

# Vorlesung GUI

## Übungsblatt 01: Setup

---

### Ziel der Übung:

In dieser Übung werden wir eine Buchsuche-Anwendung mit Vue.js erstellen. Wir werden reaktive Variablen, Zwei-Wege-Datenbindung, Computed Properties und das Styling mit TailwindCSS verwenden, um eine ansprechende und funktionale Benutzeroberfläche zu entwickeln. Am Ende binden wir eine echte Buch-API ein, um echte Daten abzurufen.

---

### 1. Theoriefragen

1. Was unterscheidet das MVVM-Modell von MVP und MVC in Bezug auf die Verbindung zwischen View und Logik?
  2. Welche Vorteile bietet die Trennung von View und Model in MVx-Modellen für die Wartbarkeit und Testbarkeit?
  3. Was ist eine computed property in Vue.js und wie unterscheidet sie sich von einer normalen Methode?
  4. Was ist der Unterschied zwischen Reaktivität und Zwei-Wege-Datenbindung in Vue.js?
  5. Warum ist der Presenter im MVP-Pattern potenziell problematisch in Bezug auf Komplexität?
- 

### 2. Projekt Setup

Setze ein typisches Vue.js Projekt mit Tailwind CSS auf. Das kannst du bereits.

---

### 3. Book Search

Die meisten Code-Stücke könnt ihr in den Vorlesungsfolien finden.

#### Basis Template

- Passe dein App.vue template an, sodass es nur noch die HelloWorld Komponente lädt
- Benenne die HelloWorld Komponente um zu "BookSearch"
- Fülle die Datei zunächst mit einem leeren Template/Script

#### Bücher als reaktive Variable anlegen und anzeigen

- Erstelle eine reaktive Variable books, die die Liste der Bücher hält
- Liste die Bücher mit einem v-for in deinem template auf

### Suchfeld hinzufügen

- Erstelle eine reaktive Variable `searchTerm`, die den Suchbegriff hält
- Füge dem Template in `<input>` Feld hinzu und verknüpfe es mit v-model zu der `searchTerm` Variable

### Bücher durchsuchen

- Lege eine Computed Property `filteredBooks` an, das die Bücher, anhand des Suchbegriffs, gefiltert anzeigt
- Anstatt über `books` zu iterieren, iteriere nun über `filteredBooks`

## 4. Styling mit TailwindCSS

### a) Container und Layout

Füge dem Hauptcontainer grundlegendes Padding und zentriere den Inhalt:

```
<template>
  <div class="container mx-auto p-4">
    ...
  </div>
</template>
```

- `container`: Zentriert den Inhalt.
  - `mx-auto p-4`: Setzt das Padding und zentriert den Container.
- 

### b) Suchfeld stylen

Stelle sicher, dass das Suchfeld gut aussieht und gut fokussiert wird:

```
<input
  ...
  class="border p-2 rounded-lg w-full shadow-lg focus:ring-2
focus:ring-blue-500"
  ...
/>
```

- `border p-2 rounded-lg`: Fügt Rahmen und abgerundete Ecken hinzu.
- `w-full`: Macht das Eingabefeld 100% der Breite.
- `shadow-lg focus:ring-2`: Fügt Schatten und Fokus-Effekte hinzu.

---

### c) Buchliste stylen

Styling der Buchliste, um sie visuell abzuheben:

```
<ul class="mt-4 space-y-2">
  <li ... class="p-4 bg-white border border-gray-300 rounded-lg
shadow-md hover:bg-gray-100">
    ...
  </li>
</ul>
```

- `p-4 bg-white border`: Fügt Padding, Hintergrund und Rahmen hinzu.
- `rounded-lg shadow-md`: Abrundung und Schatten für jedes Listenelement.
- `hover:bg-gray-100`: Verändert den Hintergrund bei Hover.

## 5. BONUS - Wir binden eine echte API an

Anstatt die Bücher selbst zu definieren:

- lege eine leere reaktive Variable für books an
- übernahm untenstehende fetchBooks Funktion
- lade die Bücher über "onMounted" beim ersten Rendern

```
const books = ref([]);

const fetchBooks = async () => {
  const response = await
  fetch('https://www.googleapis.com/books/v1/volumes?q=random&fields=items/volumeInfo
/title');
  const data = await response.json();
  books.value = data.items.map(item => ({
    title: item.volumeInfo.title
  }));
};

onMounted(fetchBooks); // Lädt die Bücher beim ersten Rendern
```