nutzer:innen gerecht zu werden. Webbasierte Anwendungen bieten großes Potenzial, um den Zugang zu Medien, die Organisation von Ausleihen und die Kommunikation zwischen Nutzer:innen und Verwaltung zeitgemäß und benutzerfreundlich zu gestalten.

Vor diesem Hintergrund wurde die Anwendung *LibraNova* entwickelt. Ziel ist es, eine moderne Full-Stack-Webanwendung bereitzustellen, die Bibliotheksprozesse effizient, sicher und barrierefrei digitalisiert – sowohl für Nutzer:innen als auch Administrator:innen.

Im Folgenden werden die Motivation und die Ziele dieser Arbeit vorgestellt.

1.1 Motivation

LibraNova adressiert genau diese Schwachstellen: Die Anwendung soll zeigen, wie durch moderne Webtechnologien – wie **Spring Boot**, **React** und **TypeScript** – eine skalierbare, intuitive und sichere Lösung für das Bibliotheksmanagement geschaffen werden kann. Besondere Schwerpunkte liegen auf der Mehrsprachigkeit (Deutsch und Englisch), Barrierefreiheit und responsiven Benutzerführung, um eine möglichst breite Zielgruppe anzusprechen und digitale Teilhabe aktiv zu fördern.

1.2 Ziele der Arbeit

Das Ziel dieser Arbeit besteht in der Konzeption, Umsetzung und Dokumentation einer webbasierten Bibliotheksanwendung, die folgende Kernanforderungen erfüllt:

- Bereitstellung einer intuitiven Benutzeroberfläche für die Recherche, Ausleihe und Bewertung von Büchern.
- Ermöglichung einer effizienten Verwaltung des Buchbestands durch Administrator:innen.
- Integration moderner Authentifizierungs- und Autorisierungsmechanismen (**JWT**, **OAuth2** via Okta).
- Einbindung eines sicheren Zahlungssystems über die **Stripe API** für kostenpflichtige Dienste.
- Mehrsprachige Bereitstellung der Anwendung (Deutsch und Englisch).
- Berücksichtigung von Aspekten der Barrierefreiheit zur Förderung inklusiver Nutzung.
- Umsetzung eines **responsiven Designs** zur optimalen Darstellung auf verschiedenen Endgeräten (Desktop, Tablet, Smartphone).
- Sicherstellung von Softwarequalität durch automatisierte Tests mit **JUnit** und **Mockito**.

Darüber hinaus soll die Anwendung modular und erweiterbar gestaltet sein, um zukünftige Funktionalitäten problemlos integrieren zu können. Durch diese Arbeit soll ein Beitrag zur praxisnahen Entwicklung moderner Webanwendungen im Bildungs- und Informationsbereich geleistet werden.

2

Verwandte Arbeiten

In diesem Kapitel werden ausgewählte bestehende Systeme untersucht, die ähnliche Funktionen wie LibraNova bereitstellen. Ziel ist es, deren Stärken und Schwächen zu analysieren, um die Positionierung und den Mehrwert der eigenen Anwendung im Vergleich zum aktuellen Stand der Technik (*State-of-the-Art*) zu verdeutlichen.

2.1 Einführung

Die Analyse bestehender Systeme ist ein zentraler Bestandteil wissenschaftlicher Arbeiten, da sie den aktuellen Stand der Technik (*State-of-the-Art*) verdeutlicht. Durch den Vergleich mit etablierten Lösungen lassen sich Stärken, Schwächen und bestehende Lücken identifizieren. Auf dieser Grundlage können die eigenen Beiträge klar positioniert und die Motivation für die Entwicklung eines neuen Systems nachvollziehbar begründet werden.

2.2 Existierende Systeme

Dieser Abschnitt untersucht ausgewählte bestehende Systeme, die ähnliche Funktionen wie LibraNova bieten. Ziel ist es, deren Merkmale, Stärken und Schwächen darzustellen und aufzuzeigen, wie sich die eigene Anwendung im Vergleich zum aktuellen Stand der Technik (*State-of-the-Art*) positioniert.

2.2.1 Thalia Website

Beschreibung: Thalia ist ein führender Buchhändler im deutschsprachigen Raum mit einem stetig wachsenden Netz aus derzeit über 500 Filialen in Deutschland, Österreich und der Schweiz. Neben dem stationären Handel betreibt Thalia einen

umfangreichen Online-Shop und baut seine Präsenz im E-Commerce kontinuierlich aus [Gmb25a].

Funktionen: Neben dem Online-Shop bietet Thalia mit der *Lesen & Hören* App eine mobile Lösung zum Lesen und Hören von eBooks und Hörbüchern. Nutzer können ihre Titel in der tolino-Cloud speichern, offline lesen, Lesefortschritte automatisch synchronisieren sowie Schriftart, -größe und Layout individuell anpassen. Zusätzliche Funktionen wie Sammlungen und ein Nachtmodus erhöhen den Lesekomfort [Gmb25c].

Sprachverfügbarkeit: Obwohl Thalia ein breites Sortiment an Büchern in verschiedenen Sprachen (z. B. Englisch, Französisch, Spanisch) anbietet, ist die Website-Oberfläche ausschließlich auf Deutsch verfügbar. Auch manche Buchbeschreibungen sind nur auf Deutsch verfügbar, was die Navigation und Informationsbeschaffung für nicht-deutschsprachige Nutzer zusätzlich erschwert und die allgemeine Nutzerfreundlichkeit beim Online-Browsing reduziert [Gmb25b].

2.2.2 Stadtbücherei Trier

Überblick: Die Stadtbücherei Trier stellt mehr als 90.000 Medien zur Verfügung, einschließlich Bücher, Hörbücher, Musik-CDs sowie Computer- und Konsolenspiele, die sowohl vor Ort genutzt als auch ausgeliehen werden können. Auf fünf lichtdurchfluteten Etagen stehen Sitz- und Lernbereiche, PC-Arbeitsplätze, Drucker und ausleihbare Laptops zur Verfügung. Informationen zum Medienangebot sind über den Onlinekatalog verfügbar (<https://opac.trier.de/>) [Tri25].

OPAC-System: Das OPAC-System der Stadtbücherei Trier ist keine Single-Page Application (SPA). Eine Analyse mit den Browser-Entwicklertools zeigt, dass bei jeder Navigation innerhalb des OPAC – selbst beim Anwenden einfacher Filter wie der Suche nach Medien eines bestimmten Jahres – die gesamte HTML-Seite vollständig neu geladen wird, anstatt Inhalte dynamisch nachzuladen, wie es bei einer echten SPA der Fall wäre. Auch die Hauptwebsite (<https://www.stadtbumcherei-trier.de/startseite/>) ist nicht vollständig responsiv, was die Nutzerfreundlichkeit insbesondere auf mobilen Geräten einschränkt.

2.3 Beitrag von LibraNova

LibraNova adressiert die identifizierten Schwächen bestehender Systeme auf mehreren Ebenen:

- **Bilinguale Verfügbarkeit:** Die Anwendung ist vollständig auf Deutsch und Englisch verfügbar, was die Nutzerbasis deutlich erweitert und die

Zugänglichkeit für internationale oder nicht-deutschsprachige Nutzer verbessert.

- **Single-Page Application (SPA):** Dank der Verwendung von React als Frontend-Framework verhält sich LibraNova als SPA. Inhalte werden dynamisch nachgeladen, wodurch bei Navigation und Filteranwendung kein vollständiger Seiten-Reload erfolgt [Pur25]. Dies erhöht die Effizienz und verbessert die Nutzererfahrung.
- **Responsives Design:** Die gesamte Anwendung ist vollständig responsiv gestaltet, wodurch alle Funktionen auf Desktop-, Tablet- und Mobilgeräten optimal genutzt werden können.

Darüber hinaus greift LibraNova auch bewährte Konzepte bestehender Systeme auf: Von Thalia wurde das grundlegende Layout der Buchdetailseite (Titel, Autor, Beschreibung, Sterne-Bewertung) übernommen, während von der Stadtbücherei Trier die Idee einer mehrsprachigen Suchplattform inspiriert wurde. Diese Stärken wurden gezielt übernommen.

2.4 Fazit

Die Analyse bestehender Systeme zeigt, dass Thalia und die Stadtbücherei Trier jeweils Stärken besitzen, jedoch auch wesentliche Einschränkungen aufweisen. So ist beispielsweise Thalia ausschließlich auf Deutsch verfügbar, während das OPAC-System der Stadtbücherei Trier keine Single-Page Application ist und die Hauptwebsite nicht vollständig responsiv ist. LibraNova positioniert sich im State-of-the-Art, indem es diese Schwächen adressiert: Die Anwendung ist vollständig bilingual (Deutsch und Englisch), vollständig responsiv und als SPA umgesetzt, sodass Inhalte dynamisch nachgeladen werden. Auf diese Weise werden bekannte Konzepte bestehender Plattformen übernommen, jedoch entscheidend verbessert, was die Notwendigkeit und den Mehrwert der eigenen Entwicklung von LibraNova verdeutlicht.

3

Grundlagen

In diesem Kapitel werden die zentralen Technologien vorgestellt, die für die Konzeption und Entwicklung der Bibliotheksanwendung *LibraNova* von Bedeutung sind.

3.1 Back-End Technologien

In diesem Abschnitt werden die eingesetzten Back-End-Technologien vorgestellt, wobei der Fokus auf Spring Boot liegt – dem zentralen Framework zur Implementierung einer robusten, skalierbaren und sicheren serverseitigen Logik sowie RESTful APIs für die Anwendung *LibraNova*.

3.1.1 Java

Java ist eine weit verbreitete, objektorientierte Programmiersprache, die für ihre Plattformunabhängigkeit, Stabilität und umfangreichen Standardbibliotheken bekannt ist. Aufgrund ihrer Vielseitigkeit und Leistungsfähigkeit wird sie häufig für die Entwicklung von Back-End-Anwendungen eingesetzt. In der Bibliotheksanwendung *LibraNova* wird Java in der Version 17 verwendet, um eine robuste, wartbare und skalierbare serverseitige Logik bereitzustellen [Ora25].

3.1.2 Spring Boot

Spring Boot ist ein Framework, das auf dem Spring Framework aufbaut und speziell entwickelt wurde, um schnelle, effiziente und skalierbare Anwendungen zu erstellen. Es bietet eine Vielzahl von Features, die den Entwicklungsprozess beschleunigen und vereinfachen, insbesondere für Java-basierte Webanwendungen [Spr25a].

Gründe für die Wahl von Spring Boot

Spring Boot wurde für *LibraNova* gewählt, da es die Entwicklung serverseitiger Anwendungen deutlich vereinfacht und beschleunigt. Entscheidende Vorteile sind:

- **Schneller Entwicklungsstart:** Vorkonfigurierte Abhängigkeiten und Best Practices ermöglichen sofortiges Beginnen der Entwicklung.
- **Einfache Bereitstellung:** Eigenständig ausführbare Anwendungen mit eingebautem Tomcat-Server benötigen keinen externen Applikationsserver.
- **Automatische Konfiguration:** Frameworks und Bibliotheken werden automatisch erkannt, was den Konfigurationsaufwand reduziert.
- **Produktionsreife Funktionen:** Integrierte Features wie Konfigurationsdateien, Monitoring und Health Checks vereinfachen den produktiven Betrieb [Spr25b].

Verwendete Abhängigkeiten

- **Spring Data REST:** Ermöglicht die automatische Bereitstellung von RESTful Webservices auf Basis von Spring-Data-Repositories. Es erstellt durchsuchbare Endpunkte, unterstützt HAL, Paginierung, Sortierung und Filterung, ganz ohne manuelle Controller [ST25b].
- **Spring Data JPA:** Vereinfacht die Implementierung von Repository-Schichten auf Basis der Java Persistence API. Entwickler definieren Interfaces, während Spring die Implementierung liefert. Unterstützt abgeleitete Suchmethoden, Paginierung, dynamische Abfragen und Auditierung von Entitäten [ST25a].

3.1.3 Hibernate

Hibernate ist ein typensicheres Java-ORM-Framework, das automatische Abbildung zwischen Objekten und relationalen Datenbanken ermöglicht, JPA unterstützt, komplexe Abfragen, Transaktionen, Caching und Performance-Optimierung bietet. Es fördert idiomatische Java-Persistenz, HQL-Abfragen, Datenbankkompatibilität, Skalierbarkeit und Entwicklerproduktivität, reduziert Boilerplate-Code, gewährleistet ACID-Eigenschaften und vereinfacht die Entwicklung von Enterprise-Anwendungen [HT25].

3.1.4 REST-API

REST (Representational State Transfer) ist ein Architekturstil für verteilte Hypermedia-Systeme, der von Roy Fielding im Jahr 2000 in seiner Dissertation vorgestellt wurde. Seitdem hat sich REST als eine der am weitesten verbreiteten Methoden zur Entwicklung von webbasierten APIs etabliert [Gup25]. Es definiert zentrale Prinzipien, die ein Webservice erfüllen muss, um als RESTful zu gelten:

- **Uniform Interface:** Ressourcen werden über eine einheitliche Schnittstelle eindeutig identifiziert und mittels standardisierter HTTP-Methoden (GET, POST, PUT, DELETE) manipuliert. Nachrichten sind selbstbeschreibend und können Hypermedia-Links enthalten.
- **Stateless:** Jeder Client-Request enthält alle notwendigen Informationen; der Server speichert keinen Sitzungszustand.
- **Client-Server-Architektur:** Trennung von Benutzeroberfläche und Datenhaltung ermöglicht unabhängige Weiterentwicklung und bessere Skalierbarkeit [Gup25]

Im *LibraNova*-Projekt wird das Backend mit Spring Boot umgesetzt, um RESTful-Endpunkte für die Verwaltung verschiedener Ressourcen bereitzustellen. Die API-Pfade, beispielsweise `/api/books`, sind klar strukturiert, um einen konsistenten Zugriff auf die Daten zu gewährleisten.

3.1.5 Spring Security

Spring Security ist ein anpassbares Framework zur Verwaltung von Authentifizierung und Autorisierung in Java-Anwendungen. Es bietet wiederverwendbare Module, Schutz gegen Web-Schwachstellen wie CSRF und flexible Integrationsmöglichkeiten für unterschiedliche Anwendungsfälle. Unsachgemäße Konfiguration kann jedoch Sicherheitsrisiken bergen [IRM⁺20].

Im *LibraNova*-Projekt wird Spring Security eingesetzt, um die REST-API-Endpunkte abzusichern und den Zugriff zu kontrollieren.

3.1.6 HTTPS und SSL/TLS

HTTPS sichert die Kommunikation zwischen Browser und Webserver durch TLS-Verschlüsselung (früher SSL). Öffentlicher Schlüssel verschlüsselt Daten, die nur

der private Schlüssel auf dem Server entschlüsseln kann. So werden sensible Informationen geschützt, und die Verbindung gewährleistet Vertraulichkeit, Integrität und Authentizität. Standardmäßig erfolgt die Kommunikation über Port 443 [Clo25].

In *LibraNova* laufen sowohl das Backend als auch das Frontend über HTTPS in ihren jeweiligen Ports unter Verwendung eines selbstsignierten Zertifikats.

3.1.7 Stripe API

Die Stripe API ermöglicht die nahtlose Integration von Zahlungsfunktionen in Anwendungen. Sie orientiert sich an REST-Prinzipien, verwendet ressourcenorientierte URLs, überträgt Anfragen im Formularformat und liefert Antworten im JSON-Format. Die Authentifizierung erfolgt über API-Schlüssel, und die API unterstützt sowohl Test- als auch Echtzeitmodi. Sie bietet umfassende Funktionen für Zahlungsabwicklung, Kundenverwaltung und Abo-Management sowie Zugriff auf das gesamte Stripe-Portfolio, einschließlich *Payments*, *Billing*, *Connect*, *Terminal*, *Issuing* und *Treasury*, wodurch individuelle Zahlungsprozesse und Automatisierungen effizient umgesetzt werden können [ST25d].

In *LibraNova* wurde Stripe eingesetzt, um eine moderne und sichere Zahlungsabwicklung zu ermöglichen.

3.2 Front-End Technologien

Dieser Abschnitt befasst sich mit den verwendeten Frontend-Technologien, darunter HTML, CSS, Bootstrap, TypeScript, React und i18next. Sie ermöglichen gemeinsam eine moderne, responsive und benutzerfreundliche Oberfläche für die Anwendung sowie eine effiziente Umsetzung der Internationalisierung.

3.2.1 HTML/CSS

HTML und CSS bilden die Grundlage für die Entwicklung und Gestaltung von Webseiten. HTML definiert die Struktur und den Inhalt der Seite, während CSS für das visuelle Erscheinungsbild zuständig ist – einschließlich Layout, Farben und Schriftarten. Gemeinsam ermöglichen sie die Erstellung ansprechender und übersichtlich strukturierter Benutzeroberflächen [HTM25a, HTM25b].

3.2.2 Bootstrap

Bootstrap ist ein weit verbreitetes, quelloffenes Framework zur Entwicklung responsiver und mobiler Webanwendungen. Es stellt eine Vielzahl vordefinierter CSS-Klassen sowie JavaScript-Komponenten bereit, die eine schnelle und konsistente Gestaltung von Benutzeroberflächen ermöglichen [Boo25].

In *LibraNova* wurde Bootstrap eingesetzt, um das Layout flexibel zu gestalten und sicherzustellen, dass sich die Benutzeroberfläche auf verschiedenen Bildschirmgrößen (z. B. Desktop, Tablet, Smartphone) dynamisch anpasst.

3.2.3 TypeScript

TypeScript wurde von Microsoft entwickelt und ist eine Programmiersprache, die JavaScript erweitert und optionale statische Typisierung sowie moderne Sprachfunktionen bietet. Sie ermöglicht es Entwickler:innen, viele häufige Fehler bereits während der Entwicklungsphase zu erkennen, wobei das ursprüngliche Laufzeitverhalten von JavaScript erhalten bleibt. Für die Entwicklung der React-basierten Benutzeroberfläche wurde bewusst TypeScript anstelle von reinem JavaScript gewählt. Die Gründe dafür liegen in der besseren Code-Wartbarkeit, der verbesserten Autovervollständigung in modernen IDEs und der höheren Typsicherheit, die gerade in größeren Anwendungen wie einem Bibliotheksverwaltungssystem eine zentrale Rolle spielt [Cor25a, Bra25].

3.2.4 React

React ist eine JavaScript-Bibliothek zum Erstellen von Benutzeroberflächen, indem kleine, wiederverwendbare Komponenten wie Buttons und Text kombiniert werden. Es verwendet einen deklarativen Ansatz, der nur die Teile der UI effizient aktualisiert, die sich ändern, wodurch der Code leichter verständlich und leichter zu debuggen ist. Komponenten kapseln ihre eigene Logik und ihren Zustand und können zu komplexen Oberflächen zusammengesetzt werden. Da die Komponenten in JavaScript geschrieben sind, ermöglichen sie einen reibungslosen Datenfluss ohne direkte Abhängigkeit vom DOM. Zur besseren Strukturierung können Komponenten in separate Dateien ausgelagert und bei Bedarf importiert werden. React ist besonders beliebt für die Entwicklung von Single-Page-Anwendungen (SPAs), die schnelle und nahtlose Nutzererlebnisse ohne vollständiges Neuladen der Seite bieten [MP25b, MP25a, MP25c].

Die Kombination von React im Frontend mit Spring Boot im Backend ist weit verbreitet, da Spring Boot RESTful APIs bereitstellt, die React über HTTP-Anfragen konsumieren kann. Diese Trennung von Frontend und Backend unterstützt eine

klare Architektur, fördert Skalierbarkeit und erleichtert die Wartung. Über REST APIs können dynamisch Daten wie Bücher, Benutzerinformationen oder Ausleihvorgänge in Echtzeit geladen und aktualisiert werden, was für eine Bibliotheksanwendung essenziell ist [Pur25].

3.2.5 i18next

i18next ist ein umfassendes Internationalisierungs-Framework für JavaScript, das Web-, Mobile- und Desktop-Plattformen unterstützt. Es bietet Funktionen wie automatische Spracherkennung, flexibles Laden von Übersetzungen und Erweiterbarkeit durch Plugins. Kompatibel mit wichtigen Frontend-Frameworks, ist es auf Skalierbarkeit und Benutzerfreundlichkeit ausgelegt. Diese Bibliothek wurde für *LibraNova* aufgrund ihrer Popularität und ihres robusten Ökosystems ausgewählt [i1825].

3.3 Datenbanktechnologien

In diesem Abschnitt werden die wichtigsten Datenbanktechnologien erläutert, die für die Implementierung der *LibraNova*-Anwendung verwendet wurden. Im Fokus stehen dabei MySQL als Datenbanksystem, MySQL Workbench als grafisches Verwaltungswerkzeug sowie SQL als zugrunde liegende Abfragesprache.

3.3.1 MySQL

MySQL ist ein weit verbreitetes Open-Source-Datenbankmanagementsystem, das Daten in relationalen Tabellen speichert und über SQL zugänglich macht. Es bietet hohe Geschwindigkeit, Zuverlässigkeit und Skalierbarkeit [Cor25c].

3.3.2 MySQL-Workbench und SQL

MySQL Workbench ist ein grafisches Werkzeug zur Modellierung, Verwaltung und Migration von MySQL-Datenbanken. Es bietet Funktionen wie visuelles Datenbankdesign, Ausführung von SQL-Abfragen, Serveradministration sowie Datenmigration aus anderen Datenbanksystemen [Cor25b].

