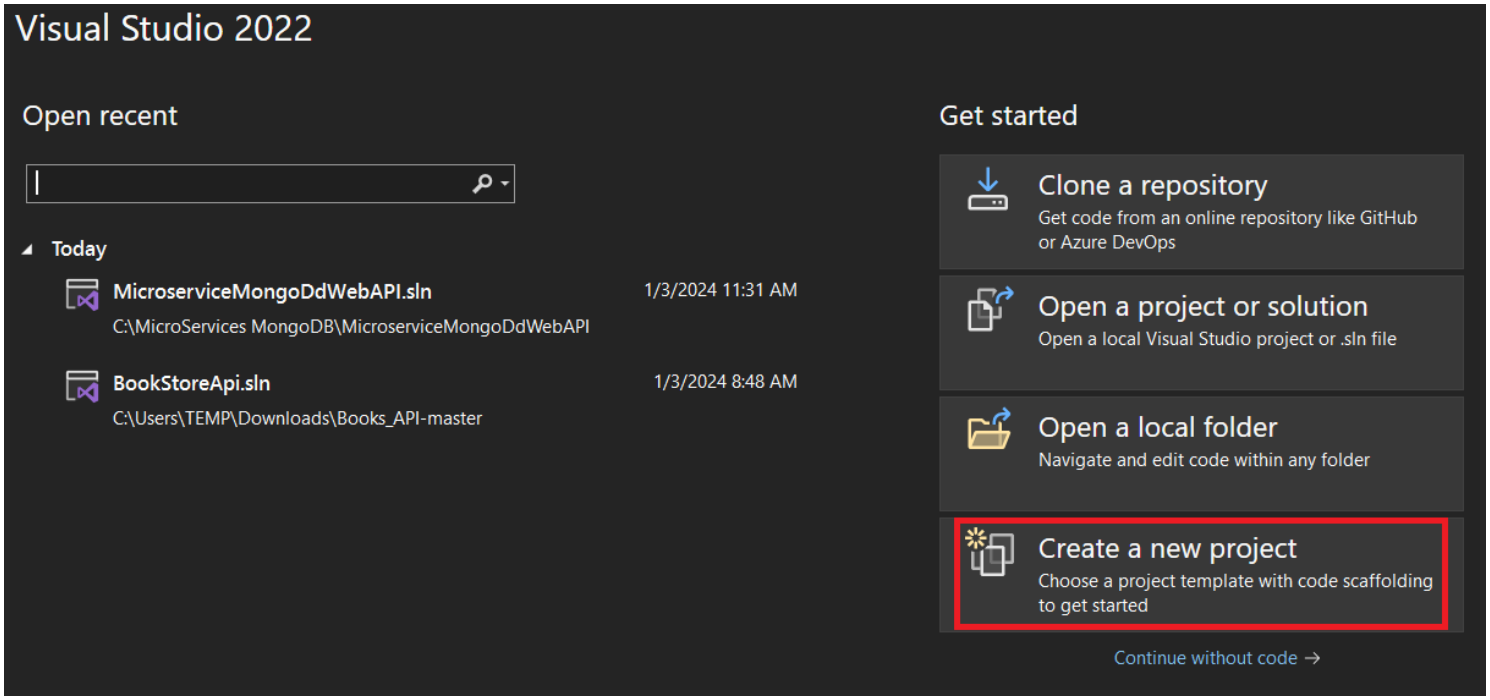
How to create a .NET8 WebAPI CRUD MongoDB Microservice

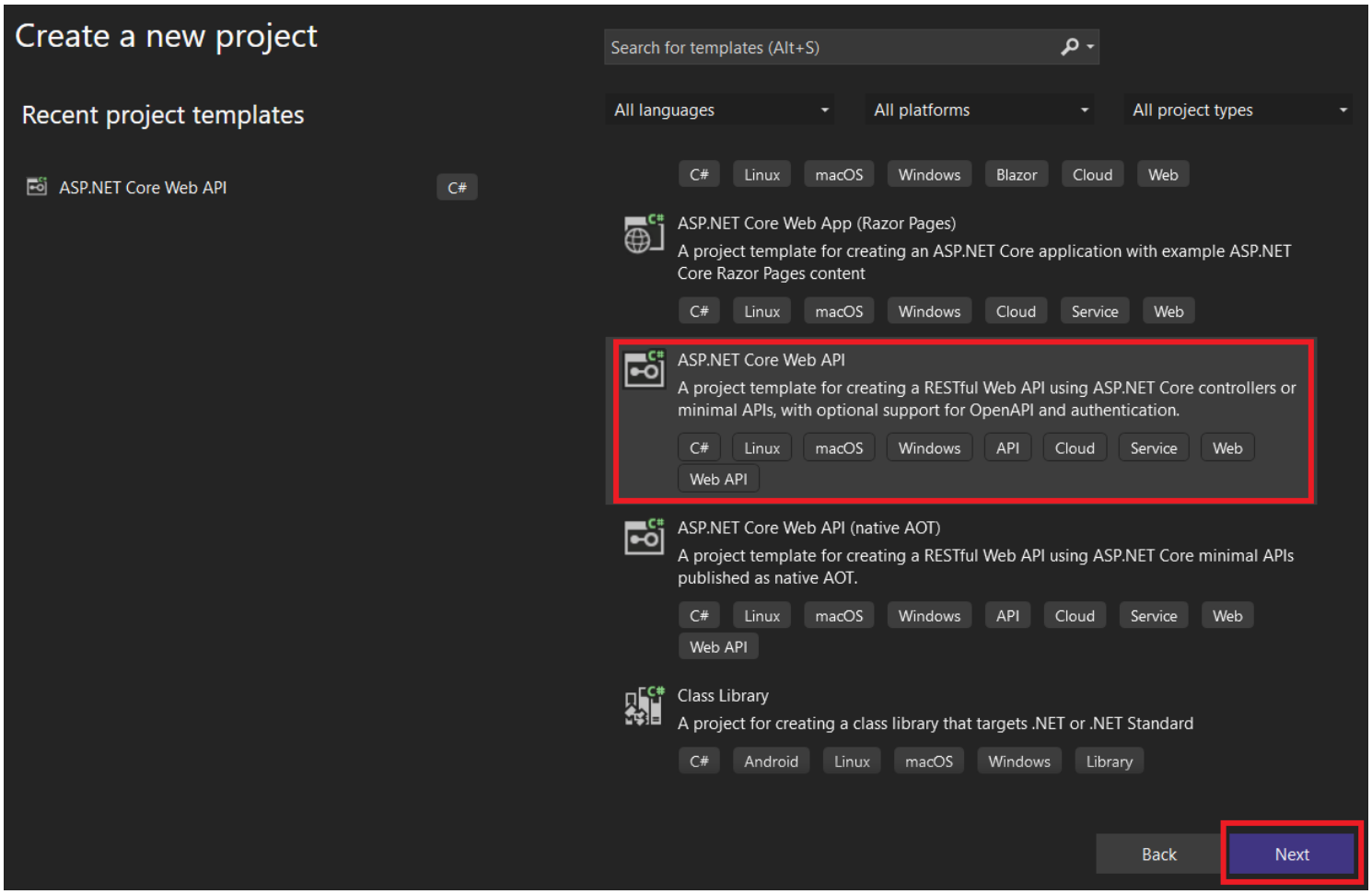
The code for this example is available in this github repo:
https://github.com/luiscoco/MicroServices_dotNET8_CRUD_WebAPI-MongoDB_deployed_to_Docker_DeskTop

