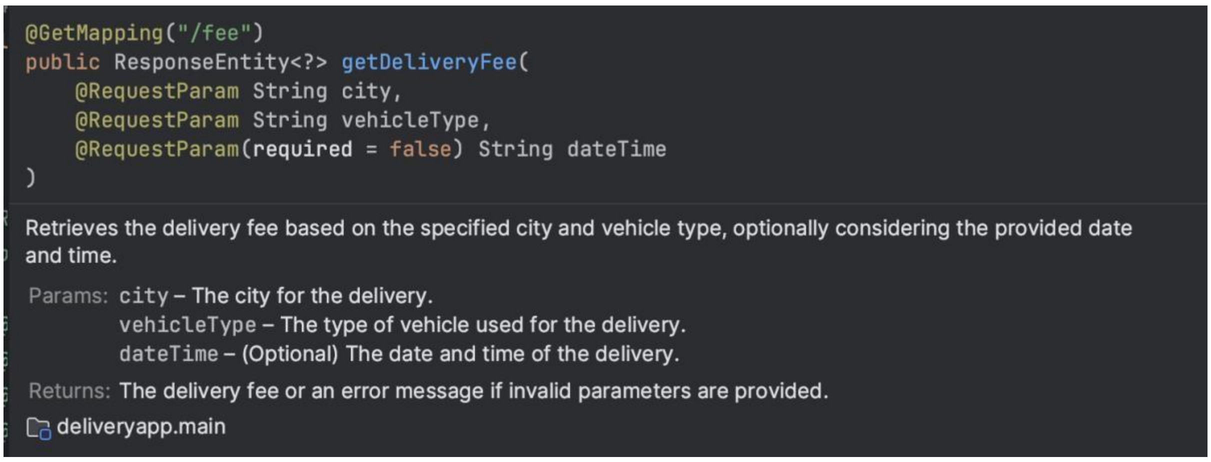
DOCUMENTATION

Documentation detailing the public methods of the sub-functionality of the food delivery application, which calculates the delivery fee for food couriers based on regional base fee, vehicle type, and weather conditions.

DeliveryController class

Controller class responsible for handling HTTP requests related to delivery operations. It provides endpoints for calculating delivery fees based on various parameters.

Public methods



```
@GetMapping("/fee")
public ResponseEntity<?> getDeliveryFee(
    @RequestParam String city,
    @RequestParam String vehicleType,
    @RequestParam(required = false) String dateTime
)
```

Retrieves the delivery fee based on the specified city and vehicle type, optionally considering the provided date and time.

Params: `city` – The city for the delivery.
`vehicleType` – The type of vehicle used for the delivery.
`dateTime` – (Optional) The date and time of the delivery.

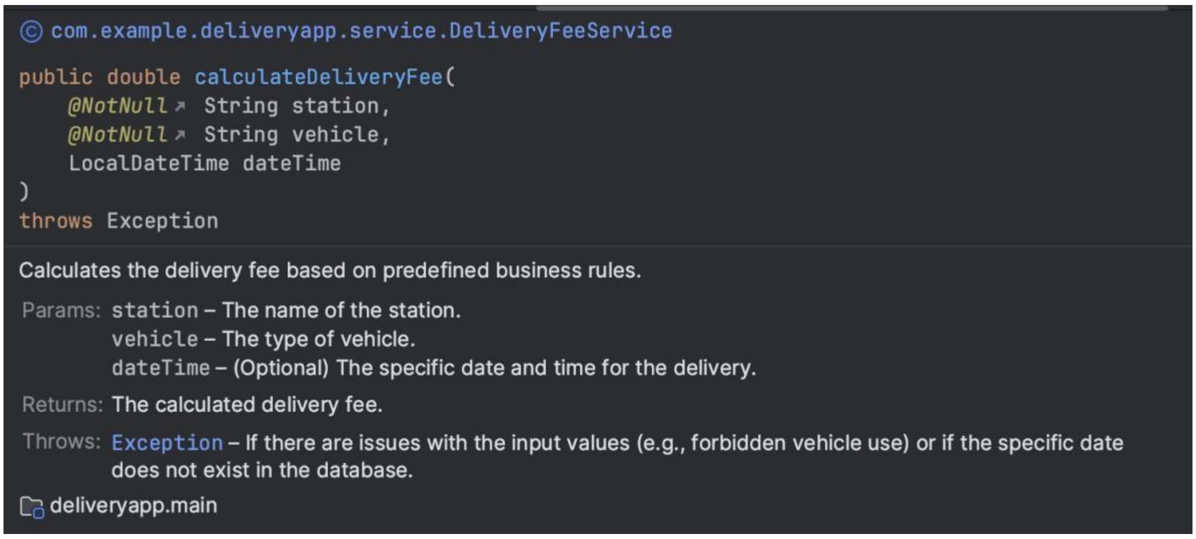
Returns: The delivery fee or an error message if invalid parameters are provided.

