REST Workshop
Lab 1

1 Lab setup

Make sure you have the following items installed or available:

- A free account at Postman and installed the Postman app
- A free account on RapidAPI and a valid credit card to register during use
- PowerShell Core 7 (if you are using Windows for this workshop)
- A suitable text editor (Notepad ++)

2 Using a browser as a REST client

There are a few sites on the Web that provide fake REST APIs for purposes of testing and prototyping. A well-known one is:

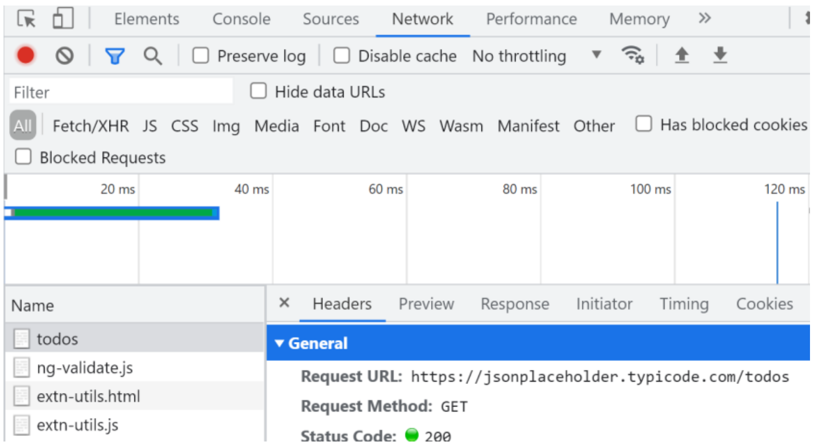
https://jsonplaceholder.typicode.com/

Open a browser tab at this URL and scroll down to the end where you see the subtopics Resources and Routes. Click on any of the links highlighted in green. This sends out a HTTP GET request from the browser to the specified API endpoint whose URL you will be able to see in the address bar. Notice that the complete URL for a given endpoint is the server name (https://jsonplaceholder.typicode.com) appended with the path portion (`/posts`) for that endpoint.

You can verify that the returned content is in the form of JSON.

If you wish to inspect the Headers in the request and response messages, and Response in detail, switch to the Network view tab in the Developer Tools when clicking on the URL (For Chrome: More

Tools -> Developer Tools to access this, a similar tab view exists for other major browser Developer Tool environments).



todos
ng-validate.js
extn-utils.html
extn-utils.js

Request URL: https://jsonplaceholder.typicode.com/todos
Request Method: GET
Status Code: 200

The Preview view shows the returned JSON formatted in a more structured form.

Notice that the links for POST, PUT, PATCH and DELETE in the ROUTES section are not highlighted because you are not able to activate any of these HTTP methods: since clicking on a browser link simply sends a HTTP GET request.

For each of the API endpoints for the 6 common resources, you can append a number to the end preceded by a forward slash to retrieve a single item from the collection of resources, for e.g.

https://jsonplaceholder.typicode.com/posts/5

https://jsonplaceholder.typicode.com/comments/2

https://jsonplaceholder.typicode.com/photos/6

This number will typically, for the majority of REST APIs, represent a unique ID for the resource in question. Make sure that the number that you append at the end does not exceed the number listed for that particular resource (e.g. 100 posts, 500 comments, etc).

You can also use query parameters (the portion after the ? at the end of the URL path) in your request to drill down to a single item or subset of items from the collection of resources available based on an id for the item, for e.g:

https://jsonplaceholder.typicode.com/comments?postId=5

https://jsonplaceholder.typicode.com/comments?postId=2&id=7

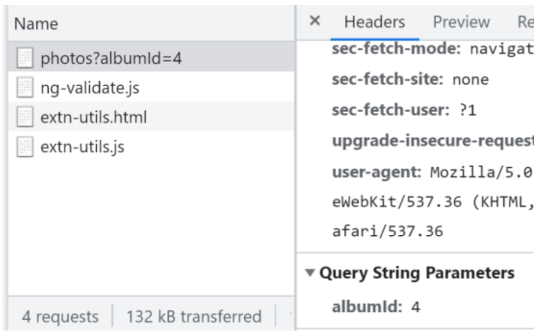
https://jsonplaceholder.typicode.com/comments?id=10

https://jsonplaceholder.typicode.com/posts?id=5

https://jsonplaceholder.typicode.com/posts?userId=3&id=23

Be careful with case as the path components of the URL after the domain name (jsonplaceholder.typicode.com) are case-sensitive: so `postid=5` is not the same as `postId=5`

If you check the Network view in the Developer Tools, you will see the query string parameters shown at the bottom, for e.g.



```
sec-fetch-mode: navigat
sec-fetch-site: none
sec-fetch-user: ?1
upgrade-insecure-request
user-agent: Mozilla/5.0
eWebKit/537.36 (KHTML,
afari/537.36
▼ Query String Parameters
albumId: 4
```

You can use any of these online JSON validators to verify that the returned response is proper JSON: just copy and paste the responses into the validator views.

https://jsonlint.com/
https://jsonformatter.org/
https://codebeautify.org/jsonvalidator

Experiment with introducing deliberate errors in the JSON formatting in these validators and see how they flag these errors. The second site (https://jsonformatter.org/) also provides options for converting the JSON to its equivalent representation in other related formats such as XML and YAML, as well

In order to test out the other REST HTTP methods such as POST, PUT, etc, you will need to run the JavaScript code to send out these methods. Go to this link:

https://jsonplaceholder.typicode.com/guide/

Switch to the Console view in the Developer Tools and copy one of the Javascript snippets and paste it into the Console and press enter. You should be able to see the returned response in the Console view. In the Network view, you will also be able to see the request details (headers and content). Select to filter based on Fetch/XHR to be able to locate the relevant request more quickly.

In a real functional REST API, requests such as PUT, POST, PATCH should modify or add to the content of the resources stored on the server side. Then, subsequent GET requests should be able to retrieve this modified or newly added content. However, for this fake API, no such modification is performed (due to the heavy traffic load from multiple users testing out this API), so subsequent GET requests that you issue will still show identical content.

Scroll down to the bottom of the page to see the nested routes that are available:

For e.g.

```
/posts/1/comments
```

would mean to retrieve all the comments related to the post with the id of 1

```
/users/1/photos
```

would mean to retrieve all the photos related to the user with the id of 1

This nesting relationship between resources would be reflective of the Entity Relationship Diagram in the relational table that is used to store the records.

If you return to the main page:

https://jsonplaceholder.typicode.com/

you will see that there is fixed number on the 6 common resources (for e.g. 100 posts, 500 comments and so on). If you attempt to access a resource ID beyond the range for the specified resource, you are effectively attempting to access a non-existent resource. Try this now by typing this URL into a browser tab:

https://jsonplaceholder.typicode.com/posts/2000

In the Network view of the Developer Tools, you should an entry with status 404 outlined in red. Responses outlined in red indicate either a client-side error (code 4xx) or server-side error (code 5xx). In this case, there is nothing contained in the response body.

