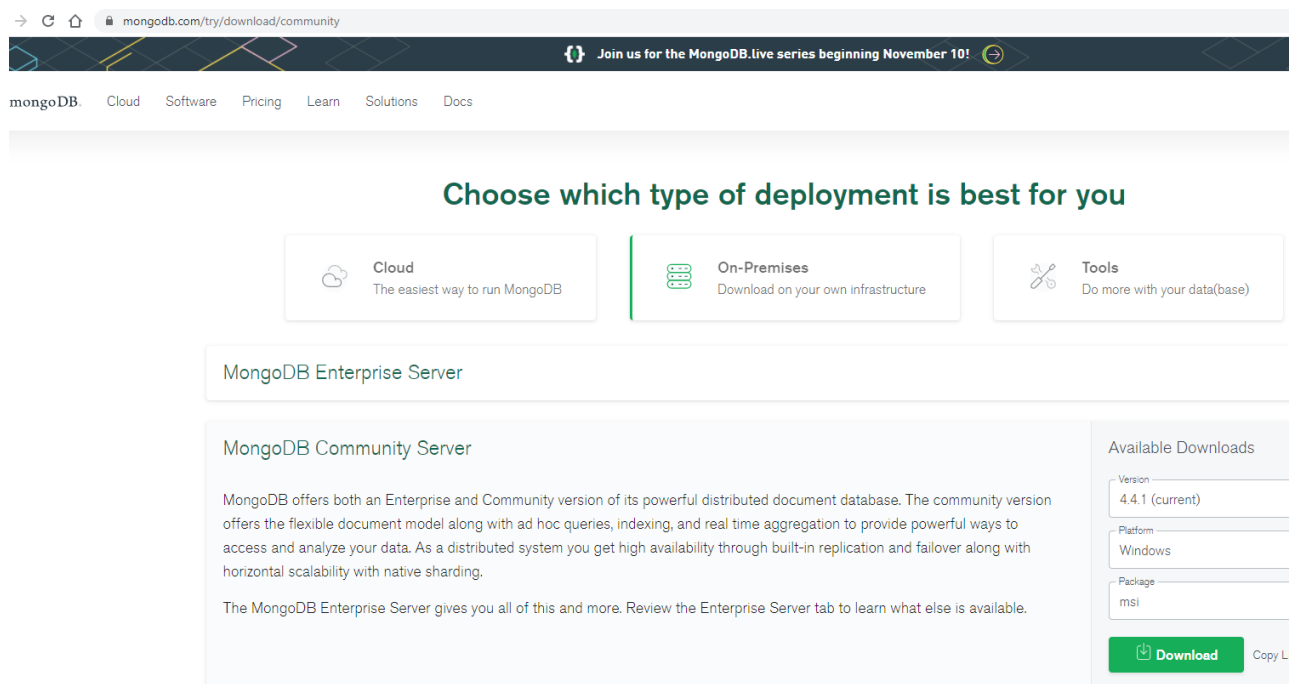


Baza danych

Do przechowywania i operowania na danych można wykorzystać dwa najbardziej znane podejścia, czyli relacyjną bazę danych np MySQL, albo nierelacyjną bazę danych taką jak na przykład mongoDB. Nierelacyjne bazy danych określane są pojęciem No-Sql. Główną różnicą między tymi bazami jest to że nie posiadają relacji. Oczywiście można te relacje zasymulować, ale domyślnie nie są odpowiedzialne za poprawne działanie bazy danych. Bazy typu No-Sql nie posiadają konkretnego wzoru struktury, w przeciwieństwie do MySQL, gdzie każde pole powinno być opisane i ma swoje dodatkowe warunki. Zastosowanie no-sql pozwala na swobodne poruszanie się po bazie danych, nie narzuca ustalania konkretnych pól, zatem w każdej kolekcji (bo tak nazywa się Tabela w przypadku No-Sql) mogą wystąpić różne pola. Zaletą stosowania MySQL jest to że w każdym momencie mamy schemat jednakowy dla wszystkich tabel w bazie, klarowne połączenia między nimi, dzięki temu wszystko jest bardzo dobrze uporządkowane. No-Sql przydatny jest zatem na początku tworzenia aplikacji, gdzie nie jest wszystko dokładnie zaplanowane i możemy rozwijać bazę danych o nowe pola i kolekcje, nie psując działania w trakcie modyfikacji kolekcji. Drugim plusem no-sql jest to że może być dzielony na różne serwery, więc No-Sql ma zastosowanie w przypadku ogromnych baz danych.

