# MongoDB

TABLE OF CONTENTS

Introduction 3

Create .NET Core Web API Project 4

Add an Entity Model 4

Add a Configuration Model 5

Add a CRUD Operation Service 7

Add a Controller 9

Testing the WEB API 12

Configure MongoDB Atlas 13

How to get Connection String 16

Create Database 17

Conclusion 19

Anbumani.T



# Introduction

MongoDB is a No SQL database. It is an open-source, cross-platform, document orientated database written in C++. Why MongoDB ? All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed. It stores data in the form of Key Value Pairs.

In this document, we are going to learn about How to create a WEB API on .NET Core that runs Create, Read, Update, and Delete (CRUD) operations on a MongoDB.

For this, we are using Visual Studio and MongoDb Atlas.

# Create ASP.NET Core web APi project

1. Select Create new Project.
2. Select the ASP.NET Core Web API project type, and select Next.
3. Name the Project and select Next.
4. Select the .NET 6.0 (Long term support) framework and select Create.
5. Go to Tools > NuGet package Manager > Manage NuGet Packages
6. Browse MongoDB.Driver and Install it in your solution.

A screenshot of a computer

Description automatically generated with medium confidence

## Add an entity model

1. Add a Models directory to the project root (solution).
2. Add a User class to the Models directory with the following code:

using MongoDB.Bson.Serialization.Attributes;

using MongoDB.Bson;

namespace MongoRepository.Models

{

public class User

{

[BsonId]

public string Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

}

}

1. Here, Id property is required for mapping the Common Language Runtime (CLR) object to the MongoDB collection.
2. Annotated with [BsonId] to make this property the document’s primary key.

## Add a configuration model

1. Add the following database configuration values to appsettings.json.

{

"UserDatabase": {

"ConnectionString": "mongodb+srv://username:password@user.c4ioll8.mongodb.net/?retryWrites=true&w=majority",

"DatabaseName": "UserDB",

"CollectionName": "Users"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

* ConnectionString will be taken from the MongoDB Atlas which will be discussed below.

1. Add a UserDatabaseSettings class to the Models directory with the following code:

namespace MongoRepository.Models

{

public class UserDatabaseSettings

{

public string ConnectionString { get; set; } = null!;

public string DatabaseName { get; set; } = null!;

public string CollectionName { get; set; } = null!;

}

}

* The preceding UserDatabaseSettings class is used to store the appsettings.json file’s UserDatabase property values. The JSON and C# property names are named identically to ease the mapping process.

1. Add the following highlighted code to the program.cs :

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.Configure<UserDatabaseSettings>(

builder.Configuration.GetSection("UserDatabase"));

* In the preceding code , the configuration instance to which the appsettings.json file’s UserDatabase section binds is registered in the Dependency Injection (DI) container. For example, the UserDatabaseSettings object’s ConnectionString property is populated with the UserDatabase:ConnectionString property in appsettings.json.

1. Add the following code to the top of program.cs to resolve the UserDatabaseSettings reference:

using MongoRepository.Models;

* Here, MongoRepository is solution name.

## Add a CRUD operations service

1. Add a Services directory to the project root.
2. Add a UserService class to the Services directory with the following code:

using Microsoft.Extensions.Options;

using MongoDB.Driver;

using MongoRepository.Models;

namespace UserApi.Services;

public class UserService

{

private IMongoCollection<User> \_user;

public UserService(IOptions<UserDatabaseSettings> userDatabaseSettings)

{

var mongoClient = new MongoClient(

userDatabaseSettings.Value.ConnectionString);

var mongoDatabase = mongoClient.GetDatabase(

userDatabaseSettings.Value.DatabaseName);

\_user = mongoDatabase.GetCollection<User>(

userDatabaseSettings.Value.CollectionName);

}

public async Task<List<User>> GetAsync() =>

await \_user.Find(\_ => true).ToListAsync();

public async Task<User?> GetAsync(string id) =>

await \_user.Find(x => x.Id == id).FirstOrDefaultAsync();

public async Task CreateAsync(User newUser) =>

await \_user.InsertOneAsync(newUser);

public async Task UpdateAsync(string id,User updatedUser) =>

await \_user.ReplaceOneAsync(x=>x.Id==id,updatedUser);

public async Task RemoveAsync(String id) =>

await \_user.DeleteOneAsync(x=>x.Id==id);

}

* In the preceding code, a UserDatabaseSettings instance is retrieved from DI via constructor injection.

1. Add the following highlighted code to Program.cs:

var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<UserDatabaseSettings>(

builder.Configuration.GetSection("UserDatabase"));

builder.Services.AddSingleton<UserService>();

1. Add the following code to the top of Program.cs to resolve the UserService reference :

using UserApi.Services;

* The UserService class uses the following MongoDB.Driver members to run CRUD operations against the database:
  1. DeleteOneAsync: Delete a single document matching the provided search criteria.
  2. Find<TDocument>: Returns the documents in the collection matching the provided search criteria.
  3. InsertOneAsync: Inserts the provided object as a new document in the collection.
  4. ReplaceOneAsync: Replaces the single document matching the provided search criteria with the provided object.

