



Vorlesung 2

03 Fortsetzung Grundlagen

- 3.1. Syntax (Text Interpolation, Reaktivität, Direktiven)
- 3.2. Event Handling
- 3.3. Components & Props
- 3.4. Forms & v-model

04 Theorie Nuxt

- 4.1. Was ist Vite ?
- 4.2. Was ist Nitro ?
- 4.3. Wie wird gerendert ?
- 4.4. Setup & Einstieg in die App



List Rendering

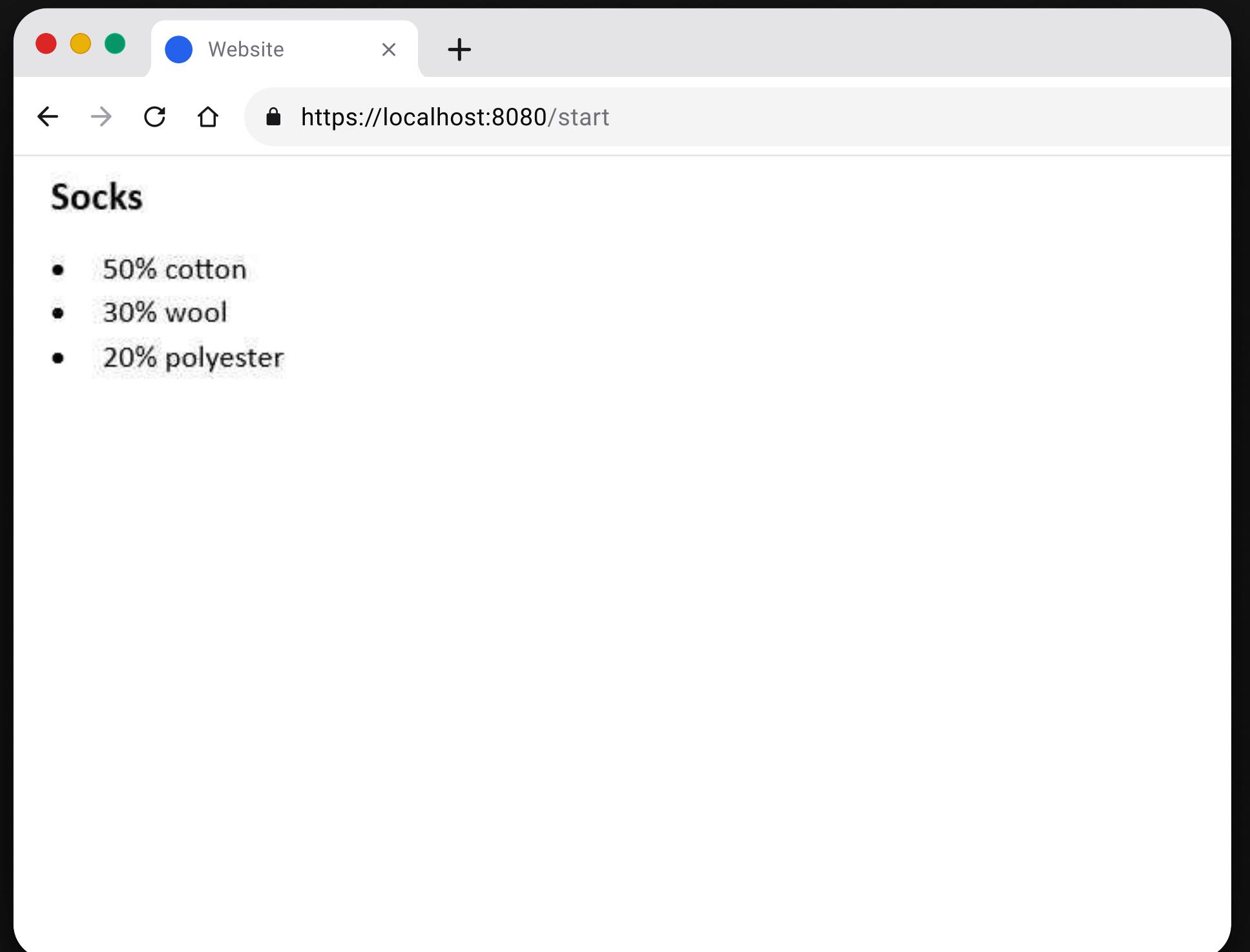
v-for

- **Funktion:** V-for ist eine Vue-Direktive, die ein Element für jedes Objekt oder jeden Wert in einer Liste wiederholt.

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from './assets/images/socks_green.jpeg'

const product = ref('Socks')
const details = ref(['50% cotton', '30% wool', '20% polyester'])
</script>

<template>
  <h1>{{ product }}</h1>
  <ul>
    <li v-for="detail in details">{{ detail }}</li>
  </ul>
</template>
```





List Rendering

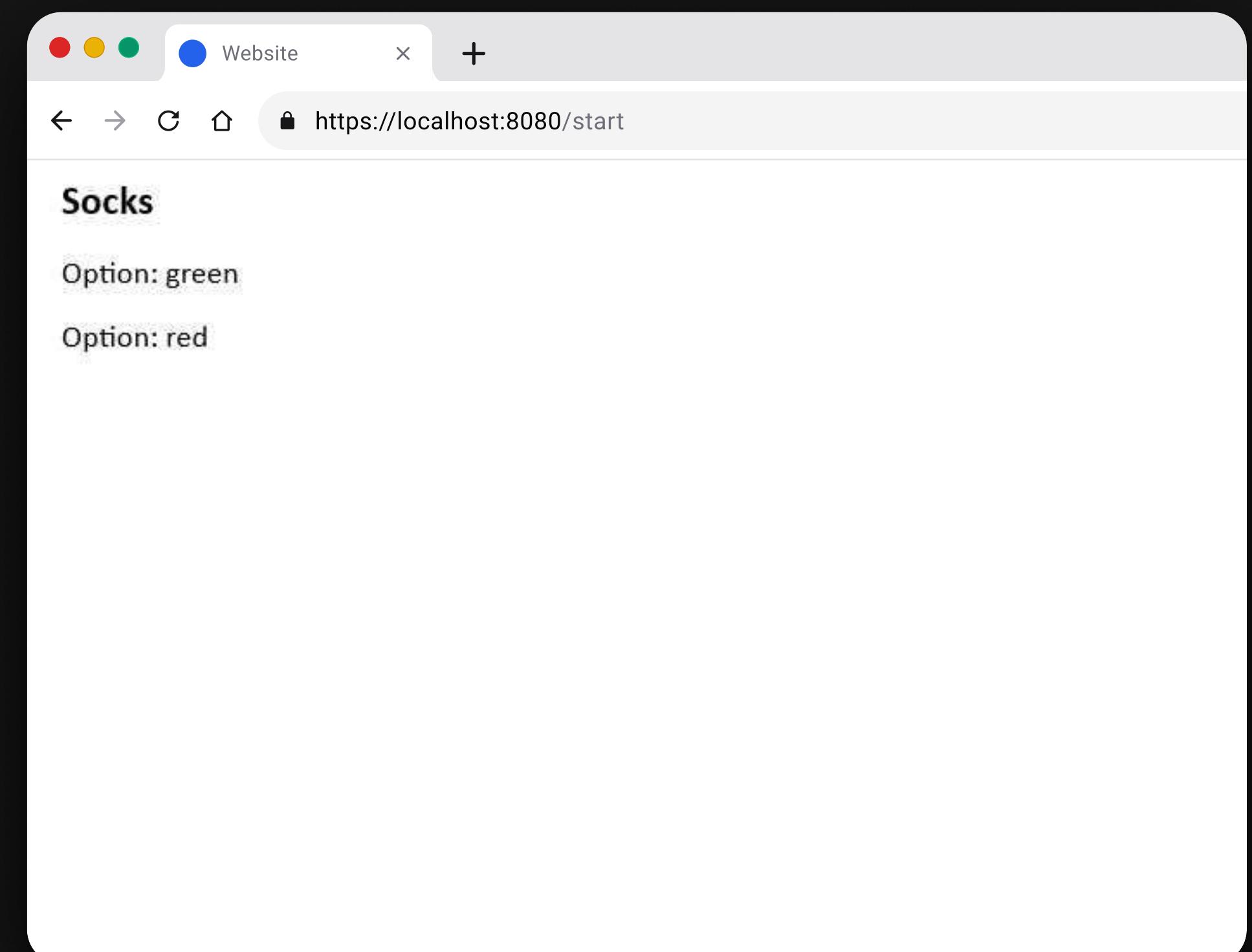
v-for

- **Funktion:** V-for ist eine Vue-Direktive, die ein Element für jedes Objekt oder jeden Wert in einer Liste wiederholt.

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from './assets/images/socks_green.jpeg'

const product = ref('Socks')
const options = ref([
  { id: 123, color: 'green' },
  { id: 456, color: 'red' }
])
</script>

<template>
  <h1>{{ product }}</h1>
  <div v-for="option in options" :key="option.id">
    Variante: {{ option.color }}
  </div>
</template>
```





Aufgabe 05

10 min

05 – List Rendering

Ziel

In dieser Übung lernst du, wie du mit der Vue-Direktive `v-for` Daten aus einer Liste dynamisch im Template renderst.

