



MAPUA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Lab 5: Using and Creating an API

CPE106L (Software Design Laboratory)

Hannah Mae Antaran

Darrel Umali Virtusio

Kathleen Joy Tupas

Group: 01

Section: E01



PreLab

Readings

Pearlman, S. (2016). *What are APIs and how do APIs work?* MuleSoft.

API stands for Application Programming Interface. It is a software intermediary that allows two applications to talk to each other. In other words, an API is the messenger that delivers the user's request to the provider and then delivers the response back to the user. One of the chief advantages of APIs is that it allows the abstraction of functionality between one system and another. An API endpoint decouples the consuming application from the infrastructure that provides a service. As long as the specification for what the service provider is delivering to the endpoint remains unchanged, the alterations to the infrastructure behind the endpoint should not be noticed by the applications that rely on that API.

Therefore, the service provider is given a great deal of flexibility when it comes to how its services are offered. For example, if the infrastructure behind the API involves physical servers at a data center, the service provider can easily switch to virtual servers that run in the cloud.

Insights

An API defines functionalities that are independent of their respective implementations, which allows those implementations and definitions to vary without compromising each other. Therefore, a good API makes it easier to develop a program by providing the building blocks.

When developers create code, they don't often start from scratch. APIs enable developers to make repetitive yet complex processes highly reusable with a little bit of code. The speed that APIs enable developers to build out apps is crucial to the current pace of application development. Developers are now much more productive than they were before when they had to write a lot of code from scratch. With an API they don't have to reinvent the wheel every time they write a new program. Instead, they can focus on the unique proposition of their applications while outsourcing all the commodity functionality to APIs.

Reflection

Application programming interfaces are a big part of the web. This experiment is all about API. An API is a tool that makes a website's data digestible for a computer. Through it, a computer can view and edit data, just like a person can by loading pages and submitting forms. There are a lot of APIs that can be used. Some of the most famous APIs are Twitter, Facebook, YouTube, Spotify, and a lot more. You can use these APIs in Python to perform tasks such as retweeting, liking videos, sharing posts, basically every stuff that you can think of.

- **Objectives**

1. To know about the importance API.
2. To learn how to use and create an API.
3. To demonstrate on using various kinds of APIs.

- **Tools Used**

1. Visual Studio Code
2. DB Browser

- **Procedure**

For this experiment, we will use the reddit API to experiment on how to use it to scrape Reddit with Python.

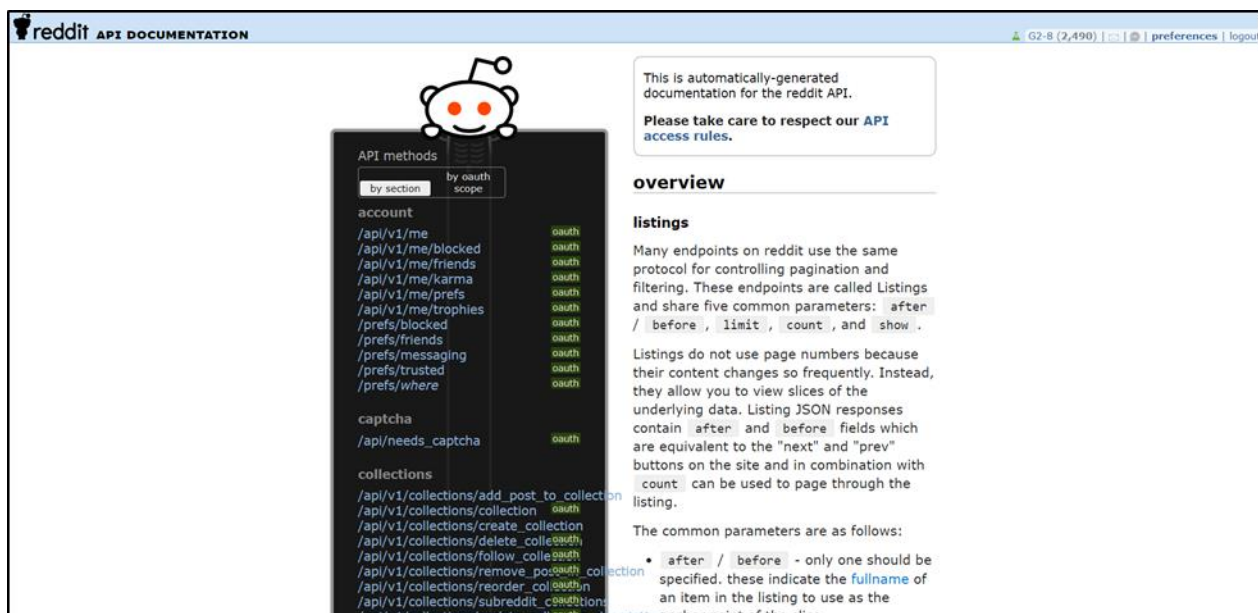


Figure 1.1 Reddit API documentation.

In this page, you can see the overview about the Reddit API and learn about how to use it and what can you do with the API. We will use this as guide as we perform the experiment.

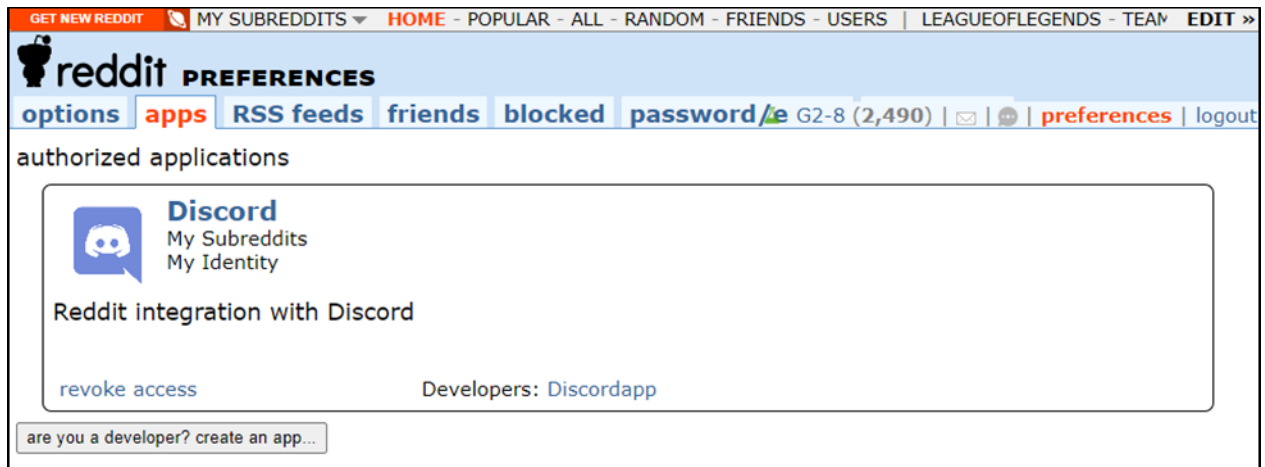


Figure 1.2 Creating an app on Reddit.

The very first thing you'll need to do is "Create an App" within Reddit to get the OAuth2 keys to access the API. To create an app, we go to <https://www.reddit.com/prefs/apps>.

create application

Please [read the API usage guidelines](#) before creating your application. After creating, you will be required to [register](#) for production API use.

name

☐ web app A web based application

☐ installed app An app intended for installation, such as on a mobile phone

☒ script Script for personal use. Will only have access to the developers accounts

description

about url

redirect uri

Figure 1.3 Form.

Pick a name for your application and add a description for reference. Also make sure you select the "script" option and don't forget to put <http://localhost:8080> in the redirect URL field.

CPE106L Experiment 5
personal use script
nu_xchEsqSZqyA

This is an example of using Reddit API

[change icon](#)

secret EpD5-bQ_TxNC2-JeaEJkx6tgvdA

name CPE106L Experiment 5

description This is an example of using Reddit API

about url

redirect uri http://localhost:8080

[update app](#)

developers G2-8 (that's you!) [remove](#)

add developer:

[delete app](#)

Figure 1.4 Successfully creating the app.

After hitting the create app button, we are ready to use the OAuth2 authorization to connect to the API and start scraping. Remember to store the 14-character personal use script and the 27-character secret key.

```

C:\Users\dtvir>pip install praw
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\dtvir>pip install praw
Collecting praw
  Downloading praw-7.1.0-py3-none-any.whl (152 kB)
    | 152 kB 595 kB/s
Collecting prawcore<2.0,>=1.3.0
  Downloading prawcore-1.4.0-py3-none-any.whl (15 kB)
Collecting websocket-client>=0.54.0
  Downloading websocket_client-0.57.0-py2.py3-none-any.whl (200 kB)
    | 200 kB 3.3 MB/s
Collecting update-checker>=0.17
  Downloading update_checker-0.17-py2.py3-none-any.whl (7.0 kB)
Collecting requests<3.0,>=2.6.0
  Downloading requests-2.24.0-py2.py3-none-any.whl (61 kB)
    | 61 kB 173 kB/s

```

Figure 1.5 Installing PRAW.

