



UNIVERSITÀ DI PISA

7/21/2021

MehrCar

Large scale and multistructured
databases



Kamran Mehravar

[Click here for the GitHub repository of the project](#)

Table of contents

Table of contents	1
INTRODUCTION.....	2
MAIN ACTORS	3
FUNCTIONAL REQUIREMENTS	3
NON-FUNCTIONAL REQUIREMENTS	4
DATASET DESCRIPTION	4
UML CLASS DIAGRAM	5
USE CASE DIAGRAM	6
DATABASE ORGANIZATION	7
MONGODB ORGANIZATION	7
SOFTWARE ARCHITECTURE: DESIGN AND IMPLEMENTATION.....	10
SOFTWARE ARCHITECTURE	10
IMPLEMENTATION PHASE	11
MOST RELEVANT QUERIES USING MONGO DB	12
ANALYTIC 1: RANKING OF THE Top 3 Regions	12
ANALYTIC 2: Average Price in Top 3 Regions.....	13
ANALYTIC 3: Favorite Brands for Men and Women	13
ANALYTIC 4: -The Most Expensive Car in Men's and Women's Favorites Brand	14
ANALYTIC 5: RANKING OF THE TOP 3 Manufacturer	14
ANALYTIC 6: RANKING OF 5 most popular paint Color.....	15
INDEXES	16
MONGODB INDEXES	16
TEST ON AVAILABILITY OF THE APPLICATION.....	18

INTRODUCTION

“MehrCar” is a second-hand Car Advertisement application whose core purpose is to provide a platform not only to finding cars but also to search Cars by the user's needs, browse through lists of advertisements, and Filter them but also to users who post new advertisement and meanwhile search for relevant cars by models and etc. The users first log in with their existing credentials (i.e. phone number and password) or sign up as new users to publish an advertisement or log in as admin to get more access and see analytics models and have full control from the application’s functionalities.

The core functions of a MehrCar include searching for Cars with respect to a few parameters. On the other hand, a user can mainly perform tasks such as publishing a new advertisement or, Edit, remove them, and browsing through list of its already published Advertisements, browsing through the list of cars and searching for Specific cars, for instance searching by specific parameters like, year and model, manufacture, region.

MAIN ACTORS

The application has **two main actors**:

- **User.** He/she is one type of user of the application, in particular it is the person who view, add and searches for an advertisement.
- **Admin.** He/she is the administrator of the application. He has special privileges that allows him, for example, to delete user accounts or to do actions that the other users cannot do.

FUNCTIONAL REQUIREMENTS

- An **anonymous user** can
 - Register as a New user
 - Login as a user/admin
- A **standard user** can
 - update (Edit)information associated to his advert
 - Search advertising by Region, by Manufacturer (like Ford, Chevrolet, BMW, ..., by Production Year, and browse them
 - view his adverts published and can do remove action on them
- The **administrator** can
 - add a new user
 - add a new Advertisement
 - Search users by their phone number
 - Search Adverts by Phone number
 - Find top 3 regions depending on submitted Adverts (where top 3 means the which Regions that published more cars (Adverts).
 - Find Average Price in Top 3 Regions
 - Find Favourite Car's Brands for Men
 - Find The Most Expensive Car in Men's Favourite Brand
 - Find Favourite Brands for Women
 - Find The Most Expensive Car in Women's Favourite Brand
 - Find the 5 most popular paint Color in our platform
 - Find the top 3 Manufacturer by submitted cars (where top 3 means the ones that have more Cars)
 - View the number of total user accounts, Total Cars in our Platform
 - update (Edit)information associated to the specific advert
 - view an advert that has been published and can do remove action on them
 - Search advertising by Region, by Manufacturer (like Ford, Chevrolet, BMW, ..., by Production Year, and browse them

NON-FUNCTIONAL REQUIREMENTS

Here there is a list of the non-functional requirements of the application:

- **Performance:** the response time of the application to user's actions under certain workload should be minimum, in order to offer to the user a good experience while interacting with it
- **Usability:** the application must be easy to use and to understand for a common user
- The application store information in a non-relational Database which is the MongoDB.
- **AP Solution (Availability and Partition Tolerance):** The application should be accessible all the time. It needs to continue to function regardless of system failures and network partitions.

