

1 Implementierung der Datenbank

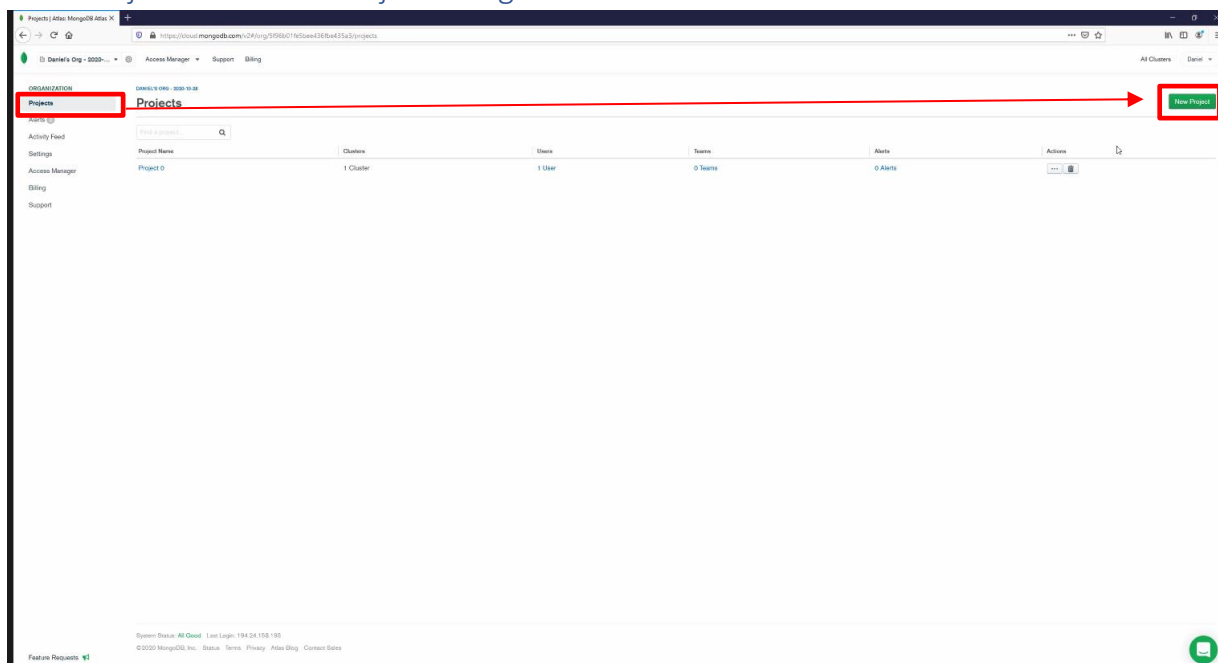
1.1 Account bei MongoDB anlegen:

<https://account.mongodb.com/account/login>

1.2 Nach Erstellung des Accounts bei MongoDB Atlas einloggen

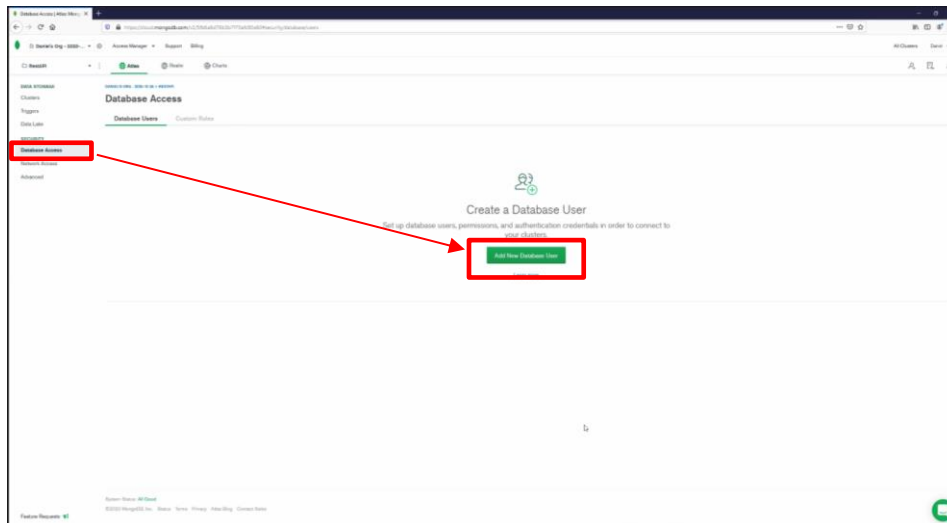
<https://account.mongodb.com/account/login>

1.3 Projects -> Neues Projekt anlegen



- Menüleiste links Button **“Projects”**
- Button rechts oben **„Create Project”** anklicken
- **Projektnamen eingeben** -> Button **„Next”** klicken
- Bei Add Members müssen keine Personen hinzugefügt werden, weil der Zugriff uneingeschränkt ist -> Button **„Create Project”** klicken

1.4 Database Access -> Datenbank User erstellen



- Menüleiste links Button **“Database Access”**
- Klick auf grünen Button **„Add new Database User“**

Add New Database User

Create a database user to grant an application or user, access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding [Access Manager](#).

