020-Internationalisierung-Lernplattform

Frontend-Aufgabe
Anforderungsniveau: Schwer

1 Ziel der Aufgabe

Sie legen einen Ausschnitt einer Lernplattform zweisprachig aus: Deutsch und Arabisch. Geprüft wird, ob Sie Internationalisierung als Architekturthema behandeln und nicht als Textaustausch. Arabisch ist der eigentliche Prüfstein: Es kehrt die Schreibrichtung um und hat sechs statt zwei Pluralformen.

Veranschlagte Bearbeitungszeit: 6–8 Stunden

Verwendete Techniken:

- Next.js (App Router) mit TypeScript
- Sprachpräfix in der Route und Aushandlung der Sprache beim Einstieg
- ICU MessageFormat für Pluralregeln
- Die `Intl`-API für Zahlen, Preise und Datumsangaben
- Schreibrichtung über `dir` und logische CSS-Eigenschaften

2 Aufgabenbeschreibung

1. **Projekt aufsetzen**

   Legen Sie ein Next.js-Projekt mit TypeScript und App Router an (`create-next-app`) und binden Sie eine Internationalisierungs-Bibliothek ein. Empfohlen ist `next-intl`; alternativ sind `i18next` mit `react-i18next` und `i18next-icu` zulässig.

2. **Sprachrouting**

   Jede Seite liegt unter einem Sprachpräfix (`/de/kurse`, `/ar/kurse`). Unterstützt werden genau `de` und `ar`; Standardsprache ist `de`. Die Pfadsegmente hinter dem Präfix bleiben in beiden Sprachen gleich, übersetzte Pfade sind nicht gefordert.

   - `/` leitet auf eine Sprache weiter. Maßgeblich ist der Header `Accept-Language`, sonst die Standardsprache.
   - Eine unbekannte Sprache im Pfad (`/fr/kurse`) führt zu einer 404-Seite, nicht zu einer stillen Weiterleitung.
   - Das `html`-Element trägt `lang` und `dir` passend zur aktiven Sprache. Beides wird serverseitig gesetzt, nicht per `useEffect` nachgereicht.
   - Ein Umschalter in der Kopfzeile wechselt die Sprache und bleibt dabei auf derselben Seite: Von `/de/kurse/7` geht es nach `/ar/kurse/7`, nicht zur Startseite. Der Wechsel erfolgt ohne vollständiges Neuladen des Dokuments.

3. **Übersetzungskataloge**

   Legen Sie je Sprache eine Katalogdatei an (`messages/de.json`, `messages/ar.json`). Gliedern Sie die Schlüssel nach Bereichen, damit die Dateien nicht zur flachen Liste werden.

```
{
  "navigation": { "courses": "Kurse" },
  "course": {
    "lessonCount": "{count, plural, one {# Lektion} other {# Lektionen}}",
    "updatedOn": "Aktualisiert am {date, date, long}"
  }
}
```

4. **Die Oberfläche**

   Bauen Sie **eine** Ansicht: die **Kursliste**. Karten mit Titel, Anzahl der Lektionen, Fortschrittsbalken und Preis. Die Daten kommen aus einer lokalen JSON-Datei im Projekt; eine externe API ist nicht nötig.

   Erfinden Sie vier Kurse; Titel und Kurzbeschreibung liegen in beiden Sprachen vor. Die arabischen Texte dürfen aus einem Übersetzungsdienst stammen, müssen aber echte arabische Schrift sein und kein Platzhaltertext.

   Kursinhalte sind Daten, keine Oberflächentexte: Sie gehören in die Datenquelle (je Kurs ein Feld pro Sprache), nicht in die Übersetzungskataloge. In die Kataloge gehört alles, was zur Oberfläche selbst zählt – Beschriftungen, Einheiten, Meldungen, Schaltflächen.

5. **Plural- und Formatregeln**

   Deutsch kennt zwei Pluralformen, Arabisch sechs: `zero`, `one`, `two`, `few`, `many`, `other`. Ihr arabischer Katalog muss für die Lektionsanzahl alle sechs bedienen.

```
const pr = new Intl.PluralRules("ar");
pr.select(0)  // "zero"   pr.select(1)   // "one"   pr.select(2)   // "two"
pr.select(3)  // "few"    pr.select(11)  // "many"  pr.select(100) // "other"
```

   Formatieren Sie ausnahmslos über `Intl`, niemals über eigene Zeichenkettenarithmetik: Zahlen und Prozentwerte über `NumberFormat`, Preise über `NumberFormat` mit `style: "currency"` (die Währung bleibt in beiden Sprachen EUR, nur die Darstellung ändert sich), Datumsangaben über `DateTimeFormat`.

   Halten Sie im README fest, welche Ausgabe Sie für `de` und `ar` bei den Lektionszahlen 0, 1, 2, 3, 11 und 100 erhalten.

