02 Exploring Payloads with Deck of Cards and OpenWeatherMap APIs

Enhanced REST API Lab: Exploring Payloads with Deck of Cards and OpenWeatherMap APIs

Objective

This lab introduces **REST API payloads** and demonstrates their use through practical examples with the **Deck of Cards API** (no authentication) and the **OpenWeatherMap API** (requires an API key, simple to configure). You'll learn how to handle JSON payloads, interact with APIs using `curl` and Python, and manage API keys for authentication. The lab includes hands-on exercises to reinforce these concepts.

Prerequisites

- **curl**: Install from https://curl.se/download.html for Mac, Linux, or Windows.
- **Python 3**: Ensure Python is installed with the `requests` library (`pip install requests`).
- **Terminal Access**: Use a terminal (e.g., Bash, PowerShell) to run commands.
- **OpenWeatherMap API Key**: Sign up at https://openweathermap.org to get a free API key (takes ~10 minutes to activate).

Section 1: Understanding REST API Payloads

The **payload** is the data sent in the body of an HTTP request or response. It carries the information needed to perform actions (e.g., create, update) or return results (e.g., retrieve data).

Payload Use Cases

- **POST, PUT, PATCH**: These verbs typically include a payload to send data for creating or updating resources.
- **GET**: Responses to GET requests usually include a payload with retrieved data.
- **Data Formats**:
  - **JSON (JavaScript Object Notation)**: Lightweight, human-readable, and widely used. Objects are defined with key-value pairs in `{}` braces, with strings in double quotes (`""`).
  - **XML (eXtensible Markup Language)**: Structured for data storage and transport, often used in enterprise systems.

Example: JSON Payload

A JSON payload for a book might look like:

```
{
  "title": "A Wrinkle in Time",
  "author": "Madeline L'Engle"
}
```

Example: XML Payload

An XML payload from a YANG model (used in network management) might look like:

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
        <name>GigabitEthernet1</name>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCsmacd</type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <address>
                <ip>198.18.133.212</ip>
                <netmask>255.255.192.0</netmask>
            </address>
        </ipv4>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
    </interface>
</interfaces>
```

Checking Payload Format

The `Content-Type` header indicates the payload format:

- `Content-Type: application/json` for JSON.
- `Content-Type: application/xml` for XML.

Use `curl --include` to inspect headers and confirm the payload format.

Section 2: Deck of Cards API (No Authentication)

The Deck of Cards API provides a simple, authentication-free interface to manage a virtual deck of playing cards. It supports actions like creating a deck, shuffling, and drawing cards.

Step 1: Create a New Deck

Run the following `curl` command to create a new deck:

```
curl https://deckofcardsapi.com/api/deck/new/ | python -m json.tool
```

Expected Response (your `deck_id` will differ):

```
{
    "success": true,
    "deck_id": "wz4csxs56693",
    "remaining": 52,
    "shuffled": false
}
```

This creates an unshuffled deck with 52 cards and a unique `deck_id`.

Step 2: Shuffle the Deck

Using the `deck_id` from the previous response, shuffle the deck:

```
curl https://deckofcardsapi.com/api/deck/<your_deck_id>/shuffle/ | python -m
json.tool
```

Replace `<your_deck_id>` with the actual `deck_id`. **Expected Response:**

```
{
    "success": true,
    "deck_id": "<your_deck_id>",
    "remaining": 52,
    "shuffled": true
}
```


Step 3: Draw Cards

Draw three cards from the shuffled deck using a query parameter:

```
curl https://deckofcardsapi.com/api/deck/<your_deck_id>/draw/?count=3 | python -m
json.tool
```

Expected Response (example):


```
{
    "success": true,
    "deck_id": "<your_deck_id>",
    "cards": [
        {
            "code": "6H",
            "image": "https://deckofcardsapi.com/static/img/6H.png",
            "value": "6",
            "suit": "HEARTS"
        },
        {
            "code": "KS",
            "image": "https://deckofcardsapi.com/static/img/KS.png",
            "value": "KING",
            "suit": "SPADES"
        },
        {
            "code": "AC",
            "image": "https://deckofcardsapi.com/static/img/AC.png",
            "value": "ACE",
            "suit": "CLUBS"
        }
    ],
    "remaining": 49
}
```


Python Script: Deck of Cards Interaction

Below is a Python script to automate deck creation, shuffling, and drawing cards:

```
import requests
import json
```

```
class DeckOfCards:
    def __init__(self):
        self.base_url = "https://deckofcardsapi.com/api/deck"
        self.deck_id = None


    def create_deck(self):
        """Create a new deck and store its deck_id."""
        try:
            response = requests.get(f"{self.base_url}/new/")
            response.raise_for_status()
            data = response.json()
            if data["success"]:
                self.deck_id = data["deck_id"]
                print(f"New deck created with ID: {self.deck_id}")
                print(f"Cards remaining: {data['remaining']}, Shuffled:
{data['shuffled']}")
            else:
                print("Failed to create deck")
        except requests.exceptions.RequestException as e:
            print(f"Error creating deck: {e}")

    def shuffle_deck(self):
        """Shuffle the current deck."""
        if not self.deck_id:
            print("No deck created. Create a deck first.")
            return
        try:
            response = requests.get(f"{self.base_url}/{self.deck_id}/shuffle/")
            response.raise_for_status()
            data = response.json()
            if data["success"]:
                print(f"Deck {self.deck_id} shuffled successfully")
                print(f"Cards remaining: {data['remaining']}, Shuffled:
{data['shuffled']}")
            else:
                print("Failed to shuffle deck")
        except requests.exceptions.RequestException as e:
            print(f"Error shuffling deck: {e}")

    def draw_cards(self, count=3):
        """Draw a specified number of cards from the deck."""
        if not self.deck_id:
            print("No deck created. Create a deck first.")
            return
        try:
            response = requests.get(f"{self.base_url}/{self.deck_id}/draw/?count=
{count}")
            response.raise_for_status()
            data = response.json()
            if data["success"]:
                print(f"Drew {count} cards from deck {self.deck_id}:")
```

```
                for card in data["cards"]:
                    print(f"- {card['value']} of {card['suit']} (Code:
{card['code']})")
                print(f"Cards remaining: {data['remaining']}")
            else:
                print("Failed to draw cards")
        except requests.exceptions.RequestException as e:
            print(f"Error drawing cards: {e}")

if __name__ == "__main__":
    deck = DeckOfCards()
    deck.create_deck()
    deck.shuffle_deck()
    deck.draw_cards(3)
```

Instructions:

1. Save the script as deck_of_cards.py.
2. Run it: python deck_of_cards.py.
3. Observe the output as it creates a deck, shuffles it, and draws three cards.

Section 3: OpenWeatherMap API (API Key Authentication)

The OpenWeatherMap API provides weather data for any location, requiring a simple API key for authentication. It's easier to configure than OAuth-based APIs and ideal for learning authenticated REST calls.

Step 1: Obtain an API Key

1. Sign up at https://openweathermap.org.
2. After activation (may take ~10 minutes), find your API key in the "API keys" section of your account dashboard.
3. Example API key (non-working): abcdef1234567890abcdef1234567890.

Security Note: Keep your API key private. Store it as an environment variable or in a secure configuration file.

Step 2: Set Up the API Key

In your terminal, export the API key:

```
export WEATHER_API_KEY="your_api_key_here"
```

Replace your_api_key_here with your actual API key.

Step 3: Fetch Weather Data

Use curl to fetch current weather for a city (e.g., London) using the q parameter and your API key:

```
curl "https://api.openweathermap.org/data/2.5/weather?
q=London&appid=$WEATHER_API_KEY&units=metric" | python -m json.tool
```

Expected Response (simplified example):


```
{
    "coord": {
        "lon": -0.1257,
        "lat": 51.5085
    },
    "weather": [
        {
            "main": "Clouds",
            "description": "overcast clouds"
        }
    ],
    "main": {
        "temp": 15.5,
        "feels_like": 14.8,
        "humidity": 82
    },
    "name": "London",
    "cod": 200
}
```


This response includes the city's coordinates, weather description, temperature (in Celsius due to units=metric), and more.

Python Script: Fetching Weather Data

Below is a Python script to fetch and display weather data for a specified city:

```
import requests
import json
import os

def fetch_weather(city, api_key, units="metric"):
    """
    Fetch current weather for a city using OpenWeatherMap API.

    Args:
        city (str): City name (e.g., "London")
        api_key (str): OpenWeatherMap API key
        units (str): Units for temperature ("metric" for Celsius, "imperial" for
Fahrenheit)
    """
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid=
{api_key}&units={units}"
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()

        print(f"Weather in {data['name']}:")
        print(f"Condition: {data['weather'][0]['main']} ({data['weather'][0]
```