Przejdźmy do stworzenia nierelacyjnej bazy danych MongoDB. Aby utworzyć taką bazę musimy pobrać MongoDB, do naszej aplikacji wykorzystajmy MongoDB Community Server <https://www.mongodb.com/try/download/community>





→ ↻ ⌂ [mongodb.com/try/download/community](https://www.mongodb.com/try/download/community)


Join us for the MongoDB Live series beginning November 10!

mongoDB. Cloud Software Pricing Learn Solutions Docs

Choose which type of deployment is best for you

 **Cloud**
The easiest way to run MongoDB

 **On-Premises**
Download on your own infrastructure

 **Tools**
Do more with your data(base)

MongoDB Enterprise Server

MongoDB Community Server

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data. As a distributed system you get high availability through built-in replication and failover along with horizontal scalability with native sharding.


The MongoDB Enterprise Server gives you all of this and more. Review the Enterprise Server tab to learn what else is available.

Available Downloads

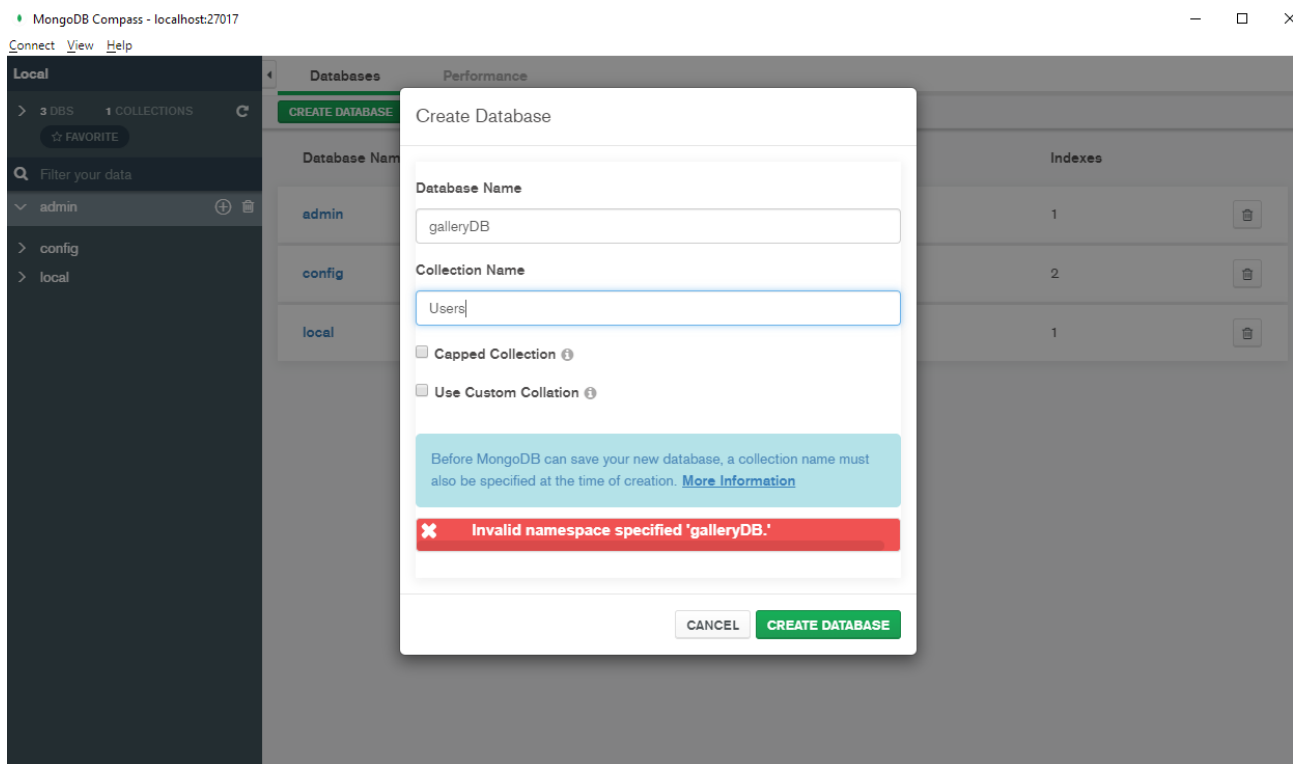
Version
4.4.1 (current)

