

Das Ziel dieses und des nächsten Arbeitsblattes ist es, dir ein paar einfache Kenntnisse zu MongoDB und Node.js zu vermitteln, sodass du für die Daten in Node.js MongoDB verwenden kannst. Mehr Kenntnisse zu MongoDB gibt es dann in INSY, wenn wir das Thema *NoSQL Datenbanken* im Detail behandeln.

Warum MongoDB?

Du kennst relationale Datenbanken wie MySQL/MariaDB/PostgreSQL. Warum eine neue Datenbankart kennenlernen?

Die digitale Welt wächst rasant und wird immer komplexer, was das Volumen (Terabyte bis Zettabyte), die Vielfalt (strukturiert, semistrukturiert und unstrukturiert und die Geschwindigkeit (hohe Wachstumsgeschwindigkeit) angeht¹.

1 Kilobyte (KB)	2 ¹⁰ Bytes	1024 Bytes = 8192 Bits
1 Megabyte (MB)	2 ²⁰ Bytes	1024 KB = 1.048.576 Bytes = 8.388.608 Bits
1 Gigabyte (GB)	2 ³⁰ Bytes	1024 MB = 1.048.576 KB = 1.073.741.824 Bytes = ...
1 Terabyte (TB)	2 ⁴⁰ Bytes	1024 GB = 1.048.576 MB = 1.073.741.824 KB = ...
1 Petabyte (PB)	2 ⁵⁰ Bytes	1024 TB = 1.048.576 GB = 1.073.741.824 MB = ...
1 Exabyte (EB)	2 ⁶⁰ Bytes	1024 PB = 1.048.576 TB = 1.073.741.824 GB = ...
1 Zettabyte (TB)	2 ⁷⁰ Bytes	1024 EB = 1.048.576 PB = 1.073.741.824 TB = ...
1 Yottabyte (YB)	2 ⁸⁰ Bytes	...

Weltweite Datenmengen sollen bis 2025 auf 175 Zettabytes wachsen! Auch wenn es sich nur um eine Schätzung handelt, so ist jedenfalls sicher, dass die Datenmenge riesig wird.

Vergleiche dazu:

- Der Mensch (80 kg) besteht aus ca. 2⁹³ Atomen (Protonen-Massen).
- Die Astronomen schätzen, dass das sichtbare Universum aus 2⁷⁰ Sternen besteht.

Dies wird als Big Data bezeichnet und ist ein globales Phänomen. Darunter versteht man typischerweise eine Datensammlung, die so groß geworden ist, dass sie mit herkömmlichen Datenmanagement-Tools nicht mehr effektiv verwaltet oder genutzt werden kann: z. B. klassische relationale Datenbankmanagementsysteme (RDBMS) oder herkömmliche Suchmaschinen.

Oft lässt sich die Struktur der Daten nicht genau im Vorhinein festlegen. Ein Datenbankdesign mit anschließender Normalisierung wäre hier viel zu rigide. Das führte dazu, dass Datenbanken entwickelt wurden, deren Struktur dynamisch verändert werden kann. MongoDB ist eine davon. Es handelt sich bei **MongoDB um eine dokumentenorientierte NoSQL Datenbank**.

MongoDB speichert Dokumente als BSON Objekte ab, was binär kodierte JSON ist. Wenn du eine Abfrage ausführst, bekommst du ein JSON-Objekt zurück (oder einen String im JSON-Format, je nach Treiber). Schau dir zum Beispiel das folgende Code Snippet an:

```
_id: ObjectId("60b8ecbdcae96c51384abcde")
name: "Bello"
born: 2020-12-01T00:00:00.000+00:00
__v: 0
```

¹ Wenn dir dieses Kapitel bekannt vorkommt, ist das gut. Wir haben diesen Inhalt in der dritten Klasse in INSY behandelt. Siehe Kapitel *NoSQL Datenbanken* in *Folien 01 – Einführung und Motivation.pdf* 🤖

Ein Dokument ist also ein Satz von Keys (aka Properties bei Objekten) (z.B. **name**) und Werten (z.B. **Be11o**). Der Key `_id` ist der Identifier, den der zugrundeliegende MongoDB-Treiber standardmäßig für jedes neue Dokument anlegt.

Stell dir ein Dokument wie eine Zeile in einer Tabelle vor. In dieser Analogie entspricht ein Key einer Column. Ein wichtiger Unterschied ist, dass nicht jedes Dokument den exakt gleichen Satz an Keys enthalten muss und dass es keine direkte Notwendigkeit gibt, Keys mit leeren Werten zu haben, die Platz wegnehmen.

Eine Sammlung von Dokumenten wird als Collection bezeichnet. Die nächstliegende Analogie ist eine Tabelle. In deiner Datenbank könntest du also mehrere Collections haben, wie z.B. eine User Collection, eine Product Collection, etc.

