

Horses Project

Refaktoryzacja

O czym jest aplikacja?

Aplikacja dotyczy pokazów (konkursów “piękności”) dla koni. Główne funkcjonalności:

- Dodawanie, edycja i usuwanie danych o sędziach, klasach oraz koniach.
- Możliwość przeniesienia konia z klasy do drugiej klasy.
- Możliwość zmiany numerów startowych.
- Możliwość edycji komisji sędziowskich dla klas.

PS Projekt pierwotnie powstał na zajęcia z dr Pawłowskim

Po co refaktoryzujemy?

- Aby dostosowywać kod do obowiązujących standardów..
- .. i dobrych praktyk
- Sprawiamy, że kod staje się prostszy i czytelniejszy, a aplikacja jest prostsza w utrzymaniu
- Zapobiegamy przed pojawieniem się błędów podczas dodawania nowych funkcjonalności
- Niwelujemy dług technologiczny
- Chcemy pisać lepszy kod i być lepszymi programistami

JavaScript (Express.js) jest trudny w utrzymaniu

Express, w porównaniu do wielu innych dostępnych frameworków, nie ma z góry narzuconej struktury katalogów i plików.

Niektóre zespoły deweloperskie przyjmują pewne konwencje, aby kod dało się utrzymywać i rozwijać bez popadania w depresję.

```
8 // Dear programmer:
9 // When I wrote this code, only god and
10 // I knew how it worked.
11 // Now, only god knows it!
12 //
13 // Therefore, if you are trying to optimize
14 // this routine and it fails (most surely),
15 // please increase this counter as a
16 // warning for the next person:
17 //
18 // total_hours_wasted_here = 254
19 //
20
```

W Express.js często wygląda to tak:

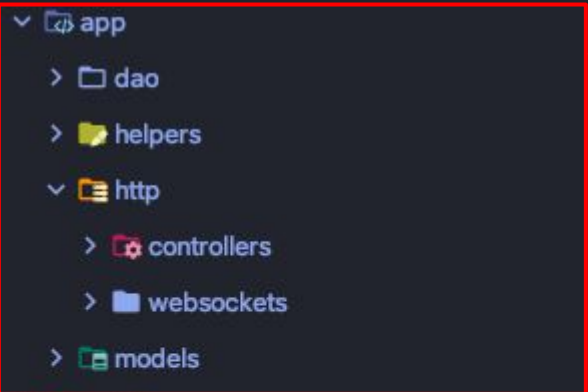
```
    }).done(function(response) {
      for (var i = 0; i < response.length; i++) {
        var layer = L.marker(
          [response[i].latitude, response[i].longitude]
          // ,{icon: myIcon}
        );
        layer.addTo(group);

        layer.bindPopup(
          "<p>" + "Species: " + response[i].species + "<br>" +
          "<p>" + "Description: " + response[i].description + "<br>" +
          "<p>" + "Seen at: " + response[i].latitude + " " + response[i].longitude + "<br>" +
          "<p>" + "On: " + response[i].sighted_at + "</p>"
        );
      }

      $('select').change(function() {
        species = this.value;
      });
    });
  }
  $.ajax({
    url: queryURL,
    method: "GET"
  }).done(function(response) {
    for (var i = 0; i < response.length; i++) {
      var layer = L.marker(
        [response[i].latitude, response[i].longitude]
        // ,{icon: myIcon}
      );
      layer.addTo(group);
    }
  });
}
```

Czyli możemy scrollować plik app.js w nieskończoność. W jednym pliku zawiera się konfigurację, łączenie z bazą danych, routingi, modele, dao, kontrolery i tak dalej... w nieskończoność.

