Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

„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. Georg Schneider

Trier, den 15.08.2025

Kurzfassung

Die Arbeit stellt „LibraNova“ vor – eine moderne und benutzerfreundliche Full-Stack-Webanwendung für das digitale Bibliotheksmanagement und die Ausleihe von Büchern. Entwickelt mit Spring Boot im Backend und React im Frontend, ermöglicht die Anwendung Nutzer:innen das Suchen von Büchern, das Prüfen ihrer Verfügbarkeit sowie das Lesen und Verfassen von Reviews. Administrator:innen können den Buchbestand verwalten und auf Anfragen der Nutzer:innen reagieren. Für Authentifizierung und Autorisierung kommen moderne Sicherheitsstandards wie Okta mit JWT, OAuth2 und OpenID Connect zum Einsatz. Die Zahlungsabwicklung kostenpflichtiger Dienste erfolgt über Stripe. Eine relationale MySQL-Datenbank gewährleistet eine strukturierte und effiziente Datenhaltung. Besonderer Fokus liegt auf intuitiver Benutzerführung, responsivem Design und automatisierten Tests (JUnit, Mockito) zur Qualitätssicherung.

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Einleitung und Anforderungen

In diesem Kapitel werden zentrale Aspekte der Anwendung „LibraNova“ vorgestellt. Der Fokus liegt dabei auf:

- Einem kompakten Überblick über die Ziele und Funktionen der entwickelten Bibliotheksanwendung.
- Sowie der Erhebung und Beschreibung der funktionalen, nicht-funktionalen und technischen Anforderungen, die im Rahmen der Entwicklung berücksichtigt wurden.

1.1 Überblick

„LibraNova“ ist ein modernes Full-Stack-Bibliotheksmanagementsystem, das eine einfache und effiziente Verwaltung von Buchausleihen ermöglicht. Nutzer können Bücher bequem suchen, ausleihen, bewerten und ihre Ausleihen verfolgen – alles über eine benutzerfreundliche und für mobile Geräte optimierte Webanwendung. Die Plattform funktioniert zuverlässig auf verschiedenen Geräten und Bildschirmgrößen.
Neben der übersichtlichen Oberfläche bietet „LibraNova“ wichtige Funktionen wie eine Echtzeit-Anzeige der Buchverfügbarkeit, ein Bewertungssystem sowie eine sichere Nutzeranmeldung mit JWT und OAuth2. Administratoren können Bücher verwalten und auf Nutzeranfragen reagieren. Die Integration externer Dienste wie Stripe ermöglicht sichere Zahlungen für kostenpflichtige Funktionen. Technologisch basiert die Anwendung auf Spring Boot, React und MySQL und ist somit gut skalierbar, sicher und leistungsfähig.

1.2 Anforderungen

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.

1.2.1 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 Tabelle 1.1 zeigt alle funktionalen Anforderungen der Anwendung.

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	Das System verwaltet Nutzerrollen (z. B. Nutzer, Administrator).
FR15	Nutzer können die Leihfrist eines Buches um bis zu 7 Tage verlängern.
FR16	Nutzer können ausgeliehene Bücher zurückgeben.
FR17	Das System zeigt an, wenn ein Buch nicht verfügbar ist.
FR18	Für kostenpflichtige Dienstleistungen ist eine Zahlungsabwicklung über Stripe integriert.
FR19	Nutzer müssen eingeloggt sein, um ein Buch auszuleihen.
FR20	Nutzer müssen eingeloggt sein, um eine Rezension verfassen zu können.
FR21	Nutzer müssen eingeloggt sein, um ein Buch bewerten zu können.
FR22	Das System zeigt an, wie viele Bücher von den maximal fünf gleichzeitig ausgeliehen sind.
FR23	Das System zeigt die Anzahl der Suchergebnisse basierend auf eingegebenen Suchbegriffen und gewählten Kategorien an.
FR24	Das System zeigt Bücherlisten (Suchergebnisse, ausgeliehene Bücher, Ausleihhistorie) paginiert an.
FR25	Das System muss sicherstellen, dass Nutzer sowohl einen Nachrichtentitel als auch den Nachrichtentext angeben, bevor sie eine Anfrage absenden können.
FR26	Das System muss sicherstellen, dass alle erforderlichen Felder beim Anlegen eines neuen Buches ausgefüllt sind.
FR27	Das System muss sicherstellen, dass Administratoren das Antwortfeld ausfüllen, bevor sie eine Antwort absenden können.

FR28	Das System muss im Ausleihverlauf eines Nutzers sowohl das Ausleihdatum als auch das Rückgabedatum jedes Buches anzeigen.

Tabelle 1.1: Funktionale Anforderungen von LibraNova

1.2.2 Nich-funktionale Anforderungen

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

Die folgende Tabelle 1.2 fasst die wichtigsten nicht-funktionalen Anforderungen für LibraNova zusammen.

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	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.
NFR4	Das System muss Cross-Origin Resource Sharing (CORS) korrekt konfigurieren, um Anfragen nur von der vertrauenswürdigen Frontend-Domain `https://localhost:3000` zuzulassen.
NFR5	Das System muss sicherstellen, dass Benutzer nur mit gültigen und nicht abgelaufenen Tokens Zugriff auf geschützte Ressourcen erhalten.
NFR6	Die Reaktionszeit der Such- und Filterfunktionen im Frontend darf 2 Sekunden nicht überschreiten, um ein flüssiges Nutzererlebnis zu gewährleisten.
NFR7	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.
NFR8	Die REST-APIs müssen robust gegen fehlerhafte Eingaben sein und validierte Anfragen verarbeiten, um die Stabilität des Systems zu gewährleisten.
NFR9	Die Backend-Services sollen modular aufgebaut sein, um einfache Wartbarkeit und Erweiterbarkeit zu ermöglichen.
NFR10	Die REST-API-Dokumentation muss aktuell und für Entwickler leicht zugänglich sein (über Swagger UI).
NFR11	Die Anwendung soll eine klare Trennung zwischen Benutzerrollen (Nutzer, Administrator) gewährleisten und Zugriffsrechte strikt durchsetzen.

ID	Anforderung (Beschreibung)
NFR12	Die Anwendung muss auf allen gängigen Webbrowsern wie Firefox, Chrome und Safari ohne Einschränkungen lauffähig sein.
NFR13	NFR31: 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.
NFR14	Die Benutzeroberfläche muss auf allen Seiten konsistente Navigationselemente, Schaltflächen-Designs und Layouts verwenden, um eine einheitliche Nutzererfahrung sicherzustellen.
NFR15	Die Benutzerführung muss intuitiv gestaltet sein, sodass typische Nutzeraktionen wie Buchsuche, Ausleihe oder Bewertung ohne zusätzliche Anleitung verständlich und ausführbar sind.
NFR16	Das System muss sicherstellen, dass nur autorisierte Clients auf geschützte Endpunkte wie `/api/books/secure/**` oder `/api/admin/secure/**` zugreifen können.
NFR17	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.
NFR18	Die Anwendung muss Sicherheitsstandards einhalten, indem sie CSRF-Schutz für REST-APIs deaktiviert, wo dieser nicht erforderlich ist, insbesondere bei stateless JWT-Authentifizierung.

Tabelle 1.2: Nicht-funktionale Anforderungen von LibraNova

1.2.3 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 folgende Tabelle 1.3 zeigt die wichtigsten technischen Anforderungen von LibraNova.

