

Using and Creating an API

What is an API?

Web API - A web API allows for information or functionality to be manipulated by other programs via the internet. For example, with Twitter's web API, you can write a program in a language like Python or Javascript that can perform tasks such as favoriting tweets or collecting tweet metadata.

What Users Want in an API?

A documentation is a user's starting place when working with a new API, and well-designed URLs make it easier for users to intuitively find resources.

When to Create an API

1. Your data set is large, making download via FTP unwieldy or resource-intensive.
2. Your users will need to access your data in real time, such as for display on another website or as part of an application.
3. Your data changes or is updated frequently.
4. Your users only need access to a part of the data at any one time.
5. Your users will need to perform actions other than retrieve data, such as contributing, updating, or deleting data.

API Terminology

- HTTP (Hypertext Transfer Protocol) is the primary means of communicating data on the web. HTTP implements a number of "methods," which tell which direction data is moving and what should happen to it. The two most common are GET, which pulls data from a server, and POST, which pushes new data to a server.
- URL (Uniform Resource Locator) - An address for a resource on the web. A URL describes the location of a specific resource, such as a web page. When reading about APIs, you may see the terms URL, request, URI, or endpoint used to describe adjacent ideas.
- JSON (JavaScript Object Notation) is a text-based data storage format that is designed to be easy to read for both humans and machines. JSON is generally the most common format for returning data through an API, XML being the second most common.
- REST (REpresentational State Transfer) is a philosophy that describes some best practices for implementing APIs. APIs designed with some or all of these principles in mind are called REST APIs. While the API outlined in this lesson uses some REST principles, there is a great deal of disagreement around this term. For this reason, I do not describe the example APIs here as REST APIs, but instead as web or HTTP APIs.

Flask Web Framework

Flask is a lightweight Web Server Gateway Interface (WSGI) web application framework, meaning that it provides functionality for building web applications, including managing HTTP requests and rendering templates. It is one of the most popular Python web application frameworks.

Aside from Flask, Python also has Django. Django is a framework that has many built-in tools. Flask applications tend to be written that are suited to a contained application.

Using and Creating an API

Flask official website is here: <https://flask.palletsprojects.com/en/1.1.x/>

Pre-requisites

- Flask installation:
- sqlitebrowser (DB browser for Sqlite): `sudo apt-get update, sudo apt-get install sqlitebrowser`

Designing Requests

The prevailing design philosophy of modern APIs is called REST. For our purposes, the most important thing about REST is that it's based on the four methods defined by the HTTP protocol: POST, GET, PUT, and DELETE. These correspond to the four traditional actions performed on data in a database: CREATE, READ, UPDATE, and DELETE.

Because HTTP requests are so integral to using a REST API, many design principles revolve around how requests should be formatted. We've already created one HTTP request, which returns all books provided in our sample data. To understand the considerations that go into formatting this request, let's first consider a weak or poorly designed example of an API endpoint:

```
http://api.example.com/getbook/10
```

The formatting of this request has a number of issues. The first is semantic—in a REST API, our verbs are typically GET, POST, PUT, or DELETE, and are determined by the request method rather than in the request URL. That means that the word “get” should not appear in our request, since “get” is implied by the fact that we're using a HTTP GET method. In addition, resource collections such as books or users should be denoted with plural nouns. This makes it clear when an API is referring to a collection (books) or an entry (book). Incorporating these principles, our API would look like this:

```
http://api.example.com/books/10
```

The above request uses part of the path (/10) to provide the ID. While this is not an uncommon approach, it's somewhat inflexible—with URLs constructed in this manner, you can generally only filter by one field at a time. Query parameters allow for filtering by multiple database fields and make more sense when providing “optional” data, such as an output format:

```
http://api.example.com/books?author=Ursula+K.+Le+Guin&published=1969&output=xml
```

When designing how requests to your API should be structured, it also makes sense to plan for future additions. Even if the current version of your API serves information on only one type of resource—books, for example—it makes sense to plan as if you might add other resources or non-resource functionality to your API in the future:

Using and Creating an API

`http://api.example.com/resources/books?id=10`

Adding an extra segment on your path such as “resources” or “entries” gives you the option to allow users to search across all resources available, making it easier for you to later support requests such as these:

`https://api.example.com/v1/resources/images?id=10`

`https://api.example.com/v1/resources/all`

Another way to plan for your API’s future is to add a version number to the path. This means that, should you have to redesign your API, you can continue to support the old version of the API under the old version number while releasing, for example, a second version (v2) with improved or different functionality. This way, applications and scripts built using the old version of your API won’t cease to function after your upgrade.

