

Entwicklung einer webbasierten Anwendung unter Verwendung des Spring Frameworks mit Test-Driven Development (TDD)

Studienarbeit T3_3101

des Studienganges Angewandte Informatik an der
Dualen Hochschule Baden-Württemberg Mosbach



von

Isabel Lind

Matrikelnummer, Kurs 8471449, INF21B

Gutachter der Dualen Hochschule Philipp Abele

23. Juni 2024

Eigenständigkeitserklärung

*Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema:
„Entwicklung einer webbasierten Anwendung unter Verwendung des Spring Frameworks
mit Test-Driven Development (TDD)“
selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel be-
nutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fas-
sung übereinstimmt.*

Ort, Datum

Unterschrift

Kurzzusammenfassung

Diese Studienarbeit behandelt die Entwicklung einer webbasierten Todo-Liste mit Vue.js, Spring Boot und MySQL mit Test-Driven Development. Ziel ist es, eine effiziente Aufgabenverwaltung zu ermöglichen, inklusive Erstellung, Bearbeitung und Löschung von Todos.

Die Arbeit beschreibt detailliert die Architektur, Implementierung und Teststrategie der Anwendung. Trotz grundlegender Funktionen wie Benutzerregistrierung und -anmeldung fehlen sichere Abmeldung und Todo-Bearbeitung, was Privatsphäre und Benutzerfreundlichkeit beeinträchtigt.

Ursachen sind mangelnde Erfahrung des Entwicklers, schlechtes Zeitmanagement und unvollständige TDD-Anwendung, was zu fehlerhaftem Code führte. Die Arbeit bietet Einblicke und Empfehlungen für zukünftige Verbesserungen und eine stärkere Anwendung bewährter Methoden.

Abstract

This student research project deals with the development of a web-based todo list using Vue.js, Spring Boot and MySQL with Test-Driven Development. The aim is to enable efficient task management, including the creation, editing and deletion of todos.

The work describes in detail the architecture, implementation and test strategy of the application. Despite basic functions such as user registration and login, secure logout and todo editing are missing, which impairs privacy and user-friendliness.

Causes include lack of developer experience, poor time management and incomplete TDD application, resulting in buggy code. The work provides insights and recommendations for future improvements and greater use of best practices.

Inhaltsverzeichnis

Abbildungsverzeichnis	I
Abkürzungsverzeichnis	II
1 Einleitung	1
2 Grundlagen	2
2.1 Test-Driven Development	2
2.2 Ablauf von Test-Driven Development	3
3 Konzeption und Design der Todo-App	6
3.1 Architektur der Anwendung	6
3.2 Datenmodell und Datenbankdesign	7
3.2.1 Entwurf des Datenmodells	7
3.2.2 Projektanforderungen an die Datenbank	9
3.2.3 Datenbankdesign	10
3.3 User Interface Design	10
3.3.1 Gestaltung der Benutzeroberfläche	10
3.3.2 Berücksichtigung von Usability-Prinzipien	13
4 Backend-Implementierung unter Verwendung von TDD	16
4.1 Entwicklungsumgebung und Werkzeuge	16
4.2 Implementierungsschritte	17
4.3 Beispieltests und Testfälle	19
4.4 Unit-Tests	19
4.5 Integrationstests	20
4.6 Akzeptanztests	21

Inhaltsverzeichnis

5	Frontend Entwicklung	23
5.1	Projektanforderung an Frontend-Entwicklung	23
5.2	Technologiestack	24
5.3	Projektstruktur	24
5.4	Routing	25
5.5	Authentifizierung und Autorisierung	25
5.6	HTTP-Anfragen	25
5.7	State Management	26
5.8	Styling	26
6	Deployment	27
6.1	Datenbank einrichten	27
6.2	Backend starten	28
6.3	Frontend starten	28
6.4	Zugriff auf Todo-Liste Webanwendung	28
7	Fazit	30
	Literaturverzeichnis	33

Abbildungsverzeichnis

2.1	schematischer Ablauf des Test-Driven Development	2
2.2	schematischer Ablauf des traditionellen Testen	3
2.3	Phasen des Test-Driven Development	4
2.4	schematischer Ablauf von Test-Driven Development in der Praxis	5
3.1	User Interface Design der Startseite	11
3.2	User Interface Design der Anmeldeseite	11
3.3	User Interface Design der Registrierungsseite	12
3.4	User Interface Design der Hauptseite	12
3.5	User Interface Design der Ansicht zum Erstellen einer neuen Todo	13
3.6	User Interface Design der Detailansicht zum Bearbeiten	14

Abkürzungsverzeichnis

TDD Test-Driven Development

1 Einleitung

Die Notwendigkeit effektiver Aufgabenverwaltungslösungen ist in unserer zunehmend digitalisierten Welt unbestreitbar. Eine Todo-Liste ist ein bewährtes Werkzeug, das Einzelpersonen und Teams hilft, ihre täglichen Aufgaben zu organisieren, zu priorisieren und zu verfolgen. In einer Zeit, in der Effizienz und Produktivität entscheidend sind, bietet eine gut gestaltete Todo-Liste klare Vorteile bei der Strukturierung des Arbeitsalltags.

Ziel dieser Studienarbeit ist es, eine webbasierte Todo-Liste mit Test-Driven Development zu entwickeln, die Benutzern eine intuitive und effiziente Möglichkeit bietet, ihre Aufgaben zu verwalten. Die Anwendung soll es Einzelpersonen ermöglichen, Todos zu erstellen, zu bearbeiten und zu löschen und die Möglichkeit bieten eine Beschreibung, ein Fälligkeitsdatum, sowie das Markieren des Erledigen einer Todo hinzuzufügen. Darüber hinaus soll die Todo-Liste Funktionen wie Benutzerregistrierung und -authentifizierung bereitstellen, um die Sicherheit und Privatsphäre der Benutzerdaten zu gewährleisten.

Im Rahmen dieser Entwicklung werden zwei Hauptkomponenten erstellt: das Frontend, das die Benutzeroberfläche der Todo-Liste bereitstellt, und das Backend, das die Datenverarbeitung und -speicherung übernimmt. Die Anwendung wird auf modernen Technologien basieren, darunter Vue.js für das Frontend, Spring Boot für das Backend, MySQL als relationale Datenbank.

Es werden grundlegende Kenntnisse in HTML, CSS, JavaScript, sowie in der Entwicklung mit Vue.js, Spring Boot, MySQL vorausgesetzt.

Diese Dokumentation bietet einen umfassenden Einblick in die Architektur, Implementierung und Teststrategie der Todo-Liste Webanwendung.

