

# Node.js + Widoki

Aby wyświetlić treść w przeglądarce dostępne są silniki widoków, na przykład Pug, Mustache, Handlebars, czy EJS. W tym tutorialu przedstawię wyświetlanie na podstawie Handlebars. Aby zacząć korzystać należy zainstalować moduł hbs (<https://www.npmjs.com/package/hbs>).

Do tej pory do przekazywania danych do widoku korzystaliśmy z metody send.

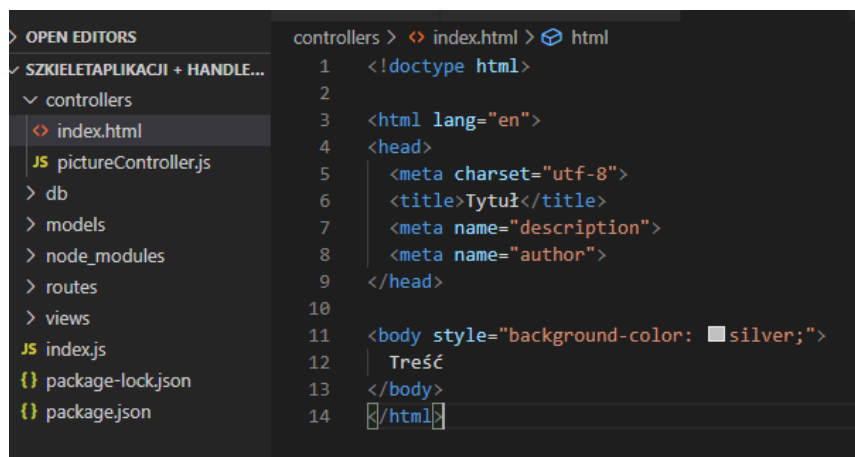
```
EXPLORER
... JS index.js JS pictureController.js X
> OPEN EDITORS
SZKIELETAPLIKACJI + HANDLE...
  controllers
    JS pictureController.js
  db
  models
  node_modules
  routes
  views
  index.html
  JS index.js
  package-lock.json
  package.json

controllers > JS pictureController.js > picture_list > picture_list > getPictures
// wywołanie get index
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 };
```

Metoda send powodowała przekazanie elementu w postaci json do widoku i nie było możliwe kreowanie wyglądu danych.

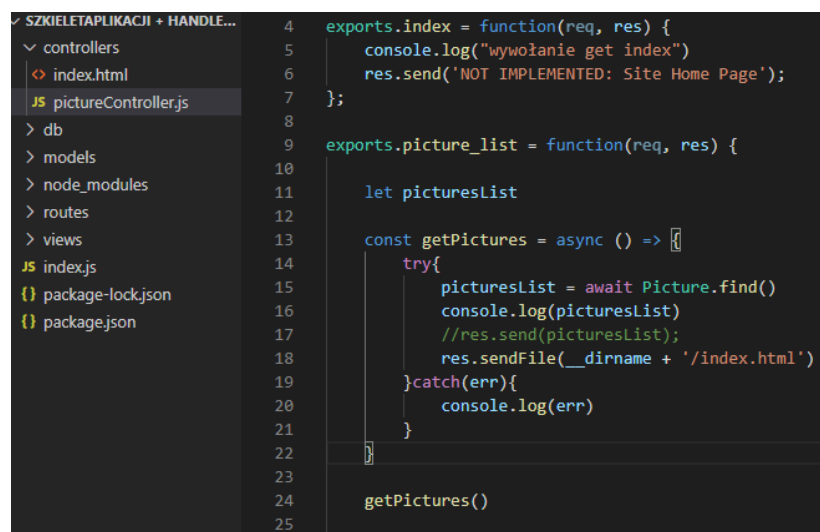


Możemy również wysłać plik metodą `sendFile`. Aby przetestować możemy wysłać prosty szablon html



```
OPEN EDITORS
SZKIELETAPLIKACJI + HANDLE...
  controllers
    index.html
    pictureController.js
  db
  models
  node_modules
  routes
  views
  index.js
  package-lock.json
  package.json

controllers > index.html > html
1 <!doctype html>
2
3 <html lang="en">
4 <head>
5   <meta charset="utf-8">
6   <title>Tytuł</title>
7   <meta name="description">
8   <meta name="author">
9 </head>
10
11 <body style="background-color: silver;">
12   Treść
13 </body>
14 </html>
```



```
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       res.sendFile(__dirname + '/index.html')
19     }catch(err){
20       console.log(err)
21     }
22   }
23
24   getPictures()
25 }
```

