1 Allgemeiner Begriff Middleware

Eigentlich ist eine Middleware eine Software, welche unterschiedliche Applikationen miteinander verbindet – also zwischen den Applikationen „vermittelt". Bei Express versteht man unter einer Middleware die Module, welche zwischen dem Empfangen der Requestnachricht und dem Senden der Responsenachricht sitzen und Zugriff auf beide Objekte (also dem Requestobjekt und dem Responseobjekt) haben.

Die Idee ist nun, dass wir Module in die Middlewareschicht einbauen können, welche standardisierte Funktionen ausführen und somit von dritten geschrieben werden können. Ein Beispiel ist das Modul „Express Rate Limit". Es verhindert zu viele Requests von einer IP Adresse in einem fest definierten Zeitraum. Um dies testen zu können, müssen wir über den Node Package Manager erstmal Express Rate Limit installieren:

```
npm install --save express-rate-limit
```

Danach importieren wir in unserer app.js Datei das Modul und erzeugen eine Instanz davon:

```
const express = require("express");
const rateLimit = require("express-rate-limit");

const limiter = rateLimit({
    windowMs: 60 * 1000, // Zeitintervall: 1 Minute
    max: 2, // Maximal 2 Anfragen von einer IP innerhalb Zeitintervall
    message: "Zu viele Anfragen pro Zeit!"
  });

const app = express();

…
```

Wir haben also unser Zeitfenster auf 60x1000 Millisekunden gesetzt und festgelegt, dass in diesem Zeitintervall maximal 2 Anfragen von einer IP Adresse kommen dürfen. Wenn diese Zahl überschritten wird, schickt Express die Nachricht „zu viele Anfragen pro Zeit!" an den Client.

Nun müssen wir dieses Modul als Middleware in unseren Verarbeitungsstrang einfügen. Hierzu dient ein „use":

```
…
const port = 8080;
app.use(limiter); //  Jeder Request wird verarbeitet
…
```

Wenn wir nun zwei Requests kurz hintereinander ausführen, arbeitet Express wie gewohnt. Beim dritten Versuch erhalten wir allerdings die Fehlermeldung. Erst wenn wir eine Minute warten, wird die nächste Anfrage wieder sauber verarbeitet.

Nun können wir die Verarbeitung auch auf gewisse Pfadbereiche beschränken. Im folgenden Code wurden einige Endpunkte erzeugt. Ergänzen sie den Code derart, dass alle Anfragen, welche mit `/api/` starten über die rate-limit Middleware überwacht werden (also `/api/eins` und `/api/zwei`, nicht aber `/test`):

```
…



app.get("/api/eins", (req, res, next) => {
```

```
    res.send("eins");
})
 app.get("/api/zwei", (req, res, next) => {
    res.send("zwei");
})
app.get("/test", (req, res, next) => {
   res.send("test");
})
…
```

Testen Sie nun Ihre Lösung.
Anmerkung: Da man mitunter von Express sowohl eine REST API als auch HTML Files zur Verfügung stellt, werden REST API Endpunkte oft mit `/api/` begonnen.

2 Eigene Middleware

Nun ist es aber auch möglich, eigene Middleware zu schreiben. Hierzu erzeugen wir ein neues File namens „`myMw.js`“ und legen es in den gleichen Pfad wie app.js. Um nun basierend auf dem Endpoint (also dem Pfad) Aktionen mit dem Request- und Response-Objekt durchführen zu können, benötigen wir eine Instanz des Express Routers. Dieser erlaubt nun die gleichen Aktionen (also GET, POST, etc.) wie der Express Server selbst. Da wir (derzeit) lediglich eine Middleware schreiben wollen, gehen wir auf alle http Befehle ein – also nutzen wir „all“:

```
const express = require('express')
const router = express.Router();

router.all("/eins", (req, res, next) => {
    res.write("I am in my middleware\n");
    next();
})

module.exports = router;
```

Zuerst nutzen wir die Express Bibliothek, indem wir über `require` „`express`“ einbinden und auf die Variable „`epxress`“ legen. Diese ermöglicht es uns nun, ein Router Objekt zu erzeugen (Zeile 2). Nun können wir mit `router.all` alle http Anfragen abarbeiten. Hier können wir nun wieder, einen Pfad angeben. Wir nutzen hier den Pfad `/eins`. Danach fügen wir in die Response Message den Text „I am in my middleware“ mit einem Zeilenumbruch ein. Damit unsere bisherigen Verarbeitungen in `app.js` noch funktionieren, fügen wir ein „`next()`“ Aufruf hinzu.

