Order Management System

**N-Views Architecture Representation Document**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | OMS Team |  |  |
| **Role** | Technical Analyst |  |  |
| **Signature** |  |  |  |
| **Date** | 22/9/2017 |  |  |

Table of Contents

[1.0 Introduction 3](#_Toc493869367)

[1.1 Purpose of this document 3](#_Toc493869368)

[1.2 Scope 3](#_Toc493869369)

[1.3 Structure of the document 4](#_Toc493869370)

[2.0 Project Overview 4](#_Toc493869371)

[2.1 Objectives 4](#_Toc493869372)

[3.0 Architecture Representation 5](#_Toc493869373)

[3.1 Architecture Overview 5](#_Toc493869374)

[3.2 Drivers 7](#_Toc493869375)

[3.2.1 Business Drivers and Goals 7](#_Toc493869376)

[3.2.2 Architectural Drivers and Goals 7](#_Toc493869377)

[3.2.3 Non-Functional Requirements 7](#_Toc493869378)

[3.3 Architecture Principles 8](#_Toc493869379)

[3.4 Key Architecture Decisions 9](#_Toc493869380)

[3.4.1 Technology Stack Rationalization 9](#_Toc493869381)

[4.0 Architecture Representational Views 10](#_Toc493869382)

[4.1 Functional View 10](#_Toc493869383)

[4.2 Layer View 10](#_Toc493869384)

[4.3 Micro-Service Architecture design patterns 12](#_Toc493869385)

[4.4 Activity Diagram 14](#_Toc493869386)

[4.4.1 Order Management System 14](#_Toc493869387)

[4.4.2 Product Management 15](#_Toc493869388)

[4.4.3 Inventory Management 16](#_Toc493869389)

[4.4.4 Order Management 17](#_Toc493869390)

[4.4.5 Customer Management 18](#_Toc493869391)

[4.5 Class Diagram 19](#_Toc493869392)

[4.6 Integration View 19](#_Toc493869393)

[4.7 Security Strategy 22](#_Toc493869394)

[4.7.1 Authentication 22](#_Toc493869395)

[4.8 Data View 22](#_Toc493869396)

[4.8.1 Executive Summary 22](#_Toc493869397)

[4.8.2 Choice of target Database 22](#_Toc493869398)

[4.8.3 Database and Shema Design 23](#_Toc493869399)

[5.0 Deployment 26](#_Toc493869400)

# Introduction

## Purpose of this document

The purpose of this software architecture is aimed at describing the architecture of the Order Management System based on multiple, concurrent views all of which can be combined to create a holistic view of the system covering both software requirements and non functional requirements. This document also addresses solutions for various functional components like

* Logging
* Exception Component
* Security
* Performance
* System management and maintainability

Describe the high-level architecture and high level design of the system.

* Describe precisely the responsibilities, relationships, and interactions of components.
* Document how application/technical parts of the system are related.
* Specify how existing and developed components are related.

## Scope

**In HLD Scope**

The scope of this document includes tools, technologies, patterns and approaches that will be used in the design and development, deployment and support and enhancements (evolution) of the proposed system. The document also brings out the rationale for arriving at a specific technology or tool or framework.

The document describes the application architecture using the N Views architecture model

The application is explained with the following architecture views

1. Functional View
2. Layered View
3. Security View
4. Data View

**In LLD Scope**

* Low level Design aspects (Class or Sequence Diagrams) of the system.

## Structure of the document

This report is structured in such a manner that it starts with the overall project context detailing the objectives and constraints, key stake holders, key areas of concerns and drivers. The subsequent section details the architectural representation of the envisaged architecture followed by a number of views of the application architecture.

# Project Overview

## Objectives

The objective of the Order Management System (OMS) Project is to provide target architecture for the

various inventory, order mangement activities . An OMS may interface with a multitude of other technical components.

Order Management System typically offer:

* Inventory Service
* Product Service
* Order Service
* Billing Service
* Customer Service
* Payment Service
* Event Service
* Mailing Service

# Architecture Representation

## Architecture Overview

The proposed application architecture is based on the following principles:

* A multi-Tiered, multi-layered architecture that leverages the patterns and open standards
* A well abstracted modular, scalable, flexible and extensible architecture
* Clear separation of roles and responsibilities of various components and sub-systems in the solution
* Enhance creation and dissemination of re-usable assets and their alignment with open industry standards available in this solution space

FUNCTIONAL VIEW

LAYER VIEW

SECURITY VIEW

DATA VIEW

Figure 1: Architecture View

|  |  |
| --- | --- |
| View | Brief Description |
| Functional View | Functional view provides a snap shot of the functional context of the system. Functional view focuses on identifying functional requirements that are important to business. This view identifies the business functions and services that are expected from the solution and its potential users. |
| Layered View | Layered view provides the perspective of how the services, Tiers/layers and components are modeled within the concerned system. This view provides the key Tiers, components and design blocks that are used in order to realize the complete architectural solution. |
| Security View | Security view provides details of the capability of the envisaged system to monitor, manage and maintain security related aspects of the system. This view provides a perspective of the system from access control, authentication and authorization perspectives. It identifies and elaborates the security mechanisms & standards that need to be realized along with identification of tools & technologies, commercial and custom components that will be leveraged to fulfill the security needs of the system. |
| Data View | Data View provides the Entity relationship between different Business Entities and also the Logical Data Model of the system |

