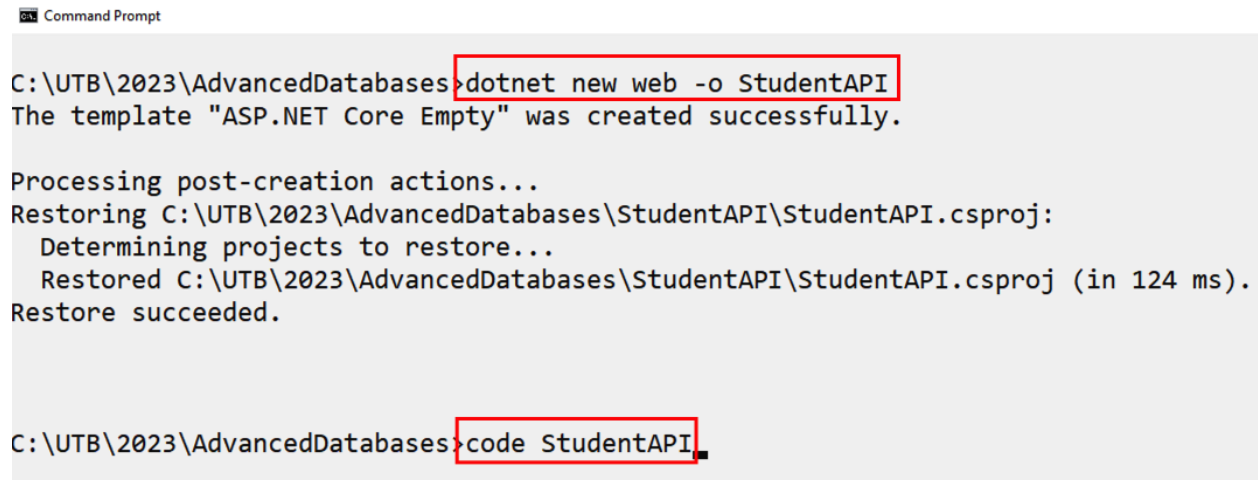


Task 1. Set up the project

Open a Terminal / CLI, navigate to any desired location, and run the following two commands in order to **generate a new empty ASP.NET Core web API project** and then **open the project in Visual Studio Code**:

```
dotnet new webapi -o StudentAPI
code StudentAPI
```



