

# Dokumentowe bazy danych – MongoDB

## Ćwiczenie 2 - zadanie do samodzielnego wykonania

Imię i nazwisko: Krzysztof Solecki

### Materialy:

Książki

Np.

- Shannon Bradshaw, Eoin Brazil, Kristina Chodorow, MongoDB: The Definitive Guide. Powerful and Scalable Data Storage, O'Reilly 2019
- Alex Giamas, Mastering MongoDB 4.x., Pact 2019

Dokumentacja

- <https://www.mongodb.com/docs/manual/reference/program/mongo/>

MongoDB University Courses

- <https://university.mongodb.com/courses/catalog>
- MongoDB Basics ○ <https://university.mongodb.com/courses/M001/about>
- The MongoDB Aggregation Framework ○ <https://university.mongodb.com/courses/M121/about>
- Data Modeling ○ <https://university.mongodb.com/courses/M320/about>

**Yelp Dataset** [www.yelp.com](http://www.yelp.com) - serwis społecznościowy – informacje o

miejscach/lokalach

- restauracje, kluby, hotele itd. (*businesses*),
- użytkownicy piszą recenzje (*reviews*) o miejscach i wystawiają oceny
- użytkownicy odwiedzają te miejsca - "meldują się" (*check-in*)
- Przykładowy zbiór danych zawiera dane z 5 miast: Phoenix, Las Vegas, Madison, Waterloo i Edinburgh.

Kolekcje:

- **42,153** businesses
- **320,002** business attributes
- **31,617** check-in sets
- **252,898** users
- **955,999** edge social graph
- **403,210** tips
- **1,125,458** reviews

## business

```
{
  'type': 'business',
  'business_id': (encrypted business id),
  'name': (business name),
  'neighborhoods': [(hood names)],
  'full_address': (localized address),
  'city': (city),
  'state': (state),
  'latitude': latitude,
  'longitude': longitude,
  'stars': (star rating, rounded to half-stars),
  'review_count': review count,
  'categories': [(localized category names)]
  'open': True / False (corresponds to closed, not business hours),
  'hours': {
    (day_of_week): {
      'open': (HH:MM),
      'close': (HH:MM)
    },
    ...
  },
  'attributes': {
    (attribute_name): (attribute_value),
    ...
  },
}
```

### review

```
{
  'type': 'review',
  'business_id': (encrypted business id),
  'user_id': (encrypted user id),
  'stars': (star rating, rounded to half-stars),
  'text': (review text),
  'date': (date, formatted like '2012-03-14'),
  'votes': {(vote type): (count)},
}
```

### user

```
{
  'type': 'user',
  'user_id': (encrypted user id),
  'name': (first name),
  'review_count': (review count),
  'average_stars': (floating point average, like 4.31),
  'votes': {(vote type): (count)},
  'friends': [(friend user_ids)],
  'elite': [(years_elite)],
  'yelping_since': (date, formatted like '2012-03'),
  'compliments': {
    (compliment_type): (num_compliments_of_this_type),
    ...
  },
  'fans': (num_fans),
}
```

### check-in

```
{
  'type': 'checkin',
  'business_id': (encrypted business id),
  'checkin_info': {
    '0-0': (number of checkins from 00:00 to 01:00 on all Sundays),
    '1-0': (number of checkins from 01:00 to 02:00 on all Sundays),
    ...
    '14-4': (number of checkins from 14:00 to 15:00 on all Thursdays),
    ...
    '23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)
  }, # if there was no checkin for a hour-day block it will not be in the dict
}
```

### tip

```
{
  'type': 'tip',
  'text': (tip text),
  'business_id': (encrypted business id),
  'user_id': (encrypted user id),
  'date': (date, formatted like '2012-03-14'),
  'likes': (count),
}
```

# Zadania

## 1. Operacje wyszukiwania danych

Dla zbioru Yelp wykonaj następujące zapytania

W niektórych przypadkach może być potrzebne wykorzystanie mechanizmu Aggregation Pipeline <https://www.mongodb.com/docs/manual/core/aggregation-pipeline/>

- a) Zwróć dane wszystkich restauracji (kolekcja `business`, pole `categories` musi zawierać wartość `Restaurants`), które są otwarte w poniedziałki (pole `hours`) i mają ocenę co najmniej 4 gwiazdki (pole `stars`). Zapytanie powinno zwracać: nazwę firmy, adres, kategorię, godziny otwarcia i gwiazdki. Posortuj wynik wg nazwy firmy.

Użyłem metody `find()` standardowo do znalezienia dokumentów spełniających zadane kryteria. Składa się ona z 3 elementów: zapytania, widoku zapytania i dodatkowych opcji (np. sortowanie wyniku).

```
// a)
db.getCollection('business').find(
  {
    'categories': { $elemMatch: { $eq: 'Restaurants' } },
    'hours.Monday': { $exists: true },
    'stars': { $gte: 4 }
  },
  {
    'name': true,
    'full_address': true,
    'categories': true,
    'hours': true,
    'stars': true
  },
  {
    sort: [ 'name' ]
  }
);
```

```
1  [
2  {
3      Edit Document
4      "_id": {
5          "$oid": "648ff925b54196a5e80eee64"
6      },
7      "full_address": "67 Nicolson Street\nNewington\nEdinburgh
8      "hours": {
9          "Monday": {
10             "close": "22:00",
11             "open": "10:00"
12         },
13         "Tuesday": {
14             "close": "22:00",
15             "open": "10:00"
16         },
17         "Friday": {
18             "close": "22:00",
19             "open": "10:00"
20         },
21         "Wednesday": {
22             "close": "22:00",
23             "open": "10:00"
24         },
25         "Thursday": {
26             "close": "22:00",
27             "open": "10:00"
28         },
29         "Sunday": {
30             "close": "22:00",
31             "open": "10:00"
32         },
33         "Saturday": {
34             "close": "22:00",
35             "open": "10:00"
36         }
37     },
38     "categories": [
39         "Food",
40         "Desserts",
41         "Coffee & Tea",
```

- b) Ile hoteli znajduje się w każdym mieście. (pole *categories* musi zawierać wartość *Hotels & Travel* lub *Hotels*). Wynik powinien zawierać nazwę miasta, oraz liczbę hoteli. Posortuj wynik malejąco wg liczby hoteli.