ID	Anforderung (Beschreibung)
TR1	Die Anwendung muss Okta als Identity Provider integrieren und dabei OAuth2, OpenID Connect und JWT verwenden.
TR2	Das Backend muss HTTPS-Verbindungen über Port 8443 ermöglichen, mithilfe eines selbstsignierten SSL-Zertifikats.
TR3	Der Zugriff auf geschützte API-Endpunkte muss über JWT-Token abgesichert werden (z.,B. /api/books/secure/**
TR4	CSRF-Schutz muss für REST-APIs deaktiviert werden, da die Authentifizierung stateless über JWT erfolgt.
TR5	Die Anwendung muss rollenbasierte Zugriffskontrolle implementieren (Nutzer vs. Administrator).

ID	Anforderung (Beschreibung)
TR6	Das Backend muss eine RESTful API zur Verfügung stellen, um Daten zwischen Frontend und Backend zu übertragen.
TR7	Die API-Endpunkte müssen CORS-konform konfiguriert sein und ausschließlich Anfragen von https://localhost:3000 akzeptieren.
TR8	Die REST-API muss mit Swagger dokumentiert sein und über Swagger UI erreichbar sein.
TR9	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.
TR10	Die Anwendung muss eine relationale MySQL-Datenbank verwenden (libranova_db), angebunden über JDBC.
TR11	Die Anwendung muss Spring Data JPA zur Datenpersistierung verwenden.
TR12	Die Anwendung muss Entities wie Book, Review und Message als JPA-Entities modellieren und mittels Repository-Schnittstellen verfügbar machen.
TR13	Die Anwendung muss Spring Data REST verwenden, um Repository-Schnittstellen automatisch als RESTful API verfügbar zu machen.
TR14	Lombok wird zur Reduzierung von Boilerplate-Code in Entities und anderen Klassen verwendet.
TR15	Die Anwendung muss Spring Boot Web Starter verwenden, um Web-Server-Funktionalitäten und REST-Controller bereitzustellen.
TR16	Die Anwendung muss Unit-Tests mit JUnit und Mocking mit Mockito implementieren, um die Funktionalität des Backends abzusichern.
TR17	Das Frontend muss in React umgesetzt sein und via HTTPS über Port 3000 laufen.
TR18	Das Frontend muss mit dem Backend über REST-APIs kommunizieren.
TR19	Die Benutzeranmeldung muss über das Okta Sign-In Widget erfolgen.
TR20	Das Frontend muss die vom Backend bereitgestellten Endpunkte nutzen und Authentifizierung über Tokens abwickeln.

Tabelle 1.3: Technische Anforderungen von LibraNova

2

Methodologie und Systementwurf

In diesem Kapitel wird erklärt, wie das Projekt „LibraNova“ geplant und technisch umgesetzt wurde. Zuerst wird die Projektidee mithilfe von UML-Diagrammen dargestellt. Danach folgen die eingesetzten Technologien im Frontend und Backend sowie der Aufbau der Datenbank. Auch Themen wie Sicherheit, Entwicklungswerkzeuge und Softwaretests werden behandelt, um einen vollständigen Überblick über den Systementwurf zu geben.

2.1 Entwurf und Konzeption

In der Softwareentwicklung spielen Entwurf und Konzeption eine zentrale Rolle. Diese Phase des Entwicklungsprozesses dient dazu, die Struktur und das Verhalten eines Systems frühzeitig zu planen, bevor mit der eigentlichen Implementierung begonnen wird. Ziel ist es, ein klares und verständliches Modell zu erstellen, das die Anforderungen und Abläufe der Anwendung abbildet. In diesem Abschnitt werden exemplarisch einige Diagrammtypen vorgestellt, die zur Modellierung des Projekts verwendet wurden.

2.1.1 Einführung in UML

Die Unified Modeling Language (UML) ist ein wesentliches Werkzeug im Bereich Entwurf. UML 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 [Bel23a].

2.1.2 Klassendiagramm

Das unten stehenden Klassendiagramm 2.1 stellt die grundlegende Struktur der Anwendung dar und dient als Übersicht über die zentralen Klassen sowie deren Beziehungen untereinander. Es bildet sowohl die Domänenlogik (z. B. Bücher, Ausleihen, Rezensionen) als auch das Benutzer- und Kommunikationsmanagement (z. B. Kunden, Administratoren, Nachrichten) ab.
Die Benutzerklassen `Admin` und `Customer` erben von der abstrakten Oberklasse

`User`, wodurch eine klare Trennung der Rollen und Verantwortlichkeiten ermöglicht wird. Die Klasse `Book` steht im Zentrum der Bibliotheksbereich und ist über Assoziationen mit `Checkout`, `Review`, `History` sowie mit den Benutzerklassen verbunden. Diese Assoziationen repräsentieren unter anderem ausgeliehene Bücher, verfasste Rezensionen oder Verwaltungsvorgänge.
Auch die Interaktionen zwischen Kunden und Administratoren über Nachrichten werden modelliert, ebenso wie die Historie vergangener Ausleihvorgänge. Das Diagramm stellt somit eine strukturierte Grundlage für die Implementierung dar und verdeutlicht die Zusammenhänge zwischen den einzelnen Komponenten der Anwendung.

