014-Antragsstrecke-Mehrstufig

Frontend-Aufgabe
Anforderungsniveau: Mittel

1 Ziel der Aufgabe

Sie bauen eine Antragsstrecke, die über drei Schritte die Angaben zu einem Vorhaben einsammelt und am Ende eine Zusammenfassung zum Absenden anbietet. Geprüft wird weniger das Formular selbst als die Zustandsverwaltung über Schrittgrenzen hinweg: Ein Antrag überlebt Schrittwechsel, Neuladen der Seite und einen Abbruch am Vortag.

Veranschlagte Bearbeitungszeit: 4–5 Stunden

Folgende Techniken kommen zum Einsatz:

- React mit TypeScript, bevorzugt Next.js (App Router) oder Vite
- Formularverwaltung mit React Hook Form, Schema-Validierung mit Zod
- Zustandsverwaltung über mehrere Routen (Context, Zustand oder vergleichbar)
- Persistenz eines Entwurfs im Browser (`localStorage`)
- Routing pro Schritt, Umgang mit Direkteinstiegen und Browser-Zurück
- Barrierefreiheit im Formular: Beschriftungen, Fehlerausgabe, Fokusführung
- Komponententests mit Vitest oder Jest und React Testing Library

2 Aufgabenbeschreibung

1. **Projekt aufsetzen**

   Legen Sie ein Projekt mit TypeScript an und richten Sie die üblichen Skripte ein (`dev`, `build`, `start`, `test`, `lint`). Ein Formatierer ist willkommen, aber nur mit einer Konfiguration im Projekt – ohne die formatiert er halbe Dateien um und macht jeden Diff unlesbar.

   ```
   npx create-next-app@latest antragsstrecke --typescript --eslint --app
   cd antragsstrecke
   npm install react-hook-form zod @hookform/resolvers
   npm install -D vitest @vitejs/plugin-react jsdom
   npm install -D @testing-library/react @testing-library/user-event @testing-library/
       jest-dom
   ```

   Vitest bringt in einem React-Projekt keine fertige Einrichtung mit: Sie brauchen eine eigene `vitest.config.ts` mit dem React-Plugin und `environment: "jsdom"` sowie ein Setup-Modul für `@testing-library/jest-dom`. Rechnen Sie diesen Schritt ein.

2. **Die drei Schritte**

   Jeder Schritt ist eine eigene Route, damit Zurück-Knopf und Lesezeichen funktionieren, zum Beispiel `/antrag/schritt-1` bis `/antrag/schritt-3`.

1. **Antragsteller** – Auswahl zwischen „Privatperson“ und „Unternehmen“, Name, E-Mail. Bei „Unternehmen“ kommt ein Pflichtfeld Firmenname hinzu, bei „Privatperson“ nicht. Die Felder richten sich also nach der getroffenen Auswahl.
2. **Vorhaben** – Titel (max. 80 Zeichen), Beschreibung (mind. 50, max. 2000 Zeichen mit sichtbarem Restzähler) und gewünschtes Startdatum.
3. **Zusammenfassung** – alle Angaben gruppiert nach Schritt, jede Gruppe mit einem Verweis „Bearbeiten“, der genau zu diesem Schritt zurückführt. Pflichtankreuzfeld für die Richtigkeitserklärung, danach Absenden.

Mehr Felder brauchen wir nicht. Uns interessiert, wie Sie die Strecke zusammenhalten, nicht wie viele Eingaben Sie darin unterbringen.

3. **Zustand über Schrittgrenzen**

Halten Sie den gesamten Antrag in einem zentralen, typisierten Zustand. Jeder Schritt liest daraus seine Vorbelegung und schreibt beim Weitergehen zurück. Vorwärts geht es nur, wenn der aktuelle Schritt gültig ist; rückwärts immer. Ein Direkteinstieg auf `/antrag/schritt-3` ohne ausgefüllte Vorschritte leitet auf den ersten unvollständigen Schritt um.

```
type Antragsart = "privat" | "unternehmen";

type AntragEntwurf = {
  schemaVersion: 1;
  aktualisiertAm: string;   // ISO-8601
  letzterSchritt: 1 | 2 | 3;
  daten: {
    schritt1?: {
      art: Antragsart;
      name: string;
      email: string;
      firmenname?: string;  // nur bei "unternehmen"
    };
    schritt2?: {
      titel: string;
      beschreibung: string;
      startdatum: string;
    };
  };
};
```

4. **Wiederaufnahme nach Abbruch**

Der Entwurf wird bei jeder Änderung gespeichert (gedrosselt, nicht bei jedem Tastendruck). Beim erneuten Aufruf der Strecke erkennt die Anwendung einen vorhandenen Entwurf und fragt, ob fortgesetzt oder neu begonnen werden soll – mit Datum der letzten Änderung und dem erreichten Schritt.

