GymInf Thesis: Webapp für Matheaufgaben

Oliver de Capitani, Patrick Weber

4. April 2024

Inhaltsverzeichnis

1 Projektskizze

1.1 Projektplanung und Design

Anforderungsanalyse

• Kernfunktionen: Die Applikation ermöglicht es Benutzern, Mathe-Aufgaben zu erstellen, durchsuchen, kommentieren und zu verwalten. Ein Warenkorb-System ermöglicht das Sammeln und Erstellen von Aufgabenblättern.

• User Stories und Use Cases: Diese helfen, die Bedürfnisse und Erwartungen der Endbenutzer zu identifizieren. Sie dienen als Grundlage für das Design und die Entwicklung der Applikation.

Datenmodellierung

• Entwurf der Datenbank-Schemata: Die Datenstruktur für Aufgaben, Kommentare, Benutzer und Aufgabenkategorien wird definiert. Wichtige Beziehungen zwischen den Entitäten werden festgelegt, wie z.B. die Zuordnung von Aufgaben zu Kategorien und von Aufgaben zu Kommentaren.

Architekturdesign

• Backend-Architektur: Entscheidung zwischen REST API oder GraphQL basierend auf den Anforderungen und Vorzügen beider Technologien.

• Frontend-Architektur: Planung der Komponentenhierarchie und des Zustandsmanagements, um eine effiziente und wartbare Codebasis zu gewährleisten.

Technologische Auswahl

• Bestätigung der Technologien: Preact für das Frontend, Express für das Backend und Prisma für die Datenbankinteraktion werden als Haupttechnologien ausgewählt. Die Notwendigkeit weiterer Bibliotheken (z.B. für Authentifizierung) wird überprüft.

1.2 Backend-Entwicklung

Einrichten der Entwicklungsumgebung

• Einrichtung von Node.js, Express und Prisma: Konfiguration der grundlegenden Entwicklungsumgebung, um mit der Entwicklung beginnen zu können.

API-Entwicklung

• CRUD-Operationen: Implementierung der Grundoperationen für die Interaktion mit Aufgaben und Kommentaren.

• Authentifizierung und Autorisierung: Sicherstellung, dass nur autorisierte Benutzer bestimmte Aktionen durchführen können.

• Entwickle eine Suchfunktion, um Aufgaben nach Stichworten oder Kategorien zu durchsuchen.

Datenbankintegration

• Prisma für das Datenbankschema: Nutzung von Prisma für das Datenbank-Management, einschließlich Schema-Definition und Migrationen.

• Teste die Datenbankoperationen über Prisma.

1.3 Frontend-Entwicklung

Einrichten der Preact-Anwendung

• Projektinitialisierung: Erstellung einer neuen Preact-Anwendung und Einrichtung des Routings.

UI-Entwicklung

• UI-Komponenten: Entwicklung der Benutzeroberfläche, einschließlich Komponenten für die Aufgabenverwaltung und das Kommentarsystem.

• Implementiere Formulare für das Erstellen und Bearbeiten von Aufgaben.

Integration mit dem Backend

• Datenabfrage und -manipulation: Implementierung der Logik für die Kommunikation mit dem Backend, um Daten zu fetchen, zu erstellen und zu aktualisieren.

1.4 Testing und Qualitätssicherung

• Backend- und Frontend-Tests: Durchführung von Tests auf beiden Ebenen, um die Funktionalität und Zuverlässigkeit der Anwendung zu gewährleisten.

• Benutzerakzeptanztests: Einbeziehung der Endbenutzer in den Testprozess, um Feedback zu sammeln und die Benutzerfreundlichkeit zu optimieren.

1.5 Deployment und Inbetriebnahme

• Vorbereitung und Auswahl einer Hosting-Plattform: Konfiguration des Projekts für die Produktionsumgebung und Auswahl einer geeigneten Plattform für das Hosting.

• Durchführung des Deployments: Veröffentlichung der Anwendung und Überprüfung der Funktionalität im Produktionsumfeld.

1.6 Dokumentation und Übergabe

• Erstellung der Dokumentation: Dokumentation des Codes, der Architektur und der Benutzung der Anwendung.

• Abschlusspräsentation: Präsentation des Entwicklungsprozesses und der fertigen Anwendung.

