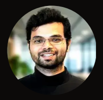
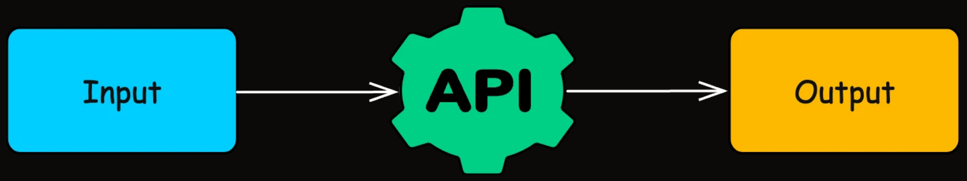
What's an API?



Ashish Pratap Singh

API stands for **Application Programming Interface**.

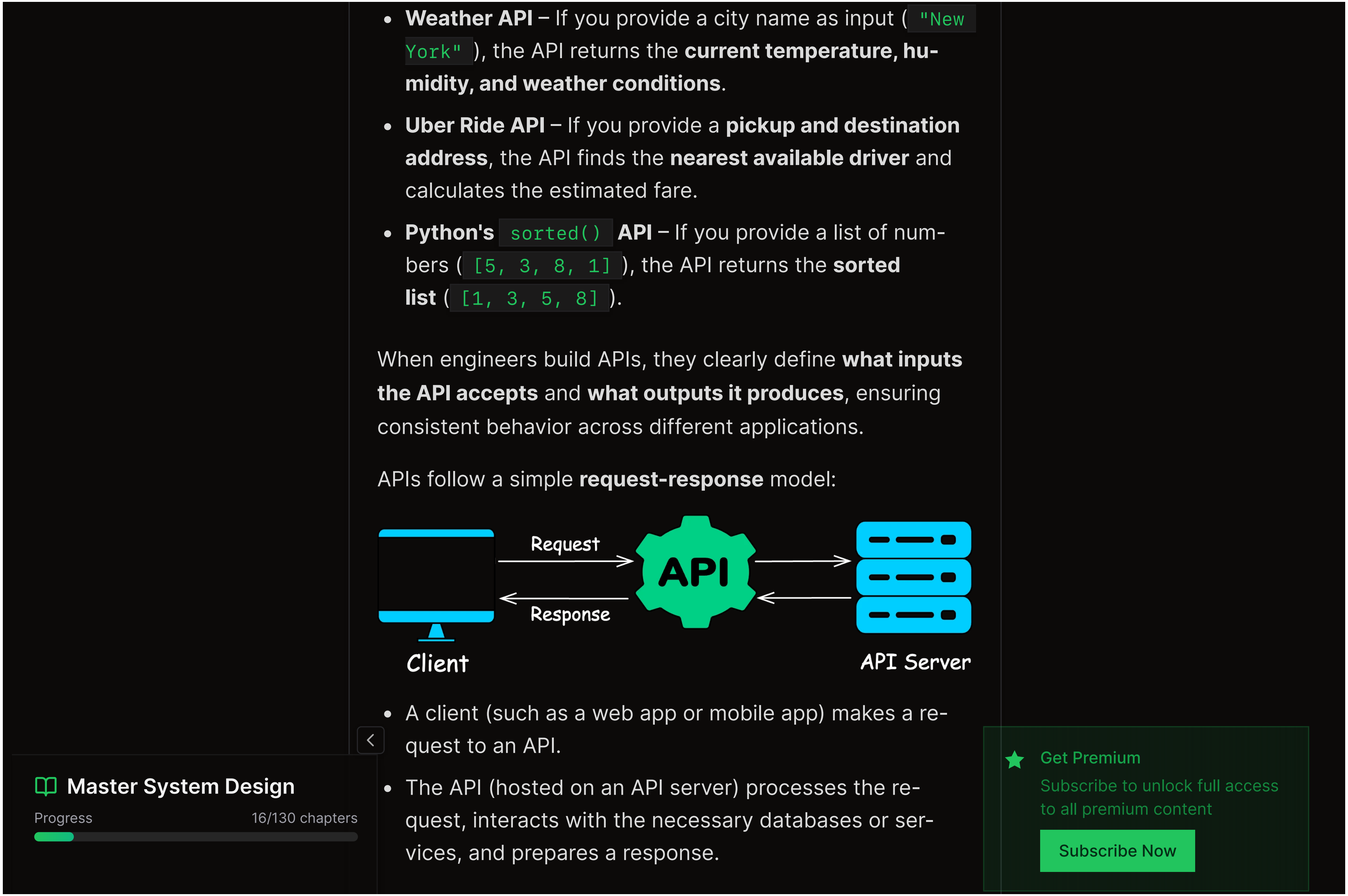
At its core, an API is a **bunch of code** that takes an **input** and gives you predictable **outputs.**



Think of an API as a **middleman** that enables applications to interact **without needing direct access to each other's code or database**.

Almost every digital service you use today—social media, e-commerce, online banking, ride-hailing apps—all of them are a bunch of APIs working together.

Examples:

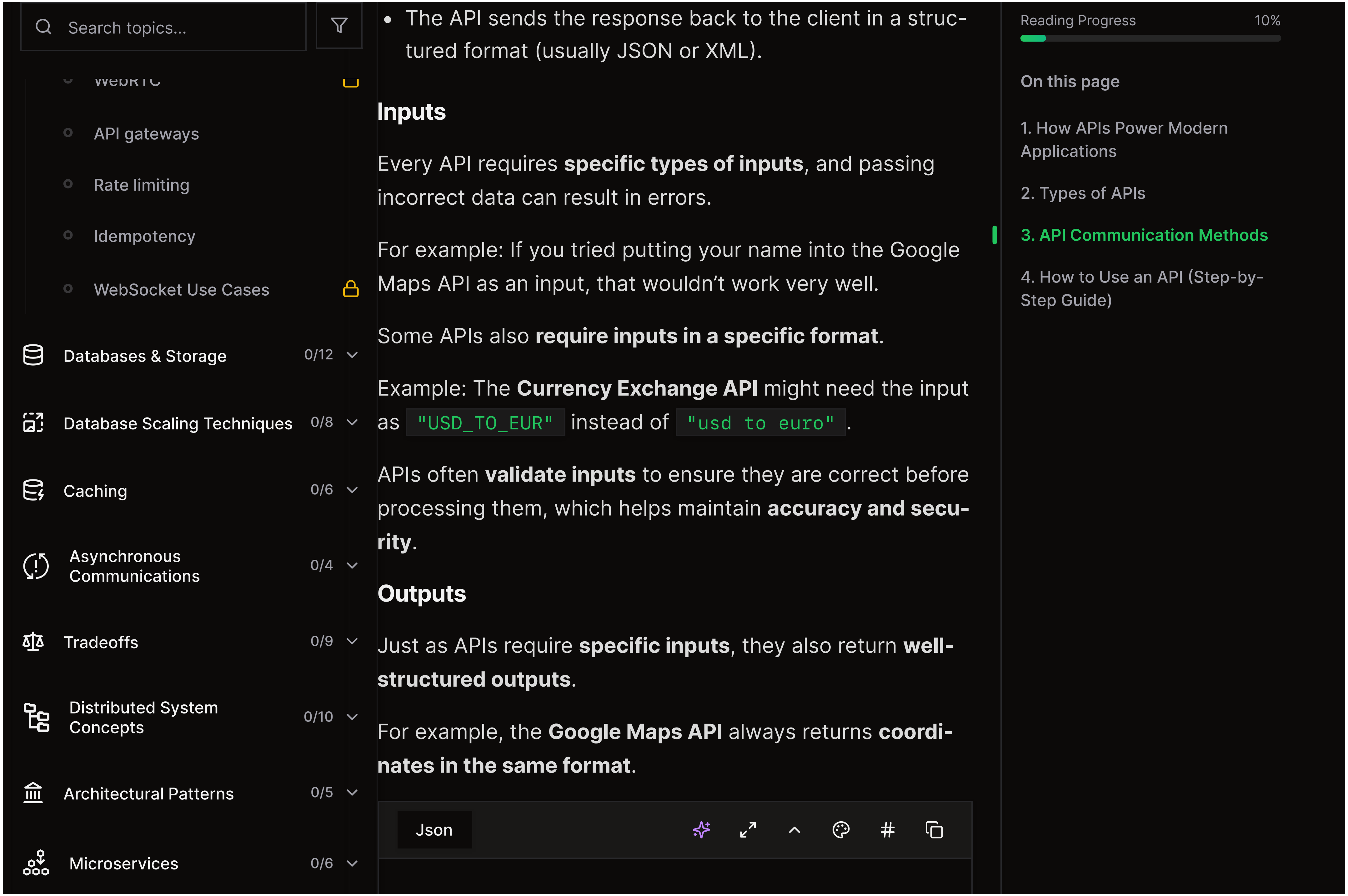


- **Weather API** – If you provide a city name as input (`"New York"`), the API returns the **current temperature, humidity, and weather conditions**.
- **Uber Ride API** – If you provide a **pickup and destination address**, the API finds the **nearest available driver** and calculates the estimated fare.
- **Python's** `sorted()` **API** – If you provide a list of numbers (`[5, 3, 8, 1]`), the API returns the **sorted list** (`[1, 3, 5, 8]`).

