Shopping System Design

Application will allow to create/update/delete order.Whenever order is created/ updated/deleted events are emitted to inventory,Billing & notification.All data would be stored in the json file.It should handle failure of events too

Functional Requirements

- User should able to Create Order
  - emit event to inventory service for the stock
  - emit event to log details for billing service
  - emit event to notification to send email
- User Should able to Update Order
  - emit event to inventory service to update stock
  - emit event to log details for billing service
  - emit event to notification to send email
- User Should able to Delete Order
  - emit event to inventory service to update stock
  - emit event to log details for billing service to deletail
  - emit event to notification to send email
- In case of failure data woule be written in failure.json

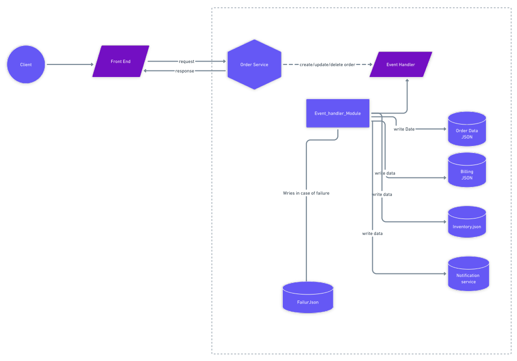
Non Functional requirements`

- System should be highly available
- Order Process has to be Asynchronous
- Maintainability of the service
- We need to think about Performance
- Monitoring, Logging & Obvserbality
- Security Compliance with TLS and HTTPs
- Documentations

Use Cases

- User should able to create order using any Interface by providing order data
- User should able to View the list of orders and Order details by ID
- User should be able to update the Order using User Interface by providing order details
- User should able to delete / cancel the order by using order ID
- User should notify upon order modified
- Whenever order is created/updated/deleted events should be emitted to inventory,billing and notification service
- In case of failure data should be stored in failure Json file

High Level Diagram



Data Storage

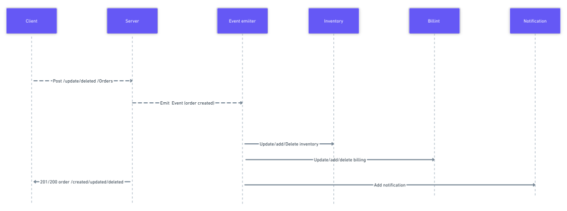
All data now will be stored in the json so we can use flat file storage as per the requirement

Order Table		
order_id	pk	
customer_name		
total_amount		
items	array (product, items, amount)	
status		
order_date		
order_update		

Service Endpoints

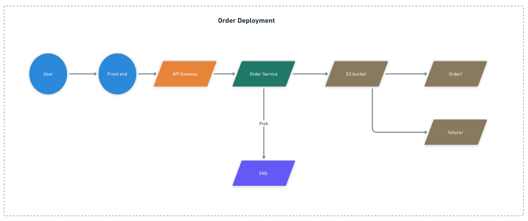
- **Post /orders**
- request : items product, quantity,amount
- response : order_id, total_amount, items[product name,qunatity,amount]
  - Emit event to Inventory
  - emit event to billing
  - emit evnet to notification
- **Patch/orders/:id**
- request : items product, quantity,amount
- response : order_id, total_amount, items[product name,qunatity,amount]
  - emit event to inventory
  - emit event to billing
  - emit event to notification
- **delete/order/:id**
- request : order_id
- response : deleted successfully (true/false)
  - emit event to inventory
  - emit event to billing
  - emit event to notification

Sequence Daigram


Client
Server
Event emiiter
Inventory
Billint
Notification
Post /update/deleted /Orders
Emit Event (order created)
Update/add/Delete inventory
Update/add/delete billing
201/200 order /created/updated/deleted
Add notification

Once customer sends a reuqest to create/update /delete order , backend creates an order once order is created successutlly events are emitted to inventory/Billing and Notifcation about it.If the event emitter fails we have json file to store all the failures.

Deployment Strategy



- The user initiates an order request through a front-end application or interface.
- The request is routed through an API Gateway, which handles forwarding it to the backend service.
- If the order is successful, it is saved in an S3 bucket under the order/ folder.
- If the order fails, it is saved under the failure/ folder in the same bucket.
- Once the order is successfully created, the Order Service pushes a notification to an SNS topic.
- This allows other services to subscribe and react to the event in real time.