Node.js & Express.js – Mitschrift

1. Einführung in Node.js und Express.js

(00:00) **Introduction**
(01:41) **What Is Node**
(02:56) **Course Requirements**
(04:16) **Course Structure**
(04:59) **Browser vs Server**

- **Node.js ist eine serverseitige JavaScript-Laufzeitumgebung**, die auf der V8-Engine basiert.
- **Express.js ist ein Web-Framework für Node.js**, das das Erstellen von APIs und Webservern erleichtert.
- **Unterschied zwischen Browser- und Server-JavaScript:**
  - **Browser**: DOM-Manipulation, UI-Interaktionen
  - **Server**: Dateioperationen, Datenbanken, HTTP-Requests
- **Kursvoraussetzungen**: Grundkenntnisse in JavaScript helfen, sind aber nicht zwingend nötig.

2. Installation & CLI

(07:50) **Install Node**
(11:08) **REPL (Read-Eval-Print Loop)**
(13:27) **CLI (Command Line Interface)**
(19:07) **Source Code**

Installation von Node.js

1. **Herunterladen & installieren**: https://nodejs.org
2. **Überprüfen der Installation:**

   ```
   node -v
   npm -v
   ```

3. **REPL (Node.js-Terminal) starten:**

   ```
   node
   ```

   - Direkt JS-Code ausführen: `console.log(2 + 2) // 4`

3. Module & Built-In Features

(20:27) **Globals**
(29:34) **Modules Setup**
(32:46) **First Module**
(45:32) **Alternative Syntax**
(49:50) **Mind Grenade**
(53:47) **Built-In Modules**

(56:31) **OS Module**
(1:04:13) **Path Module**
(1:10:06) **FS Module (Sync & Async)**

Globale Objekte in Node.js

- `__dirname`: Gibt den Pfad des aktuellen Verzeichnisses zurück
- `__filename`: Gibt den vollständigen Dateipfad zurück

```
console.log(__dirname);  // /User/MeinProjekt
console.log(__filename); // /User/MeinProjekt/index.js
```

Module in Node.js

- In Node.js können wir eigene Module schreiben:

```
// utils.js
const sagHallo = (name) => `Hallo, ${name}`;
module.exports = sagHallo;
```

```
// index.js
const sagHallo = require('./utils');
console.log(sagHallo('Max'));
```

- **Wichtige Built-In Module**:
  - `fs`: Dateioperationen
  - `path`: Arbeiten mit Dateipfaden
  - `os`: Systeminfos abrufen

4. Asynchrones Node.js

(1:27:32) **Sync vs Async**
(2:27:38) **Event Loop**
(2:37:46) **Async Patterns**
(3:06:05) **Node's Native Option**

- **Asynchrone Datei lesen:**

```
const fs = require('fs');
fs.readFile('datei.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log(data);
});
```

- **Event Loop ermöglicht nicht-blockierendes Verhalten in Node.js**.

5. HTTP & Webserver

(1:34:29) **HTTP Intro**
(1:35:58) **HTTP Module (Setup & Features)**

(3:40:46) **HTTP Request/Response Cycle**
(4:03:25) **HTTP Basics**

- **Ein einfacher Node.js-Server:**

```
const http = require('http');
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hallo Welt!');
});
server.listen(3000, () => console.log('Server läuft auf Port 3000'));
```

6. Express.js

(4:48:02) **Express Info**
(4:51:50) **Express Basics**
(5:03:05) **Express - App Example**
(5:24:07) **JSON Basics**

- **Express installieren:**

```
npm install express
```

- **Express Webserver erstellen:**

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => res.send('Hallo Welt!'));

app.listen(port, () => console.log(`Server läuft auf http://localhost:${port}`));
```

7. Routing & Middleware

(5:32:40) **Route Params & Query Strings**
(6:10:46) **Middleware Setup**
(6:28:31) **Multiple Middleware Functions**

- **Route-Parameter nutzen:**

```
app.get('/user/:id', (req, res) => {
    res.send(`User ID: ${req.params.id}`);
});
```

- **Query-Parameter nutzen:**

```
app.get('/search', (req, res) => {
    res.send(`Suchbegriff: ${req.query.q}`);
});
```

- **Middleware (Logger-Beispiel):**

```
app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next();
});
```

Wichtiges zum Schluss:

Node.js ist eine leistungsstarke JavaScript-Laufzeit für Serveranwendungen.
Express.js erleichtert die Entwicklung von APIs durch Routing, Middleware und strukturierte Anfragen.
Asynchrone Programmierung und das Event-Loop-Konzept sind entscheidend für Performance.
Eine saubere Code-Struktur mit Routern und Middleware ist wichtig für skalierbare Anwendungen.