Angular-Kurs Sommer 2025

Fachinformatiker Anwendungsentwicklung

Dokumentation zur schulischen Projektarbeit

Lernkarten Anwendung

mit CRUD Funktionalität (REST-API) und einem Web-Interface auf Basis einer Angular SPA

Abgabetermin:	Freitag 29. August 2025
Projektteilnehmer:	Max Gross
	Vasco Alexander Basque
	Artur Sanogrin
Ausbildungsstätte:	BITLC Business IT Learning Center GmbH
	Im Defdahl 10G
	44141 Dortmund
Kurs-Dozent:	Tom Selig

INHALTSVERZEICHNIS

1 EINLEITUNG

Dieses Dokument beschreibt das Softwareprojekt im Rahmen des Angular-Kurses der BITLC Learning Center GmbH. Der Dozent und Auftraggeber des Projekts ist Tom Selig.

Ziel des Projekts ist die Entwicklung einer webbasierten Anwendung zur Unterstützung der IHK-Prüfungsvorbereitung. Die Dokumentation erläutert die einzelnen Schritte von der Planung bis zum Abschluss des Projekts.

1.1 PROJEKTUMFELD

1.1.1 Unternehmensumfeld: BITLC Learning Center GmbH

Die BITLC Learning Center GmbH ist ein Bildungsträger mit Sitz in Dortmund, der Umschulungen und Fortbildungen im IT-Bereich anbietet. Der Schwerpunkt liegt auf der Betreuung von Umschülerinnen und Umschülern aus dem Großraum Dortmund.

1.1.2 Kunde: Tom Selig

Tom Selig leitet den Kurs „Angular Softwareprojekt“ und fungiert im Rahmen dieses Kurses als Auftraggeber.

1.2 PROJEKTZIEL

Das Ziel des Projekts ist die Entwicklung einer webbasierten Anwendung als Ergänzung zu den bestehenden Lernangeboten der BITLC Learning Center GmbH. Sie soll Umschülerinnen und Umschülern eine eigenständige Vorbereitung auf den ersten Teil der IHK-Prüfung ermöglichen und so zu einer höheren Erfolgsquote und besseren Noten beitragen.

Die Anwendung soll geräteunabhängig nutzbar sein. Sowohl das Backend als auch das Frontend werden auf der Infrastruktur des Kunden eigenständig installiert und betrieben.

1.3 PROJEKTBEGRÜNDUNG

Die bisherige Prüfungsvorbereitung erfolgt über heterogene analoge und digitale Materialien verschiedener Dozenten sowie externe Drittanbieter. Um den Lernprozess zu vereinheitlichen und die Pflege zu vereinfachen, wünscht die Geschäftsführung der BITLC Learning Center GmbH eine zentrale, webbasierte Lösung.

1.4 ABGRENZUNGSKRITERIEN

Eine Authentifizierung ist nicht vorgesehen. Benutzerdaten werden weder verarbeitet noch gespeichert.

1.5 PROJEKTSCHNITTSTELLEN

1.5.1 Technische Schnittstellen

Die Anwendung ist ein in sich geschlossenes System. Das Backend stellt eine REST-API[1] bereit, die vom Angular-Frontend genutzt wird. Beide Komponenten sind modular aufgebaut und können perspektivisch in ein größeres System integriert werden. Eine Integration ist jedoch nicht Teil dieses Projekts.

1.5.2 Organisatorische Schnittstellen

Grundlage des Projekts ist ein vom Auftraggeber bereitgestelltes Lastenheft. Auf dieser Basis wurde ein Pflichtenheft erstellt, das vom Auftraggeber bestätigt wurde.

2 PROJEKTPLANUNG

2.1 PROJEKTPHASEN

Das Projekt findet vom 18.08.25 bis 29.08.25 in den Schulungsräumen der BITLC GmbH statt. Täglich stehen etwa 8 Arbeitsstunden zur Verfügung, in denen das Projekt analysiert, geplant, umgesetzt und dokumentiert wird.

[1] REST = Representational State Transfer; API = Application Programming Ingerface;
Eine **REST-API** ist eine Art von Schnittstelle (API), die es verschiedenen Softwaresystemen ermöglicht, über das Internet miteinander zu kommunizieren.

Die Zeitplanung gliedert sich wie folgt:

Projektphase	Geplante Zeit
Analysephase	3 h
Konzeptionsphase	8 h
Planungsphase	5 h
Implementierung	32 h
Validierungsphase	4 h
Abschluss	4 h
Dokumentation	24 h
Gesamt	80 h

2.2 RESSOURCENPLANUNG

Für die Umsetzung stellt die BITLC GmbH einen Schulungsraum mit Linux-Desktoparbeitsplätzen bereit. Bei technischen Problemen stehen Dozenten und Systemadministratoren unterstützend zur Verfügung. Die Arbeitsplätze sind mit modernen Desktop-PCs, AMD-Prozessoren und 21-Zoll-Monitoren ausgestattet.

Die Entwicklungsumgebung wird mithilfe von VMware virtualisiert, sodass sowohl Windows 11 als auch Linux Mint flexibel genutzt werden können. Als IDEs kommen Visual Studio Code und JetBrains IntelliJ zum Einsatz. Für die Planung und Konzeption wird Open-Source-Software wie Draw.io genutzt, während Postman für das Testen der REST-API dient.

