

Mongoose & MongoDB

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

Mongoose: Verbindung zur MongoDB

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

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

```
mongoose.connect(
```

```
"mongodb+srv://admin:admin@cluster0  
.rlyud.mongodb.net/adviz?retryWrite  
s=true&w=majority", {
```

```
  useNewUrlParser: true,
```

```
  useUnifiedTopology: true
```

```
  }
```

```
);
```



Mehr Infos unter

<https://mongoosejs.com/docs/index.html>

API Rückblick: products.js POST

```
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  
  });  
});
```



Mongoose: DB Schemas => DB Models

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



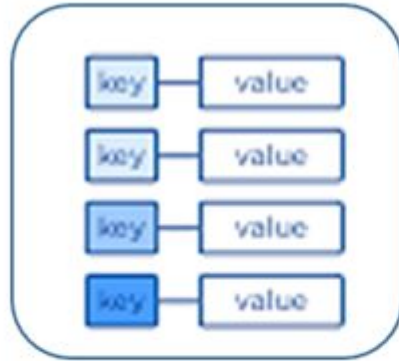
```
const Product =  
require("../models/product");  
router.post("/", (req, res, next) => {  
  const product = new Product({  
    _id: new mongoose.Types.ObjectId(),  
    name: req.body.name,  
    price: req.body.price  
  });  
  product  
    .save()  
    .then(result => {  
      console.log(result);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

Exkurs: NoSQL vs. SQL Datenbank

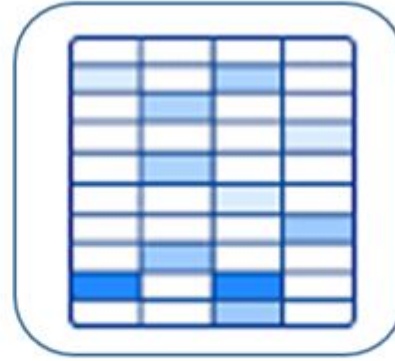
Im Gegensatz zur Speicherung in relationalen Datenbanken (RDMS) werden in NoSQL Datenbanken meist eins der folgenden Systeme zur Verfügung gestellt (MongoDB: Key-Value Store):



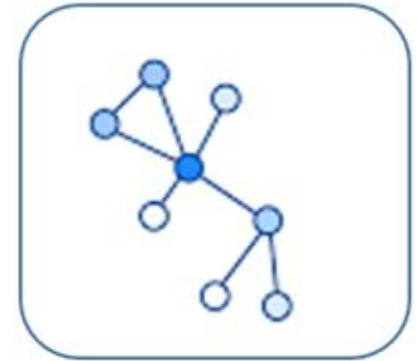
Document
Store



Key-Value
Store



Wide-Column
Store



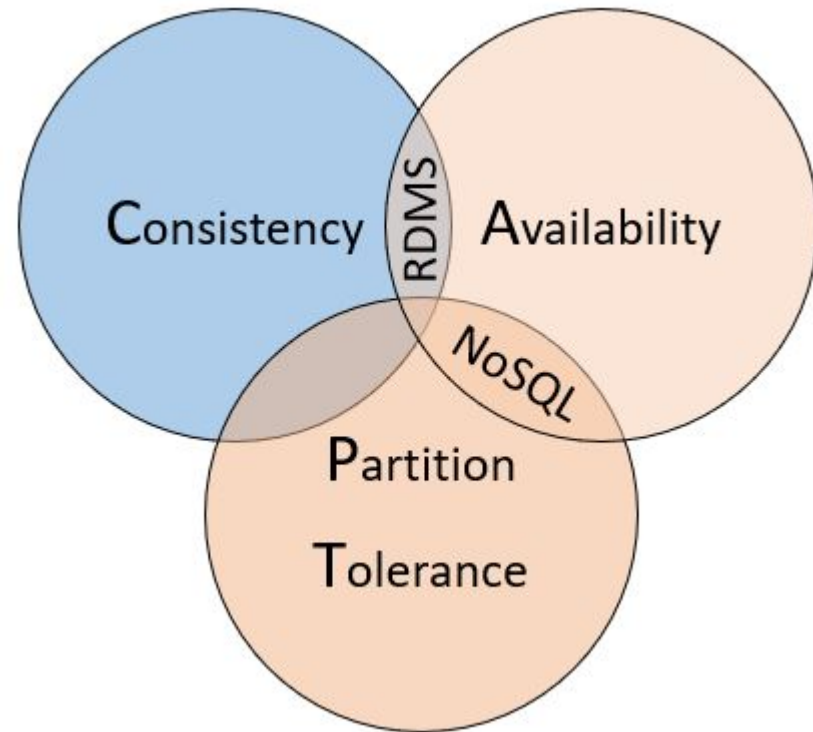
Graph
Store

Mehr Infos unter

<https://docs.microsoft.com/de-de/dotnet/architecture/cloud-native/relational-vs-nosql-data>

Exkurs: Das CAP Theorem (Brewers)

Dazu müssen Sie wissen, Datenbanken können immer nur zwei Ziele dieser drei erreichen, z.B. das RDMS erreicht Konsistenz und Verfügbarkeit, aber wenn Ein Teil fehlt, geht gar Nichts mehr. Anders bei NoSQL, hier sind dafür die Daten nicht immer 100%ig konsistent.

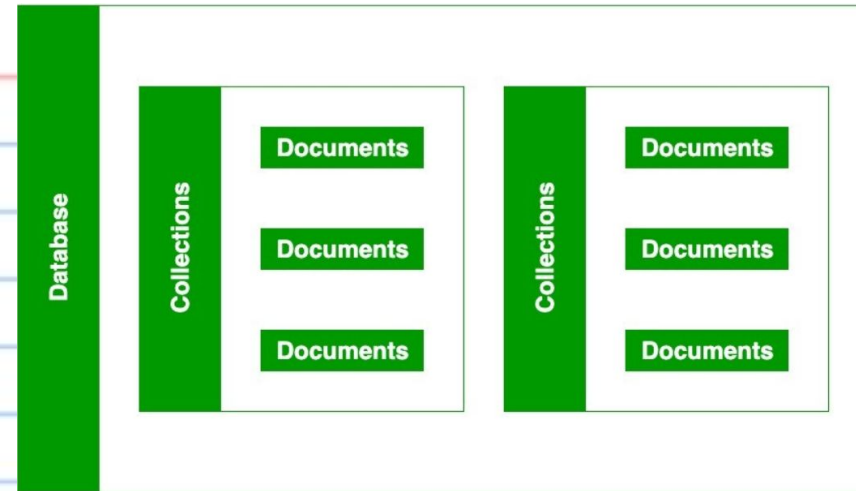


Mehr Infos unter

<https://de.wikipedia.org/wiki/CAP-Theorem>

Exkurs: NoSQL Collections vs. DBs?

Sie brauchen eine "adviz" Datenbank, in dieser können Sie dann mehrere Collections anlegen:



```
{
  na
  ag
  st
  gr
}

{
  na
  ag
  st
  gr
}

{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

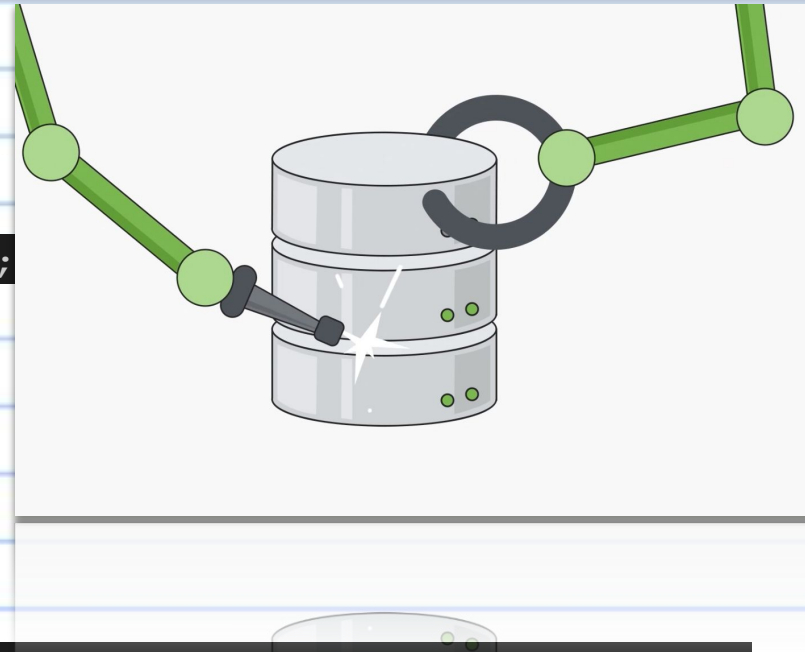
Eine Collection beinhaltet z.B. mehrere JSON-Files vergleichbaren Inhalten (keys), sogenannte "BSON"-Files

Mehr Infos Collection

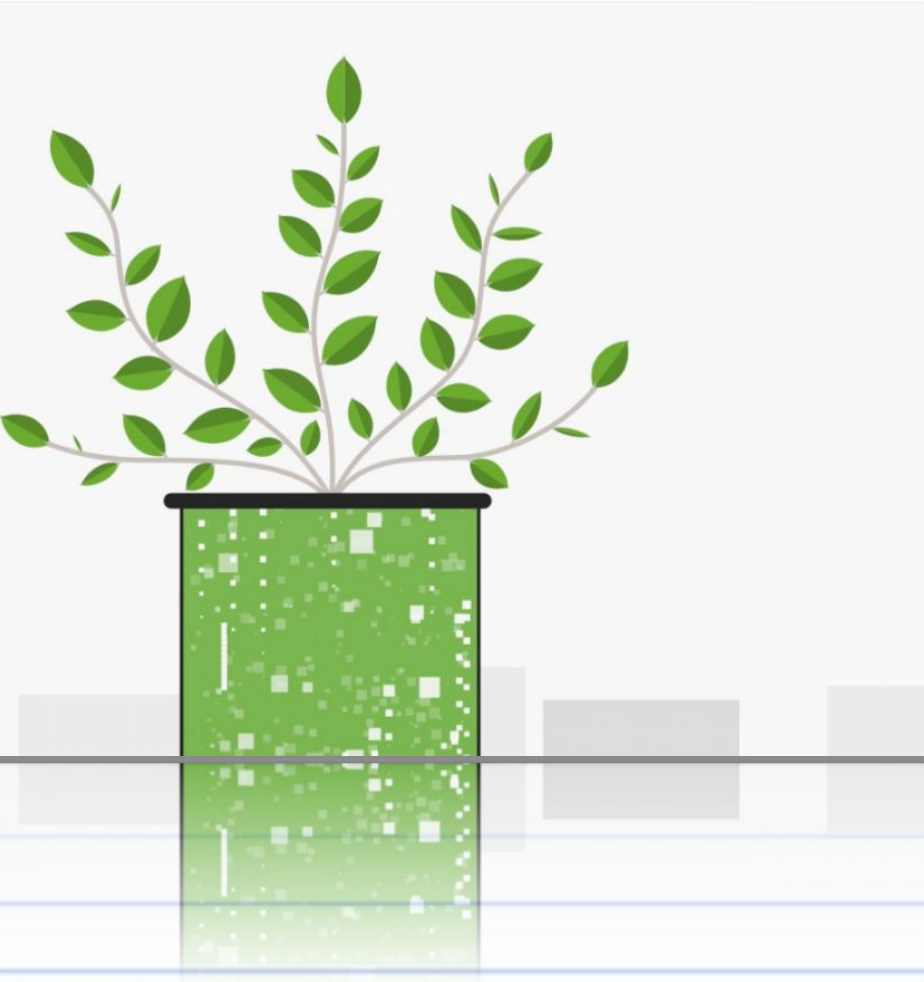
<https://docs.mongodb.com/manual/core/databases-and-collections/>

