

# Bazy danych – NoSQL MongoDB – zadania

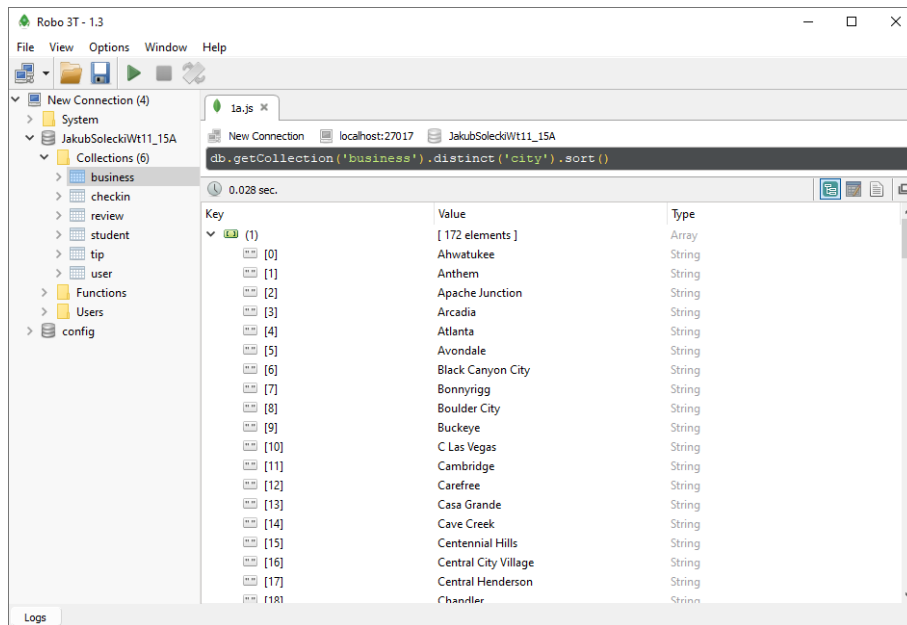
Jakub Solecki

Tydzień A, godz. lab.: 11:15

1. Wykorzystując bazę danych **yelp dataset** wykonaj zapytanie i komendy MongoDB, aby uzyskać następujące rezultaty:

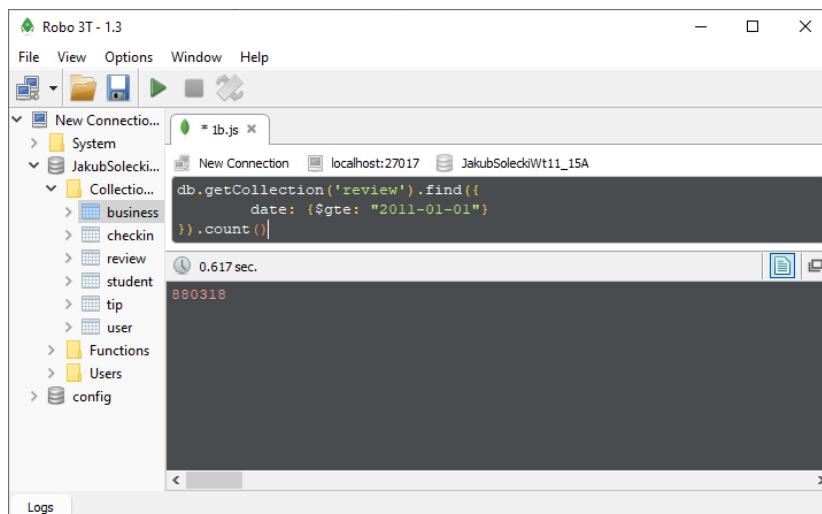
- a. Zwróć bez powtórzeń wszystkie nazwy miast w których znajdują się firmy (business). Wynik posortuj na podstawie nazwy miasta alfabetycznie.

```
db.getCollection('business').distinct('city').sort()
```



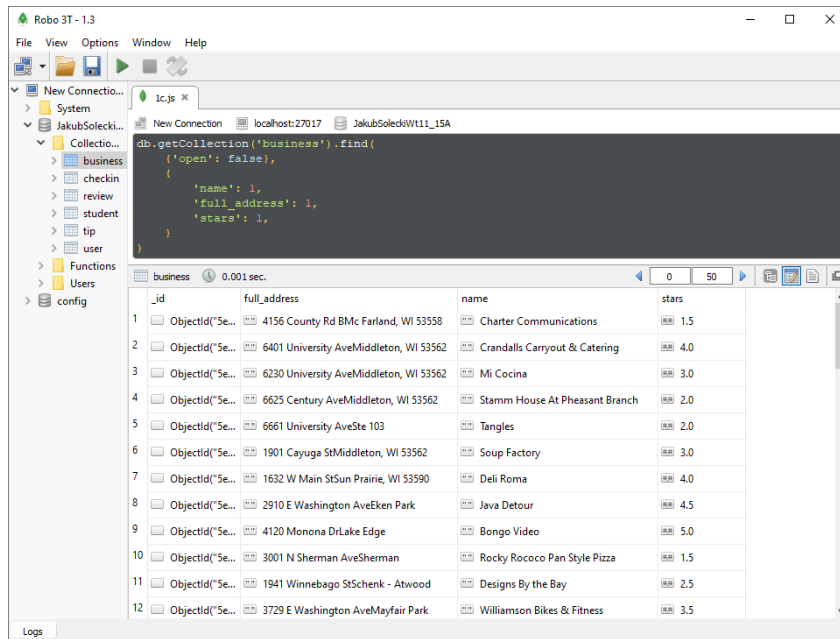
- b. Zwróć liczbę wszystkich recenzji, które pojawiły się po 2011 roku (włącznie).

```
db.getCollection('review').find({
  date: {$gte: "2011-01-01"}
}).count()
```



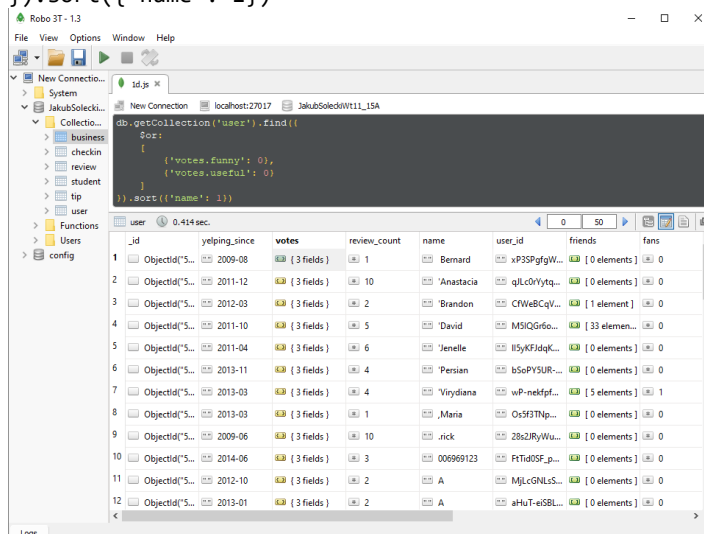
- c. Zwróć dane wszystkich zamkniętych (open) firm (business) z pól: nazwa, adres, gwiazdki (stars).

```
db.getCollection('business').find(
  {'open': false},
  {
    'name': 1,
    'full_address': 1,
    'stars': 1,
  }
)
```



- d. Zwróć dane wszystkich użytkowników (user), którzy nie uzyskali ani jednego pozytywnego głosu z kategorii (funny lub useful), wynik posortuj alfabetycznie według imienia użytkownika.

```
db.getCollection('user').find({
  $or: [
    {'votes.funny': 0},
    {'votes.useful': 0}
  ]
}).sort({'name': 1})
```



- e. Określ, ile każde przedsiębiorstwo otrzymało wskazówek/napiwków (tip) w 2012. Wynik posortuj alfabetycznie według liczby (tip).

