Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

# POLITECHNIKA LUBELSKA
# WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW
INFORMATYKA

*MATERIAŁY DO ZAJĘĆ LABORATORYJNYCH*

## Szkielety programistyczne w aplikacjach internetowych

dr Mariusz Dzieńkowski

Lublin 2022

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

# LABORATORIUM 4. IMPLEMENTACJA INTERAKCJI Z BAZĄ DANYCH

## Cel laboratorium:

Opanowanie podstawowych umiejętności pracy z bazą danych MongoDB. Nawiązywanie połączenia z aplikacją opartą na szkielecie Express. Implementacja aplikacji wykonującej podstawowe operacje na bazie danych (CRUD): tworzenia, czytania, aktualizowania i usuwania danych.

## Zakres tematyczny zajęć:

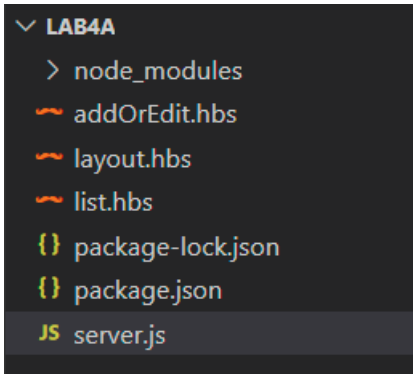
- Budowa nierelacyjnej bazy danych MongoDB.
- Podstawowe polecenia powłoki MongoDB umożliwiające korzystanie z bazy.
- Stosowanie modułu Mongoose.
- Praca z bazą danych: ustanawianie połączenia, tworzenie schematów i modeli, operacje na bazie.
- Konstruowanie routingu.

## Pytania kontrolne:

a) Opisz strukturę nierelacyjnej bazy danych.
b) Wymień podstawowe metody powłoki mongo służące do wykonywania operacji na bazie danych.
c) Za pomocą jakiej metody można nawiązać połączenie z bazą i jakich argumentów ona wymaga?
d) W jaki sposób tworzy się schemat, a następnie model przy pomocy modułu Mongoose?

## Zadanie 4.1. Interakcja z bazą danych MongoDB

W ramach tego zadania zostanie utworzony prosty system zarządzania studentami (SZS) operujący na bazie danych MongoDB, umożliwiający tworzenie, aktualizowanie i usuwanie danych o studentach. Na tym etapie wszystkie pliki będą znajdowały się w katalogu głównym projektu (Rys. 4.1). Dodatkowo cały kod aplikacji będzie znajdował się w pliku *server.js* oraz plikach widoków (*addOrEdit.hbs*, *layout.hbs*, *list.hbs*).




*Rys. 4.1 Struktura projektu aplikacji*

**Etap 1.** W katalogu projektu należy utworzyć plik *server.js* – stanowiący główny element aplikacji i łączący inne pliki projektu. Jego kod znajduje się na Listingu 4.1. Obejmuje on następujące działania:

- Import modułów: *express*, *path*, *handlebars*, *express-handlebars* oraz *@handlebars/allow-prototype-access*
- Parsowanie JSON w żądaniu HTTP realizowane za pomocą funkcji pośredniczącej:

```
app.use(express.urlencoded({
        extended: true
}));
```

- Ustawienie miejsca przechowywania widoków – katalog główny aplikacji. W tym miejscu będą znajdowały się pliki szablonów handlebars zawierające kod odpowiedzialny za utworzenie interfejsu użytkownika.
- Uruchomienie nasłuchiwania serwera.

***Listing 4.1** Fragment kodu głównej części aplikacji – plik server.js*

```
const express = require("express")
const path = require("path")
const handleBars = require("handlebars")
const exphbs = require("express-handlebars")
const {allowInsecurePrototypeAccess} = require("@handlebars/allow-prototype-access")

const app = express()

app.use(express.urlencoded({
  extended: true
}))

app.set("views", path.join(__dirname, "/"))

app.engine(
  "hbs",
  exphbs.engine({
    handlebars: allowInsecurePrototypeAccess(handleBars),
    extname: "hbs",
    defaultLayout: "layout",
    layoutsDir: __dirname,
  })
)

app.set("view engine", "hbs")

app.listen(3000, () => {
  console.log("Serwer nasłuchuje na porcie 3000")
})
```

**Etap 2.** Dodanie do pliku *server.js* kodu odpowiadającego za połączenie z bazą danych MongoDB (Listing 4.2). Kod ten obejmuje 2 działania:

- Import pakietu mongoose
  const mongoose = require("mongoose")
- Zbudowanie schematu *studentSchema* - definiującego strukturę przechowywanych w bazie dokumentów oraz utworzenie modelu *Student*
- Użycie metody *connect()* do połączenia z bazą danych: dodanie adresu URL do bazy danych, implementacja funkcji wywołania zwrotnego do obsługi wyjątków

***Listing 4.2*** *Fragment kodu niezbędnego do działań na bazie danych*

```
const studentSchema = new mongoose.Schema({
   fullName: String,
   email: String,
   mobile: Number,
   city: String,
})

const Student = mongoose.model("Student", studentSchema)

mongoose.connect("mongodb://localhost:27017/StudentDB", { useNewUrlParser: true })
   .then((result) => {
      console.log("Połączono z bazą")
   }).catch((err) => {
      console.log("Nie można połączyć się z MongoDB. Błąd: " + err)
})
```

**Etap 3.** Utworzenie tras obsługujących różne typy żądań (Listing 4.3)

- Trasa "/" służąca do obsługi żądania GET protokołu http – generowanie strony głównej aplikacji
- Trasa "/list" obsługująca żądanie GET służące do pobrania rekordów z bazy danych, wyrenderowanie strony HTML na podstawie szablonu i dostarczonych do niego zmiennych oraz wysłanie gotowego dokumentu jako odpowiedź żądania
- Trasa "/addOrEdit" obsługująca żądanie GET, dzięki któremu będzie możliwe wyświetlenie formularza obsługującego dodawanie oraz aktualizację danych studenta
- Trasa "/" obsługująca żądanie POST, dzięki któremu będzie możliwe dodawanie oraz aktualizacja danych studenta (funkcje insert() i update())
- Trasa "/:id" obsługująca żądanie GET, dzięki któremu na podstawie id studenta będzie możliwe pobranie, wyświetlenie w formularzu orazz aktualizacja danych studenta
- Trasa "/delete/:id" realizująca usuwanie studenta z bazy danych na podstawie jego id

***Listing 4.3*** *Dodawanie tras obsługujących różne typy żądań*

```
app.get("/", (req, res) => {
   res.send(`
   <h3 style="text-align:center">Baza danych studentów</h3>
   <h4 style="text-align:center">Kliknij <a href="/list"> tutaj</a>, aby uzyskać dostęp do bazy.</h4>`)
})
```

```
app.get("/list", (req, res) => {
   Student.find().then((docs) => {
      res.render("list", {
         list: docs
      })
   }).catch((err) => {
      console.log("Błąd pobierania danych" + err)
   })
})

app.get("/addOrEdit", (req, res) => {
   res.render("addOrEdit", {
      viewTitle: "Dodaj studenta"
   })
})
app.post("/", (req, res) => {
   if (req.body._id == "") {
      insert(req, res)
   } else {
      update(req, res)
   }
})
app.get("/:id", (req, res) => {
   Student.findById(req.params.id).then((doc) => {
      res.render("addOrEdit", {
         viewTitle: "Zaktualizuj dane studenta",
         student: doc
      })
   }).catch((err) => {
      console.log("Błąd podczas akutalizowania danych" + err)
   })
})
app.get("/delete/:id", (req, res) => {
   Student.findByIdAndRemove(req.params.id).then((doc) => {
      res.redirect("/list")
   }).catch((err) => {
      console.log("Błąd podczas usuwania: " + err)
   })
})
```

**Etap 4.** Implementacja funkcji realizujących zapis danych do bazy - funkcja insert() oraz aktualizację danych w bazie - funkcja update()

***Listing 4.4*** *Implementacja funkcji insert() oraz update()*

```
async function insert(req, res) {
   let student = new Student()
   student.fullName = req.body.fullName
   student.email = req.body.email
   student.mobile = req.body.mobile
```

```
    student.city = req.body.city
    try {
        await student.save()
        res.redirect("/list")
    } catch (err) {
        console.log("Błąd podczas dodawania studenta: " + err)
    }
}
async function update(req, res) {
    try {
        await Student.findOneAndUpdate({ _id: req.body._id }, req.body, { new: true })
        res.redirect("list")
    } catch (err) {
        console.log("Błąd podczas aktualizowania danych: " + err)
    }
}
```

**Etap 5.** Projektowanie wzorca szablonów (ang. layout)

Plik *hbs* jest szablonem obsługiwanym przez Handlebars, będącym silnikiem szablonów internetowych. W pliku *hbs* tworzy się szablon zawierający znaczniki HTML oraz wyrażenia Handlebars.