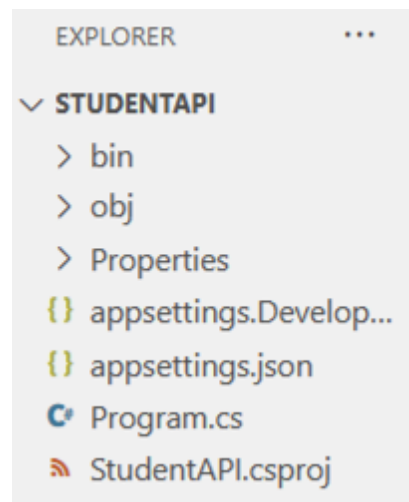
```
Command Prompt

C:\UTB\2023\AdvancedDatabases>dotnet new web -o StudentAPI
The template "ASP.NET Core Empty" was created successfully.

Processing post-creation actions...
Restoring C:\UTB\2023\AdvancedDatabases\StudentAPI\StudentAPI.csproj:
  Determining projects to restore...
  Restored C:\UTB\2023\AdvancedDatabases\StudentAPI\StudentAPI.csproj (in 124 ms).
Restore succeeded.

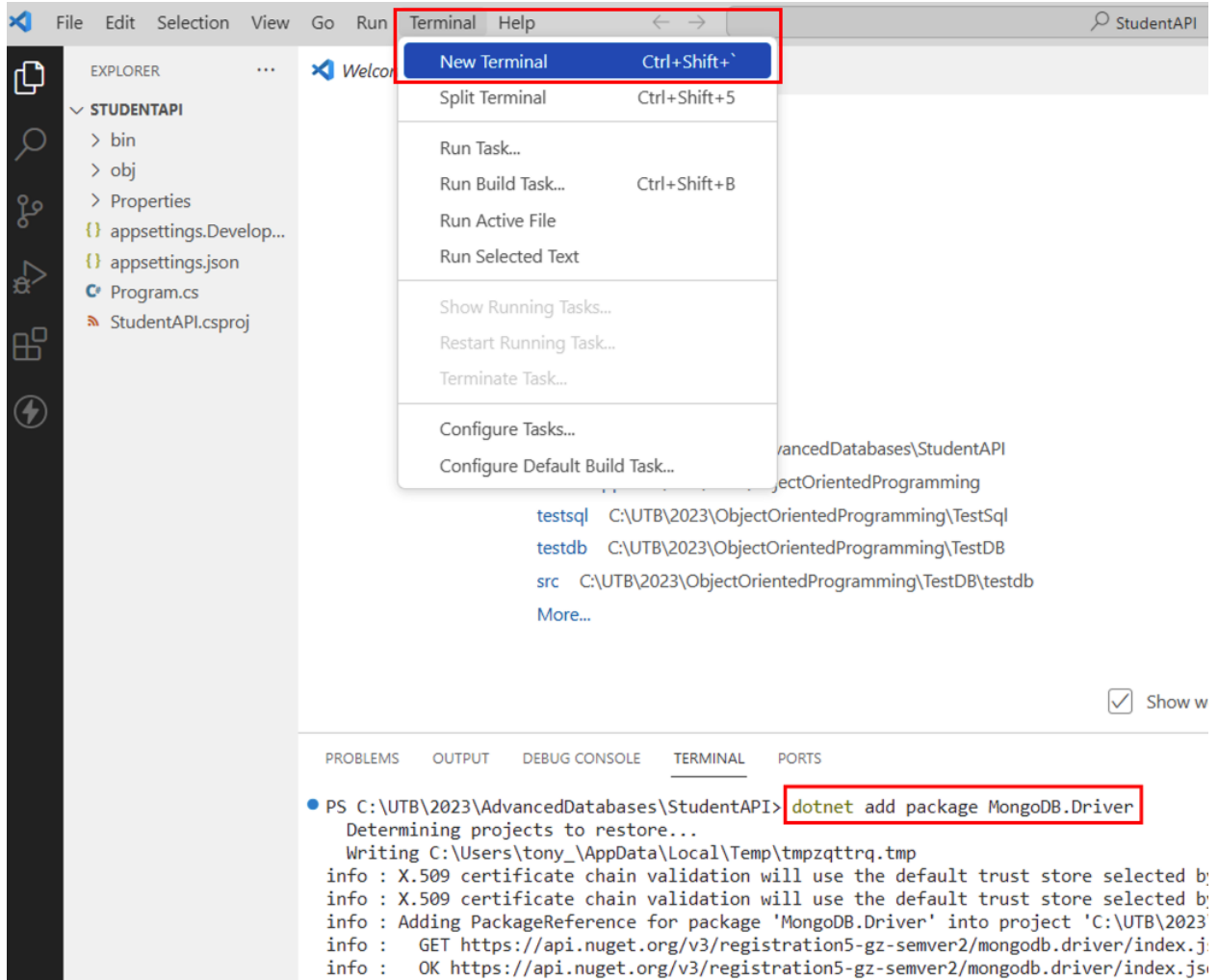
C:\UTB\2023\AdvancedDatabases>code StudentAPI
```

Once the project is opened by Visual Studio Code, you will see that several files are included:



Now, open the Integrated Terminal and run the following command to install the .NET driver for MongoDB:

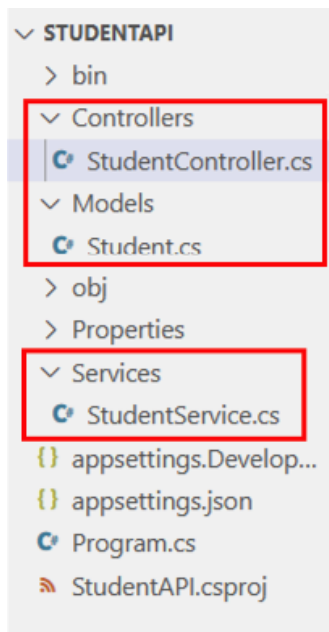
dotnet add package MongoDB.Driver



Create the following structure inside your project (files and folders):

StudentAPI

- Models (folder)
 - Student.cs
- Services (folder)
 - StudentService.cs
- Controllers (folder)
 - StudentController.cs



The **StudentAPI** directory includes the following files:

- **Models/Student.cs:** It's a class that represents the collection (table) in MongoDB.
- **Models/StudentDatabaseSettings.cs:** It's a class that stores the values from appsettings
- **Services/StudentService.cs:** It's a class that exposes a service with methods which directly interact with the MongoDB database.
- **Controllers/StudentController.cs:** It's a class that provides HTTP methods to run CRUD operations over the database through the service class.
- **appsettings.json:** It contains the database configuration values.
- **Program.cs:** It's the main class (entry point) of the application.