When engineers build APIs, they clearly define **what inputs the API accepts** and **what outputs it produces**, ensuring consistent behavior across different applications.

APIs follow a simple **request-response** model:

- A client (such as a web app or mobile app) makes a request to an API.
- The API (hosted on an API server) processes the request, interacts with the necessary databases or services, and prepares a response.



Search topics...

WebRTC

API gateways

Rate limiting

Idempotency

WebSocket Use Cases

Databases & Storage 0/12

Database Scaling Techniques 0/8

Caching 0/6

Asynchronous Communications 0/4

Tradeoffs 0/9

Distributed System Concepts 0/10

Architectural Patterns 0/5

Microservices 0/6

- The API sends the response back to the client in a structured format (usually JSON or XML).

Inputs

Every API requires **specific types of inputs**, and passing incorrect data can result in errors.

For example: If you tried putting your name into the Google Maps API as an input, that wouldn’t work very well.

Some APIs also **require inputs in a specific format**.

Example: The **Currency Exchange API** might need the input as `"USD_TO_EUR"` instead of `"usd to euro"`.

APIs often **validate inputs** to ensure they are correct before processing them, which helps maintain **accuracy and security**.

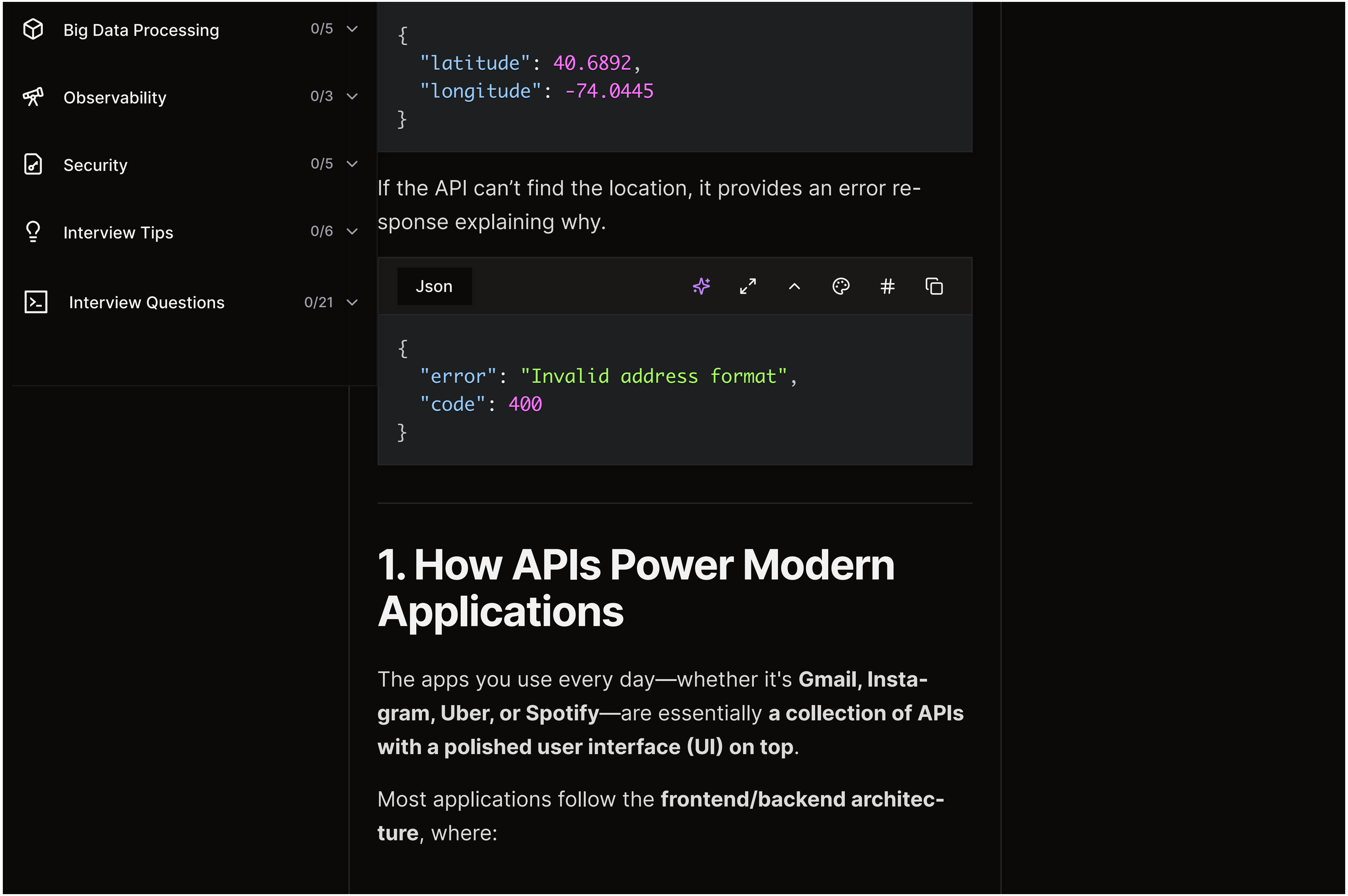
Outputs

Just as APIs require **specific inputs**, they also return **well-structured outputs**.

For example, the **Google Maps API** always returns **coordinates in the same format**.

Reading Progress 10%

On this page



```
{
  "latitude": 40.6892,
  "longitude": -74.0445
}
```

If the API can’t find the location, it provides an error response explaining why.

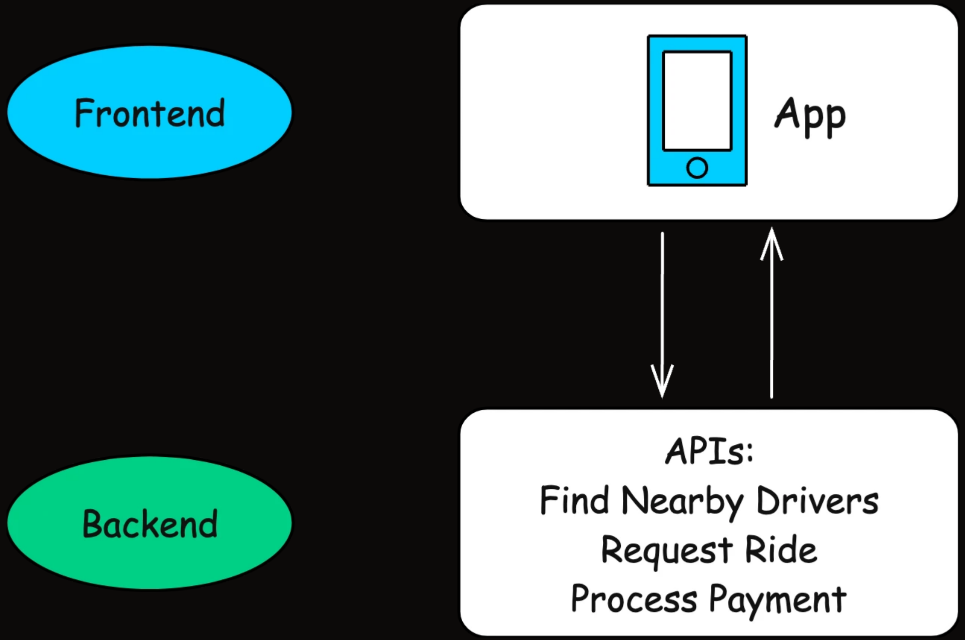
```
{
  "error": "Invalid address format",
  "code": 400
}
```

1. How APIs Power Modern Applications

The apps you use every day—whether it's **Gmail, Instagram, Uber, or Spotify**—are essentially **a collection of APIs with a polished user interface (UI) on top**.

Most applications follow the **frontend/backend architecture**, where:

- The **backend** consists of APIs that handle **data processing, business logic, and communication with databases**.
- The **frontend** is a **graphical user interface (GUI)** that interacts with these APIs, making applications user-friendly and accessible **without requiring users to write code**.



Let's break this down with a real-world example: **Uber**.

The Backend

Before the Uber app existed as a sleek, user-friendly experience, the company first built **the core APIs that power ride-hailing services:**

- Finding Nearby Drivers
- Calculating Fares & Routes
- Process Payment
- Real-Time Tracking
- Matching Riders & Drivers

These APIs run on Uber’s servers, forming the **backend infrastructure**. Every time you request a ride, track your driver, or make a payment, these backend APIs handle the request.

Backend engineers are responsible for optimizing these APIs, improving ride-matching algorithms, securing transactions, and ensuring a smooth experience for millions of users.

