Order Management System

**N-Views Architecture Representation Document**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Ashokkumar, Chandrasekaran |  |  |
| **Role** | Technical Analyst |  |  |
| **Signature** |  |  |  |
| **Date** | 20/9/2017 |  |  |

Table of Contents

[1.0 Introduction 3](#_Toc493783302)

[1.1 Purpose of this document 3](#_Toc493783303)

[1.2 Scope 3](#_Toc493783304)

[1.3 Structure of the document 4](#_Toc493783305)

[2.0 Project Overview 4](#_Toc493783306)

[2.1 Objectives 4](#_Toc493783307)

[3.0 Architecture Representation 4](#_Toc493783308)

[3.1 Architecture Overview 4](#_Toc493783309)

[3.2 Drivers 7](#_Toc493783310)

[3.2.1 Business Drivers and Goals 7](#_Toc493783311)

[3.2.2 Architectural Drivers and Goals 7](#_Toc493783312)

[3.2.3 Non-Functional Requirements 7](#_Toc493783313)

[3.3 Architecture Principles 8](#_Toc493783314)

[3.4 Key Architecture Decisions 8](#_Toc493783315)

[3.4.1 Technology Stack Rationalization 8](#_Toc493783316)

[4.0 Architecture Representational Views 9](#_Toc493783317)

[4.1 Functional View 9](#_Toc493783318)

[4.2 Layer View 10](#_Toc493783319)

[4.3 Micro-Service Architecture design patterns 11](#_Toc493783320)

[4.4 Activity Diagram 13](#_Toc493783321)

[4.4.1 Order Management System 13](#_Toc493783322)

[4.4.2 Product Management 14](#_Toc493783323)

[4.4.3 Inventory Management 15](#_Toc493783324)

[4.4.4 Order Management 16](#_Toc493783325)

[4.4.5 Customer Management 17](#_Toc493783326)

[4.5 Class Diagram 18](#_Toc493783327)

[4.6 Integration View 18](#_Toc493783328)

[4.7 Security Strategy 21](#_Toc493783329)

[4.7.1 Authentication and Authorization 21](#_Toc493783330)

[4.8 Data View 21](#_Toc493783331)

[4.8.1 Executive Summary 21](#_Toc493783332)

[4.8.2 Choice of target Database 21](#_Toc493783333)

[4.8.3 Database and Shema Design 22](#_Toc493783334)

[5.0 Deployment 27](#_Toc493783335)

# 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 management system
* Product catalog
* Order management
* Billing and Shipment
* Customer Management

# 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

DATA VIEW

LAYER VIEW

A

Y E R

V

I

E

W

SECURITY 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 improve efficiency, reduce cost, and provide increased visibility to data within OMS
2. Ease of integrating with third party system .

### Architectural Drivers and Goals

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

1. High Availability (HA)
2. Scalability
3. Reliability
4. 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 external properties across all environments.

#### oms-common-security

##### JWT

JSON Web Token (JWT) is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA**.oms-common-web

#### oms-common-web

##### Swagger

Spring Boot makes developing RESTful services ridiculously easy. And using Swagger makes documenting your RESTful services easy.

##### Logging

Spring Boot has no mandatory logging dependency, except for the Commons Logging API, of which there are many implementations to choose from. To use [Logback](http://logback.qos.ch/" \t "_top) you need to include it and jcl-over-slf4j (which implements the Commons Logging API) on the classpath. The simplest way to do that is through the starters which all depend on spring-boot-starter-logging. For a web application you only need spring-boot-starter-web since it depends transitively on the logging starter.

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.

logging.level.org.springframework.web=ERROR

logging.com.oms.Customer=DEBUG

You can also set the location of a file to log to (in addition to the console) using "logging.file".

##### Exception Management

Spring Boot helps substantially, as it removes a lot of boilerplate code and enables auto-configuration of various components.