You can repeat this with other resources, e.g.

https://jsonplaceholder.typicode.com/comments/1000
In fact, any kind of URL which is not matched by the server-side API code to a resource will result in a 404 error, for e.g.

https://jsonplaceholder.typicode.com/asdfasdf/2323

3 Using Postman as a REST client

The disadvantages of using a browser as a REST client is that it is only able to issue HTTP GET requests. To issue other HTTP REST request methods, we would have to type (or copy and paste) relevant JavaScript code snippets in the Console view of the Developer tools, which is less than ideal for rapid testing.

There are many GUI-based REST clients which provide access to the full range of REST HTTP methods, as well as ability to tailor the request headers and bodies as well as manipulate and store the results. The most popular and widely used of this at the moment is Postman

Start Postman (this process might take some time if there is an automatic update in the background during start up).

We will create a new workspace for this lab. Select Workspaces form the main menu, and select + New Workspace. Providing the following details:

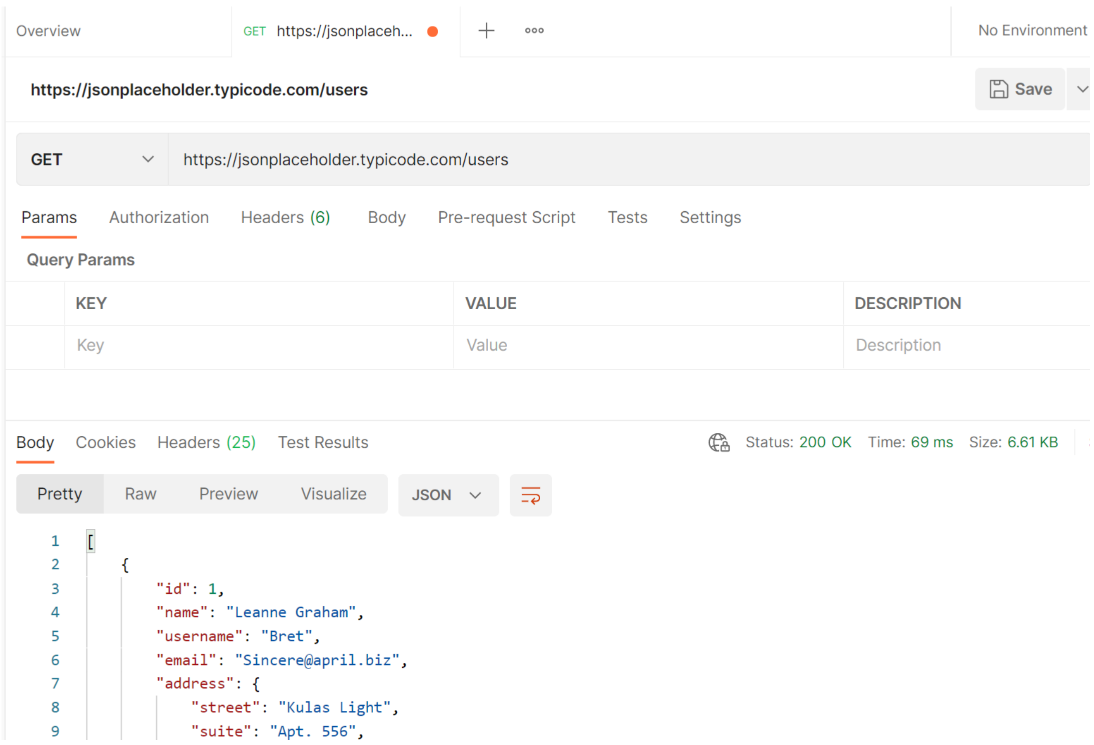
Name: `FirstLab`
Summary: `My first lab using Postman`

Then select the Create Workspace and team button at the bottom. Postman will spend a couple of minutes creating the workspace and then you can navigate to it.

To get started we will create a new request by any of these actions below:

- selecting Create a request item below the Get Started list
- clicking on the + icon
- selecting File -> New Tab from the main menu

Type in the complete URL for the API endpoint that you want to send a request to and select the method you wish to use (by default this is GET). For e.g. for
https://jsonplaceholder.typicode.com/users
then click the blue Send button



https://jsonplaceholder.typicode.com/users

GET https://jsonplaceholder.typicode.com/users

KEY	VALUE	DESCRIPTION
Key	Value	Description

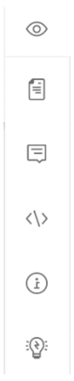
Status: 200 OK Time: 69 ms Size: 6.61 KB

```
[
    {
        "id": 1,
        "name": "Leanne Graham",
        "username": "Bret",
        "email": "Sincere@april.biz",
        "address": {
            "street": "Kulas Light",
            "suite": "Apt. 556",
```

The top part of the main view shows details regarding the outgoing HTTP request issued by Postman, for which you can customize in relevant ways: for e.g. by selecting HTTP method to be used, setting query string parameters, putting content in the body and creating new headers or modifying values for existing ones.

The bottom part of the main view show details regarding the incoming HTTP response received by Postman. This includes relevant details such as status code, headers and the body of the content (which can be viewed in a variety of ways by selecting the Pretty, Raw, Preview tabs as well as changing between JSON, XML, HTML formatting, etc).

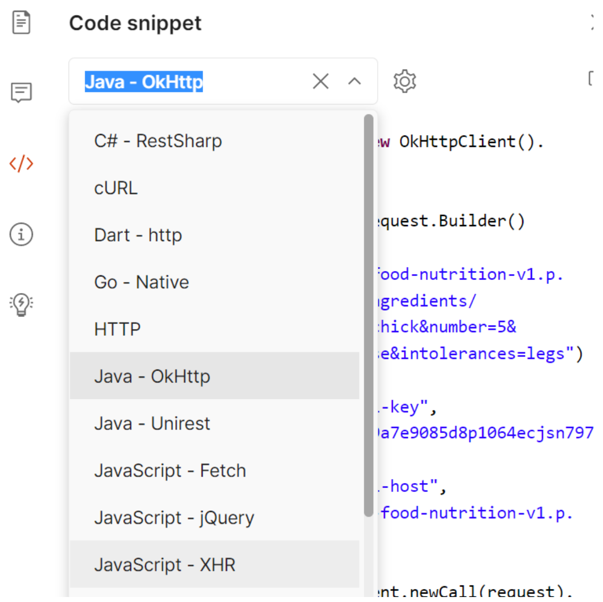
The right most pane has a series of icons:



The Documentation icon allows you to provides more information on the particular request.

The Comments icon allows you to add comments at specific portions of the request (such as the request parameters, headers, request bodies, the body, etc) to clarify their intent and help facilitate collaboration between team members.

