Ejercicios de práctica - Postman

Ejercicio #1 - Fork Collection

Cree una colección de nombre `Ejercicio 1` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Revisa la solicitud:** Crea una solicitud en la carpeta y fíjate en las diferentes partes de esta solicitud de la API.
   a. Método de solicitud HTTP `POST` - para enviar nuevos datos a una API.
   b. `https://postman-echo.com/post` URL de solicitud.
   c. En la pestaña *Body*, seleccione el tipo de cuerpo a *raw* y en el desplegable del final donde dice *Text*, seleccione *JSON*. Ingrese en el cuerpo de la solicitud el siguiente código.

```
{
  "payload": "hello world"
}
```

2. **Envíe la solicitud:** Envíe la solicitud y observe la respuesta en la parte inferior.

a. Código de estado HTTP.

b. Tiempo de respuesta.

c. Tamaño de la respuesta.

d. Cuerpo de la respuesta.

3. **Inspeccione el cuerpo de la respuesta:** Bajo la vista *Pretty* del cuerpo de la respuesta, inspeccione el objeto de datos JSON devuelto por el servidor. Este ejemplo de solicitud de la API Echo de Postman devuelve los datos proporcionados por el cliente.

Ejercicio #2 - Collection and Environment

Cree una colección de nombre `Ejercicio 2` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Cree una variable:** Fíjate en la misma petición del ejercicio anterior. Copie la primera parte de la URL de la solicitud (la parte que precede a `/post`) y sustitúyala por `{{baseURL}}`. Las dobles llaves son la forma de obtener variables en los campos de texto de Postman. Si pasa por encima de la variable, la punta de la herramienta dice "Unresolved Variable" es porque no se ha establecido todavía.
2. **Cree un entorno:** Cree un nuevo entorno con una clave `baseURL`. El valor de esta nueva variable debe ser la parte que copió de la URL de la solicitud en el paso anterior. Puedes introducirlo tanto en `INITIAL VALUE` como en `CURRENT VALUE`. Recuerda seleccionar el nuevo entorno como entorno activo para que Postman lea los valores correctamente. Si pasas el ratón por encima de la variable `{{baseURL}}` en la URL de la petición, deberías ver a Postman leyendo el valor de la variable desde el entorno seleccionado actualmente. Ahora que has almacenado la URL base en una variable llamada `baseURL`, puedes hacer referencia a ella en tus peticiones utilizando `{{baseURL}}`. Si el valor de la URL base es `https://postman-echo.com`, y aparece como parte de la URL de la solicitud utilizando `{{baseURL}}/post`, Postman enviará la solicitud a `https://postman-echo.com/post`. Guarde los cambios en la colección y el entorno.
3. **Añada una solicitud:** Duplique la solicitud en esta carpeta, y cambie el nombre de la segunda solicitud por `GET with query params`. Actualiza el nuevo método HTTP de

la petición a `GET` y actualiza la URL de la petición de `/post` a `/get`. Añade dos parámetros de consulta. Pueden ser los que quieras, por ejemplo, llave `foo` y valor `bar` o llave `name` y valor `ferret`. Envía para asegurarte de que obtienes un código de estado 200.

Ejercicio #3 - Add request details

Cree una colección de nombre `Ejercicio 3` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añadir una solicitud:** Añadir una solicitud a la carpeta `Add request details` llamada `raw JSON body` con los siguientes detalles:
   a. Método `POST`
   b. `{{baseURL}}/post` solicitud URL
   c. Selecciona el entorno que creaste en el ejercicio anterior para que Postman lea el valor apropiado para `{{baseURL}}` en la URL de la petición.
2. **Incluya un cuerpo:** Añade un objeto JSON, como el siguiente, y asegúrate de que recibes un código de estado 200 cuando pulses Enviar.

   ```
   {
     "data": "doodles"
   }
   ```

   También puedes ver los resultados de la petición para ver que el elemento `json` en la respuesta ha copiado el JSON que has añadido en el cuerpo de salida. Si la respuesta dice `"json": null`, entonces puede que hayas hecho algo mal.
   Cuando añadiste un cuerpo de solicitud JSON sin procesar, Postman autogeneró una cabecera `Content-Type: application/json` en la pestaña *Headers*. Es posible que tenga que hacer clic en el botón oculto para mostrar las cabeceras. Aquí es donde también se pueden configurar otras cabeceras de solicitud.
3. **Revise otras opciones de solicitud:** Hay un montón de otras formas de configurar peticiones, como actualizar la URL, el método y otros valores opcionales como

autenticación y parámetros. Puedes aprender más sobre cómo añadir detalles a las peticiones aquí.

