



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David
SWANSEA - ABERTAWE

Applied Computing

Wales Institute of Science and Art (WISA)

Network Programmability (ACCB5029)

Lab 5: Working with APIs

ICONS Used in this document:



Do research and complete the task. Need referencing/ in-text citations. Shall be included as a part of your coursework.



Record your Reflection.

 **Networking
Academy**

Task 1: (Explore and Discuss APIs)**Objectives**

1. Explore API Documentation Using the API Simulator
2. Record the outcomes.
3. Discuss with your group and record your reflections.

Task 2: (Use Postman)**Objectives**

4. Use Postman to Make API Calls to the API Simulator
5. Discuss with your group and record your reflections and outcomes.

Task 3: (Use Python to add records)**Objectives**

6. You Python to add 100 books to the API Simulator
7. Reflect your understanding on All the 3 tasks. Complete the individual outcomes.
Collaborate with the team and reflect your personal reflection on all the tasks after discussing with the group.

Required Resources

Host computer with DEVASC VM.

Instructions

Launch the DEVASC VM

1. Explore API Documentation Using the API Simulator

To understand how to make calls to a REST API, developers typically start by studying the API documentation. The format for requests, responses, headers, and parameter for REST APIs are typically documented using the OpenAPI Specification (formerly Swagger Specification).

- 1.1. Open the Chromium Web Browser.
- 1.2. Connect to the School Library Web Site. *(If the browser does not automatically open the School Library website, in the address bar type: library.demo.local and press return to go there.)*
- 1.3. Go to the API docs page.
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 and go to the API documentation web page.
You will now see a list of APIs in the /api/v1 Default namespace.
- 1.4. Notice the downward arrow to the far right.
 - 1.4.1. Clicking anywhere on the /api/v1 bar will minimize the API list and turn the arrow facing right.
 - 1.4.2. Click again on the same bar to re-display the API list.
 - 1.4.3. Notice the lock to the far right of several of the APIs. The lock indicates that these APIs require a token to be used.
- 1.5. List books using the GET /books API.
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.)

- 1.6. Use the Try it out feature in the API documentation.
One of the more powerful features of the OpenAPI Specification is the ability to test an API call to see if you constructed it correctly. You can also review the response to see if it is what you expected.



- ✓ You will see this same testing feature in API documentation for Cisco, MapQuest, and other organizations that use this OpenAPI Specification feature.
- ✓ Do detailed research on API documentation. Address questions like what API documentation is, what is OpenAPI Specification, how it is

useful, what are the documentations available, how to look/ access the documentation.

- ✓ Discuss with your group and record your personal reflection and your groups reflection on API Documentation. Try to access the documentations for Cisco, MapQuest, and other organizations that are available. Discuss and reflect your understanding.

1.6.1. In the GET /books API documentation, click the Try it out button.

1.6.2. Notice that you now have the option to enter information for the optional parameters. Leave the parameters blank and click the Execute button. In the Responses section you will see:

Curl: The curl command you can use to access the same information for the /books API.

Request URL: This URL is used in the API request, which can be used to request the same information using curl, Postman, and Python.

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.

1.6.3. What do you see in the response body?

1.7. Use the curl command in a terminal window.

The GET /books API provides information to access the content displayed in the response body using curl. curl is a command line tool to transfer data to or from a server, using any of the supported protocols including HTTP and HTTPS.

1.7.1. Select the curl command, right-click and copy it to your clipboard:

1.7.2. 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.

```
devasc@labvm:~$ curl -X GET "http://library.demo.local/api/v1/books"
-H "accept: application/json"
```

1.7.3. What output did you get?

1.8. List books with their ISBN using the GET /books API.

1.8.1. Return to the School Library API web site's GET /books API.

1.8.2. In the Parameters section, select the down arrow next to the includeISBN parameter and select true.

1.8.3. Click Execute.

Notice the following changes in Responses:

Curl now includes the parameter for ISBN.

```
curl -X GET "http://library.demo.local/api/v1/books?includeISBN=true" -H "accept: application/json"
```

Request URL now includes the parameter for ISBN.

```
http://library.demo.local/api/v1/books?includeISBN=true
```

