

High Level Design Document

For Project

MOIC System Development

**Ver. 0 7**

Description: cid:image006.jpg@01CCA3F2.49B084E0

Copyright © CIGNEX Datamatics

This document and content therein are for the sole use of the intended recipient(s) and may contain confidential and privileged information. Any unauthorized review, use, disclosure, dissemination, forwarding, printing or copying of this document or any action taken in reliance on this document is strictly prohibited and may be unlawful.

Visit us at [***www.CIGNEX.com***](http://www.CIGNEX.com)

Revision & Approval Chart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ver # | Date | Description | Author(s) | Reviewed/approved by/date |
| 0.1 | 07-12-20 | Draft version | Technical Team | Vatsal Thakkar |
| 0.2 | 10-12-20 | Updated as per the review comments | Technical Team | Jogen Gondalia |
| 0.3 | 12-12-20 | Updated as per review comments | Vatsal Thakkar |  |
| 0.4 | 15-12-20 | Updated as per internal review comments | Vatsal Thakkar |  |
| 0.5 | 21-12-20 | Updated based on internal discussion on download module approach | Technical Team |  |
| 0.6 | 10-01-21 | Updated for details on Integration with SAP for customer master data | Vatsal Thakkar |  |
| 0.7 | 16-10-21 | Updated for external integration details | Vatsal Thakkar |  |

Table of Contents

[1. Introduction 4](#_Toc62060000)

[1.1. Overview 4](#_Toc62060001)

[1.2. Definitions, Abbreviations, Acronyms 5](#_Toc62060002)

[1.3. Intended Audience 5](#_Toc62060003)

[1.4. References 6](#_Toc62060004)

[2. System Design Overview 7](#_Toc62060005)

[2.1. Design Criteria 7](#_Toc62060006)

[2.2. MOIC application and Components 7](#_Toc62060007)

[2.3. Common Utilities 8](#_Toc62060008)

[2.4. System Internal 8](#_Toc62060009)

[2.5. MOIC – Users Characteristics 8](#_Toc62060010)

[2.6. Design & Implementation Constraints 9](#_Toc62060011)

[2.7. Assumptions & Dependencies 9](#_Toc62060012)

[3. High Level System Design 10](#_Toc62060013)

[3.1. Design Concept Selection 10](#_Toc62060014)

[3.1.1. Orders Data from Joor (Inbound) 10](#_Toc62060015)

[3.1.2. Final Orders Data from MOIC to SAP (Outbound) 11](#_Toc62060016)

[3.1.3. Customer Data from SAP / S4 (Inbound) 12](#_Toc62060017)

[3.1.4. Customer Data from MOIC to Joor (Outbound) 13](#_Toc62060018)

[3.1.5. Product Data from PMDB (Inbound) 14](#_Toc62060019)

[3.2. Technology Stack 15](#_Toc62060020)

[3.3. Architecture Diagram 15](#_Toc62060021)

[3.3.1. Layer 1: Data Access / Integration 16](#_Toc62060022)

[3.3.2. Layer 2: Web 16](#_Toc62060023)

[3.3.3. Layer 3: Miscellaneous 16](#_Toc62060024)

[3.3.4. Layer 4: Core Container 17](#_Toc62060025)

[3.3.5. Layer 5: Test 17](#_Toc62060026)

[3.4. Design for common utilities components 17](#_Toc62060027)

[3.5. Caching 17](#_Toc62060028)

[3.6. Schedulers & Jobs 17](#_Toc62060029)

[3.7. Integrations Failure Handling 18](#_Toc62060030)

[4. System Component Design 19](#_Toc62060031)

[4.1. Communication with external applications through ESB (Inbound) 19](#_Toc62060032)

[4.1.1. Inbound Project Structure 19](#_Toc62060033)

[4.1.2. Package Details 19](#_Toc62060034)

[4.2. Communication with external applications through ESB (Outbound) 20](#_Toc62060035)

[4.2.1. Outbound Project Structure 20](#_Toc62060036)

[4.2.2. Package Details 21](#_Toc62060037)

[4.3. Download & Upload data through Excel / CSV files 21](#_Toc62060038)

[4.4. Scheduler job for CSV files process 22](#_Toc62060039)