The Frontend

The backend APIs handle **all the complex logic**, but they **only work through code**—which isn't practical for everyday users. That’s why companies build a **frontend (user interface)** on top of these APIs, allowing users to interact with the system **visually and intuitively**.

Example: When you enter your pickup & destination address, the frontend sends an API request to **find nearby drivers** and displays available cars.

Once the trip is complete, the frontend may call the process payment API to display the receipt.

---

2. Types of APIs

APIs come in different forms depending on **who can access them**, **how they are used**, and **what purpose they serve**.

1. Open APIs (Public APIs)

Open APIs, also known as **Public APIs**, are accessible to external developers with minimal restrictions.

Companies provide these APIs to encourage **third-party developers** to integrate their services and build new applications on top of them.

Example: YouTube Data API

Normally, when you use the **YouTube app**, it makes **internal API calls** to fetch your video feed, search for content, or post comments. However, YouTube also provides a **public API** that allows developers to access some of this functionality **outside of the app**.

For example, the **YouTube Search API** allows developers to fetch video results based on a keyword. If you send a request to the API with `"machine learning tutorial"` as the search term, it will return a structured response (JSON format) containing a list of relevant videos, including **titles, descriptions, thumbnails, and video links**.

This is incredibly useful because it enables developers to build custom applications on top of YouTube.

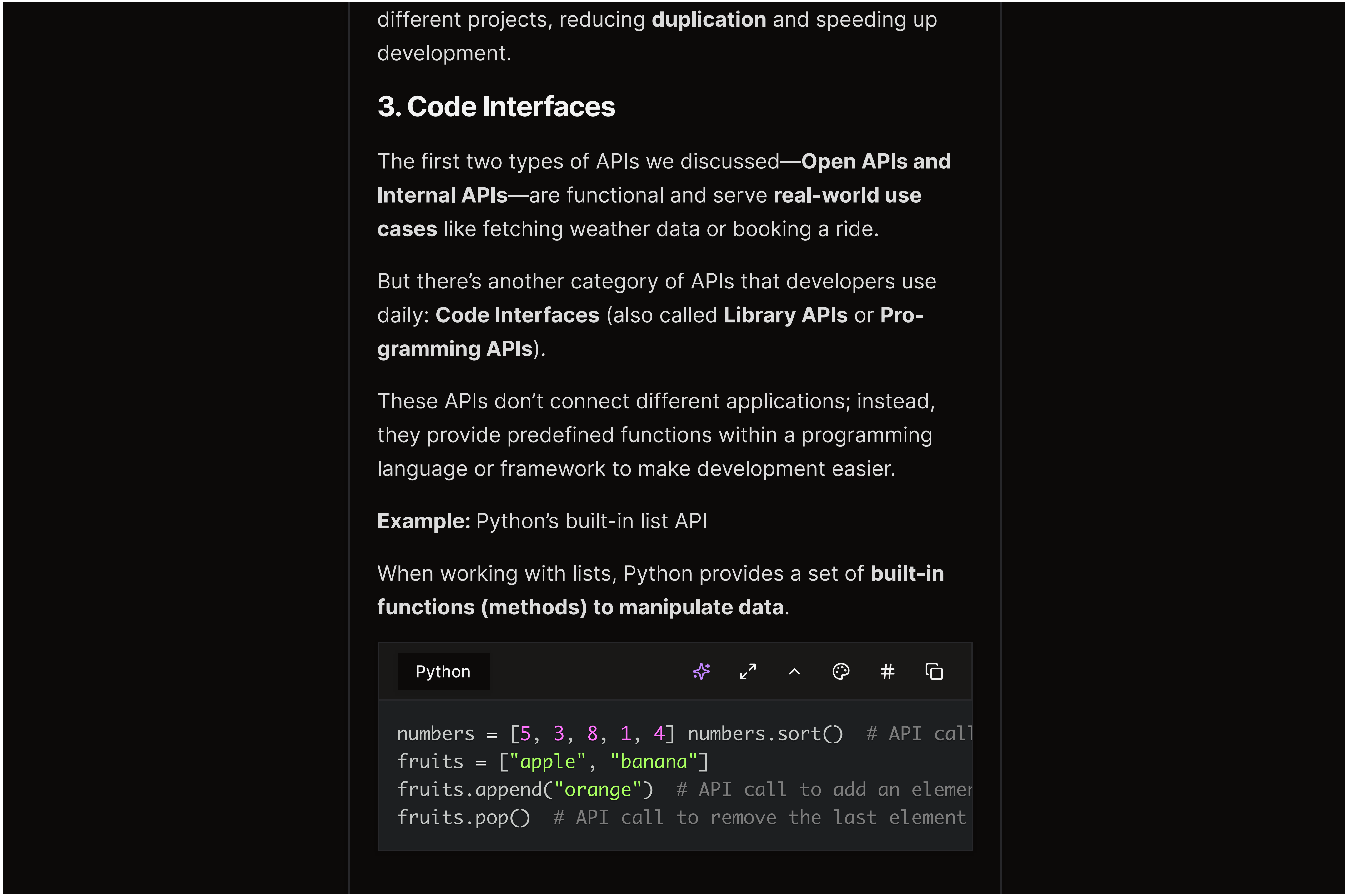
2. Internal APIs (Private APIs)

Internal APIs, also known as **Private APIs**, are designed **e**xclusively for internal use within an organization. Unlike Open APIs, these are not accessible to external developers and are used to facilitate seamless communication between different internal systems within a company.

Let’s take **Amazon** as an example. When you place an order, you might assume that a single system processes your request. In reality, **multiple internal APIs** (order processing, inventory, payment, logistics etc..) work together behind the scenes to fulfill your order efficiently.

Each of these APIs **operates independently**, but they communicate through well-defined protocols to ensure a smooth and efficient process.

Internal APIs allow companies to break down their applications into **smaller, manageable services**, making it easier to scale. Developers can **reuse internal APIs** across



different projects, reducing **duplication** and speeding up development.

3. Code Interfaces

The first two types of APIs we discussed—**Open APIs and Internal APIs**—are functional and serve **real-world use cases** like fetching weather data or booking a ride.

But there's another category of APIs that developers use daily: **Code Interfaces** (also called **Library APIs** or **Programming APIs**).

These APIs don't connect different applications; instead, they provide predefined functions within a programming language or framework to make development easier.

Example: Python's built-in list API

When working with lists, Python provides a set of **built-in functions (methods) to manipulate data**.

```python
numbers = [5, 3, 8, 1, 4] numbers.sort()  # API call
fruits = ["apple", "banana"]
fruits.append("orange")  # API call to add an elemen
fruits.pop()  # API call to remove the last element
```

Instead of writing sorting algorithms from scratch, developers can use `sort()` or `sorted()` in Python.

Code APIs are not just limited to built-in programming language functions. Take **TensorFlow**, an AI/ML library. It provides a **high-level API** for training machine learning models without needing to implement complex mathematical operations from scratch.

For example, creating a **neural network** using TensorFlow's API is as simple as:

Python

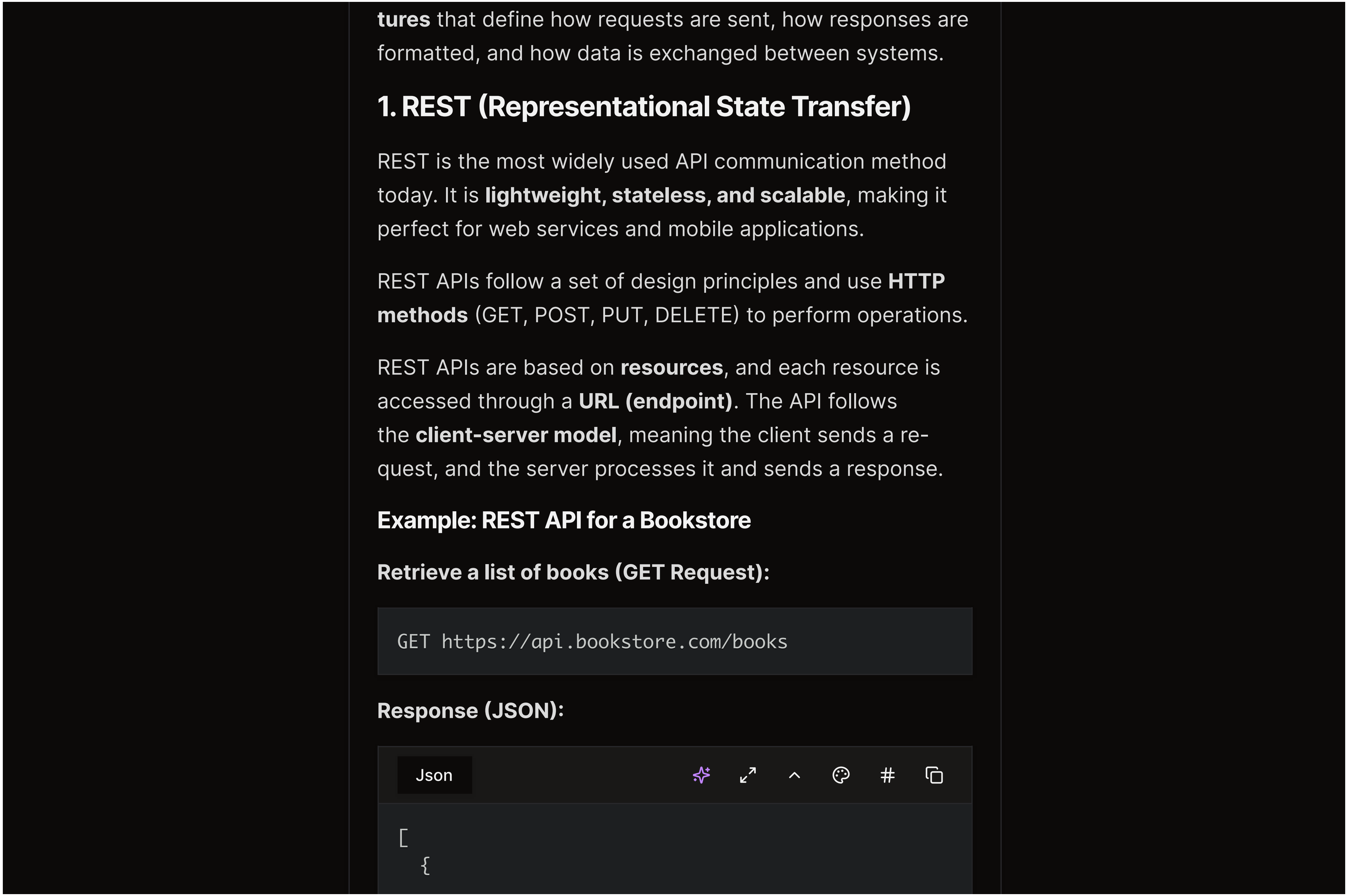
```python
import tensorflow as tf

model = tf.keras.Sequential([tf.keras.layers.Dense(
```

Programming APIs abstract away complexity so that developers can focus on building solutions rather than reinventing the wheel.

---

3. API Communication Methods

APIs communicate using different **protocols and architec-**



tures that define how requests are sent, how responses are formatted, and how data is exchanged between systems.

1. REST (Representational State Transfer)

REST is the most widely used API communication method today. It is **lightweight, stateless, and scalable**, making it perfect for web services and mobile applications.

REST APIs follow a set of design principles and use **HTTP methods** (GET, POST, PUT, DELETE) to perform operations.

REST APIs are based on **resources**, and each resource is accessed through a **URL (endpoint)**. The API follows the **client-server model**, meaning the client sends a request, and the server processes it and sends a response.

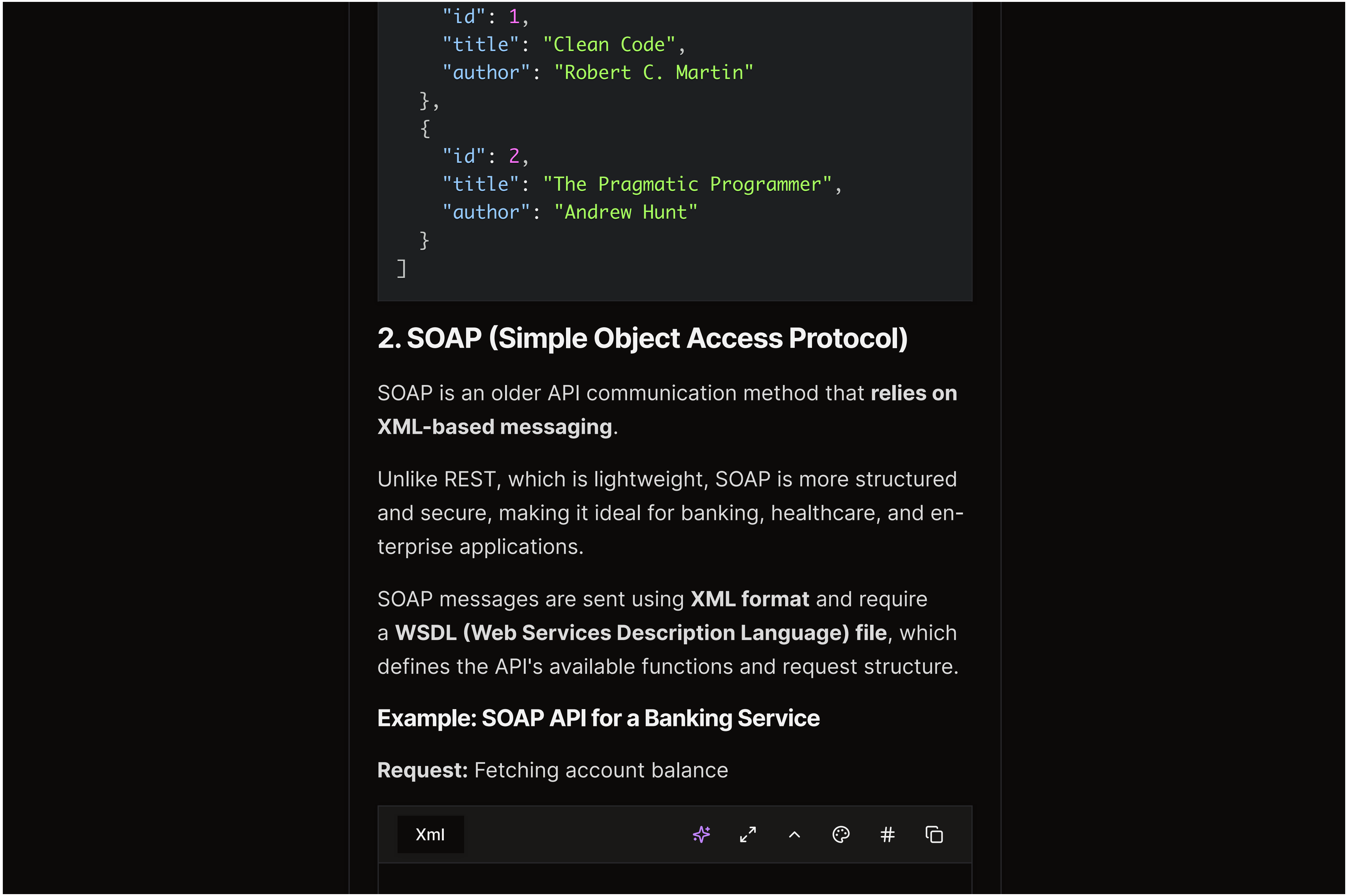
Example: REST API for a Bookstore

Retrieve a list of books (GET Request):

```
GET https://api.bookstore.com/books
```

Response (JSON):

```
[
  {
```



```
    "id": 1,
    "title": "Clean Code",
    "author": "Robert C. Martin"
  },
  {
    "id": 2,
    "title": "The Pragmatic Programmer",
    "author": "Andrew Hunt"
  }
]
```