jednak aby można było opakować zawartość w html potrzebujemy wykorzystać silnik hbs. Aby stworzyć serwis posiadający layout i treść wewnątrz layoutu możemy doinstalować pomocną paczkę <https://www.npmjs.com/package/express-handlebars>. Zaczniemy od ustawień ścieżek. Potrzebujemy utworzyć folder `views/layouts` który będzie zawierał plik `layout.hbs` oraz w folderze `view` `index.hbs` do którego testowo możemy przekazać kilka wartości. Zasada działania będzie następująca:

*renderowanie strony index w pliku kontrolera -> automatyczne wygenerowanie layoutu i wstrzyknięcie kodu z index poprzez `{{body}}`.*

Przy renderowaniu strony możemy również dodać inne informacje. Przykładowe wywołanie:

```
res.render('index', { title: 'Galeria', id: 5, admin: false });
```

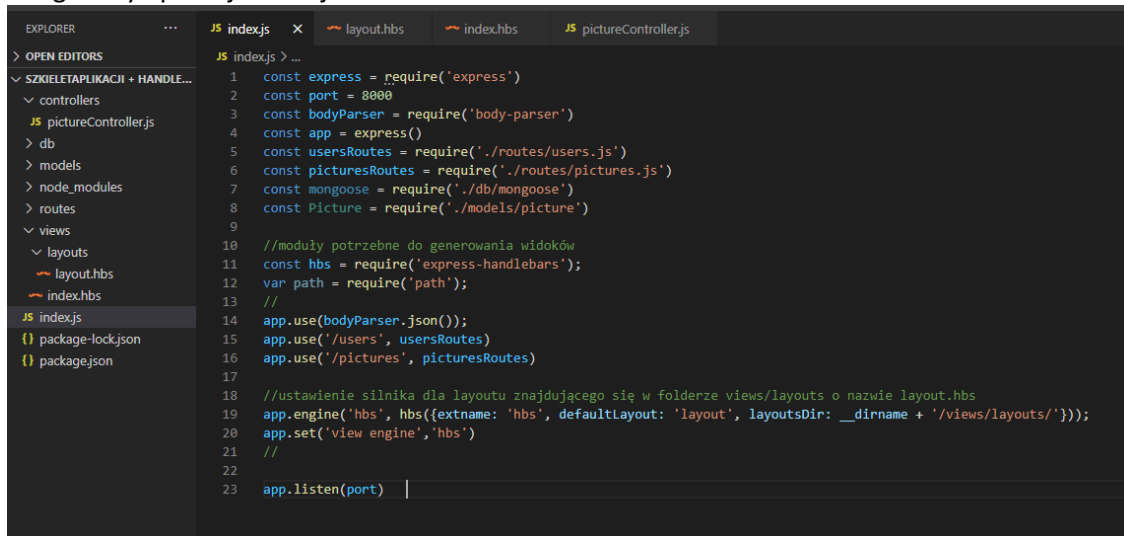
Tak przekazane dane można wyświetlić za pomocą podwójnych nawiasów klamrowych np.:

```
{{ title }}
```

Podsumowując ten fragment potrzebujemy:

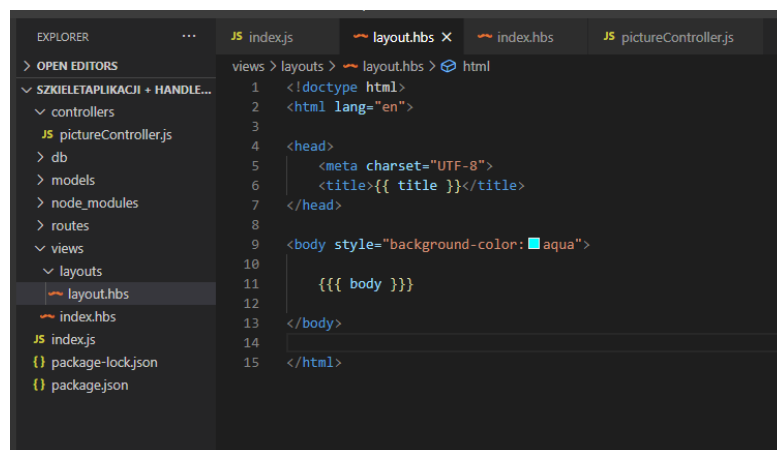
Stworzyć plik: `views/layouts/layout.hbs`, `views/index.hbs`,  
Uzupełnić plik: `controllers/pictureController.js`, `index.js`

## Plik główny aplikacji index.js



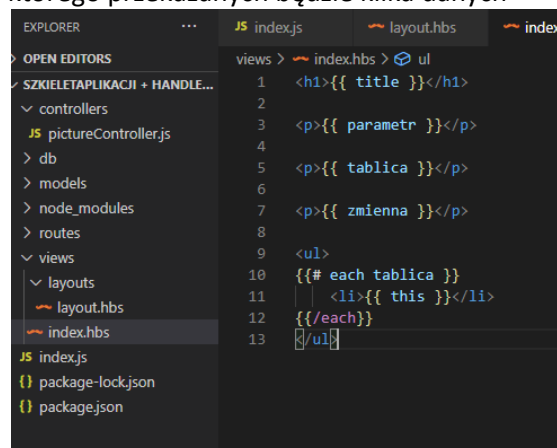
```
1 const express = require('express')
2 const port = 8000
3 const bodyParser = require('body-parser')
4 const app = express()
5 const usersRoutes = require('./routes/users.js')
6 const picturesRoutes = require('./routes/pictures.js')
7 const mongoose = require('./db/mongoose')
8 const Picture = require('./models/picture')
9
10 //moduły potrzebne do generowania widoków
11 const hbs = require('express-handlebars');
12 var path = require('path');
13 //
14 app.use(bodyParser.json());
15 app.use('/users', usersRoutes)
16 app.use('/pictures', picturesRoutes)
17
18 //ustawienie silnika dla layoutu znajdującego się w folderze views/layouts o nazwie layout.hbs
19 app.engine('hbs', hbs({extname: 'hbs', defaultLayout: 'layout', layoutsDir: __dirname + '/views/layouts/'}));
20 app.set('view engine', 'hbs')
21 //
22
23 app.listen(port)
```

## Plik layoutu



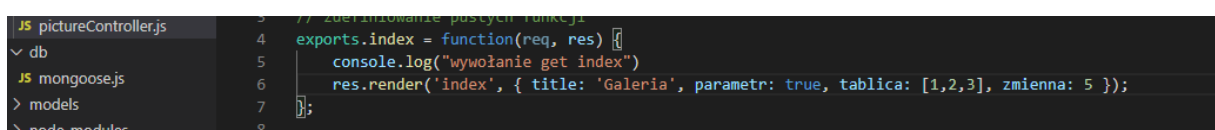
```
1 <!doctype html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>{{ title }}</title>
7 </head>
8
9 <body style="background-color: aqua">
10
11   {{ body }}
12
13 </body>
14
15 </html>
```