## Add a controller:

Add a UserController class to the Controllers directory with the following code:using Microsoft.AspNetCore.Mvc;

using MongoRepository.Models;

using UserApi.Services;

namespace MongoRepository.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class UserController:ControllerBase

{

private UserService \_userService;

public UserController(UserService userService)

{

\_userService = userService;

}

[HttpGet]

public async Task<List<User>> Get() =>

await \_userService.GetAsync();

[HttpPost]

public async Task<ActionResult> Post(User newUser)

{

await \_userService.CreateAsync(newUser);

return CreatedAtAction(nameof(Get), new { id = newUser.Id }, newUser);

}

[HttpGet("{id}")]

public async Task<ActionResult<User>> Get(string id)

{

var user=await \_userService.GetAsync(id);

if(user == null)

{

return NotFound();

}

return Ok(user);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, User updateUser)

{

var user = await \_userService.GetAsync(id);

if(user is null)

{

return NotFound();

}

updateUser.Id = user.Id;

await \_userService.UpdateAsync(id, updateUser);

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

var user=await \_userService.GetAsync(id);

if(user is null)

{

return NotFound();

}

await \_userService.RemoveAsync(id);

return NoContent();

}

}

}

The preceding web API controller:

* Uses the UserService class to run CRUD operations.
* Contains action methods to support GET, POST, PUT, DELETE HTTP requests.

# Testing the web api

1. Build and run the app.
2. Since we have created ASP.NET Core WEB API project, the Swagger is by default configured in the program.cs which will redirect you to the Swagger UI after build and run.
3. Swagger is a API Debugging tool, where you can test all the methods that you have created in the Controller.

var app = builder.Build();

// Configure the HTTP request pipeline.

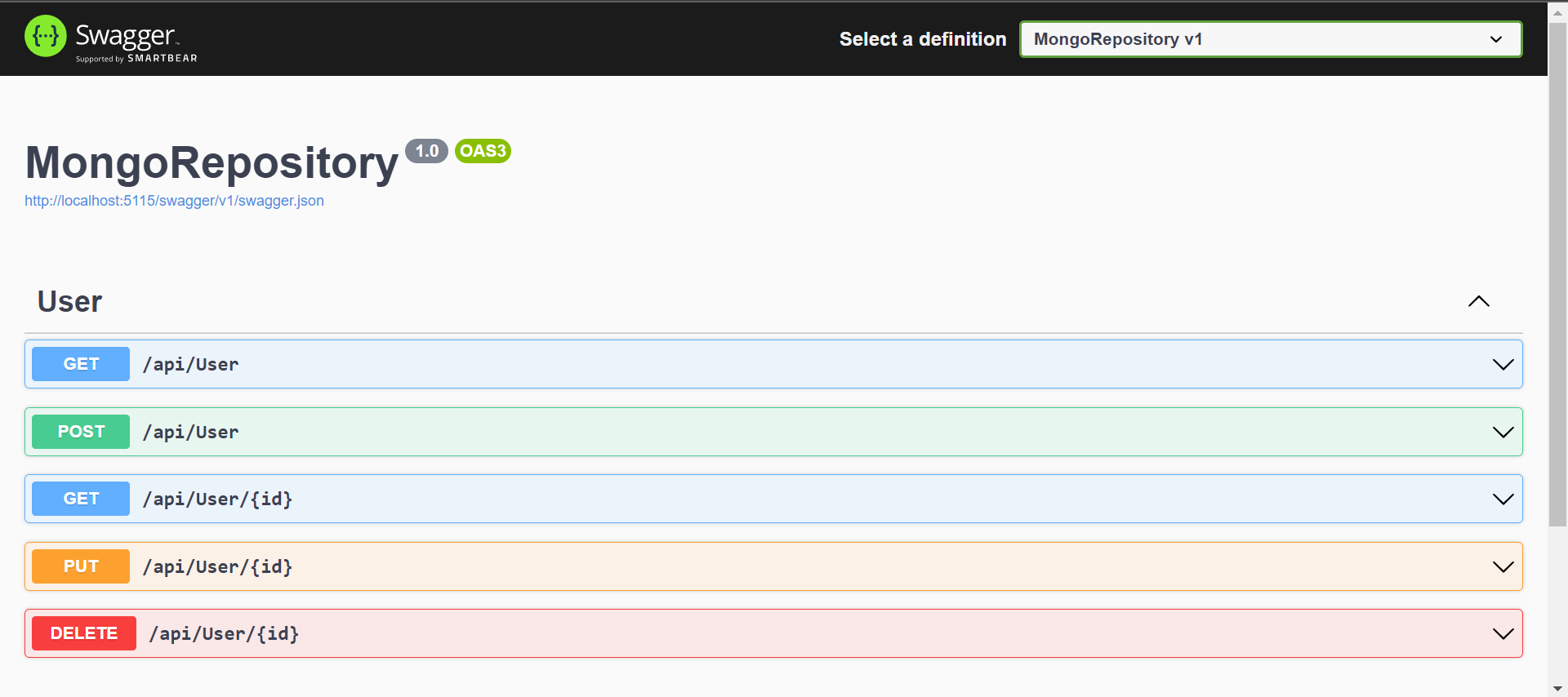
if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

