

# LEARN AND CREATE YOUR FIRST API USING PYTHON AND FLASK



BY ISMAIL CISSE

COPYRIGHT © 2024 ISMAIL CISSE

ALL RIGHTS RESERVED.

**TOOLS TO DOWNLOAD BEFORE STARTING:**

**PYTHON**

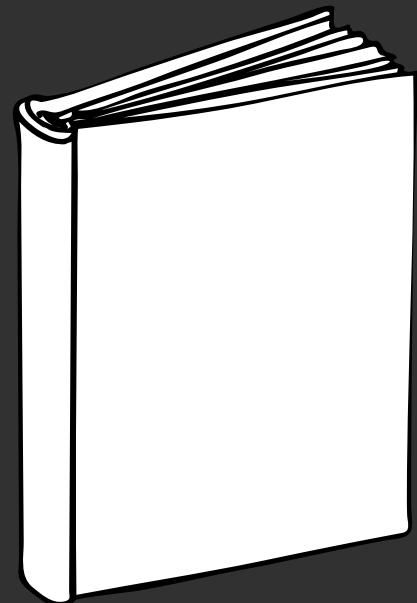
**POSTMAN**

**VISUAL STUDIO CODE**

# WHAT IS AN API?

DO YOU LIKE READING? WELL,  
WHETHER YOU DO OR DON'T, IT'S  
OKAY—BOOKS WON'T JUDGE YOU,  
 BUT THEY MIGHT JUST  
CHANGE YOUR MIND!

**AN API IS LIKE THE LIBRARIAN AT A LIBRARY IT  
TAKES YOUR REQUESTS(WHAT BOOK YOU WANT)**



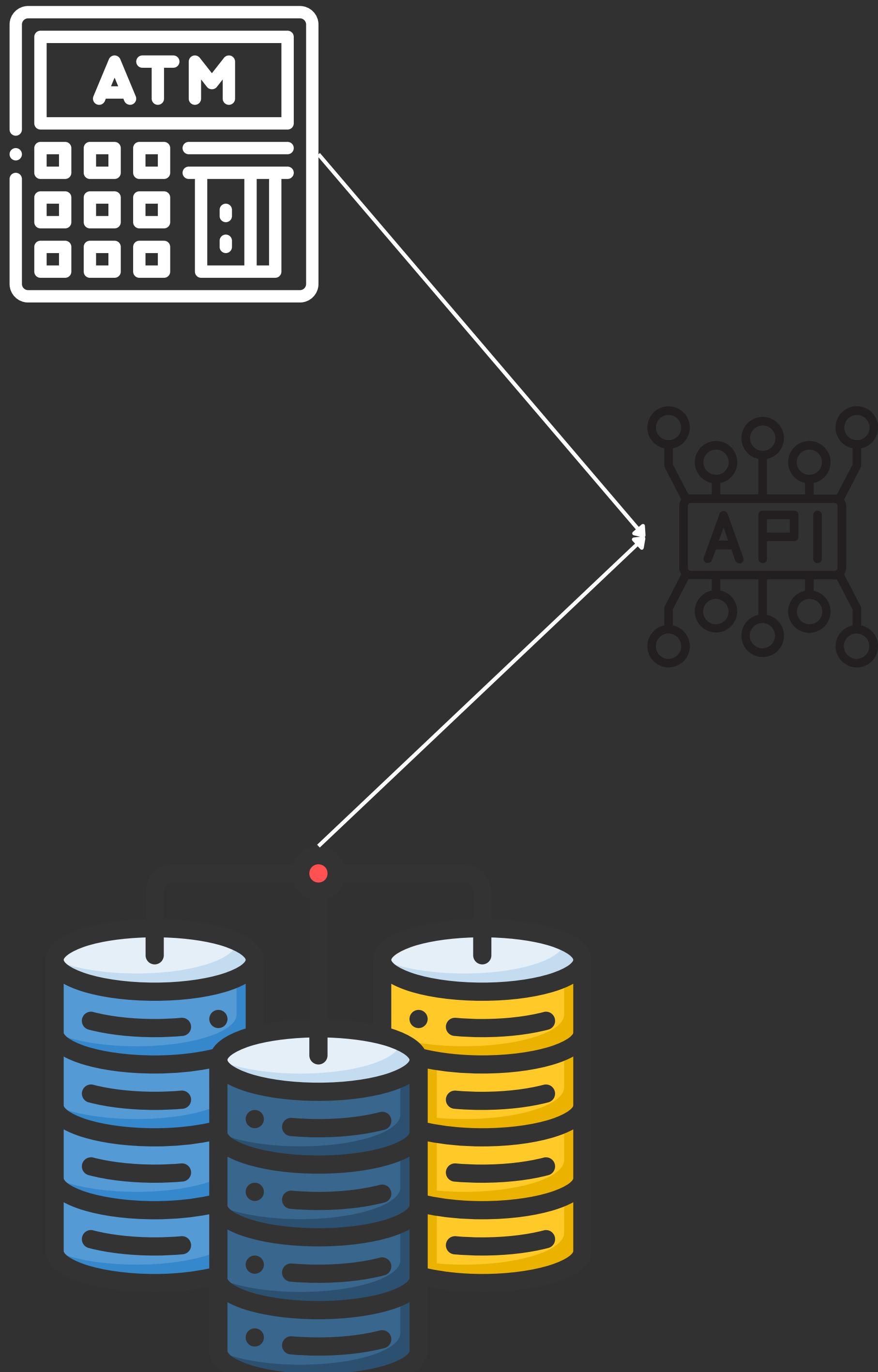
**COMMUNICATE WITH THE STACKS  
(SERVER/DATABASE)**