Routes anpassen an die neue Id

```
router.get("/:productId", (req, res, next) => {  
  const id = req.params.productId;  
  Product.findById(id)  
    .exec()  
    .then(doc => {  
      console.log("From database", doc);  
      if (doc) {  
        res.status(200).json(doc);  
      } else {  
        res  
          .status(404)  
          .json({ message: "No valid entry found for provided ID" });  
      }  
    })  
    .catch(err => {  
      console.log(err);  
      res.status(500).json({ error: err });  
    });  
});
```



Alle Produkte aus der DB holen



```
router.get("/", (req, res, next) => {  
  Product.find()  
    .exec()  
    .then(docs => {  
      console.log(docs);  
      // if (docs.length >= 0)  
      res.status(200).json(docs);  
      // } else {  
      //   res.status(404).json({  
      //     message: 'No entries  
found'  
      //   });  
      // }  
    })  
    .catch(err => {  
      console.log(err);  
      res.status(500).json({  
        error: err  
      });  
    });  
});
```

Produkt löschen aus der Datenbank

```
router.delete("/:productId", (req, res, next) => {  
  const id = req.params.productId;  
  Product.remove({ id: id })  
    .exec()  
    .then(result => {  
      res.status(200).json(result);  
    })  
    .catch(err => {  
      console.log(err);  
      res.status(500).json({  
        error: err  
      });  
    });  
});
```