Platform
Windows

Package
msi

 **Download** [Copy Link](#)

Po zainstalowaniu MongoDB dostajemy przydatne narzędzie jakim jest MongoDB Compass dzięki któremu możemy oglądać, tworzyć i modyfikować naszą bazę. Zaczniemy od stworzenia bazy danych. Podczas tworzenia musimy również utworzyć pierwszą kolekcję.



Kiedy MongoDB jest zainstalowane potrzebujemy moduł do npm który będzie obsługiwał tę bazę. Aby dodać skorzystajmy z <https://www.npmjs.com/package/mongodb> . Instrukcja instalacji zawarta jest na tej stronie, oraz podane jest przykładowe połączenie z bazą.

[npmjs.com/package/mongodb](https://www.npmjs.com/package/mongodb)

Create a new **app.js** file and add the following code to try out some basic CRUD operations using the MongoDB driver.

Add code to connect to the server and the database **myproject**:

```
const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');

// Connection URL
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'myproject';

// Use connect method to connect to the server
MongoClient.connect(url, function(err, client) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  const db = client.db(dbName);

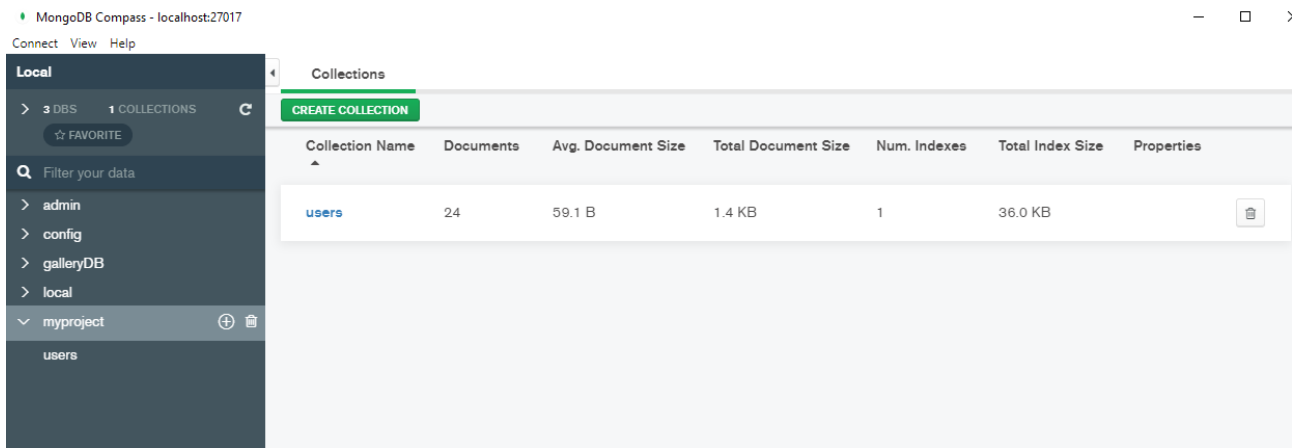
  client.close();
});
```

W powyższym przykładzie na początku musimy załadować moduł mongodb oraz assert. Url jest naszym adresem pod którym będzie się znajdować baza danych, niech to będzie dokładnie ten sam port, jest domyślny dla MongoDB. DbName to nazwa po której będziemy łączyć się do bazy danych. Jeżeli nie zmienimy ustawień zostanie utworzona baza danych o nazwie myproject. Bazując na przykładzie z poprzednich zajęć aby uruchomić należy wydać polecenie npm start. Napiszmy fragment który doda nam użytkownika.