**AND BRINGS BACK THE RESPONSE(YOUR BOOK) ALL  
WHILE YOU COMFORTABLY WAIT AT THE COUNTER  
(APPLICATION).**



# A REAL LIFE EXAMPLE IS LIKE USING AN ATM.



BANK SERVER

- YOU AS THE USER INTERACT WITH THE ATM'S INTERFACE
- THE ATM ACTS AS A CLIENT, IT SENDS REQUEST TO THE BANK'S API.
- THE BANK'S API WILL PROCESS THE REQUEST, COMMUNICATES WITH THE BANK'S DATABASE, AND SENDS BACK A RESPONSE TO THE ATM.

HTTP(HYPER, TEXT TRANSFER PROTOCOL) IS USED TO TRANSMIT DATA OVER THE INTERNET AND DEFINES HOW WEB SERVERS AND BROWSERS SHOULD RESPOND TO VARIOUS COMMANDS. THE MOST POPULAR COMMANDS ARE:

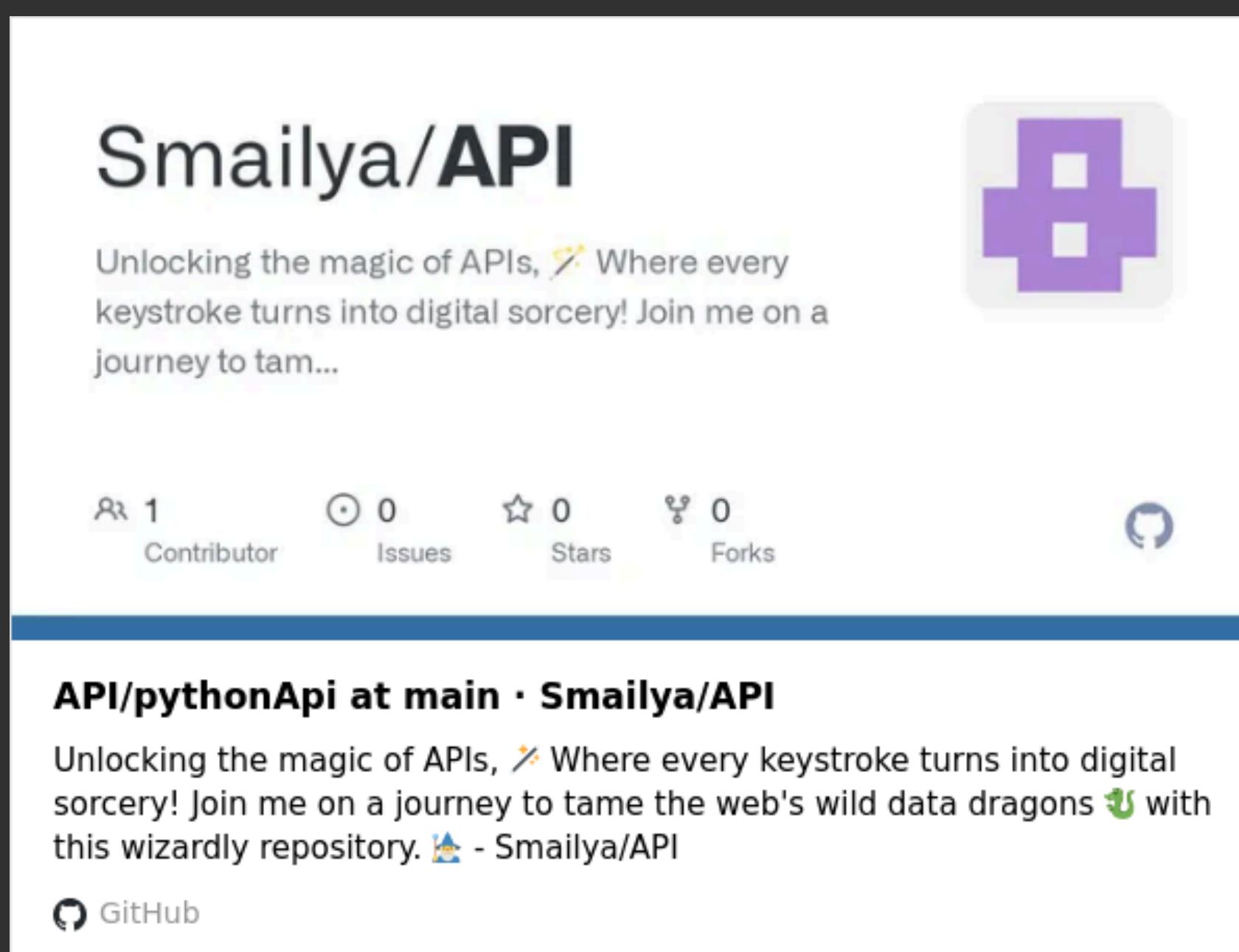
GET	#REQUEST DATA FROM A RESOURCE
POST	#CREATE A RESOURCE
PUT	#UPDATE A RESOURCE
DELETE	#DELETE A RESOURCE

