## Lab 09: Explore REST APIs with API Simulator and Postman

### Case Study

Before adopting automation tools, LearnIT Hub, an online education platform, faced difficulties in testing and teaching API integration due to limited internet connectivity and reliance on unstable third-party APIs. Developers struggled to consistently test HTTP methods like GET, POST, and DELETE, which led to slower learning, broken workflows, and an inability to confidently develop and deploy API-based solutions. Instructors also found it hard to deliver hands-on training when real-time API access was unavailable or unreliable.

After integrating the School Library API simulator within the DEVASC VM, the organization experienced a major improvement in both training and development. This offline-ready environment enabled developers to safely explore RESTful APIs, make calls using Postman, and automate tasks with Python scripts—all without needing internet access. It allowed the team to test real-world scenarios, understand API request-response cycles, and build skills in consuming and scripting APIs with confidence. The result was a faster development process, deeper technical understanding, and improved API-driven project delivery.

### Business Challenge

LearnIT Hub struggled with consistent API testing and education due to its reliance on external services. Developers could not reliably test or learn key REST methods like GET, POST, PUT, and DELETE, especially when offline. This slowed down development cycles, limited team skill-building, and created barriers to implementing automation in projects. To solve these issues, the organization introduced a local API simulator and brought you in as a certified DevNet Associate to lead the transition toward structured, offline API exploration and automation.

### Solution

In this lab, you will use the School Library API simulator to understand REST API functionality and test endpoints using Postman and Python. You will:

1. Explore API documentation using the API Simulator
2. Use Postman to make API calls (GET, POST, DELETE)
3. Use Python to add 100 books to the API Simulator

