Notizen (git: https://github.com/fatma-malik/M295)

stateless: unabhängige neue HTTP abfragen

statuscodes, https port: 443

Conditional operator: variablename = (condition) ? value1:value2

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

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 Callbacks

```
function greifeAufDatenbankZu(query, callback) {
  // Datenbankzugriff simulieren
  setTimeout(() => {
    const ergebnis = 'Datenbankergebnis: ' + query;
    callback(ergebnis);
  }, 1000);
}

greifeAufDatenbankZu('SELECT * FROM users', (ergebnis) => {
  console.log(ergebnis);
});
```

Beispielcode zu Promises

```
function greifeAufDatenbankZu(query) {
  return new Promise((resolve, reject) => {
    // Datenbankzugriff simulieren
    setTimeout(() => {
      const ergebnis = 'Datenbankergebnis: ' + query;
      resolve(ergebnis);
    }, 1000);
  });
}

greifeAufDatenbankZu('SELECT * FROM users')
  .then(ergebnis => {
    console.log(ergebnis);
  })
  .catch(err => {
    console.error('Fehler beim Datenbankzugriff:', err);
  });
```

Beispielcode zu Async/Await

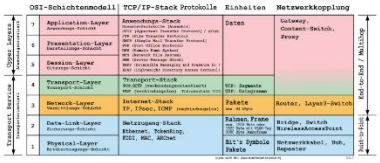
```
function greifeAufDatenbankZu(query) {
  return new Promise((resolve, reject) => {
    // Datenbankzugriff simulieren
    setTimeout(() => {
      const ergebnis = 'Datenbankergebnis: ' + query;
      resolve(ergebnis);
    }, 1000);
  });
}

async function verarbeiteDatenbankanfrage() {
  try {
    const ergebnis = await greifeAufDatenbankZu('SELECT * FROM users');
    console.log(ergebnis);
  } catch (err) {
    console.error('Fehler beim Datenbankzugriff:', err);
  }
}

verarbeiteDatenbankanfrage();
```

Softwareschnittstellen

- 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-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).



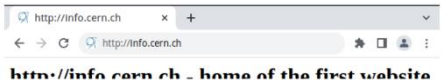
HTTP Methoden nach CRUD

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

Das HTTP Protokoll

HyperText Transfer Protocol



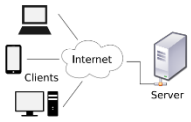
- Seit Anfang an die Art, wie der Browser mit dem WWW spricht
- Client/Server mit Request/Response
- Stateless
- Sicher nur Dank der Erweiterung HTTPS
- Standardports (HTTP: 80; HTTPS: 443) werden im Browser nicht angezeigt

HTTP Statuscodes

Server senden mit der Antwort einen Statuscode mit.

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)

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.

HTTP Headers

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

Request

- Host
- Referer
- User-Agent
- Accept, Accept-Encoding, Accept-Language
- Authorization

Response

- Content-Type
- Server
- Cache-Control
- Keep-Alive
- Set-Cookie

Content-Types

Die Art des Inhalts wird mit dem `Content-Type` Header beschrieben, damit der Client weiss, was damit zu tun ist. Die gängisten «Formate» sind:

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

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;
```