

 Vue.js +  Nuxt

113489
Web Development Workshop

Julia Ebert (je073) | Lars Gerigk (lg107) | Joel Starkov (js486)

Ablauf

Vorlesung 1

11.11.2025



Vorlesung 2

18.11.2025

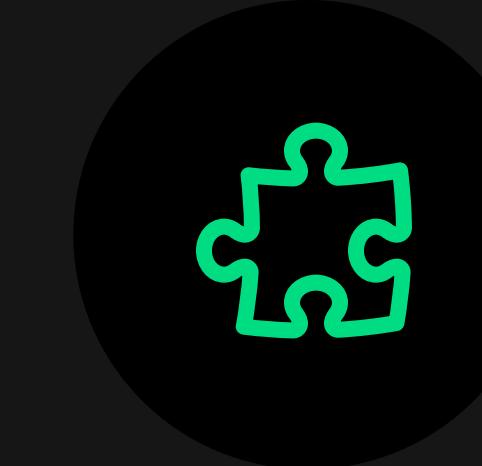


Vorlesung 3

25.11.2025



- Theorie
- Grundlagen Vue.js



- Erweiterungen durch Nuxt



- Puffer
- Wunschthemen



Vorlesung 1

01 Theorie

- 1.1. Was ist Vue.js ?
- 1.2. Was ist Nuxt ?
- 1.3. Verwendung von Vue.js & Nuxt
- 1.4. Unterschiede & Gemeinsamkeiten

02 Grundlagen

- 2.1. Setup & Einstieg in die App
- 2.2. Zwei Arten Komponenten zu erstellen
- 2.3. Syntax (Text Interpolation, Reaktivität. Direktiven)
- 2.4. Event Handling



Was ist Vue.js ?

- das proggessive Framework
- Clientseitiges Javascript-Framework
- Erstellung von Single-Page-Webanwendungen nach dem MVVM-Muster
- HTML wird um proprietäre Attribute erweitert und ergänzt, die das Rendern steuern



Einfachheit



Flexibilität

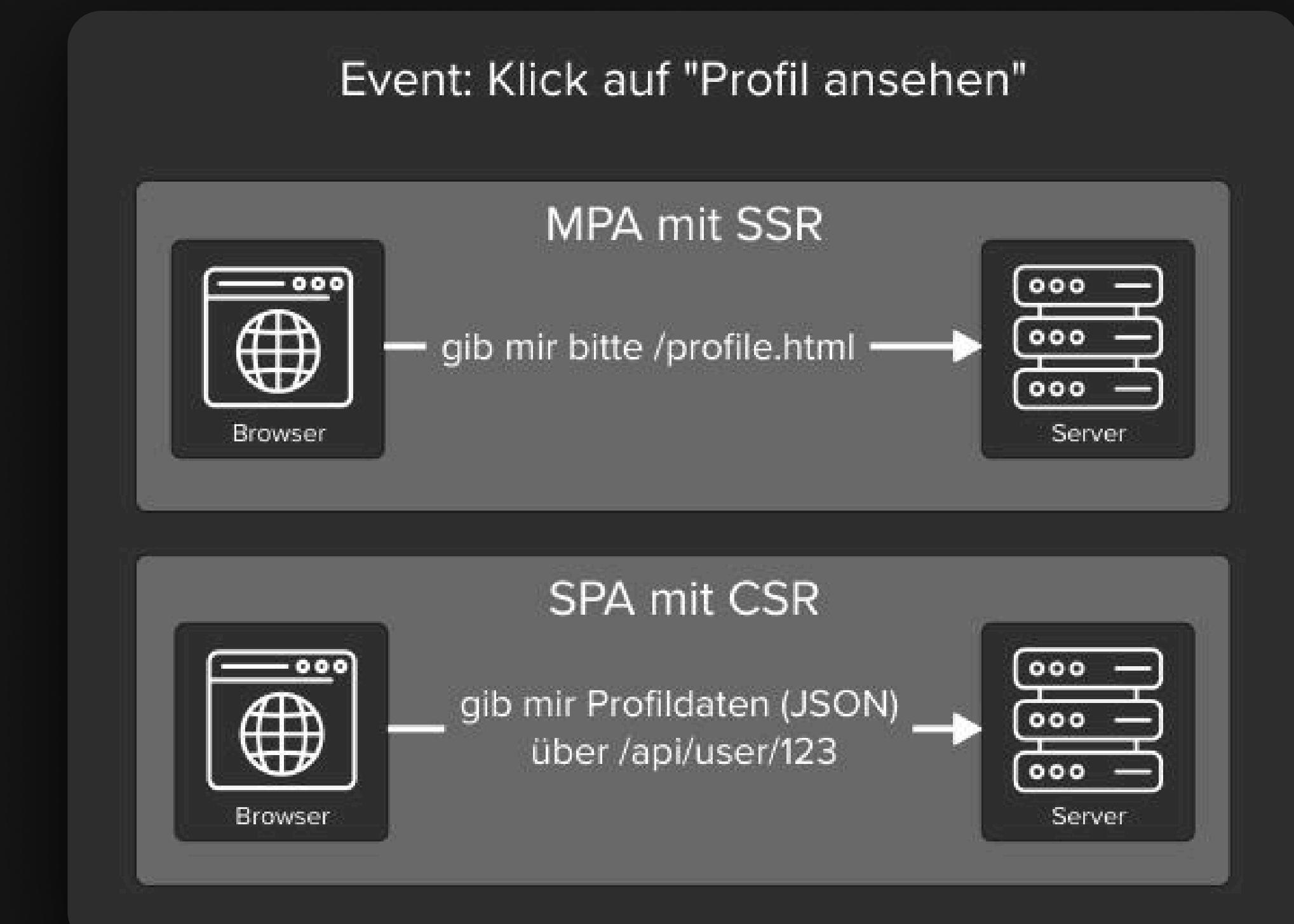


Kann schrittweise in bestehende Projekte integriert werden



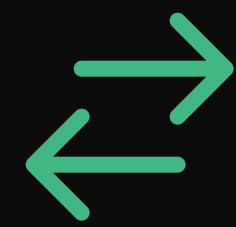
Was ist eine Single-Page-Application ?

- Entwickelten sich aus Multi-Page-Applications
 - Jede Seite hatte eigene HTML-Datei
 - Bei Klick auf Link, wurde von Server komplett neue Seite geladen
- AJAX und Start der Single-Page-Applications
 - Server liefert leere HTML-Seite und JavaScript-Bundle
 - Rendering im Browser
 - Universal Rendering





Kernfunktionen von Vue.js



Reaktivität

Datenänderungen aktualisieren
automatisch die Ansicht



Deklaratives Rendering

Bindung von Daten an HTML mit
einfacher Syntax



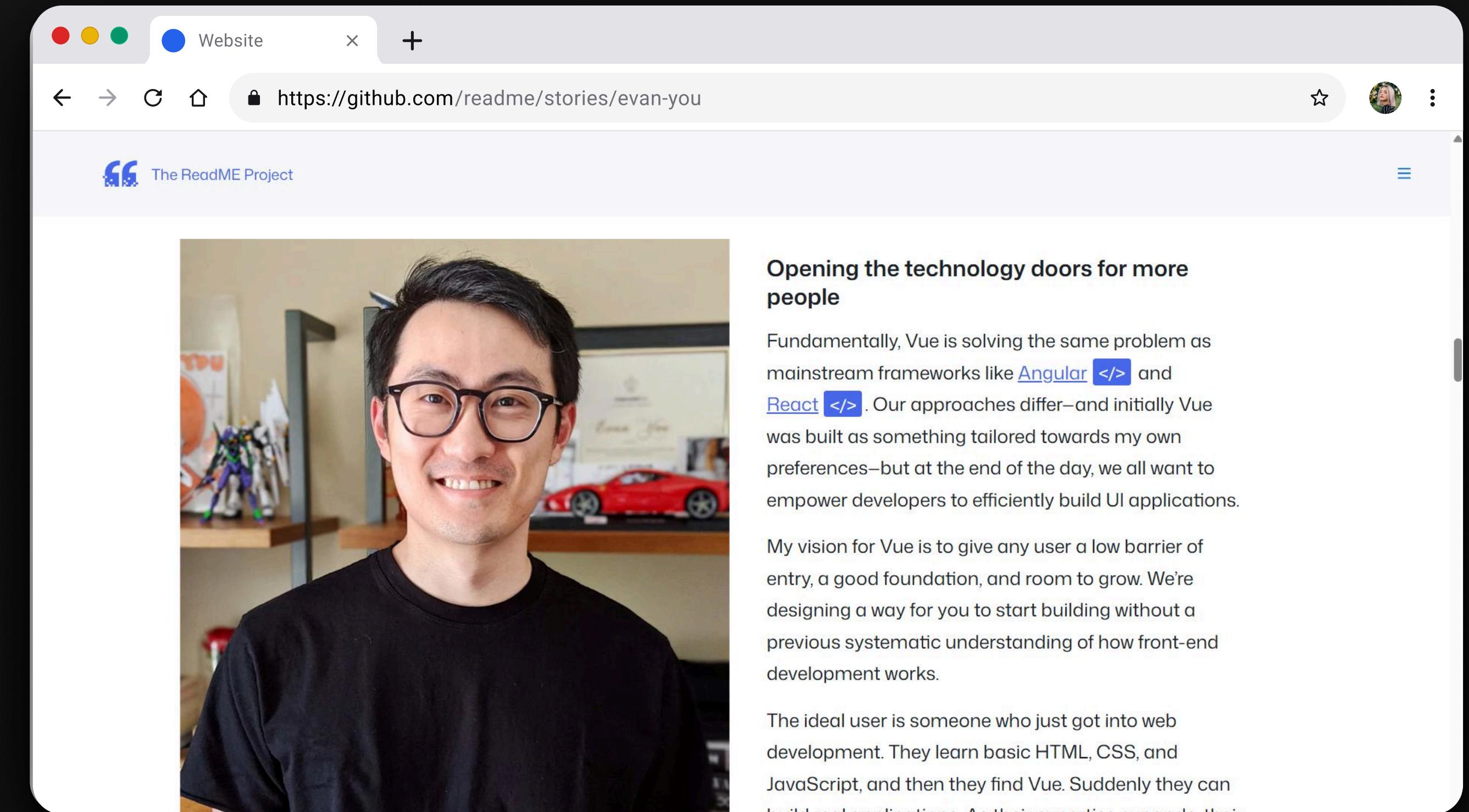
Komponenten

Wiederverwendbare, modulare
Bausteine



Fakten über Vue.js

- erste Version 2014
- Gründer: Evan You
- Vue.js → vom Hobby zum Vollzeitprojekt



Mehr dazu: <https://github.com/readme/stories/evan-you>



Fakten über Vue.js

- Funfact:
 - Versionsnamen werden aus Anime- und Mangaserien in alphabetischer Reihenfolge abgeleitet

Version	Release date	Title
3.5	September 1, 2024	Tengen Toppa Gurren Lagann ^[20]
3.4	December 28, 2023	Slam Dunk ^[21]
3.3	May 11, 2023	Rurouni Kenshin ^[22]
3.2	August 5, 2021	Quintessential Quintuplets ^[23]
3.1	June 7, 2021	Pluto ^[24]
3.0	September 18, 2020	One Piece ^[25]
2.7	July 1, 2022	Naruto ^[26]
2.6	February 4, 2019	Macross ^[27]
2.5	October 13, 2017	Level E ^[28]
2.4	July 13, 2017	Kill la Kill ^[29]
2.3	April 27, 2017	JoJo's Bizarre Adventure ^[30]
2.2	February 26, 2017	Initial D ^[31]
2.1	November 22, 2016	Hunter × Hunter ^[32]
2.0	September 30, 2016	Ghost in the Shell ^[33]
1.0	October 27, 2015	Evangelion ^[34]
0.12	June 12, 2015	Dragon Ball ^[35]
0.11	November 7, 2014	Cowboy Bebop ^[36]
0.10	March 23, 2014	Blade Runner ^[37]
0.9	February 27, 2014	Attack on Titan ^[38]



Was ist Nuxt ?

- Metaframework auf Basis von Vue.js
- Erweitert Vue um viele Funktionen, die man sonst manuell konfigurieren müsste (z.B. Routing, SSR, ...)
- Automatisiertes Rendering – wahlweise serverseitig, statisch oder clientseitig
- Integriertes Backend-Framework (Nitro)
- Klare Projektstruktur durch feste Ordner

- + Struktur
- + Performance
- + Fullstack



Vue.js vs Nuxt

	Vue.js	Nuxt.js
Routing	⚠️ (manuell)	✅ (automatisch)
SEO	🚫	✅
Rendering	⚠️ (nur Client-Side-Rendering)	✅ Server-, Static- oder Client-Side Rendering
automatische Imports	🚫	✅
Data-fetching	⚠️ (nur client-seitig)	✅ (client- und server-seitig)
Flexibilität	✅	⚠️ (viele Konventionen)



Verwendung von Vue.js & Nuxt

Vue.js

- eine einzige interaktive Seite (SPA)
- interne Web-Apps
- volle Kontrolle über Routing, Build-Prozess usw. will
- man nur Client-Side Rendering braucht