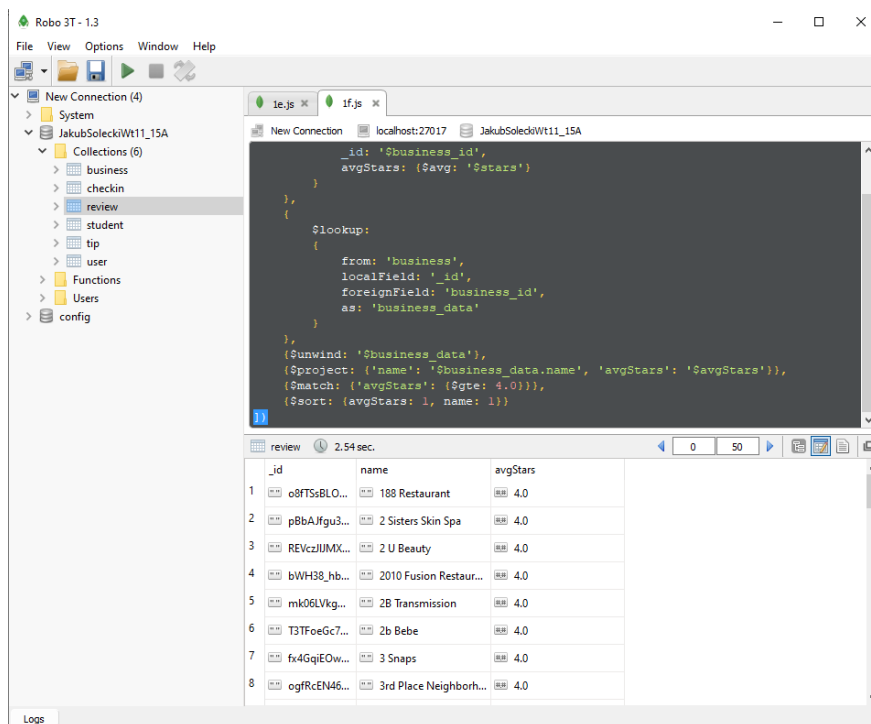
```
db.getCollection('tip').aggregate([
  {$match: {'date': '/2012/'}},
  {$group:
    {
      _id: '$business_id',
      count: {$sum: 1}
    }
  },
  {
    $lookup:
    {
      from: 'business',
      localField: '_id',
      foreignField: 'business_id',
      as: 'business_data'
    }
  },
  {$unwind: '$business_data'},
  {$project: {'name': '$business_data.name', 'count': '$count'}},
  {$sort: {count: 1, name: 1}}
])
```

The screenshot shows the Robo 3T 1.3 application window. On the left, a tree view shows the database structure with collections like 'business', 'checkin', 'review', 'student', 'tip', 'user', 'Functions', 'Users', and 'config'. The 'tip' collection is selected. The main window displays a MongoDB aggregation query in a dark-themed editor. Below the editor, a table shows the results of the query, sorted by count and then name. The table has three columns: '\_id', 'name', and 'count'. The results show 8 entries, all with a count of 1.0.

_id	name	count
kkW8V7ht...	#1 Brothers Pizza	1.0
qK_nxiXCxj...	1 Brother's Pizza	1.0
BpMdiKy1P...	1 Eastern Super Buffet	1.0
oZ1k1UxA2...	1 Hr Photo Shack	1.0
tvBC1FNxX...	101 Asian Buffet	1.0
PuuO18bxs...	1855 Saloon and Grill	1.0
o8Tss8LO...	188 Restaurant	1.0
tlA7UGjDPq...	1st Emergency Pet C...	1.0

- f. Wyznacz, jaką średnią ocen (stars) uzyskała każda firma (business) na podstawie wszystkich recenzji. Wynik ogranicz do recenzji, które uzyskały min 4.0 gwiazdki.

```
db.getCollection('review').aggregate([
  {$group:
    {
      _id: '$business_id',
      avgStars: {$avg: '$stars'}
    }
  },
  {
    $lookup:
    {
      from: 'business',
      localField: '_id',
      foreignField: 'business_id',
      as: 'business_data'
    }
  },
  {$unwind: '$business_data'},
  {$project: {'name': '$business_data.name', 'avgStars': '$avgStars'}},
  {$match: {'avgStars': {$gte: 4.0}}},
  {$sort: {avgStars: 1, name: 1}}
])
```



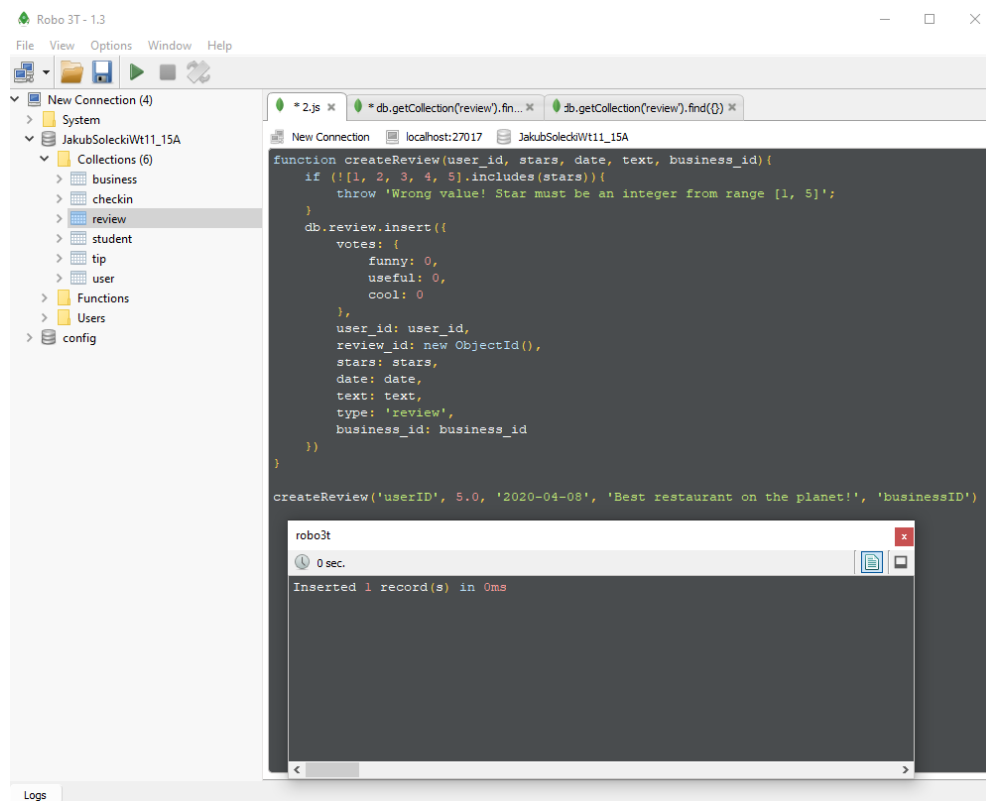
- g. Usuń wszystkie firmy (business), które posiadają ocenę (stars) równą 2.0.

