```
from flask import Flask, render_template, request, jsonify
import requests
from datetime import date, timedelta
from config import Config
#from flask_debugtoolbar import DebugToolbarExtension
app = Flask(__name__, static_url_path='/static')

#debug_toolbar = DebugToolbarExtension(app)
# Only enable DebugToolbar in development
#if app.debug:
#    toolbar = DebugToolbarExtension(app)




# NASA APOD API endpoint and API key
APOD_API_BASE_URL = "https://api.nasa.gov/planetary/apod"
APOD_API_KEY = Config.APOD_API_KEY

# Mars Rover API endpoint and API key
ROVER_API_BASE_URL = Config.ROVER_API_ENDPOINT
ROVER_API_KEY = Config.ROVER_API_KEY

# NEO API endpoint and API key
NEO_API_BASE_URL = "https://api.nasa.gov/neo/rest/v1"  # Updated the
NEO_API_BASE_URL
NEO_API_KEY = Config.NEO_API_KEY

# EPIC API endpoint and API key
EPIC_API_BASE_URL = "https://epic.gsfc.nasa.gov/api/natural"
EPIC_API_KEY = Config.EPIC_API_KEY

# ... (other configurations)

# Route for the homepage (space.html)
@app.route("/")
def space():
    # Add any logic you need for the homepage here
    # For example, you can render a template or return a simple message
    return render_template("space.html")

@app.route('/iss_location')
def iss_location():
    # API endpoint for ISS Location Now
```

```
    api_url = 'http://api.open-notify.org/iss-now.json'

    # Make a request to the API
    response = requests.get(api_url)

    # Check if the request was successful (status code 200)
    if response.status_code == 200:
        # Parse the JSON response
        iss_data = response.json()

        # Extract relevant information from the response
        latitude = iss_data['iss_position']['latitude']
        longitude = iss_data['iss_position']['longitude']
        timestamp = iss_data['timestamp']

        # Render the SpaceX template with the ISS location information
        return render_template('iss_location.html', latitude=latitude,
longitude=longitude, timestamp=timestamp)
    else:
        # If the request was not successful, return an error message
        return f"Error: Unable to fetch ISS location. Status code:
{response.status_code}"

# Route to display the APOD information
@app.route("/apod")
def index():
    # Make API request to NASA APOD
    params = {"api_key": APOD_API_KEY}
    response = requests.get(APOD_API_BASE_URL, params=params)
    data = response.json()

    # Extract relevant information
    title = data.get("title", "NASA Astronomy Picture of the Day")
    url = data.get("url", "")
    explanation = data.get("explanation", "")

    # Render the template with the APOD information
    return render_template("apod.html", title=title, url=url, explanation=explanation)

# Route to display Mars Rover photos

@app.route("/neo")
def neo():
    # Calculate the current date in the format yyyy-mm-dd
```

```
    current_date = date.today().strftime("%Y-%m-%d")

    # Set an end date in the future (e.g., one week from the current date)
    end_date = (date.today() + timedelta(days=7)).strftime("%Y-%m-%d")

    # Use the correct API endpoint for fetching asteroids based on their closest
approach date
    neo_feed_endpoint = f"{NEO_API_BASE_URL}/feed"
    params = {"start_date": current_date, "end_date": end_date, "api_key":
NEO_API_KEY}

    # Make API request to NASA NEO API with API key in parameters
    response = requests.get(neo_feed_endpoint, params=params)

    # Check if the response is successful (status code 200)
    if response.status_code == 200:
        data = response.json()

        # Extract relevant information for NEO
        neo_objects = data.get("near_earth_objects", {}).get(current_date, [])

        # Check if neo_objects is a non-empty list
        if neo_objects:
            # Assuming each NEO object has "name" and "close_approach_date" keys
            neo_info = [
                {"name": neo.get("name", "Unknown"), "approach_date":
neo.get("close_approach_date", "Unknown")}
                for neo in neo_objects
            ]

            # Render the template with NEO information
            return render_template("neo.html", neo_info=neo_info)

    # Handle the error case for unsuccessful API response or no NEO objects
    print("Error response:", response.content)
    return render_template("neo.html", neo_info=[])