## 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
* Slf4j framework shall be used for fulfilling the application logging needs.
* Common application server independent security framework (Spring Security and JWT) shall be used for Authentication and Authorization 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 management system
* Product catalog
* Order management
* Billing and Shipment
* Customer Management

## Layer View

The layering architecture consists of the following layers

* Clients
* Presentation Layer
* 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

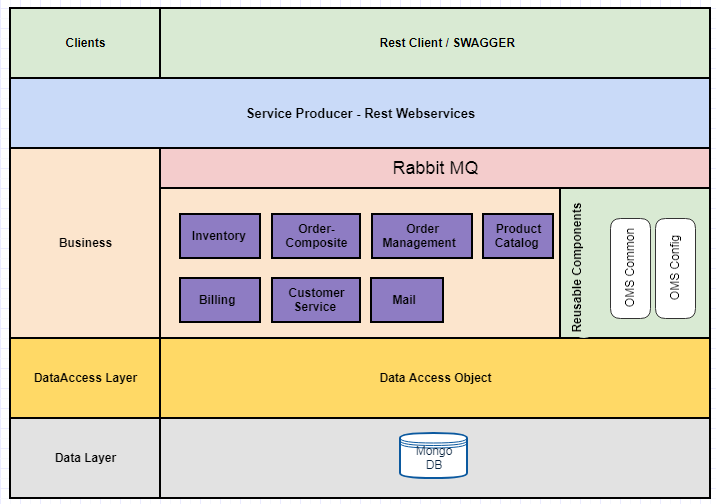


Figure 2: Layered views

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 various modes through which users can access the Order Management Systemapplication. The access to the online application can be done using any latest standard web browsers.

* **Business Layer**

The business service layer comprises of components that realizes a specific business function or services of the Order Management Systemapplication. 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](http://microservices.io/patterns/apigateway.html)  - The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

Service discovery

* [Client-side Discovery](http://microservices.io/patterns/client-side-discovery.html)  - When making a request to a service, the client obtains the location of a service instance by querying a [Service Registry](http://microservices.io/patterns/service-registry.html), which knows the locations of all service instances.
* [Server-side Discovery](http://microservices.io/patterns/server-side-discovery.html) - 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](http://microservices.io/patterns/service-registry.html), which might be built into the router, and forwards the request to an available service instance.

Communication style

* Messaging - use asynchronous messaging for inter-service communication. Services communicating by exchanging messages over messaging channels.

Deployment patterns

* [Single Service per Host](http://microservices.io/patterns/deployment/single-service-per-host.html) - Deploy each single service instance on its own host

Cross cutting concerns

* Externalized configuration - Externalize all application configuration including the database credentials and network location. On startup, a service reads the configuration from an external source, e.g. OS environment variables, etc.

Data management

* Shared database - Use a (single) database that is shared by multiple services. Each service freely accesses data owned by other services using local ACID transactions.

Security

* [Access Token](http://microservices.io/patterns/security/access-token.html) - The [API Gateway](http://microservices.io/patterns/apigateway.html) authenticates the request and passes an access token (e.g. [JSON Web Token](https://jwt.io/)) that securely identifies the requestor in each request to the services. A service can include the access token in requests it makes to other services.

Reliability

* [Circuit Breaker](http://microservices.io/patterns/reliability/circuit-breaker.html) - 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.

