# 1. Motivation und Grundentscheidungen

Zu Beginn des Projekts war es uns besonders wichtig, die Modularität der Anwendung sowohl im Frontend als auch im Backend von Anfang an in den Mittelpunkt zu stellen. Schon im ersten Semester legten wir großen Wert darauf, unseren Code so aufzubauen, dass die einzelnen Komponenten und Funktionen klar voneinander getrennt und gut wiederverwendbar sind. Dies erleichtert nicht nur die Zusammenarbeit im Team, sondern sorgt auch für eine bessere Wartbarkeit und Nachvollziehbarkeit des Projekts.

Im zweiten Semester, als der Schwerpunkt auf der Backend-Entwicklung lag, standen wir vor der grundlegenden Entscheidung, wie wir unseren Server und die technische Infrastruktur gestalten möchten. Wir entschieden uns bewusst für den Einsatz von Docker – einerseits, weil wir bereits erste Erfahrungen damit gesammelt hatten, andererseits, weil sich Anwendungen mit Docker am Ende besonders einfach und zuverlässig auf verschiedenen Systemen deployen lassen. Gerade für ein Projekt, das später produktiv auf einem Server laufen soll, ist das ein großer Vorteil.

Im Frontend legten wir großen Wert darauf, moderne Angular-Best-Practices anzuwenden. Dazu zählte insbesondere der Einsatz von Observables für asynchrone Datenflüsse. Das sorgt dafür, dass die Anwendung sehr flexibel auf Änderungen reagieren kann und die Datenhaltung stets aktuell bleibt. Ein weiteres Ziel war es, den Code so übersichtlich und verständlich wie möglich zu gestalten – sowohl durch eine saubere Struktur als auch durch ausführliche Kommentare.

Die Entscheidung für Angular als Framework fiel uns vor allem deshalb leicht, weil wir während unserer Praxisphase bereits praktische Erfahrungen damit sammeln konnten. Zwar ist der Einstieg in Angular nicht ganz einfach, da die Architektur zunächst komplex und ungewohnt wirken kann. Sobald man sich jedoch in das Framework eingearbeitet und die Grundprinzipien verstanden hat, zeigen sich die enormen Vorteile der Modularität und klaren Struktur. Anfangs mag die Struktur verschachtelt und unübersichtlich erscheinen, doch gerade bei größeren Projekten wie diesem bewährt sich die saubere Trennung in Komponenten, Services und Module.

# 2. Funktion: Anzeige, Bearbeitung und Verwaltung der Themen, Unterthemen und Karteikarten

**MatrNR: 4078886**

Im Rahmen des Projekts habe ich die Kernfunktion entwickelt, die es ermöglicht, Themen, Unterthemen (Subtopics) und Karteikarten (Flashcards) in der Anwendung anzuzeigen, zu erstellen, zu bearbeiten und zu löschen. Diese Funktion ist das Herzstück der gesamten Anwendung und bildet die Grundlage für die strukturierte Organisation und effiziente Nutzung der digitalen Karteikarten.

## 2.1 Idee und Zielsetzung

Meine Grundidee dabei war, die Daten nicht einfach nur als „flache Liste“ zu speichern, sondern klar gestaffelt und **hierarchisch** zu strukturieren:

* **Themengebiete** (z.B. „Mathematik“, „Informatik“) bilden die oberste Ebene.
* Jedes Thema enthält beliebig viele **Unterthemen** (z.B. „Algebra“, „Geometrie“).
* Jeder Unterpunkt kann wiederum eine eigene Sammlung von **Karteikarten** mit Frage und Antwort enthalten.

Dadurch lässt sich die Lernumgebung individuell und übersichtlich gestalten. Besonders wichtig war mir, dass die Nutzer ihre Inhalte **einfach verwalten, flexibel anpassen und bei Bedarf löschen** können.