Ausgangspunkt

In deinem Profil gibt es bereits den Post-Bereich (`.posts`), der bisher nur anzeigt: Noch keine Beiträge vorhanden im Skript ist nun eine Liste von Posts vorbereitet. Jeder Post besitzt folgende Eigenschaften:

Eigenschaft	Beschreibung	Beispielwert
<code>id</code>	Eindeutige ID	1
<code>title</code>	Titel des Beitrags	"Campus bei Nacht"
<code>likes</code>	Anzahl der Likes	42
<code>imageUrl</code>	Bildpfad oder URL	"https://..."

⌚ Aufgabe

1. Bedingte Anzeige:

- o Wenn `posts` leer ist, soll ein Text erscheinen: „Noch keine Beiträge vorhanden“
- o Wenn `posts` Einträge enthält, soll dieser Hinweis verschwinden und stattdessen die Galerie mit den Posts erscheinen.

2. Rendering mit `v-for`:

- o Iteriere mit `v-for` über das `posts`-Array.
- o Verwende für jedes Element einen eindeutigen Schlüssel (`:key="post.id"`).

3. Verwende folgende CSS-Klassen, um die Galerie korrekt darzustellen:

- o `posts` – umschließt den gesamten Post-Bereich
- o `post-list` – Container für die gesamte Post-Galerie (Grid)
- o `post-card` – einzelne Post-Kachel
- o `post-info` – Bereich für Titel und Like-Zahl im Overlay
- o `title` – Titeltext im Overlay
- o `likes` – Like-Anzeige mit Herzsymbol (CSS-Icon)

Event Handling

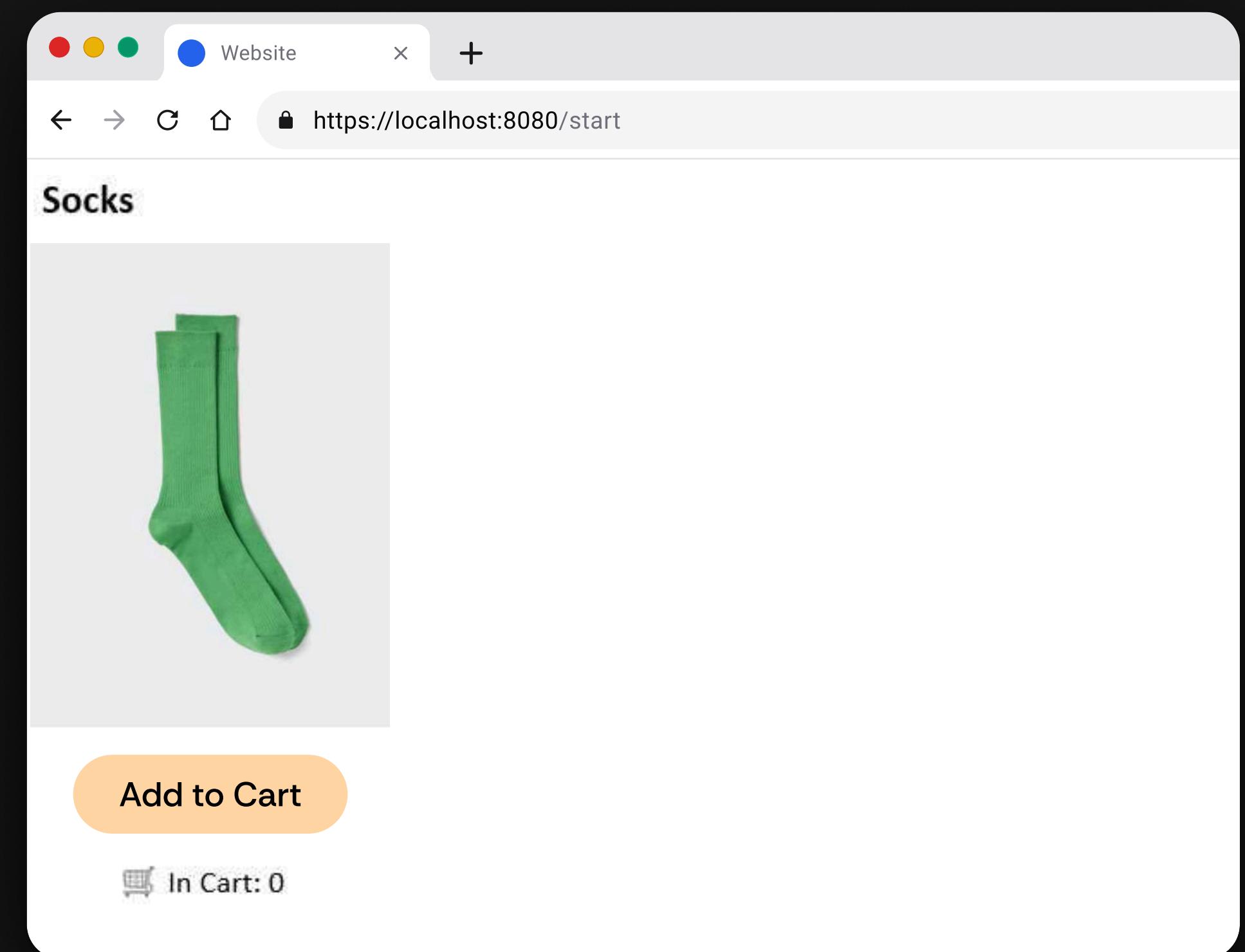
v-on

- **Funktion:** Verknüpft Benutzeraktionen mit Funktionen / Reaktion auf Events.
- **Kurzschrifweise:** z.B.: <button @click="cart += 1">

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from './assets/images/socks_green.jpeg'

const product = ref('Socks')
const image = ref(socksGreenImage)
const cart = ref(0)
</script>