Typ	Bezeichnung	Rolle
Arbeitsgerät	3x Desktop PC mit Monitor	Entwicklungsgeräte zum Arbeiten und Testen
Betriebssysteme	Windows 11	Virtualisiert auf Rocky-Linux
	Linux Mint	Virtualisiert auf Rocky-Linux
Browser	Google Chrome	Arbeits- und Testbrowser
Entwicklungstools	Visual Studio Code	Entwicklungsumgebung

	Jetbrains IntelliJ	Entwicklungsumgebung
	Java OpenJDK 21	Backend
	SpringBoot 3.x	Backend-Framework
	Angular 19	Frontend
	Postman	API Dokumentation und Test
	DBeaver	DBMS, Datenbankkontrolle und Verwaltung
	Docker	Containerisierung
	Git, Github	Versionskontrolle
Konzeptionstools	draw.io	Zeichenprogramm für Diagramme / Wireframes
Office Tools	Microsoft Word	Textverarbeitung zur Erstellung der Dokumentation

2.3 ENTWICKLUNGSPROZESS

Die Entwicklung erfolgt nach einem **agilen, iterativen Ansatz**. Nach der Erstellung des Pflichtenhefts und einer ersten Konzeptionsphase startet ein Kickoff-Meeting, in dem die Aufgaben im Team verteilt und die Vorgehensweise abgestimmt werden.

Git und GitHub dienen als Versionskontrollsystem und Remote-Host, während ein **Kanban-Board** auf GitHub den Überblick über Aufgaben, Fortschritte und offene Punkte ermöglicht. Diese Kombination erlaubt ein **schnelles, flexibles Arbeiten**, in dem einzelne Komponenten der Anwendung frühzeitig implementiert, getestet und angepasst werden können.

Regelmäßige kurze Abstimmungen sorgen dafür, dass Probleme sofort erkannt werden, Prioritäten angepasst werden können und fertige Funktionalitäten direkt getestet werden. Dadurch bleibt das Team stets handlungsfähig und kann effizient auf neue Anforderungen oder Herausforderungen reagieren.

3 ANALYSEPHASE

3.1 IST-ANALYSE

Derzeit nutzen die Lernenden der BITLC GmbH verschiedene Insellösungen zur Prüfungsvorbereitung. Materialien liegen teilweise in Papierform vor, teilweise in digitalen Dateien, die jedoch nicht zentral organisiert sind. Ein strukturierter Zugriff auf Lerninhalte ist dadurch erschwert, was vor allem bei der gezielten Vorbereitung auf die IHK-Prüfung zu Zeitverlust und Redundanzen führt.

Eine zentrale Anwendung, die Lerninhalte sammelt, strukturiert und nutzerfreundlich bereitstellt, steht aktuell nicht zur Verfügung. Auch eine gemeinsame Fortschrittskontrolle oder Teamarbeit bei der Bearbeitung von Aufgaben ist bisher nicht möglich. Dies führt dazu, dass Teilnehmende ihre Unterlagen individuell verwalten und keine einheitliche Übersicht über den Lernstand besteht.

Das geplante System soll hier Abhilfe schaffen, indem es eine einheitliche Plattform mit klarer Struktur, zentraler Datenbank und intuitiver Bedienung bereitstellt. Auf diese Weise können die Lernenden effizienter arbeiten, sich gezielter auf Prüfungen vorbereiten und die Dozenten erhalten eine bessere Möglichkeit, Lernfortschritte nachzuvollziehen.

3.2 SOLL-KONZEPT

Das Zielsystem ist eine **webbasierte Lernanwendung**, die Umschülern der BITLC GmbH eine Unterstützung bei der Vorbereitung auf die IHK-Prüfung bietet. Die Anwendung besteht aus einem **Angular-Frontend** und einem **Spring-Boot-Backend**, die über eine REST-API kommunizieren. Die Inhalte werden in einer **PostgreSQL-Datenbank** gespeichert. Das System soll plattformunabhängig, leicht erweiterbar und sowohl im Lern- als auch im Prüfungsmodus nutzbar sein.

3.2.1 Fachliche Anforderungen

- Verwaltung von Lerninhalten in **Kartensets** (Flashcard-Sets mit Fragen und Antworten).
- Unterstützung der Fragetypen **Single-Choice**, **Multiple-Choice** und **Fill-In Text**.
- **Lernmodus**: sofortiges Feedback, Möglichkeit zur Anzeige der richtigen Lösung.
- **Prüfungsmodus**: zeitlich limitierte Simulation, Ergebnisübersicht am Ende.
- Fortschrittsanzeige während der Bearbeitung („Frage x von y“).
- Keine Benutzerkonten, keine Speicherung von persönlichen Daten oder Ergebnissen.
- *[Optional]* Import und Export von Kartensets im JSON-Format.

3.2.2 Technische Anforderungen

- **Frontend**: Angular (SPA), responsive Darstellung für Desktop und Mobile.
- **Backend**: Java 21 mit Spring Boot 3.x, Bereitstellung einer REST-API.
- **Datenbank**: PostgreSQL mit relationalem Schema, Speicherung der Sets und Karten über ein Many-to-Many-Mapping.
- **Containerisierung**: Bereitstellung über Docker und Docker-Compose.
- **Kompatibilität**: Nutzung in aktuellen Browsern (Chrome, Firefox, Edge, Safari).
- **Entwicklungsumgebung**: Linux- & Windows-Arbeitsplätze, Virtualisierung mit VMware, IDEs (Jetbrains IntelliJ, Visual Studio Code).

3.3 WIRTSCHAFTLICHKEITSANALYSE

In der Wirtschaftlichkeitsanalyse wird folgend eine Make-or-buy Entscheidung getroffen, die Projektkosten ermittelt und die Amortisierungsdauer bestimmt.

