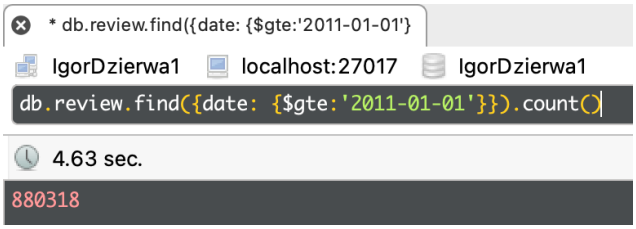
Laboratorium 3: MongoDB — zadanie

Igor Dzierwa
Grupa: 1

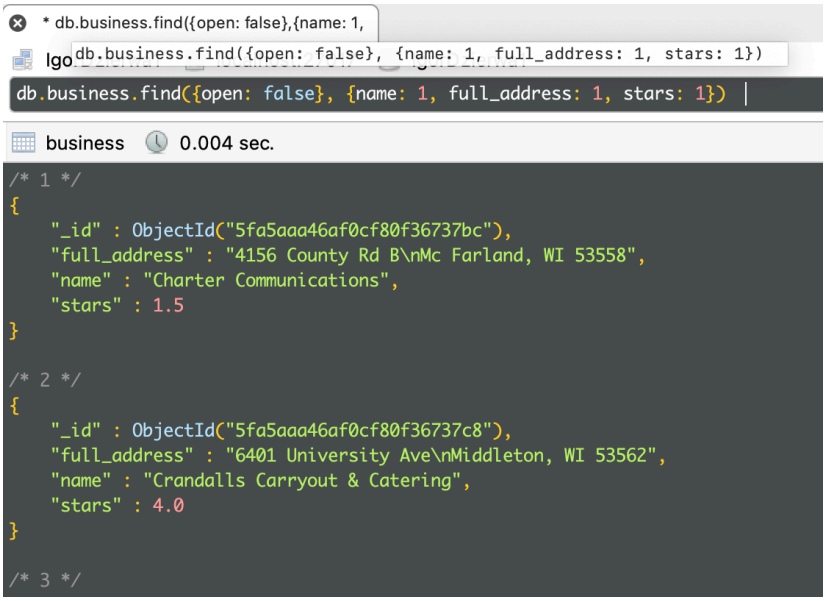
1.Wykorzystując bazę danych yelp dataset, wykonaj zapytanie i komendy MongoDB, aby uzyskać następujące rezultaty.

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

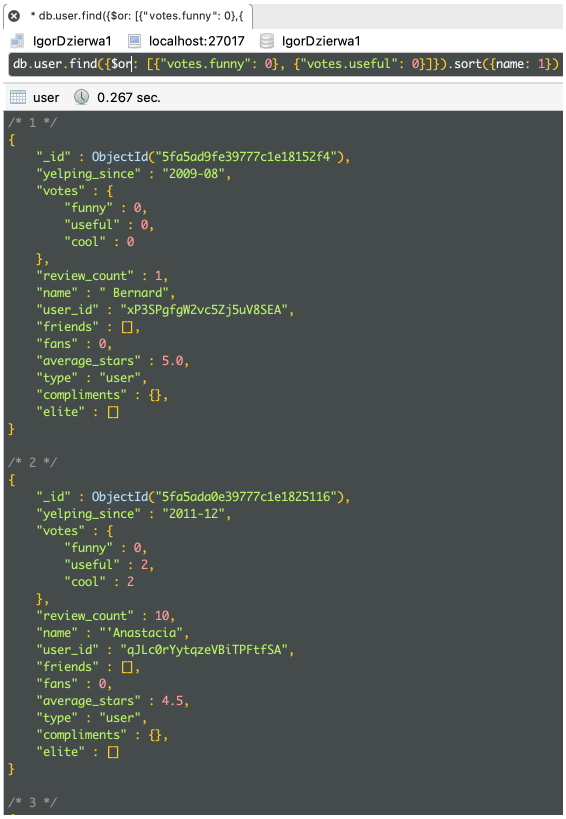
b) Zwróć liczbę wszystkich recenzji, które pojawiły się po 2011 roku *(włącznie).*