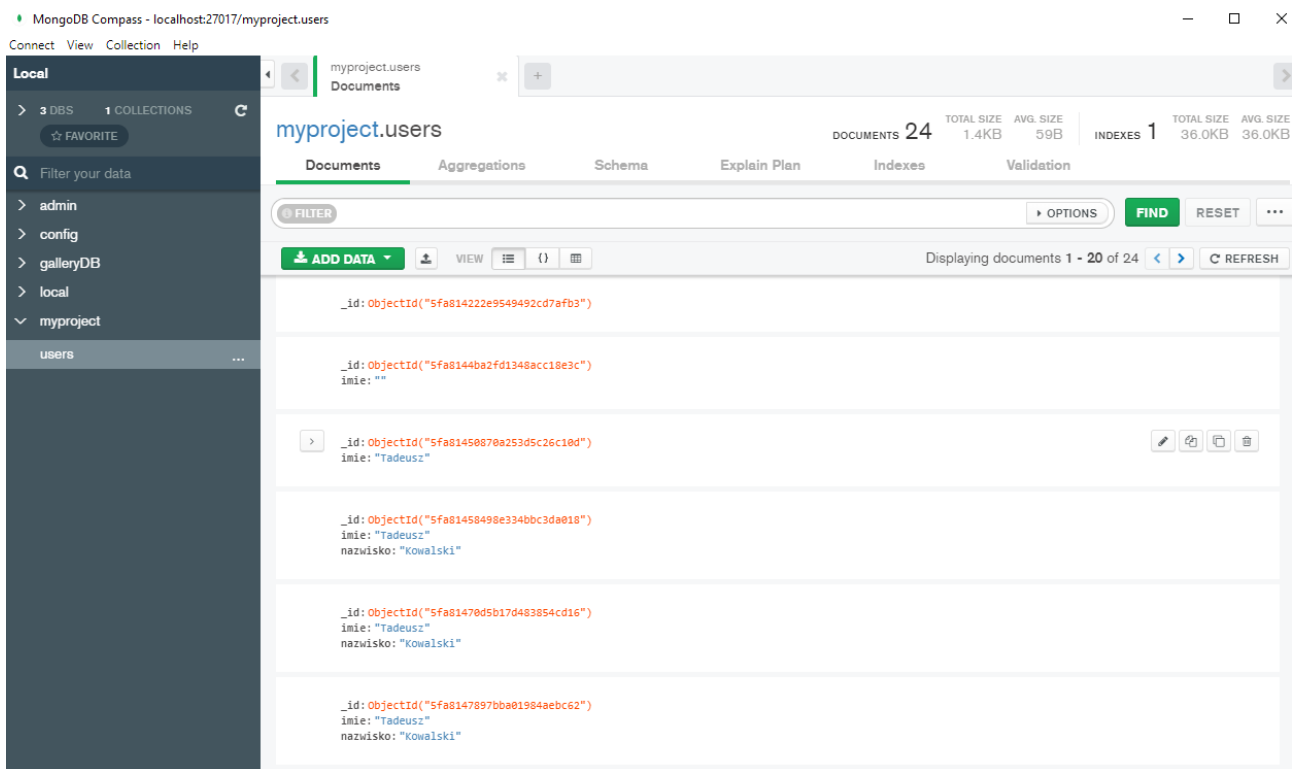
```
8   const assert = require('assert');
9
10  // Connection URL
11  const url = 'mongodb://localhost:27017';
12
13  // Database Name
14  const dbName = 'myproject';
15
16  // Use connect method to connect to the server
17  MongoClient.connect(url, function(err, client) {
18    assert.equal(null, err);
19    console.log("Connected successfully to server");
20
21    const db = client.db(dbName);
22
23    db.collection('users').insertOne({
24      imie: "Tadeusz",
25      nazwisko: "Kowalski"
26    }, (error, result) => {
27      if(error)
28        console.log("coś nie tak")
29      else
30        console.log("działa poprawnie")
31    })
32
33    client.close();
34  });
35
```

W tym przykładzie dodajemy użytkownika o imieniu Tadeusz i nazwisku Kowalski. Funkcja insertOne jak też inne funkcje oferują nam możliwość otrzymania zwrotu, czyli (error, result) gdzie możemy zobaczyć czy użytkownik został dodany poprawnie. Konstrukcja jest uniwersalna dla różnych funkcji. Jak widać dane przypominają json.