The Code icon provides code snippets in a specific library for a given programming language in order to send out that particular REST HTTP request being configured. Click on the code icon.



```
w OkHttpClient().

quest.Builder()

ood-nutrition-v1.p.
gredients/
hick&number=5&
e&intolerances=legs")

.-key",
a7e9085d8p1064ecjsn797

.-host",
food-nutrition-v1.p.

nt.newCall(request).
```

As an example, if we are working with Java, we can see code snippets using 2 different libraries for sending out the request.

1. OkHttp - https://square.github.io/okhttp/
2. Unirest - http://kong.github.io/unirest-java/

You could then copy the relevant code snippet and place in your client app at the appropriate point in the flow logic.

The Info icon provides a short summary on the Request itself, which includes a unique ID self-generated by Postman.

The Related Collections icon allows you to discover public collections of API requests that are similar to your request on the Public API Network. We will look at this later on when we finish creating our own collection at the end of this lab session.

When you are done examining the functionality of these icons, you can return back to creating more requests in the main Postman pane.

Create two more new requests to send a GET request to the following API endpoints:

https://jsonplaceholder.typicode.com/posts

https://jsonplaceholder.typicode.com/comments/2

Examine the contents of the response.

So far we have been typing in the entire URL in the address bar. Since we are essentially using the same base URL () repeatedly for all our requests, we could store it in a temporary variable which we can then substitute in when typing the URL to reduce its length. Postman provides a facility for this.

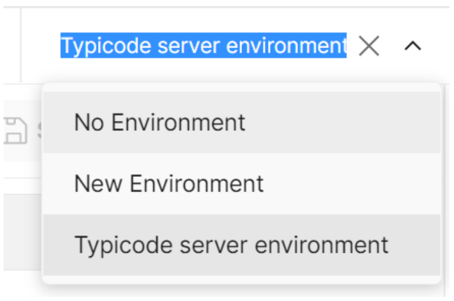
Click on the Environment quick look icon in the right hand pane. Click Add on the Environment section.

In the Environment tab, give this new environment the name: `Typicode server environment`

Add this entry
Variable: `dummyAPI`
Initial value: `https://jsonplaceholder.typicode.com`

Then click Save and close the tab.

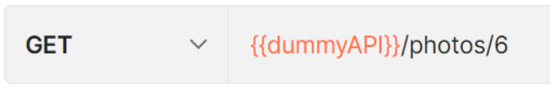
Create a new Request tab. Select the environment you just created from the drop down list at the upper right hand corner. This makes all the variables you have defined in that environment available to use in the current request you are forming.



Now type in the URL to make the request to, but this time using the variable you defined earlier

```
{{dummyAPI}}/photos/6
```

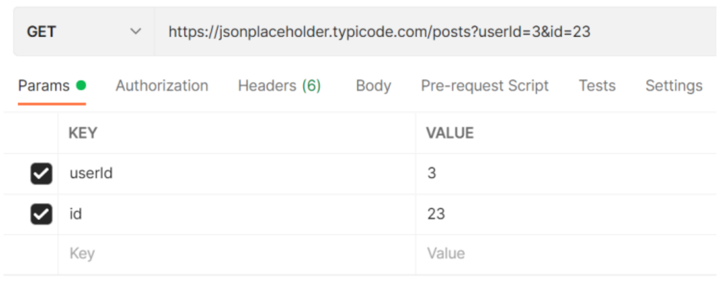
The variable should be highlighted in orange to indicate that it will be substituted with the actual value by Postman before the request is made.



Create a GET request with query parameters:

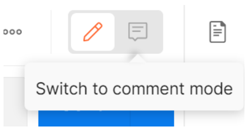
`{{dummyAPI}}/`posts?userId=3&id=23

Notice that the parameters are shown in the Params view, and you can change them in this view which will result in a reflected change in the URL itself in the main address bar and vice versa. You can deselect specific key-value pairs or add in new key-value pairs in this view.


GET https://jsonplaceholder.typicode.com/posts?userId=3&id=23

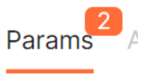
	KEY	VALUE
✓	userId	3
✓	id	23
	Key	Value

You can also place comments on these params. In the upper right hand corner, select to switch to Comment mode.

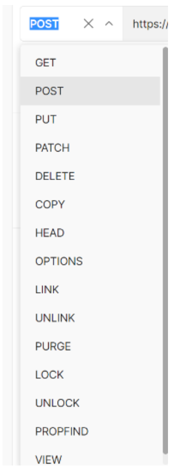


Move the cursor to point to the `userID` key: it should change to sticky note icon. Click and enter a sample comment for this key: for e.g. `This is the ID of the user`
Repeat this for the `id` key and enter the comment: `This is the id of the post`

This should result an icon above the Params tab indicating that there are 2 comments there when you switch to the Comments view.



Let's try sending other types of HTTP requests. The drop down list next to the method type gives you list of request methods available, which you can see is more than the standard ones associated with a typical REST API.



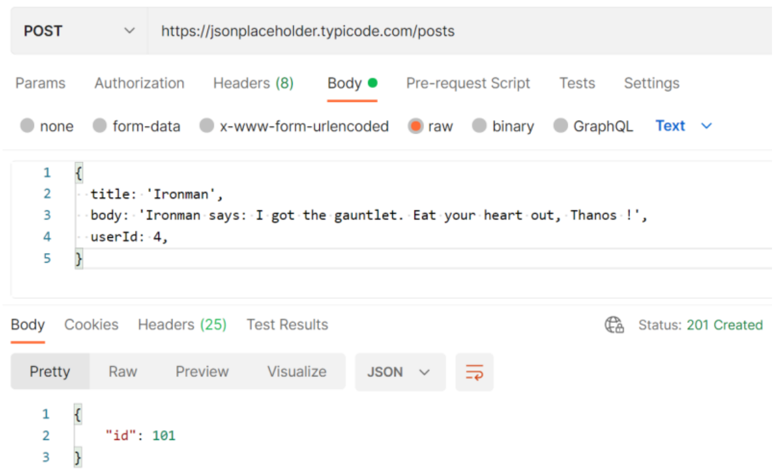
We can send a POST request to:

```
{{dummyAPI}}/posts
```

with the following body content:

```
{
  title: 'Ironman',
  body: 'Ironman beats Thanos anytime !',
  userId: 4,
}
```

This how we would place the content in Postman, using the `raw` option.



https://jsonplaceholder.typicode.com/posts

```
{
title: 'Ironman',
body: 'Ironman says: I got the gauntlet. Eat your heart out, Thanos !',
userId: 4,
}
```

Status: 201 Created

```
{
    "id": 101
}
```

You can again place comments on any content in the body using the approach we outlined earlier if you wish, then send it off.

Notice the response has a status 201 Created, which is a typical status for a successful creation of a new resource on the server end. The body of the response contains the id of the newly created resource (in this case 101) because there are currently 100 posts in the collection. However, as mentioned earlier, this is a fake API and there is no actual creation of a new resource on the server end. Thus, if you keep issuing multiple POST requests to the same API endpoint, you will constantly keep getting back the same response.