Ejercicio #4 - Authorization

La API Key de Postman le permite acceder mediante programas a los datos almacenados en su cuenta de Postman. Esta API requiere una clave de API para indicar al servidor de la API que una solicitud que recibe proviene de usted y que tiene permiso para acceder a los datos. Echa un vistazo a la documentación y explora la colección.

Cree una colección de nombre `Ejercicio 4` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una solicitud:** Copia la colección de la API de Postman a tu espacio de trabajo. A continuación, busque una solicitud `Get all collections`. Duplíquela y arrastre la copia a la carpeta Autorización. Cambie el nombre de la solicitud por el de `All Collections - header`.
2. **Añade una variable de entorno:** En el entorno creado en ejercicios anteriores, añade una clave de nombre `postman_api_key` y asígnele la clave generada por la API Postman en el `CURRENT VALUE`.
   Los valores iniciales pueden ser accesibles para otros en un espacio de trabajo público o de equipo. Los valores actuales son locales, y no se sincronizan o comparten. Dado que estamos trabajando en un espacio de trabajo público, es muy importante añadir sólo datos sensibles a los `CURRENT VALUE`. Más información sobre cómo compartir y mantener datos.
3. **Añade una cabecera:** En la documentación de la solicitud, vemos que hay dos maneras de autorizar esta solicitud en particular. Empecemos por añadir una cabecera de autorización. En la pestaña *Headers*, añade una clave `x-api-key` con el valor `{{postman_api_key}}`. Pasa por encima de la variable para asegurarte de que Postman está leyendo el valor correcto de un entorno activo. Envíe para asegurarse de que recibe un código de estado 200 y, a continuación, guarde los cambios.
4. **Añada un ayudante de autenticación:** En lugar de añadir manualmente una cabecera o params, vamos a utilizar uno de los ayudantes de autorización

proporcionados por Postman. Duplique la última solicitud y cambie el nombre de la copia a `All Collections - auth helper`. Elimine el par clave-valor en la pestaña *Params*. En la pestaña *Authorization*, seleccione el tipo `API key` en el menú desplegable. Añade `x-api-key` y `{{postman_api_key}}` para que se añada como cabecera. Envíe para asegurarse de que recibe un código de estado 200 y, a continuación, guarde los cambios. Mira en *Headers* para ver la cabecera que Postman autogeneró desde el auth helper.

5. **Añadir autenticidad a nivel de carpeta:** Dado que todas las solicitudes de esta carpeta requieren el mismo método de autorización, vamos a añadir un ayudante de autorización a nivel de carpeta. Para cada una de estas tres peticiones, elimine la cabecera añadida, elimine los parámetros y actualice el tipo de autentificación de la solicitud anterior a `Inherit auth from parent`. El padre de esta solicitud es la carpeta que contiene estas solicitudes. Seleccione la carpeta `Authorization`, y añada el auth helper de la `API key` en la pestaña `Authorization` como hizo en el paso anterior. Envíe al menos una de las solicitudes para asegurarse de que recibe un código de estado 200 y, a continuación, guarde todos los cambios.

Hay un número de maneras en que los servidores API pueden requerir autorización. Echa un vistazo a los otros auth helpers disponibles en Postman.

Ejercicio #5 - Variables

Ya hemos utilizado variables de entorno en retos anteriores. Vamos a utilizar la API de CoinDesk para aprender más sobre las variables en Postman. Esta API te permite acceder programáticamente al precio del bitcoin.

Cree una colección de nombre `Ejercicio 5` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Añade una petición llamada `collection variable` a la carpeta Variables con los siguientes detalles:
   a. Método `GET`.
   b. `https://api.coindesk.com/v1/bpi/currentprice/btc.json` URL de la petición.
2. **Añade una variable de colección:** Resalte `https://api.coindesk.com` en la URL de la solicitud y establezca una nueva variable de colección llamada `coindeskBaseUrl`

con el valor `https://api.coindesk.com`. Pasa el ratón por encima de la nueva variable para comprobar que la variable está establecida en el ámbito adecuado (colección). Una variable de colección está estrechamente vinculada a la colección y sólo puede utilizarse con esa colección. Por otro lado, las variables de entorno se pueden utilizar en muchas colecciones, de la misma manera que has almacenado tu clave de API de Postman en el mismo entorno para utilizarla con diferentes colecciones. Pulsa enviar para asegurar un código de estado 200, y guarda tus cambios.

