

Notizen

stateless: unabhängige neue HTTP abfragen

Was bedeutet Applikationen? Was bedeutet Backend? Was bedeutet realisieren?

Conditional operator

Asynchrone Programmierung

Asynchrone Programmierung ermöglicht das gleichzeitige Ausführen mehrerer Aufgaben, ohne den Ablauf des Hauptprogramms zu blockieren.

Vorteile

- Nicht-blockierend: Programm kann weiterlaufen, während auf Ergebnisse gewartet wird
- Effizient: Verbesserte Leistung und Skalierbarkeit

Callbacks in JavaScript

Callbacks sind Funktionen, die als Argumente an andere Funktionen übergeben und später aufgerufen werden.

- Ermöglicht asynchrone Programmierung
- Wird oft in Verbindung mit I/O-Operationen verwendet
- Kann zu "Callback-Hölle" führen (verschachtelte Callbacks)

Promises in JavaScript

Promises repräsentieren den zukünftigen Wert einer asynchronen Operation und helfen, asynchronen Code besser zu organisieren.

- Vereinfacht das Verwalten von asynchronem Code
- Vermeidet "Callback-Hölle"
- Zustände: pending, fulfilled (erfüllt) und rejected (abgelehnt)
- Methoden: `.then()`, `.catch()` und `.finally()`
- Promise-Ketten: Verkettung mehrerer asynchroner Aktionen

Async/Await in JavaScript

Async/Await ist eine moderne Syntax zur Vereinfachung der Arbeit mit Promises, die das Schreiben von asynchronem Code übersichtlicher und lesbarer macht.

- `"async"`: Schlüsselwort, das vor einer Funktion deklariert wird
- `"await"`: Schlüsselwort, das vor einer Promise verwendet wird
- Vereinfacht das Fehlerhandling mit `"try"` und `"catch"`
- Syntaktischer Zucker: Vereinfachung der Promise-Syntax
- Macht asynchronen Code ähnlich wie synchronen Code

HTTP-Schnittstellen

- Address:** Die eindeutige URL (Uniform Resource Locator), die eine Ressource oder einen Endpunkt im Web identifiziert, zu dem die HTTP-Anfrage gesendet wird (z.B. <https://example.com/api/users>).
- Headers:** Schlüssel-Wert-Paare, die zusätzliche Informationen über den HTTP-Anforderer und die Anfrage selbst enthalten (z.B. Content-Type, Authorization).
- Methods:** HTTP-Verben, die angeben, welche Aktion auf der angeforderten Ressource durchgeführt werden soll (z.B. GET, POST, PUT, DELETE).
- Body:** Daten, die bei bestimmten HTTP-Methoden (z.B. POST, PUT) an den Server gesendet werden, um Ressourcen zu erstellen oder zu aktualisieren.
- Status Codes:** Dreistellige Zahlen, die das Ergebnis der HTTP-Anfrage anzeigen (z.B. 200 OK, 404 Not Found, 500 Internal Server Error).

Das HTTP Protokoll

HyperText Transfer Protocol

http://info.cern.ch - home of the first website

- Seit Anfang an die Art, wie der Browser mit dem WWW spricht
- Client/Server mit Request/Response
- Stateless
- Sicher nur Dank der Erweiterung HTTPS
- Standardports (HTTP: 80; HTTPS: 443) werden im Browser nicht angezeigt

HTTP Headers

Zusätzliche Informationen zum **Request** und zur **Response** können in sogenannten Header hinterlegt werden. Übliche Header sind:

Request	Response
<ul style="list-style-type: none">HostRefererUser-AgentAccept, Accept-Encoding, Accept-LanguageAuthorization	<ul style="list-style-type: none">Content-TypeServerCache-ControlKeep-AliveSet-Cookie

Content-Types

Die Art des Inhalts wird mit dem `"Content-Type"` Header beschrieben, damit der Client weiss, was damit zu tun ist. Die gängigsten `"Format"`-sind:

- HTML: `text/html, charset=utf-8`
- JSON: `application/json`
- XML: `text/xml, application/xml`
- Bilder, Videos und andere Formate haben eigenen Content-Type

Asynchrone Programmierung

Möglichkeiten um asynchrone Programmierung umzusetzen

Möglichkeiten um asynchrone Programmierung umzusetzen:

- Callbacks:** Funktionen, die nach Abschluss einer Aufgabe aufgerufen werden
- Promises:** Objekte, die den Erfolg oder Misserfolg einer asynchronen Operation repräsentieren
- Async/Await:** Moderne Syntax, um asynchrone Funktionen übersichtlicher zu gestalten