Korzystam z metody `aggregate()`, odpowiadającej za agregację zapytania. `$match` to filtr kategorii hoteli, a `$group` to agregator grupy - grupuje każdą firmę po mieście i zlicza liczbę takich hoteli w każdej grupie. Na koniec sortuje malejąco za pomocą `$sort`.

```
db.getCollection('business').aggregate([
  {
    $match: { 'categories': {
      $in: [ 'Hotels & Travel', 'Hotels' ]
    } } },
  {
    $group: {
      '_id': '$city',
      'hotel_count': { $count: {} }
    } },
  { $sort: { 'hotel_count': -1 } }
]);
```

```
Playground Result X

1  [
2    {
3      "_id": "Las Vegas",
4      "hotel_count": 485
5    },
6    {
7      "_id": "Phoenix",
8      "hotel_count": 250
9    },
10   {
11     "_id": "Edinburgh",
12     "hotel_count": 161
13   },
14   {
15     "_id": "Scottsdale",
16     "hotel_count": 122
17   },
18   {
19     "_id": "Madison",
20     "hotel_count": 67
21   },
22   {
23     "_id": "Tempe",
24     "hotel_count": 57
25   },
26   {
27     "_id": "Mesa",
28     "hotel_count": 53
29   },
30   {
31     "_id": "Henderson",
32     "hotel_count": 41
33   },
34   {
35     "_id": "Chandler",
36     "hotel_count": 30
37   },
38   {
39     "_id": "Glendale",
40     "hotel_count": 20
41   },

```

- c) Ile każda firma otrzymała ocen/wskazówek (kolekcja *tip*) w 2012. Wynik powinien zawierać nazwę firmy oraz liczbę ocen/wskazówek. Wynik posortuj według liczby wskazówek (*tip*).

Również korzystam z metody `aggregate()`. W tym zapytaniu filtruję za pomocą wyrażenia regularnego (2012 na początku tekstu). Grupuję i zliczam wskazówki, dołączam kolekcję 'business' w celu wyciągnięcia nazwy firmy, następnie sortuję malejąco po liczbie sugestii i modyfikuję widok, aby zawierał pola: id, nazwę firmy i liczbę sugestii.

```
db.getCollection('tip').aggregate([
  { $match: { 'date': { $regex: /^2012/ } } },
  { $group: {
    '_id': '$business_id',
    'tip_count': { $count: {} }
  } },
  {
    $lookup: {
      from: 'business',
      localField: '_id',
      foreignField: 'business_id',
      as: 'business'
    }
  },
  { $sort: { 'tip_count': -1 } },
  {
    $project: {
      'business_name': { $first: '$business.name' },
      'tip_count': 1
    }
  }
]);
```



{-} Playground Result X

```
1  [
2    {
3      "_id": "jf67Z1pnwElRSXllpQHilg",
4      "tip_count": 1084,
5      "business_name": "McCarran International Airport"
6    },
7    {
8      "_id": "hw0Ne_HTHEAgGF1rAdmR-g",
9      "tip_count": 622,
10     "business_name": "Phoenix Sky Harbor International Airport"
11   },
12   {
13     "_id": "2e2e7WgqU1BnpxmQL5jbfw",
14     "tip_count": 430,
15     "business_name": "Earl of Sandwich"
16   },
17   {
18     "_id": "CsNOg-u_wCuXSt9Z-xU92Q",
19     "tip_count": 374,
20     "business_name": "Las Vegas Athletic Club Southwest"
21   },
22   {
23     "_id": "AtjsjFzalWqJ7S9DUFQ4bw",
24     "tip_count": 351,
25     "business_name": "The Cosmopolitan of Las Vegas"
26   },
27   {
28     "_id": "zt1TpTuJ6y9n551sw9TaEg",
29     "tip_count": 347,
30     "business_name": "Wicked Spoon"
31   },
32   {
33     "_id": "AeKTQBtPRDHLAFL9bzbUnA",
34     "tip_count": 258,
35     "business_name": "Sushi House Goyemon"
36   },
37   {
38     "_id": "FV16IeXJp2W6pngHTz2FAw",
39     "tip_count": 252,
40     "business_name": "Pho Kim Long"
41   },
42 ]
```

- d) Recenzje mogą być oceniane przez innych użytkowników jako *cool*, *funny* lub *useful* (kolekcja *review*, pole *votes*, jedna recenzja może mieć kilka głosów w każdej kategorii). Napisz zapytanie, które zwraca dla każdej z tych kategorii, ile sumarycznie recenzji zostało oznaczonych przez te kategorie (np. recenzja ma kategorię *funny* jeśli co najmniej jedna osoba zagłosowała w ten sposób na daną recenzję)

Zamieniam obiekt 'votes' na tablicę klucz-wartość, następnie rozpakowuję tablicę za pomocą \$unwind, pozbywam się wszystkich dokumentów z liczbą ocen z danej kategorii równą 0.

'count' to liczba wszystkich niezerowych ocen, 'total' to suma wszystkich ocen, tzn. ,że jeśli w jednym dokumencie v będzie miało wartość 2, a w drugim 3 to 'count' zwraca 2, a 'total' 5.

```
db.getCollection('review').aggregate([

  {

    $project: {

      'votes': { $objectToArray: '$votes' }

    }

  },

  { $unwind: '$votes' },

  { $match: { 'votes.v': { $gt: 0 } } },

  {

    $group: {

      '_id': '$votes.k',

      'count': { $count: {} },

      'total': { $sum: '$votes.v' }

    }

  }

]);
```

```
{...} Playground Result X
1  [
2    {
3      "_id": "useful",
4      "count": 549519,
5      "total": 1274331
6    },
7    {
8      "_id": "cool",
9      "count": 346519,
10     "total": 735341
11   },
12   {
13     "_id": "funny",
14     "count": 269256,
15     "total": 590956
16   }
17 ]
```