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



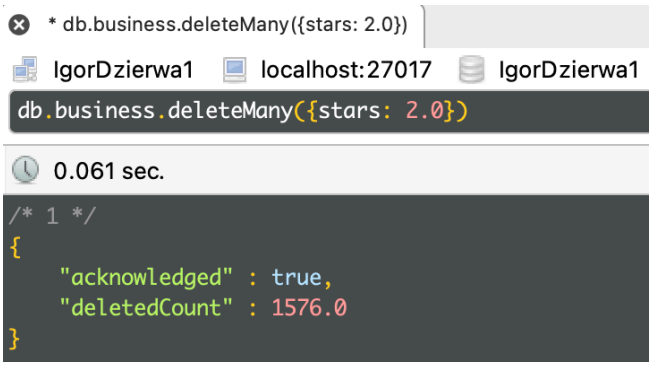
d) Zwróć dane wszystkich użytkowników *(`user`)*, którzy nie uzyskali ani jednego pozytywnego głosu z kategorii *(`funny` lub `useful`)*, wynik posortuj alfabetycznie według imienia użytkownika.



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

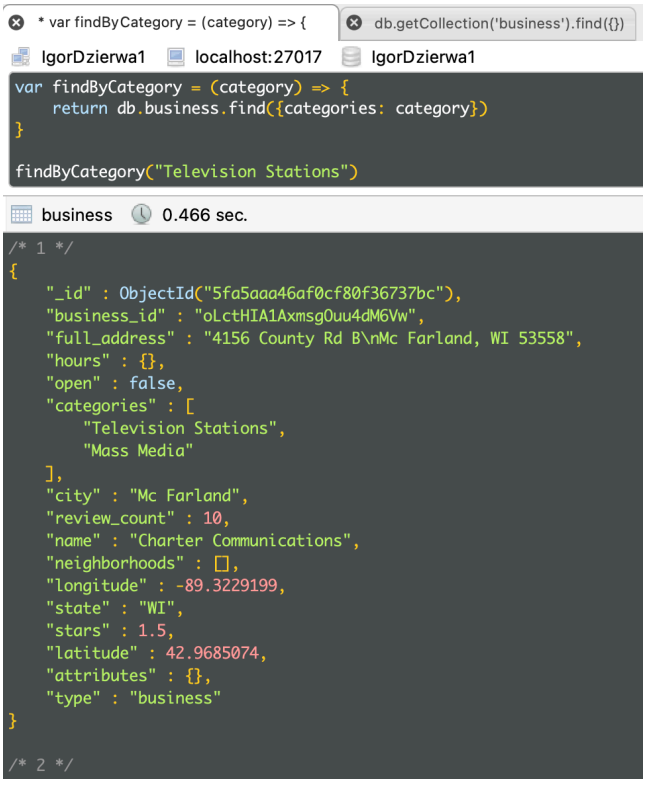
f) Wyznacz, jaką średnią ocen *(`stars`)* uzyskała każda firma *(`business`)* na podstawie wszystkich recenzji.
Wynik ogranicz do recenzji, które uzyskały min. 4.0 gwiazdki.

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

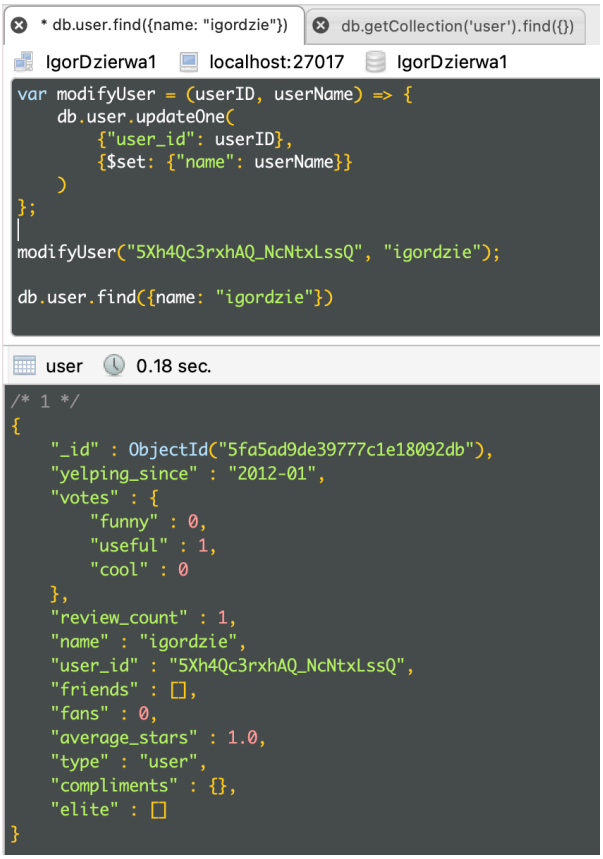


2.Zdefiniuj funkcję *(MongoDB)* umożliwiającą dodanie nowej recenzji *(`review`).*

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.