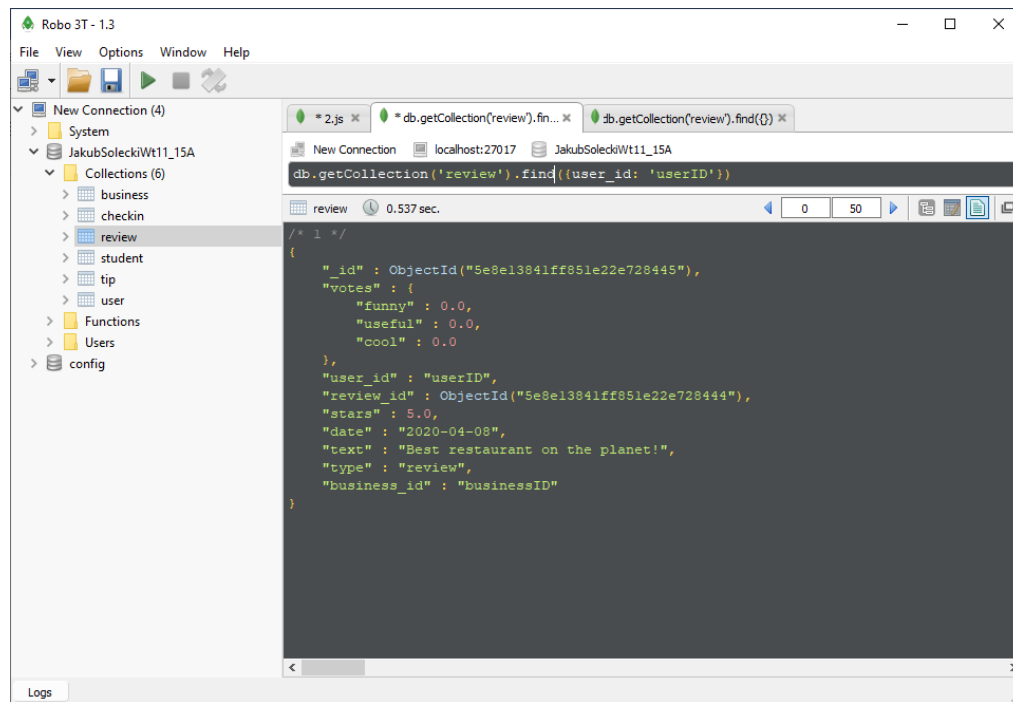
```
db.getCollection('business').deleteMany({'stars': 2.0})
```

2. Zdefiniuj funkcję (MongoDB) umożliwiającą dodanie nowej recenzji (review). Wykonaj przykładowe wywołanie.

```
function createReview(user_id, stars, text, business_id){
  if (![1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0].includes(stars)){
    throw 'Wrong value! Star must be from range [1.0, 5.0] and full' +
      'or half value.';
  }
  db.review.insert({
    votes: [
      {funny: 0},
      {useful: 0},
      {cool: 0}
    ],
    user_id: user_id,
    review_id: new ObjectId(),
    stars: stars,
    date: new Date(),
    text: text,
    type: 'review',
    business_id: business_id
  })
}

createReview('userID', 5.0, '2020-04-08', 'Best restaurant on the planet!',
'businessID')
```

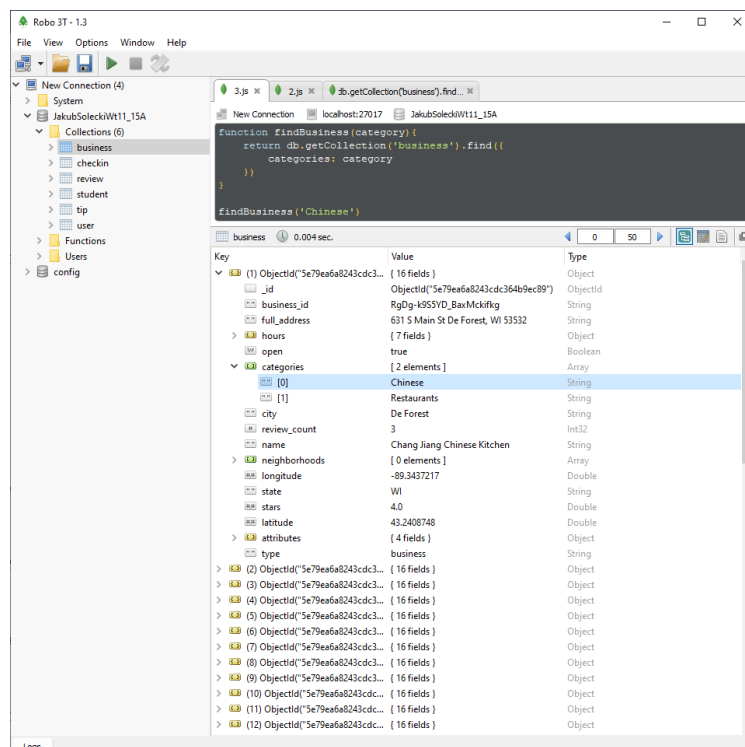




3. Zdefiniuj funkcję (MongoDB), która zwróci wszystkie biznesy (business), w których w kategorii znajduje się podana przez użytkownika cecha. Wartość kategorii należy przekazać do funkcji jako parametr. Wykonaj przykładowe wywołanie zdefiniowanej funkcji.

```
function findBusiness(category){
    return db.getCollection('business').find({
        categories: category
    })
}
```

```
findBusiness('Chinese')
```

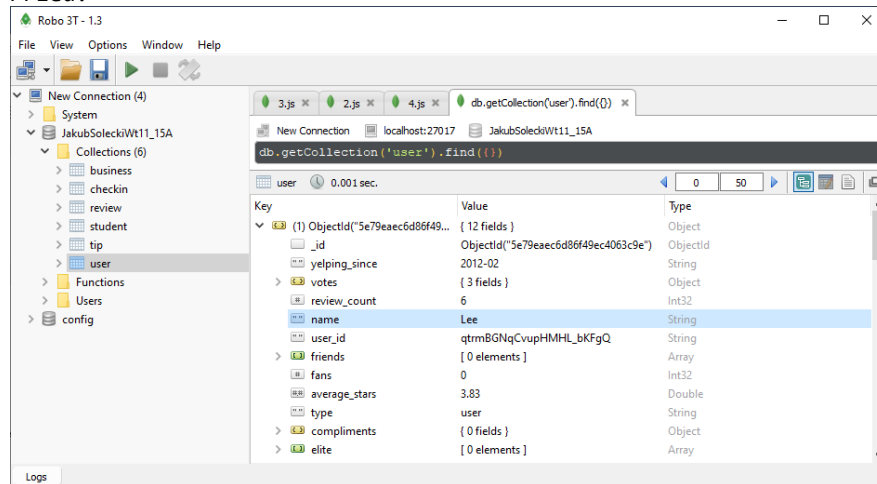


4. Zdefiniuj funkcję (MongoDB), która umożliwi modyfikację nazwy użytkownika (user) na podstawie podanego id. Id oraz nazwa mają być przekazywane jako parametry.

