

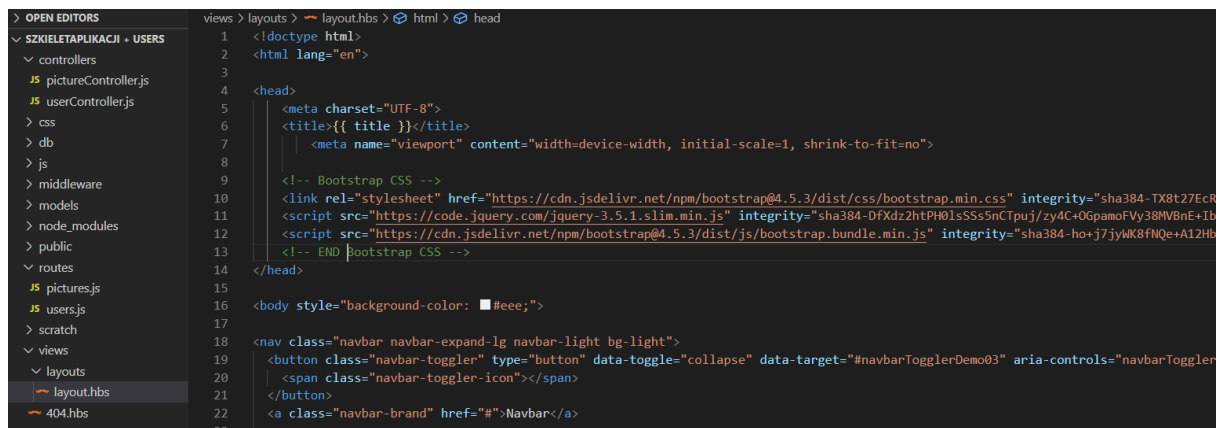
# Wygląd + pełny CRUD

Do stworzenia ładnych dla oka serwisów możemy wykorzystać jedno z najpopularniejszych narzędzi jakim jest Bootstrap, zapewniający wiele komponentów, klas, stylów oraz co najważniejsze responsywnych rozwiązań i przykładów.

<https://getbootstrap.com/>

Do aplikacji można wgrać pliki zgodnie z instrukcją na stronie i przykładem

<https://github.com/twbs/bootstrap-npm-starter>. Jednak możemy również w łatwiejszy sposób skorzystać z jsDelivr (instrukcja na stronie bootstrap) i dodać zewnętrzne linki do css i js potrzebnych do generowania ładnego wyglądu. W przykładach nie będę prezentował całej aplikacji, a jedynie istotne elementy. Grafika, style, układ jest dowolny, według upodobań twórcy (w przyszłości klienta/pracodawcy). Na początku należy dodać w sekcji head odpowiednie odnośniki zgodnie z instrukcją jsDelivr w pliku layouts/layout.hbs. W tym przykładzie został dodany na sztywno navbar, nie jest na ten moment konieczny, ale zbliży wyglądem do efektu końcowego:



```
> OPEN EDITORS
SZKIELETAPLIKACJI + USERS
  controllers
    pictureController.js
    userController.js
  css
  db
  js
  middleware
  models
  node_modules
  public
  routes
  pictures.js
  users.js
  scratch
  views
    layouts
      layout.hbs
      404.hbs
      create.hbs
      createShow.hbs

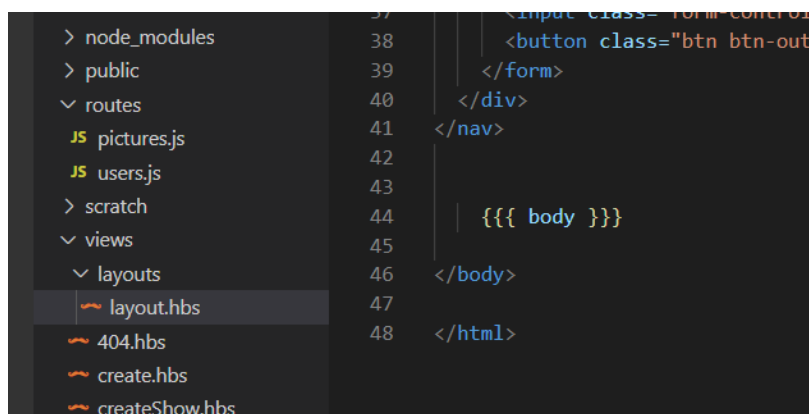
views > layouts > layout.hbs > html > head
1 <!doctype html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>{{ title }}</title>
7   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9   <!-- Bootstrap CSS -->
10  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcR"
11  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+Ib"
12  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb"
13  <!-- END Bootstrap CSS -->
14 </head>
15
16 <body style="background-color: #eee;">
17
18 <nav class="navbar navbar-expand-lg navbar-light bg-light">
19   <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo03" aria-controls="navbarToggler"
20   <span class="navbar-toggler-icon"></span>
21 </button>
22   <a class="navbar-brand" href="#">Navbar</a>
23
```