You can try sending a POST request to:

```
{{dummyAPI}}/comments
```

with the content:

```
{
  "id": 1,
  "name": "Peter Parker",
  "email": "spiderman@gmail.com",
  "body": "My webs are unbreakable !"
}
```

and you would get a response with the `id: 501` as there are currently 500 comments in the resource collection.

We can modify a resource with the PUT method. Send a PUT request to this endpoint:

```
{{dummyAPI}}/posts/1
```

with the following content:

```
{
  "userId": 1,
  "id": 1,
  "title": "Marvel comics",
  "body": "I love the MCU. Its awesome !"
}
```

Notice that the content contains all the fields expected in a single `posts` resource, because a PUT essentially overrides the entire content of that resource. The response is a 200 OK with the id of the resource being updated.

We can also modify a resource with the PATCH method. Send a PATCH request to the same endpoint:

```
{{dummyAPI}}/posts/1
```

with the following content:

```
{
  "title": "DC comics",
}
```

Notice that the content only contains the field to be changed and not all the fields expected in a single posts resource. So PUT does a complete update of the entire resource identified, while PATCH does a partial update. This is the main difference between the two. The response is still a 200 OK but the body of the response contains the entire resource that was being modified (and not just the id as in the case of a PUT). This allows the client to also verify the contents of the other fields in the modified resource. In this example here, the `title` field remains unchanged because this is a fake API, but in a real API this would now have the new value of `DC comics`

Finally, we can send a DELETE request to the same API endpoint without any body content:

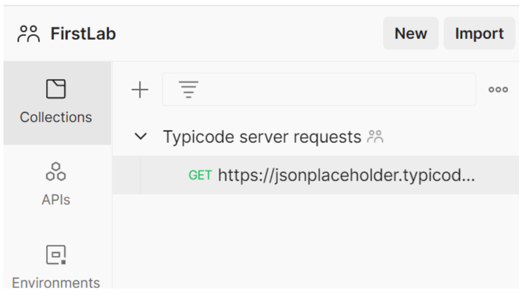
`{{dummyAPI}}/posts/1`

The response is a 200 OK with an empty body. There is no need to provide any content in the response, the 200 OK status indicates that the delete operation was successful. Again, because this is a fake API, the resource is not actually deleted so if you attempt a GET to the same API endpoint, it will succeed. In a functional REST API, attempting a GET to a resource that has already been deleted should return a 404 Not found error.

You can experiment around with sending different REST HTTP requests (GET, POST, PUT, PATH, DELETE) to multiple API endpoints, keeping in mind that you will not get proper effect on the server-side end as this is a FAKE API.

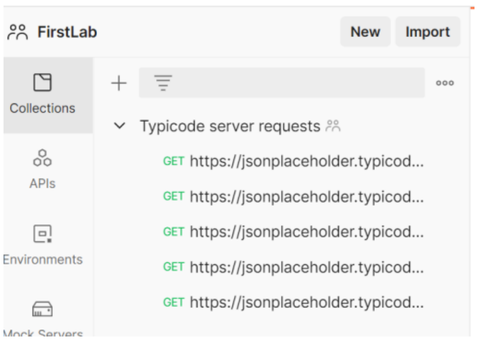
When you are done, you can save the numerous requests you have made in a Collection - which is a way to gather together related requests.

Select any request by clicking on its tab, and then select Save. You will be presented with a dialog box where you can provide an alternative name for the request and a description.
Then select Create a Collection and in the text box for the collection type: `Typicode server requests` then click Create.
Finally, select Save at the bottom of the dialog box to save the request into the newly created collection. You should now be able to see the newly created collection and the single saved request in the left hand pane.



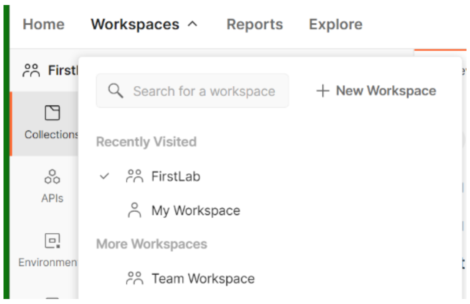
Typicode server requests
GET https://jsonplaceholder.typicod...

Close the tab for the request that you have saved. Repeat the saved operation for all the other open tabs for unsaved requests, saving them to the same collection and then close them. When you are done, you should be left with the Overview pane and the list of saved requests in the collection on the left.



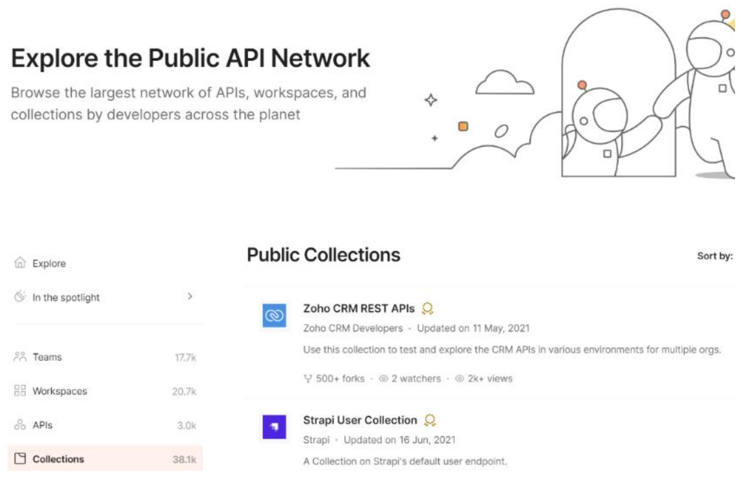
Typicode server requests
GET https://jsonplaceholder.typicod...
GET https://jsonplaceholder.typicod...
GET https://jsonplaceholder.typicod...
GET https://jsonplaceholder.typicod...
GET https://jsonplaceholder.typicod...

Close the Postman app and start it up again. Select Workspaces and open up the workspace that you created (FirstLab), and you should be able to see the collection you created and all the requests currently saved in it.



Click on any of the saved requests to retrieve it, then you can work on it in the usual manner and then send it.

Click on the Related Collections icon on the right hand pane. This allows you to discover public collections of API requests that are similar to your request on the Public API Network. This network is analogous to GitHub: it is a public repository of collections created by Postman users globally to share with others. Click on the Public API network to navigate through this list of collections


Explore the Public API Network

Browse the largest network of APIs, workspaces, and collections by developers across the planet

Public Collections

Sort by:

Explore	
In the spotlight	
Teams	17.7k
Workspaces	20.7k
APIs	3.0k
Collections	38.1k

Zoho CRM REST APIs
Zoho CRM Developers · Updated on 11 May, 2021
Use this collection to test and explore the CRM APIs in various environments for multiple orgs.
500+ forks · 2 watchers · 2k+ views

Strapi User Collection
Strapi · Updated on 16 Jun, 2021
A Collection on Strapi's default user endpoint.