Observability patterns

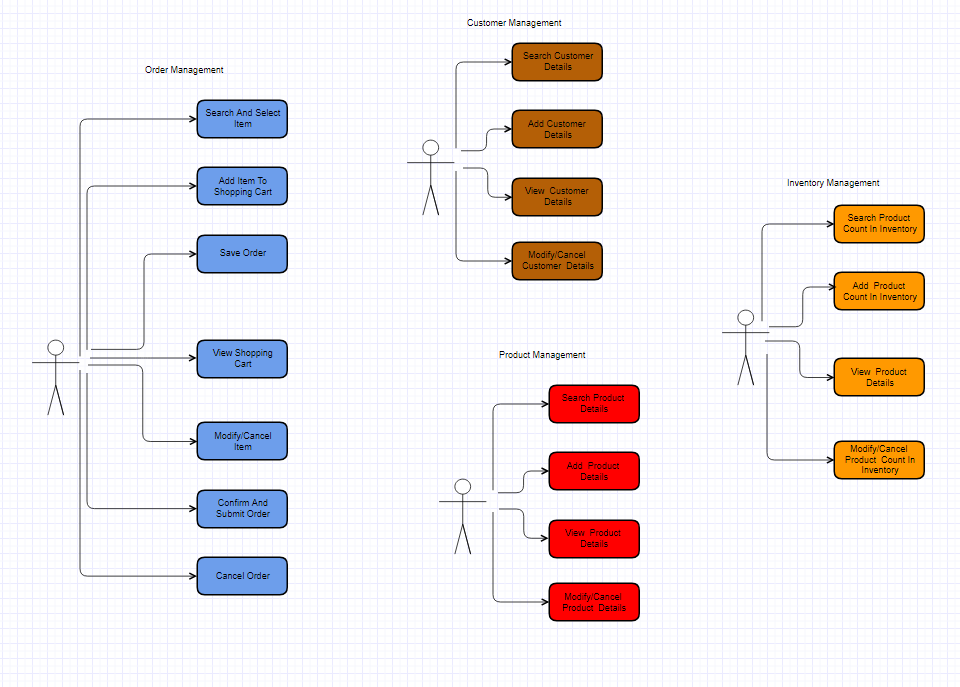
* [Exception tracking](http://microservices.io/patterns/observability/exception-tracking.html) - The application consists of multiple services and service instances that are running on multiple machines. Errors sometimes occur when handling requests. When an error occurs, a service instance throws an exception, which contains an error message and a stack trace.
* [Health check API](http://microservices.io/patterns/observability/health-check-api.html) - Sometimes a service instance can be incapable of handling requests yet still be running. For example, it might have ran out of database connections. When this occurs, the monitoring system should generate a alert. Also, the load balancer or [service registry](http://microservices.io/patterns/service-registry.html) should not route requests to the failed service instance.

Testing patterns

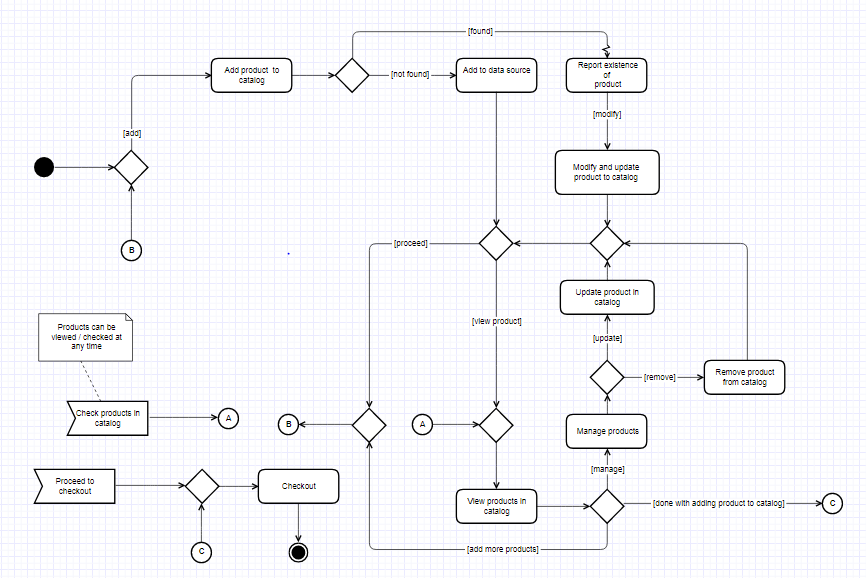
* [Service Component Test](http://microservices.io/patterns/testing/service-component-test.html) - A test suite that tests a service in isolation using test doubles for any services that it invokes.
* [Service Integration Contract Test](http://microservices.io/patterns/testing/service-integration-contract-test.html) - 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