Authentication Method

Password Certificate (M10 and up) AWS IAM (MongoDB 4.4 and up)

MongoDB uses **SCRAM** as its default authentication method.

Password Authentication

username(admin)

SHOW

Autogenerate Secure Password Copy

Database User Privileges

Select a built-in role or privileges for this user.

Read and write to any database

Restrict Access to Specific Clusters/Data Lakes

Enable to specify the resources this user can access. By default, all resources in this project are accessible.

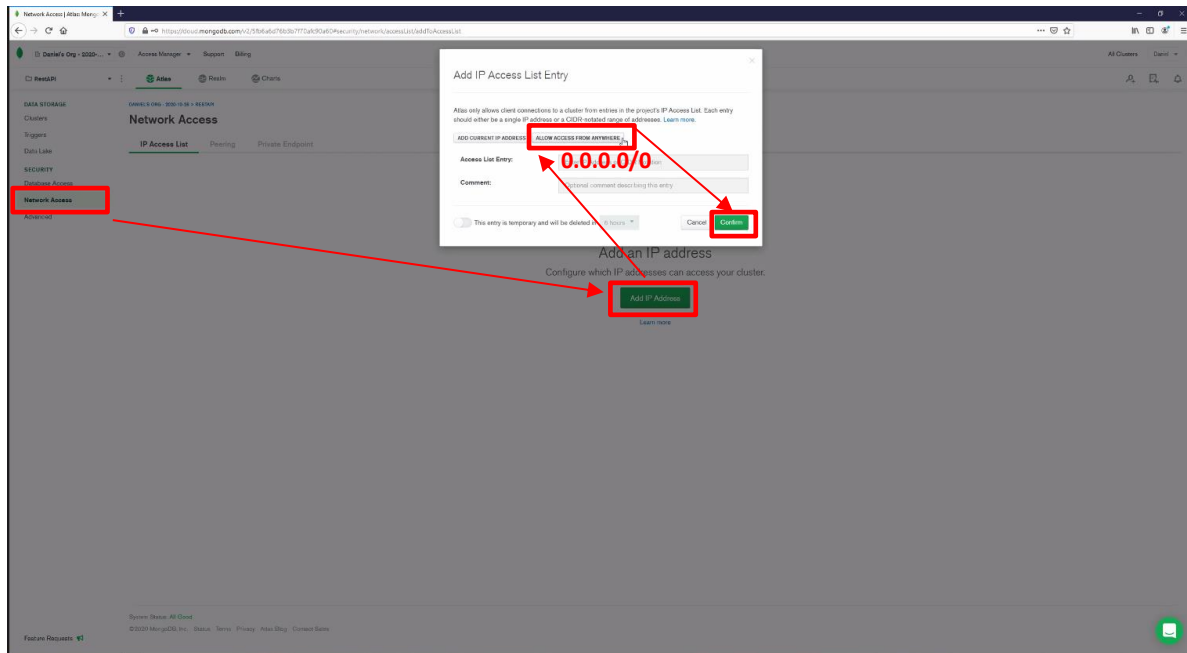
Temporary User

This user is temporary and will be deleted after your specified duration of 6 hours, 1 day, or 1 week.

