**API**

**What is an API? (Application Programming Interface)**

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you’re using an API.

**How Does A REST REQUEST WORK**

Before diving into API testing, you’ll need to learn a few of the basics. An API is made up of a set of REST requests. These are the requests made to an application server that retrieve, delete, or manipulate data in an application’s database

A REST request is made up of the following parts:

An HTTP verb that describes what action should be taken

A Uniform Resource Locator (URL) that defines the location of the request

HTTP headers that provide information to the server about the request

A request body that provides further details for the request (this can sometimes be empty)

Here are the most common HTTP verbs:

**A GET** request fetches a record from a database

**A POST** request adds a new record to a database

**A PUT** request replaces a record with a new one

**A PATCH** request replaces part of a record with new information

**A DELETE** request removes a record from a database

The URL used in the request clarifies which type of record will be altered by the request. For example, a GET request used in conjunction with the URL <https://www.example.com/cars/1> would return record number 1 in a table of cars.

Similarly, a GET request used in conjunction with the URL <https://www.example.com/trees/2> would return the second record in a table of trees. The end of the URL specifies an endpoint—a data object used in the API. In these examples, the endpoints would be /cars and /trees.

**HTTP headers can provide information to the server such as:**

**The Host:** the domain and port number of the user making the request

**Authorization:** the credentials of the user making the request

**The Content-Type:** the format of the information provided in the body of the request

The request body is used when making POST, PUT, or PATCH requests. The body specifies exactly what information should be added to the database. It is usually in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format.

Here is an example of what a JSON request body might look like for doing a POST request that adds a customer to a database:

{

“firstName”: “John”,

“lastName”: “Smith”,

“emailAddress”: “[jsmith@example.com](mailto:jsmith@example.com)”

}

**What Is Included in the Response to a REST Request?**

The response to a REST request is the information that the server sends back after it has received and processed the request. It will include

HTTP headers that describe the response,

a response code that describes the success or failure of the response, and

a response body that includes requested or relevant information (this can sometimes be empty).

The HTTP headers in the response provide information to the requesting party such as:

**• Access-Control headers:** tell the requester what types of requests and headers will be allowed

**• The Content-Type:** the format of the information returned in the response

**• The Server:** the name of the server that responded to the request

Response codes are three-digit codes used to describe the result of the REST request. The most common response codes come in one of these three categories:

• 200-level responses indicate that the request was received, understood, and processed

• 400-level responses indicate that the request was received, but that there was an error from the client

• 500-level responses indicate that there was some sort of server error

HTTP Status codes

Code Title Description

200 OK The request was successful.

201 Created The resource was successfully created.

202 Async created The resource was asynchronously created

400 Bad request Bad request

401 Unauthorized Your API key is invalid.

402 Over quota Over plan quota on this endpoint.

404 Not found The resource does not exist.

422 Validation error A validation error occurred.

50X Internal Server Error

The next step is to set up each of these requests in an API test tool.

A variety of tools can be used for this purpose. It is also possible to write API tests directly into code, but the advantage of API testing tools is that they are easy to use and provide a way to visualize the response.

**Which tool do we use**

The easiest API testing tool to use is Postman. It has a 100 percent free version that can be downloaded quickly, and there are also paid versions for teams. Once you have chosen an API testing tool, the first requests to set up are the “happy path” requests. These are the requests that the API developer would expect that users would make in the normal course of using the application.

When setting up a “happy path”-style request, it is important to include assertions. One assertion should be that the correct response code is returned (often a 200 response).

If the response includes a body, there should be an assertion on that as well. For example, if a GET request is being tested, there should be an assertion that the body of the response contains the data that was expected with that record.

Once all of the “happy path” tests have been created, negative tests can be added. Negative tests are those that make sure that any kind of error is handled correctly. It’s important that an application doesn’t crash when a user accidentally imports invalid data, and it’s important that a malicious user is prohibited from entering harmful scripts into the database. Here are some examples of negative tests:

\*Sending a request with the wrong HTTP verb

\*Sending a request with the wrong endpoint

\*Sending a request with the wrong headers

\*Sending a request with missing headers

\*Sending a request without the proper authorization

\*Requesting data for a record that does not exist

\*Sending a request with a body that has missing required fields

\*Sending a request with a body that has invalid field values

**How Can We Automate API Testing?**

Once a complete suite of positive and negative tests has been created, automation can be set up.

• Open Visual Studio

• Create a new folder

• Name the folder

• Add your class-Unit test

• You can rename your Unit test

• Go to postman

• Click on code

• Select C#(Rightsharp)

• Copy the code

• Paste into you visual studio

• Add nugget package-search for (Restsharp)

• Run the test