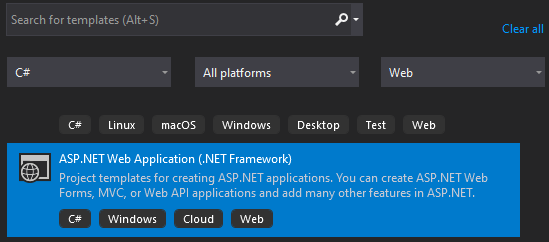
**ASP.NET Web API Tutorial (with Oauth).**

**Xamarin App:** [**GitHub - tshego3/EmployeeManagementXamarinFormsApp: Xamarin Forms App - Employee Management, consuming RESTful API with OAuth.**](https://github.com/tshego3/EmployeeManagementXamarinFormsApp)

**API:** [**GitHub - tshego3/EmployeeManagementAPI: ASP.NET - Employee Management API with OAuth.**](https://github.com/tshego3/EmployeeManagementAPI)

**YouTube:** [**https://youtube.com/playlist?list=PLpbcUe4chE78YvgIMtmgNEmRGyGJcsQdF**](https://youtube.com/playlist?list=PLpbcUe4chE78YvgIMtmgNEmRGyGJcsQdF)

Create a new ASP.NET Web Application with .NET Framework.



Select the “Web API” template, change the “Authentication” to “Individual User Accounts” and configure for HTTPS, then press “Okay”.

1. Add the ConnectionString of the Database in the “Web.Config”.

2. Add this code in the “/App\_Start/WebApiConfig.cs”:

using System.Net.Http;

using System.Net.Http.Headers;

And this in the “public static void Register(HttpConfiguration config)”:

config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));