Cancel Add User

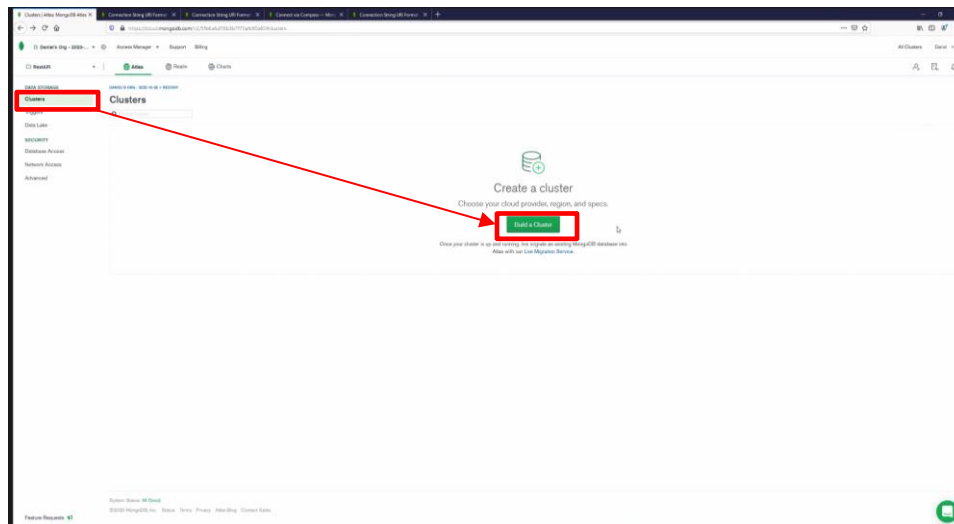
- Authentication Method = Box **„Password“**
- Password Authentication -> **Username vergeben** z.B. „admin“
- **Passwort vergeben**
- Database User Privileges => Read and write to any database (standardmäßig ausgewählt)
- Button **“Add User”**

1.5 Network Access (Zugang standortunabhängig erlauben)



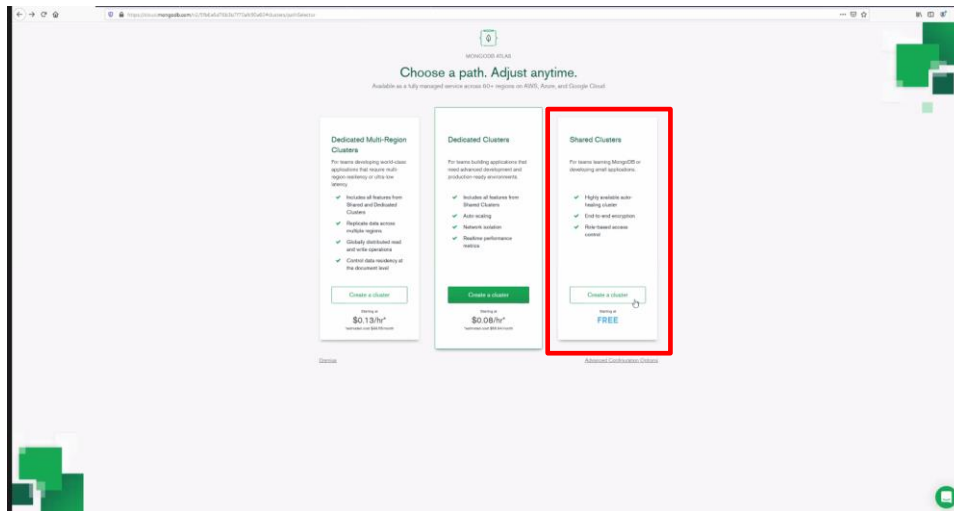
- Menüleiste links Button **“Network Access”**
- In der Mitte grüner Button „Add IP Address“ anklicken
- Button **„Allow access from everywhere“** auswählen => Access List Entry = 0.0.0.0/0 (automatisch)
- Mit Button **“Confirm”** bestätigen

1.6 Cluster erstellen

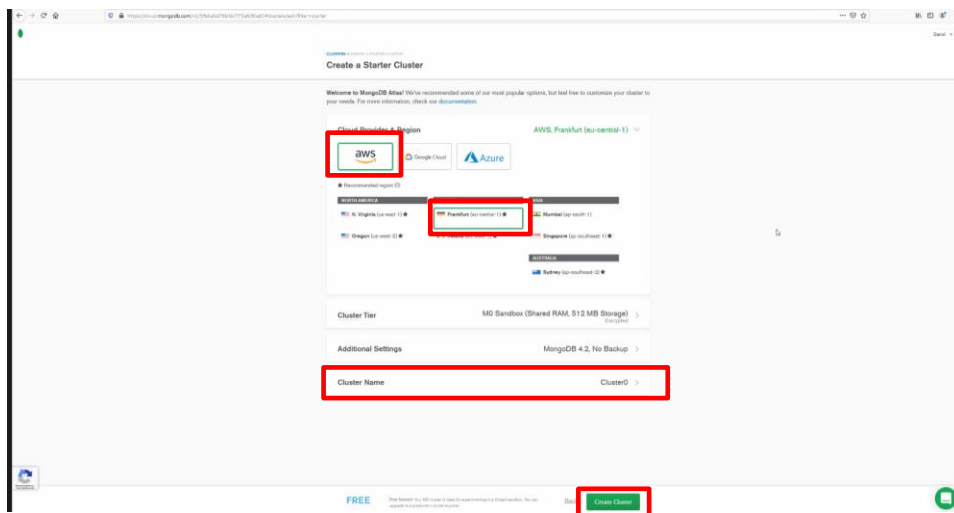


- Menüleiste links Button **“Network Access”**
- Button **“Build a cluster”** anklicken