DATASET DESCRIPTION

The data for the database of the application comes from the kaggle website

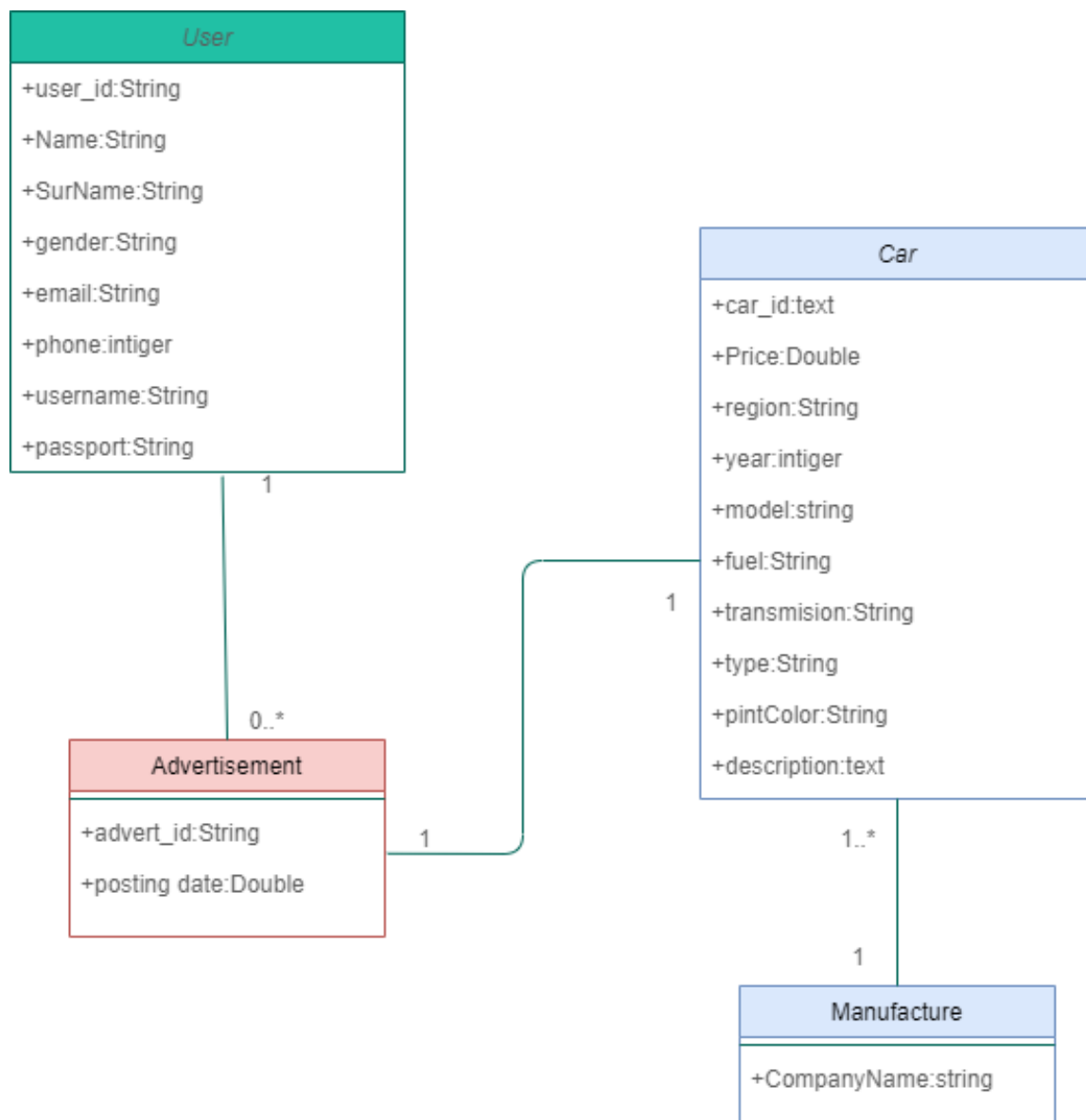
Source:

[Dataset of Cars and Users from kaggle](#)

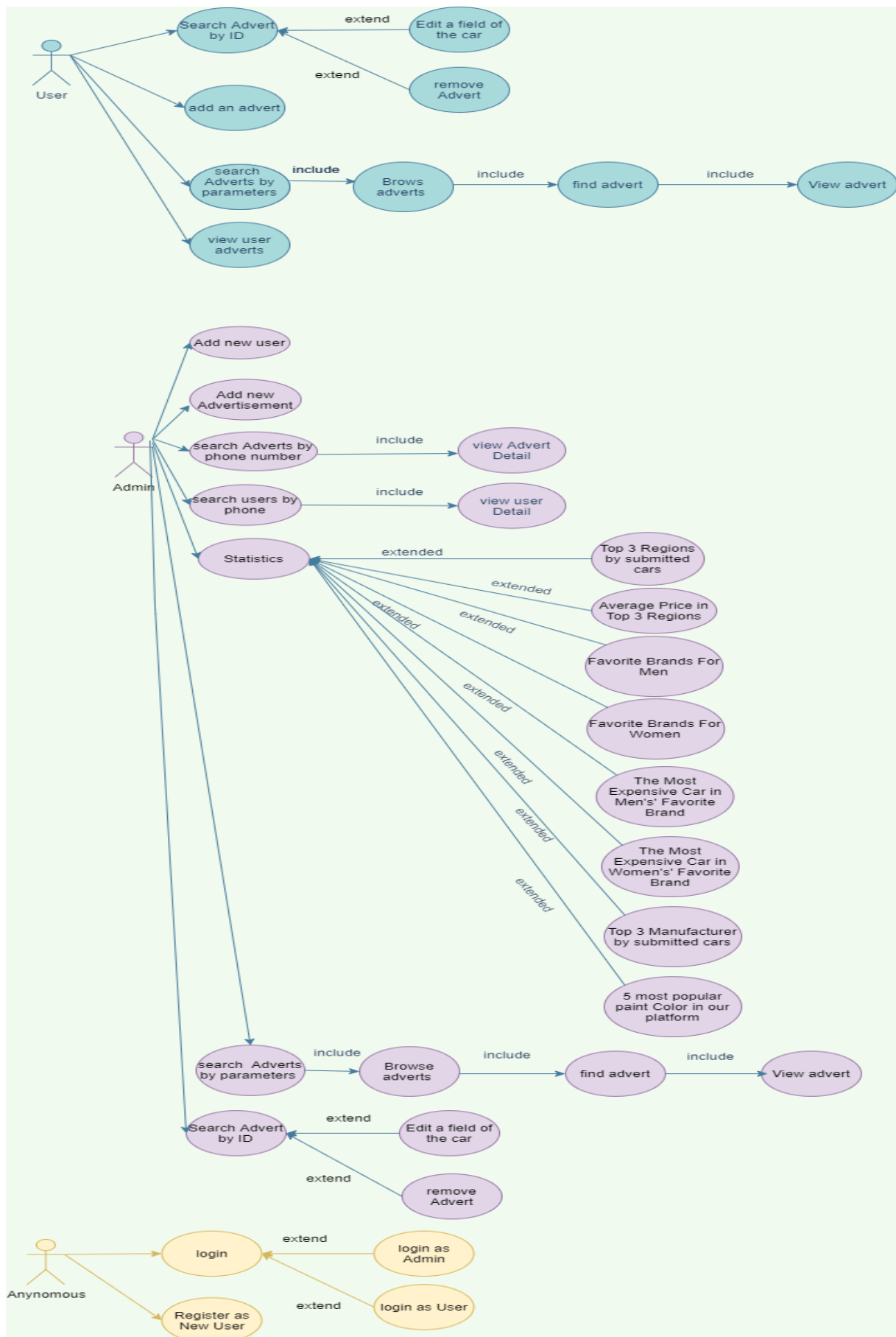
Cars: For Cars, I collect data from a dataset on Kaggle website and data scraped by ourselves from indeed website.