## Plik index.hbs w widoku do którego przekazanych będzie kilka danych



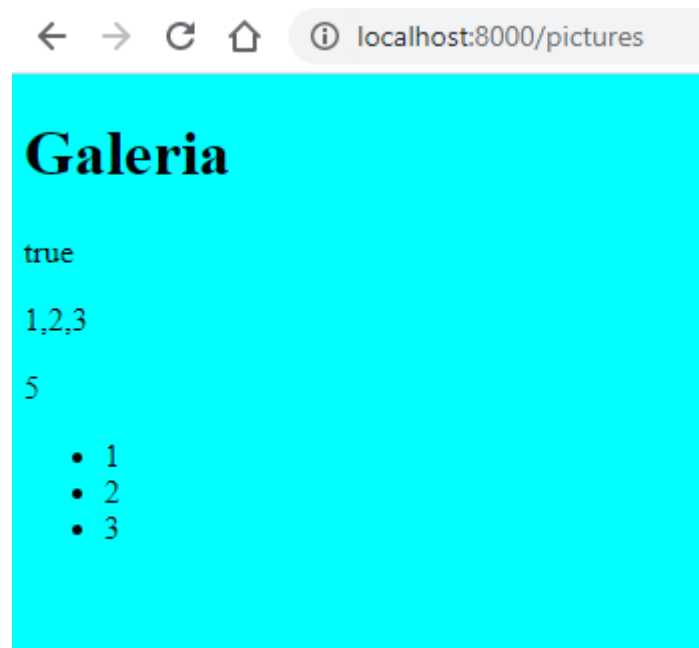
```
1 <h1>{{ title }}</h1>
2
3 <p>{{ parametr }}</p>
4
5 <p>{{ tablica }}</p>
6
7 <p>{{ zmienna }}</p>
8
9 <ul>
10   {{# each tablica }}
11     <li>{{ this }}</li>
12   {{/each}}
13 </ul>
```

W kontrolerze dla testu można zmodyfikować np. funkcję index:



```
3 // zdefiniowanie pustych funkcji
4 exports.index = function(req, res) {
5   console.log("wywołanie get index")
6   res.render('index', { title: 'Galeria', parametr: true, tablica: [1,2,3], zmienna: 5 });
7 };
8
```

Analizując aktualne zmiany powinniśmy dzięki zastosowaniu layoutu mieć niebieskie tło oraz dane przekazane podczas renderowania wyświetlone w index zagnieżdżonym w layoutcie. Aby to sprawdzić wystarczy połączyć się z adresem: <http://localhost:8000/pictures>



Szablon działa poprawnie i przekazane wartości wyświetlone zostały prawidłowo. Przeanalizujmy plik `index.hbs`:

W 1 linii kodu `<h1>{{ title }}</h1>` - wstrzyknięcie argumentu do `<h1>` jako tekst.

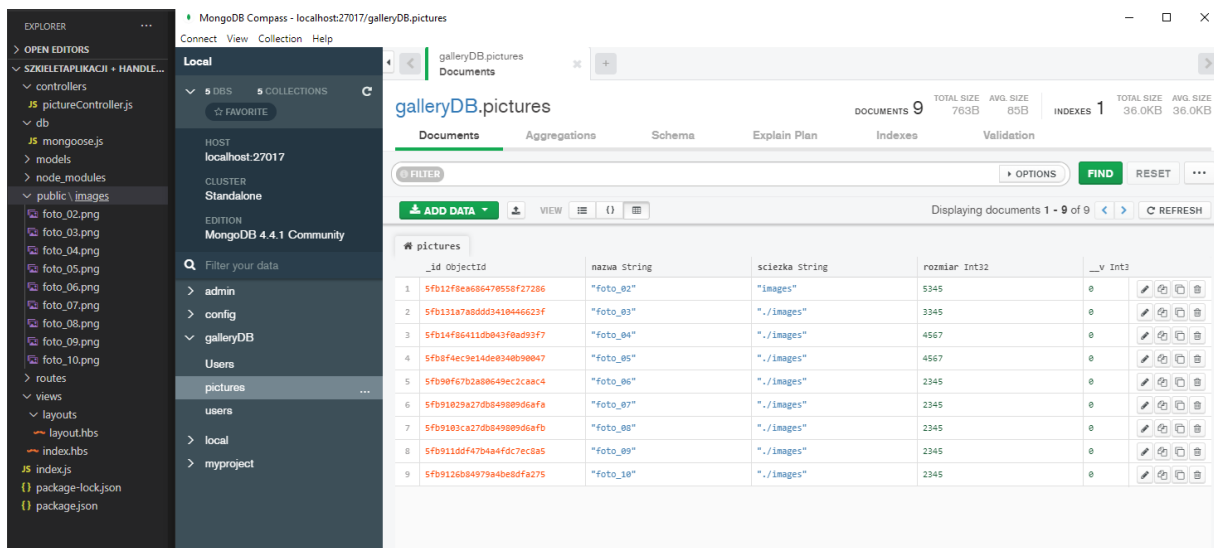
W lini 9-13 mamy wylistowanie i została zastosowana pętla `{{# each tablica }}` instrukcje `{{/each}}`. Wewnątrz za pomocą `{{ this }}` wyświetlane są kolejne elementy tablicy, natomiast żeby wylistować za pomocą znaczników html pętla obudowana została znacznikami `<ul><li>`. W rezultacie w przeglądarce możemy podejrzeć jak wygląda utworzona struktura:



Możemy spróbować wyświetlić zdjęcia w galerii. Aby tego dokonać przyda się kilka zdjęć testowych, ustawienie folderu oraz napisanie fragmentu kodu html do wyświetlenia. Zaczniemy od ustawienia folderu. Jako że Express domyślnie uniemożliwia zagłębienie do plików z poziomu przeglądarki, możemy wymusić aby jeden folder był dostępny statycznie. Aby to zrobić wystarczy dodać linijkę w głównym pliku index.js

```
JS index.js x layout.hbs index.hbs JS pictureController.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 picturesRoutes = require('./routes/pictures.js')
7  const mongoose = require('./db/mongoose')
8  const Picture = require('./models/picture')
9  const hbs = require('express-handlebars');
10 var path = require('path');
11
12 // aby nasze zdjęcia były widoczne należy dodać parametr
13 app.use(express.static(path.join(__dirname, 'public')));
14
15 app.use(bodyParser.json());
16 app.use('/users', usersRoutes) |
17 app.use('/pictures', picturesRoutes)
18
19
20 app.engine('hbs', hbs({extname: 'hbs', defaultLayout: 'layout', layoutsDir: __dirname + '/views/layouts/'}));
21 app.set('view engine', 'hbs')
22
23
24 app.listen(port)
```

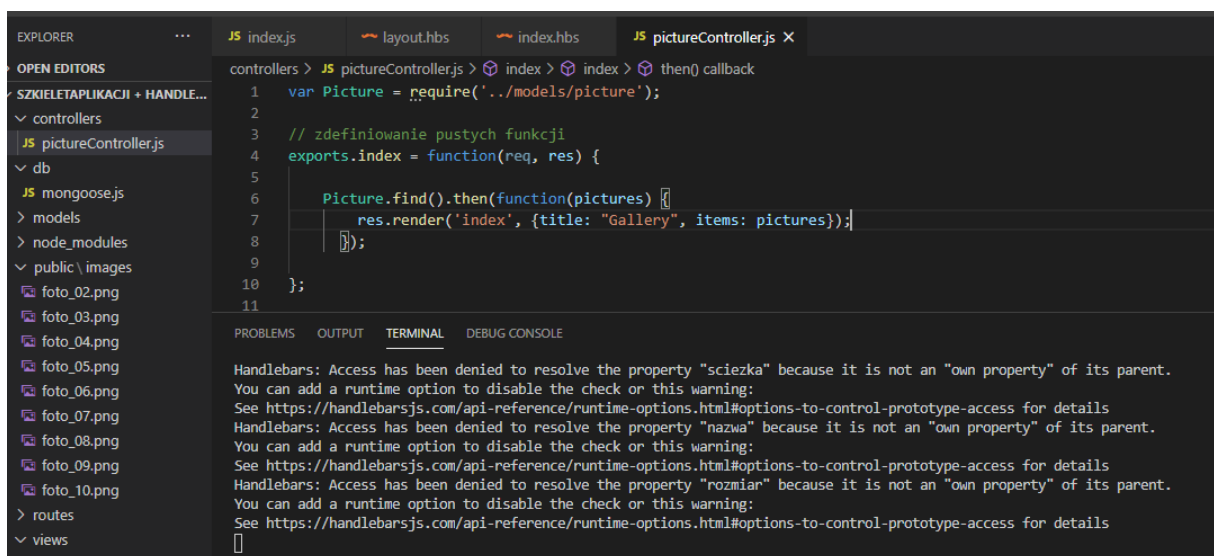
Stworzymy teraz folder Public, a w nim images i niech znajdą się w nim testowe pliki graficzne. Jeszcze nie mamy możliwości z poziomu przeglądarki dodawać zdjęć oraz informacji o nich do bazy, dlatego musimy stworzyć kilka pozycji w kolekcji o nazwie pasującej do pliku.



