BIG DATA

TOO BIG TO IGNORE

SÜMEYYE KAYNAK

OUTLINE

NoSQL

MongoDB

NOSQL

- The increase in the volume and diversity of data makes the relational database concept inadequate.
- In relational database systems, data is kept in tables and columns.
- In NoSQL database systems, data is kept in Json format.
- Read and write operations in NoSQL databases are faster than relational databases.
- NoSQL systems can scale horizontally.

MONGODB

- MongoDB is an open-source NoSQL database developed in 2009.
- There are a lot of NoSQL database today like Cassandra, BigTable, Dynamo.
- In MongoDB, each record is expressed as a document. The document is stored in Json format.
- MongoDB is one of the most preferred NoSQL databases. Because it has driver support for many programming languages.

SHARDING

- Sharding is a method for distributing data across multiple machines.
- MongoDB uses sharding to support deployments with very large data sets and high throughput operations.
- There are two methods for addressing system growth: vertical and horizontal scaling.

SHARDING

- Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space.
- Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload.

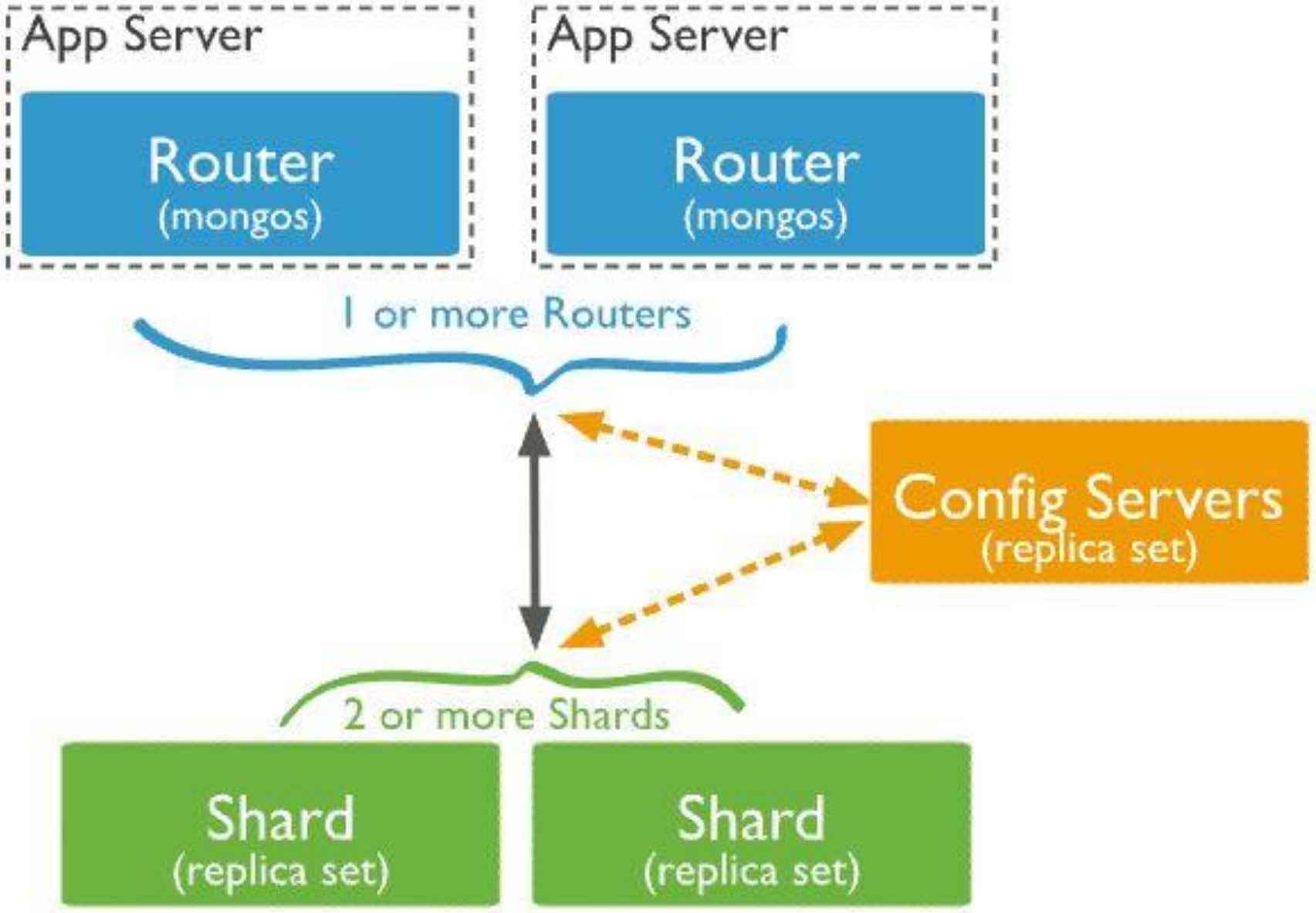
SHARDING

- Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.
- MongoDB supports horizontal scaling through sharding.

SHARDED CLUSTER

A MongoDB sharded cluster consists of the following components:

- Shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- Mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- Config servers: Config servers store metadata and configuration settings for the cluster.


App Server
Router
(mongos)
App Server
Router
(mongos)
1 or more Routers
Config Servers
(replica set)
2 or more Shards
Shard
(replica set)
Shard
(replica set)

ADVANTAGES OF SHARDING

- **Reads/Writes:** MongoDB distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations. Both read and write workloads can be scaled horizontally across the cluster by adding more shards.
- **Storage Capacity:** Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data. As the data set grows, additional shards increase the storage capacity of the cluster.

ADVANTAGES OF SHARDING

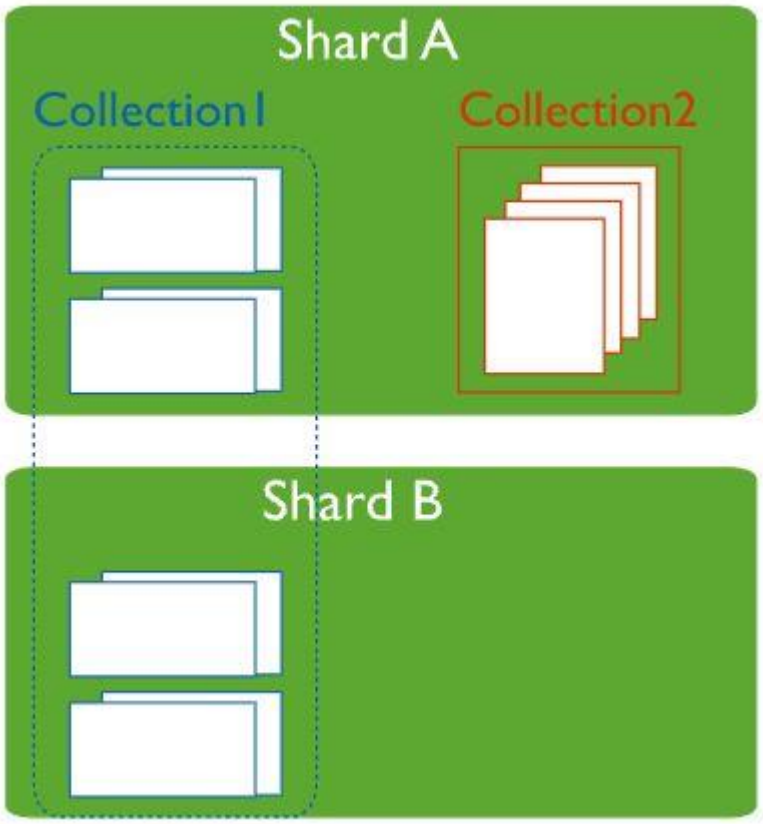
- **High Availability:** The deployment of config servers and shards as replica sets provide increased availability.
- Even if one or more shard replica sets become completely unavailable, the sharded cluster can continue to perform partial reads and writes. That is, while data on the unavailable shard(s) cannot be accessed, reads or writes directed at the available shards can still succeed.