4. Run API

5. Add users (go to "RegisterBindingModel" to see the requirements) and open Postman, then do the following:

<https://localhost:44339/api/Account/Register>

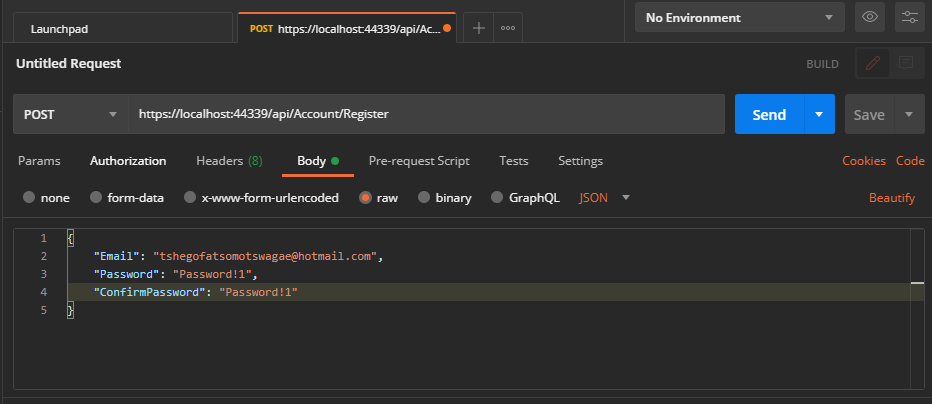
{

"Email": "",

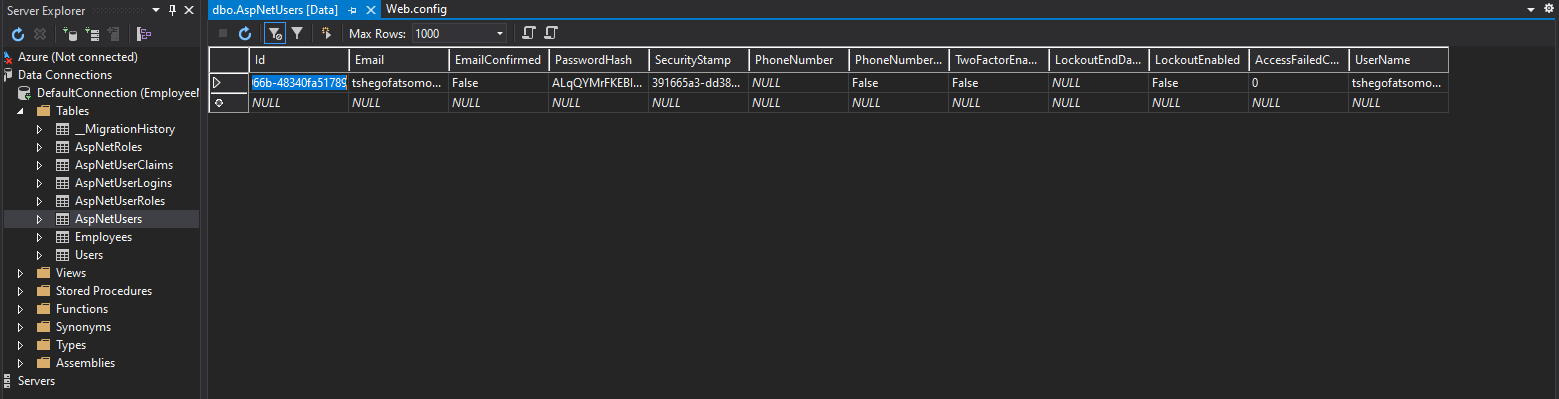
"Password": "",

"ConfirmPassword": ""

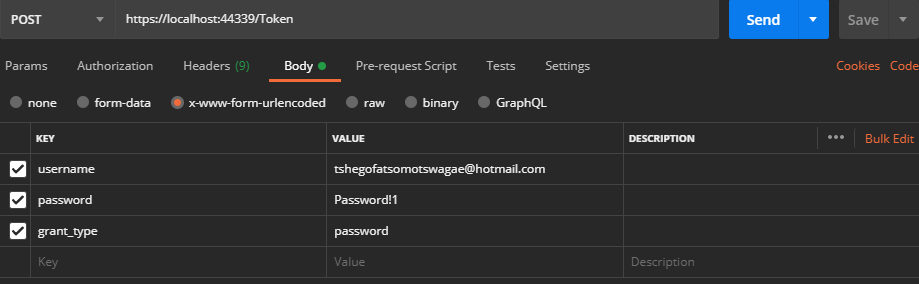
}

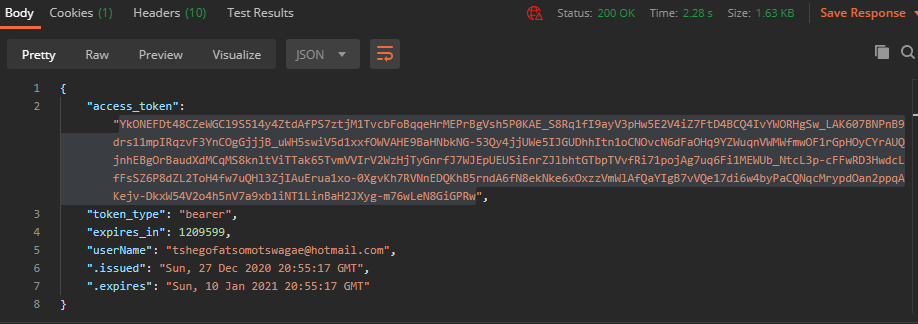


The following account would be created in the AspNetUser datable and along with the other datables as shown.

