Einsatz von GenAI-Technologien zur Bewertung von Antworten

Studienarbeit

des Studiengangs Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Len Vejsada

May 5, 2025

Matrikelnummer	9447853
Kurs	TINF22B6
Ausbildungsfirma	Kardex, Bellheim
Betreuer	Simon Franke

Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 9. 2015) Ich versichere hiermit, dass ich meinen Praxisbericht mit dem Thema: „Einsatz von GenAI-Technologien zur Bewertung von Antworten“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Lustadt, 12.12.2024	gez. Len Vejsada
Ort, Datum	Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

C# C Sharp

HTML Hypertext Markup Language

CSS Cascading Style Sheets

SCSS Sassy Cascading Style Sheets

SQL Structured Query Language

JS JavaScript

.NET Domain Network

ASP.NET Active Server Pages .NET

EF Entity Framework

DDD Domain Driven Design

ORM Object-Relational Mapping

API Application Programming Interface

SPA Single-Page Application

MPA Multi-Page Application

UI User Interface

DI Dependency Injection

TS TypeScript

CQRS Command Query Responsibility Segregation

IoC Inversion of Control

SOA Service-Oriented Architecture

CLI Command Line Interface

TFVC Team Foundation Version Control

CI Continuous Integration

CD Continuous Deployment

YAML Yet Another markup Language

REST Representational State Transfer

1 Einleitung

1.1 Ziele des Dokuments

1.2 Anlass und Motivation des Projekts

1.3 Aufbau der Arbeit

2 Hintergrund und Kontext

2.1 Technischer Hintergrund

2.2 Generative KI-Technologien: Definition und Bedeutung

2.3 Vergleich von KI- und klassischen Algorithmen in der Bewertung

3 Projektbeschreibung

3.1 Projektziele

3.2 Überblick über das Minimum Viable Product (MVP)

3.3 Abgrenzung und Einschränkungen

4 Technologien und Architektur

Die technische Umsetzung des Projekts basiert auf einem modernen Fullstack-Ansatz, der eine klare Trennung zwischen Frontend, Backend, Datenhaltung und API-Kommunikation vorsieht. Die Architektur des entwickelten Systems folgt einem serviceorientierten und modularen Design, das sich durch eine klare Trennung von Präsentation, Geschäftslogik, Datenhaltung und externen Schnittstellen auszeichnet. Im Folgenden werden die eingesetzten Technologien systematisch beschrieben.

4.1 Frontend

Die Benutzeroberfläche wurde mit Angular, einem auf TypeScript basierenden Frontend-Framework von Google, realisiert. Angular ist besonders geeignet für die Entwicklung von Single-Page Applications (SPA), bei denen der Inhalt dynamisch aktualisiert wird, ohne die gesamte Seite neu zu laden. Die Vorteile von Angular liegen in seiner klaren Struktur, der engen Verzahnung mit TypeScript sowie der komponentenbasierten Architektur. Die Wahl fiel bewusst auf Angular, da es eine solide Basis für strukturierte Projekte mit komplexer Formularlogik bietet. Der Einsatz von Reactive Forms ermöglicht eine präzise Validierung und Zustandsverwaltung der Benutzereingaben.

- **TypeScript** als Basissprache für alle Komponenten: Durch statische Typisierung erhöht sich die Fehlervermeidung bereits zur Entwicklungszeit.
- **Reactive Forms** für die Verwaltung komplexer Formulare und Zustände: Dieses Formularmodell ermöglicht es, Benutzereingaben programmatisch zu validieren und dynamisch zu reagieren.
- **Routing mit** `RouterModule` zur Navigation zwischen Komponenten wie *Login*, *Fragenverwaltung*, *Prüfungsbearbeitung* und *Ergebnisse*.
- **SCSS (Sassy CSS)** als CSS-Präprozessor: Erlaubt die Nutzung von Variablen, Verschachtelungen und Mixins, was zu einer besseren Strukturierung und Wiederverwendbarkeit des Stylesheets führt.
- **Internationalisierung mit** `ngx-translate`: Die App ist mehrsprachig konzipiert und kann problemlos um weitere Sprachpakete erweitert werden.
- **HttpClient**: Für den asynchronen Datenaustausch mit dem Backend über RESTful APIs.

Das Frontend der Anwendung ist responsiv gestaltet und stellt eine benutzerfreundliche Oberfläche bereit. Als Basissprache kommt TypeScript zum Einsatz, was durch statische Typisierung eine frühzeitige Fehlervermeidung während der Entwicklung ermöglicht. Für die Verwaltung komplexer Formulare und Zustände wird das *Reactive Forms*-Modell ver-

wendet. Dieses erlaubt es, Benutzereingaben programmatisch zu validieren und dynamisch auf Änderungen zu reagieren. Die Navigation innerhalb der Anwendung erfolgt über das *RouterModule*, welches den Wechsel zwischen Komponenten wie Login, Fragenverwaltung, Prüfungsbearbeitung und Ergebnisanzeige unterstützt.

Zur Strukturierung und besseren Wartbarkeit der Stylesheets wird der CSS-Präprozessor SCSS verwendet. Dieser ermöglicht unter anderem die Nutzung von Variablen, Verschachtelungen und Mixins. Die Internationalisierung der Anwendung erfolgt mit Hilfe von `ngx-translate`, wodurch eine mehrsprachige Nutzung realisiert und einfach erweitert werden kann. Der Datenaustausch mit dem Backend erfolgt asynchron über RESTful APIs mittels *HttpClient*.