Das `module.exports = router` führt dazu, dass das Router Objekt von dem File als Modul exportiert wird. Dadurch können wir über ein „`require`“ in `app.js` dieses Modul nun wiederum in eine eigene Variable legen und dort damit weiterarbeiten:

```
const express = require("express");
const myMw = require("./myMw");
…

app.use("/api/", limiter); //  Jeder Request wird verarbeitet

app.use("/api/", myMw);
…
```

Die Variable `myMw` wird nun zum Router Objekt aus dem File `myMw.js`. Dieses wiederum kann nun als Middleware-Modul eingebunden werden. Wir binden es nun über den Pfas `/api/` ein, wodurch alle Anfragen, welche mit `/api/` beginnen über die selbst geschriebene Middleware geleitet werden. Da wir dort `/eins` als Pfad angegeben haben, werden ausschließlich die http Endpukte `/api/eins/` dort verarbeitet. Nun wird

dort aber die `write()` Funktion aufgerufen, weshalb wir in der app.js im Pfad `/api/eins` nun ebenfalls `write()` und anschließend `end()` aufrufen müssen. Geben wir also folgenden Pfad in einem Browser ein: http://localhost:8080/api/eins, dann erhalten wir folgende Meldung auf dem Browserfenster:

```
I am in my middleware
eins
```

Die URLs http://localhost:8080/api/zwei und http://localhost:8080/test würden somit nicht von unserer Middleware verarbeitet werden. Dieses Konzept bietet uns nun eine sehr schöne Möglichkeit, die einzelnen Verarbeitungsschritte einer Web Applikation zu modularisieren. Gehen wir davon aus, dass wir Kunden und Mitarbeiterdaten verarbeiten wollen. Somit können wir zwei Module erzeugen `Kunde.js` und `Mitarbeiter.js`. Diese Files legen wir in einem Unterverzeichnis „`routes`“ in unserem Serverpfad an und legen dort die Verarbeitungen unserer einzelnen Unterpfade an. Hier ein Beispiel für `Mitarbeiter.js`:

```
const express = require('express')
const router = express.Router();

router.get("/", (req, res) => {
    res.send("List of all employees");
})

router.get("/:id", (req, res) => {
    res.send("Only employee with the ID " + req.params.id);
})

module.exports = router;
```

In `app.js` können wir nun diese Routes verarbeiten lassen:

```
const express = require("express");
const myMw = require("./myMw");
const myEmployeeRoutes = require("./routes/Mitarbeiter");

…

app.use("/api/mitarbeiter/", myEmployeeRoutes);
…
```

Alle Anfragen der Route “`/api/mitarbeiter/`“ werden somit vom Modul Mitarbeiter.js abgearbeitet. Wenn wir für die Kunden nun eine Verarbeitung benötigen, so würden wir die gleichen Aktionen für „customer“ realiseiren – also ein eigenes File customer.js, ein eigener Router in app.js und as `use()` für den Pfad `/api/customer/`. Dadurch können wir sämtliche routenspezifischen Verarbeitungsschritte in eigene Module auslagern und den Code in `app.js` somit entschlacken.

3 Aufgabenstellung

Diese Aufgabenstellung dient lediglich zum Üben beim Umgang mit Routen und Middleware. Schreiben Sie hierfür eine Applikation, welche zwei Zahlen addiert, subtrahiert, multipliziert und dividiert. Nutzen Sie für die Zahlen Route-Parameter und legen Sie je ein eigenes File für die vier Grundrechenarten im Ordner „`routes`“ an. Folgende beispielhaften URLs sollen zu den angegebenen Ergebnissen führen:

URL:	**Ergebnis:**
http://localhost:8080/api/addition/3/4	`The result is 7`
http://localhost:8080/api/division/3/4	`The result is 0.75`
http://localhost:8080/api/multiplication/3/4	`The result is 12`
http://localhost:8080/api/subtraction/3/4	`The result is -1`