3.3.1 Make or Buy-Entscheidung

Da das Projekt im Rahmen des Angular-Kurses beim Bildungsträger BITLC durchgeführt wird, liegt die interne Entwicklung der Software nahe. Ziel des Kurses ist es, praxisnah Anwendungen zu konzipieren und umzusetzen. Eine externe Beschaffung („Buy“) wäre in diesem Kontext weder wirtschaftlich sinnvoll noch zielführend, da die Lern- und Ausbildungsziele damit nicht erreicht würden.

Darüber hinaus bietet die interne Entwicklung den Vorteil, dass die Anwendung ohne zusätzliche Lizenzkosten betrieben werden kann und bei Bedarf flexibel angepasst oder erweitert werden kann. Dadurch entstehen keine langfristigen Abhängigkeiten von Drittanbietern, und Folgekosten für spätere Weiterentwicklungen bleiben gering.

3.3.2 Projektkosten

Die Projektkosten setzen sich zusammen aus den Personalkosten für drei Kursteilnehmer und einen Dozenten, sowie den Sachkosten für Räumlichkeiten, Arbeitsmaterialien, Software und Verpflegung.

3.3.2.1 Personalkosten

Als Grundlage zur Berechnung der Personalkosten werden folgende Annahmen getroffen:

Monatsgehalt Umschüler: 1150 EUR
Monatsgehalt Dozent: 4700 EUR

Bei 20 Arbeitstagen im Monat und 8 Arbeitsstunden pro Tag ergeben sich daraus folgende Stundensätze:

Umschüler: 1150 Euro / 160 Stunden = **7,19 Euro**
Dozent: 4700 Euro / 160 Stunden = **29,38 Euro**

Personal	**Stundensatz**	**Stunden**	**Gesamt**
*Umschüler **x3***	7,19 EUR	240	1725,60 EUR
Dozent	29,38 EUR	4	117,52 EUR
Gesamt			**1843,12 EUR**

3.3.2.2 Sachkosten

Die Kosten der Sachmittel setzen sich aus den häuslichen Kosten (Miete, Strom), dem Arbeitsmaterial, der Software und der Verpflegung zusammen. Alle Kosten werden auf 3 Personen und auf die Dauer des Projekts von 80 Stunden herunter gerechnet.
Sachmittel die keine Kosten verursachen werden nicht extra aufgelistet. Die Kosten für das Arbeitsmaterial (Desktop-PC) sind auf eine Gesamtbetriebszeit von 5 Jahren kalkuliert.

Sachmittel	**Gesamt bei 80h**
Miete	45,33 EUR
Strom	18,45 EUR
PC mit Bildschirm, Tastatur und Maus	49,31 EUR
Microsoft Office 365	16,43 EUR
Verpflegung (Kaffee, Wasser)	Ca. 4 EUR
Gesamt	**133,52 EUR**

3.3.3 Amortisierung

Wird die Software im Gesamtbetrieb für alle Umschulungsklassen zum Einsatz gebracht, kann die BITLC GmbH Kosten für Druckmaterialien, Personalkosten sowie Lizenzkosten durch Drittanbieter Software einsparen.

Nutzenaspekt	**Begründung**	**Monetärer Wert (EUR/Jahr)**
Einsparung Materialkosten	Weniger gedruckte Lernmaterialien (~4 € pro Teilnehmer × 80 Teilnehmer/Jahr)	360 EUR
Effizienzsteigerung Dozenten	Kürzere Prüfungswiederholungen & strukturierteres Lernen (~10 h/Jahr × 30 €/h)	300 EUR
Einsparung Drittanbieterkosten	Wegfall externer Lernsoftware (~5 € pro Teilnehmer × 80/Jahr)	400 EUR
Gesamt		**1060 EUR**

3.3.3.1 Amortisierungsrechnung

Nach circa 2 Jahren hat sich die Anwendung voraussichtlich amortisiert.

Gesamtkosten: 1976,64 EUR

Jährlicher Nutzen: 1060 EUR

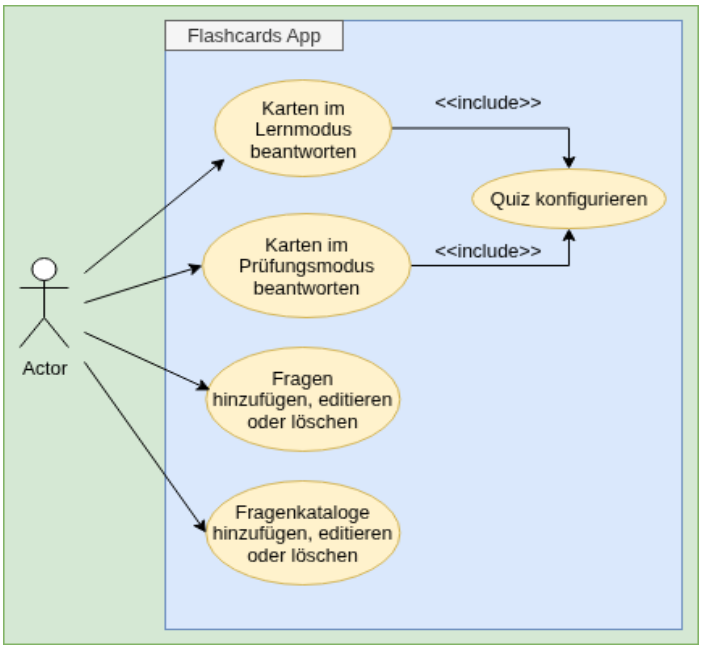
1976,64 EUR / 1060 EUR = **1,86 Jahre**

3.3.3.2 Nicht monetäre Vorteile

Eine eigene professionelle Softwarelösung für die Prüfungsvorbereitung trägt dazu bei, das **Image des Unternehmens** zu verbessern und auch künftig dank hoher Erfolgsquoten einen hohen **Zulauf an Umschulungsbewerbern** zu generieren. Die Arbeitserleichterung für Dozenten kann die **Mitarbeiterzufriedenheit** steigern.