Plik wzorca szablonów o nazwie *layout.hbs* należy umieścić w katalogu głównym aplikacji. Będzie on pełnił rolę kontenera dla interfejsu użytkownika.

Kod znajdujący się na Listingu 4.5 obejmuje:

- Dodanie podstawowego szkieletu HTML w celu wyświetlenia danych
- Wykorzystanie szkieletu Bootstrap do stylizacji elementów interfejsu
- Dodanie odpowiednich stylów dla znaczników *div*
- Użycie wyrażeń Handlebars w postaci {{body}} do umieszczenia treści pobieranych z bazy na stronie www

***Listing 4.5*** *Definicja wzorca szablonów layout.hbs*

```
<!DOCTYPE html>
<html>
<head>
    <title>Studenci</title>
    <link rel="stylesheet"
          href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
          integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
          crossorigin="anonymous">
    <style>
        #box {
            background-color: #fff;
            margin-top: 20px;
            padding: 20px;
        }
    </style>
</head>
```

```
<body class="bg-info">
  <div id="box" class="col-md-6 offset-md-3" </div>
    {{{body}}}
</body>
</html>
```

**Etap 6.** Budowa formularza służącego do dodawania i edycji danych

Również plik *addOrEdit.hbs* - zawierający kod formularza (Listing 4.6), należy utworzyć w folderze głównym aplikacji.

***Listing 4.6*** *Kod formularza – plik addOrEdit.hbs*

```
<h3>{{viewTitle}}</h3>
<form action="/" method="POST" autocomplete="off">
  <input type="hidden" name="_id" value="{{student._id}}">
  <div class="form-group">
    <label>Full Name</label>
    <input type="text" class="form-control" name="fullName" placeholder="Full Name"
           value="{{student.fullName}}">
  </div>
  <div class="form-group">
    <label>Email</label>
    <input type="text" class="form-control" name="email" placeholder="Email"
           value="{{student.email}}">
  </div>
  <div class="form-row">
    <div class="form-group col-md-6">
      <label>Mobile</label>
      <input type="text" class="form-control" name="mobile" placeholder="Mobile"
             value="{{student.mobile}}">
    </div>
    <div class="form-group col-md-6">
      <label>City</label>
      <input type="text" class="form-control" name="city" placeholder="City"
             value="{{student.city}}">
    </div>
  </div>
  <div class="form-group">
    <button type="submit" class="btn btn-info">Submit</button>
    <a class="btn btn-secondary" href="/list">View All</a>
  </div>
</form>
```

**Etap 7.** Szablon do wyświetlenia listy studentów pobranych z bazy – plik *list.hbs*

Efektem działania kodu z Listingu 4.7 będzie tabela wyświetlająca dane studentów. Nawiasy klamrowe {{}}} stanowią składnię Handlebars, która wstawia dane do tabeli, które są następnie wyświetlane w przeglądarce internetowej.

***Listing 4.7*** *Szablon tworzący listę z danymi studentów – plik list.hbs*

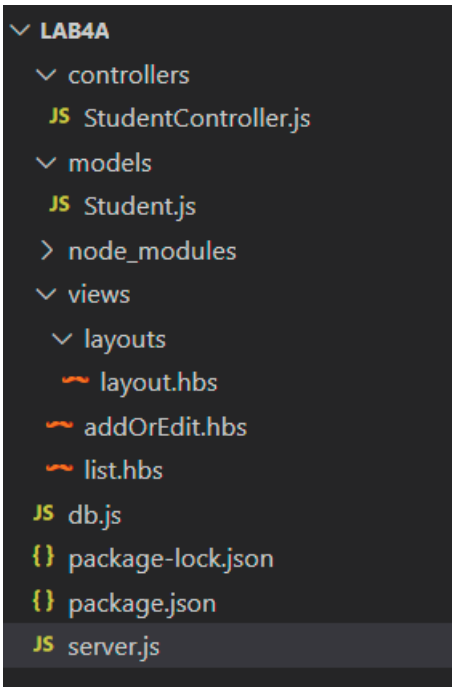
```
<h3 style="text-align:center">Lista studentów</h3>
<a class="btn btn-secondary" href="/addOrEdit">Utwórz nowy</a>

<table class="table table-striped">
  <thead>
    <tr>
      <th>Imię i nazwisko</th>
      <th>Email</th>
      <th>Telefon</th>
      <th>Miasto</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    {{#each list}}
    <tr>
      <td>{{fullName}}</td>
      <td>{{email}}</td>
      <td>{{mobile}}</td>
      <td>{{city}}</td>
      <td>
        <a class="btn btn-primary btn-sm" href="/{{_id}}">Edytuj</a>
        <a class="btn btn-danger btn-sm" href="/delete/{{_id}}"
          onclick="return confirm('Czy jesteś pewien, że chcesz usunąć ten rekord?');">Usuń</a>
      </td>
    </tr>
    {{/each}}
  </tbody>
</table>
```

### Zadanie 4.2. Modyfikacja struktury aplikacji

Należy zmodyfikować aplikację z zadania 4.1, tak aby była oparta na wzorcu MVC i miała strukturę pokazaną na rys. 4.2.

