016-Datei-Upload-Fortschritt

Frontend-Aufgabe
Anforderungsniveau: Mittel

1 Ziel der Aufgabe

Sie bauen eine Oberfläche, über die mehrere Dateien gleichzeitig hochgeladen werden. Jede Datei bekommt einen eigenen Fortschritt, lässt sich einzeln abbrechen und nach einem Fehlschlag wiederholen. Die Ablagefläche per Drag-and-Drop ist dabei nur einer von zwei gleichwertigen Wegen: alles, was mit der Maus geht, muss ebenso über Tastatur und den gewöhnlichen Dateidialog gehen. Geprüft werden:

- React mit Next.js (App Router) und TypeScript
- Upload mit echtem Fortschritt (`XMLHttpRequest` und dessen `upload.onprogress`, da `fetch` im Browser keinen Sendefortschritt liefert)
- Abbruch einzelner Uploads und Wiederholung fehlgeschlagener
- Drag-and-Drop-Ereignisse (`dragenter`, `dragover`, `dragleave`, `drop`) inklusive `dataTransfer`
- Barrierefreiheit: Tastaturbedienung, Fokus, `aria`-Attribute, Statusmeldungen in einer Live-Region
- Zustandsverwaltung einer Liste mit mehreren gleichzeitigen Übergängen

Veranschlagte Bearbeitungszeit: 6–8 Stunden

2 Aufgabenbeschreibung

1. **Projekt aufsetzen**

   Legen Sie ein Next.js-Projekt mit TypeScript und App Router an.

   ```
   npx create-next-app@latest upload-flaeche --typescript --app
   cd upload-flaeche
   npm run dev
   ```

   Ein Zustandswerkzeug ist freigestellt; React-Bordmittel genügen. Die Pfadangaben in dieser Aufgabe gehen von einem `src/`-Verzeichnis aus. Legen Sie das Projekt entweder mit `--src-dir` an, oder lassen Sie `src/` in allen folgenden Pfaden weg.

2. **Gegenstelle im Projekt**

   Schreiben Sie eine eigene Route, die die Dateien entgegennimmt. Damit sind Sie von fremden Diensten unabhängig und können Verzögerung und Fehlerfälle selbst erzeugen. Übernehmen Sie die Route so, wie sie hier steht; sie ist nicht Teil der eigentlichen Aufgabe.

   ```
   // src/app/api/upload/route.ts
   export async function POST(request: Request) {
     const datei = (await request.formData()).get("file");
     if (!(datei instanceof File)) {
   ```

```
    return Response.json({ fehler: "keine Datei" }, { status: 400 });
  }
  // kuenstliche Dauer, damit der Zustand "laeuft" lange genug anliegt,
  // um Abbrechen von Hand ausprobieren zu koennen
  await new Promise((r) => setTimeout(r, 1500 + Math.random() * 2500));
  // im Schnitt jede fuenfte Anfrage scheitert, damit "Wiederholen" pruefbar ist
  if (Math.random() < 0.2) {
    return Response.json({ fehler: "Speicher nicht erreichbar" }, { status: 503 });
  }
  return Response.json({ id: crypto.randomUUID(), name: datei.name });
}
```

Die Dateien müssen nicht dauerhaft abgelegt werden.

Beachten Sie: Die künstliche Dauer verzögert die *Antwort*, nicht das Senden. Gegen `localhost` sind auch 10 MB in Sekundenbruchteilen übertragen, der Balken springt also auf 100 Prozent und der Eintrag bleibt danach auf `laeuft` stehen, bis die Antwort eintrifft. Um den Fortschritt wirklich laufen zu sehen, drosseln Sie die Verbindung in den Entwicklerwerkzeugen des Browsers (Netzwerk-Reiter, etwa „Slow 3G“). Bauen Sie den Fortschritt nicht nach, er muss aus den echten Ereignissen kommen.

3. **Ablagefläche**

   - Beim Betreten mit Dateien wechselt die Fläche sichtbar den Zustand (Rahmen, Hintergrund). Verlassen und Ablegen setzen ihn zurück.
   - Verschachtelte Kindelemente dürfen kein Flackern erzeugen. Zählen Sie `dragenter` und `dragleave` gegeneinander auf, oder werten Sie `relatedTarget` aus.
   - `dragover` muss `preventDefault()` aufrufen, sonst löst der Browser gar kein `drop` aus, sondern öffnet die Datei.

4. **Der zweite, gleichwertige Weg**

   Die Ablagefläche ist ein Angebot, keine Bedingung.

   - Ein sichtbarer Schalter „Dateien auswählen“ öffnet den Dateidialog (ein `input` vom Typ `file` mit `multiple`).
   - Die Ablagefläche selbst ist über die Tabulatortaste erreichbar, hat einen deutlich sichtbaren Fokusring und öffnet mit Eingabe- oder Leertaste denselben Dialog.
   - Der Dateiwähler wird optisch versteckt, aber nicht mit `display: none` oder `visibility: hidden` – damit verschwindet er auch für Hilfsmittel und ist nicht mehr fokussierbar. Üblich sind eine „visually hidden“-Klasse (Ausschnitt über `clip-path`) oder eine Überlagerung mit `opacity: 0`, jeweils mit einem zugehörigen `label`. Achten Sie darauf, dass dabei keine drei Tabulatorhalte entstehen, die alle dasselbe tun.
   - Auswahl derselben Datei zweimal hintereinander muss funktionieren; setzen Sie dazu den Wert des Eingabefeldes nach der Übernahme wieder auf die leere Zeichenkette zurück.
   - Jeder Eintrag der Liste ist mit der Tastatur bedienbar: abbrechen, wiederholen, entfernen.