Konwencja przyjęta przez nasz zespół



```
▼ app
  > dao
  > helpers
  ▼ http
    > controllers
    > websockets
  > models
```

Logika biznesowa, a w niej:

- DAO (Data Access Object) umożliwia komunikację między db, a kontrolerem
- helpers, czyli klasy/metody pomocnicze
- controllers, do manipulacji danymi
- websockets, jak sama nazwa wskazuje służą do websocketów
- models, czyli odzwierciedlenie struktury tabeli w bazie danych na obiekt

```
> config
> node_modules library root
> routes
app.js
package.json
package-lock.json
```

Pliki konfiguracyjne

Routingi (endpointy dla API)

Główny plik, coś jak index.php

Dzięki temu nasz plik app.js wygląda następująco:

```
1  var express = require('express');
2  var db = require('./config/database');
3  var routes = require('./routes/index');
4  var socket = require('./app/http/websockets/socket');
5  var libraries = require('./config/libraries');
6  var properties = require('./config/properties');
7  var app = express();
8  var router = express.Router();
9
10 db();
11 libraries(app);
12
13 var server = require('http').createServer(app);
14 socket(server);
15
16 app.use(router);
17 routes(router);
18
19 server.listen(properties.PORT, () => {
20   console.log(`Serwer działa na porcie ${properties.PORT}`);
21 });
```

Pobierane są wszystkie niezbędne moduły, czyli połączenie z bazą danych, routingi, skonfigurowane paczki i sockety.

Z kolei te moduły również są łatwe w modyfikacji gdyż zawierają stosunkowo niewielką ilość kodu. Staramy się aby pliki były jak najbardziej atomowe.

Przykład dla klasy “Sędzia”

Model:

```
1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3  var judgesSchema = new Schema({
4    sedzia: {
5      type: String,
6      required: true
7    },
8    kraj: {
9      type: String,
10     required: true
11   }
12 });
13
14 module.exports = judgesSchema;
15
```

Data Access Object:

```
1  var mongoose = require('mongoose');
2  var judgesSchema = require('../models/judges.model');
3
4  judgesSchema.statics = {
5    create: function (data, cb) {
6      var judge = new this(data);
7      judge.save(cb);
8    },
9    get: function (query, cb) {
10     this.find(query, cb);
11   },
12   getById: function (query, cb) {
13     this.find(query, cb);
14   },
15   update: function (query, data, cb) {
16     this.findOneAndUpdate(query, { $set: data }, { new: true }, cb);
17   },
18   delete: function (query, cb) {
19     this.findOneAndDelete(query, cb);
20   }
21 };
22
23 var judgesModel = mongoose.model( name: 'Judges', judgesSchema);
24 module.exports = judgesModel;
25
```


Przykład dla klasy “Sędzia”

Routing:

```
1 var Judges = require('../app/http/controllers/judges.controller');
2
3 module.exports = function (router) {
4   router.post('/sedziowie', Judges.createJudge);
5   router.get('/sedziowie', Judges.getJudges);
6   router.get('/sedziowie/:id', Judges.getJudge);
7   router.put('/sedziowie/:id', Judges.updateJudge);
8   router.delete('/sedziowie/:id', Judges.removeJudge);
9 };
10
```

Przykład metody z kontrolera:

```
1 var Judges = require('../../dao/judges.dao');
2
3 exports.createJudge = function (req, res, next) {
4   var judge = {
5     sedzia: req.body.sedzia,
6     kraj: req.body.kraj
7   };
8
9   Judges.create(judge, function (err, judge) {
10     if (err) {
11       res.json({
12         error: err
13       });
14     }
15     res.send(judge);
16   });
17 }
```

A nie można tego wszystkiego w jednym pliku?