For this experiment, we will need PRAW since this is the Python Reddit API Wrapper. Along with this module, we will also use pandas and built-in python modules, datetime. To learn more about the libraries, we will refer to the documentation as we perform the experiment.

```

API.py > ...
Set as interpreter
1  #!/ python3
2  import praw
3  import pandas as pd
4  import datetime as dt
5
6  reddit = praw.Reddit(client_id='nu_xchEsqSZqyA',
7                        client_secret='EpD5-bQ_TxNC2-JeaEJkx6tgvdA',
8                        user_agent='CPE106L Experiment 5',
9                        username='G2-8',
10                       password='[REDACTED]')
11

```

Figure 1.6 Getting to Reddit instances.

In this part, we imported the necessary libraries: praw, pandas, and datetime. We also accessed to Reddit by entering the client_id which is the 14-character key, client_secret which is the 27-character key, user_agent which is the app name, username which is the username of the reddit account, and password.

```

subreddit = reddit.subreddit('leagueoflegends')

top_subreddit = subreddit.top(limit=50)

```

Figure 1.7 Accessing the subreddit r/leagueoflegends.

For this part, any subreddits can be accessed which will be the choice of the user. We chose the subreddit r/leagueoflegends because it is one of the most active and famous subreddit in Reddit. The top_subreddit variable is for accessing the contents of the subreddit. There are different ways to access posts created by redditors: .hot, .new, .top, and .gilded. We grabbed the most upvoted topics in this subreddit and limit the posts to top 50 by using the command subreddit.top(limit=50).


```

16 ∨ topics_dict = {"title": [],
17                  "score": [],
18                  "id": [], "url": [],
19                  "comms_num": [],
20                  "created": [],
21                  "body": []}
22
23 ∨ for submission in top_subreddit:
24     topics_dict["title"].append(submission.title)
25     topics_dict["score"].append(submission.score)
26     topics_dict["id"].append(submission.id)
27     topics_dict["url"].append(submission.url)
28     topics_dict["comms_num"].append(submission.num_comments)
29     topics_dict["created"].append(submission.created)
30     topics_dict["body"].append(submission.selftext)
31
32 topics_data = pd.DataFrame(topics_dict)

```

Figure 1.8 Parsing and downloading the data.

We scraped the information about the topics' title, score, URL, id, comments, and the text of the body. We stored this information in a dictionary. We iterated through the top_subreddit and append the information in the dictionary. And finally, we put the data in a data frame using pandas.

```

def get_date(created):
    return dt.datetime.fromtimestamp(created)

_timestamp = topics_data["created"].apply(get_date)

topics_data = topics_data.assign(timestamp=_timestamp)

topics_data.to_csv('LeagueOfLegends.csv', index=False)

```

Figure 1.9 Fixing the date and exporting to CSV file.

In reddit, UNIX timestamps are used for date and time, so we write a function to convert and automate the process. A new column was created for date and time and is ready to be exported in a CSV file. After exporting to CSV file, we run the program and we had successfully scraped data from a subreddit.

	A	B	C	D	E	F	G	H	I
	title	score	id	url	comms_num	created	body	timestamp	
1	OUR LORD AND SAVIOUR MARC MERRILL HAS SAID IF THIS POST GETS TO THE TOP OF /R/ALL, HE WILL GIVE GRAVES A CIGAR AGAIN!	68956	62phzt	https://www.reddit.com/r/leagueof	868	1.49E+09	Lets do this	2017-04-01 16:48:25	
2	Here is my masterpiece. A functional project ashe bow for my cosplay of her. It took me more than a hundred hours and eighty euros (with a	55975	hgd0je	https://v.redd.it/c5djk6regh751	912	1.59E+09		2020-06-28 08:54:59	
3	Tyler1 unbanned from League of Legends	53882	7o5bj9	https://www.reddit.com/r/leagueof	4837	1.52E+09	[Unbanned](https://www.reddit.com/r/leagueof	2018-01-05 11:20:22	
4	Jesus Christ. If this gets enough upvotes it will be the first result when people search for "Jesus Christ"	50977	88rpyt	https://i.redd.it/111rok6fpbp01.jpg	307	1.52E+09		2018-04-02 08:24:00	
5	DoubleLift's Statement on his Family	50035	88xaw	http://www.twitlonger.com/show/	3916	1.52E+09		2018-04-02 18:05:46	
6	Totalbiscuit has passed away, let us never forget the contribution he has given this game among many others	49025	8lxchp	https://www.reddit.com/r/leagueof	1009	1.53E+09	Totalbiscuit has	2018-05-25 15:50:28	
7	End me.	48792	f83x7	https://i.imgur.com/9Uj2ch.png	609	1.59E+09		2020-04-02 12:37:35	
8	Fake Moba. If this gets enough upvotes it will be the first image to show up when searching "Fake Moba"	46765	62t38y	http://i.imgur.com/b158KQ0.jpg	986	1.49E+09		2017-04-02 06:40:51	
9	Tahm Kenh getting married - by @Hinyoyyyy	45190	e3mnu	https://v.redd.it/6v06d16guv141	408	1.58E+09		2019-12-01 09:07:35	
10	[Spoiler] Counter Logic Gaming vs Team SoloMid / NA LCS 2015 Summer - Final / Post-Match Discussion	42491	3i506g	https://www.reddit.com/r/leagueof	9469	1.44E+09	 	2015-06-24 15:59:40	
11	Dodging is more than skill, it's Art	42593	eghz7k	https://v.redd.it/y8tp07u49741	929	1.58E+09		2019-12-28 14:28:46	
12	Tarzaned runs it down then afks, ending the game 2/11 while flaming Froggen	42543	cnt2wl	https://www.reddit.com/r/leagueof	2678	1.57E+09	https://streamable	2019-08-09 14:23:27	
13	I'm almost 43, I just hit Gold, and I have no one to tell it to.	41529	h9hxf	https://www.reddit.com/r/leagueof	1682	1.59E+09	TL;DR: I'm old but	2020-06-16 06:57:02	
14	If this post gets 32768 upvotes, I'll post it again with 32768 Merrills.	41504	88u8fo	https://i.redd.it/hojrrou0e9dp01.pn	986	1.52E+09		2018-04-02 13:37:28	
15	Every person who upvote will be gifted \$10 rp. No bamboozle	40186	88orjl	https://www.reddit.com/r/leagueof	539	1.52E+09		2018-04-01 21:47:49	
16	Riot is forcing me to change my name due to Coronavirus.	39929	g18sl2	https://www.reddit.com/r/leagueof	3465	1.59E+09	Hey all,	2020-04-15 08:32:38	
17	BREAKING NEWS: MARC MERRILL PROMISES TO UNBAN TYLER1 IF THIS POST REACHES /R/ALL, #FREETYLER1	39855	62t8sa	https://pbs.twimg.com/media/CvS	773	1.49E+09		2017-04-02 07:07:22	
18	I made a tiktok transforming myself into Sylas, hope you guys gonna like it!	38471	gwujv6	https://v.redd.it/pj54btqz251	571	1.59E+09		2020-06-05 17:05:11	
19	Lead Riot member, "he [tyler1] looks like a damn humunculous" and, "honestly.. its fine he'll die from a coke overdose or testicular cancer fro	38360	73ko9b	https://www.reddit.com/r/leagueof	8753	1.51E+09	https://imgur.com	2017-10-02 01:03:44	
20	1 upvote = 1 prayer to bring back old graves, pls help I loved him	36395	88nryg	https://i.redd.it/vb607a52K7p01.pn	693	1.52E+09		2018-04-01 18:26:41	
21	This feeling when you found your prestige mistress on ARAM	36151	g9y85	https://v.redd.it/8as827rph41	382	1.59E+09		2020-04-29 10:23:52	
22	Imagtpie, Scarra, Dyrus, Voyboy, and Shiptur are Echo Fox's new Challenger team	35964	6e074n	https://totesports.com/league-of-	3320	1.5E+09		2017-06-03 04:55:31	
23	Warriors Season 2020 Cinematic - League of Legends (ft. 2WEI and Edda Hayes)	34787	em8meb	https://www.youtube.com/watch?	4375	1.58E+09		2020-01-10 03:54:19	
24	LORD MARC SAYS THAT IF THIS POST REACHES /R/ALL, HE WILL PERSONALLY DELETE YASUO FROM LEAGUE OF LEGENDS	34717	62ulp5	http://pm1.narvii.com/5757/b38a6f	322	1.49E+09		2017-04-02 10:42:50	
25	I learned to stop worrying about Aphelios by just playing Mundo	34570	ectfym	https://www.reddit.com/r/leagueof	853	1.58E+09	I'm a simple man	2019-12-19 09:42:10	
26	Royal Never Give Up vs. G2 Esports / 2018 World Championship - Quarter-Final / Post-Match Discussion	34300	9ptz42	https://www.reddit.com/r/leagueof	7661	1.54E+09	###WORLDS 2018	2018-10-21 05:04:50	
27	Averaging 27 deaths a game, the all female team Vaevictis has been kicked from the LCL	33837	f5sc16	https://www.theloadout.com/leag	193	1.58E+09		2020-02-19 06:10:51	
28	I traveled back in time from 2050 and I am an Emerald IV toplane player. As I already know your questions, I will answer before you write.	32782	7ppcgz	https://www.reddit.com/r/leagueof	1916	1.52E+09		2018-01-12 08:41:28	
29	Voyboy's thoughts on NA solo queue	32359	gdmys3	https://www.youtube.com/watch?	5231	1.59E+09		2020-05-05 15:47:32	
30	/R/DOTA2 DOESN'T WANT THIS PIC OF OUR LORD MARC MERRILL ON /R/ALL, WHAT DO YOU THINK /R/THE_MERRILL?	32281	62p0al	http://i.imgur.com/QcfbOn2.png	857	1.49E+09		2017-04-01 15:24:41	
31	Afreeca Freecs vs. Cloud9 / 2018 World Championship - Quarter-Final / Post-Match Discussion	31649	9qlh4m	https://www.reddit.com/r/leagueof	7080	1.54E+09	###WORLDS 2018	2018-10-21 22:29:58	
32	We may lost the game, but we won the mental war	31472	gh8u91	https://v.redd.it/jp346et0y2x41	495	1.59E+09		2020-05-11 12:25:45	
33	Best Yasuo Cosplay Hands Down!	30986	gu40j0	https://v.redd.it/pjwbsie775251	419	1.59E+09		2020-06-01 10:20:35	
34	Invictus Gaming vs. Team Liquid / MSI 2019 - Semi-Final / Post-Match Discussion	29954	bprdvi	https://www.reddit.com/r/leagueof	9458	1.56E+09	###MSI 2019	2019-05-18 05:38:52	
35	I told my little brother that if he ulted a Lee-Sin mid Q that I'd give him ten bucks. I think I owe him a lot more than that.	29891	hsk8e0	https://v.redd.it/yek8lo4quab51	475	1.59E+09		2020-07-17 15:00:08	
36	SK Telecom T1 vs. G2 Esports / 2019 World Championship - Semi-Final / Post-Match Discussion	29525	drl3u2	https://www.reddit.com/r/leagueof	7560	1.57E+09	###WORLDS 2019	2019-11-04 06:43:22	