CONSIDERING BEFORE SHARDING

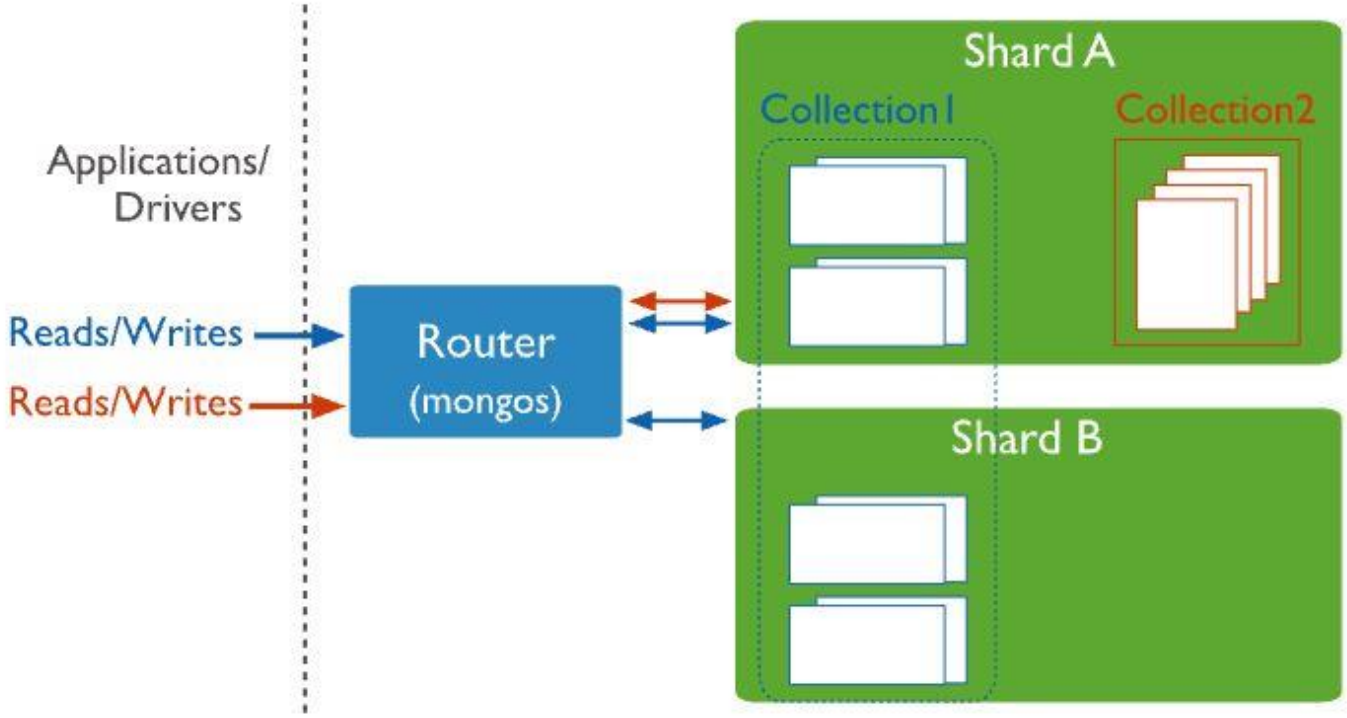
- Once a collection has been sharded, MongoDB provides no method to unshard a sharded collection.

SHARDED AND NON-SHARDED COLLECTIONS

- A database can have a mixture of sharded and unsharded collections.
- Sharded collections are partitioned and distributed across the shards in the cluster.
- Unsharded collections are stored on a primary shard. Each database has its own primary shard.



CONNECTING TO A SHARDED CLUSTER



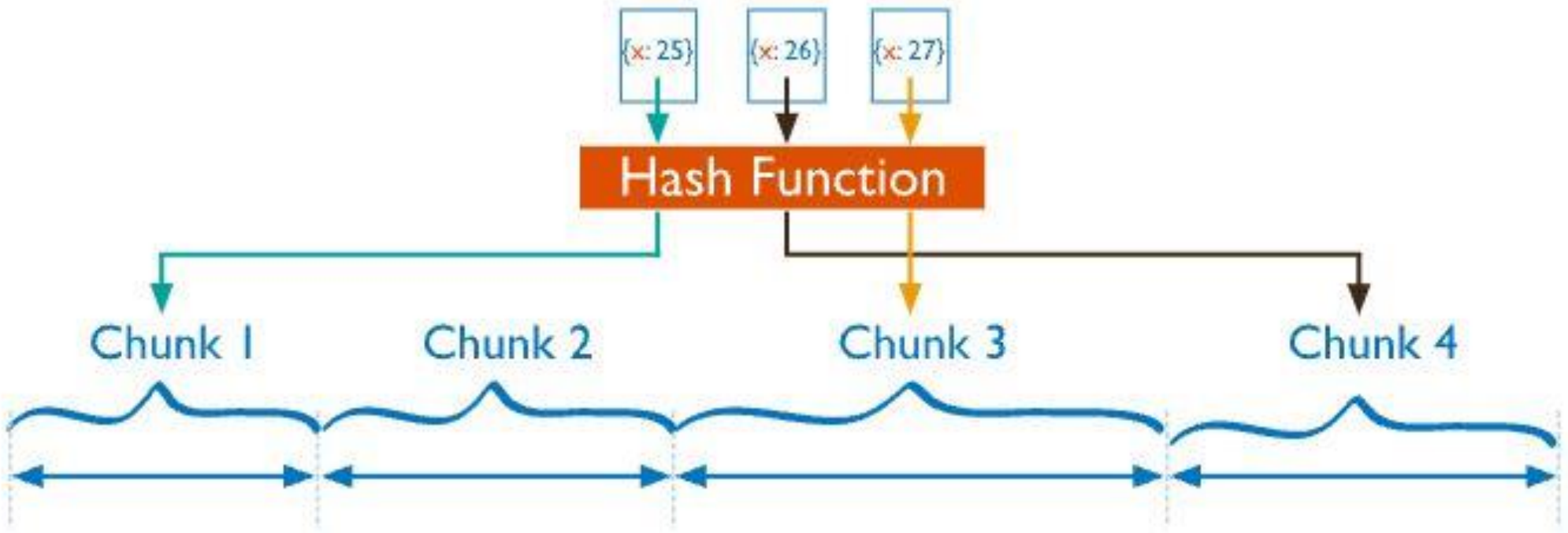
CHUNKS

- MongoDB partitions sharded data into chunks.
- A contiguous range of shard key values within a particular shard.
- Chunk ranges are inclusive of the lower boundary and exclusive of the upper boundary.
- MongoDB splits chunks when they grow beyond the configured chunk size, which by default is 64 megabytes.
- MongoDB migrates chunks when a shard contains too many chunks of a collection relative to other shards.