2 Dokumentation

2.1 Anforderungsanalyse

Die Applikation zielt darauf ab, Mathematiklehrern eine effiziente Möglichkeit zu bieten, Aufgaben für Unterrichtsblätter zu finden, sowie eine umfassende Sammlung von Aufgaben auszuwählen und zu organisieren. Kernfunktionen umfassen:

• Aufgaben erstellen: Lehrer können neue Mathematikaufgaben zur Datenbank hinzufügen.

• Aufgaben durchsuchen: Lehrer können vorhandene Aufgaben nach verschiedenen Kriterien durchsuchen.

• Aufgaben kommentieren: Nutzer können Kommentare zu Aufgaben hinterlassen, um Feedback oder Lösungsvorschläge zu geben.

• Aufgaben verwalten: Lehrer können ihre erstellten Aufgaben bearbeiten oder löschen.

• Warenkorb für Aufgaben: Lehrer können ausgewählte Aufgaben in einem Warenkorb sammeln, um sie später in ein Aufgabenblatt zu integrieren.

User Story:

Als Mathematiklehrer möchte ich eine umfassende Sammlung von Mathematikaufgaben durchsuchen und auswählen können, um effizient maßgeschneiderte Aufgabenblätter für meine Klassen zusammenzustellen.

Use Case:

• Titel: Durchsuchen von Mathematikaufgaben

• Akteure: Mathematiklehrer

• Trigger: Lehrer möchte spezifische Aufgaben für ein Aufgabenblatt finden.

• Vorbedingungen: Lehrer ist angemeldet und navigiert zur Aufgabensuche.

Hauptszenario:

1. Lehrer gibt Suchbegriffe in das Suchfeld ein.

2. System zeigt eine Liste von relevanten Aufgaben an.

3. Lehrer wählt gewünschte Aufgaben für Details aus.

4. Lehrer fügt Aufgaben zum Warenkorb hinzu.

5. Lehrer erstellt aus den Aufgaben im Warenkorb ein Aufgabenblatt.

Nachbedingungen:

Lehrer erhält ein auf die Auswahl zugeschnittenes Aufgabenblatt.

2.2 Datenmodellierung

Die Applikation nutzt ein relationales Datenmodell mit folgenden Hauptentitäten:

Benutzer (Users)

Attribute: BenutzerID, Name, Email, Passwort, Rolle

Beziehungen: 1:n zu Aufgaben, 1:n zu Kommentaren

Aufgaben (Exercises)

Attribute: AufgabenID, Titel, Inhalt, Zusammenfassung, Lösung, Erstellungs-

datum, Update-Datum, AutorID

Beziehungen: n:1 zu Benutzer, n:1 zu Kategorien, n:1 zu Subkategorien

Kategorien (Categories)

Attribute: KategorieID, Name, Beschreibung

Beziehungen: 1:n zu Aufgaben, 1:n zu Subkategorien

Subkategorien (Subcategories)

Attribute: SubkategorieID, Name, Beschreibung, KategorieID

Beziehungen: n:1 zu Kategorie, 1:n zu Aufgaben

Kommentare (Comments)

Attribute: KommentarID, Inhalt, Erstellungsdatum, BenutzerID, AufgabenID

Beziehungen: n:1 zu Benutzer, n:1 zu Aufgaben

Die Datenbankstruktur unterstützt umfassende Interaktionen zwischen Benutzern, ihren Aufgaben und Kommentaren sowie die Kategorisierung von Aufgaben.

2.3 Architekturdesign

Backend-Architektur:

Entscheidung für eine REST API, basierend auf ihrer Einfachheit, Flexibilität und weit verbreiteten Akzeptanz. REST ermöglicht es, mit standardbasierten HTTP-Methoden Ressourcen zu manipulieren, was die Entwicklung

vereinfacht.

Frontend-Architektur:

Die Komponentenhierarchie orientiert sich an einer klaren Trennung zwischen UI-Komponenten (atomar und molekular) und Seitenkomponenten, die den Anwendungsfluss darstellen. Zustandsmanagement wird durch lokale Zustände und ggf. globalen Zustand via Context API oder Zustandsmanagement-Bibliotheken gehandhabt, um einen reaktiven Datenfluss zu gewährleisten.