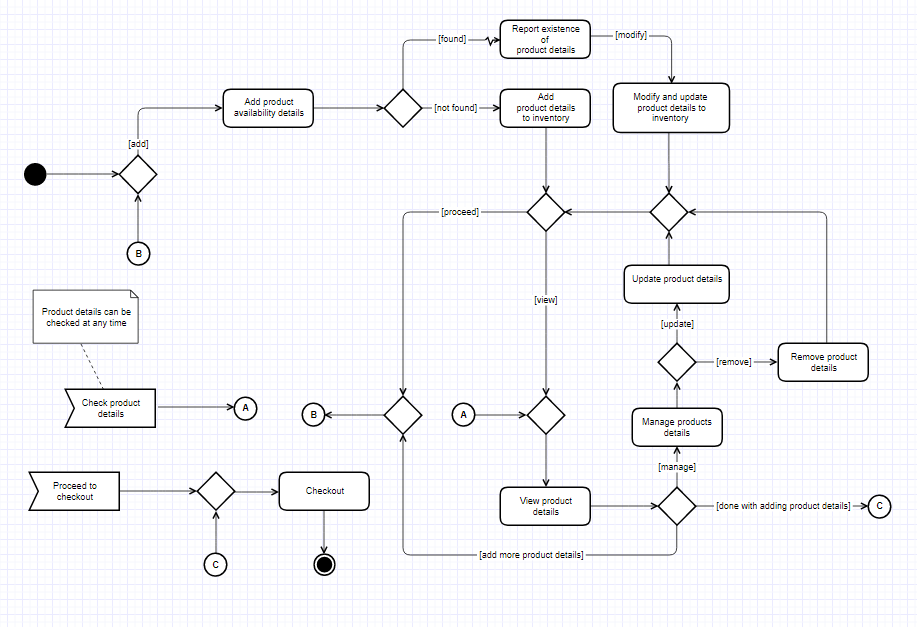
### Order Management System



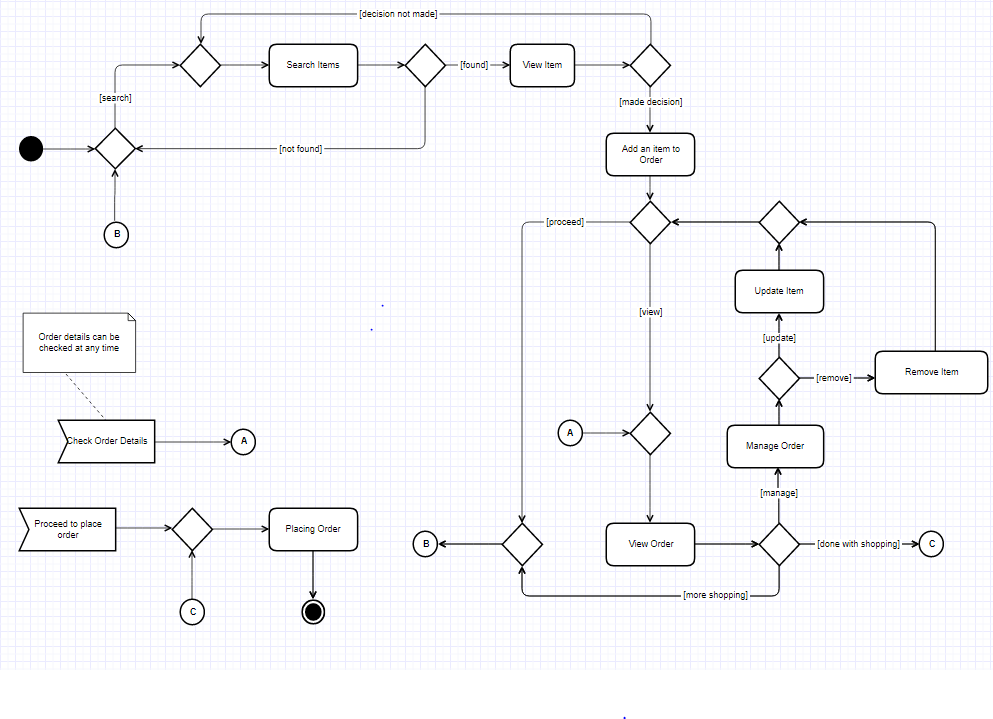
### Product Management



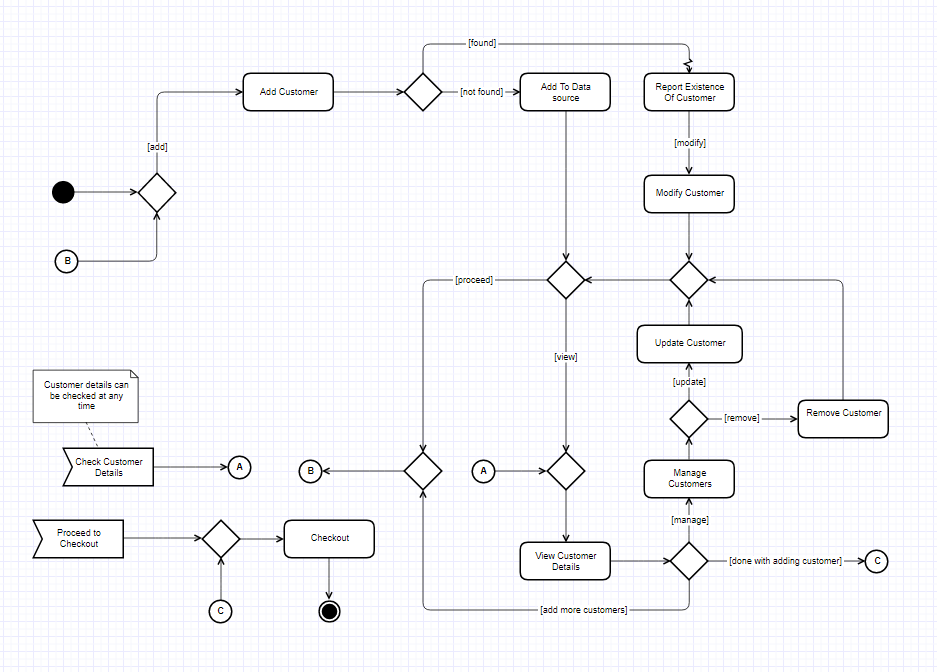
### Inventory Management



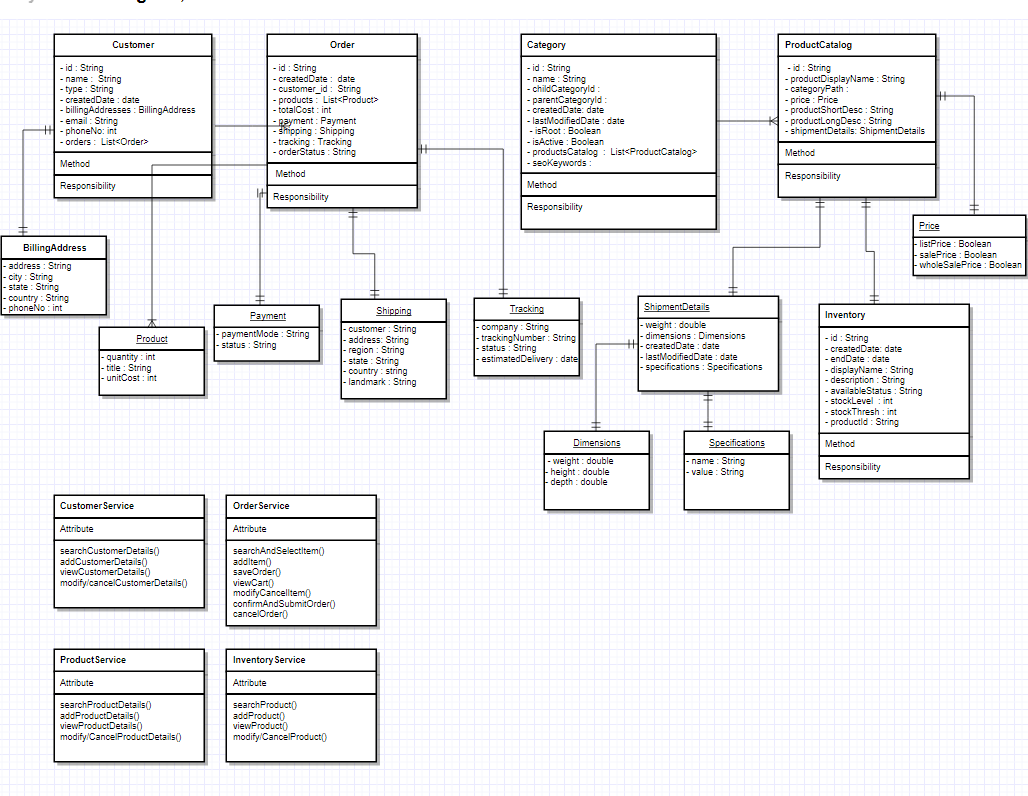
### Order Management



### 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 management system
  + Product catalog
  + Order management
  + Billing and Shippment
  + Customer Management