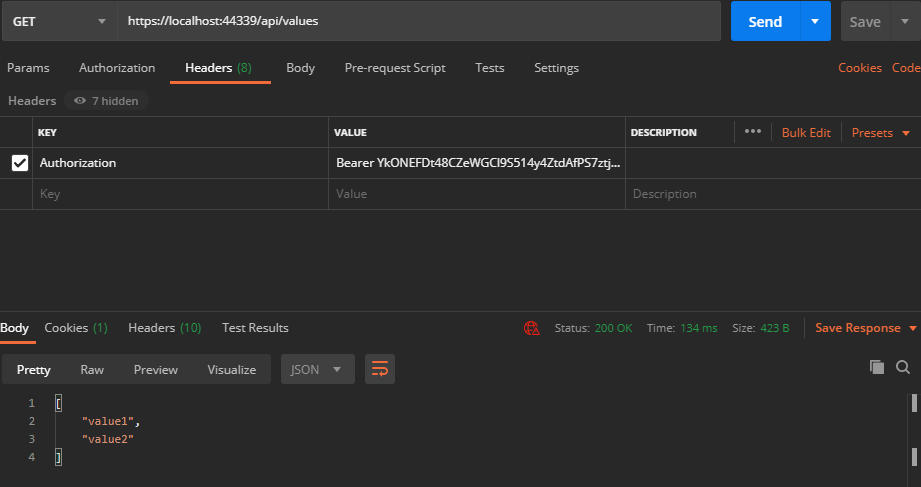


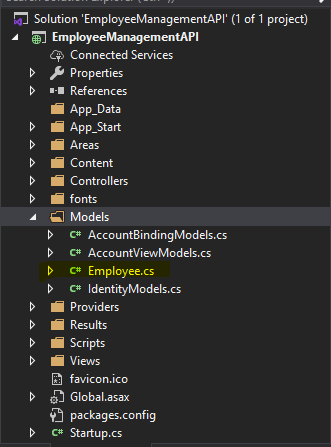
6. Run this POST HTTP request to get the access token (“grant\_type” value never changes).



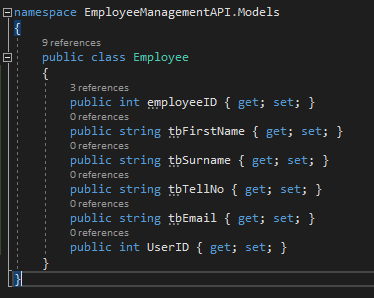


7. Run this HTTP request to test the access token.

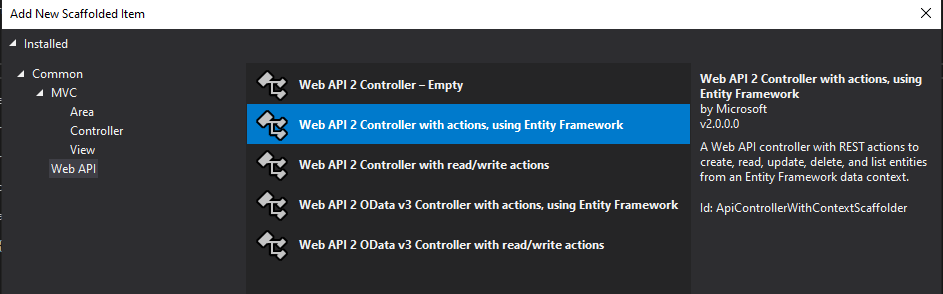


8. Add a new class in the “Models” folder.

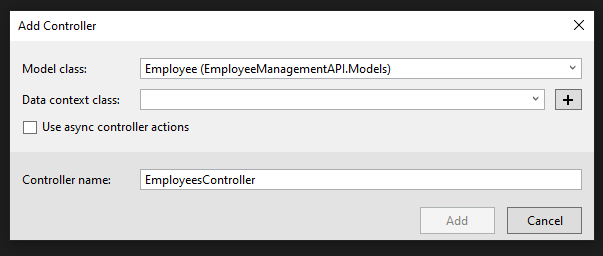
Add the following code, the build the solution. Make sure each column matches to its database data-type.



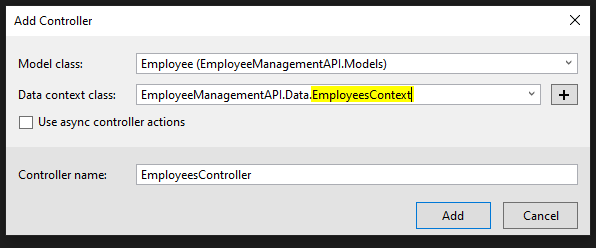
Add a new a controller (Web API 2 Controller with actions, using Entity Framework) in the “Controllers” folder.



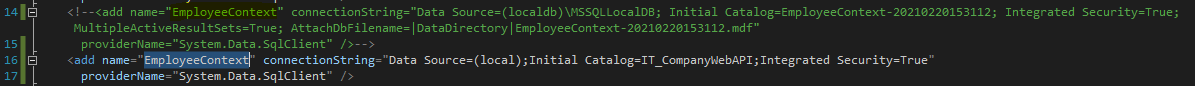
When this pop-up displays, choose the Employee model (which is the model that was just created) on “Model class:”.