2. SOAP (Simple Object Access Protocol)

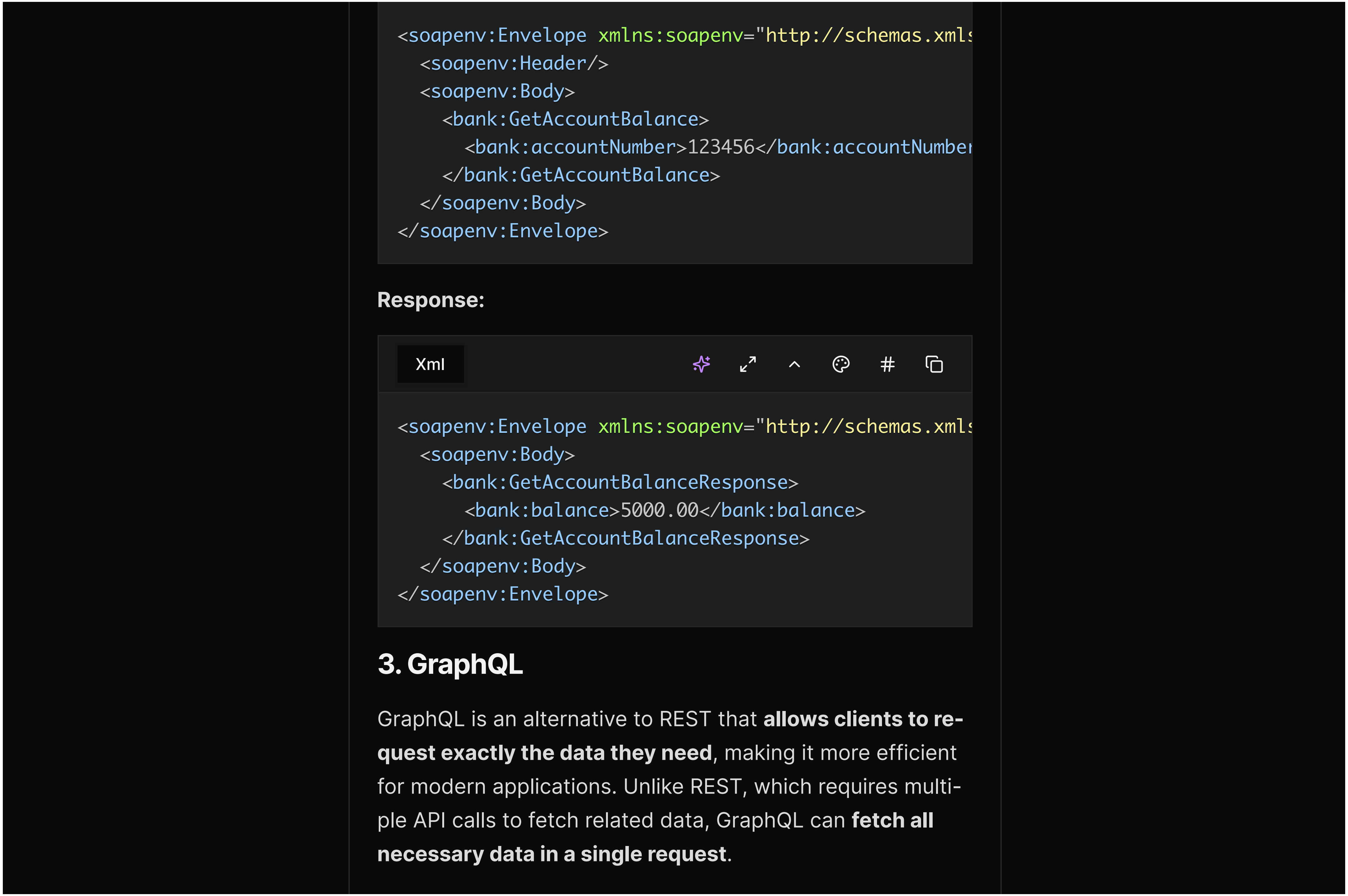
SOAP is an older API communication method that **relies on XML-based messaging**.

Unlike REST, which is lightweight, SOAP is more structured and secure, making it ideal for banking, healthcare, and enterprise applications.

SOAP messages are sent using **XML format** and require a **WSDL (Web Services Description Language) file**, which defines the API's available functions and request structure.

Example: SOAP API for a Banking Service

Request: Fetching account balance



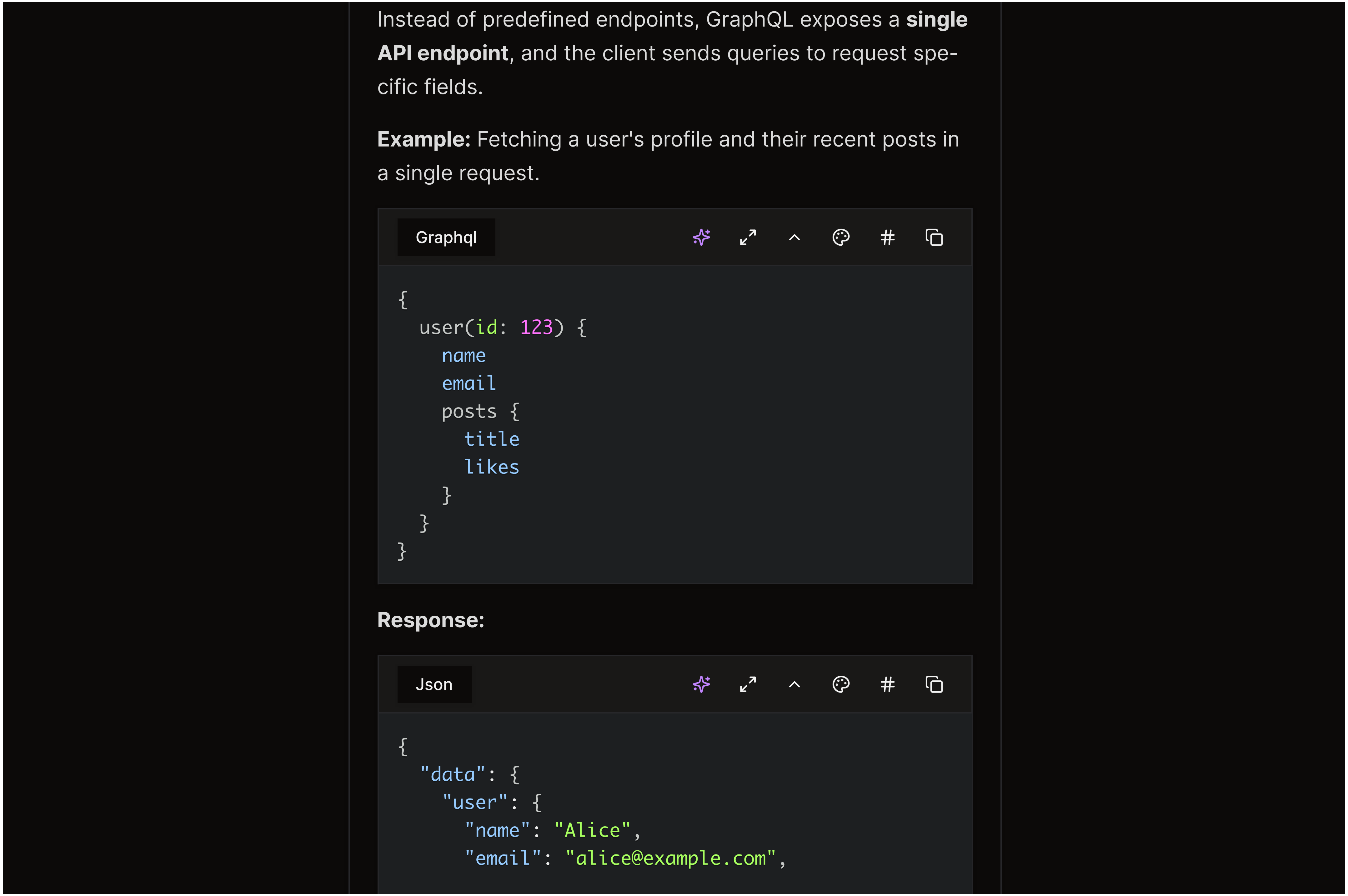
```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmls
  <soapenv:Header/>
  <soapenv:Body>
    <bank:GetAccountBalance>
      <bank:accountNumber>123456</bank:accountNumber
    </bank:GetAccountBalance>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmls
  <soapenv:Body>
    <bank:GetAccountBalanceResponse>
      <bank:balance>5000.00</bank:balance>
    </bank:GetAccountBalanceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

3. GraphQL

GraphQL is an alternative to REST that **allows clients to request exactly the data they need**, making it more efficient for modern applications. Unlike REST, which requires multiple API calls to fetch related data, GraphQL can **fetch all necessary data in a single request**.



Instead of predefined endpoints, GraphQL exposes a **single API endpoint**, and the client sends queries to request specific fields.

Example: Fetching a user's profile and their recent posts in a single request.

```graphql
{
  user(id: 123) {
    name
    email
    posts {
      title
      likes
    }
  }
}
```

Response:

```json
{
  "data": {
    "user": {
      "name": "Alice",
      "email": "alice@example.com",
```