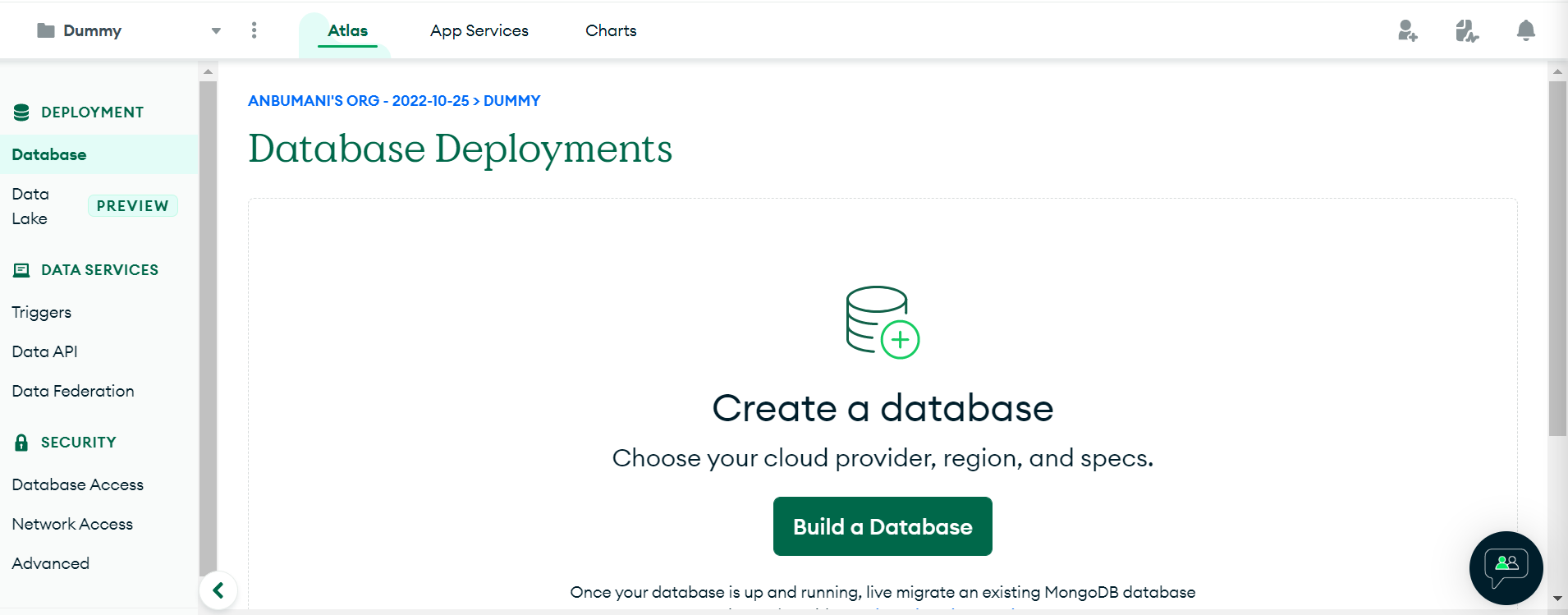
}



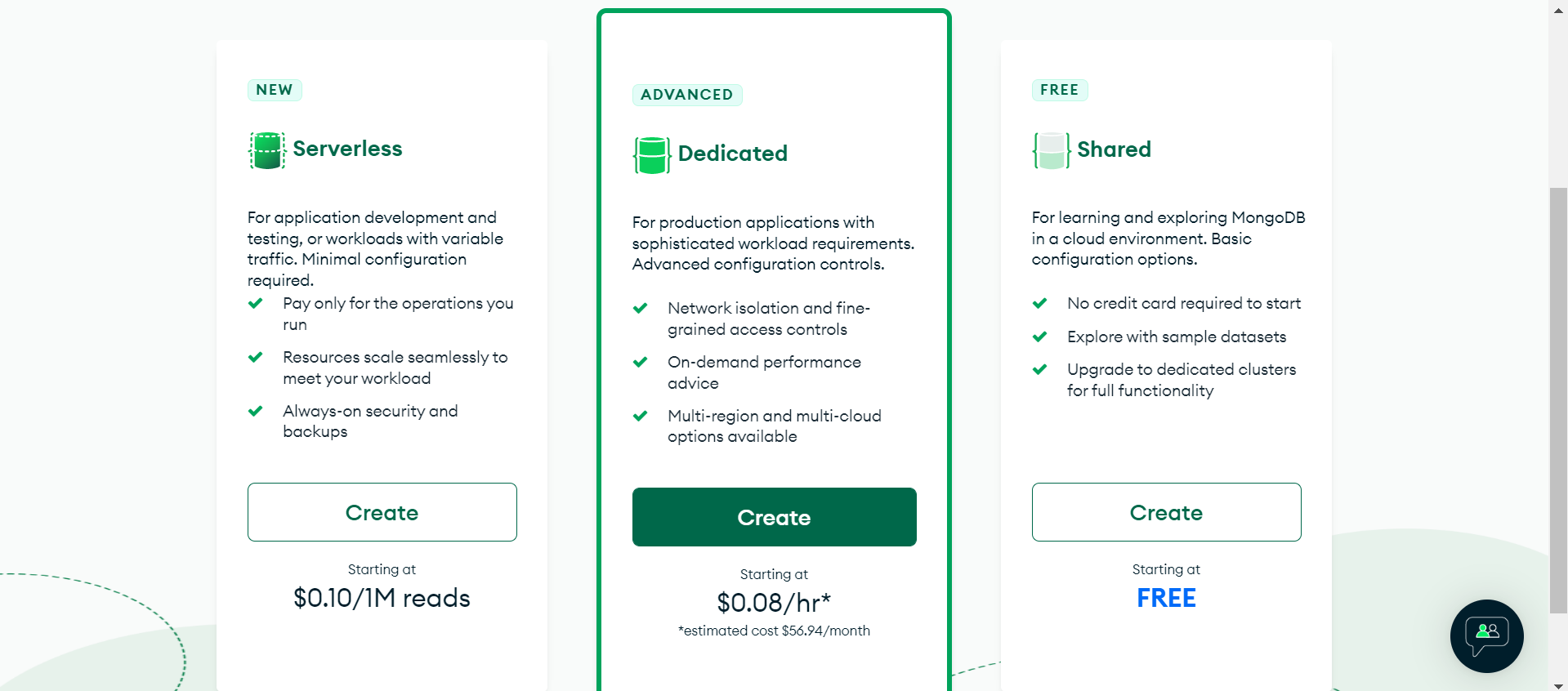
* After building, this page will be opened which contains all the methods you have implemented.
* Here you can test all your methods.

