

Zainstaluj moduł: <https://www.npmjs.com/package/mongoose>

```
Mongoose> npm install mongoose
```

MongoDB występują nieco inne typy niż w Mongoose.

Typy w MongoDB: [https://www.tutorialspoint.com/mongodb/mongodb\\_datatype.htm](https://www.tutorialspoint.com/mongodb/mongodb_datatype.htm)

Typy w Mongoose: <https://mongoosejs.com/docs/schematypes.html>

Aby połączyć się z bazą wystarczy dołączyć moduł Mongoose i skorzystać z funkcji connect:

```
11
12 //mongoose
13 const mongoose = require('mongoose')
14 const url = 'mongodb://localhost:27017/'
15 const name = 'galleryDB'
16 mongoose.connect(url+name, {
17   useNewUrlParser: true,
18   useUnifiedTopology: true,
19   useFindAndModify: false,
20   useCreateIndex: true
21 });
22
```

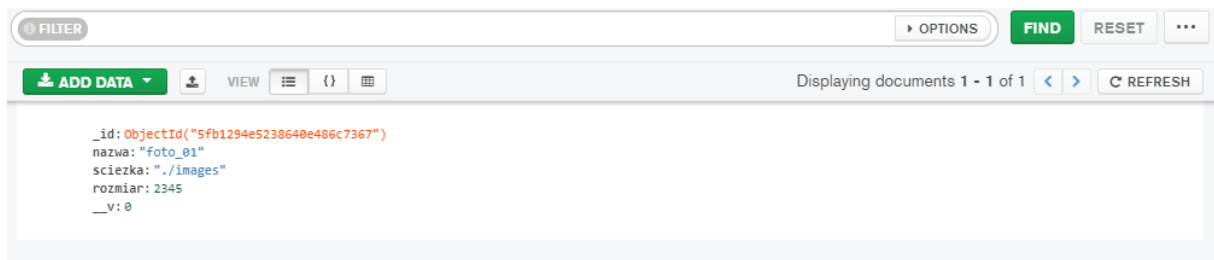
Łączenie wygląda nieco inaczej niż w poprzednim przykładzie. Aby dodać nowy obiekt do bazy stwórzmy model który będzie przechowywał podstawowe informacje o zdjęciu.

```
24 //wzór modelu
25 const Picture = mongoose.model('Picture',{
26   nazwa: String,
27   sciezka: String,
28   rozmiar: Number,
29 })
30
```

Teraz stwórzmy obiekt który dodamy do bazy danych.

```
24 //wzór modelu
25 const Picture = mongoose.model('Picture',{
26   nazwa: String,
27   sciezka: String,
28   rozmiar: Number,
29 })
30
31 //utworzenie obiektu zdjęcia i zapisanie do bazy danych
32 const picture = new Picture({nazwa: 'foto_01', sciezka: './images', rozmiar: '2345'})
33 picture.save()
34
35 app.listen(port)
```

Jak widać obiekt został dodany do bazy danych



Możemy skorzystać dodatkowo z funkcji `then()` dzięki której będziemy mogli otrzymać informację zwrotną po wykonaniu funkcji `save()`.

```
24 //wzór modelu
25 const Picture = mongoose.model('Picture',{
26   nazwa: String,
27   sciezka: String,
28   rozmiar: Number,
29 })
30
31 //utworzenie obiektu zdjęcia i zapisanie do bazy danych
32 const picture = new Picture({nazwa: 'foto_01', sciezka: './images', rozmiar: '2345'})
33 picture.save().then(() => console.log(picture)).catch(err => {console.log(err)})
34
35 app.listen(port)
```