MongoDB ist außerdem skalierbar, mit vielen eingebauten Funktionen für die Verteilung auf mehrere Server, ohne die Geschwindigkeit oder die Datenintegrität zu beeinträchtigen.

Daher lässt sich MongoDB ausgezeichnet mit Node.js integrieren. Beide Welten haben ihre eigenen Vor- und Nachteile und sind für unterschiedliche Arten von Anwendungsfällen gedacht.

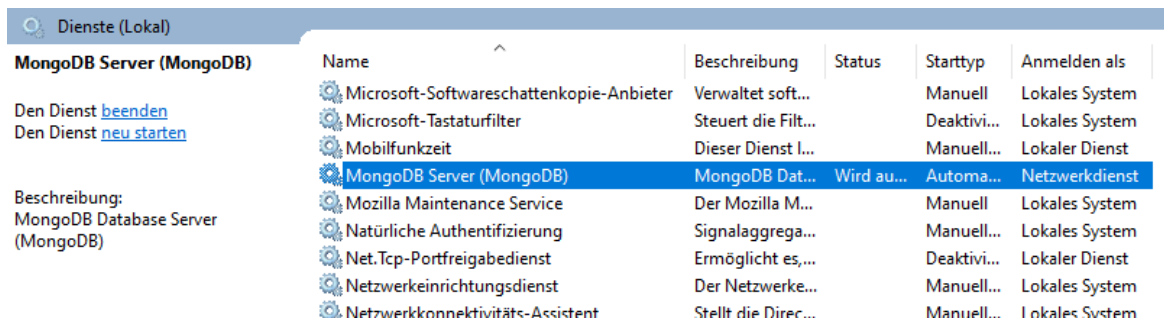
Setup

Du kannst MongoDB lokal installieren oder über die Cloud verwenden. Beachte, dass es in unserer Schule möglicherweise ein Verbindungsproblem zur Cloud gibt, da viele Ports gesperrt sind.

Lokales Setup:

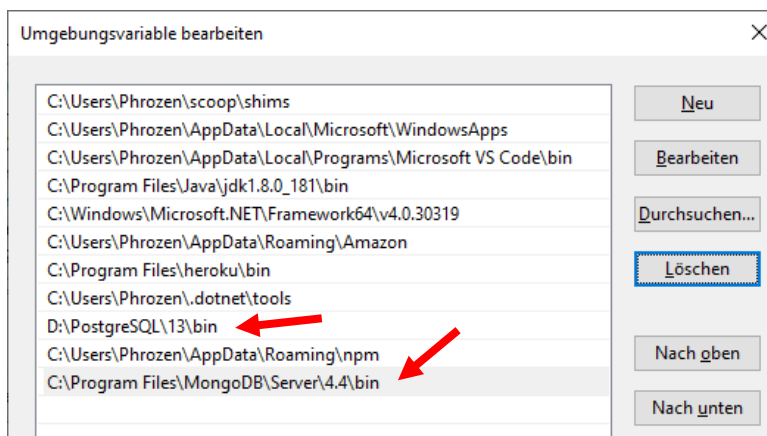
Gehe zu: <https://www.mongodb.com/try/download/community> und installiere den Server für dein Betriebssystem.

Unter Windows wird der Dienst **mongod** nach der Installation automatisch als Service gestartet.



Dienste (Lokal)					
	Name	Beschreibung	Status	Starttyp	Anmelden als
Den Dienst beenden Den Dienst neu starten	Microsoft-Softwareschattenkopie-Anbieter	Verwaltet soft...		Manuell	Lokales System
	Microsoft-Tastaturfilter	Steuert die Filt...		Deaktivi...	Lokales System
	Mobilfunkzeit	Dieser Dienst l...		Manuell...	Lokaler Dienst
	MongoDB Server (MongoDB)	MongoDB Dat...	Wird au...	Automa...	Netzwerkdienst
Beschreibung: MongoDB Database Server (MongoDB)	Mozilla Maintenance Service	Der Mozilla M...		Manuell	Lokales System
	Natürliche Authentifizierung	Signalaggrega...		Manuell...	Lokales System
	Net.Tcp-Portfreigabedienst	Ermöglicht es,...		Deaktivi...	Lokaler Dienst
	Netzwerkeinrichtungsdienst	Der Netzwerke...		Manuell...	Lokales System
	Netzwerkverbindungs-Assistent	Stellt die Direr...		Manuell...	Lokales System

Um die Shell **mongo** von überall zu benutzen, musst du die Environment Variable **path** wie bei PostgreSQL erweitern:



Teste die Datenbank in einer Shell mit dem Command **mongo** und dem Befehl **show dbs**.

```
Eingabeaufforderung - mongo
C:\Users\Phrozen>mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e450da99-8b35-45ee-9197-44bd1b1c1f48") }
MongoDB server version: 4.4.6
---
The server generated these startup warnings when booting:
  2021-05-29T08:55:10.151+02:00: Access control is not enabled for the database. Read and write a
ccess to data and configuration is unrestricted
  ---
  ---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
>
```

