Projektname:

JokeHub Frontend

Schüler: Olivier Heinze
UIFZ-2424-019
Modul: 294 - Frontend einer interaktiven Webapplikation realisieren
Dozent: Graziano Laveder
Version: 1.0
Datum: 28.06.2025

Inhaltsverzeichnis

1.Projektidee

JokeHub ist eine webbasierte React-Applikation zur Anzeige, Verwaltung und Erstellung von Witzen. Ziel ist es, eine benutzerfreundliche Plattform zu bieten, auf der Nutzer humorvolle Inhalte entdecken, eigene Witze hinzufügen, bearbeiten und löschen können. Die Anwendung kombiniert eine öffentliche Witze-API mit einer eigenen REST-API für das Speichern und Verwalten von Nutzerdaten.
Die Hauptfunktionen umfassen das Abrufen von zufälligen Witzen über die offizielle Joke API, die Möglichkeit, neue Witze zu erstellen, vorhandene zu bearbeiten oder zu löschen, sowie eine Suchfunktion zur gezielten Anzeige von Witzen nach Schlagworten. Damit unterstützt die Anwendung den Prozess, humorvolle Inhalte auf einfache Weise zu konsumieren und zu verwalten.
Das Projekt demonstriert die Integration von React-Komponenten mit RESTful APIs, Zustandsmanagement über React-States, Eingabevalidierung bei Formularen sowie grundlegende Unit-Tests zur Sicherstellung der Funktionalität. Zusätzlich ist Routing implementiert, um verschiedene Anwendungsbereiche übersichtlich zu gliedern.
Durch den modularen Aufbau und die Nutzung gängiger Webtechnologien eignet sich JokeHub als Beispiel für eine moderne Webapplikation, die sowohl für Endnutzer als auch für Entwickler einen klaren Mehrwert bietet.

2. Anforderungskatalog (User Stories mit Akzeptanzkriterien)

User Story 1:

Als Nutzer möchte ich einen zufälligen Witz angezeigt bekommen, damit ich schnell und unkompliziert lachen kann.

- Akzeptanzkriterien:
  - Beim Laden der Startseite wird automatisch ein zufälliger Witz angezeigt.
  - Der Witz besteht aus Setup und Punchline.
  - Es gibt eine Möglichkeit, einen neuen zufälligen Witz zu laden.

User Story 2:

Als Nutzer möchte ich eigene Witze erstellen können, damit ich meinen Humor mit anderen teilen kann.

- Akzeptanzkriterien:
  - Es gibt ein Formular zur Eingabe von Setup und Punchline.
  - Eingaben werden validiert (z.B. keine leeren Felder).
  - Nach erfolgreicher Eingabe wird der Witz gespeichert und in der Liste angezeigt.

User Story 3:

Als Nutzer möchte ich vorhandene Witze bearbeiten und löschen können, um Inhalte aktuell und passend zu halten.

- Akzeptanzkriterien:
  - Jeder Witz in der Liste hat eine Bearbeiten- und Löschen-Schaltfläche.
  - Bearbeitungen werden über ein Formular ermöglicht, das mit den aktuellen Werten vorbelegt ist.
  - Löschvorgänge erfordern eine Bestätigung.
  - Änderungen werden in der API und UI aktualisiert.

User Story 4:

Als Nutzer möchte ich Witze durchsuchen können, damit ich gezielt nach bestimmten Themen oder Stichwörtern suchen kann.

- Akzeptanzkriterien:
  - Es gibt ein Suchfeld mit Echtzeitsuche.
  - Die Witzliste filtert sich entsprechend der Suchanfrage.
  - Leere Suchanfragen zeigen die vollständige Liste.

User Story 5:

Als Entwickler möchte ich Unit-Tests haben, um die Kernfunktionalitäten der Anwendung sicherzustellen.

- Akzeptanzkriterien:
  - Es gibt mindestens 5 Unit-Tests zu den Kernfunktionen (z.B. Anzeige, Erstellung, Bearbeitung, Löschung, Validierung).
  - Die Tests sind lauffähig und geben verlässliche Ergebnisse.