2 Grundlagen

2.1 Test-Driven Development

Bei dem Test-Driven Development (TDD), zu Deutsch auch „test-getriebene Entwicklung“, handelt es sich um ein Entwicklungs- und Designverfahren für Software, bei dem Testfälle bereits vor oder spätestens parallel zur Implementierung spezifiziert werden. Die zu erstellende Software wird quasi über Tests entworfen. [1, S. 151]

TDD sollte bereits bei der Anforderungsanalyse angewendet werden. Die Anforderungen sind so zu definieren, dass sichergestellt werden kann, dass die Erfüllung dieser Anforderungen mithilfe von Tests validiert werden können. Dies kann durch Testdefinitionen in Textform gewährleistet werden, in die genau die Vorbedingungen, die Ausführung und die zu erwartenden Ergebnisse beschrieben werden. [2, S.188]

Die Abbildung 2.1 zeigt exemplarisch einen schematischen Ablauf des Test-Driven Development-Ansatzes. Am Anfang wird der Testfall definiert (Test Definition, TD), darauf folgt die Umsetzung und Programmierung (Programming, P). Im Anschluss werden die Testfälle ausgeführt (Test Execution, TE). [1, S. 151]

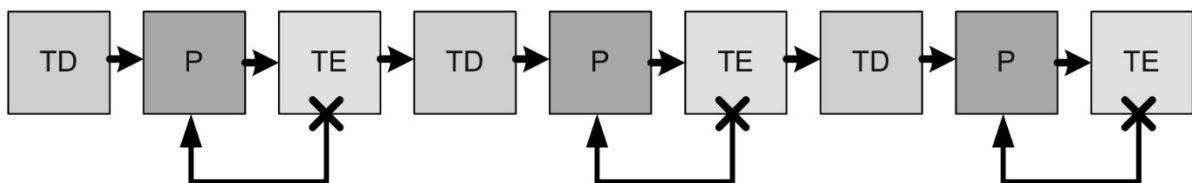


Abbildung 2.1: schematischer Ablauf des Test-Driven Development [1, S. 151]

Im Gegensatz dazu wird beim traditionellen Testen erst parallel zur Entwicklung oder nach Abschluss der Implementierung geeignete Testfälle definiert und ausgeführt, wie ein schematischer Ablauf in Abbildung 2.2 zeigt. [1, S. 150]

2.2. ABLAUF VON TEST-DRIVEN DEVELOPMENT

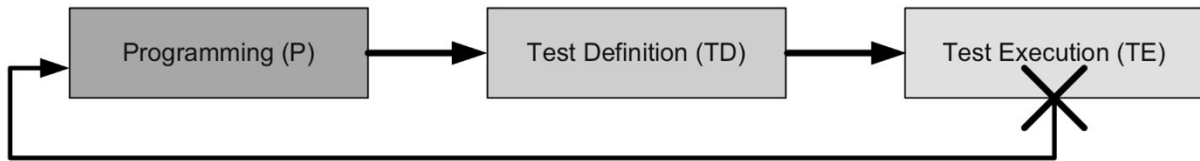


Abbildung 2.2: schematischer Ablauf des traditionellen Testen [1, S.151]

Durch das frühe Definieren und Ausführen von Testfällen im TDD-Ansatz erhalten die Entwickler ein unmittelbares Feedback zur erstellten Lösung und Probleme wie Seiteneffekte können frühzeitig erkannt werden [1, S. 151]. Da der Entwickler sich für die Testerstellung intensiv mit den Anforderungen auseinander setzen muss, erhält er ein verbessertes Verständnis für das zu entwickelnde Produkt und die eigentliche Entwicklungszeit kann deutlich verkürzt werden. Die Anzahl verschleppter und dann aufwändig zu reparierender Fehler kann zudem, durch die von der Entwicklung unabhängigen Erstellung der Testfälle, reduziert werden. Denn, wenn erst nach der Implementierung die Testfälle definiert werden, besteht eine große Wahrscheinlichkeit, dass Denkfehler bei der Implementierung bei der Testerstellung wiederholt werden und somit falsche Tests, die falsche zu testende Software, als korrekt überprüfen [2, S.188].

Die frühzeitig erstellten Tests eignen sich ferner auch zur Automatisierung und des Weiteren kann durch die Testfälle und deren Testergebnisse die Kommunikation verbessert werden. TDD wird meistens im Rahmen der agilen Software-Entwicklung verwendet und ist fester Bestandteil verschiedener Vorgehensmodelle, wie eXtreme Programming und SCRUM [1, S. 151].

2.2 Ablauf von Test-Driven Development

Der entscheidende Bestandteil des Test-Driven Development ist, dass die Testfälle vor oder spätestens parallel zur Implementierung definiert werden. Im Anschluss werden die Softwarekomponenten implementiert bzw. angepasst, bis die definierten Tests erfolgreich durchlaufen. Konkret lässt sich TDD in vier grundlegende Schritte unterteilen, wie sie in Abbildung 2.3 skizziert werden. [1, S. 153]

In dem ersten Schritt „think“ wird die Anforderung ausgewählt, die im nächsten Schritt umgesetzt werden soll. Anhand der ausgewählten Anforderung werden die geeigneten

2.2. ABLAUF VON TEST-DRIVEN DEVELOPMENT

Tests definiert. Hierbei ist sicherzustellen, dass die ausgewählte Anforderung auch tatsächlich durch die Tests abgedeckt wird. Unit-Tests können beispielsweise auf der Implementierungsebene z.B. für Komponenten verwendet werden. [1, S. 153]

In dem zweiten Schritt „red“ werden diese Tests ausgeführt. Da es noch keine Implementierung dazu gibt, schlagen die Tests fehl und sind im Status „red“. [1, S. 153]

Dann erfolgt die Implementierung, bis die erstellten Tests erfolgreich durchlaufen und den Status „green“ erreichen (Schritt drei). Dabei wird die Anforderung schrittweise implementiert und getestet. Wenn die Testfälle nicht erfolgreich durchlaufen, werden Fehler korrigiert, falls die Anforderung bereits umgesetzt wurde oder die Funktionalität implementiert, falls die Anforderung noch nicht umgesetzt wurde. [1, S. 153]

Im letzten Schritt erfolgt die Optimierung und Anpassung des geschriebenen Softwarecodes (vierter Schritt Refactoring). Da durch das Refactoring der Code verändert wird, müssen alle Testfälle im Anschluss jeweils durchgeführt werden, um sicherzustellen, dass alle Tests weiterhin erfolgreich durchlaufen und der Status „green“ nicht mehr verlassen wird. [1, S. 153]

Wenn das Refactoring erfolgreich durchgeführt wurde, ist die Implementierung für die ausgewählte Anforderung abgeschlossen und die nächste Anforderung kann durch die gleiche Schrittfolge umgesetzt werden. [1, S. 153 f.]

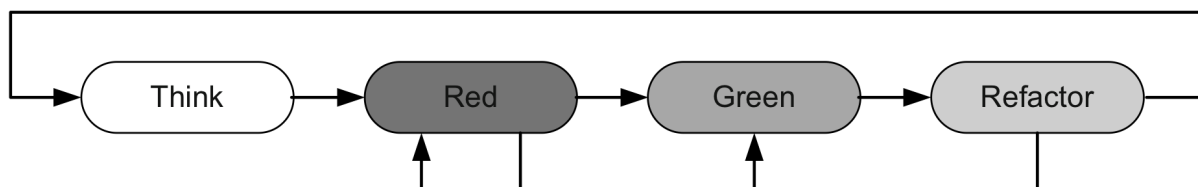


Abbildung 2.3: Phasen Ablauf des Test-Driven Development [1, S. 153]

Die Abbildung 2.4 zeigt einen beispielhaften schematischen Ablaufes von TDD in der Praxis. Dabei wird veranschaulicht, dass für die ausgewählten Anforderungen geeignete Testfälle erstellt werden, um die Erfüllung der Anforderung validieren zu können. In dem Anwendungsbeispiel wird für die Anforderung A drei Testfälle benötigt. Die Testläufe, welche auf der x-Achse dargestellt werden, geben den Status der Testfälle an und wechseln von „red“ in „green“, entsprechend den jeweiligen TDD-Schritten. Dabei wird ersichtlich, dass die Anforderungen schrittweise implementiert und getestet werden. Dieses Beispiel