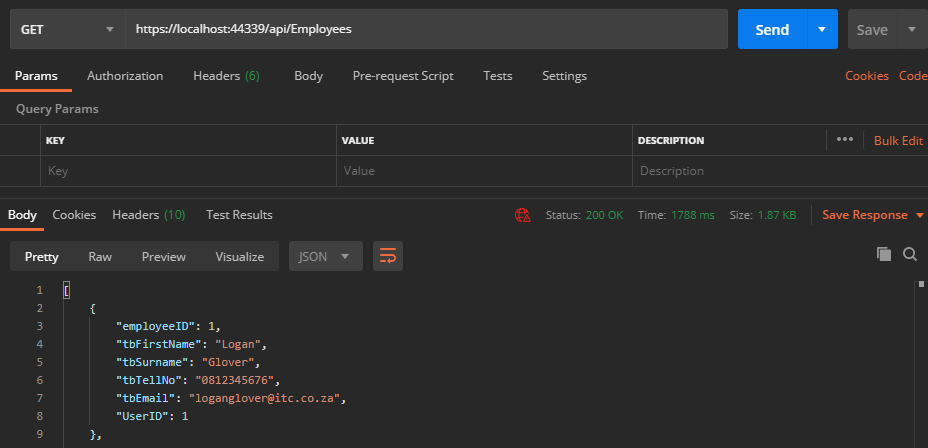
Then click on the plus sysmbol button to add a new “Data context class”, it will display another pop-up just rename it after the “EmployeeManagementAPI.Data.” to have a similar name to the model previously created. Then press add.



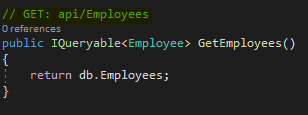
9. Testing the Employees Model API, but before doing so comment the connection string in the “Web.config” file to the following the connection name “EmployeeContext”. If its not already added, then add the connection string so it does not display the empty data.



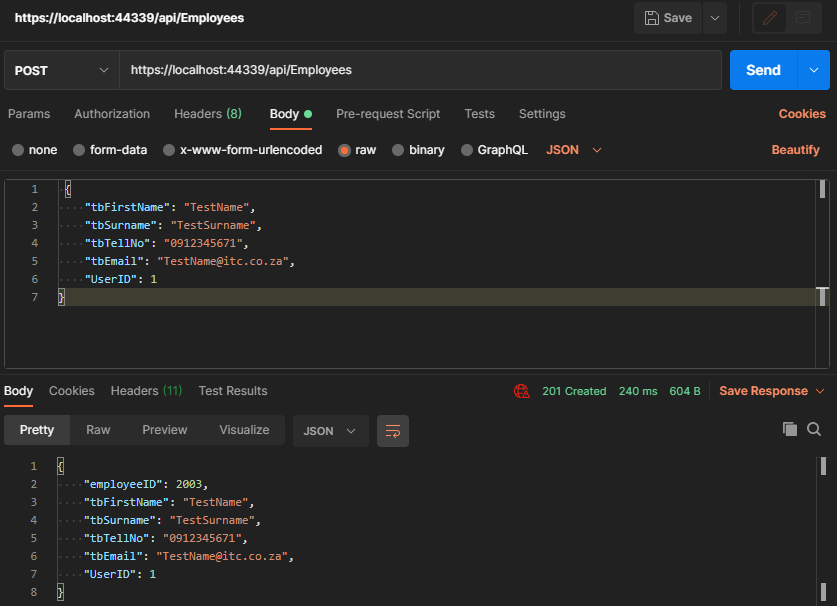
Then run the following.



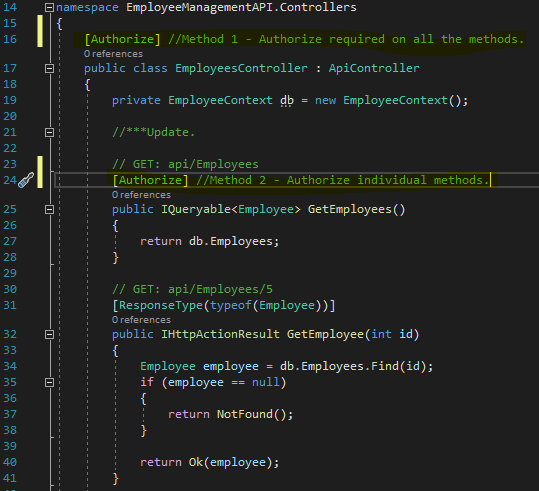
This URL “<https://localhost:44339/api/Employees>” can be found in the “\Controllers\EmployeesController.cs”.