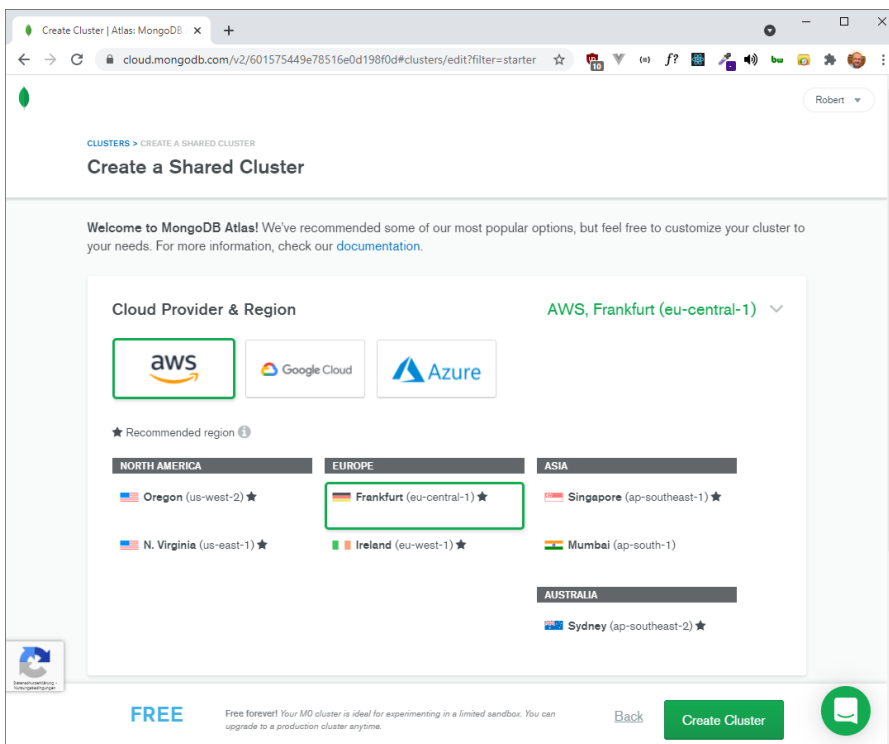
Beachte, dass standardmäßig die DB für alle offen ist (kein Superuser, kein PW)! Dadurch gibt es viele MongoDB Installationen weltweit, die nicht abgesichert sind. Unter anderem auch weil das Anlegen eines Superusers umständlich ist.²

Da deine MongoDB nur auf deinem Rechner installiert ist und vom Internet nicht zugegriffen wird, ist das für Entwicklungszwecke OK.

Cloud Setup:

Gehe zu: <https://www.mongodb.com/> und hole dir eine Account und logge ein.

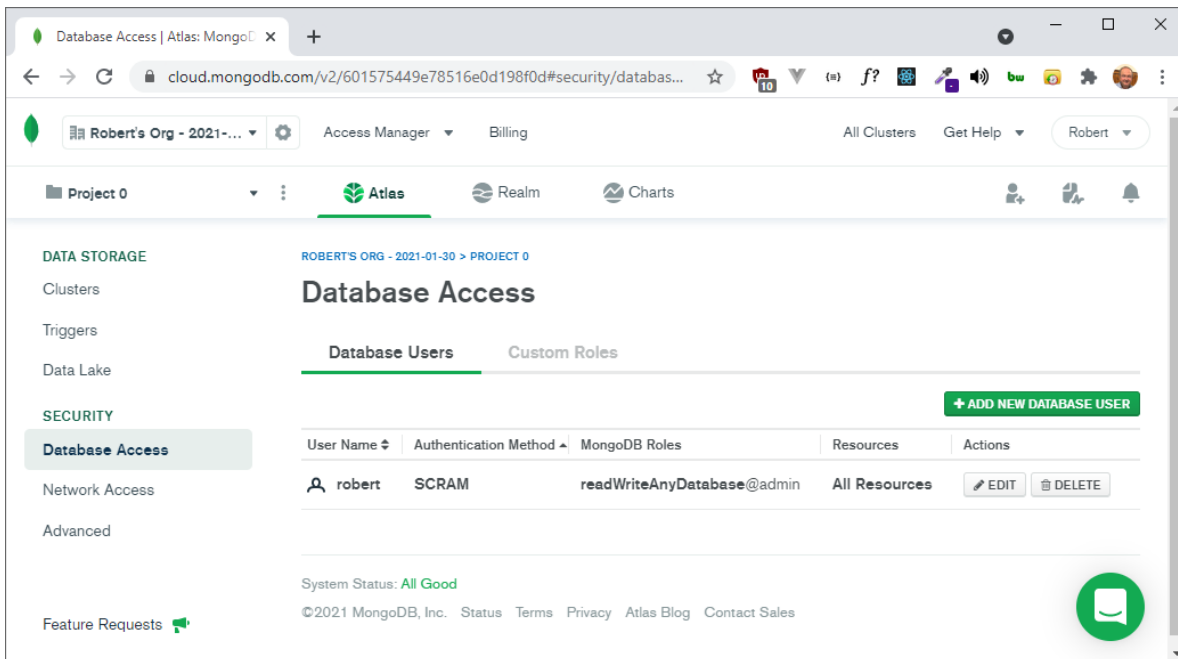
Lege einen Shared Cluster in Frankfurt an:



² Beispiel: <https://securityaffairs.co/wordpress/105485/hacking/hackers-mongodb-database.html>

Das Anlegen eines Superusers für MongoDB lernst du in INSY 5.Klasse beim Thema NoSQL Datenbanken.

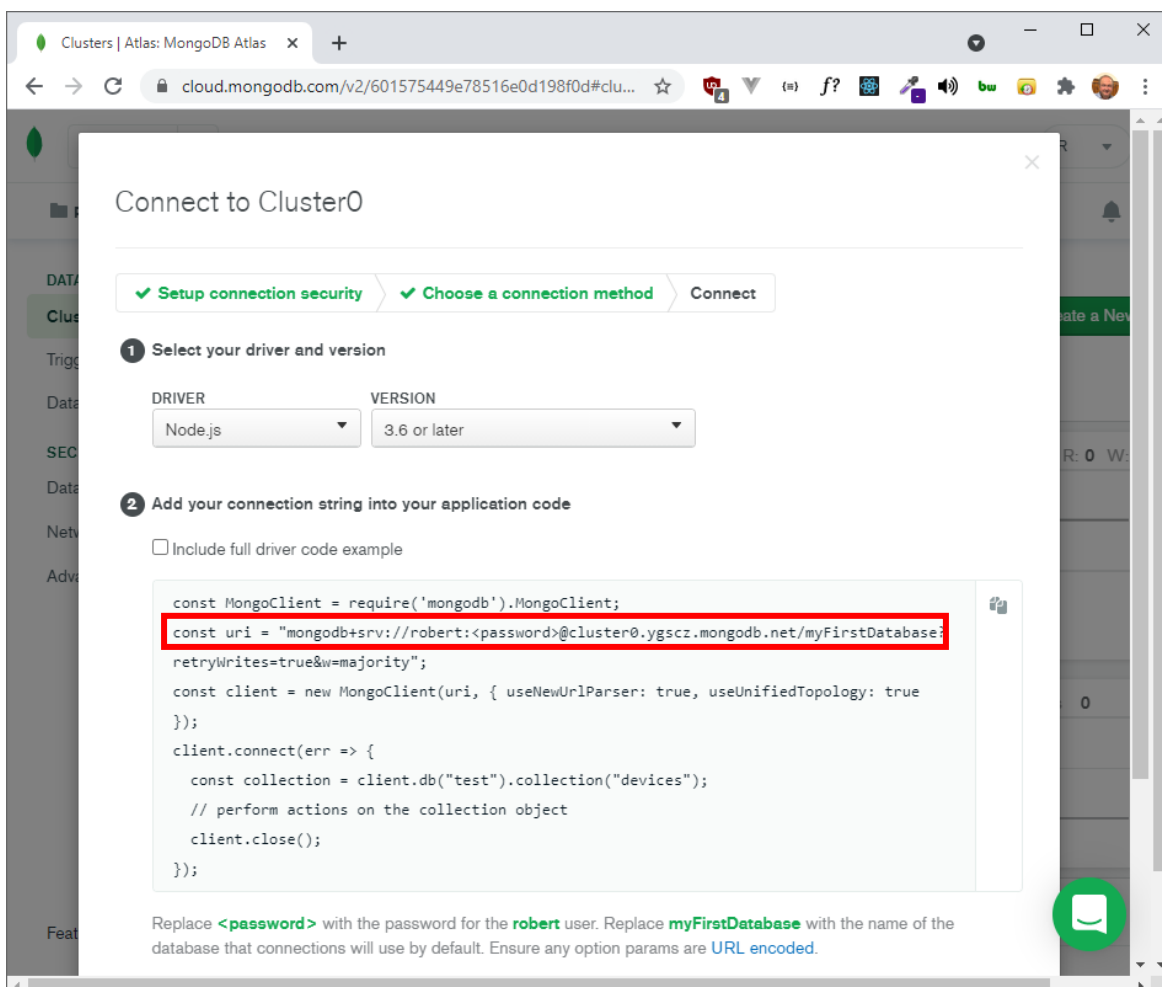
Füge einen neuen Database User hinzu.



