|  |  |
| --- | --- |
|  | LAB | REST APIs with Swagger |
|  | WORKSHOP | Design Patterns |
|  | Crystal Tenn  crtenn@microsoft.com |

# SETUP LAB ENVRIOMENT

## Prerequisites

To perform the tasks in this lab you need following:

* Visual Studio 2015 or 2017, any version

## Objectives

In this lab you will complete following tasks:

* Create a simple Console calculator application with dependency injection using Mef

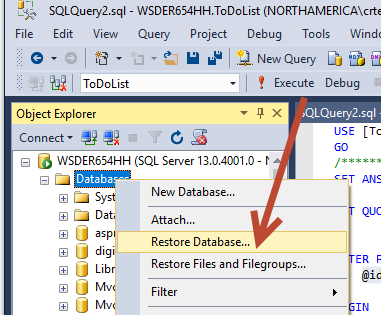
### Estimated Completion Time: 1 hour

<https://github.com/catenn/ToDoListAzure>

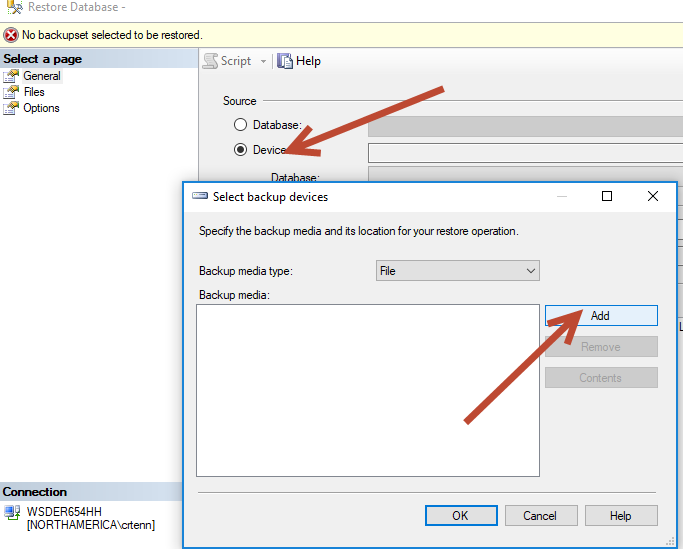
**Task: Setup SQL and Stored Procedures**

In this task you will create a new Console application.

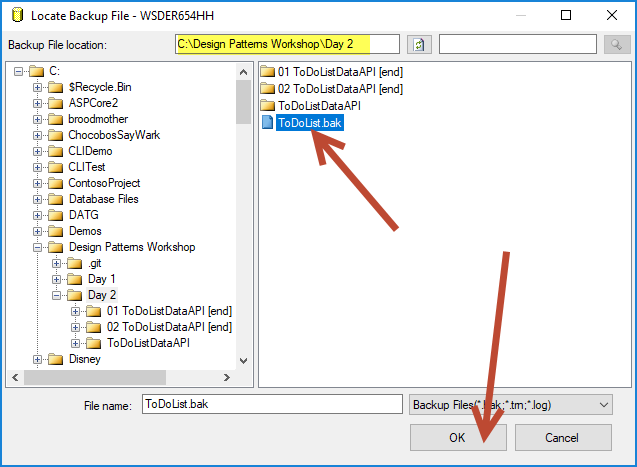
1. Open SQL Server Management Studio and login to your local database, most likely with your Windows credentials.
2. Once you are logged in, right click on Databases. Choose Restore Database..