[5. User Interface - MOIC 23](#_Toc62060040)

[5.1. Angular app component structure 23](#_Toc62060041)

[5.1.1. Common / shared components: 23](#_Toc62060042)

[5.1.2. Page level components: 23](#_Toc62060043)

[5.2. Non Functional Screens – (Developed in Angular) 24](#_Toc62060044)

[6. Error Handling 27](#_Toc62060045)

[6.1.1. Naming Conventions 27](#_Toc62060046)

[6.1.2. Deployment Design 28](#_Toc62060047)

[6.1.3. Build Tool & Code Repository 28](#_Toc62060048)

[6.1.4. Coding standard and compliance 28](#_Toc62060049)

[6.1.5. Backup 28](#_Toc62060050)

# Introduction

## Overview

This design document will represent the high level design for MOIC (Master Order Intake & Consolidation) system targeted to be developed for Tapestry users. This document will be used as a basis for development phase of proposed solution. This document will remain a living document during implementation to further define and specify component details such as table details, class, and parameter types etc.

## Definitions, Abbreviations, Acronyms

| Term/Acronym | Definition |
| --- | --- |
| OOB or OOTB | Out of the Box |
| DAO | Data Acces Object (Design pattern) |
| POJO | Plain Old Java Object |
| URL | Universal Resource Locator |
| CMS | Content Mangement System |
| SOAP | Simple Object Access Protocol |
| KPI | Key performance Indicator |
| UI | User Interface |
| PMDB | Product Master Database |
| ECV | ecVision (application) |
| M | Material |
| ESB | Enterprise Service Bus |
| S4 | SAP system used for account master data management |
| MS | Material Supplier |
| Joor | Order intake application used by end customers |
| SP | Service Provider |
| PM | Product Master |

## Intended Audience

CIGNEX team (MOIC Development), ESB team (supporting middleware layer interfaces with external applications), Joor team, S4/SAP team, PMDB team & Tapestry team members.

## References

* Business Requirements Doc Version 1.3
* Software Requirement Specification\_MOIC\_v 1.1.docx
* Functional Calls and Discussions between CIGNEX and Tapestry Team
* CustomerMaster\_SAP\_MOIC\_Data\_Mapping Spec\_01072021.xlsx

# System Design Overview

## Design Criteria

* While designing each service component, care will be been taken to address the following:
  + Efficient Performance
  + Service Components are configurable (where applicable)
  + Proper implementation of loggers to troubleshoot issue
  + Component Wiring through configuration
  + Streamlined build and deployment process

## MOIC application and Components

This section describes the system components, common utilities or components that are projected towards MOIC application development & deployment.