When you are done, click on the Workspaces -> FirstLab to return back to the current workspace.

4 Working with APIs on RapidAPI

With the rapid explosion in the popularity of REST APIs, many organizations across the world now provide services for other organizations or the public to interact via this approach. A follow up from this are the emergence of businesses that aggregate some of the more well known and well managed REST APIs in the public domain and provide a higher-level platform to access these APIs in a uniform and consistent way. RapidAPI is an example of such a business.

Login to your Rapid API account. Go to the Hub to discover APIs that you can interact with: https://rapidapi.com/hub

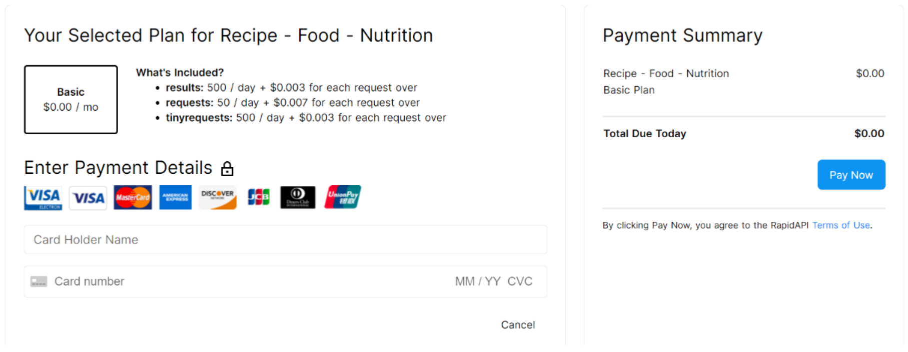
Scroll down and select one of the APIs in the Popular APIs section. Let select the Recipe - Food - Nutrition.

https://rapidapi.com/spoonacular/api/recipe-food-nutrition/

At the top part of the page, you can see the main sections to navigate to (which will be standard for all APIs on the RapidAPI Hub): Endpoints, About, Tutorials, Discussions, Pricing and Specs. Navigate to these different sections to see what are available on them.

In order to be able to interact with the API, you will need to get a subscription. Most RapidAPIs work on a freemium model which provides free subscription for basic interaction rates and charge a rated subscription fee for heavier interaction traffic.

Navigate to the Pricing section and subscribe to the Basic plan. You will transition to a payment details page to enter your credit card details. Enter the appropriate details here, and select Pay Now. For the purposes of this workshop, we will not be performing any work that will incur any charge so you can go ahead and register your credit card. Then click Pay Now.


Your Selected Plan for Recipe - Food - Nutrition

Basic
$0.00 / mo

What's Included?
- **results:** 500 / day + $0.003 for each request over
- **requests:** 50 / day + $0.007 for each request over
- **tinyrequests:** 500 / day + $0.003 for each request over

Enter Payment Details

Card Holder Name

Card number MM / YY CVC

Cancel

Payment Summary

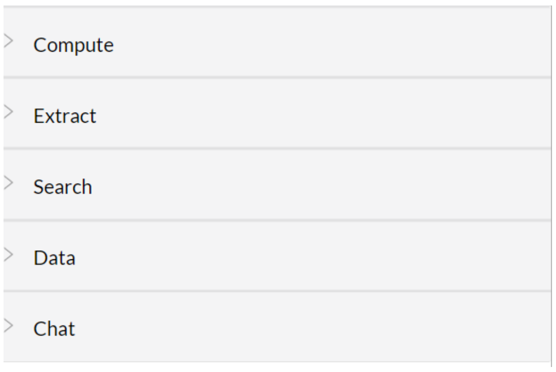
Recipe - Food - Nutrition Basic Plan	$0.00
Total Due Today	**$0.00**

Pay Now

By clicking Pay Now, you agree to the RapidAPI Terms of Use.

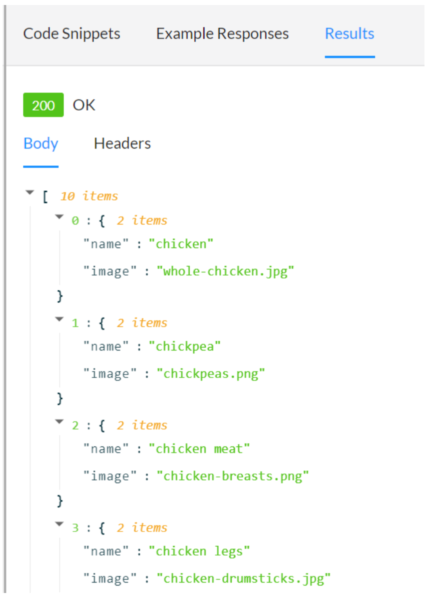
Once you have subscribed, you should get a confirmation email at the email address that you registered with your RapidAPI account.

Click Endpoints to return to the section where you can start sending HTTP REST requests. The list of all possible requests are classified into different categories in the left hand side pane.



For the first request, we will select a simple GET request. Expand on the Search category and select: GET Autocomplete Ingredient Search. The items in the required parameters section need to be completed by the user while the optional parameters can be left out or kept at their default value. Type in any string in the query: for e.g: `chick` and click Test Endpoint to send off the request.

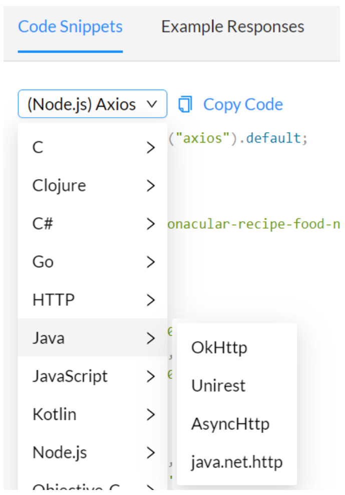
You should be able to study the body contents and response headers in the Results view on the right.



```
[ 10 items
  0 : { 2 items
    "name" : "chicken"
    "image" : "whole-chicken.jpg"
  }
  1 : { 2 items
    "name" : "chickpea"
    "image" : "chickpeas.png"
  }
  2 : { 2 items
    "name" : "chicken meat"
    "image" : "chicken-breasts.png"
  }
  3 : { 2 items
    "name" : "chicken legs"
    "image" : "chicken-drumsticks.jpg"
```

If you click on the Example Responses view, you will see a sample response based on a particular GET request, which you can use to compare with the actual response you received to see whether there is anything amiss.

If you click on the Code Snippets view, you will see the specific code snippet that you will need to write in the specific library for a given programming language in order to send out that particular REST HTTP request: this is identical to the feature that we saw earlier in Postman.



If you look at the Java libraries, in addition to the OkHttp and Unirest shown in Postman, we have 2 additional libraries:

3. AsyncHttp - https://loopj.com/android-async-http/
4. Java.net.http - https://openjdk.java.net/groups/net/httpclient/intro.html

Let's resend the same request, but this time change the optional parameter `number` to 5. Verify now that you only receive back only 5 items in the response.