## 2.2 Architektur und technische Umsetzung

## **2.2.1 Frontend:**

Die Umsetzung im Frontend erfolgte **modular nach Angular-Best-Practices**, um Wiederverwendbarkeit, Übersichtlichkeit und Wartbarkeit sicherzustellen. Die Anzeige und Bearbeitung der Daten erfolgt in eigenen Komponenten:

* TopicsComponent für Themen
* SubtopicsComponent für Unterthemen
* FlashcardsListComponent für Karteikarten

Alle Komponenten kommunizieren über einen **zentralen Service (**GetDataService**)**, der sämtliche HTTP-Requests für das Abrufen, Anlegen, Bearbeiten und Löschen von Daten kapselt.  
Die Anzeige und Bearbeitung ist dabei stets **reaktiv**: Änderungen am Datenbestand (wie Hinzufügen, Bearbeiten, Löschen) werden direkt und ohne Neuladen der Seite in der Benutzeroberfläche sichtbar.

**Ergänzende, gemeinsam genutzte Komponenten und Pipes:**

Um den **Funktionsumfang zu erweitern** und das User Interface zu vereinfachen, habe ich verschiedene **wiederverwendbare Komponenten und Pipes** entwickelt, die direkt in die oben genannten Funktionen integriert sind:

* AddItemComponent  
  Ein universell einsetzbares Formular zum Hinzufügen neuer Themen, Unterthemen oder Karteikarten. Durch die Übergabe von Parametern lässt es sich flexibel für jede Ebene verwenden und sorgt für eine einheitliche UX beim Anlegen von Inhalten.
* EditableCardComponent  
  Diese Komponente stellt eine „Karte“ dar, die sowohl angezeigt als auch direkt bearbeitet werden kann. Sie ist die Grundlage für das intuitive Inline-Editing aller Objekte (egal ob Thema, Subtopic oder Flashcard).
* SearchSortPipe  
  Eine eigens entwickelte Angular-Pipe, die es ermöglicht, alle Listen (Themen, Unterthemen, Karten) komfortabel zu durchsuchen und zu sortieren. Sie unterstützt verschiedene Suchfelder und Sortierkriterien (z.B. nach Name, Datum, Lernfortschritt).
* HeaderComponent  
  Der Header sorgt für die einheitliche Navigation, zeigt den aktuellen Status (z.B. Login), bietet Zugriff auf die Logout-Funktion, Navigation und den (optionalen) Dark Mode. Er ist in jeder Komponente eingebunden und somit ebenfalls Teil der Funktionsumsetzung.

All diese Komponenten liegen im shared**-Verzeichnis** und werden in den Feature-Komponenten (features) gezielt eingebunden. Dadurch konnte ich den Code **deutlich modularer, übersichtlicher und wartbarer gestalten**.

## **2.2.2. Backend:**

Das Backend ist in Node.js (TypeScript) mit Express und TypeORM realisiert. Die **Datenbankstruktur** bildet die Hierarchie der Themen, Unterthemen und Karteikarten **1:1** ab (Relationen).  
Es gibt für jede Ebene eigene Endpunkte (REST-API), z.B.:

* /api/topics – Themen verwalten
* /api/topics/:topicId/subtopics – Unterthemen zu einem Thema verwalten
* /api/topics/:topicId/subtopics/:subtopicId/flashcards – Karteikarten eines Unterthemas verwalten

Die API ist so konzipiert, dass alle **CRUD-Operationen (Create, Read, Update, Delete)** sowohl für Themen, Unterthemen als auch für Karteikarten sauber abgebildet und abgesichert sind.

## 2.3 Wichtige Design-Entscheidungen

* **Hierarchische Struktur:**  
  Die Unterteilung in Themen, Unterthemen und Karteikarten war von Anfang an gesetzt, um das Lernen und Verwalten großer Mengen an Karteikarten übersichtlich zu gestalten.
* **Modularität:**  
  Ich habe Wert darauf gelegt, dass jede Funktion (z.B. Hinzufügen, Bearbeiten, Löschen) in einer **eigenen kleinen Datei bzw. Komponente** gekapselt ist, statt große, schwer wartbare Dateien zu haben. Die shared-Komponenten sind überall wiederverwendbar.
* **Direkte, reaktive Anzeige:**  
  Über **Observables** werden Änderungen (z.B. neue Karte angelegt oder gelöscht) direkt in der UI sichtbar, ohne dass man die Seite neu laden muss.
* **Klare API-Struktur:**  
  Die Endpunkte spiegeln die Datenstruktur und die Besitzverhältnisse exakt wider – z.B. kann man keine Karteikarte ohne Bezug zu einem Subtopic anlegen.
* **Gute Nutzerführung:**  
  Durch Komponenten wie HeaderComponent, AddItemComponent und direkte Such-/Sortierfunktionen ist die Bedienung intuitiv und effizient.