- e) Zwróć dane wszystkich użytkowników (kolekcja *user*), którzy nie mają ani jednego pozytywnego głosu (pole *votes*) z kategorii (*funny* lub *useful*), wynik posortuj alfabetycznie według nazwy użytkownika.

```
db.getCollection('user').aggregate([
  {
    $match: {
      'votes.funny': 0,
      'votes.useful': 0,
      'type': 'user'
    }
  },
  { $sort: { 'name': 1 } }
]);
```

```
{...} Playground Result X
1  [
2    {
3      "_id": {
4        "$oid": "648ffe2a703e02d832616e52"
5      },
6      "yelping_since": "2009-08",
7      "votes": {
8        "funny": 0,
9        "useful": 0,
10       "cool": 0
11     },
12     "review_count": 1,
13     "name": " Bernard",
14     "user_id": "xP3SPgfgW2vc5Zj5uV8SEA",
15     "friends": [],
16     "fans": 0,
17     "average_stars": 5,
18     "type": "user",
19     "compliments": {},
20     "elite": []
21   },
22   {
23     "_id": {
24       "$oid": "648ffe41703e02d83263d868"
25     },
26     "yelping_since": "2013-03",
27     "votes": {
28       "funny": 0,
29       "useful": 0,
30       "cool": 0
31     },
32     "review_count": 1,
33     "name": ",Maria",
34     "user_id": "Os5f3TNpM7_A8IDNEdPX2g",
35     "friends": [],
36     "fans": 0,
37     "average_stars": 5,
38     "type": "user",
39     "compliments": {},
40     "elite": []
41   }
42 ]
```

- f) Wyznacz, jaką średnią ocenę uzyskała każda firma na podstawie wszystkich recenzji (kolekcja *review*, pole *stars*). Ogranicz do firm, które uzyskały średnią powyżej 3 gwiazdek.

przypadek 1: Wynik powinien zawierać id firmy oraz średnią ocenę. Posortuj wynik wg id firmy.

przypadek 2: Wynik powinien zawierać nazwę firmy oraz średnią ocenę. Posortuj wynik wg nazwy firmy.

Najpierw grupuję po business\_id, agregując dodatkowo średnią opinii, wyświetlam tylko opinie, które posiadają średnią większą niż 3, a potem sortuję wyniki po id firmy (\_id).

```
//przypadek 1
db.getCollection('review').aggregate([
  {
    $group: {
      '_id': '$business_id',
      'stars_mean': { $avg: '$stars' }
    }
  },
  { $match: { 'stars_mean': { $gt: 3 } } },
  { $sort: { '_id': 1 } },
]);
```

```
Playground Result X
1  [
2    {
3      "_id": "--1emggGHgoG6ipd_RMb-g",
4      "stars_mean": 3.75
5    },
6    {
7      "_id": "--5jkZ3-nUPZxUvtcbr8Uw",
8      "stars_mean": 4.615384615384615
9    },
10   {
11     "_id": "--BlvDO_RG2yElKu9XA1_g",
12     "stars_mean": 3.9696969696969697
13   },
14   {
15     "_id": "--Dl2rW_x08GuYBomlg9zw",
16     "stars_mean": 4.166666666666667
17   },
18   {
19     "_id": "--Ol5mVSMaW8ExtmWRUmKA",
20     "stars_mean": 5
21   },
22   {
23     "_id": "--XBxRlD92RaV6TyUnP80w",
24     "stars_mean": 3.6666666666666665
25   },
26   {
27     "_id": "--jFTZmywe7StuZ2hEjxyA",
28     "stars_mean": 4.333333333333333
29   },
30   {
31     "_id": "--m1g9P1wxNblrLANfVqlA",
32     "stars_mean": 4.25
33   },
34   {
35     "_id": "--qeSYxyn62mMjWvznNTdg",
36     "stars_mean": 4
37   },
38   {
39     "_id": "--05qMwbhAtrD6EiV-UElPg",
40     "stars_mean": 5
41   }
42 ]
```