Figure 1.10 Top 50 posts on r/leagueoflegends.

After running the created API, we obtain the CSV file of the top posts on the subreddit and we successfully used the Reddit API to scrape the data.

Imagine that our research area is sensationalism and the press: has newspaper coverage of major events in the United States become more or less sensational over time? Narrowing the topic, we might ask whether press coverage of, for example, urban fires has increased or decreased with government reporting on fire-related relief spending.

While we won't be able to explore this question thoroughly, we can begin to approach this research space by collecting historical data on newspaper coverage of fires using an API—in this case, the Chronicling America Historical Newspaper API. The Chronicling America API allows access to metadata and text for millions of scanned newspaper pages. In addition, unlike many other APIs, it also does not require an authentication process, allowing us to immediately explore the available data without signing up for an account.

Our initial goal in approaching this research question is to find all newspaper stories in the Chronicling America database that use the term "fire." Typically, use of an API starts with its documentation. On the Chronicling America API page, we find two pieces of information critical for getting the data we want from the API: the API's base URL and the path corresponding to the function we want to perform on the API—in this case, searching the database.

```
{
  "totalItems": 7413424,
  "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--(OCoLC)fst01204505", "Washington (D.C.)--Newspapers."],
      "city": ["Washington"],
      "date": "19220611",
      "title": "The Washington herald. [volume]",
      "end_year": 1939,
      "note": ["Also issued on microfilm from the Library of Congress, Photoduplication Service.", "Archived issues are available in digital format as part of the Library of Congress Chronicling America online collection.", "On Sunday published as: Washington times-herald, Nov. 19, 1922-Apr. 15, 1923; Washington herald times, Sept. 26, 1937-Jan. 29, 1939."],
      "state": ["District of Columbia"],
      "section_label": "",
      "type": "page",
      "place_of_publication": "Washington, D.C.",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Co.",
      "language": ["English"],
      "alt_title": ["Washington times-herald"],
      "lccn": "sn83045433",
      "country": "District of Columbia",
      "ocr_eng": "Mrs. Harding with representative of the Camp Fire Girls.",
      "batch": "dlc_greyhound-ver02",
      "title_normal": "Washington herald.",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-06-11/ed-1/seq-37.json",
      "place": ["District of Columbia--Washington"],
      "page": "Page 5",
      "sequence": 32,
      "county": [null],
      "edition": null,
      "frequency": "Daily",
      "id": "/lccn/sn83045433/1922-07-30/ed-1/seq-32/",
      "subject": ["Washington (D.C.)--fast--(OCoLC)fst01204505", "Washington (D.C.)--Newspapers."],
      "city": ["Washington"],
      "date": "19220730",
      "title": "The Washington herald. [volume]",
      "end_year": 1939,
      "note": ["Also issued on microfilm from the Library of Congress, Photoduplication Service.", "Archived issues are available in digital format as part of the Library of Congress Chronicling America online collection.", "On Sunday published as: Washington times-herald, Nov. 19, 1922-Apr. 15, 1923; Washington herald times, Sept. 26, 1937-Jan. 29, 1939."],
      "state": ["District of Columbia"],
      "section_label": "",
      "type": "page",
      "place_of_publication": "Washington, D.C.",
      "start_year": 1906,
      "edition_label": "Sunday Edition",
      "publisher": "Washington Herald Co.",
      "language": ["English"],
      "alt_title": ["Washington times-herald"],
      "lccn": "sn83045433",
      "country": "District of Columbia",
      "ocr_eng": "After attaching the hose to the hydrant. A scene of a recent fire.",
      "batch": "dlc_greyhound-ver02",
      "title_normal": "Washington herald.",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-07-30/ed-1/seq-32.json",
      "place": ["District of Columbia--Washington"],
      "page": "Page 4",
      "sequence": 4,
      "county": ["Clarendon"],
      "edition": null,
      "frequency": "Weekly",
      "id": "/lccn/sn86063760/1911-05-17/ed-1/seq-4/",
      "subject": ["Clarendon County (S.C.)--Newspapers.", "Manning (S.C.)--Newspapers.", "South Carolina--Clarendon County--fast--(OCoLC)fst01217334", "South Carolina--Manning--fast--(OCoLC)fst01228666"],
      "city": ["Manning"],
      "date": "19110517",
      "title": "The Manning times. [volume]",
      "end_year": 1999,
      "note": ["Archived issues are available in digital format as part of the Library of Congress Chronicling America online collection.", "Description based on: Vol. 1, no. 43 (Oct. 7, 1885).", "Latest issue consulted: Vol. 131, no. 17 (Apr. 25, 2013)."],
      "state": ["South Carolina"],
      "section_label": "",
      "type": "page",
      "place_of_publication": "Manning, Clarendon County, S.C.",
      "start_year": 1884,
      "edition_label": "",
      "publisher": "S.A. Nettles",
      "language": ["English"],
      "alt_title": [],
      "lccn": "sn86063760",
      "country": "South Carolina",
      "ocr_eng": "Part of my stock was then, I have placed my entire stock must be yellow homespun, Fire Sale..... 8 4c. Checked Homespun, Fire Sale..... 4 41 c. Ladies' Gauze Vests, Fire Sale..... 4 41-2c. Ladies' 75c. Pocket Books, Fire Sale..... 39c. Mens' Dress Shirts, Fire Sale..... 38c. Mens' all colors, Fire Sale..... 7 1-2c. This is an opportunity nothing must go at this Fire! nBR A4\nslightly damaged by water\nnitire stock of clothing, Shoes\nTHE RE\nsolid in order to replace the\nMens' 83.50 Shoes and Oxfords, Fire Sale..... $ 2 48 15\nBoy's 83.00 Suits, all sizes, Fire Sale..... 1 98\nMens' $15.00 Suits, all kinds, Fire Sale..... 9 85\nMens' Corset Covers, Fire Sale..... 19c. 15\n$3.00 Ladies' Oxfords, Fire Sale..... 1 98\nMens' $2.50 Pants, Fire Sale..... 1 48 $1\nfor you to secure Big Bargainsale. Come quick and get t\nMORTI\nS I\nnduring Saturday's fire, a\n3, Dress Goods, Millinery, I\nna with fresh goods.\nnc. C3hambrays, Fire Sale..... 7 71-2c. La\ntdies' 75c. Shirtwaists. Fi re Sale..... 43c. $3\nmens' $10.00 Suits. Fire Sale..... 6 98 M\nc. Embroideries, Fire Sale..... 6c. M\nmens' $5.00 Pants, Fire Sale..... 2 48 10c.\n.00 Overalls, Fire Sale..... 73c. 10c.\nnins for very little money.\nhe Big Bargains. You will\nI FORS\nI\nnrid in order to dispose of\ntc., at your mercy at\nRICE\n\ndies' $4.00 Skirts, Fire Sale..... 2 48\n.50 Ladies' Dresses, Fire Sale..... 1 98\nnns' $3.00 Low Shoes, Fire Sale..... 1 95\nnns: 2.50 Oxfords, Fire Sale..... 1.69\n. Apron Checks. Fire Sale..... 5 51-2c.\n. Calicoes, Fire Sale..... 4 41-2c.\nNothing held back, every\nget",
      "batch": "scu_babytate_ver01",
      "title_normal": "Manning times.",
      "url": "https://chroniclingamerica.loc.gov/lccn/sn86063760/1911-05-17/ed-1/seq-4.json",
      "place": ["South Carolina--Clarendon--Manning"],
      "page": "4",
      "sequence": 3,
      "county": ["Bernalillo"],
      "edition": null,
      "frequency": "Daily",
      "id": "/lccn/sn92070582/1921-"
    }
  ]
}
```