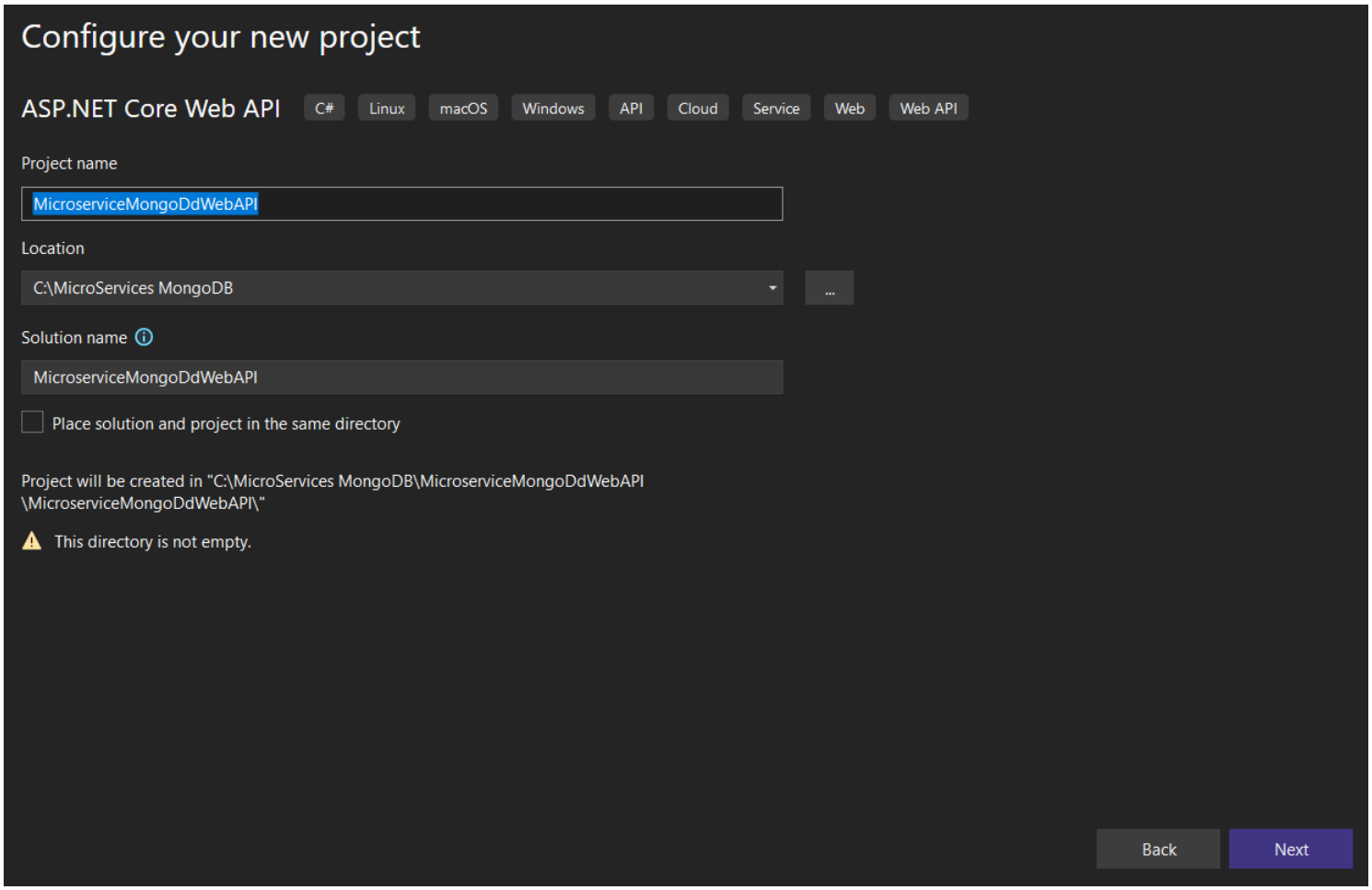
0. Prerequisites

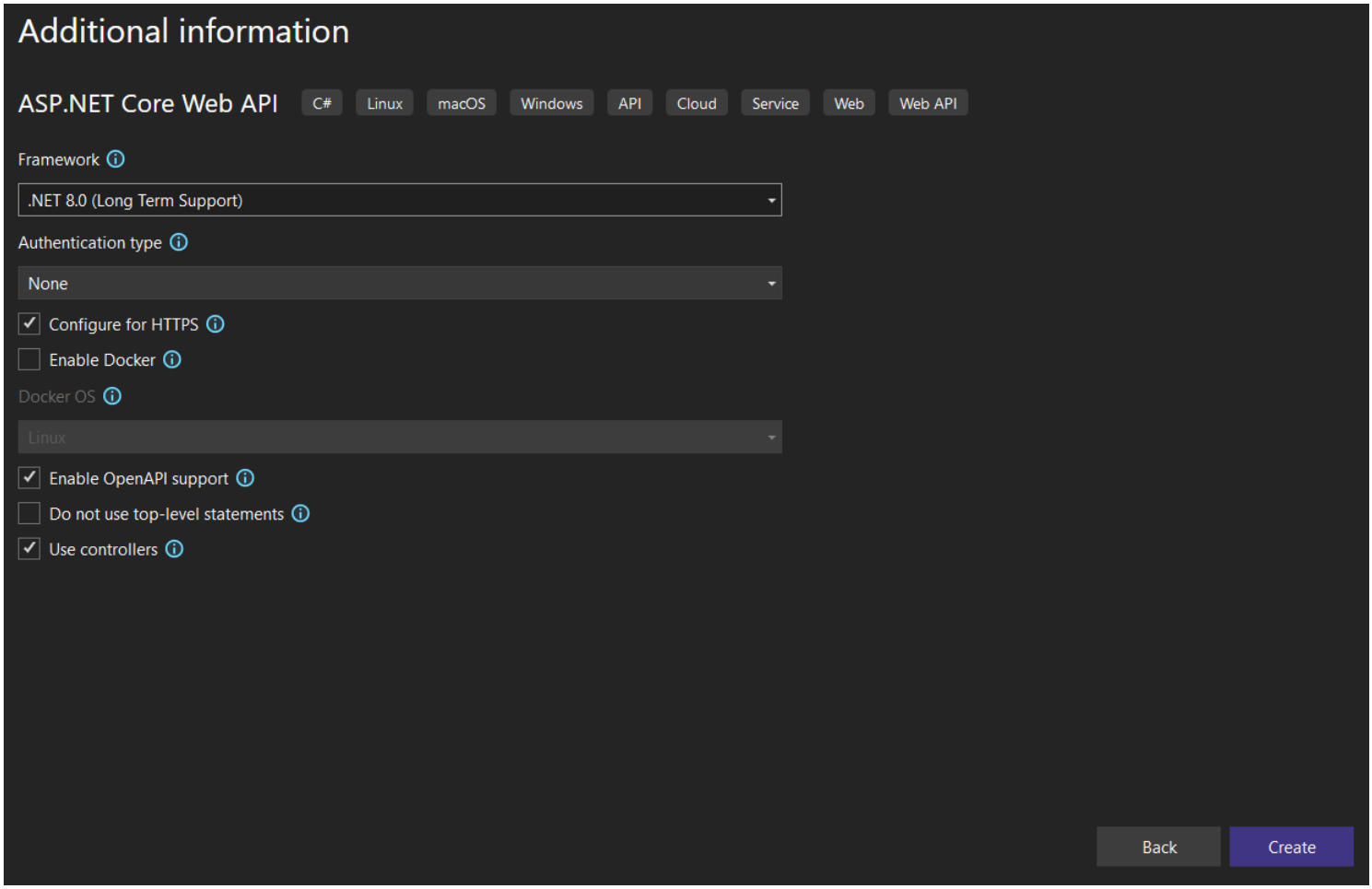
- Install Docker Desktop
- Install Visual Studio 2022 Community Edition version 17.8
- Install Studio 3T Free for MongoDB