3. **Añada una variable global:** Duplique la solicitud dentro de esta carpeta, y cambie el nombre de la segunda solicitud a `variable global`. Cree una variable global de la forma que prefiera. La variable global debe llamarse `currency` con el valor `btc` sustituyendo la parte "btc" de la ruta URL de la petición. Una variable global es accesible dentro del espacio de trabajo actual. Puedes editar manualmente las variables globales en la pestaña *Environments* de la barra lateral izquierda, en *Globals*. Hágalo ahora para actualizar `btc` a una moneda diferente de su elección, como `usd`. Envíe para asegurar un código de estado 200, y guarde sus cambios.

Postman proporciona un número de variable scope para adaptarse a su caso de uso particular. Aprenda más sobre los variable scope y cuándo elegirla.

Ejercicio #6 - Test

Mucha gente utiliza Postman para probar y explorar sus APIs. En este reto, aprenderemos a escribir pruebas para afirmar mediante programación que tu API se comporta como esperas. Vayamos a la API de Chuck Norris.

Cree una colección de nombre `Ejercicio 6` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una solicitud:** Añade una solicitud llamada `jokes` a la carpeta `Tests` con los siguientes detalles

   a. Método `GET`

   b. `https://api.chucknorris.io/jokes/random` URL de la solicitud.

   Envía la petición e inspecciona la respuesta en la parte inferior. Piensa en cómo afirmarías que la API se comporta como se espera. Tal vez hayas recibido un

determinado código de estado, el cuerpo de la respuesta tenga un formato determinado o contenga un dato específico.

2. **Añada una prueba:** De vuelta en el constructor de peticiones en la parte superior, busque la pestaña de *Tests*, y expanda los fragmentos de la derecha. Esta es una lista de los scripts de prueba más populares escritos en JavaScript. Encuentra uno llamado `Status code: Code is 200`, y haga clic en él para insertarlo.
   Puede definir las pruebas utilizando la función `pm.test`, proporcionando un nombre y una función que devuelva un valor booleano (`true` o `false`) que indique si la prueba ha pasado o no. Envíe la solicitud, y busque en la pestaña *Tests* en el visor de respuestas en la parte inferior para ver el resultado de su prueba.
3. **Escriba una prueba que falle:** En la pestaña de *Tests* en el constructor de peticiones en la parte superior, copie y pegue ese fragmento debajo del original. Actualice el nombre de la prueba para que `espere un 404`, y luego reemplace la aserción del código de estado de `200` a `404`. La sintaxis de la aserción en esta función es de una biblioteca llamada Chai.js. Esta prueba espera un código de estado 404. Envíe la solicitud de nuevo, y revise los resultados de su prueba. Deberías tener una prueba que pase y otra que falle. Guarde los cambios.

Ejercicio #7 - Debugging

Vamos a ver la API de la Astronomía del Día de la NASA para aprender más sobre la depuración de las llamadas a la API en Postman. La consola de Postman te da visibilidad para depurar durante el desarrollo.

Cree una colección de nombre `Ejercicio 7` y dentro de este, una carpeta con el nombre del ejercicio.

1. Añade una petición: Añade una petición llamada `apod` con los siguientes detalles:
   a. Método `GET`.
   b. `https://api.nasa.gov/planetary/apod` URL de la petición.
   c. parámetro de consulta `api_key` con valor `DEMO_KEY`.
   d. parámetro de consulta `count` con valor `10`.

Abre la consola de Postman en la parte inferior izquierda. Envíe la solicitud e inspeccione las distintas partes de la llamada de red (por ejemplo, red, cabeceras de solicitud, cabeceras de respuesta, cuerpo de la respuesta).