Task 2. Implement the database connection

A connection string is required to set up a communication with the database. In this case, the default values for a local MongoDB server (IP and port) are used. Add the following database (and collection) configuration values to **appsettings.json**:

```
{  
  "StudentDatabase": {  
    "ConnectionString": "mongodb://localhost:27017",  
    "DatabaseName": "StudentsDB",  
    "StudentsCollectionName": "Students"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*" }  
}
```

Add the following code to the **StudentDatabaseSettings.cs** class in the **Models** directory. This class models and stores the database configuration from **appsettings.json**:

```
Models > StudentDatabaseSettings.cs > StudentDatabaseSettings  
1 namespace StudentApi.Models;  
2  
3 0 references  
4 public class StudentDatabaseSettings  
5 {  
6 0 references  
7     public string ConnectionString { get; set; } = null!;  
8  
9 0 references  
10    public string DatabaseName { get; set; } = null!;  
11  
12 0 references  
13    public string StudentsCollectionName { get; set; } = null!;  
14 }  
15
```

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

```
Program.cs
1  using StudentApi.Models;
2
3  var builder = WebApplication.CreateBuilder(args);
4
5  builder.Services.Configure<StudentDatabaseSettings>(
6      builder.Configuration.GetSection("StudentDatabase"));
7
8  var app = builder.Build();
9
10 app.MapGet("/", () => "Hello World!");
11
12 app.Run();
13
```

In the preceding code, the **using** statement resolves the **StudentDatabaseSettings** reference. Next, the configuration instance to which the **appsettings.json** file's **StudentDatabase** section binds is registered in the Dependency Injection (DI) container. For example, the **ConnectionString** property from a **StudentDatabaseSettings** instance is populated with the **ConnectionString** property from the **StudentDatabase** section in **appsettings.json**.

Task 3. Define a model

A model class is required to reference a collection (table) in MongoDB. Add the following code for the **Student** class in the **Models** directory:

```
Models > Student.cs > ...
1  using MongoDB.Bson;
2  using MongoDB.Bson.Serialization.Attributes;
3
4  namespace StudentApi.Models;
5
6  public class Student
7  {
8      [BsonId]
9      [BsonRepresentation(BsonType.ObjectId)]
10     public string? Id { get; set; }
11
12     public string FirstName { get; set; } = null!;
13
14     public string LastName { get; set; } = null!;
15
16     public int Age { get; set; }
17 }
```

In the preceding class, the **Id** property is:

- Required for mapping the Common Language Runtime (CLR) object to the MongoDB collection.
- Annotated with **[BsonId]** to make this property the document's primary key.
- Annotated with **[BsonRepresentation(BsonType.ObjectId)]** to allow passing the parameter as type string instead of an ObjectId structure. MongoDB handles the conversion from string to ObjectId.

Task 4. Implement the Service

This is the code of the **StudentService** class in the **Services** directory:

```
Services > StudentService.cs > ...
1  using MongoDB.Driver;
2  using Microsoft.Extensions.Options;
3
4  using StudentApi.Models;
5
6  namespace StudentApi.Services;
7
8  public class StudentsService
9  {
10     6 references
11     private readonly IMongoCollection<Student> studentCollection;
12
13     0 references
14     public StudentsService(
15         IOptions<StudentDatabaseSettings> studentDatabaseSettings)
16     {
17         var mongoClient = new MongoClient(
18             studentDatabaseSettings.Value.ConnectionString);
19         var mongoDatabase = mongoClient.GetDatabase(
20             studentDatabaseSettings.Value.DatabaseName);
21         studentCollection = mongoDatabase.GetCollection<Student>(
22             studentDatabaseSettings.Value.StudentsCollectionName);
23     }
24 }
```

```

25 | 0 references
    | public async Task<List<Student>> GetAsync() =>
26 |     await studentCollection.Find(_ => true).ToListAsync();
27 |
    | 0 references
28 | public async Task<Student?> GetAsync(string id) =>
29 |     await studentCollection.Find(x => x.Id == id).FirstOrDefaultAsync();
30 |
    | 0 references
31 | public async Task CreateAsync(Student newStudent) =>
32 |     await studentCollection.InsertOneAsync(newStudent);
33 |
    | 0 references
34 | public async Task UpdateAsync(string id, Student updatedStudent) =>
35 |     await studentCollection.ReplaceOneAsync(x => x.Id == id, updatedStudent);
36 |
    | 0 references
37 | public async Task RemoveAsync(string id) =>
38 |     await studentCollection.DeleteOneAsync(x => x.Id == id);
39 | }

