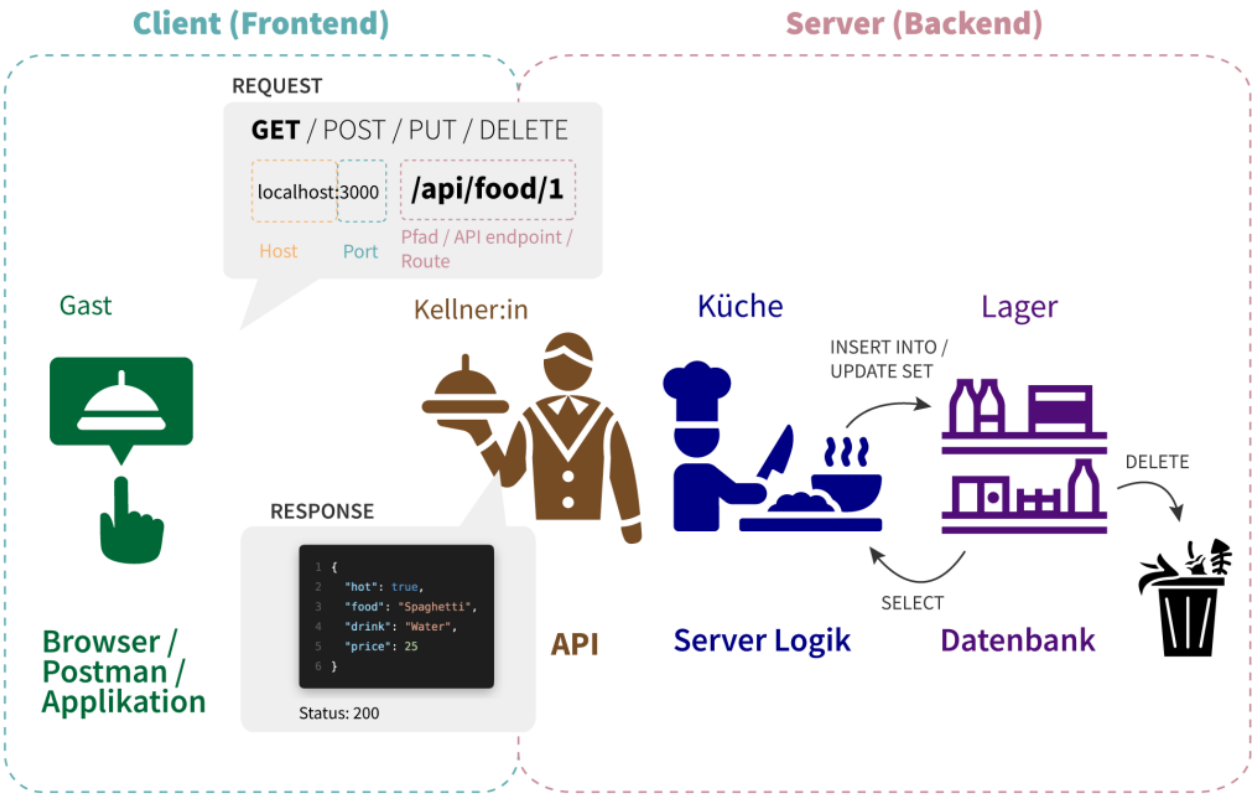
LU11 – Daten von einer API laden

Was ist eine API?

Bisher lagen alle FAQ-Daten direkt im Code – hartcodiert in einem Array. In echten Webprojekten kommen Inhalte aber fast immer von **aussen**: aus einer Datenbank, einem CMS oder einer externen Quelle.

Der Weg dorthin führt über eine **API** (Application Programming Interface) – eine Schnittstelle, über die zwei Systeme miteinander kommunizieren. Wir schicken eine Anfrage («Gib mir alle FAQs»), die API schickt die Daten zurück.



Wir (Browser / Vue App)		API / Server
«Gib mir alle FAQs»	→→ GET Request →→	Datenbank
Bekommt JSON-Daten zurück	←← Response ←←	Schickt Daten zurück

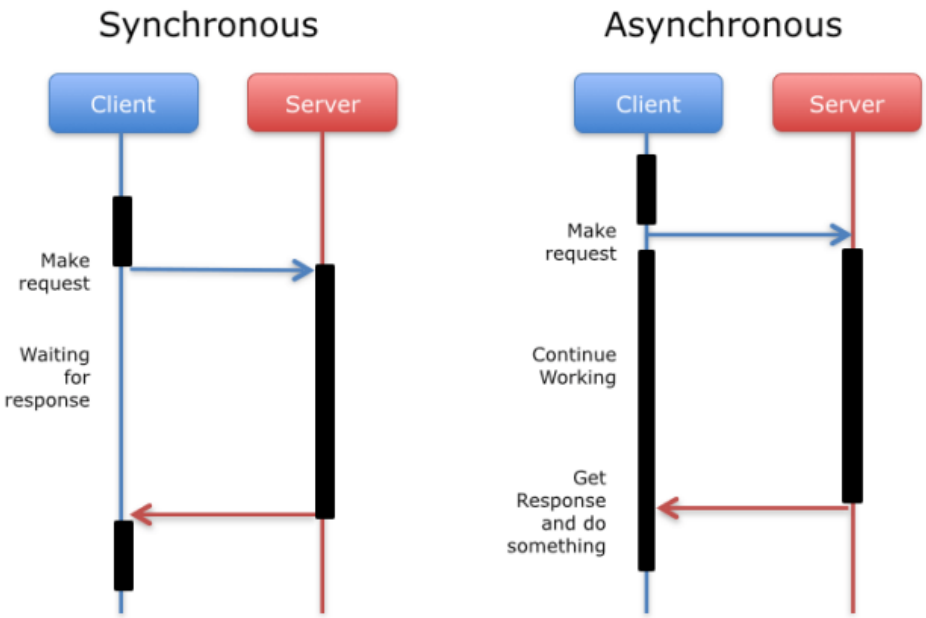
Das Problem: Netzwerkanfragen brauchen Zeit