Users: For users, I merge data for, names and passwords,username,ID,Email,phone from different sources to create users.

UML CLASS DIAGRAM



USE CASE DIAGRAM



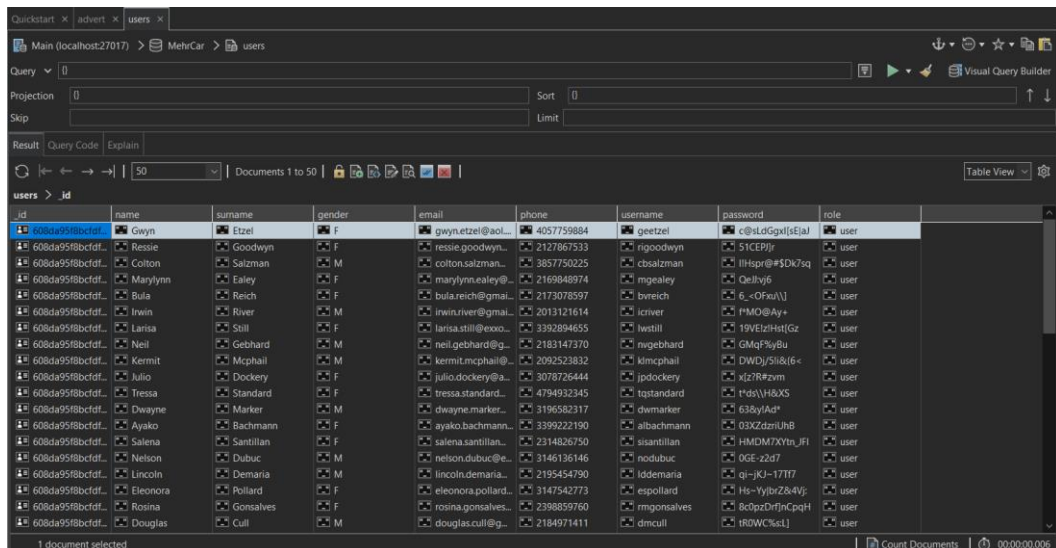
DATABASE ORGANIZATION

MONGODB ORGANIZATION

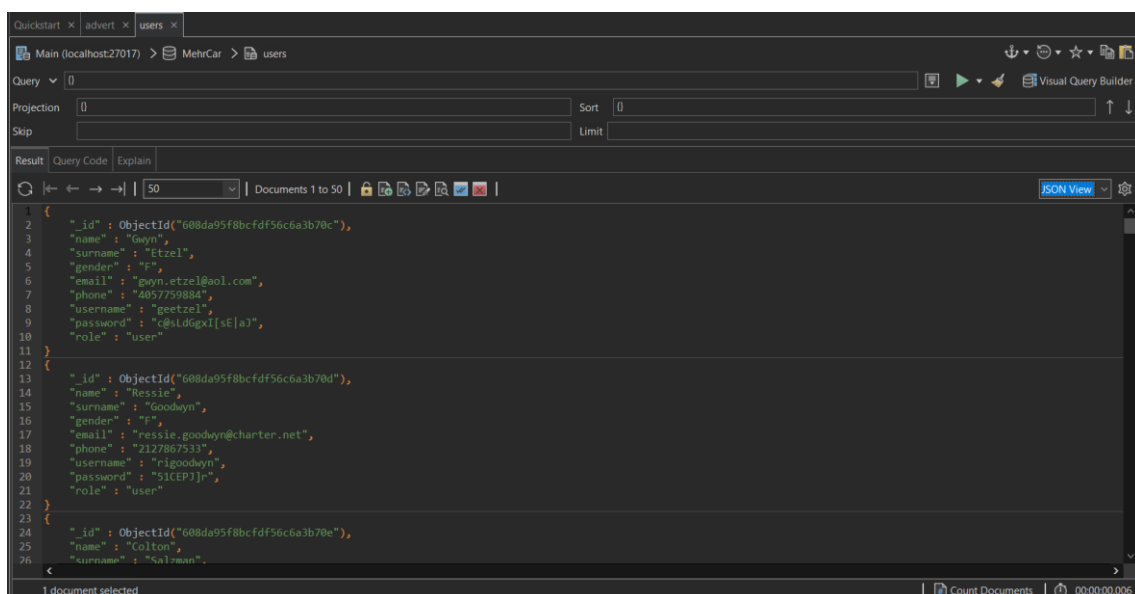
I have Two collections as follows:

- **Users:** In this collection, we store a document for each user in the database. The structure of the document is the following below:
 - **_id:** Representing the unique id for each user.
 - **password:** Representing the password.
 - **name:** Representing the first name.
 - **surname:** Representing the last name.
 - **email:** Representing the Email address.
 - **gender:** Representing the gender in a single character.
 - **email:** Representing the email.
 - **phone:** Representing the cell phone number for unique user
 - **role:** Representing the role of the user that show's this user is Admin or it is a standard user

Here is an example of a user's document:



_id	name	surname	gender	email	phone	username	password	role
608da95f8bcbdf...	Gwyn	Etzel	F	gwyn.etzel@aol...	4057759884	geetzel	c@stdgxl[sEia]	user
608da95f8bcbdf...	Ressie	Goodwyn	F	ressie.goodwyn...	2127867533	rigoodwyn	51CEPjy	user
608da95f8bcbdf...	Colton	Salzman	M	colton.salzman...	3857750225	cbsalzman	lIHspr@#5Dk7sq	user
608da95f8bcbdf...	Marilynn	Ealey	F	marilynn.ealey@...	216948974	mgealey	Qelhy6	user
608da95f8bcbdf...	Bula	Reich	F	bula.reich@gmail...	2173078597	breich	6_<Ofu\	user
608da95f8bcbdf...	Irwin	River	M	irwin.river@gmail...	2013121614	icriver	*MO@Ay+	user
608da95f8bcbdf...	Larisa	Still	F	larisa.still@exo...	3392894655	twstill	19VEzHt(Gz	user
608da95f8bcbdf...	Neil	Gebhard	M	neil.gebhard@g...	2183147370	ngebhard	GMqF%y8u	user
608da95f8bcbdf...	Kermit	Mcphail	M	kermit.mcphail@...	2092523832	kmcphail	DWDJ/5li&6<	user
608da95f8bcbdf...	Julio	Dockery	F	julio.dockery@e...	3078726444	jdockery	Qz2Rzvm	user
608da95f8bcbdf...	Tressa	Standard	F	tressa.standard...	4794932345	tostandard	*ds\ H&XS	user
608da95f8bcbdf...	Dwayne	Marker	M	dwayne.marker...	3196582317	demarkar	638yAd*	user
608da95f8bcbdf...	Ayako	Bachmann	F	ayako.bachmann...	3399222190	albachmann	03XZdmUHB	user
608da95f8bcbdf...	Salena	Santillan	F	salena.santillan...	2314826750	santillan	HMMOM7Xyn_JfI	user
608da95f8bcbdf...	Nelson	Dubuc	M	nelson.dubuc@e...	3146136146	nodubuc	OGE-zd7	user
608da95f8bcbdf...	Lincoln	Demaria	M	lincoln.demaria...	2195454790	lddemaria	qi-KJ-17T7	user
608da95f8bcbdf...	Eleonora	Pollard	F	eleonora.pollard...	3147542773	espollard	Hs-YytrZ8.4V;	user
608da95f8bcbdf...	Rosina	Gonsalves	F	rosina.gonsalves...	239859760	rmgonalves	8c0pzDrJnCpH	user
608da95f8bcbdf...	Douglas	Cull	M	douglas.cull@g...	2184971411	dmcul	IR0WCssL	user



```
{
  "_id": "608da95f8bcbdf56c6a3b70e",
  "name": "Gwyn",
  "surname": "Etzel",
  "gender": "F",
  "email": "gwyn.etzel@aol.com",
  "phone": "4057759884",
  "username": "geetzel",
  "password": "c@stdgxl[sEia]",
  "role": "user"
}
```


- **advert:** Representing the Advertisements in a document with the following fields. The structure of the document is the following below:

- **Advert_id:** Representing the unique id for each advertisement.
- **Posting_date:** Representing the date that the Advertisement posted by the user.
- **Car:** In this document, we store a document for each published car in the database.

For Car Embedded document we have this fields:

- **Car_id:** Representing the unique id for each car.
- **Price:** Representing the Price of car that user offers.
- **Year:** Representing the year of the car that it has been Produced.
- **region:** Representing the location of each car are belong to which city.
- **manufacturer:** Representing the type of the car's company (like Ford, Chevrolet, BMW, ...)
- **fuel:** Representing the type of the car's fuel consumption
- **model:** Representing the type of the Car's Model.
- **transmission:** Represent the transmission of the car
- **Type:** Representing the type of the Car's Chassis.
- **paint_color:** Representing the type of the Car's color.
- **description:** Representing the description for the car.
- **User:** In this Embedded document, we store a short document for each user in the database.

For User Embedded Document We Have this fields:

- **User_id:** Representing the unique id for each user.
- **name:** Representing the first name.
- **surname:** Representing the last name.
- **gender:** Representing the gender in a single character.
- **phone:** Representing the cell phone number for unique user

_id	posting_date	car	user
60ae66fbd4eccc41ac49e23a	2020-12-02T02:1...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e23b	2020-12-01T19:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e23d	2020-12-01T12:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e23e	2020-12-01T07:2...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e23f	2020-11-30T13:3...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e240	2020-11-29T07:3...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e241	2020-11-28T07:2...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e243	2020-11-27T12:1...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e244	2020-11-27T07:2...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e245	2020-11-26T12:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e246	2020-11-26T10:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e247	2020-11-25T14:4...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e248	2020-11-25T11:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e249	2020-11-25T07:0...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e24a	2020-11-24T12:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e24b	2020-11-24T07:2...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e24c	2020-11-23T15:0...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e24d	2020-11-23T14:5...	{ 11 fields }	{ 5 fields }
60ae66fbd4eccc41ac49e24e	2020-11-23T13:2...	{ 11 fields }	{ 5 fields }