```

In the preceding code, a **StudentDatabaseSettings** instance is retrieved from DI via **constructor injection**. This technique provides access to the **appsettings.json** database configuration values.

Constructor details:

The **StudentsService** class uses the following MongoDB.Driver members to run CRUD operations against the database:

- **MongoClient:** Reads the server instance for running database operations. The constructor of this class is provided by the **MongoDB** connection string:
- **IMongoDatabase:** Represents the MongoDB database for running operations using the generic **GetCollection** method on the interface to gain access to data in a specific collection (**Students**). You can run CRUD operations against the collection after this method is called.
- **MongoCollection:** Represents the collection and allows using the following methods, which are used in the 5 public functions:
 - **Find:** Returns all documents in the collection matching the provided search criteria.
 - **InsertOneAsync:** Inserts the provided object as a new document in the collection.
 - **ReplaceOneAsync:** Replaces the single document matching the provided search criteria with the provided object.
 - **DeleteOneAsync:** Deletes a single document matching the provided search criteria.

Finally, the 5 public methods defined which represent CRUD operations are:

- **GetAsync**: this method retrieves all students from the collection
- **GetAsync**: an overloaded method that retrieves a specific student from the collection
- **CreateAsync**: a method that inserts a student in the collection
- **UpdateAsync**: a method that updates a specific student in the collection
- **RemoveAsync**: a method that deletes a specific student in the collection

Now, add the following highlighted code to **Program.cs**:

```
Program.cs
1  using StudentApi.Models;
2  using StudentApi.Services;
3
4  var builder = WebApplication.CreateBuilder(args);
5
6  builder.Services.Configure<StudentDatabaseSettings>(
7      builder.Configuration.GetSection("StudentDatabase"));
8
9  builder.Services.AddSingleton<StudentsService>();
10
11 var app = builder.Build();
12
```

In the preceding code, the **using** statement resolves the **StudentsService** reference: the **StudentsService** class is registered with DI to support constructor injection in consuming classes. The singleton service lifetime is most appropriate because **StudentsService** takes a direct dependency on **MongoClient**. *Per the official Mongo Client reuse guidelines, MongoClient should be registered in DI with a singleton service lifetime.*

Task 5. Implement the Controller (REST Endpoints)

The goal of this tutorial is to expose **REST API** routes (action methods) to support **GET, POST, PUT, and DELETE HTTP** requests as part of a backend application.

Add the following code to the **StudentController.cs** file in **Controllers** directory:

```
Controllers > StudentController.cs > ...
1  using Microsoft.AspNetCore.Mvc;
2
3  using StudentApi.Models;
4  using StudentApi.Services;
5
6  namespace StudentApi.Controllers;
7
8  [ApiController]
9  [Route("api/[controller]")]
10 public class StudentsController : ControllerBase
11 {
12     private readonly StudentsService studentsService;
13
14     public StudentsController(StudentsService studentsService) =>
15     {
16         studentsService = studentsService;
17     }
18
19     [HttpGet]
20     public async Task<List<Student>> Get() =>
21     {
22         await studentsService.GetAsync();
23     }
24
25     [HttpGet("{id:length(24)}")]
26     public async Task<ActionResult<Student>> Get(string id)
27     {
28         var Student = await studentsService.GetAsync(id);
29
30         if (Student is null)
31         {
32             return NotFound();
33         }
34
35         return Student;
36     }
37 }
```

```

32 | [HttpPost]
    | 0 references
33 | public async Task<IActionResult> Post(Student newStudent)
34 | {
35 |     await studentsService.CreateAsync(newStudent);
36 |
37 |     return CreatedAtAction(nameof(Get),
38 |         new { id = newStudent.Id }, newStudent);
39 | }

41 | [HttpPut("{id:length(24)}")]
    | 0 references
42 | public async Task<IActionResult> Update(string id, Student updatedStudent)
43 | {
44 |     var Student = await studentsService.GetAsync(id);
45 |
46 |     if (Student is null)
47 |     {
48 |         return NotFound();
49 |     }
50 |     updatedStudent.Id = Student.Id;
51 |     await studentsService.UpdateAsync(id, updatedStudent);
52 |
53 |     return NoContent();
54 | }
55 |
56 | [HttpDelete("{id:length(24)}")]
    | 0 references
57 | public async Task<IActionResult> Delete(string id)
58 | {
59 |     var Student = await studentsService.GetAsync(id);
60 |
61 |     if (Student is null)
62 |     {
63 |         return NotFound();
64 |     }
65 |     await studentsService.RemoveAsync(id);
66 |
67 |     return NoContent();
68 | }