SQL entwickelt von Don Chamberlin und Ray Boyce bei IBM, ist eine standardisierte Sprache zur Verwaltung relationaler Datenbanken. Sie ermöglicht das Einfügen, Abfragen, Aktualisieren und Löschen von Daten mit wenigen Befehlen.

Durch JOIN-Operationen können Daten aus mehreren Tabellen effizient verknüpft und Redundanzen vermieden werden. SQL ist ein Kernbestandteil moderner datengetriebener Anwendungen und kommt in nahezu allen relationalen Datenbanksystemen zum Einsatz [IBM25].

3.4 Authentifizierungs- und Autorisierungsprotokolle

Sichere Benutzerverwaltung und Zugriffskontrolle sind für Webanwendungen entscheidend. Technologien wie Okta, JWT, OAuth2 und OpenID Connect sorgen für standardisierte Authentifizierung und Autorisierung. Im Folgenden wird ihre Nutzung in der Anwendung beschrieben.

3.4.1 Okta

Okta ist ein cloudbasierter Identitätsdienst, der sicheren Zugriff auf Anwendungen und Geräte ermöglicht. Es bietet Single Sign-On (SSO), Multi-Faktor-Authentifizierung (MFA) und Integration mit lokalen Verzeichnissen wie Active Directory. Okta erleichtert das Identitäts- und Zugriffsmanagement über verschiedene Systeme hinweg [OT25b]. Die Okta-Integration wird verwendet, um die sichere Benutzerverwaltung auszulagern, einschließlich Passwortverwaltung, Token-Ausgabe und rollenbasierter Zugriffskontrolle. Dadurch wird ein standardisierter Authentifizierungsprozess implementiert, der verschiedene Benutzerrollen unterstützt und Berechtigungen direkt in Okta verwaltet.

3.4.2 JWT

JWT ist ein offener Standard zur sicheren Übertragung von Informationen als signiertes JSON-Objekt. Es wird hauptsächlich für die Autorisierung genutzt, indem es nach der Anmeldung den Zugriff auf geschützte Ressourcen ermöglicht. Außerdem gewährleistet JWT die Integrität und Authentizität der übertragenen Daten [JT25b]. In der Anwendung sichern Okta-JWTs die API-Endpunkte. Spring Security ist als OAuth2-Resource-Server konfiguriert und validiert die Tokens, die Benutzer-Claims wie „userType“ enthalten, um rollenbasierte Zugriffssteuerung zu ermöglichen. So wird eine sichere, tokenbasierte Authentifizierung und feingranulare Autorisierung im Backend gewährleistet.

3.4.3 OAuth2

OAuth 2.0 ist ein Sicherheitsstandard, der Drittanbieteranwendungen ermöglicht, im Namen von Nutzern auf Ressourcen zuzugreifen, ohne deren Anmeldedaten

preiszugeben. Es basiert auf dem Austausch von Zugriffstokens, was die Sicherheit erhöht [RHHH20]. In der Anwendung wird OAuth 2.0 zusammen mit Okta genutzt, um JWTs auszustellen und zu validieren. Diese Tokens autorisieren den Zugriff auf geschützte Backend-Ressourcen, wodurch eine sichere und rollenbasierte Zugriffskontrolle gewährleistet wird.

3.4.4 OpenID Connect

OpenID Connect ist ein Authentifizierungsprotokoll, das auf OAuth 2.0 basiert und die sichere Überprüfung der Benutzeridentität ermöglicht. Es liefert standardisierte Nutzerinformationen und unterstützt eine einfache Integration in verschiedene Anwendungen [OT25c]. Die Anwendung nutzt OpenID Connect mit Okta zur sicheren Anmeldung und zur Verwaltung von Benutzerrollen und Zugriffsrechten.

3.5 Version Control mit Git und GitHub

Git ist ein kostenloses, verteiltes Versionskontrollsystem, das durch hohe Geschwindigkeit, effizientes Branch-Management und flexible Arbeitsabläufe besticht. Es ermöglicht die einfache Verwaltung von Projekten jeder Größe und unterstützt parallele Entwicklungsprozesse durch lokale Branches [GT25a].

GitHub ergänzt Git als cloudbasierte Plattform zum Speichern, Teilen und gemeinsamen Entwickeln von Code. Es erleichtert das Nachverfolgen von Änderungen, die Code-Review durch andere Entwickler sowie die koordinierte Zusammenarbeit, ohne unbeabsichtigte Auswirkungen auf den Hauptzweig zu riskieren [GT25b]. Beide Werkzeuge haben maßgeblich zur effizienten Verwaltung und Versionskontrolle des Codes im Verlauf dieses Projekts beigetragen.

3.6 Testen und Qualitätssicherung

Zur Sicherstellung der Softwarequalität wurden im Projekt Unit-Tests, Mocking und API-Tests eingesetzt, um Funktionen, API-Endpunkte und Abhängigkeiten zuverlässig zu überprüfen.

3.6.1 Unit-Tests mit JUnit 5

JUnit 5 ist ein modernes Testframework für Java, das aus mehreren Modulen besteht: der JUnit Platform, JUnit Jupiter und JUnit Vintage. Die JUnit Platform

bildet die Basis zur Ausführung von Tests auf der JVM und ermöglicht die Integration verschiedener Testengines. JUnit Jupiter bietet die Programmierschnittstelle und Erweiterungsmöglichkeiten für das Schreiben und Ausführen neuer Tests, während JUnit Vintage die Kompatibilität zu älteren JUnit-Versionen sicherstellt. JUnit 5 setzt mindestens Java 8 voraus und wird von gängigen Entwicklungsumgebungen und Build-Tools wie IntelliJ IDEA, Maven und Gradle unterstützt. Die modulare Architektur erleichtert sowohl das Schreiben als auch das Ausführen von Tests in modernen Java-Projekten [JT25a].

3.6.2 Mocking mit Mockito

Mockito ist ein weit verbreitetes Mocking-Framework für Java, das speziell für das Schreiben von klaren und leicht lesbaren Unit-Tests entwickelt wurde. Es ermöglicht das einfache Erstellen von Scheinobjekten (Mocks), mit denen sich das Verhalten von Abhängigkeiten gezielt simulieren und testen lässt. Dank seiner übersichtlichen API trägt Mockito zu einer besseren Verständlichkeit der Tests und zu nachvollziehbaren Fehlermeldungen bei. Es wird von der Entwicklergemeinschaft intensiv genutzt und zählt zu den beliebtesten Bibliotheken im Java-Ökosystem [MT25b].

3.6.3 API-Tests mit Postman

Postman ist eine umfassende Plattform für die Arbeit mit APIs und unterstützt den gesamten Lebenszyklus – von der Planung über das Testen bis hin zur Bereitstellung und Überwachung. Mit einer intuitiven Oberfläche und zahlreichen Funktionen ermöglicht Postman das Speichern, Dokumentieren und Testen von API-Endpunkten in einem zentralen Repository. Es bietet Werkzeuge zur Spezifikation, Mock-Erstellung und Automatisierung von Tests, wodurch die Entwicklung beschleunigt und die Zusammenarbeit im Team vereinfacht wird [PT25].

3.7 Modellierung mit UML

UML ist ein wesentliches Werkzeug im Bereich Entwurf und ist eine standardisierte Modellierungssprache, die eine Vielzahl von Diagrammen bietet, um unterschiedliche Aspekte eines Systems darzustellen. Diese Diagramme helfen dabei, komplexe Systeme zu visualisieren, zu dokumentieren und zu kommunizieren [Bel25]. Eine Auswahl relevanter Diagramme zur Konzeption der Webanwendung wird im folgenden Kapitel vorgestellt, um einen Überblick über den strukturellen Aufbau und das Zusammenspiel der Systemkomponenten zu geben.

4

Konzept

Dieses Kapitel beschreibt das grundlegende Konzept der Anwendung *Libranova*. Es erläutert zunächst die Systemarchitektur und die wichtigsten technischen Komponenten, bevor zentrale Abläufe anhand von UML-Diagrammen veranschaulicht werden. Anschließend werden die funktionalen, nicht-funktionalen und technischen Anforderungen dargestellt, gefolgt von der Beschreibung wesentlicher Algorithmen in Pseudocode. Ziel ist es, sowohl die logische Struktur als auch die Interaktion der einzelnen Systemteile transparent und nachvollziehbar darzustellen.

4.1 Blockbild der Architektur

Das Gesamtsystem basiert auf einer klassischen Client-Server-Architektur. Die Webanwendung besteht aus zwei Hauptkomponenten: dem Frontend, implementiert mit React, und dem Backend, realisiert mit Spring Boot. Die Kommunikation erfolgt über eine REST-API. Persistente Daten werden in einer MySQL-Datenbank gespeichert.

Abbildung 4.1 zeigt ein Blockdiagramm der Systemarchitektur von *Libranova*. Es veranschaulicht die Aufteilung der Anwendung in verschiedene logische Schichten und deren Zusammenspiel:

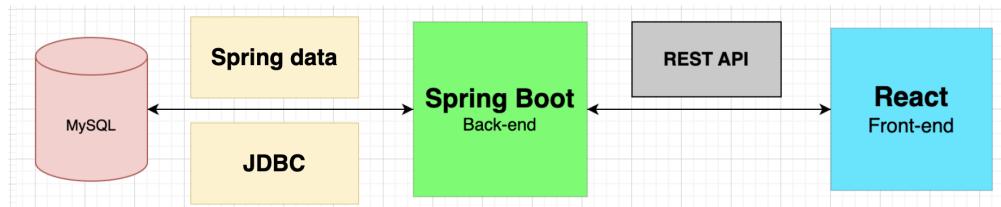


Abbildung 4.1: Blockbild der Systemarchitektur

- **React Frontend:** Die Benutzeroberfläche wurde mit React umgesetzt. Sie bietet eine moderne, komponentenbasierte Nutzererfahrung. Benutzeraktionen wie

Anmeldung, Buchsuche oder Ausleihe werden hier initiiert und über HTTP-Anfragen (im JSON-Format) an die REST-API weitergeleitet.

- **REST API:** Die Schnittstelle zwischen Frontend und Backend folgt dem REST-Architekturstil. Die Kommunikation erfolgt über klar definierte Endpunkte unter Verwendung der HTTP-Methoden **GET**, **POST**, **PUT** und **DELETE**.
- **Spring Boot Backend:** Diese Schicht verarbeitet eingehende API-Anfragen, übernimmt die Geschäftslogik, führt Validierungen durch und steuert Datenbankzugriffe. Die Benutzerverwaltung und Zugriffskontrolle erfolgt hier über die Integration mit Okta.
- **Spring Data JPA, Spring Data REST und JDBC:** Für den Datenbankzugriff wird Spring Data JPA verwendet, das eine deklarative und effiziente Abfrageerstellung über Repository-Interfaces ermöglicht. Über Spring Data REST werden ausgewählte Repository-Methoden automatisch als REST-Endpunkte bereitgestellt. Intern erfolgt die Kommunikation mit der MySQL-Datenbank über JDBC als Treiberschicht.
- **MySQL Database:** Relationale Datenbank zur Speicherung persistenter Daten, strukturiert durch Entitäten wie **Buch**, **Rezension** und **Historie**.

4.2 UML-Diagramme zur Konzeption

Zur Visualisierung des Systemkonzepts werden im Folgenden zentrale UML-Diagramme vorgestellt.

4.2.1 Klassendiagramm

Die Abbildung 4.2 zeigt das Klassendiagramm der Anwendung *LibraNova*. Das Diagramm gliedert sich in vier Hauptbereiche: Bibliotheksbereich, Zahlungssystem, Kommunikation und Benutzerverwaltung.

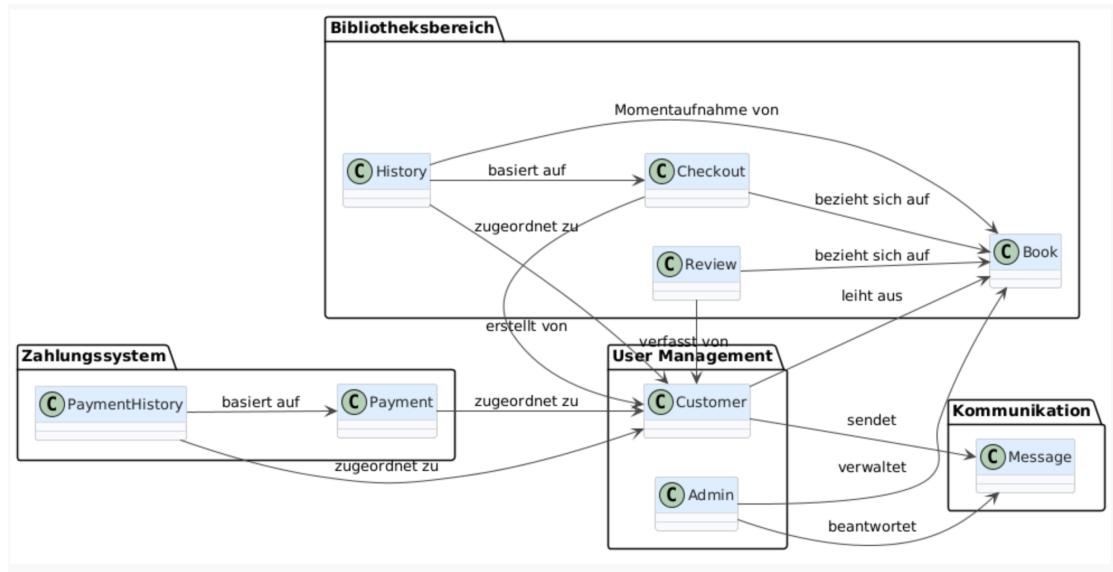


Abbildung 4.2: Klassendiagramm der Anwendung *Libranova*

Im Bibliotheksbereich repräsentiert die Klasse **Book** zentrale Buchinformationen. **Checkout** modelliert aktive Ausleihen, während **History** abgeschlossene Ausleihen als Momentaufnahme speichert und auf den **Checkout**-Vorgängen basiert. Nutzerbewertungen werden über die Klasse **Review** abgebildet, die einem Buch zugeordnet und von einem **Customer** erstellt wird.

Das Zahlungssystem besteht aus den Klassen **Payment** und **PaymentHistory**. **Payment** speichert einzelne Zahlungsinformationen, während **PaymentHistory** auf diesen Zahlungen aufbaut und dem Nutzer einen Überblick über seine bisherigen Transaktionen bietet.

Die Kommunikation zwischen Nutzern und Administratoren wird durch die Klasse **Message** modelliert. Kunden können Nachrichten an Administratoren senden, die diese beantworten.

Im Bereich Benutzerverwaltung werden die Rollen **Customer** und **Admin** dargestellt. **Admin** verwaltet Bücher und beantwortet Nachrichten, während **Customer** Bücher ausleiht, bewertet, Nachrichten versendet und Zahlungen tätigt. Die Klassen sind eng über Beziehungen verknüpft: **Review** und **Checkout** beziehen sich auf **Book** und **Customer**, **History** basiert auf **Checkout**-Daten, und **PaymentHistory** baut auf **Payment**-Vorgängen auf.

Das Diagramm verdeutlicht die logische Struktur der Anwendung, die Verantwortlichkeiten der Klassen und die Interaktionen zwischen den Systemkomponenten.

4.2.2 Sequenzdiagramme

Abbildung 4.3 zeigt das Sequenzdiagramm des Benutzer-Login-Vorgangs innerhalb der Anwendung. Ziel dieses Prozesses ist die sichere Authentifizierung des Benutzers sowie die Bereitstellung eines Access Tokens, das den Zugriff auf geschützte Ressourcen ermöglicht.

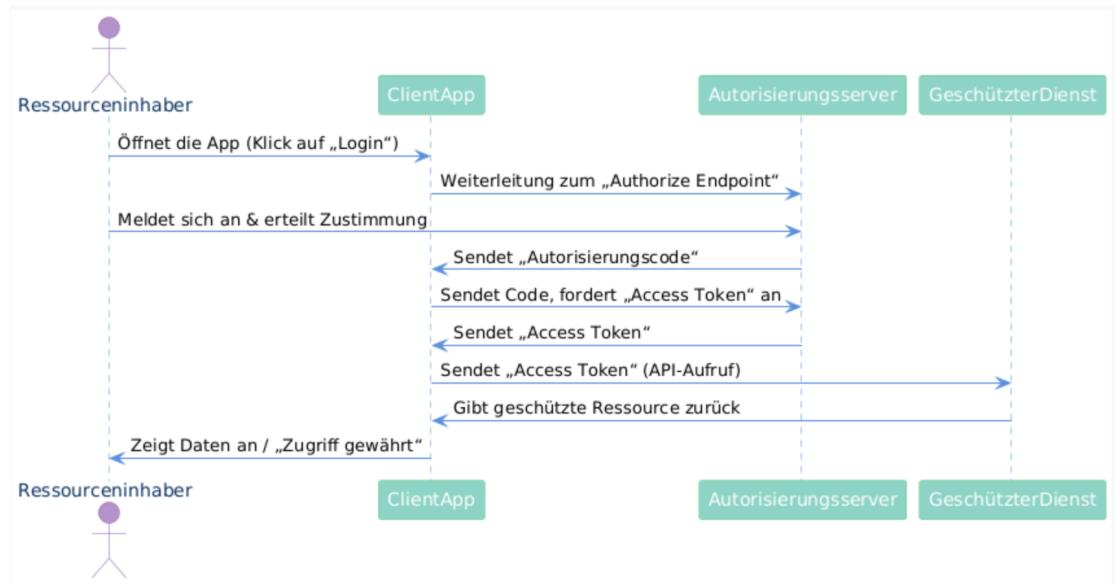


Abbildung 4.3: Sequenzdiagramm des Login-Vorgangs, angelehnt an [OT25a]

Im Login-Prozess sind mehrere Komponenten beteiligt. Der Ressourceninhaber (Benutzer) startet den Anmeldevorgang und erteilt die Zustimmung zur Autorisierung. Die Client-Anwendung (React) fungiert als Schnittstelle zwischen Benutzer und System. Der Autorisierungsserver (Okta) authentifiziert den Benutzer, stellt einen Autorisierungscode aus und tauscht diesen gegen ein Access Token ein. Schließlich überprüft der Ressourcenserver (Spring Boot) die Gültigkeit des Tokens und stellt bei erfolgreicher Authentifizierung die geschützten Ressourcen bereit.

Der Ablauf gliedert sich in folgende Schritte: Zunächst öffnet der Benutzer die React-Anwendung und startet den Login-Prozess. Die Client-Anwendung leitet den Benutzer an den Autorisierungsserver weiter, wo sich der Benutzer anmeldet und der Autorisierung zustimmt. Anschließend sendet der Autorisierungsserver einen Autorisierungscode an die Client-Anwendung, welcher gegen ein Access Token eingetauscht wird. Dieses Token wird lokal gespeichert. Bei einem geschützten API-Aufruf wird das Token an den Ressourcenserver übermittelt. Der Server prüft die Gültigkeit des Tokens und gibt bei erfolgreicher Prüfung die geschützten Daten an die Client-Anwendung zurück, sodass sie dem Benutzer angezeigt werden.

Nach der Beschreibung der Benutzeranmeldung wird nun der Bezahlvorgang über Stripe betrachtet.

Abbildung 4.4 zeigt den Ablauf eines Bezahlvorgangs mit Stripe. Das Sequenzdiagramm verdeutlicht, wie die React-Anwendung, das Spring Boot-Backend und der externe Zahlungsdienstleister Stripe zusammenwirken, um einen sicheren und effizienten Zahlungsprozess zu gewährleisten.

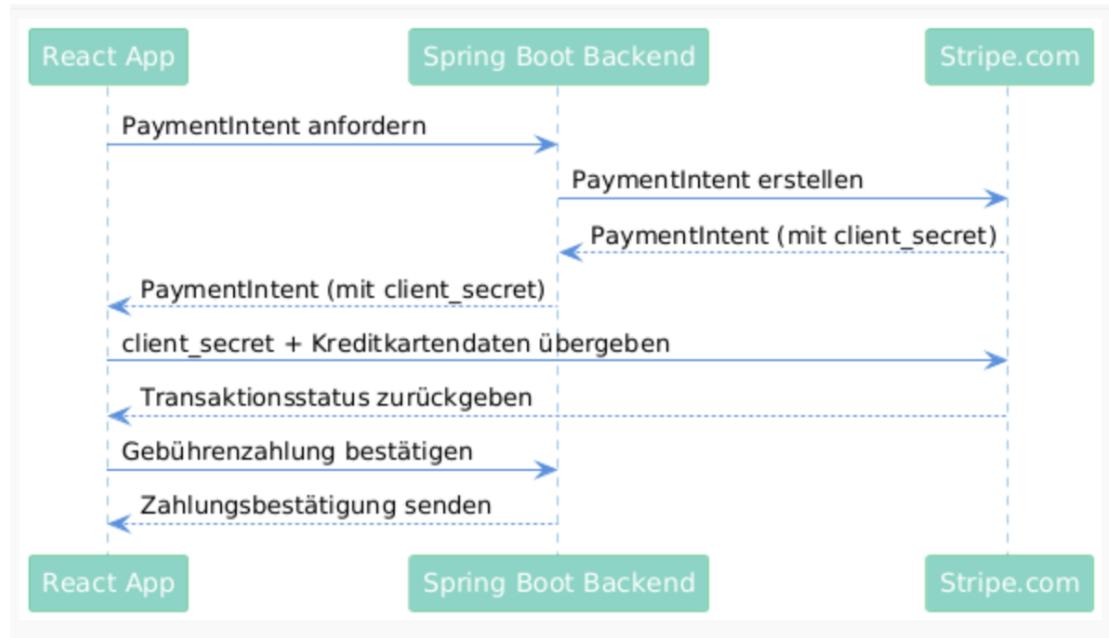


Abbildung 4.4: Zahlungsablauf mit Stripe, angelehnt an [MT25a]

Der Zahlungsprozess beginnt, wenn die React-Anwendung eine Anfrage an den Backend-Endpunkt `/api/payments/secure/intent` sendet, um einen neuen *PaymentIntent* zu erstellen. Das Backend leitet diese Anfrage an Stripe weiter und übermittelt dabei die erforderlichen Parameter wie Betrag, Währung und Zahlungsmethode. Stripe generiert daraufhin einen neuen *PaymentIntent* und gibt diesen, inklusive des *client_secret*, an das Backend zurück. Das Backend sendet den *client_secret* an die React-Anwendung, die ihn zusammen mit den Kreditkartendaten direkt an Stripe übermittelt. Stripe verarbeitet die Zahlungsinformationen und gibt den Transaktionsstatus zurück an das Frontend. Abschließend informiert das Frontend das Backend über die erfolgreiche Zahlung, sodass Datenbanken aktualisiert oder Zahlungshistorien gepflegt werden können, und das Backend bestätigt die erfolgreiche Zahlungsabwicklung.