## 2.4 Zusammenspiel von Frontend und Backend

Sobald ein Nutzer eine Aktion im Frontend ausführt (z.B. ein Thema anlegt, eine Karte bearbeitet oder eine Unterthema löscht), wird ein passender **API-Call** abgesetzt.  
Das Backend verarbeitet die Anfrage, nimmt die Änderungen in der Datenbank vor und gibt das aktualisierte Objekt oder einen Status zurück.  
Die Daten werden anschließend direkt im Frontend angezeigt bzw. aktualisiert. Dank **Angulars reaktiver Programmierung** werden auch parallele Änderungen nahtlos sichtbar.

## 2.5 Herausforderungen und Lösungen

Der **Wechsel vom frontend-basierten Ansatz** (im ersten Semester) auf die echte Backend-Anbindung (im zweiten Semester) war herausfordernd:

* **Anpassung der Datenmodelle:**  
  Die Datenmodelle und Typen mussten grundlegend angepasst werden, weil das Backend andere Datenstrukturen zurückliefert als das vorherige JSON-Server-Mockup.
* **Überarbeitung der Services und Komponenten:**  
  Die Services und Komponenten mussten überarbeitet werden, um die neuen Datentypen korrekt zu verarbeiten und darzustellen.
* **Robustes Error-Handling und Nachladen von Daten:**  
  Das Error-Handling wurde verbessert, um die UI robust zu halten, und die Nachlade-Logik wurde optimiert, sodass die Nutzer immer die aktuellen Daten sehen.
* **Einbindung von wiederverwendbaren Komponenten:**  
  Die Entwicklung von AddItemComponent, EditableCardComponent und SearchSortPipe hat den Code deutlich modularer gemacht und die Wiederverwendbarkeit sowie Konsistenz im gesamten Projekt erhöht.

# 3. Login und Registrierung

**MatrNR: 4078886**

Ein zentrales Feature unseres Karteikarten-Projekts ist die Möglichkeit für Nutzer, sich zu registrieren, einzuloggen und ihre persönlichen Daten geschützt zu verwalten. Diese Funktion wurde in Zusammenarbeit entwickelt und zeigt die Verknüpfung von Frontend- und Backend-Entwicklung.

### 3.1 Ausgangslage und erste Iteration

Im ersten Semester wurde zunächst eine Mock-Authentifizierung direkt im Frontend mit einem lokalen JSON-Server von der Matr.NR.:8742615 umgesetzt. Diese Lösung ermöglichte einen schnellen Start und einfache Tests, war aber für eine reale Nutzung weder sicher noch skalierbar. Es gab keine echte Nutzerverwaltung, keine Passworthashes und keine geschützten Sessions.

### 3.2 Ziel: Sichere, skalierbare Authentifizierung

Mit Beginn der Backend-Implementierung im zweiten Semester habe ich die Authentifizierung von Grund auf neu konzipiert – mit Fokus auf Sicherheit, Skalierbarkeit und modernen Best Practices. Die wichtigsten Ziele dabei waren:

* **Echte Nutzerverwaltung** mit Persistenz in einer echten MySQL-Datenbank (via TypeORM)
* **Sicheres Passwort-Handling**: Passwörter werden niemals im Klartext gespeichert, sondern ausschließlich mit einer modernen Hash-Funktion wie **Argon2** verschlüsselt.
* **Session-Management ohne klassische Cookies**, sondern über **stateless JWT-Token** (JSON Web Token) für mehr Sicherheit und bessere Skalierbarkeit, z.B. auch für spätere mobile Clients.
* **Zentrale Validierung aller Eingaben** mit Libraries wie Zod, um Angriffsvektoren wie SQL-Injection oder schwache Passwörter zu verhindern.
* **Trennung von Logik, Datenhaltung und Präsentation**, um die Architektur wartbar und übersichtlich zu halten.

