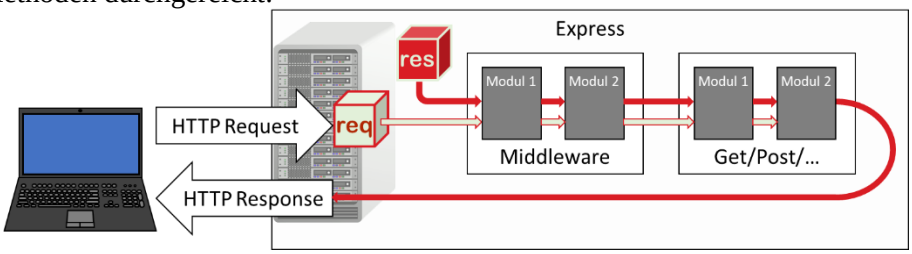
	Grundkonzept von Express		AnPr	V 1.1
	Name	Klasse	Datum	

1 Informationsfluss

Express verarbeitet http-Requests. Diese werden vom Client gesendet und innerhalb des Express Servers in ein JavaScript Objekt umgewandelt – meist „`req`“ genannt. Weiterhin wird ein JavaScript Objekt „`res`“ für den Response erzeugt. Diese beiden Objekte werden während der gesamten Verarbeitung über die Parameter der einzelnen Methoden durchgereicht:



Hierbei gibt es nun zwei verschiedene Methoden der Verarbeitung. Die „Middleware“ und die Verarbeitung der http-Methoden. Die einzelnen Verarbeitungsmethoden werden sequenziell abgearbeitet, wobei dem Express Server explizit durch einen `next()` Aufruf mitgeteilt werden muss, dass die nächste Verarbeitungsmethode nun aufgerufen werden soll.

2 Verarbeitung von http-Methoden

Jeder http-Request wird durch Express verarbeitet. Für die einzelnen http-Methoden bietet Express hierfür dezidierte Verarbeitungsmethoden an.

http-Methode:	**Verarbeitungsmethode:**
GET	app.get(path, handler);
POST	app.post(path, handler);
PUT	app.put(path, handler);
DELETE	app.delete(path, handler);
PATCH	app.patch(path, handler);

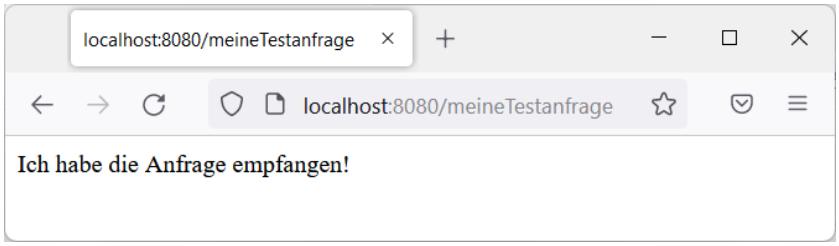
Der „`path`“ Parameter gibt an, welche URL-Ressourcenpfad verarbeitet werden soll und der `handler` ist eine Funktion, welche das `req` und `res` Objekt übergeben bekommen, um die Anfrage zu verarbeiten. Hier ein einfacher Aufbau der Verarbeitung einer GET-Anfrage:

```
const express = require('express');
const app = express();
const EXP_PORT = 8080;

// Verarbeitung aller GET-Anfragen der Route /meineTestanfrage
app.get("/meineTestanfrage", (req, res) => {
  // Senden des Textes an den Client
  res.send("Ich habe die Anfrage empfangen!");
});

app.listen(EXP_PORT, () => {
    console.log("Ich höre auf Port! " + EXP_PORT);
})
```

Rufen wir nun in einem Browser http://localhost:8080/meineTestanfrage auf, so erhalten wir:



3 Grundidee von Routing

Wir haben oben bereits `/meineTestanfrage` als Route festgelegt. Nun bietet Express Optionen an, die Routen dynamischer zu gestalten. Die wohl wichtigste ist die Wildcard „*“, welche für beliebige Zeichen steht. So würde folgender get Befehl:

```
app.get("/meineTestanfrage*", (req, res) => {
  res.send("Ich habe die Anfrage empfangen!");
});
```

alle Routen akzeptieren, welche mit `/meineTestanfrage` beginnen, bspw. `/meineTestanfragen` oder `/meineTestanfrage/ersterVersuch`. Express akzeptiert auch Reguläre Ausdrücke in den Routen:

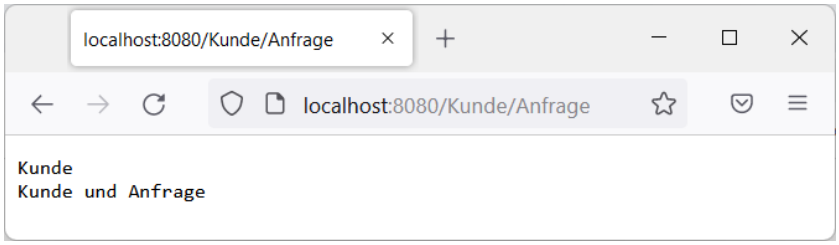
```
app.get(/\/[^\/]{3}.*/, (req, res) => {
    res.send("Anfrage mit mind. drei Zeichen bis zum 1. Slash");
  });
```

Hier würde `/abc` bzw. `/abc/` funktionieren, `/ab` bzw. `/ab/` jedoch nicht. Nun ist es möglich, verschiedene Handler auf einen Request anzuwenden. Damit jeder Handler auch Informationen ablegen kann, dürfen wir nicht mit „send“ die Funktion beenden. Wir nutzen „write“ für das Hinzufügen von Infos und „end“ für das Absenden:

```
app.get("/kunde/*", (req, res, next) => {
  res.write("Kunde\n");
  next();
});

app.get("/kunde/anfrage", (req, res) => {
  res.write("Kunde und Anfrage");
  res.end();
});
```

Wichtig ist hier, dass wir „`next()`“ über die Parameterliste erhalten und auch aufrufen. Folgendes Ergebnis:



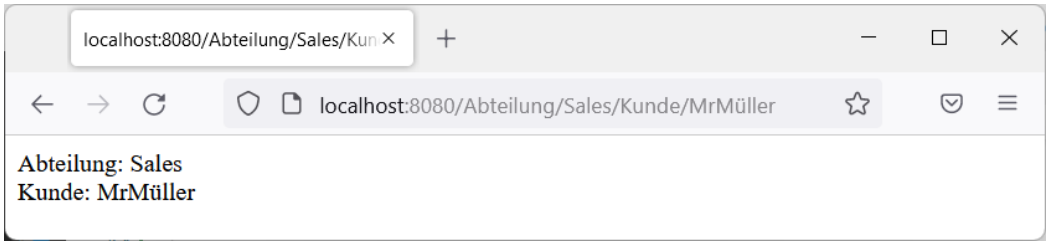
Wir müssen allerdings beachten, dass „`end`“ auch wirklich aufgerufen wird, da sonst ein Timeout entsteht. Man könnte bspw. am Ende noch alle Pfade, bspw. mit `/*` als Fehlersituation abfangen.

Grundsätzlich gilt auch, dass wir beim Design der Routen sinnvoll vorgehen. Allgemeinere Routeninformationen stehen links. Nach rechts werden die Routenelemente immer spezifischer.