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



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

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

***Wybrany język programowania:** Java

- Pobrane pliki jar:

  `mongo-java-driver-3.12.7.jar`,
  `mongodb-driver-core-4.1.1.jar`.

```java
public class Mongo {
    private MongoClient mongoClient;
    private MongoDatabase db;

    public Mongo(String hostname, int port, String databaseName) {
        this.mongoClient = new MongoClient(hostname , port);
        this.db = mongoClient.getDatabase(databaseName);
    }

    public MongoClient getMongoClient() { return this.mongoClient; }

    public MongoDatabase getDb() { return this.db; }

    public void showCollections() {
        for (String name : db.listCollectionNames()) {
            System.out.println("colection name: " + name);
        }
    }
```

```java
public class Main {

    public static void main(String[] args) {
        Mongo mongoLab = new Mongo( hostname: "localhost",  port: 27017,  databaseName: "IgorDzierwa1");

        mongoLab.showCollections();
```

```
colection name: user
colection name: students
colection name: checkin
colection name: votes_summary
colection name: business
colection name: tip
colection name: review
colection name: sum_tips
```

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

```java
public void pointA () {
    MongoCollection<Document> collection = this.db.getCollection( s: "business");
    List<String> results = new ArrayList<>();
    collection.distinct( s: "city", String.class).into(results);

    Collections.sort(results);

    for (String city : results) {
        System.out.println(city);
    }
}
```

```
Ahwatukee
Anthem
Apache Junction
Arcadia
Atlanta
Avondale
Black Canyon City
```

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

```
public void pointB () {
    MongoCollection<Document> collection = this.db.getCollection( s: "review");
    List<Document> results = new ArrayList<>();
    BasicDBObject whereQuery = new BasicDBObject();
    whereQuery.put("date", new Document("$gte", "2011-01-01"));
    collection.find(whereQuery).into(results);

    System.out.println("All reviews after 2011-01-01: " + results.size());
}
```

```
All reviews after 2011-01-01: 880320
```

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

```
public void pointC () {
    MongoCollection<Document> collection = this.db.getCollection( s: "business");
    List<Document> results = new ArrayList<>();
    BasicDBObject whereQuery = new BasicDBObject();
    whereQuery.put("open", false);
    collection.find(whereQuery)
            .projection(Projections.include( ...fieldNames: "name", "full_address", "stars"))
            .into(results);

    for (Document document : results) {
        System.out.println(document);
    }
}
```

```
Document{{_id=5fbd9ca82f691c51b61f2367, full_address=10430 S Eastern Ave
Anthem
Henderson, NV 89052, name=Carmine's, stars=3.0}}
Document{{_id=5fbd9ca82f691c51b61f2377, full_address=3000 W Ann Rd
Ste 109
North Las Vegas, NV 89031, name=Super No1 Chinese Restaurant, stars=3.5}}
Document{{_id=5fbd9ca82f691c51b61f23f0, full_address=13014 N Saguaro Blvd
Fountain Hills, AZ 85268, name=Moxie Modern Fare, stars=3.5}}
```