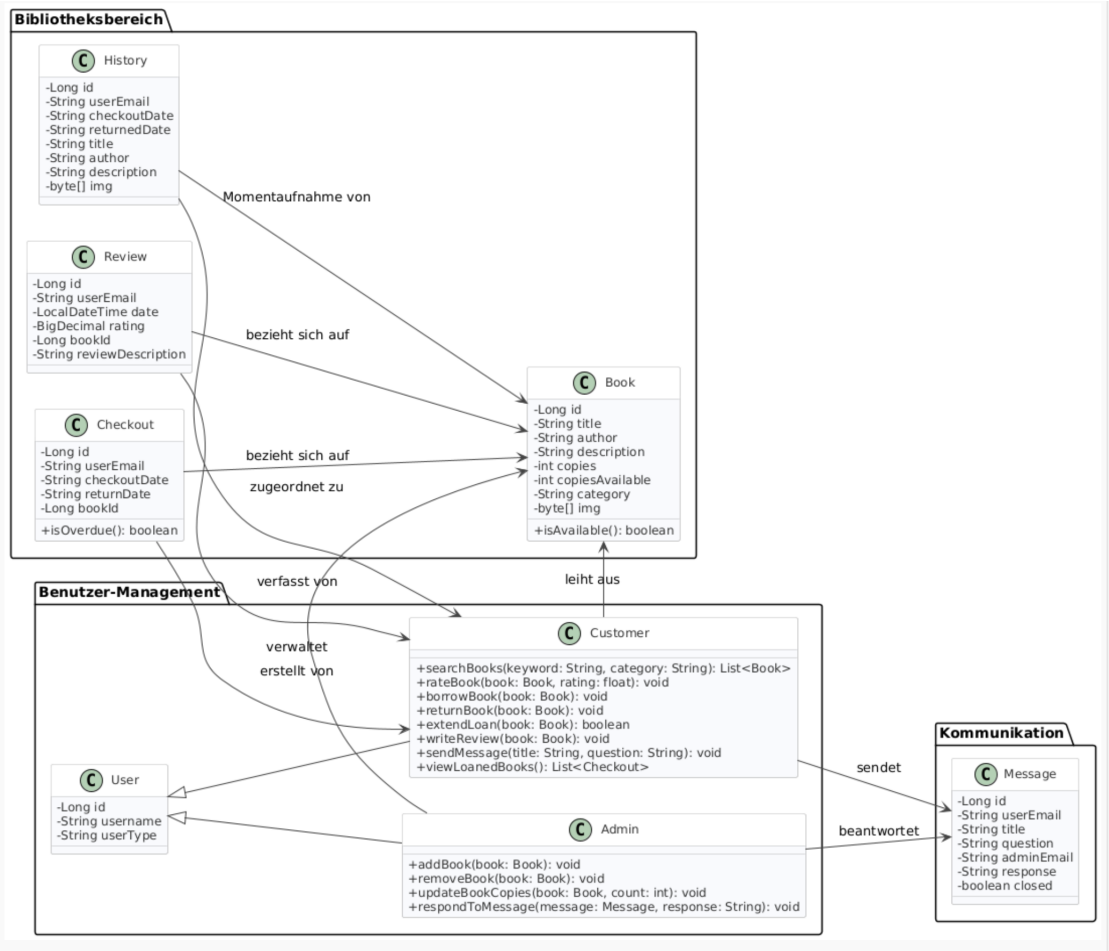

Abbildung 2.1: Klassendiagramm der Anwendung „LibraNova“

2.1.3 Sequenzdiagramm

Ein Sequenzdiagramm ist ein UML-Diagramm, das den zeitlichen Ablauf des Nachrichtenaustauschs zwischen Objekten in einem System darstellt. Es visualisiert, wie verschiedene Komponenten zusammenarbeiten, um bestimmte Funktionen oder Prozesse auszuführen [Cor21]. Sequenzdiagramme sind besonders geeignet, um

Kommunikationsabläufe und die Reihenfolge von Nachrichten in objektorientierten Systemen verständlich zu machen [SMM10, S.28].

Das folgende Sequenzdiagramm 2.2 veranschaulicht den Ablauf des Benutzer-Logins innerhalb der Anwendung. Ziel dieses Prozesses ist die sichere Authentifizierung des Benutzers sowie der Erhalt eines Tokens, um Zugriff auf geschützte Ressourcen zu ermöglichen.

Beteiligt sind die folgenden Komponenten:

- **Ressourceninhaber (Benutzer)**: startet den Anmeldevorgang und erteilt die Zustimmung zur Autorisierung.
- **Client-App (React)**: das Frontend, über das der Benutzer mit dem System interagiert.
- **Autorisierungsserver (Okta)**: authentifiziert den Benutzer und stellt einen Autorisierungscode sowie ein Access Token aus.
- **Ressourcenserver (Spring Boot)**: stellt geschützte Ressourcen bereit und prüft die Gültigkeit des Access Tokens.

Der Ablauf ist wie folgt:

1. Der Ressourceninhaber öffnet die React-Anwendung und startet den Login-Prozess.
2. Die Client-Anwendung leitet den Benutzer an den Autorisierungsserver (Okta) weiter.
3. Der Benutzer meldet sich an und stimmt der Autorisierung zu.
4. Der Autorisierungsserver sendet einen *Autorisierungscode* an die React-Anwendung.
5. Die Client-Anwendung tauscht diesen Code beim Autorisierungsserver gegen ein *Access Token* ein.
6. Das erhaltene Token wird gespeichert (z.,B. im Speicher des Browsers).
7. Bei einem geschützten API-Aufruf sendet die React-Anwendung das Token an den Ressourcenserver (Spring Boot).
8. Der Ressourcenserver prüft die Gültigkeit des Tokens.
9. Ist das Token gültig, werden die geschützten Daten zurückgegeben und dem Benutzer angezeigt.

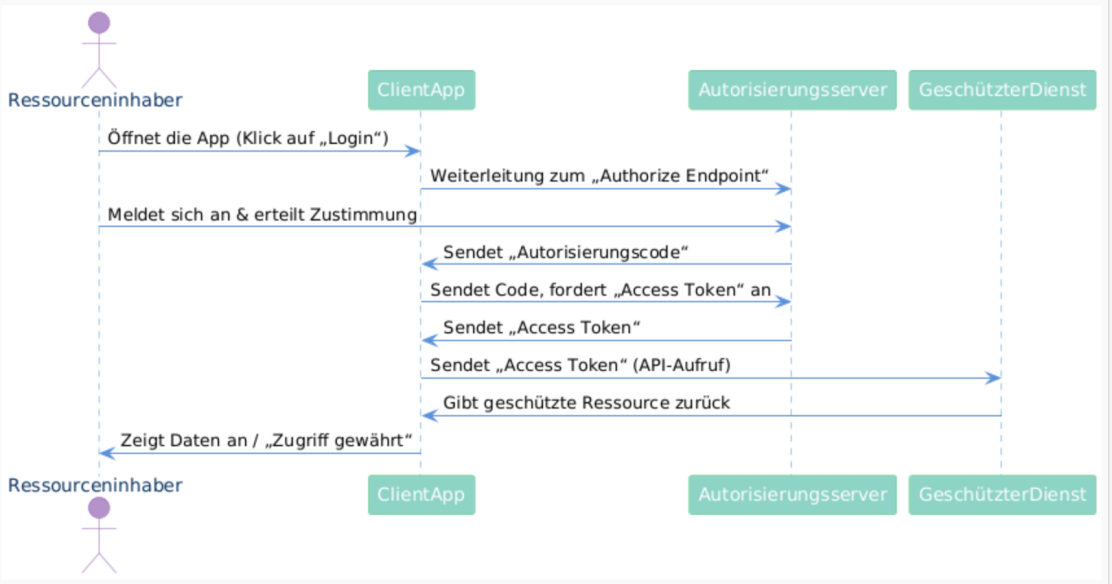

Abbildung 2.2: Sequenzdiagramm des Login-Vorgangs

2.2 Front-End Technologien

Dieser Abschnitt befasst sich mit den verwendeten Frontend-Technologien wie HTML, CSS, TypeScript, React und Bootstrap, die gemeinsam eine moderne, responsive und benutzerfreundliche Oberfläche für unsere Bibliotheksanwendung ermöglichen.

2.2.1 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 [HTM24].

2.2.2 Bootstrap

Bootstrap ist ein weit verbreitetes, quelloffenes Frontend-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 [Boo23].
In der vorliegenden Anwendung 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. Hierfür wurden gezielt Klassen wie `container`, `row`, `col-md-*` und `d-flex` verwendet.

2.2.3 Typescript

TypeScript ist eine von Microsoft entwickelte Programmiersprache, die auf JavaScript basiert und dessen Syntax sowie Laufzeitumgebung erweitert. Im Gegensatz zu JavaScript bietet TypeScript statische Typisierung, was die frühzeitige Erkennung von Fehlern bereits während der Entwicklungsphase ermöglicht [Mic25].
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 wichtige Rolle spielt [Pau25].

In einer Anwendung zur Bibliotheksverwaltung trägt TypeScript insbesondere dazu bei, die Datenstrukturen – wie Bücher oder Ausleihvorgänge – klar zu definieren und potenzielle Typfehler bei API-Aufrufen oder Zustandsänderungen frühzeitig zu verhindern.