Ein *PaymentIntent* ist ein von Stripe bereitgestelltes Objekt, das die Absicht einer Zahlung repräsentiert und den gesamten Zahlungsablauf verwaltet – von der

Erstellung über die Authentifizierung bis hin zur endgültigen Bestätigung der Zahlung.

Der *client-secret* ist ein einzigartiger, von Stripe generierter Schlüssel für jeden *PaymentIntent*. Er dient als sicherer Zugriffstoken, mit dem das Frontend bestimmte Informationen über die jeweilige Zahlung abrufen kann, beispielsweise den aktuellen Status der Transaktion. Der *client-secret* ist ausschließlich auf diese eine Zahlung beschränkt und kann nicht für andere Aktionen verwendet werden.

Der Endpunkt `POST /api/payments/secure/intent` nimmt ein JSON-Objekt mit Zahlungsinformationen wie Betrag und Währung entgegen. Intern ruft der Controller die Methode `generatePaymentIntent()` auf, die über das Stripe-SDK einen neuen *PaymentIntent* erstellt. Dabei werden der Betrag (`amount`), die Währung (`currency`) und der Zahlungstyp (auf Kreditkarte beschränkt) festgelegt. Das vom Stripe-SDK zurückgegebene *PaymentIntent*-Objekt, inklusive *client-secret*, wird anschließend vom Backend an das Frontend übermittelt, um den Bezahlvorgang fortzusetzen.

4.3 Anforderungen an das System

Die folgenden Unterabschnitte beschreiben die zentralen Anforderungen an das System. Dazu zählen funktionale, nicht-funktionale sowie technische Anforderungen, die für eine strukturierte Umsetzung der Anwendung erforderlich sind.

4.3.1 Funktionale Anforderungen

Funktionale Anforderungen definieren, welche Dienste das System leisten soll und wie es sich bei bestimmten Eingaben oder in bestimmten Situationen verhalten soll.

Die wichtigsten funktionalen Anforderungen von *LibraNova* sind:

- **Buchverwaltung:** Nutzer können Bücher suchen, deren Verfügbarkeit prüfen, ausleihen, zurückgeben, verlängern sowie Rezensionen und Bewertungen verfassen.
- **Administrationsfunktionen:** Administratoren können Bücher hinzufügen, bearbeiten (z. B. Anzahl der Exemplare ändern) und aus dem Katalog löschen.
- **Benutzerfunktionen:** Nutzer können ihre Ausleihen und die Ausleihhistorie einsehen und Zahlungen über Stripe abwickeln.

- **Such- und Filterfunktionen:** Das System bietet eine Suche nach Titel oder Kategorie, zeigt die Ergebnisse paginiert an und unterstützt eine mehrsprachige Anzeige der Buchbeschreibungen (Deutsch/Englisch).
- **Systemfeedback:** Das System informiert über nicht verfügbare Bücher, gelöschte Medien in der Ausleihliste sowie über Gebühren bei verspäteter Rückgabe.

Eine vollständige Übersicht aller funktionalen Anforderungen befindet sich in Anhang A.1.

4.3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die Qualitätsmerkmale und Randbedingungen des Systems, wie etwa Leistung, Sicherheit, Benutzbarkeit und Zuverlässigkeit.

Die wichtigsten nicht-funktionalen Anforderungen von *LibraNova* sind:

- **Benutzerfreundlichkeit:** Die Anwendung muss eine konsistente, intuitive und responsive Benutzeroberfläche bieten, die auf verschiedenen Geräten, Bildschirmgrößen und gängigen Browsern funktioniert. Zudem wird eine zweisprachige Oberfläche (Deutsch/Englisch) bereitgestellt.
- **Sicherheit:** Sichere Kommunikation über HTTPS, Schutz durch gültige Tokens, klare Trennung von Benutzerrollen sowie Zugriffsbeschränkung auf geschützte Endpunkte.
- **Leistung:** Such- und Filterfunktionen sowie Login-Vorgänge müssen kurze Reaktionszeiten gewährleisten, um eine flüssige Nutzung zu ermöglichen.
- **Barrierefreiheit:** Erfüllung grundlegender WCAG-2.1-Anforderungen (Level AA), um auch Nutzern mit Einschränkungen einen Zugang zu ermöglichen.
- **Wartbarkeit und Erweiterbarkeit:** Modularer Aufbau der Backend-Services sowie eine aktuelle und entwicklerfreundliche API-Dokumentation.

Eine vollständige Übersicht aller nicht-funktionalen Anforderungen befindet sich in Anhang A.2.

4.3.3 Technische Anforderungen

Technische Anforderungen beschreiben, welche Technologien, Werkzeuge und Methoden für die Entwicklung und den Betrieb von LibraNova verwendet werden. Dazu gehören zum Beispiel Programmiersprachen, Frameworks, Schnittstellen und Protokolle.

Die wichtigsten technischen Anforderungen von *LibraNova* sind:

- **Architektur:** Spring Boot als Backend mit RESTful APIs, React als Frontend sowie eine relationale MySQL-Datenbank für die Persistenz.
- **Sicherheit und Authentifizierung:** HTTPS, JWT-Token, OAuth2/OpenID Connect mit Okta sowie CORS-Restriktionen für das vertrauenswürdige Frontend.
- **Entwicklung und Qualitätssicherung:** Nutzung von Spring Data JPA/REST, Swagger-Dokumentation, Lombok sowie Unit-Tests mit JUnit und Mockito.

Eine vollständige Übersicht aller technischen Anforderungen befindet sich in Anhang A.3.

4.4 Wichtige Algorithmen (Pseudocode)

In diesem Abschnitt werden die wichtigsten Algorithmen dargestellt, die zentrale Abläufe des Systems beschreiben. Die Abläufe werden aus der Perspektive der Benutzerinteraktionen sowie der dahinterliegenden Systemlogik erläutert. Dabei werden wesentliche Prüfungen und Bedingungen berücksichtigt, um einen klaren und nachvollziehbaren Überblick über die Funktionsweise zu geben.

4.4.1 Buchausleihe

Der folgende Pseudocode 4.1 beschreibt den vollständigen Ablauf der Buchausleihe aus der Perspektive des Benutzers sowie der Systemlogik. Dabei werden wichtige Prüfungen berücksichtigt, wie die Anmeldung des Benutzers, die Verfügbarkeit von Exemplaren, die maximale Anzahl ausgeliehener Bücher sowie etwaige überfällige Rückgaben oder offene Gebühren.

```
1. Benutzer öffnet die Detailseite eines Buches.  
2. System prüft:  
   a. Ist der Benutzer angemeldet?
```

```

4           - Nein → Zeige Button >> Anmelden << (Weiterleitung zur
5           Login-Seite)
6           - Ja → Weiter zu Schritt 3
7   3. Hat der Benutzer dieses Buch bereits ausgeliehen?
8           - Ja → Zeige Hinweis >> Bereits ausgeliehen <<
9           - Nein → Weiter zu Schritt 4
10  4. Hat der Benutzer weniger als 5 Bücher ausgeliehen?
11           - Nein → Zeige Hinweis >> Maximale Anzahl an Büchern
12           erreicht <<
13           - Ja → Weiter zu Schritt 5
14  5. Sind Exemplare des Buches verfügbar?
15           - Nein → Zeige deaktivierten Button >> Ausleihen <<
16           - Ja → Zeige aktiven Button >> Ausleihen <<
17  6. Klickt der Benutzer auf >> Ausleihen <<:
18           a. Hat der Benutzer überfälligen Bücher oder unbezahlten Gebü
hren?
           - Nein → Legt Ausleiheintrag in der Datenbank an (7 Tage
Leihfrist)
           - Ja → Zeige Meldung: >> Bitte geben Sie überfällige Bü
cher zurück und begleichen Sie offene Gebühren, bevor
Sie neue Bücher ausleihen können. <<

```

Listing 4.1: Pseudocode für den Ausleihvorgang eines Buches

4.4.2 Verlängerung der Buchausleihe

Der folgende Pseudocode 4.2 beschreibt den vollständigen Ablauf, wie Nutzer die Leihfrist eines ausgeliehenen Buches verlängern können. Dabei werden die Bedingungen geprüft, unter denen eine Verlängerung möglich ist, sowie die Benutzeroberfläche entsprechend angepasst.

```

1   1. Benutzer öffnet seine Ausleihen und klickt bei einem Buch auf
2           >> Ausleihe verwalten <<.
3   2. System zeigt zwei Optionen:
4           a. >> Zurückgeben <<
5           b. >> Verlängern <<
6   3. System prüft für den Button >> Verlängern <<:
7           a. Ist das Rückgabedatum überschritten?
8           b. Wurde das Buch aus dem System gelöscht?
9   4. Wenn eine der Bedingungen erfüllt ist:
10           a. Zeige deaktivierten Button mit dem Text >> Verlängerung
nicht möglich <<.
11  5. Wenn keine der Bedingungen zutrifft:
12           a. Button >> Verlängern << ist aktiv.
13           b. Benutzer klickt auf >> Verlängern <<.
14           c. System verlängert die Leihfrist um 7 Tage.
           d. Verbleibende Tage werden aktualisiert.

```

Listing 4.2: Pseudocode für die Verlängerung einer Buchausleihe

5

Realisierung

In diesem Kapitel werden die zentralen Komponenten der Systemarchitektur vorgestellt. Es beginnt mit dem modular aufgebauten Backend auf Basis von Spring Boot, gefolgt von der Struktur und Internationalisierung des Frontends mit React. Abschließend wird das Datenbankmodell erläutert, einschließlich der Persistenzlogik mit Spring Data und Hibernate.

5.1 Backend-Architektur

Das Backend basiert auf Spring Boot und folgt einer klar strukturierten, modularen Architektur. Beschrieben werden der Einstiegspunkt, die Paketstruktur sowie die eingesetzte Teststrategie zur Sicherstellung der Codequalität.

5.1.1 Projektinitialisierung und Startklasse

Spring Boot vereinfacht die Entwicklung von Java-basierten Webanwendungen durch konventionsbasierte und automatisierte Projektkonfiguration. Dieses Framework bietet eine standardisierte Struktur, automatisches Abhängigkeitsmanagement und integrierte Komponenten, die den Entwicklungsprozess erheblich beschleunigen.

In diesem Abschnitt wird die Struktur des Projekts erläutert, wobei der Schwerpunkt auf der Projektinitialisierung, den zentralen Einstiegspunkt der Anwendung sowie wesentliche Konfigurationsmechanismen liegt.

- **Projektinitialisierung (Spring Initializr):** Die Anwendung *LibraNova* wurde mithilfe des Online-Tools *Spring Initializr* (<https://start.spring.io/>) erzeugt. Dieses Tool ermöglicht die einfache Auswahl von Projektparametern wie Abhängigkeiten, Java-Version und Build-Tool (z. B. Maven oder Gradle) und

generiert eine startbereite Projektstruktur inklusive grundlegender Dateien und Verzeichnisse.

- **Einstiegspunkt der Anwendung – Main-Klasse:** Die zentrale Einstiegsklasse der Anwendung befindet sich im `src/main/java`-Verzeichnis und enthält die `main`-Methode. Sie ist mit der Annotation `@SpringBootApplication` versehen, welche drei wichtige Spring-Annotationen kombiniert:

- `@Configuration`: Kennzeichnet die Klasse als Quelle für Bean-Definitionen.
- `@EnableAutoConfiguration`: Aktiviert die automatische Konfiguration basierend auf den eingebundenen Abhängigkeiten.
- `@ComponentScan`: Ermöglicht das automatische Auffinden von Komponenten, Services und Repositories im angegebenen Paket und dessen Unterpaketen [ST25c].

Nachfolgend 5.1 ist der Quellcode der Main-Klasse dargestellt:

```

1      @SpringBootApplication
2      public class LibranovaSpringBootApplication {
3
4          public static void main(String[] args) {
5              SpringApplication.run(LibranovaSpringBootApplication.class
6                  , args);
7
8      }

```

Listing 5.1: Einstiegspunkt der Spring Boot Anwendung

Erläuterung der Main-Klasse:

- `@SpringBootApplication`: Diese Annotation bündelt die Konfiguration, Auto-Konfiguration und Komponentensuche in einer einzigen Annotation und ist der zentrale Startpunkt für die Spring-Boot-Anwendung.
- `public class LibranovaSpringBootApplication`: Definition der Hauptklasse, die die Anwendung repräsentiert.
- `public static void main(String[] args)`: Die Main-Methode dient als Einstiegspunkt der Java-Anwendung. Sie wird beim Programmstart aufgerufen.
- `SpringApplication.run(...)` : Diese Methode startet den eingebetteten Server, initialisiert den Spring Application Context und lädt alle Komponenten, Beans und Konfigurationen.
- **Konfiguration über `application.properties`:** Die Datei `application.properties` im Verzeichnis `src/main/resources` enthält

zentrale Konfigurationen der Anwendung, wie z. B. Datenbankverbindung, REST-Basis-Pfad und andere anwendungsspezifische Einstellungen.

```

1      # Name der Anwendung
2      spring.application.name=libranova

4      # Datenbankverbindung (MySQL)
5      spring.datasource.url=jdbc:mysql://localhost:3306/libranova_db
6      spring.datasource.username=username
7      spring.datasource.password=password

9      # Dialekt von Hibernate
10     spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
11         MySQLDialect

12     # Basis-REST-Pfad
13     spring.data.rest.base-path=/api

```

Listing 5.2: application.properties Datei

Erläuterung der wichtigsten Einstellungen der Datei 5.2:

- `spring.application.name`: Definiert den Namen der Anwendung, z. B. für Logs oder Monitoring.
- `spring.datasource.url`: Verbindungs-URL zur MySQL-Datenbank inklusive Host, Port und Datenbankname.
- `spring.datasource.username`: Benutzername für die Datenbankverbindung.
- `spring.datasource.password`: Passwort für den Datenbankzugang.
- `spring.jpa.properties.hibernate.dialect`: Gibt den SQL-Dialekt für JPA/Hibernate an (hier MySQL).
- `spring.data.rest.base-path`: Legt den Basis-URL-Pfad für automatisch generierte REST-Endpunkte fest (z. B. `/api`).
- **Eingebetteter Webserver (Tomcat):** Spring Boot integriert standardmäßig einen eingebetteten Webserver (standardmäßig Tomcat). Dadurch kann die Anwendung direkt als eigenständiger Prozess gestartet werden, ohne dass ein separater Webserver installiert oder konfiguriert werden muss [Spr25b].

5.1.2 Modulare Paketstruktur des Backends

Im Folgenden wird die interne Struktur des Backends detailliert betrachtet. Dazu gehört die Aufteilung der Anwendung in verschiedene Pakete, die jeweils eigene Verantwortlichkeiten übernehmen und so zur Übersichtlichkeit und Wartbarkeit beitragen.

Eine schematische Übersicht der Paketstruktur ist in **Abbildung 5.1** unten dargestellt.

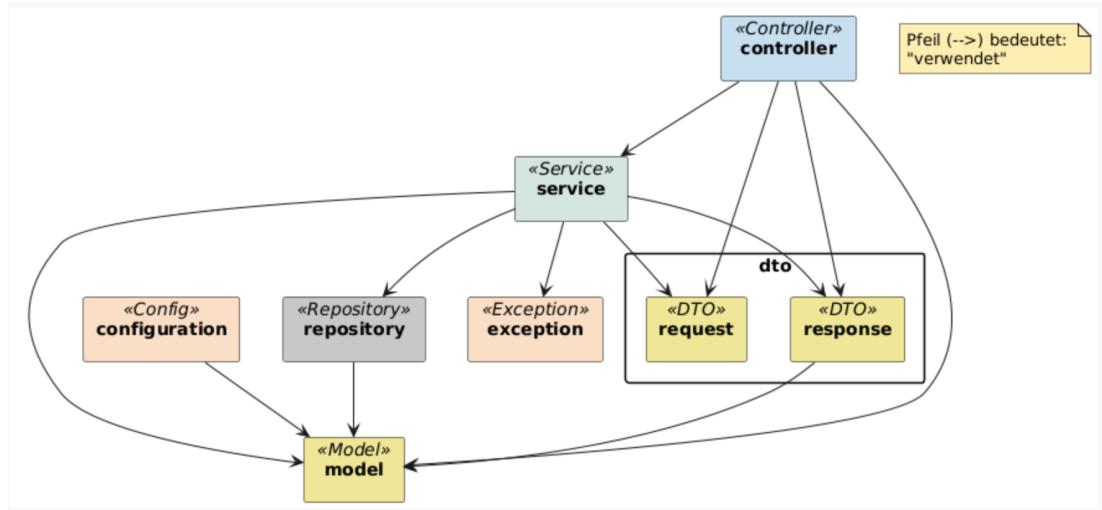


Abbildung 5.1: Modularer Aufbau des Backends

Im Folgenden werden die Pakete und ihre jeweiligen Verantwortlichkeiten beschrieben.

- **model:** Enthält die JPA-Entitäten, die die Tabellen der MySQL-Datenbank abbilden. Jede Klasse in diesem Paket entspricht einer Datenbanktabelle und definiert deren Attribute und Beziehungen. Diese Entitäten bilden die Grundlage für den Datenzugriff über das Repository.
- **dto:** Dient dem strukturierten Datenaustausch zwischen Client und Server, ohne interne Entitäten direkt preiszugeben. Es gibt zwei Unterpakete:
 - **request:** definiert Datenstrukturen für eingehende Anfragen wie etwa Formularinhalte oder Suchparameter,
 - **response:** definiert Rückgabeformate, die speziell für die Client-seitige Anzeige oder Weiterverarbeitung optimiert sind.

Die Verwendung von DTOs erhöht die Sicherheit und Flexibilität des Datenmodells.

- **exception:** Beinhaltet zentrale Komponenten zur Fehlerbehandlung. In der aktuellen Implementierung ist eine benutzerdefinierte Ausnahme enthalten, die bei nicht verfügbaren Büchern geworfen wird. Diese Ausnahme verbessert die Verständlichkeit von Fehlermeldungen auf der Client-Seite.

- **repository:** Beinhaltet Interfaces zur Datenzugriffsabstraktion mittels **Spring Data JPA**. Sie ermöglichen CRUD-Operationen auf den JPA-Entitäten, ohne dass eigene SQL-Statements geschrieben werden müssen. Damit wird der Datenzugriff stark vereinfacht und typsicher umgesetzt.
- **service:** Kapselt die Geschäftslogik der Anwendung. Hier werden Anfragen aus den Controllern verarbeitet, Daten validiert und Repository-Zugriffe koordiniert. Die Services dienen als zentrale Steuerungseinheit zwischen Controller-Logik und Datenbankzugriff.
- **controller:** Beinhaltet die REST-Controller zur Entgegennahme und Verarbeitung von HTTP-Anfragen. Sie dienen als Schnittstelle zwischen Client und Server und leiten die Anfragen zur weiteren Verarbeitung an die Service-Schicht weiter. Zudem bereiten sie die Daten so auf, dass sie für den Client verständlich und verwertbar sind.
- **configuration:** Enthält zentrale Sicherheitskonfigurationen der Anwendung. Hier werden der Zugriff auf HTTP-Endpunkte sowie Authentifizierungsmechanismen mittels JWT und OAuth2 (Okta) definiert und gesteuert.

5.1.3 Teststrategie und Testintegration

Um die Qualität und Zuverlässigkeit des Backends sicherzustellen, wurde ein automatisiertes Testkonzept implementiert. Dabei kommen vor allem Unit-Tests mit **JUnit** sowie Mocking mit **Mockito** zum Einsatz.

Die Tests befinden sich im Verzeichnis `src/test/java` und folgen der Paketstruktur des Produktivcodes, um eine klare Zuordnung zu ermöglichen.

- **JUnit** wird verwendet, um einzelne Komponenten isoliert zu testen und deren Verhalten zu validieren.
- **Mockito** ermöglicht das Erzeugen von Mock-Objekten, um Abhängigkeiten während der Tests zu simulieren und somit isolierte Testumgebungen zu schaffen.

Integration der Tests in den Build-Prozess erfolgt über das verwendete Build-Tool **Maven**, wodurch die Tests automatisiert ausgeführt werden können und eine kontinuierliche Qualitätssicherung gewährleistet ist.