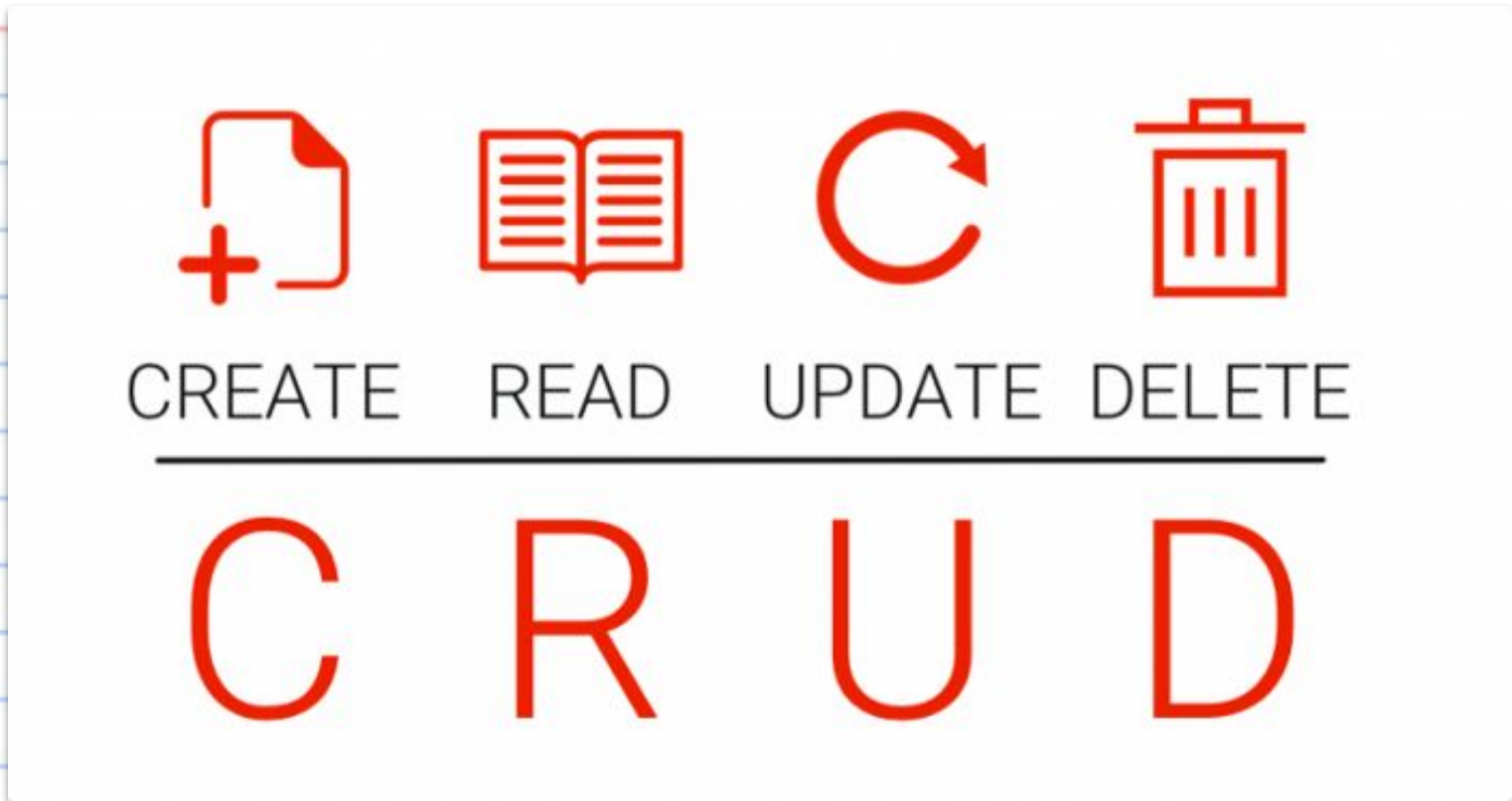


Daten ändern mit Patch (Update)

```
router.patch("/:productId", (req, res, next) =>
{
  const id = req.params.productId;
  const updateOps = {};
  for (const ops of req.body) {
    updateOps[ops.propName] = ops.value;
  }
  Product.update({ _id: id }, { $set: updateOps
}))
  .exec()
  .then(result => {
    console.log(result);
    res.status(200).json(result);
  })
  .catch(err => {
    console.log(err);
    res.status(500).json({
      error: err
```



=> RESTFUL Api mit Persistenz!



Alle Punkte können wir jetzt nicht nur mit der API sondern auch in der Datenbank dauerhaft speichern

✓ Create ✓ Read ✓ Update ✓ Delete

Validierung & Verbesserungen

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

Wie können wir Inputs validieren?

Angenommen, wir schicken statt dem Preis einen String "hallo" per POST.

=> Mongoose schickt einen Fehler, da wir im Schema eine Zahl (number) erwarten!

```
"message": "Product validation failed: price: Cast to Number failed for value \"hello\" at path \"price\""
```

Und was passiert, wenn ich gar keinen Preis schicke, sondern nur einen Namen?

=> Neuer Eintrag in der Datenbank... ups!



Schema: wichtige Felder "requieren"



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

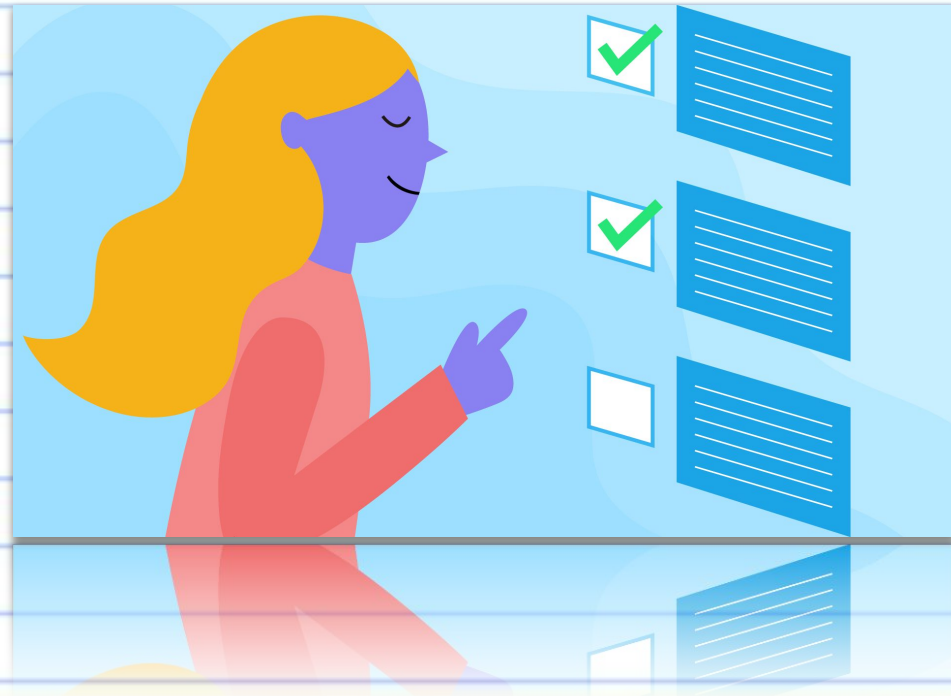
```
=> "message": "Path `price` is required.",
```


GET Response verbessern, z.B. URLs

REST Apis sollten selbsterklärend sein, nicht komplizierte kryptische Antworten enthalten...

Daher sollten die "rohen" DB-Daten mit Mongoose aufbereitet werden:

```
router.get("/", (req, res, next) => {  
  Product.find()  
    .select("name price id")  
    .exec()  
    .then(docs => {  
      const response = {  
        count: docs.length,  
        products: docs.map(doc => {  
          return {  
            name: doc.name,  
            price: doc.price,  
            id: doc._id,  
            request: {  
              type: "GET",  
              url: "http://localhost:3000/products/" + doc._id
```