After incorporating these design improvements, a request to our API might look like this:

`https://api.example.com/v1/resources/books?id=10`

Part 1. Using an API

Part 2. Creating an API

Google Shopping API

Steps

1. Issue the below command in your terminal git clone <https://github.com/googleads/googleads-shopping-samples.git>
2. Create the following directories:
 - `$(HOME)/shopping-samples`
 - `$(HOME)/shopping-samples/content`
 - `$(HOME)/shopping-samples/manufacturers`
3. Set up your desired authentication method. See below url.
`https://cloud.google.com/docs/authentication/production`
4. Open the list.py in Visual Studio Code. You will see the same below screen.
5. esdf

Open Library

1. Open the below url.
`https://chroniclingamerica.loc.gov/`
2. In the search box, type the word ‘fire’. See below display.
3. The query string is shown below.

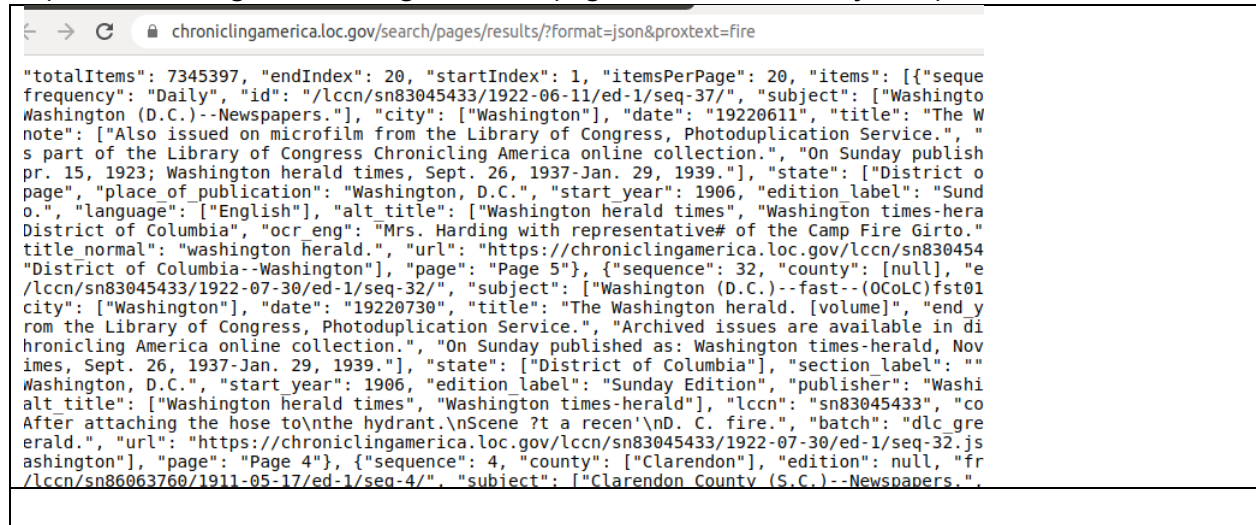
Using and Creating an API

```
https://chroniclingamerica.loc.gov/search/pages/results/?state=&date1=1789&date2=1963&prox  
text=fire&x=19&y=10&dateFilterType=yearRange&rows=20&searchType=basic
```