Table 1: Architecture Definition Views

## Drivers

### Business Drivers and Goals

The following are the business drivers considered for Order Management System architecture

1. To evaluate new technology to learn PCF and Micro services.
2. To explore Mongo DB services, Splunk and Implementing JWT Security

### Architectural Drivers and Goals

The following are the architectural drivers considered for Order Management System architecture

1. Scalability
2. Reliability
3. Performance

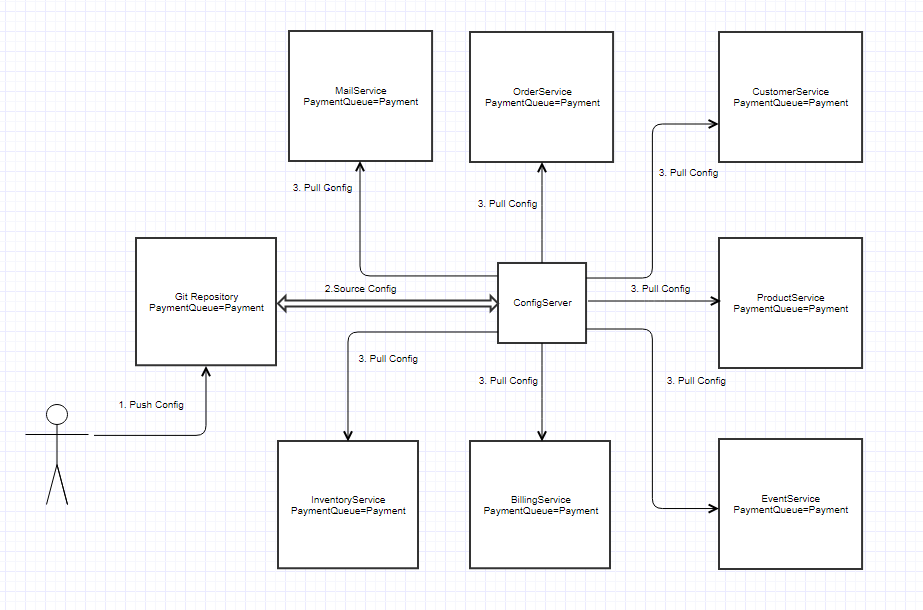
### Non-Functional Requirements

#### Configuration Management

##### Config Server

Config Server for [Pivotal Cloud Foundry](https://network.pivotal.io/products/pivotal-cf) (PCF) is an externalized application configuration service, which gives you a central place to manage an application’s common properties.

OMS uses ConfigServer (Instance name will be updated later) to load the application’s common properties from GitHub.



#### oms-common-security

##### JWT

JSON Web Token (JWT) is an open standard which is used for security implementation with the help of Spring Security Wrapper Implementation. All the external request(s) to any micro service in OMS is authenticated at **api-gateway** and all the requests between services via IPC are also validated with token.

#### oms-common-web

##### Swagger

[swagger](https://developers.helloreverb.com/swagger/) is a specification for documenting REST API. It specifies the format (URL, method, and representation) to describe REST web services. It provides also tools to generate/compute the documentation from application code.

OMS makes use of Swagger to documen RESTful services and this will also be also helpful to test.

##### Logging

OMS only need spring-boot-starter-web since it depends transitively on the logging starter (**spring-boot-starter-logging**).

Spring Boot has a LoggingSystem abstraction that attempts to configure logging based on the content of the classpath.

If the only change you need to make to logging is to set the levels of various loggers then you can do that in **application.properties** using the "logging.level" prefix, e.g.

# Enable Logging pattern

logging.level.org.springframework.web=ERROR

logging.com.oms.Customer=DEBUG

##### 

##### ExceptionHandling

OMS uses **circuit-breaker** MSAP with **Netflix** implementation of **Hystrix** for system/user defined exception handling and calling the fallback method.

##### 

## Architecture Principles

* Architecture must have distinct layers. These layers need to be loosely coupled and accessible through well defined interfaces and should be deployable independent region of the other layers
* The architecture will make use of the technical capability offered by light weight components such as Spring boot in PCF
* Communication interfaces between the layers should be based on open standard interfaces and protocols
* Common application server independent security framework (Spring Security and JWT) shall be used for Authentication purpose.

## Key Architecture Decisions

This section documents the important decisions made to select the right technology stacks for the given Order Management Systemfrom the aspect of the architecture and structure of the system that are given or have been made.