```

The **Get** route uses the **StudentService**'s **GetAsync** method to retrieve all documents in the **Students** collection, which are sent back in the response. It has an overload with an **id** argument which finds a specific document in the **Students** collection invoking the corresponding overload in the service. **NotFound** sends a **404** HTTP Status code if the student doesn't exist, while a **200** OK HTTP Status code is returned along with the data if successful.

The **Post** route **adds a new student** in the Students collection. The body of this POST method passes the **student** document that is inserted. Adding data is done via the **StudentService's CreateAsync()**. The **CreatedAtAction** function returns a **201** HTTP status code (Created) and the document (with the id) that was just added is retrieved as well.

The **Update** route looks for the **Student** document specified by the **id** argument and replaces its content with **updatedStudent** document that is also sent in the arguments (in reality, it's in the body request of a PUT request). As expected, the operation is performed by the **StudentService's UpdateAsync** method. If the document doesn't exist, a **401** Not Found HTTP Status code response is sent back.

Finally, we can remove information from the database via the **Delete** route, which includes the **id** argument that is used to find a specific student through the **StudentService's RemoveAsync** method. A **NoContent (204)** HTTP status code is returned in the response.

Now that we have everything in place, we can test the backend.

Task 6. Configure Program.cs

Add the highlighted statements in **Program.cs**:

```
Program.cs
1  using StudentApi.Models;
2  using StudentApi.Services;
3
4  var builder = WebApplication.CreateBuilder(args);
5
6  builder.Services.Configure<StudentDatabaseSettings>(
7      builder.Configuration.GetSection("StudentDatabase"));
8
9  builder.Services.AddSingleton<StudentsService>();
10
11  builder.Services.AddControllers()
12      .AddJsonOptions(
13          options => options.JsonSerializerOptions.PropertyNamingPolicy = null);
14
15  var app = builder.Build();
16
17  app.UseHttpsRedirection();
18
19  app.UseAuthorization();
20
21  app.MapControllers();
22
23  app.Run();
24  |
```

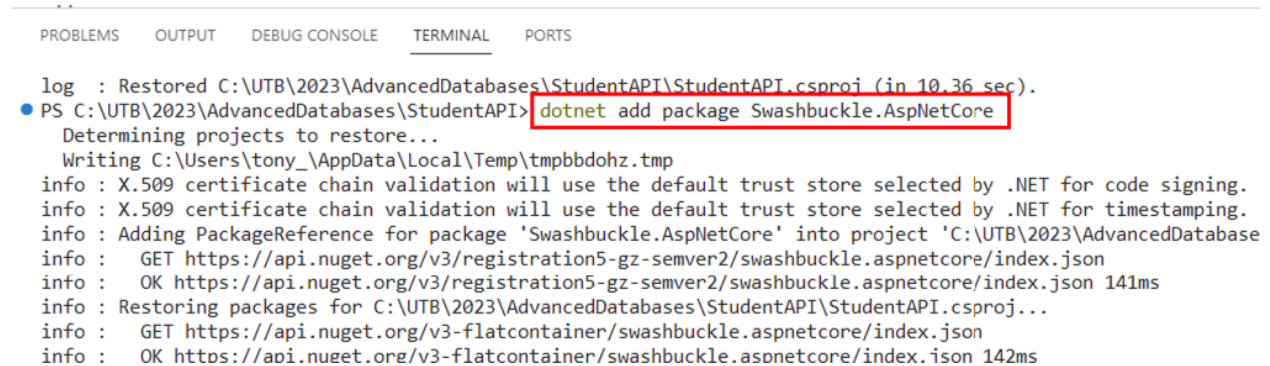
With the first preceding change, property names in the web API's serialized JSON response match their corresponding property names in the CLR object type. For example, the **Student** class's **Age** property serializes as **Age** instead of **age**.