Anleitung REST-API mit React

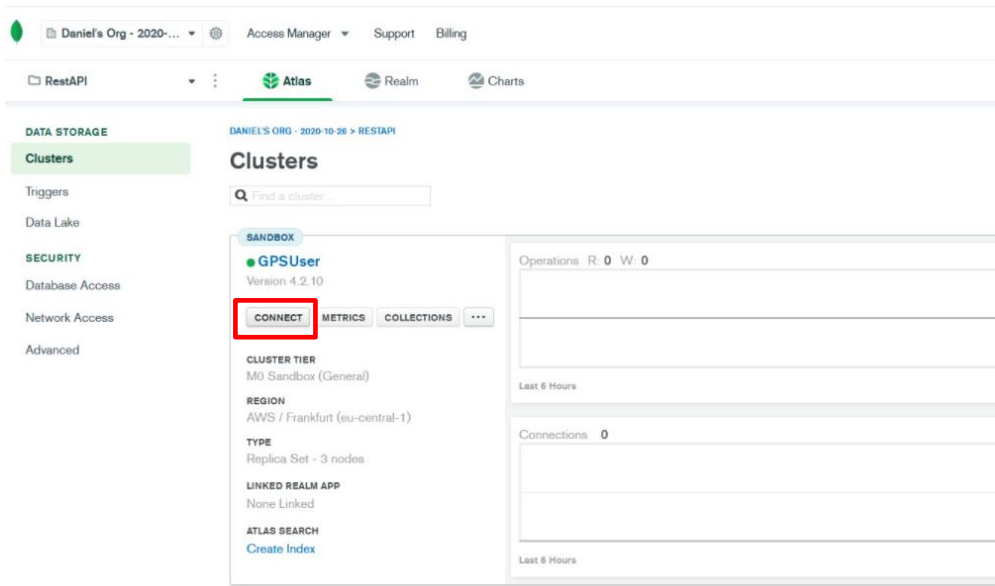


- “Shared cluster” auswählen



- Provider “AWS” und “Frankfurt” auswählen
- Optional: ClusterName vergeben
- Button „Create Cluster“ auswählen (Herstellvorgang kann bis zu 5 Minuten dauern)

Anleitung REST-API mit React



- Button „Connect“ anklicken

Connect to GPSUser

✓ Setup connection security ✓ Choose a connection method Connect

1 Choose your version of Compass:

1.12 or later

See your Compass version in "About Compass"

2 Copy the connection string, then open MongoDB Compass.

```
mongodb+srv://admin:<password>@gpsuser.onqwg.mongodb.net/test
```

Copy

You will be prompted for the password for the **admin** user's (Database User) username.
When entering your password, make sure that any special characters are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