The screenshot shows the MongoDB Atlas 'Database Access' page for 'Project 0'. The left sidebar has 'Database Access' selected under the 'SECURITY' section. The main area shows a table of 'Database Users'. One user, 'robert', is listed with the authentication method 'SCRAM' and the role 'readWriteAnyDatabase@admin'. A green '+ ADD NEW DATABASE USER' button is visible in the top right. The system status is 'All Good'.

User Name	Authentication Method	MongoDB Roles	Resources	Actions
robert	SCRAM	readWriteAnyDatabase@admin	All Resources	EDIT DELETE

Gehe dann wieder zu Clusters und klicke auf **CONNECT**. Wähle dort **Connect your application** aus. Danach siehst du den Connection String als URI:



The screenshot shows the 'Connect to Cluster0' dialog box. It has two tabs: 'Setup connection security' (active) and 'Choose a connection method'. Under 'Setup connection security', there are two steps:

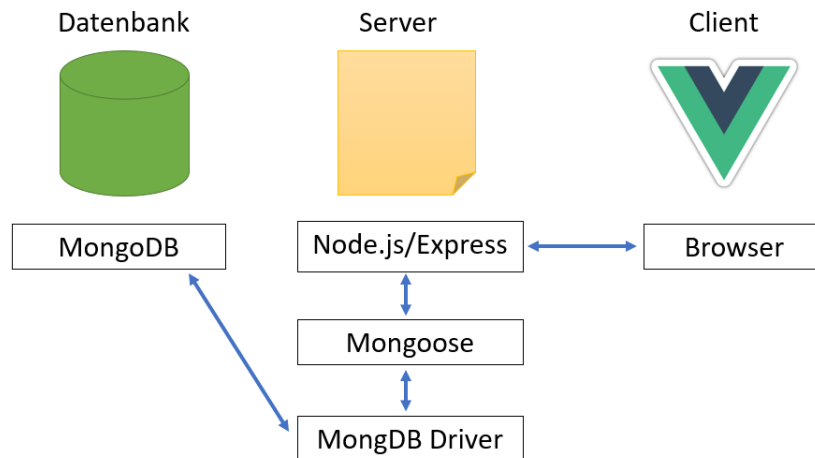
- Select your driver and version**: The 'DRIVER' is set to 'Node.js' and the 'VERSION' is '3.6 or later'.
- Add your connection string into your application code**: There is a checkbox for 'Include full driver code example' which is unchecked. Below it is a code block with a red box highlighting the connection string URI.

```
const MongoClient = require('mongodb').MongoClient;
const uri = "mongodb+srv://robert:<password>@cluster0.ygscz.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });
client.connect(err => {
  const collection = client.db("test").collection("devices");
  // perform actions on the collection object
  client.close();
});
```

Below the code block, there is a note: 'Replace <password> with the password for the robert user. Replace myFirstDatabase with the name of the database that connections will use by default. Ensure any option params are URL encoded.'

MongoDB vs. Mongoose

Mit **MongoDB** direkt zu arbeiten ist nicht schwer, allerdings ist es für simple JavaScript Objekte einfacher eine ODM (Object Data Modeling) Library wie **Mongoose** zu verwenden. Mongoose verwaltet Beziehungen zwischen Daten, bietet Schema-Validierung und erledigt das Mapping zwischen Objekten im Code und der Darstellung dieser Objekte in MongoDB.



Das erspart uns eine Menge Boilerplate-Code zu schreiben. Bei komplexeren Datenstrukturen, kann es kompliziert werden Mongoose zu verwenden, da die Library gewisse Regeln vorgibt, an die man sich besser hält. Außerdem empfehle ich dir Mongoose nicht zu verwenden, wenn:

- deine Daten keinem Schema folgen,
- deine Dokumente zufällig aus den Daten zusammengestellt werden,
- deine Daten reine Key/Value Paare sind.

Aufgabe 1: Entpacke die Zip-Datei **Server Basis Pimped V1.1.zip** und installiere das npm Modul **mongoose**.

Ändere **.env** File, sodass dort die URL zur Datenbank hinterlegt ist.

Lege einen Ordner **db** mit einer Datei **connect.js** an. In dieser Datei erfolgt die Definition der Verbindung zur Datenbank.

Mongoose erstellt automatisch einen Pool von fünf wiederverwendbaren Verbindungen (Connections), wenn du dich zum ersten Mal mit Mongoose zu einer MongoDB verbindest. Dieser Pool wird von allen nachfolgenden Requests verwendet.

Da **Mongoose** auf **MongoDB** basiert und sich beide ständig unabhängig voneinander ändern, musst du derzeit die folgenden Optionen beim Connect angeben, oder mit den Warnings (*ugly*) leben.

Siehe <https://mongoosejs.com/docs/deprecations.html>

Auf die Herstellung der Verbindung (Connect) müssten wir nicht warten (kein **await**) da alle Calls intern gepuffert werden. Geht allerdings das Connect schief, gibt es mittendrin eine Exception. Daher warten wir lieber und geben im Fehlerfall den Error aus!