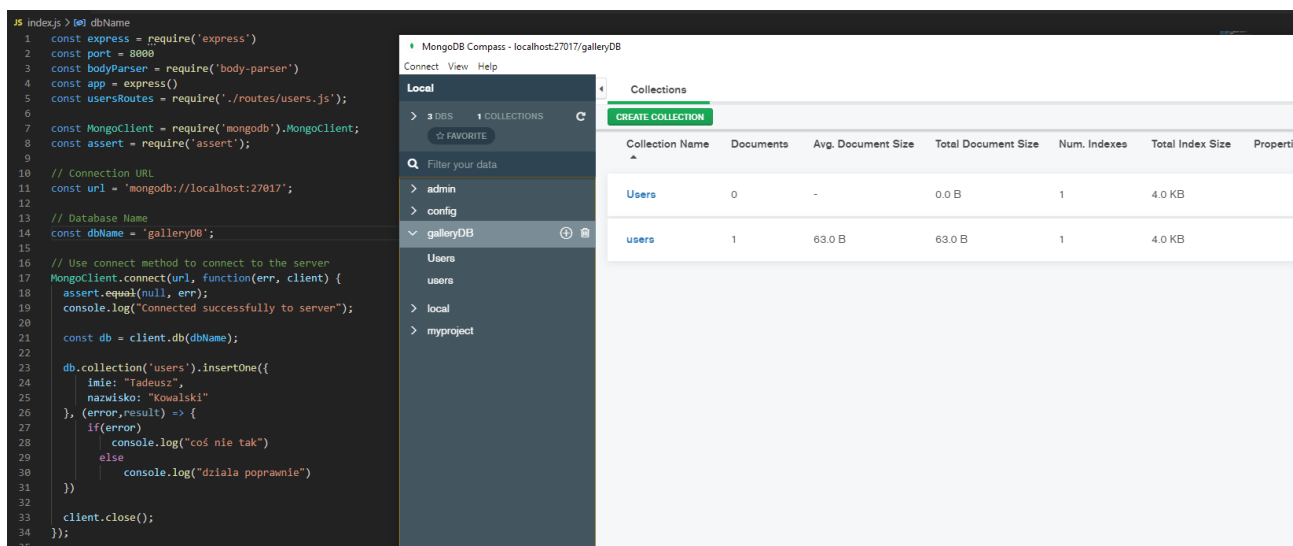
Po uruchomieniu tego skryptu możemy zobaczyć co zmieniło się w bazie danych przy pomocy MongoDBCompass.



