Backend für Applikationen Realisieren

* **Applikationen**: Applikationen sind Softwareprogramme oder Anwendungen, die spezifische Funktionen oder Dienste ausführen, um Benutzerbedürfnisse zu erfüllen. Sie können auf verschiedenen Plattformen wie Desktops, Mobilgeräten oder Webbrowsern laufen.
* **Backend**: Das Backend einer Softwareanwendung bezieht sich auf den Teil, der im Hintergrund läuft und sich um die Verarbeitung von Daten, die Geschäftslogik und die Interaktion mit der Datenbank kümmert. Es ist oft für die Serverseite einer Anwendung verantwortlich.
* **realisieren**: Realisieren bedeutet, etwas in die Tat umzusetzen oder zu verwirklichen. In Bezug auf Softwareentwicklung bedeutet es, eine Idee, Spezifikation oder Anforderung in funktionierenden Code oder ein lauffähiges Produkt umzusetzen.

# Block 1

* **Was macht ein Backend aus?**: Das Backend einer Softwareanwendung befasst sich mit der Verarbeitung von Daten, der Geschäftslogik und der Interaktion mit der Datenbank. Es stellt die grundlegenden Funktionen bereit, die von der Anwendung benötigt werden, und verarbeitet die Anfragen des Frontends.
* **Wo läuft ein Backend?**: Das Backend läuft auf einem Server, der normalerweise remote gehostet wird. Es kann auf physischen Servern oder in der Cloud betrieben werden.
* **Wie ist das Backend mit dem Frontend verbunden?**: Das Backend und das Frontend kommunizieren über definierte Schnittstellen, häufig über HTTP-Anfragen. Das Frontend sendet Anfragen an das Backend, das dann die entsprechenden Daten oder Ressourcen zurückgibt.
* **Was wissen Sie noch vom Modul 294?**: Ich habe Zugriff auf eine Vielzahl von Informationen, die bis zu meinem letzten Aktualisierungsdatum im Januar 2022 verfügbar waren. Das Modul 294 könnte sich auf Webentwicklung oder verwandte Themen beziehen, aber ich brauche mehr Kontext, um genauere Informationen bereitzustellen.
* **Welche Technologien braucht man im Backend? Welche Frameworks?**: Im Backend können verschiedene Technologien verwendet werden, darunter Programmiersprachen wie JavaScript (Node.js), Python (Django, Flask), Java (Spring), Ruby (Ruby on Rails) und viele mehr. Zur Entwicklung von Backend-Anwendungen können auch Frameworks wie Express.js (für Node.js), Django (für Python), Spring Boot (für Java) usw. eingesetzt werden.

## Einstiegsquiz

Wo kann JavaScript ausgeführt werden?  
JavaScript kann in Browsern, auf Servern (mit Node.js), in mobilen Apps (mit Frameworks wie React Native) und auf vielen anderen Plattformen ausgeführt werden.

Für JavaScript im Browser: Welches HTML-Tag wird verwendet, um eine externe JavaScript-Datei in eine Webseite einzubinden?  
**<script>**-Tag

Welche Funktion wird verwendet, um "Hello, world!" in einem Alert-Fenster anzuzeigen? Schreiben Sie den ganzen Funktionsaufruf.  
**alert("Hello, world!");**

Welche Funktion wird verwendet, um etwas auf der (Entwickler-)Konsole auszugeben?  
**console.log()**

Welches Symbol wird verwendet, um eine Anweisung in JavaScript zu beenden?  
Semikolon **;**

Können JavaScript-Anweisungen über mehrere Zeilen geschrieben werden?  
Ja

Wie werden mehrzeilige Kommentare in JavaScript formatiert?  
**/\* Kommentar \*/**

Wie deklariert man eine Variable in JavaScript?  
**var**, **let**, **const**

Welche dieser sind keine gültigen Variablennamen in JavaScript?  
**123variable**, **my-variable**, **my\_variable**, **123**

Was sind die Unterschiede zwischen var, let und const?  
**var** ist funktions- oder global-scope, **let** und **const** sind block-scope; **var** kann neu deklariert werden, **let** und **const** nicht; **const** ist unveränderlich.

Warum könnte man let über var in einer Schleife bevorzugen?  
**let** hat block scope, was das Verhalten in Schleifen und verschachtelten Blöcken besser kontrolliert.

Ordnen Sie die primitiven Datentypen in JavaScript richtig zu:  
Fliesskommazahlen, Listen, Datenstrukturen, Wahrheitswerte, Texte, Ganzzahlen  
Fliesskommazahlen, Texte, Ganzzahlen, Wahrheitswerte, Listen, Datenstrukturen

Wie ermittelt man den Typ eines Variablenwerts in JavaScript?  
**typeof**

Was ist das Ergebnis der Operation "1" + 2 in JavaScript?  
**"12"**

Was ist ein NaN-Wert und wie kann er überprüft werden?  
NaN ist "Not a Number", wird mit **isNaN()** überprüft.