Query parameters include `format=json` and `proxtext=fire`

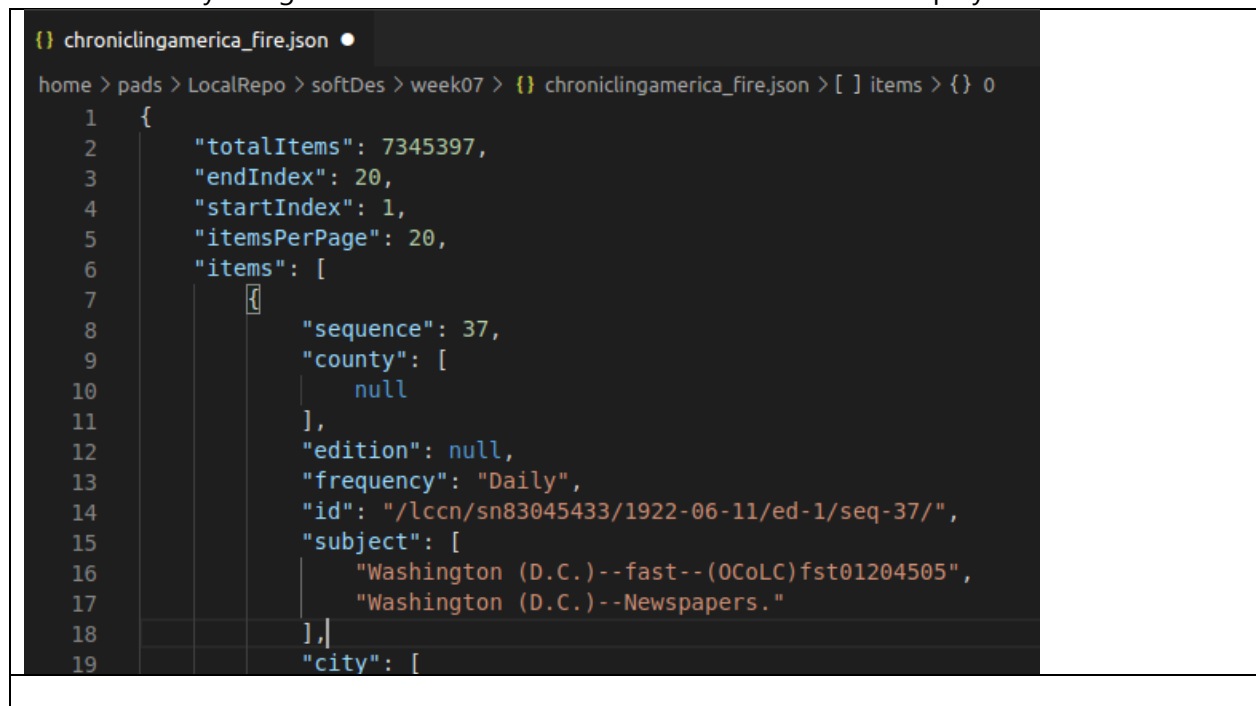
4. Typing the following link in your browser will result to the following.

```
https://chroniclingamerica.loc.gov/search/pages/results/?format=json&proxtext=fire
```



```
{
  "totalItems": 7345397,
  "endIndex": 20,
  "startIndex": 1,
  "itemsPerPage": 20,
  "items": [
    {
      "sequence": 37,
      "county": null,
      "edition": null,
      "frequency": "Daily",
      "id": "/lccn/sn83045433/1922-06-11/ed-1/seq-37/",
      "subject": [
        "Washington (D.C.)--fast--(0CoLC)fst01204505",
        "Washington (D.C.)--Newspapers."
      ],
      "city": [
        "Washington"
      ],
      "date": "19220611",
      "title": "The Washington Herald Times",
      "ocr_eng": "Mrs. Harding with representative# of the Camp Fire Girtto.",
      "alt_title": "Washington Herald Times",
      "language": "English",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Times",
      "lccn": "sn83045433",
      "batch": "dlc gre",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-06-11/ed-1/seq-37.js",
      "page": "Page 5",
      "sequence": 32,
      "county": null,
      "subject": [
        "Washington (D.C.)--fast--(0CoLC)fst01204505",
        "Washington (D.C.)--Newspapers."
      ],
      "city": [
        "Washington"
      ],
      "date": "19220730",
      "title": "The Washington Herald Times",
      "ocr_eng": "After attaching the hose to the hydrant. Scene of a recent D. C. fire.",
      "alt_title": "Washington Herald Times",
      "language": "English",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Times",
      "lccn": "sn83045433",
      "batch": "dlc gre",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-07-30/ed-1/seq-32.js",
      "page": "Page 4",
      "sequence": 4,
      "county": "Clarendon",
      "edition": null,
      "subject": [
        "Clarendon County (S.C.)--Newspapers."
      ],
      "city": [
        "Clarendon"
      ],
      "date": "1911-05-17/ed-1/seq-4/",
      "title": "Clarendon County (S.C.)--Newspapers."
    }
  ]
}
```

5. Download the json file and save as `chroniclingamerica_fire.json`. Open it in your Visual Studio code and format it by using `ctrl+shift+i`. You should have the same below display.



```
{
  "totalItems": 7345397,
  "endIndex": 20,
  "startIndex": 1,
  "itemsPerPage": 20,
  "items": [
    {
      "sequence": 37,
      "county": null,
      "edition": null,
      "frequency": "Daily",
      "id": "/lccn/sn83045433/1922-06-11/ed-1/seq-37/",
      "subject": [
        "Washington (D.C.)--fast--(0CoLC)fst01204505",
        "Washington (D.C.)--Newspapers."
      ],
      "city": [
        "Washington"
      ],
      "date": "19220611",
      "title": "The Washington Herald Times",
      "ocr_eng": "Mrs. Harding with representative# of the Camp Fire Girtto.",
      "alt_title": "Washington Herald Times",
      "language": "English",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Times",
      "lccn": "sn83045433",
      "batch": "dlc gre",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-06-11/ed-1/seq-37.js",
      "page": "Page 5",
      "sequence": 32,
      "county": null,
      "subject": [
        "Washington (D.C.)--fast--(0CoLC)fst01204505",
        "Washington (D.C.)--Newspapers."
      ],
      "city": [
        "Washington"
      ],
      "date": "19220730",
      "title": "The Washington Herald Times",
      "ocr_eng": "After attaching the hose to the hydrant. Scene of a recent D. C. fire.",
      "alt_title": "Washington Herald Times",
      "language": "English",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Times",
      "lccn": "sn83045433",
      "batch": "dlc gre",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-07-30/ed-1/seq-32.js",
      "page": "Page 4",
      "sequence": 4,
      "county": "Clarendon",
      "edition": null,
      "subject": [
        "Clarendon County (S.C.)--Newspapers."
      ],
      "city": [
        "Clarendon"
      ],
      "date": "1911-05-17/ed-1/seq-4/",
      "title": "Clarendon County (S.C.)--Newspapers."
    }
  ]
}
```