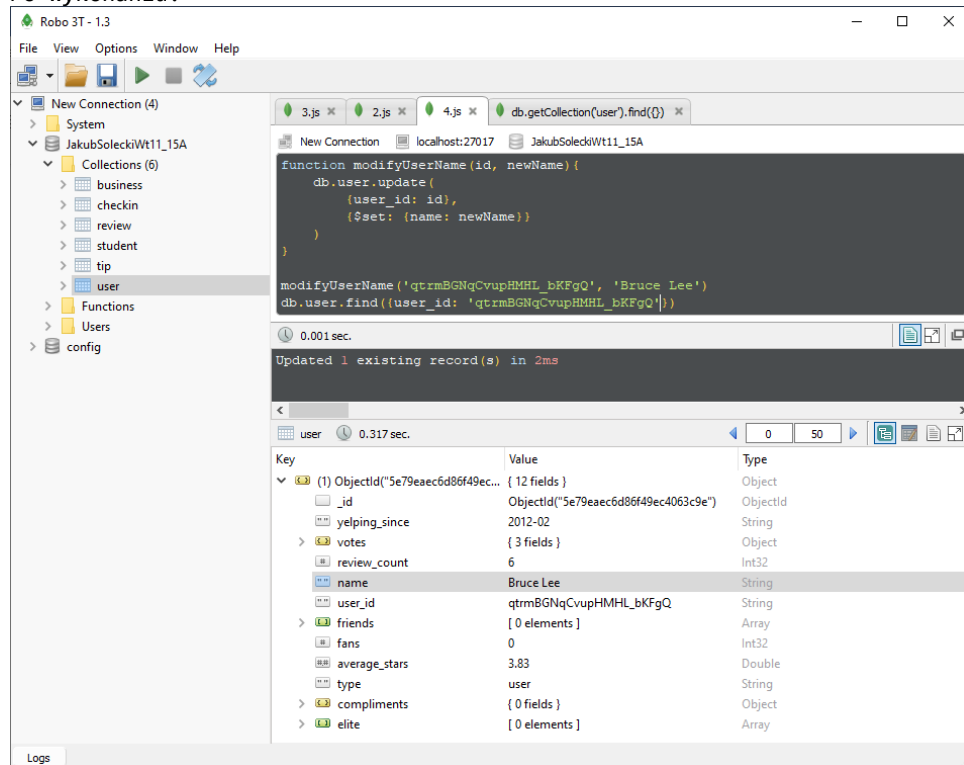
```
function modifyUserName(id, newName){
  db.user.update(
    {user_id: id},
    {$set: {name: newName}}
  )
}
```

```
modifyUserName('qtrmBGNqCvupHMHL_bKfgQ', 'Bruce Lee')
```

Przed:



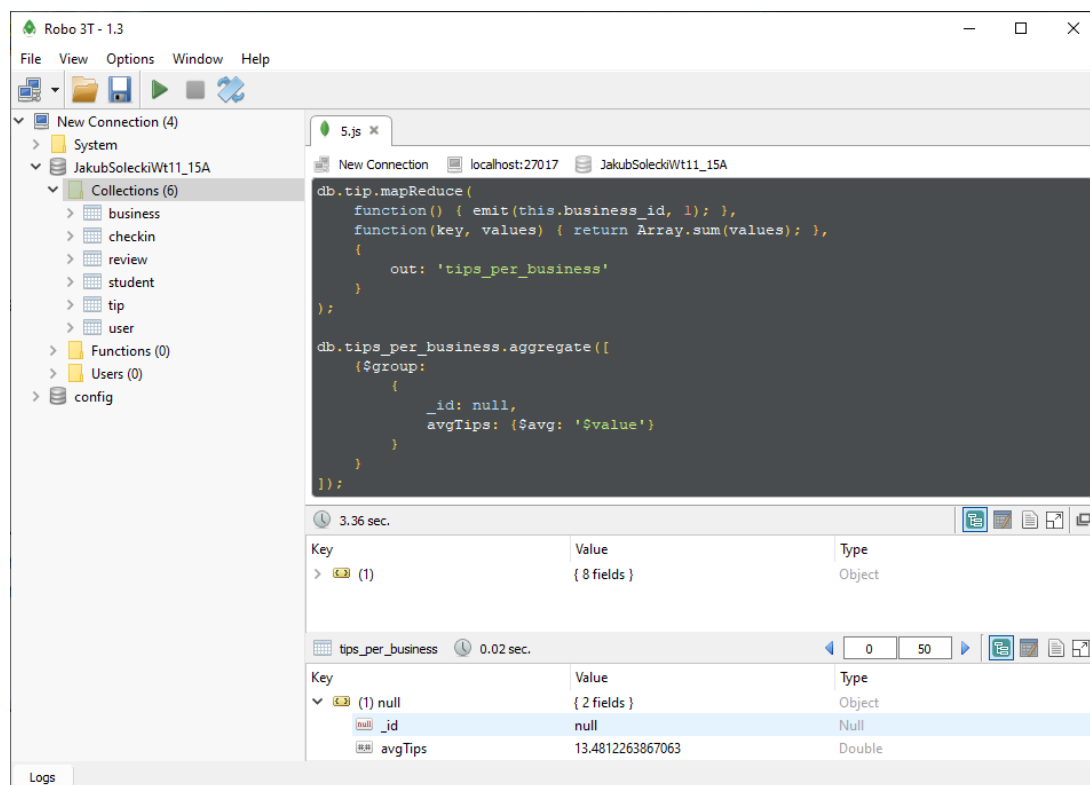
Po wykonaniu:



5. Zwróć średnią ilość wszystkich wskazówek/napiwków dla każdego z biznesów, wykorzystaj map reduce.

```
db.tip.mapReduce(
  function() { emit(this.business_id, 1); },
  function(key, values) { return Array.sum(values); },
  {
    out: 'tips_per_business'
  }
);

db.tips_per_business.aggregate([
  {$group:
    {
      _id: null,
      avgTips: {$avg: '$value'}
    }
  }
]);
```