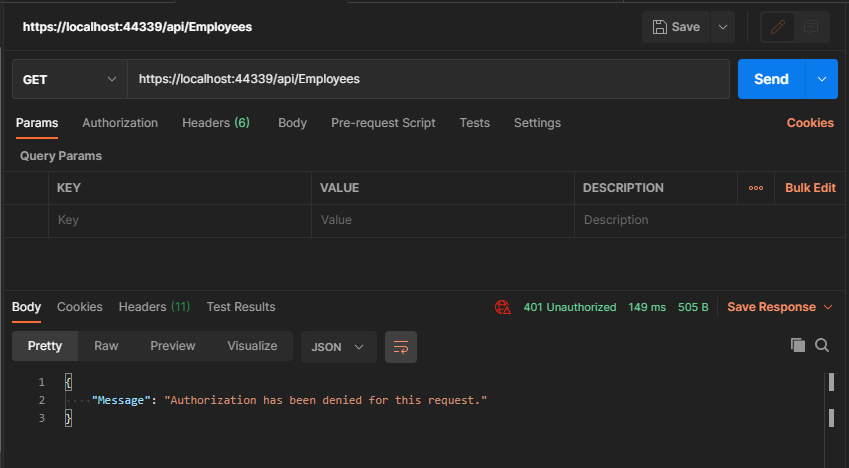
10. Test API by adding a new row.



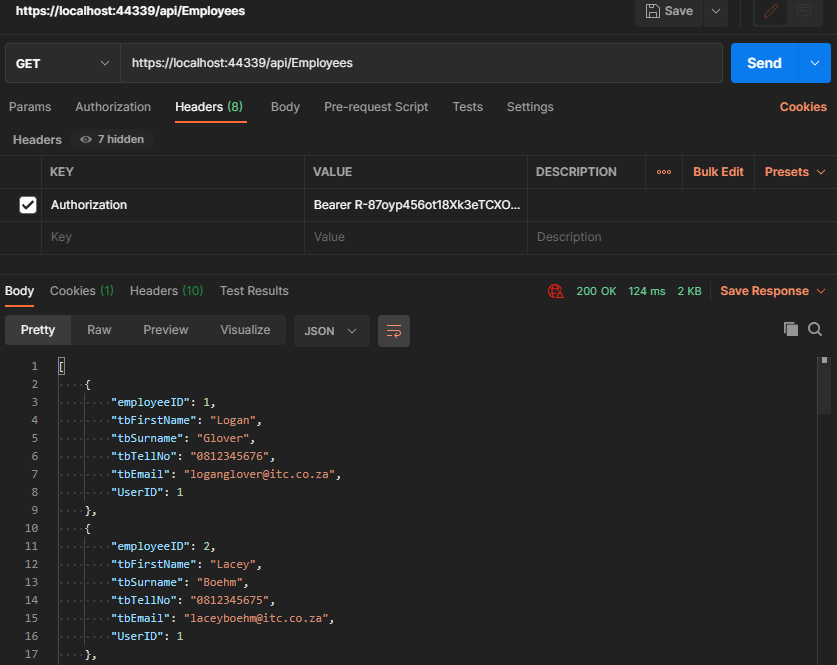
11. Make the API controller(s) require authorization before data can be viewed, edit in the “\Controllers\EmployeesController.cs”.



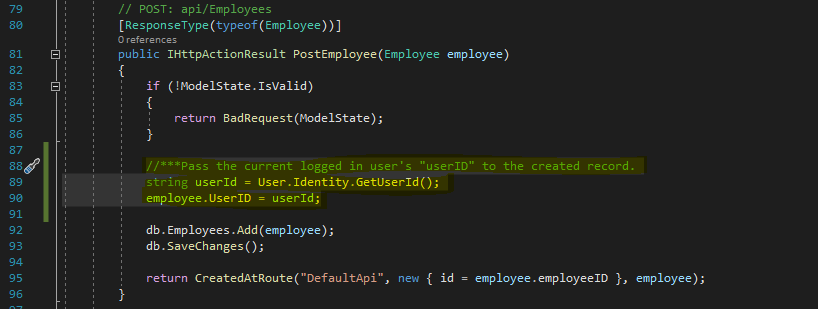
Result of accessing a method without authorization.