Mongoose verwendet den Node Event Emitter um die Events **connected**, **disconnect** zu senden. Diese fangen wir ab und protokollieren sie.

Best Practice: Öffne die Verbindung, wenn deine Anwendung startet; lass sie offen, bis deine Anwendung neu startet oder herunterfährt. Dann allerdings solltest du die Verbindung schließen, da Mongoose diese nicht automatisch schließt! Verwende dazu den von Node.js generierten SIGINT Event.

```
const { connect, connection } = require('mongoose');
require('dotenv').config();
require('colors');

async function beginConnection() {
  try {
    await connect(process.env.DATABASE_URL, {
      useUnifiedTopology: true,
      useNewUrlParser: true,
      useCreateIndex: true,
      useFindAndModify: false,
    });
  } catch (error) {
    console.error(`Error ==> ${error.message.red}`);
    process.emit('SIGINT');
  }
}

connection.on('connected', () => console.log('Mongoose connected'.blue));
connection.on('disconnected', () => console.log('Mongoose disconnected'.blue));

process.on('SIGINT', () => {
  connection.close(() => {
    console.log('Mongoose terminated through app termination'.blue);
    process.exit(0);
  });
});

beginConnection();
```

Hier wäre es cool eine **IIFE** (Immediately Invoked Function Expression) zu verwenden. Das ist einfach eine Arrow Function, die mit () umschlossen und mit () aufgerufen wird.

```
const demo = () => console.log('ich bin keine IIFE!'); // normale Arrow function
demo(); // muss aufgerufen werden
(() => console.log('ich bin eine IIFE!'))(); // wird automatisch definiert und aufgerufen
```

↑ Die Klammern wrappen die Arrow Function ↑ Das ist der Call Operator für den Aufruf

Ein weiteres Beispiel:

```
(async () => {
  const { data } = await axios.get('http://localhost:3000/movies');
  console.log(data);
})();
```

IIFEs sind ideal, wenn man async Functions erstellt und diese gleich aufrufen möchte.

Somit können wir den Code für **connect.js** vereinfachen (siehe nächste Seite).

```
// ...
(async () => {
  try {
    await connect(process.env.DATABASE_URL, {
      useUnifiedTopology: true,
      useNewUrlParser: true,
      useCreateIndex: true,
      useFindAndModify: false,
    });
  } catch (error) {
    console.error(`Error ==> ${error.message.red}`);
    process.emit('SIGINT');
  }
})();

connection.on('connected', () => console.log('Mongoose connected'.blue));
connection.on('disconnected', () => console.log('Mongoose disconnected'.blue));

process.on('SIGINT', () => {
  connection.close(() => {
    console.log('Mongoose terminated through app termination'.blue);
    process.exit(0);
  });
});
```

Aufgabe 2: Der nächste Schritt ist ein **Schema** und ein **Model** zu erstellen. Ein Schema definiert die Struktur der Daten, Standardwerte, Validatoren usw.. Ein Model ist eine kompilierte Version eines Schemas. Es ist die Schnittstelle zur Datenbank zum Erstellen, Abfragen, Aktualisieren, Löschen von Datensätzen usw..

Beginnen wir simpel. Wir wollen den Namen und das Geburtsdatum eines Hundes speichern. Erstelle ein Verzeichnis **model** und darin die Datei **dogs.js**.

Importiere **Schema** und **model** von Mongoose.

```
const { Schema, model } = require('mongoose');
```

Mongoose bietet Standarddatentypen wie String, Number, Boolean, Date, etc.

Siehe <https://mongoosejs.com/docs/schematypes.html>

Erstelle folgenden Code in **dogs.js**. Achte dabei auf Groß- und Kleinschreibung der Variablen, die standardisiert ist. Verwende Singular, MongoDB macht selbstständig Plural daraus!

```
const { Schema, model } = require('mongoose');

const dog = new Schema({
  name: String,
  born: Date
});
const Dog = model('dog', dog);

module.exports = { Dog };
```

*Name der Collection, wird zu **dogs***

Name des Schemas

3. Nun wollen wir eine Collection **dogs** in der Datenbank **puppy-love** erstellen und befüllen. Praktisch: Wenn wir ein neues Objekt mittels eines Models erzeugen, erstellt MongoDB automatisch eine Collection, wenn keine existiert.

Da das Laden der Datenbank nichts mit dem Node Server zu tun hat, erstelle dazu im Rootverzeichnis eine eigene Datei namens **loadData.js** (siehe unten).

Mithilfe des Models werden zwei Objekte in der Datenbank erzeugt. Aus Gründen der Performance wird nicht mit **await** gewartet. Stattdessen werden die Promises in Variable gespeichert und erst am Ende mit **Promise.all** gewartet.