Do zapytania z przypadku 1 dodałem relację z kolekcją business, włączając tylko nazwę firmy za pomocą pipeline. W tym zapytaniu czas jego wykonania jest długi. Spowodowane jest to sortowaniem i wyciąganiem danych spoza bazowej kolekcji za pomocą \$lookup i strukturą bazy Yelp.

```
//przypadek 2
```

```
db.getCollection('review').aggregate([

  {

    $group: {

      '_id': '$business_id',

      'stars_mean': { $avg: '$stars' }

    }

  },

  { $match: { 'stars_mean': { $gt: 3 } } },

  {

    $lookup: {

      from: 'business',

      localField: '_id',

      foreignField: 'business_id',

      pipeline: [{ $group: { '_id': '$name' } }],

      as: 'business'

    }

  },

  {

    $project: {

      '_id': 0,

      'business_name': { $first: '$business._id' },

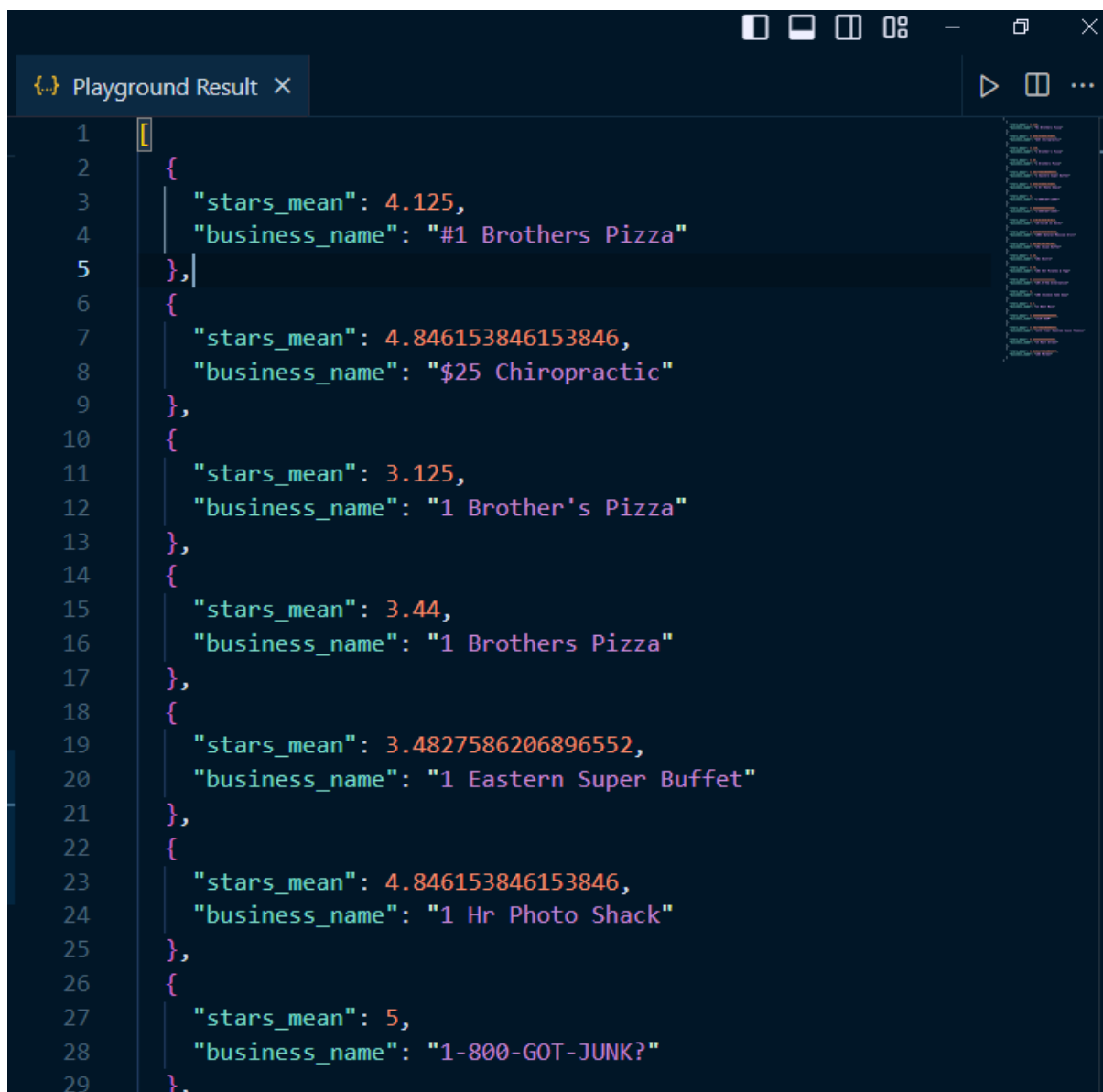
      'stars_mean': 1

    }

  },

  { $sort: { 'business_name': 1 } }

]);
```



```
1  [
2  {
3    "stars_mean": 4.125,
4    "business_name": "#1 Brothers Pizza"
5  },
6  {
7    "stars_mean": 4.846153846153846,
8    "business_name": "$25 Chiropractic"
9  },
10 {
11   "stars_mean": 3.125,
12   "business_name": "1 Brother's Pizza"
13 },
14 {
15   "stars_mean": 3.44,
16   "business_name": "1 Brothers Pizza"
17 },
18 {
19   "stars_mean": 3.4827586206896552,
20   "business_name": "1 Eastern Super Buffet"
21 },
22 {
23   "stars_mean": 4.846153846153846,
24   "business_name": "1 Hr Photo Shack"
25 },
26 {
27   "stars_mean": 5,
28   "business_name": "1-800-GOT-JUNK?"
29 },
30 ]
```

W sprawozdaniu należy umieścić zrzuty ekranów (z kodem poleceń oraz z uzyskanymi wynikami). Dodatkowo należy dołączyć plik tekstowy (najlepiej z rozszerzeniem .js) zawierający kod poleceń

## 2. Modelowanie danych

- Zaproponuj strukturę bazy danych dla wybranego/przykładowego zagadnienia/problemu
- Należy wybrać jedno zagadnienie/problem (A lub B) Przykład A
- Wykładowcy, przedmioty, studenci, oceny
- Wykładowcy prowadzą zajęcia z poszczególnych przedmiotów
- Studenci uczęszczają na zajęcia
- Wykładowcy wystawiają oceny studentom
- Studenci oceniają zajęcia



#### Przykład B

- Firmy, wycieczki, osoby
  - Firmy organizują wycieczki
  - Osoby rezerwują miejsca/wykupują bilety
  - Osoby oceniają wycieczki
- a) Warto zaproponować/rozważyć różne warianty struktury bazy danych i dokumentów w poszczególnych kolekcjach oraz przeprowadzić dyskusję każdego wariantu (wskazać wady i zalety każdego z wariantów)
- b) Kolekcje należy wypełnić przykładowymi danymi
- c) W kontekście zaprezentowania wad/zalet należy zaprezentować kilka przykładów/zapytań/zadań/operacji oraz dla których dedykowany jest dany wariantów

### Propozycja modelu danych dla problemu A:

Po zastanowieniu zdecydowałem, że baza będzie składać się z 3 kolekcji (wykładowcy, przedmioty, studenci). Alternatywnym rozwiązaniem mogłoby być trzymanie wszystkiego w jednej kolekcji, co uprościłoby wykonywanie operacji na tej bazie i zapobiegło referencji do innych kolekcji. Mimo to uważam, że rozdzielenie danych na 3 kolekcje zwiększy czytelność bazy danych, ułatwi korzystanie z niej dzięki logicznemu pogrupowaniu danych. Jedynym minusem tego rozwiązania będzie konieczność referencji do innych kolekcji (odwoływanie się po ID). Kolekcja “wykładowcy” oraz “studenci” jest oczywistym wyborem. Poza nimi zaproponowałem utworzenie kolekcji “przedmioty”, chociaż mogłem w pierwszych dwóch trzymać informację o przedmiocie. Wybrałem takie rozwiązanie, ponieważ informacji o przedmiocie jest sporo i nie każda z nich jest potrzebna w każdej z tych dwóch kolekcji, a może zdarzyć się operacja, która będzie wymagała więcej informacji, jednakże najczęściej wymagane są tylko niektóre dane, więc wybrałem najważniejsze informacje do każdej z nich oraz umieściłem referencję do kolekcji “przedmioty” w razie konieczności szczegółowych danych na temat przedmiotu. Poniżej przedstawiam strukturę dokumentu każdej z kolekcji i krótki opis wybranego rozwiązania problemu.