### Technology Stack Rationalization

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Key Architecture Decision (KAD)** | **Commercial / Open Source** | **Rationale/Driver** |
| KAD1.0 | Spring boot 1.5.4 | Open Source | Spring boot is light weight component having embedded tomcat.  Enhanced Scope Management.  Spring Security native support. |
| KAD 2.0 | MongoDB 3.4.0 | Open Source | NoSql database.  Maintaing data as a collection in json format.  Ease of deployment & maintainability. |
| KAD 3.0 | RabbitMq 1.4.6 | Open Source | Messaging broker.  Decoupling Microservices with Spring AMQP and RabbitMQ.  RabbitMQ is lightweight and easy to deploy on premises and in the cloud. |
| KAD 4.0 | Zuul | Open Source | Zuul is built to enable dynamic routing, monitoring, resiliency and security. |
| KAD 5.0 | Ribbon | Open Source | Inter Process Communication (remote procedure calls) library with built in software load balancer |
| KAD 6.0 | Eureka | Open Source | Service registry for resilient mid-tier load balancing and failover |
| KAD 7.0 | Hystrix | Open Source | Hystrix is a latency and fault tolerance library |
| KAD 8.0 | Redis | Open Source | [Caching](http://en.wikipedia.org/wiki/Cache_%28computing%29) is a way for applications to store data so that future requests |
| KAD 9.0 | Groovy 2.4.5 | Open Source | Writing Unit and Acceptance testcases using Spock framework |
| KAD 10 | Gradle | Open Source | Gradle helps teams build, automate and deliver better software, faster |

Table 2: Technology Stack Rationalization

# Architecture Representational Views

## Functional View

The functional view of Order Management System application is represented in the below pictorial view.Following are the list of the high level functional component of OMS.

* Inventory Service
* Product Service
* Order Service
* Billing Service
* Customer Service
* Payment Service
* Event Service
* Mailing Service

## Layer View

The layering architecture consists of the following layers

* Clients (Swagger, API Test suit)
* Service producer
* Business Layer
* Data Access Layer

These layers are designed in accordance with the architecture principle of enabling increased cohesion within the layer and reduced coupling across the layers. This logical separation of the

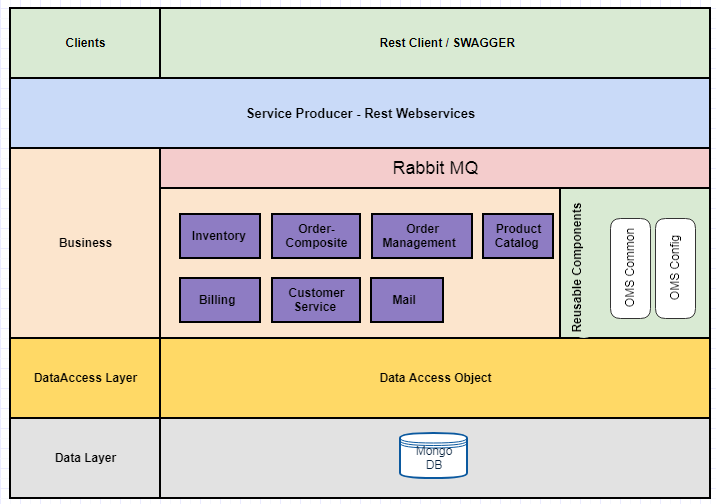


Figure 2: Layered views

architecture enables a clear separation of concerns between the systems thereby partitions the responsibility of the systems more appropriately.

* **Client**

The client layer represents the Rest Client(Postman) / Swagger / SoapUI tools through which users can access the Order Management System application.

* **Business Layer**

The business service layer comprises of components that realizes a specific business function or services of the Order Management System application. This layer includes the implementation of application specific business logic and performing business validations in addition to handling the security aspects of the application at this layer. The business layer is responsible for orchestrating with various components or services across the enterprise to realize a specific functionality in Order Management System application.

* **Data Access Layer**

This layer includes the components which are mainly supports for data access operation with underlying database server .It abstracts all the DB operation from the above layer.To improve the DB operation caching functionality also exposed in this layer.

* **Reusable Components**

This vertical column provides common, reusable components that can be used across the various components or services in different layers. The primary responsibility of this layer is to provide stand-alone components that can be plugged into any of the components in the other layers to realize their desired functionalities.

## Micro-Service Architecture design patterns

External API

* The API Gateway pattern  - The API gateway(Zuul provided by Netflix ) handles requests in one of two ways.
* External requests and Internal requests (via IPC) are simply proxied/routed to the appropriate service.
* In OMS, each external request is authenticated using JWT token procided Spring security wrapper implementation and each internal request (via IPC) is validated with the carried token.

Service Registry and Discovery

* Client-side Discovery  - When making a request to a service, the client obtains the location of a service instance by querying a Service Registry, which knows the locations of all service instances.
* In OMS, all the micro servies are registered in Service Registry (Eureka provided by Netflix) and when an external request comes to api-gateway , it routes to an actual endpoint by querying Service Registry
* Server-side Discovery - When making a request to a service, the client makes a request via a router (a.k.a load balancer) that runs at a well known location. The router queries a service registry, which might be built into the router, and forwards the request to an available service instance.
* In OMS , all the micro servies are registered in Service Registry (Eureka provided by Netflix) so when an internal request flows from one micro service to another , it will be routed/load balanced(in case of more than one instances created for target service) to an actual endpoint via Service Discovery(Ribbon provided by Netflix) which queryies Service Registry.

Communication style

* Messaging - OMS uses asynchronous messaging for inter-service communication as one of the options. Services communicating by exchanging messages via messaging broker as ***RabbitMq*** (Instance will be updated later).
* Also OMS user another way of REST API calling for internal service call via IPC using Service Discovery(Ribbon provided by Netflix) and Service Registry (Eureka provided by Netflix).