6. Odwzoruj wszystkie zadania z punktu 1 w języku programowania (np. JAVA) z pomocą API do MongoDB. Wykorzystaj dla każdego zadania odrębną metodę.

```
public class MongoZad {
    private MongoClient mongoClient;
    private MongoDBDatabase db;

    public MongoZad() throws UnknownHostException {
        mongoClient = new MongoClient("localhost", 27017);
        db = mongoClient.getDatabase("JakubSoleckiWt11_15A");
    }

    private void zad_6_1_a() {
        List<String> res = db.getCollection("business").distinct("city",
        String.class).into(new ArrayList<>());
        Collections.sort(res);
    }
}
```



```

        for(String s: res)
            System.out.println(s);
    }

```

The screenshot shows a Java IDE with a console window displaying MongoDB logs. The logs indicate a successful connection to a MongoDB instance at localhost:27017. Below the logs, a list of business names is displayed, including Ahwatukee, Anthem, Apache Junction, Arcadia, Atlanta, Avondale, Black Canyon City, Bonnyrigg, Boulder City, Buckeye, and El Las Vegas. A 'Windows Defender configuration updated' notification is visible at the bottom right.

```

private void zad_6_1_b() {
    MongoCollection<Document> review = db.getCollection("review");
    System.out.println(review.countDocuments(Filters.gte("date", "2011-01-01")));
}

```

880318

```

private void zad_6_1_c() {
    List<Document> res = db.getCollection("business")
        .find(new Document("open", false))
        .projection(
            new Document("name", 1)
            .append("full_address", 1)
            .append("stars", 1)
        ).into(new ArrayList<Document>());

    for(Document doc: res)
        System.out.println(doc.toJson());
}

```

The screenshot shows a Java IDE with a console window displaying a list of business documents. Each document contains fields for \_id, \$oid, full\_address, name, and stars. The documents are sorted by name in descending order. The list includes businesses like Charter Communications, Crandalls Carryout & Catering, Hi Cocina, Stamm House At Pheasant Branch, Tangles, Soup Factory, Deli Roma, Java Detour, Bongo Video, Rocky Rocco Pan Style Pizza, Designs By the Bay, Williamson Bikes & Fitness, Area 51 Vintage Interiors, Dorn True Value Hardware, Mong's Garden, Dimitri's Gyros, Poppa Coronofoulos Gyros & Chicago Style Deli, and Groff's Buckeye Barben Shop.

```

private void zad_6_1_d() {
    MongoCollection<Document> user = db.getCollection("user");

    DBObject clause1 = new BasicDBObject("votes.funny", 0);
    DBObject clause2 = new BasicDBObject("votes.useful", 0);
    BasicDBList or = new BasicDBList();
    or.add(clause1);
    or.add(clause2);
    BasicDBObject query = new BasicDBObject("$or", or);

    FindIterable<Document> res = user.find(query).sort(new Document("name", -1));
    for(Document doc: res)
        System.out.println(doc.toJson());
}

```

```

{"_id": {"$oid": "5e79eaf26d86f49ec48a1412"}, "yelping_since": "2011-12", "votes": {"funny": 0, "useful": 1, "cool": 0}, "review_count": 4, "name": "Allison", "user_id": "Vpm4utrHN1ZwJ1PqWzuSw", "frienc"},
{"_id": {"$oid": "5e79eaf26d86f49ec48a17b8"}, "yelping_since": "2011-01", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 1, "name": "Allison", "user_id": "SubjUyV4U-EGScEYSPR36A", "frienc"},
{"_id": {"$oid": "5e79eaf6d86f49ec48a1444"}, "yelping_since": "2012-05", "votes": {"funny": 0, "useful": 1, "cool": 0}, "review_count": 1, "name": "Allisia", "user_id": "nZEPJy_BnxVfQVhLnUK3ZQ", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec487173b"}, "yelping_since": "2010-01", "votes": {"funny": 0, "useful": 4, "cool": 1}, "review_count": 11, "name": "Allisa", "user_id": "kKKy6yUVFYt44vrea2E1YQ", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec486c7f5"}, "yelping_since": "2012-07", "votes": {"funny": 0, "useful": 1, "cool": 0}, "review_count": 3, "name": "Allie Jane", "user_id": "ALNGL0BRw9R4pnmXL_muw", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4863d67"}, "yelping_since": "2011-06", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 2, "name": "Allie", "user_id": "VH8K0sqo7AmzN6mQH3mx_g", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4864a95"}, "yelping_since": "2013-08", "votes": {"funny": 0, "useful": 5, "cool": 2}, "review_count": 9, "name": "Allie", "user_id": "4vxxLjvNFIEroe9S9q6JzQ", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4866cab"}, "yelping_since": "2011-03", "votes": {"funny": 0, "useful": 3, "cool": 0}, "review_count": 1, "name": "Allie", "user_id": "P0fAxQVvYf4btjP0tQdw", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4868f36"}, "yelping_since": "2012-08", "votes": {"funny": 0, "useful": 4, "cool": 0}, "review_count": 3, "name": "Allie", "user_id": "A8kTZal-uARo6LO1Fo1Rg", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec486d917"}, "yelping_since": "2013-07", "votes": {"funny": 0, "useful": 2, "cool": 0}, "review_count": 2, "name": "Allie", "user_id": "pvCHLW7fjizS9FAvwi6MsQ", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4870282"}, "yelping_since": "2013-11", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 1, "name": "Allie", "user_id": "Cua6g8PasFVfHXZu3J48B1u", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4871b49"}, "yelping_since": "2011-06", "votes": {"funny": 0, "useful": 1, "cool": 0}, "review_count": 1, "name": "Allie", "user_id": "TAm8QeWGH_KR-IEJqPnttA", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4871e53"}, "yelping_since": "2013-10", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 1, "name": "Allie", "user_id": "syLgtVXEXMbf3k13LzCdw", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec487381d"}, "yelping_since": "2011-04", "votes": {"funny": 0, "useful": 2, "cool": 0}, "review_count": 3, "name": "Allie", "user_id": "bmxRktB_LX_Sf8R1gan3VQ", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec48744c4"}, "yelping_since": "2012-10", "votes": {"funny": 0, "useful": 4, "cool": 0}, "review_count": 3, "name": "Allie", "user_id": "Q-U8sHzZ_Ly2Fm8ws3Scw", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4874ca8"}, "yelping_since": "2011-10", "votes": {"funny": 0, "useful": 2, "cool": 1}, "review_count": 3, "name": "Allie", "user_id": "Yy31-FTvA1Yan-ov1KS9vu", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4876367"}, "yelping_since": "2011-03", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 1, "name": "Allie", "user_id": "U5tdbc095TPUEfNn8w9QA", "frienc"},
{"_id": {"$oid": "5e79eae6d86f49ec4879554"}, "yelping_since": "2012-03", "votes": {"funny": 0, "useful": 1, "cool": 1}, "review_count": 9, "name": "Allie", "user_id": "QYTsp2wP98vcpf48g748mg", "frienc"},
{"_id": {"$oid": "5e79eaf6d86f49ec487d517"}, "yelping_since": "2011-07", "votes": {"funny": 0, "useful": 1, "cool": 1}, "review_count": 1, "name": "Allie", "user_id": "XbAjlUsgL3hezJVVfKGGw", "frienc"},
{"_id": {"$oid": "5e79eaf6d86f49ec487d8ef"}, "yelping_since": "2014-01", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 3, "name": "Allie", "user_id": "EsRour-PZfHxLP18gSHGQ", "frienc"},
{"_id": {"$oid": "5e79eaf6d86f49ec487d9a8"}, "yelping_since": "2014-02", "votes": {"funny": 0, "useful": 0, "cool": 0}, "review_count": 2, "name": "Allie", "user_id": "BhQIcL0sv-yRkMON15zq-g", "frienc"}