Figure 2.1 The Chronicling America API page.

This figure shows the content of chroniclingamerica_fire.json document.

```

API.py  {} chronicingamerica_fire.json •
LAB_5 > {} chronicingamerica_fire.json > ...
1  {}
2  "totalItems": 7413424,
3  "endIndex": 20,
4  "startIndex": 1,
5  "itemsPerPage": 20,
6  "items": [
7    {
8      "sequence": 37,
9      "county": [
10       null
11     ],
12     "edition": null,
13     "frequency": "Daily",
14     "id": "/lccn/sn83045433/1922-06-11/ed-1/seq-37/",
15     "subject": [
16       "Washington (D.C.)--fast--(OCoLC)fst01204505",
17       "Washington (D.C.)--Newspapers."
18     ],
19     "city": [
20       "Washington"
21     ],
22     "date": "19220611",
23     "title": "The Washington herald. [volume]",

```

```

{} chronicingamerica_fire.json •  api_03.py  ...  {} chronicingamerica_fire.json •
LAB_5 > {} chronicingamerica_fire.json > [ ] items > {} 1
61  ],
62  "edition": null,
63  "frequency": "Daily",
64  "id": "/lccn/sn83045433/1922-07-30/
65  "subject": [
66    "Washington (D.C.)--fast--(OCoLC)fst01204505",
67    "Washington (D.C.)--Newspapers."
68  ],
69  "city": [
70    "Washington"
71  ],
72  "date": "19220730",
73  "title": "The Washington herald. [v
74  "end_year": 1939,
75  "note": [
76    "Also issued on microfilm from
77    "Archived issues are available
78    "On Sunday published as: Washin
79  ],
80  "state": [
81    "District of Columbia"
82  ],
83  "section_label": "",
84  "type": "page",
85  "place_of_publication": "Washington
86  "start_year": 1906,
87  "edition_label": "Sunday Edition",
88  "publisher": "Washington Herald Co.
89  "language": [
90    "English"
91  ],
92  "alt_title": [
93    "Washington herald times",
94    "Washington times-herald"
95  ],
96  "lccn": "sn83045433",
97  "country": "District of Columbia",
98  "ocr_eng": "After attaching the hos
99  "batch": "dlc_greyhound_ver02",
100 "title_normal": "washington herald.
101 "url": "https://chroniclingamerica.
102 "place": [
103   "District of Columbia--Washingt
104 ],
105 "page": "Page 4"
106 },
107 {
108   "sequence": 4,
109   "county": [
110     "Clarendon"
111   ],
112   "edition": null,
113   "frequency": "Weekly",
114   "id": "/lccn/sn86063760/1911-05-17/
115   "subject": [
116     "Clarendon County (S.C.)--Newsp
117     "Manning (S.C.)--Newspapers.",
118     "South Carolina--Clarendon Cour
119     "South Carolina--Manning.--fast
120   ],

```

Figure 2.2 Formatted document of chronicingamerica_fire.json.

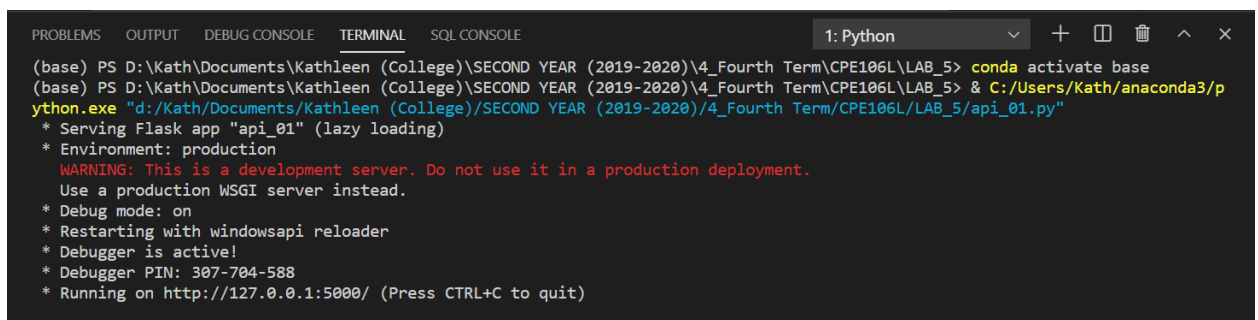
This figure shows the result of formatting the document, chronicingamerica_fire.json.



```
Python - Get Started  API.py  api_01.py X
LAB_5 > api_01.py > ...
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
10
11  app.run()
```

Figure 2.3 Source code of ap1.py.

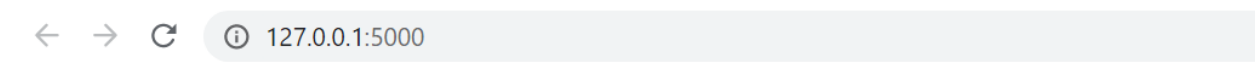
This figure shows the source code for ap1.py. The import flask component of the program imports the flask library which makes the code available to the rest of the application. The app = flask.Flask(__name__), creates the flask application object which contains all the data about the application. Lastly, the app.config["DEBUG"] = True, starts the debugger which displays an error message when the code is malformed.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE
1: Python
(base) PS D:\Kath\Documents\Kathleen (College)\SECOND YEAR (2019-2020)\4_Fourth Term\CPE106L\LAB_5> conda activate base
(base) PS D:\Kath\Documents\Kathleen (College)\SECOND YEAR (2019-2020)\4_Fourth Term\CPE106L\LAB_5> & C:/Users/Kath/anaconda3/p
ython.exe "d:/Kath/Documents/Kathleen (College)/SECOND YEAR (2019-2020)/4_Fourth Term/CPE106L/LAB_5/api_01.py"
* Serving Flask app "api_01" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 2.4 Running the ap1.py.

This figure shows some lines related to debugging which means that the flask is running the application locally at the address <http://127.0.0.1:5000/>.



Distant Reading Archive

This site is a prototype API for distant reading of science fiction novels.

Figure 2.5 The home page when rendered in a browser.

This figure shows the working web application using flask. Flask provides functionality for building web application, including managing HTTP requests and rendering templates.

```

LAB_5 > 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 form of a list of dictionaries.
8  books = [
9      {'id': 0,
10       'title': 'A Fire Upon the Deep',
11       'author': 'Vernor Vinge',
12       'first_sentence': 'The coldsleep itself was dreamless.',
13       'year_published': '1992'},
14      {'id': 1,
15       'title': 'The Ones Who Walk Away From Omelas',
16       'author': 'Ursula K. Le Guin',
17       'first_sentence': 'With a clamor of bells that set the swallows soaring, the Festival of Summer came
18       'published': '1973'},
19      {'id': 2,
20       'title': 'Dhalgren',
21       'author': 'Samuel R. Delany',
22       'first_sentence': 'to wound the autumnal city.',
23       'published': '1975'}
24  ]
25
26
27  @app.route('/', methods=['GET'])
28  def home():
29      return '''<h1>Distant Reading Archive</h1>
30      <p>A prototype API for distant reading of science fiction novels.</p>'''
31
32
33  # A route to return all of the available entries in our catalog.
34  @app.route('/api/v1/resources/books/all', methods=['GET'])
35  def api_all():
36      return jsonify(books)
37
38  app.run()

```

Figure 2.6 Source code of ap2.py.