Zustandsmanagement

Das Zustandsmanagement ist entscheidend für die Verwaltung von Daten und UI-Zuständen (wie Benutzereingaben, geladene Daten, UI-Modi). Preact bietet useState und useEffect Hooks für das lokale Zustandsmanagement in Komponenten. Für komplexere Anwendungen, die einen globalen Zustand benötigen (der über viele Komponenten hinweg geteilt wird), kannst du Bibliotheken wie Zustand oder Context API verwenden.

Zu berücksichtigen:

• Lokaler vs. globaler Zustand: Entscheide, welche Daten lokal in einer Komponente gehalten werden können und welche im gesamten App-Kontext verfügbar sein müssen.

• Datenfluss: Plane, wie Daten durch die Anwendung fließen, z.B. wie globale Daten geladen, modifiziert und zwischen Komponenten geteilt werden.

2.4 Komponentenübersicht

• Aufgabendetails (Exercise Details): Eine Detailansicht (AufgDetails) für jede Aufgabe, die Informationen wie Aufgabentext, Lösung, Kommentare und Kategoriezugehörigkeit enthält. Benutzer können Aufgaben kommentieren und in den Warenkorb legen.

• Suchergebnisse (Search Results): Komponenten für die Anzeige von Suchergebnissen, einschließlich Suchkarten (SearchCard), die einzelne Aufgaben darstellen, und einer Listenansicht (CardListSearch), die eine Sammlung von Suchkarten enthält. Ein spezieller Warenkorb für die Suche (SearchKorb) sammelt gesuchte Aufgaben für spätere Aktionen.

• Warenkorb (Warenkorb): Beinhaltet Komponenten zum Verwalten der ausgewählten Aufgaben, wie WarenCard für einzelne Aufgaben im Warenkorb und CardListWaren für die Gesamtansicht des Warenkorbs. Der WarenKorb selbst ist die Hauptkomponente, die den Nutzern die Interaktion mit dem Warenkorb ermöglicht, einschließlich der Überprüfung und Finalisierung ihrer Auswahl.

• Aufgaben hinzufügen (Add Exercises): Die ExForm Komponente ermöglicht das Erstellen neuer Aufgaben. Sie umfasst Formularfelder für die Eingabe von Titel, Inhalt, Lösung und optionalen Kategorien. Über eine API-Anfrage (askServer(/api/ex", "POST", ex)) werden die Daten an das Backend gesendet und die Aufgabe wird der Datenbank hinzugefügt.

• Aufgaben finden (Find Exercises): Dieser Bereich umfasst mehrere Komponenten, die unterschiedliche Methoden zur Aufgabensuche anbieten, darunter FindExByCategory für kategoriebasierte Suche, FindExByIdFromServer für die direkte Abfrage einer Aufgabe über ihre ID und FindExBySearchText für

textbasierte Suchanfragen.

• Benutzerregistrierung und -anmeldung (User Registration/Login): Eine Form-Komponente unterstützt die Registrierung neuer Benutzer und die Anmeldung bestehender Benutzer durch Übermittlung ihrer Anmeldeinformationen an das Backend (askServer(/api/user", "POST", user)). Authentifizierungsdaten werden überprüft, um Zugang zu geschützten Bereichen der Anwendung zu gewähren.

• Navigation (Navigation): Die Menu-Komponente stellt die Hauptnavigation der Anwendung bereit und ermöglicht den Zugriff auf die verschiedenen Seiten und Funktionen der App, darunter die Startseite (/), das Hinzufügen von Aufgaben (/add), das Finden von Aufgaben (/find), den Warenkorb (/warenkorb) und die Detailansicht einer spezifischen Aufgabe (/:id).

• Seiten (Pages): Definiert die Struktur der Anwendung und beinhaltet Seitenkomponenten wie User für die Benutzerregistrierung und -anmeldung, AddExercise für das Hinzufügen neuer Aufgaben, FindExercise für die Aufgabensuche, Letzte Suchresulate für das Anzeigen der zuletzt gesuchten Aufgaben und WarenKorb für die Ansicht und Verwaltung des Warenkorbs.