Let's resend the same request, but this time change the optional parameter `metaInformation` to `True.` Verify now that the items you receive in the response contains much more fields to provide metainformation about the item (for e.g. `aisle` and `possibleUnits`)

Experiment with different values for the required and optional parameters and study the response results carefully.

We will now replicate this API call in our Postman client. This is to demonstrate that we can use multiple GUI clients (RapidAPI dashboard / Postman) or code (multiple libraries in more than 15 languages) to produce the same HTTP Request and process the returned HTTP Response.

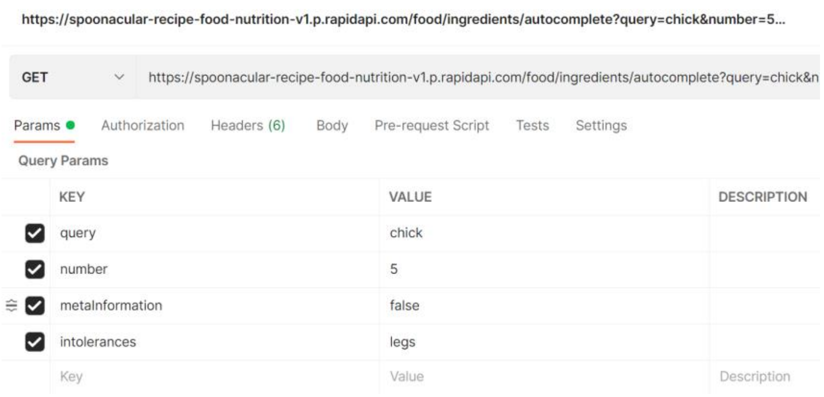
- The values for `X-RapidAPI-Key` and `X-RapidAPI-Host` must be specified as Header values.

- The values for the `required` and `optional` parameters are specified as query parameters in the form of:

`parameter1=value1&parameter2=value2&…….parameter-n=value-n`

Open up Postman and crate a new Collection called: `Recipe Food Nutrition`

Close existing requests from the previous lab / Collection and create a new request.
The URL for this new request should be the value for the path portion of the URL (i.e. the portion starting from https:// until the ?) as specified in any of the code snippets. You can simply copy and paste from there.
The query params for this new request is the query string portion of the URL (i.e. the portion after the ?) as specified in any of the code snippets. You can either append it to the previous URL or type it in via the Params box in Postman.

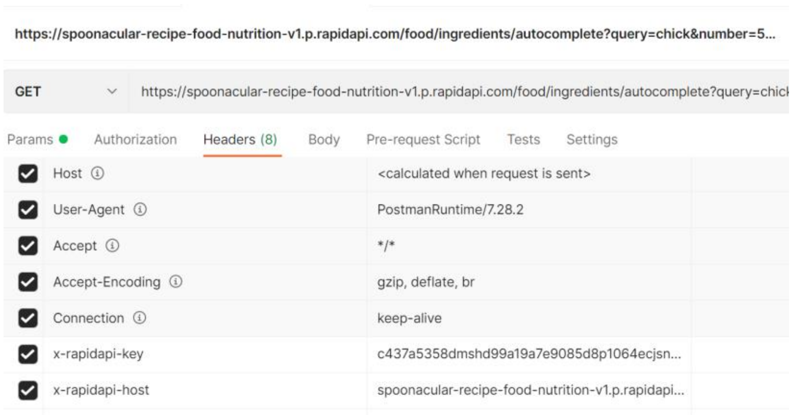

https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/ingredients/autocomplete?query=chick&number=5...

GET https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/ingredients/autocomplete?query=chick&n

Query Params

	KEY	VALUE	DESCRIPTION
✓	query	chick	
✓	number	5	
✓	metaInformation	false	
✓	intolerances	legs	
	Key	Value	Description

Switch to the Headers view in Postman and create two additional Headers with the keys of `x-rapidapi-key` and `x-rapidapi-host` and paste their values from the related portion of the code snippet.



https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/ingredients/autocomplete?query=chick&number=5...

GET https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/ingredients/autocomplete?query=chick

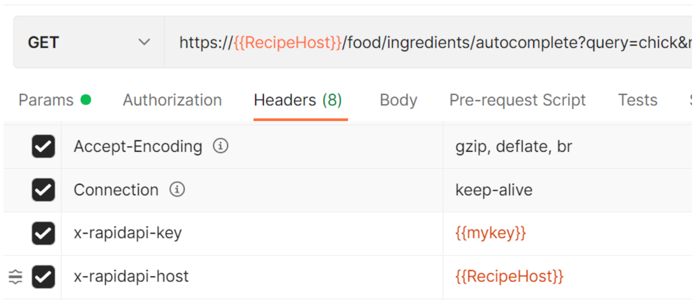
✓ Host	<calculated when request is sent>
✓ User-Agent	PostmanRuntime/7.28.2
✓ Accept	*/*
✓ Accept-Encoding	gzip, deflate, br
✓ Connection	keep-alive
✓ x-rapidapi-key	c437a5358dmshd99a19a7e9085d8p1064ecjsn...
✓ x-rapidapi-host	spoonacular-recipe-food-nutrition-v1.p.rapidapi...

When you are done, click Send. Verify that your response obtained is the same as the one sent out via the RapidAPI web-based dashboard.

If you wish now, you can create an environment for variables to work with this particular API service in a similar way that we did in the earlier lab session.
Environment: `Recipe Environment`

Variables to set :-
Variable: `RecipeHost`
Initial Value: `spoonacular-recipe-food-nutrition-v1.p.rapidapi.com`
Variable: `mykey`
Initial Value: *`Your key value`*

Using this environment, you can restructure your current request (and all subsequent requests) to reuse the variable that you have just defined:



GET https://{{RecipeHost}}/food/ingredients/autocomplete?query=chick&r

✓ Accept-Encoding	gzip, deflate, br
✓ Connection	keep-alive
✓ x-rapidapi-key	{{mykey}}
✓ x-rapidapi-host	{{RecipeHost}}

You will notice that GET requests which can return a large number of results (hundreds or even thousands of items) have two optional parameters: `offset` and `number` to allow the user to select a subset of the results to be returned in the response. This allows the implementation of pagination and also reduces latency and network bandwidth consumption in the return of the response.

Try the GET Search Grocery Products query in the Search category to experiment with how you can obtain a subset of results returned by changing the value of these two parameters. Use a query such as `chicken` which is likely to return a large number of results.

If you browse through the different categories available for this API on the left pane, you will notice that the vast majority of requests are GET, with the remaining few been POST. This is because this API is open to the public and allowing everyone to have unfettered access to modify or delete its contents would result in chaos and a meaningless API state. Even the few remaining POST methods are in actually implementing a GET functionality.