#### a) Wykładowcy (lecturers):

Kolekcja przechowuje dane na temat prowadzących. W polu *subjects* umieszczone są dwie moim zdaniem najczęściej potrzebne informacje na ich temat oraz referencja do dokumentu kolekcji “przedmioty”, jeśli potrzebne byłyby szczegółowe informacje.

#### Struktura dokumentu:

```
{
  "_id": ObjectId(),
  "faculty": [(faculty shortcut)],
  "degree": [(degree name)],
  "fname": [(first name)],
  "lname": [(last name)],
```

```

    "age": (age),
    "address": {
        "street": (street name),
        "city": (city name),
        "country": (country name),
        "zip": (zip code like '11-321'),
        "homeNumber": (home number)
    },
    "subjects": [
        {
            "subjectName": (subject name),
            "role": (lecturer or assistant),
            "subjectID": ObjectId()
        }
    ]
}

```

### Przykładowy dokument:

```

{
  "_id": {
    "$oid": "648f162c2ac76f648517fec0"
  },
  "faculty": [
    "IET",
    "EAIIB",
    "WIMiR"
  ],
  "degree": [
    "inż",
    "mgr",
    "dr",
    "prof"
  ],
  "fname": "Jan",
  "lname": "Kowalski",
  "age": 39,
  "address": {
    "street": "Kwiatowa",
    "city": "Cracow",
    "country": "Poland",
    "zip": "30-901",
    "homeNumber": 121
  },
  "subjects": [
    {
      "subjectName": "Introduction to Computer Science",
      "role": "lecturer",

```

```

        "subjectID": {
            "$oid": "648f162c2ac76f648517fec1"
        }
    },
    {
        "subjectName": "Algorithms and Data Structures",
        "role": "assistant",
        "subjectID": {
            "$oid": "648f162c2ac76f648517fec2"
        }
    }
]
}

```

#### b) Studenci (students):

Kolekcja przechowuje dane na temat studentów. W polu *subjects* umieszczona jest jedynie nazwa przedmiotu (która moim zdaniem jest najczęściej potrzebna patrząc na tę kolekcję) i referencja jeśli wymagane są szczegółowe informacje o przedmiocie. Podobnie w przypadku pola *grades*.

#### Struktura dokumentu:

```

{
    "_id": ObjectId(),
    "indexNumber": (index number),
    "semester": (semester number),
    "faculty": (faculty shortcut),
    "fname": (first name),
    "lname": (last name),
    "age": (age),
    "address": {
        "street": (street name),
        "city": (city name),
        "country": (country name),
        "zip": (zip code),
        "homeNumber": (home number)
    },
    "subjects": [
        {
            "subjectName": (subject name),
            "subjectID": ObjectId()
        }
    ],
}

```

```

    "grades": [
      {
        "subjectName": (subject name),
        "subjectGrades": [(grade)],
        "subjectID": ObjectId()
      }
    ]
  }
}

```

### Przykładowy dokument:

```

{
  "_id": {
    "$oid": "648f1c0d5153300871e2e8e3"
  },
  "indexNumber": 403231,
  "semester": 4,
  "faculty": "IET",
  "fname": "Krzysztof",
  "lname": "Solecki",
  "age": 23,
  "address": {
    "street": "Kwiatkowa",
    "city": "Cracow",
    "country": "Poland",
    "zip": "30-901",
    "homeNumber": 121
  },
  "subjects": [
    {
      "subjectName": "Discrete Mathematics",
      "subjectID": {
        "$oid": "648f1c0d5153300871e2e8e4"
      }
    },
    {
      "subjectName": "Algorithms and Data Structures",
      "subjectID": {
        "$oid": "648f1c0d5153300871e2e8e5"
      }
    }
  ]
}

```

```
},
{
  "subjectName": "Programming in C",
  "subjectID": {
    "$oid": "648f1c0d5153300871e2e8e6"
  }
}
],
"grades": [
  {
    "subjectName": "Discrete Mathematics",
    "subjectGrades": [
      5,
      4.5,
      4,
      5,
      4
    ],
    "subjectID": {
      "$oid": "648f1c0d5153300871e2e8e7"
    }
  },
  {
    "subjectName": "Algorithms and Data Structures",
    "subjectGrades": [
      3,
      4.5,
      4,
      5,
      4
    ],
    "subjectID": {
      "$oid": "648f1c0d5153300871e2e8e8"
    }
  },
  {
    "subjectName": "Programming in C",
    "subjectGrades": [
      5,
      4.5,
      4,
      5,
```

```

    4
    ],
    "subjectID": {
      "$oid": "648f1c0d5153300871e2e8e9"
    }
  }
]
}

```

### c) Przedmioty (subjects):

Kolekcja przechowuje szczegółowe dane na temat przedmiotów. W polu *reviews* (oceny przedmiotu) znajduje się referencja do konkretnego studenta, jednak najistotniejsze jest jego imię nazwisko oraz sama treść. Jeśli wymagane są dodatkowe informacje wtedy należy odwołać się do kolekcji studentów poprzez tę referencję. Oceny studentów można by przechowywać w kolekcji students jednak logicznie jest mieć tę informację w dokumencie reprezentującym dany przedmiot.