Aby przetestować wystarczy skorzystać z dowolnego komponentu:

<https://getbootstrap.com/docs/5.0/components/accordion/>

W tym momencie warto zapoznać się z layoutami i kontenerami, układem.

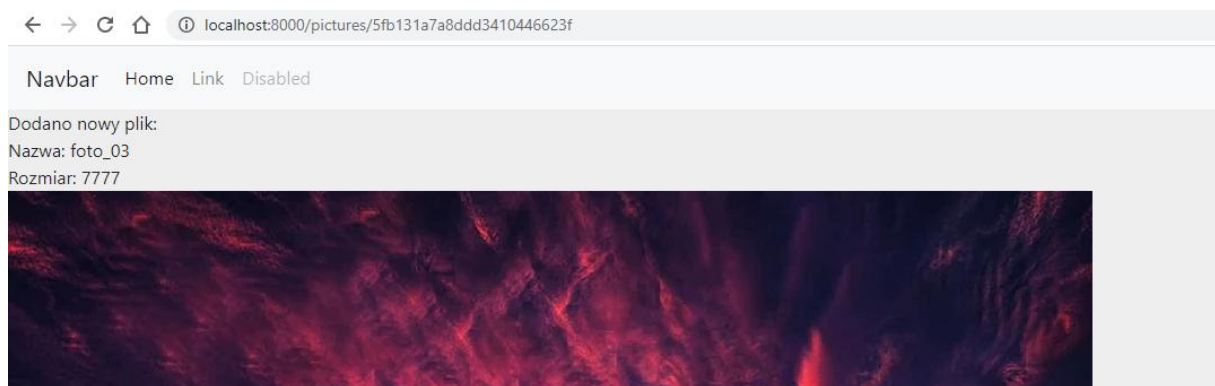
Aby wszystko działało poprawnie na podstawie aplikacji tworzonej do tej pory należy pamiętać o zagnieżdżeniu plików z widoków poprzez użycie {{{body}}}



```
> node_modules
> public
> routes
  JS pictures.js
  JS users.js
> scratch
> views
  layouts
    layout.hbs
    404.hbs
    create.hbs
    createShow.hbs

37 <input class="form-control" type="text">
38 <button class="btn btn-outline-primary" type="button">Submit</button>
39 </form>
40 </div>
41 </nav>
42
43
44 {{{ body }}}
45
46 </body>
47
48 </html>
```

Możemy uruchomić i zobaczyć efekt. W tym przykładzie skorzystałem z `Get/:id` wyświetlającej obrazek po id:



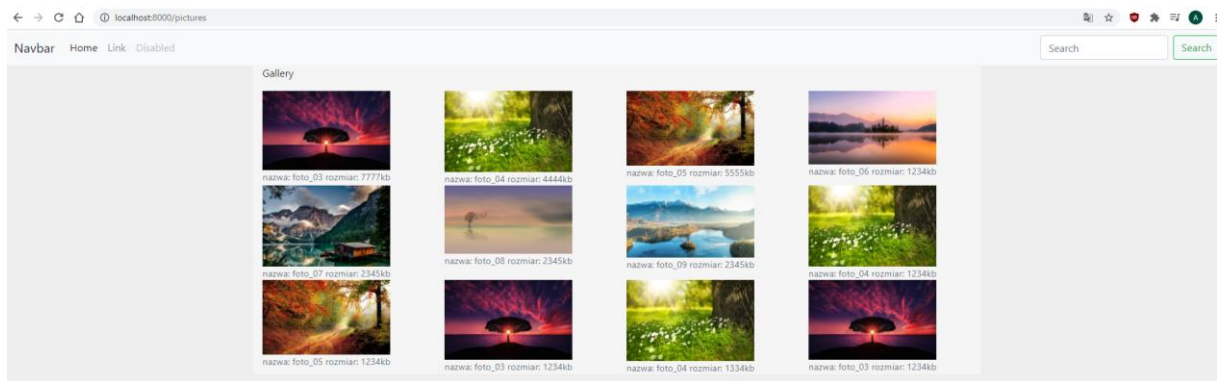
Jak widać pasek nawigacji został dodany prawidłowo i pozostanie niezmiennie.