#	pictures	_id ObjectId	nazwa String	sciezka String	rozmiar Int32	_v Int32
1		5fb12f8e686478558f27286	"foto_02"	"./images"	5345	0
2		5fb131a78d6d53418446623f	"foto_03"	"./images"	3345	0
3		5fb14f86411db043f0ad93f7	"foto_04"	"./images"	4567	0
4		5fb0f4ec9e14de0348b98047	"foto_05"	"./images"	4567	0
5		5fb98f67b2a88649ec2caac4	"foto_06"	"./images"	2345	0
6		5fb91029a27db049809dcafa	"foto_07"	"./images"	2345	0
7		5fb9103ca27db049809dcafb	"foto_08"	"./images"	2345	0
8		5fb911d5f47b4a4f9c7ec8a5	"foto_09"	"./images"	2345	0
9		5fb9126b04979a4be8dfa275	"foto_10"	"./images"	2345	0

Teraz zmodyfikujmy plik index.hbs aby faktycznie wyświetlił przekazane elementy jako zdjęcie, ustawiając ścieżkę przekazaną w argumencie jako element `<img src="">`

```
views > index.hbs > ...
1
2   {{# each items }}
3     <div >
4       
5       nazwa: {{ this.nazwa }} rozmiar: {{ this.rozmiar }}kb
6     <br />
7   </div>
8   {{/each}}
9
10  {{!--
11  <h1>{{ title }}</h1>
```



```
EXPLORER
... JS index.js layout.hbs index.hbs JS pictureController.js X
OPEN EDITORS
controllers > JS pictureController.js > index > index > then() callback
1 var Picture = require('../models/picture');
2
3 // zdefiniowanie pustych funkcji
4 exports.index = function(req, res) {
5
6   Picture.find().then(function(pictures) {
7     res.render('index', {title: "Gallery", items: pictures});
8   });
9
10 };
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Handlebars: Access has been denied to resolve the property "sciezka" because it is not an "own property" of its parent. You can add a runtime option to disable the check or this warning: See <https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access> for details

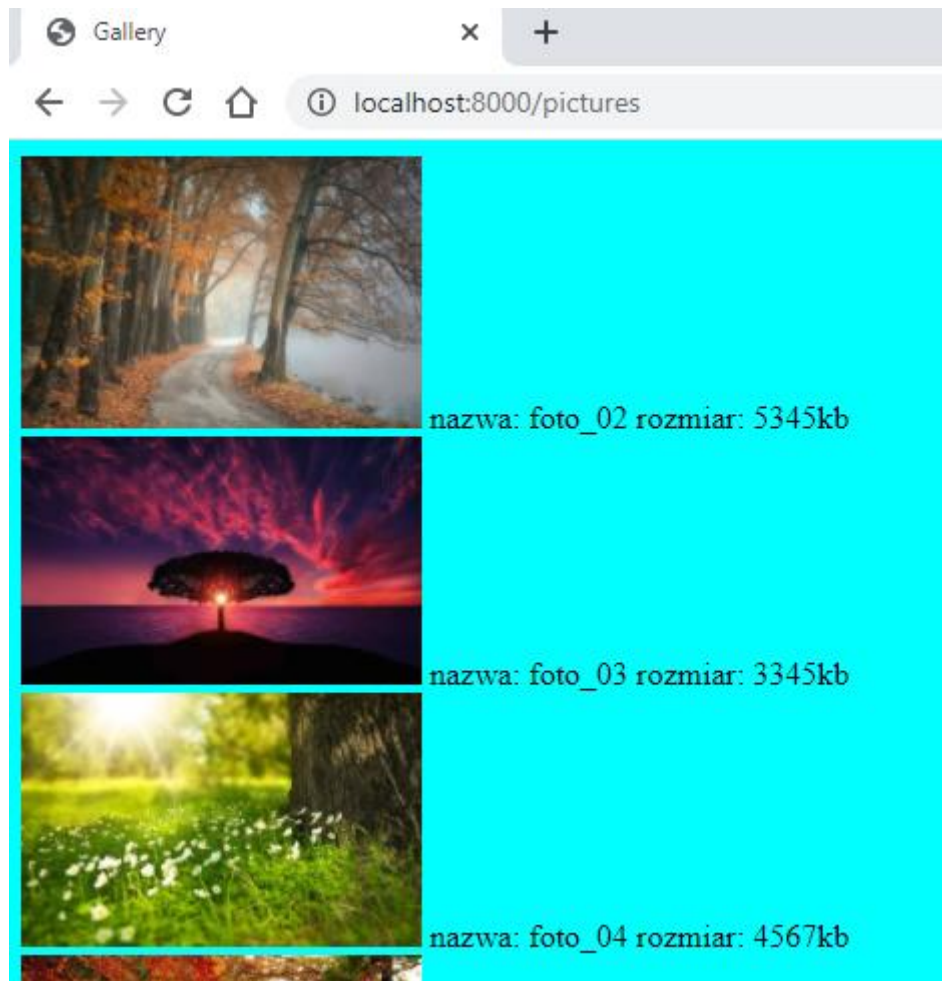
Handlebars: Access has been denied to resolve the property "nazwa" because it is not an "own property" of its parent. You can add a runtime option to disable the check or this warning: See <https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access> for details

Handlebars: Access has been denied to resolve the property "rozmiar" because it is not an "own property" of its parent. You can add a runtime option to disable the check or this warning: See <https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access> for details

Zdjęcia nie wyświetliły się, problemem okazują się przekazane argumenty, ale struktura html zadziałała poprawnie. W konsoli otrzymaliśmy komunikat ostrzegawczy „Handlebars: Access has been denied to resolve the property”. Zatem trzeba poradzić sobie dodając odpowiednie opcje do pliku głównego aplikacji.