Du kannst hier natürlich auch eine IIFE verwenden!

```
require('./db/connect');
require('colors');
const { Dog } = require('./model/schemas');

(async () => {
  try {
    const p1 = Dog.create({
      name: 'Bello',
      born: '2020-12-01',
    });
    const p2 = Dog.create({
      name: 'Beisser',
      born: '2020-12-02',
    });
    await Promise.all([p1, p2]);
    console.log('Data loaded');
  } catch (error) {
    console.error(`Error ====> ${error.message.red}`);
  } finally {
    process.emit('SIGINT');
  }
})();
```

```
> node loadData.js
```

```
Mongoose disconnected
Error ====> Authentication failed.
Mongoose terminated through app termination
```

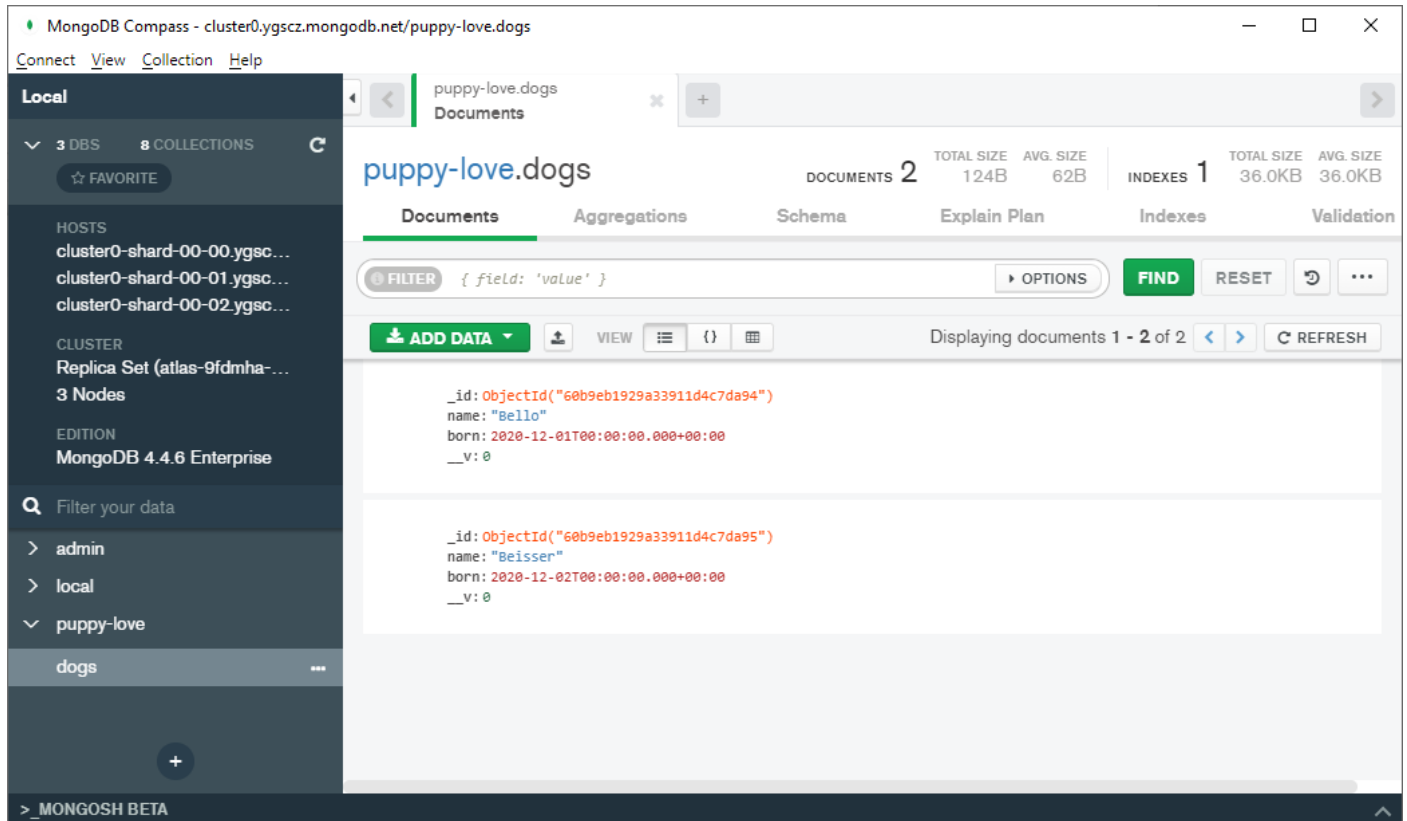
4. Baue dir ein Skript in **package.json**, um bequem das Laden durchzuführen. Überprüfe das Ergebnis in der MongoDB Datenbank.