<template>
  <h1>{{ product }}</h1>
  
  <button class="button" v-on:click="cart += 1">Add to Cart</button>
  <div class="cart-display">
    <span> In Cart: {{ cart }}</span>
  </div>
</template>
```



Event Handling

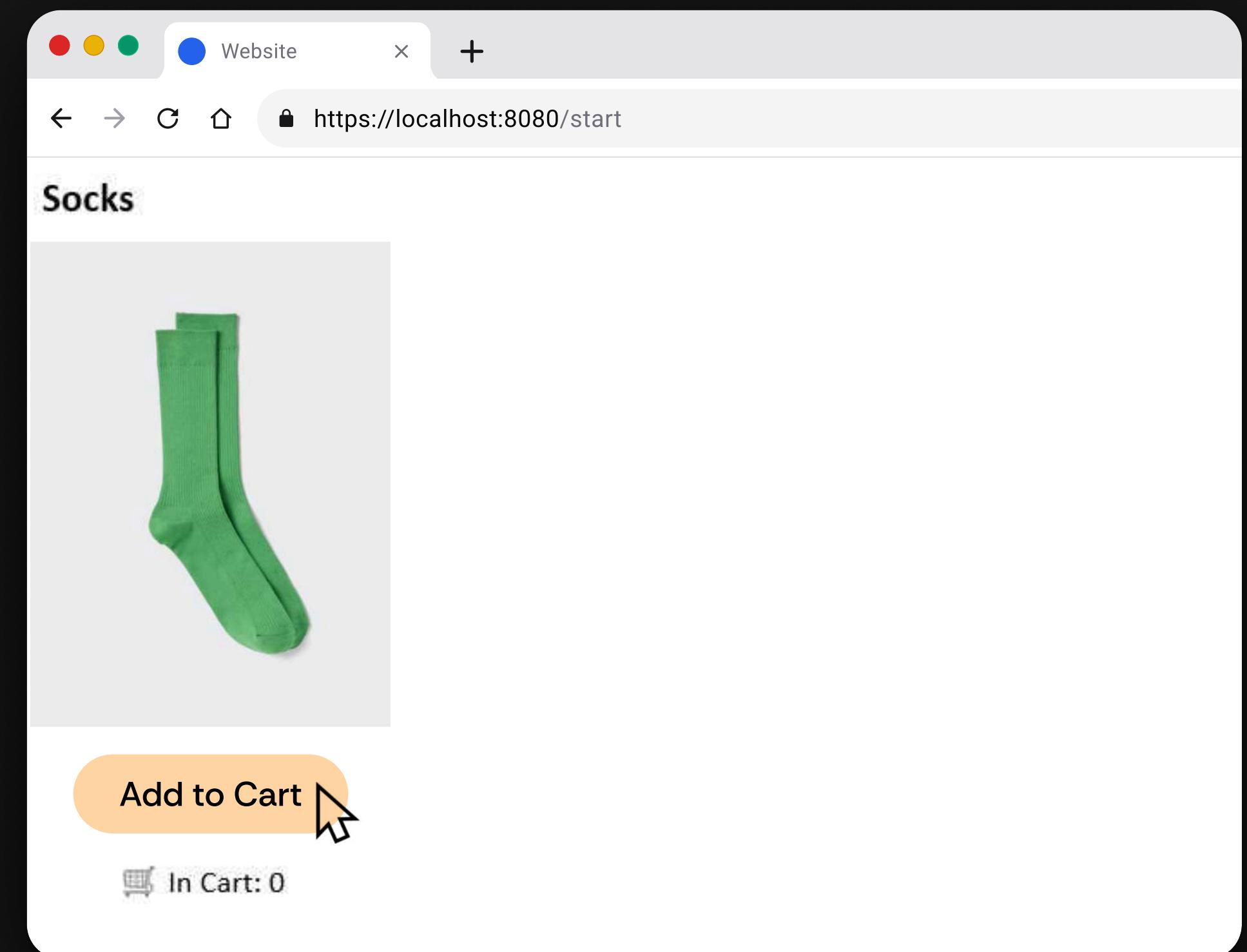
v-on

- **Funktion:** Verknüpft Benutzeraktionen mit Funktionen / Reaktion auf Events.
- **Kurzschrifweise:** z.B.: <button @click="cart += 1">

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from './assets/images/socks_green.jpeg'

const product = ref('Socks')
const image = ref(socksGreenImage)
const cart = ref(0)
</script>

<template>
  <h1>{{ product }}</h1>
  
  <button class="button" v-on:click="cart += 1">Add to Cart</button>
  <div class="cart-display">
    <span> In Cart: {{ cart }}</span>
  </div>
</template>
```





Event Handling

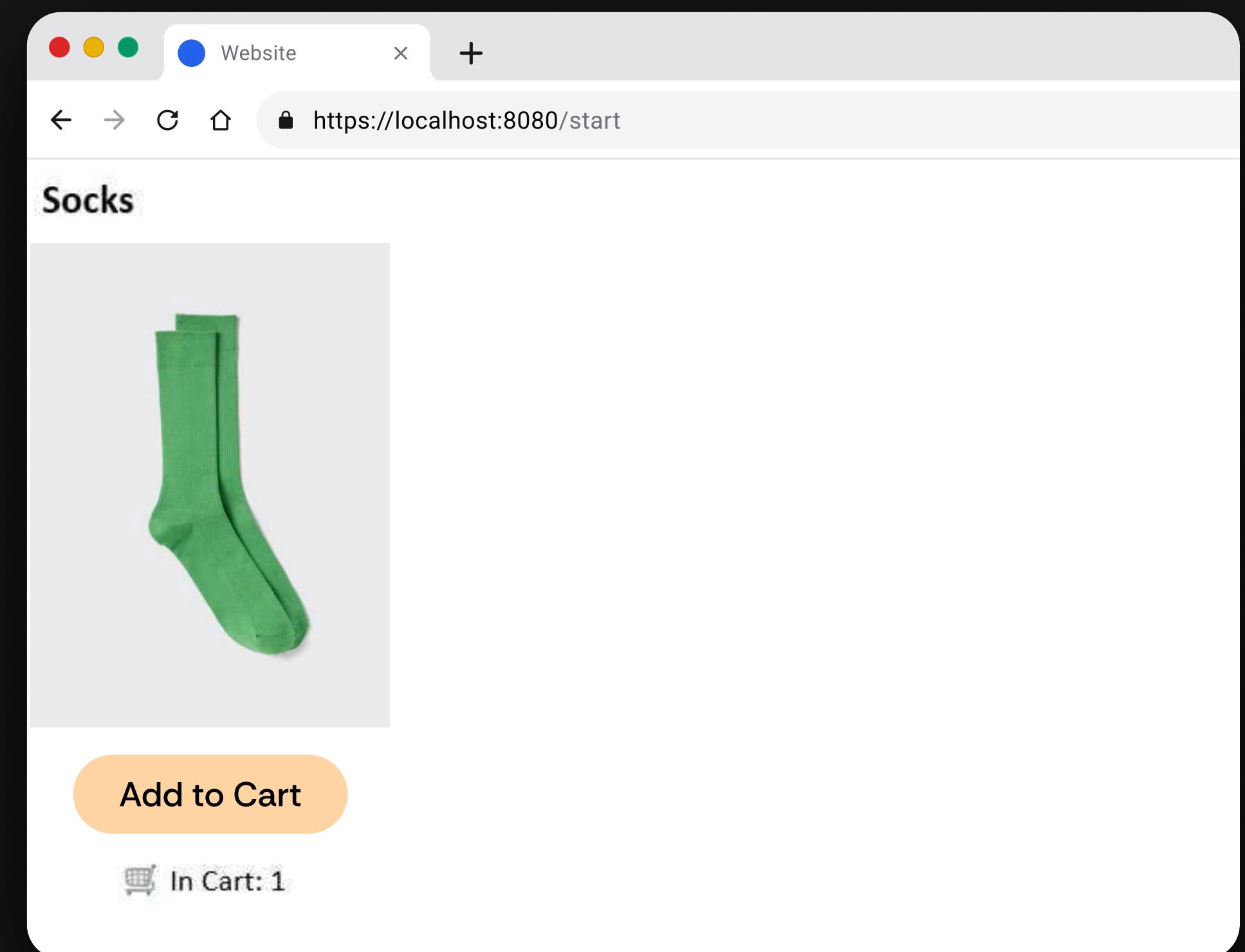
v-on

- **Funktion:** Verknüpft Benutzeraktionen mit Funktionen / Reaktion auf Events.
- **Kurzschrifweise:** z.B.: <button @click="cart += 1">

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from './assets/images/socks_green.jpeg'

const product = ref('Socks')
const image = ref(socksGreenImage)
const cart = ref(0)
</script>

<template>
  <h1>{{ product }}</h1>
  
  <button class="button" v-on:click="cart += 1">Add to Cart</button>
  <div class="cart-display">
    <span> In Cart: {{ cart }}</span>
  </div>
</template>
```





Events

Beispiele:

Event	Beschreibung	Beispiel
click	Wird ausgelöst, wenn ein Element gedrückt wird.	@click="doSomething"
dblclick	Doppelklick	@dblclick="zoomIn"
mouseover	Mauszeiger bewegt sich über ein Element.	@mouseover="highlight"
mouseout	Mauszeiger verlässt ein Element.	@mouseout="removeHighlight"
mousemove	Maus bewegt sich über das Element.	@mousemove="trackMouse"



Event Modifier

- **Funktion:** kleine Erweiterungen hinter einem Event (mit einem Punkt .), die das Verhalten eines Events verändern, ohne JavaScript schreiben zu müssen

Modifier	Beschreibung
.prevent	Verhindert das Standardverhalten des Browsers (z. B. Seiten-Reload bei Formularen)
.self	Event wird nur ausgelöst, wenn das Element selbst angeklickt wird – nicht, wenn ein Kind-Element darin angeklickt wird.
.once	Event wird nur ein einziges Mal ausgeführt.
.stop	Stoppt die Event-Weitergabe an übergeordnete Elemente.



Aufgabe 06

10 min



06 – Event Handling

Ziel

In dieser Übung lernst du, wie man auf Benutzeraktionen (Events) reagiert und mit Methoden arbeitet. Du implementierst die Logik, um auf Klick einen Beitrag groß in der Mitte des Bildschirms anzuzeigen.

Ausgangspunkt

Deine Anwendung zeigt bereits eine Galerie mit mehreren Posts. In dieser Übung ist zusätzlich ein Container für die Bildvorschau (Post-Overlay) vorbereitet. Dieser Container ist unsichtbar, bis du ihn über JavaScript aktivierst.

Die CSS- und HTML-Struktur ist also schon da – du sollst **nur die Logik** implementieren.

Aufgabe

1. Erstelle eine neue reaktive Variable `selectedPost` mit `ref(null)`.
 - o Diese Variable soll das aktuell angeklickte Post-Objekt speichern.
2. Schreibe eine Funktion `openPost(post)`, die:
 - o das angeklickte Post-Objekt in `selectedPost` speichert.
3. Füge bei jedem Post in der Liste das Click-Event hinzu
4. Die Anzeige des Overlays ist bereits an `selectedPost` gebunden.
5. Implementiere außerdem die Funktion `closePost()`,
 - o die `selectedPost` wieder auf `null` setzt.

Wichtig: Der Overlay-Container und sein Stil sind schon vorhanden — du musst nichts daran ändern. Du sollst nur die Logik schreiben, um ihn zu öffnen und zu schließen.

Challenge

- Füge später eine zusätzliche Logik hinzu, sodass man auch durch Klicken außerhalb des Bildes das Overlay schließen kann.



Components & Props

- **Funktion:** Strukturieren den Code in wiederverwendbare, eigenständige Bausteine.

App.vue