```
16
17 // dodanie opcji runtime
18 app.engine('hbs', hbs({extname: 'hbs', defaultLayout: 'layout', layoutsDir: __dirname + '/views/layouts/',
19 runtimeOptions: {
20   allowProtoPropertiesByDefault: true,
21   allowProtoMethodsByDefault: true
22 } }));
23 app.set('view engine', 'hbs')
24
```

Problem powinien zniknąć i jeżeli nie było literówek oraz innych błędów powinniśmy zobaczyć w przeglądarce zdjęcia.



Teraz spróbujmy zrobić obsługę post do wystawienia zdjęcia. Aby to zrobić potrzebujemy formularz, niech nazywa się `create.hbs` i znajduje się w folderze `views`.

```
views > create.hbs > ...  
1  
2  
3 <form action="../pictures/create" method="post">  
4   <label for="nazwa">Nazwa:</label>  
5   <input type="text" id="nazwa" name="nazwa"><br><br>  
6   <label for="rozmiar">Rozmiar:</label>  
7   <input type="number" id="rozmiar" name="rozmiar"><br><br>  
8   <input type="submit" value="Submit">  
9 </form>  
10  
11
```

W `routes/pictures.js` możemy dodać (o ile już nie są stworzone) ścieżkę do obsługi post i get dla funkcji mającej za zadanie dodać zdjęcie do bazy danych.