Deployment patterns

* Single Service per Host – OMS deploys each single service instance on its own host

Cross cutting concerns

* Externalized configuration – OMS externalizes all application configuration including the database credentials, Queue, exchange details and network location. On startup, a service reads the configuration from an external source via **ConfigServer**(Instance will be updated later) from property file placed at **GitHub**.

Data management

* Shared database - Use a (single) database that is shared by multiple services. Each service has its own collection and will access it freely, but some will refer more than one collection as per business requirement.

Security

* Access Token – In OMS, the API Gateway authenticates an external request and passes an access token (e.g. JSON Web Token) that securely identifies the requestor in each request to the services.
* In OMS, one service can include the access token in requests it makes to other services via IPC call using Service Discovery(Ribbon provided by Netflix) and the same will be validated by Spring security JWT mechanism placed in API gateway(Zuul provided by Netflix ).

Reliability

* Circuit Breaker - A service client should invoke a remote service via a proxy that functions in a similar fashion to an electrical circuit breaker. When the number of consecutive failures crosses a threshold, the circuit breaker trips, and for the duration of a timeout period all attempts to invoke the remote service will fail immediately.
* After the timeout expires the circuit breaker allows a limited number of test requests to pass through. If those requests succeed the circuit breaker resumes normal operation. Otherwise, if there is a failure the timeout period begins again.
* In OMS, the Circuit Breaker (Hystrix provided by Netflix) is used for calling fallback method on failure scenarios such as service down and exception thrown.

Observability patterns

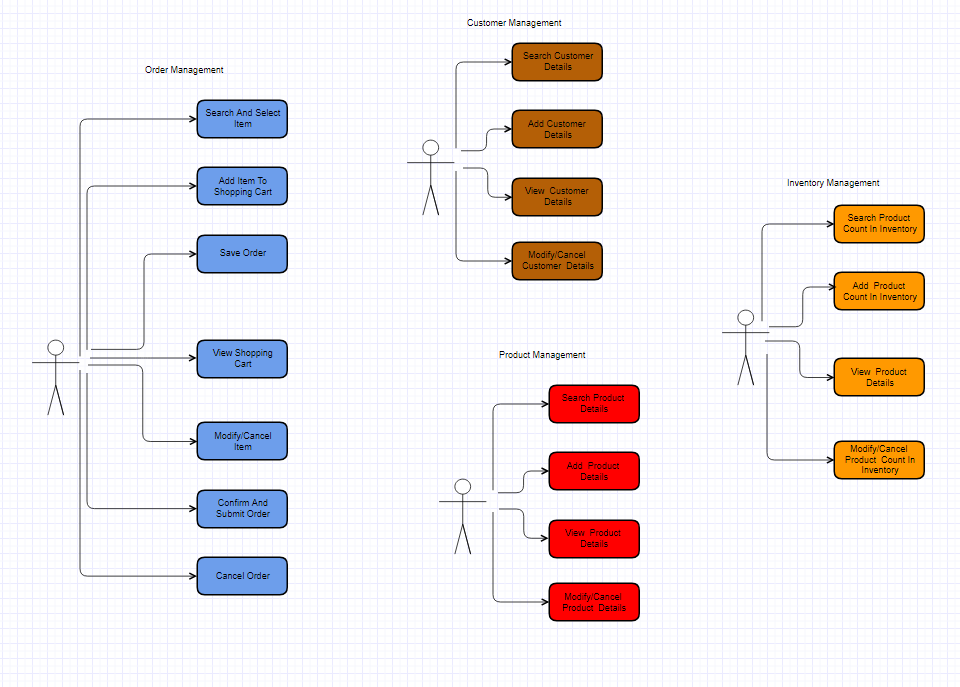
* Health check API - The application’s health will be checked using Spring Boot Actuator.

Testing patterns

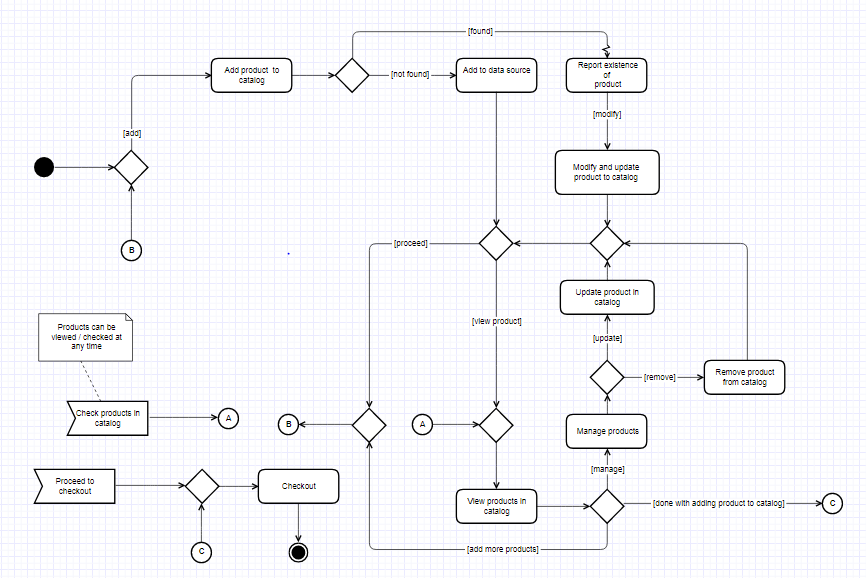
* [Service Component Test](http://microservices.io/patterns/testing/service-component-test.html) - A test suite that tests a service in isolation using test doubles for any services that it invokes.
* Service Integration Contract Test - A test suite for a service that is written by the developers of another service that consumes it. The test suite verifies that the service meets the consuming service’s expectations.