### 3.3 Technische Umsetzung und Ablauf

## **3.3.1 Frontend (Angular):**

* Die Matr.NR.:8742615 entwickelte die Benutzeroberfläche, UX und Komponenten für Login und Registrierung (**bis Commit 94e5fa9**), wobei Wert auf eine intuitive Benutzerführung und Responsive Design gelegt wurde.
* Nach Abschluss des Backends habe ich die Authentifizierungslogik im Frontend integriert:
  + **AuthService**: Ein Service, der Anmeldedaten und Registrierungsdaten an die REST-API sendet und das erhaltene JWT-Token sicher im LocalStorage speichert.
  + **HTTP-Interceptor**: Dieser fügt das JWT-Token automatisch bei allen geschützten Requests als „Bearer Token“ in den HTTP-Header ein, sodass der User im eingeloggten Zustand bleibt und keine sensiblen Daten ungeschützt übertragen werden.
  + **AuthGuard**: Über einen Route Guard werden nicht-eingeloggte Nutzer konsequent von geschützten Routen ferngehalten.
  + **Zentrale Logout-Logik**, die beim Ausloggen alle Authentifizierungsdaten sicher entfernt und den User auf die Login-Seite zurückführt.

**Backend (Node.js/Express, TypeORM, Argon2):**

* **Register-Endpoint**: Nimmt neue Nutzerdaten entgegen, prüft sie serverseitig mit Zod (u.a. Mindestlänge, gültiges Email-Format), hasht das Passwort mit Argon2 und legt den neuen Nutzer sicher in der Datenbank ab.
* **Login-Endpoint**: Prüft die Kombination aus Username/Email und Passwort, verifiziert den Passwort-Hash und gibt im Erfolgsfall ein frisch signiertes JWT-Token aus.
* **JWT-Token-Strategie**:  
  Der JWT-Token wird bei jeder Authentifizierung neu erzeugt, enthält alle für die Session notwendigen Claims (z.B. User-ID, ggf. Gültigkeitsdauer) und ist mit einem geheimen Key signiert.  
  Die Verwendung von JWT-Token war mir wichtig, da dies eine **stateless** und dadurch sehr skalierbare Lösung ist. Im Gegensatz zu klassischen Session-Cookies muss der Server keinen Zustand speichern; alles Wichtige steckt im Token.  
  Das schützt vor Cross-Site Request Forgery (CSRF), ist mit modernen Frontends (z.B. Mobile Apps) einfach kompatibel und ermöglicht bei Bedarf eine einfache Erweiterung (z.B. mit Rollen).
* **Middleware für den Zugriffsschutz**:  
  Bei jedem API-Aufruf prüft eine Middleware automatisch die Gültigkeit des JWT-Tokens, bevor Daten ausgeliefert werden. So ist sichergestellt, dass wirklich nur authentifizierte Nutzer Zugriff auf ihre persönlichen Daten oder Karteikarten bekommen.
* **Sichere Fehlerbehandlung**:  
  Alle Fehler werden klar behandelt und niemals werden sensible Informationen (wie Passworthashes, interne Stacktraces o.Ä.) an den Client übermittelt.  
  Alle Anmelde- und Registrierungsversuche werden geloggt, um z.B. Brute-Force-Angriffe erkennen zu können.

## Wichtige Überlegungen beim Backend-Aufbau

Beim Aufbau des Backends war mir wichtig, **Sicherheit, Skalierbarkeit und Verständlichkeit** zu kombinieren:

* Die Entscheidung für **Argon2** als Passwort-Hashing-Algorithmus fiel, weil es aktuell als einer der sichersten Algorithmen gilt (im Vergleich zu älteren wie bcrypt).
* Die Token-basierte Authentifizierung mit JWT sorgt für ein einfaches, aber dennoch sicheres Management der Sessions – gerade im Kontext moderner Web-Architekturen und Microservices.
* Durch **klare Trennung von Routen, Controllern, Services und Validierung** bleibt die Codebasis übersichtlich und ist gut erweiterbar.
* Die gesamte Infrastruktur wurde von Beginn an so geplant, dass sie per **Docker** einfach und konsistent auf verschiedenen Systemen (Entwicklung, Test, Produktion) ausgerollt werden kann – auch das spielt bei der Sicherheit eine Rolle, da stets eine identische Umgebung sichergestellt ist.
* Für die Zukunft ist die API so ausgelegt, dass sie problemlos um weitere Sicherheitsfeatures (z.B. 2-Faktor-Authentifizierung, Passwort-Reset, Rollen) ergänzt werden kann.