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

```
public void pointD () {
    MongoCollection<Document> collection = this.db.getCollection( s: "user");
    List<Document> results = new ArrayList<>();

    BasicDBObject orQuery = new BasicDBObject();
    List<BasicDBObject> orArguments = new ArrayList<>();
    orArguments.add(new BasicDBObject("votes.funny", 0));
    orArguments.add(new BasicDBObject("votes.useful", 0));
    orQuery.put("$or", orArguments);

    collection.find(orQuery).into(results);

    for (Document document : results) {
        System.out.println(document);
    }
}
```

```
Document{{_id=5fa5ada4e39777c1e1846eac, yelping_since=2010-11, votes=Document{{funny=0, useful=1, cool=0}}, review_count=3, name=Allister, user_id=ZRzhH3_33EkbimjsWYHRNw, friends=[], fans=0, ave
Document{{_id=5fa5ada4e39777c1e1846eaf, yelping_since=2013-06, votes=Document{{funny=0, useful=0, cool=0}}, review_count=1, name=Bibek, user_id=hmPP1xvAv9b-TY8bman85w, friends=[], fans=0, averag
Document{{_id=5fa5ada4e39777c1e1846eb1, yelping_since=2013-08, votes=Document{{funny=0, useful=4, cool=1}}, review_count=1, name=Ben, user_id=eurYEQKminFFm0ZGj1UlJw, friends=[], fans=0, average_
Document{{_id=5fa5ada4e39777c1e1846eb2, yelping_since=2012-08, votes=Document{{funny=0, useful=3, cool=3}}, review_count=2, name=Jennifer, user_id=cm2OUiKU7sE51bsCfN8jzw, friends=[YHbrdVajp-WYsj
Document{{_id=5fa5ada4e39777c1e1846eb4, yelping_since=2013-10, votes=Document{{funny=1, useful=0, cool=0}}, review_count=1, name=Anamika, user_id=CYKWssZpMJYE6uXzydKGUA, friends=[], fans=0, aver
Document{{_id=5fa5ada4e39777c1e1846eb5, yelping_since=2014-04, votes=Document{{funny=0, useful=0, cool=0}}, review_count=2, name=Stacey, user_id=lyq6kwACo7GGWVs-4VKPHQ, friends=[], fans=0, avera
Document{{_id=5fa5ada4e39777c1e1846eb7, yelping_since=2007-05, votes=Document{{funny=0, useful=1, cool=0}}, review_count=1, name=Eric, user_id=gXeEBcrUAA6mMAmDRGW78A, friends=[], fans=0, average
Document{{_id=5fa5ada4e39777c1e1846eb8, yelping_since=2011-07, votes=Document{{funny=0, useful=3, cool=0}}, review_count=1, name=Ronny, user_id=xgta1nJdBd1UW4kCVdnGCg, friends=[], fans=0, averag
Document{{_id=5fa5ada4e39777c1e1846eb9, yelping_since=2011-01, votes=Document{{funny=0, useful=0, cool=0}}, review_count=6, name=Fiana Sachi, user_id=qQBLVT7eJKD1aC2KoY4fSw, friends=[], fans=0,
```

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

```
public void pointE () {
    MongoCollection<Document> collection = this.db.getCollection( s: "tip");

    List<Bson> results = Arrays.asList(
            Aggregates.match(Filters.and(
                    Filters.gte( fieldName: "date",  value: "2012-01-01"),
                    Filters.lte( fieldName: "date",  value: "2012-12-31"))),
            Aggregates.group( id: "$business_id", Accumulators.sum( fieldName: "count",  expression: 1))
    );

    AggregateIterable iterable = collection.aggregate(results);

    for (Object i : iterable) {
        System.out.println(i);
    }
}
```

```
Document{{_id=BRcREoex-p9WLnscUjN2sQ, count=4}}
Document{{_id=Xr0SQ1i6vHgT0RBZfyzJ8Q, count=1}}
Document{{_id=_2HFbrYZFp2p59hptZeDbw, count=2}}
Document{{_id=SMUVbjzqLXXEJaGZkG134Q, count=1}}
Document{{_id=0wK--zW6Bgqgvnob8Qc77g, count=4}}
Document{{_id=CDY7AE5HWBy0TJXCakqLBQ, count=1}}
Document{{_id=7mNPyQJYtIR26Y4KQa2g1Q, count=1}}
Document{{_id=N7jFyn8PgetjOVeJG122vw, count=3}}
Document{{_id=Lc3-XLmobyijQpOxHdEPFQ, count=2}}
Document{{_id=VlEWdvhdZS_F9Lwxn67IOQ, count=10}}
Document{{_id=sLRPuboSA7Oa0yjW1q7gMA, count=3}}
Document{{_id=wx2EJUCNOCPrMC0DtKb98A, count=2}}
Document{{_id=83C1eqPfmQHTy_8MC4Ug6w, count=6}}
```