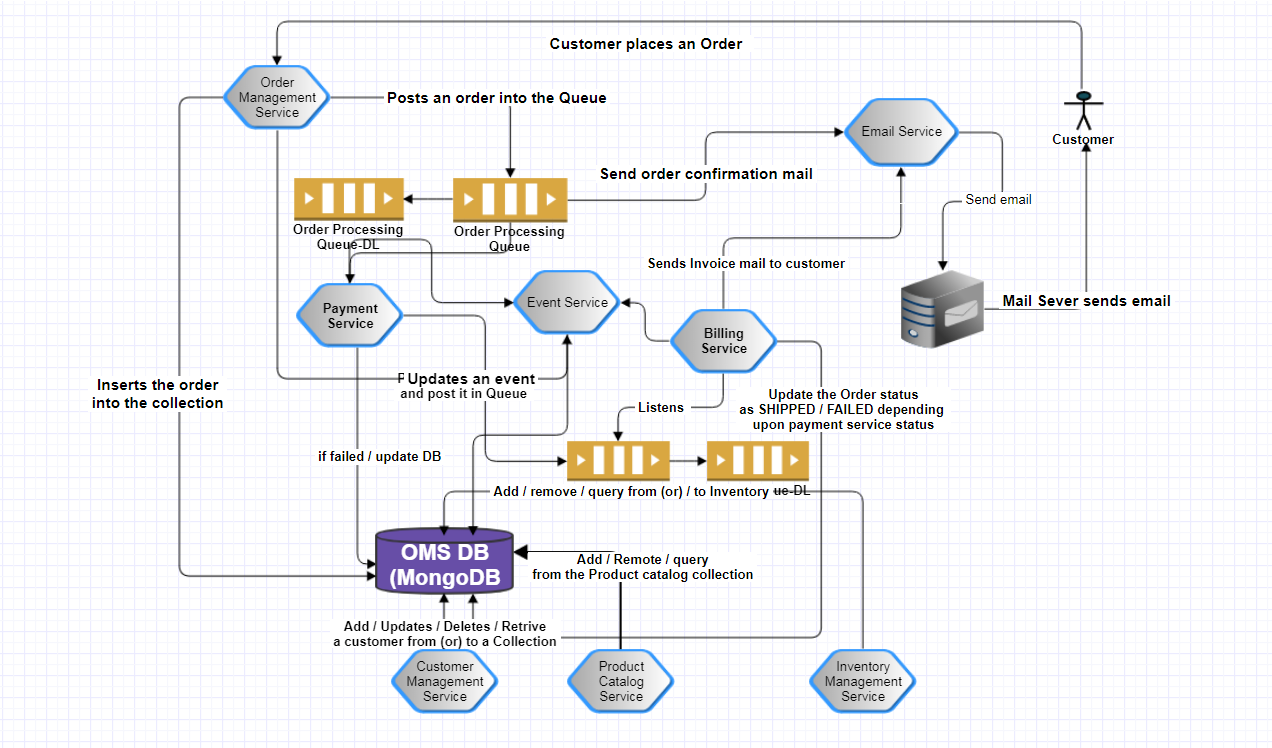


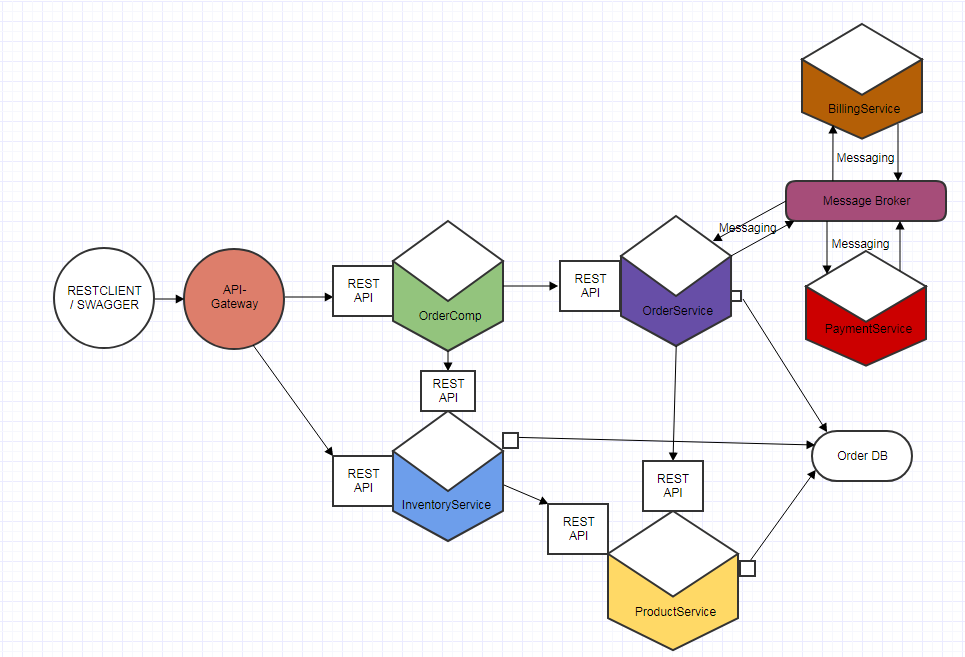
Figure 5: Even-driven flow views

**Micro-services**

The following table describes all the micro-services:

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Micro-service | Protocols | Micro-service Description |
|
| 1 | Inventory management system | HTTP REST Call | Managing the product stock details the in Inventory. |
| 2 | Product catalog | HTTP REST Call | Managing the product details. |
| 3 | Order management | HTTP REST Call | Create/Update/Cancel/Search the order details with selected line items based on the availability of the products in the inventory. |
| 4 | Billing and Shippment | HTTP REST Call | Making the payment on the confirmed order and intimate the customer on the shipment details via an email. |
| 5 | Customer Management | HTTP REST Call | Create/Update/Cancel/Search the customer details. |