# configure mongo db atlas

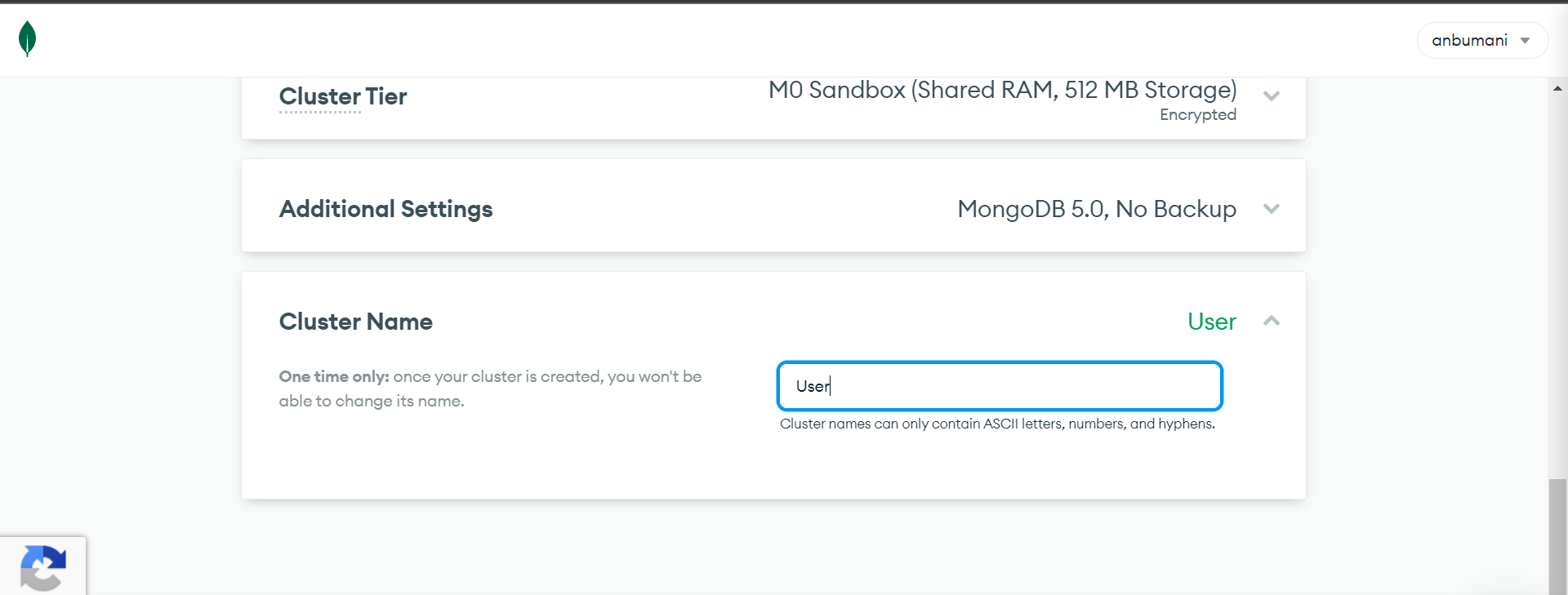
1. Create an account and login into MongoDB Atlas.
2. Select New Project
3. Enter the Project Name, Project Owner, Member details and select Create Project.



