**Online marketplace – Problem statement, overview, architecture**

This document mentions about problem statement given, proposed architecture for online market place, assumptions, design and implementation details of sub components using Java technology stack.

**Problem statement**:

*Build REST API that supports the new marketplace website.*

*In-memory database is sufficient. Optionally, you are welcome to use a persistent data store of your choice.*

*You are encouraged but not required to take advantage of a service code-generation framework of your choice when performing this exercise*.

**Overview**:

In nutshell online marketplace is a type of e-commerce website where product or service information is provided by multiple third parties (source – Wikipedia).

**Preamble and assumptions**:

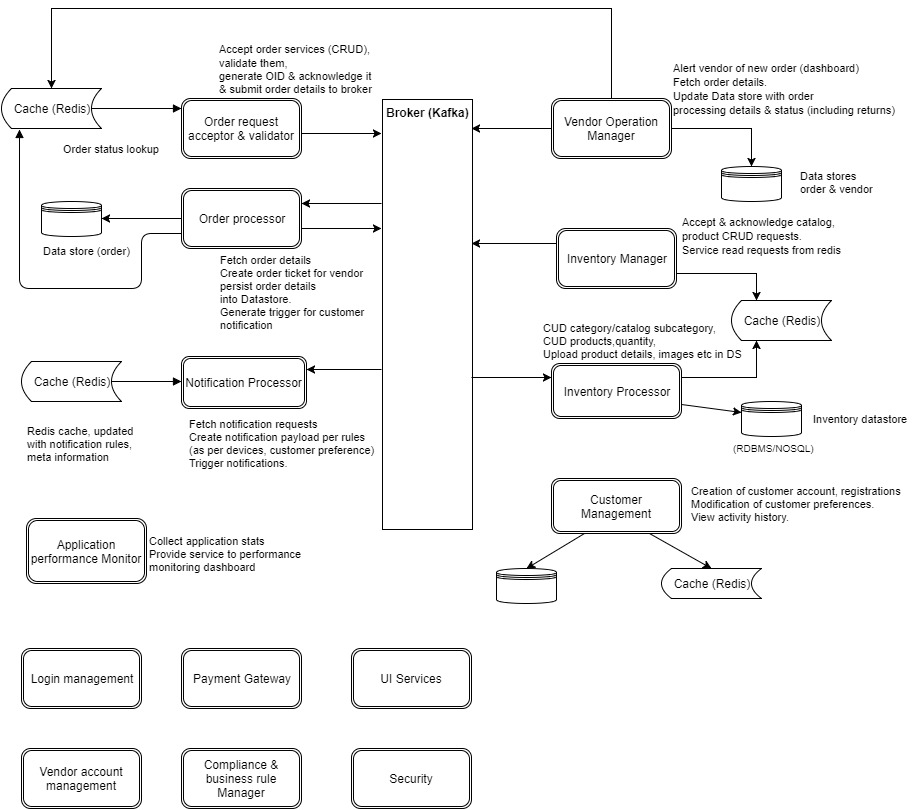
As the problem statement is open ended and subject covers a lot of sub topics, attempt is made to highlight major components & their functional detailing. And also to come up with architecture for its enterprise level implementation. (It does not detail all components)

Considering the open ended nature of problem statement and time availability at hand, the implementation provides **REST APIs for order processing, inventory and vendor activities** using in memory data base (H2) with Java tech stack. The interaction with other components and other enterprise level features like caching, usage of message broker, security, login etc. are omitted.

**Architecture**:

Following diagram details high level architecture for online market place. It details only core components, from backend, for market place as described below:

1. **Order management**: It is major component of online market place which enables customers to place order for chosen products. It is partitioned in two parts, first to accept the order related requests and other to do actual processing of the request. The first part also does the validation of the request (e.g. quantity requested against available quantity, physical deliverability, some business rules etc.). The decoupling of order request accepting from order processing allows both of them to scale independently.
2. **Inventory management**: It allows the addition/alterations of catalogs/categories, products in the system. It is kind of **min-ERP** part of the online market place that enables display of various products across categories on UI. Vendor may also use it to update some product quantities.
3. **Vendor operation management**: For vendor once received the order for delivery, his work, which is mostly offline to the system, starts which also include processing of returned products too. Across his actions, the status of the order changes (from order received, packed, shipped, delivered). This module provides services for vendor to facilitate order status updates into system.
4. **Notification manager**: Sends various notifications to customers as per their preferences registered with system on their selected devices (SMS, push notifications, emails etc.).
5. **Customer management**: Accounts for customer registration, setting/altering various preferences. View activity history, register addresses, preferred pay media etc.
6. **Login management**: Takes care of login management for different users including customer, different level of administrators, super administrators, vendors.
7. **Vendor account management**: For vendor to accept legal terms and condition updates, payment settlement and other legal requirement fulfilments.
8. **Payment gateway**: Registration, integration with various payment gateways.
9. **Compliance and business rule manager**: As online marketplace needs to adhere to various legal conditions this unit may enforce the same on all actors of system (as applicable), it may generate associated business rules as well. Also, some rules like discounts, quantity restrictions can be managed from here.
10. **UI services**: Takes care of web site static contents, site maps etc.
11. **Security**: Handles the security of the application.
12. **APM**: Application performance monitoring, event log etc. provide deep insights into application performance. It can collect these vital stats of the system and present on internal dashboard.