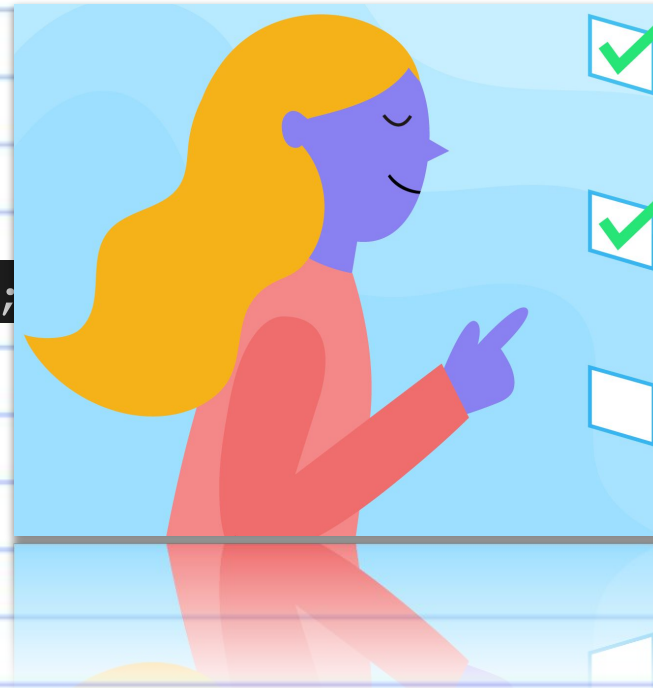
POST Request verbessern, z.B. URL



```
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: "Created product successfully",  
        createdProduct: {  
          name: result.name,  
          price: result.price,  
          id: result.id,  
          request: {  
            type: 'GET',  
            url: "http://localhost:3000/products/" + result.id  
          }  
        }  
      })  
    })  
});
```

GET :productId Response verbessern

```
router.get("/:productId", (req, res, next) => {  
  const id = req.params.productId;  
  Product.findById(id)  
    .select('name price _id')  
    .exec()  
    .then(doc => {  
      console.log("From database", doc);  
      if (doc) {  
        res.status(200).json({  
          product: doc,  
          request: {  
            type: 'GET',  
            url: 'http://localhost:3000/products'  
          }  
        })  
      }  
    })  
  next();  
})
```



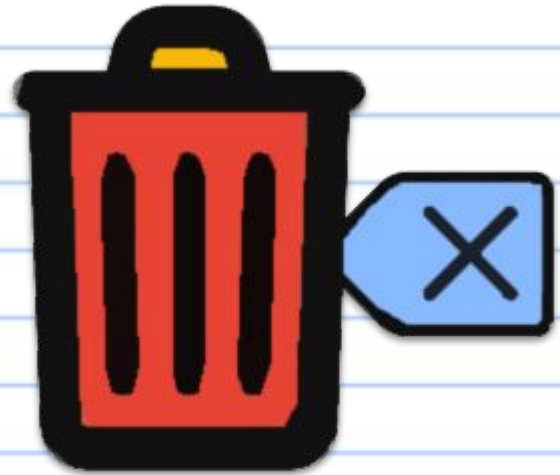
Patch Request verbessern, z.B. URL

```
router.patch("/:productId", (req, res, next) => {  
  const id = req.params.productId;  
  const updateOps = {};  
  for (const ops of req.body) {  
    updateOps[ops.propName] = ops.value;  
  }  
  Product.update({ id: id }, { $set: updateOps })  
    .exec()  
    .then(result => {  
      res.status(200).json({  
        message: 'Product updated',  
        request: {  
          type: 'GET',  
          url: 'http://localhost:3000/products/' + id
```



DELETE :productId Response improve

```
router.delete("/:productId", (req, res, next) => {  
  const id = req.params.productId;  
  Product.remove({ id: id })  
    .exec()  
    .then(result => {  
      res.status(200).json({  
        message: 'Product deleted',  
        request: {  
          type: 'POST',  
          url: 'http://localhost:3000/products',  
          body: { name: 'String', price: 'Number'  
        }  
      })  
    })  
})
```



Managing Orders mit Mongoose

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

Exkurs: Relationen mit NoSQL? Ref

- Wir können auch Relationen herstellen mit NoSQL



- Wenn wir allerdings sehr viele davon haben, macht es vielleicht doch eher Sinn, ein RDMS zu benutzen!
- Solange es aber nur ein paar wenige Relationen wie z.B. “Wem gehört der Kontakt” gibt, sollte auch die MongoDB bzw. jede andere NoSQL-Datenbank genügen

orders.js Schema & Model von project.js kopieren und anpassen

Neues File in api/models/order.js:

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

```
const orderSchema = mongoose.Schema({
```

```
  _id: mongoose.Schema.Types.ObjectId,
```

```
  product: { type: mongoose.Schema.Types.ObjectId,
```

```
  ref: 'Product', required: true },
```

```
  quantity: { type: Number, default: 1 }
```

```
});
```

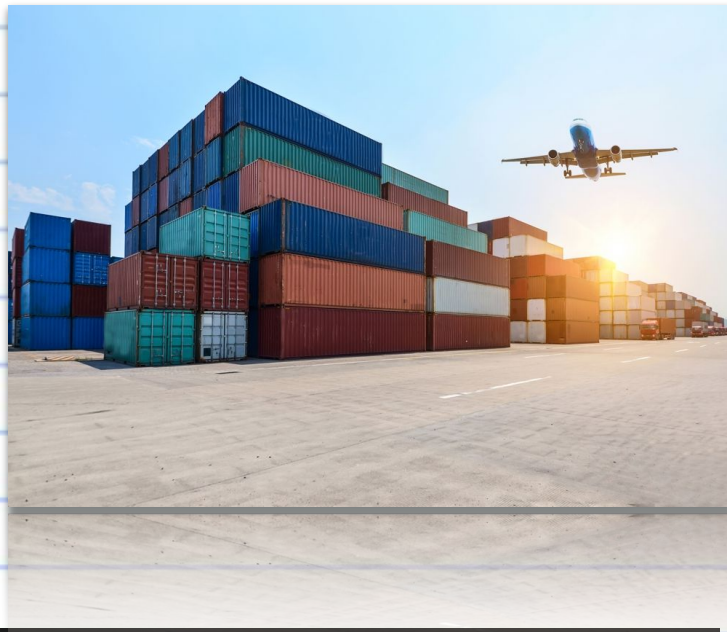
```
module.exports = mongoose.model('Order',
```

```
orderSchema);
```



Imports für orders.js ergänzen

Erstmal die Imports, nicht vergessen!