```
      "posts": [
        {
          "title": "Hello World",
          "likes": 100
        },
        {
          "title": "GraphQL is Amazing!",
          "likes": 200
        }
      ]
    }
  }
}
```

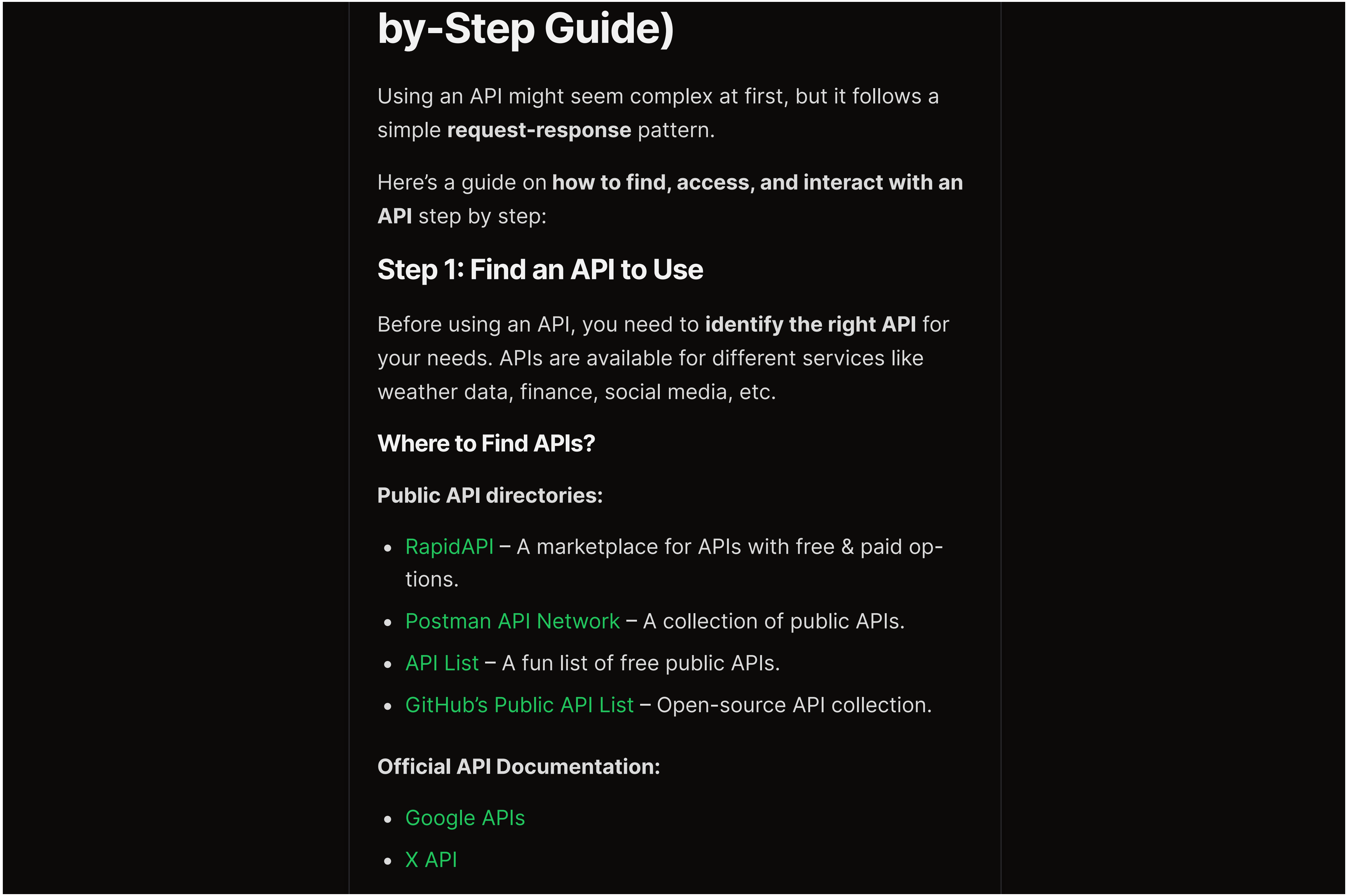
4. gRPC

gRPC (Google Remote Procedure Call) is a **high-performance API communication method** that uses **Protocol Buffers (Protobuf)** instead of JSON or XML, making it faster and more efficient.

gRPC uses **binary data format** instead of text-based formats, reducing payload size and it supports **bidirectional streaming**, meaning the client and server can send data at the same time.

---

4. How to Use an API (Step-



by-Step Guide)

Using an API might seem complex at first, but it follows a simple **request-response** pattern.

Here’s a guide on **how to find, access, and interact with an API** step by step:

Step 1: Find an API to Use

Before using an API, you need to **identify the right API** for your needs. APIs are available for different services like weather data, finance, social media, etc.

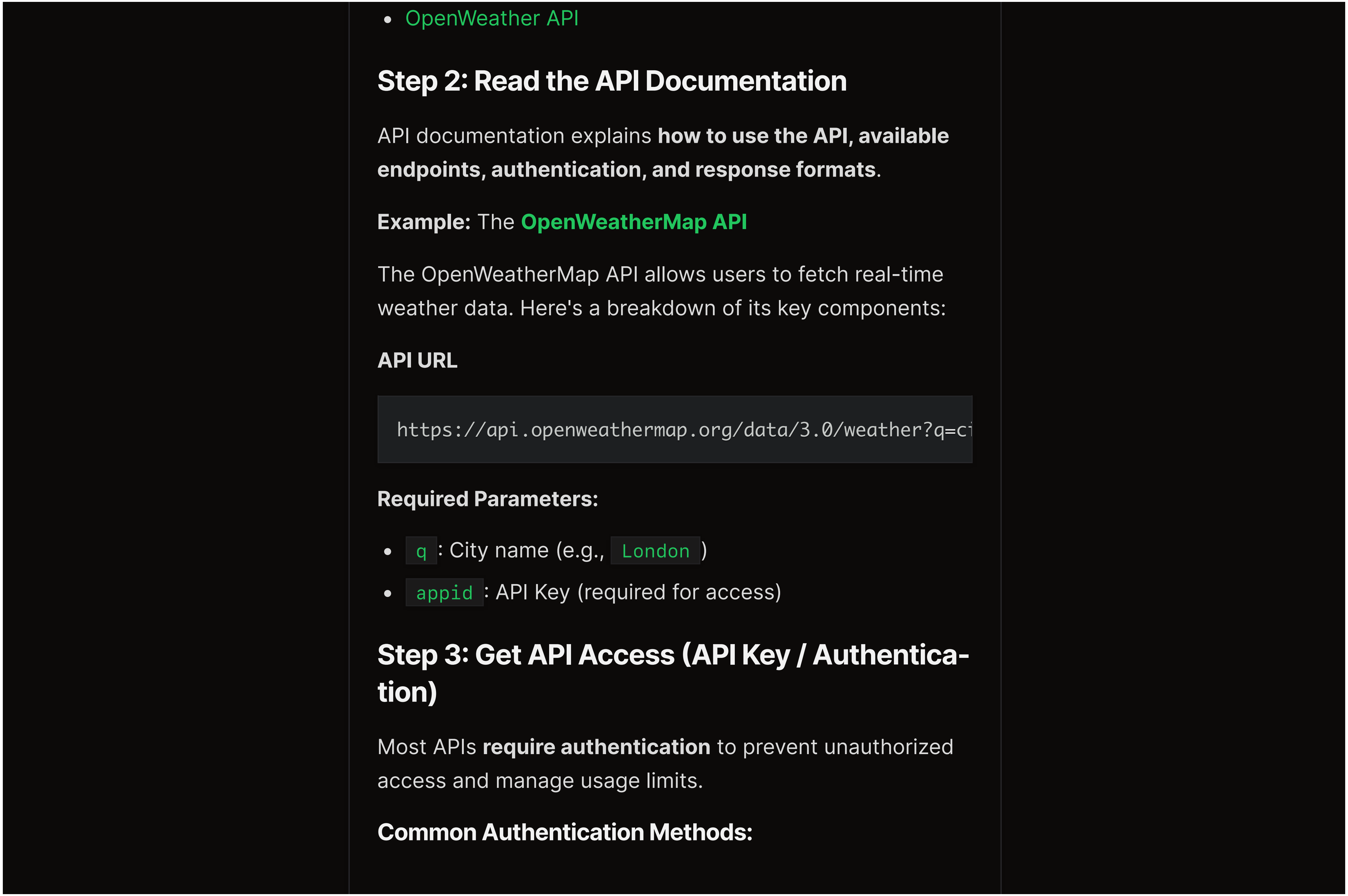
Where to Find APIs?

Public API directories:

- RapidAPI – A marketplace for APIs with free & paid options.
- Postman API Network – A collection of public APIs.
- API List – A fun list of free public APIs.
- GitHub’s Public API List – Open-source API collection.

Official API Documentation:

- Google APIs
- X API



- OpenWeather API

Step 2: Read the API Documentation

API documentation explains **how to use the API, available endpoints, authentication, and response formats**.

Example: The **OpenWeatherMap API**

The OpenWeatherMap API allows users to fetch real-time weather data. Here's a breakdown of its key components:

API URL