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

```
public void pointF () {
    MongoCollection<Document> collection = this.db.getCollection( s: "review");

    List <Bson> results = Arrays.asList(
            Aggregates.group( id: "$business_id", Accumulators.avg( fieldName: "averageStars",  expression: "$stars")),
            Aggregates.match(Filters.gte( fieldName: "averageStars",  value: 4.0))
    );

    AggregateIterable iterable = collection.aggregate(results);

    for (Object i : iterable) {
        System.out.println(i);
    }
}
```

```
Document{{_id=yGK0UuDDg9VOrNYNakIFRw, averageStars=4.4375}}
Document{{_id=Ls7XGu8JFYfGBOHaGzp-_A, averageStars=4.2}}
Document{{_id=Kx3ITctVIWmlxQ3gj6C0TQ, averageStars=4.8}}
Document{{_id=7whi5KTXSNkZaSMZ_NRAmw, averageStars=5.0}}
Document{{_id=NnoEbaGnQa44CZE_A5tBTQ, averageStars=5.0}}
Document{{_id=hHAlLNh4q7n6E5PTrtJZ8Q, averageStars=4.333333333333333}}
Document{{_id=lw74h6AbGU-zeOYpAe56PA, averageStars=4.318181818181818}}
Document{{_id=tE_ytDK-ox7a-cWX7txvIA, averageStars=4.5}}
Document{{_id=Ajhc8sooWSX5cCkz4LVUXA, averageStars=4.142857142857143}}
Document{{_id=Bl5tTS0Tie3dWA2hWn53Jw, averageStars=4.666666666666667}}
Document{{_id=O7-O54xA54g_tyP3A9pXZQ, averageStars=5.0}}
Document{{_id=w4zPPr-MOkqNq4zC11o8FA, averageStars=4.0}}
Document{{_id=TXysymbMWuKD_U2LI8yVbA, averageStars=4.666666666666667}}
Document{{_id=nMHhuYan8e3cONo3PornJA, averageStars=4.294117647058823}}
```

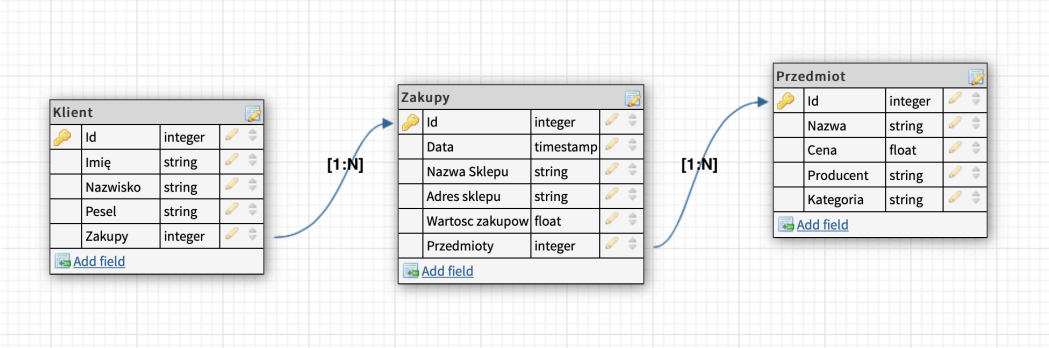
g) Usuń wszystkie firmy *(`business`)*, które posiadają ocenę *(`stars`)* równą 2.0.

```
public void pointG () {
    MongoCollection<Document> collection = this.db.getCollection( s: "business");
    collection.deleteMany(eq( fieldName: "stars",  value: 2.0));
}
```

```
* db.getCollection('business').find({stars:
IgorDzierwa1   localhost:27017   IgorDzierwa1
db.getCollection('business').find({stars: 2.0})
0.054 sec.
Fetched 0 record(s) in 54ms
```

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.

***Dokument** — odpowiednik rekordu z podejścia relacyjnego
***Kolekcja** — odpowiednik tabel z podejścia relacyjnego.



***Dokument Klient** — zawiera 5 pól:
- id — klucz główny
- Imię *(string)*, Nazwisko *(string)*, Pesel *(string)* — pozwalają na jednoznaczne zdefiniowanie klienta.
- Zakupy *(integer)*— identyfikator paragonu. W relacyjnym modelu bazy danych byłby kluczem obcym łączącym tabelę Klient z tabelą Zakupy *(relacja jeden do wielu).*

***Dokument Zakupy** — zawiera 5 pól:
- id — klucz główny.
- Data *(timestamp)*, Nazwa Sklepu *(string)*, Adres Sklepu *(string),* Wartosc zakupow *(float)* — pozwalają na jednoznaczne zdefiniowanie paragonu.
- Przedmioty (integer) — Identyfikator przedmiotu. W relacyjnym modelu bazy danych byłby kluczem obcym łączącym tabelę Zakupy z tabelą Przedmiot *(relacja jeden do wielu).*

***Dokument Przedmiot** — zawiera 5 pól:
- id — klucz główny
- Nazwa *(string),* Cena *(float),* Producent *(string)*, Kategoria *(string)* — pozwalają na jednoznaczne zdefiniowanie produktu.