Welche Funktion wird verwendet, um eine einfache Nachrichtenbox (ohne Input) für den Benutzer anzuzeigen?  
**alert()**

Was ist der Standardrückgabewert der Funktion confirm(), wenn der Abbrechen-Knopf gedrückt wird?  
**false**

Was ist der Unterschied zwischen alert() und confirm()?  
**alert()** zeigt eine Nachricht an, **confirm()** erfordert eine Bestätigung.

Wie prüft man den Wert, der von prompt() zurückgegeben wird?  
Vergleichen mit **null**.

Welche Funktion wird verwendet, um einen String in eine ganze Zahl zu konvertieren?  
**parseInt()**

Was ist das boolesche Umwandlungsergebnis eines leeren Strings?  
**false**

Was ist das Ergebnis von Boolean("0")?  
**true**

Wie wirkt sich die Typumwandlung auf den Vergleichsoperator == aus?  
Mit Zahlen führt **==** Typumwandlung durch, mit Strings führt er eine Konkatenation durch.

Was macht der + Operator, wenn er mit Zahlen verwendet wird? Und mit Strings?  
Mit Zahlen führt **+** Addition durch, mit Strings führt er eine Konkatenation durch.

Wie würde man eine Variable x um 1 erhöhen, indem man einen Kurzschreiboperator verwendet?  
**x++**

Erklären Sie die Präzedenzregeln mit einem Beispiel, das die Operatoren +, \* und = verwendet.  
**(a + b) \* c = d**

Was ist der Unterschied zwischen den Operatoren == und ===?  
**==** führt Typumwandlung durch, **===** vergleicht Typen.

Wie schreibt man eine einfache if-Anweisung, um zu überprüfen, ob eine Variable a grösser als 10 ist?  
**if (a > 10)**

Schreiben Sie ein Beispiel für eine if-else-Anweisung, die verschiedene Nachrichten basierend auf dem Wert einer Variablen ausgibt.

javascript

Copy code

if (variable === 1) { console.log("Nachricht 1"); } else if (variable === 2) { console.log("Nachricht 2"); } else { console.log("Nachricht 3"); }

Was macht der && Operator?  
logisches UND

Was gibt true || false zurück?  
**true**

Schreibe eine einfache for-Schleife, die die Zahlen 1 bis 5 ausgibt.

javascript

Copy code

for (let i = 1; i <= 5; i++) { console.log(i); }

Wie beendet man eine Schleife frühzeitig mit einer Steueranweisung?  
**break;**

Erkläre den Unterschied zwischen while und do-while Schleifen.  
**while** prüft die Bedingung vor der Ausführung, **do-while** prüft sie nach der Ausführung.

Wie wird eine Funktion in JavaScript deklariert?  
**function functionName() {}**

Was ist eine anonyme Funktion in JavaScript?  
Eine Funktion ohne Namen.

Was ist der Standardrückgabewert einer Funktion in JavaScript, die kein return-Statement verwendet?  
**undefined**

### Funktionen



### Conditional (ternary) Operator



### Was ist NodeJS?

NodeJS ist eine Open-Source-Laufzeitumgebung, die auf der V8-JavaScript-Engine von Google basiert.

Ursprung: 2009 von Ryan Dahl entwickelt, welcher V8 aus dem Chrome-Browser extrahiert hat.

Plattformübergreifend: Windows, macOS, Linux

Ereignisgesteuert: asynchrone I/O-Operationen

Anwendungsbereiche: Webserver, APIs, Echtzeitanwendungen

NPM: Node Package Manager zur Verwaltung von Modulen

Alternativen: Deno, Bun

## Learnyounode

Baby-steps.js

const args = process.argv.slice(2);

let sum = 0;

for (let i = 0; i < args.length; i++) {

sum += Number(args[i]);

}

console.log(sum)

my-first-io.js

const fs = require('fs')

const filePath = process.argv[2];

const contentBuffer = fs.readFileSync(filePath);

const contentString = contentBuffer.toString();

const numberOfLines = contentString.split('\n').length - 1;

console.log(numberOfLines)

my-first-async-io.js

var fs = require('fs')

const filePath = process.argv[2];

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) {

console.error(err);

return;

}

const numberOfLines = data.split('\n').length - 1;

console.log(numberOfLines);

});

Filtered-ls.js

const fs = require('fs');

const path = require('path');

const directoryPath = process.argv[2];

const fileExtension = '.' + process.argv[3];

fs.readdir(directoryPath, (err, files) => {

if (err) {

console.error('Error reading directory:', err);

return;

}

const filteredFiles = files.filter(file => path.extname(file) === fileExtension);

filteredFiles.forEach(file => console.log(file));

});

http-client.js

const http = require('http');

const url = process.argv[2];

http.get(url, (response) => {

response.setEncoding('utf8');

response.on('data', (data) => {

console.log(data);

});

});

http-collect.js

const http = require('http');

const url = process.argv[2];

http.get(url, (response) => {

let data = '';

response.on('data', (chunk) => {

data += chunk;

});

response.on('end', () => {

console.log(data.length);

console.log(data);