3.4 ANWENDUNGSFÄLLE

Die Anwendung soll durch eine **intuitive und benutzerfreundliche Oberfläche** eine einfache Bedienung ermöglichen, sodass sich Nutzer schnell zurechtfinden.
Der zentrale Anwendungsfall besteht darin, **Lernkarten im Lern- oder Prüfungsmodus** zu beantworten. Darüber hinaus können Nutzer bei Bedarf **eigene Karten erstellen**, diese bestehenden **Fragenkatalogen hinzufügen** oder **neue Kataloge anlegen**. Auf diese Weise bleibt die Anwendung flexibel und an individuelle Lernbedürfnisse anpassbar.



4 ENTWURFSPHASE

4.1 ZIELPLATTFORM

Für das Projekt wurde eine Client-Server-Architektur umgesetzt. Diese gewährleistet eine klare Trennung zwischen Geschäftslogik im Backend und der Benutzeroberfläche im Frontend. Die gewählte Architektur ermöglicht eine hohe Wartbarkeit, Erweiterbarkeit und Skalierbarkeit der Anwendung.

Im Backend kommt Java 21 in Verbindung mit dem Framework Spring Boot 3.x zum Einsatz. Die Entscheidung fiel auf diese Plattform, da sie eine stabile und performante Laufzeitumgebung bereitstellt und durch starke Typisierung eine langfristige Wartung erleichtert. Eine Evaluierung von Node.js ergab, dass die dortige Single-Thread-Architektur bei CPU-intensiven Aufgaben zu Einschränkungen führen kann. Aus diesem Grund wurde Java/Spring Boot als geeignete Lösung ausgewählt.

Das Frontend wurde mit Angular 19 (TypeScript) entwickelt, wie es im Lastenheft spezifiziert wurde. Für die Datenhaltung wird PostgreSQL eingesetzt, da die zugrunde liegenden Daten stark strukturiert sind und sich daher für ein relationales Datenbanksystem eignen. Die Prüfung von NoSQL-Datenbanken zeigte, dass diese für die Anforderungen keinen zusätzlichen Nutzen bieten.

Die Anwendung ist plattformunabhängig nutzbar. Auf Client-Seite wird lediglich ein aktueller Webbrowser benötigt. Auf Server-Seite genügt eine Java Virtual Machine (JVM) sowie eine Container-Laufzeitumgebung (z. B. Docker Engine). Dadurch bleibt die Lösung weitgehend unabhängig von der eingesetzten Hardware und Betriebssystemumgebung.

4.2 ARCHITEKTURDESIGN

Das Projekt ist in eine **Frontend**- und eine **Backend-Komponente** unterteilt.

Das **Frontend** („flashcards-ui“) wurde mit **Angular** (TypeScript) als Single-Page-Application umgesetzt. Es basiert auf wiederverwendbaren Komponenten für die Benutzeroberfläche. API-Aufrufe erfolgen über Angular-Services mittels HTTP/JSON, die Navigation wird durch den Angular-Router gesteuert.

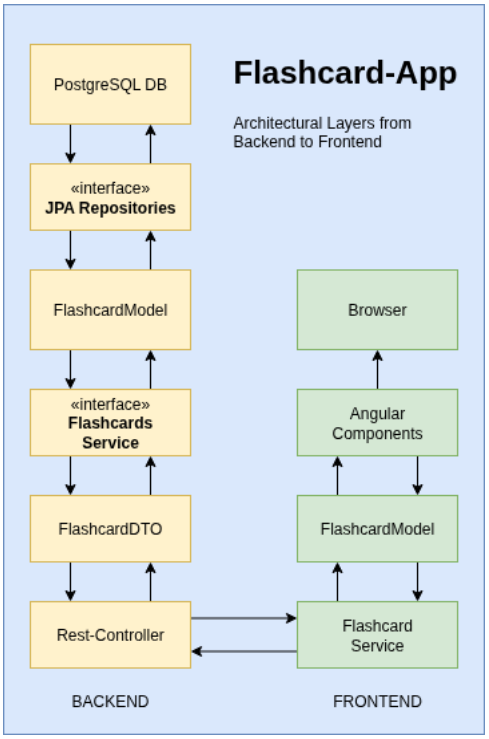
Das Backend („flashcards-api“) wurde mit Java 21 / Spring Boot 3.x entwickelt und stellt eine zustandslose REST-API bereit. Es folgt einem mehrschichtigen Aufbau:

- Controller definieren die Endpunkte und übernehmen Validierung sowie Mapping von DTOs.
- Services enthalten die Geschäftslogik, z. B. die Auswertung von Quiz-Antworten.
- Repositories realisieren den Datenzugriff über Spring Data JPA.
- Entities modellieren die Domäne und werden über Hibernate auf eine PostgreSQL-Datenbank abgebildet.

Die Datenbank wird containerisiert in Docker betrieben. Initialdaten können über SQL-Skripte eingebunden werden. Für den Datenaustausch werden ausschließlich DTOs eingesetzt. Dies reduziert die übertragene Datenmenge und sorgt für stabile Schnittstellen.

Durch die klare Trennung der Verantwortlichkeiten bleibt die Anwendung gut wartbar, testbar und skalierbar. Frontend und Backend können unabhängig voneinander weiterentwickelt und betrieben werden.