```

16 router.get('/create', picture_controller.picture_create_get);
17 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/create
18
19 router.post('/create', picture_controller.picture_create_post );
20 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/create
21

```

Oraz musimy dodać ścieżki w kontrolerze

```

controllers > JS pictureController.js > picture_create_post > picture_create_post
38
39 exports.picture_create_get = function(req, res) {
40
41
42     res.render('create');
43
44 };
45
46 exports.picture_create_post = function(req, res, next) {
47
48
49     var picture = new Picture(
50     {
51         nazwa: req.body.nazwa,
52         rozmiar: req.body.rozmiar,
53     });
54
55
56     picture.save().then(() => {
57         res.send('Add new picture');
58     }).catch(err => {
59         console.log(err)
60     })
61
62 };
63

```

W ostatnim kroku aby poprawnie zadziała formularz powinniśmy zmodyfikować ustawienia bodyParsera. Opis zmiany jest w dokumentacji: <https://github.com/expressjs/body-parser#bodyparserurlencodedoptions>. Wystarczy ustawić extended na true.

```

13
14 //zmiana w ustawieniach bodyParsera tak aby można było przysyłać dane z formularza do bazy
15 app.use(bodyParser.urlencoded({
16     extended: true
17 }));
18
19




























```

Teraz możemy uruchomić formularz pod adresem <http://localhost:8000/pictures/create>



I po akceptacji otrzymujemy komunikat zwrotny

I możemy sprawdzić w mongoDBCompass czy faktycznie została dodana pozycja:

	_id ObjectId	nazwa String	sciezka String	rozmiar Int32	__v Int3	
1	5fb12f8ea686470558f27286	"foto_02"	"images"	5345	0	  
2	5fb131a7a8ddd3410446623f	"foto_03"	"./images"	3345	0	  
3	5fb14f86411db043f0ad93f7	"foto_04"	"./images"	4567	0	  
4	5fb8f4ec9e14de0340b90047	"foto_05"	"./images"	4567	0	  
5	5fb90f67b2a80649ec2caac4	"foto_06"	"./images"	2345	0	  
6	5fb91029a27db849809d6afa	"foto_07"	"./images"	2345	0	  
7	5fb9103ca27db849809d6afb	"foto_08"	"./images"	2345	0	  
8	5fb911ddf47b4a4fdc7ec8a5	"foto_09"	"./images"	2345	0	  
9	5fc4a44d20d95d28bcc065e2	"foto_00"	"./images"	1234	0	  

Aby nie pominąć wyglądu z layoutu wynik możemy również wyświetlić renderując stronę wynikową. Dlatego można utworzyć dodatkowy plik widoku show.hbs:

```

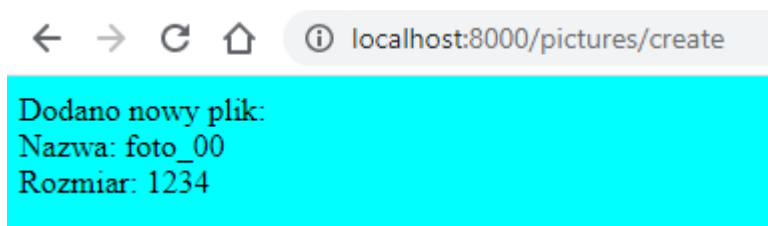
> OPEN EDITORS
SZKIELETAPLIKACJI + HANDLE...
  controllers
    JS pictureController.js
  db
    JS mongoose.js
  models
    JS picture.js
  > node_modules
  > public \ images
  routes
    JS pictures.js
    JS users.js
  views
    layouts
      layout.hbs
    create.hbs
    index.hbs
    show.hbs

views > show.hbs > ...
1 Dodano nowy plik:
2 <br />
3 Nazwa:
4 {{ item.nazwa }}
5 <br />
6 Rozmiar:
7 {{ item.rozmiar }}
```

Musimy również zmienić `res.send` na `res.render` i przekazać argumenty z `req.body` w kontrolerze `pictureControllers.js`.

```
58
59     picture.save().then(() => {
60         console.log(picture)
61         res.render('show',{item: req.body});
62     }).catch(err => {
63         console.log(err)
64     })
65
66 };
```

I teraz po dodaniu nowej pozycji informacja zwrotna generowana jest na layoutcie.



## Zadanie.

Na podstawie tych informacji stwórz widoki dla wszystkich kolekcji z możliwością wyświetlania i dodawania elementów. Rozwiń wygląd layoutu i zadbaj o czytelne wyświetlanie.