2 Implementierung von REST API in React

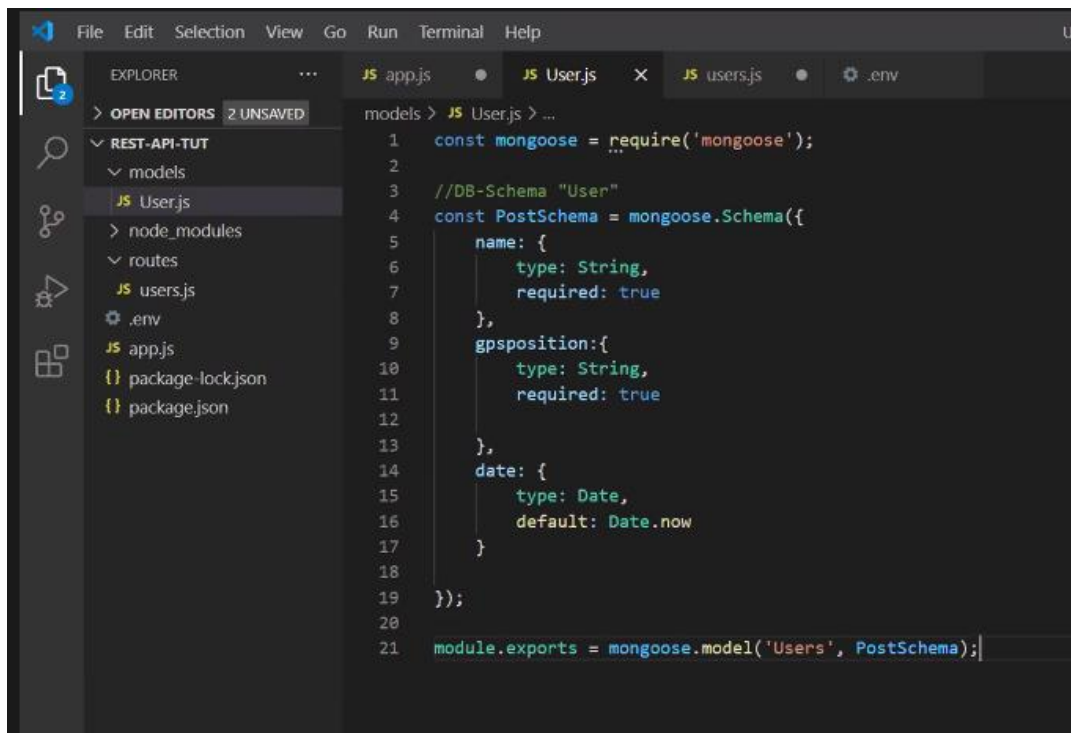
2.1 Code für Datei users.js

```
1 const express = require('express');
2 //nodemon ist notwendig um Quellcode-Änderungen automatisch zu laden
3 const { restart } = require('nodemon');
4 //Notwendig für Antworten auf Clientanforderung (HTTP)
5 const router = express.Router();
6 //Klasse "User" wird eingefügt
7 const User = require('../models/User');
8
9 //REST-API
10 //GETS BACK ALL THE USER/ENTRIES
11 //Liste von Usern wird zurückgegeben
12 router.get('/', async (req,res) => {
13   try {
14     const users = await User.find();
15     res.json(users);
16   } catch(err) {
17     res.json({message: err});
18   }
19 });
20
21 //SUBMITS A USER/ENTRY
22 //User wird folgenden Feldern angelegt (name, gpsposition, time)
23 router.post('/', async (req,res) => {
24   const user = new User({
25     name: req.body.name,
26     gpsposition: req.body.gpsposition,
27   });
28   try {
29     const savedUser = await user.save();
30     res.json(savedUser);
31   } catch(err) {
32     res.json({message: err});
33   }
34 });
35
36 //SPECIFIC USER/ENTRY BY ID
37 //User wird anhand der ID identifiziert und zurückgegeben
38 router.get('/:userId', async (req,res) => {
39   try {
40     const user = await User.findById(req.params.userId);
41     res.json(user);
42   } catch(err) {
43     res.json({message: err});
44   }
45 });
```

```
35 //SPECIFIC USER/ENTRY BY ID
36 //User wird anhand der ID identifiziert und zurückgegeben
37 router.get('/:userId', async (req,res) => {
38   try {
39     const user = await User.findById(req.params.userId);
40     res.json(user);
41   } catch(err) {
42     res.json({message: err});
43   }
44 });
45
46 //DELETE USER/ENTRY
47 //User wird anhand der ID identifiziert und gelöscht
48 router.delete('/:userId', async (req,res) => {
49   try {
50     const removeUser = await User.remove({_id: req.params.userId});
51     res.json(removeUser);
52   } catch(err) {
53     res.json({message: err});
54   }
55 });
56
57 //UPDATE USER/ENTRY
58 //User wird anhand der id identifiziert und die Felder "gpspositio" und "time" werden aktualisiert
59 router.patch('/:userId', async (req,res) => {
60   try {
61     const updatedUser = await User.updateOne(
62       {_id: req.params.userId},
63       {$set: {gpsposition: req.body.gpsposition,
64         time: Date.now}}
65     );
66     res.json(updatedUser);
67   } catch(err) {
68     res.json({message: err});
69   }
70 });
71
72 module.exports = router;
```

2.2 Code für User.js

Die Klasse der User.