Das folgende Diagramm verdeutlicht noch einmal die Struktur innerhalb des Projekts:



4.3 Entwurf der Benutzeroberfläche

Die Benutzeroberfläche besteht aus Angular-Komponenten. Die Darstellung erfolgt über HTML-Templates, SCSS und dem durch TypeScript definiertem Verhalten. Die App ist als Single-Page-Application aufgebaut. Diese Struktur ist projektseitig vorgegeben, Alternativen wurden daher nicht betrachtet.

Wireframe der Kartenset-Übersicht

Geplant ist eine Kartenset-Übersicht, die alle Lernkartensets anzeigt. Über den Button „Add Set“ können neue Sets erstellt und gespeichert werden. Das neue Set erscheint anschließend in der Liste. Per Klick werden dem Nutzer die Karten in dem Set angezeigt. Über „Edit Set “ lassen sich neue Karten, der Title sowie die Beschreibung hinzufügen. Nach dem Speichern wird die Liste aktualisiert. Löschen ist direkt aus der Liste möglich. Die Navigation zurück erfolgt über den zurück Button. Auf der Hauptseite befindet sich ein Button „Lernsession starten“ der in den Lernmodus sowie Prüfungsmodus führt, in dem Fragen nacheinander beantwortet und der Fortschritt angezeigt wird.

Wireframe der Startseite

4.4 DATENMODELL

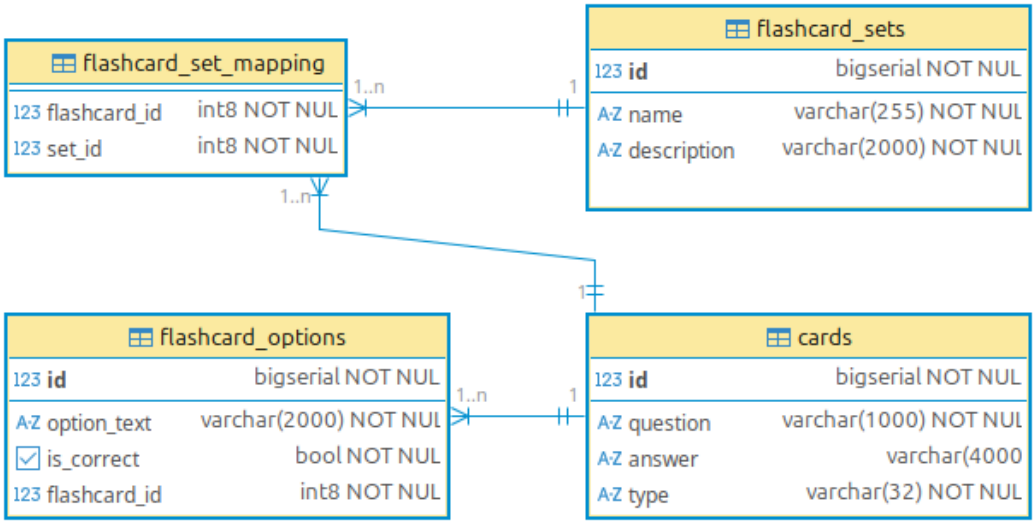
Das Datenmodell basiert auf einer **relationalen PostgreSQL-Datenbank**, die mithilfe von **Spring Data JPA/Hibernate** angebunden ist.

Wesentliche Entitäten sind:

- **Flashcard**: repräsentiert eine einzelne Lernkarte mit Frage, Antwort, Typ (Multiple Choice, Single Choice, Text) und zugehörigen Antwortoptionen.
- **FlashcardOption**: repräsentiert mögliche Antwortoptionen für Single- und Multiple-Choice-Fragen. Jede Option ist einer Karte zugeordnet.
- **FlashcardSet**: stellt eine Sammlung von Lernkarten dar, die thematisch gruppiert werden.

Die Beziehung zwischen **Flashcard** und **FlashcardSet** ist als **Many-to-Many** über eine Zwischentabelle (*flashcard_set_mapping*) realisiert. Damit können Karten mehreren Sets zugeordnet werden, ohne Daten zu duplizieren.

Zusätzlich wurden **DTOs (Data Transfer Objects)** implementiert, um eine klare Trennung zwischen Persistenzschicht und API-Schnittstelle zu gewährleisten.



5 IMPLEMENTIERUNGSPHASE

5.1 IMPLEMENTIERUNG DER REST-API

Die REST-API wurde mit **Spring Boot** realisiert und stellt die zentrale Schnittstelle zwischen Frontend und Backend dar. Die API ist nach dem Prinzip der **Ressourcenorientierung** aufgebaut und folgt gängigen Konventionen (CRUD-Endpunkte mit den HTTP-Methoden *GET, POST, PUT, DELETE*).

5.1.1 Flashcard Controller Endpunkte

Methode	Endpunkt	Funktionalität
GET	/api/flashcards?type={type}&set={setId}	Alle Karten optional gefiltert
GET	/api/sets/{setId}/flashcards	Karten eines Sets
GET	/api/flashcards/{flashcardId}	Karte nach ID
PUT	/api/flashcards	Karte aktualisieren
POST	/api/flashcards	Neue Karte anlegen
DELETE	/api/flashcards/{flashcardId}	Karte löschen

5.1.2 Flashcard Set Controller Endpunkte

Methode	Endpunkt	Funktionalität
GET	/api/sets	Alle Sets
GET	/api/sets/{setId}	Set nach ID
POST	/api/sets	Neues Set anlegen
PUT	/api/sets	Set aktualisieren
DELETE	/api/sets/{setId}	Set löschen

Die API verarbeitet und liefert **JSON-Daten** und ist so konzipiert, dass sie von einem Angular-Frontend oder auch anderen Clients problemlos konsumiert werden kann. Die Validierung von Eingaben sowie die Fehlerbehandlung erfolgt zentral über Exception Handler.