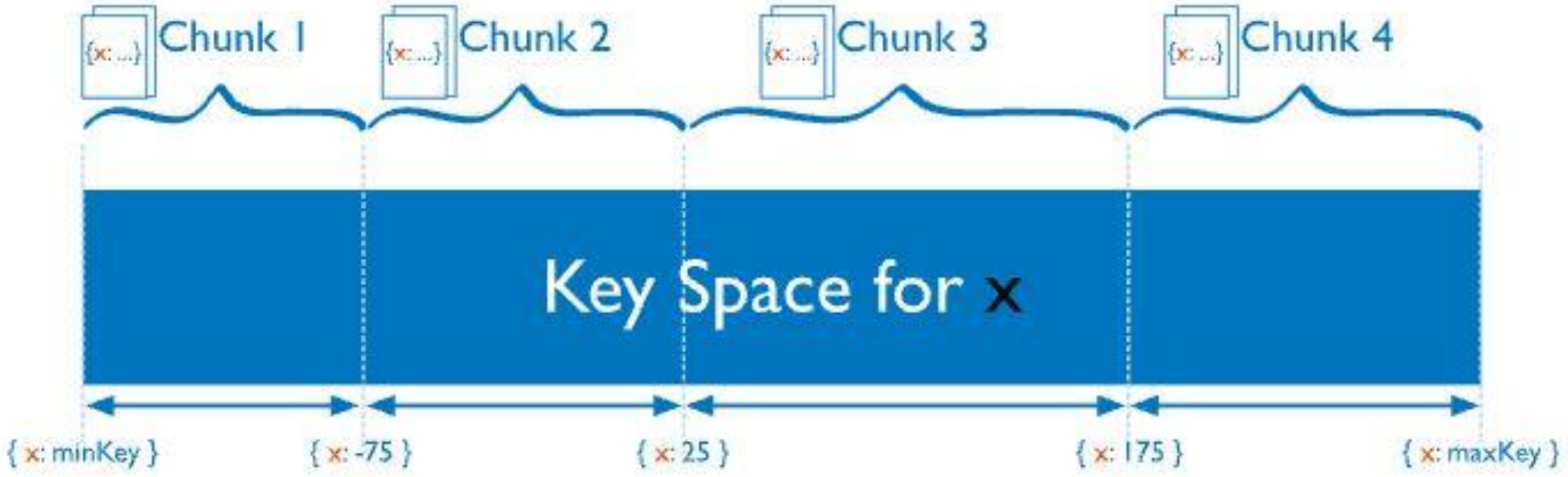
SHARDING STRATEGY

- Hashed Sharding
- Ranged Sharding

HASHED SHARDING

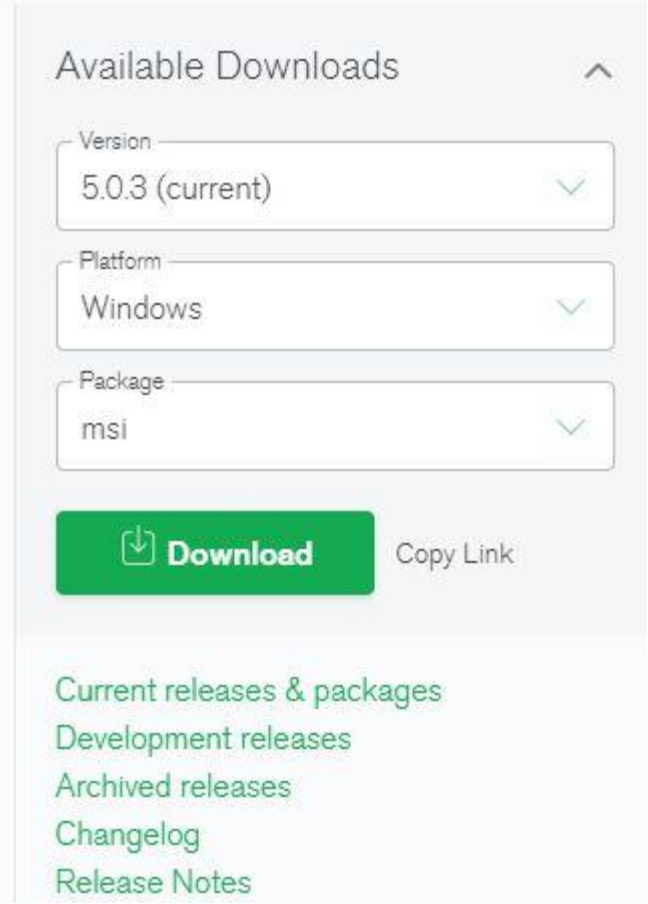


RANGED SHARDING

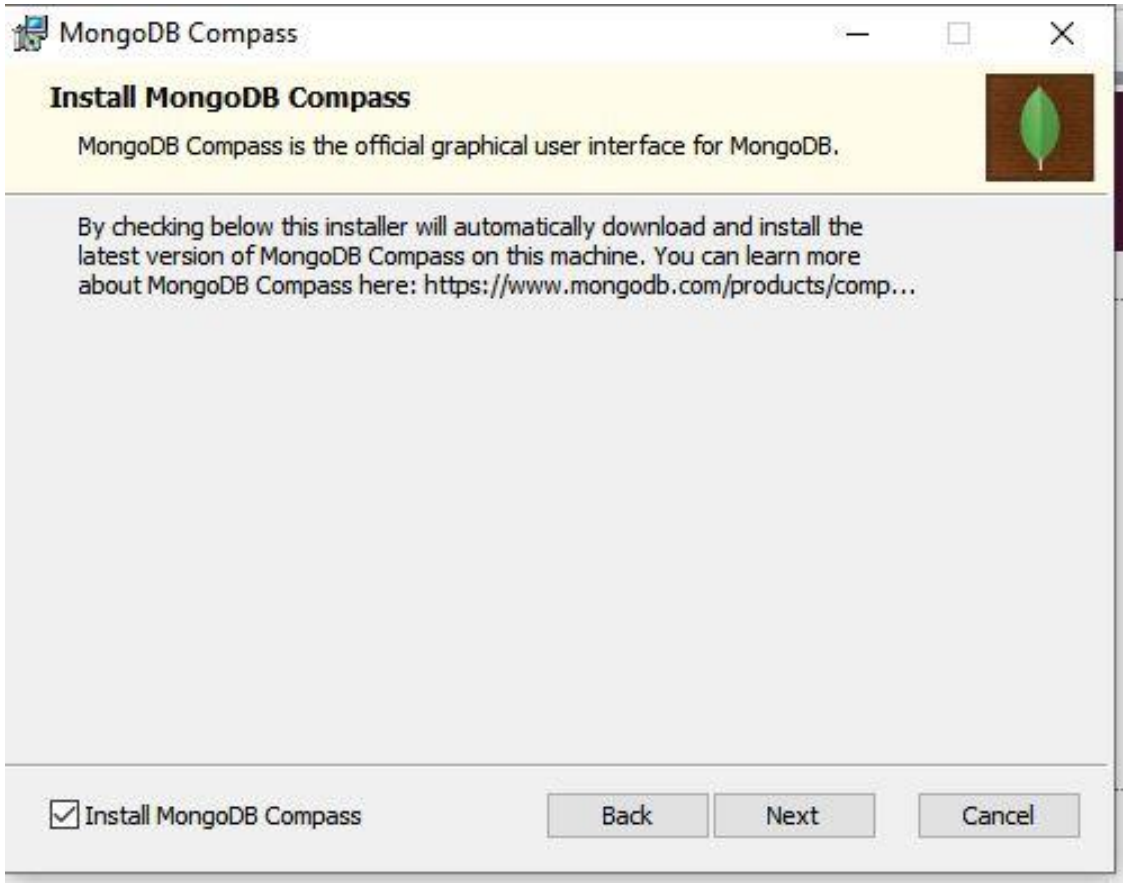


MONGODB INSTALLATION

- MongoDB Download



MONGODB INSTALLATION



MONGODB INSTALLATION

- Let's create a folder called mongodb under C: and create two files called mongo.config and mongo.log inside.
- Let's create a folder named “data” in the mongodb folder under C:
- Let's open the mongo.config file and inside the file

dbpath = C:\mongodb\data

logpath = C:\mongodb\mongo.log

Let's add the lines.

MONGODB INSTALLATION

To run the MongoDB server;

- Let's open cmd
- After coming to the mongo installation path on the command line

mongod.exe --config C:\mongodb\mongo.config

Type the line and press enter.

MONGODB INSTALLATION

Command Prompt

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sumeyye>cd..

C:\Users>cd..

C:\>cd "Program Files"

C:\Program Files>cd MongoDB

C:\Program Files\MongoDB>cd Server

C:\Program Files\MongoDB\Server>cd 5.0

C:\Program Files\MongoDB\Server\5.0>cd bin

C:\Program Files\MongoDB\Server\5.0\bin>mongod.exe --config C:\mongodb\mongo.config
```

CRUD OPERATION

Create databases

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sumeyye>cd..

C:\Users>cd..

C:\>cd "Program Files"

C:\Program Files>cd MongoDB

C:\Program Files\MongoDB>cd Server

C:\Program Files\MongoDB\Server>cd 5.0

C:\Program Files\MongoDB\Server\5.0>cd bin

C:\Program Files\MongoDB\Server\5.0\bin>mongo
MongoDB shell version v5.0.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4350277a-77af-4805-8956-22e4e0042752") }
MongoDB server version: 5.0.3
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
```

CRUD OPERATION

Create databases

```
        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> use firstexample
switched to db firstexample
>
```

```
> use firstexample
switched to db firstexample
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

```
> db
firstexample
>
```

CRUD OPERATION

Create collection

```
> db.createCollection("person");
{ "ok" : 1 }
> show collections
person
>
```

Insert data

```
> db.person.insert({adi: "Stephen Hawking"});
WriteResult({ "nInserted" : 1 })
>
```

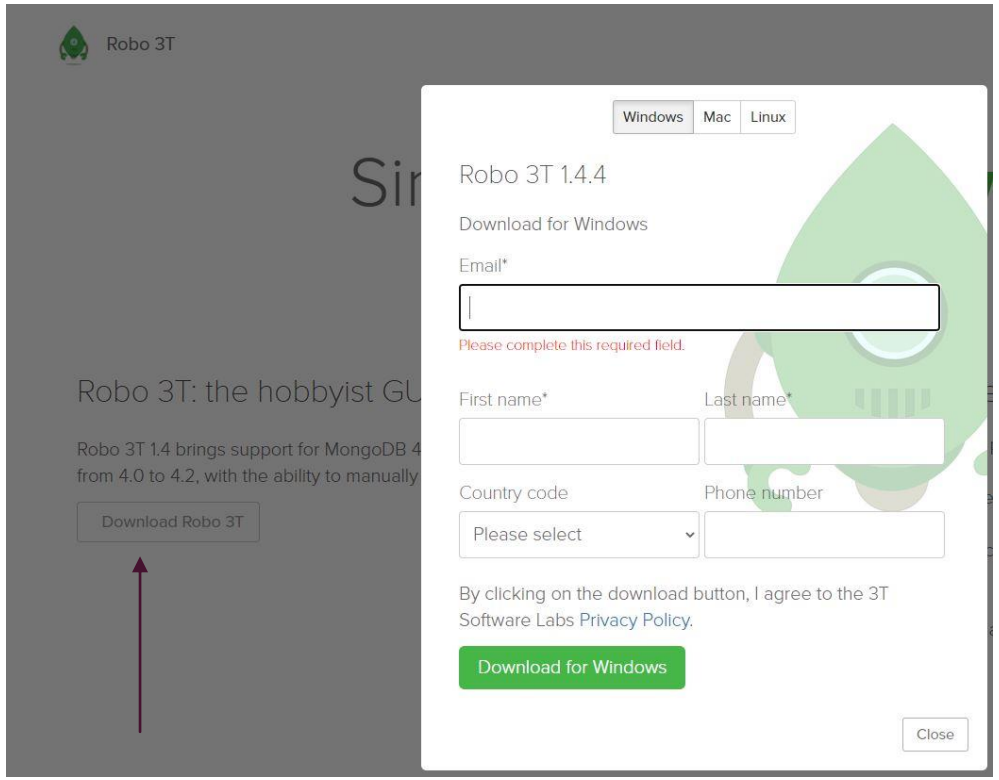
Filter data