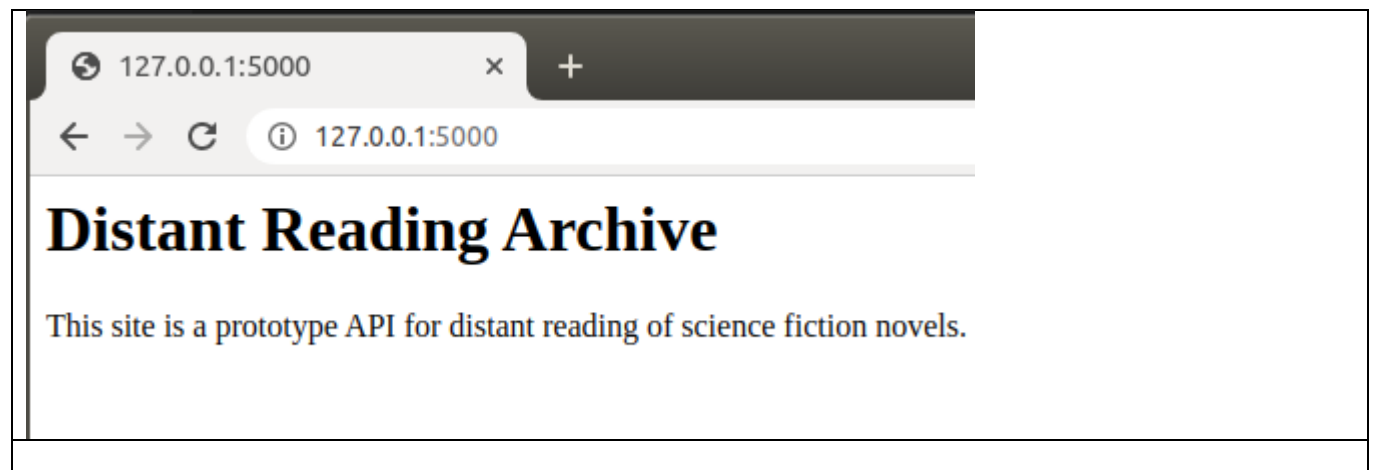
6. Open in VS Code the `api.py` found in the `chronicling+america` folder.

Using and Creating an API

<pre>api.py X api > api.py > home 1 import flask 2 3 app = flask.Flask(__name__) 4 app.config["DEBUG"] = True 5 6 7 @app.route('/', methods=['GET']) 8 def home(): 9 return "<h1>Distant Reading Archive</h1>
<p>This site is a prototype API for distant reading of science fiction novels.</p>" 10 11 app.run()</pre>	

<pre>api.py X api > api.py > home 1 import flask 2 3 app = flask.Flask(__name__) 4 app.config["DEBUG"] = True 5 6 7 @app.route('/', methods=['GET']) 8 def home(): 9 return "<h1>Distant Reading Archive</h1>
<p>This site is a prototype API for distant reading of science fiction novels.</p>" 10 11 app.run()</pre>	<pre>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL (softDes) pads@pads-Z390-AORUS-PRO-WIFI:~/LocalRepo/softDes/week07/chronicling+america: python3 api.py * Serving Flask app "api" (lazy loading) * Environment: production WARNING: This is a development server. Do not use it in production. Use a production WSGI server instead. * Debug mode: on * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) * Restarting with stat * Debugger is active! * Debugger PIN: 114-572-813</pre>

7. Copy the <http://127.0.0.1:5000> in your browser or open it by hovering your mouse on the url VS Code and pressing ctrl+c. You should have the same display below.



8. Flask maps HTTP requests to Python functions. In this case, we've mapped one URL path ('/') to one function, `home`. When we connect to the Flask server at `http://127.0.0.1:5000/`, Flask checks if there is a match between the path provided and a defined function. Since `/`, or no additional provided path, has been mapped to the `home` function, Flask runs the code in the function and displays the returned result in the browser. In this case, the returned result is HTML markup for a home page welcoming visitors to the site hosting our API.