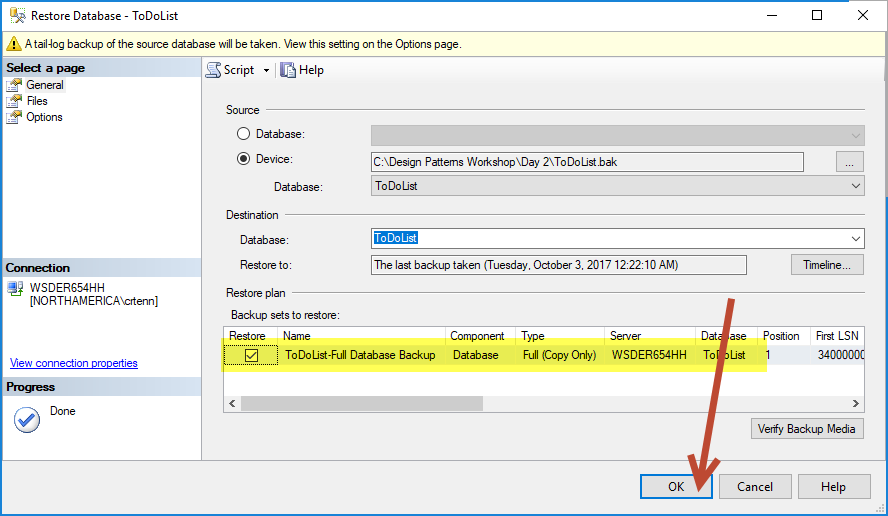
1. Select Device. On the modal pop-up, select Add.



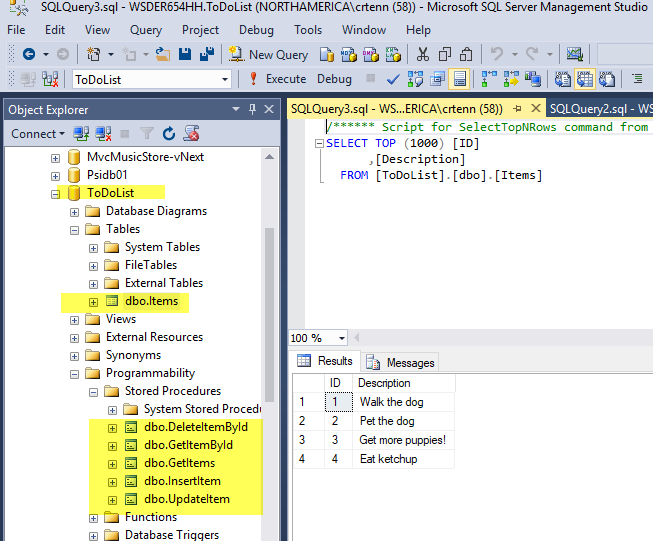
1. On the Day 2 folder, choose the ToDoList.bak file. Hit OK.



1. Hit OK to close the modal.
2. Hit OK on the Restore Database screen to start the Restore.



1. Once your DB is done restoring, you should see your ToDoList database. Click it open and check that you have one Table called dbo.Items. And check that you have 5 stored procedures. It should look like the screenshot below:

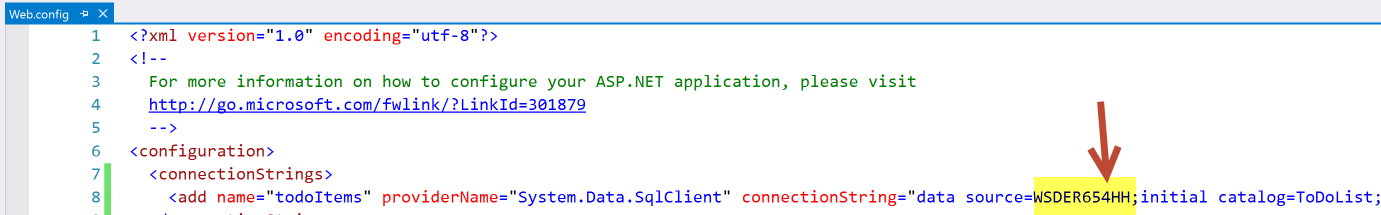


1. Right click dbo.Items to Select the top 1000 rows.
2. Check into what each Stored Procedure is doing to understand how the app will work.
3. Once you have verified everything is there you are all done setting up your local database!

## Task: Setup your REST APIs

In this task you will create a new Console application.

1. Open the ToDoListDataAPI solution.
2. Go to your web.config file and edit your connection string. Change the highlighted portion to the name of your local SQL Server. The other settings should remain the same as they are correct since you imported a copy of the same database.