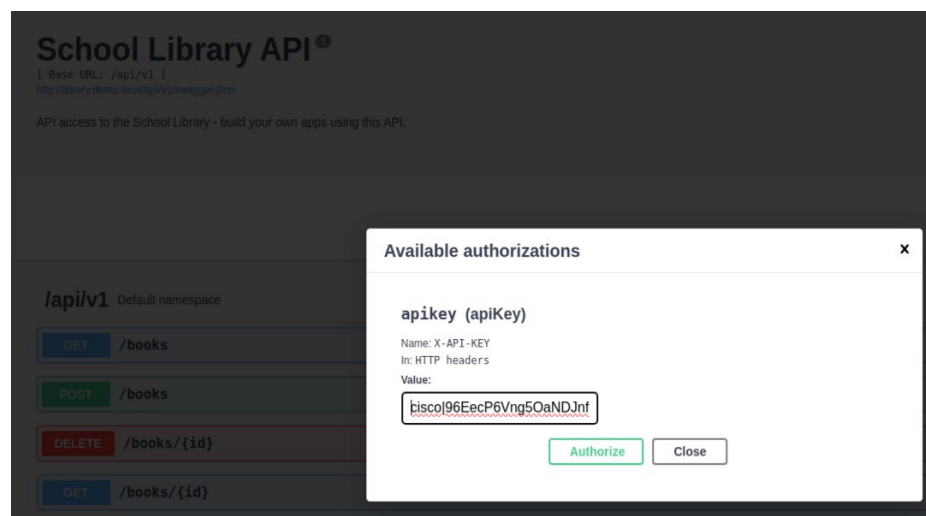
Response body has the same list of books as shown previously but now includes the book's ISBN.

Note: To minimize the scrolling, when you are done with an API you can close that specific API window by clicking anywhere on the title bar. Now you can see all the APIs more easily.

- 1.9. Get a token using the POST /loginViaBasic API.
 - 1.9.1. Click the API POST /loginViaBasic.
 - 1.9.2. Notice there are no parameters. Click Try it out, and then click Execute.
 - 1.9.3. A Sign in box will prompt you for a username & password. Enter the following information & sign in: Username: cisco | Password: Cisco123!
 - 1.9.4. The token will be displayed in the Response body. Select the information between the quotes, right-click and copy the information into your clipboard. Your token will differ from the one shown below.

```
{
  "token": "cisco|KZeQbC5iV3HKEzB7hCJ6qHqXen4rLGh72YJKeVfs"
}
```

- 1.9.5. Scroll up to the top of the School Library API page and click the green Authorize button. The Available authorizations dialogue box will appear.
- 1.9.6. Right-click and Paste the token after Value and click Authorize. Notice the Name is X-API-KEY. This information along with the token will be used later in Postman.



- 1.9.7. Close the Available authorizations dialogue box and return to the list of APIs. Notice the locks by several of the APIs have now changed. These APIs are now available for you to use.

- 1.10. Add books using the POST /books API.
 - 1.10.1. Click the API POST /books.
 - 1.10.2. Notice under Parameters that the payload is required. This means that this API requires information for this parameter in the format specified by the Parameter content type, which is JSON.
 - 1.10.3. Click Try it out.
 - 1.10.4. Modify the id, title and author with the information shown below.

```
{  
  "id": 4,  
  "title": "Network Programmability",  
  "author": "Your Name"  
}
```

- 1.10.5. Click Execute.
- 1.10.6. Verify that the post was successful in the Server response. A Code of 200 means the post was a success. You should see the book you added in the Response body along with a new id. You will also see updated information for curl and the Request URL.
- 1.10.7. To add another book, modify the id, title and author with the information shown below.

```
{  
  "id": 5,  
  "title": "1 week after reading week",  
  "author": "Your Lecturer Name"  
}
```

- 1.10.8. Click Execute.
- 1.10.9. Verify that the post was successful in the Server response. A Code of 200 means the post was a success. You should see the book you added in the Response body along with a new id. You will also see updated information for curl and the Request URL.

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.

- 1.10.10. Click the bar for the API POST /books to close the window.

You can verify the books were added to the Our Books page. 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.

- 1.11. List books using the GET /books API. Return to the School Library API tab in the browser.
 - 1.11.1. Click the GET /books API.
 - 1.11.2. Click Try it out. If you see Cancel button in red, then you are already in Try it out mode.
 - 1.11.3. Click Execute.
 - 1.11.4. Under Server response in the Response body, you will now see the two books you added. Notice they each have a unique id.
 - 1.11.5. Click the bar for the GET /books API to close the window.
- 1.12. List a specific book using the GET /books{id} API.
 - 1.12.1. Click the GET /books{id} API. Notice this API requires the id as a parameter.
 - 1.12.2. To the right of Parameters, click the Try it out button.
 - 1.12.3. Under Parameters, enter 4 for the required id.
 - 1.12.4. Click Execute. Notice the information provided by Curl and Request URL.