# SpaceX Telemetry API endpoint
# ...

# SpaceX Telemetry API endpoint
SPACEX_API_BASE_URL = "https://api.spacexdata.com/v4/launches/latest"

# SpaceX Telemetry API endpoint
```

```
# Route handling SpaceX Telemetry data
# SpaceX Telemetry API endpoint
# Route handling SpaceX Telemetry data
# Update the SpaceX API endpoint
SPACEX_API_BASE_URL = "https://api.spacexdata.com/v4/launches"

# ...

# Route handling SpaceX Telemetry data
@app.route("/spacex")
def spacex_telemetry():
    try:
        # Make API request to SpaceX Telemetry API
        response = requests.get(SPACEX_API_BASE_URL)

        # Check if the response is successful (status code 200)
        if response.status_code == 200:
            spacex_data = response.json()
            return render_template("spacex.html", spacex_launches=spacex_data)  #
Pass spacex_data as spacex_launches
        else:
            # Handle the error case for unsuccessful API response
            return jsonify({"error": "Failed to fetch SpaceX Telemetry data"}), 500
    except requests.exceptions.RequestException as e:
        # Handle the error case for any exceptions during the request
        return jsonify({"error": f"Failed to fetch SpaceX Telemetry data: {e}"}), 500
# ...

# JWST API endpoint
JWST_API_BASE_URL = "https://api.jwstapi.com"

# ...


# Route to get JWST data
@app.route("/jwst_data", methods=["GET"])
def get_jwst_data():
    try:
        # Make API request to JWST API
        response = requests.get(JWST_API_BASE_URL)

        # Check if the response is successful (status code 200)
        if response.status_code == 200:
            jwst_data = response.json()
```

```
            return jsonify(jwst_data)
        else:
            # Handle the error case for unsuccessful API response
            return jsonify({"error": "Failed to fetch JWST data"}), 500
    except requests.exceptions.RequestException as e:
        # Handle the error case for any exceptions during the request
        return jsonify({"error": f"Failed to fetch JWST data: {e}"}), 500




# ... (other imports)

# InSight: Mars Weather Service API endpoint and API key
INSIGHT_API_BASE_URL = "https://api.nasa.gov/insight_weather"
INSIGHT_API_KEY = Config.INSIGHT_API_KEY

# ... (other configurations)

# Route to display InSight Mars weather data
@app.route("/insight_weather")
def insight_weather():
    # Define API endpoint and parameters
    insight_api_endpoint = f"{INSIGHT_API_BASE_URL}/"
    params = {"api_key": INSIGHT_API_KEY, "feedtype": "json", "ver": 1.0}

    # Make API request to InSight Mars Weather Service API
    response = requests.get(insight_api_endpoint, params=params)

    # Check if the response is successful (status code 200)
    if response.status_code == 200:
        data = response.json()

        # Extract relevant information for InSight weather
        # Modify this part based on the actual structure of the API response
        # For example, you might want to extract temperature, wind, pressure, etc.

        # Render the template with InSight weather information
        return render_template("insight_weather.html", insight_info=data)

    # Handle the error case for unsuccessful API response
    print("Error response:", response.content)
    return render_template("error.html", message="Failed to fetch InSight weather
```

```
information.")

# ... (other routes)

if __name__ == '__main__':
    app.run(debug=True)


@app.route("/solar_system")
def solar_system():
    # Make API request to le-systeme-solaire API to get information for all bodies
    response = requests.get(SOLAR_SYSTEM_API_BASE_URL)

    try:
        response.raise_for_status()  # Raise HTTPError for bad responses (4xx and
5xx)
        response_data = response.json()

        # Extract relevant information for all bodies
        bodies_info = [{'id': body['id'], 'name': body['englishName']} for body in
response_data]

        return render_template('solar_system_all.html', bodies_info=bodies_info)
    except requests.exceptions.RequestException as e:
        # Handle the error case for unsuccessful API response
        print(f"Error: {e}")
        return render_template("error.html", message="Failed to fetch celestial
bodies information.")

# ...

if __name__ == '__main__':
    app.run(debug=True)
```