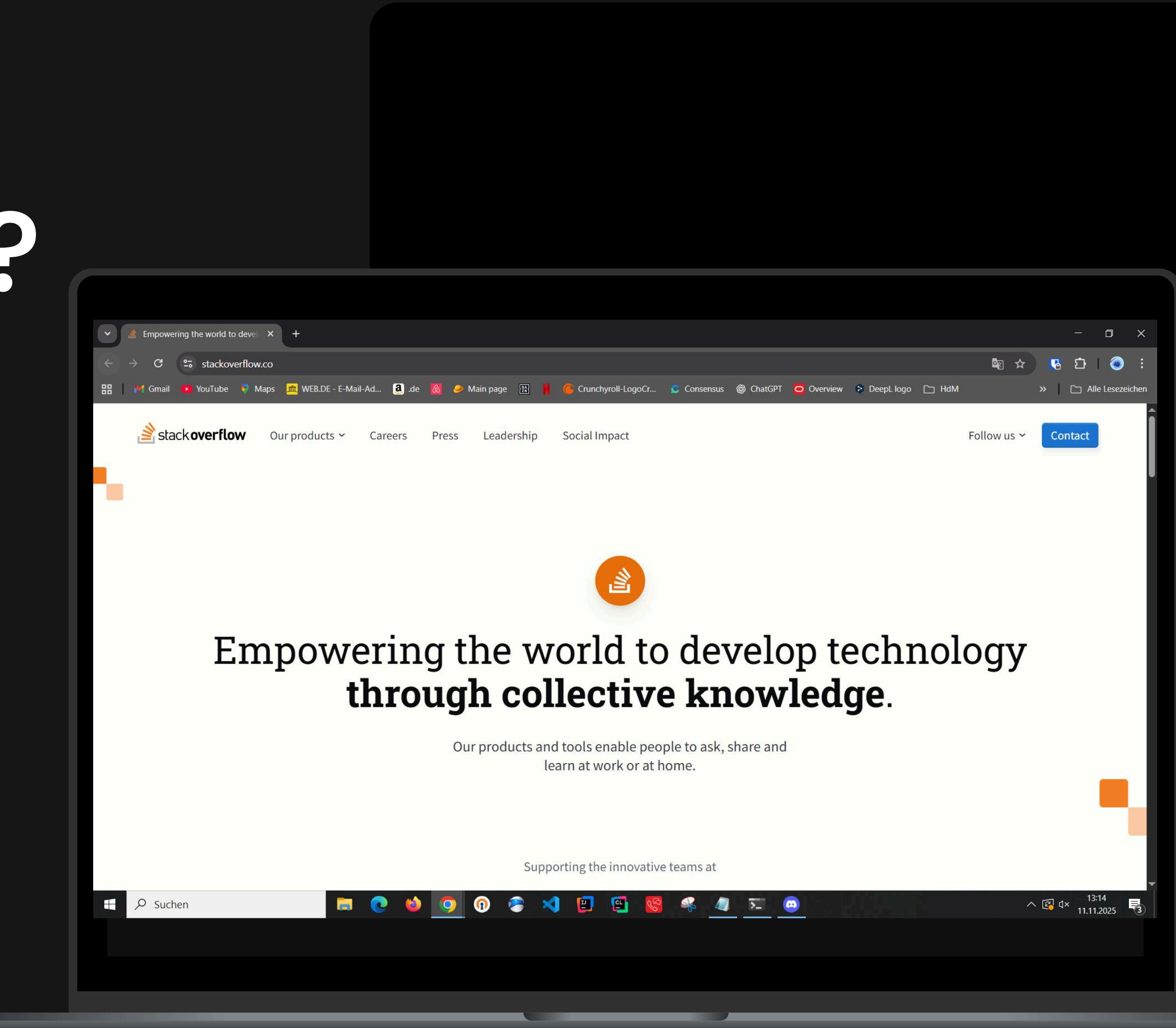
Nuxt.js

- SEO-optimierte oder öffentliche Webseite
- Seiten sollen schnell laden
- klare Projektstruktur
- man hat viele Routen



Wer nutzt Vue.js & Nuxt ?

zum Beispiel: stackoverflow





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
```



Vorlesung 1

01 Theorie

- 1.1. Was ist Vue.js ?
- 1.2. Was ist Nuxt ?
- 1.3. Verwendung von Vue.js & Nuxt
- 1.4. Unterschiede & Gemeinsamkeiten

02 Grundlagen

- 2.1. Setup & Einstieg in die App
- 2.2. Zwei Arten Komponenten zu erstellen
- 2.3. Syntax (Text Interpolation, Reaktivität. Direktiven)



Setup - Aufgabe 01

Schritt 1: git clone <https://github.com/byGamsa/wdw-nuxt-js.git>

Schritt 2: überprüfe, dass npm und node.js auf dem aktuellen Stand sind

- node.js v24.11.0
- npm 11.6.2

Schritt 3: navigiere in einen Ordner wdw-nuxt-js/exercises/01-setup-project

Schritt 4: Führe die Befehle “npm install” und “npm run dev” aus



Wie wir es von HTML kennen

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue Beispiel</title>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <style>
    button {
      background: #00ff99;
      color: #0a0a0a;
    }
  </style>
</head>
<body>
  <div id="app">
    <h1>{{ message }}</h1>
    <button @click="changeMessage">Text ändern</button>
  </div>

  <script>
    const { createApp } = Vue;

    createApp({
      data() {
        return {
          message: 'Hallo Welt!'
        };
      },
      methods: {
        changeMessage() {
          this.message = 'Text wurde geändert!';
        }
      }
    }).mount('#app');
  </script>
</body>
</html>
```



Syntax von Vue.js

- Single-File-Component
- Dateiendung: *.vue
- NEU: <template></template>



```
<template>
<div>
  <h1>{{ message }}</h1>
  <button @click="changeMessage">Text ändern</button>
</div>
</template>

<script>
export default {
  data() {
    return { message: 'Hallo Welt!' }
  },
  methods: {
    changeMessage() {
      this.message = 'Text wurde geändert!'
    }
  }
}
</script>

<style>
h1 {
  color: #00ff99;
}
</style>
```



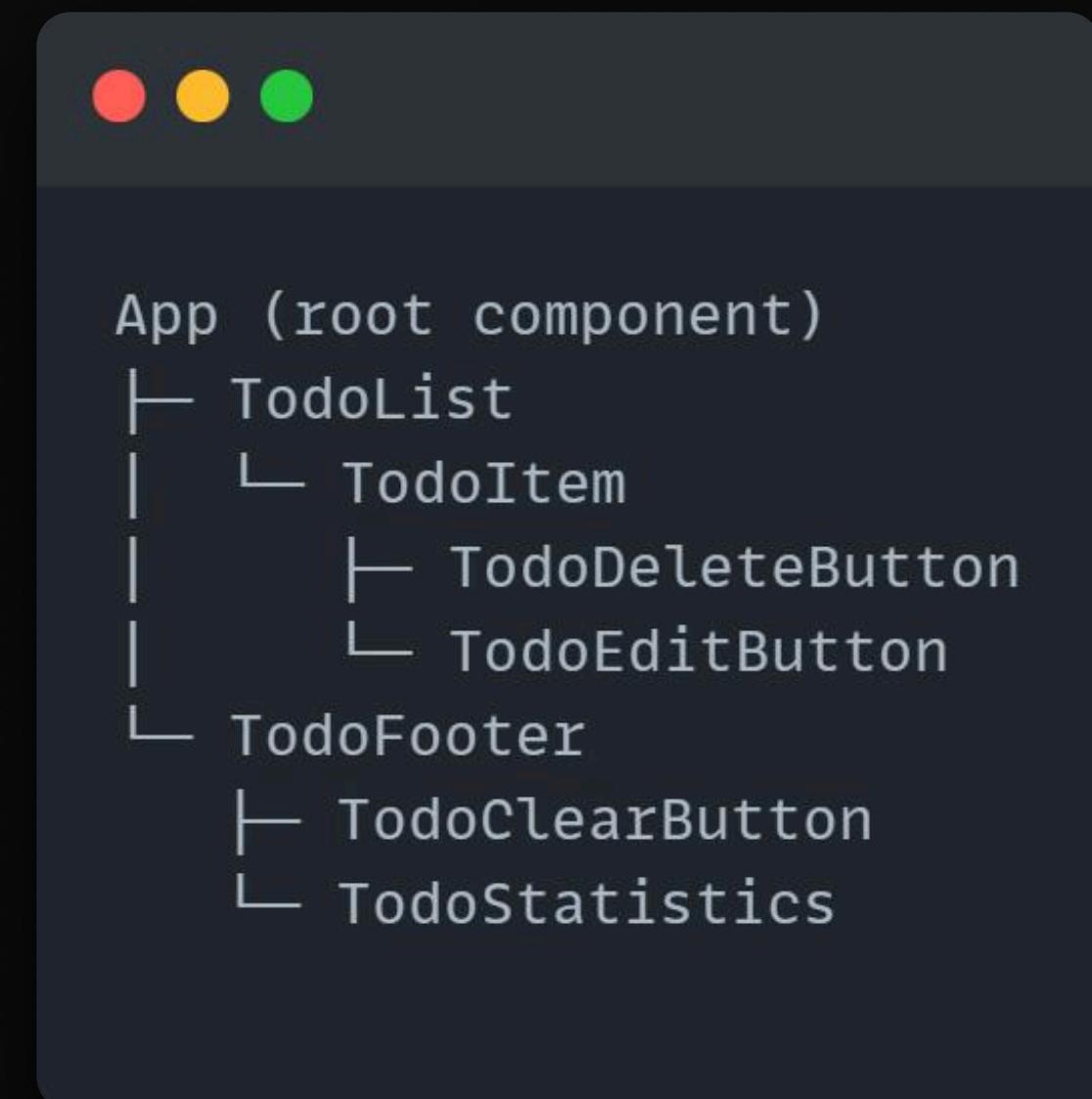
Wie starte ich eine Vue.js Applikation ?

```
import { createApp } from 'vue'  
import App from './App.vue'  
  
createApp(App).mount('#app')
```

main.js



Mögliche Ordnerstruktur





Zwei Arten Komponenten zu definieren - Options API

- Logik wird innerhalb der JS-Anweisung
"export default {}" nach Optionen gegliedert



```
<script>
  export default {
    data() {
      return {
        count: 0
      },
    },
    methods: {
      increment() {
        this.count++
      },
    },
    mounted() {
      console.log(`The initial count is ${this.count}.`)
    }
  }
</script>

<template>
  <button @click="increment">Anzahl ist: {{ count }}</button>
</template>
```



Zwei Arten Komponenten zu definieren - Compositions API

- Komponentenlogik mit importierten API-Funktionen
- Wird in Single File Components (SFCs) meist mit `<script setup>` verwendet



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

// reactive state
const count = ref(0)

// functions that mutate state and trigger updates
function increment() {
  count.value++
}

// lifecycle hooks
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```



Text Interpolation

- innerhalb der {{ ... }} kann JavaScript Code stehen
- Ähnlich wie bei JavaScript Text Interpolation

JavaScript \${ ... }

HTML {{ ... }}

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

const name = 'Alex'
const age = 29
const message = ref(`Hallo ${name}, du bist ${age.value} Jahre alt.`)
</script>

<template>
  <p>{{ message }}</p>
  <button @click="message = `Hallo ${name}, du bist jetzt ${++age}!`">Ändern</button>
</template>
```

Text Interpolation

Rechnungen



```
<script setup>
const price = 19.99
const quantity = 3
const isAdmin = true
const username = 'alex'
const today = new Date()
</script>

<template>
  <div>
    <p>💻 Rechnen: {{ price * quantity }}€</p>
    <p>⌚ Bedingung: {{ isAdmin ? 'Admin' : 'User' }}</p>
    <p>🔤 Funktion: {{ username.toUpperCase() }}</p>
    <p>📅 Formatierung: {{ today.toLocaleDateString() }}</p>
  </div>
</template>
```

Text Interpolation

Bedingung



```
<script setup>
const price = 19.99
const quantity = 3
const isAdmin = true
const username = 'alex'
const today = new Date()
</script>

<template>
  <div>
    <p>💻 Rechnen: {{ price * quantity }} €</p>
    <p>⌚ Bedingung: {{ isAdmin ? 'Admin' : 'User' }}</p>
    <p>🔤 Funktion: {{ username.toUpperCase() }}</p>
    <p>📅 Formatierung: {{ today.toLocaleDateString() }}</p>
  </div>
</template>
```

Text Interpolation

```
<script setup>
const price = 19.99
const quantity = 3
const isAdmin = true
const username = 'alex'
const today = new Date()
</script>

<template>
<div>
<p>💻 Rechnen: {{ price * quantity }} €</p>
<p>⌚ Bedingung: {{ isAdmin ? 'Admin' : 'User' }}</p>
<p>🆎 Funktion: {{ username.toUpperCase() }}</p>
<p>🕒 Formatierung: {{ today.toLocaleDateString() }}</p>
</div>
</template>
```

Funktion



Text Interpolation

```
<script setup>
const price = 19.99
const quantity = 3
const isAdmin = true
const username = 'alex'
const today = new Date()
</script>

<template>
<div>
<p>💻 Rechnen: {{ price * quantity }} €</p>
<p>⌚ Bedingung: {{ isAdmin ? 'Admin' : 'User' }}</p>
<p>🔤 Funktion: {{ username.toUpperCase() }}</p>
<p>📅 Formatierung: {{ today.toLocaleDateString() }}</p>
</div>
</template>
```

Formatierung



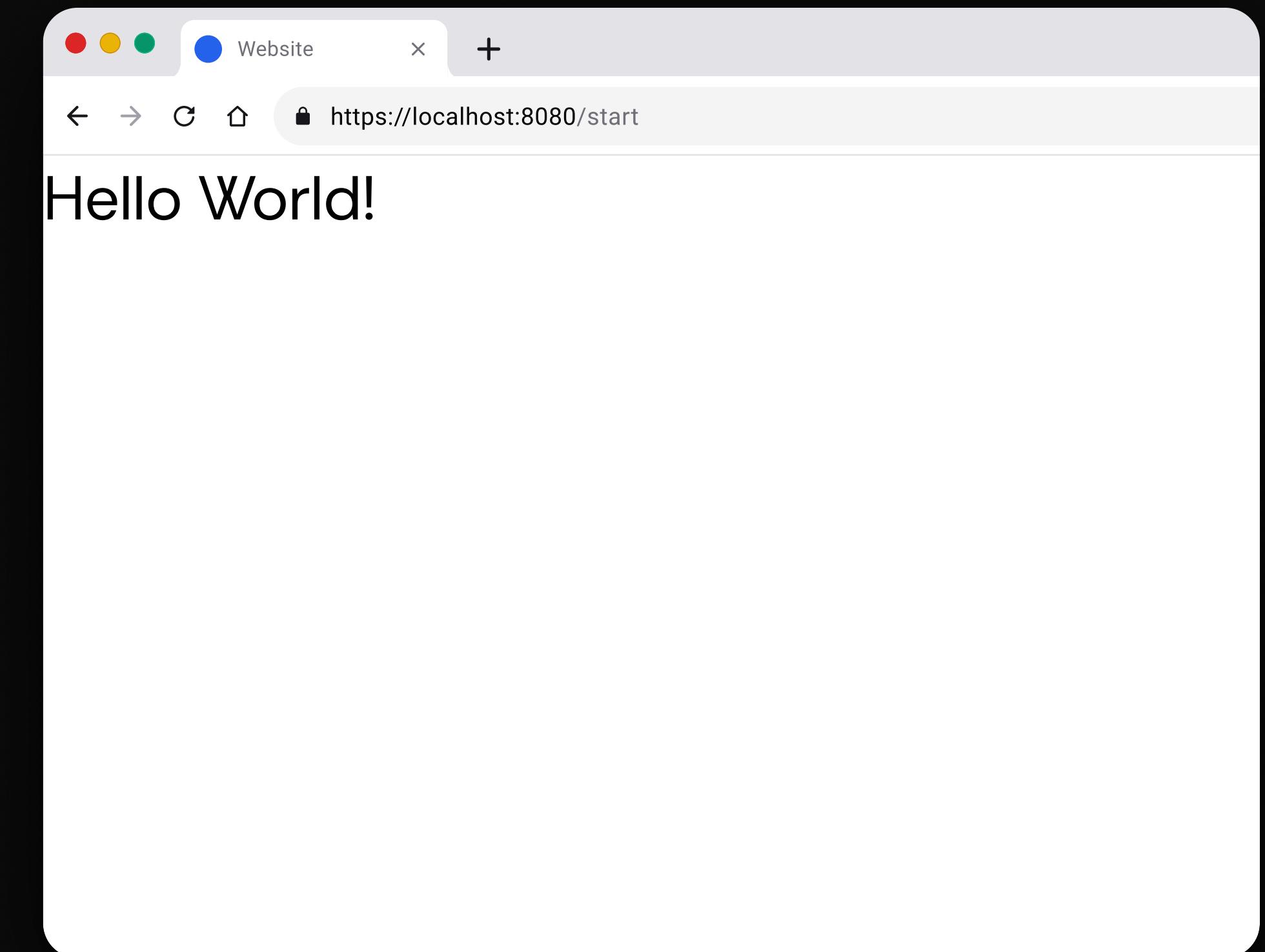


Reaktivität

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

const message = ref('Hello World!')
</script>

<template>
  <h1>{{ message }}</h1>
</template>
```



A screenshot of a web browser window titled "Website". The address bar shows "https://localhost:8080/start". The main content area displays the text "Hello World!".