Beim Lesen wird der gespeicherte Stand validiert: Passt `schemaVersion` nicht oder scheitert das Zod-Schema, wird der Entwurf verworfen statt die Anwendung in einen halb gefüllten Zustand zu bringen.

```
export function entwurfLaden(): AntragEntwurf | null {
  const roh = localStorage.getItem("antrag.entwurf.v1");
  if (!roh) return null;

  let geparst;
  try {
```

```
    geparst = entwurfSchema.safeParse(JSON.parse(roh));
  } catch {
    // kein gueltiges JSON - JSON.parse wirft, bevor das Schema greift
    localStorage.removeItem("antrag.entwurf.v1");
    return null;
  }
  if (!geparst.success) {
    localStorage.removeItem("antrag.entwurf.v1");
    return null;
  }
  return geparst.data;
}
```

5. **Absenden**

   Beim Absenden schicken Sie den Antrag als JSON an die öffentliche Test-Schnittstelle `https://jsonplaceholder.typicode.com/posts`. Sie antwortet mit einem Feld `id`, das Sie als Vorgangsnummer verwenden können. Während des Sendens ist der Absende-Knopf gesperrt und zeigt einen Spinner; ein zweiter Klick darf keinen zweiten Antrag auslösen. Bei Erfolg erscheint eine Bestätigungsseite mit Vorgangsnummer, und der gespeicherte Entwurf wird gelöscht. Bei einem Fehler bleibt der Entwurf erhalten und es gibt eine verständliche Meldung mit Wiederholmöglichkeit.

6. **Fortschritt und Führung**

   Über der Strecke steht eine Fortschrittsanzeige mit allen drei Schritten und ihrem Zustand (offen, in Arbeit, abgeschlossen). Bereits abgeschlossene Schritte sind darüber anklickbar, noch nicht erreichte nicht. Die Anzeige funktioniert auch auf schmalen Bildschirmen (ab 360 px Breite) ohne horizontales Scrollen der Seite.

3 Anforderungen

- **Typisierung:** durchgängig TypeScript, kein `any`. Die Validierungsschemata sind die einzige Quelle der Wahrheit für die Formulartypen (`z.infer`).
- **Validierung:** pro Schritt ein eigenes Schema, Fehler direkt am Feld. Beim gescheiterten Weitergehen springt der Fokus auf das erste fehlerhafte Feld.
- **Persistenz:** Entwurf mit `schemaVersion` und Zeitstempel; ungültige oder fremde Stände werden verworfen.
- **Icons:** ausschließlich aus **einem** Set – entweder Lucide oder Phosphor. Kein Mischen, keine Emojis als Bedienelemente.
- **Ladezustände:** für den Absende-Knopf ein Spinner im Knopf, kein Layout-Sprung, wenn sich die Beschriftung ändert.
- **Barrierefreiheit:** jedes Feld mit `label`, zusammengehörige Felder in `fieldset` mit `legend`, Fehler über `aria-invalid` und `aria-describedby` verknüpft, die gesamte Strecke per Tastatur bedienbar.
- **Responsiv:** nutzbar von 360 px bis Desktop. Prüfen Sie die Fortschrittsanzeige ausdrücklich auch in der schmalen Ansicht.
- **Tests:** mindestens drei Tests, darunter zwingend

  - Weitergehen mit ungültiger Eingabe bleibt auf dem Schritt,
  - ein gespeicherter Entwurf wird beim Neustart korrekt geladen,
  - ein Entwurf mit falscher `schemaVersion` wird verworfen.

- **Keine Server-Logik:** außer dem Absende-Aufruf gibt es keine Hintergrunddienste. Es ist keine Datenbank und keine Anmeldung nötig.

4 Abgabe

- Git-Repository mit nachvollziehbarer Historie – nicht ein einziger Commit „initial“. Commit-Nachrichten in ganzen Sätzen.
- `README.md` mit: Startanleitung, verwendeten Bibliotheken, Aufbau des Zustands sowie Ablageort und Format des Entwurfs.
- Ein kurzer Abschnitt „Bekannte Grenzen“: was Sie bewusst weggelassen haben und warum.
- Das Projekt startet nach `npm install` mit:

```
npm install
npm run dev
npm test
```

- Keine echten personenbezogenen Daten in Beispielen, Tests oder Screenshots. Verwenden Sie erfundene Werte (`max.mustermann@example.invalid`).

5 Bewertungskriterien

- **Zustandsführung (35 %):** Ist der Zustand an einer Stelle gebündelt und typisiert? Überlebt er Schrittwechsel, Neuladen und Browser-Zurück? Ist die Wiederaufnahme nachvollziehbar gelöst, einschließlich des Falls „kaputter Entwurf“?
- **Validierung und Führung (25 %):** Greift die Prüfung pro Schritt? Sind Fehlermeldungen verständlich und am richtigen Ort? Ist ein Direkteinstieg auf einen späteren Schritt sauber abgefangen?
- **Oberfläche und Standards (20 %):** Einheitliches Icon-Set, kein Layout-Sprung, saubere Darstellung bis hinunter zu 360 px, Beschriftungen und Fokusführung im Formular.
- **Tests (10 %):** Prüfen die Tests das Verhalten oder nur, dass etwas gerendert wurde? Ist der Wiederaufnahme-Pfad wirklich abgedeckt?
- **Code und Dokumentation (10 %):** Lesbare Komponenten in sinnvoller Größe, klare Trennung zwischen Formular, Zustand und Persistenz, README, die jemand ohne Rückfrage benutzen kann.

Wir bewerten nicht nach Zeilenzahl. Eine schlanke, an den Rändern durchdachte Umsetzung ist uns lieber als eine breite mit sechs halb fertigen Extras.