```
> db.person.insert({adi: "Stephen Hawking"});
WriteResult({ "nInserted" : 1 })
> db.person.insert({adi: "Albert Einstein"});
WriteResult({ "nInserted" : 1 })
> db.person.insert({adi: "Isaac Newton"});
WriteResult({ "nInserted" : 1 })
> db.person.find();
{ "_id" : ObjectId("614c5726a2fb35a04712baf2"), "adi" : "Stephen Hawking" }
{ "_id" : ObjectId("614c577aa2fb35a04712baf3"), "adi" : "Albert Einstein" }
{ "_id" : ObjectId("614c5792a2fb35a04712baf4"), "adi" : "Isaac Newton" }
>
```

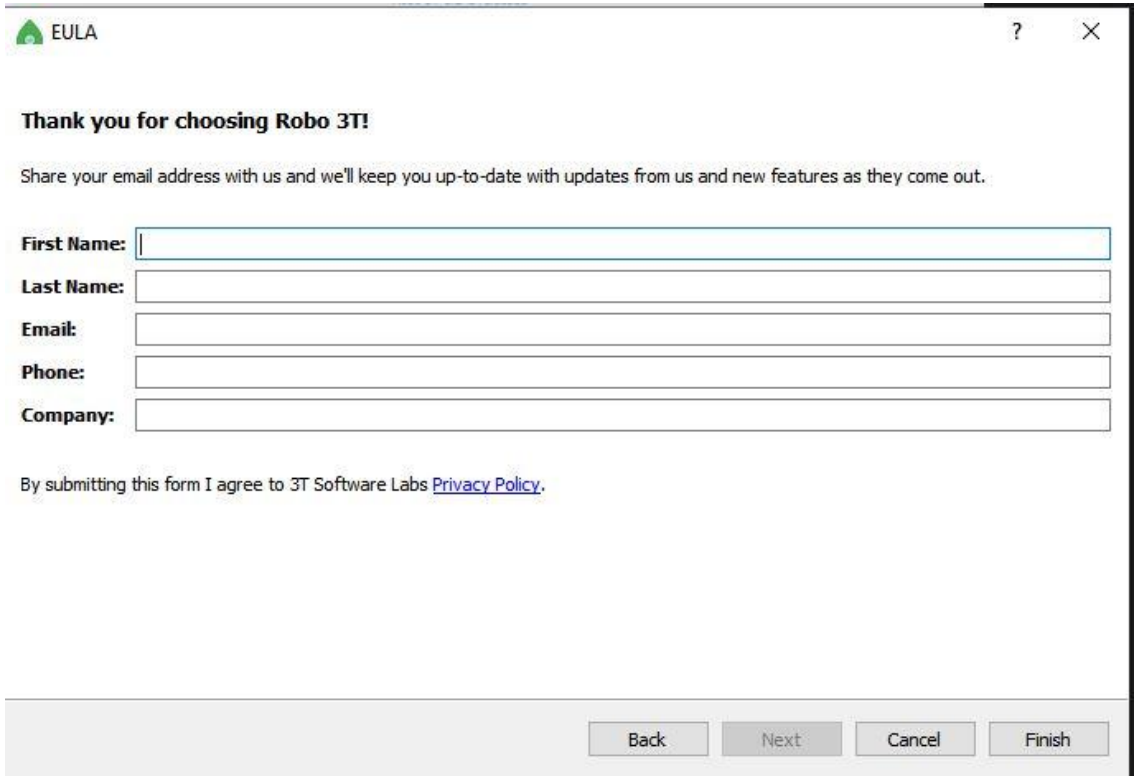
Filter data

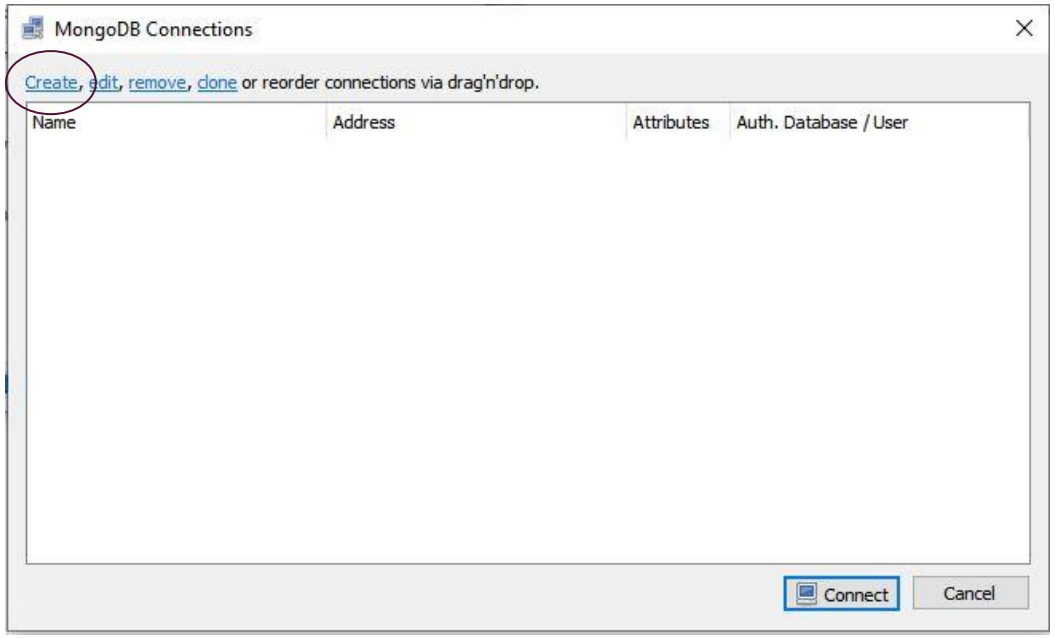
```