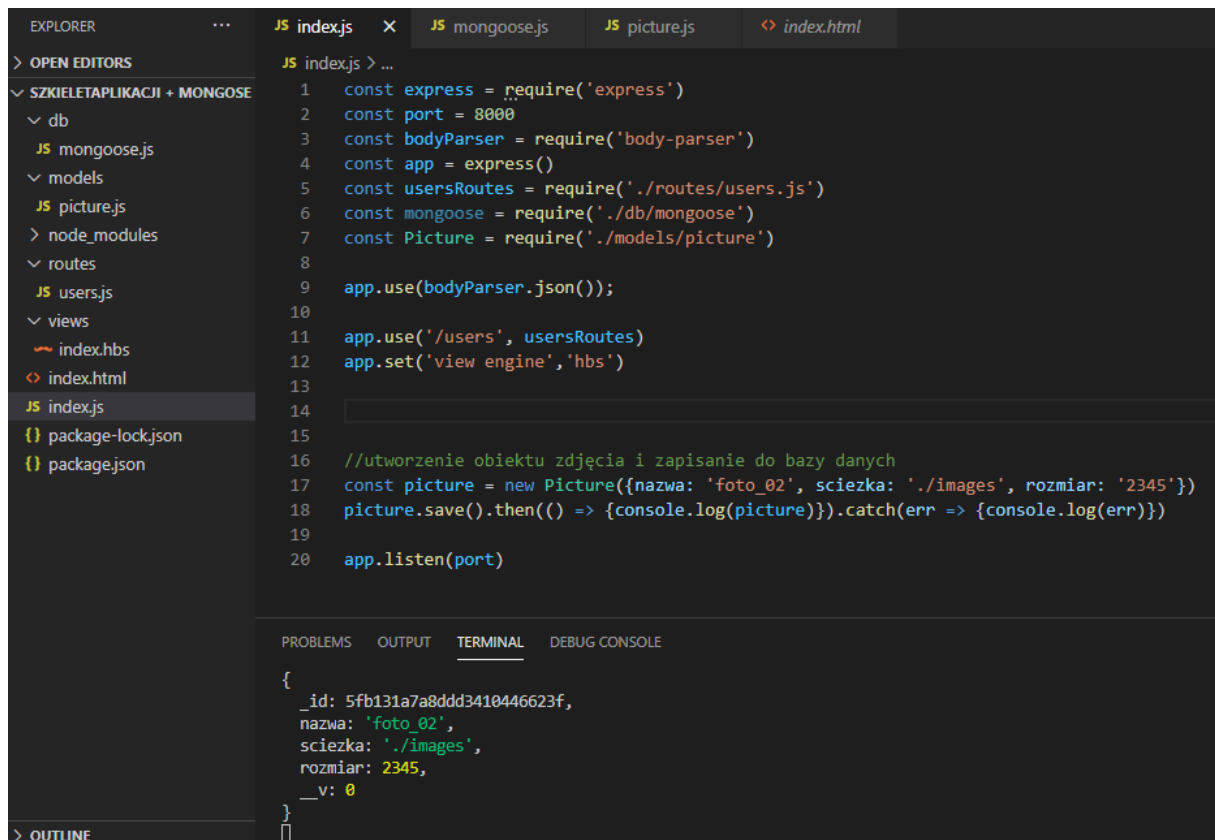
Below the code editor, the 'TERMINAL' tab is active, showing the output of the `picture.save()` operation:

```
{
  _id: 5fb12b5d46e6a50cdccd590,
  nazwa: 'foto_01',
  sciezka: './images',
  rozmiar: 2345,
  __v: 0
}
```

Do utworzonego obiektu kolekcji został dodany obiekt `_id` oraz pole `__v` w tym momencie nieistotne w projekcie. Możemy w tym momencie posprzątać nieco w plikach, zrobimy folder `db` a wewnątrz plik `mongoose.js` który będzie odpowiedzialny za łączenie z bazą, oraz folder `model` a w nim plik `picture.js` przechowujący model. Pliki te będą wyglądać następująco:

JS index.js	JS mongoose.js	JS picture.js
<pre>models &gt; JS picture.js &gt; [unknown] 1 const mongoose = require('mongoose') 2 3 const Picture = mongoose.model('Picture',{ 4   nazwa: String, 5   sciezka: String, 6   rozmiar: Number, 7 }) 8 9 module.exports = Picture</pre>	<pre>db &gt; JS mongoose.js &gt; ... 1 //mongoose 2 const mongoose = require('mongoose') 3 const url = 'mongodb://localhost:27017/' 4 const name = 'galleryDB' 5 mongoose.connect(url+name, { 6   useNewUrlParser: true, 7   useUnifiedTopology: true, 8   useFindAndModify: false, 9   useCreateIndex: true 10 });</pre>	

Musimy pamiętać że w przypadku pliku z modelem należy wyeksportować utworzony obiekt oraz dołączyć moduł 'mongoose'. W przypadku index.js należy dodać oba pliki do require aby były widoczne.



The screenshot shows the VS Code editor interface. On the left, the Explorer sidebar displays the project structure under 'SZKIELETAPLIKACJI + MONGOSE'. The main editor area shows the 'index.js' file with the following code:

```
1  const express = require('express')
2  const port = 8000
3  const bodyParser = require('body-parser')
4  const app = express()
5  const usersRoutes = require('./routes/users.js')
6  const mongoose = require('./db/mongoose')
7  const Picture = require('./models/picture')
8
9  app.use(bodyParser.json());
10
11 app.use('/users', usersRoutes)
12 app.set('view engine', 'hbs')
13
14
15
16 //utworzenie obiektu zdjęcia i zapisanie do bazy danych
17 const picture = new Picture({nazwa: 'foto_02', sciezka: './images', rozmiar: '2345'})
18 picture.save().then(() => {console.log(picture)}).catch(err => {console.log(err)})
19
20 app.listen(port)
```

Below the code editor, the TERMINAL panel shows the output of the application:

```
{
  _id: 5fb131a7a8ddd3410446623f,
  nazwa: 'foto_02',
  sciezka: './images',
  rozmiar: 2345,
  __v: 0
}
```

Spróbujmy teraz wypisać elementy przy użyciu modelu i funkcji find().

```
JS index.js x <> index.html JS mongoose.js

JS index.js > ...
1  const express = require('express')
2  const port = 8000
3  const bodyParser = require('body-parser')
4  const app = express()
5  const usersRoutes = require('./routes/users.js')
6  const mongoose = require('./db/mongoose')
7  const Picture = require('./models/picture')
8
9  app.use(bodyParser.json());
10
11 app.use('/users', usersRoutes)
12 app.set('view engine', 'hbs')
13
14
15
16 Picture.find({}).then((pictures)=>{
17   console.log(pictures)
18 })
19
20

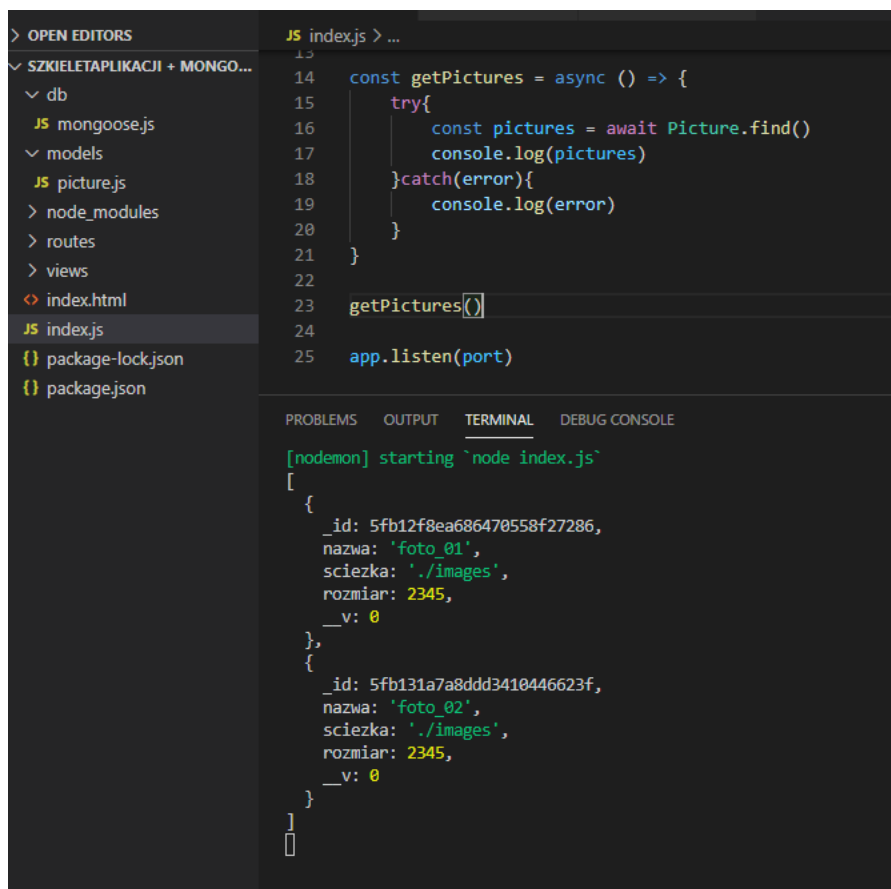
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
    at internal/main/run_main_module.js:17:47 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}
PS C:\Web Dev\szkieletAplikacji + Mongose await> node index.js
[
  {
    _id: 5fb12f8ea686470558f27286,
    nazwa: 'foto_01',
    sciezka: './images',
    rozmiar: 2345,
    __v: 0
  },
  {
    _id: 5fb131a7a8ddd3410446623f,
    nazwa: 'foto_02',
    sciezka: './images',
    rozmiar: 2345,
    __v: 0
  }
]
```