```
<script setup>
import { ref } from 'vue'
import ProductDisplay from '@components/ProductDisplay.vue'

const shipping = ref('2,99€')
</script>

<template>
  <ProductDisplay :shipping="shipping" ></ProductDisplay>

  <div class="cart-display">
    🛍 In Cart: {{ cart }}
  </div>
</template>
```

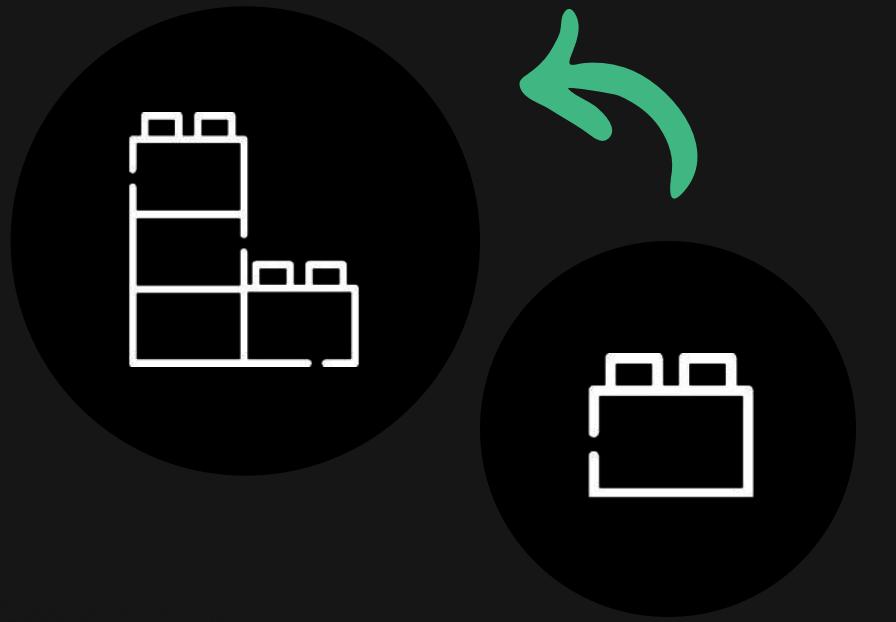
ProductDisplay.vue

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from '@assets/images/socks_green.jpeg'

const props = defineProps({
  shipping: {
    type: String,
    required: true
  }
})

const product = ref('Socks')
const image = ref(socksGreenImage)
</script>

<template>
  <h1>{{ product }}</h1>
  
  <p>Shipping: {{ props.shipping }}</p>
</template>
```





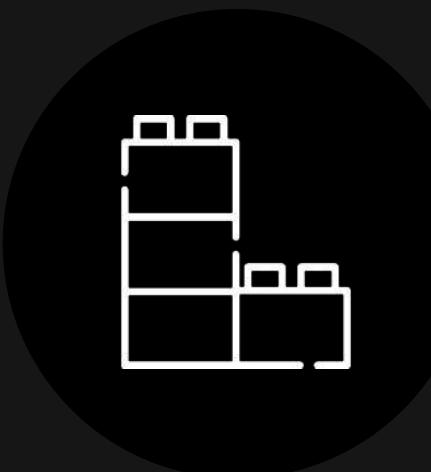
Components & Props

Fall 1: Übergabe von Daten an die aufgerufene Komponente. → Datenaustausch mithilfe von Props

→ (Parent-)Komponente über gibt Werte als Props

→ (Child-)Komponente empfängt diese mit defineProps()

(Parent-)Komponente



(Child-)Komponente



Props





Components & Props

App.vue

```
<script setup>
import { ref } from 'vue'
import ProductDisplay from '@components/ProductDisplay.vue'

const shipping = ref('2,99€')
</script>

<template>
  <ProductDisplay :shipping="shipping" ></ProductDisplay>

  <div class="cart-display">
    In Cart: {{ cart }}
  </div>
</template>
```

ProductDisplay.vue

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from '@assets/images/socks_green.jpeg'

const props = defineProps({
  shipping: {
    type: String,
    required: true
  }
})

const product = ref('Socks')
const image = ref(socksGreenImage)
</script>

<template>
  <h1>{{ product }}</h1>
  
  <p>Shipping: {{ props.shipping }}</p>
</template>
```

Components & Props

Fall 2: Die (Child-)Komponente sendet ein Ereignis an die (Parent-)Komponente

- in (Child-)Komponente wird Event mit defineEmits() definiert und ausgelöst
- (Parent-)Komponente empfängt Event über ein Listener-Attribut (@...)





Components & Props

App.vue

```
<script setup>
import { ref } from 'vue'
import ProductDisplay from '@components/ProductDisplay.vue'

const cart = ref(0)

const updateCart = () => {
  cart.value +=1
}
</script>

<template>
  <ProductDisplay @add-to-cart="updateCart" ></ProductDisplay>

  <div class="cart-display">
    In Cart: {{ cart }}
  </div>
</template>
```

ProductDisplay.vue

```
<script setup>
import { ref } from 'vue'
import socksGreenImage from '@assets/images/socks_green.jpeg'

const emit = defineEmits(['add-to-cart'])

const product = ref('Socks')
const image = ref(socksGreenImage)

const addToCart = () => {
  emit('add-to-cart')
}
</script>

<template>
  <h1>{{ product }}</h1>
  
  <button class="button" v-on:click="addToCart">Add to Cart</button>
</template>
```





Aufgabe 07

15 min

07 – Components and Props

Ziel

In dieser Übung lernst du, wie man wiederverwendbare Komponenten erstellt und mit Props und Events zwischen Parent und Child kommuniziert. Du lagerst den Overlay-Container, der beim Klick auf einen Post erscheint, in eine eigene Komponente aus.

Ausgangspunkt

In der App ist aktuell noch der Overlay-Container direkt in `App.vue` eingebaut. Dieser zeigt das ausgewählte Bild (`selectedPost`) und enthält den Schließen-Button.

Deine Aufgabe ist es nun, diesen Code in eine neue Datei zu verschieben.

Aufgabenstellung

1. Erstelle eine neue Komponente im Ordner `components/` → Dateiname: `PostOverlay.vue`
2. Kopiere den HTML-Code des Overlay-Containers aus `App.vue` (alles zwischen `<div v-if="selectedPost" class="overlay"> ... </div>`)
3. Erstelle Props für die Komponente:
 - o `post` – das aktuell ausgewählte Post-Objekt
 - o `avatarImageRef` – das Profilbild
 - o `username` – den Namen des Accounts
4. Erstelle ein Custom-Event, um das Overlay zu schließen:
 - o `defineEmits(['close'])`
 - o Der Close-Button und der Klick auf den Hintergrund sollen das Event auslösen.
5. Binde das neue Event und die Props in `App.vue` ein:
 - o Importiere `PostOverlay`
 - o Verwende sie nur noch so:

```
<PostOverlay  
  v-if="selectedPost"  
  :post="selectedPost"  
  :avatarImageRef="avatarImageRef"  
  :username="username"  
  @close="closePost"  
/>
```

6. Entferne den alten Overlay-Code aus `App.vue`. Nur die neue Komponente soll jetzt die Anzeige übernehmen.

Forms & v-model

v-model

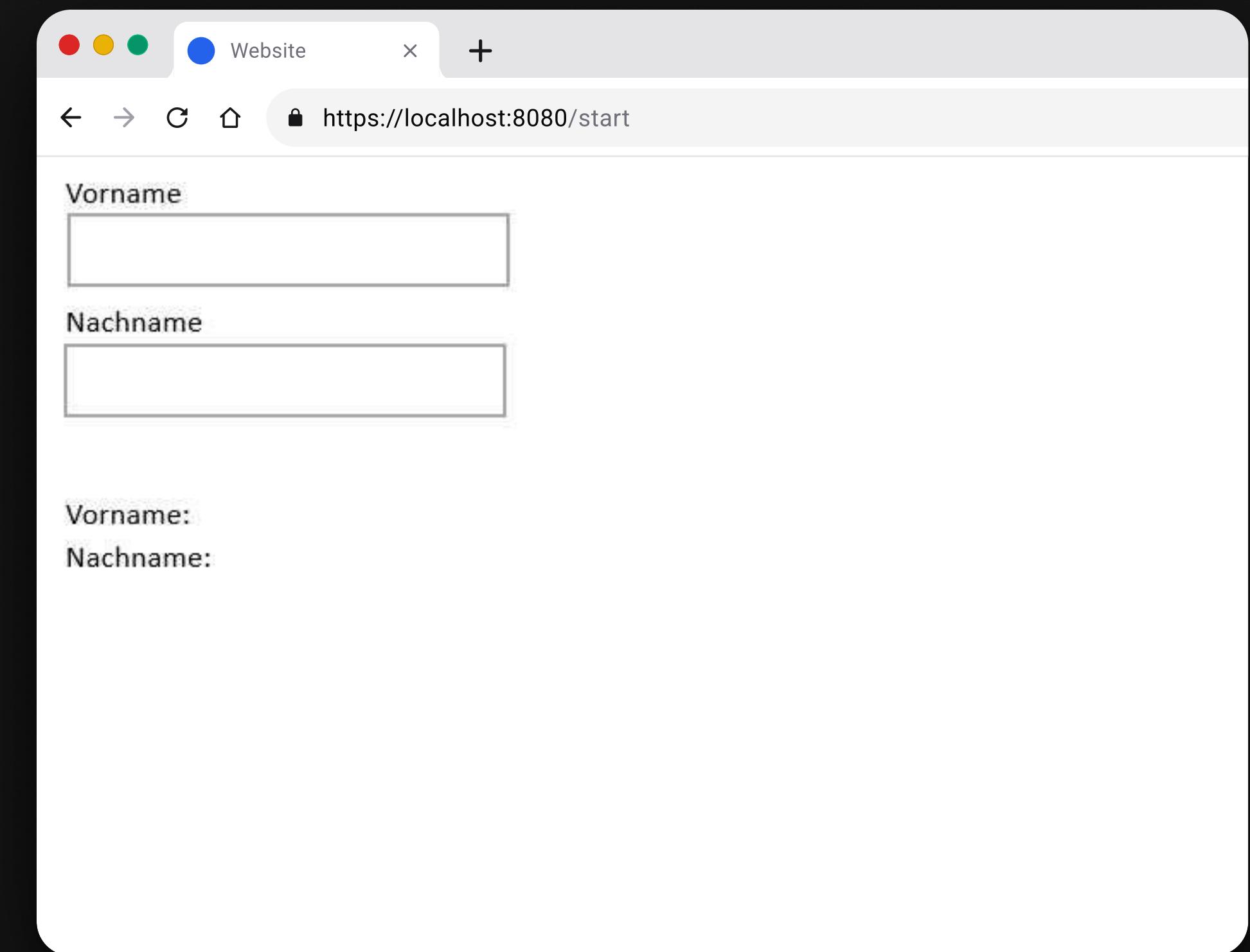
```
<script setup>
import { ref } from 'vue'

const firstname = ref('')
const lastname = ref('')

</script>

<template>
  <input v-model="firstname" placeholder="Vorname" />
  <input v-model="lastname" placeholder="Nachname" />

  <p>Vorname: {{ firstname }}</p>
  <p>Nachname: {{ lastname }}</p>
</template>
```





Forms & v-model

v-model

```
<script setup>
import { ref } from 'vue'

const firstname = ref('')
const lastname = ref('')

</script>

<template>
  <input v-model="firstname" placeholder="Vorname" />
  <input v-model="lastname" placeholder="Nachname" />

  <p>Vorname: {{ firstname }}</p>
  <p>Nachname: {{ lastname }}</p>
</template>
```



Vorname
Max

Nachname
Mustermann

Vorname:
Nachname:



Forms & v-model

v-model

```
<script setup>
import { ref } from 'vue'

const firstname = ref('')
const lastname = ref('')

</script>

<template>
  <input v-model="firstname" placeholder="Vorname" />
  <input v-model="lastname" placeholder="Nachname" />