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

```
const Order = require("../models/order");
```

```
const Product = require("../models/product");
```

Routen für orders.js anpassen POST

```
router.post("/", (req, res, next) => {  
  Product.findById(req.body.productId)  
    .then(product => {  
      if (!product) {  
        return res.status(404).json({  
          message: "Product not found"  
        });  
      }  
      const order = new Order({  
        id: mongoose.Types.ObjectId(),  
        quantity: req.body.quantity,  
        product: req.body.productId  
      });  
      return order.save();  
    })  
    .then(result => {  
      console.log(result);  
      res.status(201).json({  
        message: "Order stored",  
        createdOrder: {  
          id: result.id,  
          product: result.product,  
          quantity: result.quantity  
        },  
        request: {  
          type: "GET",  
          url: "http://localhost:3000/orders/" + result.id
```



GET (all) orders.js anpassen

```
// Handle incoming GET requests to /orders
router.get("/", (req, res, next) => {
  Order.find()
    .select("product quantity _id")
    .exec()
    .then(docs => {
      res.status(200).json({
        count: docs.length,
        orders: docs.map(doc => {
          return {
            id: doc._id,
            product: doc.product,
            quantity: doc.quantity,
            request: {
              type: "GET",
              url: "http://localhost:3000/orders/" + doc._id
            }
          }
        })
      })
    })
})
```



GET 1 (:orderId) order.js anpassen



```
router.get("/:orderId", (req, res, next) => {  
  Order.findById(req.params.orderId)  
    .exec()  
    .then(order => {  
      if (!order) {  
        return res.status(404).json({  
          message: "Order not found"  
        });  
      }  
      res.status(200).json({  
        order: order,  
        request: {  
          type: "GET",  
          url: "http://localhost:3000/orders"        }  
      });  
    })  
});
```

Nur noch Delete OrderId fixen :)

```
router.delete("/:orderId", (req, res, next) => {  
  Order.remove({ _id: req.params.orderId })  
    .exec()  
    .then(result => {  
      res.status(200).json({  
        message: "Order deleted",  
        request: {  
          type: "POST",  
          url: "http://localhost:3000/orders",  
          body: { productId: "ID", quantity:  
"Number" }  
        }  
      })  
    })  
    .catch(err => {  
      res.status(500).json({  
        message: "Error deleting order",  
        error: err  
      })  
    })  
  next()  
})
```

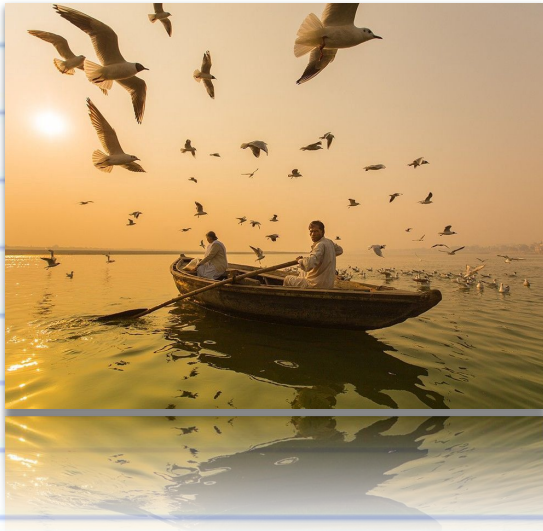


Populäres Populate()

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

Relation "benutzen" z.B. orders.js

- Nach dem `select()` und vor dem `exec()` haben wir die Möglichkeit, die Relation einzubinden:



```
.populate('product') // Alle Daten  
.populate('product', 'name')  
// Nur den Namen z.B.
```