This figure shows the source code for ap2.py. The code contains entries on three science fiction novels as a list of dictionaries. Each dictionary contains an ID number, title, author, first sentence, and a year of publication for each book.

```

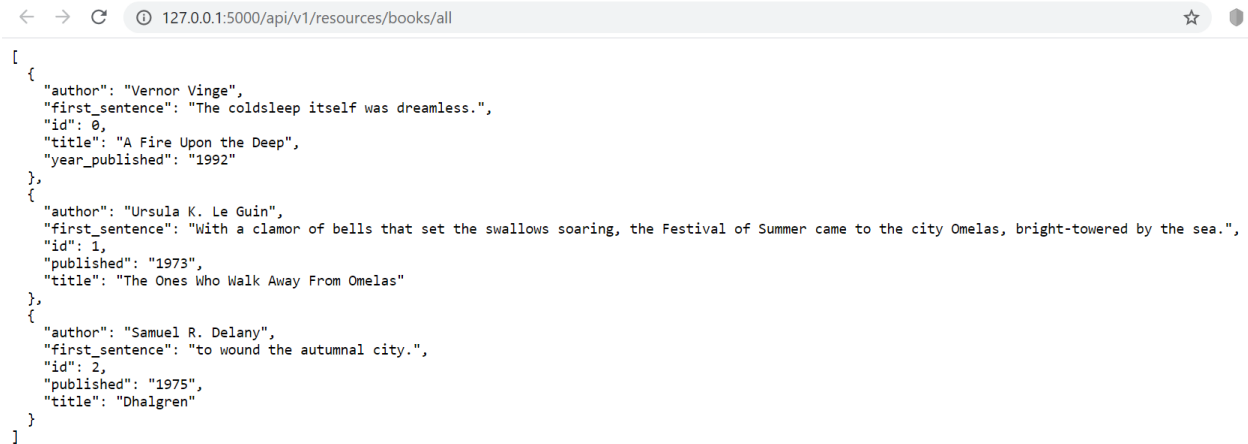
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE: MESSAGES
1: Python

* Serving Flask app "api_02" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

Figure 2.7 Running the ap2.py.

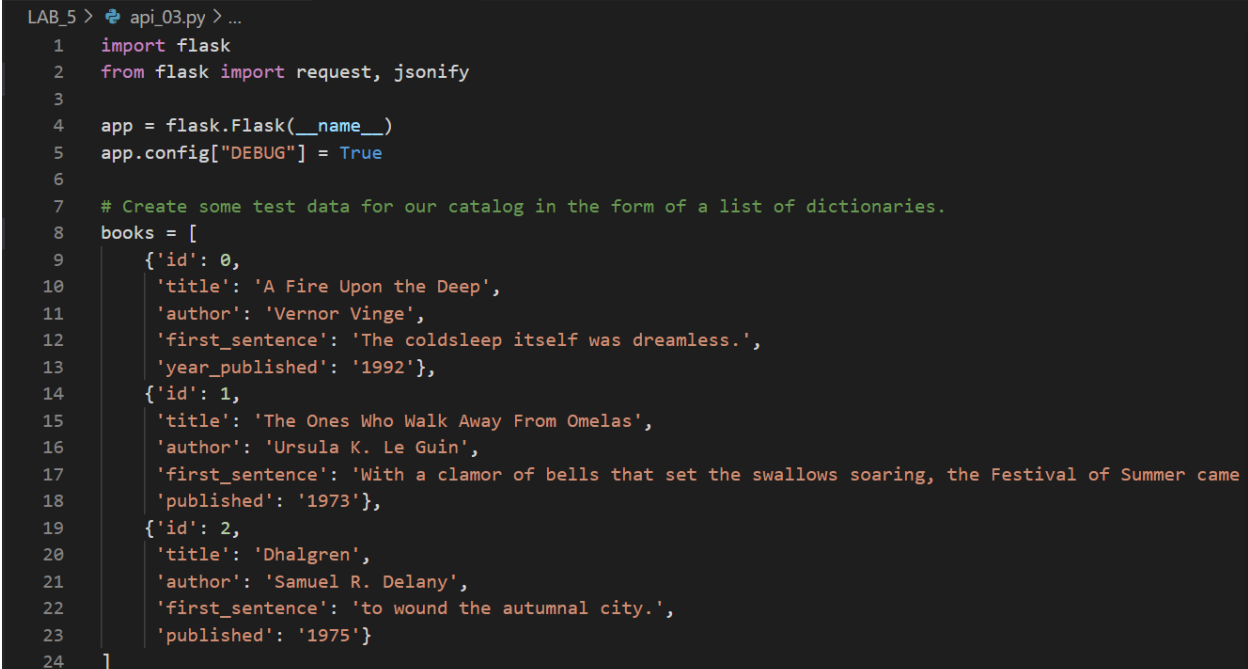
This figure shows that the server is running and to view the data in the browser, the route URL <http://127.0.0.1:5000/api/v1/resources/books/all> was visited.



```
[
  {
    "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 swallows soaring, the Festival of Summer came to the city Omelas, bright-towered by the sea.",
    "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"
  }
]
```

Figure 2.8 The entries on the request made.

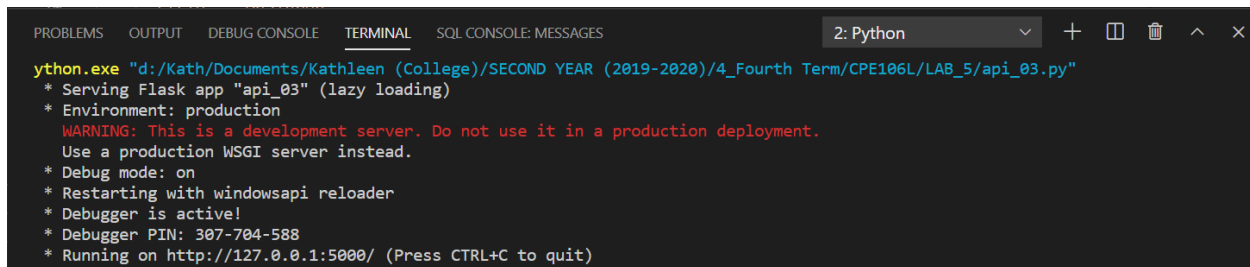
This figure shows all the three entries rendered in the browser. It provided all the details of each entries such as its ID number, title, author, first sentence, and year of publication.



```
LAB_5 > 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 in the form of a list of dictionaries.
8  books = [
9      {
10         'id': 0,
11         'title': 'A Fire Upon the Deep',
12         'author': 'Vernor Vinge',
13         'first_sentence': 'The coldsleep itself was dreamless.',
14         'year_published': '1992'},
15     {
16         'id': 1,
17         'title': 'The Ones Who Walk Away From Omelas',
18         'author': 'Ursula K. Le Guin',
19         'first_sentence': 'With a clamor of bells that set the swallows soaring, the Festival of Summer came',
20         'published': '1973'},
21     {
22         'id': 2,
23         'title': 'Dhalgren',
24         'author': 'Samuel R. Delany',
25         'first_sentence': 'to wound the autumnal city.',
26         'published': '1975'}
27 ]
```

Figure 2.9 Source code of ap3.py.

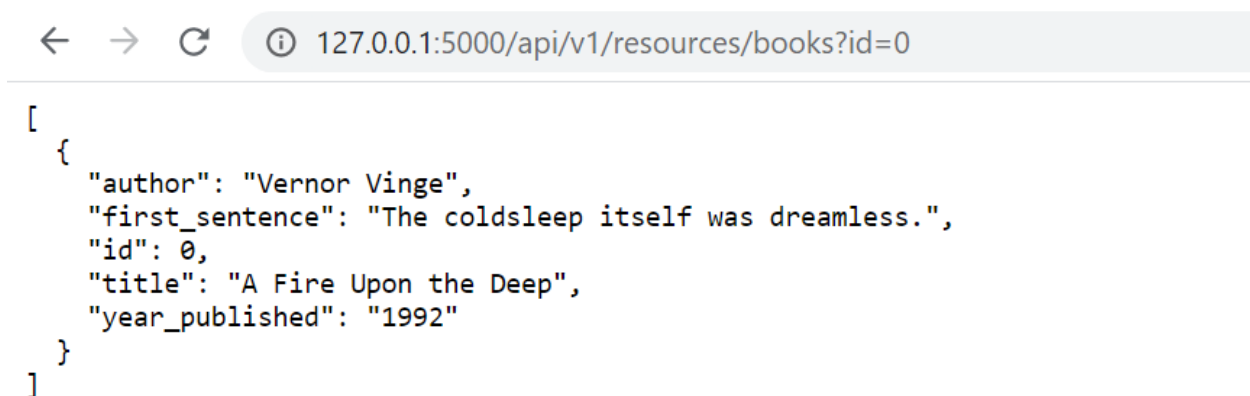
This figure shows the source code for ap3.py. In here, functions were added in the set of codes which allows users to filter their returned results using a more specific request.