5.2 Frontend-Struktur

In diesem Abschnitt wird die Architektur des Frontends erläutert. Beginnend mit dem Einstiegspunkt der React-Anwendung, werden anschließend die Projektstruktur sowie die Integration der Internationalisierung mittels i18next vorgestellt.

5.2.1 Einstiegspunkt der React-Anwendung

Das Frontend der Anwendung wurde mit **React** und **TypeScript** auf Basis von **Create React App (CRA)** entwickelt. CRA bietet eine sofort einsatzbereite Entwicklungsumgebung inklusive Webpack-Konfiguration, Hot-Reloading, Testing-Setup und TypeScript-Support [Fac25].

In diesem Abschnitt wird die Struktur der React-Anwendung beschrieben, wobei der Schwerpunkt auf der Projektinitialisierung, dem Einstiegspunkt sowie zentralen Konfigurationsdateien liegt.

- **Projektinitialisierung mit CRA:** Die Anwendung wurde über folgendes Kommando initialisiert (siehe Listing 5.3):

```
1 npx create-react-app react-library-app --template typescript
```

Listing 5.3: Projektinitialisierung mit Create React App

Dieses Kommando setzt sich wie folgt zusammen:

- **npx:** Führt ein npm-Paket temporär aus, ohne es global zu installieren.
- **create-react-app:** Das offizielle Tool zur Erzeugung von React-Projekten, welches ein komplettes Setup mit Build-Tooling, Linter und Tests erstellt.
- **react-library-app:** Der Name des Projektverzeichnisses, das automatisch erstellt wird.
- **--template typescript:** Gibt an, dass die Anwendung mit TypeScript anstelle von JavaScript erstellt werden soll.

Dadurch wurde eine vollständige Projektstruktur erzeugt, einschließlich Konfigurationsdateien, TypeScript-Unterstützung und einer initialen Komponentenstruktur.

- **Einstiegspunkt – index.tsx:** Die Datei `src/index.tsx` (siehe Listing 5.4) bildet den Einstiegspunkt der React-Anwendung. Dort wird die Hauptkomponente `<App />` in das DOM eingebunden:

```

1      import React from 'react';
2      import ReactDOM from 'react-dom/client';
3      import './index.css';
4      import { App } from './App';
5      import { BrowserRouter } from 'react-router-dom';
6      import 'bootstrap-icons/font/bootstrap-icons.css';
7      import { loadStripe } from '@stripe/stripe-js';
8      import { Elements } from '@stripe/react-stripe-js';
9      import './i18n';

11     const stripePromise = loadStripe('my_stripe_public_key');

13     const root = ReactDOM.createRoot(
14       document.getElementById('root') as HTMLElement
15     );
16     root.render(
17       <BrowserRouter>
18         <Elements stripe={stripePromise}>
19           <App />
20         </Elements>
21       </BrowserRouter>
22     );

```

Listing 5.4: Einstiegspunkt der React-Anwendung

Die Datei `index.tsx` dient als zentrales Einstiegsskript für die React-Anwendung. Sie importiert zunächst alle notwendigen Module und Abhängigkeiten wie React selbst, den ReactDOM-Client, die globale CSS-Datei, die Hauptkomponente `App` sowie zusätzliche Bibliotheken für Routing (`react-router-dom`), UI-Icons (Bootstrap Icons), Zahlungsintegration (Stripe) und Internationalisierung (`i18n`).

Im nächsten Schritt wird Stripe über `loadStripe` mit dem öffentlichen Schlüssel initialisiert und in einer Promise-Variable gespeichert. Anschließend wird mit Hilfe von `ReactDOM.createRoot(...)` eine sogenannte „Root“-Instanz erzeugt, welche das `<div>` mit der ID `root` im HTML-Dokument referenziert. Das `<div>` mit der ID `root` befindet sich in der Datei `public/index.html`.

Im letzten Schritt erfolgt das eigentliche Rendern der Anwendung. Hierbei wird die Komponente `<App />` in den Kontext von `<BrowserRouter>` und `<Elements>` eingebettet, um Routing- und Zahlungsfunktionen global bereitzustellen.

- **Backend-Kommunikation über Umgebungsvariablen:** Hier wird die Port-URL des Backends definiert, welche von der React-Anwendung zur Kommunikation mit dem Backend genutzt wird. Diese Konfiguration erfolgt über die Datei `.env` (siehe Listing 5.5).

```
1      REACT_APP_API_URL='https://localhost:8443/api'
```

Listing 5.5: Frontend-Umgebungsvariable für Backend-Zugriff in `.env`-Datei

Die Umgebungsvariable `REACT_APP_API_URL` gibt die Adresse an, unter der die REST-API des Backends erreichbar ist. Damit diese Variable im Code verwendet werden kann, muss sie zwingend mit dem Präfix `REACT_APP_` beginnen. Diese Variable wird beispielsweise im Code zur Konfiguration der API-Endpunkte verwendet, z. B. `fetch(process.env.REACT_APP_API_URL + '/books')`.

5.2.2 React-Projektstruktur

Im Folgenden wird die interne Struktur der React-Frontend-Anwendung detailliert betrachtet. Die Anwendung ist in verschiedene Verzeichnisse unterteilt, die jeweils spezifische Verantwortlichkeiten übernehmen und dadurch zur Übersichtlichkeit, Modularität und Wartbarkeit beitragen.

Eine schematische Übersicht der Verzeichnisse und zentralen Dateien ist in **Abbildung 5.2** unten dargestellt.

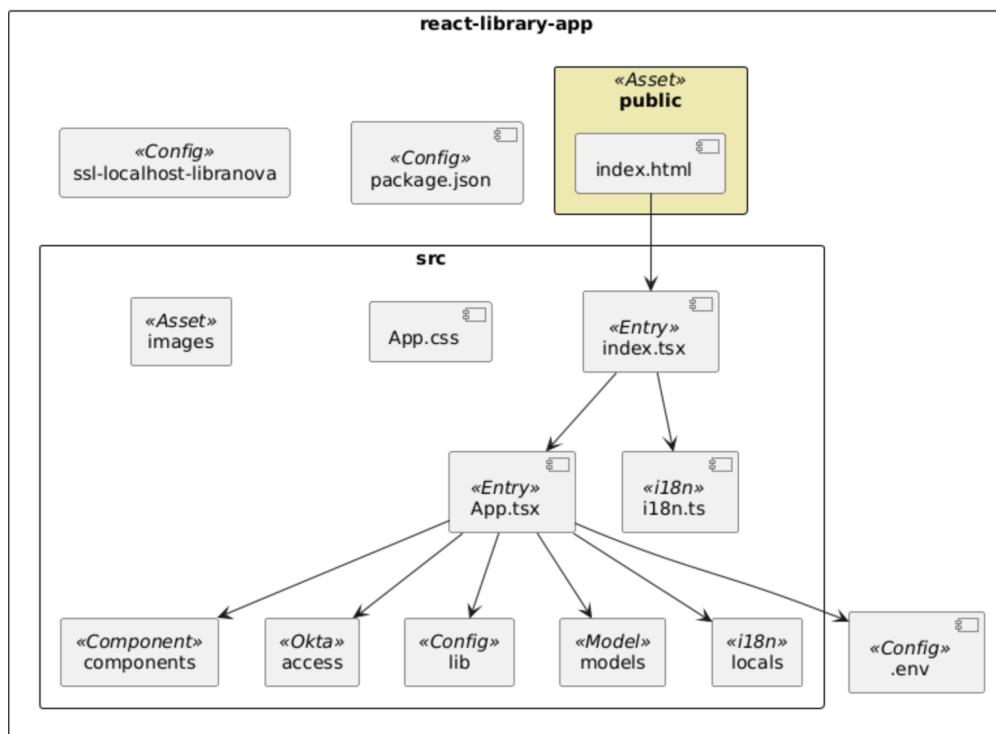


Abbildung 5.2: Modularer Aufbau des Frontends

Anschließend werden die wichtigsten Ordner und Dateien sowie ihre jeweiligen Aufgabenbereiche beschrieben.

- **index.html:** Statisches HTML-Grundgerüst der Anwendung. Enthält das `<div>` mit der ID `root`, in das React die App rendernt. Zusätzlich werden hier externe Ressourcen wie Bootstrap und Stripe eingebunden.
- **package.json:** Zentrale Konfigurationsdatei für das React-Projekt. Definiert alle Projektabhängigkeiten, Skripte zur Ausführung (z. B. Starten, Bauen, Testen), sowie weitere Metadaten der Anwendung.
- **ssl-local-libranova:** Enthält SSL-Zertifikat (`localhost.crt`) und privaten Schlüssel (`localhost.key`) für die lokale HTTPS-Entwicklung.
- **index.tsx:** (siehe Listing 5.4) — bereits zuvor beschrieben im Abschnitt zur Einstiegspunktstruktur.
- **App.css:** Enthält benutzerdefinierte globale CSS-Regeln zur Gestaltung zentraler UI-Komponenten, darunter Header, Buttons, Bilder, Effekte sowie Media Queries zur responsiven Darstellung.
- **images:** Enthält statische Bilddateien, die in der Benutzeroberfläche verwendet werden.
- **i18n.ts:** Initialisiert die Mehrsprachigkeit mittels `i18next` mit englischen und deutschen Übersetzungen sowie automatischer Spracherkennung.
- **App.tsx:** Zentrale Komponente der Anwendung, die das Routing und die Navigation steuert. Sie bindet verschiedene Seiten und Komponenten ein und integriert die Authentifizierung mit Okta. Header und Footer sorgen für das Layout, während geschützte Routen durch `Security` verwaltet werden.
- **components:** Enthält die wiederverwendbaren UI-Komponenten der Anwendung, wie Header, Footer, Hauptseite, Suchseite und weitere funktionale Elemente. Diese Komponenten bilden die Benutzeroberfläche und sind modular aufgebaut, um Wartbarkeit und Erweiterbarkeit zu gewährleisten.
- **access:** Beinhaltet die Integration und Steuerung des Okta-Authentifizierungswidgets. Die Komponenten `OktaLoginWidget` und `OktaSignInWidget` ermöglichen das Anmelden, Handhaben von Login-Events und Weiterleitung nach erfolgreicher Authentifizierung.
- **lib:** Enthält die Datei `oktaConfig.ts`, die die Einstellungen für die Okta-Authentifizierung definiert, wie Client-ID, Autorisierungsserver (Issuer), Redirect-URI und Berechtigungen (Scopes). Diese Konfiguration ist zentral für die Anbindung der Okta-Authentifizierung im Frontend.
- **models:** Enthält Klassen, die zentrale Datenstrukturen der Anwendung modellieren, wie z. B. `Book`. Sie dienen als Grundlage für den strukturierten Datenaustausch innerhalb des Frontends sowie zwischen Frontend und Backend.

- **locales:** Beinhaltet sprachspezifische JSON-Dateien (`en`, `de`) für die Internationalisierung der UI mittels `react-i18next`.
- **.env:** Diese Datei definiert Umgebungsvariablen für das Projekt. Dazu gehören Pfade zu SSL-Zertifikat und -Schlüssel für die HTTPS-Kommunikation im lokalen Entwicklungsumfeld sowie die Backend-URL (`REACT_APP_API_URL`) für die React-Anwendung.

5.2.3 Internationalisierung mit i18next

Zur Umsetzung der Mehrsprachigkeit im Frontend wurde das `i18next`-Framework in Kombination mit `react-i18next` und `i18next-browser-languagedetector` verwendet. Nach der Installation dieser Abhängigkeiten wird in der Datei `i18n.ts` (siehe Listing 5.6) die Initialisierung des Übersetzungssystems vorgenommen.

```

1 import i18n from "i18next";
2 import { initReactI18next } from "react-i18next";
3 import LanguageDetector from "i18next-browser-languagedetector";

5 import translationEN from "./locales/en/translation.json";
6 import translationDE from "./locales/de/translation.json";

8 const resources = {
9     en: { translation: translationEN },
10    de: { translation: translationDE }
11};

13 i18n
14 .use(LanguageDetector) // Detektiert die Sprache des Benutzers
15 .use(initReactI18next) // Bindet i18next an React
16 .init({
17     resources,
18     fallbackLng: "en", // verwenden Sie Englisch als Fallback-Sprache
19     interpolation: {
20         escapeValue: false // React bereits vor XSS-Angriffen schützt
21     }
22 });

24 export default i18n;

```

Listing 5.6: Initialisierung von i18next in `i18n.ts`

Hier ist eine Zeile-für-Zeile-Erklärung der Datei `i18n.ts`, die die Internationalisierung der React-Anwendung initialisiert:

- `import i18n from "i18next";`
Importiert die Hauptbibliothek `i18next`, die die Internationalisierung ermöglicht.

- `import { initReactI18next } from "react-i18next";`
Importiert die React-spezifische Integration, um `i18next` mit React zu verbinden.
- `import LanguageDetector from "i18next-browser-languagedetector";`
Importiert ein Modul zur automatischen Erkennung der Sprache des Benutzers im Browser.
- `import translationEN from "./locales/en/translation.json";`
Importiert die englischen Übersetzungen aus der JSON-Datei.
- `import translationDE from "./locales/de/translation.json";`
Importiert die deutschen Übersetzungen aus der JSON-Datei.
- `const resources = {
 en: { translation: translationEN },
 de: { translation: translationDE }
};`
Definiert die verfügbaren Sprachressourcen mit den jeweiligen Übersetzungen.
- `i18n
 .use(LanguageDetector)
 .use(initReactI18next)
 .init({
 resources,
 fallbackLng: "en",
 interpolation: {
 escapeValue: false
 }
 });`
Initialisiert `i18next` mit: automatischer Spracherkennung, React-Integration, Sprachressourcen, Standard-Fallbacksprache Englisch, und deaktiviert Escape-Mechanismen, da React bereits sicher ist.
- `export default i18n;`
Exportiert die konfigurierte Instanz, damit sie im Projekt verwendet werden kann.

Die sprachspezifischen Übersetzungen sind in den Ordner `en` und `de` als strukturierte `.json`-Dateien organisiert.

Ein typisches Beispiel (siehe Listing 5.7) für die Verwendung in einer React-Komponente ist:

```
1  import { useTranslation } from 'react-i18next';  
3  const { t } = useTranslation();
```

```
5 <button>{t("checkout.thankYouReview")}</button>
```

Listing 5.7: Beispielhafte Nutzung von `useTranslation`

Hierbei wird die Funktion `useTranslation` aus `react-i18next` importiert und aufgerufen. Die Rückgabe wird destrukturiert, wobei die Variable `t` extrahiert wird. Die Variable `t` ist eine Funktion, die einen Übersetzungsschlüssel als Argument entgegennimmt, den entsprechenden Text aus den JSON-Übersetzungsdateien auswählt und diesen korrekt für die aktuelle Sprache zurückgibt. In diesem Beispiel liefert `t("checkout.thankYouReview")` je nach eingestellter Sprache den Text aus `en.json` oder `de.json`.

In den entsprechenden JSON-Dateien sind die Übersetzungen wie folgt definiert:

en.json:

```
1 "checkout": {
2     "thankYouReview": "Thank you for your review"
3 }
```

Listing 5.8: Englische Übersetzung in `en.json`

de.json:

```
1 "checkout": {
2     "thankYouReview": "Vielen Dank für deine Bewertung"
3 }
```

Listing 5.9: Deutsche Übersetzung in `de.json`

Wie in Listing 5.8 und Listing 5.9 zu sehen, werden die Schlüssel `checkout.thankYouReview` mit den jeweiligen Texten in Englisch und Deutsch verknüpft. Die Funktion `t` stellt dabei sicher, dass der passende Text basierend auf der aktuellen Sprache dynamisch angezeigt wird.

Zur Laufzeit kann die Sprache über ein Dropdown-Menü gewechselt werden. Im `Header.tsx` (siehe Listing 5.10) der Anwendung befindet sich ein Sprachumschalter, der die aktuelle Sprache visuell mit einem Icon anzeigt und dem Benutzer den Wechsel zwischen Deutsch und Englisch ermöglicht:

```
1 <div className="dropdown">
2     <button>
3         {i18n.language === 'de' ? '[DE]' : '[EN]'}
4     </button>
5     <ul className="dropdown-menu">
6         <li><button onClick={() => i18n.changeLanguage('en')}>[EN] English
7             </button></li>
```

```
7      <li><button onClick={() => i18n.changeLanguage('de')}>[DE] Deutsch
8      </button></li>
9    </ul>
</div>
```

Listing 5.10: Sprachumschalter im Header

Hier ist die Zeilen-für-Zeilen-Erklärung des Sprachumschalters:

- `<div className="dropdown">`: Definiert einen Dropdown-Container für die Sprachwahl.
- `<button>`: Der Button zeigt die aktuell ausgewählte Sprache an. Die Darstellung erfolgt dynamisch: Falls die Sprache Deutsch ist (`i18n.language === 'de'`), wird `[DE]` angezeigt, sonst `[EN]`.
- `<ul className="dropdown-menu">`: Die Dropdown-Liste mit Sprachoptionen.
- `<button onClick={() => i18n.changeLanguage('en')}>`: Ein Button zum Wechseln auf Englisch. Beim Klick wird die Sprache zu Englisch gewechselt.
- `<button onClick={() => i18n.changeLanguage('de')}>`: Ein Button zum Wechseln auf Deutsch. Beim Klick wird die Sprache zu Deutsch gewechselt.

Damit wird gewährleistet, dass die Benutzeroberfläche abhängig von der gewählten Sprache dynamisch angepasst wird.

5.3 Datenbankmodell und Persistenz

In diesem Abschnitt werden das Datenbankschema sowie die Umsetzung der Datenpersistenz mit Spring Data und Hibernate erläutert.

5.3.1 Datenbankschema

Die Anwendung verwendet ein relationales MySQL-Datenbanksystem zur Speicherung persistenter Informationen wie Bücher, Ausleihen, Zahlungen, Bewertungen und Nachrichten. Das ER-Modell der Datenbank ist in Abbildung 5.3 dargestellt.

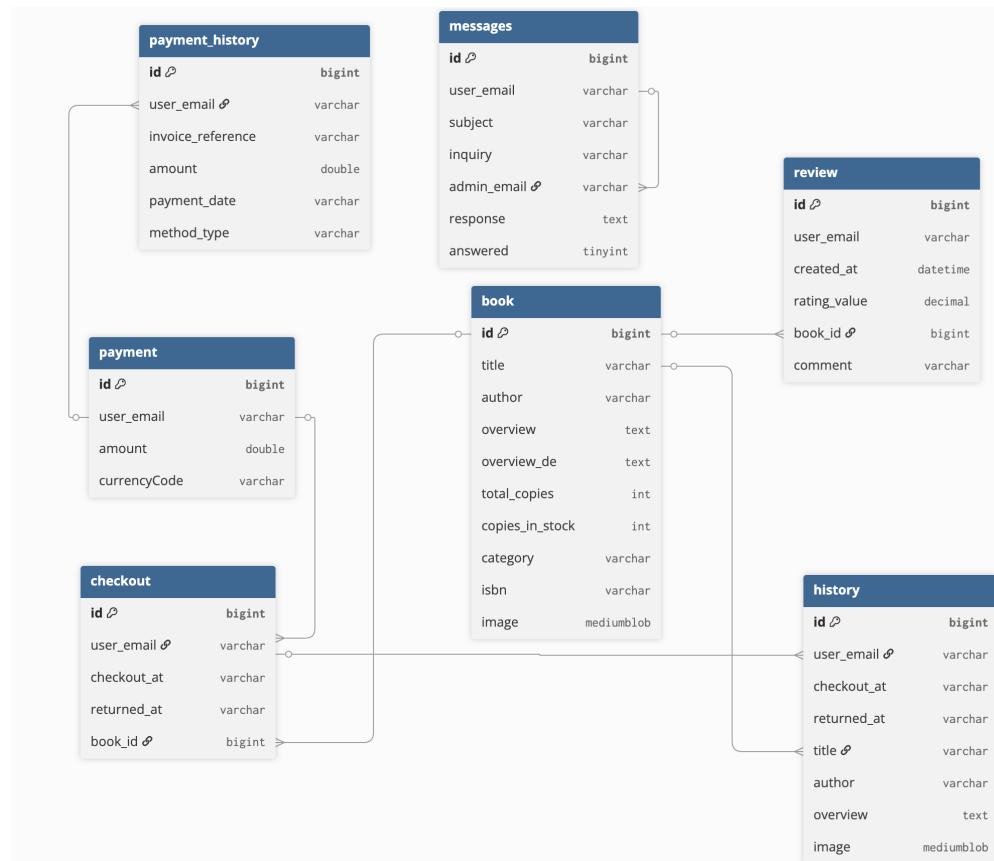


Abbildung 5.3: ER-Modell der Datenbank

Zur besseren Übersicht werden im Folgenden die einzelnen Tabellen des Datenbankschemas beschrieben. Dabei werden sowohl die gespeicherten Informationen als auch die zentralen logischen Beziehungen zu anderen Tabellen erläutert.