The technology stack is as mentioned in diagram.

The **decoupling** of request acceptance and processor allows for independent scaling of the components.

The system is mostly **distributed** wherein different components are communicating with each other via message queue/broker (e.g. Kafka). High traffic end points like order placement can be put behind **load balancer** to scale the request accepting.

Most of the static or less prone to change data is **cached** in Redis, which is expected to be updated as the information is updated by its source (by some source-sink or backend reconciler component).

With adoption of distributed processing along with above mentioned technologies, the overall system is aimed for **eventual consistency**.

**Demo Implementation**:

As mentioned above the implementation code details are for providing REST APIs for **order processing, inventory and vendor activities** using in memory data base (H2) with Java tech stack. In altogether, following use cases are implements:

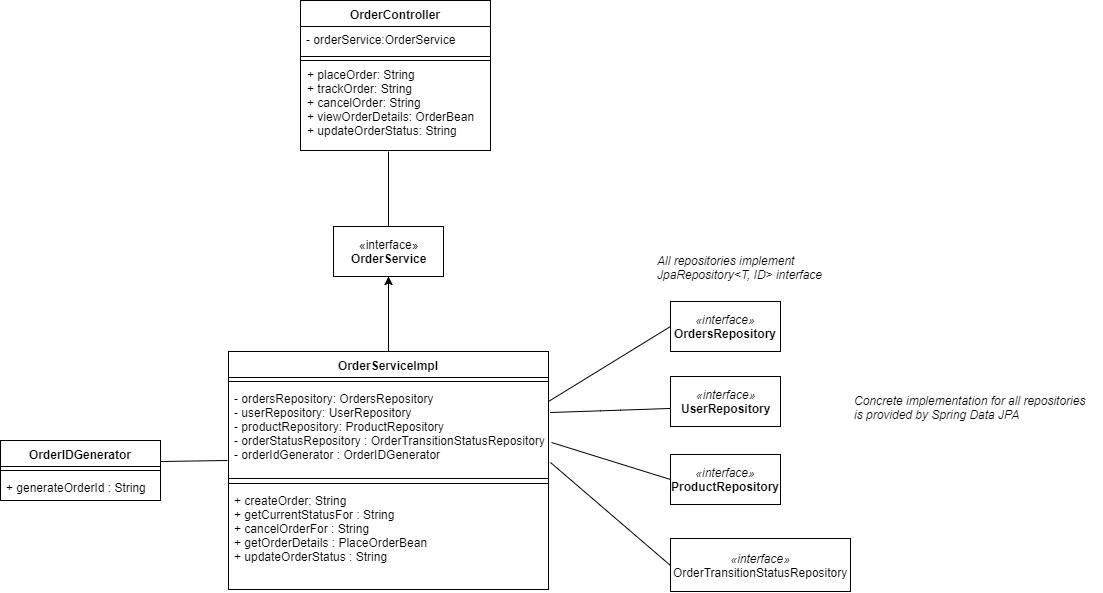
1. Place order
2. Track order with order code
3. Cancel order
4. View complete order details
5. Update order status (vendor activity)
6. Claim order (by vendor)
7. Add categories
8. Add products across categories
9. Update product quantities
10. Alter product availability flag (service to show/hide products on UI)

As application is not built using enterprise level solutions, it makes certain assumptions as commented in code.

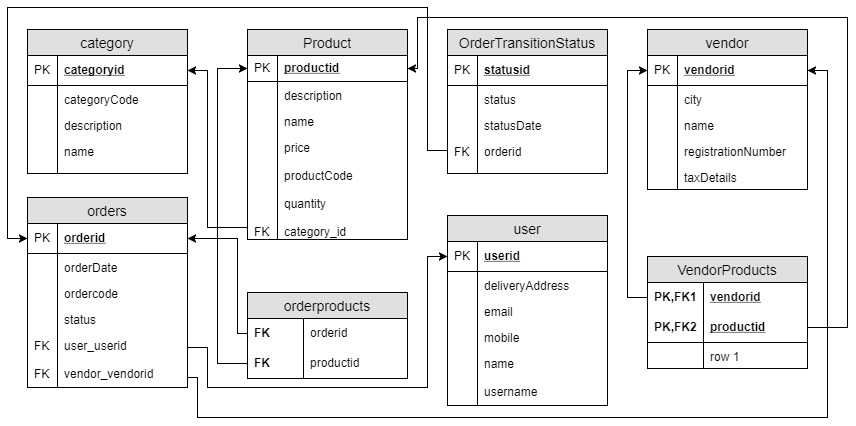
**Technologies used**:

The application is built using **java, Spring boot, spring data REST, spring data JPA, H2** data base (in memory).

Application uses controller, Service, Repository pattern primarily. The controllers are as per responsibility e.g. a controller for order related request processing, similarly there are controllers for vendor and inventory related tasks. Following class diagram highlights the same for order management. Other controllers also follow similar design pattern for this demonstration code.



Following diagram shows the DB tables involved and their relationships.



For demo applications, tables are created by spring boot data JPA component (vendor- Hibernate). The tables are populated with some default information from data.sql file.