Curl - This is the curl command to perform the same function using curl.

Request URL - This is the URL that can be used to get the same information using Postman and Python.

Verify that the get was successful in the Server response. A Code of 200 means the post was a success. In the Response body you will see the book you requested with the id of 4.

What is your outcome?

- 1.12.5. Click the bar for the GET /books{id} API to close the window.
- 1.13. Delete a specific book using the DELETE /books{id} API.
 - 1.13.1. Click the DELETE /books{id} API. Notice this API requires the id as a parameter.
 - 1.13.2. Click Try it out.
 - 1.13.3. Under Parameters, enter 4.
 - 1.13.4. Click Execute.
 - 1.13.5. Verify that the delete was successful in the Server response. A Code of 200 means the post was a success. In the Response body you will see the book you deleted with the id of 4.

What is your outcome?

- 1.13.6. Click the bar for the DELETE /books{id} API to close the window.
- 1.14. List books using the GET /books API.
 - 1.14.1. Click the GET /books API.
 - 1.14.2. Click Try it out. If you see Cancel button in red, then you are already in Try it out mode.
 - 1.14.3. Click Execute.
 - 1.14.4. Under Server response in the Response body, you will no longer see the book with id or 4.

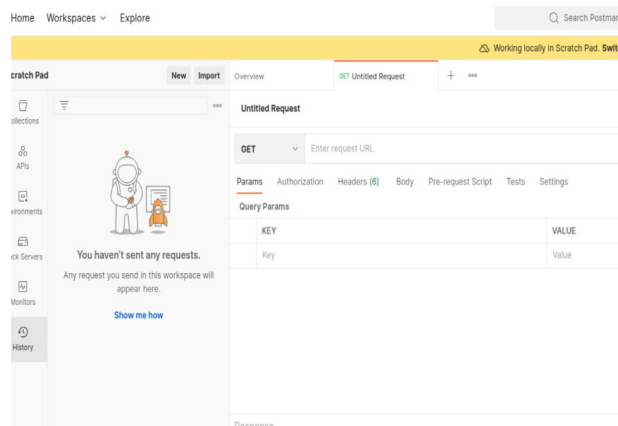
Note: Do not close the School Library API tab in the Chromium browser. You will use the API documentation in the next part.

Task 2: Use Postman to Make API Calls to the API Simulator

In this Part, you will use Postman to make the same API calls you made in the Student Library API documentation. Postman is a useful tool when an API developer web site is not available while providing the ability to easily save, organize, and reuse APIs.

2. Open Postman (Double-click the Postman icon on the desktop. Normally, you would sign into Postman. However, it is not necessary to get an account and login to Postman for labs in this course.)

- 2.1. List the books using the GET /books API.
- 2.2. In the main window next to the Launchpad tab, click the plus icon "+" to create an Untitled Request. By default, this will be a GET request.



- 2.3. Click the down arrow next to GET to view the different API operations including GET, POST, and DELETE. Leave the selection on GET. Click the up arrow next to GET to close the list.

- 2.4. Enter request URL.

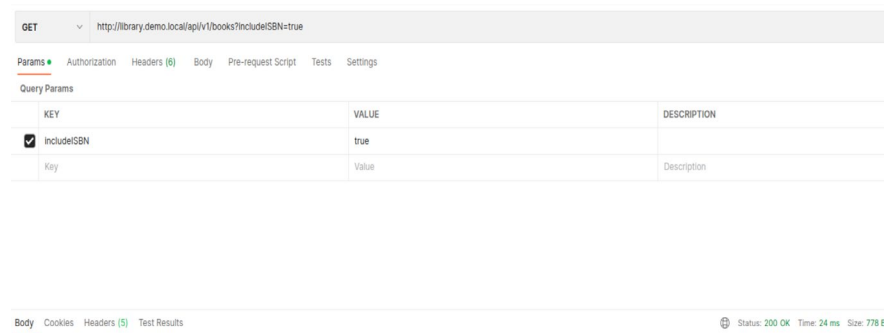
- 2.4.1. Return to the School Library API tab in Chromium and, if necessary, expand the GET /books API.