Let's try out a POST method. First, randomly select one of the entries returned by the query: chicken in the GET Search Grocery Products query in the Seach category. Get the value for its title field for e.g.: `"Anytizers Tyson Any'tizer Buffalo Style Hot Wings"`

Scroll up to the Compute category and select the POST Classify a Grocery Product request. For the request body, use the value that you obtained earlier, i.e.

```
{
    "plu_code": "",
    "title": "Anytizers Tyson Any'tizer Buffalo Style Hot Wings",
    "upc": ""
}
```

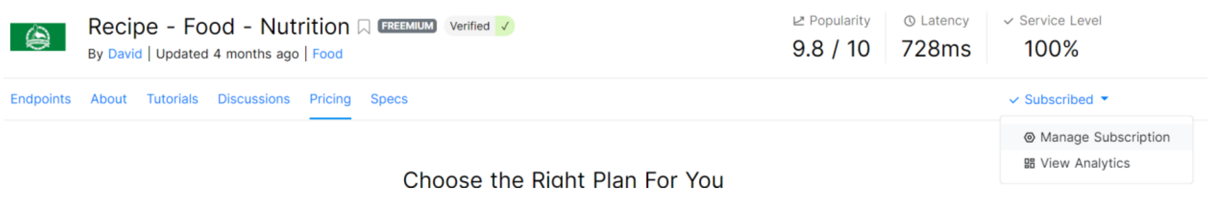
And you will see from the response that its basic ingredients are:

```
0:"chicken wings"
1:"chicken"
2:"poultry"
3:"meat"
4:"animal product"
```

Spend some time exploring the various API endpoints exposed by this particular service, the content returned and the way the URL paths are structured to reflect the content returned. This will give you some idea of how to go about designing a REST API of your own. You can return to the API Hub (https://rapidapi.com/hub) to subscribe to other REST APIs in other categories and explore interacting with them.

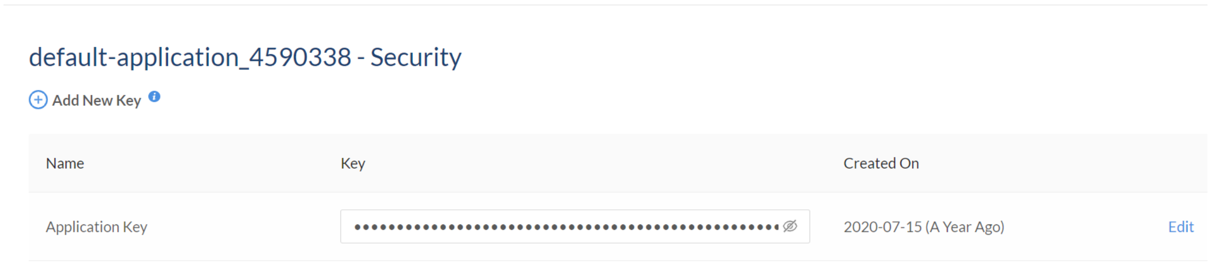
Work at interacting with them via the RapidApi dashboard and also through Postman until they become second nature to you. You can save all your Postman requests in the collection you created in order for you to refer back to them later.

When you are done working with the various APIs you have subscribed to and no longer wish to interact with any one of them, you can opt to unsubscribe from it. This will remove your credit card details from being associated with it. Select Manage subscription from the drop down menu at the right.



On the Subscriptions & Usage page, you can see a list of all the public APIs that you are subscribed to and you can then click on Unsubscribe.

If you click on the My Apps, you will see a single entry for the default application that is created for you to send the various HTTP REST requests while you are logged into your account. You can edit this to change the name, and select Configuration, Analytics and Security to update or view details for each of these sections. In particular, in the Security section, you can create a new Application Key for use with future REST API calls.



default-application_4590338 - Security

Add New Key

Name	Key	Created On	
Application Key	•••	2020-07-15 (A Year Ago)	Edit

This Application Key uniquely identifies your account identity for authentication, billing and monitoring purposes when you make API calls to a given REST API server that is part of the RapidAPI platform. This means that if this key is used in any piece of code (either written you or someone else), it will still count towards any billing / monitoring usage. Thus, you should always safeguard the confidentiality of your key and ensure that you only give it to other trusted people to use on your behalf.

5 Working with other public REST APIs

The majority of public REST APIs require some form of registration and authentication to work with via an API key that is included in the requests to the API server. They typically offer a freemium model similar to RapidAPI where you are able to access a limited form of the API functionality available or interact with a lower traffic rate for free, and then subscribe for more complete functionality or higher traffic interaction rates.

A good illustration of this is:
https://www.exchangerate-api.com/

Once you have registered an account with them, you will be given an API key which you will include as part of the URL path in all your outgoing requests, for e.g.

```
GET https://v6.exchangerate-api.com/v6/YOUR-API-KEY/latest/USD
```

To specify another country as the base currency for conversion, supply the country's 3 letter code in the URL path. For a full listing see:
https://www.exchangerate-api.com/docs/supported-currencies
For e.g. to use RM as the base currency for conversion, use:

```
GET https://v6.exchangerate-api.com/v6/YOUR-API-KEY/latest/MYR
```

This organization provides proper documentation of the API endpoints to use as well as code examples from a variety of languages.

https://www.exchangerate-api.com/docs/overview

You can explore making calls to this API using Postman.

Many organizations have a very extensive REST API to allow developers to create apps that interact with the resources that they provide. For e.g. GitHub, the world's largest online code repository, provides a REST API that allows the development of app that interact with the code repos of an organization hosted on their site.

https://docs.github.com/en/rest

You can browse through the documentation to get an idea of how extensive their REST API is and how it is documented.

You can also find a list of public APIs here:

https://any-api.com/
https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/

6 Using command line tools as a REST client

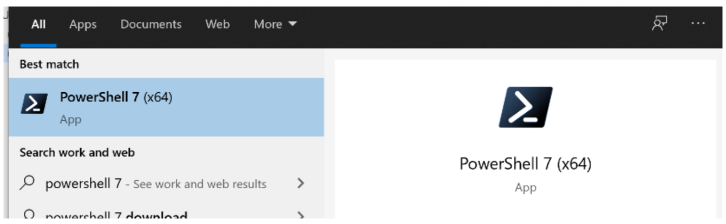
In addition to using Postman (or other similar GUI-based clients such as Insomnia, HTTPie, Fiddler, etc), we can also interact with a REST API service from the command line.
The most common approach is the cURL tool in Linux / Unix environments.

In Windows, there are several options:

i. curl.exe - which is nearly similar functionality to the Linux cURL, but with slightly different delimiter symbols and options.
ii. Invoke-WebRequest - a native Powershell cmdlet in Windows Powershell that is conceptually similar to Linux cURL, but with completely different option specifications.
iii. Invoke-RestMethod - a native Powershell cmdlet in Powershell Core > 6.0 that has the roughly the same functionality but is more tailored to manipulating and processing JSON / XML that is used in the majority of REST API interactions.

We will be using the third approach in this lab.