Można. Ale... wtedy kod w tym jednym pliku wydłużyłby się o 115 linii. I to tylko dla jednego modelu! A przecież mamy ich trzy: Koń, Sędzia i Klasa. Gdybyśmy całą logikę biznesową zamieścili w jednym pliku, miałby on długość 642 linii. I wtedy przychodzi klient i chce zmienić nazwę jednego pola w klasie Sędzia. Jaki jest efekt?

```
8 // Dear programmer:
9 // When I wrote this code, only god and
10 // I knew how it worked.
11 // Now, only god knows it!
12 //
13 // Therefore, if you are trying to optimize
14 // this routine and it fails (most surely),
15 // please increase this counter as a
16 // warning for the next person:
17 //
18 // total_hours_wasted_here = 254
19 //
20
```

+n

A wystarczy w odpowiednich plikach
dopisać / zmienić kilka linijek.

Co refaktoryzujemy - utrzymanie kolejności numerów klas oraz koni = dwa razy ten sam kod!

```
let classUpdate = req.body;
let beforeUpdate = db.getCollection("class").findOne({
  "$loki": classUpdate.$loki
});
if (classUpdate.number !== beforeUpdate.number) {
  let lastClass = db.getCollection("class").chain().find().simplsort("number",
true).data()[0];
  if (classUpdate.number > lastClass.number) {
    if (classUpdate.$loki !== lastClass.$loki) {
      let classesToChangeNumberDown = db.getCollection("class").find({
        "number": {
          "$gt": beforeUpdate.number
        }
      });
      classesToChangeNumberDown.forEach(el => {
        el.number--;
        db.getCollection("horse").update(el);
      });
      classUpdate.number = lastClass.number + 1;
    } else {
      classUpdate.number = lastClass.number;
    }
  }
} else {
```

```
if (beforeUpdate.number < classUpdate.number) {
  let classesToChangeNumberDown = db.getCollection("class").find({
    "number": {
      "$between": [beforeUpdate.number + 1, classUpdate.number]
    }
  });
  classesToChangeNumberDown.forEach(el => {
    el.number--;
    db.getCollection("class").update(el);
  });
} else if (beforeUpdate.number > classUpdate.number) {
  let classesToChangeNumberUp = db.getCollection("class").find({
    "number": {
      "$between": [classUpdate.number, beforeUpdate.number - 1]
    }
  });
  classesToChangeNumberUp.forEach(el => {
    el.number++;
    db.getCollection("class").update(el);
  });
}
}
db.getCollection("class").update(classUpdate);
return res.status(201).json(classUpdate);
```

Data Clump - wykonywanie zapytań do bazy danych

Wydzielamy DAO dla każdego modelu

```
const db = require("../database/db");
const Class = {
  get : () =>{
    return db.getCollection("class").chain().find().simplesort("number", false).data();
  },
  getSortByNumberDesc : () => {
    return db.getCollection("class").chain().find().simplesort("number", true).data();
  },
  getByNumerBetween: (nr1, nr2) => {
    return db.getCollection("class").find({ "number": { "$between": [nr1, nr2] } });
  },
  save: (data) => {
    return db.getCollection("class").insert(data);
  },
  update: (data) => {
    db.getCollection("class").update(data);
  },
  remove: (data) => {
    db.getCollection("class").remove(data);
  }
};
module.exports = Class;
```

Zastosowanie DAO

```
let classUpdate = req.body;
let beforeUpdate = Class.getById(classUpdate.$loki);
if (classUpdate.number !== beforeUpdate.number) {
  let lastClass = Class.getSortByNumberDesc()[0];
  if (classUpdate.number > lastClass.number) {
    if (classUpdate.$loki !== lastClass.$loki) {
      Class.getByNumberGreaterThan(beforeUpdate.number)
        .forEach(el => {
          el.number--;
          Horse.update(el);
        });
      classUpdate.number = lastClass.number + 1;
    } else {
      classUpdate.number = lastClass.number;
    }
  } else {
```

```
    if (beforeUpdate.number < classUpdate.number) {
      Class.getByNumberBetween(beforeUpdate.number + 1,
        classUpdate.number)
        .forEach(el => {
          el.number--;
          Class.update(el);
        });
    } else if (beforeUpdate.number > classUpdate.number) {
      Class.getByNumberBetween(classUpdate.number,
        beforeUpdate.number - 1)
        .forEach(el => {
          el.number++;
          Class.update(el);
        });
    }
  }
}
```

Pamiętajmy, że mamy dwa razy ten sam kod!