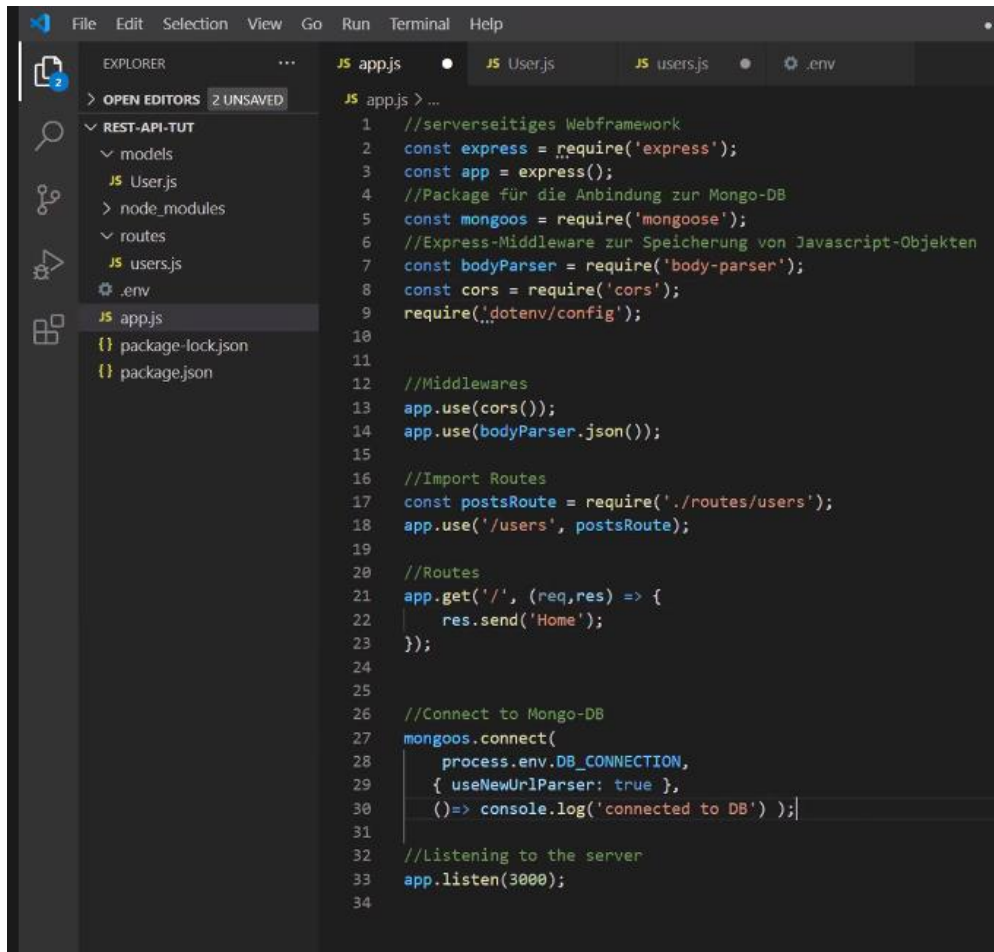
The screenshot shows a Visual Studio Code editor window with the following details:

- Explorer Panel:** Shows the project structure. The 'models' folder is expanded, and 'User.js' is selected. Other files visible include 'app.js', 'users.js', '.env', 'package-lock.json', and 'package.json'.
- Editor Panel:** Displays the code for 'User.js'. The code defines a Mongoose schema for a 'User' model.

```
1  const mongoose = require('mongoose');
2
3  //DB-Schema "User"
4  const PostSchema = mongoose.Schema({
5    name: {
6      type: String,
7      required: true
8    },
9    gpsposition:{
10     type: String,
11     required: true
12   },
13   date: {
14     type: Date,
15     default: Date.now
16   }
17 });
18
19
20
21 module.exports = mongoose.model('Users', PostSchema);
```

2.3 Code für app.js

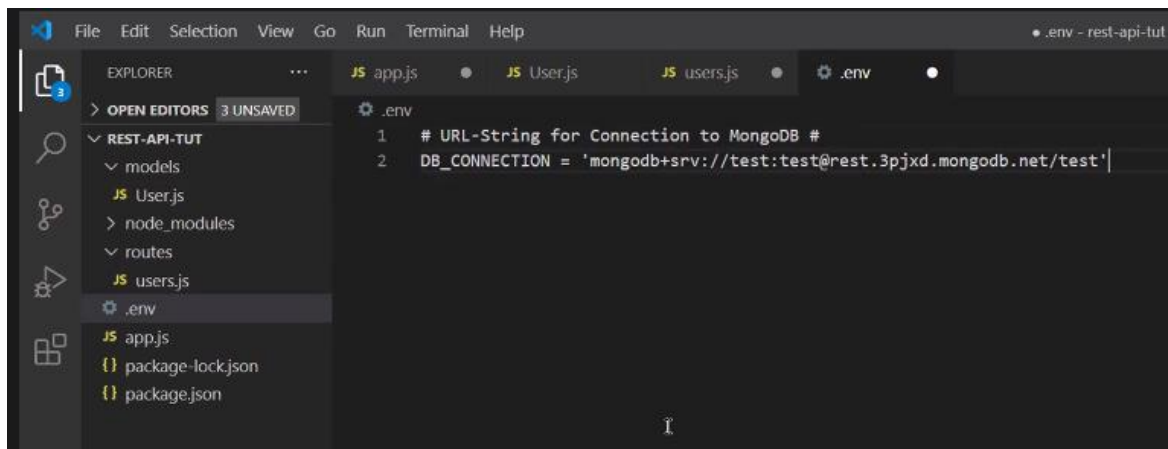
Hier wird die Verbindung zur Datenbank aufgebaut und mit dem Port 3000 verbunden.



```
1 //serverseitiges Webframework
2 const express = require('express');
3 const app = express();
4 //Package für die Anbindung zur Mongo-DB
5 const mongoos = require('mongoose');
6 //Express-Middleware zur Speicherung von Javascript-Objekten
7 const bodyParser = require('body-parser');
8 const cors = require('cors');
9 require('dotenv/config');
10
11
12 //Middlewares
13 app.use(cors());
14 app.use(bodyParser.json());
15
16 //Import Routes
17 const postsRoute = require('./routes/users');
18 app.use('/users', postsRoute);
19
20 //Routes
21 app.get('/', (req,res) => {
22   res.send('Home');
23 });
24
25
26 //Connect to Mongo-DB
27 mongoos.connect(
28   process.env.DB_CONNECTION,
29   { useNewUrlParser: true },
30   ()=> console.log('connected to DB') );
31
32 //Listening to the server
33 app.listen(3000);
34
```