### Struktura dokumentu:

```

{
  "_id": ObjectId(),
  "name": (subject name),
  "fieldOfStudy": (field of study),
  "mandatory": (obligatory or elective),
  "lectureLanguage": (language),
  "faculty": (faculty shortcut),
  "form of verification": (exam / assessment),
  "numberOfHours": {
    "noLectures": (number of lectures),
    "noExercises": (number of exercises),
    "noLaboratories": (number of laboratories)
  },
  "assistant": {

```

```

    "name": (first name and last name),

    "id": ObjectId()

},

"lecturers": [

{

    "name": (first name and last name),

    "id": ObjectId()

}

],

"literature": [(book info)],

"ECTS": (number of ECTS points)

"goals": (goal of the subject),

"studyContent": (study content),

"additionalInformation": (additional information)

"reviews": [

{

    "studentID": ObjectId(),

    "studentName": (first name and last name),

    "content": (review content)

}

]

}

```

### Przykładowy dokument:

```

{

  "_id": {

    "$oid": "648f6749c27ba66abe6df672"

  },


```

```
"name": "Discrete Mathematics",
"fieldOfStudy": "Computer Science",
"mandatory": "obligatory",
"lectureLanguage": "Polish",
"faculty": "IET",
"form of verification": "exam",
"numberOfHours": {
  "noLectures": 30,
  "noExercises": 30,
  "noLaboratories": 0
},
"assistant": {
  "name": "Jan Kowalski",
  "id": {
    "$oid": "648f6749c27ba66abe6df673"
  }
},
"lecturers": [
  {
    "name": "Jan Kowalski",
    "id": {
      "$oid": "648f6749c27ba66abe6df674"
    }
  }
],
"literature": [
  "Discrete Mathematics",
  "Discrete Mathematics and its Applications"
],
```



```
"ECTS": "5",

"goals": "The aim of the course is to introduce students to the basic
concepts of discrete mathematics and to develop their skills in the use
of these concepts in solving problems related to computer science.",

"studyContent": "1. Set theory. 2. Relations and functions. 3.
Combinatorics. 4. Graph theory. 5. Boolean algebra. 6. Logic. 7. Number
theory.",

"additionalInformation": "The course is a continuation of the course
'Introduction to Computer Science'.",

"reviews": [

  {

    "studentID": {

      "$oid": "648f6749c27ba66abe6df675"

    },

    "studentName": "Krzysztof Solecki",

    "content": "Very interesting subject, I recommend it to
    everyone."

  }

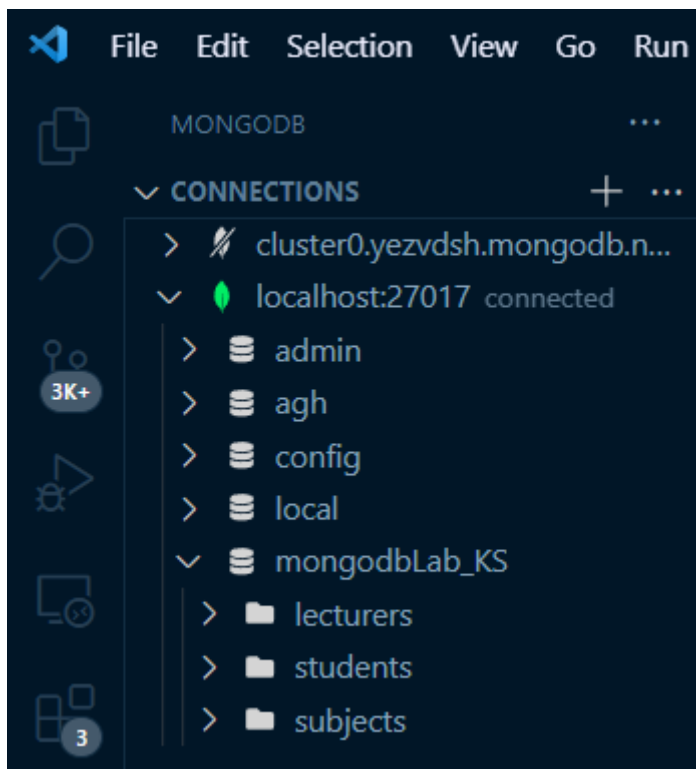
]

}
```

## Utworzenie bazy danych i wypełnienie kolekcji przykładowymi danymi:

Wprowadziłem kilka przykładowych przedmiotów, studentów oraz wykładowców. Poniżej widoczny jest przykładowych poleceń insertOne() oraz find() na każdej z nich.

### Utworzenie bazy danych:



Wypełnienie danymi:

```
db.getCollection('lecturers').insertOne(  
{  
  "_id": ObjectId(),  
  "faculty": ["IET", "EAIIB", "WIMiR"],  
  "degree": ["inż", "mgr", "dr", "prof"],  
  "fname": "Jan",  
  "lname": "Kowalski",  
  "age": 39,  
  "address": {  
    "street": "Kwiatowa",  
    "city": "Cracow",  
    "country": "Poland",  
    "zip": "30-901",  
    "homeNumber": 121
```

```

    },
    "subjects": [
        {
            "subjectName": "Introduction to Computer Science",
            "role": "lecturer",
            "subjectID": ObjectId()
        },
        {
            "subjectName": "Algorithms and Data Structures",
            "role": "assistant",
            "subjectID": ObjectId()
        }
    ]
}
}))

```

```

db.getCollection('students').insertOne(
{

```

```

    "_id": ObjectId(),
    "indexNumber": 403231,
    "semester": 4,
    "faculty": "IET",
    "fname": "Krzysztof",
    "lname": "Solecki",
    "age": 23,
    "address": {
        "street": "Kwiatkova",
        "city": "Cracow",
        "country": "Poland",
        "zip": "30-901",
        "homeNumber": 121
    }
}

```

```
},
"subjects": [
  {
    "subjectName": "Discrete Mathematics",
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Algorithms and Data Structures",
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Programming in C",
    "subjectID": ObjectId()
  }
],
"grades": [
  {
    "subjectName": "Discrete Mathematics",
    "subjectGrades": [5.0, 4.5, 4.0, 5.0, 4.0],
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Algorithms and Data Structures",
    "subjectGrades": [3.0, 4.5, 4.0, 5.0, 4.0],
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Programming in C",
    "subjectGrades": [5.0, 4.5, 4.0, 5.0, 4.0],
```

```

        "subjectID": ObjectId()

    }

]

}

)

db.getCollection('subjects').insertOne(
{
    "_id": ObjectId(),
    "name": "Discrete Mathematics",
    "fieldOfStudy": "Computer Science",
    "mandatory": "obligatory",
    "lectureLanguage": "Polish",
    "faculty": "IET",
    "form of verification": "exam",
    "numberOfHours": {
        "noLectures": 30,
        "noExercises": 30,
        "noLaboratories": 0
    },
    "assistant": {
        "name": "Jan Kowalski",
        "id": ObjectId()
    },
    "lecturers": [
        {
            "name": "Jan Kowalski",
            "id": ObjectId()
        }
    ]
}
)