2. **Añade una declaración de registro:** En la pestaña *Tests* de la solicitud, añada una función con `console.log()` como se describe al final, para dar salida al título de cada imagen en la consola de Postman. Esto le permite analizar una respuesta rápidamente mientras está en desarrollo, en lugar de navegar por toda la carga útil. El uso de sentencias de consola como `console.log()`, `console.info()`, `console.warn()`, y `console.error()` en las pestañas de *Pre-request Script* o de pruebas le permite rastrear las ejecuciones de las solicitudes, las excepciones y los errores durante el desarrollo. Siéntase libre de añadir más declaraciones de registro, tal vez condicionalmente, para proporcionar más información sobre lo que se devuelve desde el servidor.

```
let pics = pm.response.json()
pics.forEach((pic) => { console.log(pic.title, pic.url) })
```

Ejercicio #8 - Run a collection

Utilicemos la API de usuarios aleatorios para aprender a ejecutar una colección en Postman.

Cree una colección de nombre `Ejercicio 8` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Añade una petición llamada `get random user` a la carpeta con los siguientes detalles:

   a. Método `GET`.

   b. `https://randomuser.me/api` URL de la petición.

   c. Al menos una prueba para determinar una respuesta exitosa de esta API.

   Envíe la solicitud para asegurarse de que sus pruebas pasan.

2. **Añade otra petición:** Duplicar la petición para crear una segunda petición llamada `get female user`. Actualice la solicitud añadiendo los parámetros necesarios para

obtener un único usuario femenino. Es posible que tenga que actualizar sus pruebas para garantizar una respuesta satisfactoria. Envía la petición para asegurarte de que tus pruebas pasan.

3. **Añade una tercera petición:** Duplique la solicitud para crear una tercera solicitud llamada `get french user`. Actualice la solicitud para recuperar un único usuario que sea a la vez mujer y francés. Una vez más, es posible que tengas que actualizar tus pruebas para asegurarte de que la respuesta sea satisfactoria. Envíe la solicitud para asegurarse de que sus pruebas pasan.
4. **Ejecute la carpeta:** Ahora que tiene tres solicitudes con pruebas, puede recorrer y ejecutar la carpeta pasando por cada una de las solicitudes y pulsando Enviar. Una forma más fácil de ejecutar las peticiones es utilizar el collection runner. Ejecute sólo la carpeta `Run a collection` para asegurarse de que sus pruebas pasan. Tómese un minuto para revisar las opciones de configuración, los resultados y el resumen de la ejecución.

Ejercicio #9 - Scripting

Utilicemos la API de usuarios aleatorios para aprender a hacer scripts en Postman.

Cree una colección de nombre `Ejercicio 9` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Añade una petición llamada `get random user` a la carpeta con los siguientes detalles:
   a. Método `GET`.
   b. `https://randomuser.me/api/` URL de la petición.

   Envía la petición para asegurar una respuesta exitosa y guarda los cambios.
2. **Añade otra petición:** Añade una segunda petición llamada `echo the user` después de la primera con los siguientes datos:
   a. Método `POST`.
   b. `https://postman-echo.com/post` URL de la petición.
   c. Cuerpo de la petición JSON con el formato descrito al final.

```
{
  "name": "Dawn Ellis",
  "email": "dawn.ellis@example.com",
  "id": "334eecea-607d-4111-90ee-e57bea971654"
}
```

3. **Encadene las solicitudes:** Vuelva a la pestaña *Tests* de la primera solicitud para capturar los datos necesarios para la segunda solicitud. Utilice esta vez las variables de la colección. Recorra el flujo de trabajo ejecutando cada solicitud de una en una para asegurarse de que está obteniendo y configurando las variables adecuadamente y pasando los datos correctamente.
4. **Ejecute la colección:** Una vez que crea que lo tiene, ejecute esta carpeta usando el collection runner para asegurarse de que no hay errores en la carpeta. Si quieres ejecutar toda la colección en lugar de la carpeta, añade `postman.setNextRequest(null)` en la pestaña *Tests* de la segunda petición para evitar que la ejecución de la colección continúe con la siguiente carpeta. Recuerde guardar los cambios.

Ejercicio #10 - Visualizations

Utilicemos la PokéAPI para aprender a visualizar los datos de respuesta en Postman.

Cree una colección de nombre `Ejercicio 10` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Agrega una nueva solicitud llamada `visualizer` a la carpeta `Visualizations` con los siguientes detalles.
   a. Método `GET`.
   b. `https://pokeapi.co/api/v2/type` URL de la solicitud.