1. Create .NET8 WebAPI in Visual Studio 2022 Community Edition




Create a new project
Search for templates (Alt+S)
Recent project templates
All languages
All platforms
All project types
ASP.NET Core Web API
C#
C# Linux macOS Windows Blazor Cloud Web
ASP.NET Core Web App (Razor Pages)
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content
C# Linux macOS Windows Cloud Service Web
ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# Linux macOS Windows API Cloud Service Web Web API
ASP.NET Core Web API (native AOT)
A project template for creating a RESTful Web API using ASP.NET Core minimal APIs published as native AOT.
C# Linux macOS Windows API Cloud Service Web Web API
Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library
Back
Next

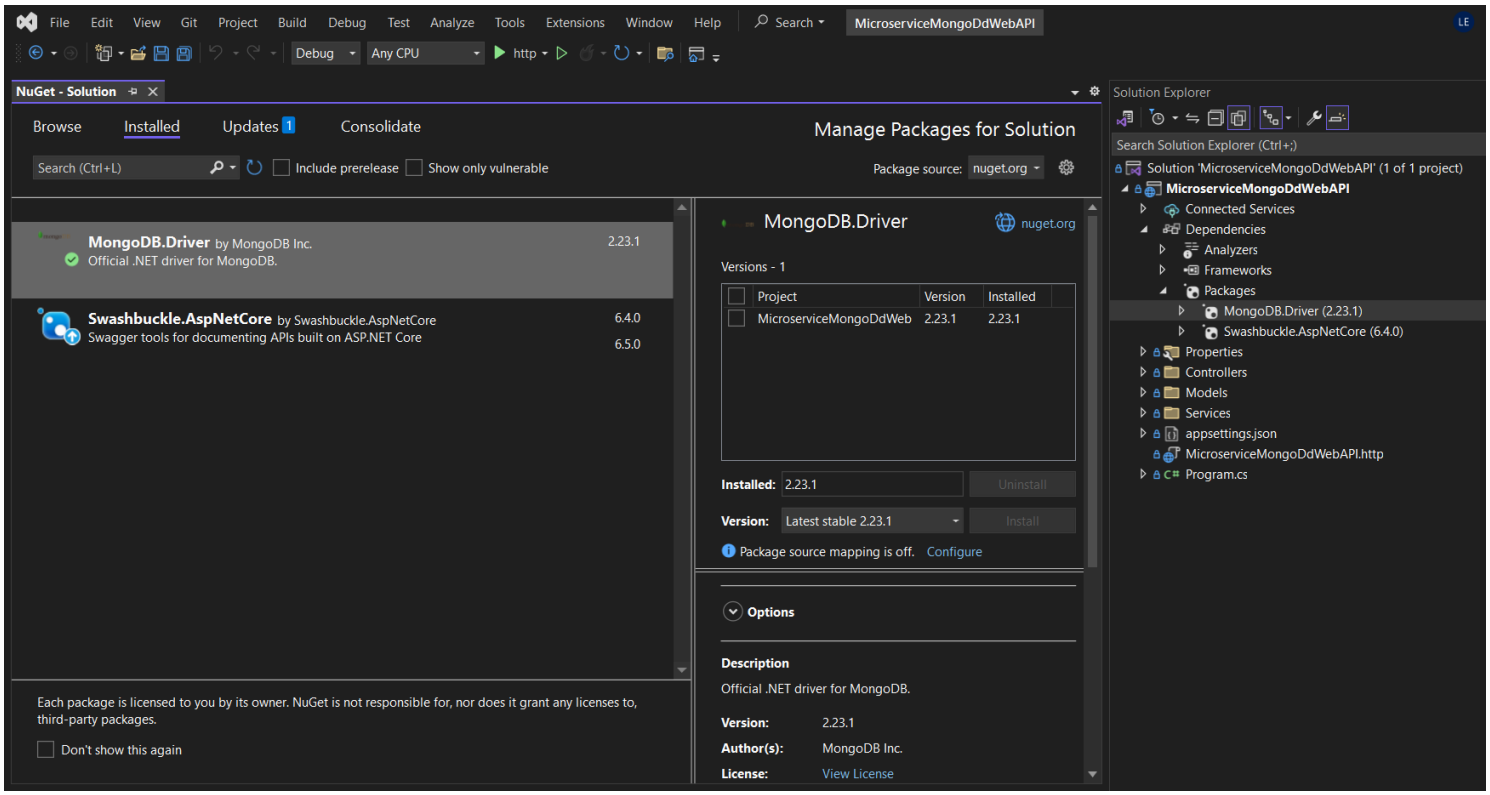

Configure your new project
ASP.NET Core Web API
C#
Linux
macOS
Windows
API
Cloud
Service
Web
Web API
Project name
MicroserviceMongoDdWebAPI
Location
C:\MicroServices MongoDB
...
Solution name
MicroserviceMongoDdWebAPI
Place solution and project in the same directory
Project will be created in "C:\MicroServices MongoDB\MicroserviceMongoDdWebAPI
\MicroserviceMongoDdWebAPI\"
This directory is not empty.
Back
Next



2. Add the MongoDB.Driver dependency

Select the menu option Tools->Nuget Package Manager->Manage Nuget Packages for Solution...

Then browse Mongo.DB.Driver and install it in your solution



3. Add the Models

Create the Models folder and inside include the following two files:

Book.cs

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System.Text.Json.Serialization;

namespace BookStoreApi.Models
{
    public class Book
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string? Id { get; set; }

        [BsonElement("Name")]
        [JsonPropertyName("Name")]
        public string BookName { get; set; } = null!;

        public decimal Price { get; set; }

        public string Category { get; set; } = null!;
```

```
        public string Author { get; set; } = null!;
    }
}
```

BookStoreDatabaseSettings.cs

```
namespace BookStoreApi.Models
{
    public class BookStoreDatabaseSettings
    {
        public string ConnectionString { get; set; } = null!;

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

        public string BooksCollectionName { get; set; } = null!;
    }
}
```

4. Add the Service

Create a Services folder with the following file:

BookService.cs

```
using BookStoreApi.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace BookStoreApi.Services
{
    public class BooksService
    {
        private readonly IMongoCollection<Book> _booksCollection;

        public BooksService(
            IOptions<BookStoreDatabaseSettings> bookStoreDatabaseSettings)
        {
            var mongoClient = new MongoClient(
                bookStoreDatabaseSettings.Value.ConnectionString);

            var mongoDatabase = mongoClient.GetDatabase(
                bookStoreDatabaseSettings.Value.DatabaseName);

            _booksCollection = mongoDatabase.GetCollection<Book>(
                bookStoreDatabaseSettings.Value.BooksCollectionName);
```

```
        }

        public async Task<List<Book>> GetAsync() =>
            await _booksCollection.Find(_ => true).ToListAsync();

        public async Task<Book?> GetAsync(string id) =>
            await _booksCollection.Find(x => x.Id == id).FirstOrDefaultAsync();

        public async Task CreateAsync(Book newBook) =>
            await _booksCollection.InsertOneAsync(newBook);

        public async Task UpdateAsync(string id, Book updatedBook) =>
            await _booksCollection.ReplaceOneAsync(x => x.Id == id, updatedBook);

        public async Task RemoveAsync(string id) =>
            await _booksCollection.DeleteOneAsync(x => x.Id == id);
    }
}
```

5. Add the Controller

In the Controllers folder include the following file:

BooksController.cs

```
using BookStoreApi.Models;
using BookStoreApi.Services;
using Microsoft.AspNetCore.Mvc;
using System.Data;

namespace BookStoreApi.Controllers;

[ApiController]
[Route("api/[controller]")]
public class BooksController : ControllerBase
{
    private readonly BooksService _booksService;

    public BooksController(BooksService booksService) =>
        _booksService = booksService;