```

```

],

"literature": ["Discrete Mathematics", "Discrete Mathematics and
its Applications"],

"ECTS": "5",

"goals": "The aim of the course is to introduce students to the
basic concepts of discrete mathematics and to develop their skills
in the use of these concepts in solving problems related to
computer science.",

"studyContent": "1. Set theory. 2. Relations and functions. 3.
Combinatorics. 4. Graph theory. 5. Boolean algebra. 6. Logic. 7.
Number theory.",

"additionalInformation": "The course is a continuation of the
course 'Introduction to Computer Science'.",

"reviews": [
{
  "studentID": ObjectId(),
  "studentName": "Krzysztof Solecki",
  "content": "Very interesting subject, I recommend it to
everyone.",
}
]
})

```

```

db.getCollection('lecturers').insertOne(
{
  "_id": ObjectId(),
  "faculty": ["IET", "EAIIB"],
  "degree": ["inż", "mgr"],
  "fname": "Marian",

```

```
    "lname": "Nowak",

    "age": 43,

    "address": {

        "street": "Pszczelna",

        "city": "Cracow",

        "country": "Poland",

        "zip": "30-920",

        "homeNumber": 124

    },

    "subjects": [

        {

            "subjectName": "Haskell Programming",

            "role": "assistant",

            "subjectID": ObjectId()

        },

        {

            "subjectName": "Advanced Machine Learning",

            "role": "assistant",

            "subjectID": ObjectId()

        }

    ]

})
```

```
db.getCollection('lecturers').insertOne(

{

    "_id": ObjectId(),

    "faculty": ["IET", "EAIIB"],

    "degree": ["inż", "mgr", "dr"],

    "fname": "Faustyna",
```

```
"lname": "Kowalska",

"age": 41,

"address": {

    "street": "Opolska",

    "city": "Cracow",

    "country": "Poland",

    "zip": "30-920",

    "homeNumber": 23

},

"subjects": [

    {

        "subjectName": "Numerical Methods",

        "role": "lecturer",

        "subjectID": ObjectId()

    },

    {

        "subjectName": "Probability Theory and Statistics",

        "role": "assistant",

        "subjectID": ObjectId()

    },

    {

        "subjectName": "Operating Systems",

        "role": "assistant",

        "subjectID": ObjectId()

    }

]

})
```

```
db.getCollection('students').insertOne(
```



```
{  
  
  "_id": ObjectId(),  
  
  "indexNumber": 402122,  
  
  "semester": 4,  
  
  "faculty": "IET",  
  
  "fname": "Bartosz",  
  
  "lname": "Walczak",  
  
  "age": 22,  
  
  "address": {  
  
    "street": "Akacjowa",  
  
    "city": "Cracow",  
  
    "country": "Poland",  
  
    "zip": "30-901",  
  
    "homeNumber": 23  
  
  },  
  
  "subjects": [  
  
    {  
  
      "subjectName": "Operating Systems",  
  
      "subjectID": ObjectId()  
  
    },  
  
    {  
  
      "subjectName": "Numerical Methods",  
  
      "subjectID": ObjectId()  
  
    },  
  
    {  
  
      "subjectName": "Automata and Formal Languages",  
  
      "subjectID": ObjectId()  
  
    }  
  
  ],  
  
}
```

```

"grades": [
    {
        "subjectName": "Operating Systems",
        "subjectGrades": [3.0, 3.5, 4.0, 5.0, 4.0],
        "subjectID": ObjectId()
    },
    {
        "subjectName": "Numerical Methods",
        "subjectGrades": [3.0, 4.5, 4.0, 5.0, 4.0],
        "subjectID": ObjectId()
    },
    {
        "subjectName": "Automata and Formal Languages",
        "subjectGrades": [4.0, 2.0, 4.0, 5.0, 4.0],
        "subjectID": ObjectId()
    }
]
}
)

```

```

db.getCollection('students').insertOne(

```

```

{
    "_id": ObjectId(),
    "indexNumber": 408122,
    "semester": 1,
    "faculty": "IET",
    "fname": "Anna",
    "lname": "Rogowska",
    "age": 21,

```

```
"address": {
  "street": "Głogowska",
  "city": "Cracow",
  "country": "Poland",
  "zip": "30-901",
  "homeNumber": 244
},
"subjects": [
  {
    "subjectName": "Algebra",
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Introduction to Computer Science",
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Interpersonal Communication",
    "subjectID": ObjectId()
  }
],
"grades": [
  {
    "subjectName": "Algebra",
    "subjectGrades": [3.0, 3.5, 4.0, 5.0, 4.0],
    "subjectID": ObjectId()
  },
  {
    "subjectName": "Introduction to Computer Science",
```

```

        "subjectGrades": [3.0, 4.5, 4.0, 5.0, 4.0],
        "subjectID": ObjectId()
    },
    {
        "subjectName": "Interpersonal Communication",
        "subjectGrades": [4.0, 2.0, 4.0, 5.0, 4.0],
        "subjectID": ObjectId()
    }
]
}
)

```

```

db.getCollection('subjects').insertOne(
    {
        "_id": ObjectId(),
        "name": "Algorithms and Data Structures",
        "fieldOfStudy": "Computer Science",
        "mandatory": "obligatory",
        "lectureLanguage": "Polish",
        "faculty": "IET",
        "form of verification": "exam",
        "numberOfHours": {
            "noLectures": 30,
            "noExercises": 30,
            "noLaboratories": 0
        },
        "assistant": {
            "name": "Piotr Horban",
            "id": ObjectId()
        }
    }
)