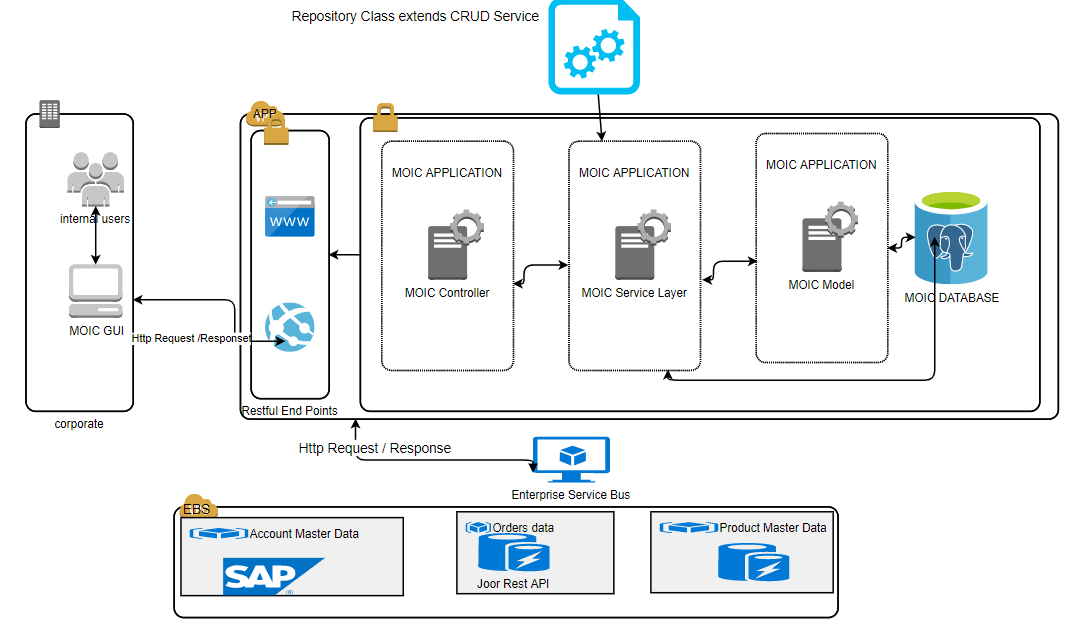


Fig 1.0 System Component Design

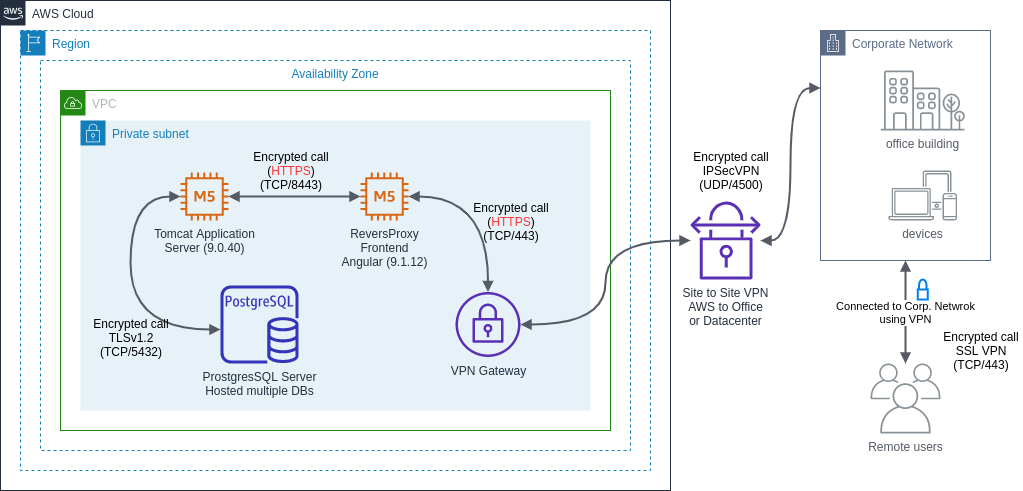


Fig 2.0 Application Architecture Diagram

## Common Utilities

* MOIC application - Routing Module
* MOIC application - Component
* MOIC application - Component Template
* MOIC application - Module
* MOIC application – Database

## System Internal

* MOIC GUI will be created which will interact with MOIC Restful API.
* MOIC application Controller will have DAO layer services.
* Service layer will interact with MOIC Database.
* Any modification to service layer mandates the service jar to be generated again and placed in global class path.
* *S*erver restart is required if a new service-jar is placed in global class path.

## MOIC – Users Characteristics

* There will be no login authentication for MOIC system.
* If required, the restriction will be applied at the port level permissions.
* MOIC is not a public facing application and will only be used by internal Tapestry team.

## Design & Implementation Constraints

* MOIC application will communicate with all external systems through middleware layer. There will be no direct interface from MOIC to any external system.
* All services developed will be RESTful web services.
* For excel uploads into MOIC, file format and data type validations will be done. Data validation will not be done for data authenticity.
* There will be no login authentication for users to get access to MOIC application.

## Assumptions & Dependencies

The assumptions are as under:

* Services that will be developed in Spring Framework will be RESTful web services.
* Majority of integration work with respect to extract, transformation from SAP, PMDB, and Joor will be done by the Tapestry middleware team via ESB or Informatica.
  + Product Master Data and Account Master Data are not required in real time. The communication of data between MOIC and PMDB, MOIC and SAP will happen through batch processing.
* The excel file upload and download will be based on the listed fields under Reporting section. Any deviation from this in the format will result in the transaction failure and error display.
* The “default” values for Fashion Season and Channel dropdown if other than what is mentioned in this document are required, Tapestry team will let the development team know earlier in the development cycle.
* No pre-load of Joor Order data is required.
* Pre-load of Account Master Data from SAP will be concluded prior to the development cycle start.