```
python.exe "d:/Kath/Documents/Kathleen (College)/SECOND YEAR (2019-2020)/4_Fourth Term/CPE106L/LAB_5/api_03.py"
* Serving Flask app "api_03" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 2.10 Running the ap3.py.

This figure shows that the server is running and to view the data in the browser, four different URLs were visited.

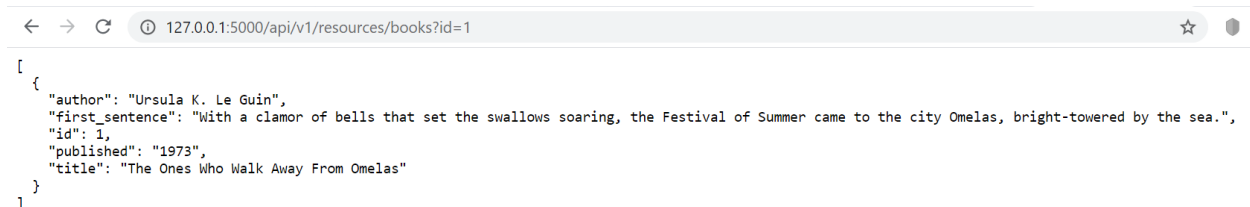


```
127.0.0.1:5000/api/v1/resources/books?id=0

[
  {
    "author": "Vernor Vinge",
    "first_sentence": "The coldsleep itself was dreamless.",
    "id": 0,
    "title": "A Fire Upon the Deep",
    "year_published": "1992"
  }
]
```

Figure 2.11 The entry of the first request.

This figure shows all the data about the first book entry entitled A Fire Upon the Deep after visiting the URL <http://127.0.0.1:5000/api/v1/resources/books?id=0>. As observed, the ID number 0 was used to filter the returned results.

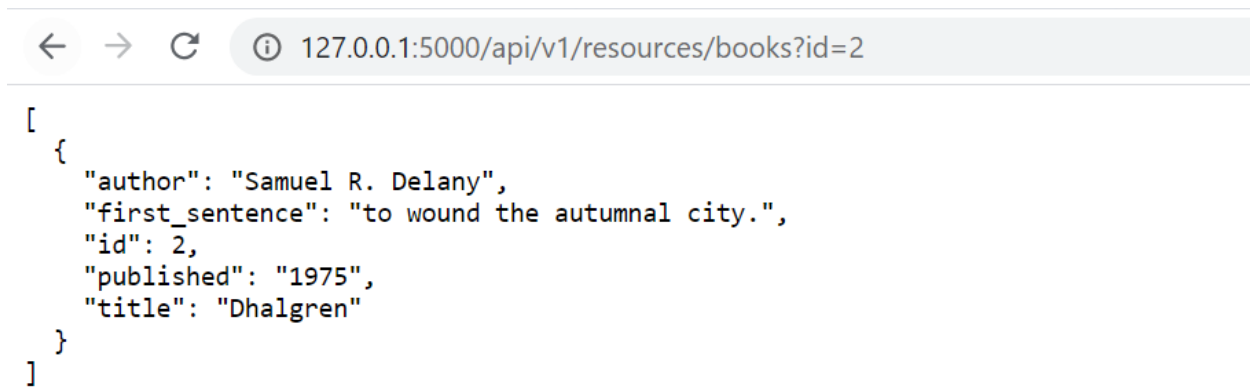


```
127.0.0.1:5000/api/v1/resources/books?id=1

[
  {
    "author": "Ursula K. Le Guin",
    "first_sentence": "With a clamor of bells that set the swallows soaring, the Festival of Summer came to the city Omelas, bright-towered by the sea.",
    "id": 1,
    "published": "1973",
    "title": "The Ones Who Walk Away From Omelas"
  }
]
```

Figure 2.12 The entry on the second request.

This figure shows all the data about the second book entry entitled The One's Who Walk Away from Omelas after visiting the URL <http://127.0.0.1:5000/api/v1/resources/books?id=1>. As observed, the ID number 1 was used to filter the returned results.

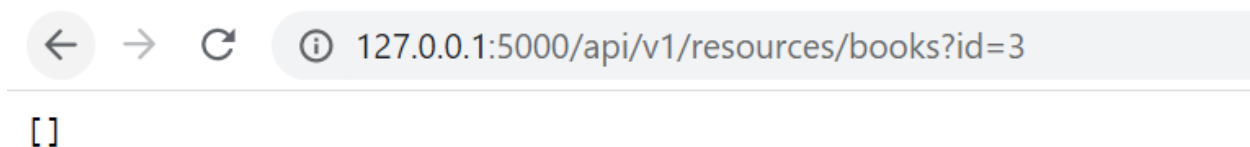


A screenshot of a web browser's address bar and response area. The address bar shows the URL `127.0.0.1:5000/api/v1/resources/books?id=2`. Below the address bar, a JSON array is displayed, containing a single object representing a book entry.

```
[
  {
    "author": "Samuel R. Delany",
    "first_sentence": "to wound the autumnal city.",
    "id": 2,
    "published": "1975",
    "title": "Dhalgren"
  }
]
```

Figure 2.13 The entry on the third request.

This figure shows all the data about the third book entry entitled Dhalgren after visiting the URL <http://127.0.0.1:5000/api/v1/resources/books?id=2>. As observed, the ID number 2 was used to filter the returned results.

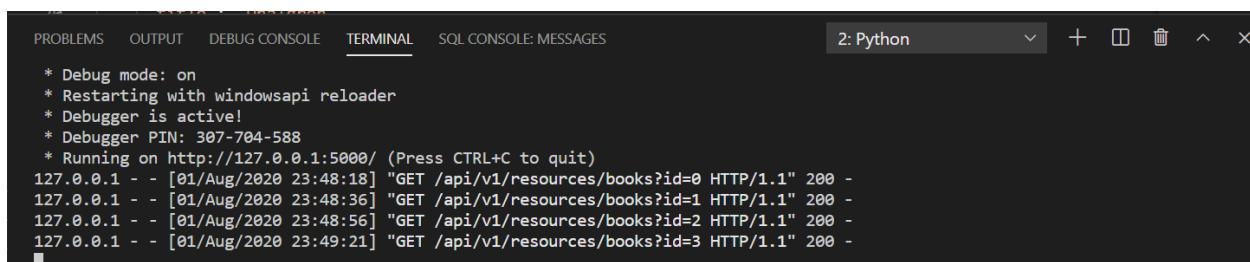


A screenshot of a web browser's address bar and response area. The address bar shows the URL `127.0.0.1:5000/api/v1/resources/books?id=3`. Below the address bar, an empty JSON array is displayed.

```
[]
```

Figure 2.14 The entry on the fourth request.

This figure shows a nonexistent item since there's no book for which the ID value is 3. Hence, visiting the URL <http://127.0.0.1:5000/api/v1/resources/books?id=3> would result to an empty list.



A screenshot of a terminal window with a dark background. It shows the output of a Python application running on `http://127.0.0.1:5000/`. The terminal displays several status messages and a log of four HTTP GET requests to the `/api/v1/resources/books?id=0` through `id=3` endpoints, all returning a `200` status.

```
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [01/Aug/2020 23:48:18] "GET /api/v1/resources/books?id=0 HTTP/1.1" 200 -
127.0.0.1 - - [01/Aug/2020 23:48:36] "GET /api/v1/resources/books?id=1 HTTP/1.1" 200 -
127.0.0.1 - - [01/Aug/2020 23:48:56] "GET /api/v1/resources/books?id=2 HTTP/1.1" 200 -
127.0.0.1 - - [01/Aug/2020 23:49:21] "GET /api/v1/resources/books?id=3 HTTP/1.1" 200 -
```

Figure 2.15 All the requests made.

This figure shows all the query made and the result of each request.