2.2.4 React

React ist eine von Meta (ehemals Facebook) entwickelte JavaScript-Bibliothek zur Erstellung moderner Benutzeroberflächen. Sie basiert auf einem komponentenbasierten Architekturmodell, das eine modulare und wiederverwendbare Struktur von UI-Elementen ermöglicht. Dabei wird der Zustand der Komponenten typischerweise mit React-eigenen Mechanismen wie Hooks (useState, useEffect) verwaltet, was eine einfache und effektive Steuerung von Benutzerinteraktionen und UI-Aktualisierungen erlaubt. Besonders bekannt ist React für die Entwicklung von Single-Page Applications (SPA), bei denen Benutzerinteraktionen ohne vollständiges Neuladen der Seite verarbeitet werden, was eine schnelle und flüssige Nutzererfahrung gewährleistet [Met24b, Met24a].

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 [Rak23].
React genießt große Popularität durch seine Performance, seine aktive Community sowie die umfangreiche Unterstützung durch Meta und das wachsende Ökosystem an Tools und Bibliotheken.

2.3 Back-End Technologien

In diesem Abschnitt werden die eingesetzten Back-End-Technologien analysiert, wobei der Fokus auf Spring Boot liegt – dem zentralen Framework zur Imple-

mentierung einer robusten, skalierbaren und sicheren serverseitigen Logik sowie RESTful APIs für die Anwendung LibraNova.

2.3.1 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“ kommt Java in der Version 17 zum Einsatz, um eine robuste, wartbare und skalierbare serverseitige Logik bereitzustellen [Jav24].

2.3.2 Spring Boot

In diesem Abschnitt wird die Verwendung von Spring Boot innerhalb der Bibliotheksanwendung „LibraNova“ näher erläutert. Dabei wird aufgezeigt, welche zentralen Funktionen das Framework übernimmt, welche Module und Abhängigkeiten integriert wurden, wie Spring Boot grundsätzlich funktioniert und welche Vorteile es im Kontext dieser Anwendung bietet. Zudem wird begründet, warum sich das Projektteam bewusst für den Einsatz von Spring Boot entschieden hat.

2.3.2.1 Einführung

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 und Microservices [Spr].

2.3.2.2 Gründe für die Wahl von Spring Boot

Spring Boot wurde für die Umsetzung dieser Anwendung gewählt, da es eine Reihe bedeutender Vorteile bietet, die die Entwicklung von serverseitigen Anwendungen erheblich vereinfachen und beschleunigen. Die wichtigsten Gründe für die Entscheidung zugunsten von Spring Boot sind:

- **Schneller Entwicklungsstart:** Spring Boot verfolgt einen „Opinionated“-Ansatz und stellt vorkonfigurierte Abhängigkeiten bereit, die gängige Standards und Best Practices bereits enthalten. Dadurch kann sofort mit der Entwicklung begonnen werden – ohne langwierige manuelle Konfiguration.
- **Einfache Bereitstellung:** Spring Boot-Anwendungen sind eigenständig lauffähig und benötigen keinen externen Applikationsserver. Der eingebaute Tomcat-Server ermöglicht eine unkomplizierte Ausführung der Anwendung mit einem einfachen Befehl.
- **Automatische Konfiguration:** Viele Frameworks und Bibliotheken werden automatisch erkannt und konfiguriert. Dies reduziert Boilerplate-Code und minimiert Konfigurationsaufwand.

- **Produktionsreife Funktionen:** Funktionen wie externe Konfigurationsdateien, Monitoring, Health Checks und Metriken sind bereits integriert. Dies vereinfacht nicht nur die Entwicklung, sondern auch den späteren produktiven Betrieb [Spr].

2.3.2.3 Funktionsweise von Spring Boot

Spring Boot vereinfacht die Entwicklung von Java-basierten Webanwendungen durch einen automatisierten und konventionsbasierten Startprozess. In diesem Abschnitt wird erläutert, wie ein typisches Spring-Boot-Projekt aufgebaut ist, welche Rolle die Hauptklasse spielt und wie die automatische Konfiguration sowie die Verwaltung von Anwendungseinstellungen funktioniert.

- **Projektinitialisierung (Spring Initializr):** Das Projekt wurde mit dem Tool *Spring Initializr* erzeugt, das eine schnelle Auswahl von Abhängigkeiten und die Generierung eines startbereiten Projekts ermöglicht.
- **Main-Klasse der Anwendung::** Die Annotation `@SpringBootApplication` ist eine Meta-Annotation, die `@Configuration`, `@EnableAutoConfiguration` und `@ComponentScan` kombiniert. Dadurch kann Spring Boot automatisch die benötigten Konfigurationen basierend auf den eingebundenen Abhängigkeiten vornehmen und nach Komponenten, Services und Repositories suchen, um diese als Beans im Anwendungskontext zu registrieren.

```
1  @SpringBootApplication
2  public class SpringBootLibraryApplication {

4      public static void main(String[] args) {
5          SpringApplication.run(SpringBootLibraryApplication.class,
               args);
6      }

8  }
```

Listing 2.1: Einstiegspunkt der Spring Boot Anwendung

- **Konfigurationsdateien (`application.properties`):** Zentrale Einstellungen wie der Serverport, die Datenbankverbindung oder benutzerdefinierte Werte werden in `application.properties` definiert. Diese Datei bietet eine zentrale Steuerung der Anwendungskonfiguration.

```
1  spring.application.name=libranova
2  # Datenbankkonfiguration
3  spring.datasource.url=jdbc:mysql://localhost:3306/libranova_db
4  spring.datasource.username=your-username
5  spring.datasource.password=your-password
6  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
       MySQLDialect
7  # REST Pfad
8  spring.data.rest.base-path=/api
9  # Logging
10 logging.level.org.springframework.security=DEBUG
```

Listing 2.2: Beispielhafte application.properties Datei

- **Eingebaute Webserver (Tomcat):** Spring Boot integriert standardmäßig einen eingebetteten Webserver wie Tomcat. Dadurch kann die Anwendung ohne externes Deployment direkt ausgeführt werden.

2.3.2.4 Eingesetzte Abhängigkeiten (Dependencies)

Eine Aufzählung und kurze Erklärung der wichtigsten genutzten Abhängigkeiten:

- **Spring Web**: Für die Erstellung von RESTful APIs.
- **Spring Data JPA**: Für die Datenzugriffslogik mit JPA/Hibernate.
- **Spring Data REST**: Automatische Bereitstellung von REST-Endpunkten für JPA-Repositories.
- **Spring Boot Starter**: Vereinfacht die Projektkonfiguration durch sinnvolle Voreinstellungen.
- **Lombok (extern)**: Reduziert Boilerplate-Code durch automatische Generierung von Gettern, Settern usw.

2.3.2.5 Spring Security

Spring Security ist ein umfassendes und hochgradig anpassbares Framework, das speziell für die Verwaltung der Authentifizierung und Autorisierung in Java-Anwendungen entwickelt wurde. Seine Architektur ermöglicht es Entwicklern, Sicherheitsmaßnahmen zu implementieren, die auf die einzigartigen Anforderungen ihrer Anwendungen zugeschnitten sind, sodass es eine bevorzugte Wahl für Enterprise-Level-Lösungen. Das Framework bietet nicht nur wiederverwendbare Module für die Authentifizierung und Autorisierung, sondern umfasst auch Standardschutzmaßnahmen gegen gängige Web-Schwachstellen wie Cross-Site Request Forgery (CSRF) und verschiedene Exploit-Schutzmaßnahmen [IRM+20].