> db.person.find( {adi:"Isaac Newton"})
{ "_id" : ObjectId("614c5792a2fb35a04712baf4"), "adi" : "Isaac Newton" }
>
```

ROBOMONGO-ROBO 3T

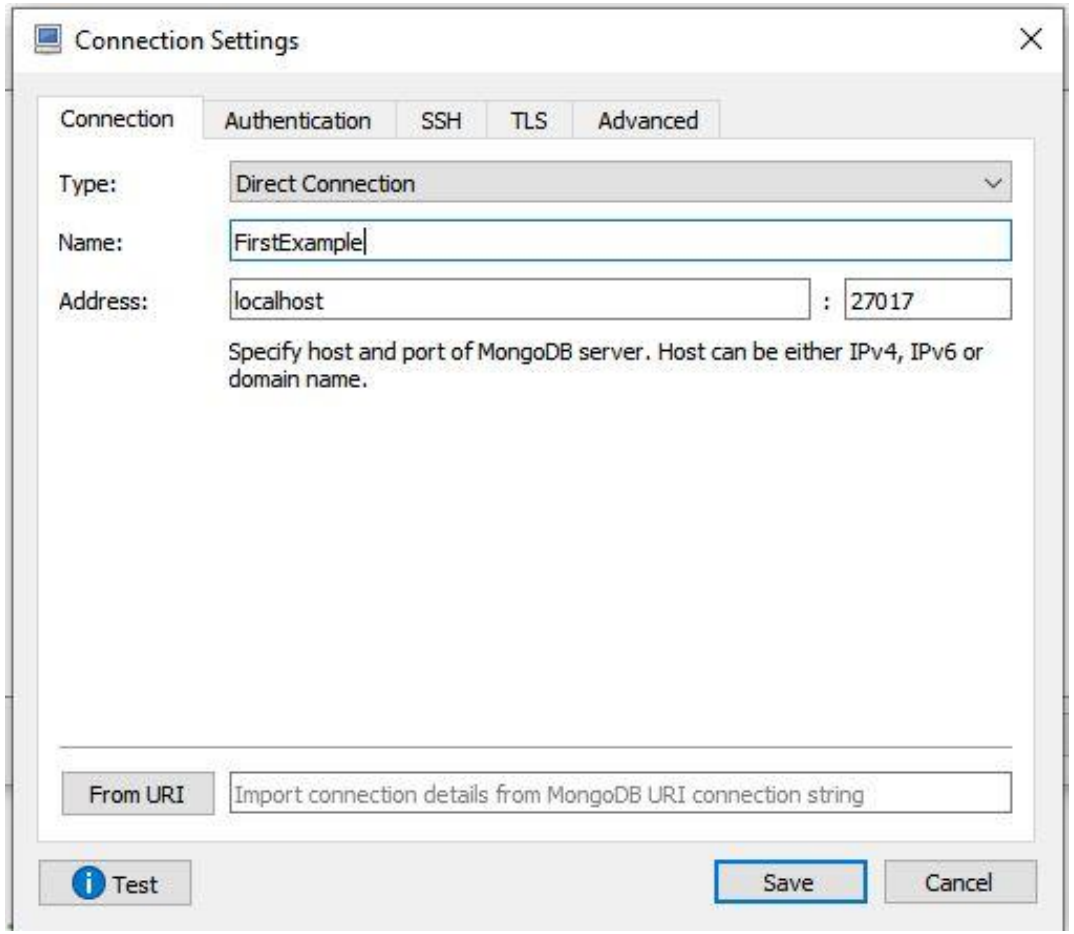


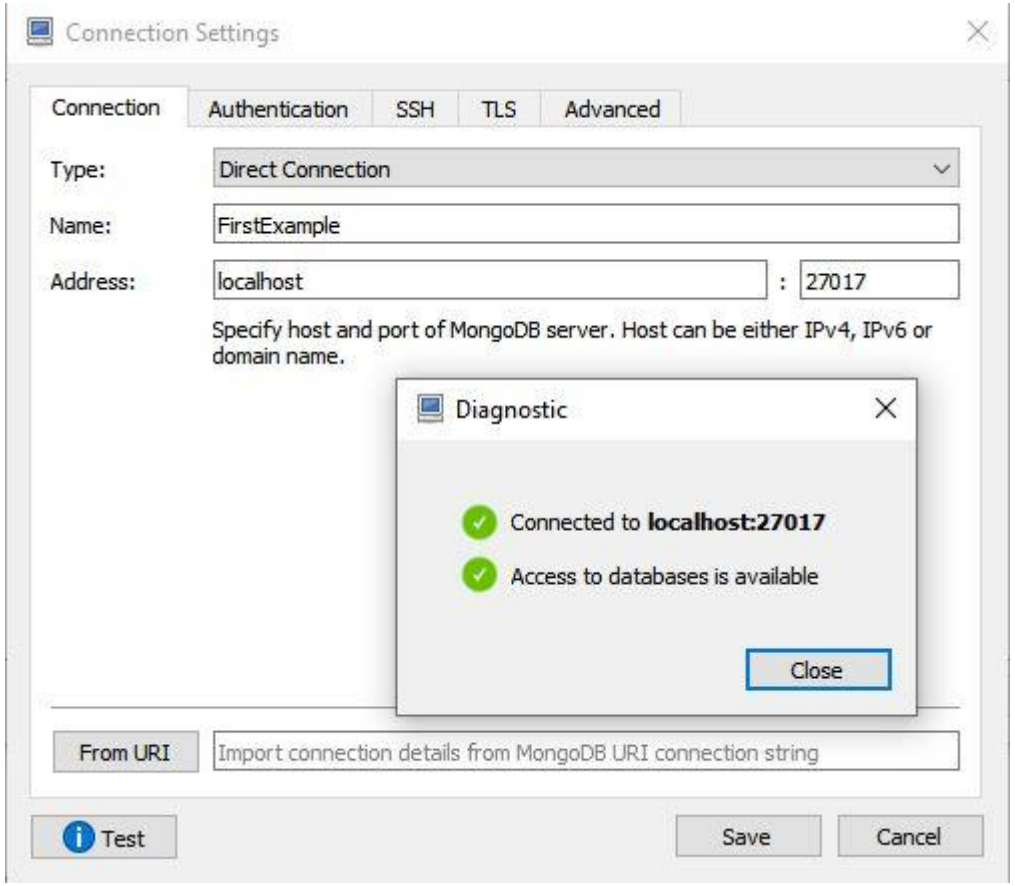
ROBOMONGO-ROBO 3T





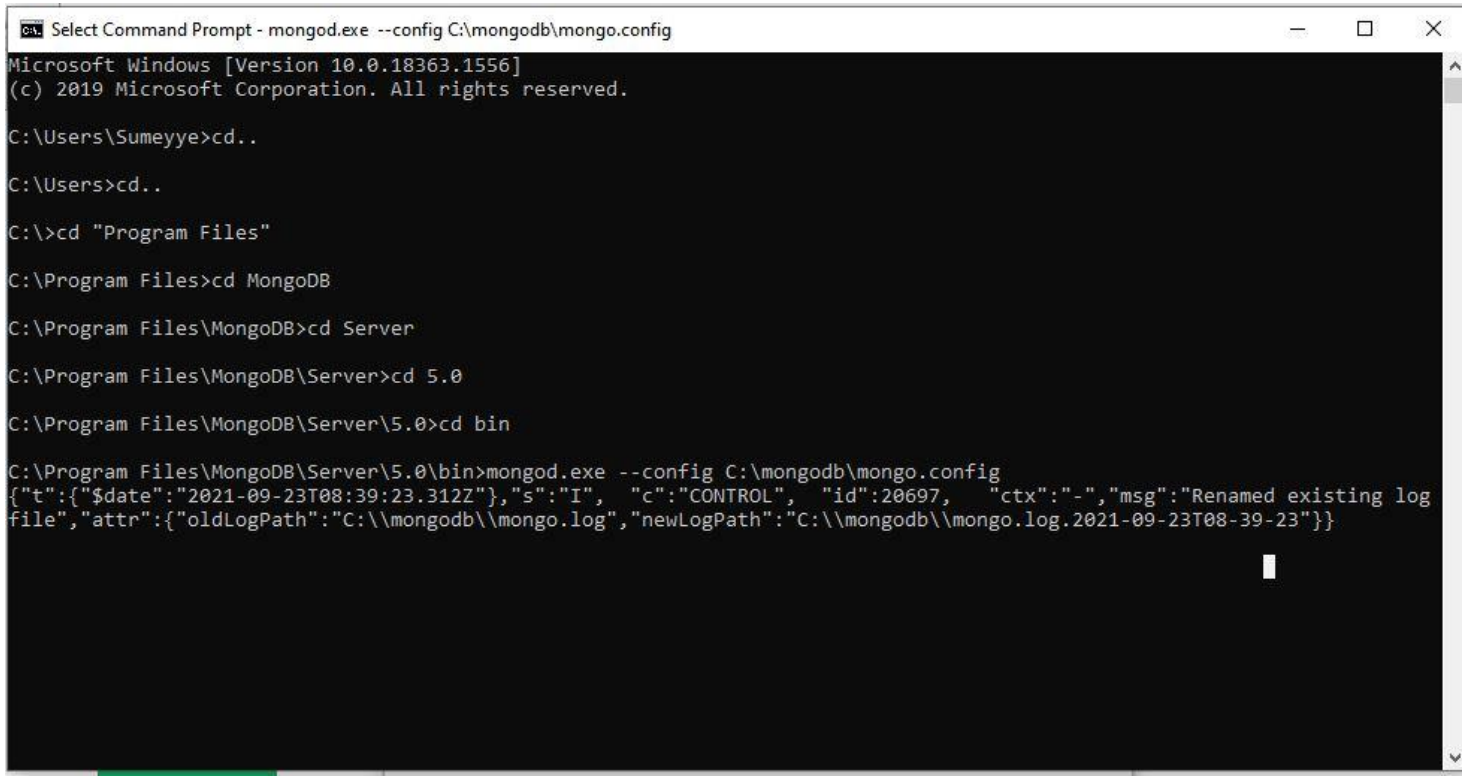
ROBOMONGO-ROBO 3T



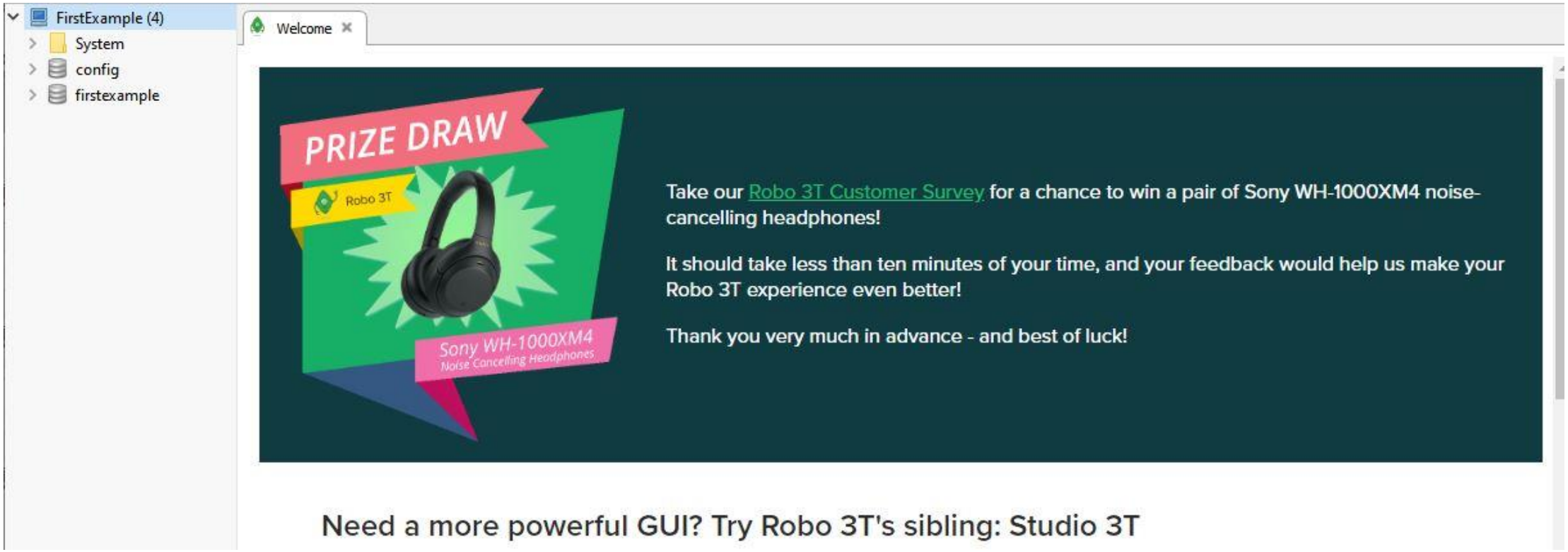


ROBOMONGO-ROBO 3T

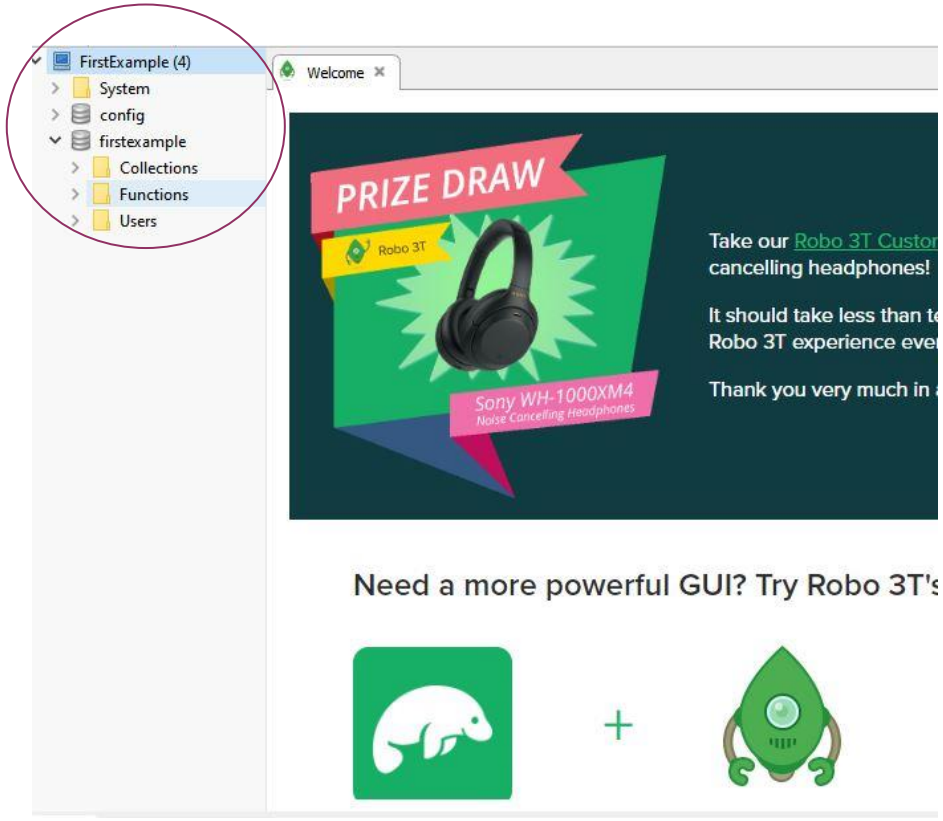
MongoDB server



ROBOMONGO-ROBO 3T



ROBOMONGO-ROBO 3T



ROBOMONGO-ROBO 3T

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

FirstExample | localhost:27017 | firstexample