2.4 Code für .env

Hier wird der URL-String für die Verbindung zur Mongo-Datenbank angegeben.



```
1 # URL-String for Connection to MongoDB #
2 DB_CONNECTION = 'mongodbsrv://test:test@rest.3pjxd.mongodb.net/test'
```


3 Erstellung Postman-Account für Testen der REST API

3.1 Einrichtung Postman Applikation

- 1) Account erstellen

<https://www.postman.com/product/api-client/>

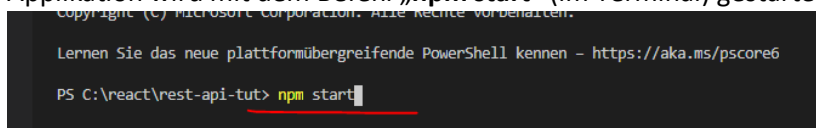
- 2) Anwendung „Postman“ downloaden
- 2) Applikation starten

4 REST-API Applikation

4.1 REST-API starten

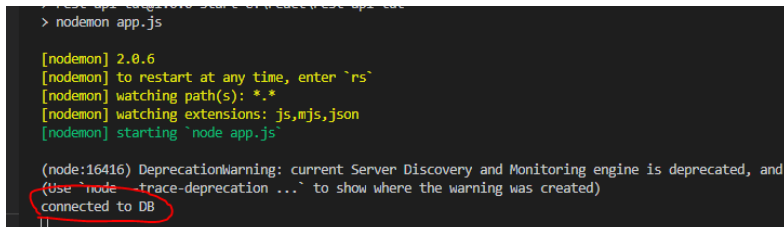
Rest-API wird mit Postman getestet

- 1) Applikation wird mit dem Befehl „**npm start**“ (im Terminal) gestartet



```
Copyright (c) Microsoft Corporation. Alle Rechte vorbehalten.  
Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6  
PS C:\react\rest-api-tut> npm start
```

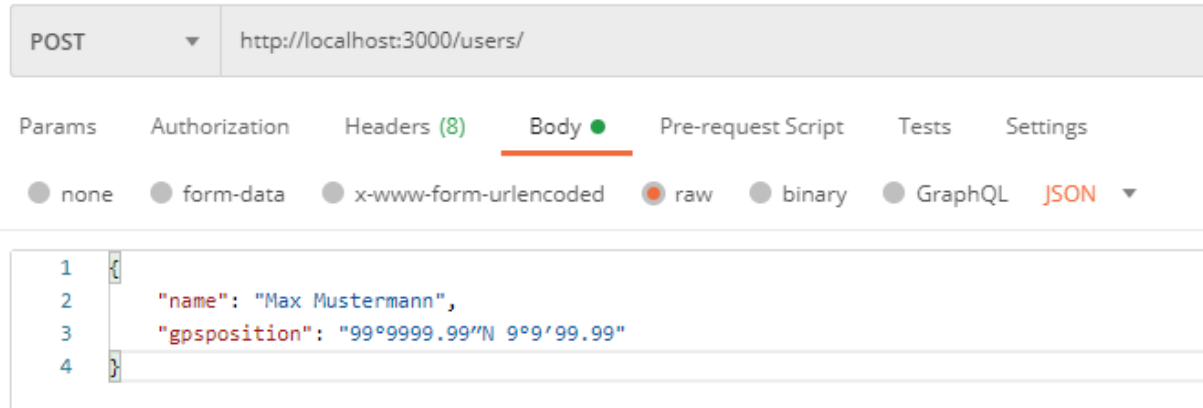
- 2) Wenn der Befehl erfolgreich durchgeführt wurde, wird die Erfolgsbestätigung „**connected to DB**“ angezeigt.



```
> nodemon app.js  
[nodemon] 2.0.6  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node app.js`  
  
(node:16416) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version.  
(Use `node --trace-deprecation ...` to show where the warning was created)  
connected to DB
```