## Activity Diagram

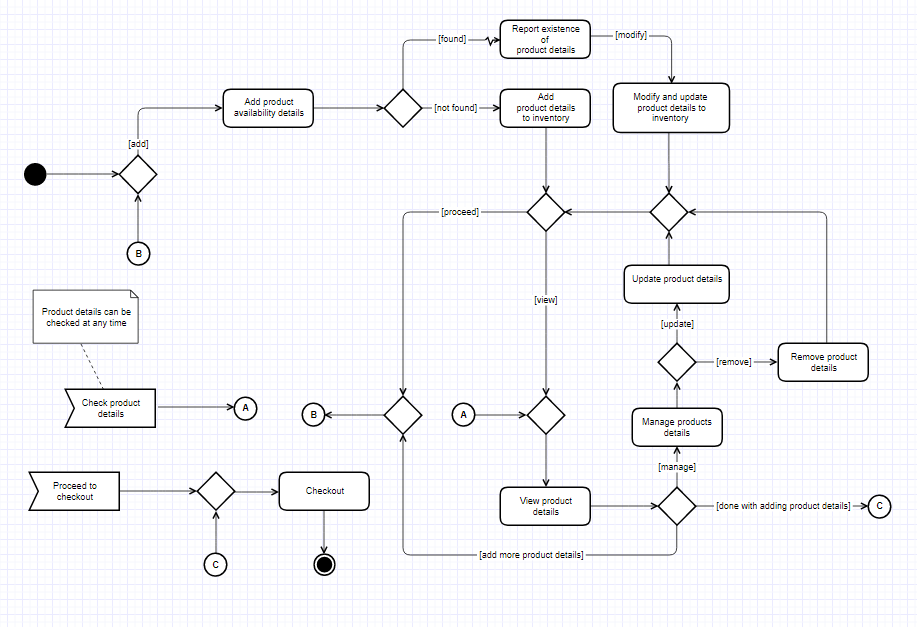
### Order Management System



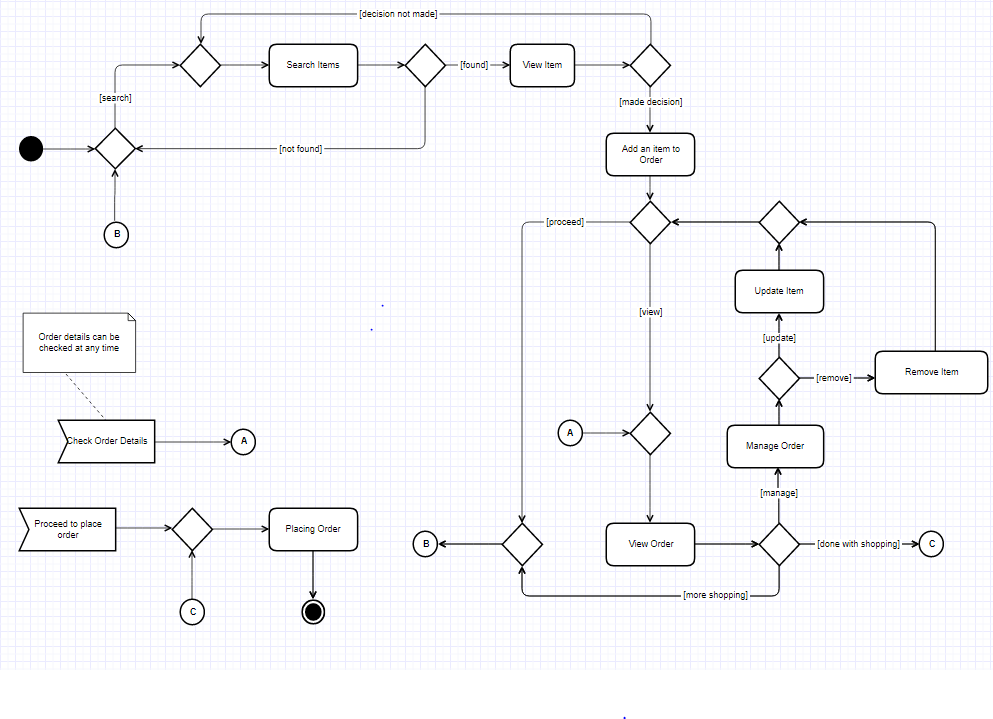
### Product Management



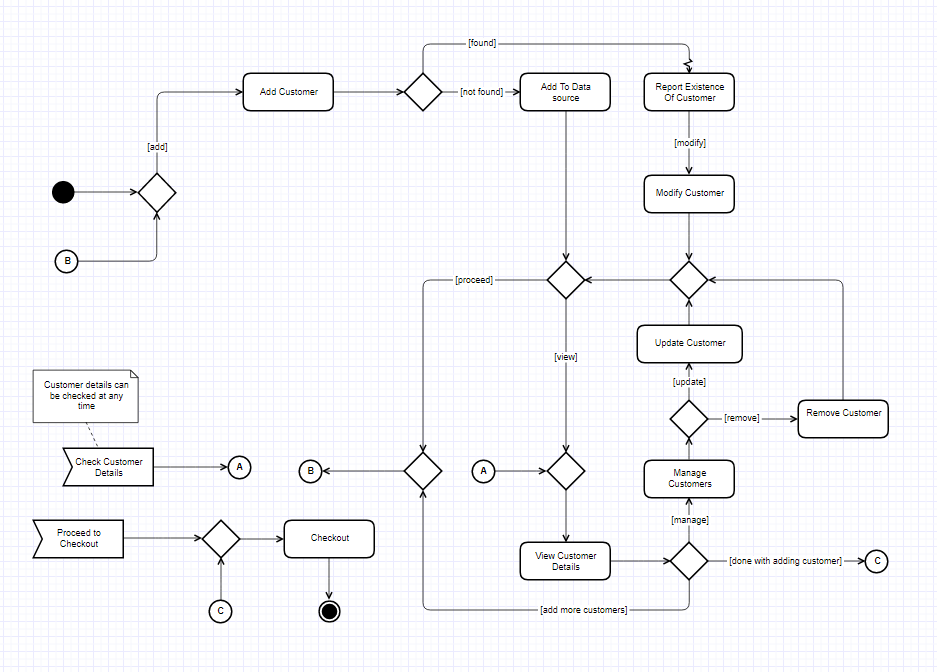
### Inventory Management



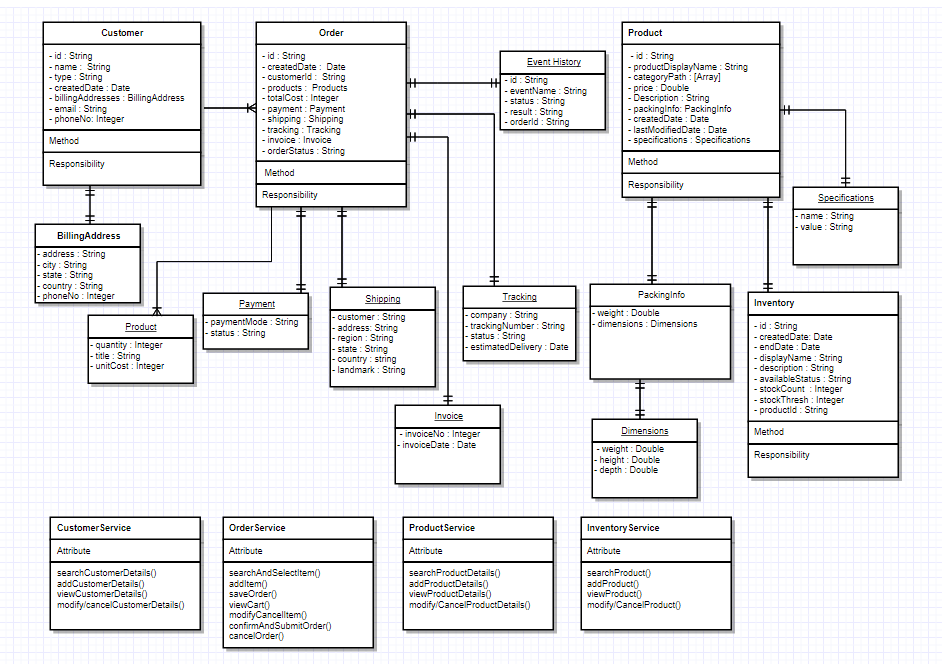
### Order Management



### Customer Management



## Class Diagram



## Integration View

This section specifies the technical interface between micro services. Please note that this section only considers the interfaces of these systems with the OMS

* + Inventory Service
  + Product Service
  + Order Service
  + Billing Service
  + Customer Service
  + Payment Service
  + Event Service
  + Mailing Service

