Documentation for Performance Check Application

This documentation provides an overview of the backend FastAPI application and the Flutter frontend that works together to collect race results, view performance stats, and provide improvement tips.

Backend (FastAPI) - main.py

1. API Endpoints:

1.1 Add Race Result:

- **Endpoint:** /add_race_result
- **Method:** POST
- **Description:** Adds a race result to the database.
- **Request Body:**

```
{
  "name": "John Doe",
  "timing": 18.5,
  "date": "2024-12-17",
  "status": "Below Threshold",
  "level": "Intermediate",
  "tips": [
    "Great performance! Focus on speed drills and endurance.",
    "Incorporate interval training into your routine to enhance both speed and stamina
    "Refine your running technique to minimize energy wastage during sprints."
  ]
}
```

1.2 View Performance:

- **Endpoint:** /view_performance/{name}
- **Method:** GET
- **Description:** Retrieves performance metrics and details for a specific user.
- **Path Parameters:**
    - name: The name of the person whose performance is to be viewed.


```
  "name": "John Doe",
  "timing": 18.5,
  "date": "2024-12-17",
  "status": "Below Threshold",
  "level": "Intermediate",
  "tips": [
    "Great performance! Focus on speed drills and endurance.",
    "Incorporate interval training into your routine to enhance both speed and sta
    "Refine your running technique to minimize energy wastage during sprints."
  ]
},
{
  "name": "John Doe",
  "timing": 20.1,
  "date": "2024-12-16",
  "status": "Above Threshold",
  "level": "Novice",
  "tips": [
    "Good effort! Work on strength training and agility.",
    "Add hill sprints to your workouts to build power and explosiveness.",
    "Improve your diet by including more protein-rich foods to support muscle reco
  ]
}
```


    - Returns a summary of the user's race data, including:
        - Total races
        - Average timing
        - Number of races below and above the threshold
        - Detailed race results.

1.3 Get Improvement Tips by Timing:

- **Endpoint:** /get_improvement_tips/{timing}
- **Method:** GET
- **Description:** Retrieves improvement tips based on a specific race timing.
- **Path Parameters:**
    - timing: The time of the race (in seconds).
- **Response:**
    - Returns the improvement level (e.g., Elite, Novice) and associated tips.
    - Example


```
{
  "level": "Intermediate",
  "tips": [
    "Great performance! Focus on speed drills and endurance.",
    "Incorporate interval training into your routine to enhance both speed and stamina
    "Refine your running technique to minimize energy wastage during sprints."
  ]
}
```


---

2. Helper Functions:

- **load_data()**: Loads race data from a JSON file (race_data.json) when the API starts.
- **save_data()**: Saves the race data back to the race_data.json file.
- **get_improvement_tip(timing)**: Determines the improvement tips based on the provided race timing.

Frontend (Flutter) - main.dart

1. User Interface:

- The application is a simple form where users can:
    - Input their name and race timing.
    - View improvement tips based on their timing.
    - Submit their race results to the backend.
    - View their historical race performance, including race times, improvement level, and tips.

2. Features:

- **Add Race Result**: Users can submit their race timing and name to the backend, and the app will display any improvement tips and a message based on the submitted timing.
- **View Performance**: Users can enter their name to retrieve detailed statistics about their past race performances, including:
    - Total races.
    - Average timing.
    - Number of races below and above the threshold.
    - List of past race results with timing, status, and improvement level.

3. Networking:

- **API Requests**:
    - The app uses the http package to interact with the FastAPI backend.
    - POST /add_race_result to submit a race result.
    - GET /view_performance/{name} to retrieve user performance data.
    - GET /get_improvement_tips/{timing} to get tips based on a race timing.

4. State Management:

- The app maintains state using setState () to update UI elements dynamically based on API responses. This ensures the user receives immediate feedback after submitting race data or viewing performance.

Running the Application

1. Backend Setup (FastAPI):

- Install dependencies:

```
pip install fastapi pydantic
```

2. Frontend Setup (Flutter):

- Ensure you have Flutter and Dart installed.
- Run the app using:

```
flutter run
```

- The app will run on your connected device/emulator.

3. Backend & Frontend Integration:

- Make sure the apiBaseUrl in the Flutter app points to the correct backend URL, especially if running on a device or emulator.
    - For local development, you can keep it as http://127.0.0.1:8000.

Conclusion

This system allows users to input their race results, view their performance history, and receive tailored improvement tips based on their timing. The combination of Fast API backend and Flutter frontend ensures a responsive and interactive experience for tracking and improving race performance.