```
https://api.openweathermap.org/data/3.0/weather?q=ci
```

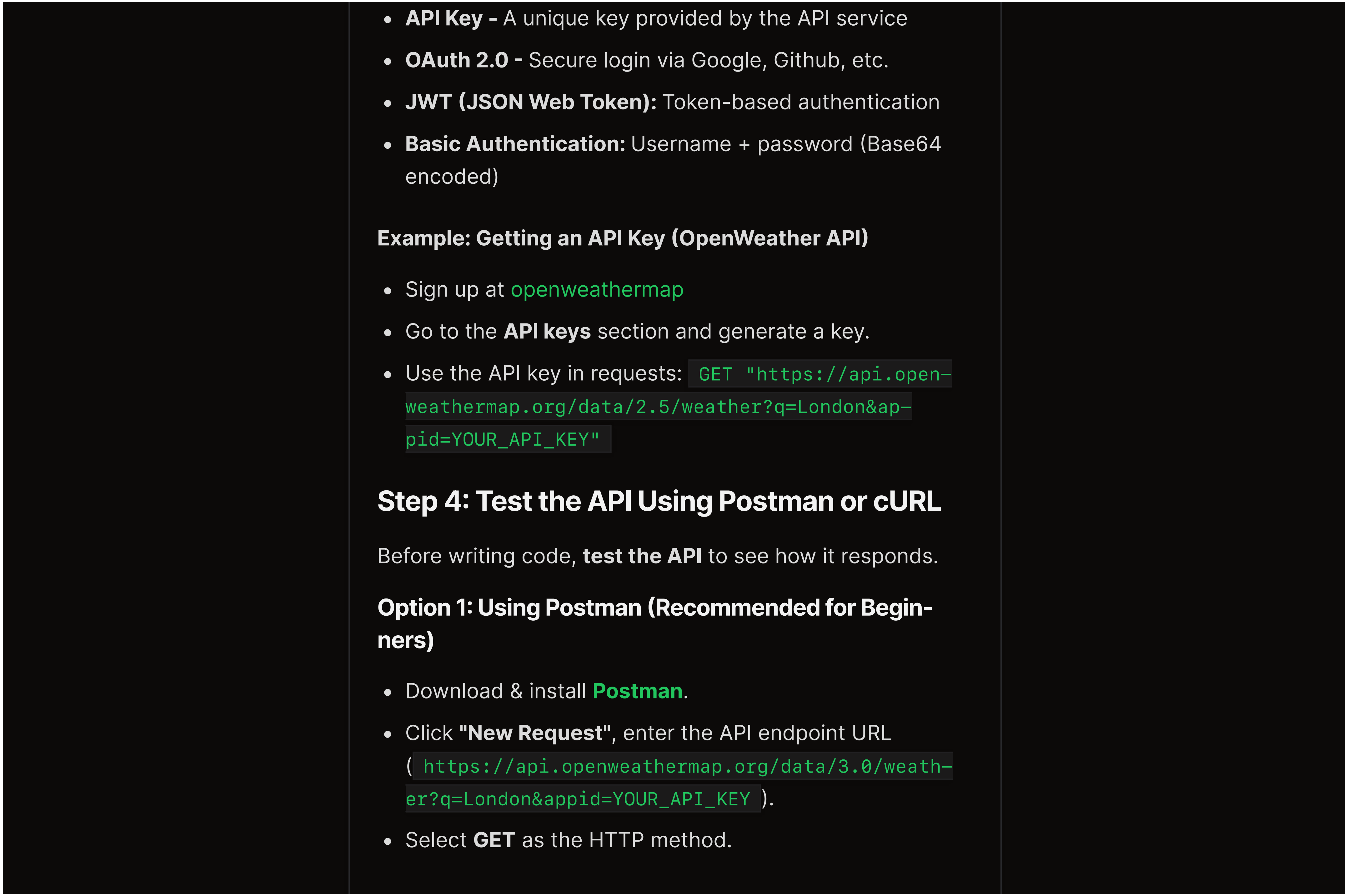
Required Parameters:

- `q` : City name (e.g., `London`)
- `appid` : API Key (required for access)

Step 3: Get API Access (API Key / Authentication)

Most APIs **require authentication** to prevent unauthorized access and manage usage limits.

Common Authentication Methods:



- **API Key -** A unique key provided by the API service
- **OAuth 2.0 -** Secure login via Google, Github, etc.
- **JWT (JSON Web Token):** Token-based authentication
- **Basic Authentication:** Username + password (Base64 encoded)

Example: Getting an API Key (OpenWeather API)

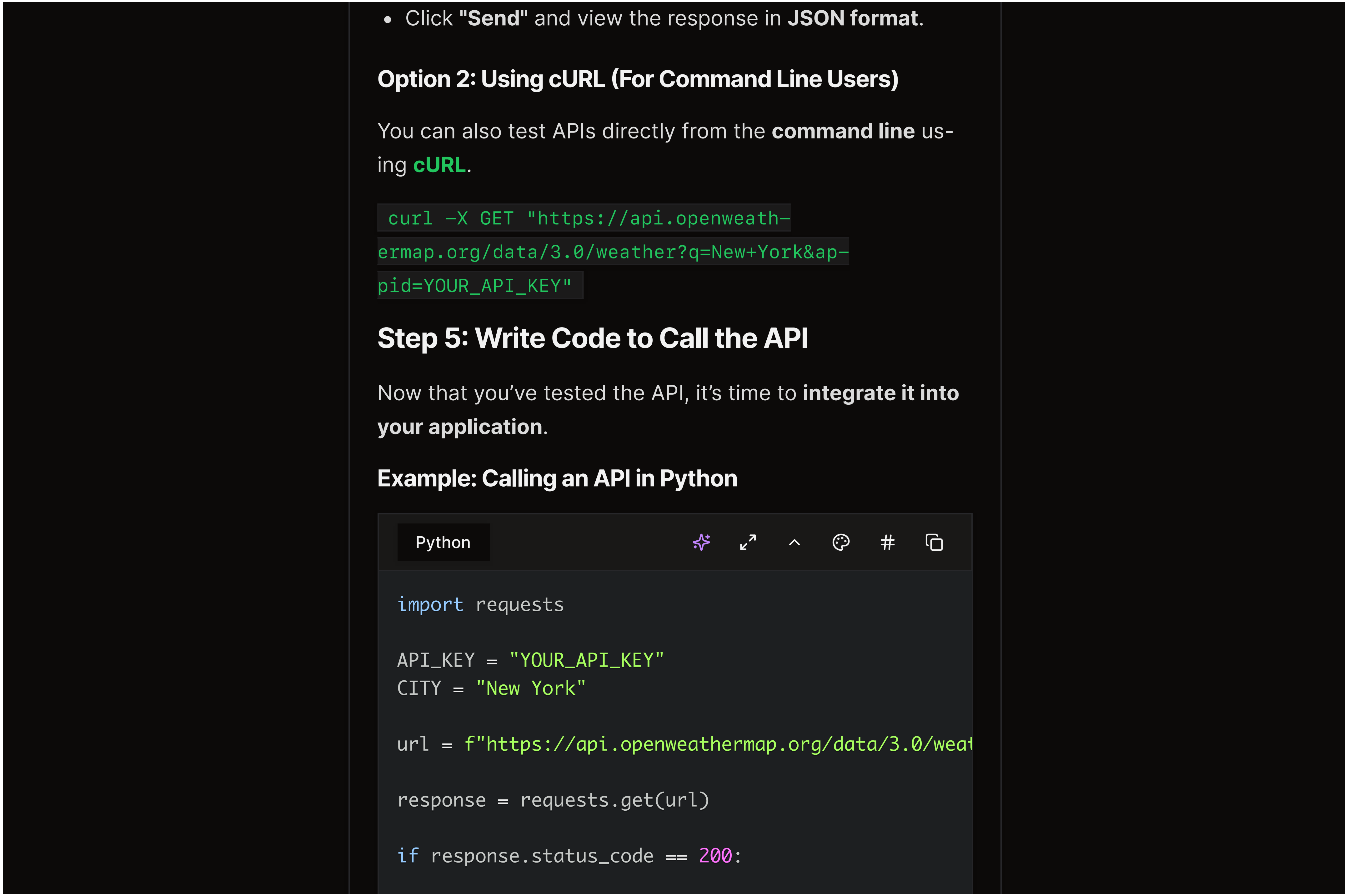
- Sign up at openweathermap
- Go to the **API keys** section and generate a key.
- Use the API key in requests: `GET "https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY"`

Step 4: Test the API Using Postman or cURL

Before writing code, **test the API** to see how it responds.

Option 1: Using Postman (Recommended for Beginners)

- Download & install **Postman**.
- Click **"New Request"**, enter the API endpoint URL (`https://api.openweathermap.org/data/3.0/weather?q=London&appid=YOUR_API_KEY`).
- Select **GET** as the HTTP method.



- Click **"Send"** and view the response in **JSON format**.

Option 2: Using cURL (For Command Line Users)

You can also test APIs directly from the **command line** using **cURL**.

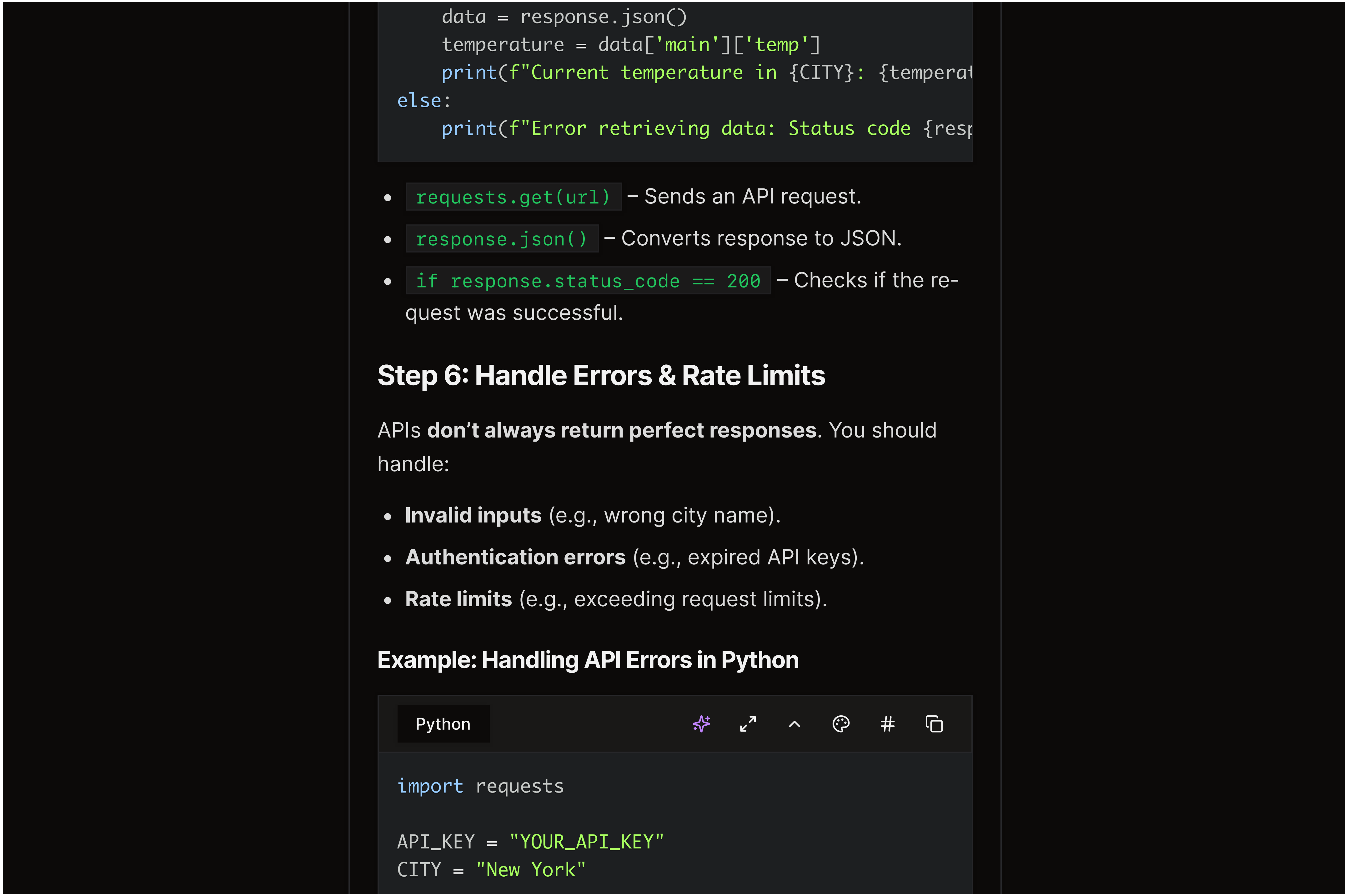
`curl -X GET "https://api.openweathermap.org/data/3.0/weather?q=New+York&appid=YOUR_API_KEY"`

Step 5: Write Code to Call the API

Now that you've tested the API, it's time to **integrate it into your application**.

Example: Calling an API in Python