Teraz możemy zmienić wygląd również wyświetlania poszczególnych elementów. Niezmiennie ważne jest aby zapoznać się z klasami `row`, `container`, `col`. Zmodyfikujmy zatem plik nazwany `index.hbs` który jest odpowiedzialny za wyświetlanie zdjęć w galerii:

```
EXPLORER    ...    ller.js    JS authenticate.js    UserLogin.hbs    createUser.hbs    userCreate.hbs    JS pict
> OPEN EDITORS
SZKIELETAPLIKACJI + USERS
  controllers
    JS pictureController.js
    JS userController.js
  css
  db
  js
  middleware
  models
  node_modules
  public
  routes
    JS pictures.js
    JS users.js
  scratch
  views
    layouts
      layout.hbs
    404.hbs
    create.hbs
    createShow.hbs
    createUser.hbs
    edit.hbs
    index.hbs
    show.hbs

views > index.hbs > div.container > div.row
1  <div class="container" style="background-color: whitesmoke">
2
3  <p>{{ title }}</p>
4
5  <div class="row">
6    {{# each items }}
7      <div class="col-sm">
8        
9        <figcaption class="figure-caption text-left">
10         nazwa: {{ this.nazwa }} rozmiar: {{ this.rozmiar }}kb
11        </figcaption>
12      </div>
13    </div>
14  </each>
15
16
17
18
```

Po uruchomieniu powinniśmy otrzymać czytelny wygląd. Powyższy przykład zawiera komponenty bootstrapa, ale to jakie komponenty wybierze jest kwestią indywidualną



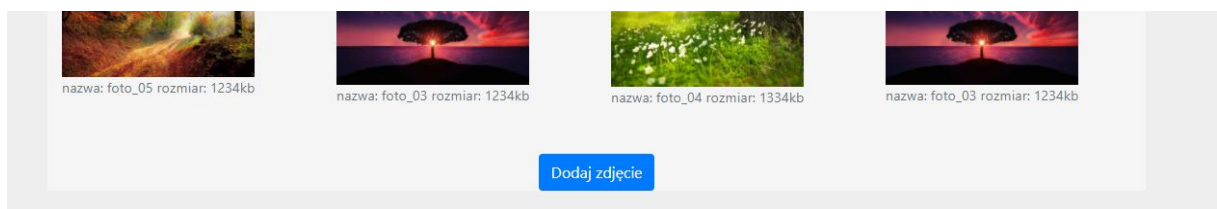
Jak widać galeria ma już zwarty wygląd. W wersji backendowej oczywiście może być to lista, ale ładny i czytelny wygląd w każdym miejscu jest istotny. Możemy teraz przejść do modyfikacji plików widoków oraz dopełnieniu CRUD. W aplikacji backendowej powinniśmy mieć bardzo elastyczne podejście i możliwość edycji, dlatego pełny CRUD jest bardzo potrzebny.

Na początku możemy umieścić w widocznym miejscu przycisk do dodawania zdjęć. Zatem dopiszmy formularz do pliku index.

```

18  {{/each}}
19
20  </div>
21  <br />
22  <br />
23  <div style="text-align: center;">
24    <form action="pictures/create" method="get">
25      <button class="btn btn-primary" type="submit" class="btn-link">Dodaj zdjęcie</button>
26    </form>
27  </div>
28
29  </div>
30
31

```



Aby zadziałało powinniśmy zdefiniować Get w route i obsłużyć renderując formularz dodawania zdjęć.