5. **Liste und Fortschritt**

   Jede übernommene Datei wird ein Eintrag mit eigenem Zustand. Alle übernommenen Dateien starten sofort; eine Warteschlange mit Parallelitätsbegrenzung ist *nicht* gefordert.

```
type UploadZustand = "laeuft" | "fertig" | "abgebrochen" | "fehler";

type Eintrag = {
```

```
  id: string;            // eigene Kennung, nicht der Dateiname
  datei: File;
  zustand: UploadZustand;
  prozent: number;       // 0..100
  meldung?: string;      // Fehlertext zum Anzeigen
  xhr?: XMLHttpRequest;
};
```

Anforderungen an den Ablauf:

- Der Fortschritt kommt aus `xhr.upload.onprogress`. Rechnen Sie aus `event.loaded` und `event.total`; prüfen Sie `event.lengthComputable`.
- Ein Balken je Datei mit Prozentwert.
- Der Balken trägt die Rolle `progressbar`. Setzen Sie darauf `aria-valuenow`, `aria-valuemin` und `aria-valuemax` sowie eine Beschriftung, die die Datei benennt.

```
const xhr = new XMLHttpRequest();
const daten = new FormData();
daten.append("file", eintrag.datei);
xhr.upload.onprogress = (e) => {
  if (e.lengthComputable) setProzent(Math.round((e.loaded / e.total) * 100));
};
xhr.open("POST", "/api/upload");
xhr.send(daten);   // xhr merken, sonst kein xhr.abort()
```

6. **Abbrechen und Wiederholen**

   - Ein laufender Upload lässt sich einzeln abbrechen (`xhr.abort()`). Der Eintrag geht auf `abgebrochen` und bleibt in der Liste stehen.
   - Abgebrochene und fehlgeschlagene Einträge lassen sich wiederholen. Der Fortschritt beginnt dabei wieder bei null.
   - Nach dem Abbruch darf kein verspäteter `onprogress`- oder `onload`-Aufruf den Zustand mehr verändern. Räumen Sie die Rückrufe auf.

7. **Prüfung vor dem Senden**

   Prüfen Sie vor dem Start und melden Sie Verstöße am jeweiligen Eintrag, nicht in einem Sammelhinweis: höchstens 10 MB je Datei, erlaubt sind nur `image/png`, `image/jpeg` und `application/pdf`. Die Prüfung genügt im Browser; die Route bleibt wie oben abgedruckt.

8. **Anzeige der Liste**

   Je Eintrag zeigen Sie: Name, Größe in lesbarer Form („2,4 MB“), Zustand, Fortschrittsbalken und die passenden Schalter. Eine Bildvorschau ist nicht gefordert.

3 Anforderungen

- **Icons ausschließlich aus einem Satz** – entweder Lucide oder Phosphor, nicht beides. Keine Emojis als Bedienzeichen.
- **Kein Layout-Sprung.** Ein Eintrag hat vor, während und nach dem Upload dieselbe Höhe; Zustandstext und Schalter dürfen die Zeile nicht umbrechen lassen.

- **Tooltips** an gekappten Dateinamen (voller Name) und an reinen Icon-Schaltern (Abbrechen, Wiederholen, Entfernen). Nicht mehr als nötig – beschriftete Schalter brauchen keinen.
- **Barrierefreiheit:** Zustandswechsel werden in einer höflichen Live-Region angesagt, etwa „bericht.pdf hochgeladen“. Der Fokus geht beim Entfernen eines Eintrags nicht verloren. Kein Zustand ist allein über Farbe unterscheidbar.
- **Fehlertexte sind verständlich** und nennen den nächsten Schritt. „503“ ist kein Fehlertext.
- **Kein Formatierer ohne Projektkonfiguration.** Wenn Sie Prettier oder ESLint verwenden, legen Sie die Konfiguration mit ins Repository.
- Die Oberfläche ist ab 360 px Breite benutzbar, TypeScript ohne `any` an den tragenden Stellen.
- Mindestens zwei Tests, die den Kern abdecken und nicht nur den Sonnenschein-Fall: Übernahme über den Dateidialog und Abbruch eines laufenden Uploads. Testwerkzeug frei (Vitest, Jest, Playwright).

4 Abgabe

- Ein Git-Repository mit nachvollziehbaren Commits. Kein einzelner Commit „initial“ über den gesamten Stand. `node_modules/` und hochgeladene Dateien gehören nicht hinein.
- `README.md` mit Startanleitung (Installieren, Starten, Tests ausführen), verwendetem Icon-Satz und einer kurzen Begründung, warum `XMLHttpRequest` statt `fetch` zum Einsatz kommt.
- Darin zusätzlich ein Abschnitt „Bedienung ohne Maus“: welche Tasten welchen Vorgang auslösen.
- Das Projekt läuft nach `npm install` und `npm run dev` ohne weitere Handgriffe.

5 Bewertungskriterien

- **Funktion (35%)** – korrekter Fortschritt je Datei, Abbruch und Wiederholung tun das, was draufsteht, Prüfung greift vor dem Senden.
- **Bedienung ohne Maus (25%)** – der Weg über Tastatur und Dateidialog ist gleichwertig, nicht nachgereicht. Fokus, Ansage, Beschriftung.
- **Zustandsführung (15%)** – saubere Übergänge, keine verwaisten Rückrufe nach Abbruch, kein Flackern der Ablagefläche.
- **Oberfläche (15%)** – ein Icon-Satz, keine Layout-Sprünge, sinnvolle Tooltips, verständliche Fehlertexte, schmaler Bildschirm.
- **Code-Qualität (10%)** – Komponentenschnitt, Typen, Commits.

Halten Sie Annahmen, die Sie treffen mussten, in der `README.md` fest.