Projektbeschreibung: liste.live - Mitbringliste Web-App

📋 Übersicht

liste.live ist eine Web-Anwendung zur Organisation von Mitbringlisten für Events, Partys und gemeinsame Veranstaltungen. Gäste können sich eintragen, was sie mitbringen möchten, und die Anzahl der begleitenden Personen angeben.

Projektziel: Entwicklung einer Single-Page-Application (SPA) mit React, die eine intuitive Benutzeroberfläche für die Verwaltung von Mitbringlisten bietet.

🎯 Lernziele

- **React-Grundlagen**: Komponenten, Props, State, Hooks
- **React Router**: Client-seitiges Routing und Navigation
- **Formular-Handling**: Validierung und Fehlerbehandlung
- **API-Integration**: REST-API Kommunikation mit fetch
- **State Management**: Lokaler State mit useState und useEffect
- **UI/UX Design**: Responsive Design mit Tailwind CSS
- **Fehlerbehandlung**: Graceful Error Handling und User Feedback

---

🔧 Technologie-Stack

Vorgegebene Technologien

- **Framework**: React 19
- **Build Tool**: Vite
- **Routing**: React Router v7
- **Styling**: Tailwind CSS v4 (Optional)
- **Sprache**: JavaScript (TypeScript optional)

Backend (bereits vorhanden)

- Express.js mit REST API
- Endpunkte sind dokumentiert und funktionsfähig

✨ Funktionale Anforderungen

1. Startseite (Home)

- Willkommensseite mit Branding "liste.live"
- Navigation zur Listenerstellung (Route 2)
- Kurze Erklärung des Konzepts
- Als Verbesserung: die Startseite könnte auch ein Input-Feld haben, um den Titel der neuen Liste direkt schon zu erfassen. Der Submit führt dann mit vorbereitetem Titel zur Route 2

2. Liste Erstellen

Funktionen:

- Formular mit folgenden Feldern:
  - **Titel** (Pflichtfeld)
  - **Beschreibung** (optional)
  - **E-Mail** (Pflichtfeld)
- Formular-Validierung (keine leeren Pflichtfelder)
- Absenden an API: `POST /api/lists`
- Nach Erstellung: Navigation zur Listen-Ansicht (Route 3)

3. Listen-Ansicht

Funktionen:

- Anzeige der Listen-Details (Titel, Beschreibung)
- Anzeige eines Shareable Link zum Kopieren (nur Listen-URL)
- **Beitragsformular**:
  - Name (Pflichtfeld)
  - Was wird mitgebracht? (Pflichtfeld)
  - Anzahl Personen (Pflichtfeld, numerisch)
  - Submit-Button
- **Liste aller Beiträge** in tabellarischer Ansicht:
  - Name | Was mitgebracht wird | Anzahl Personen
  - Responsive Design (mobile optimiert)
- **Zusammenfassung**: Gesamtzahl der Personen
- Nach Erstellen eines Beitrags: Weiterleitung zur Bearbeitungsansicht (Route 4)

4. Beitrags-Bearbeitungsansicht

Funktionen:

- **Prominenter Link-Bereich** (oberhalb des Formulars):
  - Hinweis-Kasten
  - Text: "☝️ Link zu deinem Beitrag:"
  - Hinweis: "💡 Speichere diesen Link, um deinen Beitrag später zu bearbeiten!"
- **Beitragsformular**:
  - Titel: "Beitrag bearbeiten"
  - Vorausgefüllte Felder mit bestehenden Daten
  - "Neuer Beitrag"-Button (wechselt zu Create-Modus)
  - Update-Button und Löschen-Button
- **Liste aller Beiträge** in tabellarischer Ansicht:
  - Name | Was mitgebracht wird | Anzahl Personen
  - Responsive Design (mobile optimiert)

    - Hervorgehobener eigener Beitrag

- **Zusammenfassung**: Gesamtzahl der Personen
- **Fehlerbehandlung**:
    - Ungültiger Beitrags-Key → Fehlermeldung + Weiterleitung zur Liste

---

Hinweise

- **Routes**: Ansichten 3 und 4 können zusammengefasst werden
- **Components**: Eine separate Komponente für das Beitragsformular könnte hilfreich sein
- **URL-State**: Nutzt den URL-State aus:
    - Welche Liste soll dargestellt werden?
    - Welcher Beitrag kann editiert werden

API-Endpunkte

Base URL: `http://localhost:3000`

Listen

Liste erstellen

```
POST /api/lists
Content-Type: application/json

{
  "title": "Geburtstag Anna",
  "description": "Was bringt ihr mit?",
  "email": "anna@example.com"
}

Response: 201 Created
{
  "id": 1,
  "key": "abc123",
  "title": "Geburtstag Anna",
  "description": "Was bringt ihr mit?",
  "email": "anna@example.com",
  "createdAt": "2024-01-01T12:00:00.000Z"
}
```

Liste abrufen

```
GET /api/lists/:key
```

```
Response: 200 OK
{
  "id": 1,
  "key": "abc123",
  "title": "Geburtstag Anna",
  "description": "Was bringt ihr mit?",
  "email": "anna@example.com",
  "submissions": [
    {
      "id": 1,
      "key": "def456",
      "submittedBy": "Max",
      "bringing": "Salat",
      "attendees": 2,
      "createdAt": "2024-01-01T12:30:00.000Z"
    }
  ]
}
```

Beiträge (Submissions)