W pliku router/pictures.js

```

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

```

Oraz w pliku controllers/picturesController.js

```

39
40 exports.picture_create_get = function (req, res) {
41
42     res.render('create');
43
44 };
45

```

Plik create powinniśmy utworzyć pod ścieżką views/create.hbs. Przykład formularza zawiera ścieżkę do której odnosimy się oraz metodę post, a także poszczególne inputy których nazwa powinna być adekwatna do pola w przechwytywającej metodzie post. W tym przypadku jedno pole jest text, drugie number. Dostępne typy można znaleźć pod adresem:

[https://www.w3schools.com/html/html\\_form\\_input\\_types.asp](https://www.w3schools.com/html/html_form_input_types.asp)

```

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

```

Teraz w routes możemy dodać metodę post:

```

routes
19 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/create
20
JS pictures.js
21 router.post('/create', picture_controller.picture_create_post );
JS users.js
22 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/create

```

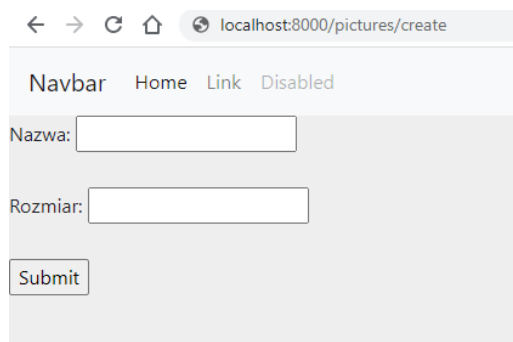
I zdefiniować :

```

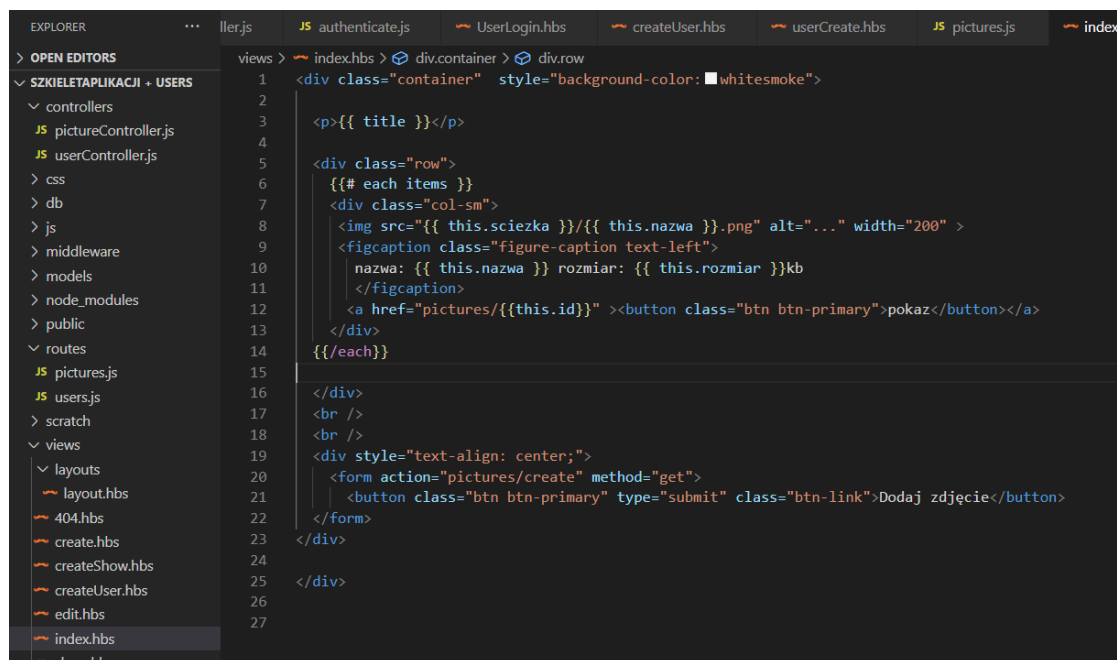
JS pictureController.js
JS userController.js
> css
> db
> js
> middleware
> models
> node_modules
> public
> routes
JS pictures.js
JS users.js
> scratch
> views
  > layouts
    layout.hbs
  404.hbs
  create.hbs
  createShow.hbs
  createUser.hbs
43
44 };
45
46 exports.picture_create_post = function (req, res) {
47
48     console.log(req.body.rozmiar)
49
50     const picture = new Picture(
51     {
52         nazwa: req.body.nazwa,
53         rozmiar: req.body.rozmiar,
54     });
55
56
57     picture.save().then(() => {
58         console.log(picture)
59         res.render('createShow', { item: req.body , path: "../images/"});
60     }).catch(err => {
61         console.log(err)
62     })
63
64 };
65