    [HttpGet]
    public async Task<List<Book>> Get() =>
        await _booksService.GetAsync();

    [HttpGet("{id:length(24)}")]
    public async Task<ActionResult<Book>> Get(string id)
```

```
    {
        var book = await _booksService.GetAsync(id);

        if (book is null)
        {
            return NotFound();
        }

        return book;
    }

    [HttpPost]
    public async Task<IActionResult> Post(Book newBook)
    {
        await _booksService.CreateAsync(newBook);

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

    [HttpPut("{id:length(24)}")]
    public async Task<IActionResult> Update(string id, Book updatedBook)
    {
        var book = await _booksService.GetAsync(id);

        if (book is null)
        {
            return NotFound();
        }

        updatedBook.Id = book.Id;

        await _booksService.UpdateAsync(id, updatedBook);

        return NoContent();
    }

    [HttpDelete("{id:length(24)}")]
    public async Task<IActionResult> Delete(string id)
    {
        var book = await _booksService.GetAsync(id);

        if (book is null)
        {
            return NotFound();
        }

        await _booksService.RemoveAsync(id);

        return NoContent();
```

```
    }
}
```

6. Modify Program.cs file

In the Program.cs file include the following code:

```
using BookStoreApi.Models;
using BookStoreApi.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
ConfigurationManager Configuration = builder.Configuration;

// Add services to the container.
builder.Services.Configure<BookStoreDatabaseSettings>(
    builder.Configuration.GetSection("BookStoreDatabase"));

builder.Services.AddSingleton<BooksService>();

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

7. Modify appsettings.json file

In the appsettings.json file include the following code:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "BookStoreDatabase": {
    "ConnectionString": "mongodb://localhost:27017",
    "DatabaseName": "BookStore",
    "BooksCollectionName": "Books"
  },
  "AllowedHosts": "*"
}
```

Logging Configuration:

"**Logging**": This section configures the logging behavior of the application.

"**LogLevel**": This subsection specifies the minimum level of events to log.

"**Default**": "Information": By default, the application logs events that are at the "Information" level or higher. "Information" level typically includes general application flow events and operational information.

"**Microsoft.AspNetCore**": "**Warning**": For components from the "Microsoft.AspNetCore" namespace, only "Warning" level events or higher are logged. "Warning" level logs are used for potentially harmful situations or cautionary messages.

Database Configuration for a Book Store:

"**BookStoreDatabase**": This section contains settings specific to a database used by a Book Store application.

"**ConnectionString**": "mongodb://localhost:27017": Defines the connection string for the database. This particular string indicates that the application connects to a MongoDB instance running on localhost (the same machine where the application is running) and listening on port 27017.

"**DatabaseName**": "**BookStore**": Specifies the name of the database within MongoDB to be used, which is "BookStore" in this case.

"**BooksCollectionName**": "**Books**": Indicates the name of the collection within the "BookStore" database that will store the book data. In MongoDB, a collection is analogous to a table in a relational database.

Allowed Hosts Configuration:

"**AllowedHosts**": "": *This setting configures which hosts are allowed to send requests to the application. The asterisk ()* is a wildcard that means any host can send requests. This is an important setting for web applications, especially concerning CORS (Cross-Origin Resource Sharing) policies.

Overall, this JSON file is used to configure logging, database connections, and security policies for a web application, likely built with technologies like ASP.NET Core (indicated by the Microsoft.AspNetCore logging configuration).

8. How to run the application

8.1. First, we need to pull and run the MondoDB database Docker container image

For pulling the MondoDB docker image from Docker hub we run these commands

```
docker login
```

```
docker pull mongo
```

For running the Mongodb docker image we type the command:

```
docker run -d -p 27017:27017 --name mongodb-container mongo
```

To enter in the Mongodb database we type this command:

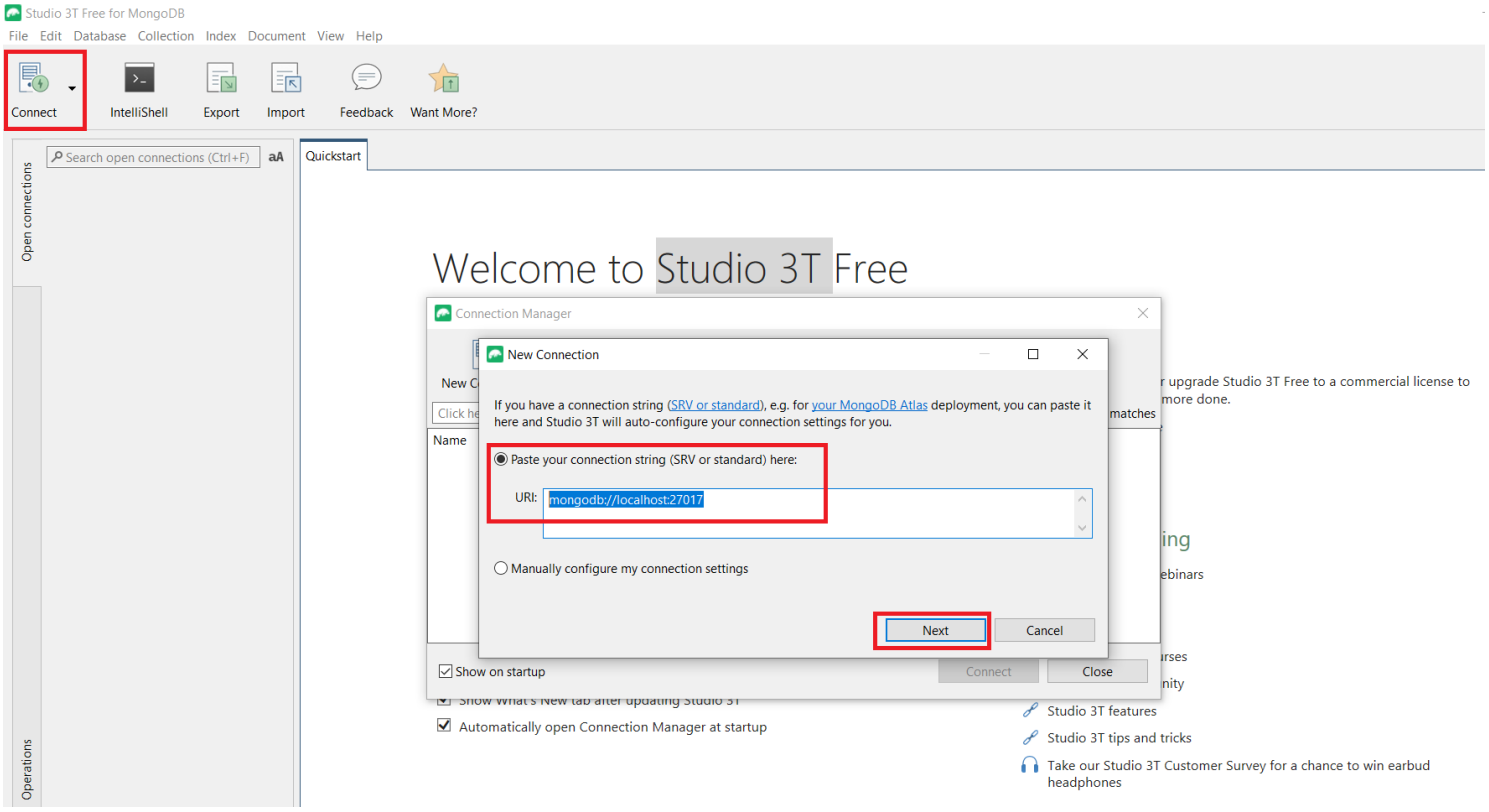
```
docker exec -it mongodb-container mongosh
```

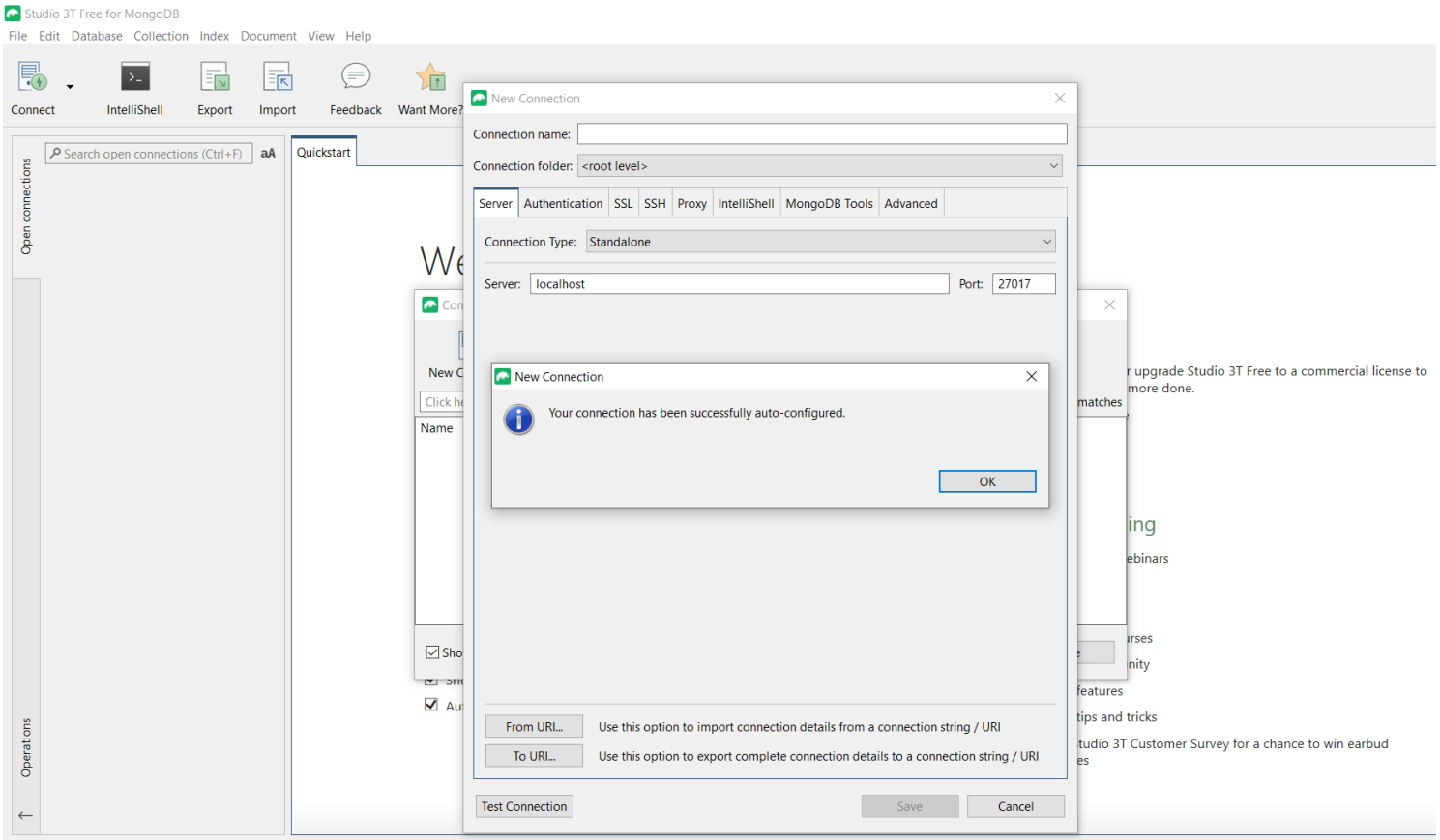
And also we create a new database with a collections and two documents inside, see this picture:

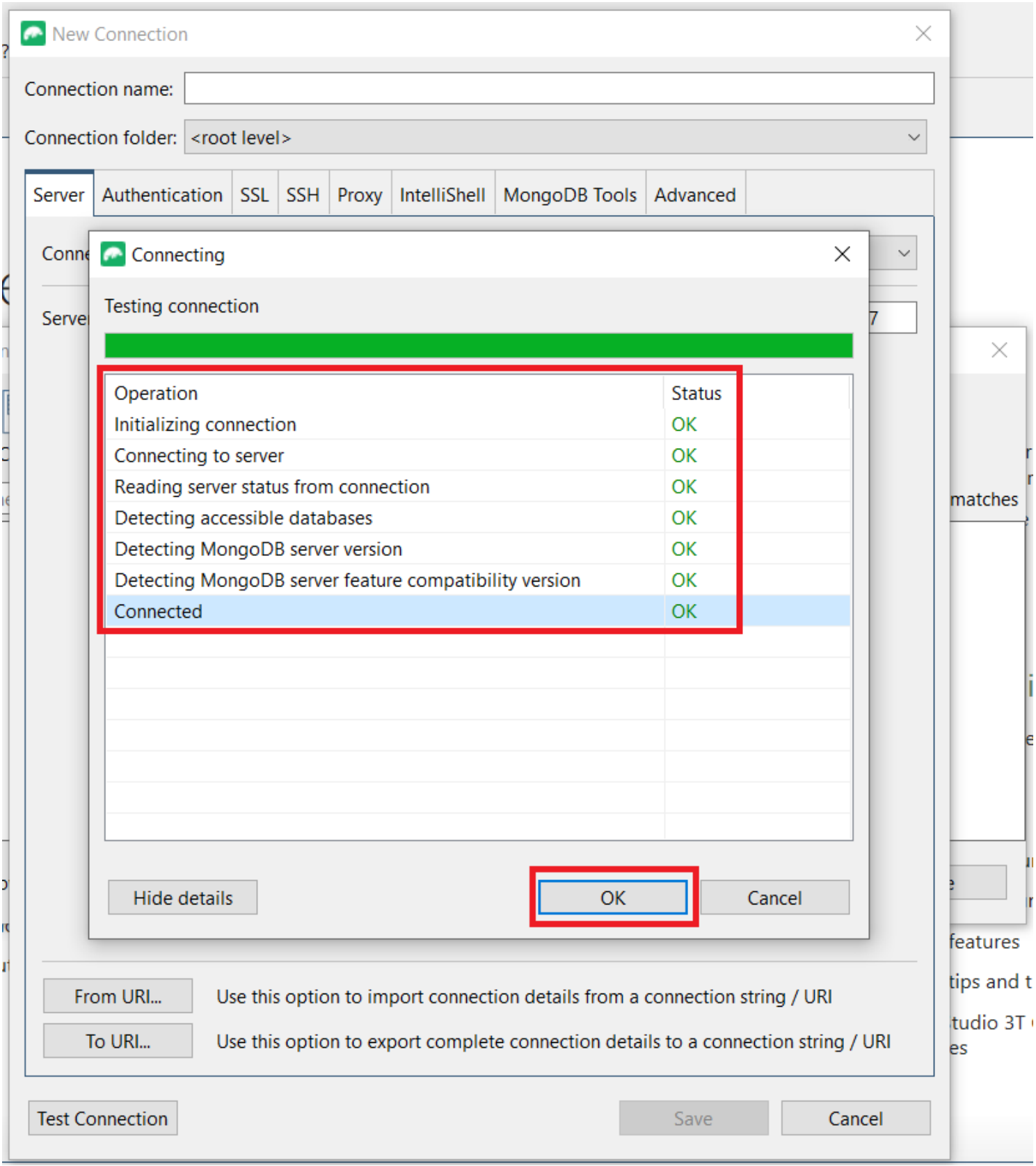
```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