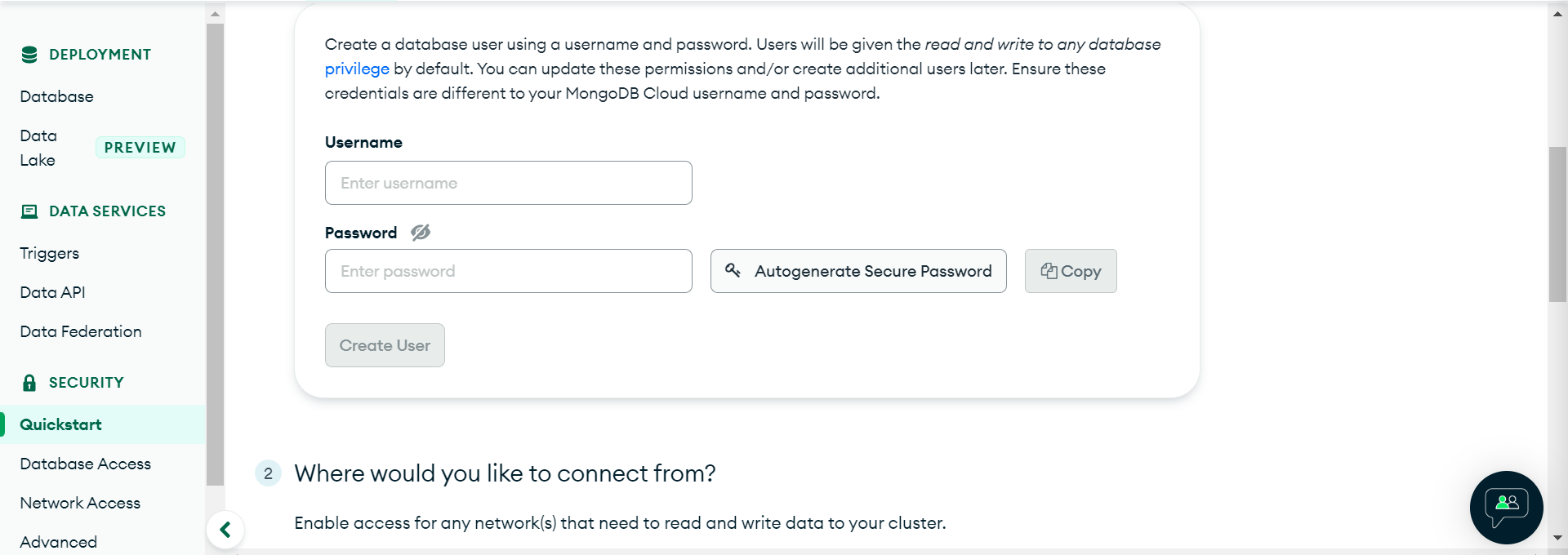
1. After creating the Project, Select the Build a Database Button.



1. Select the Shared version which is free.



1. Enter the Cluster name. Shared Cluster in MongoDB means collection of database distributed across many servers in order to maintain the horizontal scalability and better performance in read and write operations. Select Create Cluster.



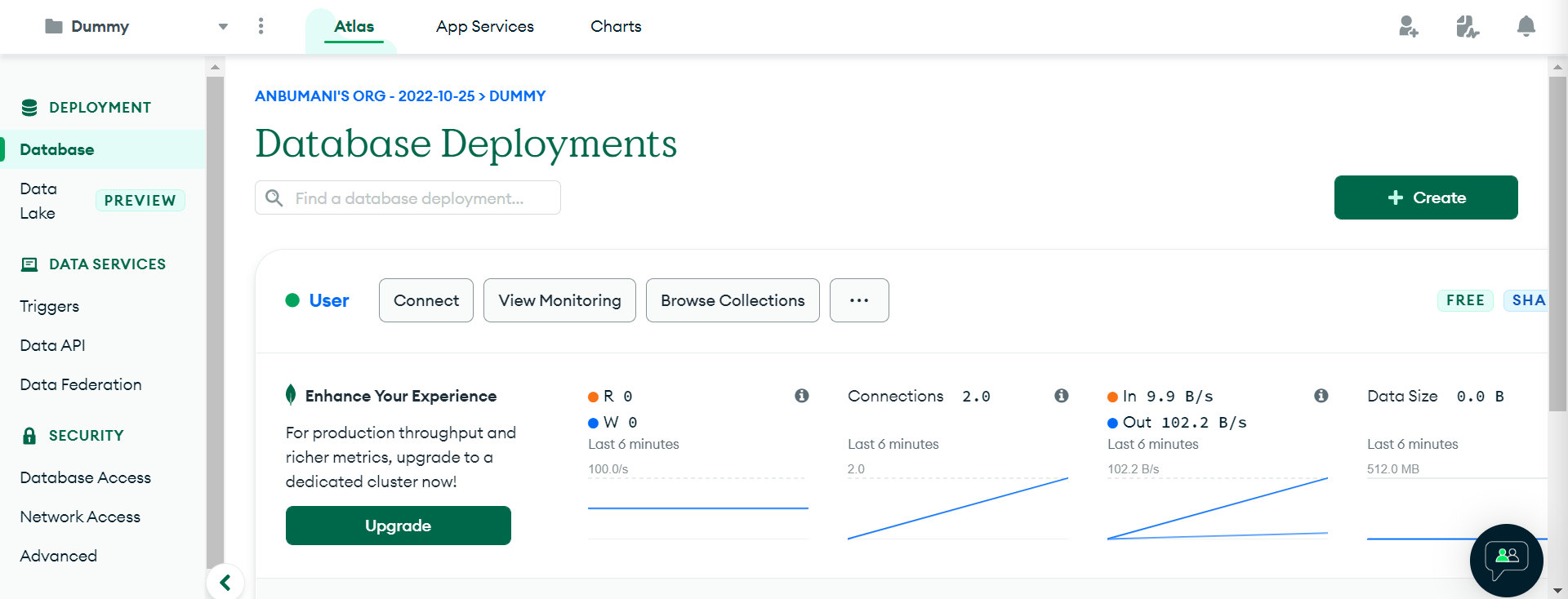
1. Create a new user by entering the Username and Password.