**GET READY TO EMBARK  
ON THIS EPIC JOURNEY  
I'LL BE GANDALF THE  
CREAT, GUIDING OUR  
WAY TO THE MISTY  
CODING, AND YOU'RE  
OUR BRAVE BILBO  
BAGGINS!**



DOUBLE CLICK THE IMAGE OR GO TO THE FOLLOWING  
LINK TO DOWNLOAD THE ENTIRE CODE FROM MY  
GITHUB.

[HTTPS://GITHUB.COM/SMAILYA/API/TREE/MAIN/PYTHONAPI](https://github.com/smailya/API/tree/main/pythonapi)



DOWNLOAD THE FILE `main.py`

AFTER DOWNLOADING YOU WILL SEE THAT I EXPLAINED EACH LINE OF CODE TO MAKE IT EASY FOR YOU TO UNDERSTAND WHAT'S REALLY GOING ON.

NOW OPEN VISUAL STUDIO

CREATE A NEW FOLDER YOU NAME IT API

CREATE A NEW FILE NAME IT main.py

OPEN YOUR TERMINAL AND TYPE:

```
pip install flask| ENTER
```

IT WILL INSTALL THE FLASK FRAMEWORK FOR YOU, FLASK IS THE SERVER FROM WHICH YOU WILL BE RUNNING YOUR API

NOW I WANT YOU TO TYPE OUT EVERYTHING FROM THE FILE main.py

RESIST THE URGE TO COPY-PASTE! LET THOSE FINGERS DANCE ON THE KEYS—TYPING IS THE PATH TO TRUE UNDERSTANDING!

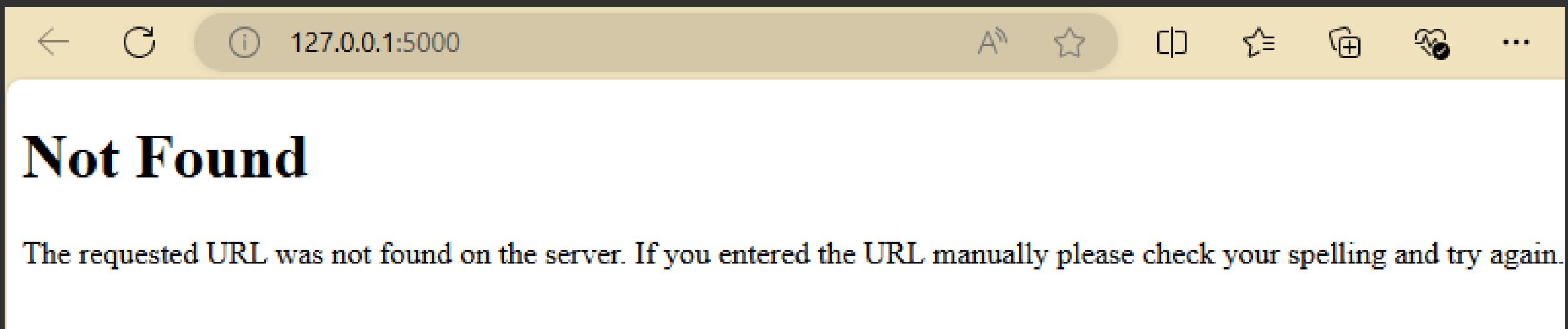
WHEN YOU DONE TYPING TO RUN THE SERVER TYPE

```
\API> python main.py| ENTER
```

YOU WILL SEE A URL SIMILAR TO THE ONE BELOW OPEN IT IN YOUR BROWSER

```
* Running on http://127.0.0.1:5000
```

# OPEN YOUR BROWSER YOU WILL SEE THE FOLLOWING:



IT'S BECAUSE YOU DON'T HAVE A HOME ROUTE IF YOU WANT YOU CAN DEFINE A FUNCTION FOR THAT. BY TYPING

```
@app.route("/", methods=["GET"])
def home():
    return "home"
```

EXPLANATIONS OF THE CODE:

# @app.route USING THE ROUTE() DECORATOR TO BIND A FUNCTION TO A URL.

("/", METHODS=["GET"]) INDICATES THAT THE FOLLOWING FUNCTION HANDLES GET REQUESTS TO THE ROOT URL ("/")

def home(): # DEFINING A VIEW FUNCTION NAMED 'HOME' THAT WILL BE CALLED WHEN THE ROOT URL IS ACCESSED.

return "home" # RETURNING A SIMPLE STRING "HOME" AS THE RESPONSE TO THE GET REQUEST.