1. Please go to the ToDoListManager.cs file and understand what is happening in this file.
2. Please go to the ToDoListRepo.cs file and understand what is happening in this file.
3. Create a new Controller in your Controllers file directory called ToDoListController.cs. Add the following code to your controller:

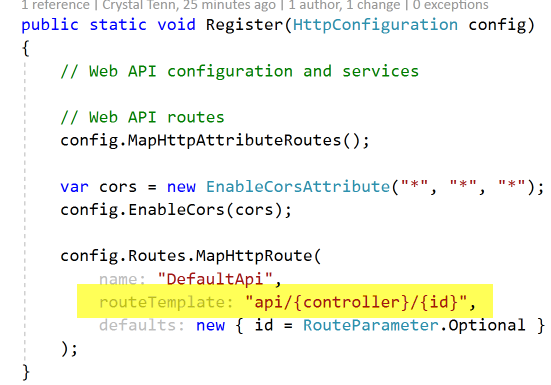
public class ToDoListController : ApiController

{

ToDoListManager \_manager = new ToDoListManager();

}

1. Go to the App\_Start folder and open up the WebApiConfig.cs file. Note the route template on line 23:



1. Now let’s add a Get method for getting all of our ToDoItems. You should add a comment with a URL example. We know the format will be api/ToDoList because of the format in the route template in the screenshot above that showed api/{controller} as the format. Make sure you use the plural “Items”, optionally to be more specific you could say “GetAllTodoItems” and add the word “All”. The naming conventions are important because many times the developer who makes the API will not be the one implementing it in every scenario. APIs are often reused in different parts of the code or just used by another developer. Or, they are given out like a Google Maps API so that other devs all over the world can use it.

// GET: api/ToDoList

public IEnumerable<ToDoItem> GetTodoItems()

{

return \_manager.GetTodoItems();

}

1. Let’s add a Get method for Id. It is important to name it with the singular “Item”. We want to specify in the naming that it is “ById” in case we need to get them by some other property in another API method later down the line.

// GET: api/ToDoList/5

public ToDoItem GetTodoItemById(int id)

{

return \_manager.GetTodoItemById(id);

}

1. Let’s add an Insert / Post method. Add this [HttpPost] attribute so you can rename this method to UpdateToDoList. If you do not add this, then you have to name it “Post” by Swagger’s convention, otherwise it will not work. It is good practice to add the corresponding attribute to all of your APIs.

// POST: api/ToDoList

[HttpPost]

public void UpdateToDoList(ToDoItem todo)

{

\_manager.InsertTodoItem(todo);

}

1. Let’s add an Edit / Put method. We are only able to edit the description in this case so we specify in the name what we are editing. Often if you are editing a whole object, you would just say something like “EditToDoItem”.

// PUT: api/ToDoList

[HttpPut]

public void EditToDoItemDescription(ToDoItem todo)

{

\_manager.EditDescription(todo);

}

1. Finally, we will add a Delete method.

// DELETE: api/ToDoList/5

public void DeleteToDoItemById(int id)

{

\_manager.DeleteById(id);

}

1. Add the corresponding attributes to all of the APIs that do not have one.
2. Your full ToDoListController.cs file should look like the following:

public class ToDoListController : ApiController

{

ToDoListManager \_manager = new ToDoListManager();

// GET: api/ToDoList

[HttpGet]

public IEnumerable<ToDoItem> GetTodoItems()

{

return \_manager.GetTodoItems();

}

// GET: api/ToDoList/5

[HttpGet]

public ToDoItem GetTodoItemById(int id)

{

return \_manager.GetTodoItemById(id);

}

// POST: api/ToDoList

[HttpPost]

public void UpdateToDoList(ToDoItem todo)

{

\_manager.InsertTodoItem(todo);

}

// PUT: api/ToDoList

[HttpPut]

public void Put(ToDoItem todo)

{

\_manager.EditDescription(todo);

}

// DELETE: api/ToDoList/5

[HttpDelete]

public void Delete(int id)

