Task for Python (API Development) – Sports Data Integration and Display

Objective: Assess the ability to develop APIs in Python that interact with **sports applications (like Garmin, Coros, or any other fitness platform of your choice)**, pull relevant data (such as activity metrics, health statistics, etc.), and display this data in a simple dummy app. This task evaluates skills in API integration, data manipulation, and backend development.

Task Overview

Deliverables: Develop a **Python-based API** that pulls data from a sports application (Garmin, Coros, or any other fitness platform of your choice) and displays the data in a **dummy app** (which can be a simple console-based app or a basic web app).

Requirements & Details:

1. **API Development**

- **Choose a Sports Platform API**:

  Select one of the following platforms (Garmin, Coros, Strava, or any other fitness tracking app that provides public or accessible API endpoints).

  If the platform requires authentication (e.g., OAuth), implement the necessary authentication flow.

- **Data Retrieval**:

  The API should be able to pull the following types of data:

  - Activity data (e.g., distance, time, pace, steps, calories burned)
  - Health metrics (e.g., heart rate, sleep data, recovery metrics)
  - Recent workouts or challenges completed by the user

- **Data Handling**:

  - Ensure that the data pulled is clean and structured (using JSON or another format).
  - Filter out unnecessary information, leaving only relevant activity or health data.

2. Dummy App Development

- **Display Data**:
  Build a basic front-end (can be a console-based or web app) to display the data fetched from the API. For example:
  - A user dashboard showing their most recent activity (steps, distance, calories, etc.).
  - A simple graph or chart representing metrics like heart rate over time (using libraries like **Matplotlib** or **Plotly** for Python).
- **User Interface**:
  - Keep the UI simple but clear. If it's a web app, you can use **Flask** or **Django** to render data in a user-friendly format.
  - Ensure responsiveness (especially if web-based), though a basic layout is sufficient for this task.

3. Authentication and Error Handling

- Implement **OAuth 2.0** or **API key-based authentication** as required by the platform.
- Handle **error cases** (e.g., when the API request fails, invalid user input, or missing data) gracefully and display appropriate error messages.

4. Documentation

- Provide clear instructions on how to run the project, set up API keys, and authenticate the app to pull data.
- Include comments in the code to explain the key steps of data fetching and display.

Submission Guidelines

Deadline: Submit the task within **7 days** of receiving it.

Submission Format:

- Python code files, including scripts for the API and the app.
- Instructions for setting up the environment (e.g., dependencies, API keys, etc.).
- If a web app is used, provide a link to the GitHub repository or share the app with a basic walkthrough.

Evaluation Criteria

API Integration:

- Ability to successfully integrate with external sports APIs and fetch relevant data.
- Correct implementation of authentication and error handling.

Data Handling:

- Effective cleaning and filtering of fetched data.
- Proper formatting and presentation of the data.

App Development:

- A clear and simple UI for displaying the fetched data.
- For web-based apps, use of Python frameworks like Flask or Django is appreciated, but a console-based app is acceptable.

Documentation:

- Clarity in the setup instructions, ensuring that others can easily run the app and view the data.

Code Quality:

- Well-structured and clean code, with appropriate comments and error handling.