Zum Beispiel mit der Mongo Shell.



```
Eingabeaufforderung - mongo mongodb+srv://robert:alleswirdgut2020@cluster0.ygscz.mongodb.net/puppy-love
switched to db puppy-love
MongoDB Enterprise atlas-9fdmha-shard-0:PRIMARY> db.dogs.find()
{ "_id" : ObjectId("60b8ecbdcae96c51384abcde"), "name" : "Bello", "born" : ISODate("2020-12-01T00:00:00Z"), "__v" : 0 }
{ "_id" : ObjectId("60b8ecbdcae96c51384abcdf"), "name" : "Beisser", "born" : ISODate("2020-12-02T00:00:00Z"), "__v" : 0 }
MongoDB Enterprise atlas-9fdmha-shard-0:PRIMARY>
```


Oder via MongoDB Compass (<https://www.mongodb.com/products/compass>):



Oder direkt am Web, wenn Atlas verwendet wurde.

Nun können wir auch die **GET /dogs** Route bauen.

5. Importiere **connect.js** in **app.js**. Somit wird die Verbindung nur beim Hochfahren des Servers aufgebaut.

Erstelle im Verzeichnis **model** eine zusätzliche Datei **dogs.js**:

```
const { Dog } = require('./schemas');  
  
const getDogs = () => Dog.find();  
  
module.exports = { getDogs };
```

Wird auf dem Model **Dog** die **find** Methode aufgerufen, werden alle Dokumente in der Collection zurückgeliefert!

Erstelle die Route **GET /dogs** erstellen und teste mittels REST Client!

5. Lösche die Daten in der Datenbank. und lade die Datei **dogs.json** in **loadData.js**.

Erstelle zum Löschen eine Datei **removeData.js** und verwende dazu:

```
await Dog.deleteMany({});
```

6. Lade die Datei **dogs.json** in **loadData.js**. Tipp: Du kannst mit **create** ein Array von Objekten übergeben!

MongoDB Compass - cluster0.ygscz.mongodb.net/puppy-love.dogs

Connect View Collection Help

Local

3 DBS 8 COLLECTIONS

☆ FAVORITE

HOSTS

- cluster0-shard-00-00.ygsc...
- cluster0-shard-00-01.ygsc...
- cluster0-shard-00-02.ygsc...

CLUSTER

Replica Set (atlas-9fdmha-...)

3 Nodes

EDITION

MongoDB 4.4.6 Enterprise

Filter your data

- admin
- local
- puppy-love
- dogs

puppy-love.dogs Documents

DOCUMENTS 2 TOTAL SIZE 124B AVG. SIZE 62B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND RESET ↺ ...

ADD DATA VIEW {}

Displaying documents 1 - 20 of 20 REFRESH

<pre>{ "_id": ObjectId("60ba19a80ead46362caa5900"), "born": 2019-10-16T00:00:00.000+00:00, "name": "Cupcake", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5901"), "born": 2021-04-22T00:00:00.000+00:00, "name": "Britney Ears", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5902"), "born": 2019-10-09T00:00:00.000+00:00, "name": "Buddy", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5903"), "born": 2021-02-28T00:00:00.000+00:00, "name": "Oscar", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5904"), "born": 2019-11-05T00:00:00.000+00:00, "name": "Milo", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5905"), "born": 2019-11-20T00:00:00.000+00:00, "name": "Bones", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5906"), "born": 2018-11-17T00:00:00.000+00:00, "name": "ollie", "__v": 0 }</pre>
<pre>{ "_id": ObjectId("60ba19a80ead46362caa5907"), "born": 2019-03-18T00:00:00.000+00:00, "name": "Milo", "__v": 0 }</pre>

> _MONGOSH BETA



```
1 // 20210604142709
2 // http://localhost:3000/dogs
3
4 [
5   {
6     "_id": "60ba19a80ead46362caa5900",
7     "born": "2019-10-16T00:00:00.000Z",
8     "name": "Cupcake",
9     "__v": 0
10  },
11  {
12    "_id": "60ba19a80ead46362caa5901",
13    "born": "2021-04-22T00:00:00.000Z",
14    "name": "Britney Ears",
15    "__v": 0
16  },
17  {
18    "_id": "60ba19a80ead46362caa5902",
19    "born": "2019-10-09T00:00:00.000Z",
20    "name": "Buddy",
21    "__v": 0
22  },
23  {
24    "_id": "60ba19a80ead46362caa5903",
25    "born": "2021-02-28T00:00:00.000Z",
26    "name": "Oscar",
27    "__v": 0
28  },
29  {
30    "_id": "60ba19a80ead46362caa5904",
31    "born": "2019-11-05T00:00:00.000Z",
32    "name": "Milo",
33    "__v": 0
34  },
35  {
36    "_id": "60ba19a80ead46362caa5905",
37    "born": "2019-11-20T00:00:00.000Z",
38    "name": "Bones",
39    "__v": 0
40  },
41  {
42    "_id": "60ba19a80ead46362caa5906",
```