Beitrag erstellen

```
POST /api/lists/:key/submissions
Content-Type: application/json

{
  "submittedBy": "Max",
  "bringing": "Salat für 8 Personen",
  "attendees": 2
}

Response: 201 Created
{
  "id": 1,
  "key": "def456",
  "submittedBy": "Max",
  "bringing": "Salat für 8 Personen",
  "attendees": 2,
  "createdAt": "2024-01-01T12:30:00.000Z"
}
```

Beitrag aktualisieren

```
PUT /api/submissions/:key
Content-Type: application/json

{
```

```
  "submittedBy": "Max Mustermann",
  "bringing": "Salat für 10 Personen",
  "attendees": 3
}

Response: 200 OK
{
  "id": 1,
  "key": "def456",
  "submittedBy": "Max Mustermann",
  "bringing": "Salat für 10 Personen",
  "attendees": 3,
  "updatedAt": "2024-01-01T13:00:00.000Z"
}
```

Beitrag löschen

```
DELETE /api/submissions/:id

Response: 204 No Content
```

🧩 Implementierungs-Schritte

Phase 1: Grundgerüst (Pflicht)

1. **Projekt-Setup**
   - Vite + React initialisieren
   - Tailwind CSS v4 installieren und konfigurieren
   - React Router v7 einrichten
2. **Basis-Routing**
   - Home-Seite
   - Create-List-Seite
   - List-View-Seite
3. **Layout-Komponente**
   - Header mit Logo/Branding
   - Pathless Layout Route

Phase 2: Listen-Verwaltung (Pflicht)

4. **Listen erstellen**
   - Formular
   - Validierung

- API-Integration
- Navigation nach Erfolg

5. **Listen anzeigen**
    - Listen-Details darstellen
    - Shareable Link anzeigen

Phase 3: Beiträge (Pflicht)

6. **Beiträge erstellen**
    - Formular
    - Validierung
    - API-Integration
    - Weiterleitung zur Bearbeitungsansicht
7. **Beiträge anzeigen**
    - Tabular List Design
    - Responsive Layout
    - Zusammenfassung (Personenzahl)

Phase 4: Bearbeitung (Pflicht)

8. **Beitrags-Bearbeitung**
    - Pre-filled Formular
    - Shareable Link anzeigen zum Beitrag anzeigen
    - Update-Funktionalität
    - Löschen-Funktionalität

Phase 5: Fehlerbehandlung (Optional)

10. **Error Handling**
    - Ungültige Listen-Keys
    - Ungültige Submission-Keys
    - API-Fehler abfangen

🎓 Bewertungskriterien

Funktionalität (70%)

- [ ] Alle Routes funktionieren korrekt
- [ ] Listen können erstellt und angezeigt werden
- [ ] Beiträge können erstellt, bearbeitet und gelöscht werden
- [ ] API-Integration funktioniert fehlerfrei
- [ ] Fehlerbehandlung implementiert

Code-Qualität (20%)

- [ ] Saubere Komponenten-Struktur
- [ ] Korrekte Hook-Verwendung (useState, useEffect)
- [ ] Code-Kommentare an komplexen Stellen

UI/UX (+20%)

- [ ] Design
- [ ] Intuitive Benutzerführung
- [ ] Toast-Benachrichtigungen implementieren
- [ ] Collapsible Forms funktional
- [ ] Visuelles Feedback bei Interaktionen

Best Practices (10%)

- [ ] ESLint ohne Fehler
- [ ] Sinnvolle Dateistruktur

🚀 Bonus-Features (Optional)

Erweiterte Funktionen

- [ ] **Loading States**: Spinner während API-Calls
- [ ] **Optimistic Updates**: UI-Update vor API-Response
- [ ] **Debouncing**: Bei Search/Filter-Funktionen
- [ ] **Local Storage**: Listen-Keys speichern
- [ ] **Export-Funktion**: Liste als PDF/CSV
- [ ] **Drag & Drop**: Beiträge sortieren
- [ ] **Dark Mode**: Theme-Switcher

Technische Verbesserungen

- [ ] **TypeScript**: Vollständige Typisierung
- [ ] **Testing**: Unit-Tests mit Vitest
- [ ] **Error Boundary**: React Error Boundaries
- [ ] **Accessibility**: ARIA-Labels, Keyboard-Navigation
- [ ] **Performance**: React.memo, useMemo optimizations

📚 Hilfreiche Ressourcen

Dokumentation

- **React**: https://react.dev
- **React Router**: https://reactrouter.com
- **Tailwind CSS**: https://tailwindcss.com
- **Vite**: https://vitejs.dev

Tutorials

- React Hooks: https://react.dev/reference/react
- Form Handling: https://react.dev/reference/react-dom/components/form
- Fetch API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Tipps

1. **Früh anfangen**: React-Setup kann knifflig sein
2. **Klein beginnen**: Erst Basis-Features, dann erweitern
3. **Häufig testen**: Im Browser während der Entwicklung
4. **Git nutzen**: Regelmäßige Commits, Feature-Branches
5. **Fragen stellen**: Bei Problemen nicht zu lange hängen bleiben
6. **Code Review**: Gegenseitig Code reviewen
7. **UX beachten**: Nicht nur Features, sondern Nutzererlebnis

---

Projekterfolg

Ein erfolgreiches Projekt zeigt:

- ☑ Verständnis von React-Grundkonzepten
- ☑ Fähigkeit zur API-Integration
- ☑ Saubere Komponenten-Architektur
- ☑ Responsive und intuitive UI
- ☑ Robuste Fehlerbehandlung
- ☑ Professionelle Code-Qualität

Viel Erfolg!