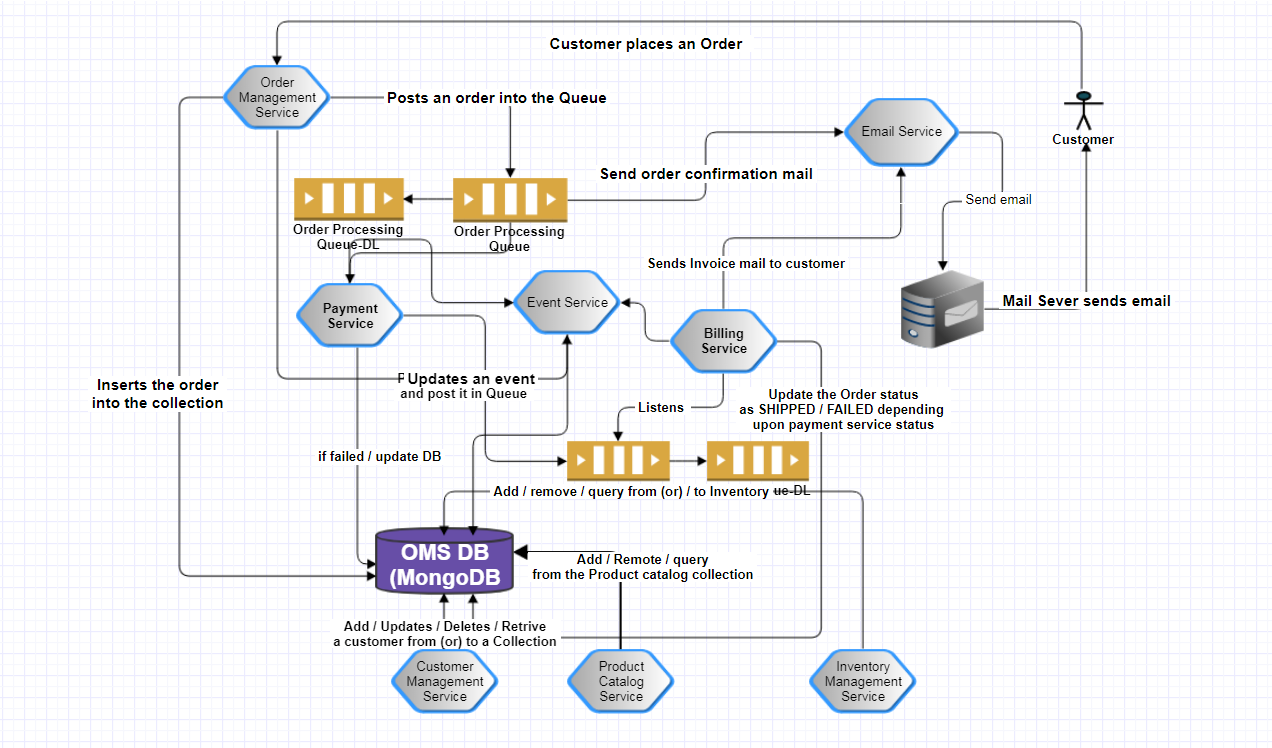


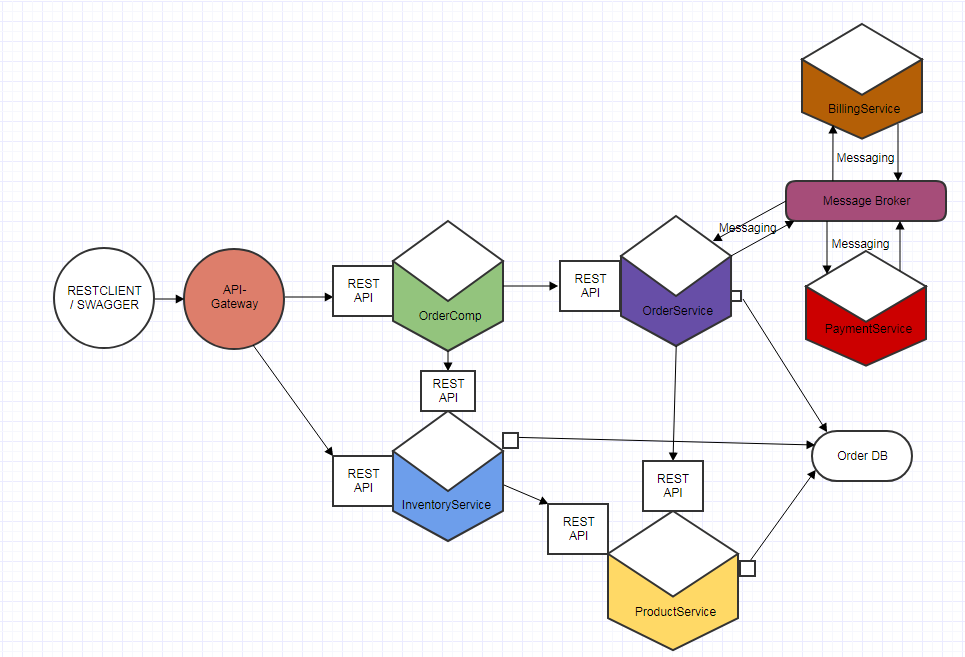
Figure 5: Even-driven flow views

**Micro-services**

The following table describes all the micro-services:

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Micro-service | Protocols | Micro-service Description |
|
| 1 | Inventory service | HTTP REST Call | Managing the product stock details the in Inventory. |
| 2 | Product service | HTTP REST Call | Managing the product details. |
| 3 | Order service | HTTP REST Call  and  Message Broker | Create/Update/Cancel/Search the order details with selected line items based on the availability of the products in the inventory. |
| 4 | Billing service | HTTP REST Call | Making the payment on the confirmed order and intimate the customer on the shipment details via an email. |
| 5 | Customer service | HTTP REST Call | Create/Update/Cancel/Search the customer details. |
| 6 | Payment Service | HTTP REST Call  and  Message Broker | OMS is using an opensource payment gate way (OpenSourceGateWay) |
| 7 | Event Service | HTTP REST Call | Create/Update event history table |
| 8 | Mailing Service | HTTP REST Call | Send an email over success or failure of order creation and Invoice Generation |

##### Service interaction view



## Security Strategy

### Authentication

Security in web application refers to Authentication. Authentication is the process of establishing a user.