JSON Format zum Datenaustausch zwischen Backend und Frontend (beispielhaft)

```
{
        "question": "Nenne eine HTTP Methode",
        "answer": "GET",
        "type": "TEXT",
        "options": []
}
```

5.2 Implementierung der Benutzeroberfläche

5.2.1 Implementierung Grundstruktur Sets und Karten

Im ersten Schritt haben wir die Grundstruktur für Sets und Karten umgesetzt.

Serverseitig haben wir die Domänenschicht (Entities FlashcardSet, Flashcard, FlashcardOption) sowie Repositories und Services angelegt. Parallel dazu haben wir im Shared-Bereich DTOs definiert und Mapper für Entities sowie DTOs erstellt. Dabei achten wir strikt auf konsistente Datentypen: Das Kartenfeld type führen wir im Backend als ENUM, in DTOs/Frontend als String-Literal. Mehrfachantworten bilden wir über flashcard_options mit mit dem Feld is_correct ab, Freitext nutzt das Feld cards.answer.

Für die CRUD-Funktionalität haben wir zwei zentrale REST-Controller bereitgestellt (/sets, /cards). Die Controller validieren Eingaben und delegieren an die Services. Datenzugriffe laufen über Spring Data JPA. Zur Konsistenz setzen wir u. a. eindeutige Constraints (z. B. Set-Name) und modellieren die n:m-Beziehung zwischen Karten und Sets über eine Mapping-Tabelle.

Clientseitig haben wir die Angular-Grundstruktur folgendermaßen aufgebaut: Standalone-Komponenten, Router und Services. Die Sets-Übersicht lädt alle Sets, unterstützt Such- und Filter Funktionen und bietet die Moeglichkeit Sets anzulgen. Die Set-Detailansicht zeigt die zugehörigen Karten. Hier können Karten erstellt, bearbeitet und gelöscht werden. API-Aufrufe kapseln wir im FlashcardsService (HTTP/JSON).

Formulare sind als Reactive Forms umgesetzt, inkl. Inline-Validierung (Pflichtfelder, Antwortoptionen, Fragetyp).

Die Zuordnung Karte und Set nehmen wir nach der Set-UI vor: Karten werden zunächst erstellt und anschließend dem gewählten Set zugeordnet. Für Multiple-Choice erzwingen wir mindestens zwei Optionen und mindestens eine korrekte, bei Freitext ist answer pflichtig.

Damit haben wir die Grundfunktionen für Sets und Karten umgesetzt.

5.2.2 Implementierung Grundstruktur Quiz

Wir haben als Nächstes die Quiz-Funktion umgesetzt: Set wählen, Fragen beantworten und am Ende das Ergebnis sehen.

Im FlashcardSetController steht dafür der Endpunkt GET /api/sets/{setId} bereit. Der Service lädt alle Karten des Sets (inklusive Antwortoptionen), mischt die Reihenfolge optional pro Sitzung und gibt die Daten als DTOs zurück. Bei Textfragen enthält das DTO die erwartete Antwort, bei Single-/Multiple-Choice die Optionen mit Markierung der korrekten Antworten.

Die angelegte Quiz-Struktur haben wir wie folgt befüllt: quiz-config zeigt die verfügbaren Sets; mit „Quiz starten“ navigieren wir zu /quiz/play/:setId. quiz-question rendert je nach Fragetyp (Text/Single/Multi) das passende UI-Element, nimmt die Antwort entgegen und zeigt direkt Feedback (richtig/falsch). Mit „Nächste Frage“ geht es weiter, bis alle Fragen durch sind. quiz-result fasst richtig/gesamt und Prozent zusammen und bietet Neustart oder Rückkehr zur Übersicht, optional wird eine kurze Fehlerliste angezeigt.

Die Logik liegt im quiz.service: startQuiz(setId) lädt die gemischten Karten, setzt den Index auf 0 und leert den Antwortzustand. submitAnswer(…) speichert die Eingabe und prüft die Korrektheit (Single: genau eine richtige, Multi: Mengenvergleich, Text: getrimmt und case-insensitive). getProgress() und getResults() liefern Fortschritt und Endergebnis.

Technisch halten wir den UI-Zustand mit Signals (Index, Auswahl, Feedback). API-Aufrufe laufen über HttpClient/Observables und werden bei Bedarf mit toSignal() überführt das bleibt performant und gut lesbar. Leere Auswahl ist bei Single/Multi nicht möglich (Button

bleibt deaktiviert), bei Text ist die Eingabe Pflicht. Netzwerkfehler signalisieren wir per Snackbar mit Retry. Falls ein Set keine Karten hat, zeigen wir einen Leerzustand mit Link zur Sets-Übersicht. Optional sichern wir den Fortschritt im sessionStorage, damit ein Refresh nicht alles löscht.

Damit ist die Grundstruktur für den Quiz-Bereich umgesetzt.

6 TESTPHASE UND QUALITÄTSKONTROLLE

Wir haben die Anwendung während der Implementierung fortlaufend manuell getestet und die definierten Testszenarien T1–T5 am Stück ausgeführt und protokolliert. Jeder neue Build musste die bereits bestandenen Tests erneut bestehen. Backend (Spring Boot + PostgreSQL in Docker) und Frontend (Angular) liefen in einer gemeinsamen Testumgebung.

6.1 T1 - BROWSERZUGRIFF (BEDIENBARKEIT DER ANWENDUNG)

Die Startseite lud ohne Fehlermeldungen. Wir konnten zur Sets-Verwaltung, Karten-Verwaltung und in den Quiz-Bereich navigieren. Im Quiz-Auswahlbildschirm waren „Set wählen“ und „Karten mischen“ bedienbar. Beim Umschalten auf den Prüfungsmodus wurde die Zeitlimit-Eingabe korrekt eingeblendet. Es traten keine UI-Blocker und keine Konsolenfehler auf.