RUN THE SERVER AGAIN YOU WILL SEE IT DISPLAY HOME. NOW ADD THE FOLLOWING TO YOUR URL: get-user/422

IT DOESN'T MATTER WHAT USER\_ID IS PUT IN THE URL PATH BECAUSE THE GET\_USER FUNCTION IS HARD-CODED TO RETURN THE SAME DATA FOR ANY USER\_ID. SPECIFICALLY, THE USER\_DATA DICTIONARY ALWAYS CONTAINS THE SAME NAME AND EMAIL VALUES REGARDLESS OF THE USER\_ID. ONLY THE USER\_ID FIELD IN THE RESPONSE CHANGES TO REFLECT THE USER\_ID PROVIDED IN THE URL.

YOU WILL SEE THE FOLLOWING JSON RESPONSE IN YOUR BROWSER OR API CLIENT:

```
{  
    "email": "Bilbo.baggins@example.com",  
    "name": "Bilbo Baggins",  
    "user_id": "2"  
}
```

THE FLASK APPLICATION MATCHES THE URL /GET-USER/2 TO THE ROUTE DEFINED BY THE @APP.ROUTE("/GET-USER/<USER\_ID>") DECORATOR.

THE GET\_USER FUNCTION IS CALLED WITH USER\_ID SET TO 2.

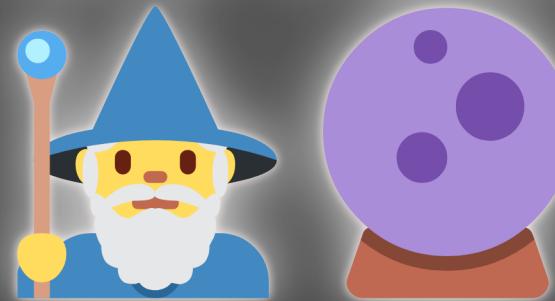
INSIDE THE GET\_USER FUNCTION, A DICTIONARY USER\_DATA IS CREATED WITH THE KEYS "USER\_ID", "NAME", AND "EMAIL", AND THEIR CORRESPONDING VALUES.

THE FUNCTION CHECKS FOR AN OPTIONAL QUERY PARAMETER EXTRA USING REQUEST.ARGS.GET("EXTRA"). IF THE EXTRA PARAMETER IS PROVIDED IN THE REQUEST, IT IS ADDED TO THE USER\_DATA DICTIONARY.

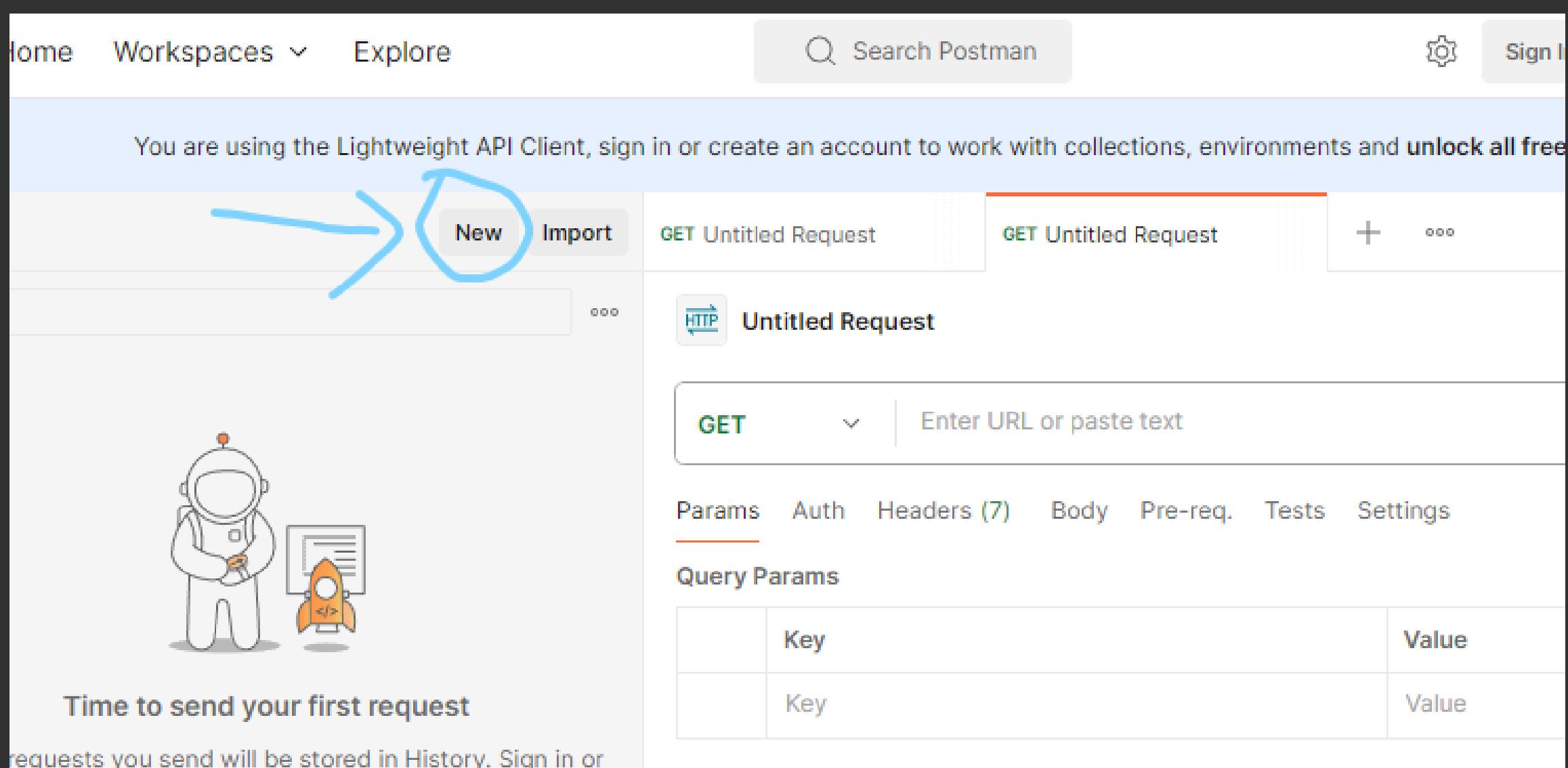
THE USER\_DATA DICTIONARY IS CONVERTED TO A JSON RESPONSE USING JSONIFY(USER\_DATA) AND RETURNED TO THE CLIENT WITH A STATUS CODE 200 (OK).