```

```

private void zad_6_1_e() {
    MongoCollection<Document> tip = db.getCollection("tip");

    AggregateIterable<Document> agg = tip.aggregate(Arrays.asList(
        Aggregates.match(Filters.regex("date", "2012")),
        Aggregates.group("$business_id", sum("count", 1)),
        Aggregates.lookup("business", "_id", "business_id", "business_data"),
        Aggregates.unwind("$business_data"),
        Aggregates.project(
            new Document("name", "$business_data.name")
                .append("count", "$count")
        ),
        Aggregates.sort(Sorts.ascending("count"))
    ));

    for(Document doc: agg)
        System.out.println(doc.toJson());
}

```

```

Run: MongoZad
d": {"_id": "y2PjF0B1z3P0z-kHf2U5eQ", "name": "Smoothie King", "count": 1}
{"_id": "392pl_1hDtpAsqAc6fN7fg", "name": "Joolz", "count": 1}
{"_id": "RXi32yMcePy-1lHtye6vg", "name": "Jiffy Lube", "count": 1}
{"_id": "E_NKYE0HR-RR_Xox9VHCND", "name": "The Z Spot LV", "count": 1}
{"_id": "s5Mw61CnZLohZwFVCVPTQ", "name": "Domino's Pizza", "count": 1}
{"_id": "DvalhTH-RJlUa_Bf1QlZ91w", "name": "Walgreens", "count": 1}
{"_id": "qvcW143C8H9dX641NEEfQ", "name": "Paradise Car Wash", "count": 1}
{"_id": "J4L_M302snux3xCU2bLP8Q", "name": "Cranberry Hills Eatery & Catering", "count": 1}
{"_id": "cB8HkCvUj09DBaAMUDSMw", "name": "Renaissance Fine Jewelry", "count": 1}
{"_id": "bFosvxlK1PBI6dqvIKpQ5w", "name": "Great Dane Pub", "count": 1}
{"_id": "PaM_LRVZGvg1-e9fzeczPw", "name": "HomeGoods", "count": 1}
{"_id": "0QFCVAIRw6eqtRhxK3Z8BMA", "name": "Ryan's Bar", "count": 1}
{"_id": "kn3nVeUfMvYuhd1xonb4Thg", "name": "Robb & Stucky", "count": 1}
{"_id": "usw2n3CrgLjPaTijVwoH8A", "name": "Shade Apartments at Desert Ridge", "count": 1}
{"_id": "qwi36CWFyYxrBnyf5jA2FQ", "name": "Verizon Wireless", "count": 1}
{"_id": "nVsABtNAvGoiMoS0EfjTzQ", "name": "John Henry's", "count": 1}
{"_id": "tHkZego3MH6GdHhpsdkumA", "name": "AT&T", "count": 1}
{"_id": "7Y1lfzKwRohZK1MwZMw", "name": "El Pollo Loco", "count": 1}
{"_id": "iFEFF_yxCr64REdq9yrWmw", "name": "Scottsdale Cat Clinic", "count": 1}
{"_id": "hq_V09Vm86V_Xn_3S2GtA", "name": "Advanced Family Medicine", "count": 1}
{"_id": "SPSv9v66RD-lqcyMgGdg", "name": "Desert Willow Golf Course", "count": 1}
{"_id": "bL2fxmLTcQAS3QHMDwzXtHQ", "name": "Barnes & Noble Booksellers", "count": 1}

```

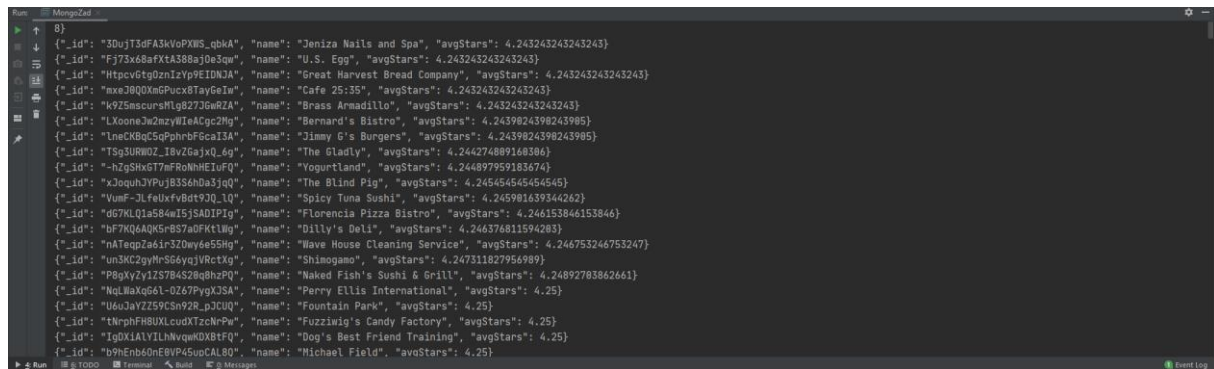
```

private void zad_6_1_f() {
    MongoCollection<Document> tip = db.getCollection("review");

    AggregateIterable<Document> agg = tip.aggregate(Arrays.asList(
        Aggregates.group("$business_id", avg("avgStars", "$stars")),
        Aggregates.lookup("business", "_id", "business_id", "business_data"),
        Aggregates.unwind("$business_data"),
        Aggregates.project(
            new Document("name", "$business_data.name")
                .append("avgStars", "$avgStars")
        ),
        Aggregates.match(Filters.gte("avgStars", 4.0)),
        Aggregates.sort(Sorts.ascending("avgStars"))
    ));

    for(Document doc: agg)
        System.out.println(doc.toJson());
}

```



```
private void zad_6_1_g() {
    db.getCollection("business").deleteMany(Filters.eq("stars", 2));
}

public static void main(String[] args) throws UnknownHostException {
    MongoZad mongoZad = new MongoZad();
    mongoZad.zad_6_1_a();
    mongoZad.zad_6_1_b();
    mongoZad.zad_6_1_c();
    mongoZad.zad_6_1_d();
    mongoZad.zad_6_1_e();
    mongoZad.zad_6_1_f();
    mongoZad.zad_6_1_g();
}
}
```