```

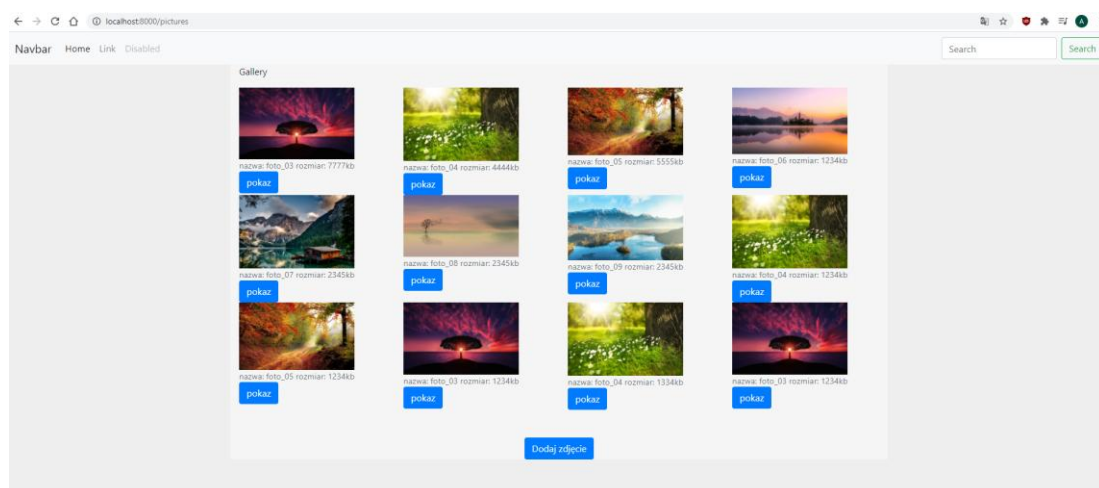
Teraz po kliknięciu przycisku dodaj zdjęcie powinniśmy otrzymać formularz. W przykładzie wygląd nie został jeszcze obrobiony przy użyciu bootstrap, warto w tym momencie upiększyć wygląd:



W pliku widoku głównego index.hbs dla zdjęć możemy dodać teraz przycisk który spowoduje wyświetlenie konkretnego zdjęcia (linijka 12):



```
1 <div class="container" style="background-color: #whitesmoke">
2
3   <p>{{ title }}</p>
4
5   <div class="row">
6     {{# each items }}
7     <div class="col-sm">
8       {{ this.nazwa }}.png alt="..." width="200" >
9       <figcaption class="figure-caption text-left">
10        nazwa: {{ this.nazwa }} rozmiar: {{ this.rozmiar }}kb
11      </figcaption>
12      <a href="pictures/{{this.id}}" ><button class="btn btn-primary">pokaz</button></a>
13    </div>
14  </div>
15  {{/each}}
16
17 </div>
18 <br />
19 <br />
20 <div style="text-align: center;">
21   <form action="pictures/create" method="get">
22     <button class="btn btn-primary" type="submit" class="btn-link">Dodaj zdjęcie</button>
23   </form>
24 </div>
25
26 </div>
```



Aby zadziało musimy dodać ścieżkę do routes/pictures.js ścieżkę dla Get i zdefiniować w kontrolerze

Routes/pictures.js:

```
404.hbs      33
create.hbs   34   router.get('/:id', picture_controller.picture_info);
createShow.hbs 35   //Dostęp do funkcji pod adresem http://localhost:8000/pictures/id
createUser.hbs 36
```

Controllers/pictureController.js (w dowolnym miejscu, tutaj jest w linii 151 ponieważ zdążyły inne funkcje się pojawić)

```
150
151   exports.picture_info = function (req, res) {
152     var id = req.params.id
153     Picture.findOne({ _id: id }).then(function (pictures) {
154       console.log(pictures)
155       res.render('show', { item: pictures });
156     });
157     console.log(req.params)
158
159   };
160
```