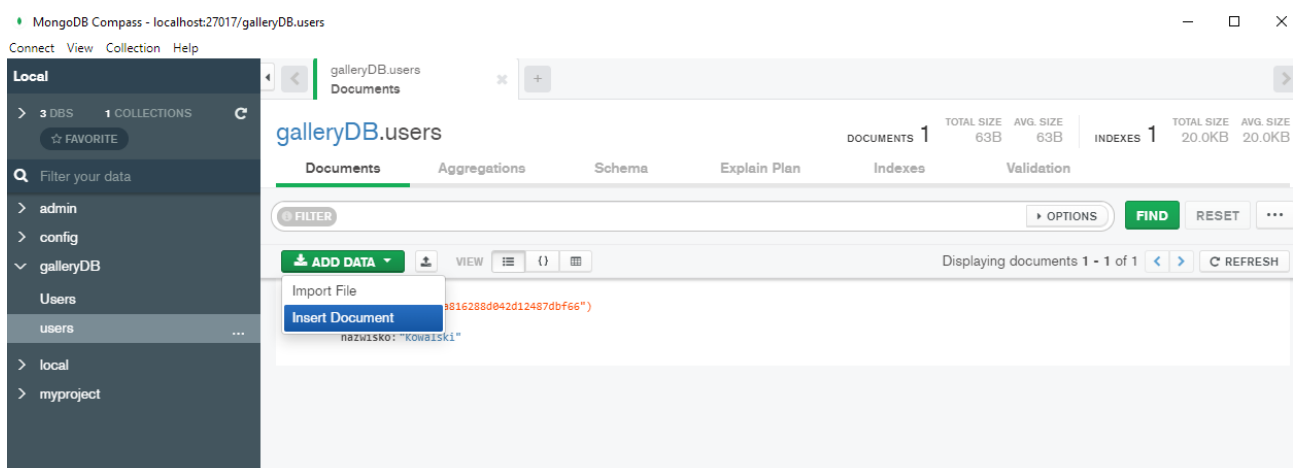
Jak widać została utworzona kolekcja users. Jednak otwarcie kolekcji może w niektórych przypadkach przerazić, ponieważ możemy zobaczyć wielu użytkowników. Dzieje się tak ponieważ na ostatnich zajęciach skorzystaliśmy z nodemon który na bieżąco aktualizuje, oraz funkcję automatycznego zapisu. Aby tego uniknąć można skorzystać z pierwotnego polecenia `node index.js` lub wyłączyć automatyczny zapis. Poniżej przykład tego problemu.



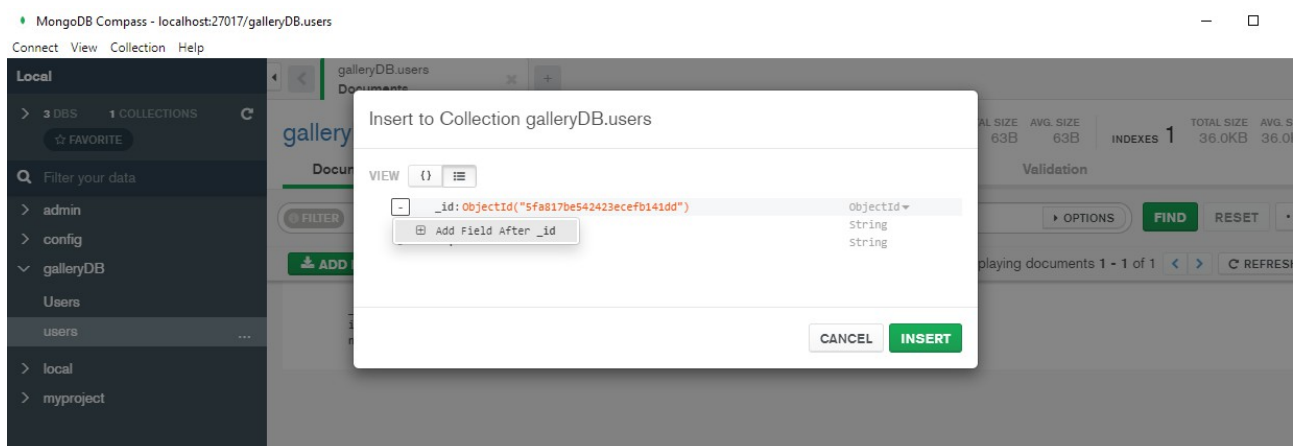
Nie chcemy jednak bazować na przykładzie, tylko aktualizować wcześniej stworzoną bazę danych "GalleryDB", zatem pozmieniamy w kodzie i dodajmy użytkownika Tadeusz Kowalski do naszej kolekcji użytkowników



Jak widać mamy teraz dwie kolekcje, problemem który tutaj wyniknął była zła nazwa kolekcji, pierwsza wielka litera z przodu nazwy spowodowała że została dodana nowa kolekcja. Aplikacja MongoDBCompass umożliwia usunięcie takiej pomyłki. Aplikacja ta posiada również możliwość dodania elementów kolekcji