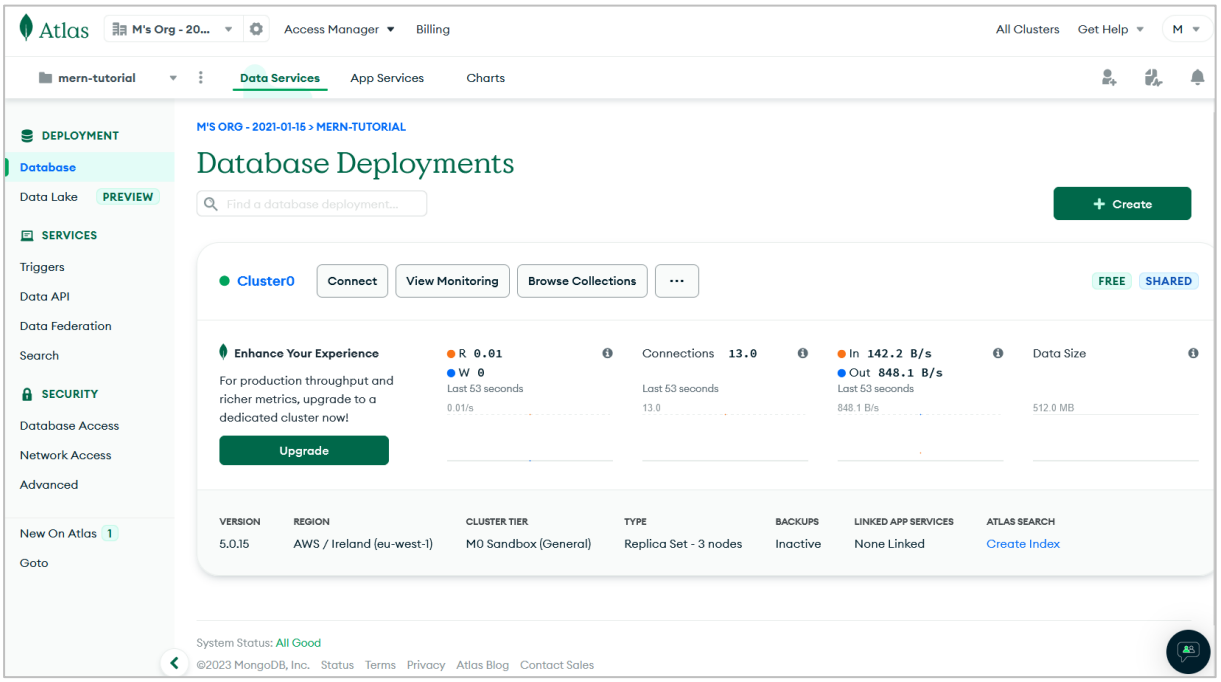
- Plik *db.js* będzie realizował połączenie z bazą danych (*mongoose.connect()*)
- W pliku *Student.js* powinna znajdować się definicja schematu *studentSchema* oraz modelu *Student*
- W folderze *controllers* powinien się znajdować plik *StudentController.js*, który będzie zawierał funkcje obsługujące żądania z listingów 4.3 i 4.4
- Plik szablonu *layout.hbs* oraz pliki widoków *list.hbs* i *addOrEdit.hbs* należy umieścić w folderze *views*




*Rys. 4.2 Struktura projektu aplikacji MVC*

### Zadanie 4.3. Połączenie i praca z bazą danych w chmurze

Należy zmodyfikować zadanie 4.1 i skorzystać z bazy danych w chmurze MongoDB Atlas (https://www.mongodb.com/). W tym celu najpierw należy założyć bezpłatne konto, a następnie utworzyć bazę i użytkownika. W kolejnym etapie należy skopiować ścieżkę do bazy i skonfigurować połączenie w aplikacji.




*Rys. 4.4 Okno platformy MongoDB Atlas*