# Entwicklung der Flashcard-Preview (Frontend)

(Martikel-Nr: 6418650)

Ich war verantwortlich für die vollständige Implementierung des Frontend-Bereichs, der die Karteikarten-Vorschau ermöglicht. Dabei habe ich eine komponentenbasierte Architektur in Angular genutzt, um folgende Funktionalitäten umzusetzen:

* Dynamisches Laden der Karteikarten: Die Flashcards werden über REST-API vom Backend geladen und im Interface angezeigt.
* Modus-Auswahl: Nutzer können zwischen einem Endlos-Modus („infinite“) und einem begrenzten Modus („limited“) wählen. Im begrenzten Modus werden nur Karten angezeigt, die noch nicht vollständig gelernt wurden.
* Interaktive Karteikarten: Karten können durch Klick umgedreht werden, um Vorder- und Rückseite anzuzeigen. Die Animation basiert auf CSS 3DTransformationen.
* Navigation: Die Nutzer können zwischen den Karten vor- und zurückspringen, unterstützt durch visuelle Pfeile.
* Lernfortschritt: Die Fortschrittsanzeige zeigt, wie oft eine Karte schon richtig beantwortet wurde, und wird sowohl in der UI als auch im Backend synchron gehalten.
* Endscreen mit Auswertung: Nach Abschluss des Lernens im begrenzten Modus erscheint ein Endscreen mit einer Diagramm-Auswertung, die den Anteil richtiger und falscher Antworten visualisiert. Die Darstellung erfolgt mit Chart.js.

## Entwicklung des ToDo-Features (Frontend & Backend)

(Martikel-Nr: 6418650)

Ich habe außerdem das ToDo-Feature vollständig realisiert, inklusive Frontend und

Backend:

* Erstellung der Datenbank-Entity „Todo“ mit TypeORM, inklusive Feldern für Text, Status (erledigt/nicht erledigt) und Nutzerzuordnung.
* Implementierung eines Service mit CRUD-Funktionalitäten: ToDos anlegen, abrufen, Status toggeln und löschen.

Frontend:

* Umsetzung der UI mit klaren Checkboxen für den Status, Lösch-Buttons und visuellen Effekten (Hover, Klick).
* Anbindung an die Backend-API, um Änderungen live zu synchronisieren.
* Implementierung von Nutzerfeedback, z.B. visuelle Rückmeldungen beim Klicken auf Checkbox oder Löschen.
* Bugfixes und Verbesserungen

Neben meinen Hauptaufgaben habe ich auch an anderen Bereichen des Projekts mitgearbeitet und verschiedene Fehler behoben, um die Stabilität und Benutzerfreundlichkeit der Webseite zu erhöhen. Dazu gehörten:

* Korrekturen bei der Synchronisation von Lernfortschritten
* Verbesserungen bei der Performance der Karten-Animation
* Optimierungen der Datenbankabfragen für schnellere Ladezeiten

## Zusammenfassung

Meine Arbeit konzentrierte sich auf die Kernfeatures der Lernplattform: Die FlashcardPreview, die essenziell für das Lernerlebnis ist, und das ToDo-Feature, das den Nutzern hilft, ihre Lernaufgaben strukturiert zu verwalten. Durch die Kombination von AngularFrontend, Node.js-Backend und einer MySQL-Datenbank konnte ich ein stabiles, skalierbares und benutzerfreundliches System entwickeln. Die enge Zusammenarbeit im Team und die Nutzung von modernen Tools wie Docker ermöglichten einen effizienten Entwicklungsprozess