Database Structure		
Browse Data Edit Pragmas Execute SQL		
Create Table Create Index Print		
Name	Type	Schema
▼ Tables (13)		
albums	CREATE TABLE "albums" ([AlbumId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [Title] NVARCHAR(120) NOT NULL, [ArtistId] INTEGER NOT NULL, CONSTRAINT [FK_AlbumArtistId] FOREIGN KEY ([ArtistId]) REFERENCES "artists" ([ArtistId]))	
artists	CREATE TABLE "artists" ([ArtistId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [Name] NVARCHAR(120) NOT NULL, [Albums] INTEGER NOT NULL)	
customers	CREATE TABLE "customers" ([CustomerId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [FirstName] NVARCHAR(40) NOT NULL, [LastName] NVARCHAR(40) NOT NULL, [Address] NVARCHAR(70) NOT NULL, [City] NVARCHAR(40) NOT NULL, [State] NVARCHAR(40) NOT NULL, [Country] NVARCHAR(40) NOT NULL, [SupportRepId] INTEGER NOT NULL, CONSTRAINT [FK_CustomerSupportRepId] FOREIGN KEY ([SupportRepId]) REFERENCES "employees" ([EmployeeId]))	
employees	CREATE TABLE "employees" ([EmployeeId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [LastName] NVARCHAR(40) NOT NULL, [FirstName] NVARCHAR(40) NOT NULL, [Title] NVARCHAR(80) NOT NULL, [ReportsTo] INTEGER NOT NULL, [BirthDate] DATETIME NOT NULL, [HireDate] DATETIME NOT NULL, CONSTRAINT [FK_EmployeeReportsTo] FOREIGN KEY ([ReportsTo]) REFERENCES "employees" ([EmployeeId]))	
genres	CREATE TABLE "genres" ([GenreId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [Name] NVARCHAR(120) NOT NULL)	
invoice_items	CREATE TABLE "invoice_items" ([InvoiceItemId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [InvoiceId] INTEGER NOT NULL, [TrackId] INTEGER NOT NULL, [UnitPrice] NUMERIC(10,4) NOT NULL, [Quantity] INTEGER NOT NULL, [Discount] NUMERIC(10,4) NOT NULL, CONSTRAINT [FK_InvoiceInvoiceId] FOREIGN KEY ([InvoiceId]) REFERENCES "invoices" ([InvoiceId]))	
invoices	CREATE TABLE "invoices" ([InvoiceId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [CustomerId] INTEGER NOT NULL, [InvoiceDate] DATETIME NOT NULL, [BillingAddress] NVARCHAR(70) NOT NULL, [BillingCity] NVARCHAR(40) NOT NULL, [BillingState] NVARCHAR(40) NOT NULL, [BillingCountry] NVARCHAR(40) NOT NULL, [SupportRepId] INTEGER NOT NULL, CONSTRAINT [FK_InvoiceCustomerId] FOREIGN KEY ([CustomerId]) REFERENCES "customers" ([CustomerId]))	
media_types	CREATE TABLE "media_types" ([MediaTypeId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [Name] NVARCHAR(120) NOT NULL)	
playlist_track	CREATE TABLE "playlist_track" ([PlaylistId] INTEGER NOT NULL, [TrackId] INTEGER NOT NULL, CONSTRAINT [PK_PlaylistTrack] PRIMARY KEY ([PlaylistId], [TrackId]))	
playlists	CREATE TABLE "playlists" ([PlaylistId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [Name] NVARCHAR(120) NOT NULL)	
sqlite_sequence	CREATE TABLE sqlite_sequence(name,seq)	
sqlite_stat1	CREATE TABLE sqlite_stat1(tbl,idx,stat)	
tracks	CREATE TABLE "tracks" ([TrackId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, [AlbumId] INTEGER NOT NULL, [GenreId] INTEGER NOT NULL, [MediaTypeId] INTEGER NOT NULL, [Title] NVARCHAR(120) NOT NULL, [ArtistId] INTEGER NOT NULL, [Duration] NUMERIC(8,3) NOT NULL, [UnitPrice] NUMERIC(10,4) NOT NULL, [Discount] NUMERIC(10,4) NOT NULL, CONSTRAINT [FK_TrackAlbumId] FOREIGN KEY ([AlbumId]) REFERENCES "albums" ([AlbumId]))	
▼ Indices (10)		
IFK_AlbumArtistId	CREATE INDEX [IFK_AlbumArtistId] ON "albums" ([ArtistId])	
IFK_CustomerSupportRepId	CREATE INDEX [IFK_CustomerSupportRepId] ON "customers" ([SupportRepId])	
IFK_EmployeeReportsTo	CREATE INDEX [IFK_EmployeeReportsTo] ON "employees" ([ReportsTo])	
IFK_InvoiceCustomerId	CREATE INDEX [IFK_InvoiceCustomerId] ON "invoices" ([CustomerId])	
IFK_InvoiceLineInvoiceId	CREATE INDEX [IFK_InvoiceLineInvoiceId] ON "invoice_items" ([InvoiceId])	
IFK_InvoiceLineTrackId	CREATE INDEX [IFK_InvoiceLineTrackId] ON "invoice_items" ([TrackId])	
IFK_PlaylistTrackTrackId	CREATE INDEX [IFK_PlaylistTrackTrackId] ON "playlist_track" ([TrackId])	
IFK_TrackAlbumId	CREATE INDEX [IFK_TrackAlbumId] ON "tracks" ([AlbumId])	
IFK_TrackGenreId	CREATE INDEX [IFK_TrackGenreId] ON "tracks" ([GenreId])	
IFK_TrackMediaTypeId	CREATE INDEX [IFK_TrackMediaTypeId] ON "tracks" ([MediaTypeId])	

Database Structure		
Browse Data Edit Pragmas Execute SQL		
Table: albums <input type="text" value="Filter in any column"/>		
AlbumId	Title	ArtistId
Filter	Filter	Filter
1	1 For Those About To Rock We ...	1
2	2 Balls to the Wall	2
3	3 Restless and Wild	2
4	4 Let There Be Rock	1
5	5 Big Ones	3
6	6 Jagged Little Pill	4
7	7 Facelift	5
8	8 Warner 25 Anos	6
9	9 Plays Metallica By Four Cellos	7
10	10 Audioslave	8
11	11 Out Of Exile	8
12	12 BackBeat Soundtrack	9
13	13 The Best Of Billy Cobham	10
14	14 Alcohol Fueled Brewtality Live!...	11
15	15 Alcohol Fueled Brewtality Live!...	11
16	16 Black Sabbath	12
17	17 Black Sabbath Vol. 4 (Remaster)	12
18	18 Body Count	13
19	19 Chemical Wedding	14

Figure 2.16 The entries from chinook.db.

This figure shows the all the entries from the chinook.db. It represents a digital media store, including tables for artists, albums, media tracks, invoices and customers.

```
{} chronicleingamerica_firejson • api_04.py X
LAB_5 > api_04.py > ...
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
15 @app.route('/', methods=['GET'])
16 def home():
17     return '''<h1>Accessing Chinook Database</h1>
18     <p>A prototype API for accessing the chinook sqlite sample database.</p>'''
19
20
21
22 @app.route('/api/v1/resources/albums/all', methods=['GET'])
23 def api_all():
24     conn = sqlite3.connect('chinook.db')
25     conn.row_factory = dict_factory
26     cur = conn.cursor()
27     all_books = cur.execute('SELECT * FROM albums;').fetchall()
28
29     return jsonify(all_books)
```

Figure 2.17 Source code of ap3.py.

This figure shows the source code for ap4.py. In here, API was connected to the database chinook.db using the sqlite3 library. The database used is SQLite, a lightweight database engine that is supported in Python by default.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE: MESSAGES 2: Python
python.exe "d:/Kath/Documents/Kathleen (College)/SECOND YEAR (2019-2020)/4_Fourth Term/CPE106L/LAB_5/api_04.py"
* Serving Flask app "api_04" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 2.18 Running the ap4.py.

This figure shows some lines related to debugging which means that the flask is running the application locally at the address <http://127.0.0.1:5000/>.

Accessing Chinook Database

A prototype API for accessing the chinook sqlite sample database.

Figure 2.19 The home page when rendered in a browser.

The figure shows the working web application connected to chinook.db.

← → ↻ ⓘ 127.0.0.1:5000/api/v1/resources/albums/all ☆

```
[
  {
    "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": 3,
    "Title": "Big Ones"
  },
  {
    "AlbumId": 6,
    "ArtistId": 4,
    "Title": "Jagged Little Pill"
  },
  {
    "AlbumId": 7,
    "ArtistId": 5,
    "Title": "Facelift"
  },
]
```

Figure 2.20 The entries on the request made.