Żeby dodatkowo wyłapać błędy możemy skorzystać z catch()

```
17
18 const getPictures = Picture.find().then(() => {console.log(getPictures)}).catch(err => {console.log(err)})
19
20 app.listen(port)

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell + [ ] ^
Promise { <pending> }
```

Możemy również skorzystać z asynchroniczności, zatem poprzedni kod można zastąpić następującym.

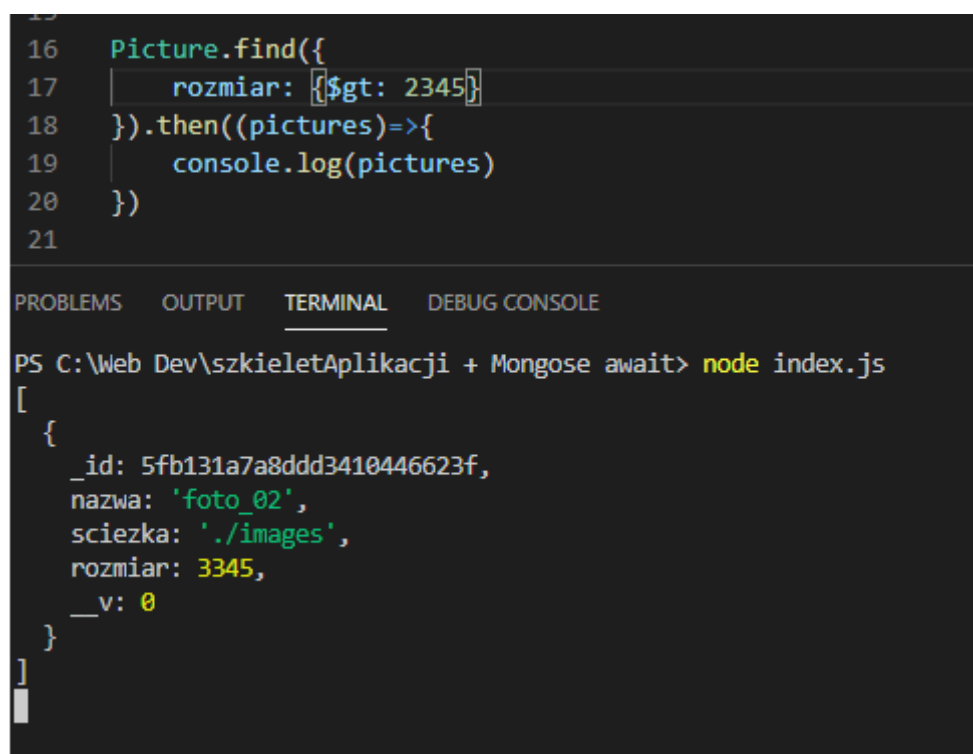


```
13
14 const getPictures = async () => {
15   try{
16     const pictures = await Picture.find()
17     console.log(pictures)
18   }catch(error){
19     console.log(error)
20   }
21 }
22
23 getPictures()
24
25 app.listen(port)
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
[nodemon] starting `node index.js`
[
  {
    _id: 5fb12f8ea686470558f27286,
    nazwa: 'foto_01',
    sciezka: './images',
    rozmiar: 2345,
    __v: 0
  },
  {
    _id: 5fb131a7a8ddd3410446623f,
    nazwa: 'foto_02',
    sciezka: './images',
    rozmiar: 2345,
    __v: 0
  }
]
```