C:\>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED       STATUS       PORTS                      NAMES
d70cfd450dc5   mongo     "docker-entrypoint.s…"   3 hours ago   Up 3 hours   0.0.0.0:27017->27017/tcp   mongodb-container

C:\>docker exec -it mongodb-container mongosh
Current Mongosh Log ID: 65953a9609f7dbf441d31980
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.1
Using MongoDB:          7.0.4
Using Mongosh:          2.1.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2024-01-03T07:59:00.138+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-01-03T07:59:00.766+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-01-03T07:59:00.766+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
   2024-01-03T07:59:00.766+00:00: vm.max_map_count is too low
------

test> use BookStore
switched to db BookStore
BookStore> db.createCollection('Books')
{ ok: 1 }
BookStore> show collections
Books
BookStore> db.Books.insertMany([{ "Name": "Design Patterns", "Price": 54.93, "Category": "Computers",
... "Author": "Ralph Johnson" }, { "Name": "Clean Code", "Price": 43.15, "Category":
... "Computers","Author": "Robert C. Martin" }])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65953adb09f7dbf441d31981'),
    '1': ObjectId('65953adb09f7dbf441d31982')
  }
}
BookStore> db.Books.find().pretty()
[
  {
    _id: ObjectId('65953adb09f7dbf441d31981'),
    Name: 'Design Patterns',
    Price: 54.93,
    Category: 'Computers',