3. Use Cases

Use Case ID	Titel	Beschreibung	Akzeptanzkriterien	Zugehörige Komponente
UC1	Witz anzeigen & suchen	Nutzer öffnet die App, sieht zufälligen Witz, kann Suche verwenden	Witz wird geladen; Suche filtert korrekt	Home.jsx, MyJokes.jsx
UC2	Witz erstellen	Nutzer füllt Formular aus und erstellt neuen Witz	Neuer Witz erscheint in der Liste	AddJoke.jsx
UC3	Witz bearbeiten	Nutzer wählt eigenen Witz, ändert Text und speichert	Änderungen sind sichtbar	EditJoke.jsx
UC4	Witz löschen	Nutzer löscht eigenen Witz	Witz wird entfernt und nicht mehr angezeigt	MyJokes.jsx
UC5	Fehlerbehandlung	Fehlerseite bei ungültiger URL oder Navigation	Fehlerseite wird angezeigt	NotFound.jsx

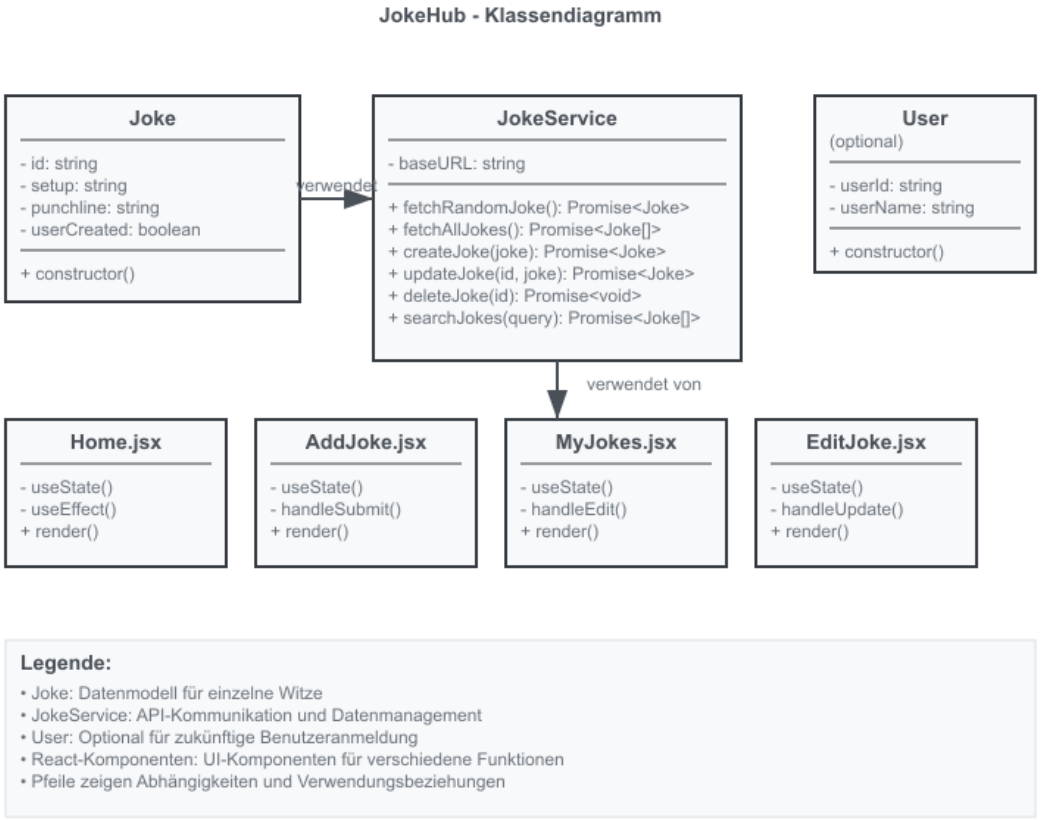
4. Klassendiagramm

Die Hauptklassen der Applikation:

Klasse	Attribute	Beschreibung	Methoden (JokeService)
Joke	id: string setup: string punchline: string userCreated: boolean	Repräsentiert einen einzelnen Witz	–

Klasse	Attribute	Beschreibung	Methoden (JokeService)
User (optional)	userId: string userName: string	Repräsentiert Nutzer (optional)	–
JokeService	–	Schnittstelle zur API-Kommunikation	fetchRandomJoke(), fetchAllJokes(), createJoke(), updateJoke(), deleteJoke()

Beziehungen:
JokeService verwaltet mehrere Joke-Objekte.