deliveryapp.main

DeliveryFeeService class

Service class responsible for calculating delivery fees based on predefined business rules. It handles the logic for determining the delivery fee considering factors such as city, vehicle type, and weather conditions.

Public methods



```
com.example.deliveryapp.service.DeliveryFeeService

public double calculateDeliveryFee(
    @NotNull String station,
    @NotNull String vehicle,
    LocalDateTime dateTime
)
throws Exception
```

Calculates the delivery fee based on predefined business rules.

Params: `station` – The name of the station.
`vehicle` – The type of vehicle.
`dateTime` – (Optional) The specific date and time for the delivery.

Returns: The calculated delivery fee.

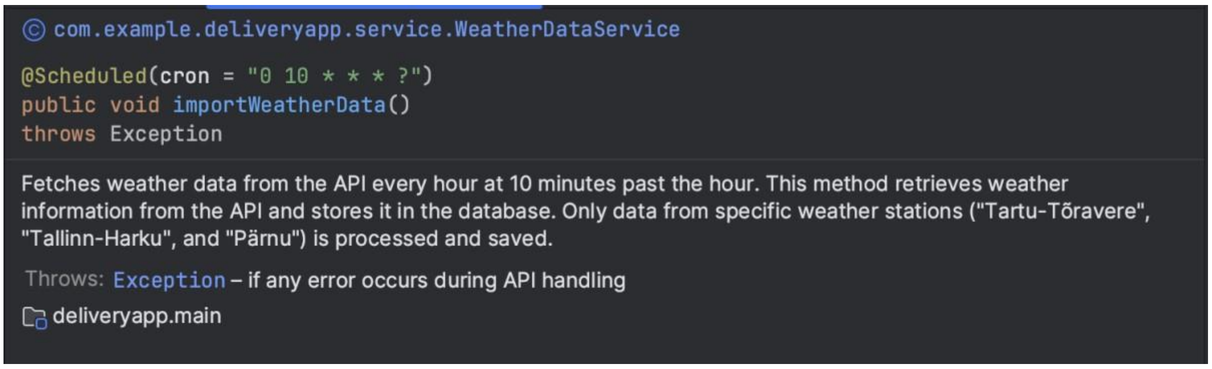
Throws: `Exception` – If there are issues with the input values (e.g., forbidden vehicle use) or if the specific date does not exist in the database.

deliveryapp.main

WeatherDataService class

Service class responsible for retrieving and importing weather data from an external API. It handles the logic for fetching weather data, processing it, and storing it in the database. Since weather observations from ilmateenistus.ee are updated every 10 minutes past the full hour, I import weather data every hour at the 10th minute.

Public methods



```
com.example.deliveryapp.service.WeatherDataService

@Scheduled(cron = "0 10 * * * ?")
public void importWeatherData()
throws Exception
```

Fetches weather data from the API every hour at 10 minutes past the hour. This method retrieves weather information from the API and stores it in the database. Only data from specific weather stations ("Tartu-Tõravere", "Tallinn-Harku", and "Pärnu") is processed and saved.

Throws: Exception – if any error occurs during API handling

deliveryapp.main