- **Book:** Enthält zentrale Buchinformationen wie Titel, Autor, Beschreibung, Kategorie, ISBN, Bild, Gesamtanzahl und verfügbare Exemplare. Jedes Book kann in mehreren **Checkouts** ausgeliehen werden (1:N) und mehrere **Reviews** besitzen (1:N). Historische Einträge in **History** können ebenfalls den ursprünglichen Buchtitel referenzieren.
- **Checkout:** Repräsentiert aktive Ausleihen eines Benutzers. Enthält u. a. Benutzer-E-Mail, Ausleih- und Rückgabedatum sowie book_id. Jede Ausleihe ist eindeutig einem Book und einem User zugeordnet. Frühere Ausleihen werden in **History** gespeichert. Zahlungen (Payment) sind zusätzlich über die Benutzer-E-Mail mit Ausleihen verknüpft.
- **Review:** Speichert Bewertungen von Büchern durch Benutzer, inklusive Bewertungspunktzahl, Kommentar und Erstellungsdatum. Jede Bewertung ist über

`book_id` einem Buch und über `user_email` einem Benutzer zugeordnet (N:1-Beziehung).

- **Messages:** Dient der Kommunikation zwischen Benutzern und Administratoren. Enthält Betreff, Nachricht, Antwort, Status (beantwortet/nicht beantwortet) sowie Benutzer- und Administrator-Mailadressen. Damit wird eine Beziehung zwischen Nutzer und Administrator hergestellt.
- **History:** Enthält vergangene Ausleihen eines Benutzers und speichert redundante Buchinformationen (Titel, Autor, Beschreibung, Bild). Verknüpft über `user_email` mit `Checkout`-Einträgen, um frühere Ausleihvorgänge nachzuvollziehen.
- **Payment:** Speichert Zahlungsinformationen wie Benutzer-E-Mail, Betrag und Währung. Zahlungen sind über die Benutzer-E-Mail mit den jeweiligen Ausleihen verbunden. Die Historisierung erfolgt über `PaymentHistory`.
- **PaymentHistory:** Enthält eine Übersicht aller abgeschlossenen Zahlungen eines Benutzers, verknüpft über die Benutzer-E-Mail mit `Payment`. Zusätzlich werden Informationen wie Rechnungsreferenz, Zahlungsdatum, Betrag und Zahlungsmethode gespeichert.

5.3.2 Datenpersistenz mit Spring Data und Hibernate

Die Anwendung nutzt eine Kombination aus **Hibernate**, **Spring Data JPA** und **Spring Data REST**, um eine effiziente und wartbare Persistenzschicht bereitzustellen. Die drei Komponenten übernehmen dabei unterschiedliche Rollen:

- **Hibernate:** Implementiert die Java Persistence API (JPA) und führt die eigentliche Kommunikation mit der Datenbank aus.
- **Spring Data JPA:** Stellt auf Basis von Hibernate eine vereinfachte Abstraktion zur Verfügung, die generische CRUD-Operationen und automatisch abgeleitete Queries ermöglicht.
- **Spring Data REST:** Exportiert die Repository-Schnittstellen automatisch als REST-Endpunkte, sodass CRUD-Operationen auch über HTTP verfügbar sind.

Beispiel: Entity-Klasse für Payments

```
1  @Entity
2  @Table(name = "payment")
3  @Data
4  public class Payment {
6      @Id
```

```

7     @GeneratedValue(strategy = GenerationType.IDENTITY)
8     private Long id;
9
10    @Column(name = "user_email")
11    private String userEmail;
12
13    @Column(name = "amount")
14    private double amount;
15 }
```

Listing 5.11: JPA-Entity Payment

Die Klasse `Payment` 5.11 ist als JPA-Entity mit der Annotation `@Entity` markiert, wodurch Hibernate die Persistenz dieser Klasse übernimmt. Mit `@Table(name="payment")` wird die Entity explizit der Datenbanktabelle `payment` zugeordnet. Die Annotation `@Data` von Lombok generiert automatisch Getter, Setter sowie die Methoden `equals()`, `hashCode()` und `toString()`, wodurch Boilerplate-Code reduziert wird. Das Attribut `id` ist mit `@Id` als Primärschlüssel gekennzeichnet, während `@GeneratedValue(strategy = GenerationType.IDENTITY)` die automatische Generierung des Schlüssels durch die Datenbank (z.B. MySQL AUTO_INCREMENT) sicherstellt. Die Felder `userEmail` und `amount` sind mittels `@Column(name = "...")` den entsprechenden Datenbankspalten zugeordnet, was die Trennung von Java-Feldnamen und Spaltennamen ermöglicht und somit eine konsistente Abbildung zwischen Domänenmodell und Datenbank garantiert.

Hibernate interpretiert die JPA-Annotationen der Klasse `Payment` und sorgt dafür, dass die Entity korrekt auf die Tabelle `payment` abgebildet wird, wobei es die SQL-Befehle für Insert, Update, Delete und Select ausführt.

Beispiel: Datenzugriff mit Spring Data

```

1  public interface PaymentRepository extends JpaRepository<Payment, Long> {
2      Payment findPaymentsByUserEmail(String userEmail);
3 }
```

Listing 5.12: Payments-Repository-Schnittstelle

Dieses Interface 5.12 definiert die Datenzugriffsschicht für die `Payment`-Entität. Durch die Erweiterung von `JpaRepository<Payment, Long>` stellt Spring automatisch alle grundlegenden CRUD-Operationen bereit. `Payment` bezeichnet die Entitätsklasse, `Long` ist der Datentyp des Primärschlüssels.

Die Methode `findPaymentsByUserEmail(String userEmail)` nutzt die Konventionen von Spring Data JPA: Anhand des Methodennamens erkennt Spring automatisch, dass nach dem Attribut `userEmail` gesucht werden soll, und generiert im Hintergrund die entsprechende SQL-Abfrage.

Da Spring Data REST als Abhängigkeit eingebunden ist, werden automatisch REST-Endpunkte für das Repository verfügbar gemacht – ohne dass man eigene Controller oder Services definieren muss. Die Endpunkte folgen dabei einer standardisierten Struktur:

- GET `/payments` → Alle Zahlungen abrufen
- POST `/payments` → Neue Zahlung erstellen
- GET `/payments/1` → Zahlung mit ID 1 abrufen
- DELETE `/payments/1` → Zahlung mit ID 1 löschen

Durch die gemeinsame Nutzung von Spring Data JPA und Spring Data REST wird eine klare Trennung zwischen Datenmodell und Zugriffsschicht erreicht, bei gleichzeitig minimalem Implementierungsaufwand.

6

Implementierung

Im Rahmen dieses Kapitels wird exemplarisch eine der wichtigsten Funktionalitäten der Webanwendung behandelt – der Ausleihprozess sowie das Verfahren zur Rückgabe eines Buches.

6.1 Ausleihprozess eines Buches

Im Folgenden wird der Ausleihprozess eines Buches aus Sicht des Backends sowie des Frontends detailliert beschrieben.

6.1.1 Backend-Prozess

In diesem Abschnitt wird der gesamte Backend-Prozess der Buchausleihe detailliert beschrieben. Die folgende Darstellung umfasst die beteiligten Repositories, die Service-Schicht inklusive aller Hilfsmethoden sowie den REST-Endpunkt im Controller.

Zugriff auf Daten: Repositories

1. PaymentRepository

Da das `PaymentRepository` bereits im vorherigen Kapitel (siehe Listing 5.12) im Zusammenhang mit Spring Data REST und JPA vorgestellt wurde, wird an dieser Stelle lediglich darauf verwiesen. Es wird für die Überprüfung offener Zahlungen verwendet.

2. CheckoutRepository

Für den Ausleihprozess wurde in diesem Repository (siehe Listing 6.1) diese Methoden genutzt:

```

1  public interface CheckoutRepository extends JpaRepository<Checkout,
2      Long> {
3      List<Checkout> findByUserEmail(String userEmail);
4      Checkout findByBookIdAndUserEmail(Long bookId, String userEmail);
5  }

```

Listing 6.1: CheckoutRepository.java

Die erste Methode ruft alle ausgeliehenen Bücher eines bestimmten Nutzers anhand seiner E-Mail-Adresse ab. Die zweite Methode ruft einen einzelnen Checkout-Eintrag aus der Datenbank ab, der zur angegebenen E-Mail-Adresse des Benutzers und der ID des Buches gehört.

Geschäftslogik: Die Methode checkoutBook im BookService

Die Methode `checkoutBook` (siehe Listing 6.2) enthält den gesamten Ablauf der Buchausleihe. Im Folgenden wird die Methode vollständig dargestellt und im Anschluss schrittweise erklärt:

```

1  public Book checkoutBook(String userEmail, Long bookId) throws
2      Exception {
3      Book book = bookRepository.findById(bookId)
4          .orElseThrow(() -> new BookNotFoundException("Book with ID
5              " + bookId + " is not available."));
6
7      checkAvailability(book, userEmail);
8
9      List<Checkout> userCheckouts = checkoutRepository.
10         findByUserEmail(userEmail);
11      boolean hasOverdueBooks = hasOverdueBooks(userCheckouts);
12
13      Payment payment = paymentRepository.findByUserEmail(userEmail)
14          ;
15
16      if ((payment != null && payment.getAmount() > 0) || (payment
17          != null && hasOverdueBooks)) {
18          throw new Exception("The loan has been blocked due to
19              outstanding payments or overdue books.");
20      }
21
22      if (payment == null) {
23          createZeroPayment(userEmail);
24      }
25
26      decrementBookStock(book);
27      createCheckoutRecord(userEmail, book);
28
29      return book;
30  }

```

Listing 6.2: checkoutBook() Methode im BookService.java

Erklärung:

- **Zeile 2–3:** Das Buch wird anhand der ID geladen. Wenn es nicht existiert, wird eine Ausnahme geworfen.
- **Zeile 5:** Es wird geprüft, ob das Buch verfügbar ist und noch nicht vom Benutzer ausgeliehen wurde.
- **Zeile 7:** Alle bisherigen Ausleihen des Benutzers werden geladen.
- **Zeile 8:** Es wird überprüft, ob überfällige Bücher dabei sind.
- **Zeile 10:** Die Zahlungsinformationen des Benutzers werden geladen.
- **Zeile 12–14:** Falls offene Zahlungen oder überfällige Bücher vorhanden sind, wird eine Sperre ausgelöst.
- **Zeile 16–18:** Wenn kein Zahlungseintrag vorhanden ist, wird einer mit 0 Euro erstellt.
- **Zeile 20:** Der Buchbestand wird um eins reduziert.
- **Zeile 21:** Ein neuer Ausleihdatensatz wird erstellt.
- **Zeile 23:** Das Buchobjekt wird zurückgegeben.

Hilfsmethoden:

```

1     private void checkAvailability(Book book, String userEmail) {
2         if (checkoutRepository.findByIdAndUserEmail(book.getId(),
3             userEmail) != null) {
4             throw new BookNotAvailableException("The book has already
5                 been borrowed.");
6         }
7         if (book.getCopiesInStock() <= 0) {
8             throw new BookNotAvailableException("No copies available
9                 for loan.");
10        }
11    }

```

Listing 6.3: Überprüfung der Verfügbarkeit eines Buches

Diese Methode 6.3 überprüft, ob das angegebene Buch bereits von dem betreffenden Benutzer ausgeliehen wurde. Falls dies zutrifft, wird eine entsprechende Ausnahme ausgelöst, um eine doppelte Ausleihe zu verhindern. Zusätzlich wird geprüft, ob noch verfügbare Exemplare des Buches vorhanden sind. Ist dies nicht der Fall, wird ebenfalls eine Ausnahme ausgelöst, sodass keine Ausleihe ohne Bestand erfolgen kann.

```

1     private boolean hasOverdueBooks(List<Checkout> checkouts) {
2         LocalDate today = LocalDate.now();
3
4         for (Checkout checkout : checkouts) {
5             LocalDate returnDate = LocalDate.parse(checkout.getReturnedAt());
6             if (returnDate.isBefore(today)) {
7                 return true;
8             }
9         }
10        return false;
11    }

```

Listing 6.4: Prüfung auf überfällige Ausleihen

Diese Methode 6.4 überprüft, ob in der übergebenen Liste von Checkout-Einträgen mindestens ein Buch enthalten ist, dessen Rückgabedatum vor dem heutigen Datum liegt – also überfällig ist.

```

1  private void createZeroPayment(String userEmail) {
2      Payment newPayment = Payment.builder()
3          .userEmail(userEmail)
4          .amount(0.0)
5          .build();
6      paymentRepository.save(newPayment);
7  }

```

Listing 6.5: Erstellung einer Nullzahlung mit createZeroPayment()

Diese Methode 6.5 erstellt einen neuen Zahlungseintrag mit dem Betrag 0.0 für die angegebene Benutzer-E-Mail und speichert sie in der Datenbank über das PaymentRepository.

```

1  private void decrementBookStock(Book book) {
2      book.setCopiesInStock(book.getCopiesInStock() - 1);
3      bookRepository.save(book);
4  }

```

Listing 6.6: Reduzierung des Buchbestands

Diese Methode 6.6 reduziert den Lagerbestand des übergebenen Buchs um eins und speichert die Änderung in der Datenbank.

```

1  private void createCheckoutRecord(String userEmail, Book book) {
2      Checkout checkout = new Checkout(userEmail, LocalDate.now().
3          toString(),
4          LocalDate.now().plusDays(CHECKOUT_PERIOD_DAYS).toString(), book.
5          getId());
6      checkoutRepository.save(checkout);
7  }

```

Listing 6.7: Erstellung eines Ausleihdatensatzes

Diese Methode 6.7 erstellt einen neuen Ausleihdatensatz für das angegebene Buch und den Benutzer mit dem aktuellen Datum und speichert ihn in der Datenbank.

Schnittstelle zum Frontend: BookController

Der REST-Endpunkt (siehe 6.8) empfängt Anfragen zur Ausleihe und leitet sie an den Service weiter:

```

1  @PutMapping("/secure/loans/checkout")
2  public Book checkoutBook(Authentication authentication,
3      @RequestParam Long bookId) throws Exception {
4      String userEmail = authentication.getName();
5      return bookService.checkoutBook(userEmail, bookId);
6  }

```

Listing 6.8: REST-Endpunkt checkoutBook() im BookController

- `@PutMapping`: Definiert den Pfad zum Ausleih-Endpunkt.
- `Authentication`: Ermöglicht Zugriff auf die Benutzerinformationen über Spring Security.
- Die Methode ruft den Ausleihprozess im Service auf und gibt das ausgeliehene Buch zurück.

6.1.2 Frontend-Prozess

Im Frontend wird der Ausleihprozess in der Komponente `CheckoutBook` umgesetzt. Hierzu werden die Methode `checkoutBook()` für den API-Aufruf und die Konstante `checkoutButton` zur Darstellung der passenden Benutzeroberfläche verwendet.

API-Aufruf zur Ausleihe eines Buches

Die folgende Methode (siehe 6.9) übernimmt den API-Aufruf an das Backend, um ein Buch auszuleihen:

```

1  async function checkoutBook() {
2      const apiUrl = `${process.env.REACT_APP_API_URL}/books/secure/
3          loans/checkout?bookId=${bookId}`;
4      const response = {
5          method: 'PUT',
6          headers: {
7              Authorization: `Bearer ${authState?.accessToken?.
8                  accessToken}`,
9              'Content-Type': 'application/json'
10         }
11     };
12     const res = await fetch(apiUrl, response);
13     if (!res.ok) {
14         setShowError(true);
15         return;
16     }
17     setShowError(false);
18     setIsBookCheckedOut(true);
19 }

```

Listing 6.9: Implementierung der checkoutBook()-Funktion in CheckoutBook.tsx

Erklärung:

- `apiUrl`: Baut die URL für den API-Endpunkt mit dem Buch-ID als Parameter.
- `response`: Enthält die Methode (PUT) und den Authentifizierungs-Token.
- `fetch()`: Sendet die Anfrage an das Backend.
- `!res.ok`: Falls der Server einen Fehler zurückgibt, wird ein Fehler angezeigt.
- `setShowError(false)`: Versteckt die Fehlermeldung, falls alles korrekt lief.
- `setIsBookCheckedOut(true)`: Setzt den Status, dass das Buch nun ausgeliehen ist.

Benutzeroberfläche: Auswahl der richtigen Aktion

Die Konstante `checkoutButton` (siehe Listing B.1) bestimmt dynamisch, welcher Button oder welche Nachricht dem Benutzer im Ausleihbereich angezeigt wird.

- **Zeilen 1–2:** `useMemo(() => { ... }, [...])` stellt sicher, dass die Berechnung des Buttons nur dann neu ausgeführt wird, wenn sich die abhängigen Werte ändern (`props.isAuthenticated`, `props.isCheckedOut`, `props.currentLoans`, `props.checkoutBook`, `t`).
- **Zeilen 3–8:** Wenn der Benutzer nicht authentifiziert ist (`!props.isAuthenticated`), wird ein Link-Element zur Login-Seite zurückgegeben, das den Text **Anmelden** anzeigt.
- **Zeilen 10–31:** Wenn das Buch noch nicht ausgeliehen wurde (`!props.isCheckedOut`) und der Benutzer noch weniger als 5 Bücher ausgeliehen hat (`props.currentLoans < 5`):
 - **Zeilen 12–20:** Überprüft, ob keine Exemplare verfügbar sind (`!props.book?.copiesInStock || props.book.copiesInStock <= 0`). Ist dies der Fall, wird ein deaktivierter Button angezeigt.
 - **Zeilen 22–31:** Andernfalls wird ein aktiver Checkout-Button zurückgegeben, der beim Klicken die Methode `props.checkoutBook()` ausführt. Der Button zeigt ebenfalls **Ausleihen** an.
- **Zeilen 33–40:** Wenn das Buch bereits ausgeliehen ist (`props.isCheckedOut`), wird eine Erfolgsmeldung in einem `<p>`-Element angezeigt, inklusive eines grünen Häkchen-Icons und des Textes **Bereits ausgeliehen**.
- **Zeilen 42–47:** In allen anderen Fällen, z. B. wenn der Benutzer die maximale Anzahl an Ausleihen erreicht hat, wird eine Warnmeldung in einem `<p>`-Element mit rotem Icon und dem Text **Die maximale Anzahl an Checkouts wurde erreicht** angezeigt.

- **Zeilen 49–54:** Die Werte `props.isAuthenticated`, `props.isCheckedOut`, `props.currentLoans`, `props.checkoutBook` und `t` werden als Abhängigkeiten übergeben, sodass die Berechnung des Buttons nur erneut erfolgt, wenn sich einer dieser Werte ändert. Sie repräsentieren den Authentifizierungsstatus, den Ausleihstatus des Buches, die Anzahl der aktuellen Ausleihen, die Funktion zum Ausleihen eines Buches sowie die Übersetzungsfunktion.

Auf diese Weise sorgt `checkoutButton` dafür, dass die Benutzeroberfläche stets den aktuellen Status des Benutzers und des Buches korrekt darstellt, während `useMemo` unnötige Neuberechnungen verhindert und die Performance optimiert.

Einbindung des Buttons in die Oberfläche

Dieses `checkoutButton`-Konstantenobjekt wird anschließend innerhalb des `return()`-Statements aufgerufen, als `{checkoutButton}`, innerhalb der Datei `ReviewCheckoutPanel.tsx`.

6.2 Verfahren zur Rückgabe eines Buches

Im Folgenden wird das Verfahren zur Rückgabe eines Buches sowohl im Backend als auch im Frontend erläutert.

6.2.1 Backend-Prozess

Der Rückgabeprozess nutzt die bereits beschriebenen Repositories 6.1 und 5.12.

Geschäftslogik: Die Methode `returnBook` im `BookService.java`

Die wesentliche Logik für die Rückgabe eines Buches ist in der Methode `returnBook` der Datei `BookService.java` implementiert (siehe Listing B.2).

Erklärung:

- **Zeilen 1–3:** Das Buch mit der angegebenen `bookId` wird aus der Datenbank abgerufen. Existiert das Buch nicht, wird eine Ausnahme ausgelöst, die signalisiert, dass das Buch nicht verfügbar ist.

- **Zeilen 5–8:** Es wird nach einem aktiven Ausleihdatensatz (**Checkout**) für dieses Buch und diesen Benutzer gesucht. Falls kein Eintrag gefunden wird, wird eine Ausnahme ausgelöst, um eine ungültige Rückgabe zu verhindern.
- **Zeilen 10–11:** Der Lagerbestand des Buchs wird um eins erhöht und die geänderte Buch-Entität in der Datenbank gespeichert.
- **Zeilen 13–15:** Das Rückgabedatum des Ausleihdatensatzes wird geparsst, das aktuelle Datum ermittelt und die Anzahl der überfälligen Tage berechnet.
- **Zeilen 17–28:** Wenn das Buch überfällig ist, wird der Zahlungsdatensatz des Benutzers abgerufen. Existiert kein Eintrag, wird ein neuer Zahlungsdatensatz mit 0.0 erstellt. Der Betrag wird basierend auf den überfälligen Tagen (2 Einheiten pro Tag) aktualisiert und gespeichert.
- **Zeile 30:** Der Ausleihdatensatz wird aus der Datenbank gelöscht, wodurch das Buch offiziell als zurückgegeben markiert wird.
- **Zeilen 32–42 (Historie erfassen):** Ein Historieneintrag für diese Ausleihe wird erstellt, einschließlich Buchinformationen, Ausleihdatum und Rückgabedatum. Dieser Eintrag wird gespeichert, um eine dauerhafte Aufzeichnung der Transaktion zu gewährleisten.

Rückgabe-Endpunkt im Controller

Die Controller-Methode(siehe 6.10) ist für die Entgegennahme der Anfrage aus dem Frontend zuständig:

```

1  @PutMapping("/secure/loans/return")
2  public void returnBook(Authentication authentication,
3  @RequestParam Long bookId) throws ParseException {
4      String userEmail = authentication.getName();
5      bookService.returnBook(userEmail, bookId);
6  }

```

Listing 6.10: returnBook() Endpoint in BookController.java

Erklärung:

- Der Endpunkt verarbeitet eine PUT-Anfrage mit dem Buch-ID als Parameter.
- Die E-Mail des Benutzers wird über das Authentication-Objekt extrahiert.
- Der Service übernimmt die Logik für die Rückgabe.