Aufgabe 02

10 min

02 – Reactive Data

Ziel

In dieser Übung lernst du, wie du mit der Vue-Funktion `ref()` reaktive Daten erzeugst und diese dynamisch im Template anzeigenst.

Ausgangspunkt

Deine Anwendung enthält bereits:

- Ein Benutzerprofil mit `username`, `fullname` und `stats`
- Eine Sidebar und einen Profil-Bereich im Layout

Aufgabe

1. Definiere im `<script setup>`-Bereich eine neue reaktive Variable namens `bio` mit `ref()`.
2. Diese Variable soll einen kurzen Beschreibungstext (Bio) des Benutzers enthalten.
3. Binde diese `bio`-Variable im Template unterhalb der Statistik ein. Verwende dafür ein `p` Element.
4. Verwende für das neue Element die CSS-Klasse `description`.



Direktiven

- Präfix: “v-”
- DOM Manipulation mit speziellen HTML-Attributen

	Direktive	Funktion
Attribut Bindung	v-bind	dynamische Datenbindung
Conditional Rendering	v-if	erstellt HTML-Tags abhängig von einer Bedingung
List Rendering	v-for	erstellt wiederholte Elemente aus einer Liste
Event Handling	v-on	Reaktion auf Events

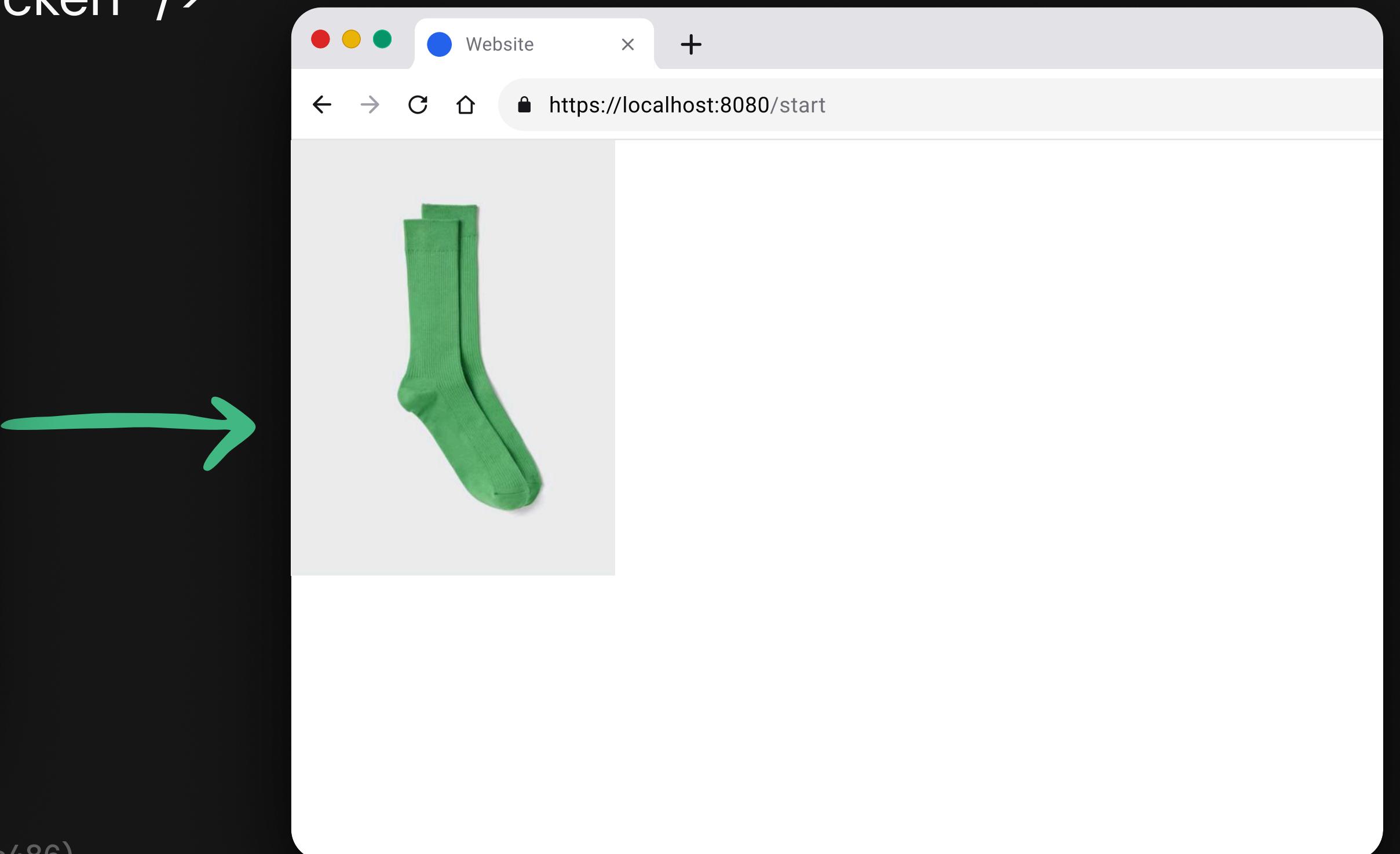
Attribut Binding

v-bind

- **Funktion:** dynamische Datenbindung an Attribute
- **Kurzschrifweise:** z.B.: ``

```
  

  <template>
    
  </template>
```



Attribut Binding

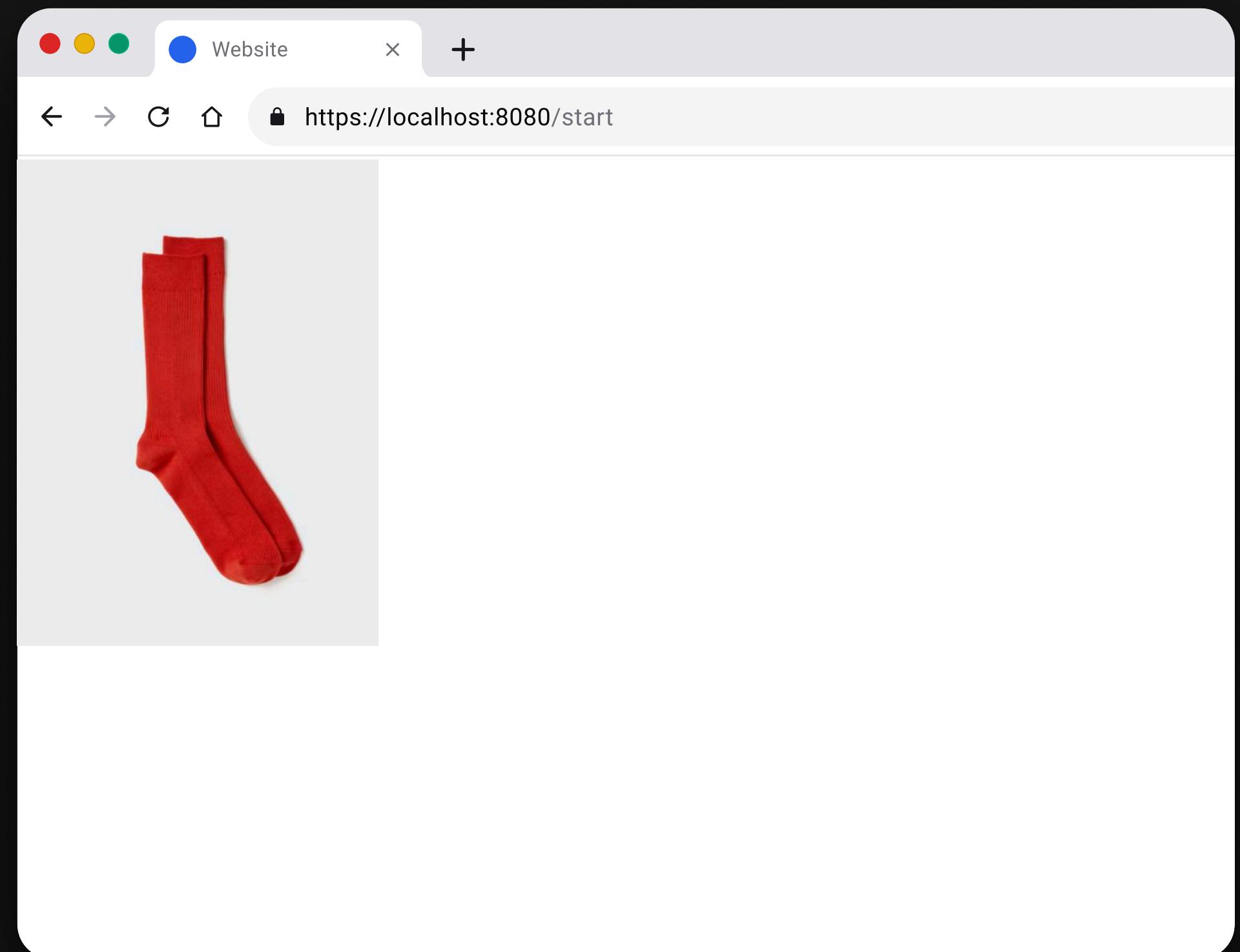
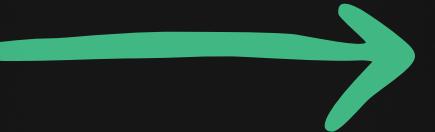
v-bind

- **Funktion:** dynamische Datenbindung an Attribute
- **Kurzschrifweise:** z.B.: ``

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

const image = ref(socksRedImage)
</script>

<template>
  
</template>
```





Aufgabe 03

10 min

03 – Attribute-Binding

Ziel

In dieser Übung lernst du, wie man mit der Vue-Direktive `v-bind` (oder der Kurzform `:`) reaktive Daten an HTML-Attribute bindet.

Ausgangspunkt

Deine Anwendung zeigt bereits ein Profil mit Avatar, Name, Statistik und Beschreibung. Jetzt soll ein Social Link hinzugefügt werden, der auf die externe Webseite oder ein Social-Media-Profil verweist.

Aufgabe

1. Definiere im `<script setup>`-Bereich eine neue reaktive Variable `socialLink` mit `ref()`.
2. Diese Variable soll eine vollständige URL enthalten (z. B. zur offiziellen Website oder einem Instagram-Profil).
3. Binde diese `socialLink`-Variable im Template mit `v-bind` an das `href`-Attribut eines `<a>`-Tags.
4. Füge den Link **unterhalb** der Beschreibung (`.description`) ein.
5. Gib dem Link die CSS-Klasse `social-link`.

Hinweis: Verwende die Schreibweise `v-bind:href`, um das Konzept klar zu üben.



Conditional Rendering

v-if

v-else

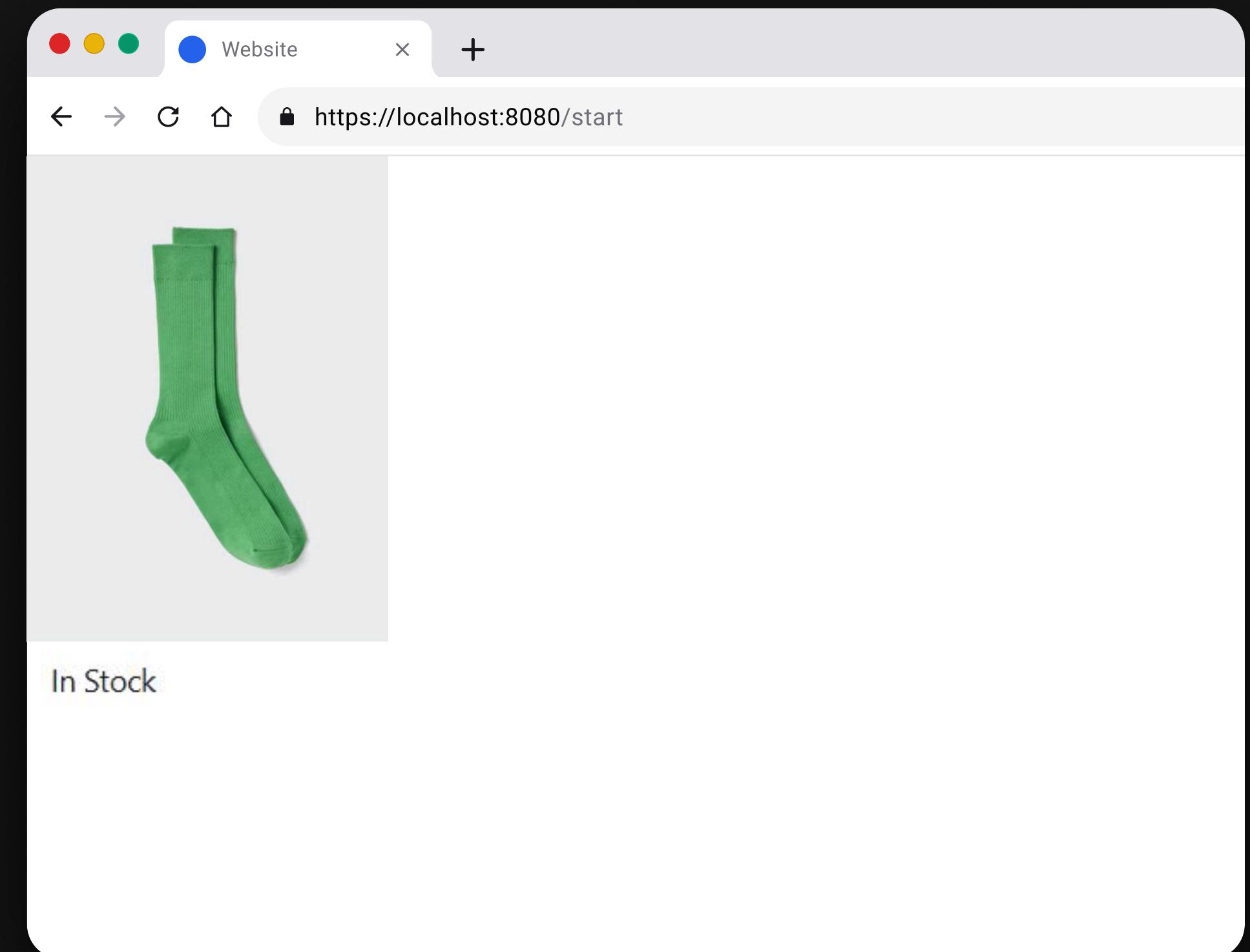
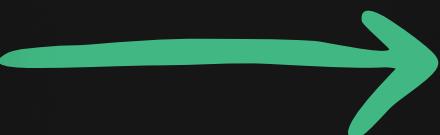
- **Funktion:** erstellt HTML-Tags abhängig von einer Bedingung
- **Weitere Direktiven:** v-if-else

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

const image = ref(socksRedImage)
const inStock = ref(true)

</script>

<template>
  
  <p v-if="inStock">In Stock</p>
  <p v-else>Out of Stock</p>
</template>
```