6.2 T2 - WEB API ANBINDUNG (DATENABRUF, CRUD, FEHLERBEHANDLUNG)

Die Liste der Sets und die Karten eines gewählten Sets wurden korrekt geladen. Anlegen, Bearbeiten und Löschen von Set und Karte funktionierten und waren in der Datenbank nachvollziehbar. Der Abruf der GET-Methode übernahm die Optionen (Karten mischen, Modus, Zeitlimit) wie konfiguriert. Ein absichtlich provozierter API-Ausfall zeigte eine verständliche Snackbar mit Retry, die App blieb bedienbar.

6.3 T3 - Reihenfolge und Konsistenz (Listen & Zufall)

Ohne „Karten mischen“ blieb die Reihenfolge stabil (deterministisch nach ID) über mehrere Reloads. Anzahl und Inhalte waren identisch, es gab keine Duplikate. Mit aktivierter Option änderte sich die Reihenfolge mindestens an einer Position pro Lauf, die Anzahl blieb unverändert und es gab keine Duplikate.

6.4 T4 - Validierung und Bewertung (Formulare & Quiz-Ergebnis)

In den Formularen wurden Pflichtfelder erzwungen, sowie doppelte Set-Namen wurden abgewiesen. Bei Multiple-Choice musste mindestens eine korrekte Option gesetzt sein und bei Textfragen wurde die Antwort validiert. Im Lernbereich stimmten die Bewertungen „richtig/falsch“ mit den Katalogdaten überein. Bei „Lösung anzeigen“ zeigte die korrekten Optionen bzw. die erwartete Textantwort. Im Prüfungsmodus wurden Antworten ohne vorzeitige Lösungseinblendung gespeichert und ein gesetztes Zeitlimit sowie „Quiz beenden“ beendete den Durchlauf wie vorgesehen. Die Ergebnissicht zeigte Gesamt, richtig, falsch, übersprungen sowie die Benotung korrekt. In „Details anzeigen“ funktionierten die Filterfunktion von Alle, Richtig, Falsch und Übersprungen. Der Zähler und die Listen stimmten überein.

6.5 T5 Navigation und Datenintegrität

Die Navigation zwischen Start, Sets-Liste, Karten-Liste/-Detail und Quiz funktionierte. Im Lernbereich arbeiteten „Vorherige/Nächste Frage“, „Antwort anzeigen“ und „Quiz beenden“ wie erwartet. Im Prüfungsmodus funktionierten „Vorherige/Nächste Frage“, „Antwort speichern“ und „Quiz beenden“. Beim Verlassen von Bearbeitungsformularen ohne Speichern erscheint die Abfrage zum Verwerfen, gespeicherte Daten blieben erhalten. Aus der Ergebnisansicht führten „Details anzeigen“ und „Neue Prüfung starten“ zuverlässig zurück bzw. in eine neuen Quiz-Vorbereich.

Gesamtfazit: Alle Tests wurden durchgeführt und erfolgreich bestanden. Bedienbarkeit, API-Anbindung inkl. CRUD, Reihenfolge, Logik, Ergebnisdarstellung und Navigation sind funktionsfähig und stabil.

7 ABNAHME

Nach Abschluss der Anwendung wurden alle fünf von uns konkret ausgearbeiteten Testszenarien erfolgreich durchgeführt. Die Software zeigte in sämtlichen Szenarien das erwartete Verhalten und alle Funktionen verhielten sich wie vorgesehen. Während des Projekts wurden regelmäßige Abstimmungen durchgeführt, sodass die Abnahme ohne Schwierigkeiten verlief. Das Projekt erfüllt alle Kriterien, die für die Anwendung wichtig und funktional notwendig sind. Neben den geforderten Kriterien wurden auch optionale Anforderungen berücksichtigt, die jedoch nicht vollständig umgesetzt werden konnten.

Ein solches Kriterium war beispielsweise die Erweiterbarkeit der Anwendung um verschiedene Themenbereiche, sodass zu jedem gewählten Thema passende Kataloge angeboten werden können. Derzeit wird jedoch nur ein Themenbereich unterstützt, der aus festgelegten Kartensets besteht.

Neben dem normalen Lernmodus wurde ein Prüfungsmodus implementiert, der in Zukunft um zusätzliche Funktionen erweitert werden soll. Wir konnten eine stabile Basis für die Modularität der Anwendung aufbauen. Dadurch ist eine schnellere und einfachere Wartung und Erweiterbarkeit der Anwendung in Zukunft möglich. Somit kann die Anwendung besser um weitere Funktionen erweitert werden. Das Projekt wurde termingerecht und gemäß den Vorgaben umgesetzt. Das Pflichtenheft wurde vollständig und in vollem Umfang erfüllt. Auf dieser Grundlage wurde die Anwendung als funktionsfähig bewertet und als abgenommen eingestuft. Im weiteren Verlauf ist eine Präsentation des Projekts für den Auftraggeber vorgesehen.

8 DOKUMENTATION

Im Rahmen des Projekts wurde eine Anwenderdokumentation erstellt. Diese beschreibt die wichtigsten Funktionen sowie die Bedienung der Anwendung und dient als Unterstützung für neue Anwenderinnen und Anwender. Die entsprechende Anleitung ist als separate Datei dem Dokumentations-Ordner (*/flashcards-app/docu*) zu entnehmen: *Benutzer_Handbuch_Lernkarten_Anwendung.pdf*.