Ein normaler JavaScript-Funktionsaufruf ist sofort fertig. Eine Anfrage ans Netzwerk dauert dagegen eine Weile – die App weiss nicht, wie lange.

Wenn JavaScript einfach **wartet**, bis die Daten da sind, friert die ganze Seite ein. Buttons reagieren

nicht mehr, nichts bewegt sich. Das wäre eine schlechte User Experience.

Die Lösung: **asynchrones JavaScript** – der Browser wartet auf die Daten, aber der Rest der App läuft währenddessen weiter.



async / await

Mit den Schlüsselwörtern `async` und `await` schreiben wir asynchronen Code, der trotzdem einfach zu lesen ist.

```
async function fetchFaqs() {
  const response = await
fetch('https://...mockapi.io/faqs');
  const data = await response.json();
  console.log(data);
}
```

Schlüsselwort	Bedeutung
`async`	Diese Funktion ist asynchron. Sie erlaubt die Verwendung von `await` und blockiert nicht den Rest der App.
`await`	Halt – warte hier, bis das Ergebnis vorliegt. Danach mach weiter. Nur innerhalb einer `async`-Funktion erlaubt.
`fetch()`	Sendet einen HTTP GET-Request an die angegebene URL und gibt ein Promise zurück.
`response.json()`	Liest den Body der Antwort und wandelt ihn von JSON in ein JavaScript-Objekt um. Auch das ist asynchron – daher auch hier `await`.

Warum zwei `await`? `fetch()` liefert zuerst nur die Antwort-Hülle (Statuscode, Header). Der eigentliche Inhalt (Body) kommt als Stream und muss erst mit `response.json()` ausgelesen werden. Beides dauert einen Moment – daher je ein `await`.

try / catch / finally

Was passiert, wenn die API nicht erreichbar ist oder einen Fehler zurückgibt? Ohne Fehlerbehandlung crashed die Applikation. Mit `try / catch` fangen wir Fehler ab und zeigen eine verständliche Meldung an.

```
try {
  // Code, der einen Fehler verursachen könnte
  const response = await fetch('https://...');
} catch (err) {
  // Wird ausgeführt, wenn im try-Block etwas schiefläuft
  console.log('Fehler:', err.message);
} finally {
  // Wird IMMER ausgeführt – egal ob Erfolg oder Fehler
  isLoading.value = false;
}
```

Block	Wann?
`try`	Enthält den Code, der fehlschlagen könnte
`catch`	Wird ausgeführt, wenn im `try`-Block ein Fehler auftritt
`finally`	Wird immer ausgeführt – ideal um z.B. `isLoading` zurückzusetzen

onMounted

Wann soll die Funktion `fetchFaqs()` ausgeführt werden? Sobald die Komponente im Browser eingebunden ist – dafür gibt es den **Lifecycle-Hook** `onMounted`:

```
import { ref, onMounted } from 'vue';

onMounted(fetchFaqs);
```

`onMounted` registriert eine Funktion, die Vue automatisch aufruft, sobald die Komponente das erste Mal im DOM gerendert wurde. Das ist der richtige Zeitpunkt, um Daten zu laden – das HTML ist bereit, die Daten können sofort angezeigt werden.

v-if / v-else-if / v-else

Beim Laden von Daten gibt es immer drei mögliche Zustände, die wir im Template abbilden:

```
<div v-if="isLoading">Daten werden geladen...</div>

<div v-else-if="error">{{ error }}</div>

<div v-else>
  <!-- Daten sind da: AccordionItems anzeigen -->
</div>
```

Direktive	Wann wird dieses Element angezeigt?
`v-if=„isLoading“`	Solange die Daten noch geladen werden
`v-else-if=„error“`	Wenn ein Fehler aufgetreten ist
`v-else`	Wenn das Laden fertig ist und kein Fehler vorliegt

Diese drei Blöcke schliessen sich gegenseitig aus – Vue zeigt immer nur einen davon an.

Der vollständige Ablauf

```
1. Komponente wird gerendert (onMounted)
        ↓
2. fetchFaqs() wird aufgerufen
        ↓
3. isLoading = true  →  «Daten werden geladen...» erscheint
        ↓
4. fetch() sendet GET-Request an MockAPI
        ↓
5. Antwort kommt zurück  →  response.json() wandelt sie um
        ↓
6. faqItems.value = data  →  Vue rendert die AccordionItems
        ↓
7. isLoading = false  →  Ladeanzeige verschwindet
```