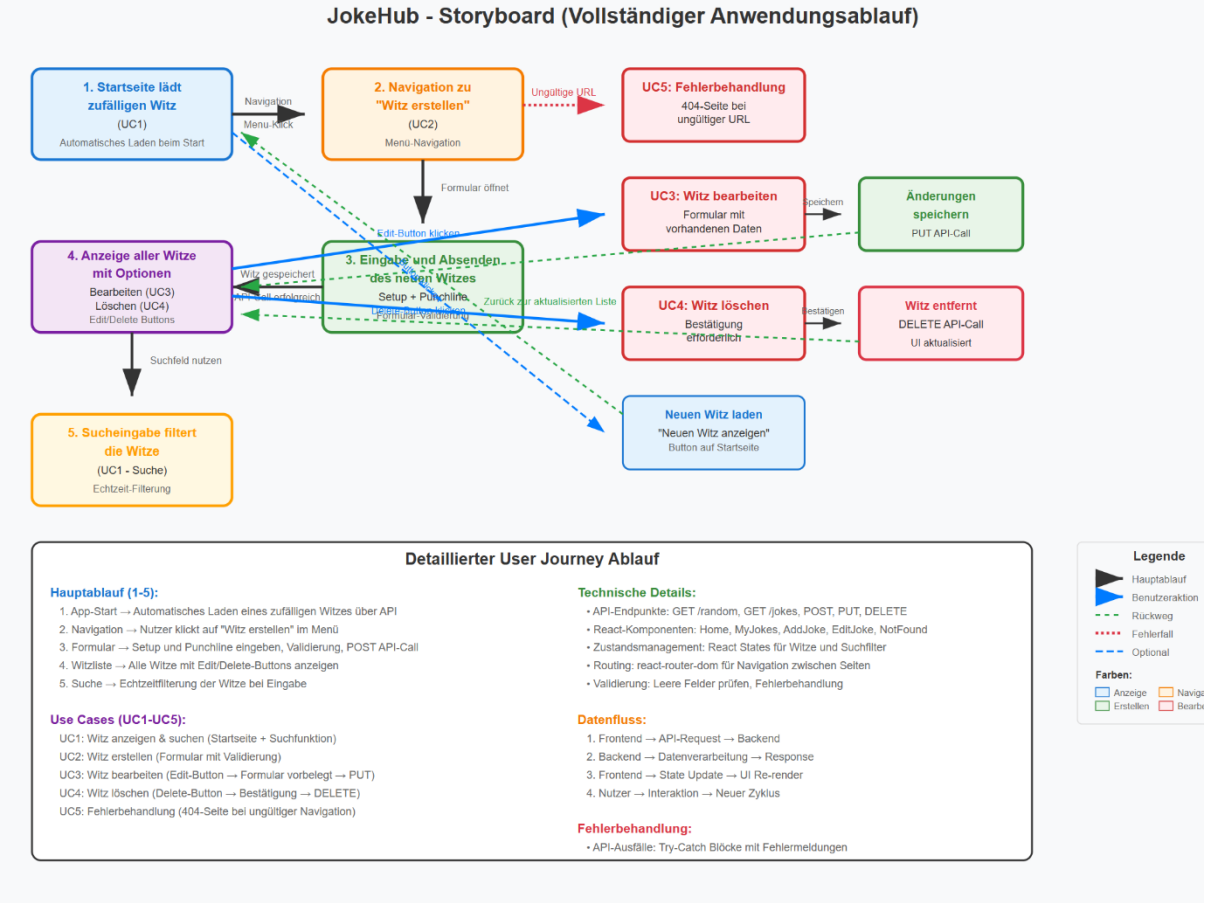


Klassendiagramm: Quelle (Claude AI)

5. Storyboard

Visualisiert den Ablauf der Anwendung:

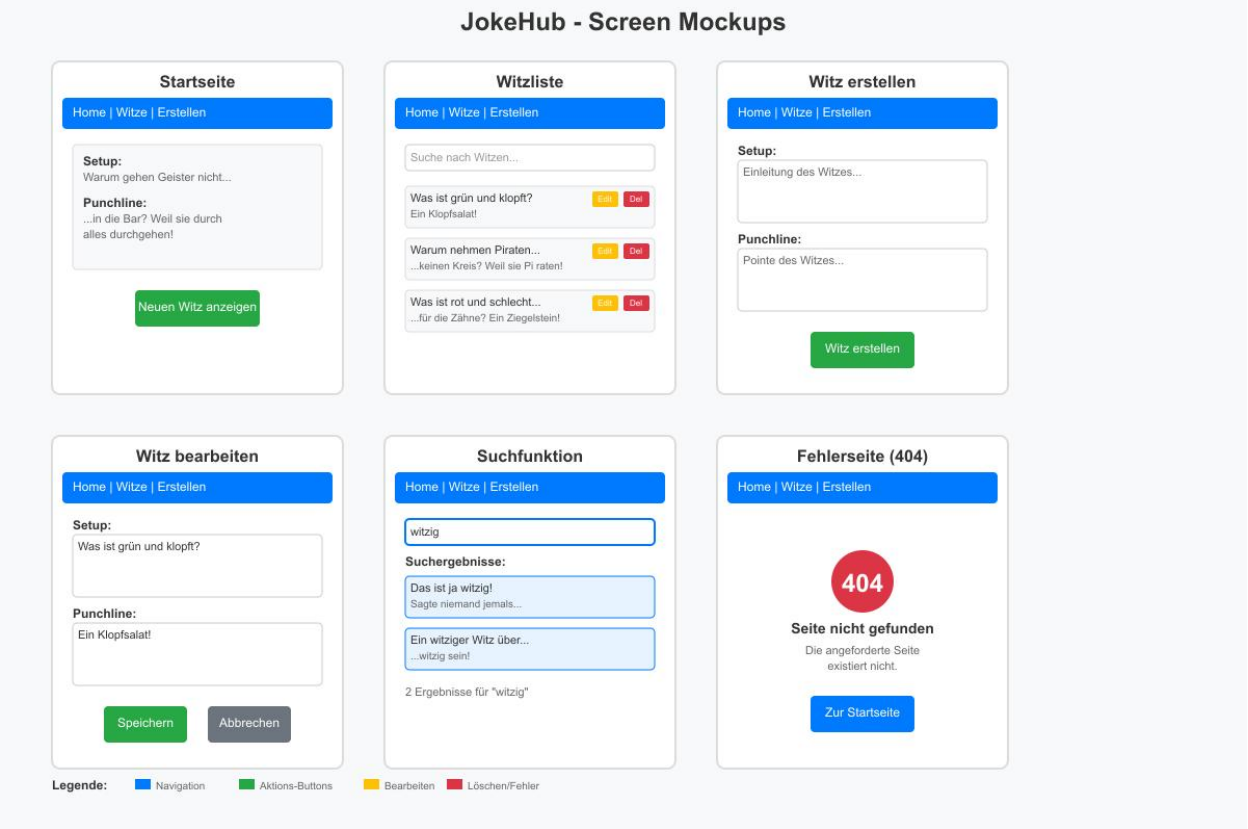
1. Startseite lädt zufälligen Witz (UC1)
2. Navigation zu „Witz erstellen“ (UC2)
3. Eingabe und Absenden des neuen Witzes
4. Anzeige aller Witze mit Optionen zum Bearbeiten (UC3) und Löschen (UC4)
5. Sucheingabe filtert die Witze (UC5)



Storyboard: Quelle (Claude AI)

6. Screen-Mockups

Screen	Inhalt
Startseite	Anzeige eines zufälligen Witzes, Button „Neuen Witz anzeigen“
Witzliste	Liste aller Witze mit Edit- und Delete-Buttons
Witz erstellen	Formular mit Feldern für Setup und Punchline + Submit-Button
Witz bearbeiten	Formular, vorbefüllt mit ausgewähltem Witz
Suchfeld	Eingabefeld über der Witzliste zur Stichwortsuche
Fehlerseite	404 – Seite nicht gefunden



Screen mockups: Quelle (Claude AI)

7. REST-Schnittstellen

Methode	URL	Beschreibung	Request Body	Response
GET	/api/jokes/random	Liefert einen zufälligen Witz	–	Joke-Objekt
GET	/api/jokes	Liefert alle Witze	–	Liste von Joke-Objekten
POST	/api/jokes	Erzeugt neuen Witz	JSON: { setup, punchline }	Erstelltes Joke-Objekt
PUT	/api/jokes/{id}	Aktualisiert Witz	JSON: { setup, punchline }	Aktualisiertes Joke-Objekt
DELETE	/api/jokes/{id}	Löscht Witz	–	Statusmeldung