2.2. ABLAUF VON TEST-DRIVEN DEVELOPMENT

zeigt ebenfalls auf, wie durch die schnelle Rückmeldung, durch die Tests, Seiteneffekte frühzeitig entdeckt werden können. Eine Implementierung, welche für den Testfall C2 bestimmt war, bewirkte, dass nicht nur der Test C2 weiterhin fehlschlägt, sondern auch der Testfall B2. Dieser Testfall wäre wohlmöglich bei einem traditionellen Testanfall erst sehr spät entdeckt worden und müsste aufwendig korrigiert werden. [1, S. 154]

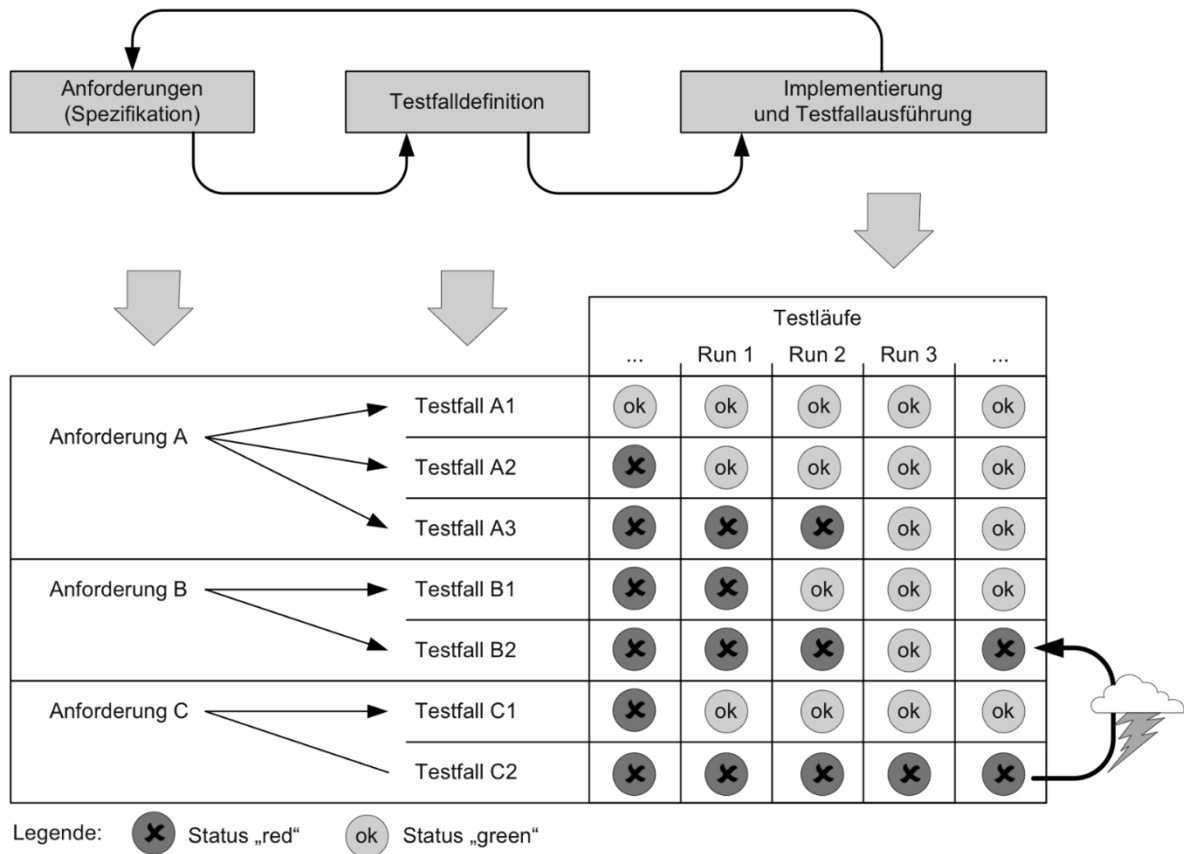


Abbildung 2.4: schematischer Ablauf von Test-Driven Development in der Praxis [1, S. 154]

3 Konzeption und Design der Todo-App

3.1 Architektur der Anwendung

Die Todo-App ist als eine mehrschichtige Architektur konzipiert, die eine klare Trennung der Verantwortlichkeiten zwischen den verschiedenen Schichten sicherstellt. Diese Architektur umfasst die folgenden Schichten:

- **Präsentationsschicht:** Diese Schicht besteht aus REST-Controllern, die HTTP-Anfragen entgegennehmen und HTTP-Antworten zurückgeben. Sie interagiert mit der Service-Schicht, um Geschäftslogik zu implementieren.
- **Service-Schicht:** Diese Schicht enthält die Geschäftslogik der Anwendung. Sie validiert die Daten und ruft die entsprechenden Methoden der Repository-Schicht auf.
- **Repository-Schicht:** Diese Schicht besteht aus JPA-Repositories, die für die Datenzugriffslogik verantwortlich sind. Sie interagiert mit der MySQL-Datenbank, um Daten zu speichern und abzurufen.
- **Sicherheitsschicht:** Diese Schicht nutzt Spring Security, um Authentifizierung und Autorisierung zu implementieren. JWT (JSON Web Tokens) wird verwendet, um die Benutzersitzungen zu verwalten.
- **Datenbank-Schicht:** Diese Schicht besteht aus einer MySQL-Datenbank, in der alle Daten der Anwendung gespeichert werden.

3.2 Datenmodell und Datenbankdesign

3.2.1 Entwurf des Datenmodells

Das Datenmodell der Todo-App umfasst mehrere Entitäten, um die Beziehungen zwischen Benutzern und ihren Aufgaben zu verwalten. Die Hauptentitäten sind User und Todo.

User Entität

Die User-Entität repräsentiert einen Benutzer der Anwendung und enthält folgende Attribute:

- id: Ein eindeutiger Bezeichner für jeden Benutzer.
- username: Der Benutzername, den der Benutzer zur Anmeldung verwendet.
- password: Das verschlüsselte Passwort des Benutzers.

Die User-Entität wird als Java-Klasse implementiert und mit JPA-Anmerkungen versehen, um die Zuordnung zur Datenbank zu erleichtern.

```
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    // Getter und Setter
}
```

Todo Entität

3.2. DATENMODELL UND DATENBANKDESIGN

Die Todo-Entität repräsentiert eine Aufgabe und enthält folgende Attribute:

- id: Ein eindeutiger Bezeichner für jede Todo.
- title: Der Titel der Todo.
- description: Eine Beschreibung der Todo.
- dueDate: Das Fälligkeitsdatum der Todo.
- completed: Ein boolescher Wert, der angibt, ob die Todo abgeschlossen ist.
- user: Eine Beziehung zum Benutzer, der die Todo erstellt hat.