Beispielcode zu Promises

```
1 function greifeAufDatenbankZu(query) {
2   return new Promise((resolve, reject) => {
3     // Datenbankzugriff simulieren
4     setTimeout(() => {
5       const ergebnis = 'Datenbankergebnis: ' + query;
6       resolve(ergebnis);
7     }, 1000);
8   });
9 }
10
11 greifeAufDatenbankZu('SELECT * FROM users')
12   .then(ergebnis => {
13     console.log(ergebnis);
14   })
15   .catch(err => {
16     console.error('Fehler beim Datenbankzugriff:', err);
17   });
```

Beispielcode zu Callbacks

```
1 function greifeAufDatenbankZu(query, callback) {
2   // Datenbankzugriff simulieren
3   setTimeout(() => {
4     const ergebnis = 'Datenbankergebnis: ' + query;
5     callback(ergebnis);
6   }, 1000);
7 }
8
9 greifeAufDatenbankZu('SELECT * FROM users', (ergebnis) => {
10   console.log(ergebnis);
11 });
```

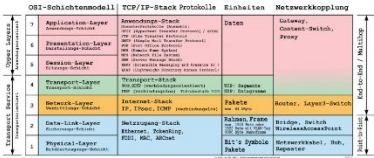
Beispielcode zu Async/Await

```
1 function greifeAufDatenbankZu(query) {
2   return new Promise((resolve, reject) => {
3     // Datenbankzugriff simulieren
4     setTimeout(() => {
5       const ergebnis = 'Datenbankergebnis: ' + query;
6       resolve(ergebnis);
7     }, 1000);
8   });
9 }
10
11 async function verarbeiteDatenbankAnfrage() {
12   try {
13     const ergebnis = await greifeAufDatenbankZu('SELECT * FROM users');
14     console.log(ergebnis);
15   } catch (err) {
16     console.error('Fehler beim Datenbankzugriff:', err);
17   }
18 }
19
20 verarbeiteDatenbankAnfrage();
```



Softwarechnittstellen

- Es wird zwischen internen und externen Schnittstellen unterschieden.
- Intern ist, wenn ein Programm mit einem anderen Programm auf demselben Gerät kommuniziert.
- Extern ist, wenn ein Programm mit einem anderen Programm über das Netzwerk kommuniziert.



HTTP Methoden nach CRUD

Clients geben die Art der Anfrage mit sog. Methoden (manchmal auch Verben genannt) bekannt:

Anfrage	Methode
Erstellen (Create)	POST
Lesen (Read)	GET
Ändern/Ersetzen (Update/Replace)	PUT
Ändern/Anpassen (Update/Patch)	PATCH
Löschen (Delete)	DELETE

Softwareverteilungsarchitekturen

- Standalone-Anwendungen:** Unabhängige Programme, die auf einem einzelnen Computer laufen und keine Netzwerkkommunikation benötigen.
 - Beispiel: Texteditor, Taschenrechner.
- Client-Server-Anwendungen:** Anwendungen, die auf einer Client-Server-Architektur basieren, wobei der Client Anfragen an den Server sendet und der Server darauf reagiert.
 - Beispiel: Webbrowser und Webserver, E-Mail-Clients und -Server.
- Peer-to-Peer-Anwendungen (P2P):** Anwendungen, bei denen alle Teilnehmer gleichberechtigt sind und direkt miteinander kommunizieren, ohne zentralen Server.
 - Beispiel: BitTorrent, Blockchain-Netzwerke.



Funktionen

In JavaScript können funktionen auf verschiedene Arten definiert werden. Alle Beispiele bezwecken das Gleiche!

1. "Normale" Funktion:

```
function addiere(zahl1: number, zahl2: number) {
  return zahl1 + zahl2;
}

addiere(5, 2); // Resultat: 7
```

2. "Normale" anonyme Funktion:

```
const addiere = function (zahl1: number, zahl2: number) {
  return zahl1 + zahl2;
}

addiere(5, 2); // Resultat: 7
```

3. "Arrow"-Funktion:

```
const addiere = (zahl1: number, zahl2: number) => {
  return zahl1 + zahl2;
};

addiere(5, 2); // Resultat: 7
```

4. "Arrow"-Funktion mit abgekürztem `"return"`:

```
const addiere = (zahl1: number, zahl2: number) => zahl1 + zahl2;
```