7. Zaproponuj bazę danych składającą się z 3 kolekcji pozwalającą przechowywać dane dotyczące klientów, zakupu oraz przedmiotu zakupu. W bazie wykorzystaj: pola proste, złożone i tablice. Zaprezentuj strukturę dokumentów w formie JSON dla przykładowych danych. Uzasadnij swoją propozycję

```
// product
db.createCollection("product", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: [ "name", "product_id", "category", "manufacturer", "price",
"unitsInStock" ],
            properties: {
                name: {
                    bsonType: "string",
                    description: "Must be a string and is required"
                },
                product_id: {
                    bsonType: "string",
                    description: "Must be a string and is required"
                },
                category: {
                    bsonType: "string",
                    description: "Must be a string and is required"
                },
                manufacturer: {
                    bsonType: "string",
                    description: "Must be a string and is required"
                },
                price: {
                    bsonType: "double",
                    description: "Must be a double and is required"
                }
            }
        }
    }
});
```

```

    },
    unitsInStock: {
      bsonType: "int",
      description: "Must be a int and is required"
    }
  }
}
})

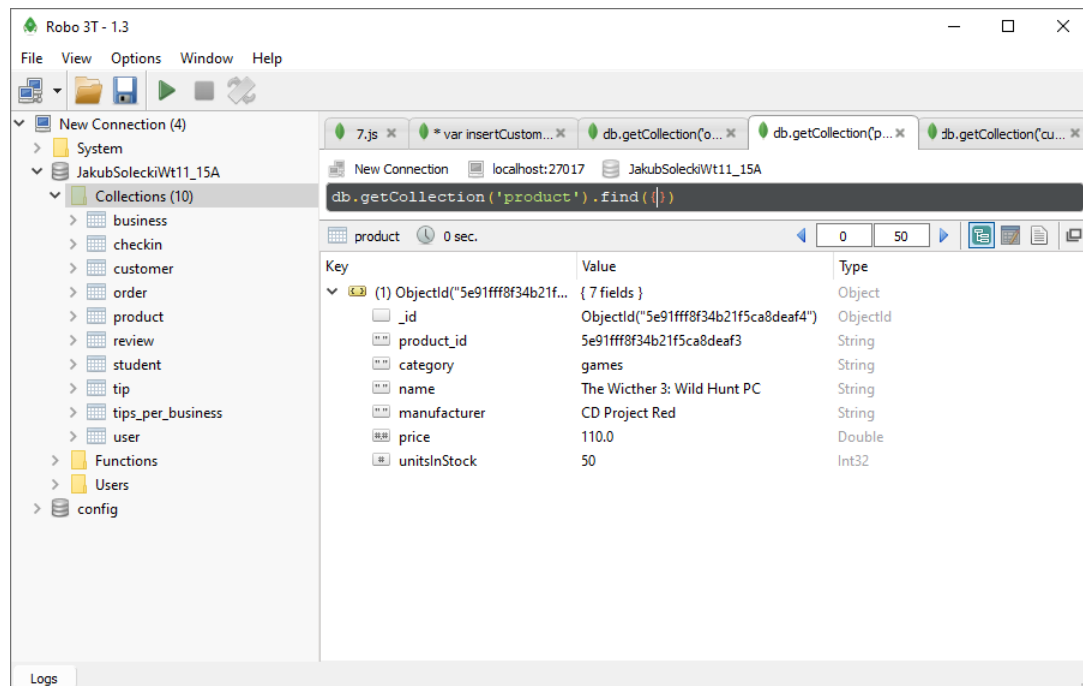
```

```

var insertProduct = function(name, category, manufacturer, price, unitsInStock) {
  prodID = new ObjectId().valueOf()
  db.product.insert({
    product_id: prodID,
    category: category,
    name: name,
    manufacturer: manufacturer,
    price: price,
    unitsInStock: NumberInt(unitsInStock)
  })
}

```

```
insertProduct("The Wicther 3: Wild Hunt PC", "games", "CD Project Red", 110.0, 50)
```



```

// customer
db.createCollection("customer", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "customer_id", "name", "phone", "email", "address", "orders" ],
      properties: {
        customer_id: {
          bsonType: "string",
          description: "Must be a string and is required"
        },
        name: {
          bsonType: "object",
          required: [ "firstname", "lastname" ],
          properties: {
            firstname: {
              bsonType: "string",

```

```

        description: "Must be a string and is required"
    },
    lastname: {
        bsonType: "string",
        description: "Must be a string and is required"
    }
},
phone: {
    bsonType: "string",
    description: "Must be a string and is required"
},
email: {
    bsonType: "string",
    description: "Must be a string and is required"
},
address: {
    bsonType: [ "object" ],
    required: [ "country", "city", "postalCode", "street", "buildingNumber"
],
    properties: {
        country: {
            bsonType: "string",
            description: "Must be a string and is required"
        },
        city: {
            bsonType: "string",
            description: "Must be a string and is required"
        },
        postalCode: {
            bsonType: "string",
            description: "Must be a string and is required"
        },
        street: {
            bsonType: "string",
            description: "Must be a string and is required"
        },
        buildingNumber: {
            bsonType: "string",
            description: "Must be a string and is required"
        }
    }
},
orders: {
    bsonType: "array",
    description: "Must be an array and is required (can be empty)"
}
}
}
})

```

```

var insertCustomer = function
(firstname, lastname, phone, email, country, city, postalCode, street, buildingNumber) {
    custID = new ObjectId().valueOf()
    db.customer.insert({
        customer_id: custID,
        name:
        {
            firstname: firstname,
            lastname: lastname
        },
        phone: phone,
        email: email,
        address:
        {

```

```

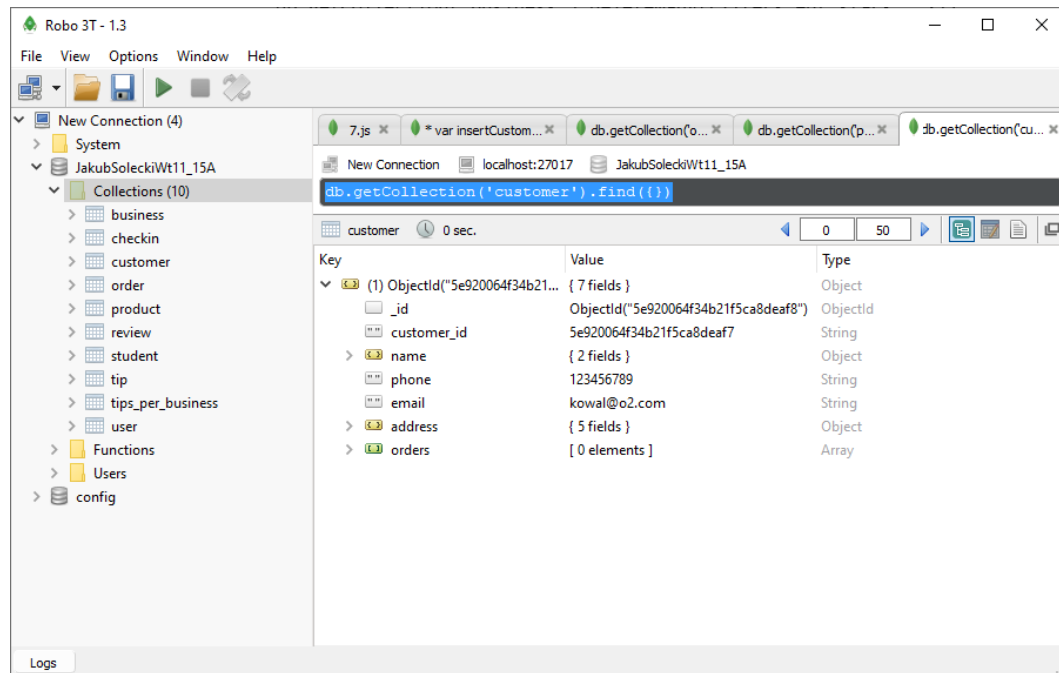
        country: country,
        city: city,
        postalCode: postalCode,
        street: street,
        buildingNumber: buildingNumber
    },
    orders: []
}
})
}