- 2.4.2. Under Request URL, select, right-click and Copy the URL to your clipboard:

- 2.4.3. Return to Postman and paste the URL next to GET where it states, "Enter request URL".

Note: If pasting adds a line below the URL, remove the extra line.

- 2.4.4. Click Send. To verify that the API request was a success, you will now see a response that include the Status code 200 OK in green. Scroll down to the Body section to see the response. Notice that the default is Pretty and json.



Note: You can save the JSON output to a file using the Save Response button above the output. This is not required for this lab.

- 2.5. Get a Token using the POST /loginViaBasic API. In the main window, click the plus icon "+" to create a new Untitled Request.

- 2.5.1. Click the down arrow next to GET and select POST.

- 2.5.2. Enter request URL.

- 2.5.2.1. Return to the School Library API tab in Chromium and expand the POST /loginViaBasic API, if necessary.

- 2.5.2.2. Under Request URL, select, right-click and Copy the URL to your clipboard:

<http://library.demo.local/api/v1/loginViaBasic>

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!.

- 2.5.2.3. Return to Postman and paste the URL next to POST where it states, "Enter request URL".

Note: If pasting adds a line below the URL, remove the extra line.

- 2.5.3. Click Authorization. Within this area, complete the following:

- 2.5.3.1. In the drop-down list for Type, choose Basic Auth.

- 2.5.3.2. For the Username and Password fields, fill in the following:
Username: cisco | Password: Cisco123!

- 2.5.4. Click Send.

- 2.5.5. If necessary, scroll down to the Body section to see your new token.
Your token will be different than the one shown here.

2.5.6. What is your new token?

- 2.6. Add a book using the POST /books API. Now you will add the No.4 book **Network Programmability** you deleted in Part 2 when using the Try it out feature in the School Library API documentation.

- 2.6.1. In the main window, click the plus icon "+" to create an Untitled Request.

- 2.6.2. Click the down arrow next to GET and select POST.

- 2.6.3. Enter request URL.

- 2.6.3.1. Return to the School Library API tab in Chromium and expand the POST /books API.

- 2.6.3.2. 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 cancelled "Try it out". Click Try it out, and then Execute to show the **Request URL**.

- 2.6.3.3. Return to Postman and paste the URL next to POST where it states, "Enter request URL".
Note: If pasting adds a line below the URL, remove the extra line.
- 2.6.4. Click Authorization. Within this area, complete the following:
 - 2.6.4.1. In the drop-down list for Type, choose API Key.
 - 2.6.4.2. 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.
 - 2.6.4.3. Return to the Post tab in Postman and copy the token you received in 2.5.6. Be sure to include everything within the quotation marks.
 - 2.6.4.4. Go back to the second Post tab in Postman. Paste the token in the Value field
- 2.6.5. In the same row with the Authorization tab, click Body. This section will allow you to choose the format of your input.
 - 2.6.5.1. Click the raw radio button.
 - 2.6.5.2. Click Text and change this option to JSON.

In the input area you will see the number 1, for "line 1". Enter the following JSON object.

```
{
  "id": 4,
  "title": "IPv6 Fundamentals",
  "author": "Rick Graziani",
  "isbn": "978 158144778"
}
```
- 2.6.6. Click Send.
- 2.6.7. To verify that the API request was a success, you will now see a response that include the Status code 200 OK in green.
- 2.7. Verify the additional book with the Get /books API. Return to the first GET tab. As you can see, Postman makes it easy to switch between different API calls.
 - 2.7.1. Click Send.
 - 2.7.2. To verify that the API request was a success, you will now see a response that include the Status code 200 OK in green.
 - 2.7.3. Click Body to see the response. Notice that the default is Pretty and json. (You must be able to see the added book here)
- 2.8. Use additional parameters with the Get /books API.
 - 2.8.1. Go to the School Library API web site. Scroll up to GET /books API and expand it, if necessary.
 - 2.8.2. Notice the parameters that are available:

includeISBN: Includes in the results the ISBN numbers. Default=false

sortBy: Sort results using the specified parameter. Default=id

author: Return only books by the given Author.

page: Used to specify a page number.

2.8.3. Click Try it out. If you see a Cancel button in red, then you do not need to select this button.

2.8.4. Under parameters:

2.8.4.1. Click includeISBN and select true

2.8.4.2. Click sortBy and select author

2.8.5. Click Execute.

2.8.6. In the Response body you will see the list of books now sorted by author and including the ISBNs. Notice that the Request URL now includes the parameters. You will see this again in Postman.