The process of mapping URLs to functions is called routing. The line

	Description
<code>@app.route('/', methods=['GET'])</code>	Flask know that this function, <code>home</code> , should be mapped to the path <code>/</code> . The methods list (<code>methods=['GET']</code>) is a keyword argument that lets Flask know what kind of HTTP requests are allowed.
<code>import flask</code>	Imports the Flask library, making the code available to the rest of the application.
<code>app = flask.Flask(__name__)</code>	Creates the Flask application object, which contains data about the application and also methods (object functions) that tell the application to do certain actions. The last line, <code>app.run()</code> , is one such method.
<code>app.config["DEBUG"] = True</code>	Starts the debugger. With this line, if your code is malformed, you'll see an error when you visit your app. Otherwise you'll only see a generic message such as Bad Gateway in the browser when there's a problem with your code.
<code>app.run()</code>	A method that runs the application server.

API Creation

Data will be added as Python list. Dictionaries in Python group pairs of keys and values as shown below program listing (a). The program listing (b) shows two keys, `name` and `number` with their corresponding values.

<pre>{ 'key': 'value', 'key': 'value' }</pre>	<pre>[{ 'name': 'Alexander Graham Bell', 'number': '1-333-444-5555' }, { 'name': 'Thomas A. Watson', 'number': '1-444-555-6666' }]</pre>
(a) Python Dictionary Syntax	(b) Python Dictionary Example

9. Open the `api_02.py` in VS Code.

Using and Creating an API

<pre>api_02.py • api > api_02.py > ... 1 import flask 2 from flask import request, jsonify 3 4 app = flask.Flask(__name__) 5 app.config["DEBUG"] = True 6 7 # Create some test data for our catalog in the f 8 books = [9 {'id': 0, 10 'title': 'A Fire Upon the Deep', 11 'author': 'Vernor Vinge', 12 'first_sentence': 'The coldsleep itself was 13 'year_published': '1992'}, 14 {'id': 1, 15 'title': 'The Ones Who Walk Away From Omela 16 'author': 'Ursula K. Le Guin', 17 'first_sentence': 'With a clamor of bells t 18 'published': '1973'}, 19 {'id': 2,</pre>	<pre>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL (softDes) pads@pads-Z390-AORUS-PRO-WIFI:~/LocalRepo/softDes/w hon /home/pads/LocalRepo/softDes/week07/chronicling+america/a * Serving Flask app "api_02" (lazy loading) * Environment: production WARNING: This is a development server. Do not use it in a Use a production WSGI server instead. * Debug mode: on * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) * Restarting with stat * Debugger is active! * Debugger PIN: 114-572-813 []</pre>
---	---

10. Follow step 7. You will be seeing the same below display.

<pre>127.0.0.1:5000/api/v1/res x + 127.0.0.1:5000/api/v1/resources/books/all [{ "author": "Vernor Vinge", "first_sentence": "The coldsleep itself was dreamless.", "id": 0, "title": "A Fire Upon the Deep", "year_published": "1992" }, { "author": "Ursula K. Le Guin", "first_sentence": "With a clamor of bells that set the s "id": 1, "published": "1973", "title": "The Ones Who Walk Away From Omelas" }, { "author": "Samuel R. Delany", "first_sentence": "to wound the autumnal city.", "id": 2, "published": "1975", "title": "Dhalgren" }]</pre>

11. Run the code (navigate to your api folder in the command line and enter `python api.py`). Once the server is running, visit our route URL to view the data in the catalog:

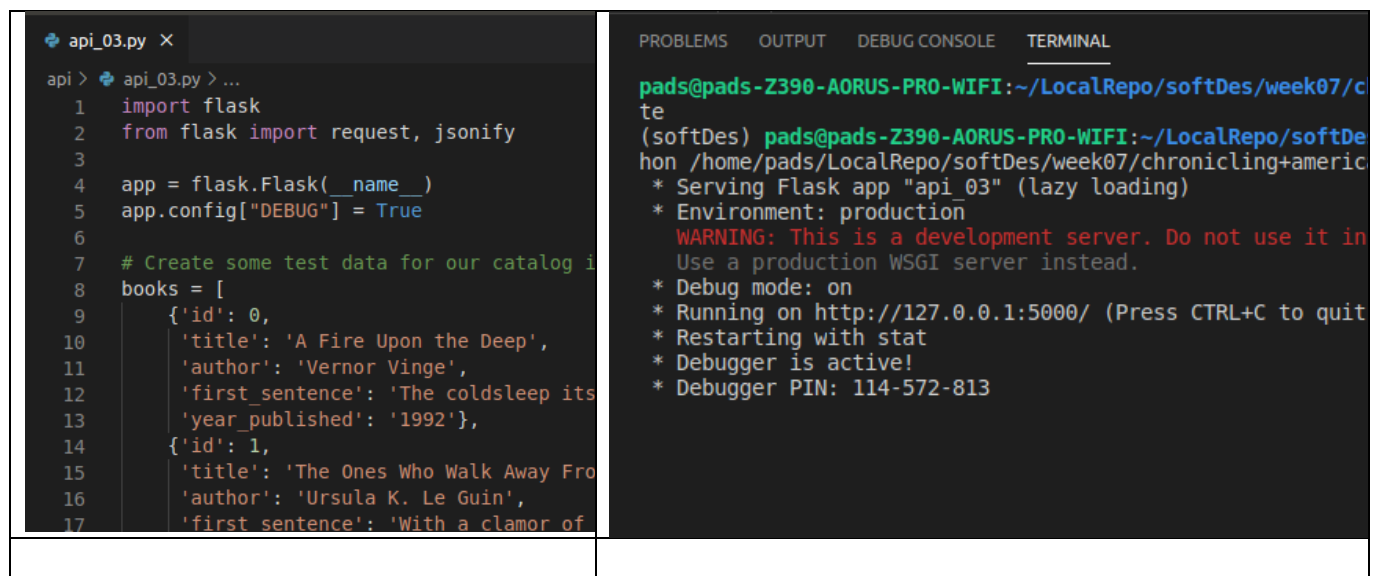
<http://127.0.0.1:5000/api/v1/resources/books/all>

Using and Creating an API

You should see JSON output for the three entries in our test catalog. Flask provides us with a jsonify function that allows us to convert lists and dictionaries to JSON format. In the route we created, our book entries are converted from a list of Python dictionaries to JSON before being returned to a user.

Finding Specific Resources

12. Open and run the api_03.py in your VS Code.



The image shows a VS Code editor with a file named `api_03.py` open. The code defines a Flask application with a list of book data. The terminal on the right shows the output of running the application, indicating it is running on `http://127.0.0.1:5000/`.

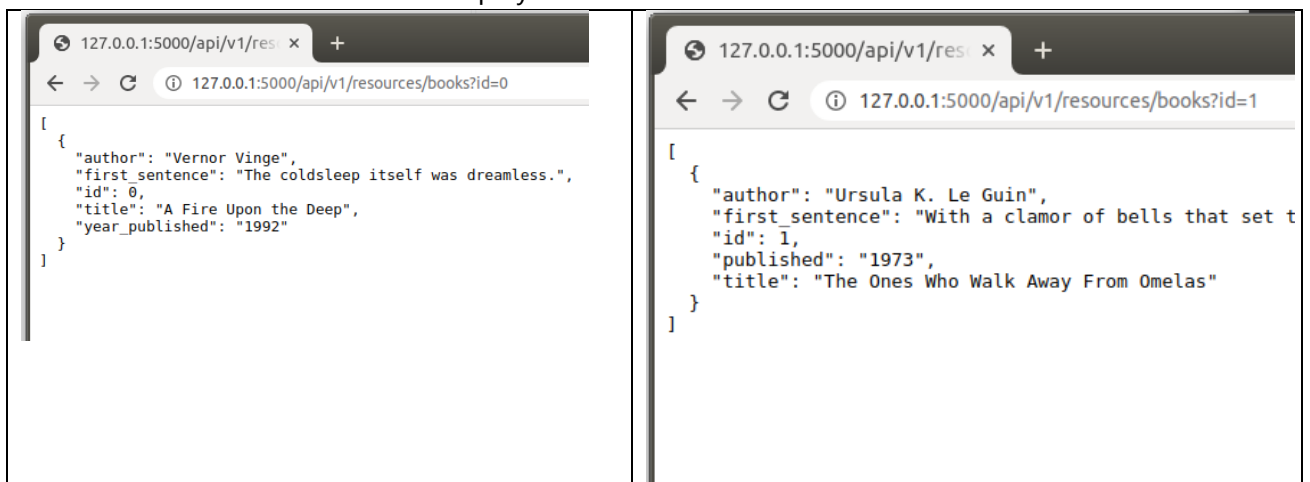
```
api_03.py X
api > api_03.py > ...
1 import flask
2 from flask import request, jsonify
3
4 app = flask.Flask(__name__)
5 app.config["DEBUG"] = True
6
7 # Create some test data for our catalog i
8 books = [
9     {'id': 0,
10      'title': 'A Fire Upon the Deep',
11      'author': 'Vernor Vinge',
12      'first_sentence': 'The coldsleep its
13      'year_published': '1992'},
14     {'id': 1,
15      'title': 'The Ones Who Walk Away Fro
16      'author': 'Ursula K. Le Guin',
17      'first sentence': 'With a clamor of
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
pads@pads-Z390-AORUS-PRO-WIFI:~/LocalRepo/softDes/week07/c
te
(flask) pads@pads-Z390-AORUS-PRO-WIFI:~/LocalRepo/softDe
hon /home/pads/LocalRepo/softDes/week07/chronicling+americ
* Serving Flask app "api_03" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 114-572-813
```