```
['description']})")
        print(f"Temperature: {data['main']['temp']} °{'C' if units == 'metric'
else 'F'}")
        print(f"Feels Like: {data['main']['feels_like']} °{'C' if units ==
'metric' else 'F'}")
        print(f"Humidity: {data['main']['humidity']}%")
        print(f"Status Code: {data['cod']}")

    except requests.exceptions.RequestException as e:
        print(f"Error fetching weather: {e}")

if __name__ == "__main__":
    # Replace with your API key and city
    API_KEY = os.getenv("WEATHER_API_KEY") or "your_api_key_here"
    CITY = "London"

    fetch_weather(CITY, API_KEY, "metric")
```

Instructions:

1. Save the script as weather_api.py.
2. Set the WEATHER_API_KEY environment variable or replace "your_api_key_here" with your API key.
3. Update CITY to any city (e.g., "New York", "Tokyo").
4. Run the script: python weather_api.py.

Section 4: Comparing JSON and XML Payloads

To illustrate the difference between JSON and XML, let's create a Python script that fetches a JSON payload from the Deck of Cards API and converts it to a simplified XML format for comparison.

```
import requests
import xml.etree.ElementTree as ET
import xml.dom.minidom as minidom

def fetch_deck_and_convert_to_xml():
    """Fetch a new deck and convert the JSON response to XML."""
    try:
        response = requests.get("https://deckofcardsapi.com/api/deck/new/")
        response.raise_for_status()
        data = response.json()

        # Create XML structure
        root = ET.Element("Deck")
        ET.SubElement(root, "Success").text = str(data["success"])
        ET.SubElement(root, "DeckID").text = data["deck_id"]
        ET.SubElement(root, "Remaining").text = str(data["remaining"])
        ET.SubElement(root, "Shuffled").text = str(data["shuffled"])

        # Convert to pretty-printed XML
        rough_string = ET.tostring(root, "utf-8")
        reparsed = minidom.parseString(rough_string)
```

```
        pretty_xml = reparsed.toprettyxml(indent="  ")

        print("JSON Response:")
        print(json.dumps(data, indent=2))
        print("\nEquivalent XML:")
        print(pretty_xml)

    except requests.exceptions.RequestException as e:
        print(f"Error fetching deck: {e}")

if __name__ == "__main__":
    fetch_deck_and_convert_to_xml()
```

Instructions:

1. Save the script as json_to_xml.py.
2. Run it: python json_to_xml.py.
3. Compare the JSON and XML outputs to understand their structural differences.

Sample Output:

```
JSON Response:
{
  "success": true,
  "deck_id": "wz4csxs56693",
  "remaining": 52,
  "shuffled": false
}

Equivalent XML:
<?xml version="1.0" ?>
<Deck>
  <Success>true</Success>
  <DeckID>wz4csxs56693</DeckID>
  <Remaining>52</Remaining>
  <Shuffled>false</Shuffled>
</Deck>
```

Section 5: Hands-On Exercises

1. **Deck of Cards Exploration**:
   - Run the deck_of_cards.py script to create, shuffle, and draw cards.
   - Modify the script to draw 5 cards instead of 3.
   - Add a method to reshuffle the deck if remaining is less than 10 cards.
2. **Weather API Exploration**:
   - Run the weather_api.py script to fetch weather for a city.
   - Modify the script to fetch weather for a different city or in imperial units (units="imperial" for Fahrenheit).
   - Add a feature to display wind speed and direction from the response (data['wind']['speed'] and data['wind']['deg']).

3. **JSON vs. XML**:
   - Run the json_to_xml.py script to compare JSON and XML formats.
   - Extend the script to fetch drawn cards and convert their details (value, suit) to XML.
4. **Error Handling**:
   - Intentionally use an invalid deck_id in the Deck of Cards script and observe the error response.
   - Use an invalid city name in the weather_api.py script (e.g., q=InvalidCity) and check the status code (e.g., 404).
5. **Explore Headers**:
   - Use curl --include https://deckofcardsapi.com/api/deck/new/ to inspect response headers.
   - For OpenWeatherMap, use curl --include "https://api.openweathermap.org/data/2.5/weather?q=London&appid=$WEATHER_API_KEY" to check headers like Content-Type.

Section 6: Key Takeaways

- **Payloads** carry data in REST API requests and responses, typically in JSON or XML.
- **JSON** is lightweight and human-readable, ideal for most APIs like Deck of Cards and OpenWeatherMap.
- **XML** is structured and common in enterprise systems.
- **Authentication** varies: Deck of Cards API requires none, while OpenWeatherMap uses a simple API key.
- **Python** and curl are versatile for interacting with REST APIs.
- Always check **API documentation** for endpoints, parameters, and authentication details.

Section 7: Additional Resources

- **Deck of Cards API**: https://deckofcardsapi.com
- **OpenWeatherMap API**: https://openweathermap.org/api
- **Python Requests Library**: https://requests.readthedocs.io
- **HTTP Headers**: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- **JSON vs. XML**: https://www.w3schools.com/js/js_json_xml.asp