Graphical user interface, text, application, email

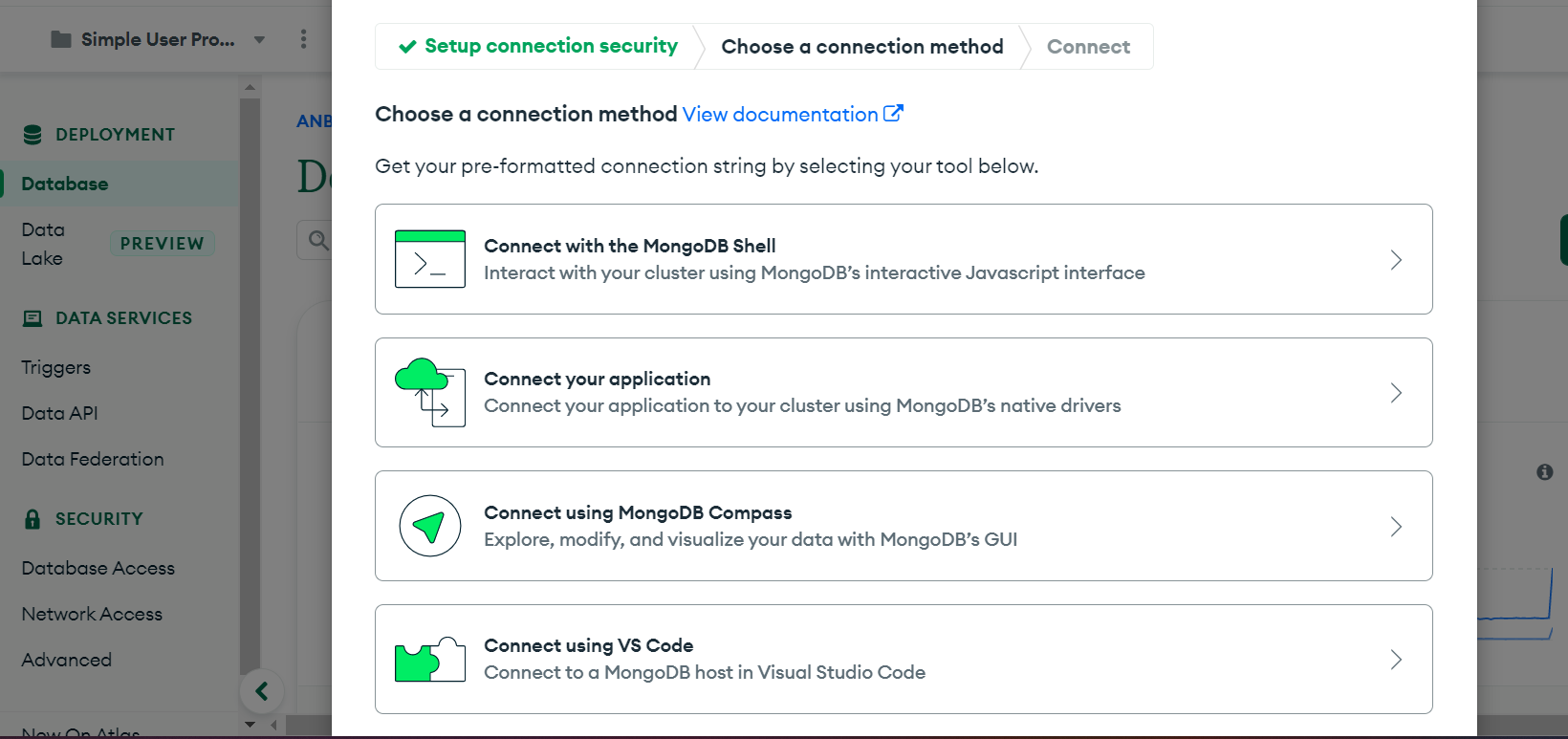
Description automatically generated

1. Select the My Local Environment and Enter **0.0.0.0** in the IP Address field and select Add Entry. It means it will allow the requests coming from any IP Address.