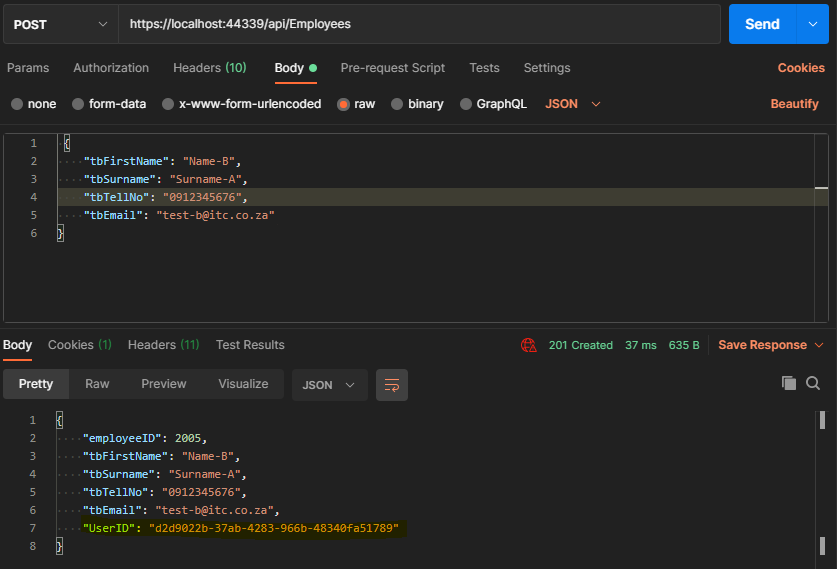
Result of accessing a method with authorization (To get the Bearer token, follow **Step 6 & 7** and copy it into the Headers, with “Bearer” then the token).



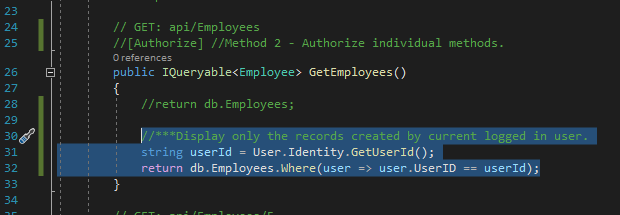
12. Add the following code to pass the current logged in user's "userID" to the created record, edit in the “\Controllers\EmployeesController.cs”.



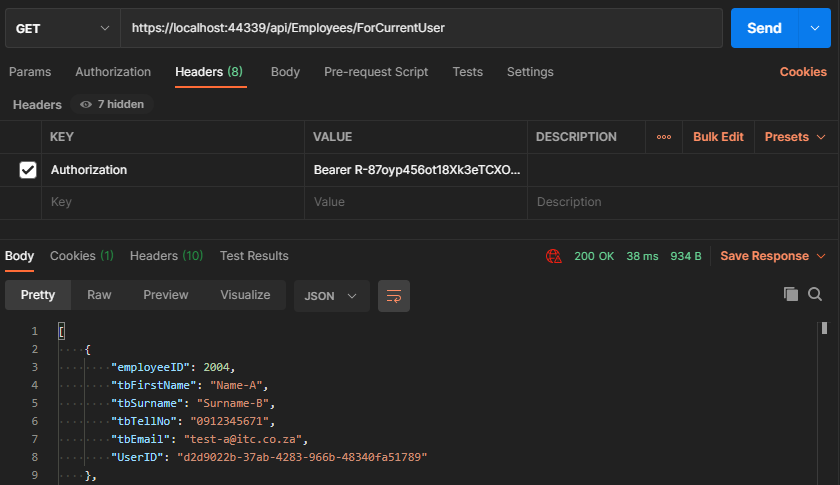
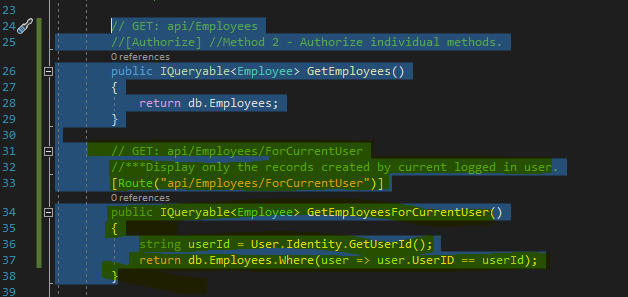
Result of the current user’s User-ID passed to the created record (with authorization).