   Envíe la solicitud e inspeccione la respuesta. También podemos formatear los datos de la respuesta de manera que nos ayude a entender estos datos usando visualizaciones.
2. **Visualiza la respuesta:** En la pestaña *Test*, utiliza el método `pm.visualizer.set()` para inicializar una cadena de plantilla HTML y pasar los datos de la respuesta. Renderiza una tabla de dos columnas mostrando los nombres y las URLs de cada

tipo de Pokemon mediante un bucle sobre un array. Los títulos de cada columna deben ser `Name` y `URL` respectivamente. Si necesitas una pista, mira el código de ejemplo en la documentación.

3. **Dale estilo a la tabla:** Utiliza cualquier método de estilo que prefieras, pero por favor haz que el color de fondo de la tabla sea precisamente `#FFFFFF`. Puede añadir cualquier otro estilo personal que desee. Asegúrese de que la solicitud devuelve una respuesta satisfactoria y guarde los cambios.

Este era un ejemplo sencillo de representación de una tabla. Puedes hacer visualizaciones más avanzadas trabajando con bibliotecas externas, e incluso incluyendo interacciones de JavaScript. Si no sabes por dónde empezar, navega por la Red de APIs de Postman para encontrar plantillas de inicio o echa un vistazo a algunos tutoriales de frameworks populares que aparecen en la documentación de esta colección.

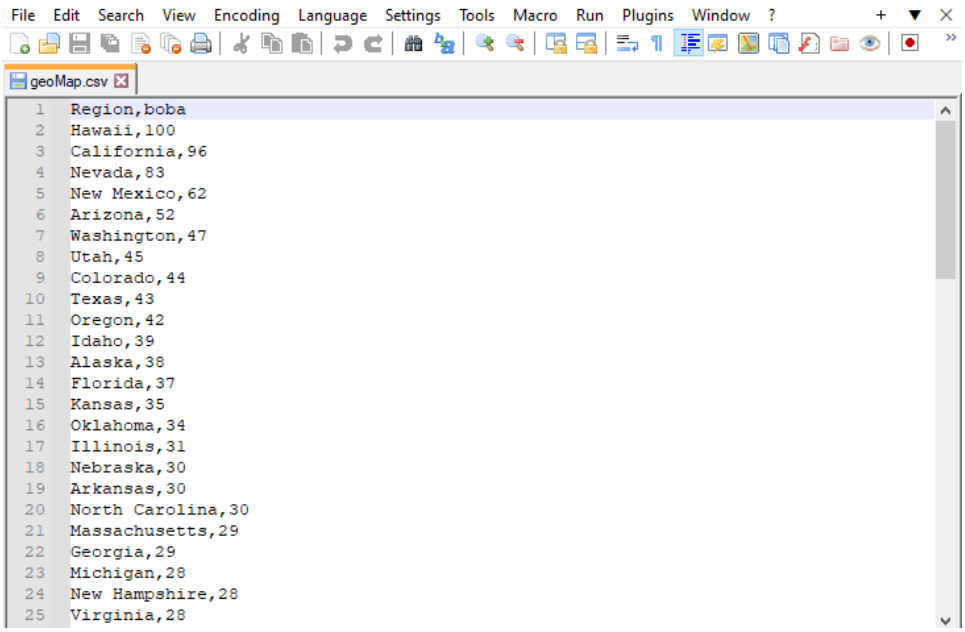
Ejercicio #11 - Documentation

Cree una colección de nombre `Ejercicio 11` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Documentar una solicitud:** Añade una única petición a la carpeta `Documentation`. Puede ser una solicitud de una API que hayamos utilizado anteriormente o cualquier API que te guste. Estas descripciones proporcionan más información sobre una colección, una carpeta, una solicitud, un parámetro o una cabecera y están formateadas en Markdown. Incluya lo siguiente para explicar cómo utilizar la solicitud:
    a. Descripción de la solicitud formateada en Markdown.
    b. Incluya al menos un hipervínculo a la documentación oficial en la descripción de la solicitud.
    c. Descripción de los parámetros si añade algún parámetro.
    d. Descripción de la cabecera si añade alguna cabecera.
    e. Guarde una respuesta satisfactoria del servidor como ejemplo.

Ejercicio #12 - Data files

Vamos a realizar una colección utilizando los datos de un archivo. Completaremos este reto utilizando un archivo de datos sobre las bebidas boba. Exporta tu propio archivo de Google Search Trends sobre el volumen de búsquedas de "boba" por estado. El archivo debe tener el siguiente formato:



Cree una colección de nombre `Ejercicio 12` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Añade una petición llamada `boba` a la carpeta `Data files` con los siguientes detalles:

   a. Método `GET`.

   b. `https://postman-echo.com/get` URL de la petición.