The second section:

- Adds a middleware for redirecting HTTP requests to HTTPS.
- Adds a middleware for authorization capabilities.
- Adds endpoints for controller actions.

Now, use the integrated terminal and run the following command to add a new NuGet package:


dotnet add package Swashbuckle.AspNetCore



```
log : Restored C:\UTB\2023\AdvancedDatabases\StudentAPI\StudentAPI.csproj (in 10.36 sec).
● PS C:\UTB\2023\AdvancedDatabases\StudentAPI> dotnet add package Swashbuckle.AspNetCore
Determining projects to restore...
Writing C:\Users\tony\AppData\Local\Temp\tmpbbdohz.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Swashbuckle.AspNetCore' into project 'C:\UTB\2023\AdvancedDatabase
info : GET https://api.nuget.org/v3/registration5-gz-semver2/swashbuckle.aspnetcore/index.json
info : OK https://api.nuget.org/v3/registration5-gz-semver2/swashbuckle.aspnetcore/index.json 141ms
info : Restoring packages for C:\UTB\2023\AdvancedDatabases\StudentAPI\StudentAPI.csproj...
info : GET https://api.nuget.org/v3-flatcontainer/swashbuckle.aspnetcore/index.json
info : OK https://api.nuget.org/v3-flatcontainer/swashbuckle.aspnetcore/index.json 142ms
```

This open source package allows you to add **Swagger** tools support for documenting your API. Among other capabilities, you will be able to directly test your endpoints without adding any other software (such as Postman or the Thunder Client extension).

In order to enable it, you need to add **four** new statements in **Program.cs**. They are highlighted in the following picture:



```
11 builder.Services.AddControllers()
12     .AddJsonOptions(
13         options => options.JsonSerializerOptions.PropertyNamingPolicy = null);
14
15 builder.Services.AddEndpointsApiExplorer();
16 builder.Services.AddSwaggerGen();
17
18 var app = builder.Build();
19
20 app.UseSwagger();
21 app.UseSwaggerUI();
22
23 app.UseHttpsRedirection();
24
25 app.UseAuthorization();
26
27 app.MapControllers();
28
29 app.Run();
```

Task 7. Build and run the app

Save all files, then run the **dotnet build** command using the integrated terminal:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\UTB\2023\AdvancedDatabases\StudentAPI> dotnet build
MSBuild version 17.8.3+195e7f5a3 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
StudentAPI -> C:\UTB\2023\AdvancedDatabases\StudentAPI\bin\Debug\net8.0\StudentAPI.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)
```

If an error is found, fix it. On the other hand, if the build was successful, then execute the **dotnet run** following command to run your app:

```
Time Elapsed: 00:00:00.775
PS C:\UTB\2023\AdvancedDatabases\StudentAPI> dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5055
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\UTB\2023\AdvancedDatabases\StudentAPI
■
```

In this test, the app is running in port **5055** so open a browser and use the following URL:
<http://localhost:5055/swagger/index.html>

NOTE: If you get a different port, use it.

The result is presented in the next page:

Swagger UI

localhost:5055/swagger/index.html

Select a definition **StudentAPI v1**

StudentAPI ^{1.0} ^{OAS3}

<http://localhost:5055/swagger/v1/swagger.json>

Students

- GET** /api/Students
- POST** /api/Students
- GET** /api/Students/{id}
- PUT** /api/Students/{id}
- DELETE** /api/Students/{id}

Schemas

- Student** >

This view was generated by Swashbuckle and displays each controller included in your project. By clicking on the arrow, the details to send a request to the endpoint will be shown.

For example, in the next page you can see the details of the Post endpoint to add a new student:

Students

GET /api/Students

POST /api/Students

Parameters

No parameters

Request body

application/json

Try it out

Example Value | Schema