6.2.2 Frontend-Prozess

Die Rückgabe eines Buches erfolgt durch einen Button innerhalb der Benutzeroberfläche der ausgeliehenen Bücher. Der Button ist wie folgt definiert (siehe 6.11):

```

1      <button
2        onClick={() => props.returnBook(props.userLoanSummary.book.id)}
3        type="button"
4        data-bs-dismiss="modal"
5        className="btn btn-outline-success rounded-pill py-1 px-2 mb-2
6          small"
7      >
8        {t("loanDetails.returnBook")}
9      </button>

```

Listing 6.11: returnBook-Button in LoanDetailsModal.tsx

Erklärung:

- `onClick={() => props.returnBook(props.userLoanSummary.book.id)}`: Beim Klick wird die Funktion `returnBook()` mit der `bookId` des ausgeliehenen Buches aufgerufen.
- `data-bs-dismiss="modal"`: Schließt das Bootstrap-Modal nach dem Klick automatisch.
- `className="..."`: Definiert das Styling des Buttons (grün, umrandet, abgerundet, klein).
- `t("loanDetails.returnBook")`: Holt den lokalisierten Text für „Buch zurückgeben“ aus der Übersetzungsdatei.

Beim Klicken auf den Button wird die Methode `returnBook()` (siehe 6.12) aufgerufen, welche als `async` Funktion in der Datei `loans.tsx` definiert ist:

```

1      async function returnBook(bookId: number) {
2        const apiUrl = `${process.env.REACT_APP_API_URL}/books/secure/
3          loans/return?bookId=${bookId}`;
4        const requestOptions = {
5          method: 'PUT',
6          headers: {
7            Authorization: `Bearer ${authState?.accessToken?.
8              accessToken}`,
9            'Content-Type': 'application/json'
10         }
11       };
12       const data = await fetch(apiUrl, requestOptions);
13       if (!data.ok) {
14         throw new Error('Something went wrong while returning the
15           book!');
16       }
17       setCheckout(!checkout);
18     }

```

Listing 6.12: returnBook() in Loans.tsx

Erklärung:

- `const apiUrl = ...`: Erstellt die API-URL mit Query-Parameter `bookId`, um das Rückgabe-Ende im Backend anzusprechen.
- `const requestOptions = {}`: Konfiguriert die HTTP-Anfrage mit Methode und Headern.
- `method: 'PUT'`: Gibt an, dass es sich um eine PUT-Anfrage handelt (für Rückgabe geeignet).
- `Authorization: Bearer ...`: Hängt das JWT-Token im Header an, um die Anfrage zu authentifizieren.
- `'Content-Type': 'application/json'`: Gibt das Format der übertragenen Daten an.
- `const data = await fetch(...)`: Führt die Anfrage asynchron aus und wartet auf die Antwort.
- `if (!data.ok)`: Überprüft, ob ein Fehler vom Server zurückgegeben wurde.
- `throw new Error(...)`: Löst bei Fehlern eine Ausnahme mit passender Meldung aus.
- `setCheckout(!checkout)`: Aktualisiert den `checkout`-State, um die Oberfläche neu zu laden bzw. den Rückgabeszustand zu reflektieren.

Beispiel

Dieses Kapitel bietet einen umfassenden Überblick über die Funktionalität und Benutzerfreundlichkeit der Anwendung. Anhand einer Reihe detaillierter Screenshots wird die Bedienung des Systems anschaulich dargestellt und analysiert.

7.1 Startseite

Die Startseite der Anwendung dient als zentraler Einstiegspunkt für alle Nutzergruppen. Sie bietet einen Überblick über die wichtigsten Funktionen der App, wie die Buchsuche, aktuelle Empfehlungen und den Zugang zu weiterführenden Diensten. Die Benutzeroberfläche ist übersichtlich gestaltet und ermöglicht eine intuitive Navigation durch die verschiedenen Inhalte. In diesem Teil wird gezeigt, wie die Startseite strukturiert ist und welche interaktiven Elemente den Nutzerinnen und Nutzern zur Verfügung stehen.

7.1.1 Kopfzeile

Die folgende Abbildung 7.1 zeigt den Aufbau des Headers mit dem dynamischen Navigationsmenü, das sich je nach Benutzerrolle unterscheidet:



Abbildung 7.1: Kopfzeile der App *Libranova*

- **Besucher:** Anzeige des *Libranova*-Logos sowie der Menüpunkte **Startseite**, **Über uns** und **Bücher Suchen**. Rechts oben befindet sich ein **Login**-Button. Außerdem steht ein Dropdown-Menü zur Auswahl der Sprache (Deutsch oder Englisch) zur Verfügung.

- **Eingeloggte Nutzer:** Zusätzlich zu den Besucher-Menüpunkten sind **Bibliotheksaktivität** (Einblick in ausgeliehene Bücher und Ausleihhistorie), **Überfällige Gebühren** (Anzeige möglicher Gebühren für verspätete Rückgaben) sowie **Posteingang** (Nachrichten senden und beantwortete Nachrichten einsehen) sichtbar. Rechts oben wird ein **Logout**-Button angezeigt.
- **Administratoren:** Ein **Admin**-Link erscheint für den Zugriff auf den Buchbestand und die Bearbeitung von Anfragen.

7.1.2 Footer

Die folgende Abbildung 7.2 zeigt die Fußzeile der Anwendung. Sie enthält das Copyright © Libranova App, Inc sowie die Menüpunkte **Startseite**, **Bücher suchen**, **Über uns** und **Barrierefreiheit**.



Abbildung 7.2: Fußzeile der App *Libranova*

7.1.3 Karussell und Buch-Browser

Die folgende Abbildung 7.3 zeigt ein Karussell zur Darstellung ausgewählter Bücher sowie einen Button, der zur Suchseite für Bücher weiterleitet.

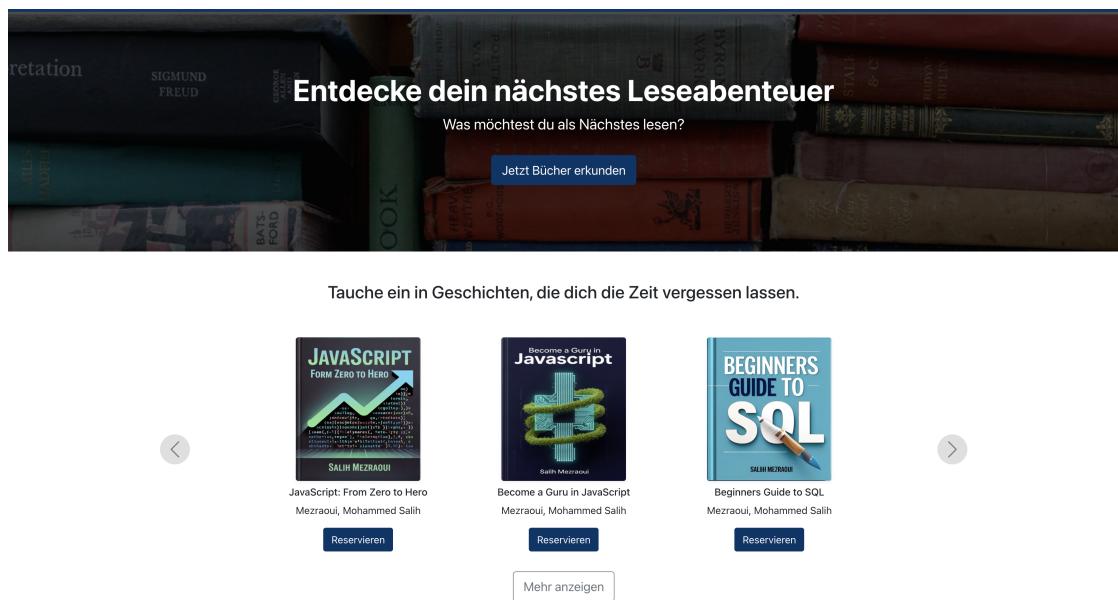


Abbildung 7.3: Karussell und Buch-Browser-Schaltfläche

7.2 Seitenübersicht

Dieser Abschnitt gibt einen Überblick über zentrale Benutzeroberflächen der Anwendung, insbesondere die Buchsuchseite und die Buchdetailseite, und beschreibt deren wichtigste Funktionselemente.

7.2.1 Buchsuche

Die untenstehende Abbildung 7.4 zeigt die Benutzeroberfläche der Suchseite. Sie enthält ein Suchfeld mit Schaltfläche sowie ein Dropdown-Menü zur Auswahl von Kategorien. Nutzer können entweder nach Stichwörtern, Kategorien oder einer Kombination aus beiden suchen. Die Suchergebnisse werden in paginierter Form dargestellt und beinhalten jeweils den Buchtitel, den Autor, eine Kurzbeschreibung, Verfügbarkeit sowie eine Schaltfläche zur Detailansicht des jeweiligen Buches.

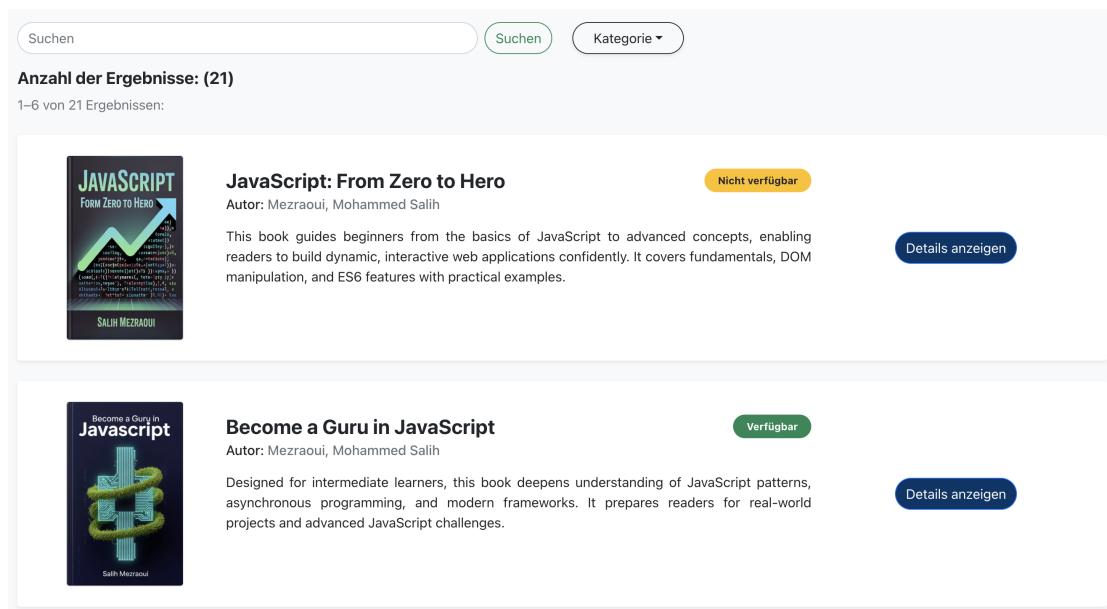


Abbildung 7.4: Benutzeroberfläche der Suchseite

7.2.2 Buchseite

Die untenstehende Abbildung 7.5 zeigt die Benutzeroberfläche der Buchseite.

Introduction to Spring Boot

Mezraoui, Mohammed Salih

Show English

Dieses Buch stellt die Grundlagen von Spring Boot vor und befähigt Entwickler, robuste und skalierbare Java-Anwendungen schnell zu erstellen. Es behandelt Einrichtung, Dependency Injection, REST-APIs und Datenzugriff mit praxisnahen Anleitungen.

★★★★★

5/5 ausgeliehene Bücher

Nicht verfügbar

0 Exemplare 0 Verfügbar

⚠ Du hast die maximale Anzahl an ausgeliehenen Büchern erreicht.

Diese Zahl kann sich bis zur Bestellung noch ändern.

Eine Bewertung schreiben ▾

★★★★★

Neueste Rezensionen

testuser@email.com 10. August 2025

Great Book!! I would Absolutely recommend it

★★★★★

[Alle Rezensionen anzeigen >](#)

Abbildung 7.5: Benutzeroberfläche der Buchseite

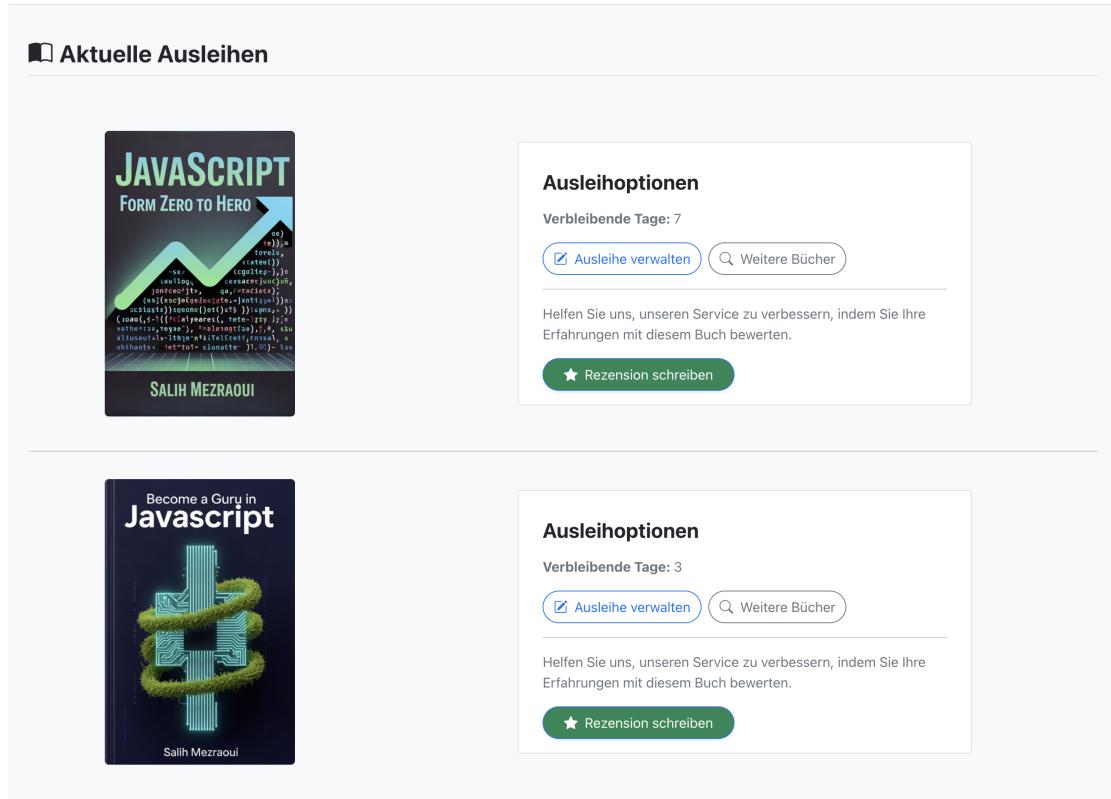
Die Buchdetailseite stellt umfassende Informationen zu einem einzelnen Buch bereit. Sie zeigt das Buchcover, den Titel sowie den Autor an und bietet eine Buchbeschreibung, bei der zwischen deutscher und englischer Sprache gewechselt werden kann, unabhängig von der gewählten Sprache der Anwendung. Zudem enthält die Seite ein Bewertungssystem zur Anzeige der durchschnittlichen Nutzerbewertung. Auf der rechten Seitenleiste werden Informationen wie die Anzahl der vom aktuellen Benutzer ausgeliehenen Exemplare, die Anzahl der derzeit verfügbaren Exemplare sowie die Gesamtanzahl der im Bestand befindlichen Exemplare angezeigt. Zusätzlich bietet die Seite eine Übersicht der Nutzerbewertungen zum Buch sowie einen Link zur vollständigen Liste aller Reviews.

7.3 Bibliotheksaktivität

In diesem Abschnitt werden die Funktionen zur Verwaltung aktueller und vergangener Ausleihen dargestellt.

7.3.1 Ausleihen

Die untenstehende Abbildung 7.6 zeigt das Design der aktuellen **Ausleihen** Seite.



The screenshot displays a library loan interface with two book entries. Each entry includes the book cover, title, author, loan details, and a 'Borrow Options' section.

Book 1: JavaScript Form Zero to Hero

- Cover:** The cover features the title 'JAVASCRIPT FORM ZERO TO HERO' and the author 'SALIH MEZRAOUI'.
- Title:** JavaScript Form Zero to Hero
- Author:** SALIH MEZRAOUI
- Borrow Options:**
 - Verbleibende Tage: 7
 - Ausleihe verwalten
 -
- Review:** ★ Rezension schreiben

Book 2: Become a Guru in Javascript

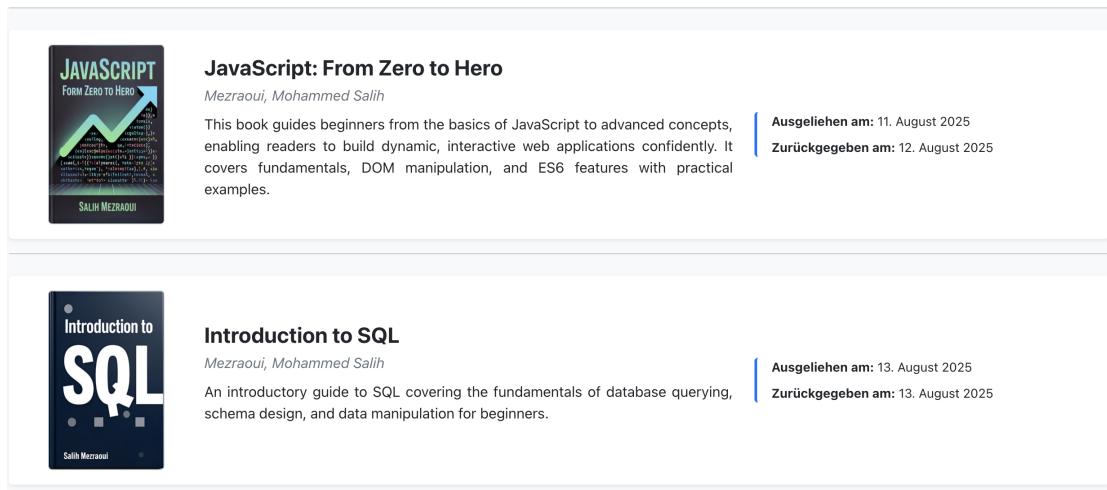
- Cover:** The cover features the title 'Become a Guru in Javascript' and the author 'Salih Mezraoui'.
- Title:** Become a Guru in Javascript
- Author:** Salih Mezraoui
- Borrow Options:**
 - Verbleibende Tage: 3
 - Ausleihe verwalten
 -
- Review:** ★ Rezension schreiben

Abbildung 7.6: Benutzeroberfläche der Ausleihseite

Die Ausleiheseite ermöglicht es Nutzenden, ihre aktuell ausgeliehenen Bücher zu verwalten. Sie zeigt eine Liste aller ausgeliehenen Bücher an und gibt für jedes Buch die verbleibenden Tage bis zur Rückgabe an. Für jedes Buch stehen Verwaltungsoptionen wie die Rückgabe oder die Verlängerung der Leihfrist zur Verfügung. Zusätzlich enthält die Seite eine Schaltfläche zur Suche nach weiteren Büchern sowie einen Link, um eine Rezension für das jeweilige Buch zu verfassen.

7.3.2 Ausleihhistorie

Die untenstehende Abbildung 7.7 zeigt das Design der Seite **Ausleihverlauf**. Diese Seite enthält die Historie der vom Nutzer ausgeliehenen Bücher, einschließlich des Ausleih- und Rückgabedatums.



The screenshot displays a library loan history page with two entries. The first entry is for the book 'JavaScript: From Zero to Hero' by Mezraoui, Mohammed Salih. The second entry is for 'Introduction to SQL' by Mezraoui, Mohammed Salih. Both entries show the loan date as 11. August 2025 and the return date as 12. August 2025.

Book Title	Author	Borrowed On	Due On
JavaScript: From Zero to Hero	Mezraoui, Mohammed Salih	11. August 2025	12. August 2025
Introduction to SQL	Mezraoui, Mohammed Salih	13. August 2025	13. August 2025

Abbildung 7.7: Benutzeroberfläche der Ausleihverlaufsseite

7.4 Bibliotheksdienste

In diesem Abschnitt werden die Bibliotheksdienste vorgestellt, insbesondere die Benutzeroberfläche zur Übermittlung von Anfragen sowie der persönliche Nachrichtenverlauf mit den jeweiligen Antworten.

Die folgende Abbildung 7.8 zeigt die Benutzeroberfläche, über die Benutzer einzelne Anfragen oder Nachrichten an die Administratoren der Bibliothek senden können.

The screenshot shows a user interface for sending a new message. At the top, a dark header bar contains the text 'Neue Nachricht senden' in white. Below this, the form is divided into two main sections: 'Betreff' (Subject) and 'Anfrage' (Query). The 'Betreff' section contains a text input field with the placeholder 'Betreff eingeben'. The 'Anfrage' section contains a larger text input field with the placeholder 'Schreiben Sie Ihre Nachricht...'. At the bottom of the form is a blue button with the text 'Nachricht absenden' (Send message) in white.