});

});

Juggling-async.js

const http = require('http');

const results = [];

let count = 0;

function getData(url, index) {

http.get(url, (response) => {

let data = '';

response.on('data', (chunk) => {

data += chunk;

});

response.on('end', () => {

results[index] = data;

count++;

if (count === 3) {

printResults();

}

});

}).on('error', (error) => {

console.error('Fehler beim Abrufen der Daten:', error);

});

}

function printResults() {

for (let result of results) {

console.log(result);

}

}

for (let i = 0; i < 3; i++) {

getData(process.argv[2 + i], i);

}

Time-server.js

const net = require('net');

function formatDate(date) {

const year = date.getFullYear();

const month = String(date.getMonth() + 1).padStart(2, '0');

const day = String(date.getDate()).padStart(2, '0');

const hours = String(date.getHours()).padStart(2, '0');

const minutes = String(date.getMinutes()).padStart(2, '0');

return `${year}-${month}-${day} ${hours}:${minutes}`;

}

const server = net.createServer((socket) => {

const date = new Date();

const formattedDate = formatDate(date);

socket.end(`${formattedDate}\n`);

});

const port = process.argv[2];

server.listen(port);

# Block 2

## 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

### Möglichkeiten um asynchrone Progammierung umzusetzten

**Callbacks:** Funktionen, die nach Abschluss einer Aufgabe aufgerufen werden

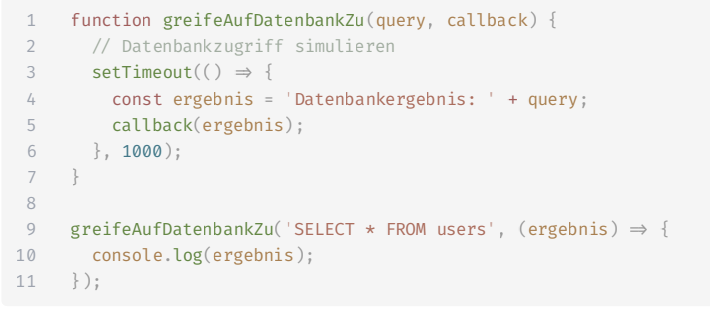
**Promises:** Objekte, die den Erfolg oder Misserfolg einer asynchronen Operation repräsentieren

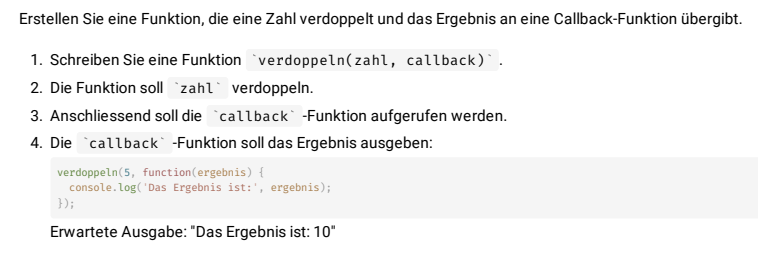
**Async/Await:** Moderne Syntax, um asynchrone Funktionen übersichtlicher zu gestalten

## Callbacks

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)





function verdoppeln(zahl, callback) {

callback(zahl \* 2);

}

verdoppeln(5, function(ergebnis) {

console.log('Das Ergebnis ist:', ergebnis);

});

## Promises

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





const fs = require('node:fs');

function leseDateiInhalt(filepath) {

return new Promise((resolve, reject) => {

fs.readFile(filepath, 'utf8', (err, data) => {

if (err) {

reject(err);

} else {

resolve(data);

}

});

});

}

leseDateiInhalt('./promises.txt')

.then(inhalt => {

console.log('Die Länge des Dateiinhalts beträgt:', inhalt.length);

})

.catch(err => {

console.error('Fehler beim Lesen der Datei:', err);

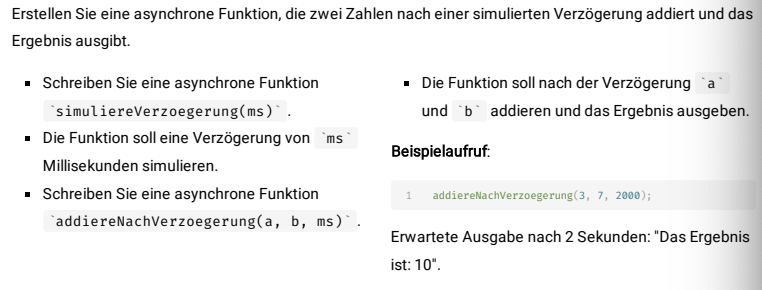
});

## Async/Await

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





async function simuliereVerzoegerung(ms) {

return new Promise(resolve => setTimeout(resolve, ms));

}

async function addiereNachVerzoegerung(a, b, ms) {

await simuliereVerzoegerung(ms);

const ergebnis = a + b;

console.log('Das Ergebnis ist:', ergebnis);

}

addiereNachVerzoegerung(3, 7, 2000);

# Block 3

# Block 4

# Block 5