## Data View

### Executive Summary

As required by OMS on-boarded to analyze the database design and practices that are being followed in the database area and come up with the documentation of the changes that are expected in the current database design and allied recommendations .Based on the outcome of the said discussions, further analysis was done to come up with the recommendations on the following areas. These include high level changes to the database design. Any database design change needs to be interpreted in conjunction with the accompanying Conceptual models, which have been produced wherever possible. The database design changes have only been conceived at a very high level without taking into considerations detailed level design considerations like application requirements, different types of accesses against data, storage requirements, specific database platform etc. As such these would have to be reviewed in details during the Design phase at the time of executing the project.

### Choice of target Database

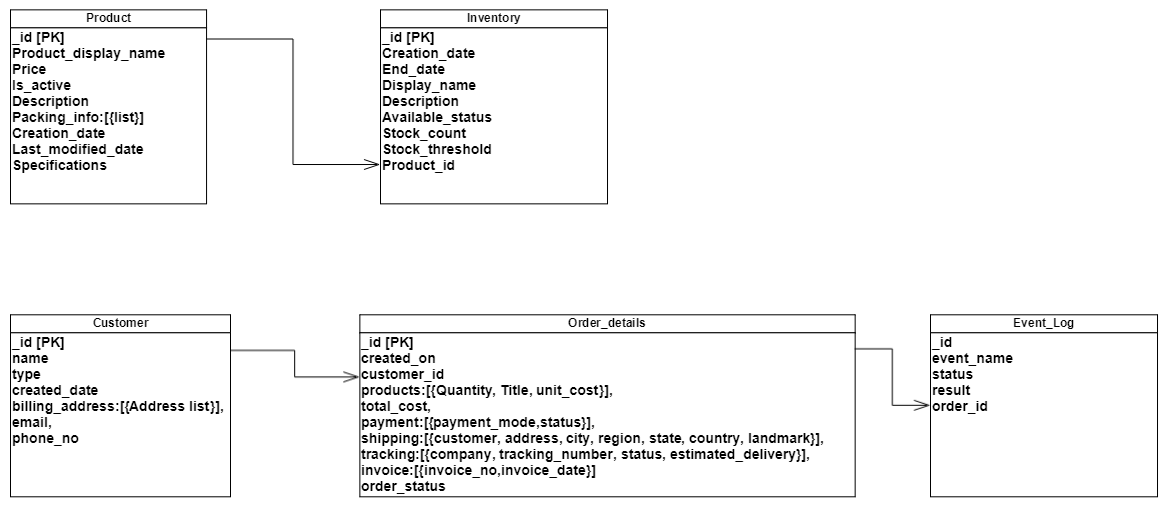
OMS is using MongoDB . MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.The document model maps to the objects in your application code, making data easy to work with MongoDB is free and open-source, published under the GNU Affero General Public License. Here MongoDB is installed outside of the PCF environment and prove the connection using Create a User-Provided Service Instance

(CUPS).

### Database and Shema Design

#### ER Diagram

#### Database Name : OMS



#### DB Details:

Mongo server details for OMS project.

Host: 10.231.173.31

Port:27017

User: OmsReadWrite

Password: OMS1234

#### Collection Details:

##### Product

{

\_id: String,

Product\_display\_name: String,

Category\_path:[Array],

Price:Double

Is\_active: Boolean,

Description: String,

Packing\_info:[

{ weight: Double,

Dimensions:[{weight: Double, height: Double, Depth: Double}]

}

],

Creation\_date: Date,

Last\_modified\_date: Date,

Specifications: [{name: String, Value: String}]

}

##### Inventory

{

\_id:ObjectId,

Creation\_date:Date,

End\_date:Date,

Display\_name:String,

Description:String,

Available\_status:String,

Stock\_count:Int,

Stock\_thresh:Int,

Product\_id:String

}

##### Order\_Details

{

\_id:ObjectId

created\_on: date,

customer\_id: string

products: {

quantity: int,

sku: String

title: String,

unit\_cost: int

},

total\_cost: int,

payment: {payment\_mode: string,

status: string

}

shipping: {

customer: string,

address: string,

city: string,

region: string,

state: string,

country: string,

landmark: string

},

tracking: {

company: string,

tracking\_number: string,

status: string,

estimated\_delivery: date

},

invoice: [

{invoice\_no:int,invoice\_date:date}

],

Order\_status: string

}

##### Customer details

{

"\_id" : ObjectId,

"name" :string,

"type" : string,

"created\_date" :date,

"billing\_addresses" : [{

"address" : string,

"city" : string,

"state" : string,

"country" : string,

“phone\_no”:int

}],

"email”:string,

"phone\_no" :int

}

##### Event History

{

“\_id”:ObjectId,

“event\_name”:String,

“status”:String,

“result”:String,

“order\_id”:ObjectId

}

# Deployment

CLI deployment - The pcf utility provides a command line interface to Pivotal Cloud Foundry for the purpose of deploying and testing tiles. Its primary reason for existence is to enable Ops Manager access from CI pipelines, but developers also find it convenient to use this CLI rather than the Ops manager GUI.

