E-Commerce Shipping Charge Estimator

Problem Statement

Build a simple application with APIs to calculate the shipping charge for delivering a product in a B2B e-commerce marketplace.

Context

Imagine building a B2B e-commerce marketplace that helps Kirana stores discover and order the products they need to run their shops. It works similarly to consumer platforms like Flipkart or Amazon but focuses on the specific needs of small retailers.

Here are the entities involved in the marketplace:

Entities

Customer

Customers are Kirana stores. Assume the system stores complete customer details, including their location:

Customer	Name	Phone number	location
Cust-123	Shree Kirana Store	9847******	{ lat: 11.232, lng: 23.445495}
Cust-124	Andheri Mini Mart	9101******	{ lat: 17.232, lng: 33.445495}

Seller and Product

Sellers can sell products with attributes like weight and dimensions. Assume sellers are located anywhere in India.

Seller Name	**Product Name**	**Selling Price**	**Attributes**
Nestle Seller	Maggie 500g Packet	10 Rs	{ weight: 0.5kg, dimension: 10cmx10cmx10cm }
Rice Seller	Rice Bag 10Kg	500 Rs	{ weight: 10kg, dimension: 1000cmx800cmx500cm }
Sugar Seller	Sugar Bag 25kg	700 Rs	{weight: 25kg, dimension: 1000cmx900cmx600cm }

Warehouse

Our marketplace warehouses are present across the country.

Warehouse Name	Attributes
BLR_Warehouse	{lat: 12.99999, long: 37.923273}
MUMB_Warehouse	{lat: 11.99999, long: 27.923273}

When a customer places an order for a seller's product, the seller finds the nearest warehouse to his location and drops the items.

Delivery Logistics

After the seller drops the items at the warehouse, products are shipped from the warehouse to the customer's location.

- The transport mode depends on the distance.
- Each of the above transport modes has different rates charged.

Transport Mode	**Distance**	**Rate**
Aeroplane	500Km+	1 Rs per km per kg
Truck	100Km+	2 Rs per km per kg
Mini Van	0-100Km	3 Rs per km per kg

At the same time, the customer has multiple delivery speeds depending on their needs.

Delivery Speed	**Amount**
Standard	Rs 10 standard courier charge + **calculated shipping charge on items**
Express	Rs 10 standard courier charge + Rs 1.2 per Kg Extra for express charge + **calculated shipping charge on items**

Assignment Requirements

You need to implement the following APIs to support the above functionality:

---

1. Get the Nearest Warehouse for a Seller

Description: Given a seller and product, return the nearest warehouse where the seller can drop off the product.

Endpoint:

```
GET /api/v1/warehouse/nearest
```

Sample Request:
http
Copy code
```
GET /api/v1/warehouse/nearest?sellerId=123&productId=456
```

Sample Response:

```
{
  "warehouseId": 789,
  "warehouseLocation": { "lat": 12.99999, "long": 37.923273 }
}
```

---

2. Get the Shipping Charge for a Customer from a Warehouse

Description: Given the warehouse ID and customer ID, return the shipping charge based on the distance and transport mode.

Endpoint:

```
GET /api/v1/shipping-charge
```

Sample Request:

```
GET
/api/v1/shipping-charge?warehouseId=789&customerId=456&deliverySpeed=standard
```

Sample Response:

```
{
  "shippingCharge": 150.00
}
```

---

3. Get the Shipping Charges for a Seller and Customer

Description: Given a seller and customer ID, calculate the shipping charges by combining the nearest warehouse retrieval and shipping charge calculation.

Endpoint:

```
POST /api/v1/shipping-charge/calculate
```

Sample Request:

```
{
  "sellerId": 123,
  "customerId": 456,
  "deliverySpeed": "express"
}
```

Sample Response:

```
{
  "shippingCharge": 180.00,
  "nearestWarehouse": {
    "warehouseId": 789,
    "warehouseLocation": { "lat": 12.99999, "long": 37.923273 }
  }
}
```

- ## Additional Guidelines

All the entities mentioned in the document are for example, do not limit yourself to the attributes given for the entities and be creative in identifying more by taking inspiration from e-commerce

applications and keeping yourself in the shoes of a Kirana owner who is ordering products on this app.

Must have

1. Store these entities in **any of the data stores** you are comfortable with.
2. Ensure the APIs handle invalid or missing parameters, no warehouses found, unsupported delivery locations, or any such situations gracefully with good **exception handling**.
3. Write clean, modular, and well-documented code. Include comments where necessary`.

Good to have

1. Using **Design Patterns** to make code extensible
2. **Testing**: Write unit tests to validate API functionality and edge cases.
3. **Caching** the response for any of the APIs for faster response times

In case of any queries, please mail at shreya.palit@jumbotail.com