```python
import requests

API_KEY = "YOUR_API_KEY"
CITY = "New York"

url = f"https://api.openweathermap.org/data/3.0/weat

response = requests.get(url)

if response.status_code == 200:
```



```
    data = response.json()
    temperature = data['main']['temp']
    print(f"Current temperature in {CITY}: {temperat
else:
    print(f"Error retrieving data: Status code {resp
```

- `requests.get(url)` – Sends an API request.
- `response.json()` – Converts response to JSON.
- `if response.status_code == 200` – Checks if the request was successful.

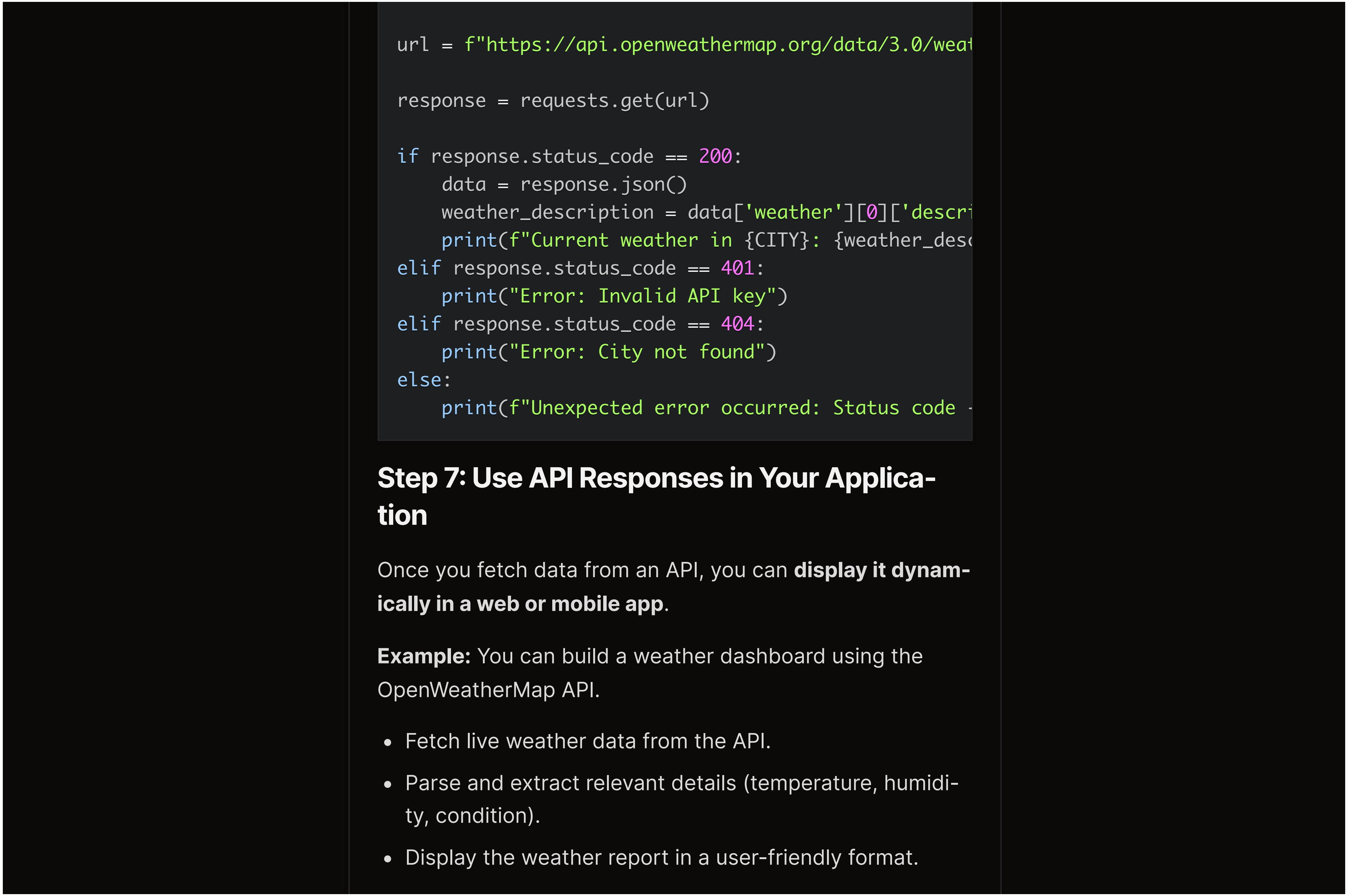
Step 6: Handle Errors & Rate Limits

APIs **don't always return perfect responses**. You should handle:

- **Invalid inputs** (e.g., wrong city name).
- **Authentication errors** (e.g., expired API keys).
- **Rate limits** (e.g., exceeding request limits).

Example: Handling API Errors in Python

```python
import requests

API_KEY = "YOUR_API_KEY"
CITY = "New York"
```



```
url = f"https://api.openweathermap.org/data/3.0/weat

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    weather_description = data['weather'][0]['descri
    print(f"Current weather in {CITY}: {weather_desc
elif response.status_code == 401:
    print("Error: Invalid API key")
elif response.status_code == 404:
    print("Error: City not found")
else:
    print(f"Unexpected error occurred: Status code
```

Step 7: Use API Responses in Your Application

Once you fetch data from an API, you can **display it dynamically in a web or mobile app**.

Example: You can build a weather dashboard using the OpenWeatherMap API.

- Fetch live weather data from the API.
- Parse and extract relevant details (temperature, humidity, condition).
- Display the weather report in a user-friendly format.