Do wyszukiwania można również dodać warunki, zmienię rozmiar pliku 'foto\_02' na rozmiar 3345, a następnie poszukam pliki większe od 2345.



```
16 Picture.find({
17   rozmiar: { $gt: 2345 }
18 }).then((pictures)=>{
19   console.log(pictures)
20 })
21
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
PS C:\Web Dev\szkieletAplikacji + Mongose await> node index.js
[
  {
    _id: 5fb131a7a8ddd3410446623f,
    nazwa: 'foto_02',
    sciezka: './images',
    rozmiar: 3345,
    __v: 0
  }
]
```

Do aktualizacji możemy wykorzystać funkcję `updateOne`. Dla przykładu:

The screenshot shows a development environment with a code editor on the left and MongoDB Compass on the right. The code editor displays the following JavaScript code in `index.js`:

```
JS index.js > $set > rozmiar
6 const mongoose = require('../db/mongoose')
7 const Picture = require('../models/picture')
8
9 app.use(bodyParser.json());
10
11 app.use('/users', usersRoutes)
12 app.set('view engine', 'hbs')
13
14
15
16 Picture.updateOne(
17   {nazwa: 'foto_02'},
18   {$set: {
19     rozmiar: 5345
20   }}
21 ).then((pictures)=>{
22   console.log(pictures)
23 })
24
25
26 /*
27
```

The terminal output shows the result of the update operation:

```
PS C:\Web Dev\szkieletAplikacji + Mongose awai
[
  {
    _id: 5fb131a7a8ddd3410446623f,
    nazwa: 'foto_02',
    sciezka: './images',
    rozmiar: 3345,
    __v: 0
  }
]
PS C:\Web Dev\szkieletAplikacji + Mongose awai
{ n: 1, nModified: 1, ok: 1 }
```

MongoDB Compass on the right shows the `galleryDB` database with the `pictures` collection. The document being updated is visible in the `Documents` tab:

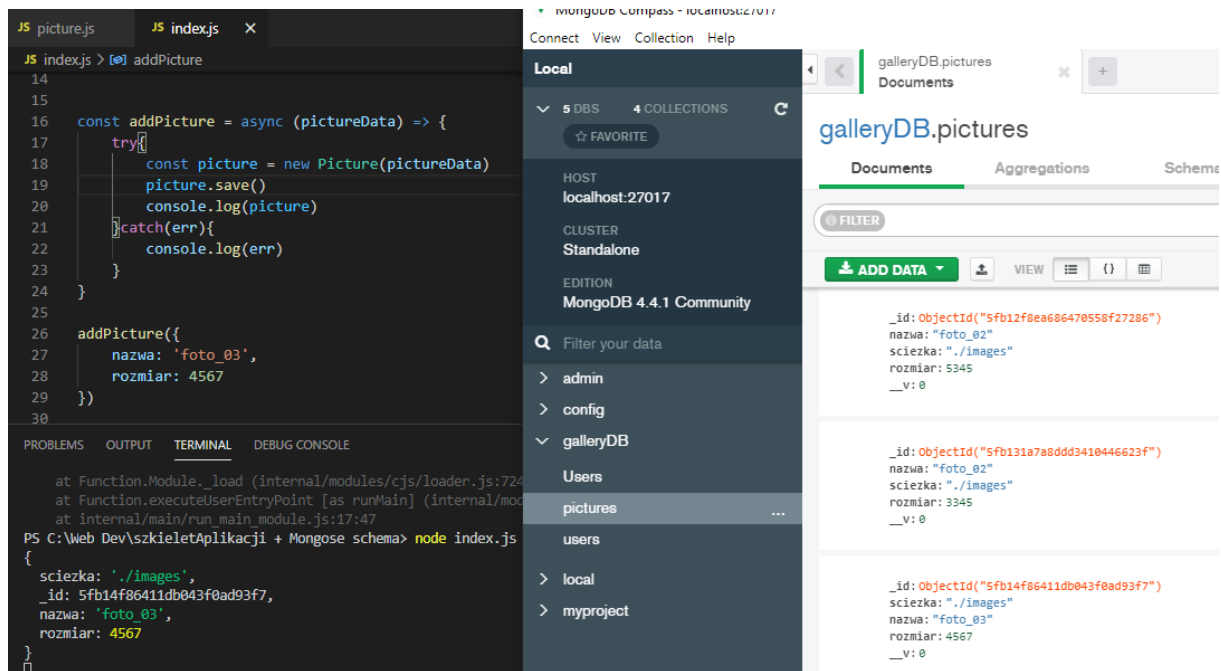
```
{
  "_id": "5fb12f8ea686470558f27286",
  "nazwa": "foto_02",
  "sciezka": "./images",
  "rozmiar": 5345,
  "__v": 0
}
```

Kolejnym udogodnieniem stosowania mongoose jest obiekt `Schema`. Dzięki niemu możemy narzucić dodatkowe wymagania takie jak ograniczenia czy domyślne wartości. Aby skorzystać ze `Schema` musimy wrócić do modelu i trochę go zmodyfikować.

The screenshot shows a code editor with the following JavaScript code in `picture.js`:

```
JS picture.js > JS index.js > JS mongoose.js
models > JS picture.js > ...
1 const mongoose = require('mongoose')
2
3 // korzystając z obiektu Schema udostępnionym przez mongoose, można dodać dodatkowe parametry
4 // takie jak wymagane pola czy różne ograniczenia i elementy domyślne
5 const PictureSchema = new mongoose.Schema({
6   nazwa: {
7     type: String,
8     required: true,
9   },
10  sciezka: {
11    type: String,
12    default: './images'
13  },
14  rozmiar: {
15    type: Number,
16    max: 10000
17  }
18 })
19
20 //dopiero tutaj tworzony jest model
21 const Picture = mongoose.model('Picture', PictureSchema)
22
23 module.exports = Picture
```

Po stworzeniu modelu możemy przetestować działanie, na przykład nie wpisujemy ścieżki. Aby przetestować musimy napisać funkcję która przyjmuje model a następnie zapisuje, jednocześnie zamykając w bloki try-catch możemy wyłapać ewentualne błędy. Po wywołaniu funkcji otrzymujemy wynik, w którym widać że ścieżka została dodana domyślnie.



## Zadanie.

Utwórz kolekcje zgodnie ze stworzonym API, a następnie stwórz modele do wszystkich kolekcji z bazy danych i przetestuj działanie na przykładowych danych.