|  |  |  |
| --- | --- | --- |
| **// Explore API documentation using the API Simulator**  1. Launch the DEVASC VM. Open the **Chromium** Web Browser by double-clicking its icon on the desktop. If the School Library website does not load automatically, type **library.demo.local** in the address bar and press **Enter** to view the API documentation, which follows the OpenAPI Specification format for requests, responses, headers, and parameters. The web site defaults to the **Our Books** tab and displays a list of books. In the upper right corner where it states, **Click here for API docs**, click **here** to go to the API documentation web page.    2. You will now see a list of APIs in the **/api/v1 Default namespace**. Notice the downward arrow to the far right. Clicking anywhere on the **/api/v1** bar will minimize the API list and turn the arrow facing right. Click again on the same bar to re-display the API list.  Notice the lock to the far right of several of the APIs. The lock indicates that these APIs require a token to be used.    3. Click anywhere on the bar for the **GET /books** API. This API returns a list of books in the school library.   * **Parameters** – There are several optional API parameters. These can be used to filter, sort, or paginate the output. These will be referred to later in this lab. * **Response content type** – Click **application/json** to see a list of the different types of data formats the information can be viewed. Leave the selection as **application/json**. * **Code** – The code displays 200 by default, which indicates the API request from the sever was a success as displayed in the **Description**. (You have not sent an API request yet.)       4. One of the powerful features of the OpenAPI Specification is the ability to test API calls directly within the documentation to verify if they are constructed correctly and to review the expected responses. This testing capability is also present in API documentation from Cisco, MapQuest, and other organizations that use OpenAPI Specification.  Click the **Try** **it out** button in the GET /books section of the API documentation, leave the optional parameters blank, and click **Execute**.    5. In the **Responses** section, you will see the **Curl** command that can be used to access the same endpoint, along with the **Request** **URL**, which can be reused with curl, Postman, or Python to send the same request.   * **Code**: This is the HTTP response code. 200 indicates a successful call. * **Response body**: List of books in JSON format. * **Response headers**: Information about the API returned from the server.     6. In the **Response body** you will see a list of books in JSON format.    7. The GET /books API shows how to access the content in the response body using **curl**, a command-line tool used to transfer data to or from a server over supported protocols such as HTTP and HTTPS. Copy the command **curl -X GET "http://library.demo.local/api/v1/books" -H "accept: application/json"**, to copy it to your clipboard.  8. Open a terminal window. Right-click and **Paste** the contents from the clipboard into the terminal and press Enter. Notice this provides the same information as the library’s OpenAPI interface.    9. To list books along with their ISBN using the GET /books API, return to the School Library API website’s GET /books section. In the **Parameters** section, click the dropdown arrow next to the **includeISBN** parameter and select **true**, then click **Execute**.    10. In the **Responses** section, notice that the **curl and Request URL** command now includes the parameter for ISBN, indicating that ISBN information will be included in the API response.    11. Click the **POST /loginViaBasic** API to begin the login process. Since there are no parameters required, click **Try it out**, then click **Execute**.    12. A sign-in prompt will appear, enter the following credentials: **Username**: cisco, **Password**: Cisco123! And click **Sign in**.    13. The authentication token will appear in the **Response** body. Select the content between the quotation marks, right-click, and choose **Copy** to copy the token to your clipboard. Note that your token will be unique and different from the example shown.    14. Scroll up to the top of the School Library API page and click the green **Authorize** button. The **Available authorizations** dialogue box will appear.    15. Right-click and **Paste** the copied token into the **Value** field next to **X-API-KEY**, then click **Authorize**. This token, along with the header name **X-API-KEY**, will be used later in Postman. Close the **Available authorizations** dialog box and return to the list of APIs.    16. You will notice that the lock icons next to several APIs have changed, indicating that those APIs are now available for use. Click the bar for **POST /loginViaBasic** again to close that section.    17. To add books using the **POST /books** API, click on **POST /books**. Under **Parameters**, note that a payload is required—this means you must provide input data in JSON format, as specified by the parameter’s content type. Click **Try it out.**    18. Update the values for **id, title**, and **author** with the required information. Click on **Execute**.    19. Verify that the POST request was successful by checking the **Server response**. A **Status Code 200** indicates the post was successful. In the **Response body**, you should see the details of the book you added, including a new **id**. The **curl** command and **Request URL** will also reflect the updated request. To add another book, update the values for **id**, **title**, and **author** with the new information provided below, and click **Execute** again.  **Note**: If you got a **401** code, check the **Response body** text. Most likely you received an **“error”: “Invalid API key”** response. This is because you did not enter all the characters for your API key. Or possibly, you add an unnecessary space. Return to the previous step and repeat the authorization process.  20. You can verify the books were added to the **Our Books** Return to the **School Library** tab in your browser (http://library.demo.local) and refresh the page. Be careful not to close the School Library API tab. If you do, then you will need to reauthenticate.    21. Return to the **School Library API** tab in the browser. Click the **GET /books.**  Click **Try it out**. If you see **Cancel** button in red, then you are already in **Try it out.** Click **Execute**. Under **Server response** in the **Response body**, you will now see the two books you added. Notice they each have a unique **id**.    22. Click the **GET /books{id}** API, which requires an id parameter. To the right of **Parameters**, click **Try it out**. Under **Parameters**, enter 4 for the required id, then click **Execute**. Observe the **Curl** and **Request URL** sections:  **Curl** – Shows the command to perform the same GET request using curl.  **Request URL** – This URL can be used to retrieve the same information using Postman or Python.    23. Verify the request was successful by checking the **Server response**. A **Status Code 200** indicates success. In the **Response body**, you will see the details of the book with the id of 4.    24. Click the **DELETE /books{id}** API, which requires an id parameter to specify the book to be deleted. Click **Try it out**, then enter 4 under **Parameters**. Click **Execute** to send the request. Verify the deletion was successful by checking the **Server response**—a **Status Code 200** confirms the operation was successful. In the **Response body**, you will see the details of the book that was deleted, including the id value of 4.    