IF YOU INCLUDE AN EXTRA QUERY PARAMETER IN THE REQUEST, SUCH AS HTTP://127.0.0.1:5000/GET-USER/2?EXTRA=SOME\_VALUE, THE JSON RESPONSE WILL INCLUDE THE EXTRA FIELD:

**ALRIGHT, FOLKS, BRACE  
YOURSELVES—IT'S  
TIME TO SUMMON OUR  
MYSTICAL POWERS AND  
UNLEASH OUR MAGIC  
WAND API TESTER,  
POSTMAN!**



# CLICK ON NEW THEN SELECT HTTP

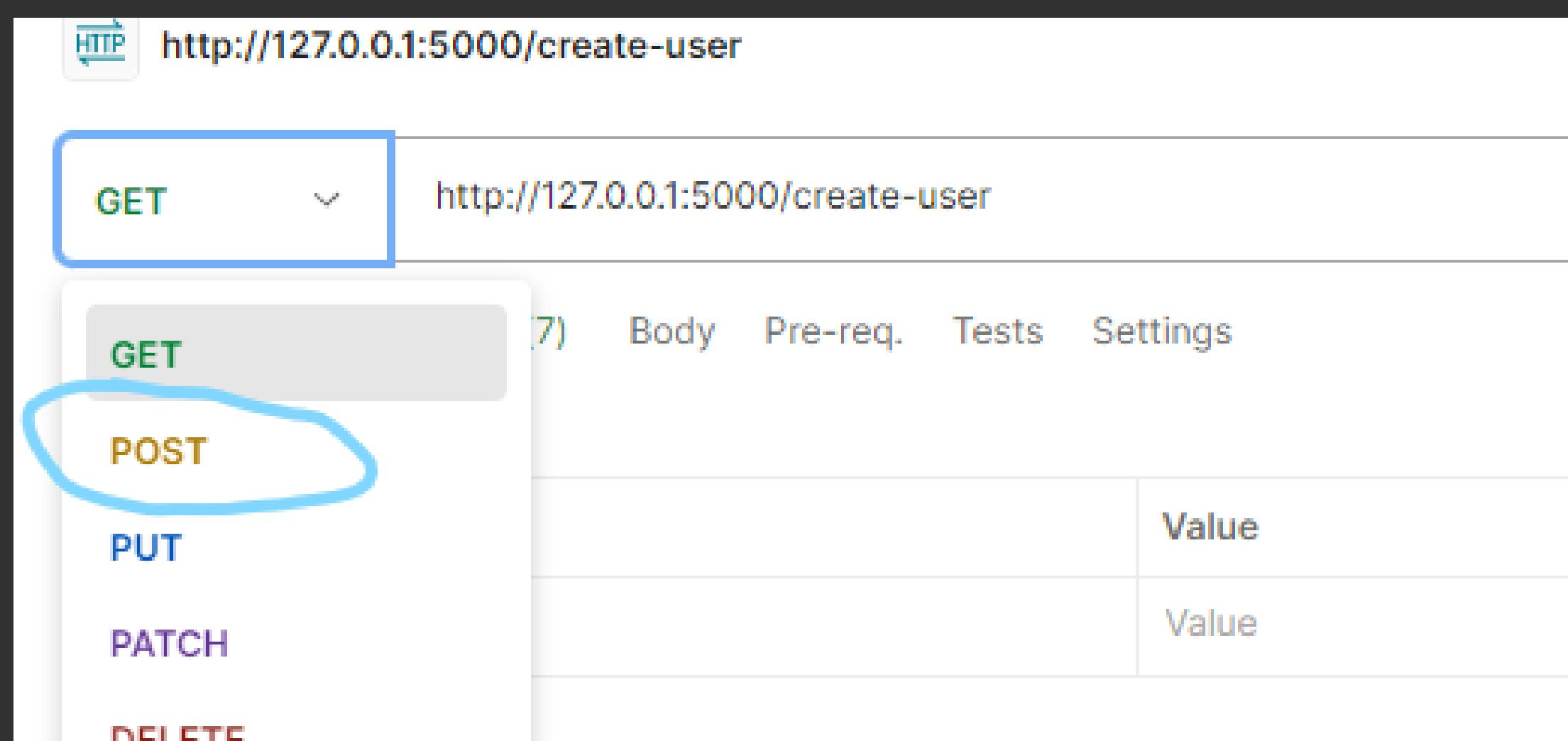


The screenshot shows the Postman application interface. At the top, there are navigation links for Home, Workspaces, Explore, and a search bar. Below the header, a message encourages users to sign in or create an account. A large blue arrow points from the left towards the 'New' button in the top-left corner of the main workspace. The 'New' button is highlighted with a blue circle. To its right is the 'Import' button. The main workspace displays a request titled 'Untitled Request' with a GET method selected. The URL field contains 'http://127.0.0.1:5000/Untitled Request'. Below the URL, tabs for Params, Auth, Headers (7), Body, Pre-req., Tests, and Settings are visible. The Headers tab is currently active. Under Headers, there is a