## PHASE 3: APEX CODE IMPLEMENTATION

**NearestDriverService Apex Implementation**

Class Architecture

Wrapper Classes:

* DriverRequest: Takes parameters such as deliveryId, latitude, longitude, and hubId (as flow action inputs).
* DriverResponse: Returns driverId (assigned), estimated distance (km), and ETA (hours) to flow.

Core Method:

text

@InvocableMethod(label='Find Nearest Driver')

public static List<DriverResponse> findNearestDrivers(List<DriverRequest> requests) {

DriverRequest request = requests[0];

Decimal deliveryLat = request.latitude;

Decimal deliveryLon = request.longitude;

List<Driver\_\_c> drivers = [

SELECT Id, Current\_Latitude\_\_c, Current\_Longitude\_\_c

FROM Driver\_\_c

WHERE Status\_\_c = 'Available'

AND Current\_Latitude\_\_c != null

AND Current\_Longitude\_\_c != null

];

// (See distance/selection logic below)

}

Algorithm & Integration

Distance Calculation:

text

public static Decimal haversine(Decimal lat1, Decimal lon1, Decimal lat2, Decimal lon2) {

Decimal R = 6371; // Earth's radius in km

Decimal dLat = toRad(lat2 - lat1);

Decimal dLon = toRad(lon2 - lon1);

Decimal a = Math.sin(dLat/2) \* Math.sin(dLat/2) +

Math.cos(toRad(lat1)) \* Math.cos(toRad(lat2)) \*

Math.sin(dLon/2) \* Math.sin(dLon/2);

Decimal c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

return R \* c;

}

public static Decimal toRad(Decimal deg) {

return deg \* Math.PI / 180;

}

Driver Selection Logic:

text

Decimal minDist = Decimal.valueOf(999999);

Id minDriverId;

for(Driver\_\_c driver : drivers) {

Decimal dist = haversine(deliveryLat, deliveryLon, driver.Current\_Latitude\_\_c, driver.Current\_Longitude\_\_c);

if(dist < minDist) {

minDist = dist;

minDriverId = driver.Id;

}

}

DriverResponse response = new DriverResponse();

if(minDriverId != null) {

response.driverId = minDriverId;

response.estDistanceKm = minDist.setScale(2);

response.estETAHours = (minDist / 30).setScale(2); // ETA: km/30kmh

}

return new List<DriverResponse>{response};

Flow Integration

* Inputs: deliveryId, latitude, longitude, hubId
* Outputs: driverId, estDistanceKm, estETAHours
* Error Handling: Handles no-available-driver case with null driverId in response; works with Flow checks for assignment attempts and empty results.

This code encapsulates the business rule that the nearest available driver by geospatial distance is always assigned; results integrate seamlessly with Salesforce Flow for further automation (assignment, task creation, ETA calculation, etc.). This structure is robust for production logic and highly extensible for future optimization.Here’s a complete summary and clarification of your NearestDriverService Apex implementation for Smart Delivery Hub:

**NearestDriverService Class Architecture**

* DriverRequest: Input wrapper for Flow or Apex actions (deliveryId, latitude, longitude, hubId).
* DriverResponse: Output to Flow (driverId, estimated distance in km, ETA in hours).

**Key Methods & Logic**

Method:

text

@InvocableMethod(label='Find Nearest Driver')

public static List<DriverResponse> findNearestDrivers(List<DriverRequest> requests)

* Handles a single delivery/location per call (for flow use).

Distance Calculation (Haversine Formula):

* Calculates driving distance between delivery and each available driver.

text

public static Decimal haversine(Decimal lat1, Decimal lon1, Decimal lat2, Decimal lon2)

Driver Selection:

* Loops through all "Available" drivers with coordinates.
* Compares distance, keeps track of the nearest.
* Sets response fields (driverId, estDistanceKm, estETAHours [distance ÷ 30km/h]).

Result Returned To Flow:

* Flow receives driverId, estimated distance, ETA (hours), then updates the Delivery record.

Error Handling:

* If no drivers found, driverId is null and Flow can handle exceptions/notifications.