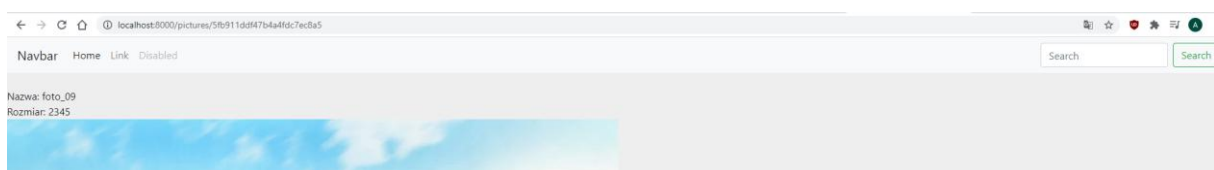
W powyższym przykładzie pobieramy id z parametrów i wyszukujemy w modelu Pictures po id, a następnie renderujemy plik views/show.hbs wraz z obiektem item zawierającym wszystkie pola z modelu Pictures, które możemy obsłużyć w widoku odnosząc się poprzez {{ item.element }}

Plik views/show.hbs

```
views > show.hbs > img
1   <br />
2   Nazwa:
3   {{ item.nazwa }}
4   <br />
5   Rozmiar:
6   {{ item.rozmiar }}
7   <br />
8   
```

W tym miejscu również należy dodać elementy bootstrap. W przykładzie nie zostało to zrobione.

Po kliknięciu w przycisk „pokaż” powinniśmy otrzymać obrazek wraz z informacjami zagnieżdżony w layoucie:



Teraz możemy dodać kolejne przyciski do edycji i usuwania. W przykładzie wykorzystam metodę post i delete, jednak domyślnie powinny być adekwatnie put i delete.

W pliku route zamieszczam ścieżki:

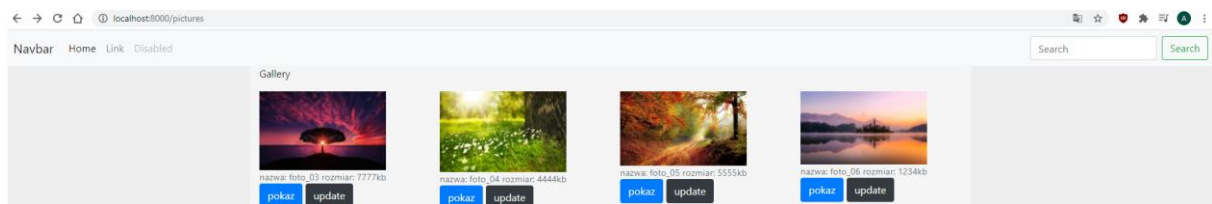
```
22 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/create
23
24 router.get('/update/:id', picture_controller.picture_update_get );
25 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/update:id
26
27 router.post('/update/:id', picture_controller.picture_update_post );
28 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/update:id
29
30 router.post('/delete', picture_controller.picture_delete_post );
31
32
33
34 router.get('/:id', picture_controller.picture_info);
35 //Dostęp do funkcji pod adresem http://localhost:8000/pictures/id
36
```

W pliku kontrolera:

```
81 exports.picture_update_post = function (req, res) {
82
83     let picture = {}
84     picture.nazwa = req.body.nazwa
85     picture.rozmiar = req.body.rozmiar
86
87     const id = req.params.id
88
89     Picture.findByIdAndUpdate(id, picture,
90         function (err, docs) {
91             if (err) {
92                 console.log(err)
93             }
94             else {
95                 res.redirect('/pictures');
96             }
97         });
98
99 };
100
101
102 exports.picture_update_get = function (req, res) {
103
104     const id = req.params.id
105     let data;
106     Picture.findOne({ _id: id }).then(function (picture) {
107         res.render('edit', { item: picture, id: id });
108     });
109
110
111
112 };
```

W powyższym przykładzie wykorzystane zostały funkcje findByIdAndUpdate oraz findOne. Do dyspozycji jest kilka, można je zobaczyć w dokumentacji. W przypadku pierwszej funkcji wykorzystany jest redirect, to znaczy jeśli funkcja nie miała problemów z aktualizacją wówczas przekieruje adres do adresu głównego z listą zdjęć.

