

# REST API mit Node.js Express

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# Representational State Transfer

RESTful API = Abkürzung für  
HTTP Transfer mit {JSON Daten}

{ REST

Beliebte Alternative für traditionelle  
Websites, besonders für Single Page Apps

Wie die asynchrone Geolocation API von  
Google / Leaflet Geocoding zum Beispiel

Web-Server, der je nach URL kein HTML,  
sondern nur {JSON Daten} schickt

# REST Constraints

- Client-Server Architektur: RESTful API sollte sich nicht um die UI kümmern
- Stateless: Kein Client-Context (z.B. Session!) wird auf dem Server gespeichert
- Cacheability: Responses müssen selbst definieren ob sie gecacht werden dürfen oder nicht
- Layered System: Zwischenserver können benutzt werden, ohne dass der Client davon weiß
- Einheitliches Interface: Resources werden im Request identifiziert, Daten sind vom DB-Schema entkoppelt

{ REST }

# REST API: Was bauen wir?

Unsere API

/products

GET

POST

/orders

GET

POST

/products/{id}

GET

PATCH

DELETE

/orders/{id}

GET

DELETE

# REST API: Express installieren



- Die Kursdateien für die RESTful API mit NodeJS Tutorial:
- In den Branches finden Sie die einzelnen "Lessons" 01-13
- <https://github.com/htw-web/no-de-restful-api-tutorial>

● `npm install express --save`

● Zwei Dateien kreieren  
`server.js` & `app.js`

Mehr Infos unter

<https://expressjs.com/de/>

# Hello World 1: server.js (Express)

```
const http = require('http');
```

```
const app = require('./app');
```

```
const port = process.env.port || 3003;
```

```
const server = http.createServer(app);
```

```
server.listen(port, function () {
```

```
  console.log('Example app listening on port',
```

```
port);
```

```
});
```

Mehr Infos unter

[https://www.w3schools.com/nodejs/met\\_http\\_createserver.asp](https://www.w3schools.com/nodejs/met_http_createserver.asp)



# Hello World 2: app.js (Express)

```
const express = require('express');
```

```
const app = express();
```

```
app.use((req, res, next) => {
```

```
  res.status(200).json({
```

```
    message: 'It works!'
```

```
  });
```

```
});
```

```
module.exports = app;
```



Mehr Infos unter

<https://expressjs.com/de/starter/hello-world.html>

# REST API: Express Server starten

Express Node-Server  
starten mit

```
node server.js
```

Stoppen mit

CTRL + C

Testen mit

<http://localhost:3003/>

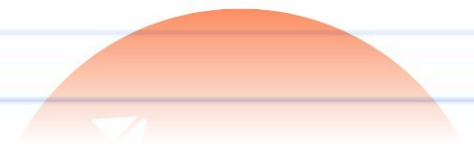




# Exkurs:

## Postman: Praktisches Test-Tool

- Für Windows, Mac & Linux
- Kostenlos verfügbar
- <https://www.postman.com/downloads/>
- Keine Registrierung notwendig!
- Kann Requests speichern & zeigt alles im Response an von A-Z



# VS Code & Postman Demo




# Mehr Routen zur API hinzufügen

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# Routen anlegen unter api/routes/..

## 1. 2 Neue Ordner api + routen innerhalb

## 2. Neue Datei products.js innerhalb



```
const express = require('express');  
const router = express.Router();  
router.get('/', (req, res, next) => {  
  res.status(200).json({  
    message: 'Handling GET requests to /products'  
  });  
});  
module.exports = router;
```

## 3. app.js anpassen

```
const productRoutes = require('./api/routes/products');  
app.use('/products', productRoutes);
```

# REST API: Was bauen wir?

Unsere API

/products

GET

POST

/orders

GET

POST

/products/{id}

GET

PATCH

DELETE

/orders/{id}

GET

DELETE

# GET POST PATCH DELETE Routen für api/router/products.js

```
router.post('/', (req, res, next) => {  
  res.status(201).json({  
    message: 'Handling POST requests to /products'  
  });  
});  
  
router.get('/:productId', (req, res, next) => {  
  const id = req.params.productId;  
  if (id === 'special') {  
    res.status(200).json({  
      message: 'You discovered the special ID'  
    });  
  } else {  
    res.status(200).json({  
      message: 'You passed an ID'  
    });  
  }  
});  
  
router.patch('/:productId', (req, res, next) => {  
  res.status(200).json({  
    message: 'Updated product!'  
  });  
});  
  
router.delete('/:productId', (req, res, next) => {  
  res.status(200).json({  
    message: 'Deleted product!'  
  });  
});
```



# VS Code & Postman Demo



# Kopiere products zu orders.js & anpassen nach Design

```
router.get('/', (req, res, next) => {  
  res.status(200).json({  
    message: 'Orders were fetched'  
  });  
});
```

```
router.post('/', (req, res, next) => {  
  res.status(201).json({  
    message: 'Order was created'  
  });  
});
```

```
router.get('/:orderId', (req, res, next) => {  
  res.status(200).json({  
    message: 'Order details',  
    orderId: req.params.orderId  
  });  
});
```

```
router.delete('/:orderId', (req, res, next) => {  
  res.status(200).json({  
    message: 'Order deleted'  
  });  
});
```