Rozwiązanie to wyłączenie metody.

```
const changeNumberAndUpdate = (database, val, nr1, nr2) => {
  database.getByNumberBetween(nr1, nr2)
    .forEach(el => {
      el.number = el.number + val;
      database.update(el);
    });
};

exports.changeNumberOnUpdate = (database, beforeUpdate, toUpdate) => {
  let all = database.getSortByNumberDesc();
  let lastByNumber = all[0];
  if (toUpdate.number > lastByNumber.number) {
    toUpdate.number = lastByNumber.number;
    changeNumberAndUpdate(database, -1, beforeUpdate.number + 1, toUpdate.number);
  } else {
    if (beforeUpdate.number < toUpdate.number) {
      changeNumberAndUpdate(database, -1, beforeUpdate.number + 1, toUpdate.number);
    } else if (beforeUpdate.number > toUpdate.number) {
      changeNumberAndUpdate(database, 1, toUpdate.number, beforeUpdate.number - 1);
    }
  }
  return toUpdate.number;
};
```


Efektem tego jest:

```
let classUpdate = req.body;
let beforeUpdate = Class.getById(classUpdate.$loki);
if (classUpdate.numer !== beforeUpdate.numer) {
    classUpdate.numer = guardNumer.changeNumerOnUpdate(Class, beforeUpdate, classUpdate);
}
Class.update(classUpdate);
return res.status(201).json(classUpdate);
```


Ale to jeszcze nie koniec!


```
exports.changeNumberOnUpdate = (database, beforeUpdate, toUpdate) => {  
  let all = database.getSortByNumerDesc();  
  let lastByNumer = all[0];  
  if (toUpdate.numer > lastByNumer.numer) {  
    toUpdate.numer = lastByNumer.numer;  
  }  
  if (beforeUpdate.numer < toUpdate.numer) {  
    changeNumberAndUpdate(database, -1, beforeUpdate.numer + 1, toUpdate.numer);  
  } else {  
    changeNumberAndUpdate(database, 1, toUpdate.numer, beforeUpdate.numer - 1);  
  }  
  return toUpdate.numer;  
};
```


Dodaliśmy to nieszczęsne pole. Co dalej?

Wrzucamy na produkcję, a użytkownicy przetestują. 

Testujemy API przez Postmana. 

Testujemy ręcznie? 

Wrzucamy to na produkcję w piątek o 16:00 i idziemy do domu ? 

I wtedy wchodzą one, całe ubrane na białą (zielono) ->

TESTY, TESTY I JESZCZE RAZ TESTY

Po co? Dlaczego? Co nam to da? Komu to potrzebne?

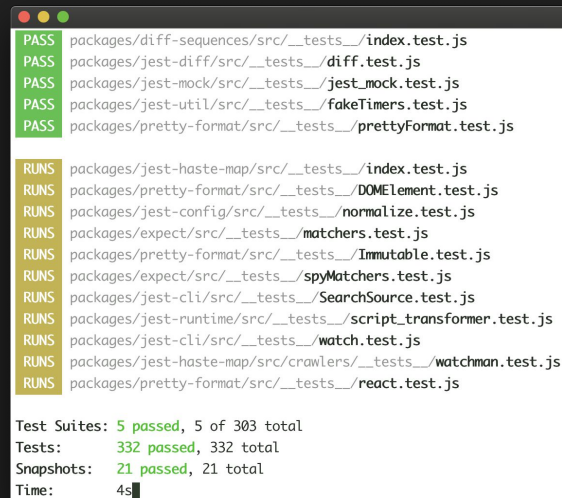
- Testujemy my, a nie klient i w dodatku nie robimy tego ręcznie
- Zapobiegamy (no prawie) przed pojawieniem się błędów na produkcji
- Piszemy je **RAZ** i nie używamy w kółko Postmana lub innych narzędzi
- Wyłapujemy błędy często niewidoczne na pierwszy rzut oka

Ciekawostka:

Chociaż TDD (Test Driven Development) wydaje się wydłużać proces tworzenia oprogramowania, jest zupełnie odwrotnie. Według badań przyspiesza proces o około 60%. I mamy w 100% przetestowany kod.