Abbildung 7.8: Benutzeroberfläche zum Versenden von Anfragen

Die Abbildung 7.9 veranschaulicht den Verlauf sämtlicher ausgetauschter Nachrichten, einschließlich der gestellten Fragen und der dazugehörigen Antworten.



Abbildung 7.9: Nachrichtenverlauf zwischen Nutzer und Bibliothek

7.5 Bezahlungsseite

In diesem Abschnitt wird die Benutzeroberfläche zur Verwaltung von Zahlungsinformationen und offenen Gebühren vorgestellt.

Abbildung 7.10 zeigt das Layout, das angezeigt wird, wenn ein Benutzer ausstehende Zahlungen zu begleichen hat.

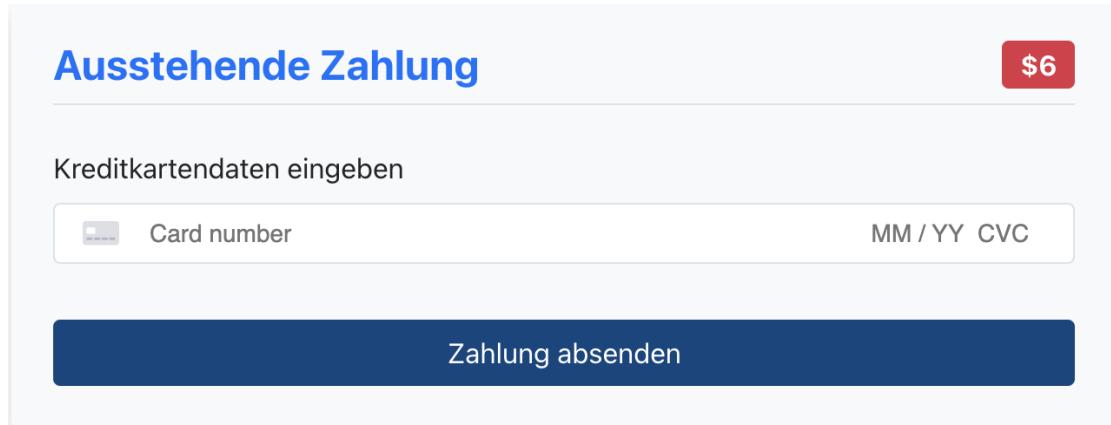


Abbildung 7.10: Benutzeroberfläche bei ausstehenden Zahlungen

Wenn keine offenen Gebühren vorliegen, wird dem Nutzer die in Abbildung 7.11 dargestellte Ansicht präsentiert. Sie bestätigt, dass derzeit keine Zahlungen erforderlich sind.

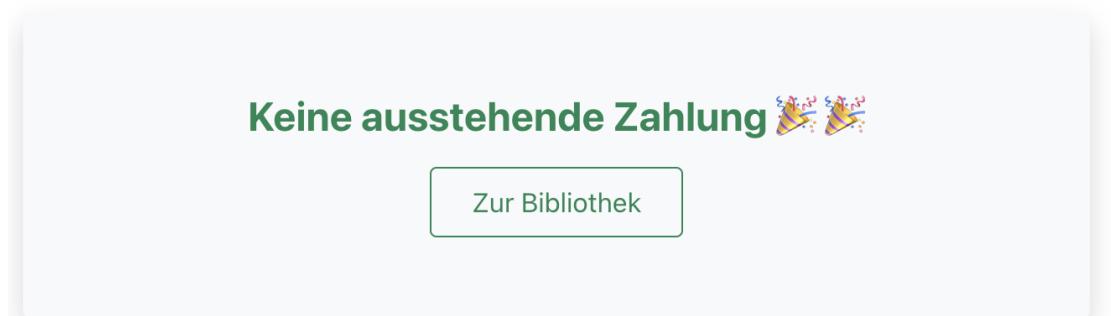


Abbildung 7.11: Benutzeroberfläche bei keinen offenen Zahlungen

7.6 Admin-Bereich

Der Admin-Bereich bietet eine dedizierte Oberfläche zur Verwaltung der Bibliotheksressourcen und zur Interaktion mit Benutzeranfragen. In diesem Abschnitt werden die Verwaltungsfunktionen dargestellt, die einem Administrator zur Verfügung stehen.

7.6.1 Neues Buch hinzufügen

Die erste Funktion (siehe 7.12) ermöglicht es dem Administrator, neue Bücher in das System aufzunehmen. Hierzu gibt er relevante Informationen wie Titel, Autor, Kategorie, eine kurze Beschreibung sowie die Anzahl der verfügbaren Exemplare an. Zusätzlich kann ein Bild des Buchcovers hochgeladen werden. Nach dem Ausfüllen der Felder wird durch Klicken auf die Schaltfläche *"Buch hinzufügen"* ein neuer Eintrag erstellt.

The screenshot shows a form titled "Neues Buch hinzufügen" (Add New Book). The form is divided into several sections: "Titel" (Title) with a text input field "Buchtitel eingeben" (Enter book title); "Autor" (Author) with a text input field "Name des Autors eingeben" (Enter author name); "Kategorie" (Category) with a dropdown menu "Kategorie auswählen" (Select category); "Überblick" (Overview) with a text input field "Buchüberblick eingeben" (Enter book overview); "Gesamtanzahl" (Total quantity) with a text input field "0" (0); and a file input field "addBook.coverImageLabel" with the placeholder "Datei auswählen" (Select file) and the message "Keine Datei ausgewählt" (No file selected). At the bottom right is a blue button with a plus sign and the text "Buch hinzufügen" (Add book).

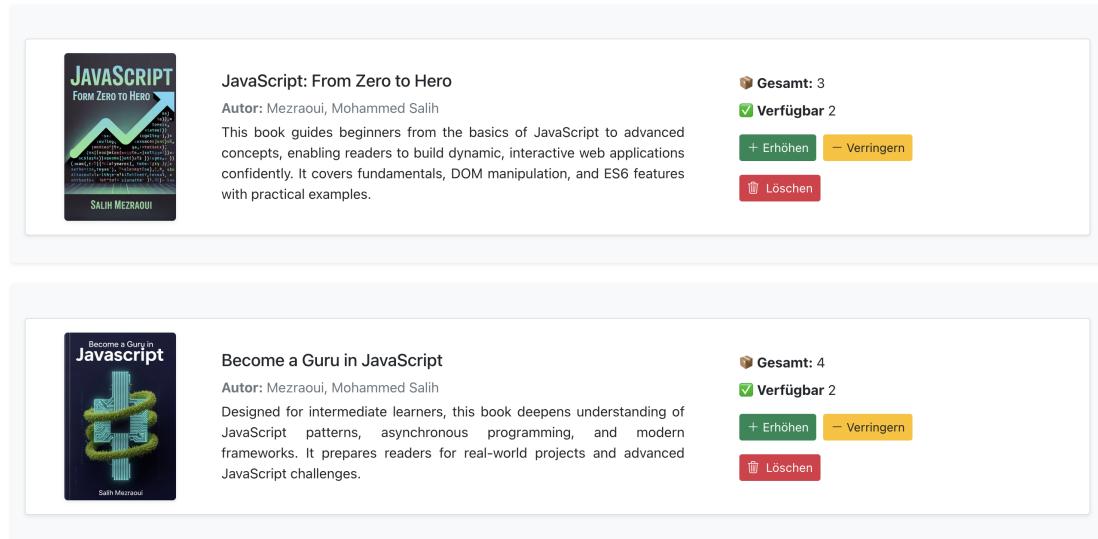
Abbildung 7.12: Das Design zum Hinzufügen eines neuen Buches

7.6.2 Bücher verwalten

Der Administrator kann bestehende Bücher verwalten, indem er die Anzahl der verfügbaren Exemplare anpasst oder Bücher vollständig aus dem System entfernt. Die folgende Abbildung 7.13 zeigt die Verwaltungsoberfläche, über die solche Änderungen vorgenommen werden können.

Gesamtanzahl Bücher: 21

1–6 von 21 Ergebnissen werden angezeigt



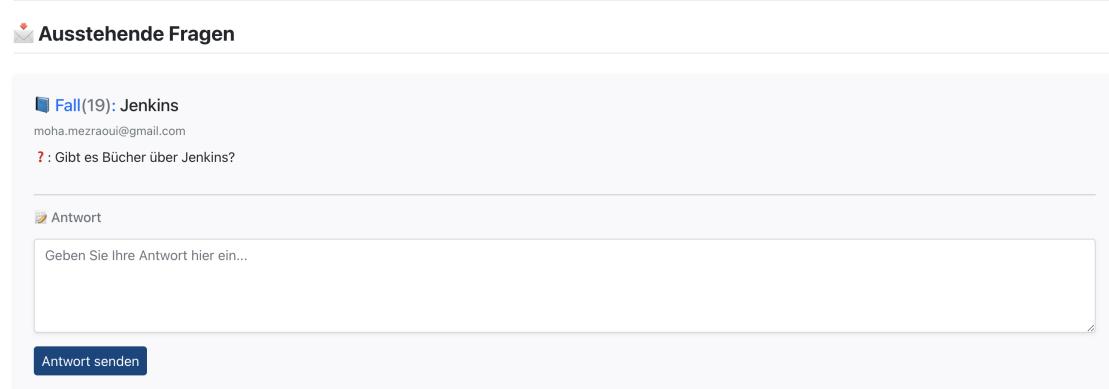
The screenshot shows a list of books in a management interface. Each book entry includes the book cover, title, author, a brief description, and a control panel with buttons for managing the book's availability and deleting it.

Book	Author	Description	Availability	Actions
JavaScript: From Zero to Hero	Mezraoui, Mohammed Salih	This book guides beginners from the basics of JavaScript to advanced concepts, enabling readers to build dynamic, interactive web applications confidently. It covers fundamentals, DOM manipulation, and ES6 features with practical examples.	Gesamt: 3 Verfügbar 2	+ Erhöhen - Verringern Löschen
Become a Guru in JavaScript	Mezraoui, Mohammed Salih	Designed for intermediate learners, this book deepens understanding of JavaScript patterns, asynchronous programming, and modern frameworks. It prepares readers for real-world projects and advanced JavaScript challenges.	Gesamt: 4 Verfügbar 2	+ Erhöhen - Verringern Löschen

Abbildung 7.13: Das Design der Buchverwaltung

7.6.3 Nachrichten

In diesem Bereich kann der Administrator auf Anfragen von Benutzern antworten. Die Benutzerfrage wird angezeigt und kann direkt über das vorgesehene Textfeld beantwortet werden. Eine Schaltfläche ermöglicht das Senden der Antwort. Die Abbildung 7.14 zeigt die zugehörige Benutzeroberfläche.



The screenshot shows a message interface. It displays an incoming message from 'Jenkins' with a question about books. Below it is a text input field for the administrator's response, with a 'Senden' (Send) button at the bottom.

Frage	Aktion
Fall(19): Jenkins moha.mezraoui@gmail.com ?: Gibt es Bücher über Jenkins?	Antwort Geben Sie Ihre Antwort hier ein... Antwort senden

Abbildung 7.14: Die Schnittstelle für die Beantwortung von Anfragen

Anwendungsszenarien

Die im Rahmen dieses Projekts entwickelte Anwendung zur Verwaltung von Büchern ist flexibel einsetzbar und bietet zahlreiche Erweiterungsmöglichkeiten über den ursprünglichen Anwendungsfall hinaus. Dank ihrer modularen Architektur, der Nutzung von REST-APIs und der durchdachten Benutzeroberfläche lässt sich das System leicht an verschiedene Nutzungsszenarien anpassen.

Im Folgenden werden exemplarisch mehrere Anwendungsbereiche vorgestellt, in denen das System sinnvoll eingesetzt oder erweitert werden könnte. Dazu zählen insbesondere der Bildungsbereich und E-Learning-Plattformen, die mobile Nutzung durch Erweiterung zu nativen Apps sowie innovative Funktionen wie personalisierte Buchempfehlungen sowie Echtzeit-Reservierungsbenachrichtigungen. Diese Szenarien verdeutlichen das Potenzial des Systems, weit über die klassische Bibliotheksverwaltung hinaus einen Mehrwert für Nutzerinnen und Nutzer zu schaffen.

8.1 Bildungs- und E-Learning-Plattformen

E-Learning-Plattformen und Bildungseinrichtungen könnten das entwickelte System als Backend-Lösung zur Bereitstellung von digitalen Lernmaterialien wie Kursunterlagen, wissenschaftlichen Artikeln oder Videoinhalten nutzen. Die integrierte Bewertungs- und Kommentarfunktion ermöglicht es Lernenden, Inhalte zu bewerten und Rezensionen zu verfassen, was wiederum anderen Nutzerinnen und Nutzern bei der Auswahl geeigneter Materialien hilft. Das System könnte zudem erweitert werden, um PDF-Dateien, E-Books und Videos mit Download- oder Streamingrechten zu verwalten und bereitzustellen. Darüber hinaus könnten individuelle Lernpfade unterstützt werden, indem relevante Ressourcen personalisiert vorgeschlagen werden.

8.2 Mobile Nutzung

Durch die RESTful-Architektur der Anwendung ist eine einfache Erweiterung um eine mobile App möglich. Dies ermöglicht Nutzerinnen und Nutzern, bequem von unterwegs auf ihre ausgeliehenen Bücher, Rezensionen und weiteren Inhalten zuzugreifen. Die mobile Nutzung steigert die Flexibilität und Benutzerfreundlichkeit der Anwendung erheblich, indem sie den Zugriff jederzeit und überall ermöglicht – sei es auf iOS- oder Android-Geräten.

8.3 Personalisierte Empfehlungen

Eine zukünftige Erweiterung des Systems könnte die Implementierung KI-basierter Vorschläge umfassen, die Nutzerinnen und Nutzern auf Basis ihrer Lesehistorie und Präferenzen individuell zugeschnittene Buchempfehlungen anbieten. Dies könnte durch die Integration von Machine-Learning-Algorithmen realisiert werden, die das Nutzerverhalten analysieren. Dadurch würde die Nutzererfahrung verbessert und die Nutzung des Systems attraktiver gestaltet.

8.4 Reservierungsbenachrichtigungen

Das System kann erweitert werden, um Nutzern Echtzeit-Benachrichtigungen zu senden, sobald ein reserviertes Buch wieder verfügbar ist. Diese Funktion verbessert die Nutzerzufriedenheit, da Interessenten sofort informiert werden und somit ihre Ausleihe schneller planen können. Die Integration solcher Benachrichtigungen kann über E-Mail, Push-Nachrichten oder andere Kommunikationskanäle erfolgen.

Resümee und Ausblick

In dieser Arbeit wurde die Konzeption und Realisierung einer modernen, benutzerfreundlichen und sicheren Bibliotheksmanagement-Plattform namens LibraNova vorgestellt. Das Ziel war es, eine skalierbare und wartbare Lösung zu entwickeln, die sowohl den Anforderungen der Nutzerinnen und Nutzer als auch der Bibliothekadministration gerecht wird. Dabei stand die Integration aktueller Webtechnologien wie Spring Boot im Backend, React mit TypeScript im Frontend sowie eine umfassende Sicherheitsarchitektur mit Okta, JWT, OAuth2 und OpenID Connect im Fokus. Zudem wurde die Plattform um eine Zahlungsfunktion erweitert, die die Stripe API nutzt, um beispielsweise etwaige Gebühren reibungslos und sicher abzuwickeln.

Die Anwendung ist mehrsprachig ausgelegt und unterstützt sowohl Deutsch als auch Englisch, um eine breitere Nutzerbasis anzusprechen. Darüber hinaus legt das System großen Wert auf responsives Design, sodass die Plattform auf verschiedenen Endgeräten — sei es Desktop, Tablet oder Smartphone — optimal genutzt werden kann. Die Umsetzung erfolgte durch modulare Architekturen, die REST-APIs nutzen, um eine flexible Anpassung an verschiedene Nutzungsszenarien zu ermöglichen. Wesentliche Funktionen wie die Buchsuche, das Ausleihmanagement, Nutzerbewertungen und die administrative Verwaltung wurden effizient implementiert und durch systematische Tests abgesichert.

Die Plattform unterstützt zudem zukünftige Erweiterungen, etwa Cloud-Bereitstellung, erweiterte Testabdeckung sowie personalisierte Empfehlungen und Benachrichtigungen. Mit LibraNova wurde eine innovative Lösung geschaffen, die den digitalen Wandel im Bibliothekswesen aktiv mitgestaltet und die Nutzererfahrung durch intuitive Bedienung, hohe Sicherheitsstandards sowie eine responsive Gestaltung deutlich verbessert. Das Projekt bildet eine solide Grundlage für zukünftige Entwicklungen und bietet zahlreiche Potenziale zur weiteren Optimierung und Erweiterung.

Literaturverzeichnis

- Bel25. BELL, DONALD: *An introduction to the Unified Modeling Language*. <https://developer.ibm.com/articles/an-introduction-to-uml/>, 2025. Veröffentlicht am 25. Juni 2023, Download am 20.07.2025.
- Boo25. BOOTSTRAP AUTHORS: *Get started with Bootstrap*. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>, 2025. Download am 22. Juli 2025.
- Bra25. BRATSLAVSKY, PAUL: *Top 6 Benefits of Implementing TypeScript*. <https://strapi.io/blog/benefits-of-typescript>, 2025. Veröffentlicht am 22. Februar 2025, Download am 22. Juli 2025.
- Clo25. CLOUDFLARE: *What is HTTPS?* <https://www.cloudflare.com/learning/ssl/what-is-https/>, 2025. Download am 02. Juli 2025.
- Cor25a. CORPORATION, MICROSOFT: *TypeScript for JavaScript Programmers*. <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>, 2025. Download am 21. Juli 2025.
- Cor25b. CORPORATION, ORACLE: *MySQL Workbench Manual - General Information*. <https://dev.mysql.com/doc/workbench/en/wb-intro.html>, 2025. Download am 04. Juli 2025.
- Cor25c. CORPORATION, ORACLE: *What is MySQL?* <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>, 2025. Download am 04. Juli 2025.
- Fac25. FACEBOOK: *Create React App - Getting Started*. <https://create-react-app.dev/docs/getting-started/>, 2025. Download am 03.08.2025.
- Gmb25a. GMBH, THALIA BÜCHER: *Das ist Thalia*. <https://unternehmen.thalia.de/unternehmen/>, 2025. Download am 14. August 2025.

- Gmb25b. GMBH, THALIA BÜCHER: *Fremdsprachige Bücher*. <https://www.thalia.de/kategorie/fremdsprachige-buecher-880/>, 2025. Download am 14. August 2025.
- Gmb25c. GMBH, THALIA BÜCHER: *Lesen & hören so flexibel wie nie*. <https://www.thalia.de/vorteile/thalia-lesen-und-hoeren-app>, 2025. Download am 14. August 2025.
- GT25a. GIT-TEAM: *Git About - Branching and Merging*. <https://git-scm.com/about/branching-and-merging>, 2025. Download am 01. August 2025.
- GT25b. GITHUB-TEAM: *About GitHub*. <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git#about-github>, 2025. Download am 01. August 2025.
- Gup25. GUPTA, LOKESH: *What is REST?* <https://restfulapi.net/>, 2025. Veröffentlicht am 1. April 2025, Download am 19. Mai 2025.
- HT25. HIBERNATE-TEAM: *Idiomatic persistence for Java and relational databases*. <https://hibernate.org/orm/>, 2025. Download am 06.08.2025.
- HTM25a. HTML.COM: *HTML5 Basics For Everyone Tired Of Reading About Deprecated Code*. <https://html.com/html5/>, 2025. Download am 02.06.2025.
- HTM25b. HTML.COM: *Intimidated By CSS? The Definitive Guide To Make Your Fear Disappear*. <https://html.com/css/>, 2025. Download am 02.06.2025.
- i1825. i18NEXT ORGANISATION: *i18next documentation*. <https://www.i18next.com/>, 2025. Download am 2. August 2025.
- IBM25. IBM: *What is a relational database?* <https://www.ibm.com/think/topics/relational-databases>, 2025. Veröffentlicht am 20. Oktober 2021, Download am 04. Juli 2025.
- IRM⁺20. ISLAM, MAZHARUL, SAZZADUR RAHAMAN, NA MENG, BEHNAZ HASSANSHAH, PADMANABHAN KRISHNAN und DANFENG DAPHNE YAO: *Coding Practices and Recommendations of Spring Security for Enterprise Applications*. Seiten 49–57, 09 2020.
- JT25a. JUNIT-TEAM: *Junit 5 User Guide*. <https://docs.junit.org/current/user-guide/>, 2025. Download am 02. August 2025.
- JT25b. JWT-TEAM: *Introduction to JSON Web Tokens*. <https://jwt.io/introduction>, 2025. Download am 19. Juli 2025.