```

```

    },
    "lecturers": [
      {
        "name": "Daniel Lewandowski",
        "id": ObjectId()
      }
    ],
    "literature": ["Cormen, Leiserson, Rivest, Stein: Introduction
to Algorithms, MIT Press, 2009", "T. H. Cormen, C. E.
Leiserson, R. L. Rivest, C. Stein: Wprowadzenie do algorytmów,
Wydawnictwo Naukowo-Techniczne, 2001"],
    "ECTS": "6",
    "goals": "The aim of the course is to acquaint students with
the basic algorithms and data structures used in computer
science. The course is a continuation of the course
Introduction to Computer Science.",
    "reviews": [
      {
        "studentID": ObjectId(),
        "studentName": "Bartosz Walczak",
        "content": "Hard but interesting subject. I recommend it to
everyone who wants to learn something about algorithms and
data structures.",
      }
    ]
  })

db.getCollection('subjects').insertOne(
  {

```

```
"_id": ObjectId(),

"name": "Haskell Programming",

"fieldOfStudy": "Computer Science",

"mandatory": "elective",

"lectureLanguage": "English",

"faculty": "WIMiR",

"form of verification": "assessment",

"numberOfHours": {

    "noLectures": 14,

    "noExercises": 0,

    "noLaboratories": 30

},

"assistant": {

    "name": "Szymon Chojnacki",

    "id": ObjectId()

},

"lecturers": [

{

    "name": "Urszula Kaczmarek",

    "id": ObjectId()

}

],

"literature": ["Graham Hutton: Programming in Haskell,

Cambridge University Press, 2007", "Graham Hutton:

Programowanie w Haskellu, Helion, 2016"],

"ECTS": "3",

"goals": "The aim of the course is to acquire knowledge about

functional programming in Haskell language. The course is a

continuation of the course Introduction to Computer Science.",
```

```

    "reviews": [
      {
        "studentID": ObjectId(),
        "studentName": "Anna Rogowska",
        "content": "Boredom. I don't recommend it to anyone.",
      }
    ]
  }
)

```

## Przykładowe operacje na bazie danych:

- a) Zapis studenta na przedmiot:

W poniższym przykładzie zapis studenta o numerze indeksu 408122 na przedmiot "Haskell Programming".

```

db.getCollection('students').updateOne(
  {
    "indexNumber": 408122,
  },
  {
    $push: {
      "subjects": {
        "subjectName": "Haskell Programming",
        "subjectID": ObjectId()
      }
    }
  }
)

db.getCollection('students').updateOne(
  {
    "indexNumber": 408122,
  },

```

```
{
  $push: {
    "grades": {
      "subjectName": "Haskell Programming",
      "subjectGrades": [],
      "subjectID": ObjectId()
    }
  }
}
)
```

Playground Result X

```
1 {
2   "acknowledged": true,
3   "insertedId": null,
4   "matchedCount": 1,
5   "modifiedCount": 1,
6   "upsertedCount": 0
7 }
```

b) Przyznanie prowadzenia zajęć danemu prowadzącemu:

Przykładowo Urszula Kaczmarek będzie prowadzić przedmiot Haskell Programming

```
db.getCollection('subjects').updateOne(
  {
    "name": "Haskell Programming",
  },
  {
    $push: {
      "lecturers": {
        "name": "Urszula Kaczmarek",
        "id": ObjectId()
      }
    }
  }
)
```



```
{ } Playground Result X
1 {
2   "acknowledged": true,
3   "insertedId": null,
4   "matchedCount": 1,
5   "modifiedCount": 1,
6   "upsertedCount": 0
7 }
```

c) Przyznanie koordynowania przedmiotu danemu prowadzącemu:

Przykładowo Szymon Chojnacki będzie koordynatorem przedmiotu Haskell Programming.

```
db.getCollection('subjects').updateOne(
  {
    "name": "Haskell Programming",
  },
  {
    $set: {
      "assistant": {
        "name": "Szymon Chojnacki",
        "id": ObjectId()
      }
    }
  }
)
```

```
{...} Playground Result X
1  {
2    "acknowledged": true,
3    "insertedId": null,
4    "matchedCount": 1,
5    "modifiedCount": 1,
6    "upsertedCount": 0
7  }
```

d) Wykładowca wystawia ocenę:

Przykładowo prowadzący wystawia ocenę studentowi o indeksie 408122 z przedmiotu Haskell Programming.

```
db.getCollection('students').update(
  {
    "grades.subjectName": "Haskell Programming",
    "indexNumber": 408122
  },
  {
    $push: {
      "grades.$.subjectGrades": 5.0
    }
  }
)
```

```
{...} Playground Result X
1  {
2    "acknowledged": true,
3    "insertedId": null,
4    "matchedCount": 1,
5    "modifiedCount": 1,
6    "upsertedCount": 0
7  }
```

e) Ocena przedmiotu przez studenta:

Przykładowo student ocenia przedmiot Discrete Mathematics.

```
db.getCollection('subjects').updateOne(
  (
    { "name": "Discrete Mathematics", "faculty": "IET" }
  ),
  {
    $push: {
      "reviews": {
        "studentID": ObjectId(),
        "studentName": "Bartosz Walczak",
        "content": "Interesting and useful subject."
      }
    }
  }
)
```

```
{...} Playground Result X
1  {
2    "acknowledged": true,
3    "insertedId": null,
4    "matchedCount": 1,
5    "modifiedCount": 1,
6    "upsertedCount": 0
7  }
```

W sprawozdaniu należy zamieścić przykładowe dokumenty w formacie JSON ( pkt a) i b)), oraz kod zapytań/operacji (pkt c)), wraz z odpowiednim komentarzem opisującym strukturę dokumentów oraz polecenia ilustrujące wykonanie przykładowych operacji na danych

Do sprawozdania należy kompletny zrzut wykonanych/przygotowanych baz danych (taki zrzut można wykonać np. za pomocą poleceń mongoexport, mongodump ...) oraz plik z kodem operacji zapytań (załącznik powinien mieć format zip).

**Punktacja za zadanie (razem 2pkt)**