##### Service interaction view



## Security Strategy

### Authentication and Authorization

Security in web application refers to Authentication and Authorization. Authentication is the process of establishing a user and authorization pertains to what a user may do in application.

Following figure explains the solution for authentication and authorization in the Order Management System application.

## 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

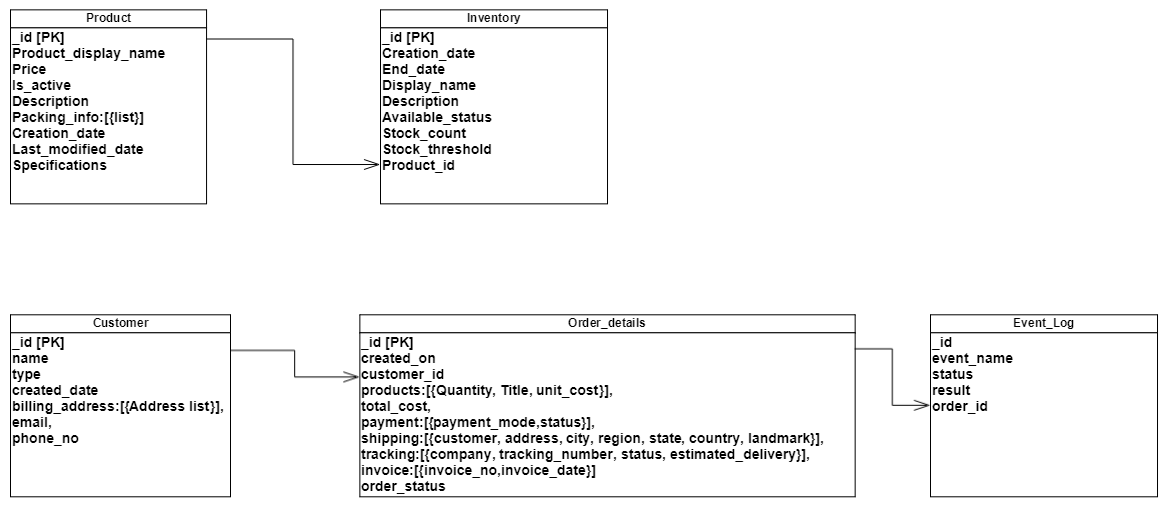
OMS is using MongoDB . MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.The document model maps to the objects in your application code, making data easy to work with MongoDB is free and open-source, published 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



#### Collection Details:

##### Category

{

\_id:String,

Name:String,

Child\_categorie\_id:[Array],

Parent\_categorie\_id:[Array],

Image\_path:String,

Creation\_date:Date,

Last\_modified\_date:Date,

Is\_root:Boolean,

Is\_active:Boolean,

Seo\_Keywords:[Array],

}

##### 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.