   La API Echo de Postman se hace eco de lo que enviamos en la petición. Vamos a recuperar la información del archivo de datos para enviarla a esta API.

2. **Acceder a los datos en los campos de texto:** Añadir un parámetro de consulta llamado `{{region}}`, con un valor llamado `{{boba}}`. Estas variables no se definirán hasta que hagamos un bucle en el archivo de datos.

3. **Acceda a los datos en un script:** En la pestaña de *Pre-request Script*, registre un log de su archivo de datos para ambos parámetros.
4. **Ejecute la carpeta:** Ejecute la carpeta `Data files` en el collection runner utilizando los datos del archivo CSV. Verifique que su script de pre-solicitud, los parámetros de la consulta y la prueba están extrayendo los datos del archivo de datos como se esperaba.

Ejercicio #13 - Dynamic request bodies

Cree una colección de nombre `Ejercicio 13` y dentro de este, una carpeta con el nombre del ejercicio.

1. **Añade una petición:** Añade una petición llamada `color` a la carpeta `Dynamic request bodies` con los siguientes detalles:
   a. Método de solicitud `POST`.
   b. `https://postman-echo.com/post` URL de la petición.
   c. Crea una variable de colección y añádela cuerpo de la petición formateado raw como JSON como `{{payload}}`.
2. **Establezca dinámicamente la carga útil:** En la pestaña *Pre-request Script*, establezca una variable (del tipo que prefiera) llamada `payload` para enviar en el cuerpo de la solicitud. El payload debe consistir en las siguientes partes:
   a. Generar dinámicamente un código hexadecimal aleatorio utilizando la biblioteca faker.
   b. Usar `pm.sendRequest()` para hacer una petición `GET` al endpoint de la API de color .

   Obtener el valor rgb y el nombre del código hexadecimal aleatorio, para esto deberá transformar la respuesta en JSON para poder construir la variable. Construya el `payload` completo en el script de pre-request para que la variable payload pueda ser enviada como `{{payload}}` en la pestaña *Body*. Debería tener este aspecto:

```
{
 hex: "#170d58",
 rgb: "rgb(23, 13, 88)",
 name: "Violent Violet"
}
```

Asegúrate de que la carga útil se envía como esperas. Enviar la solicitud de nuevo debería cambiar la carga útil dinámicamente. Recuerde que puede consultar la consola de Postman para depurar comportamientos inesperados.

