Aggregation Servie

Technical decisions:

AggreagationFacade

1. When a request comes in is handled by AggreagationFacade, which is a thin layer between controller and services it will delegate and coordinate all service calls
2. The Facade will split request the data from each service (PricingService, TrackService, ShipmentService). it creates a task that will schedule with Schedulers.bondedElastic() that will manage on what thread to run the task. Waits for all task to complete and aggregate the data.

Services (Shipment, Pricing, Track)

1. All services are very similar(TODO: enhance the reusability, super class or more generic)
2. The service receives a list of keys that has to find a corresponding response for (Key-Value)
3. The service has a Queue with main fields RequestSet and ResponseMap
4. In order to be able to find the Key-Value pair, the service creates a Result object with the keys that has to find responses for and it subscribes to the Queue.ResponseMap to check the responses regularly. When it finds all responses it finishes and return the result back to aggregation Service. In the same time it cleans the ResponseMap.
5. In order to match any response the Service registers the received keys in the Queue.RequestSet that will trigger a WebClient call to fetch the data and fill up the responseMap. The call is triggered in two cases
    a. When the Set reaches the throttle limit and
    b. If the limit is not reached. When a result object subscribes to check the responseQueue, it schedules a task with throttle timeout.

WebClient(Pricing, Shipment, Track)

Simple webClients requests, in case of 503 they will return a map with key- null.

Tests

1. I add more integration tests as it's more resilient in case the implementation changes, more often the API specs remain the same or with minor changes.
2. WireMock was used to mock the requests