Conditional Rendering

v-if

v-else

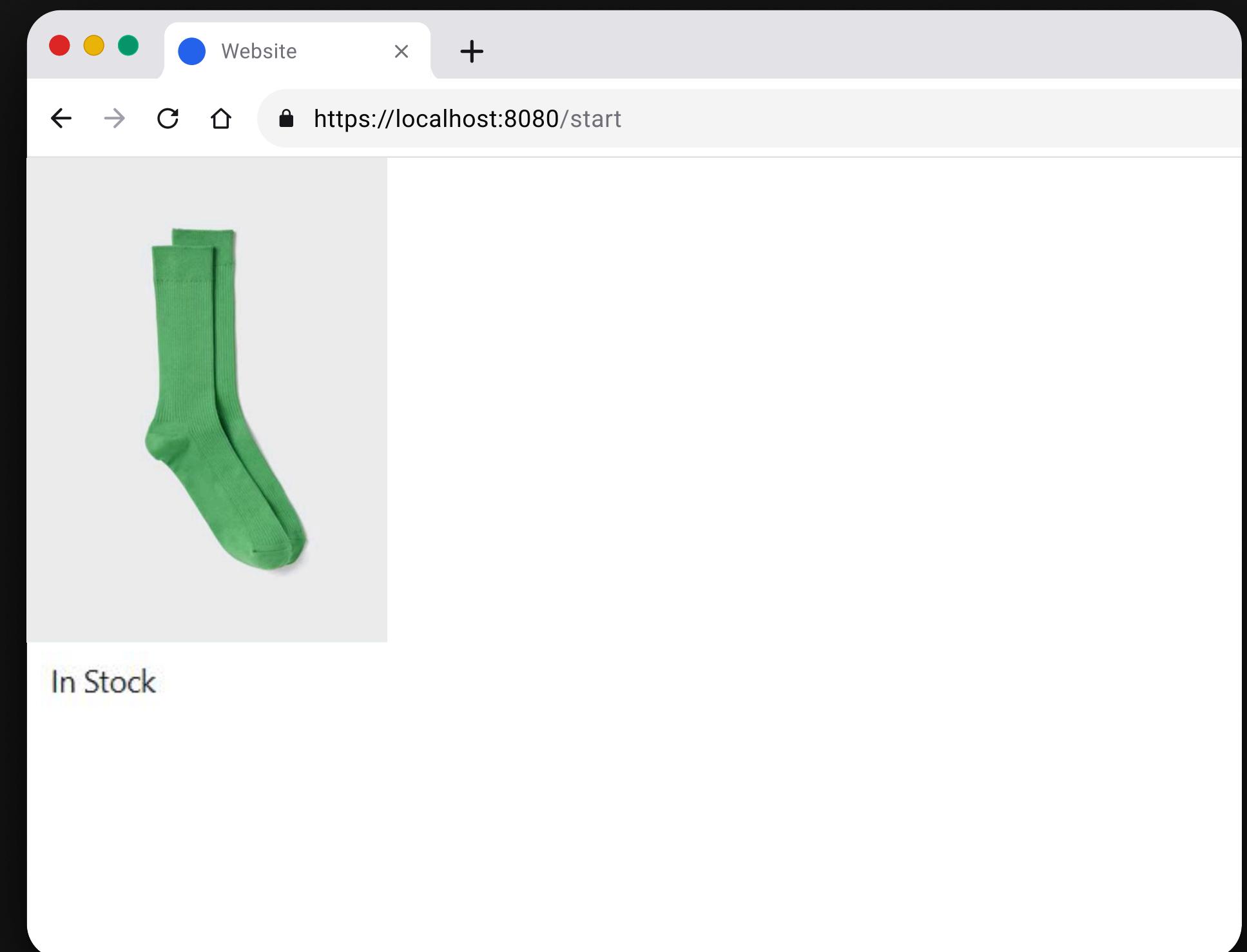
- **Funktion:** erstellt HTML-Tags abhängig von einer Bedingung
- **Weitere Direktiven:** v-if-else

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

const image = ref(socksRedImage)
const inStock = ref(true)

</script>

<template>
  
  <p v-if="inStock">In Stock</p>
  <p v-else>Out of Stock</p>
</template>
```





Conditional Rendering

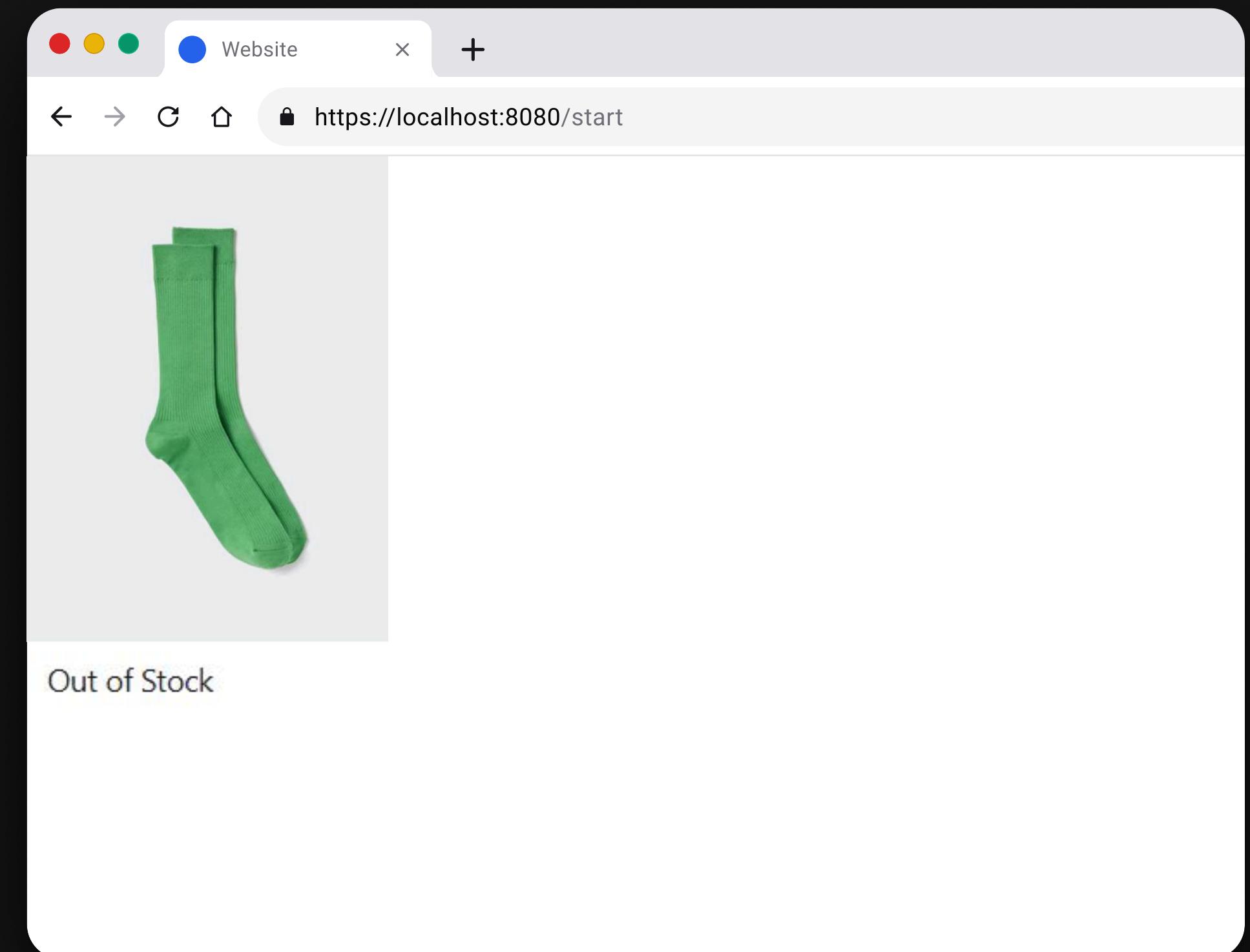
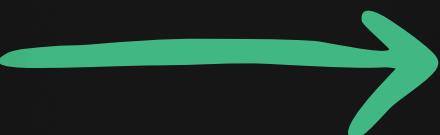
v-if

v-else

- **Funktion:** erstellt HTML-Tags abhängig von einer Bedingung
- **Weitere Direktiven:** v-if-else

```
  

  <template>
    
    <p v-if="inStock">In Stock</p>
    <p v-else>Out of Stock</p>
  </template>
```





Conditional Rendering

v-show

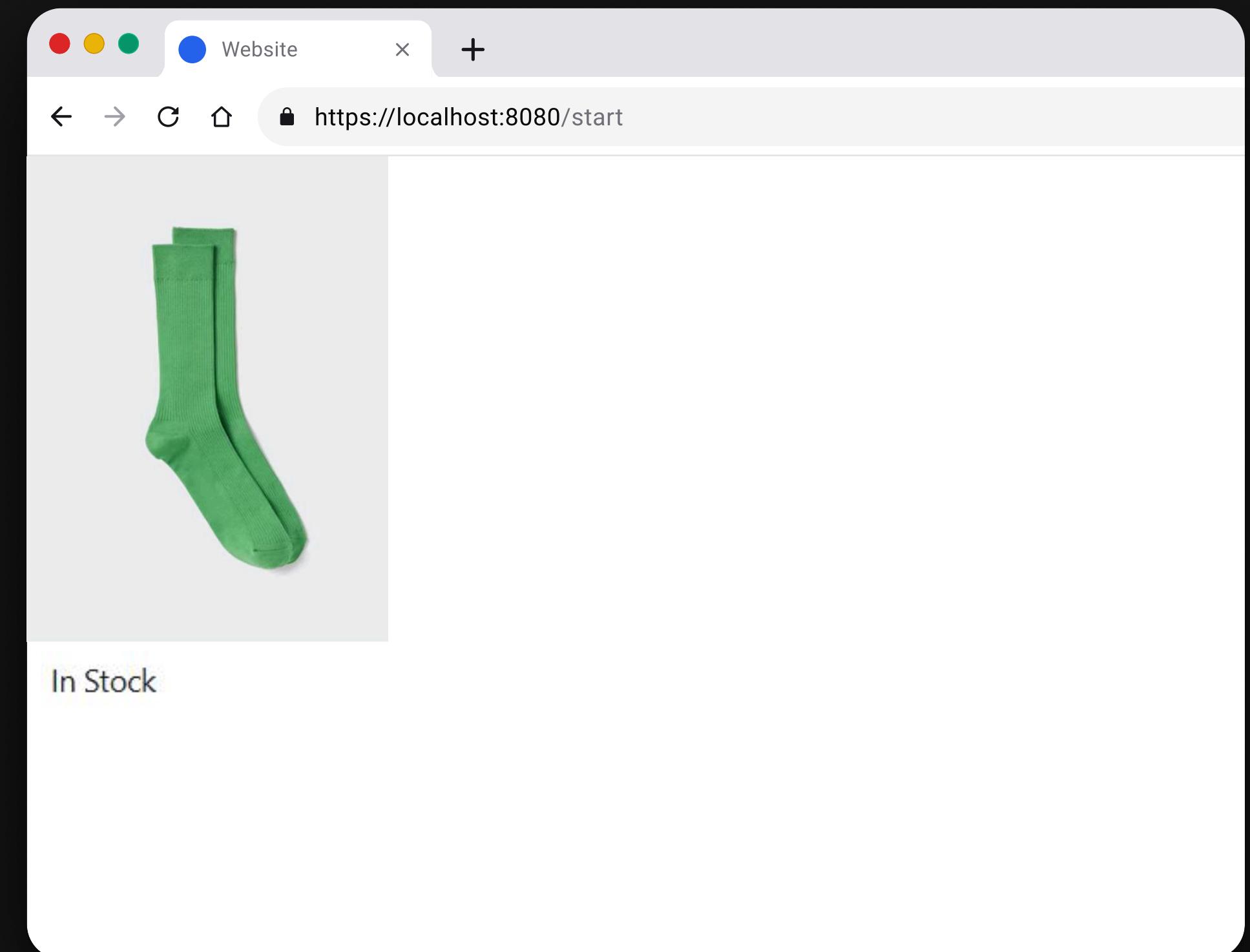
- **Funktion:** Gibt an, ob ein HTML-Element je nach Bedingung sichtbar sein soll oder nicht.

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

const image = ref(socksRedImage)
const inStock = ref(true)

</script>

<template>
  
  <p v-show="inStock">In Stock</p>
</template>
```