```

```

insertCustomer('Jan', 'Kowalski', '123456789', 'kowal@o2.com', 'Polska', 'Sosnowiec',
'21-370', 'Długa', '5A')

```



```

// order
db.createCollection("order", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "order_id", "customer_id", "name", "date", "prodcut_id",
"prodcut_name",
      "quantity", "price" ],
      properties: {
        order_id: {
          bsonType: "string",
          description: "Must be a string and is required"
        },
        customer_id: {
          bsonType: "string",
          description: "Must be a string and is required"
        },
        name: {
          bsonType: "string",
          description: "Must be a string and is required"
        },
        date: {
          bsonType: [ "string" ],
          description: "Must be a string and is required"
        },
        prodcut_id: {
          bsonType: "string",

```

```

        description: "Must be a string and is required"
    },
    prodcut_name: {
        bsonType: "string",
        description: "Must be a string and is required"
    },
    quantity: {
        bsonType: "int",
        description: "Must be a int and is required"
    },
    price: {
        bsonType: "double",
        description: "Must be a double and is required"
    }
}
}
}
}))

var addOrder = function(productID, customerID, quantity) {
    crs = db.product.find({product_id: productID})
    prod = crs.next()
    if(quantity > prod.unitsInStock)
        throw "Not enough units in stock"

    crs2 = db.customer.find({customer_id: customerID})
    cust = crs2.next()

    ordID = new ObjectId().valueOf()

    db.order.insert({
        order_id: ordID,
        customer_id: customerID,
        name: cust.name.firstname + ' ' + cust.name.lastname,
        date: new Date().toISOString().slice(0, 16).replace('T', ' '),
        prodcut_id: productID,
        prodcut_name: prod.name,
        quantity: NumberInt(quantity),
        price: quantity * prod.price
    })

    db.customer.update(
        {customer_id: customerID},
        {$addToSet: {orders: ordID}}
    )

    db.product.update(
        {product_id: productID},
        {$inc: {unitsInStock: NumberInt(-quantity)}}
    )
}

addOrder('5e91f5d1f34b21f5ca8deac1', '5e91f78ef34b21f5ca8dead1', 1)

```

Robo 3T - 1.3

File View Options Window Help

New Connection (4)

- System
- JakubSoleckiWt11\_15A
  - Collections (10)
    - business
    - checkin
    - customer
    - order
    - product
    - review
    - student
    - tip
    - tips\_per\_business
    - user
    - Functions
    - Users
    - config

7.js \* var addOrder = ... db.getCollection(o... db.getCollection(p... db.getCollection(cu... x

New Connection localhost:27017 JakubSoleckiWt11\_15A

db.getCollection('order').find({})

order 0 sec.

Key	Value	Type
(1) ObjectId("5e920117f34b21f5ca8deafa")	{ 9 fields }	Object
_id	ObjectId("5e920117f34b21f5ca8deafa")	ObjectId
order_id	5e920117f34b21f5ca8deaf9	String
customer_id	5e920064f34b21f5ca8deaf7	String
name	Jan Kowalski	String
date	2020-04-11 17:40	String
product_id	5e91fff8f34b21f5ca8deaf3	String
product_name	The Witcher 3: Wild Hunt PC	String
quantity	1	Int32
price	110.0	Double

Logs

Robo 3T - 1.3

File View Options Window Help

New Connection (4)

- System
- JakubSoleckiWt11\_15A
  - Collections (10)
    - business
    - checkin
    - customer
    - order
    - product
    - review
    - student
    - tip
    - tips\_per\_business
    - user
    - Functions
    - Users
    - config

7.js \* var addOrder = ... db.getCollection(o... db.getCollection(p... db.getCollection(cu... x

New Connection localhost:27017 JakubSoleckiWt11\_15A

db.getCollection('product').find({})

product 0 sec.

Key	Value	Type
(1) ObjectId("5e91fff8f34b21f5ca8deaf4")	{ 7 fields }	Object
_id	ObjectId("5e91fff8f34b21f5ca8deaf4")	ObjectId
product_id	5e91fff8f34b21f5ca8deaf3	String
category	games	String
name	The Witcher 3: Wild Hunt PC	String
manufacturer	CD Project Red	String
price	110.0	Double
unitsInStock	49	Int32

Logs

Robo 3T - 1.3

File View Options Window Help

New Connection (4)

- System
- JakubSoleckiWt11\_15A
  - Collections (10)
    - business
    - checkin
    - customer
    - order
    - product
    - review
    - student
    - tip
    - tips\_per\_business
    - user
    - Functions
    - Users
    - config

7.js \* var addOrder = ... db.getCollection(o... db.getCollection(p... db.getCollection(cu... x

New Connection localhost:27017 JakubSoleckiWt11\_15A

db.getCollection('customer').find({})

customer 0 sec.

Key	Value	Type
(1) ObjectId("5e920064f34b21f5ca8deaf8")	{ 7 fields }	Object
_id	ObjectId("5e920064f34b21f5ca8deaf8")	ObjectId
customer_id	5e920064f34b21f5ca8deaf7	String
name	{ 2 fields }	Object
phone	123456789	String
email	kowal@o2.com	String
address	{ 5 fields }	Object
orders	[ 1 element ]	Array
[0]	5e920117f34b21f5ca8deaf9	String

Logs



Zdecydowałem się na powyższy schemat, ponieważ wydaje mi się on odpowiedni do wymagań postawionych w treści zadania. 3 stworzone kolekcje znajdują się w tej samej bazie co yelp z zajęć, jednak zasada działania pozostaje niezmienna. Kolekcja *product* przechowuje informacje o produktach, *customer* o kupujących a *order* o ich zamówieniach. W moim zamyśle klienci istnieją niezależnie od produktów i na odwrót. Kiedy klient składa zamówienie tworzy nowy dokument w kolekcji *order*, zawierający informacje o zakupie konkretnej pozycji spośród istniejących produktów. Jednocześnie aktualizowane są odpowiednie pola w dokumencie produktu oraz klienta: zmniejsza się liczba dostępnych produktów (o liczbę wykupionych sztuk) a w dokumencie klienta id produktu zostaje dodane do tablicy z dokonanymi zamówieniami (*orders*). Uważam takie rozwiązanie za czytelne i pozwalające szybko orientować się w historii zamówień, jednak bez zbędnego przeładowywania dokumentów danymi.