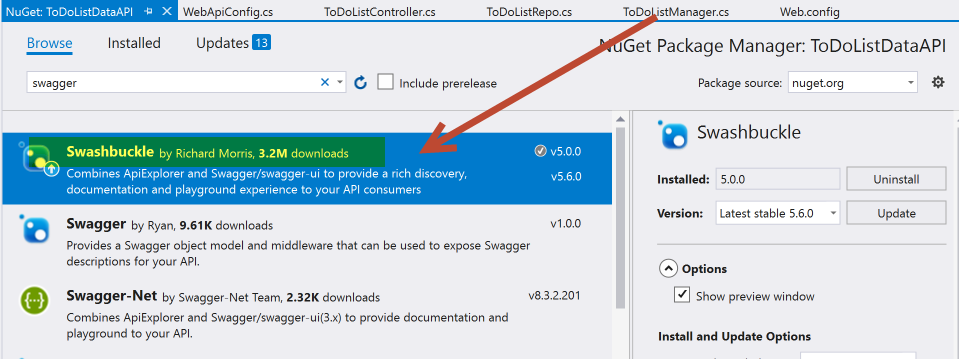
{

\_manager.DeleteById(id);

}

}

1. Now right click on references and select Manage NuGet packages.
2. Under Browse, search for Swagger or Swashbuckle, and you will see Swashbuckle by Richard Morris.



This is the NuGet package that you should download into your future projects to use Swagger. Note that it is already installed into this one, so no action is required. For your reference, here is the difference between the two packages:

**What is Swagger?**

**Swagger** is a specification on documentation an API. As I am sure we all know API documentation tends to get out of date fast and a lot of times is a low priority. Swagger aims to help solve that problem using a format that is both human and machine readable which can be maintained in either JSON or YAML and can be auto generated using a tool like Swashbuckle. Check out this post by the Swagger team for the full introduction.

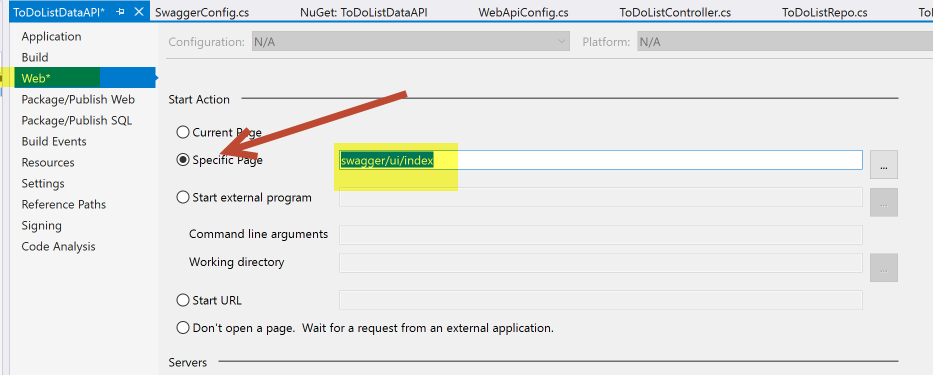
**What is Swashbuckle?**

**Swashbuckle** provides auto generation of Swagger 2.0, swagger-ui integration, etc. The project takes all the pain out of getting going with Swagger as well as providing tools and hooks for using and customizing Swagger related items. The full description can be found here.

Reference: <https://elanderson.net/2016/12/swagger-and-swashbuckle-with-asp-net-core-api/>

1. Go into the App\_Start folder again, but this time click on the SwaggerConfig.cs file. If you read through the other options that are commented out you can understand additional options you can configure here.
2. Go to line 176, notice the .EnableSwaggerUi, this is what enables you to add “/swagger” to the end of your URL and see a Swagger UI that lets you manipulate the APIs and the data.
3. Open the project properties by right clicking on your ToDoListDataAPI project and hitting Properties.
4. Click on the Web tab, select Specific Page, and add the following:

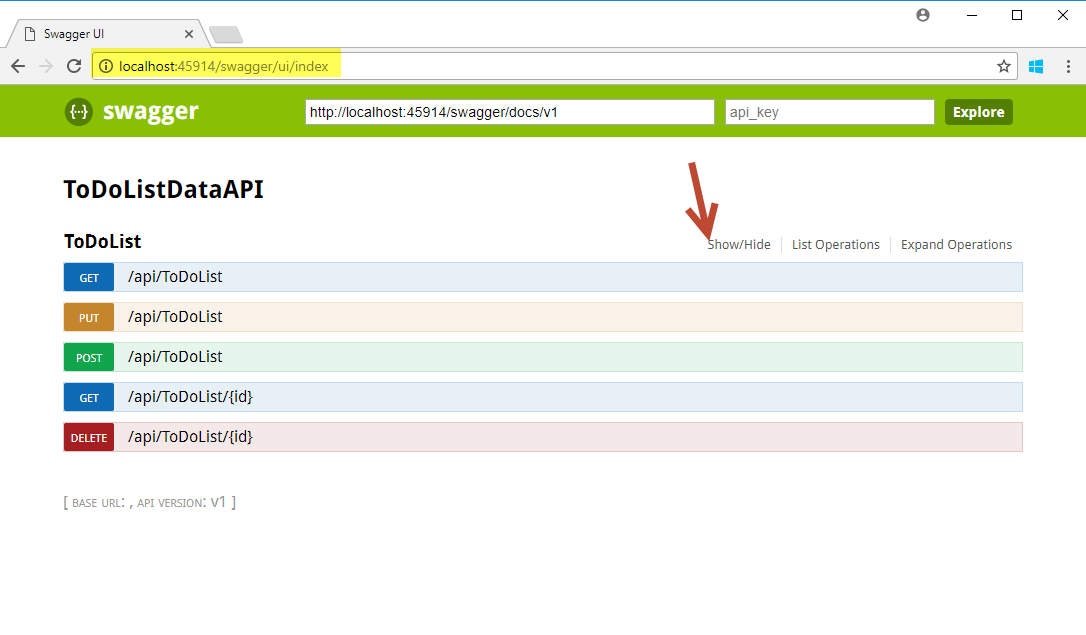
swagger/ui/index



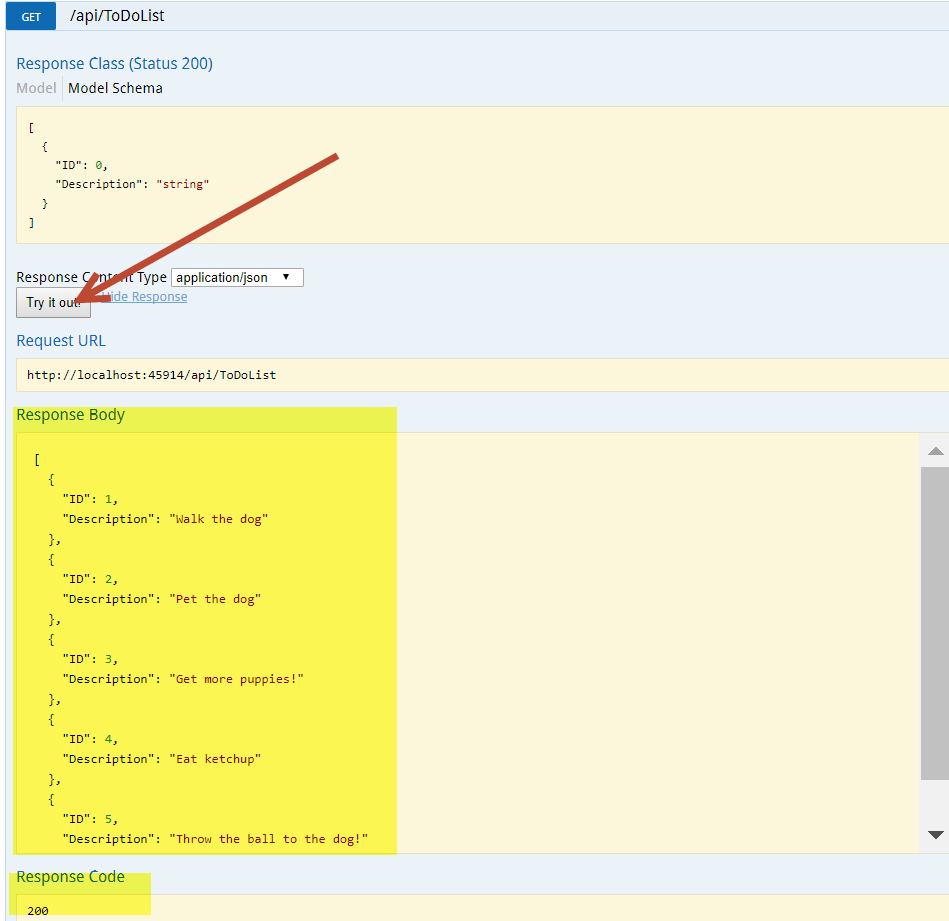
## Task: Run Swagger

In this task you will create a new Console application.

1. Run the project.
2. Add "/swagger" to the end of your URL if it is not already there (it should be though since we set it in the project settings a few steps ago), and you should see a page like