Die Stärke von Spring Security liegt in seiner Flexibilität, die eine nahtlose Integration in ein breites Spektrum von Anwendungsfällen ermöglicht. Entwickler können die Funktionen leicht erweitern, um spezifische Sicherheitsanforderungen zu erfüllen, was in den dynamischen Anwendungsumgebungen von heute entscheidend ist. Dieses Anpassungspotenzial birgt jedoch auch Risiken, da unsachgemäße Konfigurationen zu erheblichen Sicherheitsschwachstellen führen können [IRM+20].

In der Anwendung wird Spring Security verwendet, um die REST-API-Endpunkte abzusichern und den Zugriff zu kontrollieren.

2.3.3 REST-API

REST ist ein Architekturstil für die Entwicklung vernetzter Anwendungen. Er basiert auf einem zustandslosen, Client-Server- und cachefähigen Kommunikationsprotokoll, und in fast allen Fällen wird das HTTP-Protokoll verwendet. REST-APIs sind so konzipiert, dass sie einfach, leichtgewichtig und skalierbar sind, was sie zu einer beliebten Wahl für Webdienste macht [RES24].

2.3.3.1 Einführung in REST

Angelehnt an [RES24] bieten REST-APIs eine Möglichkeit, über HTTP auf die Funktionalität und Daten einer Anwendung zuzugreifen. Sie folgen den Prinzipien von REST, die auf Ressourcen und deren Repräsentationen basieren. Jede Ressource wird durch eine eindeutige URI identifiziert und kann laut [KMM21, S.4] durch standardisierte HTTP-Methoden manipuliert werden:

- GET: Abrufen von Ressourcen
- POST: Erstellen neuer Ressourcen
- PUT: Aktualisieren bestehender Ressourcen
- DELETE: Löschen von Ressourcen

REST-APIs sind aufgrund ihrer Architektur und Funktionsweise weit verbreitet und bieten eine Reihe von Vorteilen:

- **Skalierbarkeit:** Durch das stateless Design können REST-APIs leicht skaliert werden. Jeder HTTP-Request enthält alle notwendigen Informationen, um ihn zu verarbeiten, ohne dass der Server den vorherigen Zustand kennen muss.
- **Flexibilität:** REST-APIs sind flexibel und können mit verschiedenen Datenformaten arbeiten. JSON ist das gebräuchlichste Format, da es leichtgewichtig und gut lesbar ist.
- **Interoperabilität:**[1] REST-APIs nutzen standardisierte HTTP-Methoden, was ihre Interoperabilität mit verschiedenen Clients und Plattformen gewährleistet.
- **Leichte Integration:** REST-APIs lassen sich leicht in bestehende Systeme integrieren, da sie auf bekannten Webstandards basieren.

2.3.3.2 Absicherung der REST-Endpunkte mit Spring Security

Zur Absicherung der REST-Endpunkte in „LibraNova“ wurde das Framework **Spring Security** in Kombination mit OAuth2 und JSON Web Tokens (JWT) eingesetzt. Dabei schützt eine zentrale Sicherheitskonfiguration gezielt sensible Pfade wie `/api/books/secure/**`, Alle übrigen Endpunkte bleiben öffentlich zugänglich. Diese Trennung zwischen geschützten und öffentlichen Routen erlaubt eine kontrollierte Zugriffskontrolle und gewährleistet gleichzeitig Offenheit für nicht sensible Daten.
Die Konfiguration erfolgt über eine `SecurityFilterChain`-Bean. Hierbei wurde die CSRF-Absicherung deaktiviert, da sie bei stateless JWT-Authentifizierung überflüssig ist. Gleichzeitig wird eine OAuth2-Resource-Server-Konfiguration mit JWT-Validierung verwendet. Die Einrichtung einer `ContentNegotiationStrategy` unterstützt eine saubere Inhaltsaushandlung zwischen Client und Server. Die Integration der `Okta.configureResourceServer401ResponseBody(http)`-Methode verbessert zudem die Fehlerbehandlung bei unautorisierten Zugriffen.

[1] Interoperabilität ist die Fähigkeit verschiedener Systeme oder Software, zusammenzuarbeiten und Informationen nahtlos auszutauschen[wik].

```
1       http
2       .csrf(csrf -> csrf.disable())
3       .authorizeHttpRequests(configurer -> configurer
4       .requestMatchers("/api/books/secure/**",
5       "/api/reviews/secure/**",
6       "/api/messages/secure/**",
7       "/api/admin/secure/**").authenticated()
8       .anyRequest().permitAll())
9       .oauth2ResourceServer(oauth2 -> oauth2.jwt())
10      .cors(cors -> cors.configurationSource(corsConfigurationSource()));
```

Listing 2.3: Spring Security-Konfiguration

Durch diese Konfiguration wird sichergestellt, dass nur authentifizierte Nutzer mit gültigem JWT-Token auf geschützte Ressourcen zugreifen können, während gleichzeitig der Zugriff auf öffentliche Inhalte uneingeschränkt möglich bleibt.

2.3.3.3 CORS-Konfiguration und Einschränkung von HTTP-Methoden

Zur zusätzlichen Absicherung des Backends wurden zwei Maßnahmen umgesetzt: Erstens wurde eine **CORS-Konfiguration** implementiert, die ausschließlich Anfragen vom React-Frontend (`https://localhost:3000`) erlaubt. Zweitens wurden mithilfe von `RepositoryRestConfigurer` bestimmte HTTP-Methoden wie `POST`, `PUT`, `PATCH` und `DELETE` auf ausgewählten Entitäten deaktiviert, um ungewollte Änderungen über Spring Data REST zu verhindern.

```
1       config.exposeIdsFor(Book.class, Review.class, Message.class);

3       HttpMethod[] unsupported = {POST, PUT, PATCH, DELETE};
4       restrictHttpMethods(Book.class, config, unsupported);

6       corsRegistry.addMapping(config.getBasePath() + "/**")
7       .allowedOrigins("https://localhost:3000");
```

Listing 2.4: CORS und HTTP-Methodenbeschränkung

Diese Konfiguration trägt maßgeblich zur Sicherheit und Stabilität der Anwendung bei, indem sie sowohl die erlaubten Ursprünge als auch die zulässigen Zugriffsarten explizit definiert.

2.3.4 HTTPS und SSL/TLS

In diesem Abschnitt wird erläutert, wie die Anwendung durch die Integration von HTTPS und SSL/TLS eine sichere und verschlüsselte Kommunikation zwischen Client und Server gewährleistet.

2.3.4.1 Grundlagen von HTTPS und SSL/TLS

HTTPS ist die verschlüsselte Version von HTTP und schützt die Kommunikation zwischen Client und Server [HTT24]. Dabei kommen die Protokolle SSL/TLS zum Einsatz, wobei TLS als moderner und sicherer Standard gilt [Bel23b]. Sie gewährleisten:

- **Verschlüsselung:** Schutz der übertragenen Daten vor unbefugtem Zugriff.
- **Authentifizierung:** Sicherstellung, dass der Server legitim ist.
- **Datenintegrität:** Verhinderung der Manipulation von Daten während der Übertragung.