6. **Schreibrichtung**

   Für Arabisch steht `dir="rtl"`, für Deutsch `dir="ltr"`. Das Layout muss sich daraus ableiten, nicht aus einer zweiten Stilvariante:

   - Logische CSS-Eigenschaften (`margin-inline-start`, `inset-inline-start`, `text-align: start`) statt `left` und `right`. In Tailwind die Varianten `ms-*`, `me-*`, `ps-*`, `pe-*`, `start-*`.
   - Richtungsgebundene Symbole – etwa ein Chevron auf der Karte – werden im RTL-Modus gespiegelt. Gegenständliche Symbole (Uhr, Nutzer, Haken) bleiben unverändert.
   - Der Fortschrittsbalken wächst in Leserichtung: in `ar` von rechts nach links.
   - Zahlen und lateinische Eigennamen innerhalb arabischer Sätze dürfen die Richtung nicht zerreißen. Prüfen Sie einen solchen Satz.

```
.card__meta { margin-inline-start: 1rem; text-align: start; }
[dir="rtl"] .icon--directional { transform: scaleX(-1); }
```

7. **Fehlende Schlüssel**

   Ein fehlender Schlüssel darf weder die Seite abstürzen lassen noch unbemerkt als leerer Text durchrutschen. Legen Sie ein Verhalten fest und begründen Sie es im README – etwa: in der Entwicklung sichtbar markieren (`[[course.lessonCount]]`) samt Warnung auf der Konsole, in der Produktion auf die Standardsprache zurückfallen.

   Schreiben Sie zusätzlich ein kurzes Prüfskript, das die Schlüsselmengen beider Kataloge vergleicht und bei Abweichung mit einem Fehlercode endet. Es muss über ein npm-Skript aufrufbar sein.

```
npm run i18n:check

  ar.json: fehlend     course.lessonCount
  ar.json: überzählig  debug.placeholder
```

8. **Tests**

   Schreiben Sie mindestens drei automatisierte Tests. Vitest oder Jest für die Formatierungs- und Katalogprüfung, ein Werkzeug Ihrer Wahl für die Oberfläche. Mindestens ein Test bestätigt, dass `/ar/kurse` tatsächlich mit `dir="rtl"` ausgeliefert wird.

```
test("ar deckt alle Pluralkategorien ab", () => {
  const kat = new Intl.PluralRules("ar").resolvedOptions().pluralCategories;
  for (const k of kat) expect(ar.course.lessonCount).toContain(k);
});
```

3 Anforderungen

- Next.js mit App Router und TypeScript, `strict` aktiv.
- Kein Übersetzungstext im Quelltext der Komponenten. Jede sichtbare Zeichenkette kommt aus einem Katalog – auch Fehlermeldungen, `aria-label`-Werte und der Dokumenttitel.
- Keine Zeichenkette wird aus Teilen zusammengesetzt (`t("Sie haben") + n + t("Kurse")`). Solche Sätze lassen sich nicht übersetzen; nutzen Sie Platzhalter innerhalb einer Nachricht.
- Symbole ausschließlich aus **einem** Satz – Lucide oder Phosphor. Kein Mischen, keine Emojis als Bedienelemente.
- Eine eingebundene arabische Schriftart liegt lokal im Projekt, nicht auf einem fremden CDN.
- Die Anwendung startet mit `npm install` und `npm run dev` ohne weitere Handgriffe.

4 Abgabe

- Ein Git-Repository mit nachvollziehbaren Commits, nicht ein einzelner Commit „initial“ mit dem fertigen Projekt.
- Eine `README.md` mit Startanleitung (Voraussetzungen, Installation, Entwicklungsstart, Testlauf), der Format-Tabelle aus Punkt 5, Ihrer begründeten Entscheidung zu fehlenden Schlüsseln und einer Notiz, was Sie mit mehr Zeit anders gelöst hätten.
- Je ein Bildschirmfoto der Kursliste in `de` und in `ar`.
- `node_modules` und Bauartefakte gehören nicht ins Repository.

5 Bewertungskriterien

- **Sprachrouting** – Aushandlung beim Einstieg, saubere 404 bei unbekannter Sprache, `lang` und `dir` serverseitig gesetzt, Pfaderhalt beim Umschalten.
- **Pluralbehandlung** – alle sechs arabischen Kategorien bedient, keine selbstgebaute Fallunterscheidung nach `n === 1`.
- **Formatierung** – durchgängig über `Intl`, keine fest verdrahteten Trennzeichen, Datumsmuster oder Währungssymbole.
- **Schreibrichtung** – logische CSS-Eigenschaften statt Richtungsangaben, korrekt gespiegelte Symbole, Fortschrittsbalken in Leserichtung, gemischte Sätze ohne Richtungsbruch.
- **Fehlende Schlüssel** – begründete Strategie und ein Prüfskript, das die Lücke wirklich findet.
- **Katalogaufbau** – sinnvolle Gliederung, keine zusammengesetzten Sätze, vollständige Abdeckung sichtbarer Texte.
- **Tests** – prüfen sie Verhalten, und wird der RTL-Zustand von einem Test tatsächlich betreten?
- **Codequalität** – Typen, Struktur, Lesbarkeit, kein `any`.