# High Level System Design

## Design Concept Selection

Tapestry intends to build a solution that can provide real time reconciliation between demand (Orders) and production / targets. MOIC web application will be communicating with external systems through the middleware layer (ESB) for data (Inbound & Outbound).

* Order Data from Joor (Inbound)
* Final Order Data from MOIC to SAP (Outbound)
* Customer Data from SAP / S4 (Inbound)
* Customer Data from MOIC to Joor (Outbound)
* Product Data from PMDB (Inbound)

The business users will be able to view the consolidated data through MOIC application screens and manipulate / change as required. The changes will be updated in MOIC database and the relevant data will be communicated back to external systems (Joor / SAP) through middleware layer.

MOIC application will be developed using Spring Boot which is an open source JAVA based framework for the backend and Angular JS for the front end. MOIC will use Postgres as the relational database engine.

MOIC will in take the Orders data from Joor, Account Master Data from SAP and Product Master Data from PMDB. For this, RESTful web services will be developed as part of MOIC application. These services will be invoked by the middleware layer and the relevant data will be communicated to external applications (Outbound) and MOIC database (Inbound). The middleware layer already has interfaces for talking with the external applications and the same will be used or enhanced as required.

Updates to the consolidated data will be done through MOIC screens and also through Excel imports in case of bulk updates.

### Orders Data from Joor (Inbound)

Joor is a 3rd party SAAS based solution – external application which will be used by Tapestry customers (wholesale) for placing their orders.

Joor has its own APIs (legacy set, new set). JOOR provides a sandbox environment that can be used to test our application against the JOOR API. The sandbox environment replicates all features and functions of their production environment.

The JOOR AIS APIs employ two domains, one for developing and testing the integration and one for our production account. The development domain is [https://apisandbox.jooraccess.com](https://apisandbox.jooraccess.com/) while the production domain is [https://api.jooraccess.com](https://api.jooraccess.com/). Both domains require secure HTTP connections.

MOIC application will not have a direct communication / connection to Joor. As a middleware, ESB will be used. ESB will invoke Joor APIs and send the required orders data to MOIC application in a JSON file. MOIC will have a job which will pick up the JSON file and process the orders to bring them into MOIC.

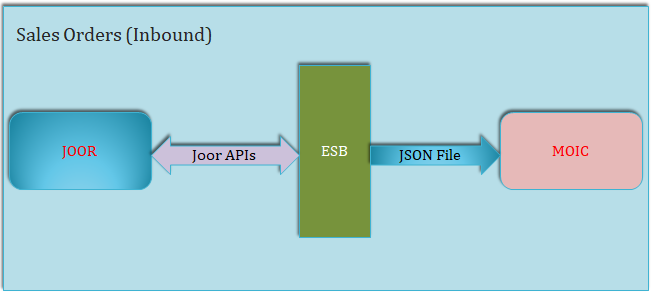


Fig 3.0 Sales Order Inbound

* Source Application: Joor
* Middleware: ESB
* Destination Application: MOIC
* Frequency: Twice a day (to be finalized)
* Load: Full (to be finalized based on order status field)

### Final Orders Data from MOIC to SAP (Outbound)

After the orders get into MOIC from Joor, the orders will be worked upon by MOIC users through MOQ and SKU Change screens. The users will be able to manipulate some necessary data and when they are sure, they will send the finalized orders to SAP for further processing.

As the users finalize the orders through MOIC screen, the final order rows will get inserted into “FinalBuyTable” within MOIC schema.

From the screen “S4 Upload”, the user will confirm which final orders (based on the Fashion Season and Channel) should be sent to SAP. When the user clicks on “Upload to S4” button on the screen, they will be asked for their confirmation. Upon confirming, the relevant finalized orders will be moved from “FinalBuyOrders” in main MOIC schema to a “FinalBuyOrders\_staging” table in the outbound schema.

MOIC will also archive the outgoing orders into SAP for a period of 2 years. MOIC users can download archived orders using the “Download archive” button on the Overview screen. MOIC will have a job to clean up archived data older than 2 years.

ETL will have a job to pick up these finalized orders from outbound schema and sent it to SAP.

