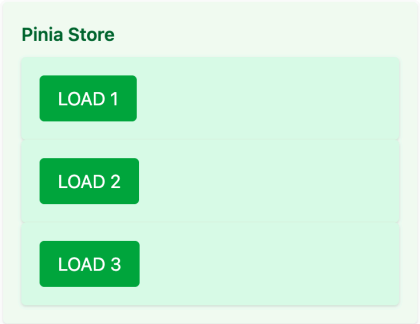
Hochschule **Rosenheim**
University of Applied Sciences

Vorlesung GUI

Übungsblatt 07: Parallelität und State Management

Ziel der Übung:
Du lernst, wie du Zustände in Vue teilst, wann Props oder provide/inject reichen – und wann ein Store wie Pinia nötig ist. Tailwind ist diesmal nicht nötig, du kannst die Komponente(n) aber gerne wieder stylen.



---

1. Theoriefragen

1. Was bedeutet „asynchrone Programmierung“ im Kontext von Web-UIs?
2. Warum kann eine blockierende Funktion die Benutzeroberfläche einfrieren?
3. Was ist ein Promise? Was bewirkt async und was await?
4. Wann ist Promise.all() nützlich? Wann Promise.race()?
5. Erkläre den Unterschied zwischen provide/inject und einem Store wie Pinia.

---

2. Projekt Setup

Setze ein typisches Vue.js Projekt auf und lösche unnötige Komponenten. Das kannst du bereits aus vorherigen Übungen.

---

3. Lokaler State in mehreren Komponenten

Ziel: Drei Komponenten verwalten unabhängig voneinander einen eigenen Ladezustand (loading).

Anleitung:

1. Lege die Komponentenhierarchie an:
   - App.vue lädt Parent.vue
   - Parent.vue lädt ChildOne.vue, ChildTwo.vue, ChildThree.vue

2. Jede Child*.vue-Komponente enthält:
   - Einen lokalen ref für loading
   - Eine Anzeige: „Lädt …“ oder „Bereit“
   - Einen Button „Ladevorgang starten“ → setzt loading = true für 2 Sekunden
   - Ein mit dem Text “loading” wird angezeigt während geladen wird

```
function startLoading() {
   loading.value = true
   setTimeout(() => {
     loading.value = false
   }, 2000)
 }
```

---

4. Gemeinsamer Ladezustand über Props (Props Drilling)

Ziel: Ein loading-State wird von Parent.vue zentral verwaltet und an alle Kinder weitergegeben.

Anleitung:

1. Entferne den lokalen loading-State in allen Child*.vue-Komponenten

2. Definiere in Parent.vue:
   - const loading = ref(false)
   - Eine Funktion startLoading() (setzt loading.value = true für 2 Sekunden)

3. Gib loading und startLoading als Props an die drei Child*.vue-Komponenten weiter

```
<ChildOne :loading="loading" :startLoading="startLoading" />
```

4. In Child*.vue:
    - Nutze defineProps() für loading und startLoading
    - Button klickt startLoading()

---

5. Zustand teilen mit provide / inject

Ziel: Vermeide Props-Drilling, indem loading zentral bereitgestellt und in den Kindern gelesen wird.

Anleitung:

1. In Parent.vue:
    - Importiere ref und provide
    - Definiere const loading = ref(false)
    - Verwende provide('loading', loading)
2. In Child*.vue:
    - Importiere inject
    - Verwende const loading = inject('loading')
    - Zeige Ladezustand wie zuvor an

---

6. Einführung von Pinia

Ziel: Ersetze provide/inject durch einen zentralen Store mit Lese- und Schreibzugriff.

Anleitung:

1. Installiere Pinia:

```
npm install pinia
```

2. Registriere Pinia in main.js:

```
const app = createApp(App)
app.use(createPinia())
app.mount('#app')
```

3. Erstelle src/stores/useAppStore.js:

```
import { defineStore } from 'pinia'

export const useAppStore = defineStore('app', {
 state: () => ({
   loading: false
 }),
 actions: {
   setLoading(val) {
     this.loading = val
   }
 }
})
```

4. In allen Komponenten (Child*.vue):

   - Entferne den inject der vorherigen Aufgabe
   - Importiere useAppStore
   - const app = useAppStore()
   - Nutze app.loading und app.setLoading(true) in deiner startLoading Funktion, sowie im <template>