    Author: 'Ralph Johnson'
BookStore>
```

8.2. Second, we verify the data with 3T Studio Free for MongoDb

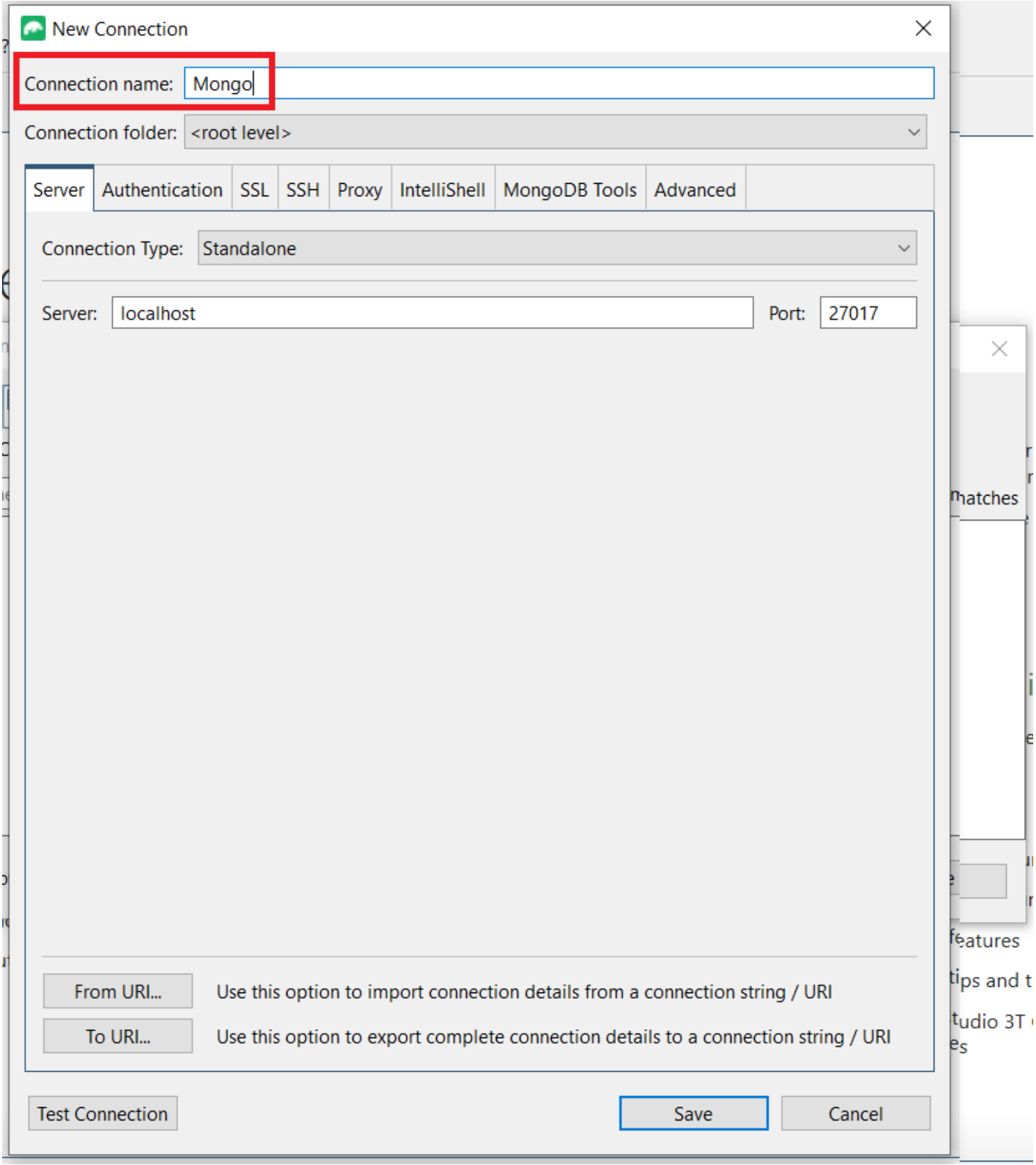
We connect to the MongoDB running container from 3T Studio setting the connection string: **mongodb://localhost:27017**


Studio 3T Free for MongoDB
File Edit Database Collection Index Document View Help
Connect
IntelliShell
Export
Import
Feedback
Want More?
Search open connections (Ctrl+F)
Quickstart
Open connections
Operations
Welcome to Studio 3T Free
Connection Manager
New Connection
If you have a connection string (SRV or standard), e.g. for your MongoDB Atlas deployment, you can paste it here and Studio 3T will auto-configure your connection settings for you.
Paste your connection string (SRV or standard) here:
URI: mongodb://localhost:27017
Manually configure my connection settings
Next
Cancel
Show on startup
Connect
Close
Automatically open Connection Manager at startup
Studio 3T features
Studio 3T tips and tricks
Take our Studio 3T Customer Survey for a chance to win earbud headphones

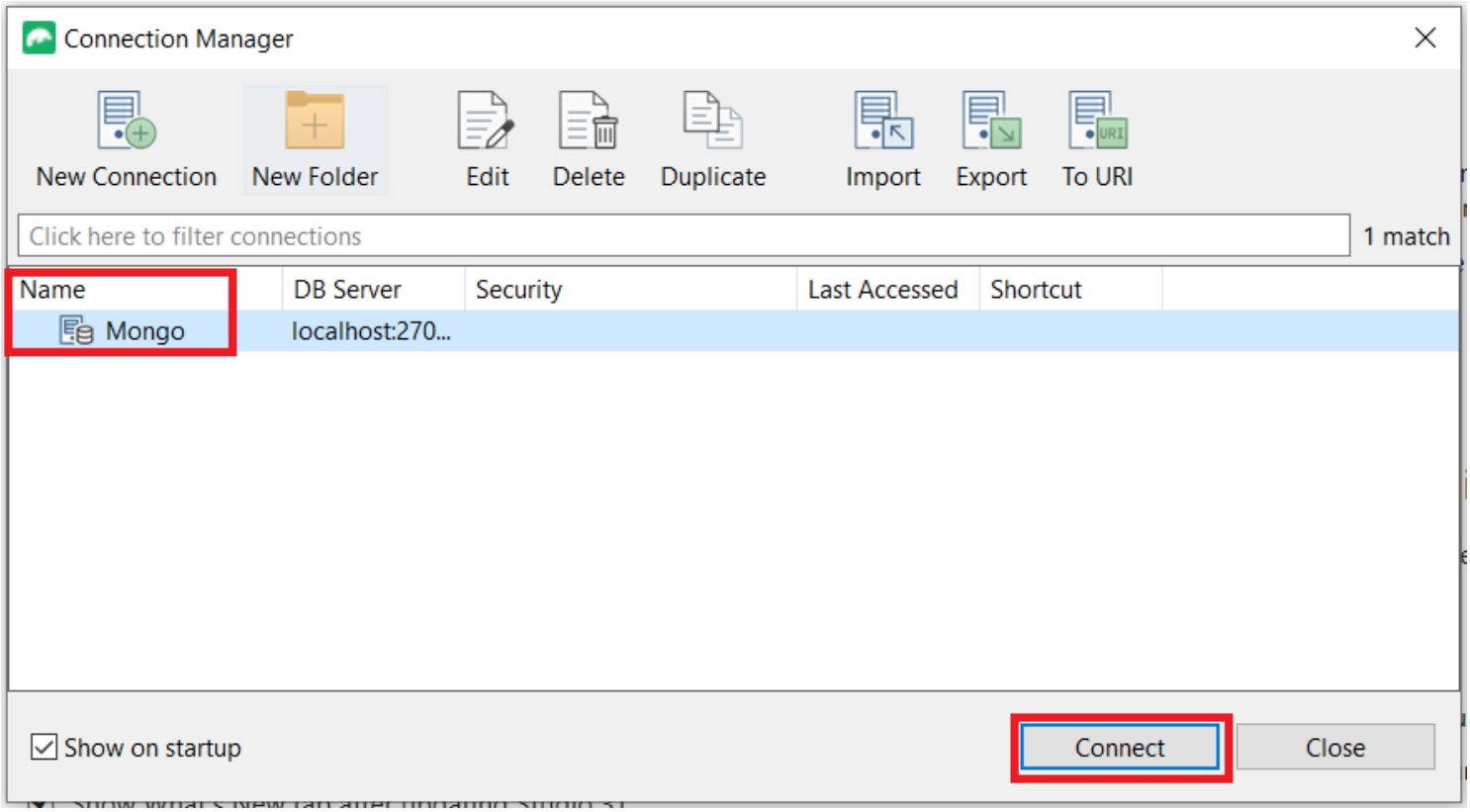

Studio 3T Free for MongoDB
File Edit Database Collection Index Document View Help
Connect
IntelliShell
Export
Import
Feedback
Want More?
Search open connections (Ctrl+F)
Quickstart
Open connections
Operations
New Connection
Connection name:
Connection folder: <root level>
Server Authentication SSL SSH Proxy IntelliShell MongoDB Tools Advanced