This figure shows the result of the first request which returned all the entries in the database, similar to the /all request implemented for the previous version of the API. It provided all the details of each entries such as its Album ID number, Artist ID number, and the title of the song.

```

26
27 @app.route('/', methods=['GET'])
28 def home():
29     return '''<h1>Distant Reading Archive</h1>
30 <p>A prototype API for distant reading of science fiction novels.</p>'''
31
32
33 # A route to return all of the available entries in our catalog.
34 @app.route('/api/v1/resources/books/all', methods=['GET'])
35 def api_all():
36     return jsonify(books)
37
38 @app.errorhandler(404)
39 def page_not_found(e):
40     return "<h1>404</h1><p>The resource could not be found.</p>", 404
41
42 app.run()

```

Figure 2.21 Edited source code of ap2.py.

This figure shows the edited ap2.py. The purpose of the page_not_found function is to create an error page seen by the user if they encounter an error or inputs a route that hasn't been defined.

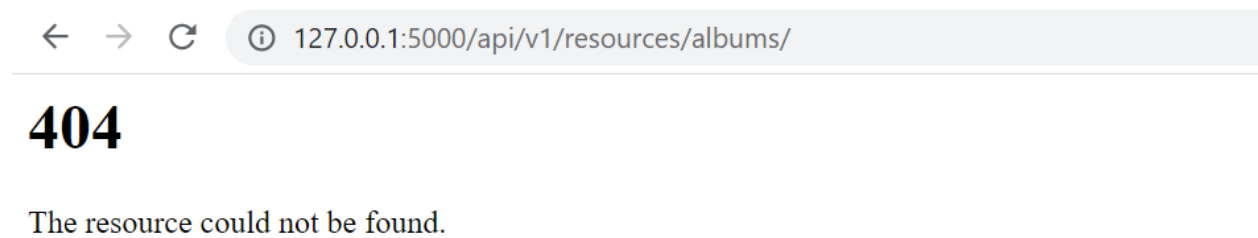


Figure 2.22 The 404 page when rendered in a browser.

This figure shows the result of using the page_not_found function. In HTML responses, the code 200 means "OK" wherein the expected data was transferred, while the code 404 means "Not Found" wherein there are no resources available at the URL given. This function allows to return 404 pages when something goes wrong in the application.

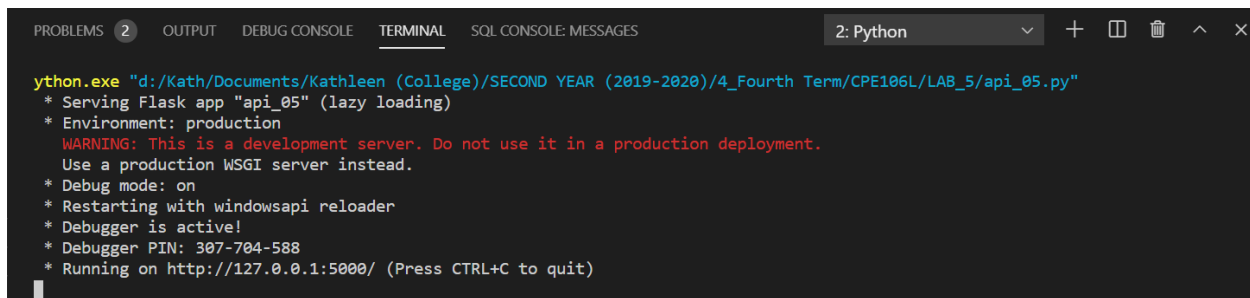
```

{} chronicleingamerica_fire.json • api_05.py •
LAB_5 > api_05.py > ...
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[idx]] = row[idx]
12     return d
13
14 @app.route('/', methods=['GET'])
15 def home():
16     return '''<h1>Accessing Chinook Database</h1>
17     <p>A prototype API for accessing the chinook sqlite sample database.</p>'''
18
36 def api_filter():
37     query_parameters = request.args
38
39     id = query_parameters.get('albumid')
40     title = query_parameters.get('title')
41
42     query = "SELECT * FROM albums WHERE"
43     to_filter = []
44
45     if id:
46         query += ' albumid=? AND'
47         to_filter.append(id)
48     if title:
49         query += ' title=? AND'
50         to_filter.append(title)
51     if not (id or title):
52         return page_not_found(404)
53
54     query = query[:-4] + ';'
55
56     conn = sqlite3.connect('chinook.db')
57     conn.row_factory = dict_factory
58     cur = conn.cursor()
59
60     results = cur.execute(query, to_filter).fetchall()
61
62     return jsonify(results)
63
64 app.run()

```

Figure 2.23 Source code of ap5.py.

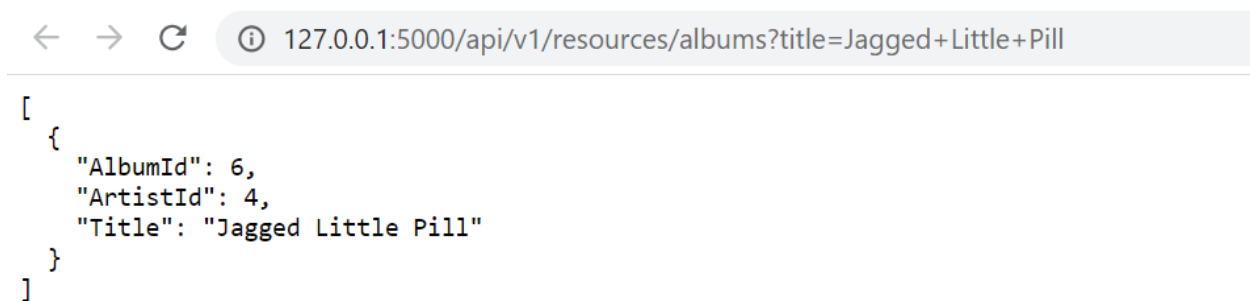
This figure shows the source code of ap5.py. As observed, this version of API serves a larger number of results that are stored in an SQLite database (chinook.db). When the user requests an entry or set of entries, the API pulls the information from the database by building and executing an SQL query. This iteration of the API also allows filtering by more than one field.



```
python.exe "d:/Kath/Documents/Kathleen (College)/SECOND YEAR (2019-2020)/4_Fourth Term/CPE106L/LAB_5/api_05.py"
* Serving Flask app "api_05" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 307-704-588
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 2.24 Running the ap5.py.

This figure shows that the server is running and to view the data in the browser, two different URLs were visited.

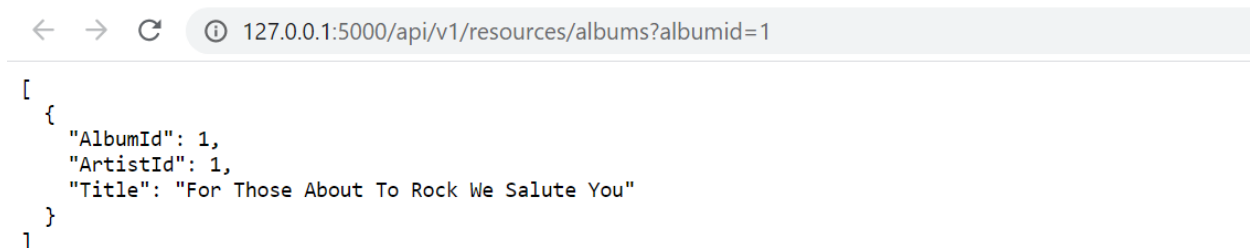


```
<  →  ↻  ⓘ  127.0.0.1:5000/api/v1/resources/albums?title=Jagged+Little+Pill

[
  {
    "AlbumId": 6,
    "ArtistId": 4,
    "Title": "Jagged Little Pill"
  }
]
```

Figure 2.25 The entry on the first request.

This figure shows all the data about the song Jagged Little Pill after visiting the URL <http://127.0.0.1:5000/api/v1/resources/albums?title=Jagged+Little+Pill>. As observed, the title of the song was used to filter the results of the request.



```
<  →  ↻  ⓘ  127.0.0.1:5000/api/v1/resources/albums?albumid=1

[
  {
    "AlbumId": 1,
    "ArtistId": 1,
    "Title": "For Those About To Rock We Salute You"
  }
]
```

Figure 2.26 The entry on the second request.

This figure shows all the data about the song For Those About to Rock We Salute You after visiting the URL <http://127.0.0.1:5000/api/v1/resources/albums?albumid=1>. As observed, the Album ID number was used to filter the results of the request.

OneDrive:

Group 1: <https://bit.ly/33j71nK>

Github:

Darrel Virtusio: <https://bit.ly/2PgDOSn>

Hannah Antaran: <https://bit.ly/3hYjP7b>

Kathleen Tupas: <https://bit.ly/2DrsGzk>