```
db.getCollection('person').find({})
```

person | 0.002 sec. | 0 | 50

Key	Value	Type
(1) ObjectId("614c5726a2fb35a04712baf2")	{ 2 fields }	Object
_id	ObjectId("614c5726a2fb35a04712baf2")	ObjectId
adi	Stephen Hawking	String
(2) ObjectId("614c577aa2fb35a04712baf3")	{ 2 fields }	Object
_id	ObjectId("614c577aa2fb35a04712baf3")	ObjectId
adi	Albert Einstein	String
(3) ObjectId("614c5792a2fb35a04712baf4")	{ 2 fields }	Object
_id	ObjectId("614c5792a2fb35a04712baf4")	ObjectId
adi	Isaac Newton	String

ROBOMONGO-ROBO 3T

Commands are written here.

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

FirstExample localhost:27017 firstexample

```
db.getCollection('person').find({})
```

person 0.002 sec.

run

ample (4)
em
fig
example
ollections (1)
person
Indexes (1)

* db.getCollection('person').fin...

FirstExample localhost:27017 firstexample

```
db.person.insert({adi: "Thomas Edison", dt: "11.02.1847"})
```

person 0.002 sec.

Key Value

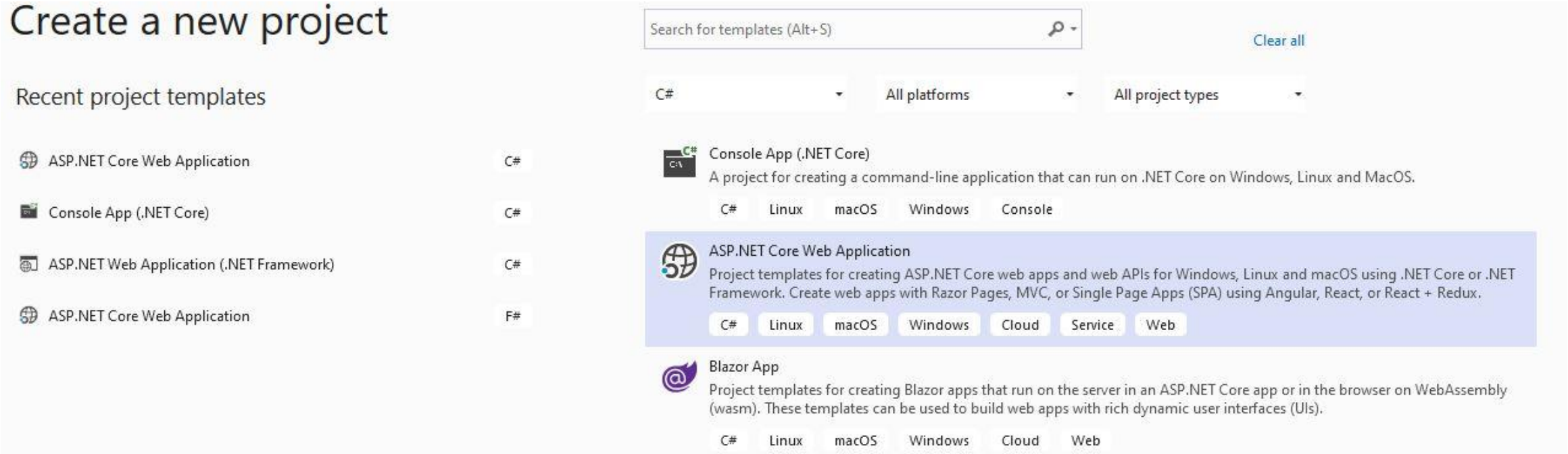
BUILD A WEB APP WITH ASP.NET CORE AND MONGODB

Car Gallery

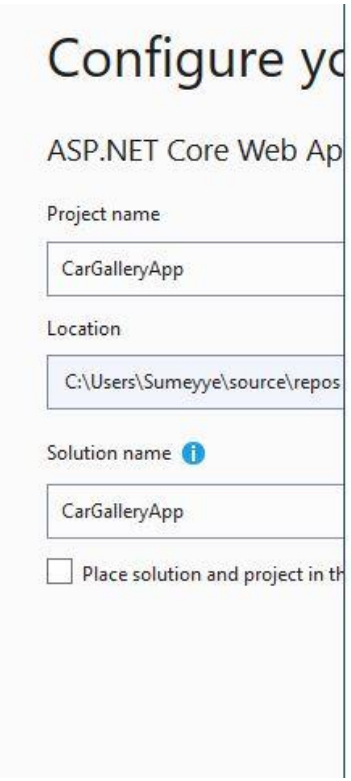
Add New Car

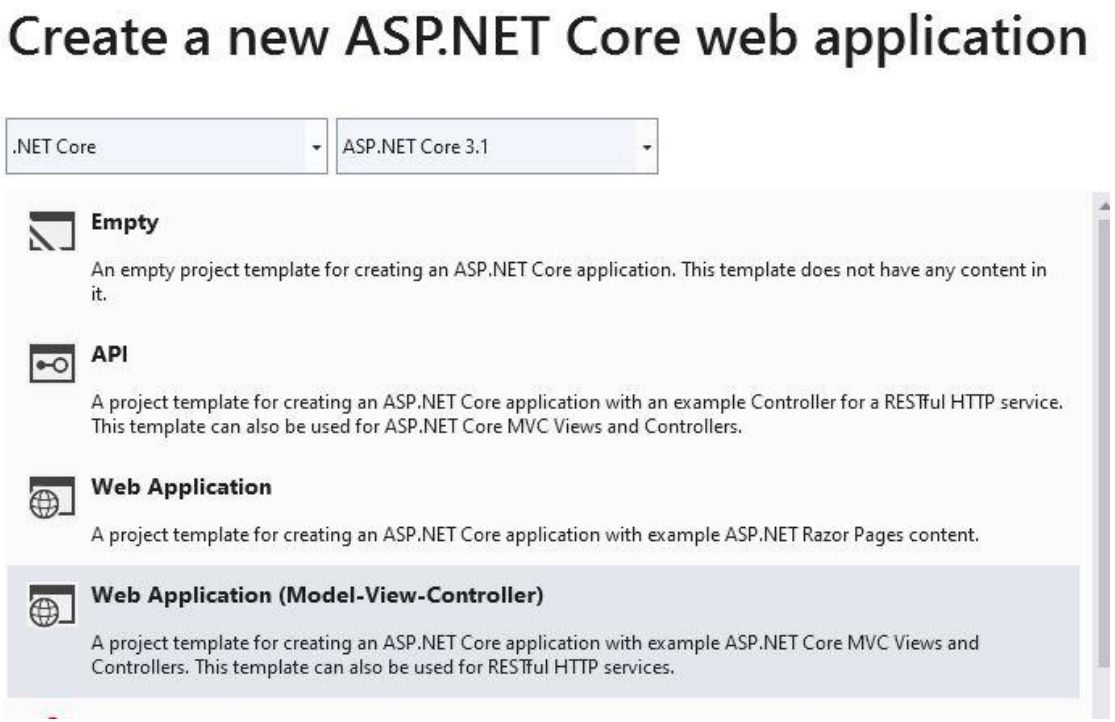
Photo	Brand	Model	Year	Price($)	
	Porsche	Cayenne	2018	60.600	Edit | Details | Delete
	Maserati	Quattroporte	2019	107.680	Edit | Details | Delete
	Bentley	Bentayga	2019	165.000	Edit | Details | Delete
	Chrysler	300	2018	28.995	Edit | Details | Delete

CREATE A NEW ASP.NET CORE PROJECT

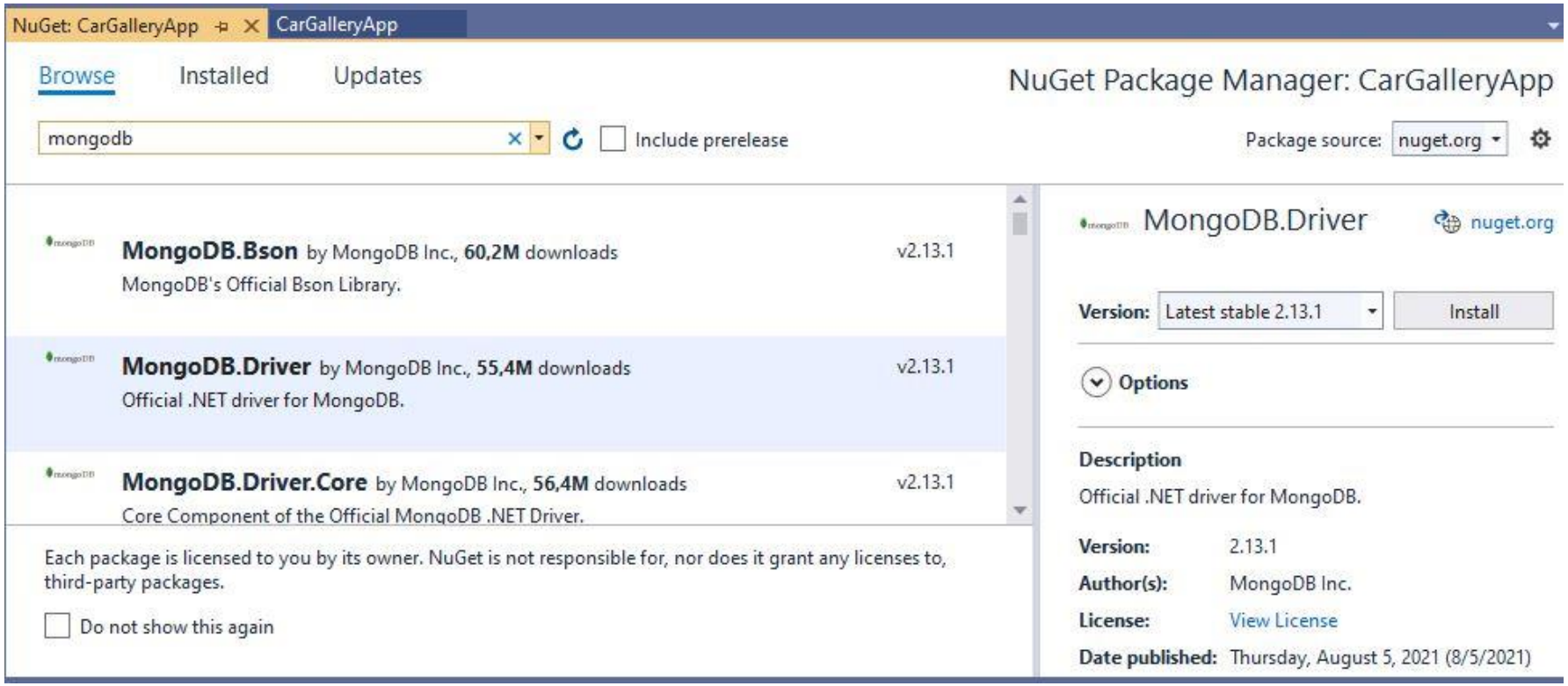


CREATE MVC PROJECT





ADD MONGODB DRIVER



LAYOUT EDITING

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Car Gallery App</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
```

```
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Car Gallery App</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
```

```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2021 - Car Gallery App - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @RenderSection("Scripts", required: false)
```