```
{
  "Id": "string",
  "FirstName": "string",
  "LastName": "string",
  "Age": 0
}
```

Responses

Code	Description	Links
200	Success	No links

You can see in the above picture information about this method that you already know (because it was added in the code of this endpoint):

- The endpoint is <http://localhost:5055/api/Students>
- The endpoint accepts a POST request
- The request body expects a JSON string content. An example value, with the expected format that includes the properties (Id, FirstName...) from a Student document, is presented.
- The method returns the 200 Success HTTP Status code.

You need to click on the **Try it out** button to get access to an entry and a button to test your endpoint. You can see this in the next page:

POST /api/Students

Parameters

Cancel

No parameters

Request body

application/json

```
{
  "Id": "string",
  "FirstName": "string",
  "LastName": "string",
  "Age": 0
}
```

Execute

When you click on Execute, the corresponding code is invoked. You can see the HTTP response after this.

Let's test each of the endpoints in the next task so you can observe what's the response you get in each case!

Task 8. Test the routes

a) Add a student

Sample request:

POST	/api/Students
Parameters	
No parameters	
Request body	
<pre>{ "Id": "61a6058e6c43f32854e51f51", "FirstName": "John", "LastName": "Doe", "Age": 40 }</pre>	

Sample response:

Curl	
<pre>curl -X 'POST' \ 'http://localhost:5055/api/Students' \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "Id": "61a6058e6c43f32854e51f51", "FirstName": "John", "LastName": "Doe", "Age": 40 }'</pre>	
Request URL	
<pre>http://localhost:5055/api/Students</pre>	
Server response	
Code	Details
201 <i>Undocumented</i>	Response body <pre>{ "Id": "61a6058e6c43f32854e51f51", "FirstName": "John", "LastName": "Doe", "Age": 40 }</pre>

b) **Get students**

Sample request:

GET	/api/Students
Parameters	
No parameters	
Execute	

Sample response:

Curl

```
curl -X 'GET' \
  'http://localhost:5055/api/Students' \
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5055/api/Students
```

Server response

Code

Details

200

Response body

```
[
  {
    "Id": "61a6058e6c43f32854e51f51",
    "FirstName": "John",
    "LastName": "Doe",
    "Age": 40
  }
]
```

c) Get a student

Sample request:

GET	/api/Students/{id}
Parameters	
Name	Description
id * required	
string (path)	61a6058e6c43f32854e51f51

Sample response:

Curl

```
curl -X 'GET' \
  'http://localhost:5055/api/Students/61a6058e6c43f32854e51f51' \
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5055/api/Students/61a6058e6c43f32854e51f51
```

Server response

Code

Details

200

Response body

```
{
  "Id": "61a6058e6c43f32854e51f51",
  "FirstName": "John",
  "LastName": "Doe",
  "Age": 40
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Tue,05 Dec 2023 12:55:44 GMT
server: Kestrel
transfer-encoding: chunked
```

d) Update a student

Sample request:

PUT	/api/Students/{id}
Parameters	
Name	Description
id * required	
string	61a6058e6c43f32854e51f51
(path)	
Request body	
<pre>{ "Id": "61a6058e6c43f32854e51f51", "FirstName": "John Michael", "LastName": "Doe III", "Age": 35 }</pre>	

Sample response:

Curl	
<pre>curl -X 'PUT' \ 'http://localhost:5055/api/Students/61a6058e6c43f32854e51f51' \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "Id": "61a6058e6c43f32854e51f51", "FirstName": "John Michael", "LastName": "Doe III", "Age": 35 }'</pre>	
Request URL	
<pre>http://localhost:5055/api/Students/61a6058e6c43f32854e51f51</pre>	
Server response	
Code	Details
204	Response headers
Undocumented	<pre>date: Tue,05 Dec 2023 13:10:35 GMT server: Kestrel</pre>

e) Delete a student:

Sample request:

DELETE /api/Students/{id}	
Parameters	
Name	Description
id * required	
string	61a6058e6c43f32854e51f51
(path)	

Sample response:

Curl	
<pre>curl -X 'DELETE' \ 'http://localhost:5055/api/Students/61a6058e6c43f32854e51f51' \ -H 'accept: */*'</pre>	
Request URL	
<pre>http://localhost:5055/api/Students/61a6058e6c43f32854e51f51</pre>	
Server response	
Code	Details
204	
Undocumented	Response headers
	<pre>date: Tue,05 Dec 2023 13:13:00 GMT server: Kestrel</pre>