Quickstart x IntelliShell: Main x advert x users x

Main (localhost:27017) > MehrCar > advert

Query { } Projection { } Sort { } Skip Limit { } Visual Query Builder

Result Query Code Explain

50 Documents 1 to 50 JSON View

```
1 {
2   "_id" : ObjectId("60ae66fbdaeccc41ac49e23a"),
3   "posting_date" : "2020-12-02T02:11:50-0600",
4   "car" : {
5     "_id" : ObjectId("6051ed390a85c466ff1347f5"),
6     "region" : "auburn",
7     "price" : NumberInt(7500),
8     "year" : "2014",
9     "manufacturer" : "hyundai",
10    "model" : "sonata",
11    "fuel" : "gas",
12    "transmission" : "automatic",
13    "type" : "sedan",
14    "paint_color" : null,
15    "description" : "I'll move to another city and try to sell my car. The car is in very good condition, everything works and fully cleaned. It equipped with a heater",
16  },
17  "user" : {
18    "_id" : ObjectId("608da95f8bcfdf56c6a3b70d"),
19    "name" : "Ressie",
20    "surname" : "Goodwyn",
21    "phone" : "2127867533",
22    "gender" : "F"
23  }
24 }
25 {
26   "_id" : ObjectId("60ae66fbdaeccc41ac49e23b").
```

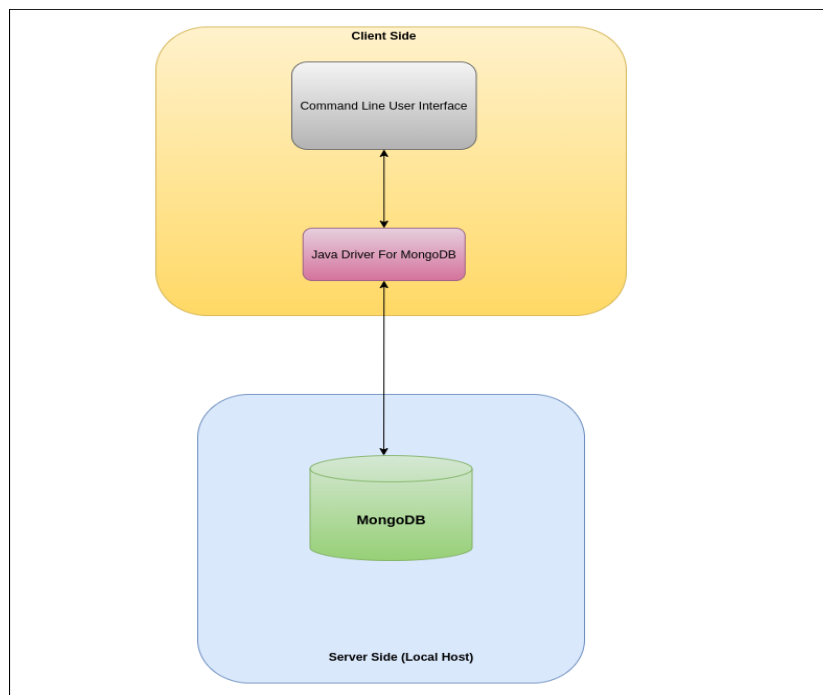
1 document selected | Count Documents | 00:00:00.017

SOFTWARE ARCHITECTURE: DESIGN AND IMPLEMENTATION

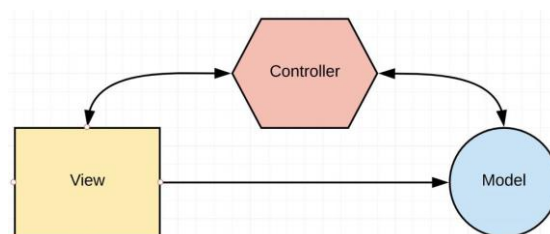
SOFTWARE ARCHITECTURE

This application is a client-server application which server is deployed locally. In addition, On the client-side, a command-line user interface, and users can have interaction with the application. Moreover, a Java driver has been used to bridge data between the CUI (command-line user interface) and the server-side: this driver is used for handling MongoDB connections. On the server-side, there is a document database

The document database is a MongoDB replica set. The MongoDB replica set consists of one primary and two secondary replicas to ensure eventual consistency.



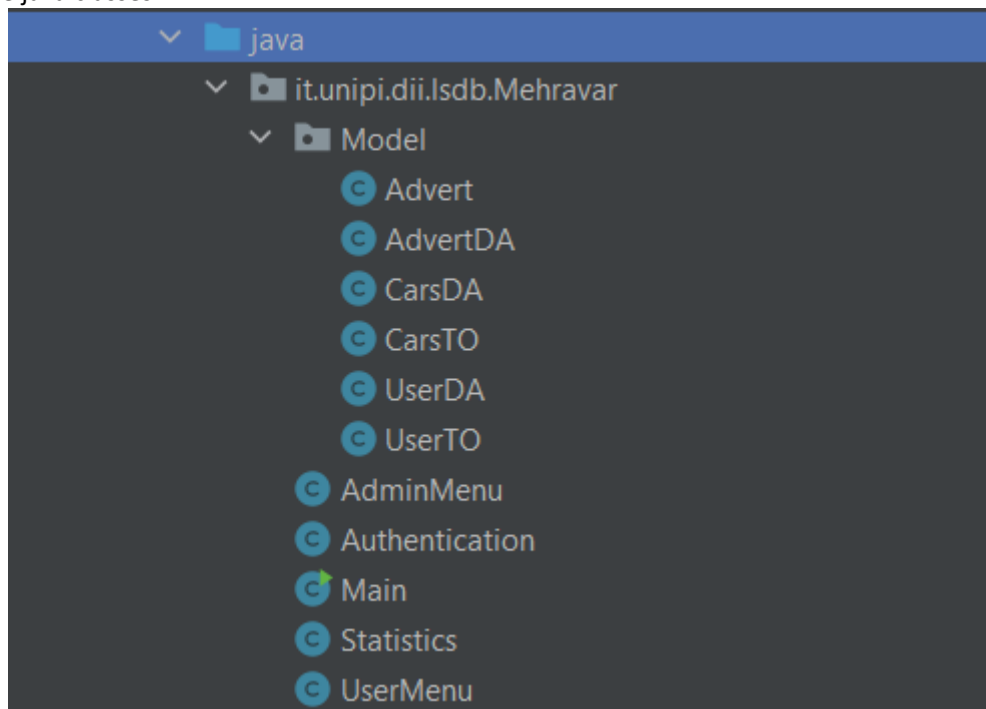
In addition, the architecture of my Software Design Pattern is MVC. The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.