Connection Type: Standalone
Server: localhost
Port: 27017
New Connection
Your connection has been successfully auto-configured.
OK
From URI...
Use this option to import connection details from a connection string / URI
To URI...
Use this option to export complete connection details to a connection string / URI
Test Connection
Save
Cancel



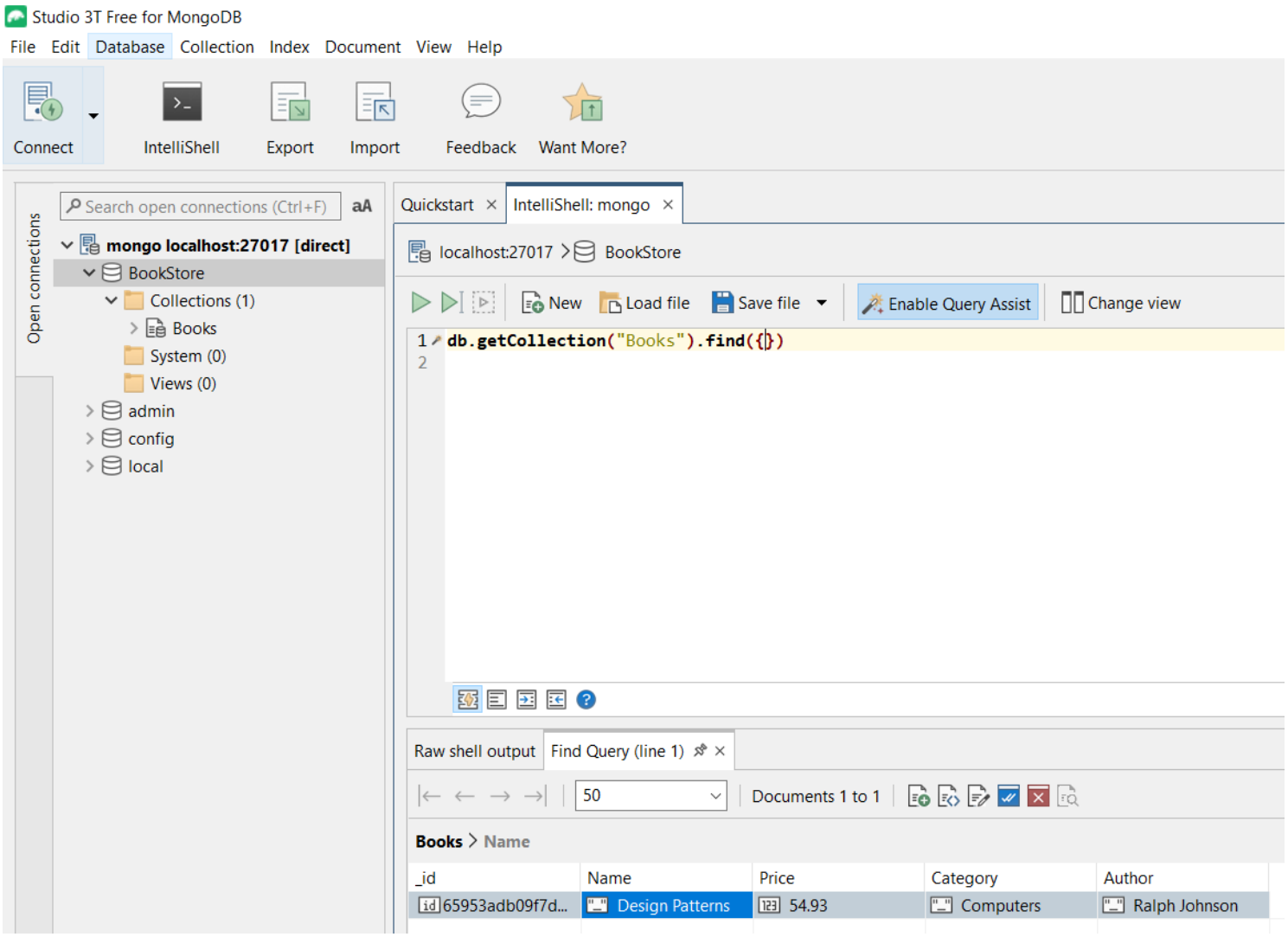
We set the connection name



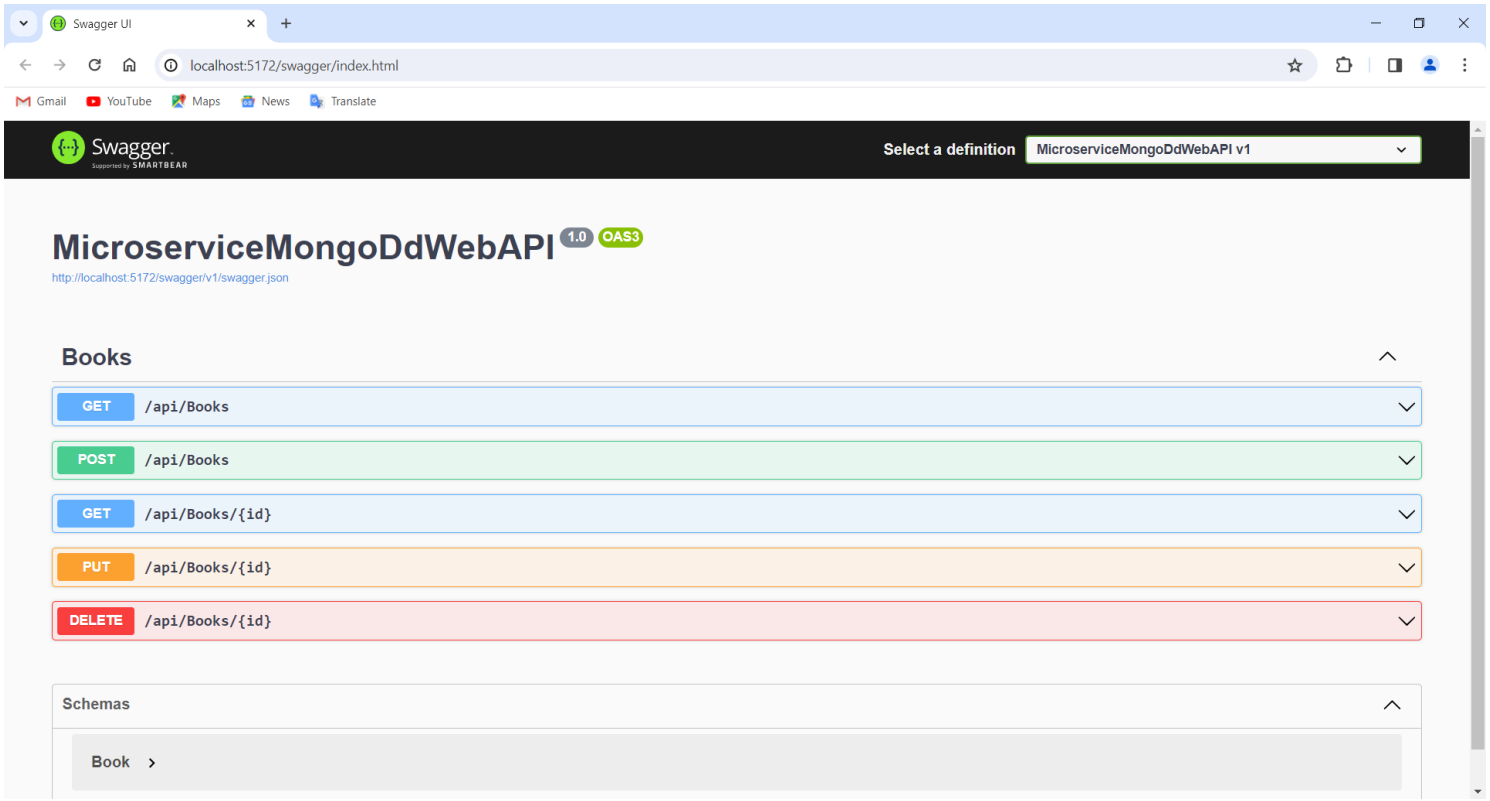
And we connect to the database

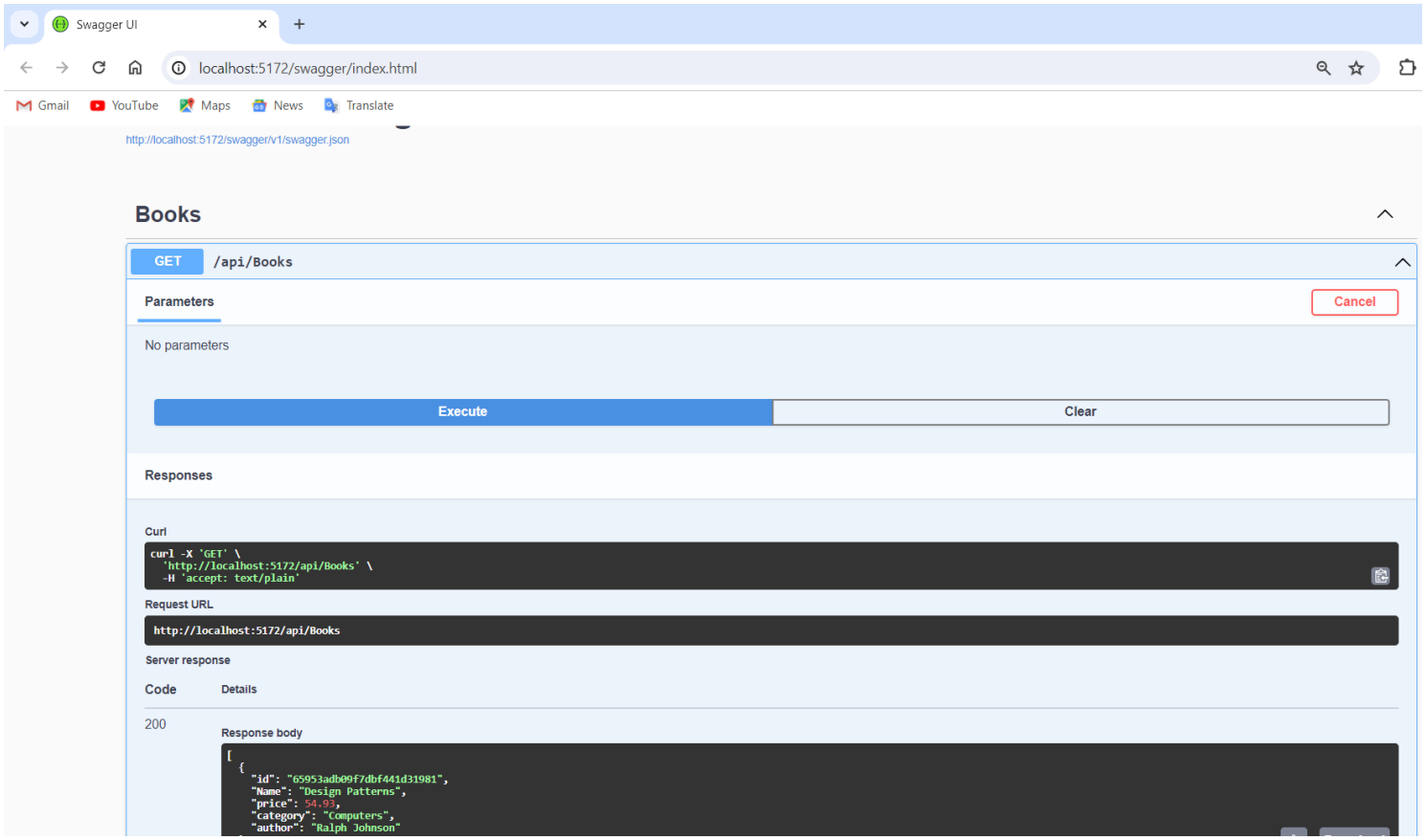


See the data inside the new database and collection



8.3. Third, we build and run the WebAPI application with HTTP protocol


Swagger UI
localhost:5172/swagger/index.html
Gmail
YouTube
Maps
News
Translate
Swagger
Select a definition
MicroserviceMongoDdWebAPI v1
MicroserviceMongoDdWebAPI
1.0
OAS3
http://localhost:5172/swagger/v1/swagger.json
Books
GET /api/Books
POST /api/Books
GET /api/Books/{id}
PUT /api/Books/{id}
DELETE /api/Books/{id}
Schemas
Book


Swagger UI
localhost:5172/swagger/index.html
Gmail
YouTube
Maps
News
Translate
http://localhost:5172/swagger/v1/swagger.json
Books
GET /api/Books
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:5172/api/Books' \
-H 'accept: text/plain'
Request URL
http://localhost:5172/api/Books
Server response
Code
Details
200
Response body
[
{
"id": "65953adb09f7dbf441d31981",
"Name": "Design Patterns",
"price": 54.93,
"category": "Computers",
"author": "Ralph Johnson"