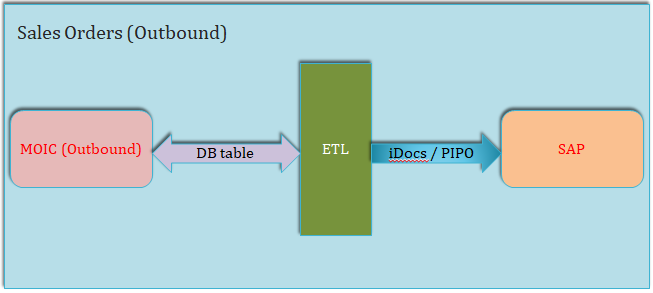


Fig 4.0 Sales Orders Outbound

* Source Application: MOIC
* Middleware: ETL
* Destination Application: SAP
* Frequency: Twice a day (to be finalized)
* Load: Only finalized orders

### Customer Data from SAP / S4 (Inbound)

Customer data (wholesale customers only) will be brought into MOIC from SAP through this process. MOIC users will update / manipulate customer data through the Config screen.

MOIC will have a customer master table within main schema and will have a customer master staging table in the inbound schema. ETL will pick up customer data from SAP (full load) and dump it into the customer master staging table. MOIC will have a job which will pick up the data from staging table and process it into main MOIC customer master table.

As the users will be updating customer data (editable fields) through Config screen, the job will ensure that the updated customer data is not overwritten when bringing from staging into main. At the same time if any of the key fields will have a different value in the incoming data from SAP, we will create a new customer record when bringing the data from staging into main. Also the “isActive” flag will be updated for all customer records in the main schema as per what comes in from SAP every time.

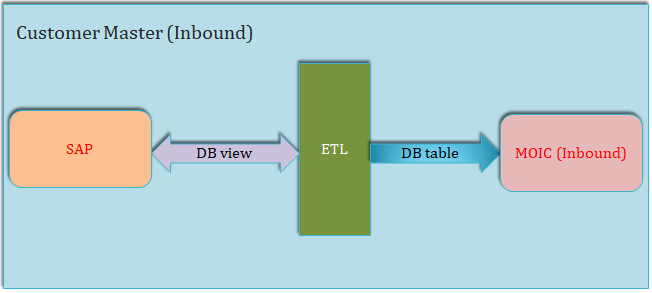


Fig 5.0 Customer Master Inbound

* Source Application: SAP
* Middleware: ETL
* Destination Application: MOIC
* Frequency: Once a day
* Load: Full

### Customer Data from MOIC to Joor (Outbound)

The customer data from SAP comes into MOIC and the MOIC users update / edit the data (only for editable fields) including “Channel”, “Segment”, etc.

The updated customer data has to be sent to Joor application. Joor is a 3rd party SAAS based solution – external application which will be used by Tapestry customers (wholesale) for placing their orders.

Joor has its own APIs (legacy set, new set). JOOR provides a sandbox environment that can be used to test our application against the JOOR API. The sandbox environment replicates all features and functions of their production environment.

MOIC application will not have a direct communication / connection to Joor. As a middleware, ESB will be used. ESB will invoke Joor APIs to send the customer data over to Joor. MOIC will produce JSON file(s) with required customer data in the format that ESB can take and make the Joor API call to send the data over.

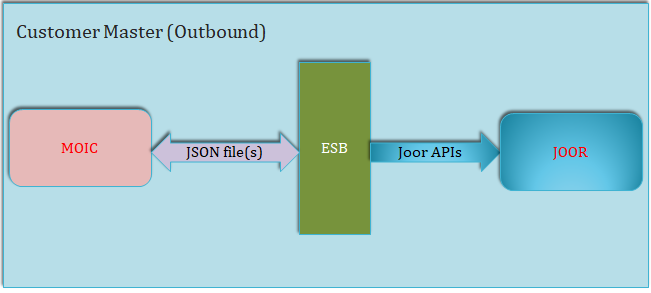


Fig 6.0 Customer Master Outbound

* Source Application: MOIC
* Middleware: ESB
* Destination Application: Joor
* Frequency: Once a day
* Load: Full

### Product Data from PMDB (Inbound)

The product master data which resides in PMDB will be brought into MOIC. Since both; PMDB and MOIC are internal facing applications, the communication between the two will be comparatively simple. The product data does not need to be brought into MOIC so batch processing will be the method.

