

Schemat aplikacji (źródło: [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/routes](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes))

Database – mongoDB, do zarządzania MongoDBCompass, do obsługi mongoose

Models – modele, schematy, czyli to wszystko jak wygląda dany element kolekcji

Routes – do przekazywania żądań i informacji pochodzących z adresu url

Controllers – łączy modele się z modelami, generuje widoki i w tym miejscu wykonywane są wszystkie obróbki danych.

Views – widoki, czyli szablony do wyświetlania danych, warstwa interfejsu użytkownika, może już zawierać wszystkie graficzne technologie np. html i css

## Porządki zgodnie ze schematem

Wszystkie operacje na bazie danych będzie trzeba przenieść z pliku index.js. Zatem na początku zgodnie z tutorialiem ([https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/routes](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)) zakomentujmy dotychczasowe operacje i ustalmy routers dla Picture.

Od razu możemy stworzyć:

1. folder controllers, a w środku pictureController.js,
2. folder views
3. plik w folderze models picture.js
4. plik w folderze routes pictures.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 picturesRoutes = require('./routes/pictures.js') //musimy dodać połączenie z plikiem pictures.js
7  const mongoose = require('mongoose')
8  const Picture = require('./models/picture')
9
10 app.use(bodyParser.json());
11
12 app.use('/users', usersRoutes)
13 app.use('/pictures', picturesRoutes) // dodajemy ścieżkę do naszej aplikacji
14 app.set('view engine', 'hbs')
15
16 /*
17 const getPictures = async () => {
18   try{
19     const pictures = await Picture.find()
20     //console.log(pictures)
21   }catch(err){
22     console.log(err)
23   }
24 }
```

Należy zatem dodać linijkę 6 i 13. Po lewej stronie widać od razu układ plików. Dzięki dodaniu ścieżki mamy teraz możliwość przerzucić poszczególne operacje do innych plików robiąc aplikację bardziej czytelną.

Zajmijmy się plikiem routes/pictures.js – w tym pliku będą ścieżki po których będziemy odnosić się do funkcji kontrolera.

```
JS index.js JS pictures.js X JS pictureController.js
routes > JS pictures.js > ...
1  const express = require('express')
2  const router = express.Router();
3
4  //musimy dodać kontroler aby był widoczny dla naszych ścieżek
5  var picture_controller = require('../controllers/pictureController');
6
7  //w tym miejscu zamiast definiować operacje, ustalamy jedynie ścieżki
8  //przypisujemy potrzebne elementy do budowy CRUD
9
10 router.get('/', picture_controller.index);
11 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/
12
13 router.get('/picturesList', picture_controller.picture_list);
14 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/picturesList
15
16
17 console.log("router gotowy")
18 module.exports = router ;
19 |
```

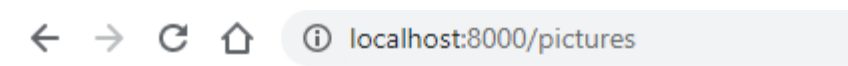
Kolejnym krokiem będzie zdefiniować kontroler dla tych funkcji, dla przykładu niech to będzie funkcja obsługująca router.get('/') dla domyślnej strony dla ścieżki pictures, oraz router.get('picturesList') do obsługi wylistowania wszystkich elementów kolekcji zdjęć. W komentarzu podane dokładne ścieżki po których możemy je uruchomić.

```
JS index.js JS pictures.js JS pictureController.js X
controllers > JS pictureController.js > ...
1  var Picture = require('../models/picture');
2
3  // zdefiniowanie pustych funkcji
4  exports.index = function(req, res) {
5      console.log("wywołanie get index")
6      res.send('NOT IMPLEMENTED: Site Home Page');
7  };
8
9  exports.picture_list = function(req, res) {
10     console.log("wywołanie get pictures")
11     res.send('NOT IMPLEMENTED: Picture list');
12 };
13
14 console.log("kontroler gotowy")
```

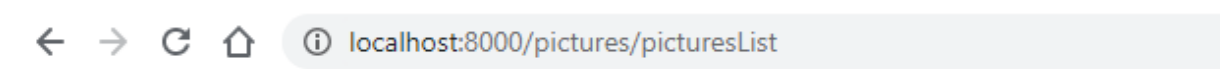
Wygląda że wszystko jest poprawnie przeniesione, dodatkowo dołączyłem w każdym pliku odpowiednie komunikaty, aby zobaczyć czy wszystkie połączenia są prawidłowe i aplikacja je widzi.

```
PS C:\Web Dev\szkieletAplikacji + controllers> node index.js
model gotowy
kontroler gotowy
router gotowy
baza danych uruchomiona
```

Możemy również sprawdzić działanie przy pomocy ścieżki:



NOT IMPLEMENTED: Site Home Page



NOT IMPLEMENTED: Picture list

Teraz możemy uzupełnić kontroler aby pobierał dane z bazy danych. Możemy zatem przenieść utworzone wcześniej operacje na bazie danych do konkretnych funkcji w kontrolerze. W przykładzie poniżej będzie wypisanie elementów kolekcji zdjęć.

