Distributed Systems Exercise 8 (Web service)

Riccardo Benedetti
Leon Harz

Overview of weather client

```python
class SimpleClient:
    lharz
    def __init__(self, server_address, port):
        self.server_address = server_address
        self.port = port

    1 usage  lharz *
    def get_weather_info(self, city, time_frame, hourly):
        with open("skeleton.json") as file:
            weather_request = json.load(file)
        weather_request["type"] = "weather_request"
        weather_request["location"] = city
        weather_request["time_frame"] = time_frame
        weather_request["day_format"] = hourly
        weather_request = json.dumps(weather_request)
        response = requests.post(
            url: f"http://{self.server_address}:{self.port}/forecast",
            json=weather_request
        )
        if response.status_code == 200:
            weather_info = json.loads(response.text)[0]["weather"]
            print_weather_info(weather_info)
        else:
            print(f"Server problems :S")
```

- Simple client for communication with REST API server
- get_weather_info builds weather request using a predefined json
- transmits the location, time frame and if a daily or hourly report is needed
- processes server responds, either failure or prints the weather information in a readable way

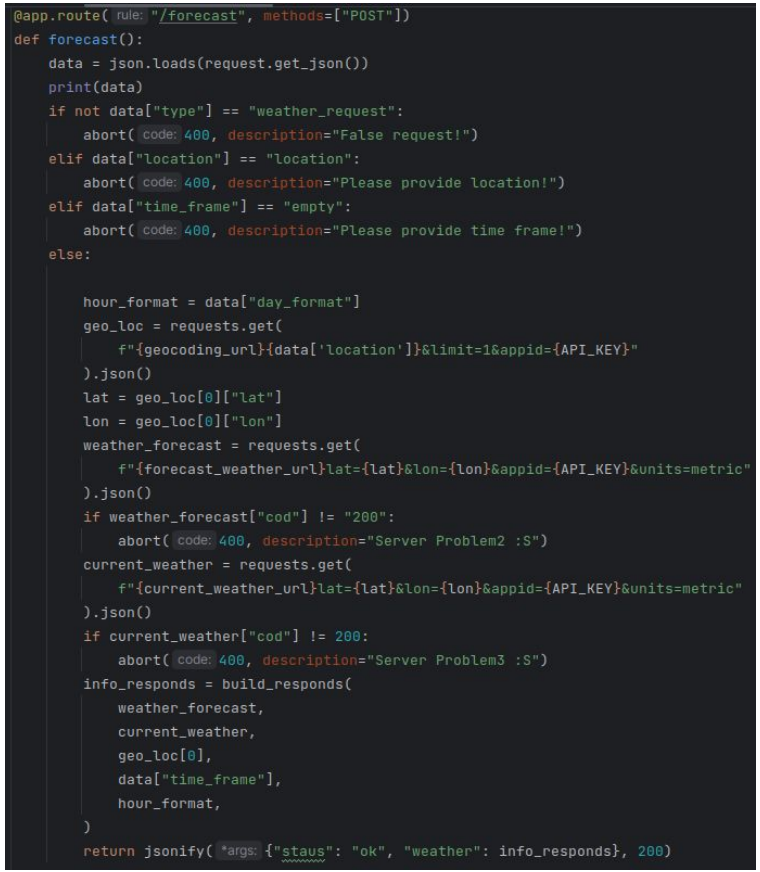
```
#########CURRENT WEATHER#########
Bielefeld, North Rhine-Westphalia, DE, 19-11-2024
currently: overcast clouds temperature: 0.97°C
feels_like: -3.14 max_temp: 1.38°C
humidity: 99  min_temp: 0.97°C

Bielefeld, North Rhine-Westphalia, DE, 19-11-2024 19:46:58
Temperature: overcast clouds temperature: 1.23°C
feels_like: -2.79 max_temp: 1.76°C
humidity: 98  min_temp: 1.23°C
```

How it works and Execution Flow

1. User runs the script with arguments (python client.py <city> <time_frame> <daily||hourly>)
2. Script initializes the SimpleClient
3. Read the skeleton.json and fill in request parameters
4. Send as POST to request to the server
5. If the server responds successfully (status_code = 200), parses and prints weather data
6. Display error message if server communication fails

Overview of weather server



```python
@app.route( rule: "/forecast", methods=["POST"])
def forecast():
    data = json.loads(request.get_json())
    print(data)
    if not data["type"] == "weather_request":
        abort( code: 400, description="False request!")
    elif data["location"] == "location":
        abort( code: 400, description="Please provide location!")
    elif data["time_frame"] == "empty":
        abort( code: 400, description="Please provide time frame!")
    else:

        hour_format = data["day_format"]
        geo_loc = requests.get(
            f"{geocoding_url}{data['location']}&limit=1&appid={API_KEY}"
        ).json()
        lat = geo_loc[0]["lat"]
        lon = geo_loc[0]["lon"]
        weather_forecast = requests.get(
            f"{forecast_weather_url}lat={lat}&lon={lon}&appid={API_KEY}&units=metric"
        ).json()
        if weather_forecast["cod"] != "200":
            abort( code: 400, description="Server Problem2 :S")
        current_weather = requests.get(
            f"{current_weather_url}lat={lat}&lon={lon}&appid={API_KEY}&units=metric"
        ).json()
        if current_weather["cod"] != 200:
            abort( code: 400, description="Server Problem3 :S")
        info_responds = build_responds(
            weather_forecast,
            current_weather,
            geo_loc[0],
            data["time_frame"],
            hour_format,
        )
        return jsonify( *args: {"staus": "ok", "weather": info_responds}, 200)
```

- routes and processes incoming HTTP POST requests for weather data
- Loads an API key from an .env file to access OpenWeatherMap
- /forecast-Endpoint handles weather requests sent by the client
- /forecast-Endpoint validates input JSON and aborts with errors for invalid requests
- Geocoding API converts location names into latitude and longitude
- Weather API fetches current and forecasted weather data based on coordinates

How it works and Execution Flow

1. Receiving Request:
    a. Parse incoming JSON request to extract data
    b. Validates required field and handles missing/incorrect data
2. Data Fetching:
    a. Retrieves geographic coordinates using the Geocoding API
    b. Queries OpenWeatherMap’s current weather and forecast APIs using coordinates
3. Response Construction:
    a. Builds detailed weather report
    b. Current conditions: weather temperature, humidity,...
    c. forecasts either in a daily (aggregated data, averages and min/max) or hourly
4. Returns the weather report as JSON object to the client