13. Use the following urls to view the filtering capability provided by the API.

```
127.0.0.1:5000/api/v1/resources/books?id=0
127.0.0.1:5000/api/v1/resources/books?id=1
127.0.0.1:5000/api/v1/resources/books?id=2
127.0.0.1:5000/api/v1/resources/books?id=3
```

14. You should have the same below displays.

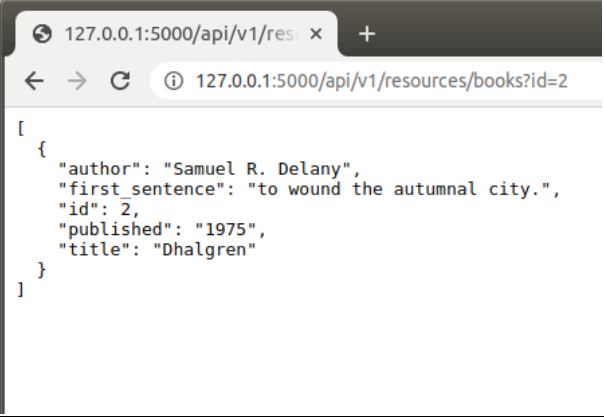
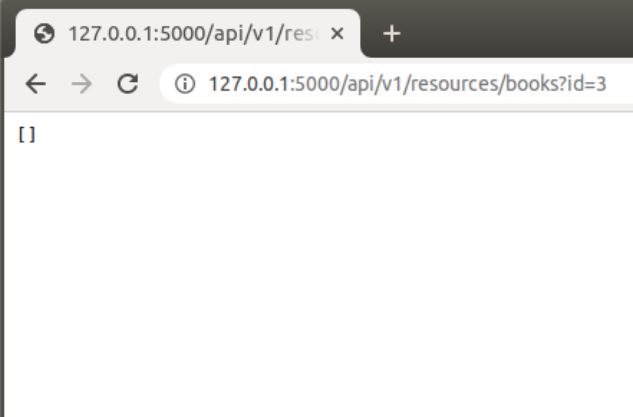


The image shows two browser windows displaying JSON responses from the API. The first window shows the response for `id=0`, and the second window shows the response for `id=1`.

```
127.0.0.1:5000/api/v1/res x +
127.0.0.1:5000/api/v1/resources/books?id=0
[
  {
    "author": "Vernor Vinge",
    "first_sentence": "The coldsleep itself was dreamless.",
    "id": 0,
    "title": "A Fire Upon the Deep",
    "year_published": "1992"
  }
]
```

```
127.0.0.1:5000/api/v1/res x +
127.0.0.1:5000/api/v1/resources/books?id=1
[
  {
    "author": "Ursula K. Le Guin",
    "first_sentence": "With a clamor of bells that set t
    "id": 1,
    "published": "1973",
    "title": "The Ones Who Walk Away From Omelas"
  }
]
```


Using and Creating an API

(a)	(b)
	
(c)	(d)