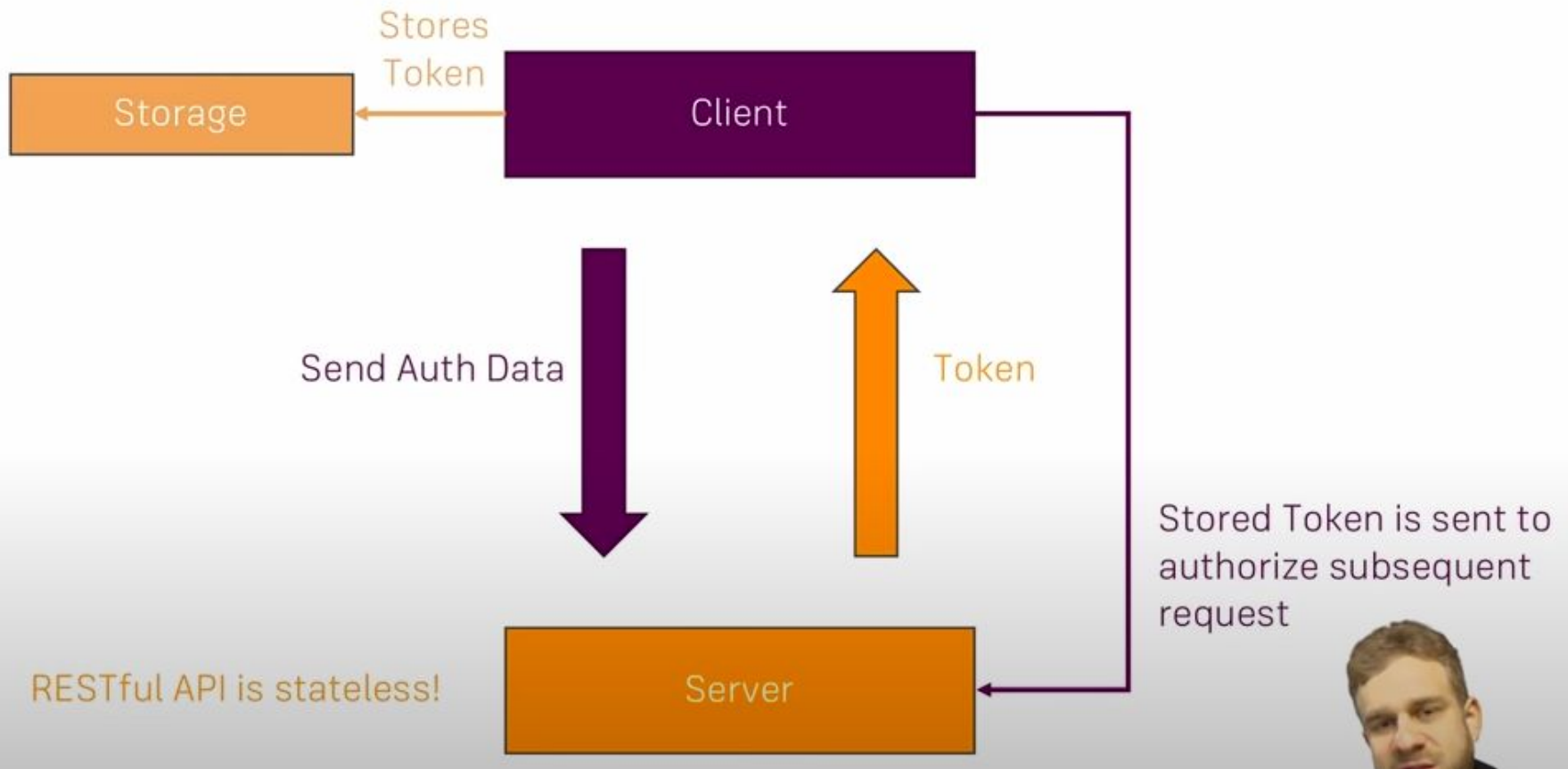
- Und damit können wir sie benutzen:

```
product: doc.product,
```

User Signup hinzufügen opt.

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

Wie Authentifizierung funktioniert



JSON Web Token (JWT)

Mehr Infos unter <https://jwt.io/>

Einfaches JSON Data Object

=> Typischerweise unverschlüsselt [OBJ]

+

Signatur

=> Kann vom Server verifiziert werden

=

JSON Web Token (JWT)



JWT

User.js Model/Schema wie Order.js

Neues File in api/models/user.js:

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

```
const userSchema = mongoose.Schema({  
  id: mongoose.Schema.Types.ObjectId,
```

```
  email: {
```

```
    type: String,
```

```
    required: true,
```

```
    unique: true,
```

```
    match:
```

```
    /[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/
```

```
  },
```

```
  password: { type: String, required: true }
```

```
});
```

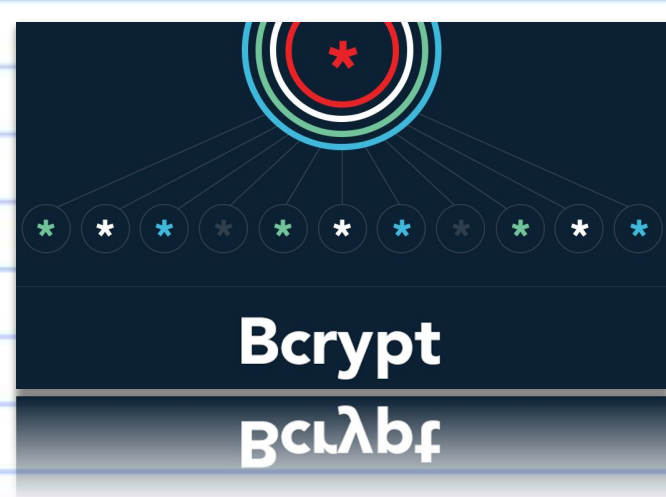
```
module.exports = mongoose.model('User', userSchema);
```



Exkurs: bcrypt Hash Password (opt)

<https://github.com/kelektiv/node.bcrypt.js>

One Way Hashing mit Salt



Passwörter sollten nie im Klartext in der Datenbank gespeichert werden, daher optional bitte dieses Node Paket installieren!