ADDING A MODEL

```
25 references
public class Cars
{
    [BsonId]
    [BsonRepresentation(MongoDB.Bson.BsonType.ObjectId)]
    14 references
    public string ID { get; set; }

    [Required]
    [BsonElement("Model")]
    12 references
    public string Model { get; set; }

    [Required]
    [BsonElement("Year")]
    12 references
    public int Year { get; set; }

    [BsonElement("Price")]
    12 references
    public decimal Price { get; set; }

    [BsonElement("ImageURL")]
    [Required]
    12 references
    public string ImageURL { get; set; }
}
```

ADDING A MODEL

```
0 references
public class Cars
{
    [BsonId]
    [BsonRepresentation(MongoDB.Bson.BsonType.ObjectId)]
    0 references
    public int ID { get; set; }

    [Required]
    [BsonElement("Model")]
    0 references
    public string Model { get; set; }

    [Required]
    [BsonElement("Year")]
    0 references
    public int Year { get; set; }

    [BsonElement("Price")]
    0 references
    public decimal Price { get; set; }

    [BsonElement("ImageURL")]
    [Required]
    0 references
    public string ImageURL { get; set; }
}
```

fail

```
25 references
public class Cars
{
    [BsonId]
    [BsonRepresentation(MongoDB.Bson.BsonType.ObjectId)]
    14 references
    public string ID { get; set; }

    [Required]
    [BsonElement("Model")]
    12 references
    public string Model { get; set; }

    [Required]
    [BsonElement("Year")]
    12 references
    public int Year { get; set; }

    [BsonElement("Price")]
    12 references
    public decimal Price { get; set; }

    [BsonElement("ImageURL")]
    [Required]
    12 references
    public string ImageURL { get; set; }
}
```

ADDING A CRUD SERVICES CLASS

```
4 references
public class CarServices
{
    private readonly IMongoCollection<Cars> cars;
    0 references
    public CarServices(IConfiguration config)
    {
        MongoClient client = new MongoClient(config.GetConnectionString("CarGalleryDb"));
        IMongoDatabase database = client.GetDatabase("CarGalleryDb");
        cars = database.GetCollection<Cars>("Cars");
    }
    1 reference
    public List<Cars> Get()
    {
        return cars.Find(car => true).ToList();
    }

    4 references
    public Cars Get(string id)
    {
        return cars.Find(car => car.ID == id).FirstOrDefault();
    }

    1 reference
    public Cars Create(Cars car)
    {
        cars.InsertOne(car);
        return car;
    }
```

```
1 reference
public void Update(string id, Cars carIn)
{
    cars.ReplaceOne(car => car.ID == id, carIn);
}

0 references
public void Remove(Cars carIn)
{
    cars.DeleteOne(car => car.ID == carIn.ID);
}

1 reference
public void Remove(string id)
{
    cars.DeleteOne(car => car.ID == id);
}
```

ADDING A CRUD SERVICES CLASS

- The “CarServices” class uses the following MongoDB.Driver members to perform CRUD operations against the database.

```
private readonly IMongoCollection<Cars> cars;
0 references
public CarServices(IConfiguration config)
{
    MongoClient client = new MongoClient(config.GetConnectionString("CarGalleryDb"));
    IMongoDatabase database = client.GetDatabase("CarGalleryDb");
    cars = database.GetCollection<Cars>("Cars");
}
```

ADDING A CRUD SERVICES CLASS

- **MongoClient:** Reads the server instance for performing database operations. The constructor of this class is provided the MongoDB connection string.
- **IMongoDatabase:** Represents the Mongo database for performing operations.

“COLLECTION” DEFINITION

- GetCollection<T>(collection)
- ”collection” represents the collection name in the database.
- “T” represents the CLR object type stored in the collection.