25. Click the **GET /books** API to list all available books. Click **Try it out** (if a red **Cancel** button is visible, you are already in Try it out mode), then click **Execute**. In the **Server response**, under the **Response body**, you will notice that the book with **id 4** is no longer listed, confirming that it was successfully deleted.    **// Use Postman to Make API Calls to the API Simulator**  1. Double-click the **Postman** icon on the desktop. Normally, you would sign in to Postman. However, it is not necessary to get an account and login to Postman for labs in this course.    2. To retrieve the book list using the **GET /books API**, click the **plus icon** in Postman to open a **new request tab**. Ensure the request method is set to **GET**. If it is not, click the dropdown and change it to **GET**.    3. Next, go to the **Chromium browser**, open the **School Library API documentation**, and copy the URL for the **GET /books** endpoint: [**http://library.demo.local/api/v1/books**](http://library.demo.local/api/v1/books) **and** paste this URL into the **Enter request URL** field in Postman. **Remove any extra characters or lines** that may have been copied along with the URL. Now click the **Send** button.    4. If successful, the response section will display a **status of 200 OK in green**, along with a **JSON body** showing the list of books available in the system. Notice that the default is **Pretty** and **json.**    5. In the main window, click the plus icon **+** to create a new **Untitled Request**. Click the down arrow next to **GET** and select **POST**. Enter request URL. Return to the **School Library API** tab in Chromium and expand the **POST /loginViaBasic** API, if necessary. Under **Request URL**, select, right-click and **Copy** the URL to your clipboard: **http://library.demo.local/api/v1/loginViaBasic** and paste it into the request field.  **Note**: If the **Request URL** is no longer showing, then you probably closed and re-opened the **School Library API** documentation page and are no longer authenticated. Click **Try it out**, then **Execute**, and then re-authenticate with username **cisco** and password **Cisco123!**.  6. Return to **Postman**and paste the URL next to POST where it states, **Enter request URL**. Click **Authorization**. Within this area, in the drop-down list for **Type**, choose **Basic Auth**. For the **Username** enter **cisco** and in **Password** enter **Cisco123**! and click **Send**.  7. You can scroll down to the **Body** section to see your new token**.**Your token will be different than the one shown here.    8. Now add the **Learning DevNet** book that you deleted in Part 2 using the Try it out feature in the School Library API documentation. In the main window, click the plus icon **+** to create an Untitled Request. Click the down arrow next to GET and select **POST**. Return to the School Library API tab in Chromium and expand the POST /books. Under Request URL, select, right-click, and Copy the URL to your clipboard:  **http://library.demo.local/api/v1/books.**  **Note**: If the **Request URL** is no longer showing, then you probably canceled **Try it out**. Click **Try it out**, and then **Execute** to show the **Request URL**.  9. Return to **Postman**and paste the URL next to POST where it states, **Enter request URL**. Click **Authorization**. In the drop-down list for **Type**, choose **API Key**. In the **Key** field, enter **X-API-KEY**.  **Note**: Recall that you saw **X-API-KEY** in the School Library API web page when you got a token selecting the green **Authorize** button.  10. Return to the **Post** tab in Postman and copy the token you received in Step 3. Be sure to include everything within the quotation marks. Your token will be different than the one shown here.  Example: cisco|5xSUHYFDvIAoCRv0LqWVSDcjJAwWjg18vMml6u2lm1I  11. Go back to the second Post tab in Postman. Paste the token in the Value field.    12. In the same row with the Authorization tab, click Body. This section will allow you to choose the format of your input. Click the **raw** radio button. Then, click **Text** and change this option to **JSON**.    13. In the input area you will see the number 1, for “line 1”. Enter the following JSON object and click **Send**.   |  | | --- | | {  "id": 4,  "title": "Learning DevNet",  "author": "Alicia",  "isbn": "978 158144778"  } |     14. To verify that the API request was a success, you will now see a response that include the **Status** code 200 OK in green.  **// Use Python to Add 100 Books to the API Simulator**  1. You could use the OpenAPI Specification Try It tool or Postman to add as many books as you want. However, you would have to add them one at a time. A better solution would be to write a program to add the books. In this Part, you will simulate the process of adding 100 books by using the Python **faker** library.  2. Open **VS Code** from the **Menu** button or by double-clicking the icon on the desktop. Click **File > Open Folder…**, navigate to the **labs/devnet-src/school-library** folder, and click **OK**. In VS Code **EXPLORER** pane on the left, click **py** to open it, if necessary.  3. At the top, notice the “shebang” that sets the interpreter to Python 3 and then the three libraries that are imported.  4. This Python script **automates the addition of 100 books** to the **School Library API simulator** by using the **Faker library** to generate **random book titles, authors, and ISBNs**, and the **requests library** to send **POST requests** to the API. It starts by calling **getAuthToken()** to authenticate using predefined **login credentials** **(cisco / Cisco123!)** and retrieves an **API token**. Then, a loop runs from **ID 4 to 103**, and in each iteration, it generates fake book data using **fake.catch\_phrase(), fake.name(),** and **fake.isbn13(),** forms a JSON object, and sends it using the **addBook()** function. Each book is added by sending a **POST request** to the **/api/v1/books** endpoint with the proper **headers and token**. Upon success, each book’s data is printed in the terminal as confirmation.   |  | | --- | | #!/usr/bin/env python3  import requests  import json  from faker import Faker  APIHOST = "http://library.demo.local"  LOGIN = "cisco"  PASSWORD = "Cisco123!"  def getAuthToken():  authCreds = (LOGIN, PASSWORD)  r = requests.post(  f"{APIHOST}/api/v1/loginViaBasic",  auth=authCreds  )  if r.status\_code == 200:  return r.json()["token"]  else:  raise Exception(f"Status code {r.status\_code} and text {r.text}, while trying to Auth.")  def addBook(book, apiKey):  r = requests.post(  f"{APIHOST}/api/v1/books",  headers={  "Content-type": "application/json",  "X-API-Key": apiKey  },  data=json.dumps(book)  )  if r.status\_code == 200:  print(f"Book {book} added.")  else:  raise Exception(f"Error code {r.status\_code} and text {r.text}, while trying to add book {book}.")  # Main script logic  apiKey = getAuthToken()  fake = Faker()  for i in range(4, 104): # Add 100 books with IDs 4 through 103  fakeTitle = fake.catch\_phrase()  fakeAuthor = fake.name()  fakeISBN = fake.isbn13()  book = {  "id": i,  "title": fakeTitle,  "author": fakeAuthor,  "isbn": fakeISBN  }  addBook(book, apiKey) |     **To verify e**xecute the command **python3 add100RandomBooks.py** in the terminal to run the script. You should see output with different randomly generated titles, authors, and ISBNs. |