Utworzenie bazy danych składającej się z 3 kolekcji *(Klient, Zakupy, Przedmiot)

```
> use ShoppingDB
switched to db ShoppingDB
```

```
> db.createCollection("Klient")
{ "ok" : 1 }
> db.createCollection("Zakupy")
{ "ok" : 1 }
> db.createCollection("Przedmiot")
{ "ok" : 1 }
```

***Funkcja dodająca Klienta:**

```
var addClient = (imie, nazwisko, pesel) => {
    db.Klient.insertOne({
        "imie": imie,
        "nazwisko": nazwisko,
        "pesel": pesel,
        "zakupy": []
    })
}

addClient("Jan", "Kowalski", "97021002999");
addClient("Robert", "Nowak", "99023001222");
```

***Funkcja dodająca Zakupy:**

```
var addShopping = (nazwaSklepu, adresSKlepu, wartoscZakupow) => {
    db.Zakupy.insertOne({
        "date": new Date(),
        "nazwaSklepu": nazwaSklepu,
        "adresSklepu": adresSKlepu,
        "wartoscZakupow": wartoscZakupow,
        "przedmioty": []
    })
}

addShopping("Zabka", "Dobrego Pasterza 103", 100);
addShopping("Lidl", "Czarnowiejska 90", 50);
```

***Funkcja dodająca Przedmiot:**

```
var addProduct = (nazwa, cena, producent, kategoria) => {
    db.Przedmiot.insertOne({
        "nazwa": nazwa,
        "cena": cena,
        "producent": producent,
        "kategoria": kategoria,
    })
}

addProduct("mleko", 10, "mullermilch", "przetwory mleczne");
addProduct("bombonierka", 40, "Lindt lindor", "slodycze");
```

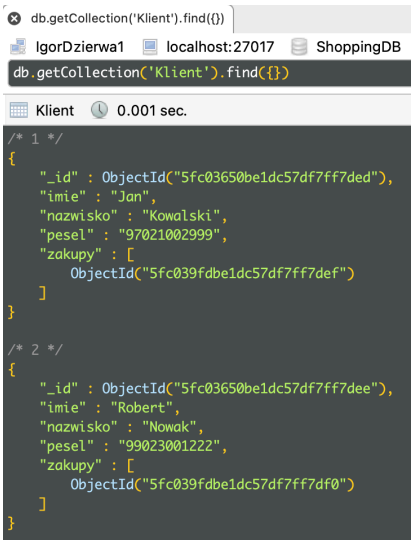
***Funkcja dodająca Przedmioty do Zakupów:**

```
var addProductToShopping = (shoppingID, productID) => {
    db.Zakupy.update(
        {_id: ObjectId(shoppingID)},
        {$push: {przedmioty: new ObjectId(productID)}}
    )
}

addProductToShopping("5fc039fdbe1dc57df7ff7def", "5fc03ce6be1dc57df7ff7df1");
addProductToShopping("5fc039fdbe1dc57df7ff7def", "5fc03ce6be1dc57df7ff7df1");
addProductToShopping("5fc039fdbe1dc57df7ff7def", "5fc03ce6be1dc57df7ff7df2");
addProductToShopping("5fc039fdbe1dc57df7ff7def", "5fc03ce6be1dc57df7ff7df2");

addProductToShopping("5fc039fdbe1dc57df7ff7df0", "5fc03ce6be1dc57df7ff7df1");
addProductToShopping("5fc039fdbe1dc57df7ff7df0", "5fc03ce6be1dc57df7ff7df2");
```

***Funkcja dodająca Zakupy do Klientów:**

```
var addShoppingToClient = (clientID, shoppingID) => {
    db.Klient.update(
        {_id: ObjectId(clientID)},
        {$push: {zakupy: new ObjectId(shoppingID)}}
    )
};

addShoppingToClient("5fc03650be1dc57df7ff7ded", "5fc039fdbe1dc57df7ff7def");
addShoppingToClient("5fc03650be1dc57df7ff7dee", "5fc039fdbe1dc57df7ff7df0");
```

***Przykładowe dane kolekcji Klient:**



***Przykładowe dane kolekcji Zakupy:**

***Przykładowe dane kolekcji Przedmiot:**