npm install bcrypt --save

Mindestens

```
"bcrypt": "^5.0.0",
```

Bcrypt benutzen: user.js (optional)

```
const express = require("express");
const router = express.Router();
const mongoose = require("mongoose");
const bcrypt = require("bcrypt");
const User = require("../models/user");

router.post("/signup", (req, res, next) => {
  User.find({ email: req.body.email })
    .exec()
    .then(user => {
      if (user.length >= 1) {
        return res.status(409).json({
          message: "Mail exists"
        });
      } else {
        bcrypt.hash(req.body.password, 10, (err, hash) => {
          if (err) {
            return res.status(500).json({
              error: err
            });
          } else {
            const user = new User({
              id: new mongoose.Types.ObjectId(),
              email: req.body.email,
              password: hash
            });
```

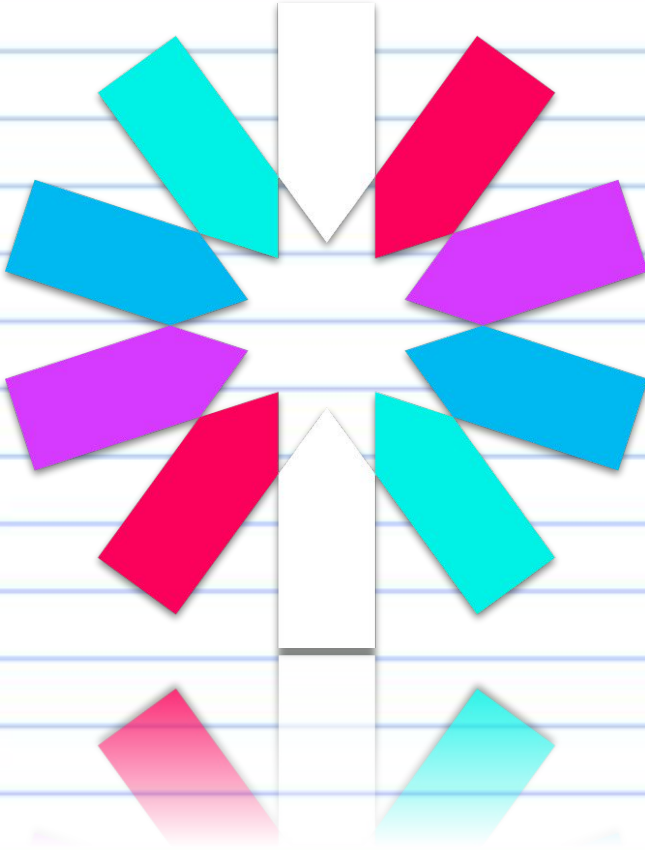
```
            user.save()
              .then(result => {
                console.log(result);
                res.status(201).json({
                  message: "User created"
                });
              })
              .catch(err => {
                console.log(err);
                res.status(500).json({
                  error: err
                });
              });
          });
        });
      });
    });
  router.delete("/:userId", (req, res, next) => {
    User.remove({ id: req.params.userId })
      .exec()
      .then(result => {
        res.status(200).json({
          message: "User deleted"
        });
      })
      .catch(err => {
        console.log(err);
        res.status(500).json({
          error: err
        });
      });
    });
  });
});
```

User Login & Token liefern

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

Exkurs: JSON Web Token

```
npm install jsonwebtoken --save
```



```
const token = jwt.sign({  
  email: user[0].email,  
  userId: user[0].id  
},  
  process.env.JWT_KEY,  
  {expiresIn: "1h"}  
);
```

Mehr Infos unter

<https://www.npmjs.com/package/jsonwebtoken>

JWT + Bcrypt: /login in user.js

```
router.post("/login", (req, res, next) => {
```

```
  User.find({ email: req.body.email })
```

```
    .exec()
```

```
    .then(user => {
```

```
      if (user.length < 1) {
```

```
        return res.status(401).json({
```

```
          message: "Auth failed"
```

```
        });
```

```
      }
```

```
      bcrypt.compare(req.body.password,
```

```
      user[0].password, (err, result) => {
```

```
        if (err) {
```

```
          return res.status(401).json({
```

```
            message: "Auth failed"
```

```
          });
```

```
        }
```

```
        if (result) {
```

TOKEN (siehe letzte Folie)

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

```
            message: "Auth successful",
```

```
            token: token
```

```
          });
```

```
        res.status(401).json({
```

```
          message: "Auth failed"
```

```
        });
```

```
      });
```

```
    }.catch(err => {
```

```
      console.log(err);
```

```
      res.status(500).json({
```

```
        error: err
```

```
      });
```

```
    });
```

```
  });
```


Routen sichern (protecting)

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

Middleware, die den JWT checkt



Neuer Order middleware/

Neues File check-auth.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  try {
    const token =
req.headers.authorization.split(" ")[1];

    const decoded = jwt.verify(token,
process.env.JWT_KEY); // check und decode
    req.userData = decoded;
    next(); // wenn es weitergehen darf
  } catch (error) {
    return res.status(401).json({
      message: 'Auth failed'
    });
  }
}
```

Middleware zu Routen hinzufügen

```
const checkAuth =  
require('../middleware/check-auth');  
  
router.post("/", checkAuth, (req, res,  
next) => { ...
```



Statt im Body besser im Header JWT

Header "Authorization": "Bearer <TOKEN>"

So holen wir es einfach aus dem aktuellen Request.



```
try {  
    const token =  
req.headers.authorization.split("  
")[1];  
    const decoded =  
jwt.verify(token,  
process.env.JWT_KEY);  
    req.userData = decoded;  
    next();  
}
```

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

The MongoDB 4.2 Manual

<https://docs.mongodb.com/manual/>

Vielen Dank für Ihre Aufmerksamkeit!