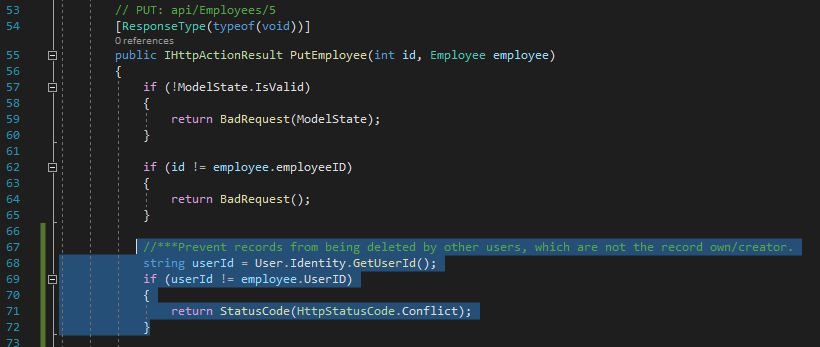
13. Display only the records created by current logged in user, edit in the “\Controllers\EmployeesController.cs” (with authorization).



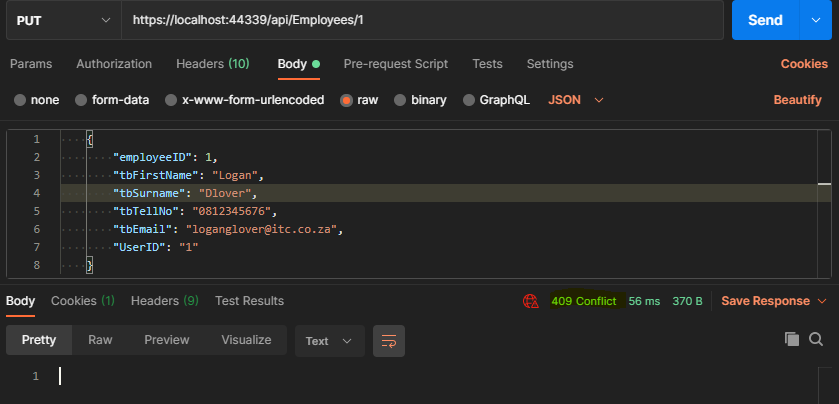
To display all records of the current logged in user or only the records the current user made create a duplicate of the code and revert the first one as it was originally, then on the second add a re-route so that the API url-link does not conflict.

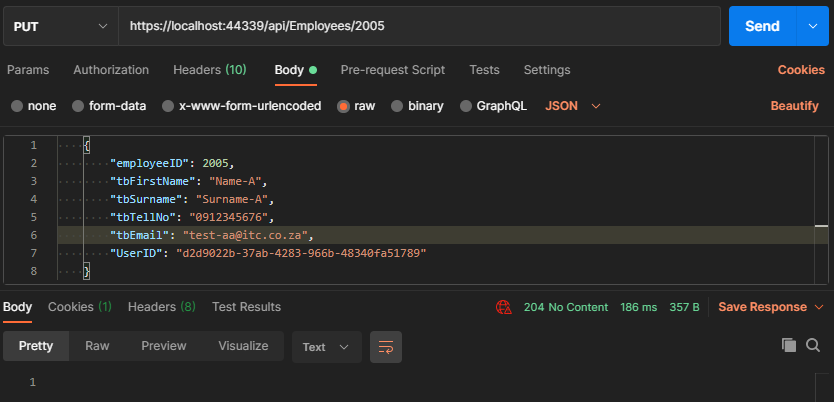
14. Add the following code in the “\Controllers\EmployeesController.cs”, to prevent records from being edited by other users, which are not the record owner/creator (with authorization).



