# Erläuterung der grundsätzlichen Struktur

Das Projekt ist als Monorepo strukturiert und besteht aus zwei klar getrennten Hauptbereichen:

Backend (/backend)

Frontend (/frontend)

Zusätzlich gibt es im Wurzelverzeichnis verschiedene Konfigurations- und Setup-Dateien (z.B. docker-compose.yml, Setup-Skripte, .env.example).

# 1. Backend (/backend)

## 1.1. Genutzte Frameworks & Bibliotheken:

* Express: Haupt-Framework für den HTTP-Server.
* TypeORM: Objekt-Relationales Mapping für die Verbindung zu einer MySQL-Datenbank.
* MySQL2: Datenbank-Client für Node.js.
* Zod: Für die Validierung von Request-Daten (bspw. Body, Params).
* envalid: Für das sichere und typsichere Einlesen und Validieren von Umgebungsvariablen.
* argon2: Zum sicheren Hashen von Passwörtern.
* jsonwebtoken: Für das Ausstellen und Verifizieren von JWT-Tokens (User-Authentifizierung).
* helmet: Fügt Sicherheits-Header hinzu.
* express-rate-limit: Schützt die API vor zu vielen Anfragen.
* cookie-parser: Für das Auslesen von Cookies (insbesondere Auth-Token).
* dotenv: Lädt Umgebungsvariablen aus .env-Dateien.
* ESLint, Prettier: Für Code-Qualität und Formatierung.

## 1.2. Aufbau im Backend:

Das Backend läuft mit Node.js, Express und TypeScript.

TypeORM dient als ORM für die Anbindung an eine MySQL-Datenbank.

### 1.2.1. Wichtige Strukturelemente:

* controllers/: Enthält die Controller-Logik für die REST-API-Endpunkte.
* entity/: Die TypeORM-Entities – also die Klassen, die den Tabellen (User, Topic, Subtopic, Flashcard) in der Datenbank entsprechen.
* middleware/: Zentrale Express-Middlewares (Authentifizierung, Fehlerbehandlung, Rate-Limiting etc.).
* routes/: Routen-Definitionen für API-Endpunkte (z.B. /auth, /topics, /flashcards).
* config/: Konfigurationen, z.B. für Umgebungsvariablen und Datenbank-Settings.

# 2. Frontend (/frontend)

## 2.1. Genutzte Frameworks & Bibliotheken:

Angular (v18): Moderne Single-Page-Application mit TypeScript.

RxJS: Für reaktive Datenströme (z.B. State-Management, asynchrone Kommunikation).

Chart.js: Für die grafische Darstellung des Lernerfolgs (Doughnut-Charts).

concurrently: Zum parallelen Starten von Front- und Backend im Dev-Modus.

Karma/Jasmine: Für Unit-Testing.

TypeScript: Strikt typisierte Entwicklung.

## 2.2. Aufbau im Frontend:

Das Frontend basiert auf Angular (TypeScript). Die Komponenten sind nach Best Practices modular aufgebaut, um die Wartbarkeit und Skalierbarkeit der App zu verbessern.

Die drei wichtigsten Hauptverzeichnisse innerhalb von src/app/ sind:

### 2.2.1. core/

Enthält globale, anwendungsweite Services und zentrale Logik, die im gesamten Projekt benötigt werden, zum Beispiel:

* Authentifizierungs-Service (auth.service.ts)
* HTTP-Interceptor für das automatische Anhängen von Tokens an API-Requests
* Datenmodelle (Interfaces für Topics, Flashcards, etc.)
* Guards (z.B. authguard.service.ts)
* Weitere zentrale Dienste (z.B. für Darkmode)

Wird nur einmalig geladen und stellt grundlegende Funktionalität für alle Features bereit.

2.2.2. features/

Enthält die eigentlichen Hauptfunktionen und Seiten der App.

Jedes größere Feature (bspw. Login, Themengebiete, Unterthemen, Lernmodus) ist als eigener Unterordner mit eigener Komponente, Logik, HTML und CSS umgesetzt.

Beispielhafte Unterordner:

* auth/ – Login und Registrierung
* topics/ – Themengebiete
* subtopics/ – Unterthemen
* flashcards-list/ – Liste und Bearbeitung von Karteikarten
* flashcard-preview/ – Lernmodus

Diese Struktur sorgt dafür, dass jede Funktionalität für sich entwickelt, getestet und ggf. erweitert werden kann.

2.2.3. shared/

Hier befinden sich wiederverwendbare UI-Komponenten und Pipes (Hilfsfunktionen).

Beispiele:

* header/ – Die Header-Komponente, die auf jeder Seite genutzt wird
* add-item/ – Komponente zum Hinzufügen von neuen Elementen (Themen, Unterthemen, Karten)
* editable-card/ – Karte für editierbare Inhalte
* search-sort-pipe/ – Pipe für Suchen und Sortieren von Listen

shared wird projektweit genutzt, um Redundanzen zu vermeiden und ein einheitliches Design zu gewährleisten.

# 3. Gemeinsame Tools/Konfiguration

* Docker / Docker Compose:

Wird für die lokale Entwicklungsumgebung und das Setup der Datenbank genutzt. Optional kann das Backend auch containerisiert gestartet werden.

* Setup-Skript:

Zentrales Skript im Root, das alle Abhängigkeiten installiert, Umgebungsvariablen vorbereitet und (optional) alle Services startet.

* Node.js & npm:

Wird für beide Teile benötigt.