Die Todo-Entität wird als Java-Klasse implementiert und mit JPA-Anmerkungen versehen, um die Zuordnung zur Datenbank zu erleichtern.

```
@Entity
public class Todo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column
    private String description;

    @Column
    private LocalDate dueDate;

    @Column(nullable = false)
    private boolean completed;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
```

```
// Getter und Setter  
}
```

3.2.2 Projektanforderungen an die Datenbank

Das Projekt stellt spezifische Anforderungen an die Datenbank, die durch MySQL erfüllt werden:

- **Open source:** Die Datenbank muss open source sein, da keine Kosten für die Verwendung der Datenbank anfallen dürfen. MySQL bietet eine MySQL Community Edition an, die open source ist [3].
- **Datenintegrität und Konsistenz:** Die Datenbank muss in der Lage sein, Daten in einer strukturierten und konsistenten Weise zu speichern. MySQL gewährleistet dies durch die Verwendung relationaler Tabellen und die Unterstützung von Transaktionen [3].
- **Sicherheit:** Der Schutz sensibler Daten ist von entscheidender Bedeutung. MySQL bietet umfassende Sicherheitsfunktionen wie SSL-Verschlüsselung, Datenmaskierung und Authentifizierungs-Plugins, um die Datensicherheit zu gewährleisten [3].
- **Performance:** Das Projekt erfordert schnelle Datenverarbeitungszeiten, um eine reibungslose Benutzererfahrung zu gewährleisten. MySQL bietet eine hohe Geschwindigkeit und Effizienz bei der Verarbeitung großer Datenbanken [4].
- **Flexibilität und Integration:** Die Datenbank muss flexibel genug sein, um mit verschiedenen Programmiersprachen und Technologien integriert zu werden. MySQL bietet eine breite Unterstützung für verschiedene Systeme und Schnittstellen [4].
- **Benutzerfreundlichkeit:** Eine einfache Installation und Verwaltung der Datenbank ist erforderlich, um den Entwicklungsprozess zu optimieren. MySQL ist benutzerfreundlich und bietet zahlreiche Verwaltungstools, die die Bedienung erleichtern [4].

Diese Anforderungen des Projekts werden durch die funktionalen und technischen Merkmale von MySQL umfassend abgedeckt, was die Entscheidung für MySQL als Datenbanklösung rechtfertigt.

3.3. USER INTERFACE DESIGN

3.2.3 Datenbankdesign

Das Datenbankdesign umfasst zwei Tabellen: users und todos. Die Struktur der Tabellen ist wie folgt:

Tabelle users

- id (BIGINT, AUTO_INCREMENT, PRIMARY KEY)
- username (VARCHAR, NOT NULL, UNIQUE)
- password (VARCHAR, NOT NULL)

Tabelle todos

- id (BIGINT, AUTO_INCREMENT, PRIMARY KEY)
- title (VARCHAR, NOT NULL)
- description (TEXT)
- dueDate (DATE)
- completed (BOOLEAN, NOT NULL)
- user_id (BIGINT, FOREIGN KEY)

Durch diese Struktur wird sichergestellt, dass jede Aufgabe eindeutig einem Benutzer zugeordnet ist. Diese Beziehung ermöglicht es, dass Benutzer nur ihre eigenen Todos sehen und verwalten können.

3.3 User Interface Design

3.3.1 Gestaltung der Benutzeroberfläche

Die Benutzeroberfläche der Todo-App ist so gestaltet, dass sie intuitiv und benutzerfreundlich ist. Die Anwendung besteht aus mehreren Ansichten, die jeweils spezifische Funktionen bereitstellen:

- **/IndexView**: Die Startseite der Anwendung siehe Abbildung 3.1. Diese Seite bietet zwei Hauptoptionen: „Log in“ und „Sign up“. Dies ermöglicht neuen Benutzern die Registrierung und bestehenden Benutzern die Anmeldung.

3.3. USER INTERFACE DESIGN

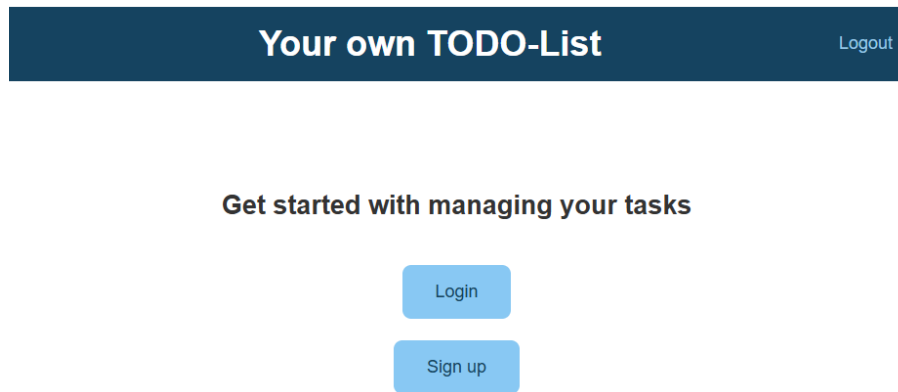


Abbildung 3.1: User Interface Design der Startseite [Eigene Darstellung]

- **/LoginView**: Die Anmeldeseite siehe Abbildung 3.2. Hier können Benutzer ihren Benutzernamen und ihr Passwort eingeben, um auf ihre persönlichen Aufgabenlisten zuzugreifen. Ein Link zur Registrierung ist ebenfalls vorhanden.

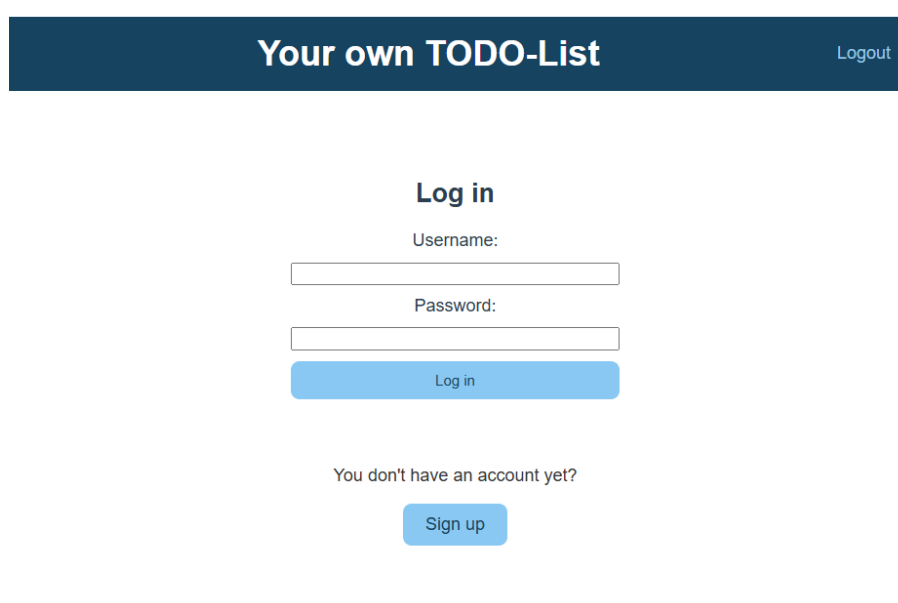
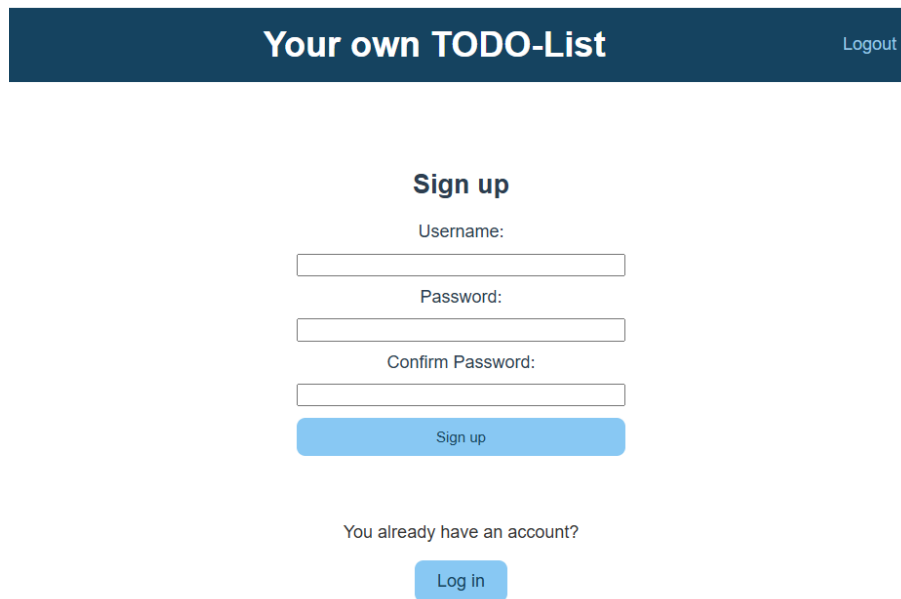