Jest

- Narzędzie stworzone przez Facebooka do testowania JavaScriptowego kodu.
- Możliwość testowania frontendu, jak i backendu (API).
- Szybszy od podobnych narzędzi ze względu na równoległe przeprowadzanie testów.



```
PASS packages/diff-sequences/src/__tests__/index.test.js
PASS packages/jest-diff/src/__tests__/diff.test.js
PASS packages/jest-mock/src/__tests__/jest_mock.test.js
PASS packages/jest-util/src/__tests__/fakeTimers.test.js
PASS packages/pretty-format/src/__tests__/prettyFormat.test.js

RUNS packages/jest-haste-map/src/__tests__/index.test.js
RUNS packages/pretty-format/src/__tests__/DOMElement.test.js
RUNS packages/jest-config/src/__tests__/normalize.test.js
RUNS packages/expect/src/__tests__/matchers.test.js
RUNS packages/pretty-format/src/__tests__/Immutable.test.js
RUNS packages/expect/src/__tests__/spyMatchers.test.js
RUNS packages/jest-cli/src/__tests__/SearchSource.test.js
RUNS packages/jest-runtime/src/__tests__/script_transformer.test.js
RUNS packages/jest-cli/src/__tests__/watch.test.js
RUNS packages/jest-haste-map/src/crawlers/__tests__/watchman.test.js
RUNS packages/pretty-format/src/__tests__/react.test.js

Test Suites: 5 passed, 5 of 303 total
Tests:       332 passed, 332 total
Snapshots:   21 passed, 21 total
Time:        4s
```

Test #1 DELETE Judge

```
17 describe("DELETE Judge", () => {
18   test("It responds with a message of Deleted", async () => {
19     const newJudge = await request(app)
20       .post("/sedziowie")
21       .send({
22         sedzia: "Khasjan Yksahek",
23         kraj: "Wenezuela"
24       });
25
26     const removedJudge = await request(app).delete(
27       `/sedziowie/${newJudge.body._id}`
28     );
29     expect(removedJudge.body).toEqual({
30       message: "Judge deleted successfully"
31     });
32     expect(removedJudge.statusCode).toBe(202);
33   }, 30000);
34 });
```

Test #2 UPDATE Class

- Dodajemy sędziego.
- Tworzymy klasę z utworzonym sędzią.
- Udatujemy klasę.

```
54 | describe("UPDATE klasy", () => {
55 |   test("It responds with an updated class", async () => {
56 |     const newJudge = await request(app)
57 |       .post("/sedziowie")
58 |       .send({
59 |         sedzia: "Jerzy Tymbark",
60 |         kraj: "Polska"
61 |       });
62 |
63 |     const newClass = await request(app)
64 |       .post("/klasy")
65 |       .send({
66 |         numer: 1,
67 |         kat: 'klacze jednoroczne',
68 |         czempionat: 1,
69 |         komisja: [newJudge.body._id]
70 |       });
71 |
72 |     const updatedClass = await request(app)
73 |       .put(`/klasy/${newClass.body._id}`)
74 |       .send({
75 |         numer: 2,
76 |         kat: 'klacze jednoroczne',
77 |         czempionat: 1,
78 |         komisja: [newJudge.body._id]
79 |       });
80 |
81 |     expect(updatedClass.statusCode).toBe(200);
82 |   }, 30000);
83 | });
```