Das Frontend der Anwendung ist responsiv gestaltet und stellt eine benutzerfreundliche Oberfläche zur Verfügung. Entwickelt wurde es mit TypeScript als Basissprache, wodurch durch statische Typisierung bereits zur Entwicklungszeit eine höhere Fehlervermeidung gewährleistet wird. Die Verwaltung komplexer Formulare erfolgt mithilfe des *Reactive Forms*-Modells, das eine programmgesteuerte Validierung von Eingaben sowie eine dynamische Reaktion auf Zustandsänderungen ermöglicht. Die Navigation zwischen den einzelnen Komponenten – etwa Login, Fragenverwaltung, Prüfungsbearbeitung oder Ergebnisanzeige – wird durch das *RouterModule* realisiert.

Für die Gestaltung des Layouts kommt der CSS-Präprozessor SCSS zum Einsatz, der unter anderem die Nutzung von Variablen, Verschachtelungen und Mixins erlaubt und somit zu einer besseren Strukturierung und Wiederverwendbarkeit des Stylesheets beiträgt. Die Anwendung ist mehrsprachig konzipiert und nutzt zur Internationalisierung die Bibliothek `ngx-translate`, wodurch eine einfache Erweiterung um zusätzliche Sprachpakete möglich ist. Der Datenaustausch mit dem Backend erfolgt über den *HttpClient* asynchron über RESTful APIs.

Funktional bietet das Frontend unter anderem ein Benutzerprofil, das Zugriff auf persönliche Einstellungen sowie die Ergebnisse abgeschlossener Prüfungen ermöglicht. Nutzerinnen und Nutzer können eigenständig Fragen erstellen, bearbeiten und löschen. Diese Fragen können im Anschluss Prüfungen zugewiesen werden, wobei sich die Prüfungen zusätzlich um ein Zeitlimit konfigurieren lassen. Während der Durchführung einer Prüfung stehen Funktionen wie eine Fortschrittsanzeige, die Navigation zwischen einzelnen Fragen sowie ein Echtzeit-Timer zur Verfügung. Nach Abschluss wird die Abgabe automatisiert an eine externe Schnittstelle gesendet, die auf der ChatGPT-API basiert. Diese übernimmt die KI-gestützte Korrektur der Prüfung und liefert eine entsprechende Bewertung samt Feedback zurück.

4.2 Backend

Das serverseitige System wurde mit ASP.NET Core 7 entwickelt – einem modernen, plattformunabhängigen Webframework von Microsoft, das sich durch hohe Performance, Skalierbarkeit und Flexibilität auszeichnet. Das Backend folgt dem Prinzip einer klar strukturierten Schichtenarchitektur. In der obersten Ebene befindet sich die Controller-Schicht, die für das Routing und die Entgegennahme von HTTP-Anfragen verantwortlich ist. Sie übernimmt außerdem die Autorisierung und delegiert fachliche Aufgaben an die zugrunde liegende Service-Schicht. Diese Schicht implementiert die Geschäftslogik der Anwendung, etwa die Validierung von Prüfungen, das Anlegen von Versuchen oder die Übergabe von Abgaben an das Bewertungssystem.

Die Repository-Schicht bildet die unterste Ebene der Architektur und ist für den Datenzugriff zuständig. Sie nutzt das Object-Relational-Mapping-Framework Entity Framework Core (EF Core), um eine effiziente und typensichere Interaktion mit der Datenbank zu gewährleisten.

Zur weiteren Unterstützung einer sauberen Softwarearchitektur kommen zusätzliche Konzepte und Technologien zum Einsatz. Die native Unterstützung von Dependency Injection (DI) in ASP.NET Core ermöglicht eine klare Trennung zwischen Implementierung und Abhängigkeiten durch die Verwendung von Interfaces. Sämtliche Datenzugriffe und API-Aufrufe sind asynchron realisiert, wodurch eine nicht-blockierende Verarbeitung durch das `async/await`-Paradigma sichergestellt wird. Die Authentifizierung erfolgt mittels JSON Web Tokens (JWT), die nach dem Login generiert, clientseitig gespeichert und bei jeder weiteren Anfrage im HTTP-Header mitgesendet werden.

Die bereitgestellte REST-API umfasst Endpunkte für die Benutzerregistrierung und -anmeldung, die Verwaltung von Fragen und Prüfungen, das Starten und Abschließen von Prüfungsversuchen sowie die Rückgabe von Ergebnissen, einschließlich einer automatisierten Bewertung durch die Integration eines generativen KI-Moduls auf Basis von ChatGPT.

4.3 Datenbank

4.4 API-Kommunikation

4.5 Teststrategie

5 Implementierung

5.1 Implementierung der Fragetypen

5.2 Integration der generativen KI-Bewertung

5.3 Datenbankstruktur und Datenmodelle

5.4 Entwicklung der Benutzeroberfläche

6 Evaluierung

6.1 Vergleich von GenAI-Lösungen

6.2 Leistung und Genauigkeit der Bewertung

6.3 Kostenanalyse der verschiedenen KI-APIs

7 Ergebnisse und Diskussion

7.1 Zusammenfassung der Implementierungsergebnisse

7.2 Herausforderungen bei der Entwicklung

7.3 Potenzial für zukünftige Erweiterungen

8 Zukünftige Arbeiten und Erweiterungsmöglichkeiten

8.1 Adaptive Lernpfade

8.2 Integration von Sprach- und Bilderkennung

8.3 Mehrsprachigkeit

9 Fazit

9.1 Erfüllung der Projektziele

9.2 Bedeutung der Ergebnisse

9.3 Ausblick