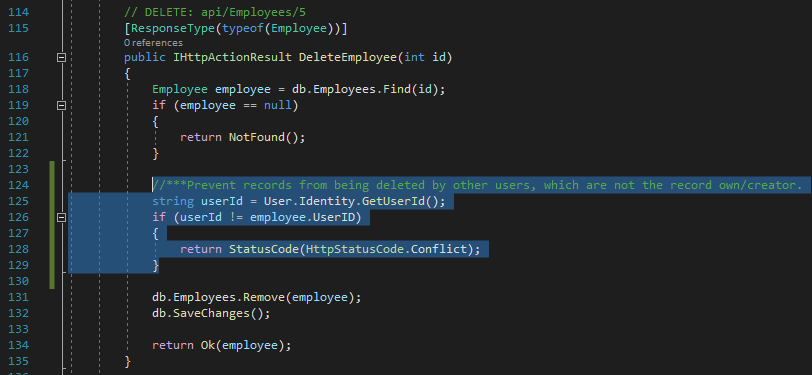
Result of a user trying to edit a record that is not theirs. **HTTPS Status 409-Conlfilct**.



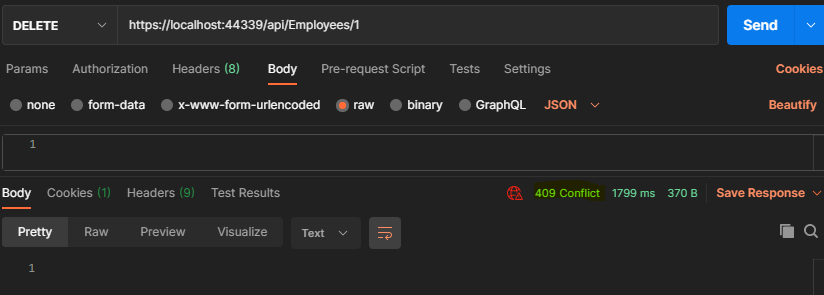
Result of a user trying to edit a record that is theirs. **HTTPS Status 204-No Content (its updated)**.



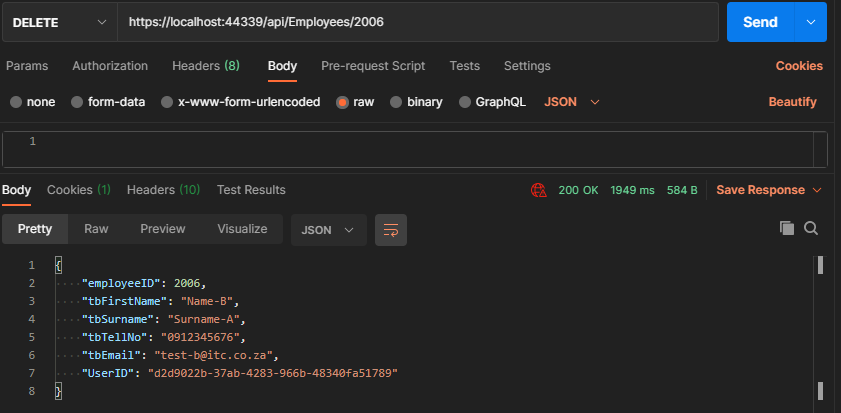
15. Add the following code in the “\Controllers\EmployeesController.cs”, to prevent records from being deleted by other users, which are not the record owner/creator (with authorization).



Result of a user trying to edit a record that is not theirs. **HTTPS Status 409-Conlfilct**.



Result of a user trying to edit a record that is theirs. **HTTPS Status 200 (its deleted)**.



16. Add the following code in the Web.Config file of the API to allow REST (enabling PUT & DELETE) on IIS Manager.

<system.webServer>

<modules>

<remove name="WebDAVModule" />

<remove name="FormsAuthentication" />

</modules>

<handlers>

<remove name="WebDAV" />

<remove name="ExtensionlessUrlHandler-Integrated-4.0" />

<remove name="OPTIONSVerbHandler" />

<remove name="TRACEVerbHandler" />

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="\*." verb="\*" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" />

</handlers>

</system.webServer>

17. Add the following code in the “\Controllers\EmployeesController.cs” to display only the records containing the "keyword".

// GET: api/Employees/Search/Peter

//\*\*\*Display only the records containing the "keyword".

[Route("api/Employees/Search/{keyword}")]

[HttpGet]

public IQueryable<Employee> SearchEmployees(string keyword)

{

return db.Employees.Where(employee => employee.tbFirstName.Contains(keyword) || employee.tbSurname.Contains(keyword));

}