4 Datenübergabe

Wir haben bei PHP gesehen, dass wir Daten über GET- oder POST Parameter übertragen können. Mit Hilfe von Express öffnen sich hier noch sehr viel mehr Möglichkeiten. Beginnen wir mit den Route Parametern. Diese werden in der URL hinterlegt und sind nicht mit dem Querystring zu verwechseln. Sie sind eine Art „dynamische Route“. Wir fügen in unserem Express Server folgende Methode mit ein:

```
app.get("/abteilung/:abteilungId/kunde/:kundeID", (req, res) => {
  const {abteilungId, kundeID} = req.params;
  res.send(`Abteilung: ${abteilungId}<br/>Kunde: ${kundeID}`);
});
```



Die Route beinhaltet nun zwei Route-Paraneter, gekennzeichnet durch den Doppelpunkt:

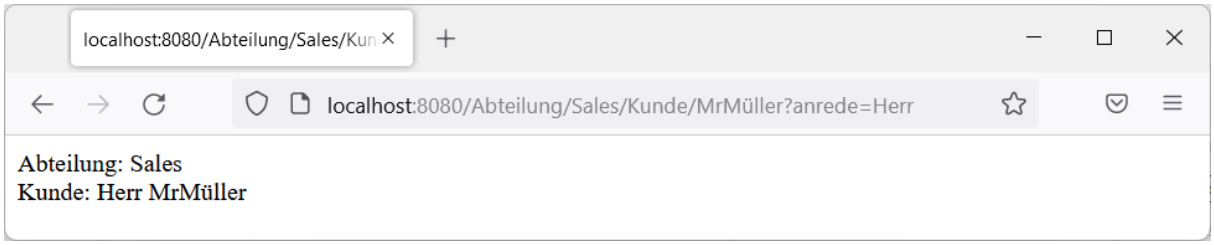
Parameter1:		Parameter2:	

Mit der Zeile `const {abteilungId, kundeID} = req.params;` übernehmen wir die beiden Werte aus der URL `req.params.abteilungId` und `req.params.kundeID` direkt in die lokalen Variablen gleichen Namens. Dieses „Konstrukt“ nennt man auch „Object Destructuring“. Die Ausgabe wurde hier der Einfachheit halber mit Hilfe von „template literals“ gemacht. Hierfür nutzt man „backtick“ Zeichen, wodurch der Ausdruck `${abteilungId}` mit dem Wert der Variablen `abteilungId` ersetzt wird.

Querystrings werden in der Variablen `req.query.*` abgegriffen:

```
app.get("/abteilung/:abteilungId/kunde/:kundeID", (req, res) => {
  const {kundeID, abteilungId} = req.params;
  const anrede = req.query.anrede ? req.query.anrede : "";
  console.log(req.query.anrede);
  res.send(`Abteilung: ${abteilungId}<br/>Kunde: ${anrede} ${kundeID}`);
});
```

Bei Querystrings sollten wir immer prüfen, ob sie auch wirklich gesetzt sind. Hierfür eignet sich sehr gut der ternäre Operator `?`. Ich prüfe hier direkt `req.query.anrede`, wobei hier auch ein object destrucutring möglich gewesen wäre – also `const {anrede} = req.query`; und anschließend eine einfach if-Abfrage auf anrede.



Weitere Parameteroptionen wären die POST-Parameter. Um dies zu testen, benötigen wir erstmal eine HTML-Seite, welche einen Post generiert:

```
<!DOCTYPE html>
<html>
<head><title>Post a msg</title></head>
<body>
<form action="/testMessage"  method="post">
  <label for="fname">Enter Message:</label><br>
  <input type="text" name="myMessage"><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Da die Daten im Body „versteckt“ sind, müssen wir diese erst zugreifbar machen. Hierfür muss der Body vorverarbeitet werden:

```
const express = require('express');
const app = express();
// parsen des Body-Inhaltes und vorbereiten der Daten in req.body
app.use(express.urlencoded({ extended: true }));
//…
```

Ab hier stehen die Form-Daten zur Verfügung und können in `req.body` abgegriffen werden:

```
//…
app.post('/testMessage', function (req, res, next) {
  res.send("I received: " + req.body.myMessage);
});
```

5 Aufgabenstellung

Erstellen Sie eine serverseitige Applikation, welche zwei Zahlen addiert. Diese Zahlen sollen entweder über Route-Parameter, dem Querystring oder POST-Parameter übergeben werden.