2.3.4.2 SSL-Konfiguration im Backend

Zur Absicherung des Datenverkehrs wurde in der Anwendung eine HTTPS-Konfiguration implementiert, bei der der Server auf Port `8443` HTTPS-Anfragen entgegennimmt (siehe 2.6). Dafür wurde SSL/TLS aktiviert, um eine verschlüsselte Kommunikation zwischen Client und Server zu gewährleisten.
Die Verschlüsselung basiert auf einem selbstsignierten SSL-Zertifikat, das mit dem folgenden Befehl generiert wurde:

```
keytool -genkeypair -alias libranova \
-keystore src/main/resources/libranova-keystore.p12 \
-keypass secret -storepass secret -storeType PKCS12 \
-keyalg RSA -keysize 2048 -validity 365 \
-dname "C=DE, ST=Rhineland-Palatinate, L=Trier, O=libranova, OU=
    Studies Backend, CN=localhost" \
-ext "SAN=dns:localhost"
```

Listing 2.5: Generierung eines SSL-Zertifikats

Dabei wurde eine Keystore-Datei im `PKCS12`-Format (`libranova-keystore.p12`) erstellt, in der das Zertifikat unter dem Alias `libranova` gespeichert ist. Diese Datei wird in der `application.properties` wie folgt eingebunden:

```
# HTTPS settings
server.port=8443
server.ssl.enabled=true
server.ssl.key-alias=libranova
server.ssl.key-store=classpath:libranova-keystore.p12
server.ssl.key-store-password=secret
server.ssl.key-store-type=PKCS12
```

Listing 2.6: HTTPS- und SSL-Konfiguration

Durch diese Konfiguration wird sichergestellt, dass alle übermittelten Daten verschlüsselt und vor unbefugtem Zugriff geschützt sind. Die Anwendung erfüllt damit grundlegende Sicherheitsanforderungen moderner Webanwendungen.

2.3.4.3 SSL-Konfiguration im Frontend

Auch im Frontend wurde eine lokale HTTPS-Verbindung eingerichtet, um eine verschlüsselte Kommunikation mit dem Backend zu ermöglichen. Dazu wurde mithilfe von OpenSSL ein selbstsigniertes Zertifikat erstellt:

```
openssl req -x509 \
-out ssl-localhost-libranova/localhost.crt \
-keyout ssl-localhost-libranova/localhost.key \
-newkey rsa:2048 -nodes -sha256 -days 365 \
-config localhost.conf
```

Listing 2.7: Generierung eines Frontend-Zertifikats

Dieses Kommando erzeugte zwei Dateien:

- `localhost.crt` – das Zertifikat
- `localhost.key` – der zugehörige private Schlüssel

In der `.env`-Datei wurden diese Dateien anschließend referenziert, um die React-Entwicklungsumgebung über HTTPS zu betreiben:

```
SSL_CRT_FILE=ssl-localhost-libranova/localhost.crt
SSL_KEY_FILE=ssl-localhost-libranova/localhost.key
REACT_APP_API_URL='https://localhost:8443/api'
```

Listing 2.8: .env Konfiguration für HTTPS und API-Zugriff

Die Konfigurationsdatei `localhost.conf` enthielt die benötigten Informationen für das Zertifikat (wie Standort und Common Name), um den Generierungsprozess zu automatisieren:

```
[req]
prompt = no
distinguished_name = dn

[dn]
C = DE
ST = Rhineland-Palatinate
L = Trier
O = Libranova
OU = Studies
CN = localhost
```

Listing 2.9: localhost.conf

Diese Konfiguration ermöglicht eine sichere Verbindung zwischen Frontend und Backend während der lokalen Entwicklung.

Die HTTPS-Implementierung mit SSL/TLS gewährleistet die Einhaltung von Sicherheitsstandards und schützt sensible Nutzerdaten zuverlässig.

2.3.5 Stripe API

2.3.5.1 Überblick

2.3.5.2 Hauptmerkmale der Stripe API

2.3.5.3 Stripe API Integration für nahtlose Zahlungsabwicklung

2.4 Datenbankstruktur

In diesem Abschnitt wird die Datenbankstruktur der Webanwendung untersucht, um ein umfassendes Verständnis für die Organisation, den Zugriff und die Verwaltung der Daten zu vermitteln. Der Schwerpunkt liegt auf dem relationalen

Datenbankmodell und den spezifischen Technologien und Frameworks, die zur Handhabung der Datenpersistenz und der Transaktionen verwendet werden. Dabei wird MySQL als das gewählte Datenbanksystem zusammen mit Spring Data JPA für eine nahtlose Dateninteraktion untersucht. In den folgenden Unterabschnitten werden diese Aspekte im Detail behandelt, wobei relationale Datenbanken, die Auswahl von MySQL, Datenpersistenz und Transaktionsmanagement sowie die Integration von MySQL mit Spring Boot behandelt werden.

2.4.1 Übersicht über relationale Datenbanken und Auswahl von MySQL

Relationale Datenbanken speichern Daten in Tabellen und nutzen Schlüssel zur Definition von Beziehungen. Für ein Bibliotheksverwaltungssystem eignet sich dieses Modell gut, da es komplexe Verknüpfungen zwischen Entitäten wie Büchern und Ausleihvorgängen unterstützt. MySQL wurde gewählt, da es strukturierte Daten zuverlässig verarbeitet, komplexe Abfragen effizient unterstützt und sich stabil mit Spring Boot integrieren lässt [IBM].

2.4.2 Entitäten und Datenbankabbildung mit Spring Data JPA

Zur Modellierung der Datenstruktur der Bibliotheksanwendung kommen mehrere Entitäten zum Einsatz, die jeweils einer Tabelle in der zugrundeliegenden relationalen Datenbank entsprechen. Die Abbildung erfolgt über JPA-Annotationen wie `@Entity`, `@Id` und `@Column`. Dadurch wird eine klare, objektorientierte Repräsentation der Daten erreicht.

Zu den zentralen Entitäten gehören:

- **Book**: Enthält grundlegende Informationen über verfügbare Bücher, wie Titel, Autor, Kategorie, verfügbare Exemplare und Beschreibung.
- **Checkout**: Repräsentiert die Ausleihvorgänge, wobei jedem Eintrag eine Benutzer-E-Mail, ein Buch und Ausleih- sowie Rückgabedatum zugeordnet sind.
- **History**: Dokumentiert abgeschlossene Ausleihvorgänge und speichert relevante Buch- und Benutzerdaten.
- **Review**: Ermöglicht es Nutzern, Bewertungen zu Büchern zu hinterlassen. Enthält Angaben wie Bewertung, Rezensionstext und das Erstellungsdatum.
- **Message**: Dient der Kommunikation zwischen Nutzern und Administratoren, insbesondere für Supportanfragen.

Die Verwaltung der Entitäten erfolgt über Spring Data JPA. Interfaces wie `JpaRepository` ermöglichen CRUD-Operationen und komplexe Abfragen ohne eigenen SQL-Code. Das `BookRepository` (siehe 2.10) bietet Methoden mittels benutzerdefinierter Queries.

```
1       public interface BookRepository extends JpaRepository<Book, Long> {
2           Page<Book> findByTitleContainingIgnoreCase(@RequestParam("title")
                String title, Pageable pageable);

4           Page<Book> findByCategory(@RequestParam("category") String
                category, Pageable pageable);

6           @Query("SELECT␣b␣FROM␣Book␣b␣WHERE␣b.id␣IN␣:book_ids")
7           List<Book> findBooksByBookIds(@Param("book_ids") List<Long> bookId
                );
8       }
```

Listing 2.10: Ausschnitt aus dem BookRepository

Diese strukturierte und erweiterbare Modellierung bildet das Fundament für die Datenpersistenz und stellt sicher, dass alle Anwendungskomponenten konsistent mit der Datenbank interagieren.