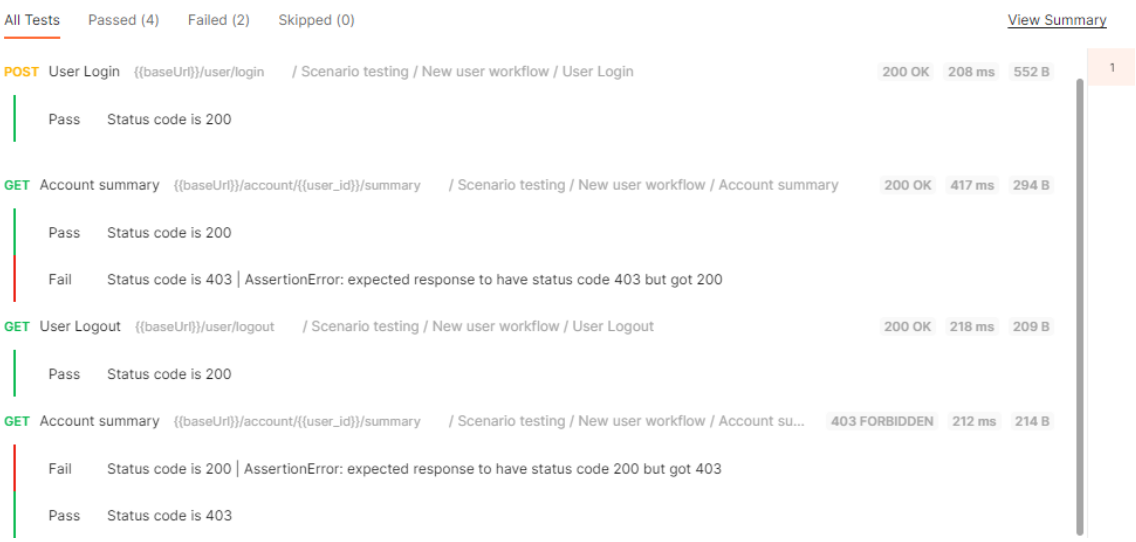
Ejercicio #14 - Scenario testing

1. **Cree una API y una colección:** Importe esta especificación de la API y genere una colección a partir de la especificación. Una vez completado este paso, asegúrese de que tiene una API y una colección asociada, ambas llamadas `Good Bank API`.
2. **Añada carpetas:** Añade una nueva carpeta llamada `New user workflow` anidada bajo la carpeta `Scenario testing`.
3. **Añadir URL:** Encuentre la `baseUrl` utilizada en la colección `API de Good Bank`, y añádala como una variable de colección llamada `baseUrl` a la colección `Good Bank API`.
4. **Configure un nuevo usuario:** Busque la solicitud `POST` *Create User* en la colección de la `API de Good Bank` y duplíquela en la carpeta `Setup`.
   En la pestaña *Body*, actualice el nombre de usuario y la contraseña de su nuevo usuario. Anote esta información utilizando el método que prefiera. Pulse *Enviar* para crear un nuevo usuario. La respuesta contendrá un `user_id`. Cree una variable de colección llamada `user_id` con esta información, utilizando el método que prefiera.
5. **Añada un escenario:** Encuentre las siguientes solicitudes en la colección de la `API de Good Bank`, y duplíquelas en la carpeta `New user workflow`.
   a. `POST` User Login.
   b. `GET` Account summary.
   c. `GET` User Logout.

Encuentra la solicitud de inicio de sesión de usuario, y actualiza el cuerpo de la solicitud con el `username` y `password` del nuevo usuario que creaste en el paso anterior. Añade una prueba de Postman para verificar una respuesta exitosa. A continuación, añade algo de código en la pestaña *Tests* para establecer dos variables de colección (`user_id` para almacenar el id del usuario conectado y `token` para almacenar el token de sesión del usuario conectado). Envíe la solicitud de nuevo para verificar que las variables se establecen correctamente.

6. **Añadir autorización:** Establezca un tipo de autorización de `clave de API` para la carpeta `New user workflow` utilizando la variable de nuevo `token`. Esta API en particular requiere que se envíe un token como una cabecera de solicitud llamada `x-api-key`. Cada solicitud dentro de la carpeta `New user workflow` debe heredar la autorización de la carpeta.
7. **Continúe con el escenario:** Agregue una prueba a cada solicitud dentro del `New user workflow` para verificar una respuesta exitosa.
    a. `GET` Account summary - Actualice el parámetro de ruta para utilizar su nueva variable `user_id`.
    b. `GET` User Logout.
    c. `GET` Account summary - Añade dos pruebas una con respuesta satisfactoria `200` y otra con respuesta para verificar un código de estado HTTP `403`, ya que después de que el usuario cierre la sesión, no debería poder acceder a su resumen de cuenta.
8. **Ejecutar la carpeta:** Utilice el *Run* para ejecutar todas las solicitudes de la carpeta `New user workflow`. Antes de correr las pruebas añade código para llamar a solicitud Account summary luego de hacer el User Logout y comprobar que la respuesta `403` se está recibiendo. Asegúrese de que, al recibir el código de respuesta `403`, se detenga el flujo añadiendo `postman.setNextRequest(null)`.

    La respuesta luego de correr la carpeta debería mostrarse como la siguiente imagen, en donde se ven las fallas de la solicitud Account summary dependiendo de la respuesta obtenida.


All Tests
Passed (4)
Failed (2)
Skipped (0)
View Summary
POST User Login {{baseUrl}}/user/login / Scenario testing / New user workflow / User Login 200 OK 208 ms 552 B
Pass Status code is 200
GET Account summary {{baseUrl}}/account/{{user_id}}/summary / Scenario testing / New user workflow / Account summary 200 OK 417 ms 294 B
Pass Status code is 200
Fail Status code is 403 | AssertionError: expected response to have status code 403 but got 200
GET User Logout {{baseUrl}}/user/logout / Scenario testing / New user workflow / User Logout 200 OK 218 ms 209 B
Pass Status code is 200
GET Account summary {{baseUrl}}/account/{{user_id}}/summary / Scenario testing / New user workflow / Account su... 403 FORBIDDEN 212 ms 214 B
Fail Status code is 200 | AssertionError: expected response to have status code 200 but got 403
Pass Status code is 403
1