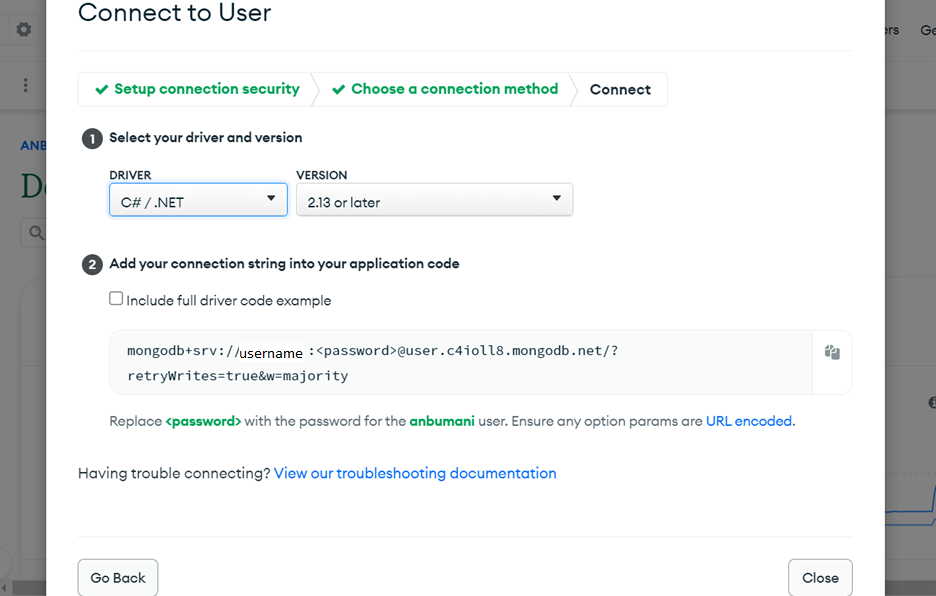
## How to get Connection String



1. Select the **Connect** and it will show all the possible ways to connect your database to the API.



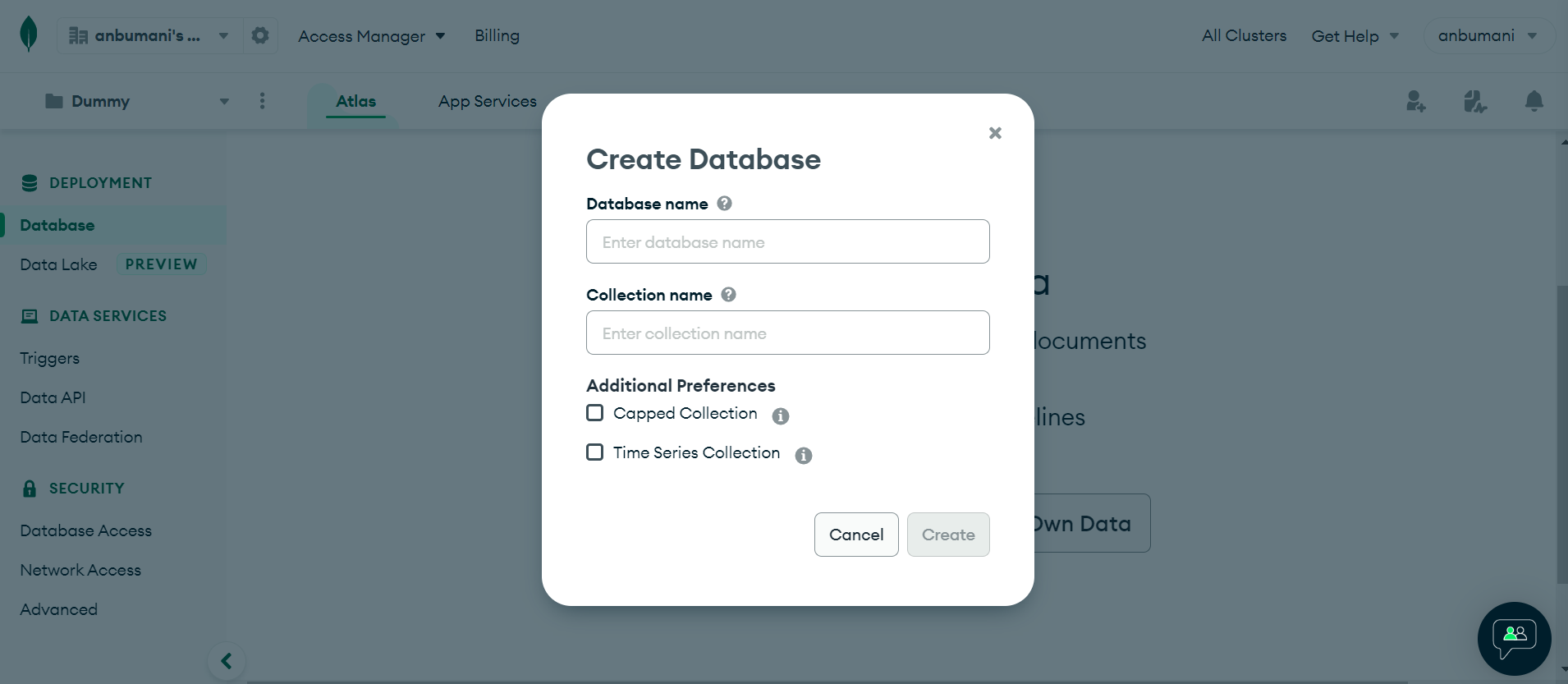
1. Select the second option **Connect your Application**.



