# Running the code –

* The solution has 3 projects –
  + Cylance.UnitTests
  + CylanceClient (console application)
  + CylanceGUID (API)
* The project has been configured so that when you click “Start”, both the API and the console application will start up at the same time.
* The API is configured to run on port 44361 (this can be changed from the launchsettings.json file.
* The application uses Redis as the cache, and that must be configured by running the Redis server on port 6379. I used a standalone version of the Redis server (version 3.2.1). I have never worked on this before, so my knowledge of configuring and running this is limited.
* Once you click Start, the API starts running on IIS Express, and you can directly hit the API using endpoints as such –

<https://localhost:44361/api/Guid/25087bff-1b4d-4347-a215-ca1f441d5a56>

* You may also use the Console application to call the API’s.
  + The first parameter on the console is the Crud operation.
  + Depending on the input, the user will be prompted for the Guid, Expire field and User fields.
  + To skip entering any field, simply press the Enter key, and it will move on to the next element. For example if you doing a POST and don’t want to enter a GUID, press Enter when the program prompts you for a GUID.
  + After the last field is entered for the CRUD operation specified, the system will automatically make a call to the API using those parameters and display the result once the API has responded.
  + If an invalid value is entered (For example you enter “ABC” as a Crud operation, the console will tell you that the value is invalid and the user needs to Stop and Start the Visual Studio project again at that point. I have not handled the scenario in which the user is allowed to re-run.
* The solution stores the data using SQL Server. This can be accessed using the Sql Server Object Explorer. There is a Data folder in the solution folder. The DB name is “Cylance” and the table name is “GuidModel”.
* You many also hit the API using tools like Postman if you do not wish to use the console application.
* The API takes a json input of the model that was specified in the requirements.

# About the Solution –

# CylanceGUID (API)

* The entry point is in the Controllers class. Depending on the route, the controller calls the Get/Put/Post/Delete methods.
* The control is then passed to the manager class along with the inputs that were received. This is the main business logic class.
* For Get requests, the manager uses the cache to search for existing values and return those. If nothing is found, it searches the DB and adds the result to the cache for future Get calls.
* For Update, we update the value in the DB, and then delete the value from the cache (if present) and add it to the cache.
* For Post, we save the value to the DB.
* For Delete, we delete the value from the DB as well as from the cache.
* There are 2 different models here – first is the GuidAPIModel which is the model in which input requests are received, and the second is the GuidDataModel, which is the model in which the data is stored in the DB or the cache. Although for this project, we can make do with one model, I created it to emphasize the importance of separating business (API) models and data models.
* For exception handling, I have a separate class. Right now, it handles 3 different types of custom exceptions, but this can be extended to use as many more custom exceptions as we need.
* Constants are kept in a separate file, so if some text needs to be changed, we need to look and change it only in one place. This can also be done with a resource file (ideal for internationalization, but serves the purpose of a constants file as well).

# CylanceClient (Console Application)

* The entry point is the Main static function in the Program.cs file. I read the inputs from the command prompt and based on the inputs request the user for other information. For example, the Get and delete operation require only the Guid and no other information.
* All the API requests are handled in the HttpClients.cs file. From there I use httpclient to call the API passing the required fields and displaying the response on the command prompt once a response is received.
* I have used a separate model for this project as well to emphasize the need for a separate Client model (along with the Business and Data models explained above).
* Please note – I have not tried to make the Console application too flowery, because I hope that the intent was to see how to call the API and not how the user experience for using the console application is. If the users tries to break the application, it will break 😊. I have tried to handle most of the basic error conditions here.

# Cylance.UnitTests (xUnit)

* I have some basic units tests in place which check for common valid values, common invalid requests and also tests for some custom exceptions. There is scope to increase the code coverage of the unit tests by handling other less probable scenarios.

Please feel free to reach out to me if there are any questions/concerns.