Conditional Rendering

v-show

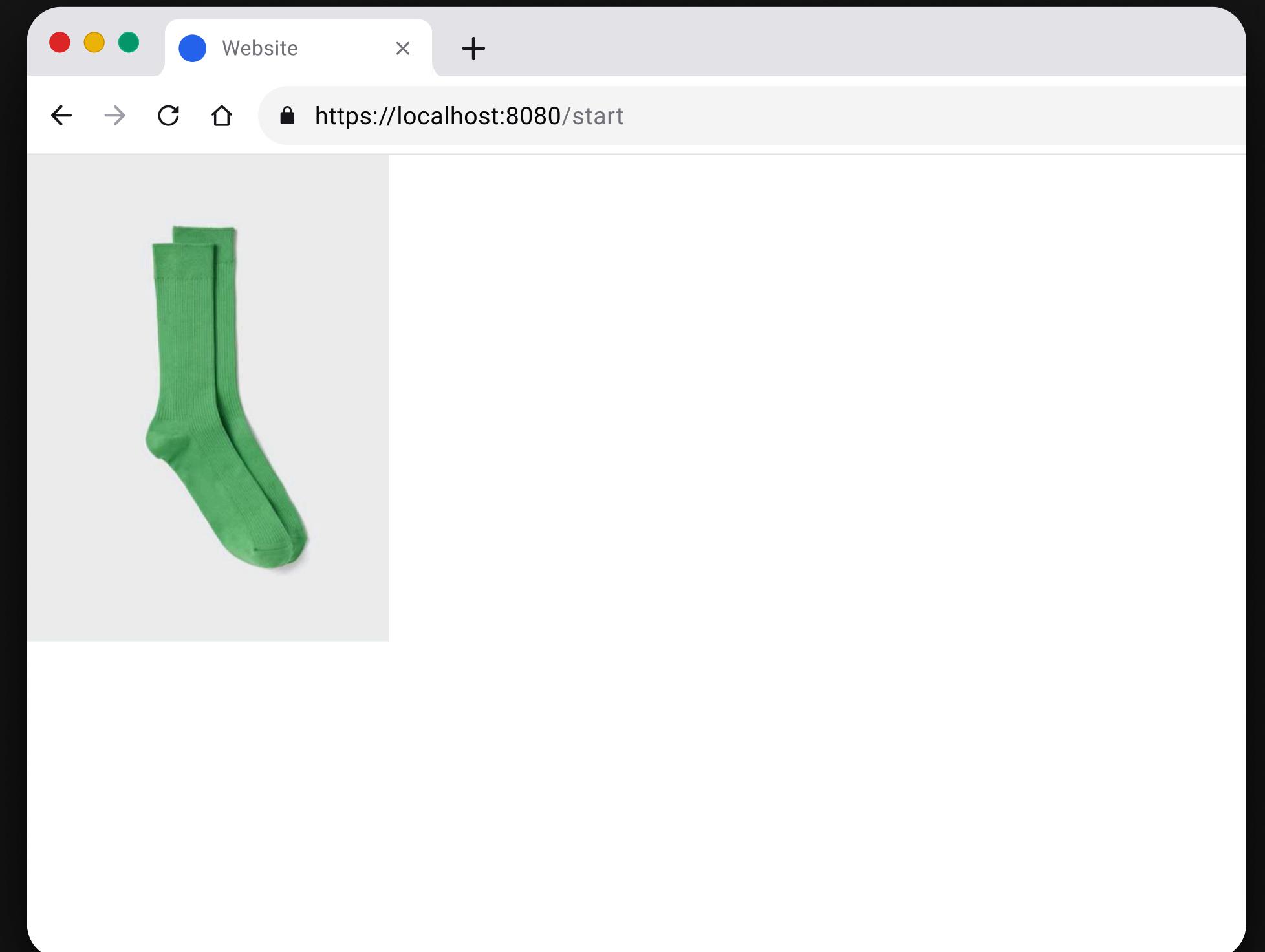
- **Funktion:** Gibt an, ob ein HTML-Element je nach Bedingung sichtbar sein soll oder nicht.

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

const image = ref(socksRedImage)
const inStock = ref(false)

</script>

<template>
  
  <p v-show="inStock">In Stock</p>
</template>
```





Aufgabe 04

10 min

04 – Conditional Rendering

Ziel

In dieser Übung lernst du, wie du Inhalte im Template abhängig von Bedingungen anzeigen kannst. Dafür verwenden wir die Vue-Direktiven `v-if`, `v-else` und `v-show`.

Ausgangspunkt

Dein Profil enthält nun bereits:

- Avatar, Benutzername und Statistik
- Social-Link
- Follow-/Unfollow-Buttons Unterhalb des Profils ist ein leerer Post-Bereich vorbereitet.

Aufgabe

1. Definiere im `<script setup>` eine neue reaktive Variable `hasPosts` mit `ref(false)`.
2. Im Template, innerhalb des `<div class="posts">`, soll Folgendes geschehen:
 - Wenn `hasPosts` `false` ist, soll der Text „Noch keine Beiträge vorhanden“ angezeigt werden.
 - Wenn `hasPosts` `true` ist, soll der Text „Beiträge werden geladen...“ erscheinen.
3. Verwende dafür die Vue-Direktive `v-if` und `v-else`.



Checkerfragen





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

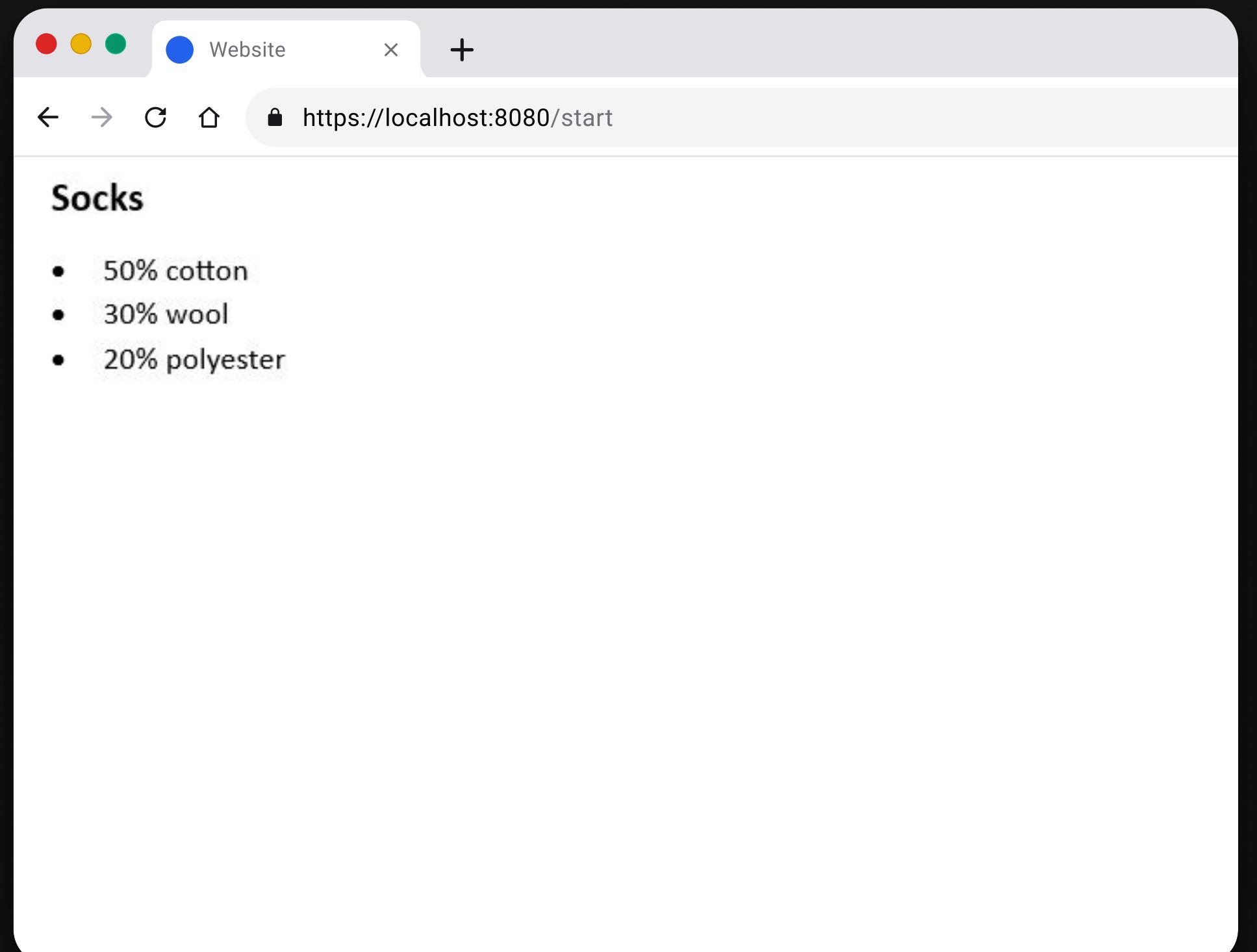
v-for

- **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

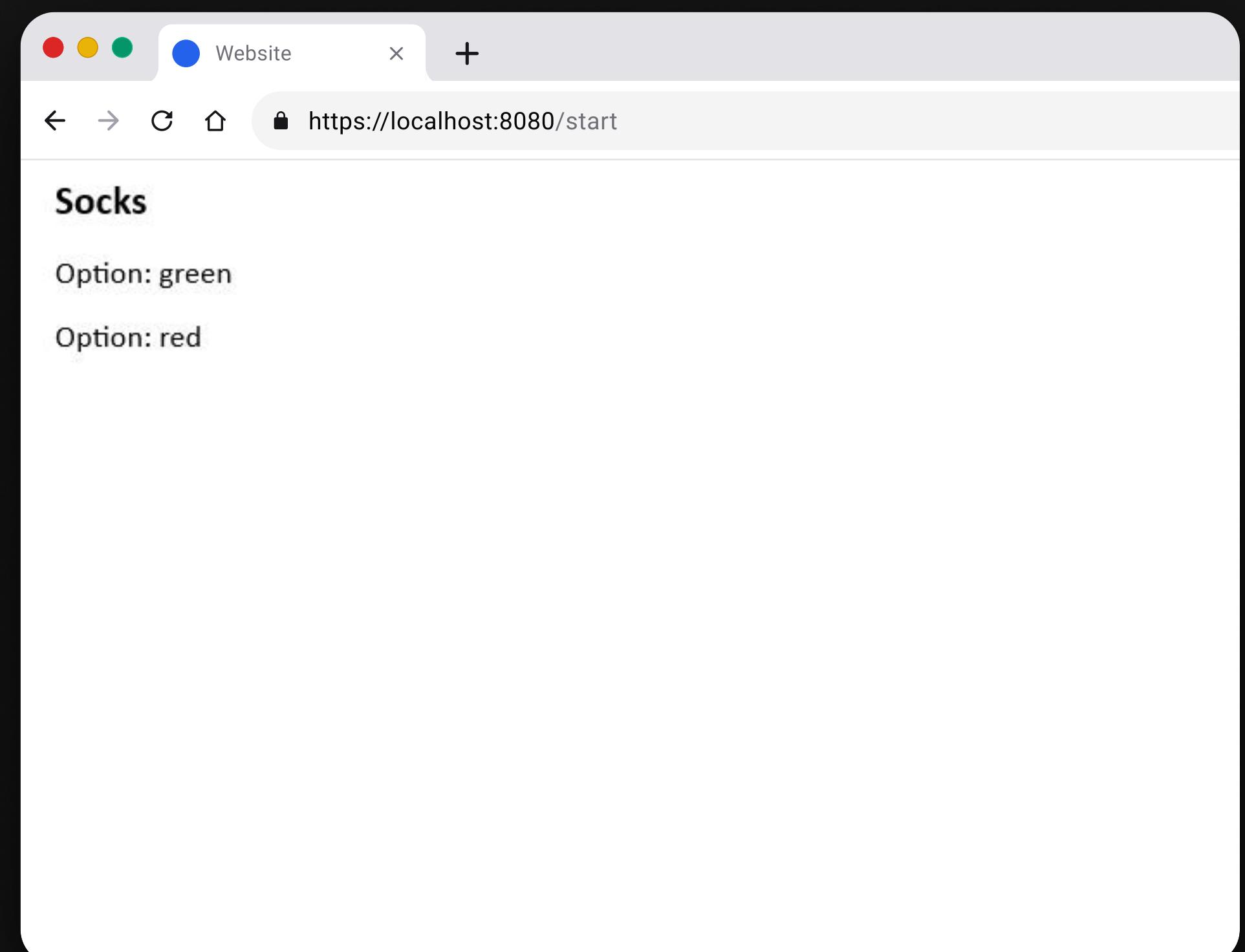
v-for

- **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:

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

2. Rendering mit `v-for`:

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

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

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

Event Handling

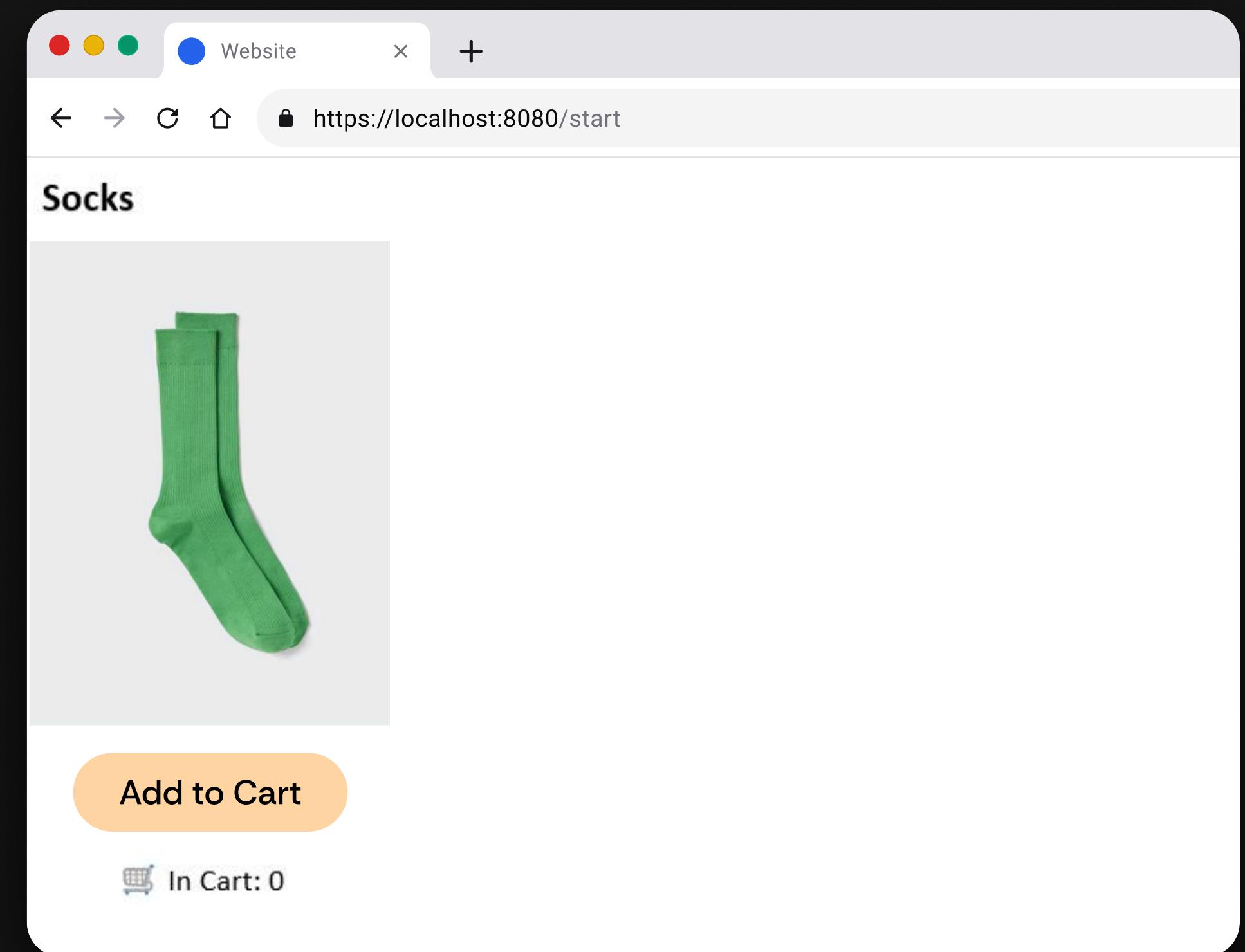
v-on

- **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

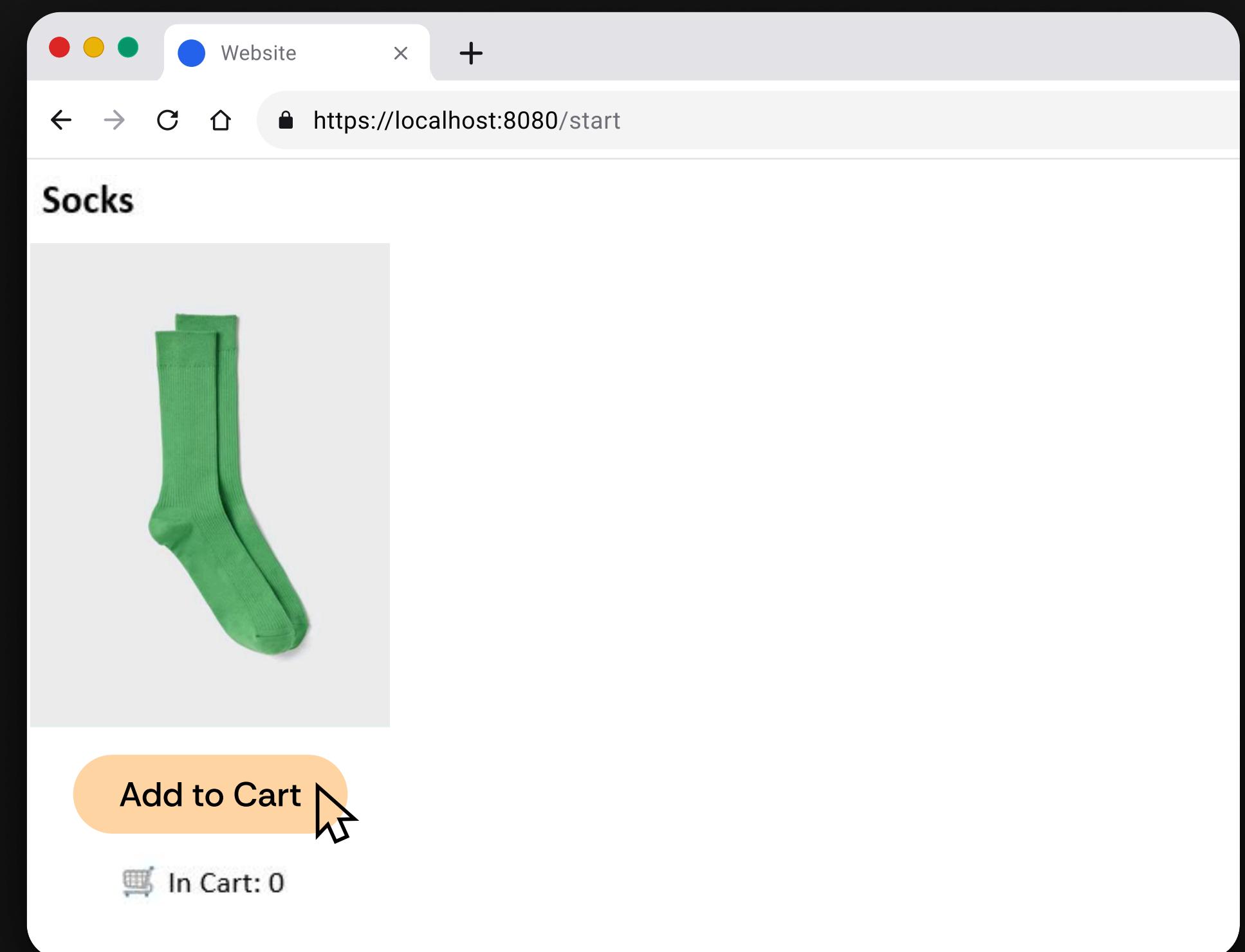
v-on

- **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

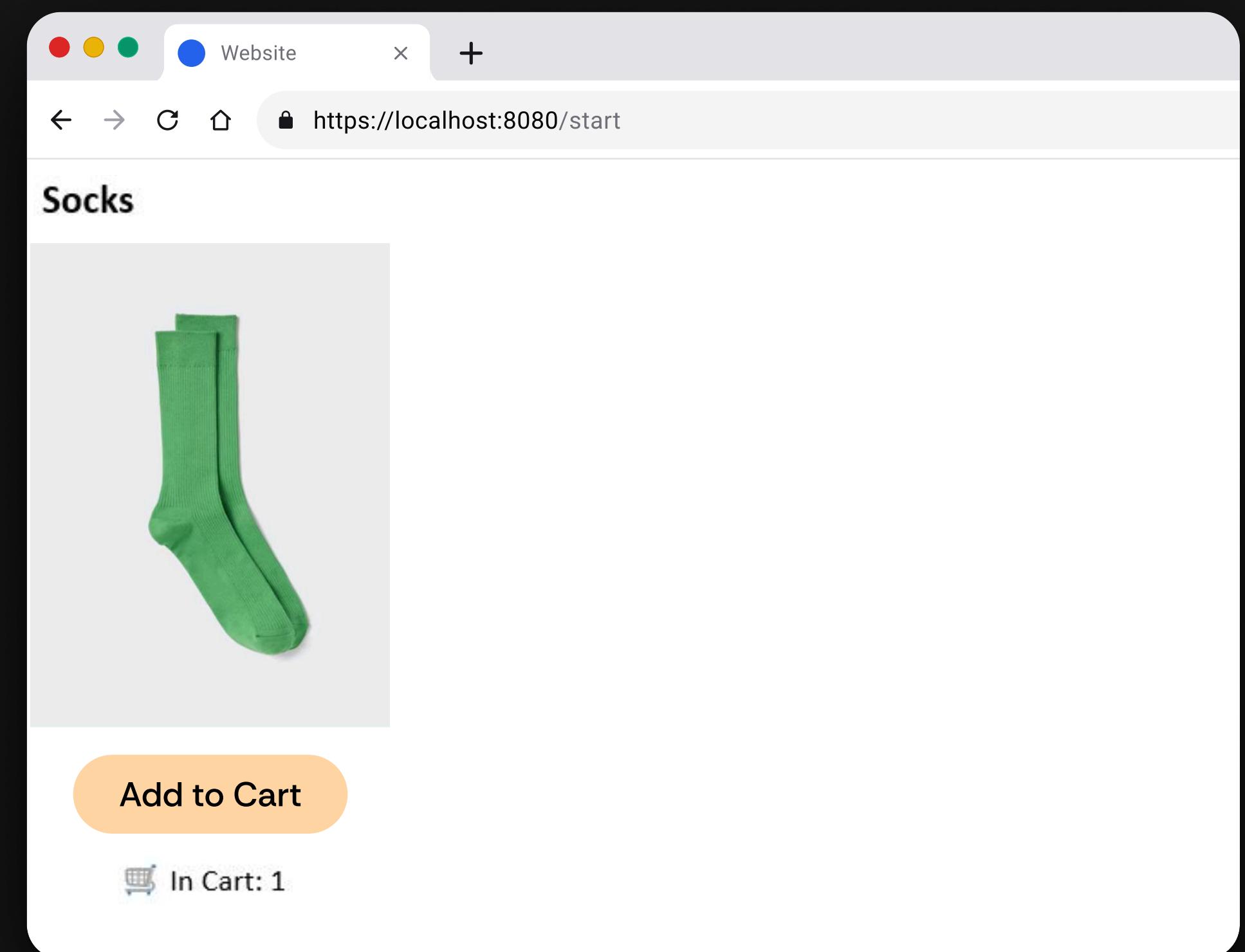
v-on

- **Funktion:** Verknüpft Benutzeraktionen mit Funktionen / Reaktion auf Events.
- **Kurzschreibweise:** 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)`.
 - Diese Variable soll das aktuell angeklickte Post-Objekt speichern.
2. Schreibe eine Funktion `openPost(post)`, die:
 - 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()`,
 - 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>
```