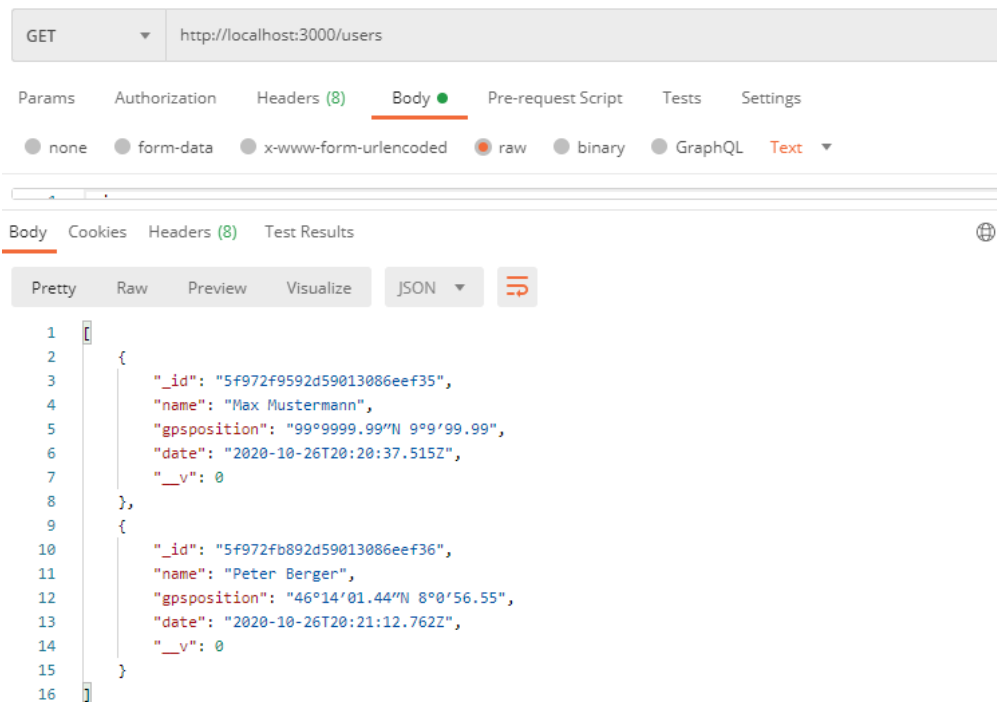
4.2 POST-Methode

- Neuen Datensatz in der Datenbank anlegen.
- Die ID wird automatisch generiert und mit dem Datensatz verknüpft.



4.3 GET-Methode

- Es werden alle Datensätze der „Tabelle“ angezeigt.



Anleitung REST-API mit React

- Es werden bestimmte Datensätze über eine Query gefiltert, `_id` wird als Identifier verwendet.

GET `http://localhost:3000/users/5f972f9592d59013086eef35`

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **Text**

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "5f972f9592d59013086eef35",
3   "name": "Max Mustermann",
4   "gpsposition": "99°9999.99"N 9°9'99.99",
5   "date": "2020-10-26T20:20:37.515Z",
6   "__v": 0
7 }
```

4.4 PATCH-Methode

- Updaten eines Datensatzes über die ID (mit der Angabe der zu ändernden Felder, z.B. "name").

PATCH `http://localhost:3000/users/5f972f9592d59013086eef35`

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Miriam Musterfrau"
3 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "n": 1,
3   "nModified": 1,
4   "opTime": {
5     "ts": "6891630671198945320",
6     "t": 4
7   },
8   "electionId": "7fffffff0000000000000004",
9   "ok": 1,
10  "$clusterTime": {
11    "clusterTime": "6891630671198945320",
12    "signature": {
13      "hash": "FDj1yTixx4wwjpTjLwwafb9NdP4=",
14      "keyId": "6886926359159898115"
15    }
16  },
17  "operationTime": "6891630671198945320"
18 }
```

4.5 DELETE-Methode

- Löschen eines Datensatzes über die ID

DELETE	▼	http://localhost:3000/users/5f972fb892d59013086eef36
--------	---	---