2.5 Technologische Auswahl

2.6 Backend-Entwicklung

2.6.1 CRUD- und HTTP-Operationen

CRUD steht für Create, Read, Update, Delete und bezieht sich auf die vier grundlegenden Operationen, die in vielen Anwendungen zur Interaktion mit Datenbanken oder Datenspeichern verwendet werden. Die Implementierung von CRUD-Operationen ermöglicht es Benutzern (oder Systemen), Daten zu erstellen, abzurufen (lesen), zu aktualisieren und zu löschen.

In einer Webanwendung werden CRUD-Operationen typischerweise über das Backend realisiert, wobei das Frontend (die Benutzeroberfläche) Anfragen an das Backend sendet, um Daten zu erstellen, abzurufen, zu aktualisieren oder zu löschen. Das Backend interagiert dann mit der Datenbank, um die angeforderten Aktionen durchzuführen und das Ergebnis (z.B. die abgerufenen Daten oder eine Bestätigung der Aktion) an das Frontend zurückzusenden, wo es dem Benutzer angezeigt wird.

Für das Backend, das oft eine API bereitstellt, können diese Operationen spezifischen Endpunkten entsprechen, z.B.:

• GET /api/users für das Abrufen von Benutzern (Read)

• POST /api/user für das Erstellen eines neuen Benutzers (Create)

• PUT /api/user/ für das Aktualisieren eines spezifischen Benutzers (Update)

• POST /api/ex für das Erstellen einer neuen Aufgabe (Create)

• GET /api/ex für das Abrufen von Aufgaben (Read)

• PUT /api/ex/:id für das Aktualisieren einer spezifischen Aufgabe (Update)

• DELETE /api/ex/:id für das Löschen einer Aufgabe (Delete)

• GET /api/ex/search/:searchText für das Abrufen von Aufgaben nach Suchbegriffen (Read)

• GET /api/ex/category/:category für das Abrufen von Aufgaben nach Kategorien (Read)

• GET /api/ex/:id für das Abrufen einer spezifischen Aufgabe (Read)

• POST /api/download/ für das Herunterladen der spezifischen Aufgabe im Warenkorb (Download)

2.6.2 Authentifizierung und Autorisierung

Argon2:

Argon2 nimmt als Eingabe das Passwort des Benutzers, einen Salz (eine zufällig generierte Zeichenfolge, die jedem Passwort-Hash hinzugefügt wird, um die Einzigartigkeit zu gewährleisten), und mehrere Parameter, die die Komplexität des Hashing-Vorgangs steuern (wie Speicherbedarf, Rechenzeit und Parallelität).

Basierend auf diesen Eingaben führt Argon2 eine Reihe von komplexen, rechenintensiven Operationen durch. Diese Operationen sind so gestaltet, dass sie sowohl eine hohe Menge an CPU-Ressourcen als auch Speicher benötigen. Dies macht es sehr schwierig für Angreifer, Passwörter zu erraten oder Brute-Force-Angriffe durchzuführen, selbst wenn sie über leistungsfähige Hardware

verfügen.

Das Ergebnis des Prozesses ist ein Passwort-Hash, der in der Datenbank gespeichert wird. Da der gleiche Prozess (mit dem gleichen Salz und denselben Parametern) immer denselben Hash erzeugt, kann das System das vom Benutzer bei der Anmeldung eingegebene Passwort überprüfen, ohne das eigentliche Passwort kennen oder speichern zu müssen.

Sicherheit gegen Brute-Force-Angriffe: Die rechen- und speicherintensive Natur von Argon2 macht es teuer und zeitaufwändig, Hashes zu knacken. Dies schützt gegen Angriffe, die darauf abzielen, Passwörter durch Ausprobieren vieler möglicher Kombinationen (Brute-Force) zu erraten.

A Anhang

B Bilderverzeichnis

Abbildungsverzeichnis

C Literaturverzeichnis

Literatur

[1] Nachname, Vorname *Titel eines ersten Buchs* Jahr: Verlag.

[2] Nachname, Vorname *Titel eines zweiten Buchs* Jahr: Verlag.

[3] Webseite: `http://img.lum.dolimg.com/v1/images/tfa_poster_wide_header_adb92fa0.jpeg?region=61%2C271%2C1937%2C1089&width=600`
(zuletzt kontrolliert am 4.1.2016)