2.4.3 MySQL-Workbench und SQL

MySQL Workbench dient als Werkzeug für den Entwurf und die Verwaltung der Datenbank, während SQL als Sprache zur Erstellung von Abfragen und zur Datenbankverwaltung eingesetzt wird.

Mit MySQL Workbench wurde das Datenbankschema visualisiert, was eine intuitive Gestaltung und Pflege der Datenbankstrukturen ermöglicht. So konnten Tabellen, Beziehungen und Indizes einfach erstellt und angepasst werden. Zusätzlich kam SQL zum Einsatz, um Abfragen für Tests durchzuführen sowie Aufgaben wie Datenmanipulation und Schemaänderungen vorzunehmen [myS].

2.5 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.

2.5.1 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 [OTa].

In dieser Anwendung dient Okta als externer Identitätsanbieter zur sicheren und effizienten Verwaltung von Authentifizierung und Autorisierung. Das Frontend in React nutzt das Okta Sign-In Widget und SDK, um Benutzeranmeldungen, Sitzungsmanagement und Token-Verwaltung abzuwickeln. Im Backend validiert die

Spring Boot-Anwendung diese Tokens mithilfe des Okta Spring Boot Starters, um sicherzustellen, dass nur authentifizierte und autorisierte Nutzer auf geschützte API-Endpunkte zugreifen können.

Der Hauptzweck der Okta-Integration bestand darin, die Komplexität der sicheren Nutzerverwaltung auszulagern – einschließlich Passwortverwaltung, Token-Ausgabe und rollenbasierter Zugriffskontrolle. So konnte ein standardisierter und effizienter Authentifizierungsprozess umgesetzt werden, der unterschiedliche Benutzerrollen (z. B. Admins und normale Nutzer) unterstützt und die Berechtigungen direkt in Okta verwaltet. Insgesamt verbessert die Einbindung von Okta die Sicherheit, Skalierbarkeit und Wartbarkeit des Authentifizierungsflusses, ohne das Rad neu erfinden zu müssen.

2.5.2 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. [JT].

In der Anwendung werden von Okta ausgestellte JWTs verwendet, um API-Endpunkte zu sichern. Spring Security ist als OAuth2-Resource-Server konfiguriert, der die JWTs validiert. Die Tokens enthalten Benutzer-Claims wie den benutzerdefinierten „userType“, der in den Controllern ausgelesen wird, um rollenbasierte Zugriffssteuerung umzusetzen. So wird sichergestellt, dass z. B. nur Admins bestimmte Aktionen durchführen können. Dieses Setup gewährleistet eine sichere, tokenbasierte Authentifizierung und feingranulare Autorisierung im Backend.

2.5.3 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 [Car23].

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.

2.5.4 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 [OTb].

In der Anwendung wird OpenID Connect über Okta genutzt, um eine zuverlässige und interoperable Benutzeranmeldung sicherzustellen. Dabei stellt OpenID Connect die Identitätsinformationen als ID-Token bereit, die vom Backend zur Authentifizierung und Autorisierung der Benutzer überprüft werden. So kann die Anwendung Benutzerrollen sicher handhaben und den Zugriff auf geschützte Ressourcen kontrollieren.

2.6 Entwicklungsumgebung und Versionskontrolle

In diesem Abschnitt werden die integrierten Entwicklungsumgebungen (IDEs) und Versionskontrollsysteme beschrieben, die im Rahmen des Projekts eingesetzt wurden. Er bietet einen Überblick über die für die Entwicklung ausgewählten IDEs und die Versionskontrollwerkzeuge, die zur effektiven Verwaltung von Codeänderungen eingesetzt wurden.

2.6.1 Entwicklungsumgebung

Für die Entwicklung des Backends wurde IntelliJ IDEA Community Edition verwendet, eine kostenlose und weit verbreitete IDE, die besonders gut für Spring Boot Projekte geeignet ist. Für das Frontend kam Visual Studio Code zum Einsatz, ebenfalls eine kostenlose und sehr beliebte Entwicklungsumgebung, die sich durch ihre Flexibilität und umfangreiche Erweiterungen auszeichnet. Beide Tools gehören zu den meistgenutzten in der Softwareentwicklung und haben die effiziente Umsetzung der Anwendung unterstützt.

2.6.2 Versionskontrollsysteme: Git und GitHub

In diesem Unterabschnitt werden die im Projekt eingesetzten Versionskontrollsysteme Git und GitHub beschrieben. Git ist ein kostenloses, verteiltes Versionskontrollsystem, das durch schnelle Performance, effizientes Verzweigungsmanagement und flexible Arbeitsabläufe überzeugt. Es ermöglicht eine einfache Verwaltung von Projekten jeder Größe und fördert die parallele Entwicklung durch lokale Branches [Teaa].

GitHub ergänzt Git als cloudbasierte Plattform für das Speichern, Teilen und die Zusammenarbeit an Code. Es erleichtert das Verfolgen von Änderungen, die Überprüfung durch andere Entwickler sowie die koordinierte Zusammenarbeit, ohne unbeabsichtigte Auswirkungen auf den Hauptcode zu riskieren [Teab]. Beide Tools haben wesentlich zur effizienten Codeverwaltung und Versionskontrolle im Verlauf des Projekts beigetragen.

2.7 Testen und Qualitätssicherung

2.7.1 Backend-Tests

- Unit-Tests mit JUnit
- Mocking und Dependency Injection mit Mockito
- Integrationstests mit Spring Test

2.7.2 Frontend-Tests

- Komponenten- und UI-Tests mit React Testing Library und Jest

2.7.3 API-Tests

- Manuelle und automatisierte API-Tests mit Postman

3

Implementierung

In diesem Kapitel wird die Implementierung und die prinzipielle Funktionsweise der Anwendung beschrieben.

3.1 Architektur und Projektstruktur

3.1.1 Backend-Struktur

3.1.2 Frontend-Struktur

3.2 Backend-Implementierung

3.2.1 Spring Boot Hauptklasse und Konfiguration

3.2.2 REST API Endpunkte

3.2.3 Datenzugriff

3.3 Frontend-Implementierung

3.3.1 Hauptkomponenten und Routing

3.3.2 Authentifizierung und Autorisierung im Frontend

3.3.3 State Management

3.4 Teststrategien und Qualitätssicherung

3.4.1 Backend-Tests

3.4.2 Frontend-Tests

4

Ergebnisse und Analyse

4.1 Allgemeine Benutzeroberfläche

4.1.1 Header und Navigation

4.1.2 Footer

4.1.3 Startseite (Main Page)

4.1.3.1 Karussell (Carousel)

4.1.3.2 Links und Informationen

4.2 Seitenübersicht

4.2.1 Buchsuche (Search Page)

4.2.2 Buchdetails (Book Page)

4.2.3 Rezensionsseite (Reviews Page)

4.3 Benutzerbezogene Funktionen

4.3.1 Bibliotheksaktivität (Library Activity)

4.3.1.1 Ausleihen (Loans)

4.3.1.2 Ausleihhistorie (History)

4.3.2 Bibliotheksservice (Library Service)

4.4 Admin-Bereich

4.4.1 Buchverwaltung

4.4.2 Benutzeranfragenverwaltung

5

Zusammenfassung und Ausblick

Diese Dokumentation hat die Konzeption, Implementierung und Analyse der Full-Stack-Bibliotheksmanagement-Anwendung „LibraNova“ umfassend dargestellt. Ziel des Projekts war es, eine moderne, benutzerfreundliche und sichere Plattform zu entwickeln, die den Anforderungen der Bibliotheksnutzer und Administratoren gerecht wird. Mit bewährten Technologien wie Spring Boot im Backend, React mit TypeScript im Frontend sowie einer Sicherheitsarchitektur mit Okta, JWT, OAuth2 und OpenID Connect konnte eine skalierbare und wartbare Lösung realisiert werden.