9 FAZIT

9.1 SOLL-/IST-VERGLEICH

Das Projekt erfüllt alle Kriterien, die für die Anwendung wichtig und funktional notwendig sind. Nach 80 Stunden ist die Anwendung erfolgreich entwickelt worden und neben den geforderten Kriterien wurden auch optionale Anforderungen berücksichtigt. Bei der Konzeptionsphase hatten wir mehr Zeit benötigt als vorgesehen. Dies stellte anfangs Sorgen dar, ob das Projekt gefährdet sein könnte. In der Projektdurchführung konnten wir feststellen, dass wir einen Vorteil hatten, weil wir eine gute Grundstruktur in der Konzeption für die Entwicklung der Anwendung aufbauen konnten und dadurch die Implementierung der Funktionen strukturierter und schneller durchführen konnten. Wir benötigten wir weniger Zeit und waren eine Stunde schneller in der Projektdurchführung im Ist-Zeit fertig als im Soll-Zeit. Der vollständige Soll-/Ist-Vergleich ist unter Anhang 10.2 zu finden.

9.2 AUSBLICK

Die im Soll-/Ist-Vergleich genannten Erweiterungen und Funktionen könnten in Zukunft implementiert werden, um den vollen Funktionsumfang der Anwendung anbieten zu können. Weitere Erweiterungen und neue Funktionalitäten sind derzeit nicht vorgesehen und ergeben sich möglicherweise erst, wenn Lernende die Anwendung nutzen und durch ihr Feedback sowie konstruktive Kritik Verbesserungsvorschläge und Wünsche einbringen.

9.3 GEWONNENE ERKENNTNISSE

Abschließend werden die gewonnenen Erkenntnisse der Projektteilnehmer dargestellt und eine Reflexion über den Projektverlauf vorgenommen. Wir haben ein vertiefendes Verständnis des Angular-Frameworks sowie des Zusammenspiels von Frontend und Backend gewonnen.

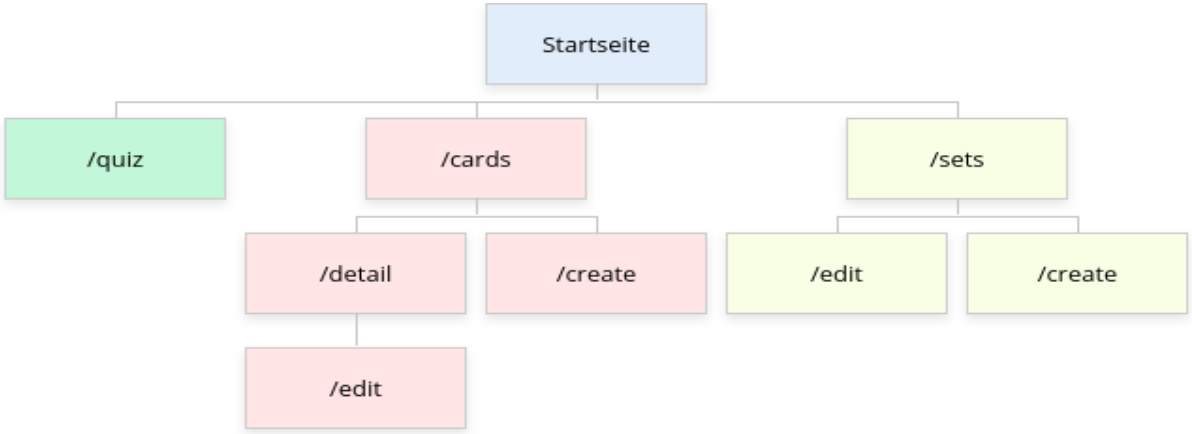
Besonders Letzteres war interessant, da das Backend abweichend von der empfohlenen Variante des Auftraggebers umgesetzt und mithilfe von Docker für die Containerisierung realisiert wurde. Insbesondere die Auseinandersetzung mit Docker ermöglichte es uns, die Anwendung in einer stabilen und reproduzierbaren Umgebung

bereitzustellen. Dies trug zu einer professionelleren Entwicklung der Anwendung bei. Dadurch konnten wir wertvolle Erfahrungen im Umgang mit modernen Entwicklungsprozessen sammeln.

Auch der Wissensaustausch zwischen uns erwies sich als große Bereicherung für die Technologien, die wir im Projektverlauf eingesetzt haben und auch auf die Zusammenarbeit im Team.

10 Anhang

10.1 Navigation in der Anwendung



10.2 Zeitplanung mit Soll-/Ist-Vergleich

Projektphase	Soll-Zeit	Ist-Zeit	Differenz
Projektdefinition			
Ist-Analyse	1 h	1 h	0 h
Use-Case	2 h	2 h	0 h
Planung			
Soll-Konzept	1 h	2 h	0 h
Wirtschaftlichkeitsanalyse	2 h	2 h	0 h
Zeitplanung	2 h	2 h	0 h
Konzeptionsphase			
Konzeptskizzen erstellt	4 h	5 h	+1 h
Architektur und Datenmodell gestalten	4 h	5 h	+1 h
Projektdurchführung			
Datenstrukturen implementieren	10 h	10 h	0 h
REST-API implementieren	10 h	9 h	-1 h
Benutzeroberfläche entwickelt	12 h	11 h	-1 h
Validierungsphase			
Testszenarien durchführen	4 h	4 h	0 h
Abschluss			
Soll-/Ist-Vergleich	2 h	2 h	0 h
Abgabe	2 h	2 h	0 h
Dokumentation			
Dokumentation schreiben	21 h	21 h	0 h
Benutzerhandbuch	1 h	1 h	0 h
Schnittstellen der Web-API beschreiben	1 h	1 h	0 h
Gesamt	80 h	80 h	0 h