this (Click on Show/Hide where the red arrow is to expand and see all your APIs):

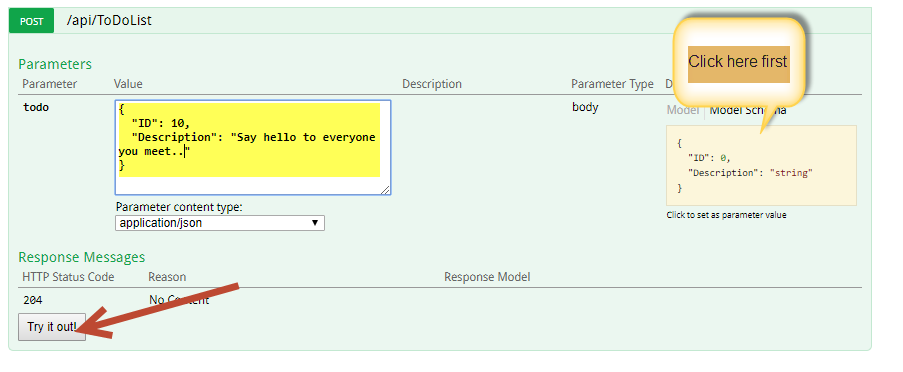


1. Try to run a GET all which is the first API on the page /api/ToDoList, you should see:

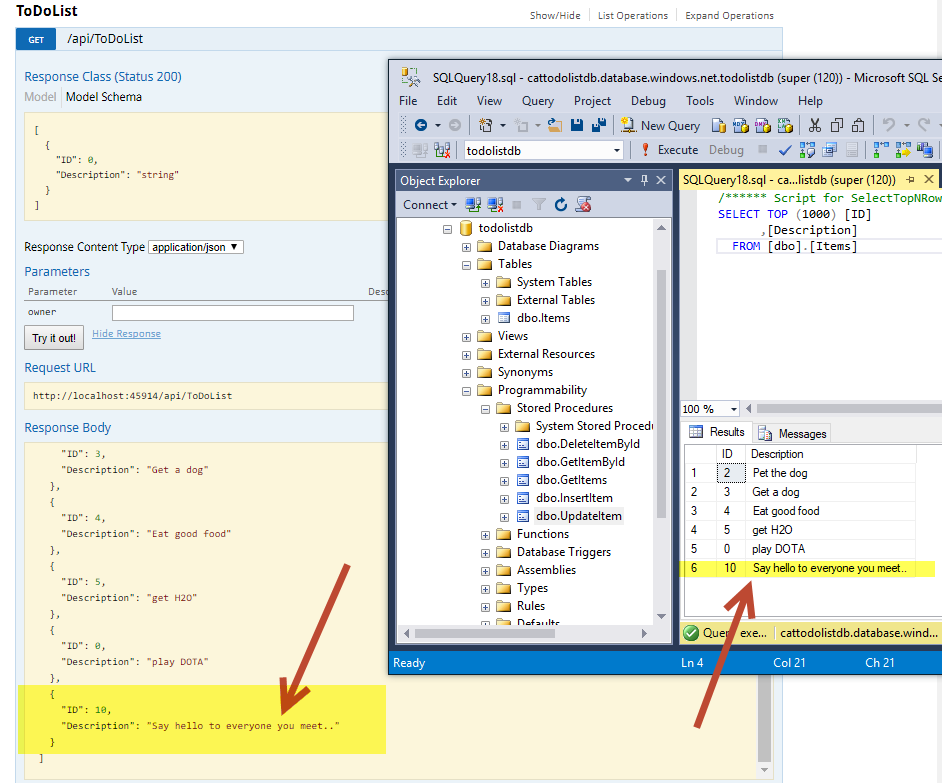


Note here you are checking that the response body should be a JSON representation of what is in your local database.