IMPLEMENTATION PHASE

The final implementation of the Java application project consists of 1 module and 11 classes written in Java (in the “java” folder)

These are the java classes:



The application is based on **one main modules** as follows:

- **Model:** This module corresponds to the model in the MVC software design pattern that I have implemented in my code.
 - **Advert:** Representing the model for the advert.
 - **AdvertDA:** Representing the advert data access.
 - **CarsDA:** Representing the cars data access.
 - **CarsTO:** Representing the cars table object.
 - **UserDA:** Representing the user data access
 - **UserTO:** Representing the user table object.

In addition, these are the other classes that I have implemented in the application code.

- **AdminMenu:** Representing the menu for the admin
- **Authentication:** Representing the authentication of the system.
- **Main:** Representing the main class where the application runs.
- **Statistics:** Representing the statistics of the database for the admin.
- **UserMenu:** Representing the menu for the user.

MOST RELEVANT QUERIES USING MONGO DB

In this section, the most relevant queries performed with MongoDB are briefly explained and discussed. i will show only the queries.

ANALYTIC 1: RANKING OF THE Top 3 Regions

DESCRIPTION: Find the top 3 Regions by submitted Cars, find top 3 Regions depending on submitted cars (where top 3 means the which Regions that published more cars (Adverts)).

- This is the code for this analytic:

```
// top 3 regions
System.out.println("*****Top 3 Regions!*****");
advert.aggregate(
    Arrays.asList(
        Aggregates.group( "$car.region", Accumulators.sum( "count", "expression: 1")),
        new Document().append("$sort", new Document().append("count", -1)),
        new Document().append("$limit", 3)
    )
).forEach(document ->{ top3Regions.add(document.getString( "key: "_id")); System.out.println("Region Name: " + document.getString( "key: "_id") + " " +document.getInteger( "key: "count"));});
```

ANALYTIC 2: Average Price in Top 3 Regions

Description: Average Price in Top 3 Regions (where top 3 means the ones that have more Adverts and I calculate the average Price between all adverts in this top 3 Regions).

- This is the code for this analytic:

```
// average price in top 3 regions
System.out.println("-----Average Price in Top 3 Regions!-----");
for (String r:top3Regions) {
    advert.aggregate(
        Arrays.asList(
            Aggregates.match(new Document().append("car.region", r)),
            Aggregates.group( id: "$car.region", Accumulators.avg( fieldName: "average", expression: "$car.price"))
        )
    ).forEach(document -> System.out.println("Region: " + document.getString( key: "_id") + " Avg Price: " + document.getDouble( key: "average")));
}
```

ANALYTIC 3: Favorite Brands for Men and Women

Description: Favourite Brands for Men and Women (where most favourite Brands the ones that have more interested between Men and Women in all Adverts).

- This is the code for this analytic:

```
// Favorite brands for men
System.out.println("-----Favorite Brands For Men!-----");
advert.aggregate(
    Arrays.asList(
        Aggregates.match(new Document().append("user.gender", "M")),
        Aggregates.group( id: "$car.manufacturer", Accumulators.sum( fieldName: "count", expression: 1)),
        new Document().append("$sort", new Document().append("count", -1)),
        new Document().append("$limit", 3)
    )
).forEach(document -> {favoriteBrandsForMen.add(document.getString( key: "_id")); System.out.println("Brand Name: "+document.getString( key: "_id") + " Count: " + document.getInteger( key: "count"));});

// Favorite brands for women
System.out.println("-----Favorite Brands For Women!-----");
advert.aggregate(
    Arrays.asList(
        Aggregates.match(new Document().append("user.gender", "F")),
        Aggregates.group( id: "$car.manufacturer", Accumulators.sum( fieldName: "count", expression: 1)),
        new Document().append("$sort", new Document().append("count", -1)),
        new Document().append("$limit", 3)
    )
).forEach(document -> {favoriteBrandsForWomen.add(document.getString( key: "_id")); System.out.println("Brand Name: "+document.getString( key: "_id") + " Count: " + document.getInteger( key: "count"));});
```

ANALYTIC 4: -The Most Expensive Car in Men's and Women's Favorites Brand

Description: The Most Expensive Car in Men's and Women's Favourites Brand (where most favourites Brands the ones that have more interested between Men and Women in all Adverts and I calculate the Highest Price between them).