<http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author>

2.8.7. Return to Postman and go to the first API tab, GET <http://library.demo.local/api/v1/books>. You will now include some of the parameters from the School Library API web site.

2.8.8. Click Params. You will see under Query Params input boxes for KEY and VALUE. Enter the following information:

2.8.8.1. Under KEY, enter includeISBN and under Value enter true. Notice a check mark will automatically be included to the left of the value and a new row added.

2.8.8.2. Under KEY, enter sortBy and under Value enter author.

Notice that when entering these query parameters, it has updated the original URL next to the GET. This is the same Request URL you saw in the School Library API web site for this same API call. This is the URL Postman will be using, with these query parameters when making the API call.

<http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author>

2.8.9. Click Send. Notice in the Body, it now shows the same list of books, sorted by author and including the ISBNs that you saw in the School Library API web site.

2.9. Record your Reflection: Discuss with your team on this task. If needed, try out additional features you wish or you discussed with the team. What is Postman? How do you feel the use of Postman? Do a research and find out other relevant tools like Postman.

Task 3: Use Python to Add 45 Books to the API Simulator

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**.

3. Open Visual Studio (VS) Code and navigate to the school-library directory.

- 3.1. Investigate the libraries used by add100RandomBooks.py program. In VS Code EXPLORER pane on the left, click add100RandomBooks.py to open it, if necessary.
- 3.2. At the top, notice the “shebang” that sets the interpreter to Python 3 and then the three libraries that are imported.

```
#!/usr/bin/env python3
```

```
import requests
```

```
import json
```

```
from faker import Faker
```

- 3.3. You will use the Python requests library throughout this course. The requests library is required if you want to use Python to make API requests using GET, POST, DELETE and other HTTP methods.
- 3.4. Faker is a Python library that generates 'fake' data for you. This program uses the Python faker library to generate random book titles, authors, and ISBNs.
- 3.5. Do research and discuss with your group for more information on faker library. However, complete the following steps to see all the 252 methods for the faker library.
 - 3.5.1. Open a terminal window and start Python 3.
 - 3.5.2. From faker import the Faker() module.
 - 3.5.3. Assign the Faker() module to fake.
 - 3.5.4. To see all the methods, enter fake. and then press the tab key twice. Notice the method fake.name(), which you will use in the next step. In the next step and later in this lab, you will also use the three highlighted methods (prefaced with fake.): catch_phrase(), isbn13(), and name().

```
devasc@labvm:~$ python3
Python 3.8.2 (default, Apr 27 2020, 15:53:34)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from faker import Faker
>>> fake=Faker()
>>> fake.
Display all 252 possibilities? (y or n)
fake.add_provider(                                fake.longitude(
fake.address(                                       fake.mac_address(
fake.am_pm(                                         fake.mac_platform_token(
fake.android_platform_token(                       fake.mac_processor(
fake.ascii_company_email(                          fake.md5(
fake.ascii_email(                                  fake.military_apo(
fake.ascii_free_email(                             fake.military_dpo(
fake.ascii_safe_email(                             fake.military_ship(
fake.bank_country(                                 fake.military_state(
fake.bban(                                          fake.mime_type(
fake.binary(                                        fake.month(
fake.boolean(                                       fake.month_name(
fake.bothify(                                       fake.msisdn(
fake.bs(                                             fake.name(
fake.building_number(                               fake.name_female(
fake.cache_pattern                                fake.name_male(
```

3.6. Practice generating random data using the faker library.

- 3.6.1. Enter the following to generate a fake name. Your output will be a different fake name each time you execute the command.

```
>>> print('My name is {}'.format(fake.name()))
My name is Robert Foster.
>>> print('My name is {}'.format(fake.name()))
My name is Benjamin Salazar.
>>> print('My name is {}'.format(fake.name()))
My name is Mrs. Charlotte Rojas.
```

3.6.2. Use appropriate commands to print an output like this:

My name is somename and I wrote "something" (ISBN some numbers).

Later in the program, a loop is used to iterate through these three methods to create entries for the School Library.