PMDB will create a view (with MOIC needed product / SKU attributes). ETL will pick up the data from PMDB view and dump the data into MOIC staging table in inbound schema. MOIC will have a job which will pick up the data from staging and process it into main MOIC schema.

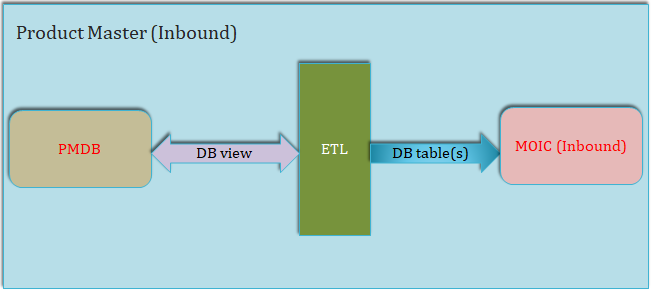


Fig 7.0 Product Master Inbound

* Source Application: PMDB
* Middleware: ETL
* Destination Application: MOIC
* Frequency: Once a day
* Load: Full (filtered by Brand / Department)

## Technology Stack

* Spring Boot 2.2.7
* Spring Framework 5.2.5
* Hibernate 5.4.15 / JPA 5.4.15
* Java 1.8
* Build tool: Gradle 6.3
* Database: PostgresSQL version xx
* Front End Framework: Angular version 9.0

## Architecture Diagram

Spring Framework Architecture

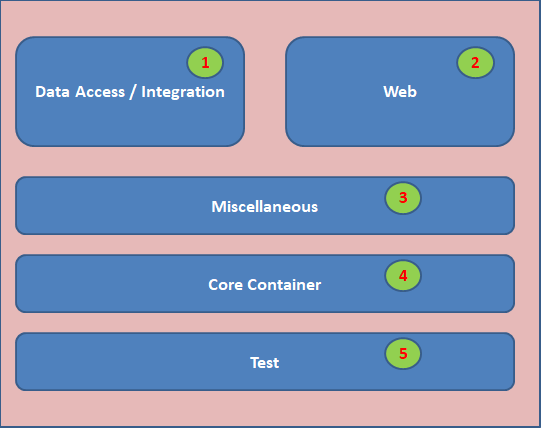


Fig 8.0 Development Framework

### Layer 1: Data Access / Integration

* Spring Data JPA – The Spring Data JPA module implements the Spring Data Commons repository abstraction to ease the repository implementations even more, making a manual implementation of a repository obsolete in most casesORM – Provides the integration layers for popular object-relational mapping APIs
* OXM – Provides an abstraction layer that supports XML mapping implementations
* Transactions – Provides support for all POJOs

### Layer 2: Web

* Angular 9 - MOIC Application will use Spring Boots for the backend and Angular 9 for the frontend. More detail is in section 5.

### Layer 3: Miscellaneous

* AOP – Provides an aspect oriented programming implementation which helps writing decoupled code.
* Aspects – Provides integration with AspectJ.
* Instrumentation – Provides class instrumentation support and classloader implementations to be used in certain application servers.

### Layer 4: Core Container

* Beans – Provides the fundamental part of the framework.
* Core – Provides the fundamental part of the framework.
* Context – Means to access objects in a framework style manner.

### Layer 5: Test

* Testing – Supports the testing of Spring components with JUnit or TestNG.

## Design for common utilities components

* All the common methods will be moved to MoicUtil class.
* In this project JPA is used, which will be used for database interactions
* Any modification to the application code mandates the application jar to be re-generated.
* This Spring boot application jar will contains an embedded tomcat server which needs to be stopped and the application jar should be replaced with the new one and needs to be started.