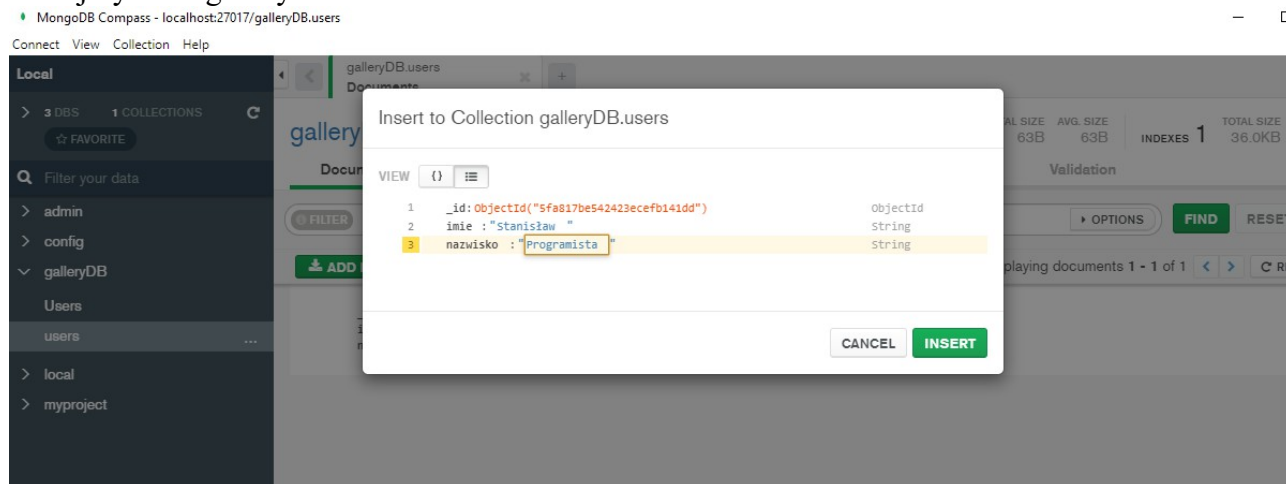
Możemy dodać kolejną pozycję na różne sposoby. W tym momencie można przetestować jakie funkcje są udostępnione.



Można zauważyć że każdy obiekt ma jakieś id, ale nie jest to tekst tylko obiekt, aby zrozumieć na czym to polega skorzystaj z dokumentacji

<https://docs.mongodb.com/manual/reference/method/ObjectId/>

Dodajmy nowego użytkownika



Aby znaleźć dodanego użytkownika z poziomu naszego skryptu wystarczy skorzystać z metody find dla kolekcji 'users'.

```
34 db.collection('users').find({
35   imie: 'Stanisław'
36 }).toArray((err,res) =>{
37   console.log(res)
38 })
39
40 /*
```

```
[
  {
    _id: 5face9058fed73c351fa0744,
    imie: 'Stanisław',
    nazwisko: 'Programista'
  }
]
```

W tym przypadku użytkownik składa się jedynie z pola imie i nazwisko, jednak jeśli chcemy skorzystać z pól liczbowych wówczas możemy skorzystać z operatorów porównania

<https://docs.mongodb.com/manual/reference/operator/query-comparison/>

W naszym przypadku korzystając z operatora porównania możemy znaleźć użytkownika.

```
41 db.collection('users').find({
42   "imie": { $eq: "Stanisław" }
43 }).toArray((err,res) =>{
44   console.log(res)
45 })
46 /*
47
48 db.collection('users').updateOne({
```

```
[
  {
    _id: 5face9058fed73c351fa0744,
    imie: 'Stanisław',
    nazwisko: 'Programista'
  }
]
```

Do dyspozycji mamy również aktualizację, dlatego sprawdźmy funkcję updateOne.

```
48 db.collection('users').updateOne({
49   imie: "Tadeusz"
50 }, {
51   $set:{
52     imie: "Tymoteusz"
53   }
54 })
```

Funkcja ta spowoduje zmianę imienia użytkownika Tadeusz na Tymoteusz. Możemy również usunąć użytkownika.

```
56 db.collection('users').deleteOne({
57   imie: "Stanisław"
58 }, (err, res) => {
59   if(err) console.log("błąd");
60   else console.log(res)
61 })
```