```
1 var Picture = require('../models/picture');
2
3 // zdefiniowanie pustych funkcji
4 exports.index = function(req, res) {
5   console.log("wywołanie get index")
6   res.send('NOT IMPLEMENTED: Site Home Page');
7 };
8
9 exports.picture_list = function(req, res) {
10
11   let picturesList
12
13   const getPictures = async () => {
14     try{
15       picturesList = await Picture.find()
16       console.log(picturesList)
17       res.send(picturesList);
18     }catch(err){
19       console.log(err)
20     }
21   }
22
23   getPictures()
24
25 };
26
27 console.log("kontroler gotowy")
```

localhost:8000/pictures/picturesList

```
[
  {
    "sciezka": "./images",
    "_id": "5fb12f8ea686470558f27286",
    "nazwa": "foto_02",
    "rozmiar": 5345,
    "__v": 0
  },
  {
    "sciezka": "./images",
    "_id": "5fb131a7a8ddd3410446623f",
    "nazwa": "foto_02",
    "rozmiar": 3345,
    "__v": 0
  },
  {
    "sciezka": "./images",
    "_id": "5fb14f86411db043f0ad93f7",
    "nazwa": "foto_03",
    "rozmiar": 4567,
    "__v": 0
  },
  {
    "sciezka": "./images",
    "_id": "5fb8f4ec9e14de0340b90047",
    "nazwa": "foto_03",
    "rozmiar": 4567,
    "__v": 0
  }
]
```

Jeśli chodzi o metodę Post, możemy stworzyć z tymczasowymi danymi na sztywno i dodać nowe zdjęcie. Zaczniemy od ścieżki.

```
EXPLOER  ...  JS index.js  JS pictures.js X  JS pictureController.js  JS picture.js
> OPEN EDITORS
SZKIELETAPLIKACJI + CONTRO...
  controllers
    JS pictureController.js
  db
    JS mongoose.js
  models
    JS picture.js
  > node_modules
  > routes
    JS pictures.js
    JS users.js
  > views
    index.html
  JS index.js
  {} package-lock.json
  {} package.json

routes > JS pictures.js > ...
1 const express = require('express')
2 const router = express.Router();
3
4 //musimy dodać kontroler aby był widoczny dla naszych ścieżek
5 var picture_controller = require('../controllers/pictureController');
6
7 //w tym miejscu zamiast definiować operacje, ustalamy jedynie ścieżki
8 //przypisujemy potrzebne elementy do budowy CRUD
9
10 router.get('/', picture_controller.index);
11 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/
12
13 router.get('/picturesList', picture_controller.picture_list);
14 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/picturesList
15
16 router.post('/pictureinstance/create', picture_controller.picture_create_post );
17 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/pictureinstance/create
18
19
20 console.log("router gotowy")
21 module.exports = router ;
22
```

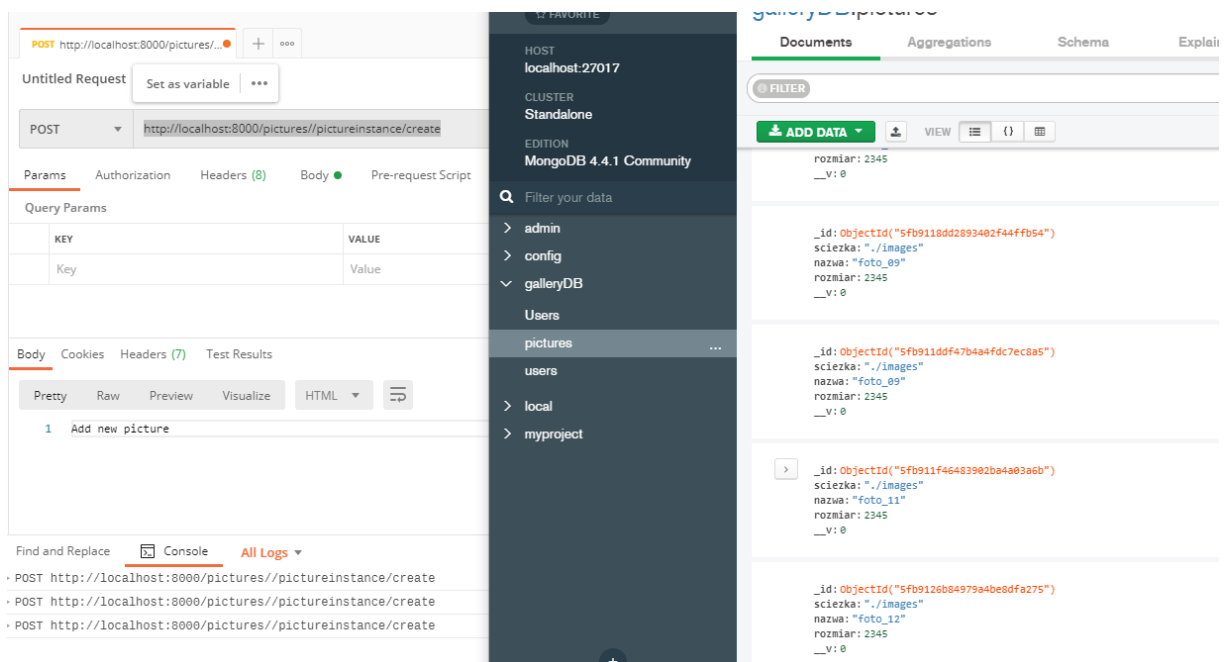
A następnie w kontrolerze musimy mieć możliwość dodać użytkownika, nie mamy jeszcze żadnych formularzy w widoku dlatego zrobmy na sztywno.

```

27
28 exports.picture_create_post = function(req, res) {
29
30   const picture = new Picture({nazwa: 'foto_12', sciezka: './images', rozmiar: '2345'})
31   picture.save().then(() => {
32     console.log(picture)
33     res.send('Add new picture');
34   }).catch(err => {
35     console.log(err)
36   })
37
38 };
39
40 console.log("kontroler gotowy")

```

Teraz aby przetestować musimy użyć Postmana i dla podanego url wybrać metodę post i wysłać.



Po odświeżeniu w Postmanie możemy zobaczyć że pozycja została dodana.

## Zadanie.

Na podstawie tego tutorialu utwórz wszystkie potrzebne kontrolery i przetestuj działanie. Wszystkie potrzebne funkcje możemy zaprojektować na podstawie API.