A CRUD SERVICES CLASS

```
public List<Cars> Get()
{
    return cars.Find(car => true).ToList();
}

4 references
public Cars Get(string id)
{
    return cars.Find(car => car.ID == id).FirstOrDefault();
}

1 reference
public Cars Create(Cars car)
{
    cars.InsertOne(car);
    return car;
}

1 reference
public void Update(string id, Cars carIn)
{
    cars.ReplaceOne(car => car.ID == id, carIn);
}

0 references
public void Remove(Cars carIn)
{
    cars.DeleteOne(car => car.ID == carIn.ID);
}
```

```
1 reference
public void Remove(string id)
{
    cars.DeleteOne(car => car.ID == id);
}
```

ADD THE MONGODB CONNECTION STRING

- appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },

  "ConnectionStrings": {
    "CarGalleryDb": "mongodb://localhost:27017"
  },
  "AllowedHosts": "*"
}
```

REGISTERING SERVICE WITH THE DEPENDENCY INJECTION SYSTEM

- Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<CarServices>();
    services.AddControllersWithViews();
}
```

ADDING A CONTROLLER

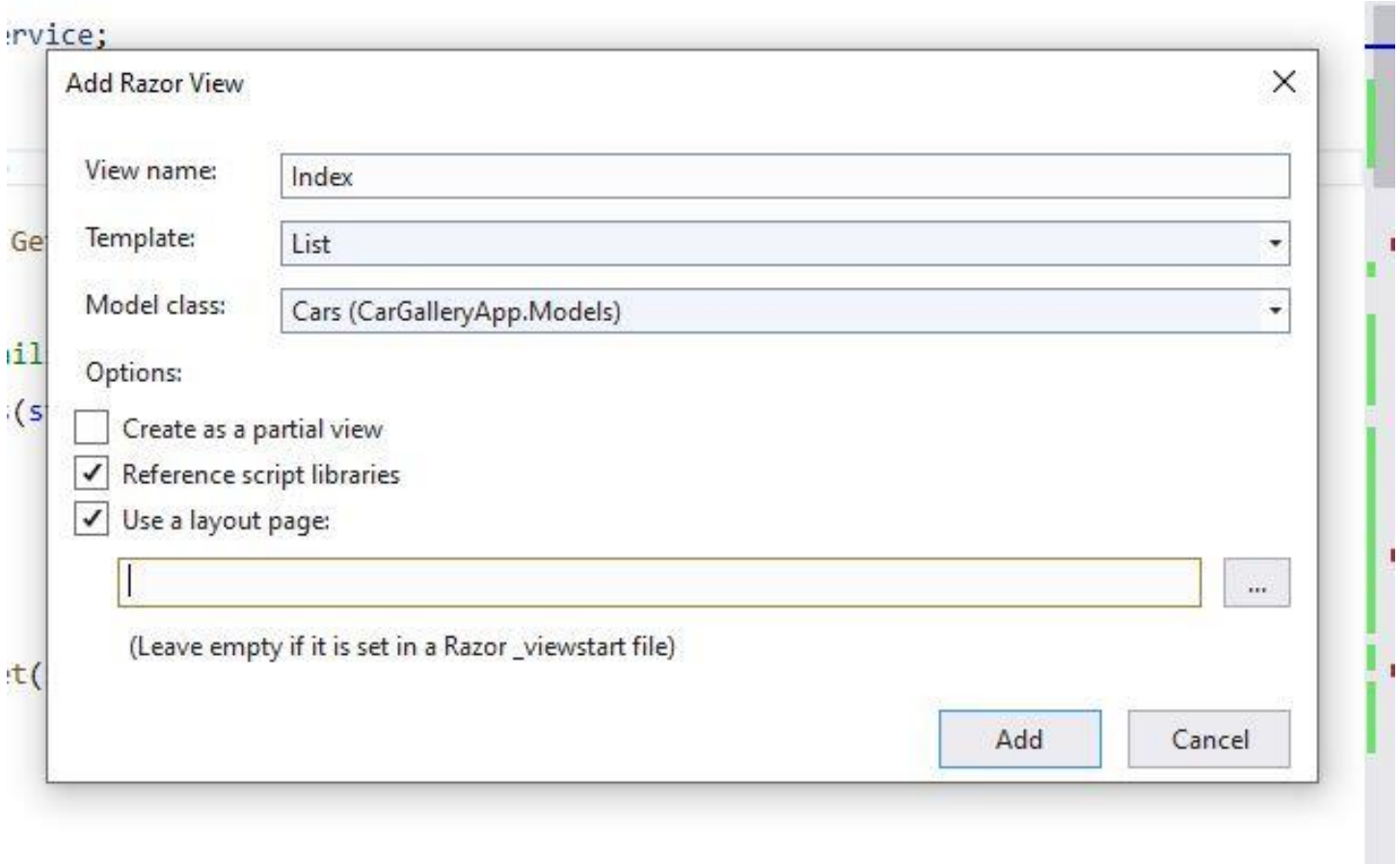
```
1 reference
public class CarsController : Controller
{
    private readonly CarServices carservice;

    0 references
    public CarsController(CarServices carservice)
    {
        this.carservice = carservice;
    }
    // GET: CarsController
```

INDEX METHOD AND VİEW

- Change the Index method in *CarsController.cs* as follows:

```
3 references
public ActionResult Index()
{
    return View(carservice.Get());
}
```

ADD INDEX VIEW



INDEX VIEW

Open the ”Index.cshml” and make the following changes:

- Remove the Id fields
- Change the title Index to **Car Gallery**
- Change Create New to **Add New Car**
- Change the **ImageUrl** field to:

```
<td>
    @if (item.ID != null)
    {
        <img src="@Url.Content(item.ImageURL)" alt="Image" />
    }
</td>
```

INDEX VIEW

- Update the action links as below:

```
<td>
    @Html.ActionLink("Edit", "Edit", new { id=item.ID}) |
    @Html.ActionLink("Details", "Details", new { id=item.ID}) |
    @Html.ActionLink("Delete", "Delete", new { id=item.ID})
</td>
```

CREATE VIEW

```
<h4>Cars</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Model" class="control-label"></label>
                <input asp-for="Model" class="form-control" />
                <span asp-validation-for="Model" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Year" class="control-label"></label>
                <input asp-for="Year" class="form-control" />
                <span asp-validation-for="Year" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ImageURL" class="control-label"></label>
                <input asp-for="ImageURL" class="form-control" />
                <span asp-validation-for="ImageURL" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
```

CREATE POST METHOD

```
// POST: CarsController/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Create(Cars car)
{

        if (ModelState.IsValid)
        {
            carservice.Create(car);
            return RedirectToAction(nameof(Index));
        }

    return View(car);
}
```

ADDING A NEW RECORD

Car Gallery App Home Privacy

Create

Cars

Model

Year

Price

ImageURL

Create

Back to List

Index

Add New Car

Model	Year	Price	ImageURL	
Porsche	2020	1120000,00		Edit | Details | Delete

EDIT VIEW

```
<div class="col-md-4">
    <form asp-action="Edit">
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <div class="form-group">
            <label asp-for="Model" class="control-label"></label>
            <input asp-for="Model" class="form-control" />
            <span asp-validation-for="Model" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Year" class="control-label"></label>
            <input asp-for="Year" class="form-control" />
            <span asp-validation-for="Year" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Price" class="control-label"></label>
            <input asp-for="Price" class="form-control" />
            <span asp-validation-for="Price" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="ImageURL" class="control-label"></label>
            <input asp-for="ImageURL" class="form-control" />
            <span asp-validation-for="ImageURL" class="text-danger"></span>
        </div>
        <div class="form-group">
            <input type="submit" value="Save" class="btn btn-primary" />
        </div>
    </form>
```

EDIT GET AND POST METHODS

Model	Year	Price	ImageURL	
Porsche	2020	1120000,00		Edit | Details | Delete

Edit

Cars

Model

Porsche

Year

2020

Price

1120000,00

ImageURL

/images/Porsche.JPG

Save

EDIT GET AND POST METHODS

```
// GET: CarsController/Edit/5
0 references
public ActionResult Edit(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var car = carservice.Get(id);
    if (car == null)
    {
        return NotFound();
    }
    return View(car);
}
```

```
// POST: CarsController/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Edit(string id, Cars cars)
{
    if (id != cars.ID)
    {
        return NotFound();
    }
    if (ModelState.IsValid)
    {
        carservice.Update(id, cars);
        return RedirectToAction(nameof(Index));
    }
    else
    {
        return View(cars);
    }
}
```

DETAILS VIEW

```
<dl class="row">
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Model)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Model)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Year)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Year)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Price)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Price)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.ImageURL)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.ImageURL)
    </dd>
</dl>
```

DETAILS GET METHOD

```
// GET: CarsController/Details/5
0 references
public ActionResult Details(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var car = carservice.Get(id);
    if (car == null)
    {
        return NotFound();
    }
    return View(car);
}
```

DELETE VIEW

```
<dl class="row">
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Model)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Model)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Year)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Year)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Price)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Price)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.ImageURL)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.ImageURL)
    </dd>
</dl>
```

DELETE GET AND POST METHODS

```
// GET: CarsController/Delete/5
0 references
public ActionResult Delete(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var car = carservice.Get(id);
    if (car == null)
    {
        return NotFound();
    }
    return View(car);
}
```

```
// POST: CarsController/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult DeleteConfirmed(string id)
{
    try
    {
        var car = carservice.Get(id);

        if (car == null)
        {
            return NotFound();
        }

        carservice.Remove(car.ID);

        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}
```