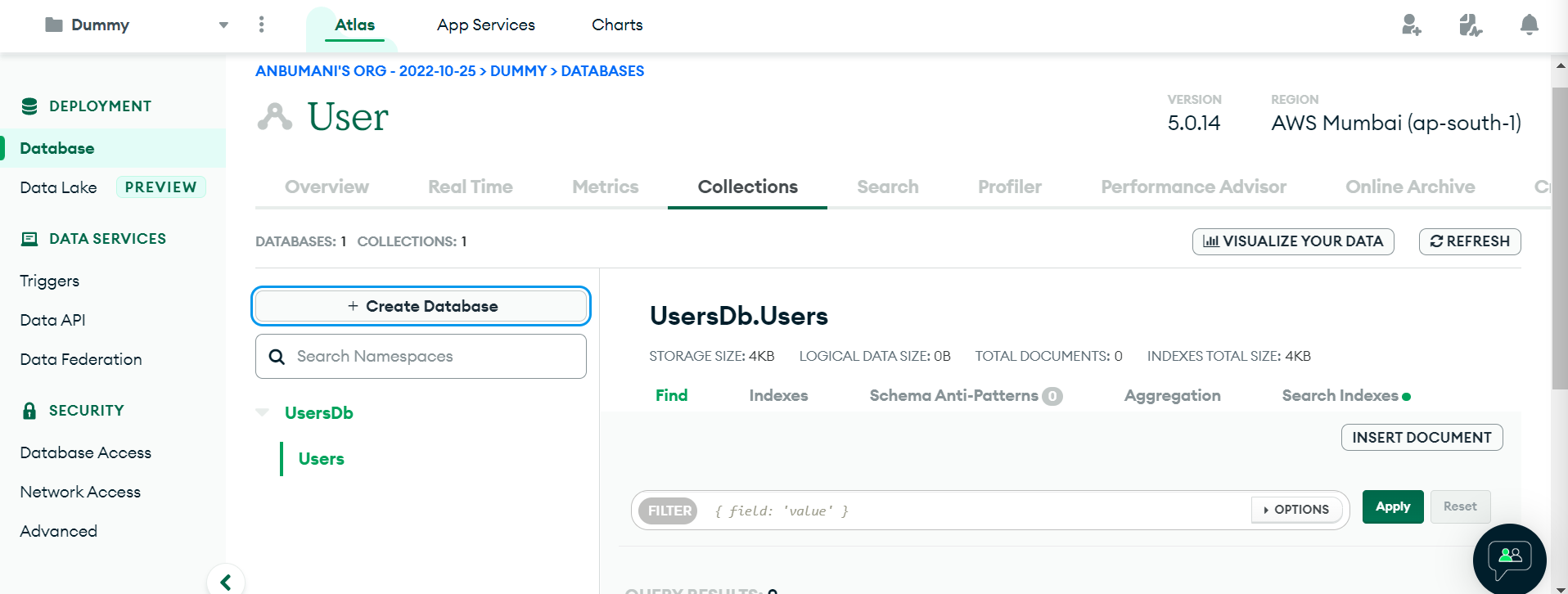
1. Select the Driver (C#/.NET) and the version. Copy the Connection String and paste it in the appsettings.json.

## Create Database:

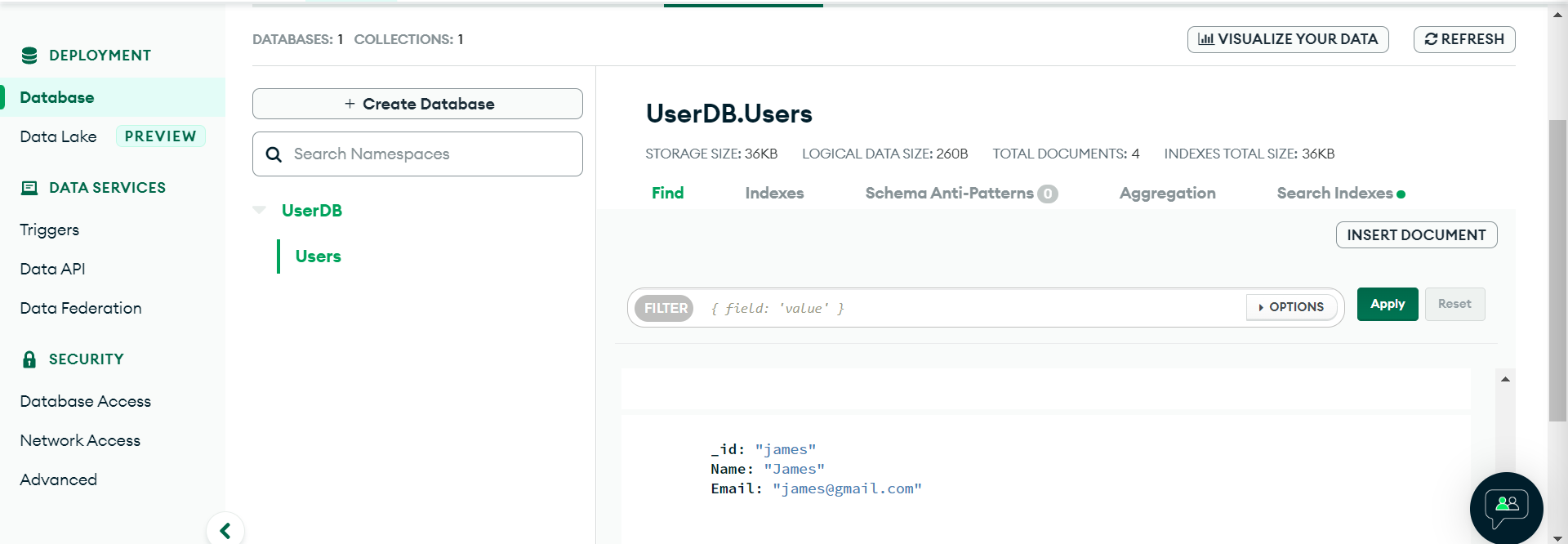
1. Now the Cluster and the user is created. Next we have to Add our Database configuration. For that Select **Browse Collections -> Add My Own Data.**



1. Enter the Database Name an Collection Name which we have given in the app.settings.json while configuring the Database settings.



1. Now the UsersDB is created with Collection Users.
2. You can either manually insert the data or you can use the Swagger to do the CRUD operations.



1. One user is added to the database successfully. You can also Edit, Delete and Read all the data in the database.

# conclusion

In this document, we have seen how to create the .NET web API project in visual studio and configure the database settings to connect the MongoDB. Here we are using MongoDB Atlas. We have seen how to create shared cluster in MongoDB Atlas and create database user and also how to add collections in it. Finally we have to copy the connection string and connect to the API.