- 3.6.3. Enter the following to generate 10 random names. After the "..." you will need to press return a second time.

```
>>> for i in range(10):
...     print(fake.name())
...
David Wilkinson
Katherine Schmidt
David Mcdaniel
Joshua Griffin
Katrina Kerr
Michelle Nguyen
Brandy Barnes
Richard Giles
Troy Smith
Stephanie Davis
```

- 3.6.4. Quit the Python interpreter when done investigating the faker library.

3.7. Review the function variables.

- 3.7.1. The two functions in the program use three variables to get an authorization token from the School Library API service.

```
APIHOST = "http://library.demo.local"
```

```
LOGIN = "cisco"
```

```
PASSWORD = "Cisco123!"
```

3.8. Review the getAuthToken function.

- 3.8.1. The getAuthToken function uses three variables to build a request. The r variable invokes the POST method for the loginViaBasic API and stores the token value if the call is successful. Notice the use of an f-string to build the request URL.

```
def getAuthToken():
```

```

authCreds = (LOGIN, PASSWORD)

r = requests.post(
    f"{APIHOST}/api/v1/loginViaBasic",
    auth = authCreds
)

```

- 3.8.2. If the call is unsuccessful (HTTP status code is not equal to 200), an exception is raised and printed to the terminal window. Again, notice the use of an f-string to build the exception message. You can test the exception code by changing the one of the variables in 3.7.

```

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.")

```

3.9. Review the addBook function.

- 3.9.1. Similar to the addAuthToken function, the addBook function uses the three variables shown in 3.7 to build a request. The r variable invokes the POST method for the books API. The data comes from a variable called book, which is specified in the final part of the program.

```

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)
    )

```

- 3.9.2. If the call is unsuccessful, an exception is raised and printed to the terminal window. You can test it by changing the one of the variables in 3.7.

```

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}.")
```

3.10. Review the code that invokes the two functions.

3.10.1. The addAuthToken function is invoked and the results are stored in the variable apiKey.

```
apiKey = getAuthToken()
```

3.10.2. The Faker() module is set to a variable named fake. A for loop then iterates 100 times. The i variable is used later in the loop to set the value for the id key for each new book from 4 up to and not including 104.

3.10.3. Note: If you want to keep the two previous books added previously in this lab, change range to (6, 106).

```
fake = Faker()
```

```
for i in range(4, 104):
```

3.10.4. Next, three variables hold the value of methods invoked from the Faker() module: catch_phrase(), name(), and isbn13().

```
fakeTitle = fake.catch_phrase()
```

```
fakeAuthor = fake.name()
```

```
fakeISBN = fake.isbn13()
```

3.10.5. Recall that the payload parameter for the books API requires JSON in the following format:

```
{  
    "id": 0,  
    "title": "string",  
    "author": "string"  
}
```

The book variable is built using the three required keys for the payload parameter and values from the three fake variables.

```
book = {"id":i, "title": fakeTitle, "author": fakeAuthor, "isbn": fakeISBN}
```

3.10.6. Finally, the addBook function is called passing the book and apiKey variables. Because addBook is part of the loop, it will be called as many times it is initiated, one time each for book ID.

3.11. Run and verify the add100RandomBooks.py program.

3.11.1. Enter the Python 3 command to run the add100RandomBooks.py program. You should get output like the following although your book titles and ISBN numbers will be different fake values.

```
devasc@labvm:~/labs/devnet-src/school-library$ python3  
add100RandomBooks.py
```

```
Book {'id': 4, 'title': 'Assimilated client-server frame', 'author': 'Chelsea  
Mitchell', 'isbn': '978-0-411-83123-3'} added.
```

```
Book {'id': 5, 'title': 'Adaptive tangible conglomeration', 'author': 'Edward  
Ryan', 'isbn': '978-1-64406-014-8'} added.
```

<output omitted>

```
Book {'id': 103, 'title': 'Fundamental uniform data-warehouse', 'author':  
'Dennis David', 'isbn': '978-1-68465-896-1'} added.
```

```
Book {'id': 104, 'title': 'Organic 4thgeneration functionalities', 'author':  
'Nicole Gilbert', 'isbn': '978-0-13-176202-2'} added.
```

3.11.2. Return to the Chromium browser and refresh the <http://library.demo.local/> webpage. You should now see your 100 new books added.

Note: If you got to the API documentation page instead of the main page. (<http://library.demo.local/api/v1/docs>) and use Try It out, you will only get a list of the first 10 books. You can enter a value from 2 to 10 the page parameter to see the other books.

3.11.3. How would you add another 100 books?

End of document.