# VS Code & Postman Demo



# Verbesserungen & Errorhandling

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# Exkurs: Nodemon(itor) installieren



- `npm install --save-dev nodemon`
- Beobachtet unsere Dateien und startet den Server neu, sobald was geändert wird => "Watcher"
- `nodemon server.js`
  - => **Befehl nicht gefunden!**
- Fix im Startup Script anpassen, so dass es automatisch gestartet wird, wenn wir "npm start" schreiben:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "nodemon server.js"  
},
```

# Exkurs: Morgan Logging

Damit wir incoming Requests im Server direkt sehen:

```
npm install --save morgan
```

Im app.js importieren:

```
const morgan = require('morgan');
```

```
app.use(morgan('dev'));
```

Dann den NPM Server neu starten, und

jetzt können wir alle Requests in der Console sehen :)



HTTP  
LOGS



# Errorhandling (1)

Normalerweise bekommen wir automatisch eine Standard-HTML-Fehler-Seite zurück, wenn wir eine falsche Adresse aufrufen

Stattdessen wollen wir aber JSON zurückschicken

D.h. wir müssen in der app.js den Fehler handeln

```
// Server Errorhandling  
app.use((req, res, next) => {  
  const error = new Error('Not found');  
  error.status = 404;  
  next(error);  
});
```



# Errorhandling (2)

Falls wir noch mehr Fehler einbauen, können wir gleich eine zweite Abstraktionsebene für generelle Error (z.B. auch aus der Datenbank) einbauen, wie hier:



```
// General error handling
app.use((error, req, res, next) => {
  res.status(err.status || 500);
  res.json({
    error: {
      message: error.message
    }
  });
});
```

Testen nicht vergessen!

# Body parsen & CORS fixen

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# Exkurs: Body-Parser installieren

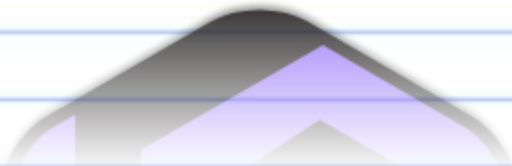
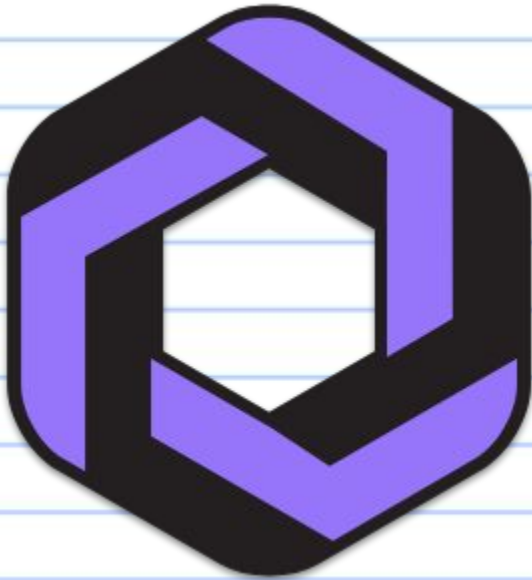
- `npm install --save body-parser`

- Damit können wir Request-Bodys auslesen/parsen

z.B. unterstützt es URL-Encoded Bodies und POST Requests, die JSON Payload mitbringen

In `app.js` einbinden:

```
const bodyParser = require("body-parser"); // importieren
app.use(bodyParser.urlencoded({ extended: false })); // true =
RICH DATA
app.use(bodyParser.json()); // packt JSON aus
```





# Body-Parser benutzen: products.js

```
router.post('/', (req, res, next) => {  
  const product = {  
    name: req.body.name,  
    price: req.body.price  
  };  
  res.status(201).json({  
    message: 'Handling POST requests to /products',  
    createdProduct: product  
  });  
});
```



# Body-Parser benutzen: orders.js



```
router.post('/', (req, res, next) =>
{
  const order = {
    productId: req.body.productId,
    quantity: req.body.quantity
  };
  res.status(201).json({
    message: 'Order was created',
    order: order
  });
});
```

# POST JSON Bodies in Postman

Um die Hinzufügen-Requests zu simulieren, können wir im Postman auch einen JSON Body hinzufügen. Achten Sie auf korrekte Schreibweise, alles muss als String übergeben werden:

```
{  
  "productId":  
  "alksdaölskdöalskd",  
  "quantity": "10"  
}
```



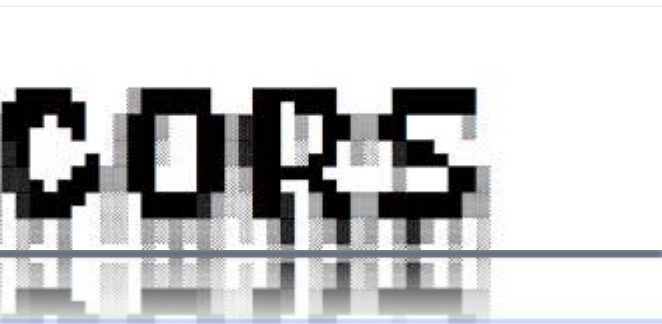
# VS Code & Postman Demo



# CORS: Cross Origin Resource Sharing

Eigentlich ein gutes Sicherheitskonzept, dass Websites nur von ihrem eigenen Server (und Port!) Daten abrufen können.

=> Problematisch bei RESTful APIs, die sind ja extra für externe Anfragen gedacht



=> CORS muss im Header disabled werden, damit unsere API angefragt werden kann (falls Fehlermeldung wie z.B. **"no Access-Control-Allow-Origin Header present"** vorhanden ist)

Mehr Infos unter

<https://developer.mozilla.org/de/docs/Web/HTTP/CORS>

# In app.js CORS Header hinzufügen

```
app.use((req, res, next) => {  
  res.header("Access-Control-Allow-Origin",  
    "*");  
  res.header(  
    "Access-Control-Allow-Headers",  
    "Origin, X-Requested-With, Content-Type,  
Accept, Authorization"  
  );  
  if (req.method === 'OPTIONS') {  
  
    res.header('Access-Control-Allow-Methods',  
      'PUT, POST, PATCH, DELETE, GET');  
    return res.status(200).json({});  
  }  
  next();  
});
```



CORS

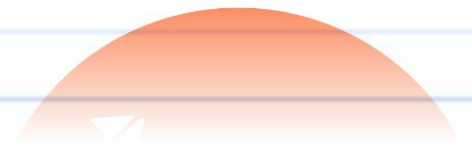
# Postman hat kein Problem mit CORS!

Postman ist extra zum Testen gemacht.

CORS ist ein Konzept, das nur im Browser funktioniert.

Postman interessiert sich nicht dafür und daher gibt es hier damit auch nie Probleme!

=> Aber für unsere SPA!!



# DB : MongoDB & Mongoose

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*



# MongoDB: NoSQL Datenbank

Supereinfaches Setup, keine Kreditkarte nötig & kostenlos



<https://www.mongodb.com/cloud/atlas/register>

Datenbank in der AWS Cloud z.B. auch mit Google Login möglich

=> Geht natürlich auch lokal, wie Sie möchten!

Mehr Infos unter <https://university.mongodb.com/>

# MongoDB: NoSQL Datenbank

## Erste Schritte mit MongoDB:

- 1) Beide Nutzer für die Abgabe anlegen (admin und no-admin)
- 2) IP-Adresse freischalten, entweder generell für alle zugänglich machen oder ihre eigene eintragen
- 3) Beispiel-Daten reinladen (wenn Sie schon welche haben)
- 4) Applikation verbinden => Link von Website kopieren, User + Passwort ändern nicht vergessen!



# Mongoose: Verknüpfung zur MongoDB

```
npm install --save mongoose
```

```
const mongoose = require('mongoose');
```

```
mongoose.connect(
```

```
"mongodb://node-shop:" +
```

```
process.env.MONGO_ATLAS_PW +
```

```
"@node-rest-shop-shard-00-00-wovcj.mongodb.net:27017,node-rest-shop-shard-00-01-wovcj.mongodb.net:27017,node-rest-shop-shard-00-02-wovcj.mongodb.net:27017/test?ssl=true&replicaSet=node-rest-shop-shard-0&authSource=admin",
```

```
{
```

```
  useMongoClient: true
```

```
}
```

```
); Mehr Infos unter
```

```
https://mongoosejs.com/docs/index.html
```



# Mongoose: DB Schema erstellen

Neuer Ordner und File in `api/models/product.js`:

```
const mongoose = require('mongoose');
```

```
const productSchema = mongoose.Schema({  
  id: mongoose.Schema.Types.ObjectId,  
  name: String,  
  price: Number  
});
```

```
module.exports = mongoose.model('Product',  
productSchema); // wie es intern als Konstruktor  
genannt/benutzt werden soll => new Product()
```



# Speichern des Posts in der DB



```
router.post("/", (req, res, next) =>
{
  const product = new Product({
    id: new
mongoose.Types.ObjectId(),
    name: req.body.name,
    price: req.body.price
  });
  product
    .save()
    .then(result => {
      console.log(result);
      res.status(201).json({
        message: "Handling POST
requests to /products",
```

# VS Code & Postman Demo



# Credits & Mehr Lesestoff

Die Kursdateien für die RESTful API mit NodeJS Tutorial:

In den Branches finden Sie die einzelnen "Lessons" 01-13

<https://github.com/htw-web/node-restful-api-tutorial>

Creating a REST API with Node.js - Maximilian Schwarzmüller

[https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-B63KBnyqU6opNPFk\\_q](https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-B63KBnyqU6opNPFk_q)

Express JS Library Documentation

<https://expressjs.com/de/>

Vielen Dank für Ihre Aufmerksamkeit!