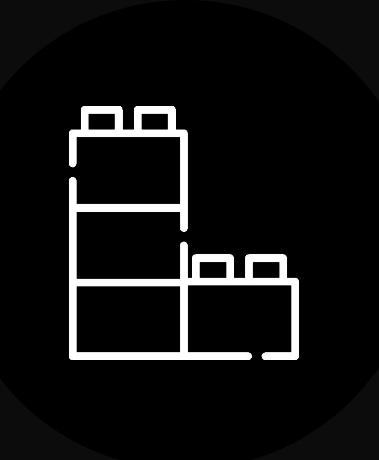
Components & Props

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

→ (Parent-)Komponente über gibt Werte als Props

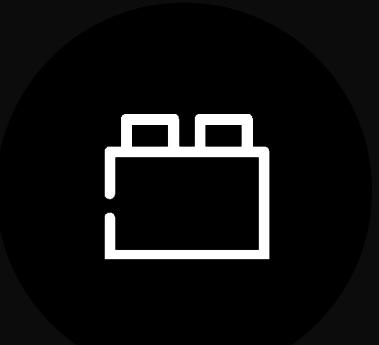
→ (Child-)Komponente empfängt diese mit defineProps()

(Parent-)Komponente



Props

(Child-)Komponente





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>

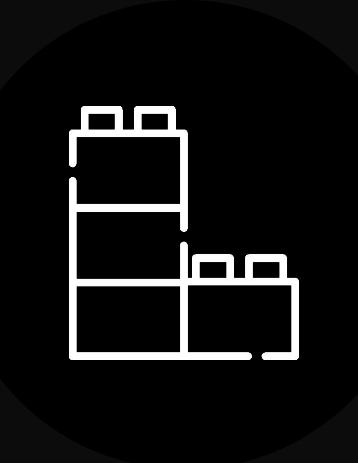
<template>
  <h1>{{ product }}</h1>
  
  <p>Shipping: {{ props.shipping }}</p>
</template>
```

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 (`@...`)

(Parent-)Komponente



(Child-)Komponente



Event



Components & Props

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')
}
```

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

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>
```





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

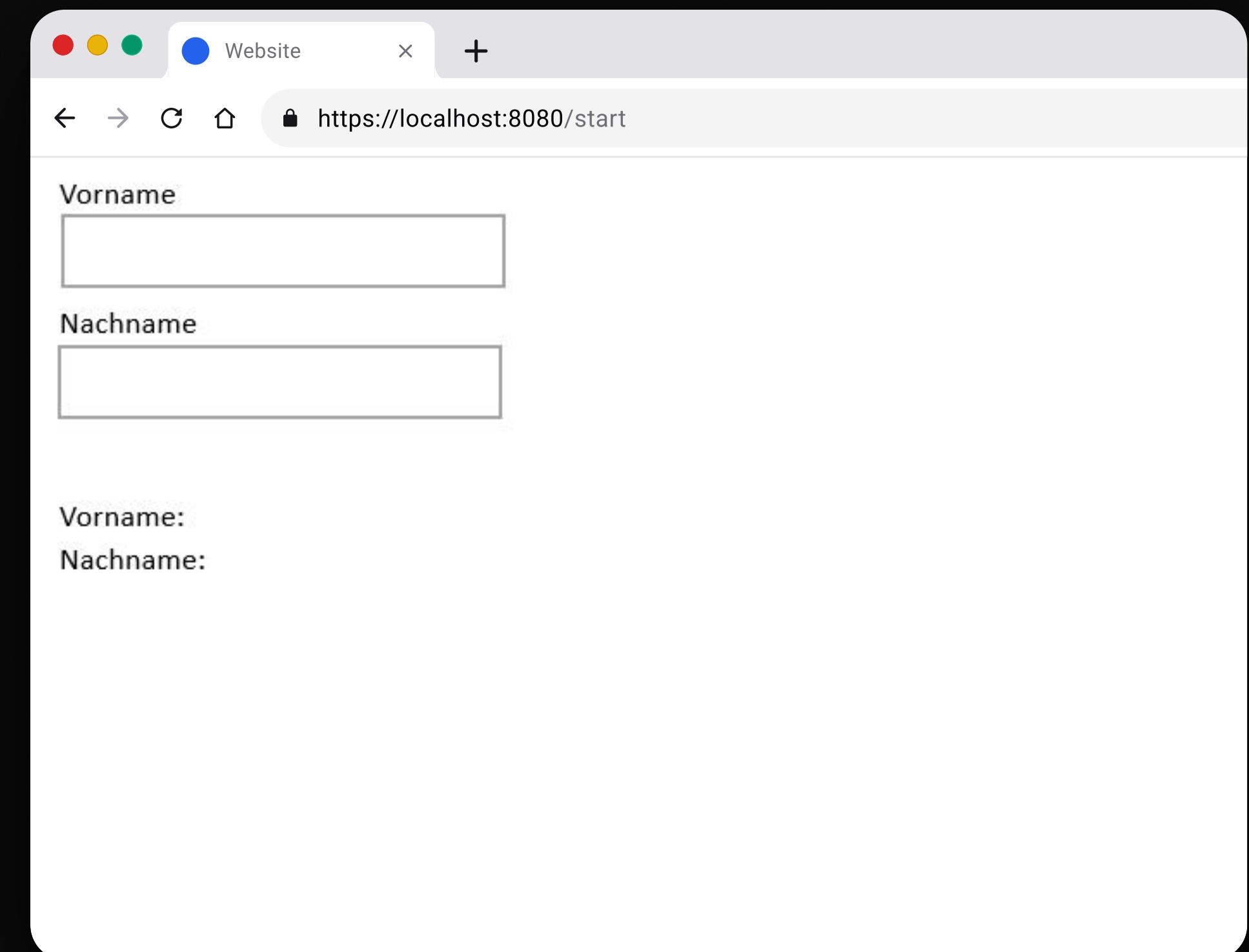
```
<script setup>
import { ref } from 'vue'