Note that there are currently two versions of PowerShell: a classic Windows PowerShell (the latest version is 5.1 and it is no longer developed) and a new PowerShell Core platform (version 7.1 is available now). Despite the version numbering continuing on from 5.1 (6.0, 6.1, 7.0, 7.1, etc.), these are two different platforms. In order to use all the options for Invoke-RestMethod, you will need to have PowerShell Core 7 installed. This is how PowerShell 7 looks like when accessed from the Search Bar:



The installation file for this can be found at:
https://github.com/PowerShell/PowerShell

Even if you plan to work with a GUI client rather than from the command line, knowing the most common forms of the command line invocations (particularly for cURL) can be useful as a lot of documentation on REST APIs demonstrate how they work using CuRL commands rather than GUI client setups. An example is the GitHub REST API reference:
https://docs.github.com/en/rest/reference/repos

Knowing how to map from a cURL command to setting up an equivalent HTTP REST request (and vice versa) will facilitate your work, particularly in certain environments where you only have access to the command line and cannot run a REST GUI client.

6.1 cURL in Linux

Some examples are given below based on the fake REST API site:
https://jsonplaceholder.typicode.com/

To send a HTTP GET request to a given URL:

```
curl https://jsonplaceholder.typicode.com/users
```

When sending request with query parameters, ensure you enclose the entire URL in quotes to avoid confusion with the & character that is used to separate key-value pair parameters.

```
curl "https://jsonplaceholder.typicode.com/comments?postId=2&id=7"
```

To send a HTTP GET request to a RapidAPI REST API service, we simply add the relevant values for `x-rapidapi-key` and `x-rapidapi-host` as header arguments using -H.

```
curl -H "Content-Type: application/json" -H "x-rapidapi-key:key-value" -H "x-rapidapi-host:host-value" "https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/ingredients/autocomplete?query=chicken&number=10&intolerances=egg"
```

When sending data via a POST, PUT or PATCH request, two common formats (specified via the Content-Type header) are:

- `application/json`
- `application/x-www-form-urlencoded`

Many APIs will accept both formats, and if you use curl without explicitly specifying a format in the Header, then the default will be: `application/x-www-form-urlencoded`

To send a POST / PUT / PATCH request to a given URL in form-urlencoded format:

```
curl -d 'title=Ironman&body=Ironman beats Thanos&userId=4' -X POST https://jsonplaceholder.typicode.com/users
```

To send a POST / PUT / PATCH request to a given URL in JSON format:

```
curl -d '{"title":"Ironman", "body":"Ironman beats Thanos", "userId":"4"}' -H "Content-Type: application/json" -X POST https://jsonplaceholder.typicode.com/users
```

Alternatively, you can save the data content to a file and specify the file name instead. For example if the JSON content is saved to `data.json`, then use:

```
curl -d @data.json -H "Content-Type: application/json" -X POST https://jsonplaceholder.typicode.com/users
```

For a PUT / PATCH, just use `-X PUT` or `-X PATCH` instead of `-X POST`

To send a HTTP DELETE request to a given URL:

```
curl -X DELETE https://jsonplaceholder.typicode.com/posts/1
```

To list the headers in the response, simply add the -i option:

```
curl https://jsonplaceholder.typicode.com/users -i
```

To get verbose output, use the -v option:

```
curl https://jsonplaceholder.typicode.com/users -v
```

This provides output on the entire SSL handshake process as well as the headers in the response.

To summarize the key cURL options:
- -X specifies the HTTP method
- -d specifies the content in the body to be sent, either explicitly or by specifying a file
- -H specifies individual headers to be included in the request
- -i lists the headers in the response
- -v provides more information which is useful for debugging

As an exercise, you can try using cURL to send HTTP REST requests to the public APIs on the RapidAPI platform or elsewhere.

Both Postman and RapidAPI provide the cURL snippets corresponding to the HTTP requests that you create in them. You can also copy and paste the snippets from there for more complex requests in case you are not clear about what the parameters are.

6.2 Invoke-RestMethod on Windows

Some examples are given below based on the fake REST API site:
https://jsonplaceholder.typicode.com/

To send a HTTP GET request to a given URL:

```
Invoke-RestMethod https://jsonplaceholder.typicode.com/users
```

When sending request with query parameters, ensure you enclose the entire URL in quotes to avoid confusion with the & character that is used to separate key-value pair parameters.

```
Invoke-RestMethod
"https://jsonplaceholder.typicode.com/comments?postId=2&id=7"
```

To send a HTTP GET request to a RapidAPI REST API service, we simply add the relevant values for `x-rapidapi-key` and `x-rapidapi-host` as values to a $Header object which we subsequently use in our invocation:

```
$Header = @{
"x-rapidapi-key" = "key-value"
"x-rapidapi-host" = "host-value"
}

Invoke-RestMethod  -Uri  "https://spoonacular-recipe-food-nutrition-
v1.p.rapidapi.com/food/ingredients/autocomplete?query=chicken&number
=10&intolerances=egg" -Headers $Header
```

When sending data via a POST, PUT or PATCH request, two common formats (specified via the Content-Type header) are:

- `application/json`
- `application/x-www-form-urlencoded`

To send a POST / PUT / PATCH request to a given URL in JSON format, we declare the body and parameters as separate objects which are subsequently passed to the Invoke-RestMethod.

```
$Body = @{
    title = "Ironman"
    body = "Ironman beats Thanos"
}

$Parameters = @{
    Method = "POST"
    Uri = "https://jsonplaceholder.typicode.com/users"
    Body = ($Body | ConvertTo-Json)
    ContentType = "application/json"
}

Invoke-RestMethod @Parameters
```

To send in form-urlencoded format, simply change in $Parameters:
`ContentType = "application/x-www-form-urlencoded"`

Alternatively, you can save the data content to a file and specify the file name instead. For example if the JSON content is saved to `data.json`, then use:

```
$Parameters = @{
    Method = "POST"
    Uri = "https://jsonplaceholder.typicode.com/users"
    ContentType = "application/json"
}
```

```
Invoke-RestMethod @Parameters -InFile data.json
```

For a PUT / PATCH, just use `Method = "PUT"` or `Method = "PATCH"` in `$Parameters`

To send a HTTP DELETE request to a given URL:

```
Invoke-RestMethod -Method Delete
https://jsonplaceholder.typicode.com/posts/1
```

To view the response headers, store them into a variable first and retrieve them after the invocation:

```
Invoke-RestMethod -ResponseHeadersVariable RespHeaders -Uri
"https://jsonplaceholder.typicode.com/comments?postId=2&id=7"

$RespHeaders
```

As an exercise, you can try using Invoke-RestMethod to send HTTP REST requests to the public APIs on the RapidAPI platform or elsewhere.

Postman provides the Invoke-RestMethod snippets corresponding to the HTTP requests that you create in them. You can also copy and paste the snippets from there for more complex requests in case you are not clear about what the parameters are.