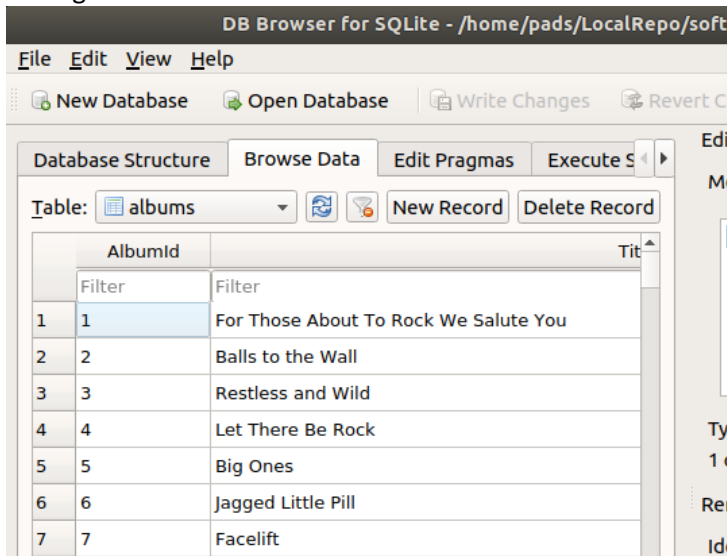
Notice that every time a query is made, the VS Code will also display the status of the request as shown below.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 114-572-813
127.0.0.1 - - [06/Jul/2020 23:21:19] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:21:19] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [06/Jul/2020 23:23:48] "GET /api/v1/resources/books?id=0 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:28:32] "GET /api/v1/resources/books?id=1 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:30:59] "GET /api/v1/resources/books?id=2 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:31:33] "GET /api/v1/resources/books?id=4 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:31:52] "GET /api/v1/resources/books?id=3 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:31:56] "GET /api/v1/resources/books?id=2 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:32:04] "GET /api/v1/resources/books?id=3 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:38:37] "GET /api/v1/resources/books HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:51:59] "GET /api/v1/resources/books?id=3 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:52:11] "GET /api/v1/resources/books?id=100 HTTP/1.1" 200 -
127.0.0.1 - - [06/Jul/2020 23:52:17] "GET /api/v1/resources/books?id=0 HTTP/1.1" 200 -
```

Connecting an API to a Database

15. Open the chinook database.

Using and Creating an API



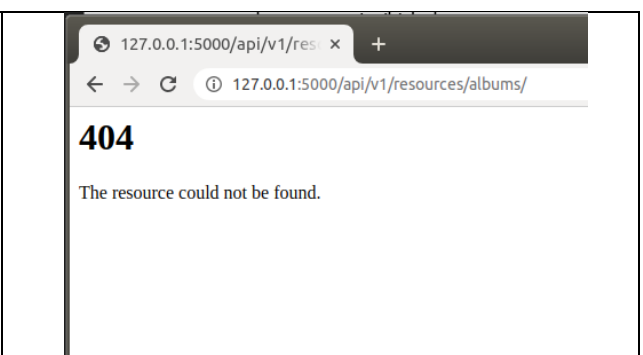
The screenshot shows the 'DB Browser for SQLite' application. The 'Database Structure' tab is active, displaying a table named 'albums'. The table has two columns: 'AlbumId' and 'Title'. The data is as follows:

AlbumId	Title
1	For Those About To Rock We Salute You
2	Balls to the Wall
3	Restless and Wild
4	Let There Be Rock
5	Big Ones
6	Jagged Little Pill
7	Facelift

16. Open and run the api_02.py in VS Code.

 <pre>api_04.py x api > api_04.py > api_all 1 import flask 2 from flask import request, jsonify 3 import sqlite3 4 5 app = flask.Flask(__name__) 6 app.config["DEBUG"] = True 7 8 def dict_factory(cursor, row): 9 d = {} 10 for idx, col in enumerate(cursor.description): 11 d[col[0]] = row[idx] 12 return d 13 14 @app.route('/', methods=['GET']) 15 def home(): 16 return '<h1>Accessing Chinook Database</h1>' 17 18 <p>A prototype API for accessing the chinook sqlite</pre>	 <p>127.0.0.1:5000/api/v1/res x +</p> <p>127.0.0.1:5000/api/v1/resources/albums/all</p> <pre>[{ "AlbumId": 1, "ArtistId": 1, "Title": "For Those About To Rock We Salute You" }, { "AlbumId": 2, "ArtistId": 2, "Title": "Balls to the Wall" }, { "AlbumId": 3, "ArtistId": 2, "Title": "Restless and Wild" }, { "AlbumId": 4, "ArtistId": 1, "Title": "Let There Be Rock" }, { "AlbumId": 5, "ArtistId": 1, "Title": "Big Ones" }, { "AlbumId": 6, "ArtistId": 1, "Title": "Jagged Little Pill" }, { "AlbumId": 7, "ArtistId": 1, "Title": "Facelift" }]</pre>
(a)	(b)

17. Add the below code above the app.run() line. Run the program and in your browser, remove the 'all' in the 'http://127.0.0.1:5000/api/v1/resources/albums/all' url. The display of the browser should be the same as in (b) below.

<pre>@app.errorhandler(404) def page_not_found(e): return "<h1>404</h1><p>The resource could not be found.</p>", 404</pre>	 <p>127.0.0.1:5000/api/v1/res x +</p> <p>127.0.0.1:5000/api/v1/resources/albums/</p> <p>404</p> <p>The resource could not be found.</p>
(a)	(b)

18. Open and run the api_05.py in your VS Code. Enter in your browser the following:
- http://127.0.0.1:5000/api/v1/resources/albums?title=Jagged+Little+Pill

Using and Creating an API

b) `http://127.0.0.1:5000/api/v1/resources/albums?albumid=1`

	
(a)	(b)

References:

References

Flask	https://palletsprojects.com/p/flask/