section for 'Query Params' with two rows: one for 'Key' and one for 'Value'. On the left side of the workspace, there is a cartoon illustration of an astronaut and a rocket.

COPY AND PASTE YOUR LOCAL SERVER URL INTO POSTMAN.

ADD THE FOLLOWING TO IT: /CREATE-USER

CHANGE THE METHOD TO BE POST



The screenshot shows the Postman interface with a blue arrow pointing from the left towards the method dropdown menu. The method dropdown is currently set to 'GET', which is highlighted with a blue box. Below the dropdown, a list of other methods is visible: 'POST' (circled with a blue oval), 'PUT', 'PATCH', and 'DELETE'. The URL field above the dropdown contains 'http://127.0.0.1:5000/Untitled Request'. To the right of the URL, there are tabs for Params, Auth, Headers (7), Body, Pre-req., Tests, and Settings. The Headers tab is active. The Headers table has two rows: one for 'Key' and one for 'Value'. The Body tab is also visible on the right side of the interface.

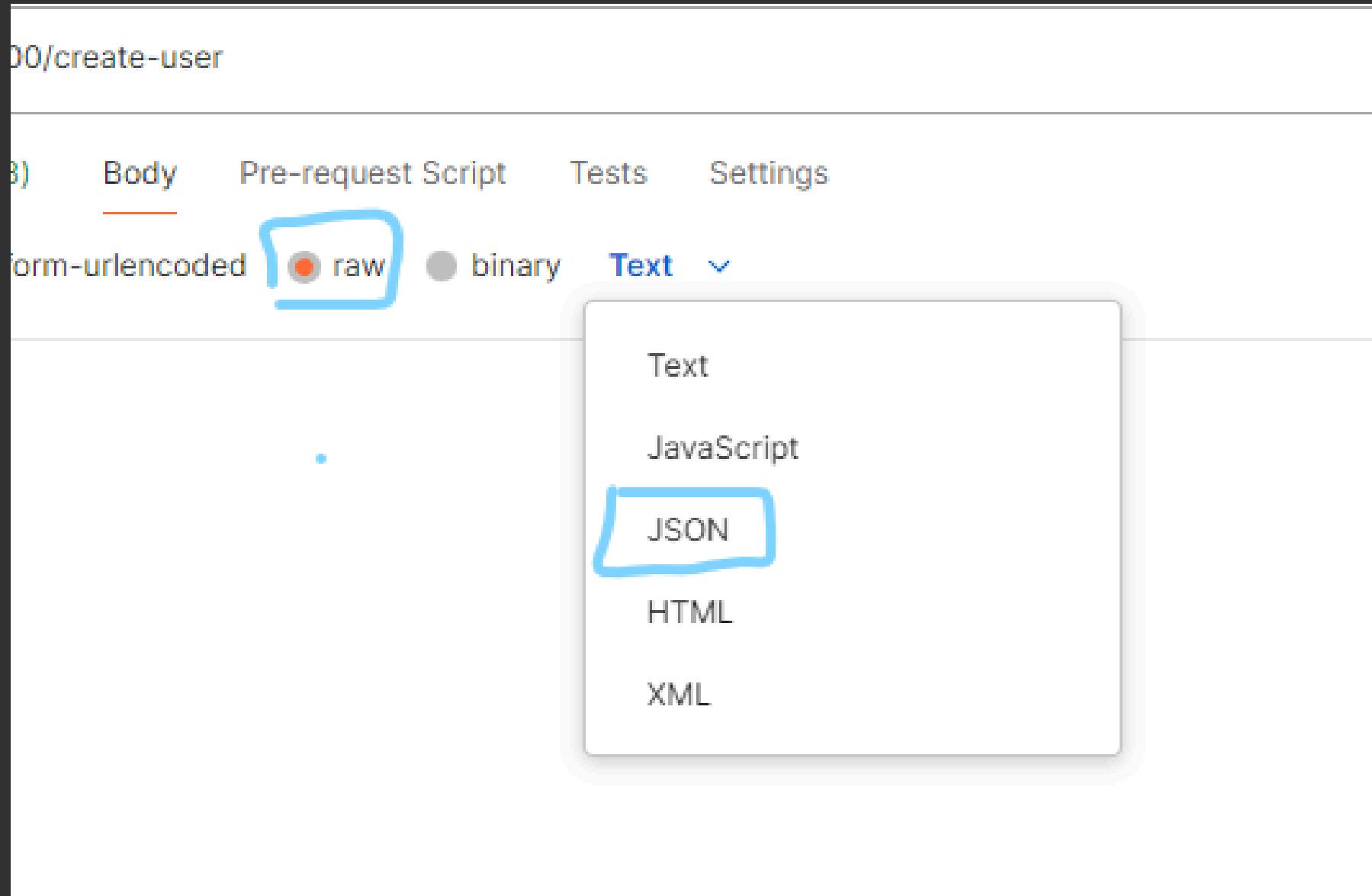
THEN GO TO BODY TO INSERT JSON DATA, SELECT RAW, CHANGE TEXT TO JSON, YOU CAN PASS A JSON OBJECT THE FOLLOWING WAY