  <p>Vorname: {{ firstname }}</p>
  <p>Nachname: {{ lastname }}</p>
</template>
```



Vorname
Max

Nachname
Mustermann

Vorname: Max

Nachname: Mustermann



Aufgabe 08

10 min



08 – Forms & v-model

Ziel

In dieser Übung lernst du, wie du Formulare mit `v-model` in Vue erstellst und Benutzereingaben reaktiv anzeigen. Du ergänzt das Post-Overlay um ein Kommentarformular, bei dem neue Kommentare direkt im Kommentarbereich erscheinen.

Ausgangspunkt

In deiner App existiert bereits:

- Ein Post-Overlay (`PostOverlay.vue`), das Bild, Titel und Likes eines Beitrags anzeigt.
- Ein Kommentarbereich (`<div class="comments">`), in dem die Kommentare erscheinen sollen.
- Das Formular mit einem Eingabefeld und Button ist im Template schon enthalten.
- Die CSS-Styles und Grundlogik (Array `comments`, Funktion `addComment()`) sind bereits vorbereitet.

Aufgabe

1. Lege eine neue reaktive Variable an, die das aktuelle Texteingabefeld des Benutzers speichert. Diese Variable soll automatisch mit dem Eingabefeld verbunden werden.
2. Verbinde das Eingabefeld mit dieser Variable:
 - a. Verwende dazu die Direktive `v-model` im `<input>`-Element.
3. Implementiere die Logik in der Funktion `addComment()`:
 - a. Wenn der eingegebene Kommentar nicht leer ist:
 - Füge den Wert in das Array `comments` ein.
 - Leere danach das Eingabefeld, damit ein neuer Kommentar geschrieben werden kann.
4. Sorge dafür, dass im Template die Kommentare automatisch aktualisiert angezeigt werden.

Challenge

Erweitere das Formular später so, dass:

- der Kommentar auch per Enter-Taste abgeschickt werden kann (`@keyup.enter="addComment"`).
- der Kommentartext automatisch getrimmt wird (keine Leerzeichen am Anfang/Ende).



Vorlesung 2

03 Fortsetzung Grundlagen

- 3.1. Syntax (Text Interpolation, Reaktivität, Direktiven)
- 3.2. Event Handling
- 3.3. Components & Props
- 3.4. Forms & v-model

04 Theorie Nuxt

- 4.1. Was ist Vite ?
- 4.2. Was ist Nitro ?
- 4.3. Wie wird gerendert ?
- 4.4. Setup & Einstieg in die App



Was ist Vite?

- Build-Tool für schnelleres Entwickeln von modernen Web-Applikationen
- Besteht aus Entwicklungs- und Produktionsmodus
- Entwicklungsmodus:
 - Kompiliert nicht ganzes Projekt, sondern nur das, was du gerade ansiehst
 - Hot Reload → nur geänderte Module werden neu geladen
- Produktionsmodus erstellt Rollup um Module zu bündeln



Einfache Entwicklung



Performance



Was ist Nitro?

- Server-Engine von Nuxt
- verantwortlich für Server-Side Rendering (SSR), API-Routen und Serverfunktionen
- Läuft in jeder Umgebung
- Ermöglicht serverseitige Logik direkt im Nuxt-Projekt
- Unterstützt Dateibasiertes Routing für APIs
- Nicht als Voll-Backend gedacht (Serverless)

+ SSR

+ API-Routing



Wie wird gerendert?

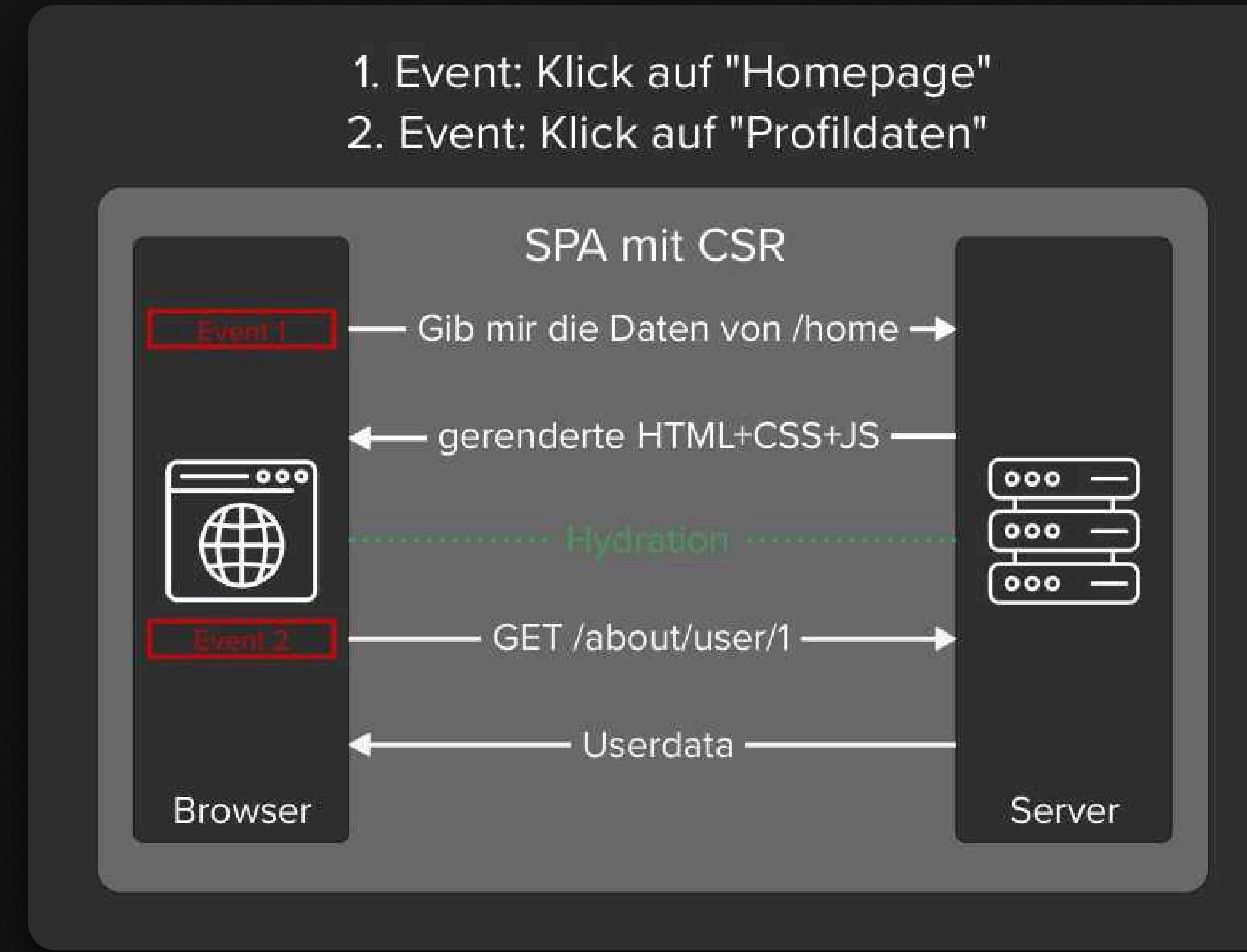
- SPA + SSR hybrid: Single Page App, die Server-Side Rendering nutzt
- Nitro rendert HTML auf dem Server für die erste Anfrage
- Nach dem Laden übernimmt Vue die Seite (Hydration)
- Navigation zwischen Seiten ohne Reload

- + SEO
- + Schnelles Laden
- + Automatisiert

“Früheres” Rendering



Nuxt Rendering



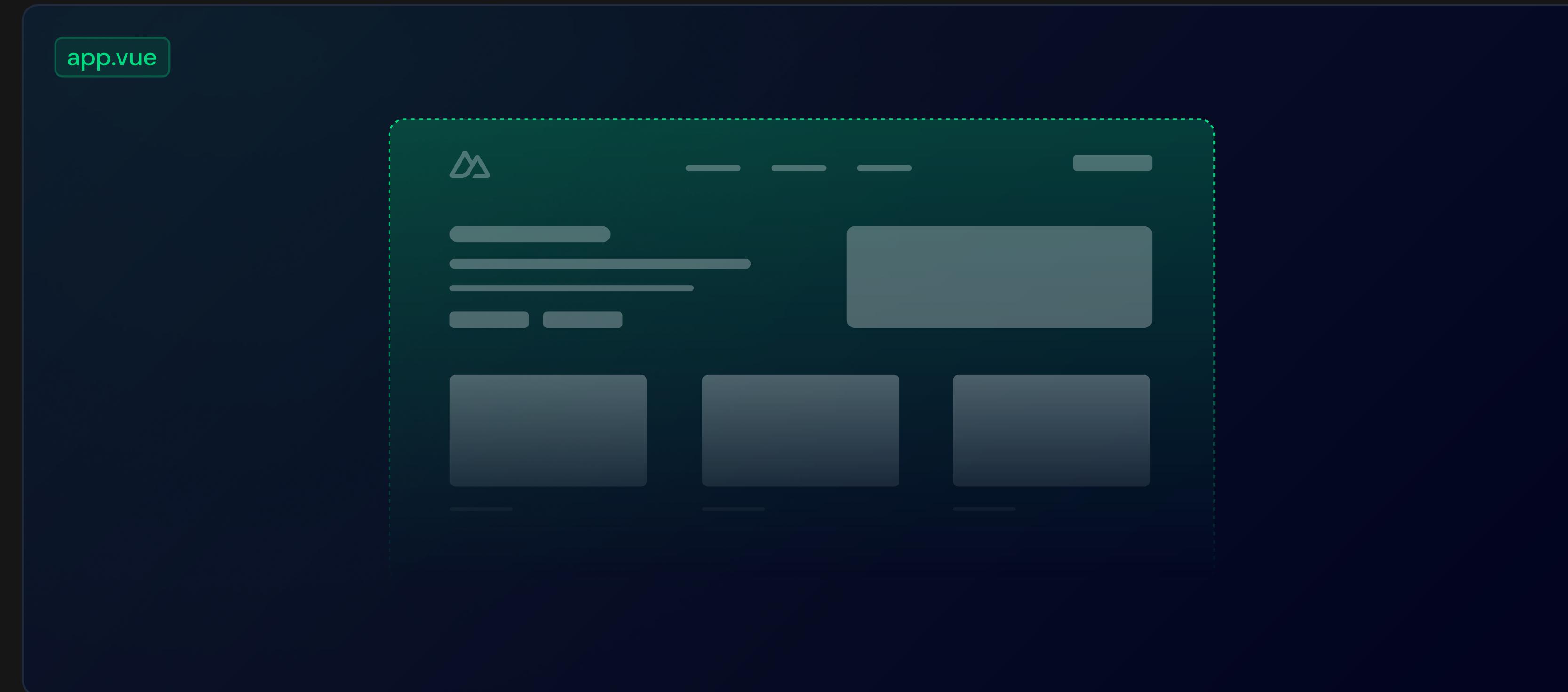


Ordnerstruktur