-
- MP25a. META PLATFORMS, INC.: *Build a React app from Scratch*. <https://react.dev/learn/build-a-react-app-from-scratch#step-2-build-common-application-patterns>, 2025. Download am 22. Juli 2025.
- MP25b. META PLATFORMS, INC.: *Describing the UI*. <https://react.dev/learn/describing-the-ui>, 2025. Download am 22. Juli 2025.
- MP25c. META PLATFORMS, INC.: *React — A JavaScript library for building user interfaces*. <https://legacy.reactjs.org/>, 2025. Download am 22. Juli 2025.
- MT25a. MEDIUM-TEAM: *Stripe Integration Using Flutter Web and NestJS*. <https://medium.com/%40akshelar.18119/stripe-integration-using-flutter-web-and-nestjs-4e94ccd33b9f>, 2025. Download am 15. August 2025.
- MT25b. MOCKITO-TEAM: *Tasty mocking Framework for unit tests in Java*. <https://site.mockito.org/>, 2025. Download am 02. August 2025.
- Ora25. ORACLE CORPORATION: *What is Java technology and why do I need it?* https://www.java.com/en/download/help/whatis_java.html, 2025. Download am 19. Mai 2025.
- OT25a. OKTA-TEAM: *Oauth 2.0 and OpenID Connect overview*. <https://developer.okta.com/docs/concepts/oauth-openid/>, 2025. Download am 15. August 2025.
- OT25b. OKTA-TEAM: *What is Okta and what does Okta do?* https://support.okta.com/help/s/article/what-is-okta?language=en_US, 2025. Download am 19. Mai 2025.
- OT25c. OPENID-TEAM: *How OpenId Connect Works*. <https://openid.net/developers/how-connect-works/>, 2025. Download am 19. Juli 2025.
- PT25. POSTMAN-TEAM: *What is Postman?* <https://www.postman.com/product/what-is-postman/>, 2025. Download am 02. August 2025.
- Pur25. PUROHIT, RAKESH: *A Step-by-Step Guide to Implementing React Spring Boot in Your Application*. <https://www.dhiwise.com/post/a-step-by-step-guide-to-implementing-react-spring-boot>, 2025. Veröffentlicht am 17. Juni 2025, Download am 23. Juli 2025.
- RHHH20. RAHMAN, SHAWON, NAZMUL HOSSAIN, MD. ALAM HOSSAIN und MD. ZOBAYER HOSSAIN: *OAuth 2.0: A Framework to Secure the OAuth-Based Service for Packaged Web Application*. Seiten 92–139, 01 2020.

- Spr25a. SPRING-TEAM: *Spring Boot*. <https://spring.io/projects/spring-boot>, 2025. Download am 15. Mai 2025.
- Spr25b. SPRING-TEAM: *Spring Boot*. <https://spring.io/projects/spring-boot>, 2025. Download am 15. Mai 2025.
- ST25a. SPRING-TEAM: *Spring Data JPA*. <https://spring.io/projects/spring-data-jpa>, 2025. Download am 06.08.2025.
- ST25b. SPRING-TEAM: *Spring Data Rest*. <https://spring.io/projects/spring-data-rest>, 2025. Download am 06.08.2025.
- ST25c. SPRING-TEAM: *Using the @SpringBootApplication Annotation*. <https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html>, 2025. Download am 16. August 2025.
- ST25d. STRIPE-TEAM: *Stripe API Documentation*. <https://docs.stripe.com/api>, 2025. Download am 03. Juli 2025.
- Tri25. TRIER, STADTBÜCHEREI: *Informationen für Sie*. <https://www.stadtbuecherei-trier.de/informationen-fuer-sie/>, 2025. Download am 14. August 2025.

A

Systemanforderungen

ID	Anforderung (Beschreibung)
FR1	Nutzer können Bücher nach Titel oder Kategorie suchen.
FR2	Nutzer können die Verfügbarkeit eines Buches in Echtzeit einsehen.
FR3	Nutzer können bis zu fünf Bücher gleichzeitig ausleihen.
FR4	Nutzer können ihre ausgeliehenen Bücher einsehen.
FR5	Nutzer können ihre Ausleihhistorie einsehen.
FR6	Nutzer können ein Buch bewerten.
FR7	Nutzer können eine Rezension verfassen.
FR8	Nutzer können Rezensionen anderer Nutzer lesen.
FR9	Das System zeigt eine Übersicht der verfügbaren Bücher im Katalog an.
FR10	Administratoren können neue Bücher zum Katalog hinzufügen.
FR11	Administratoren können die Anzahl der Exemplare eines vorhandenen Buches erhöhen oder verringern.
FR12	Administratoren können Bücher aus dem Katalog löschen.
FR13	Nutzer können sich in ihr persönliches Konto einloggen.
FR14	Nutzer können die Leihfrist eines Buches um bis zu 7 Tage verlängern.
FR15	Nutzer können ausgeliehene Bücher zurückgeben.
FR16	Das System zeigt an, wenn ein Buch nicht verfügbar ist.
FR17	Für kostenpflichtige Dienstleistungen ist eine Zahlungsabwicklung über Stripe integriert.
FR18	Nutzer müssen eingeloggt sein, um ein Buch auszuleihen.
FR19	Nutzer müssen eingeloggt sein, um eine Rezension verfassen zu können.
FR20	Nutzer müssen eingeloggt sein, um ein Buch bewerten zu können.
FR21	Das System zeigt an, wie viele Bücher von den maximal fünf gleichzeitig ausgeliehen sind.
FR22	Das System zeigt die Anzahl der Suchergebnisse basierend auf eingegebenen Suchbegriffen und gewählten Kategorien an.
FR23	Das System zeigt Bücherlisten (Suchergebnisse, ausgeliehene Bücher, Ausleihhistorie) paginiert an.

FR24	Das System muss sicherstellen, dass Nutzer sowohl einen Nachrichtentitel als auch den Nachrichtentext angeben, bevor sie eine Anfrage absenden können.
FR25	Das System muss sicherstellen, dass alle erforderlichen Felder beim Anlegen eines neuen Buches ausgefüllt sind.
FR26	Das System muss sicherstellen, dass Administratoren das Antwortfeld ausfüllen, bevor sie eine Antwort absenden können.
FR27	Das System muss im Ausleihverlauf eines Nutzers sowohl das Ausleihdatum als auch das Rückgabedatum jedes Buches anzeigen.
FR28	Bei verspäteter Rückgabe von Büchern soll das System die fälligen Gebühren automatisch berechnen und über die Stripe-API zur Zahlung auffordern.
FR29	Für jede Buchbeschreibung muss ein Button vorhanden sein, mit dem der Nutzer die Beschreibung zwischen Deutsch und Englisch umschalten kann.
FR30	Wenn ein Buch gelöscht wurde, darf der Benutzer die Ausleihe dieses Buches nicht mehr verlängern können.
FR31	Wenn ein Buch gelöscht wird, muss der Benutzer in der Ausleihliste darüber informiert werden, dass dieses Buch gelöscht wird.

Tabelle A.1: Funktionale Anforderungen von LibraNova

ID	Anforderung (Beschreibung)
NFR1	Das System muss eine responsive Benutzeroberfläche bieten, die auf verschiedenen Geräten und Bildschirmgrößen funktioniert.
NFR2	Die Anwendung muss eine sichere Kommunikation über HTTPS (TLS) gewährleisten, um die Datenübertragung zwischen Frontend und Backend zu schützen (Ports 8443 und 3000).
NFR3	Das System muss sicherstellen, dass Benutzer nur mit gültigen und nicht abgelaufenen Tokens Zugriff auf geschützte Ressourcen erhalten.
NFR4	Die Reaktionszeit der Such- und Filterfunktionen im Frontend darf 2 Sekunden nicht überschreiten, um ein flüssiges Nutzererlebnis zu gewährleisten.
NFR5	Die Anwendung muss grundlegende Barrierefreiheitsanforderungen gemäß WCAG 2.1 (Level AA) erfüllen, um auch Nutzern mit Einschränkungen einen uneingeschränkten Zugang und eine barrierefreie Nutzung zu ermöglichen.
NFR6	Die REST-APIs müssen robust gegen fehlerhafte Eingaben sein und validierte Anfragen verarbeiten, um die Stabilität des Systems zu gewährleisten.
NFR7	Die Backend-Services sollen modular aufgebaut sein, um einfache Wartbarkeit und Erweiterbarkeit zu ermöglichen.
NFR8	Die API-Dokumentation muss aktuell und entwicklerfreundlich sein.

ID	Anforderung (Beschreibung)
NFR9	Die Anwendung muss eine klare Trennung zwischen Benutzerrollen (Nutzer, Administrator) ermöglichen.
NFR10	Die Anwendung muss auf allen gängigen Webbrowersn wie Firefox, Chrome und Safari ohne Einschränkungen lauffähig sein.
NFR11	Der Anmeldevorgang eines Nutzers (ab dem Klick auf den Login-Button bis zur erfolgreichen Authentifizierung und Weiterleitung zur Startseite) darf nicht länger als 3 Sekunden dauern.
NFR12	Die Benutzeroberfläche muss auf allen Seiten konsistente Navigationselemente, Schaltflächen-Designs und Layouts verwenden, um eine einheitliche Nutzererfahrung sicherzustellen.
NFR13	Die Benutzerführung muss intuitiv gestaltet sein, sodass typische Nutzeraktionen wie Buchsuche, Ausleihe oder Bewertung ohne zusätzliche Anleitung verständlich und ausführbar sind.
NFR14	Das System muss sicherstellen, dass nur autorisierte Clients auf geschützte Endpunkte wie <code>/api/books/secure/**</code> oder <code>/api/admin/secure/**</code> zugreifen können.
NFR15	Die Anwendung muss vollständig auf Deutsch und Englisch verfügbar sein, wobei alle Seiten und Benutzeroberflächen entsprechend übersetzt werden.

Tabelle A.2: Nicht-funktionale Anforderungen von LibraNova

ID	Anforderung (Beschreibung)
TR1	Das Backend muss HTTPS-Verbindungen über Port 8443 ermöglichen, mithilfe eines selbstsignierten SSL-Zertifikats.
TR2	Der Zugriff auf geschützte API-Endpunkte muss über JWT-Token abgesichert werden (z., B. <code>/api/books/secure/**</code>).
TR3	CSRF-Schutz muss für REST-APIs deaktiviert werden, da die Authentifizierung stateless über JWT erfolgt.
TR4	Das Backend muss eine RESTful API zur Verfügung stellen, um Daten zwischen Frontend und Backend zu übertragen.
TR5	Die API-Endpunkte müssen CORS-konform konfiguriert sein, sodass ausschließlich Anfragen von der vertrauenswürdigen Frontend-Domain <code>https://localhost:3000</code> zugelassen werden.
TR6	Die Authentifizierung und Autorisierung erfolgt über Okta als Identity Provider unter Verwendung von OAuth2, OpenID Connect und JWT, um sichere und standardisierte Benutzerzugriffe zu gewährleisten.
TR7	Die REST-API muss mit Swagger dokumentiert sein und über Swagger UI erreichbar sein.
TR8	Die Anwendung muss HTTP-Methoden wie POST, PUT, DELETE und PATCH für bestimmte Ressourcen (z. B. Book, Review, Message) deaktivieren, wenn sie nicht benötigt werden.

ID	Anforderung (Beschreibung)
TR9	Die Anwendung muss eine relationale MySQL-Datenbank verwenden (libranova_db), angebunden über JDBC.
TR10	Die Anwendung muss Spring Data JPA zur Datenpersistenz verwenden.
TR11	Die Anwendung muss Entities wie Book, Review und Message als JPA-Entitäten modellieren und mittels Repository-Schnittstellen verfügbar machen.
TR12	Die Anwendung muss Spring Data REST verwenden, um Repository-Schnittstellen automatisch als RESTful API verfügbar zu machen.
TR13	Lombok wird zur Reduzierung von Boilerplate-Code in Entities und anderen Klassen verwendet.
TR14	Die Anwendung muss Spring Boot Web Starter verwenden, um Web-Server-Funktionalitäten und REST-Controller bereitzustellen.
TR15	Die Anwendung muss Unit-Tests mit JUnit und Mocking mit Mockito implementieren, um die Funktionalität des Backends abzusichern.
TR16	Das Frontend muss in React umgesetzt sein und via HTTPS über Port 3000 laufen.
TR17	Das Frontend muss mit dem Backend über REST-APIs kommunizieren.
TR18	Das Frontend muss die vom Backend bereitgestellten Endpunkte nutzen und Authentifizierung über Tokens abwickeln.
TR19	Die Internationalisierung der Anwendung wird im Frontend mit React unter Verwendung der Bibliothek i18next umgesetzt.

Tabelle A.3: Technische Anforderungen von LibraNova

B

Quellcode

```
1      const checkoutButton = useMemo(() => {
2        if (!props.isAuthenticated) {
3          return (
4            <Link to="/login" className="btn btn-success btn-lg w-
5              100">
6              {t("checkout.signIn")}
7            </Link>
8          );
9        }
10
11        if (!props.isCheckedOut && props.currentLoans < 5) {
12          if (!props.book?.copiesInStock || props.book.copiesInStock
13              <= 0) {
14            return (
15              <button
16                className="btn btn-success btn-lg w-100 shadow-sm"
17                disabled
18              >
19                {t("checkout.checkout")}
20              </button>
21            );
22          }
23
24          return (
25            <button
26              onClick={() => props.checkoutBook()}
27              className="btn btn-success btn-lg w-100 shadow-sm"
28              aria-label={t("checkout.checkout")}
29            >
30              {t("checkout.checkout")}
31            </button>
32          );
33
34        if (props.isCheckedOut) {
35          return (
36            <p className="text-success fw-semibold text-center">
37              <i className="bi bi-check-circle-fill me-2"></i>
38              {t("checkout.alreadyCheckedOut")}
39            </p>
40          );
41      }
```

```

42         return (
43             <p className="text-danger fw-semibold text-center">
44                 <i className="bi bi-exclamation-triangle-fill me-2"></i>
45                 {t("checkout.maxReached")}
46             </p>
47         );
48     },
49     [
50         props.isAuthenticated,
51         props.isCheckedOut,
52         props.currentLoans,
53         props.checkoutBook,
54         t,
55     ]);

```

Listing B.1: Prozess des Auscheckens eines Buches im Frontend

```

1  public void returnBook(String userEmail, Long bookId) throws
2      ParseException {
3      Book book = bookRepository.findById(bookId)
4          .orElseThrow(() -> new BookNotAvailableException("Book with ID
5              " + bookId + " is not available."));
6
7      Checkout checkout = checkoutRepository.
8          findByBookIdAndUserEmail(bookId, userEmail);
9      if (checkout == null) {
10          throw new BookNotAvailableException("No active checkout
11              found for book with ID " + bookId + ".");
12      }
13
14      book.setCopiesInStock(book.getCopiesInStock() + 1);
15      bookRepository.save(book);
16
17      LocalDate dueDate = LocalDate.parse(checkout.getReturnDate());
18      LocalDate today = LocalDate.now();
19      long daysOverdue = ChronoUnit.DAYS.between(dueDate, today);
20
21      if (daysOverdue > 0) {
22          Payment payment = paymentRepository.
23              findPaymentsByUserEmail(userEmail);
24          if (payment == null) {
25              payment = Payment.builder()
26                  .userEmail(userEmail)
27                  .amount(0.0)
28                  .build();
29          }
30
31          payment.setAmount(payment.getAmount() + daysOverdue * 2);
32          paymentRepository.save(payment);
33      }
34
35      checkoutRepository.deleteById(checkout.getId());
36
37      History history = History.builder()
38          .userEmail(userEmail)
39          .checkoutDate(checkout.getCheckoutDate())
40          .returnedDate(LocalDate.now().toString())

```

```
36     .title(book.getTitle())
37     .author(book.getAuthor())
38     .overview(book.getOverview())
39     .image(book.getImage())
40     .build();
41
42     historyRepository.save(history);
43 }
```

Listing B.2: Die Geschäftslogik der Rückgabe eines Buches im Backend

C

Glossar

ACID	Atomicity, Consistency, Isolation, Durability (dt.: Atomarität, Konsistenz, Isolation, Dauerhaftigkeit)
API	Application Programming Interface (dt.: Programmierschnittstelle)
CORS	Cross-Origin Resource Sharing (dt.: Ressourcenfreigabe zwischen verschiedenen Ursprüngen)
CRA	Create React App (dt.: React-App-Generator)
CRUD	Create, Read, Update, Delete (dt.: Erstellen, Lesen, Aktualisieren, Löschen)
CSRF	Cross-Site Request Forgery (dt.: Webseiten-übergreifende Anfragenfälschung)
CSS	Cascading Style Sheets (dt.: Gestufte Stylesheets)
DOM	Document Object Model (dt.: Dokumentobjektmodell)
DTO	Data Transfer Object (dt.: Datenübertragungsobjekt)
HAL	Hypertext Application Language (dt.: Hypertext-Anwendungssprache)
HTML	Hypertext Markup Language (dt.: Hypertext-Auszeichnungssprache)
HTTP	Hypertext Transfer Protocol (dt.: Hypertext-Übertragungsprotokoll)
HTTPS	Hypertext Transfer Protocol Secure (dt.: Sicheres Hypertext-Übertragungsprotokoll)
HQL	Hibernate Query Language (dt.: Hibernate-Abfragesprache)
IDE	Integrated Development Environment (dt.: Integrierte Entwicklungsumgebung)
ISBN	International Standard Book Number (dt.: Internationale Standardbuchnummer)
JDBC	Java Database Connectivity (dt.: Java-Datenbankanbindung)

JPA	Java Persistence API (dt.: Java Persistenz-Programmierschnittstelle)
JSON	JavaScript Object Notation (dt.: JavaScript-Objektnotation)
JSX	JavaScript XML (dt.: JavaScript-XML-Syntax)
JWT	JSON Web Token (dt.: JSON-Web-Token)
MFA	Multi-Factor Authentication (dt.: Mehrfaktor-Authentifizierung)
MVC	Model-View-Controller (dt.: Modell-Ansicht-Steuerung)
OAuth2	Open Authorization 2 (dt.: Offene Autorisierung 2)
OPAC	Online Public Access Catalogue (dt.: Online-Bibliothekskatalog)
ORM	Object-Relational Mapping (dt.: Objekt-relationales Abbilden)
PDF	Portable Document Format (dt.: Portables Dokumentenformat)
REST API	Representational State Transfer (dt.: Repräsentative Zustandsübertragung)
SPA	Single-Page Application (dt.: Einzelseitenanwendung)
SQL	Structured Query Language (dt.: Strukturierte Abfrage-sprache)
SSO	Single Sign-On (dt.: Einmalanmeldung)
SSL	Secure Sockets Layer (dt.: Sichere Socket-Schicht)
TLS	Transport Layer Security (dt.: Transportschichtsicherheit)
UI	User Interface (dt.: Benutzeroberfläche)
UML	Unified Modeling Language (dt.: Vereinheitlichte Modellierungssprache)
URI	Uniform Resource Identifier (dt.: Einheitlicher Bezeichner für Ressourcen)
URL	Uniform Resource Locator (dt.: Einheitlicher Ressourcen-anzeiger)
UX	User Experience (dt.: Nutzungserlebnis)
WCAG	Web Content Accessibility Guidelines (dt.: Richtlinien für barrierefreie Webinhalte)
iOS	iPhone Operating System (dt.: iPhone-Betriebssystem)

D

Eigenständigkeitserklärung

- Die vorliegende Arbeit wurde als Einzelarbeit angefertigt.
- Die vorliegende Arbeit wurde als Gruppenarbeit angefertigt. Mein Anteil an der Gruppenarbeit ist im untenstehenden Abschnitt *Verantwortliche* dokumentiert:

- Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche kenntlich gemacht. Darüber hinaus erkläre ich, dass ich die vorliegende Arbeit in dieser oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht habe.

- Es ist keine Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln erfolgt.
- Die Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln wurde von der/dem Prüfenden ausdrücklich gestattet. Die von der/dem Prüfenden mit Ausgabe der Arbeit vorgegebenen Anforderungen zur Dokumentation und Kennzeichnung habe ich erhalten und eingehalten. Sofern gefordert, habe ich in der untenstehenden Tabelle *Nutzung von KI-Tools* die verwendeten KI-basierten text- oder inhaltgenerierenden Hilfsmittel aufgeführt und die Stellen in der Arbeit genannt. Die Richtigkeit übernommener KI-Aussagen und Inhalte habe ich nach bestem Wissen und Gewissen überprüft.

Datum

Unterschrift der Kandidatin/des Kandidaten

Verantwortliche

Der alleinige Autor und Verantwortliche für sämtliche Kapitel der vorliegenden Dokumentation sowie den Quellcode ist:

Mohammed Salih Mezraoui

Nutzung von KI-Tools

KI-Tool	Genutzt für	Warum?	Wann?	Mit welcher Eingabefrage bzw. -aufforderung?	An welcher Stelle der Arbeit übernommen?
ChatGPT	Zusammenfassungen und einleitende Absätze	Unterstützung beim Formulieren von verständlichen Zusammenfassungen und Einleitungen	Während der Erstellung von Kapitel	Ist es eine bessere Formulierung für diesen Abschnitt?“ „Wie kann ich diese beiden Ideen zu einem Abschnitt kombinieren?“ „Welche Reihenfolge der Abschnitte ist sinnvoll?“ „Auf welchen Punkt sollte ich mich hier konzentrieren?“	Kapitel <i>Einleitung, Verwandte Arten, Anwendungsszenarien</i>
ChatGPT	Fehlerbehebung und Installationshinweise	Hilfe beim Lösen von Laufzeit- und Exceptions-Fehlern sowie Installationshinweisen	Während der Implementierung der Anwendung	„Wie behebe ich diesen Fehler?“ „Wie installiere ich Bibliothek X?“	Bei jedem Auftreten eines Fehlers während der Implementierung
DeepL Write	Korrektur Übersetzung Texten	Verbesserung von der Lesbarkeit und Übersetzung von Quellen ins Deutsche	Während der Erstellung von Kapitel	„Formuliere diesen Text in korrektes Deutsch“ „Übersetze den englischen Text sinngemäß ins Deutsche“	Zusammenfassungen, Abstract, Kapitel <i>Grundlagen, Resümee und Ausblick</i>