http://127.0.0.1:5000/create-user

Body Pre-request Script Tests Settings

form-urlencoded  raw  binary Text ▾

Text  
JavaScript  
**JSON**  
HTML  
XML



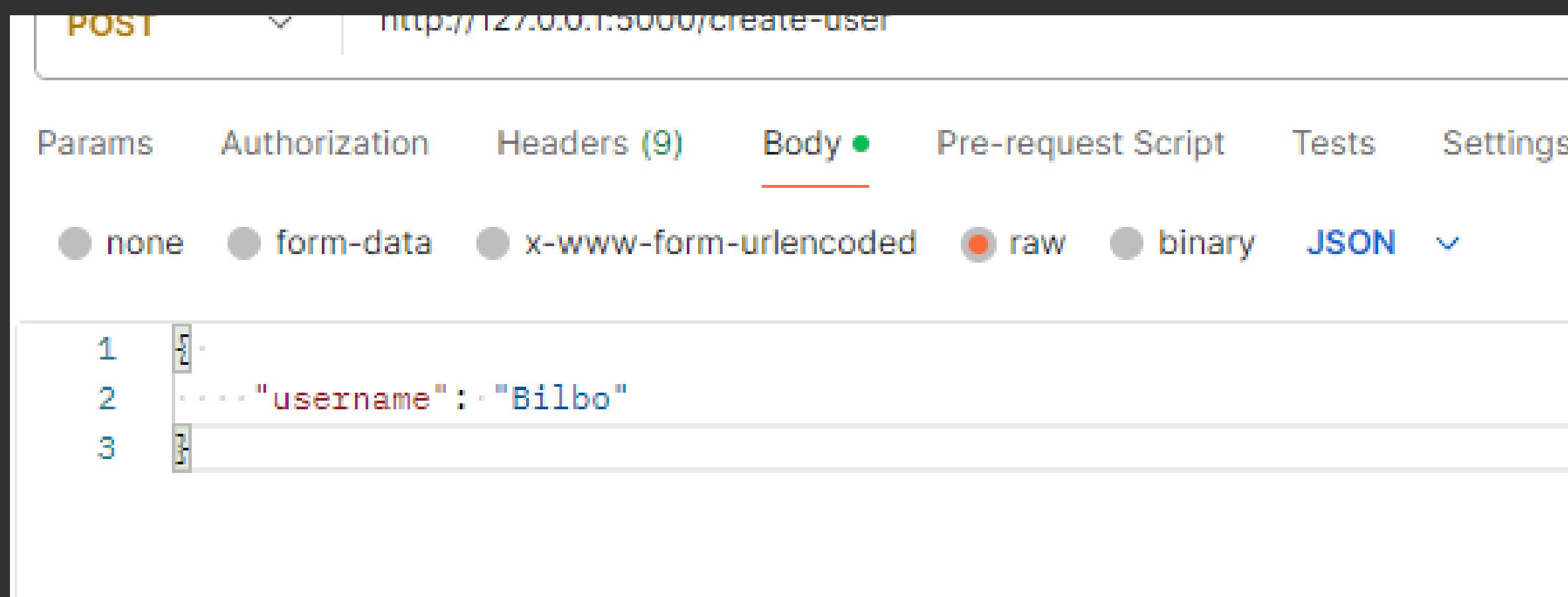
## YOU CAN PASS A JSON OBJECT THE FOLLOWING WAY

POST | http://127.0.0.1:5000/create-user

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary **JSON** ▾

```
1 {  
2   "username": "Bilbo"  
3 }
```