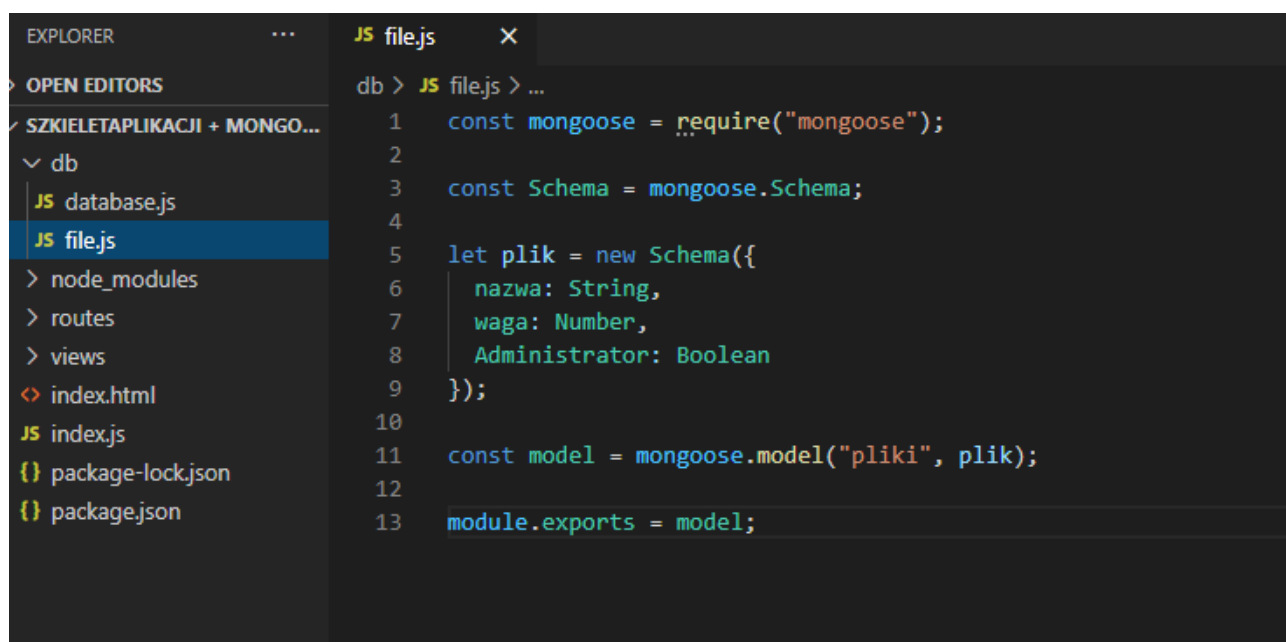
Zadanie.

Przetestuj działanie różnych funkcji operujących na bazie MongoDB

(<http://mongodb.github.io/node-mongodb-native/3.1/api/>) , sprawdź możliwości

MongoDB Compass i stwórz przykładowe dane bazując na schemacie z poprzednich zajęć.

Zapoznaj się z pakietem mongoose (<https://www.npmjs.com/package/mongoose>) i zacznij tworzyć modele na podstawie dokumentacji (<https://mongoosejs.com/docs/guide.html>). Przykład modelu:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with folders like 'node_modules', 'routes', and 'views', and files like 'index.html', 'index.js', 'package-lock.json', and 'package.json'. The 'db' folder is expanded, showing 'database.js' and 'file.js'. The 'file.js' file is selected and its content is shown in the main editor. The code in 'file.js' defines a Mongoose schema for a model named 'pliki'. It includes fields for 'nazwa' (String), 'waga' (Number), and 'Administrator' (Boolean). The schema is instantiated as 'plik' and then used to create a Mongoose model 'model'. Finally, the model is exported as 'module.exports = model;'.

```
1  const mongoose = require("mongoose");
2
3  const Schema = mongoose.Schema;
4
5  let plik = new Schema({
6    nazwa: String,
7    waga: Number,
8    Administrator: Boolean
9  });
10
11  const model = mongoose.model("pliki", plik);
12
13  module.exports = model;
```