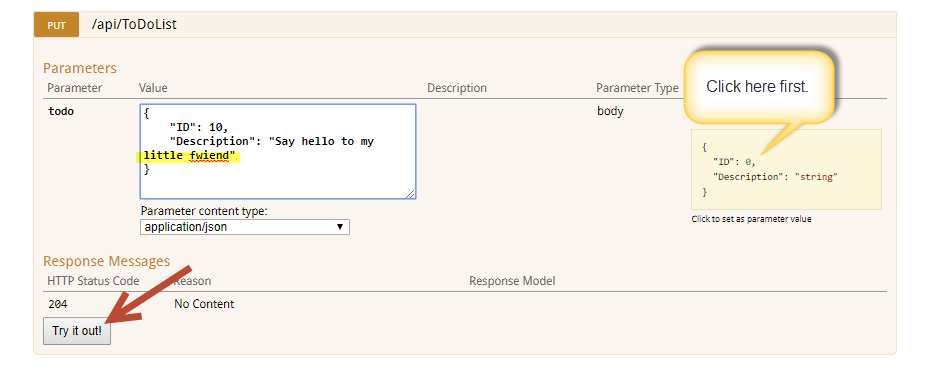
1. Try to run a POST, first click where the screenshot shows, and fill in an ID with 10 (or any number higher than your maximum ID pulled from the last step) and any description you want.



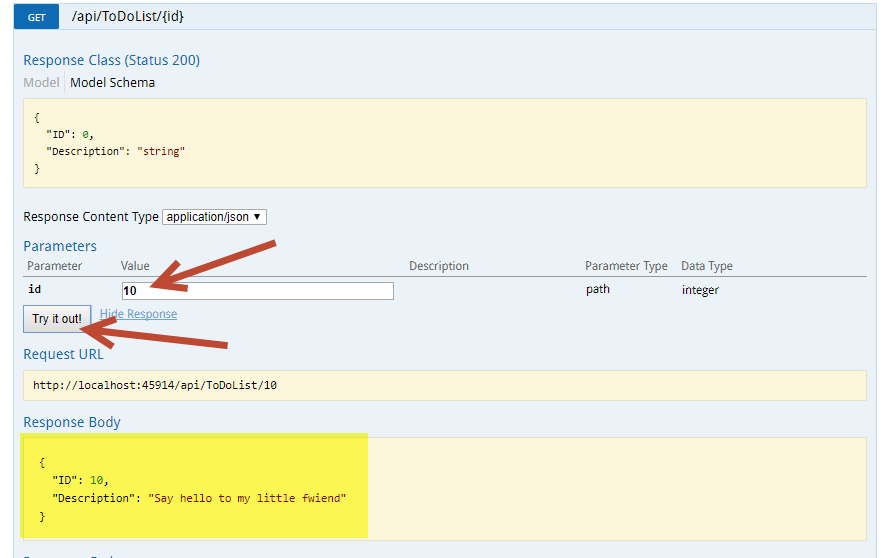
1. Try to run a GET all again, you should see your added value.



1. You should also check in SSMS and refresh your table to ensure you can see it in your database table.
2. Try to run a PUT, again click to get the format from where it's shown in the screenshot and modify an existing record's description, you can use 10 as was in the example or any number of an existing entry.



1. Try to run a GET by ID, use 10 (or whichever ID you just modified):



1. Run another Get All to make sure they all exist.
2. Try running a delete, and removing one record.
3. Run a Get All and ensure it has been deleted.
4. Go to your SSMS and check your database table has been modified and reflects your changes.