ZEROBYTE

Internship Assignment

Problem Statement: Cheapest Transfer Route

A logistics company is optimizing its package delivery network. Packages need to be transferred between multiple cities, and each transfer has an associated cost. Your task is to help the company find the **best combination of transfers** while ensuring that the total package weight stays within a given limit.

Requirements

Inputs

- A list of available transfers, where each transfer has:
  - **weight** (in kg): The weight the transfer can handle.
  - **cost** (in currency): The fee for that transfer.
- A maximum weight limit (**maxWeight**): The total weight that can be transferred in one route.

Objective

- Find the combination of transfers that maximizes the total cost while ensuring the total weight of the transfers is less than or equal to **maxWeight**.

Output

- A list of transfers (weights and costs) selected for the route.
- The total cost of the selected transfers.

Example Input

```
{
  "maxWeight": 15,
  "availableTransfers": [
    {
      "weight": 5,
      "cost": 10
    },
    {
      "weight": 10,
      "cost": 20
    },
    {
      "weight": 3,
      "cost": 5
    },
    {
      "weight": 8,
      "cost": 15
    }
  ]
}
```

Example Output

```
{
  "selectedTransfers": [
    {
      "weight": 5,
      "cost": 10
    },
    {
      "weight": 10,
      "cost": 20
    }
  ],
  "totalCost": 30,
  "totalWeight": 15
}
```

Constraints

1. If no valid combination of transfers is possible (e.g., all transfers exceed the weight limit), return an empty list and **`totalCost = 0`**.
2. Transfers cannot be split (e.g., you must take the full weight of each transfer if selected).

Deliverables

1. A fully functional **Spring Boot** application built using **Java** or **Kotlin**.
2. The application should:
    a. Have a **REST API** for calculating the cheapest transfer route.
    b. Split the logic into **appropriate layers** (**Controllers, Services, Repositories**) as necessary.
3. Include well-written **unit tests** and **integration tests** for the **logic** and **API endpoints**.
4. The application should be configured to run using **Gradle** or **Maven** for building and dependency management.
5. The application should run as a **JAR** file after being built. No Dockerization is required.
6. Upload the code to a **public GitHub repository**.
7. Include a **README** file in the repository with:
    a. Clear instructions on how to build and run the application.
    b. Example **CURL** requests and responses for the API.