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

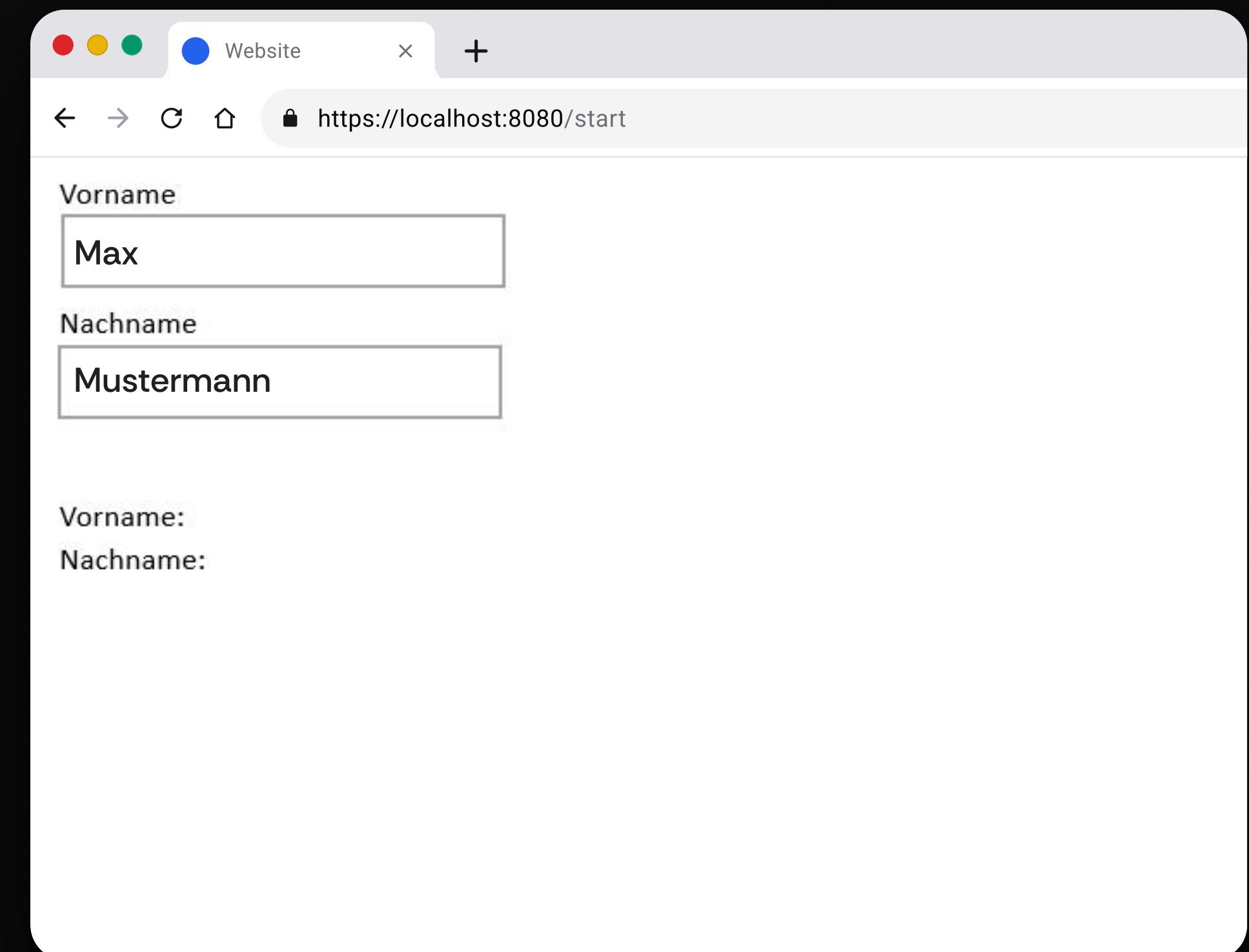
```
<script setup>
import { ref } from 'vue'

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>
```



Forms & v-model

v-model

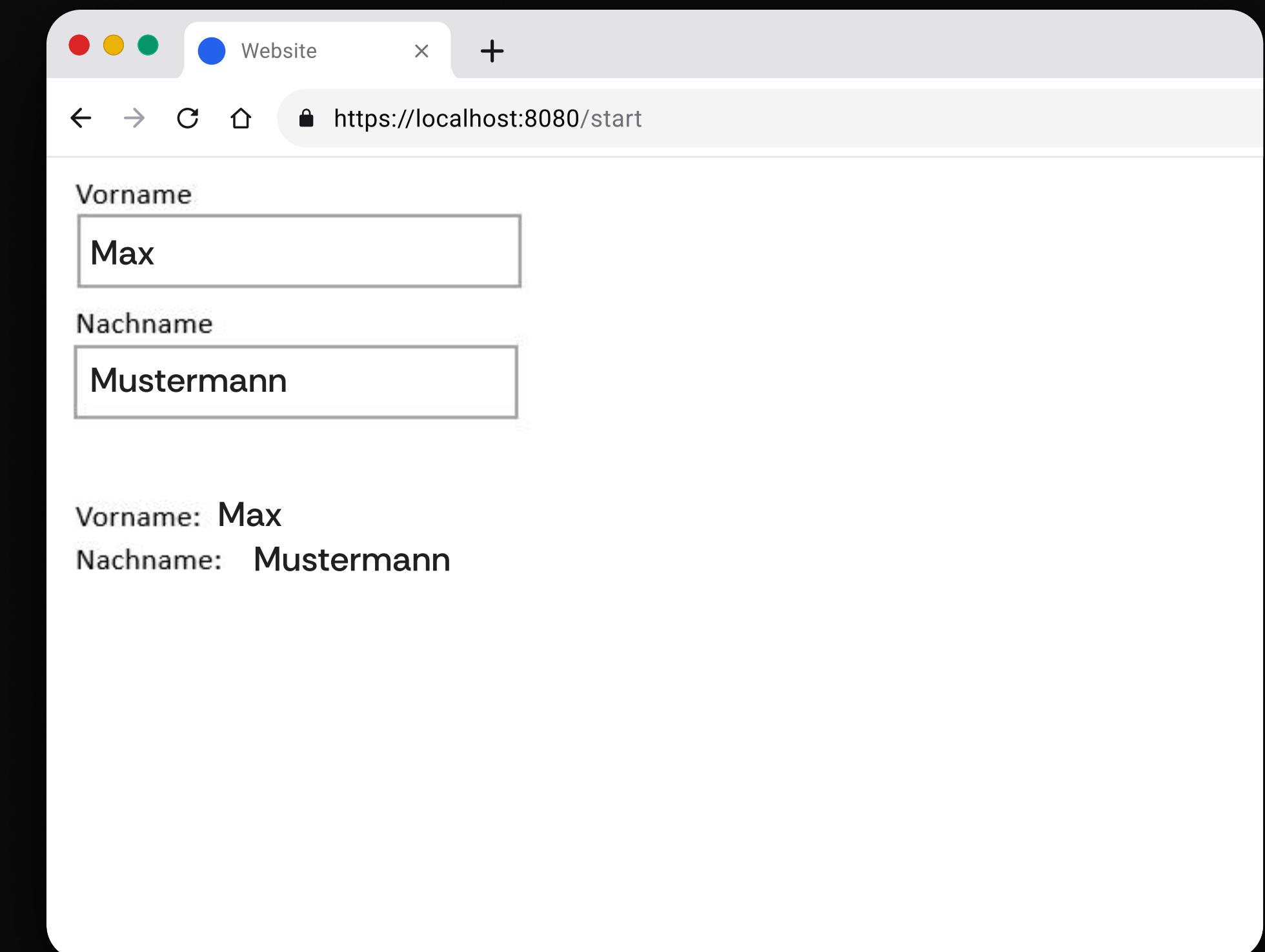
```
<script setup>
import { ref } from 'vue'

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>
```





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:
 - Verwende dazu die Direktive `v-model` im `<input>`-Element.
3. Implementiere die Logik in der Funktion `addComment()`:
 - 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?

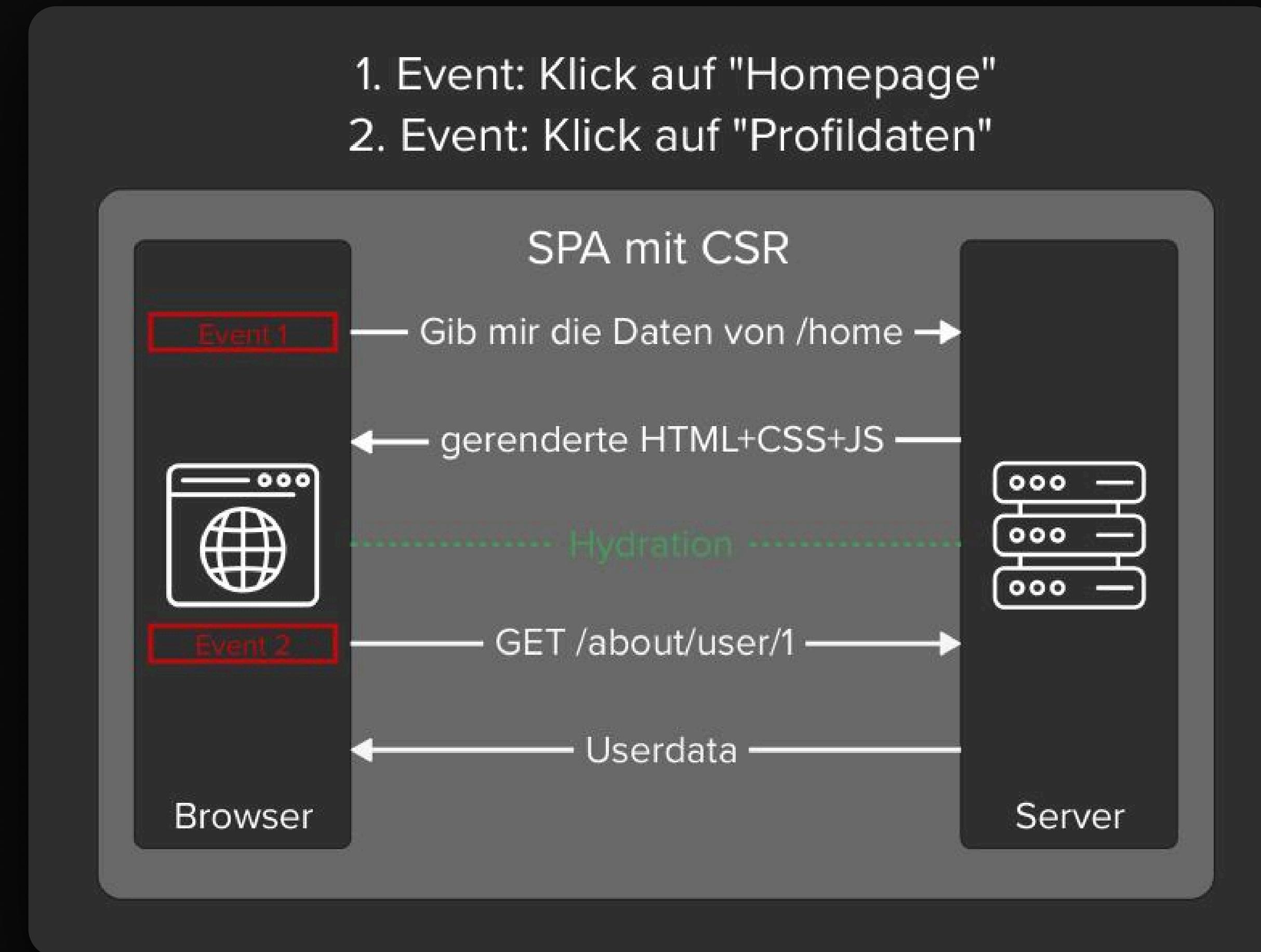
- SPA + SSR hybrid: Single Page App, die Server-Side Rendering nutzt
- Nitro rendert HTML auf dem Server für die erste Anfrage
- Nach dem Laden übernimmt Vue die Seite (Hydration)
- Navigation zwischen Seiten ohne Reload