Abbildung 3.2: User Interface Design der Anmeldeseite [Eigene Darstellung]

- **/RegisterView**: Die Registrierungsseite siehe Abbildung 3.3. Neue Benutzer können hier ein Konto erstellen, indem sie ihren Benutzernamen, ihr Passwort und die Bestätigung des Passworts eingeben. Ein Link zur Anmeldung für bestehende Benutzer ist ebenfalls verfügbar.

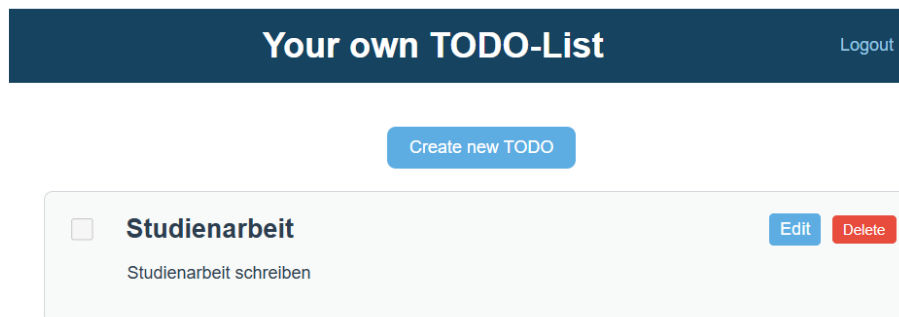
3.3. USER INTERFACE DESIGN



The registration page features a dark blue header with the title "Your own TODO-List" and a "Logout" link. Below the header, the "Sign up" section includes three input fields for "Username:", "Password:", and "Confirm Password:", followed by a blue "Sign up" button. A link "You already have an account?" points to a blue "Log in" button.

Abbildung 3.3: User Interface Design der Registrierungsseite [Eigene Darstellung]

- **/HomeView**: Die Hauptseite nach der Anmeldung siehe Abbildung 3.4. Diese Seite zeigt die Aufgabenliste des angemeldeten Benutzers. Benutzer können neue Aufgaben erstellen, vorhandene Aufgaben bearbeiten oder löschen.



The home page features a dark blue header with the title "Your own TODO-List" and a "Logout" link. Below the header, there is a blue "Create new TODO" button. A task card for "Studienarbeit" is shown, including a checkbox, the task name, a description "Studienarbeit schreiben", and "Edit" and "Delete" buttons.

Abbildung 3.4: User Interface Design der Hauptseite [Eigene Darstellung]

- **/NewTodoForm**: Diese Seite bietet ein Formular zum Erstellen einer neuen Todo siehe Abbildung 3.5.
- **/TodoDetailView**: Die Detailansicht einer spezifischen Todo siehe Abbildung 3.6. Diese Seite ermöglicht es Benutzern, die Details einer ausgewählten Todo zu bear-

3.3. USER INTERFACE DESIGN

The image shows a user interface for a TODO application. At the top is a dark blue header bar with the text "Your own TODO-List" in white and a "Logout" link on the right. Below the header is a section titled "Create New Todo". The form includes a "Completed:" checkbox, a "Title:" text input field, a "Description:" text input field with a small edit icon, and a "Due Date:" date input field with a calendar icon. At the bottom of the form are two buttons: "Create new TODO" (blue) and "Cancel" (dark grey).

Abbildung 3.5: User Interface Design der Ansicht zum Erstellen einer neuen Todo [Eigene Darstellung]

beiten.

Jede Seite hat ein konsistentes Layout mit einem zentralen Header. Der Header ist als eine Komponente in der **/HeaderBar** definiert. Er enthält das Logo „Your own TODO-List“ und die Logout-Option, abhängig davon, ob der Benutzer angemeldet ist.

3.3.2 Berücksichtigung von Usability-Prinzipien

Das User Interface Design der Todo-App folgt mehreren grundlegenden Usability-Prinzipien nach Jakob Nielsen [5], um sicherzustellen, dass die Anwendung einfach zu bedienen und effizient ist:

1. **Sichtbarkeit des Systemstatus:** Die App hält den Benutzer stets über den aktuellen Status und die Ergebnisse ihrer Aktionen informiert. Beispielsweise werden Änderungen an Aufgaben sofort angezeigt und erfolgreiche Anmeldungen führen direkt zur Hauptseite mit der Aufgabenliste.
2. **Übereinstimmung zwischen System und realer Welt:** Die Anwendung ver-

3.3. USER INTERFACE DESIGN

The screenshot displays the 'Your own TODO-List' application interface. At the top, a dark blue header bar contains the title 'Your own TODO-List' in white and a 'Logout' link on the right. Below the header, the 'Edit Todo' form is centered. It includes a 'Completed:' checkbox, a 'Title:' text field with the value 'Studienarbeit', a 'Description:' text field with the value 'Studienarbeit abgeben', and a 'Due Date:' text field with the value '23.06.2024'. At the bottom of the form are two buttons: 'Save and Go Back' (blue) and 'Cancel' (dark grey).

Abbildung 3.6: User Interface Design der Detailansicht [Eigene Darstellung]

wendet Begriffe und Konzepte, die den Benutzern vertraut sind. Schaltflächen und Symbole sind intuitiv und leicht verständlich, was die Bedienung erleichtert.

3. **Benutzerkontrolle und Freiheit:** Benutzer können Aktionen rückgängig machen. Es gibt deutlich sichtbare Optionen zum Abbrechen von Aktionen, um Fehlaktionen leicht korrigieren zu können.
4. **Konsistenz und Standards:** Das Layout und das Design der Benutzeroberfläche sind auf allen Seiten der Anwendung konsistent. Dies erleichtert den Benutzern das Verständnis und die Navigation durch die App.
5. **Wiedererkennung statt Erinnerung:** Die Benutzeroberfläche macht alle wichtigen Optionen und Funktionen sichtbar, damit Benutzer sie leicht wiedererkennen können, anstatt sich an sie erinnern zu müssen. Wichtige Elemente wie Schaltflächen und Eingabefelder sind prominent platziert und leicht zu finden. Der Einsatz von Farben und Schriftgrößen hilft, die Aufmerksamkeit der Benutzer auf wichtige Aktionen und Informationen zu lenken.
6. **Ästhetik und minimalistisches Design:** Das Design ist einfach und übersichtlich gehalten, um die Benutzerfreundlichkeit zu erhöhen. Unnötige Elemente wurden vermieden, um Ablenkungen zu minimieren und den Fokus auf die Hauptfunktionen

3.3. USER INTERFACE DESIGN

der Anwendung zu legen.