```
my-nuxt-app/
|
|   .nuxt/          # Automatisch generiert (Build-Output, Cache, Routen, usw.)
|   node_modules/    # Abhängigkeiten
|   public/          # Statische Dateien (z.B. favicon.ico, images, robots.txt)
|   app/
|     assets/        # Unkompilierte Ressourcen (z.B. Fonts, Bilder für Komponenten)
|     components/    # Globale Vue-Komponenten
|     composables/   # Reaktive Hilfsfunktionen (z.B. useAuth(), useFetchData())
|     layouts/        # Seiten-Layouts (z.B. default.vue, admin.vue)
|     middleware/    # Middleware für Routen (z.B. Auth-Check)
|     pages/          # Definiert automatisch das Routing-System
|     plugins/        # Client-/Server-Plugins (z.B. Axios, Pinia, i18n)
|     utils/          # Allgemeine Hilfsfunktionen
|     app.vue         # Haupteinstiegspunkt
|     app.config.ts   # API-Endpunkte und Server-Routen (Nuxt Server Engine)
|
|   nuxt.config.ts    # Hauptkonfigurationsdatei (Plugins, Build, RuntimeConfig)
|   package.json      # Projektabhängigkeiten
|   server            # API-Endpunkte und Server-Routen
```

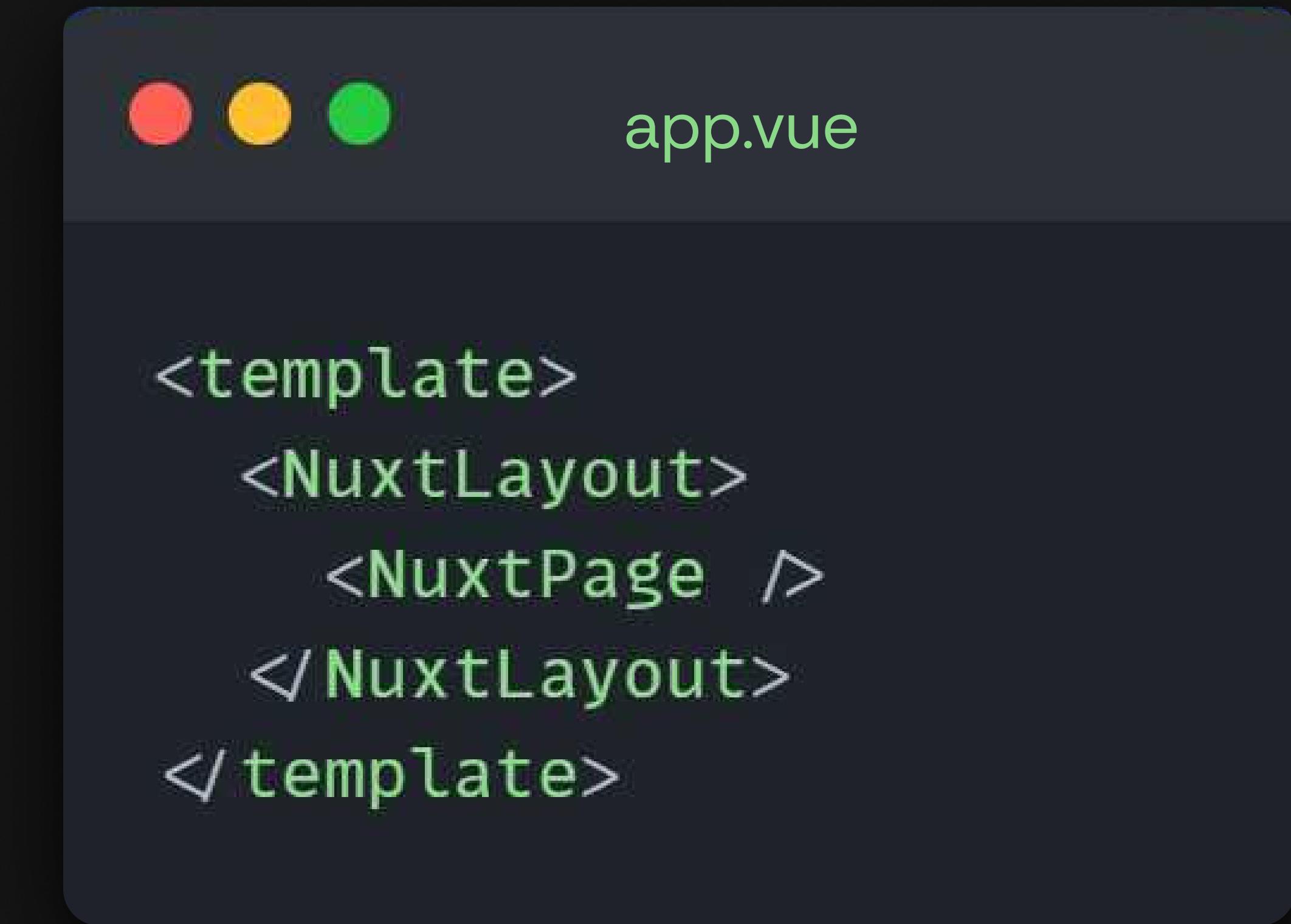


Ordnerstruktur - App





Ordnerstruktur - App

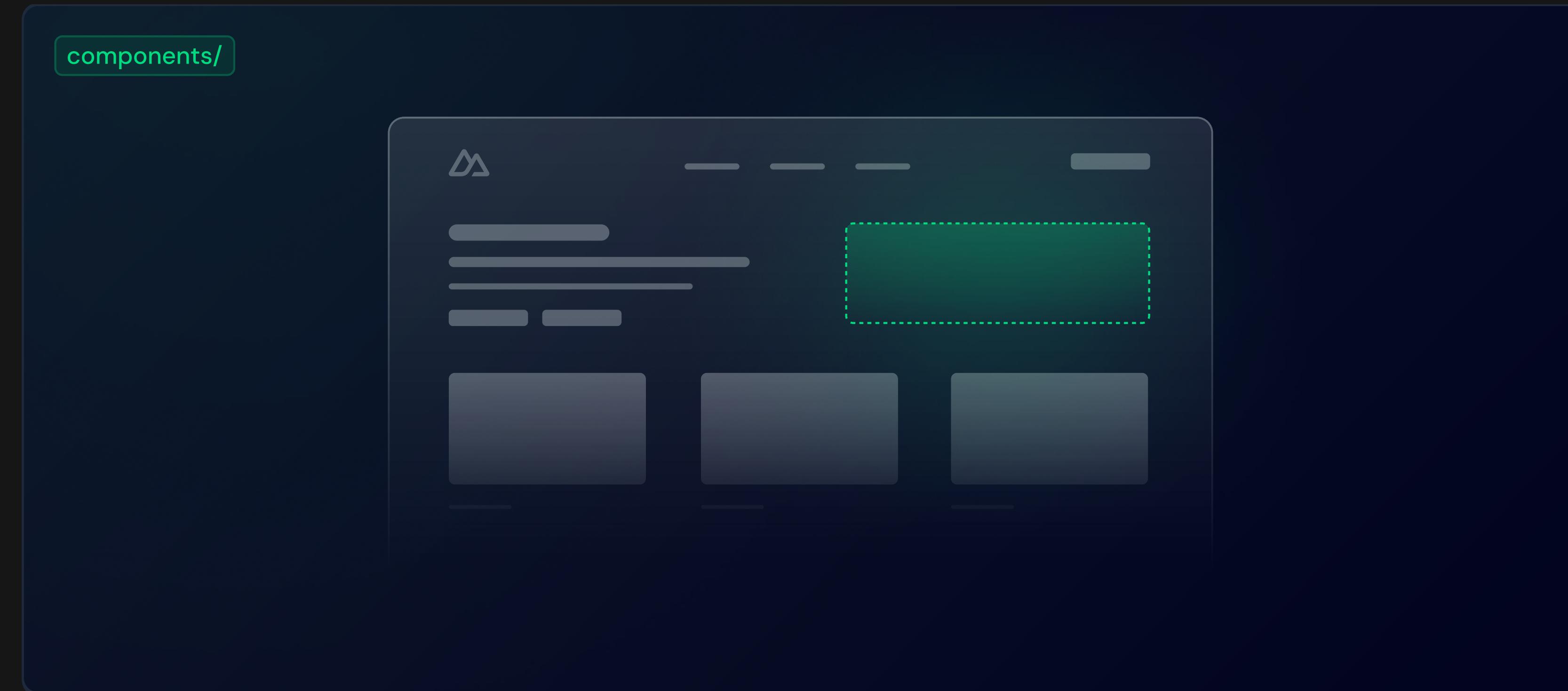


The image shows a dark-themed code editor window with a file named "app.vue". In the top right corner of the editor area, there are three colored circular icons: red, yellow, and green. To the right of the editor area, the file name "app.vue" is displayed in white text. The code itself is written in Vue.js and consists of the following template structure:

```
<template>
  <NuxtLayout>
    <NuxtPage />
  </NuxtLayout>
</template>
```

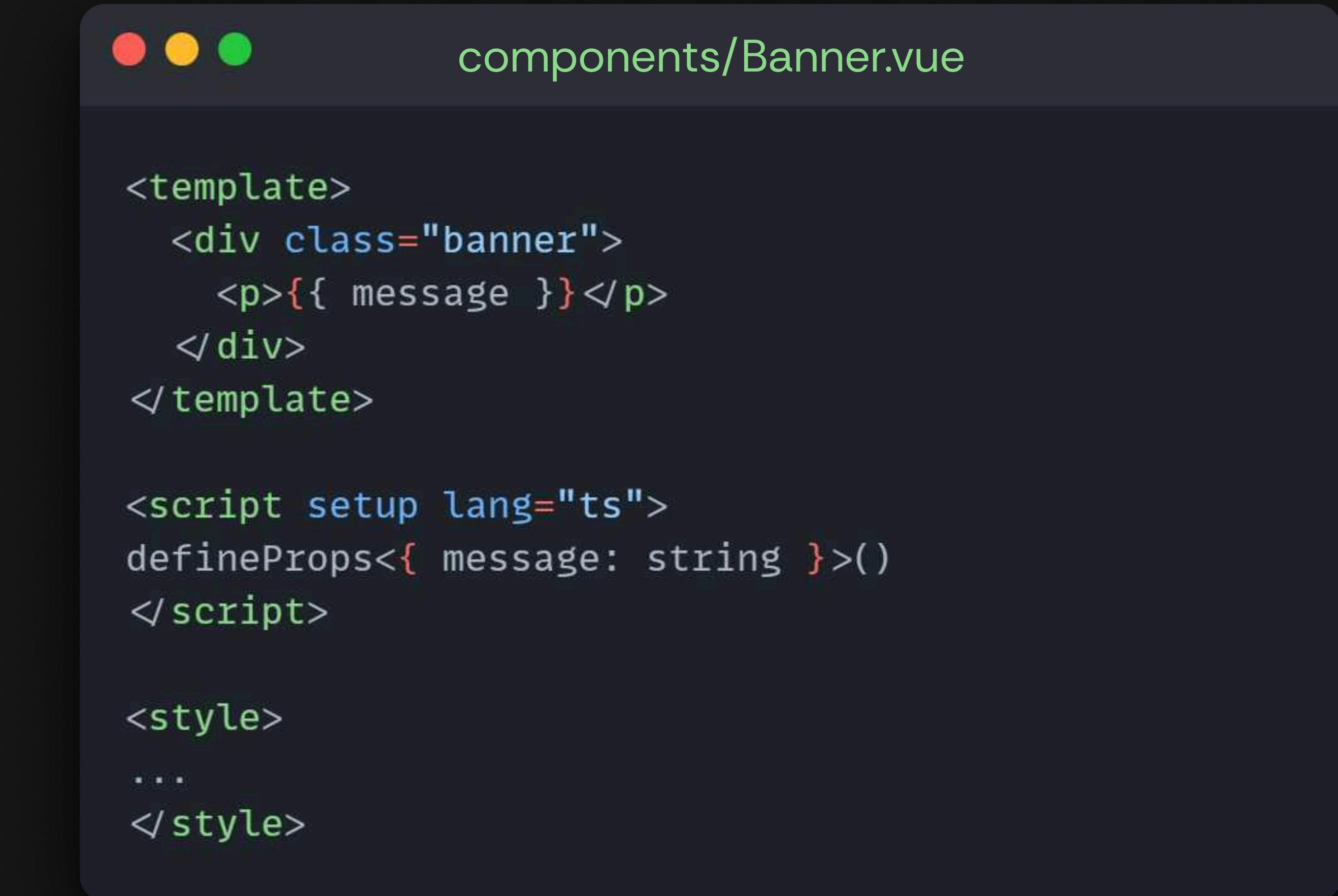


Ordnerstruktur - Komponente





Ordnerstruktur - Komponente



The screenshot shows a Mac OS X window titled "components/Banner.vue". The window contains the following Vue.js component code:

```
<template>
  <div class="banner">
    <p>{{ message }}</p>
  </div>
</template>

<script setup lang="ts">
defineProps<{ message: string }>()
</script>

<style>
...
</style>
```



Ordnerstruktur - Pages





Ordnerstruktur - Pages

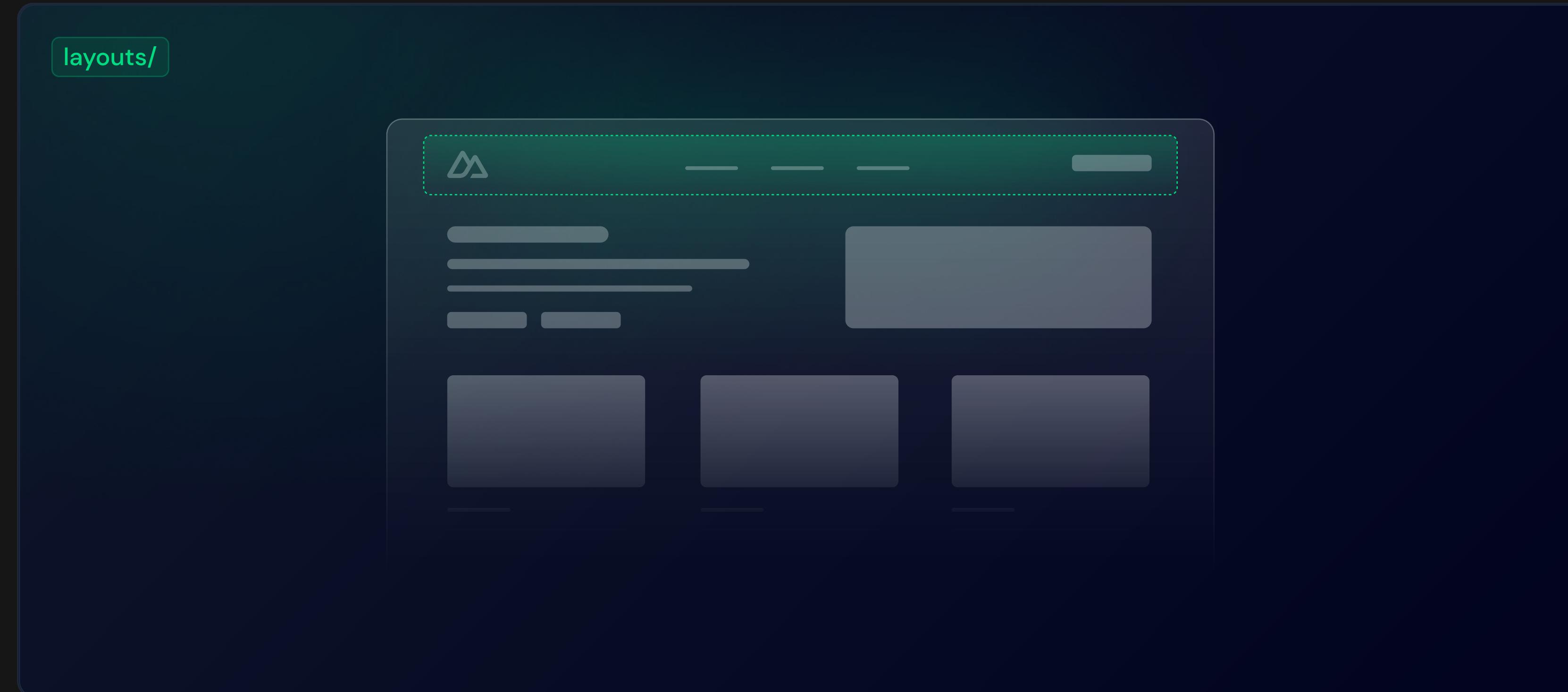


pages/index.vue

```
<template>
  <div>
    <h2>Homepage</h2>
  </div>
</template>
<script setup lang="ts">
</script>
```



Ordnerstruktur - Layouts





Ordnerstruktur - Layouts

layouts/default.vue

```
<template>
  <div class="layout">
    <header>
      <h1>Meine Nuxt App</h1>
      <nav>
        <NuxtLink to="/">Home</NuxtLink> | 
        <NuxtLink to="/about">About</NuxtLink>
      </nav>
    </header>
    <Banner message="Willkommen auf meiner Seite!" />
    <main>
      <NuxtPage />
    </main>

    <footer>
      <p>© 2025 Mein Projekt</p>
    </footer>
  </div>
</template>
<script setup lang="ts">
</script>
<style>
...
</style>
```

Two orange arrows point from the text "Layouts" in the main heading to the highlighted code snippets in the screenshot. The first arrow points to the opening of the `<NuxtLink>` tag, and the second arrow points to the opening of the `<NuxtPage />` tag.



Ordnerstruktur - Layouts



```
<script setup lang="ts">
definePageMeta({
  layout: 'custom' // oder layout: false, für Deaktivierung
})
</script>

<template>
  <div>
    <h1>Kontaktseite mit Custom-Layout</h1>
  </div>
</template>
```



Setup - Aufgabe 09 + 10





Checkerfragen