W tym momencie powinniśmy dodać odpowiedni button do widoku głównego aby mieć możliwość modyfikacji konkretnego zdjęcia



Należy stworzyć formularz który automatycznie uzupełni się danymi ale jednocześnie pozwoli nam wczytać zmiany:

```
views > edit.hbs > ...
1
2 <div class="col-sm-2">
3 <form action="../../update/{{id}}" method="post">
4 <div class="form-group">
5   <label for="nazwa">Nazwa:</label>
6   <input class="form-control" type="text" id="nazwa" name="nazwa" value="{{item.nazwa}}">
7   <label for="rozmiar">Rozmiar:</label>
8   <input class="form-control" type="number" id="rozmiar" name="rozmiar" value="{{item.rozmiar}}">
9 </div>
10 <input type="submit" value="Submit" class="btn btn-primary">
11 </form>
12 </div>
13
```

Tym razem wykorzystujemy `{{id}}` aby wiedzieć do jakiego zdjęcia się odnosimy. W value zamieszczamy również aktualne wartości pobrane za pomocą obiektu `item`. Po akceptacji buttonem wykonana zostanie metoda `post` dla `update/konkretny_id`.

Ostatnim krokiem jest możliwość usunięcia zdjęcia. Musimy zdefiniować w kontrolerze funkcję.

```
66 exports.picture_delete_post = function (req, res) {
67   id = req.body.id
68   console.log(id)
69   Picture.findByIdAndDelete(id,
70     function (err, docs) {
71     if (err) {
72       console.log(err)
73     }
74     else {
75       console.log("usuwa")
76       res.redirect('/pictures');
77     }
78   });
79 }
80
```

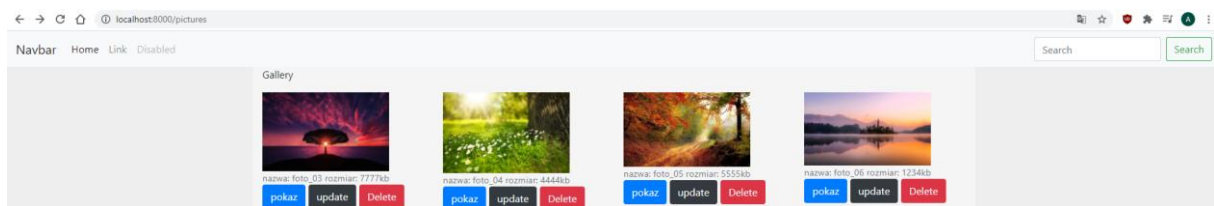
W tym przypadku metoda `delete` nie ma w ścieżce `id`, ale możemy `id` przekazać podczas wywołania przycisku usuwania, a następnie pobrać `id` za pomocą `req.body.id` i usunąć wykorzystując metodę `findByIdAndDelete` na obiekcie `Picture`. W tym przypadku znowu po wykonaniu przekierowujemy na stronę główną.



Do przesłania id możemy wykorzystać formularz będący samym przyciskiem i przekazać id. Do tego celu dodajemy ostatni przycisk do index.hbs obsługujący galerię:

```
8      
9      <figcaption class="figure-caption text-left">
10         nazwa: {{ this.nazwa }} rozmiar: {{ this.rozmiar }}kb
11      </figcaption>
12      <a href="pictures/{{this.id}}" ><button class="btn btn-primary">pokaz</button></a>
13      <a href="pictures/update/{{this.id}}"><button class="btn btn-dark">update </button> </a>
14      <form action="pictures/delete" method="post" style="display: inline;">
15      <button class="btn btn-danger" type="submit" name="id" id="{{this.id}}" value="{{this.id}}" class="btn-link">Delete</button>
16    </form>
17  </div>
18  {{/each}}
19
```

Tak wysłane id powinno trafić do metody post usuwającej, usunąć i przekierować na stronę główną. A wygląda to następująco:



## Zadanie.

Zdefiniuj CRUD dla galerii z formularzami, posprzątaj w plikach widoku dodając odpowiednie foldery, zapoznaj się z metodami Get i Delete i przekształć na ich podstawie kod. Zapoznaj się z bootstrapem i wykorzystaj do zbudowania ładnego layoutu galerii.