7. **Hilfe beim Erkennen, Diagnostizieren und Beheben von Fehlern:** Fehler-
nachrichten sind in einfacher Sprache verfasst und bieten klare Hinweise zur Pro-
blemlösung. Beispielsweise wird bei einer fehlerhaften Anmeldung eine verständliche
Fehlermeldung angezeigt und mögliche Lösungen vorgeschlagen.

4 Backend-Implementierung unter Verwendung von TDD

4.1 Entwicklungsumgebung und Werkzeuge

Die Entwicklung der Todo-App erfolgte in einer modernen Java-Entwicklungsumgebung, die auf dem Spring Boot Framework basiert. Die wichtigsten verwendeten Tools und Technologien sind:

- **IDE:** IntelliJ IDEA wurde als Integrated Development Environment (IDE) verwendet, da es umfangreiche Unterstützung für Java und das Spring Framework bietet, einschließlich leistungsstarker Debugging- und Refactoring-Tools [6].
- **Build-Tool:** Maven wurde für das Build-Management und die Abhängigkeitsverwaltung eingesetzt. Es ermöglicht die einfache Verwaltung von Bibliotheken und Plugins sowie die Konfiguration von Build-Prozessen [7].
- **Versionierung:** GitHub wurde für die Quellcodeverwaltung und Versionskontrolle verwendet. GitHub ermöglichte zudem die Zusammenarbeit und den Austausch von Code [8].
- **Test-Frameworks:** JUnit 5 und Spring Boot Test wurden für die Implementierung und Ausführung von Unit- und Integrationstests verwendet. Mockito diente zur Erstellung von Mock-Objekten für die Isolierung von Testfällen.
- **Datenbank:** MySQL wurde als relationale Datenbank verwendet. Für die Tests wurde eine MySQL-Testdatenbank konfiguriert, um schnelle und isolierte Testausführungen zu ermöglichen. Für die Entwicklungsumgebung wurde XAMPP verwendet, das eine einfache Möglichkeit bietet, einen lokalen Webserver mit MySQL-Datenbank zu betreiben [9].

4.2 Implementierungsschritte

Die Implementierung am Beispiel der Benutzerregistrierung in der Todo-App folgte nach dem klassischen TDD-Zyklus: **Red-Green-Refactor**.

1. **Red Phase:** Zunächst wurde ein fehlgeschlagener Test (Red) geschrieben, der die Registrierung eines neuen Benutzers beschreibt, ohne dass die Implementierung vorhanden war.

Beispiel: Ein Test für die Registrierung eines neuen Benutzers über den UserController.

```
@Test
public void testRegisterUser_success() throws Exception {
    UserRegistrationRequest request = new UserRegistrationRequest();
    request.setUsername("integrationtestuser");
    request.setPassword("password");
    request.setConfirmPassword("password");

    ResultActions result = mockMvc.perform(post("/register")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)));

    result.andExpect(status().isOk());
}
```

2. **Green Phase:** Anschließend wurde der minimal notwendige Code geschrieben, um den Test erfolgreich zu bestehen (Green). In diesem Schritt wird nur so viel implementiert, dass der Testfall erfüllt wird.

Beispiel: Die grundlegende Implementierung des registerUser-Endpunkts im UserController.

```
@PostMapping("/register")
public ResponseEntity<?> registerUser(@RequestBody
    UserRegistrationRequest request) {
    User user = new User();
    user.setUsername(request.getUsername());
}
```

4.2. IMPLEMENTIERUNGSSCHRITTE

```
user.setPassword(passwordEncoder.encode(request.getPassword()));

return ResponseEntity.ok(user);
}
```

3. **Refactor Phase:** Nach dem erfolgreichen Bestehen des Tests wurde der Code optimiert und verbessert, ohne die Funktionalität zu ändern. Dabei wurde auf Sauberkeit, Lesbarkeit und Wartbarkeit des Codes geachtet.

Beispiel: Die vollständige Implementierung des registerUser-Endpunktes im UserController zeigt, dass ein Refactoring erfolgte, nachdem die Implementierung zum erfolgreichen Ausführen von Test, wie zum Beispiel des Testens der Fehlerbehandlung von bereits existierenden Benutzernamen, hinzugefügt wurde.

```
@PostMapping("/register")
public ResponseEntity<?> registerUser(@Valid @RequestBody
    UserRegistrationRequest request, BindingResult result) {
    // Check if username already exists
    if (userService.existsByUsername(request.getUsername())) {
        return ResponseEntity.status(HttpStatus.CONFLICT).body("Username
            already exists");
    }

    // Check if password and confirmation match
    if (!request.getPassword().equals(request.getConfirmPassword())) {

        return ResponseEntity.status(HttpStatus.CONFLICT).body("Password
            and confirm password do not match");
    }

    // Check for validation results
    if (result.hasErrors()) {

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("
            Validation error: " + result.getAllErrors());
    }
}
```

4.3. BEISPIELTESTS UND TESTFÄLLE

```
// Register user
try {
    User user = authenticationService.register(request);
    return ResponseEntity.ok(user);
} catch (Exception e) {
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).
        body("Error registering user");
}
}
```

Die Anwendung des TDD-Zyklus unterstützt eine stabile Implementierung und kann zu einem fehlerarmen Code führen.

4.3 Beispieltests und Testfälle

4.4 Unit-Tests

Unit-Tests wurden verwendet, um einzelne Komponenten isoliert zu testen.

Ein Beispiel ist der Test der `UserController`-Klasse, um sicherzustellen, dass eine Registrierung eines Benutzers erfolgreich verläuft.

```
@Test
public void testRegisterUserPasswordMismatch() {
    UserRegistrationRequest request = new UserRegistrationRequest("testuser", "
        password1", "password2");

    BindingResult result = mock(BindingResult.class);
    ResponseEntity<?> response = userController.registerUser(request, result);

    assertEquals(HttpStatus.CONFLICT, response.getStatusCode());
    assertEquals("Password and confirm password do not match", response.getBody
        ());
}
```

4.5 Integrationstests

Integrationstests wurden durchgeführt, um das Zusammenspiel verschiedener Komponenten zu überprüfen.

Ein Beispiel ist der Integrationstest für die Benutzerregistrierung über den `UserController`.

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class UserRegistrationAcceptanceTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void testRegisterUser() {
        String url = "http://localhost:" + port + "/register";
        UserRegistrationRequest request = new UserRegistrationRequest("testuser"
            , "password", "password");

        HttpHeaders headers = new HttpHeaders();
        HttpEntity<UserRegistrationRequest> entity = new HttpEntity<>(request,
            headers);

        // Send HTTP POST request
        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.
            POST, entity, String.class);

        // Check if the response status code is 409 (CONFLICT)
        assertEquals(HttpStatus.CONFLICT, response.getStatusCode());
    }
}
```

4.6 Akzeptanztests

Akzeptanztests wurden verwendet, um sicherzustellen, dass die Anwendung den Anforderungen der Benutzer entspricht. Diese Tests wurden aus der Sicht des Endbenutzers geschrieben und überprüfen die Funktionalität der gesamten Anwendung.

Ein Beispiel ist der Akzeptanztest für die Benutzerregistrierung über die REST-Schnittstelle.