- This is the code for this analytic:

```
// The most expensive car in mens' favorite car
System.out.println("-----The Most Expensive Car in Mens' Favorite Brand!-----");
for (String mb:favoriteBrandsForMen) {
    advert.find(new Document().append("car.manufacturer", mb).append("user.gender", "M")).sort(new Document().append("car.price", -1)).limit(1)
        .forEach(document -> {System.out.println("Model: "+((Document)document.get("car")).getString( key: "model") + " price: " +((Document)document.get("car")).getInteger( key: "price" )});});
}

// The most expensive car in womens' favorite car
System.out.println("-----The Most Expensive Car in Women's' Favorite Brand!-----");
for (String wb: favoriteBrandsForWomen) {
    advert.find(new Document().append("car.manufacturer", wb).append("user.gender", "F")).sort(new Document().append("car.price", -1)).limit(1)
        .forEach(document -> {System.out.println("Model: "+((Document)document.get("car")).getString( key: "model") + " price: " +((Document)document.get("car")).getInteger( key: "price" )});});
}
```

ANALYTIC 5: RANKING OF THE TOP 3 Manufacturer

Description: Find the top 3 Manufacturer by submitted cars (where top 3 means the ones that have more Cars).

- This is the code for this analytic:

```
//Top 3 Manufacturer
System.out.println("-----Top 3 Manufacturer by submitted cars -----");
cars.aggregate(Arrays.asList(
    Aggregates.group( id: "$manufacturer", Accumulators.sum( fieldName: "count", expression: 1)),
    new Document().append("$sort", new Document().append("count", -1)),
    new Document().append("$limit", 3)
)).forEach(doc -> System.out.println("Manufacturer: " + doc.getString( key: "_id") + " " + "submitted cars: " + doc.getInteger( key: "count")));
```

ANALYTIC 6: RANKING OF 5 most popular paint Color

Description: Find all the car paint colors that have the 5 most popular paint Color on our platform

- This is the code for this analytic:

```
//5 most popular paint Color
System.out.println("----- 5 most popular paint Color in our platform -----");
cars.aggregate(Arrays.asList(
    Aggregates.group( id: "$paint_color", Accumulators.sum( fieldName: "count", expression: 1)),
    new Document().append("$sort", new Document().append("count", -1)),
    new Document().append("$limit", 5)
)).forEach(doc -> System.out.println("Paint Color: " + doc.getString( key: "_id") + " " + " number of cars with this Color: " + doc.getInteger( key: "count")));

System.out.println("*****");
```


INDEXES

One of the non-functional requirements for the application is the fast response. In this case, indexing is of great importance, because it aids in an efficient execution of read queries. In the final version of my project I decided to have two single property indexes:

- An **index for the *phone* in the *Cars* collection**, sorted in descending order
- An **index for the *phone* in the *User* collection**, sorted in descending order

In the following subsections, I briefly explain the choice of introducing these indexes and I show some tests just for the phone in Cars collection done in order to verify their benefits.

MONGODB INDEXES

In all the queries on Cars collection, I sort the retrieved cars from the newest to old published cars, so it proved to be a good approach to index *phone* in a descending order.

Attached below is an example test query used to experiment my approach:

```
> db.cars.find({"phone":"4057759884"}).explain("executionStats")
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "MehrCar.cars",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "phone" : {
        "$eq" : "4057759884"
      }
    }
  }
```

I performed this simple query to test the performance of execution of query with and without indexing.

The execution time without indexing during sort operation is 8414 ms:

```
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 4,
    "executionTimeMillis" : 8414,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 458268,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "phone" : {
          "$eq" : "4057759884"
        }
      }
    }
  },
  "nReturned" : 4,
  "executionTimeMillisEstimate" : 6832,
  "works" : 458270,
  "advanced" : 4,
  "needTime" : 458265,
  "needYield" : 0,
  "saveState" : 565,
```

Let me make the *phone* field indexed now:

```
> db.cars.createIndex({phone:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

With indexing the execution time reduced by more than 95 percent of the time before indexing. The attached picture supports my choice:

```
> db.cars.find({"phone":"4057759884"}).explain("executionStats")
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "MehrCar.cars",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "phone" : {
        "$eq" : "4057759884"
      }
    }
  },
  "winningPlan" : {
    "stage" : "FETCH",
    "inputStage" : {
      "stage" : "IXSCAN",
      "keyPattern" : {
        "phone" : 1
      },
      "indexName" : "phone_1",
      "isMultiKey" : false,
      "multiKeyPaths" : {
        "phone" : [ ]
      },
      "isUnique" : false,
      "isSparse" : false,
      "isPartial" : false,
      "indexVersion" : 2,
      "direction" : "forward",
      "indexBounds" : {
        "phone" : [
          ["4057759884\","4057759884\"]
        ]
      }
    }
  },
  "rejectedPlans" : [ ]
},
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 4,
    "executionTimeMillis" : 3,
```

TEST ON AVAILABILITY OF THE APPLICATION

One of the main non-functional requirements of my application, is the availability. In particular, I want that even in case of failure of the primary, the application continues working in a properly manner and a user can still use it.

To prove that my requirement is satisfied, I performed a test shutting down the primary and making read requests with the application. **I proved that even if the primary fails, the application continues to offer its service to the user. Applying the same reasoning for the other logical servers in different ports, I can say that, since I have three replicas, I can handle two server crashes. In this last case, the application will still work using only one logical server (in other ports of local host).** A little note must be done: it could happen that the primary fails when a writing operations is being done. In this case, if the write operation isn't still being replicated among the secondary logical servers, the write update will be lost.

In the following lines, I will briefly discuss about the test I performed.

First of all, I should talk about the configuration of my mongoDB replica set. I set it as is shown in the following image:

```
"members" : [
  {
    "_id" : 0,
    "name" : "localhost:27018",
    "health" : 1,
    "state" : 1,
    "stateStr" : "PRIMARY",
    "uptime" : 36,
    "optime" : {
      "ts" : Timestamp(1625077515, 1),
      "t" : NumberLong(5)
    },
    "optimeDate" : ISODate("2021-06-30T18:25:15Z"),
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "electionTime" : Timestamp(1625077495, 1),
    "electionDate" : ISODate("2021-06-30T18:24:55Z"),
    "configVersion" : 1,
    "configTerm" : 5,
    "self" : true,
    "lastHeartbeatMessage" : ""
  },
  {
    "_id" : 1,
    "name" : "localhost:27019",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 25,
    "optime" : {
      "ts" : Timestamp(1625077505, 1),
      "t" : NumberLong(5)
    },
    "optimeDurable" : {
      "ts" : Timestamp(1625077505, 1),
      "t" : NumberLong(5)
    },
    "optimeDate" : ISODate("2021-06-30T18:25:05Z"),
    "optimeDurableDate" : ISODate("2021-06-30T18:25:05Z"),
    "optimeDuration" : ISODate("2021-06-30T18:25:05Z"),
    "lastHeartbeat" : ISODate("2021-06-30T18:25:15.236Z"),
    "lastHeartbeatRecv" : ISODate("2021-06-30T18:25:14.272Z"),
    "pingMs" : NumberLong(0),
    "lastHeartbeatMessage" : "",
    "syncSourceHost" : "localhost:27018",
    "syncSourceId" : 0,
    "infoMessage" : "",
    "configVersion" : 1,
    "configTerm" : 5
  },
  {
    "id" : 2,
    "name" : "localhost:27020",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 12,
    "optime" : {
      "ts" : Timestamp(1625077505, 1),
      "t" : NumberLong(5)
    },
    "optimeDurable" : {
      "ts" : Timestamp(1625077505, 1),
      "t" : NumberLong(5)
    },
    "optimeDate" : ISODate("2021-06-30T18:25:05Z"),
    "optimeDurableDate" : ISODate("2021-06-30T18:25:05Z"),
    "lastHeartbeat" : ISODate("2021-06-30T18:25:15.237Z"),
    "lastHeartbeatRecv" : ISODate("2021-06-30T18:25:16.072Z"),
    "pingMs" : NumberLong(0),
    "lastHeartbeatMessage" : "",
    "syncSourceHost" : "localhost:27019",
    "syncSourceId" : 1,
    "infoMessage" : "",
    "configVersion" : 1,
    "configTerm" : 5
  }
],
```

So, as we can see, the local machine with IP address **127.0.0.1:27018** has the greater priority and, if available, will be always chosen as the primary during the leader election of the replica set.

Now I start the application and I perform some reading operations. Checking the status of the replica set, as I expect, I have as primary the local machine(port:27018) with highest priority.

Now, while the application is still running, I shut down the primary:

```
---
lsmdb:PRIMARY> use admin
switched to db admin
lsmdb:PRIMARY> db.shutdownServer()
server should be down...
>
```

The application is still working and I can correctly retrieve data of our database using it. **So, when a failure occurs in the primary, the application can still offer its service to the user.**

Connecting to the local machine with IP address **127.0.0.1:27019** we can see (through the `rs.status()` command) that now it has been elected as the primary, and that the local machine with IP address **127.0.0.1:27018** has a not reachable state and on the other hand the local machine with IP address **127.0.0.1:27020** shown as a **secondary** :

```
{
  "members" : [
    {
      "id" : 0,
      "name" : "localhost:27018",
      "health" : 0,
      "state" : 8,
      "stateStr" : "(not reachable/healthy)",
      "uptime" : 0,
      "optime" : {
        "ts" : Timestamp(0, 0),
        "t" : NumberLong(-1)
      },
      "optimeDurable" : {
        "ts" : Timestamp(0, 0),
        "t" : NumberLong(-1)
      },
      "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
      "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
      "lastHeartbeat" : ISODate("2021-06-29T21:05:18.984Z"),
      "lastHeartbeatRecv" : ISODate("2021-06-29T20:59:25.207Z"),
      "pingMs" : NumberLong(0),
      "lastHeartbeatMessage" : "Error connecting to localhost:27018 (127.0.0.1:27018) :: caused by :: Connection refused",
      "syncSourceHost" : "",
      "syncSourceId" : -1,
      "infoMessage" : "",
      "configVersion" : 1,
      "configTerm" : 3
    },
  ],
}
```

```

},
{
  "_id" : 1,
  "name" : "localhost:27019",
  "health" : 1,
  "state" : 1,
  "stateStr" : "PRIMARY",
  "uptime" : 419,
  "optime" : {
    "ts" : Timestamp(1625000714, 1),
    "t" : NumberLong(4)
  },
  "optimeDate" : ISODate("2021-06-29T21:05:14Z"),
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "infoMessage" : "",
  "electionTime" : Timestamp(1625000364, 1),
  "electionDate" : ISODate("2021-06-29T20:59:24Z"),
  "configVersion" : 1,
  "configTerm" : 4,
  "self" : true,
  "lastHeartbeatMessage" : ""
},
{
  "_id" : 2,
  "name" : "localhost:27020",
  "health" : 1,
  "state" : 2,
  "stateStr" : "SECONDARY",
  "uptime" : 394,
  "optime" : {
    "ts" : Timestamp(1625000714, 1),
    "t" : NumberLong(4)
  },
  "optimeDurable" : {
    "ts" : Timestamp(1625000714, 1),
    "t" : NumberLong(4)
  },

```