Die klare Trennung von Backend und Frontend sowie moderne Frameworks ermöglichten eine effiziente Entwicklung. Spring Security in Verbindung mit Okta sorgt für einen robusten Authentifizierungs- und Autorisierungsmechanismus, der die Sicherheit sensibler Daten gewährleistet. Die Integration von Redux im Frontend unterstützt ein effektives State-Management für eine reaktive und performante Benutzeroberfläche.

Die Anwendung umfasst Kernfunktionen zur Buchsuche, Ausleihe und Bewertung sowie administrative Werkzeuge zur Verwaltung des Buchkatalogs und der Nutzeranfragen. Die Qualitätssicherung erfolgte mit JUnit, Mockito und React Testing Library, um Backend- und Frontend-Komponenten systematisch zu testen und die Stabilität der Anwendung sicherzustellen. Postman diente als praktisches API-Testtool und ergänzte den Entwicklungsprozess.

Zukünftige Arbeiten könnten Cloud-Bereitstellung mit CI/CD-Pipelines, erweiterte Testabdeckung sowie Funktionen wie Benachrichtigungen oder personalisierte Empfehlungen umfassen. Die modulare Architektur erlaubt eine flexible Anpassung von LibraNova an unterschiedliche Bibliotheksanforderungen.

Insgesamt bietet LibraNova eine zeitgemäße Lösung, die den digitalen Wandel im Bibliotheksmanagement unterstützt und die Nutzererfahrung durch intuitive Bedienung und sichere Prozesse verbessert. Diese Dokumentation bildet eine Grundlage für weitere Entwicklungen und zeigt, wie moderne Webtechnologien effektiv kombiniert werden können, um praxisnahe und nachhaltige Anwendungen zu schaffen.

Literaturverzeichnis

Bel23a. Bell, Donald: *An introduction to the Unified Modeling Language*. https://developer.ibm.com/articles/an-introduction-to-uml/, 2023. Abgerufen am 20.07.2025.

Bel23b. Bell, Donald: *What is SSL/TLS: An In-Depth Guide*. https://www.ssl.com/article/what-is-ssl-tls-an-in-depth-guide/, 2023. Abgerufen am 19.08.2024.

Boo23. Bootstrap Authors: *Bootstrap - The most popular HTML, CSS, and JS library in the world*. https://getbootstrap.com/docs/5.3/getting-started/introduction/, 2023. Zugegriffen am 22. Juli 2025.

Car23. Carlier, Bertrand: *An Introduction to OAuth2.0*. IDPro Body of Knowledge, 1, 10 2023.

Cor21. Corporation, IBM: *Sequenzdiagramme*. https://www.ibm.com/docs/de/radfws/9.6.1?topic=diagrams-sequence, 2021. abgerufen am 28.08.2024.

HTM24. *HTML For Beginners*. https://html.com, 2024. Abgerufen am 02.06.2024.

HTT24. *An overview of HTTP*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview, 2024. Abgerufen am 19.08.2024.

IBM. IBM: *What is a relational database?* https://www.ibm.com/topics/relational-databases/. Abgerufen am 14.08.2024.

IRM+20. Islam, Mazharul, Sazzadur Rahaman, Na Meng, Behnaz Hassanshahi, Padmanabhan Krishnan, Danfeng und Yao: *Coding Practices and Recommendations of Spring Security for Enterprise Applications*. 07 2020.

Jav24. *What is Java technology and why do I need it?* https://www.java.com/en/download/help/whatis_java.html, 2024. Abgerufen am 19.08.2024.

JT. JWT-Team: *Introduction to JSON Web Tokens*. https://jwt.io/introduction. Abgerufen am 19. August 2024.

KMM21. Kornienko, D V, S V Mishina und M O Melnikov: *The Single Page Application architecture when developing secure Web services*. Journal of Physics: Conference Series, 2091(1), 2021.

Met24a. 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, 2024. Zugegriffen am 22. Juli 2025.

Met24b. Meta Platforms, Inc.: *Describing the UI.* https://react.dev/learn/describing-the-ui, 2024. Zugegriffen am 22. Juli 2025.

Mic25. Microsoft Authors: *TypeScript for JavaScript Programmers.* https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html, 2025. Zugegriffen am 22. Juli 2025.

myS. mySQL: *MySQL Workbench Manual.* https://dev.mysql.com/doc/workbench/en/wb-intro.html. Abgerufen am 14.08.2024.

OTa. Okta-Team: *What is Okta and what does Okta do?* https://support.okta.com/help/s/article/what-is-okta?language=en_US. Abgerufen am 19. August 2024.

OTb. OpenId-Team: *How OpenId Connect Works.* https://openid.net/developers/how-connect-works/. Abgerufen am 19. August 2024.

Pau25. Paul Bratslavsky: *Top 6 Benefits of Implementing TypeScript.* https://strapi.io/blog/benefits-of-typescript, 2025. Zugegriffen am 22. Juli 2025.

Rak23. Rakesh Purohit: *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, 2023. Zugegriffen am 23. Juli 2025.

RES24. *What is REST?* https://www.codecademy.com/article/what-is-rest, 2024. Abgerufen am 18.08.2024.

SMM10. Swain, Santosh Kumar, Durga Prasad Mohapatra und Rajib Mall: *Test case generation based on use case and sequence diagram.* International Journal of Software Engineering, 3(2):21–52, 2010.

Spr. Spring: *Spring Framework.* https://spring.io/projects/spring-boot. Abgerufen am 28. August 2024.

Teaa. Team, Git: *Git About.* https://git-scm.com/about. Abgerufen am 14. August 2024.

Teab. Team, GitHub: *GitHub About.* https://docs.github.com/en/get-started/start-your-journey/about-github-and-git#about-github. Abgerufen am 14. August 2024.

wik. *Wikipedia: Interoperability.* https://en.wikipedia.org/wiki/Interoperability. Abgerufen am 27.07.2024.

A

Abkürzungsverzeichnis

SQL	Structured Query Language
API	Application Programming Interface
JPA	Java Persistence API
MVC	Model-View-Controller
OAuth2	Open Authorization 2
JWT	JSON Web Token
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
ACID	Atomicity, Consistency, Isolation, Durability
CRUD	Create, Read, Update, Delete
ORM	Object-Relational Mapping
JDBC-URL	Java Database Connectivity Uniform Resource Locator
IDE	Integrated Development Environment
SCM	Source Code Management
CVS	Concurrent Versions System
HTTPS	Hypertext Transfer Protocol Secure
UML	Unified Modeling Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SPA	Single-Page Application
UI	User Interface
RxJs	Reactive Extensions for JavaScript
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
REST API	Representational State Transfer
DTO	Data Transfer Object

B

Selbstständigkeitserklärung

☐ Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Imad-Eddine ABDESSAMI, Mohammed Salih MEZRAOUI

Datum	Unterschrift der Kandidatin/des Kandidaten
Datum	Unterschrift der Kandidatin/des Kandidaten