```

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class UserRegistrationAcceptanceTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void testRegisterUser() {
        String url = "http://localhost:" + port + "/register";
        UserRegistrationRequest request = new UserRegistrationRequest("testuser"
            , "password", "password");

        HttpHeaders headers = new HttpHeaders();
        HttpEntity<UserRegistrationRequest> entity = new HttpEntity<>(request,
            headers);

        // Send HTTP POST request
        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.
            POST, entity, String.class);

        // Check if the response status code is 409 (CONFLICT)
        assertEquals(HttpStatus.CONFLICT, response.getStatusCode());
    }
}

```

Diese strukturierte Vorgehensweise durch TDD unterstützt eine robuste und fehlerarme

4.6. AKZEPTANZTESTS

Implementierung der Todo-Liste.

5 Frontend Entwicklung

5.1 Projektanforderung an Frontend-Entwicklung

Das Projekt stellt vielfältige Anforderungen an die Entwicklung des Frontends:

- **Einfachheit und Flexibilität:** Vue.js bietet eine sanfte Lernkurve und ist flexibel einsetzbar, was es ideal für verschiedene Projektanforderungen macht [10].
- **Einfache Integrität:** Vue.js ermöglicht eine einfache Integration in bestehende Projekte und Technologiestacks [10].
- **Reaktive Datenbindung:** Durch die reaktive Datenbindung können Änderungen automatisch auf der Benutzeroberfläche reflektiert werden, was die Entwicklung dynamischer Anwendungen erleichtert [10].
- **Komponentenbasierte Architektur:** Die komponentenbasierte Architektur fördert die Wiederverwendbarkeit und Wartbarkeit des Codes, indem sie die Trennung von Logik und Darstellung erleichtert [10].
- **Leistungsfähigkeit:** Vue.js bietet eine hohe Leistung und Effizienz bei der Erstellung von Benutzeroberflächen [10].
- **Große Community und gute Dokumentation:** Eine aktive Community und umfassende Dokumentation erleichtern die Lösung von Problemen und den Zugriff auf Ressourcen [10].

Die Wahl von Vue.js als Framework erfüllt diese Anforderungen und rechtfertigt daher die Entscheidung für seine Verwendung im Projekt.

5.2 Technologiestack

Das Frontend verwendet folgende Technologien und Frameworks:

- **Vue.js (Version 3.2.13)**: Haupt-Framework für die Erstellung der Benutzeroberfläche.
- **Vue Router (Version 4.4.0)**: Routing-Library für die Navigation innerhalb der Anwendung.
- **Vuex (Version 4.1.0)**: State-Management-Bibliothek für die zentralisierte Speicherung von Daten.

Das Projekt wurde mit Hilfe des Vue CLI (Command Line Interface) initialisiert, um eine standardisierte Projektstruktur und die notwendigen Abhängigkeiten bereitzustellen. Nach der Erstellung des Projekts wurden die zusätzliche Bibliotheken **axios** für HTTP-Anfragen und **vue-router** für die Navigation installiert.

5.3 Projektstruktur

Die Projektstruktur wurde so gestaltet, dass sie eine klare Trennung der einzelnen Komponenten und Funktionalitäten ermöglicht. Die Verzeichnisstruktur ist wie folgt:

- **/src**: Hauptverzeichnis für den Quellcode.
- **/src/components**: Enthält wiederverwendbare Vue-Komponenten.
- **/src/components/HeaderBar.vue**: Die Kopfzeile der Anwendung, die das Logo und die Logout-Option enthält, abhängig davon, ob der Benutzer angemeldet ist.
- **/src/views**: Enthält die Hauptansichten der Anwendung.
- **/src/views/IndexView.vue**: Die Startseite der Anwendung, die Optionen zum Anmelden und Registrieren bereitstellt.
- **/src/views/LoginView.vue**: Das Anmeldeformular, das Benutzernamen und Passwort für die Authentifizierung erfordert.
- **/src/views/RegisterView.vue**: Das Registrierungsformular, das Benutzer zur Erstellung eines Kontos ermöglicht.

5.4. ROUTING

- `/src/views/HomeView.vue`: Die Hauptansicht nach der Anmeldung, die eine Liste von Todos anzeigt.
- `/src/views/NewTodoForm.vue`: Ein Formular zur Erstellung neuer Todos.
- `/src/views/ToDoDetailView.vue`: Ansicht zum Bearbeiten eines bestimmten Todos.
- `/src/router.js`: Konfiguration der Vue Router-Instanz.
- `/src/store.js`: Vuex Store-Konfiguration für das State-Management.
- `/src/App.vue`: Wurzelkomponente der Anwendung.
- `/src/main.js`: Einstiegspunkt, wo die Vue-Instanz erstellt und konfiguriert wird.

5.4 Routing

Der Router wurde in der Datei `src/router.js` konfiguriert. Die Konfiguration umfasst die Definition der verschiedenen Routen, die jeweils einer spezifischen Komponente zugeordnet sind. Dies ermöglicht eine einfache Navigation zwischen den verschiedenen Ansichten der Anwendung.

5.5 Authentifizierung und Autorisierung

Die Webanwendung verwendet JSON Web Tokens (JWT) zur Authentifizierung von Benutzern. Nach erfolgreicher Anmeldung wird ein Token im Local Storage gespeichert und für alle folgenden Anfragen an den Server verwendet. Die Authentifizierung wird durch Vue Router Navigation Guards implementiert, um sicherzustellen, dass nur authentifizierte Benutzer auf geschützte Routen zugreifen können.

5.6 HTTP-Anfragen

HTTP-Anfragen werden mit Axios durchgeführt, um mit dem Backend-Server zu kommunizieren. Diese Anfragen werden in den Vue-Komponenten ausgeführt, um Daten zu

5.7. STATE MANAGEMENT

laden, zu speichern oder zu aktualisieren. Für jede wird der JWT-Token automatisch zum Header hinzugefügt, um sicherzustellen, dass der Server die Anfrage autorisiert.

5.7 State Management

Vuex wird verwendet, um den globalen Zustand der Anwendung zu verwalten. Es speichert den Anmeldestatus und den JWT-Token des Benutzers und bietet zentralisierten Zugriff auf diese Daten. Die State-Management-Logik wurde in der Datei `src/store.js` definiert.

5.8 Styling

Das Styling erfolgt hauptsächlich mit CSS innerhalb der einzelnen Vue-Komponenten. Das Scoped Styling von Vue.js sorgt dafür, dass die Styles nur auf die jeweilige Komponente angewendet werden.

6 Deployment

Um die Todo-Liste Webanwendung bereitzustellen und zu starten, sind mehrere Schritte erforderlich, einschließlich der Einrichtung der Datenbank und der Ausführung der Anwendung sowohl für das Frontend als auch das Backend.

6.1 Datenbank einrichten

Zunächst muss die MySQL-Datenbank eingerichtet werden, um die Daten der Todo-Liste speichern zu können. Für die Entwicklungsumgebung wurde XAMPP verwendet, das eine einfache Möglichkeit bietet, einen lokalen Webserver mit MySQL-Datenbank zu betreiben. Für die Konfiguration können die folgenden Informationen aus der `application.properties` verwendet werden:

```
spring.datasource.url=jdbc:mysql://localhost:3306/todoappdb
spring.datasource.username=springuser
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

das notwendige Datenbankdesign, kann aus dem Kapitel 3.2.3 Datenbankdesign entnommen werden.