- + SEO
- + Schnelles Laden
- + Automatisiert

“Früheres” Rendering



Nuxt Rendering



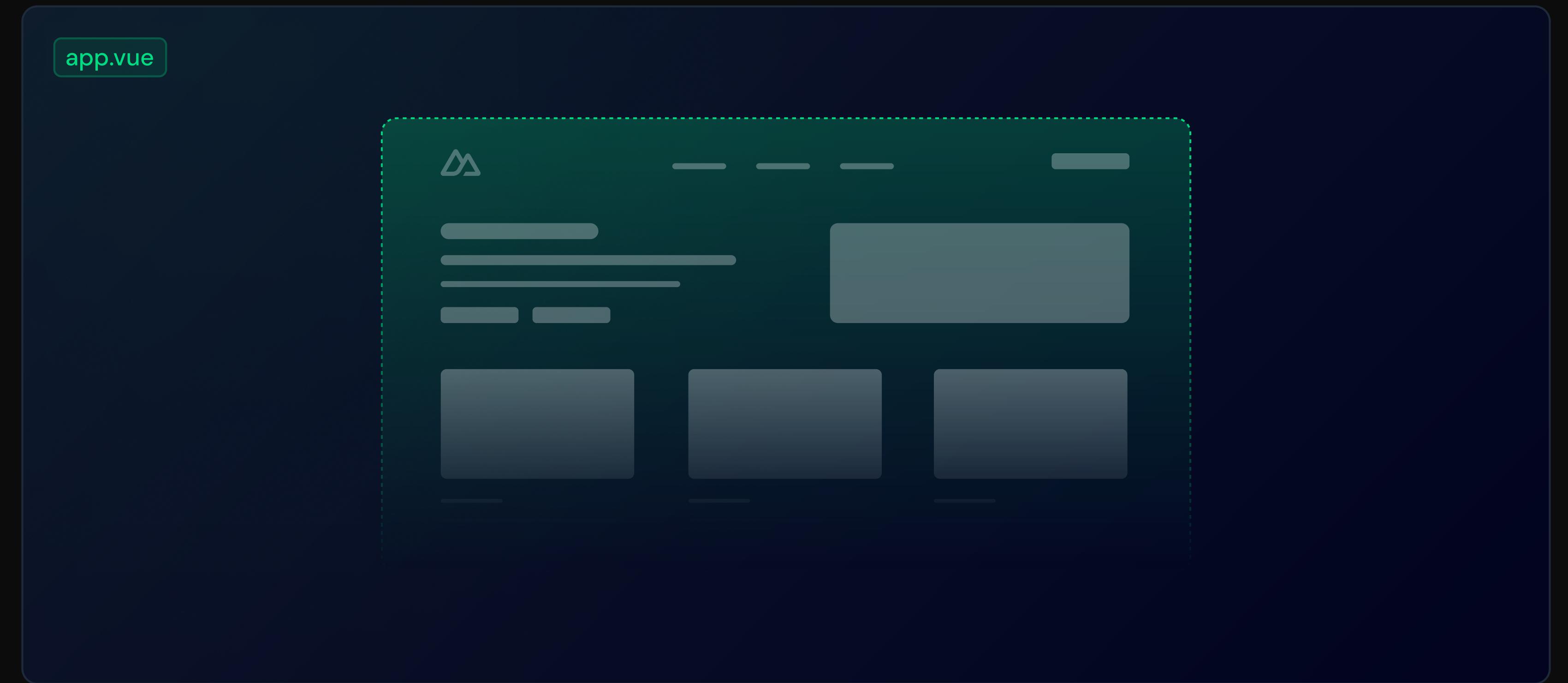


Ordnerstruktur

```
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
```

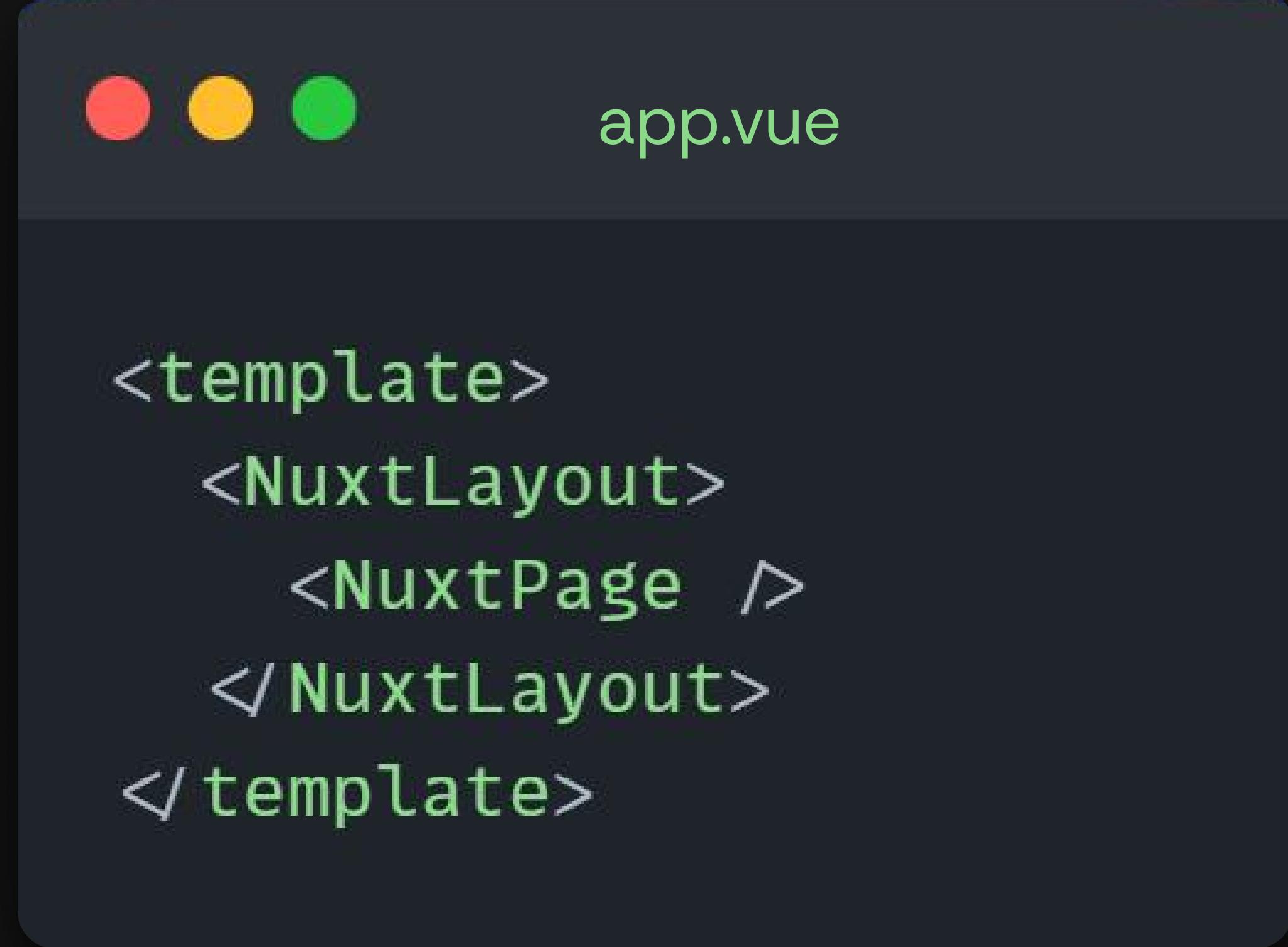


Ordnerstruktur - App





Ordnerstruktur - App

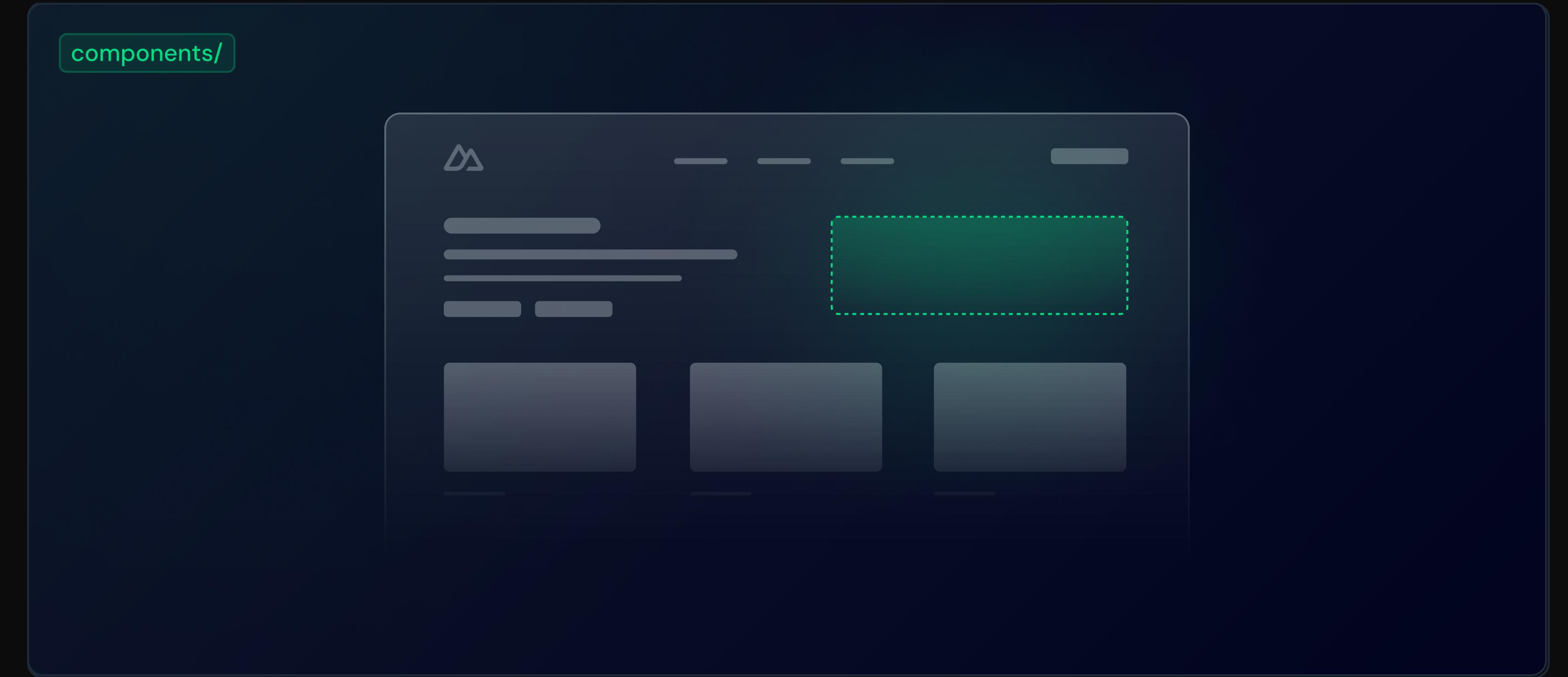


The image shows a dark gray rounded rectangle containing a file structure diagram and some code. At the top left are three colored circles (red, yellow, green) representing files. To their right is the text "app.vue". Below this is a block of code:

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



Ordnerstruktur - Komponente





Ordnerstruktur - Komponente



```
components/Banner.vue

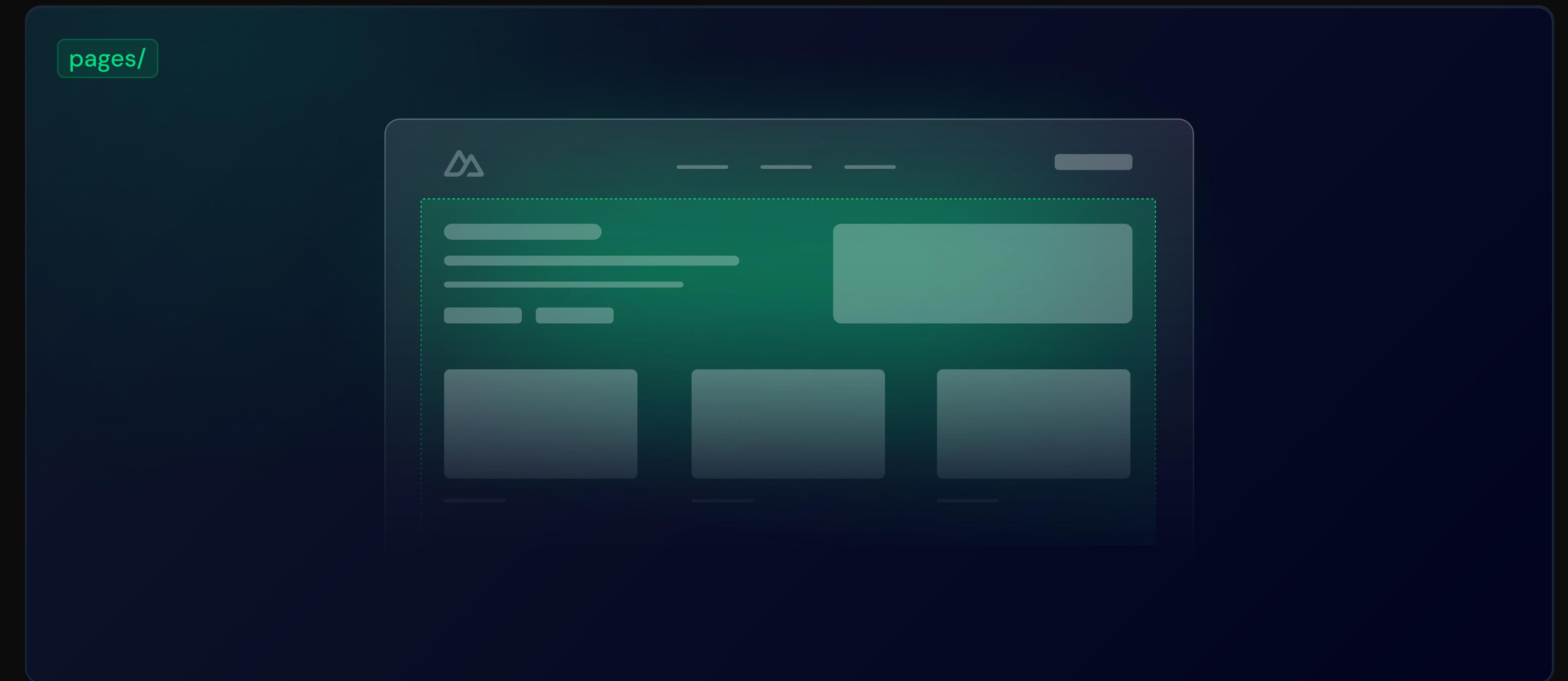
<template>
  <div class="banner">
    <p>{{ message }}</p>
  </div>
</template>

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

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



Ordnerstruktur - Pages





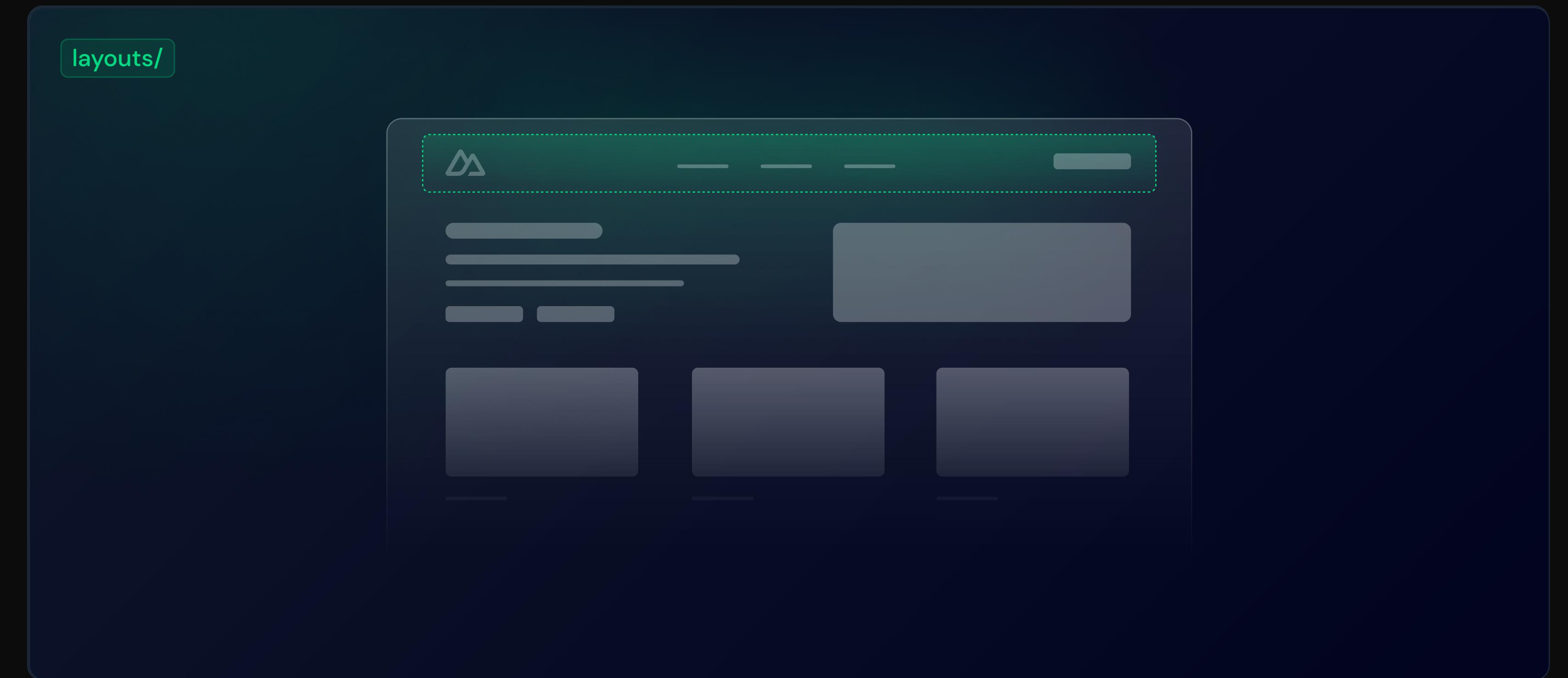
Ordnerstruktur - Pages

● ● ● pages/index.vue

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



Ordnerstruktur - Layouts





Ordnerstruktur - Layouts



```
<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 01

Schritt 1: git clone <https://github.com/byGamsa/wdw-nuxt-js.git>

Schritt 2: Projekt erstellen

Schritt 3: Evtl für Verständnis Layout und Homepage erstellen? Oder zweite Aufgabe

dafür machen und das machen, ich denke das macht Sinn



Checkerfragen



DANKE FÜR'S ZUHÖREN!