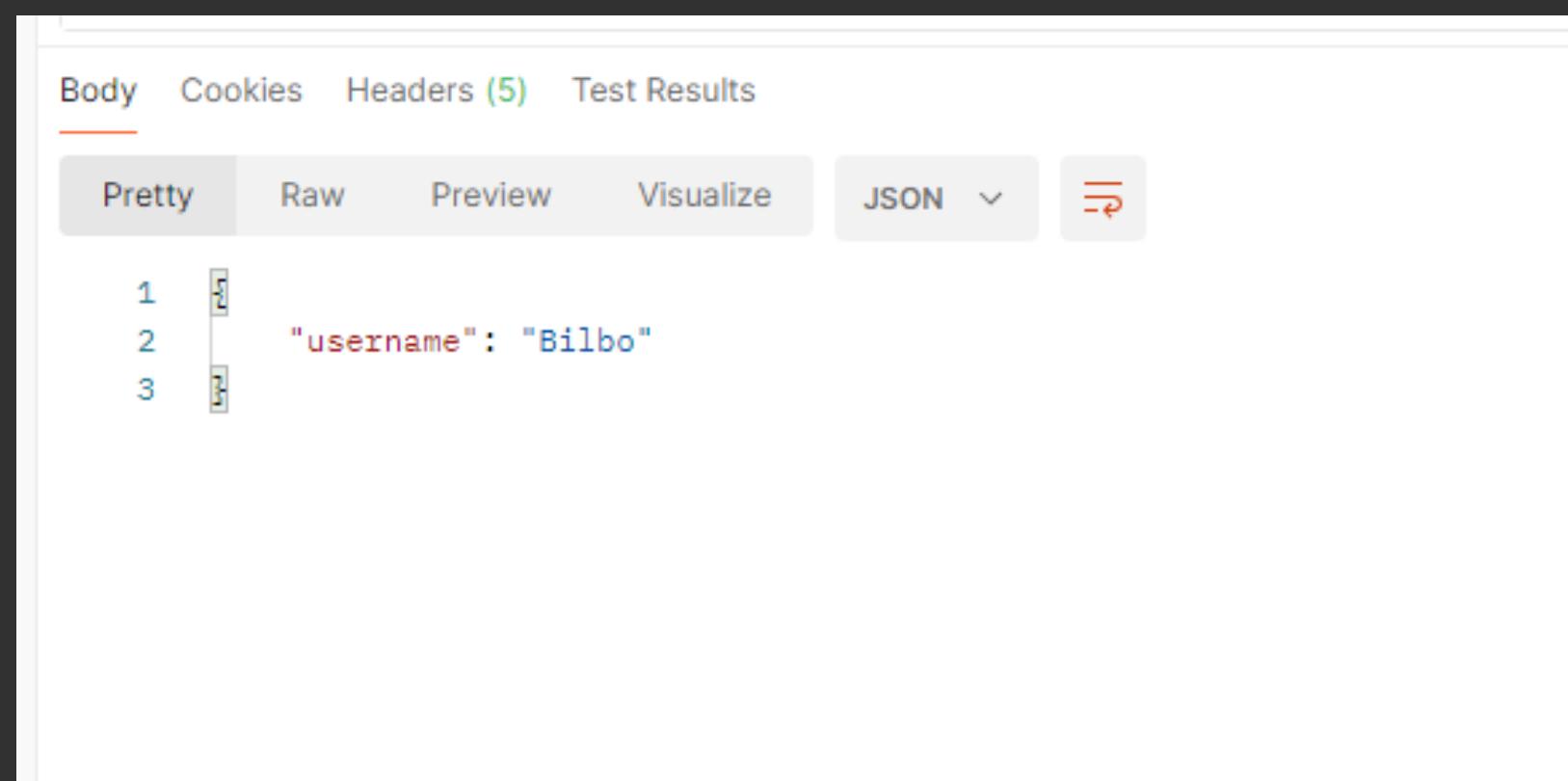


PRESS SEND, YOU WILL SEE SOMETHING DO

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▾

```
1 {  
2   "username": "Bilbo"  
3 }
```



**CONGRATULATIONS ON COMPLETING THE BOOK!**

AS A FELLOW DEVELOPER, I KNOW THAT CODING CAN SOMETIMES BE CHALLENGING. BUT DON'T LET THAT STOP YOU! KEEP GOING, KEEP PUSHING FORWARD. THE MORE YOU PRACTICE, THE BETTER YOU'LL BECOME AT IT. ONE THING THAT HAS HELPED ME IS TO TRY TO HAVE FUN WHILE CODING. WHEN YOU ENJOY THE PROCESS, IT CAN MAKE THE CHALLENGES FEEL LESS DAUNTING.

I SPENT A LOT OF TIME PUTTING THIS BOOK TOGETHER I HOPE THAT YOU FOUND A LOT OF VALUE FROM IT. FEEL FREE TO SHARE IT.

IF YOU HAVE ANY QUESTIONS ABOUT THE BOOK OR PROGRAMMING IN GENERAL, MY DOOR IS ALWAYS OPEN. YOU CAN REACH ME ON MY LINKEDIN PROFILE AT [HTTPS://WWW.LINKEDIN.COM/IN/ISMAIL-CISSE/](https://www.linkedin.com/in/ismail-cisse/). I'M HERE TO HELP YOU, SO DON'T HESITATE TO CONNECT WITH ME IF YOU NEED ANY ASSISTANCE.