6.2 Backend starten

Das Backend der Todo-Liste Webanwendung ist mit Spring Boot implementiert. Hier sind die Schritte zum Starten des Backends:

1. **Projekt herunterladen und öffnen:** Das Projekt muss zunächst von dem Versionskontrollsystem Github heruntergeladen werden und anschließend in einer bevorzugten IDE geöffnet werden [11].
2. **Konfigurieren der Datenbankverbindung:** Die Datenbankeinstellungen in der Datei „src/main/resources/application.properties“ müssen so angepasst sein, dass Zugriff auf die zuvor erstellte Datenbank ermöglicht werden kann.
3. **Starten der Backends:** Die Spring Boot-Anwendung kann mittels dem Aufrufen der „main“-Methode in der Hauptklasse („TodoApplication“) gestartet werden.

6.3 Frontend starten

Das Frontend der Todo-Liste Webanwendung basiert auf Vue.js. Hier sind die Schritte zum Starten des Frontends:

1. **Projekt herunterladen und öffnen:** Das Projekt muss zunächst von dem Versionskontrollsystem Github heruntergeladen werden und anschließend in einer bevorzugten IDE geöffnet werden [11].
2. **Installieren der Abhängigkeiten:** Mittels einer Befehlszeile kann zum Verzeichnis des Frontend-Projekts navigiert werden. Durch den Befehl „npm install“ werden alle Abhängigkeiten installiert, die in der „package.json“-Datei aufgeführt sind.
3. **Starten der Frontends:** Nachdem alle Abhängigkeiten installiert wurden, kann das Frontend mit dem Befehl „npm run serve“ gestartet werden.

6.4 Zugriff auf Todo-Liste Webanwendung

Nachdem sowohl das Backend als auch das Frontend gestartet wurde, kann auf die Todo-Liste Webanwendung zugegriffen werden. Hierfür muss ein Webbrowser geöffnet und zu

6.4. ZUGRIFF AUF TODO-LISTE WEBANWENDUNG

der URL „localhost:8081“ navigiert werden. Es erscheint die Startseite der Todo-Liste, in der sich registriert oder angemeldet werden kann.

7 Fazit

Diese Studienarbeit zielt darauf ab, eine intuitive Benutzererfahrung zu bieten, die es Benutzern ermöglicht, Todos zu erstellen, zu bearbeiten und zu löschen, sowie zusätzliche Details wie Beschreibungen und Fälligkeitsdaten hinzuzufügen.

Die entwickelte Todo-Liste besteht aus zwei Hauptkomponenten: dem Frontend, das auf Vue.js basiert und die Benutzeroberfläche bereitstellt, und dem Backend, das auf Spring Boot aufbaut und die Datenverarbeitung und -speicherung übernimmt. Die Integration mit einer MySQL-Datenbank bietet eine robuste Grundlage für die Speicherung von Benutzerdaten und Todos.

Trotz der erfolgreichen Implementierung der grundlegenden Funktionen der Todo-Liste gibt es noch einige Herausforderungen und unvollständige Aspekte, die erwähnt werden müssen. Ein Benutzer kann sich erfolgreich registrieren und anmelden, jedoch sind die Funktionen zur sicheren Abmeldung und zur Bearbeitung von Todos nicht vollständig implementiert. Dies führt zu Bedenken hinsichtlich der Privatsphäre und der Benutzerfreundlichkeit der Anwendung.

Die Gründe für diese unvollständigen Funktionen sind vielschichtig. Der begrenzte Erfahrungsschatz des Entwicklers mit den verwendeten Technologien sowie unzureichendes Zeitmanagement haben dazu geführt, dass der ursprüngliche Zeitplan nicht eingehalten werden konnte. Zudem wurde der Zyklus des Test-Driven Developments (TDD) nicht konsequent durchgeführt vor allem im Hinblick des Refactoring, was zu nicht bestandenen Tests und vermutlich fehlerhaftem Code führte. Diese Probleme haben sich direkt auf die Funktionalität der Anwendung ausgewirkt, insbesondere auf Bereiche wie das sichere Ausloggen und die Bearbeitung von Todos.

Für zukünftige Weiterentwicklungen und Verbesserungen der Todo-Liste Webanwendung ist es entscheidend, diese Herausforderungen zu adressieren. Ein verstärkter Fokus auf

Schulung und Weiterbildung in den verwendeten Technologien sowie eine verbesserte Projektplanung und -überwachung können helfen, ähnliche Probleme in zukünftigen Entwicklungsprojekten zu vermeiden. Darüber hinaus sollte die Implementierung von Best Practices im Bereich Softwareentwicklung, einschließlich eines strengeren TDD-Ansatzes und einer umfassenden Testabdeckung, Priorität haben, um die Qualität und Zuverlässigkeit der Anwendung sicherzustellen.

Insgesamt bietet diese Studienarbeit einen detaillierten Einblick in die Architektur, Implementierung und die Lessons Learned bei der Entwicklung der Todo-Liste Webanwendung. Trotz der identifizierten Herausforderungen legt sie den Grundstein für künftige Verbesserungen und Erweiterungen dieser digitalen Werkzeuge zur Aufgabenverwaltung.

Literatur

- [1] Alexander Schatten u. a. *Best Practice Software-Engineering*. de. Heidelberg: Spektrum Akademischer Verlag, 2010. ISBN: 978-3-8274-2486-0 978-3-8274-2487-7. DOI: 10.1007/978-3-8274-2487-7. URL: <http://link.springer.com/10.1007/978-3-8274-2487-7> (besucht am 18.03.2024).
- [2] Stephan Kleuker. *Qualitätssicherung durch Softwaretests: Vorgehensweisen und Werkzeuge zum Testen von Java-Programmen*. de. Wiesbaden: Springer Fachmedien, 2019. ISBN: 978-3-658-24885-7 978-3-658-24886-4. DOI: 10.1007/978-3-658-24886-4. URL: <http://link.springer.com/10.1007/978-3-658-24886-4> (besucht am 18.03.2024).
- [3] *MySQL :: MySQL Community Edition*. URL: <https://www.mysql.com/products/community/> (besucht am 23.06.2024).
- [4] *Was ist MySQL? | Data Basecamp*. de-DE. Section: Daten. März 2022. URL: <https://databasecamp.de/daten/mysql> (besucht am 23.06.2024).
- [5] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. en. URL: <https://www.nngroup.com/articles/ten-usability-heuristics/> (besucht am 01.06.2024).
- [6] *IntelliJ IDEA – die führende Java- und Kotlin-IDE*. de. URL: <https://www.jetbrains.com/de-de/idea/> (besucht am 23.06.2024).
- [7] *Maven – Welcome to Apache Maven*. URL: <https://maven.apache.org/> (besucht am 23.06.2024).
- [8] *Build software better, together*. en. 2024. URL: <https://github.com> (besucht am 23.06.2024).
- [9] *XAMPP Installers and Downloads for Apache Friends*. URL: <https://www.apachefriends.org/de/index.html> (besucht am 23.06.2024).

Literatur

- [10] *Vue.js*. en-US. URL: <https://vuejs.org/> (besucht am 23.06.2024).
- [11] *philippabele/tdd-with-java*. URL: <https://github.com/philippabele/tdd-with-java> (besucht am 23.06.2024).