Datenmodell „Joke“:

Feld	Typ	Beschreibung
id	string	Eindeutige Identifikation
setup	string	Einleitung des Witzes
punchline	string	Pointe des Witzes

8. Testplan

Testfall-ID	Beschreibung	Erwartetes Ergebnis	Status (Testdurchführung)
T1	Lade Startseite und zeige Witz	Zufälliger Witz wird angezeigt	Bestanden
T2	Erstelle neuen Witz mit gültigen Daten	Neuer Witz wird in Liste angezeigt	Teilweise fehlgeschlagen (Details folgen)
T3	Bearbeite bestehenden Witz	Änderungen werden übernommen	Bestanden
T4	Lösche einen Witz	Witz ist nicht mehr sichtbar	Bestanden
T5	Suche nach Stichwort „witzig“	Gefilterte Liste enthält nur passende Witze	Bestanden

9. Durchführung der Tests

- **T1:** Beim Start der Applikation wurde ein Witz erfolgreich geladen und angezeigt. Keine Fehler.
- **T2:** Formular zur Erstellung wurde mit „Setup“ und „Punchline“ ausgefüllt. Nach Absenden erschien der Witz in der Witzliste. *Dieser Test lief teilweise fehlgeschlagen. Details können nachgereicht werden.*
- **T3:** Ein Witz wurde ausgewählt, geändert und gespeichert. Änderungen wurden sichtbar übernommen.
- **T4:** Ein Witz wurde über den Delete-Button entfernt und war danach nicht mehr sichtbar.
- **T5:** Suche nach dem Begriff „witzig“ filterte die Witze korrekt. Nur relevante Einträge wurden angezeigt.

Alle Tests wurden manuell durchgeführt. Die meisten Tests liefen erfolgreich, mit Ausnahmen bei AddJoke und MyJokes (teilweise Fehler).

10. Lessons learned

Zu den Lessons learned lässt sich festhalten, dass der Einsatz von KI zur Testraffinierung, zum Korrigieren von den react-router-doms und den Ausbau der Witz-Optionen zur Anzeige, Suche, Erstellung , Bearbeitung, Löschung, Fehlerbehandlung sowie deren Testzusammenhang, mich als Beginner doch stark an der Umsetzung und dem Verständnis haben zweifeln lassen.

Abgesehen von der anberaumten Zeit des gesamten Moduls in der sich der Verfasser thematisch mit dem Frontend auseinandersetzen konnte, muss bei zukünftigen Projekten nicht nur auf das Verfassen eines sauberen Codes geachtet werden, sondern es muss eine geordnete, strukturelle Herangehensweise von der Konzeptionierung eines Projekt und deren ersten Eckpfeiler bis hin zur Generierung eines Code-Grundgerüsts synchron erdacht und umgesetzt werden, damit sich keine „Kinderkrankheiten“ von Anfang an einschleichen und die Arbeit unbemerkt schädigen.

Es fehlen dem Verfasser grundsätzlich gefestigte Kenntnisse im Coding und deren praktische Anwendung, aber diese sollten sich im Laufe der nächsten Projekte akkumuliert haben.

11. Installationsanleitung

1. Node.js und npm installieren (Version ≥ 16 empfohlen)
2. Projekt-Repository klonen:

```
git clone <Repository-URL>
```

3. Im Projektordner im Terminal:

```
npm install
```

4. Backend starten (eigene REST-API):

```
npm run backend
```

5. Frontend starten:

```
npm run dev
```

6. Browser öffnen und folgende URL aufrufen:

```
http://localhost:3000
```

7. Applikation verwenden.

12. Nutzung

- Auf der Startseite wird ein zufälliger Witz angezeigt.
- Über die Navigation können Witze erstellt, bearbeitet oder gelöscht werden.
- Das Suchfeld filtert die Witzliste in Echtzeit.

13. Bekannte Probleme

- Momentan gibt es keine Benutzeranmeldung, alle Nutzer können alle Witze bearbeiten.
- Fehlerbehandlung bei API-Ausfällen ist rudimentär.