Server

Ein *Server* stellt *Clients* über ein Netzwerk bestimmte Funktionen zur Verfügung. Dabei kann man zwischen *Hardware-Servern*, auch *Hosts* genannt, und *Server-Software* unterscheiden, wobei sich ein Hardware-Server vor allem dadurch auszeichnet, dass auf ihm Server-Software läuft.

Mithilfe von serverseitigem JavaScript (also JavaScript auf Hardware-Servern) kann man einen Server (also die benötigte Software) aufsetzen. Als Host kann dabei beispielsweise der eigene Computer dienen.

node.js

Um *serverseitiges JavaScript* nutzen zu können, muss auf dem Host node.js installiert sein*. Um zu überprüfen, ob node.js erfolgreich installiert wurde, kann die installierte Version überprüft werden.

```
$ node -v
vxx.xx.x
```

Mit node.js wird auch npm, der *Node Package Manager* installiert. Das ist ein Paketmanager für node.js, der u. a. bei der Installation von Paketen und bei der Organisation von node.js-Projekten hilft. Um zu überprüfen, ob npm erfolgreich installiert wurde, kann die installierte Version überprüft werden.

```
$ npm -v
x.x.x
```

Ein Projekt anlegen

Bevor programmiert wird, sollte ein passender Ort für alle Dateien gesucht werden. Beispielsweise kann innerhalb eines Projektordners 🗀 `Mein Projekt` ein Ordner für die Dokumentation 🗀 `Mein Projekt` ▸ `doc` und ein weiterer Ordner für den eigentlichen Quellcode 🗀 `Mein Projekt` ▸ `src` angelegt werden. Innerhalb von 🗀 `src` kann dann mit dem Terminal gearbeitet werden.

Mithilfe von npm wird ein node.js-Projekt angelegt. Dabei werden verschiedene Eigenschaften des Projekts abgefragt, die aber alle im Nachhinein bearbeitet werden können.

```
$ npm init
```

Aus den Angaben wird die Datei 🗀 `package.json` erzeugt. Sie enthält alle Infos, um das Projekt zu einem späteren Zeitpunkt oder auf einem anderen Gerät wieder zu installieren.

Skripte definieren

Innerhalb von 🗀 `package.json` können Skripte definiert werden. Für den Server verwenden wir zwei Skripte, um den Server bequem entweder im Entwicklermodus mit nodemon oder regulär mit node.js starten zu können. Das Skript `test` wird nicht mehr benötigt.

```
"scripts": {
  "dev": "nodemon server.js",
  "start": "node server.js"
},
```

*nodejs.org/de/download/

Pakete installieren

Für den Server werden zusätzliche Pakete benötigt. Dabei kann *nodemon* mit der Option -g global installiert werden und steht dann auch für zukünftige Projekte zur Verfügung.

```
# nodemon
# global
$ npm install -g nodemon
# oder nur für das Projekt als Entwicklungsabhängigkeit
$ npm install --save-dev nodemon

# socket.io
$ npm install socket.io

# express
$ npm install express
```

Einstiegswebsite anzeigen

Der Server soll zu Beginn eine statische Website anzeigen. Lege dazu eine statische Website nach deinen Vorstellungen unter 🗀src▸public▸index.html an.

Um den Server einzurichten, müssen innerhalb von 🗀src▸server.js die installierten Pakete eingebunden, die statische Website an den Client übergeben und der Port bestimmt werden.

```
// benötigte Pakete einbinden
const path = require('path');
const http = require('http')
const express = require('express');
const socketio = require('socket.io');

// zu verwendenden Port definieren
const PORT = process.env.PORT || 3000;

// Server einrichten
const app = express();
const server = http.createServer(app);
const io = socketio(server);

// Server starten
server.listen(PORT, () =>
    console.log(`Server listening on port ${PORT}...`));

// statische Website bereitstellen
app.use(express.static(path.join(__dirname, 'public')));
```

Client-Server-Kommunikation

Damit Client und Server miteinander kommunizieren können, müssen noch zusätzliche Einstellungen vorgenommen werden.

Innerhalb von 🗀src▸public▸index.html müssen die benötigten Skripte eingebunden werden.

```
<!-- Skript für socket.io -->
<script src="/socket.io/socket.io.js"></script>

<!-- JavaScript -->
<script src="js/client.js"></script>
```

Das benötigte clientseitige JavaScript wird dabei in 🗀 src ▸ public ▸ js ▸ client.js ausgelagert. Hier muss ein Socket initialisiert und ein Listener eingerichtet werden.

```
const socket = io();

// Nachricht empfangen
socket.on('message', msg => {
    // Was soll der Client tun, wenn er eine Nachricht bekommt?
});

// Nachricht senden
socket.emit('message', {...});
```

Der Socket muss außerdem auf der Serverseite eingerichtet werden. Dazu wird in 🗀 src ▸ server.js das Verhalten des Servers definiert. Jeder Client hat seine eigene Socket-Verbindung, die durch die Nachricht `connection` aufgebaut und durch `disconnect` beendet wird.

```
io.on('connection', socket => {
    // Was soll beim Aufbau der Socket-Verbindung passieren?

    // Nachricht empfangen
    socket.on('message', msg => {
        // Was soll der Server tun, wenn er eine Nachricht bekommt?

        // Nachricht an den Client senden
        socket.emit('message', {...});

        // Nachricht an alle anderen Clients senden
        socket.broadcast.emit('message', {...});

        // Nachricht an alle Clients senden
        socket.emit('message', {...});
    });

    socket.on('disconnect', () => {
        // Was soll beim Beenden der Socket-Verbindung passieren?
    });
});
```

Anstelle von `message` können die Nachrichten passend benannt werden. So können Server und Client auf unterschiedliche Nachrichten richtig reagieren. Die Nachrichten können Objekte mit beliebigem Inhalt enthalten, von einfachen Strings bis hin zu komplexeren Datenelementen.