## Caching

Not applicable

## Schedulers & Jobs

* Scheduled Job 1 - DeleteArchivedFinalBuy
  + We will write a job and schedule it for cleaning up the archived final buy data. As per the requirements, the data is to be kept for a maximum of 2 years. This job will run once a month and delete archived data which is older than 2 years.
* Scheduled Job 2 – GetCustomerMasterData
  + Details: TBD
* Scheduled Job 3 – GetProductMasterData
  + Details: TBD
* Scheduled Job 4 – PostFinalOrdersData
  + Details: TBD
* Scheduled Job 5 – PostCustomerMasterData
  + Details: TBD

## Integrations Failure Handling

* MOIC may talk with other applications through ESB using webservice as a part of integration. It is very likely that one of the webservice may be down and because of this system may not perform as required.
* In order to quickly identify any such integration issues, mail will be sent to system admins notifying about the unavailability of the service.
* As of now following are the web service integrations planned
  + Integration with ESB
* Whenever the above webservices goes down a mail will be triggered to notify the sytem admin.

# System Component Design

## Communication with external applications through ESB (Inbound)

### Inbound Project Structure

**/moic-inbound**

**/moic-inbound/src/main/java**

**com.coach.moic.ib**

**com.coach.moic.ib.api.controller**

**com.coach.moic.ib.api.service**

**com.coach.moic.ib.common.constants**

**com.coach.moic.ib.common.dto**

**com.coach.moic.ib.common.vo**

**com.coach.moic.ib.db**

**com.coach.moic.ib.db.repository**

**com.coach.moic.ib.model**

**com.coach.moic.ib.model.entity**

**com.coach.moic.ib.util**

**/moic-inbound/src/main/resources**

**/moic-inbound/src/main/resources/META-INF**

**/moic-inbound/src/main/resources/application.yml**

**/moic-inbound/src/test/java**

**/moic-inbound/bin**

**/moic-inbound/doc**

**/moic-inbound/gradle**

**/moic-inbound/src**

**/moic-inbound/build.gradle**

**/moic-inbound/gradlew**

**/moic-inbound/settings.gradle**

* All the java classes of MOIC application will reside in src/main/java within appropriate packages.
* All property files, YML files and any other files used for configurations will reside in src/main/resources.
* MOIC-API project will consist of RestFul web services for Inbound and Oubound communication.
* We will have controller interfaces and controller classes which will deal with all the Inbound and Outbound data manipulation and data validations.
* All the business logic and data persisting logic will be handled through service interfaces and service classes.
* All the common utility methods will be kept under MOICUtil class and ensure re-usability.
* build.gradle file will consist of all the project dependencies.

### Package Details

|  |  |
| --- | --- |
| com.coach.moic.ib | Main spring boot Service |
| com.coach.moic.ib.api.controller | Controls the request URL, GET/POST method |
| com.coach.moic.ib.api.service | Service (Main Business Logic) |
| com.coach.moic.ib.common.constants | Over All Constants |
| com.coach.moic.ib.common.dto | Request , Response Classes and Object Classes |
| com.coach.moic.ib.common.vo | VO class (if required) |
| com .coach.moic.ib.db | Nothing in there right now. Most probably for the external data base configuration |
| com.coach.moic.ib.db.repository | JPA Respository |
| com.coach.moic.ib.model | Nothing |
| com.coach.moic.ib.model.entity | Entity Classes which is mapped with the db |
| com.coach.moic.ib.util | Generic utility methods used in overall codes. Like exception handler , debug Classes , Format Handler |
| application.yml | Configurations |
| doc | Generated Documents |
| build.gradle | Project Dependencies |

## Communication with external applications through ESB (Outbound)

### Outbound Project Structure

**/moic-outbound**

**/moic-outbound/src/main/java**

**com.coach.moic.ib**

**com.coach.moic.ib.api.controller**

**com.coach.moic.ib.api.service**

**com.coach.moic.ib.common.constants**

**com.coach.moic.ib.common.dto**

**com.coach.moic.ib.common.vo**

**com.coach.moic.ib.db**

**com.coach.moic.ib.db.repository**

**com.coach.moic.ib.model**

**com.coach.moic.ib.model.entity**

**com.coach.moic.ib.util**

**/moic-outbound/src/main/resources**

**/moic-outbound/src/main/resources/META-INF**

**/moic-outbound/src/main/resources/application.yml**

**/moic-outbound/src/test/java**

**/moic-outbound/bin**

**/moic-outbound/doc**

**/moic-outbound/gradle**

**/moic-outbound/src**

**/moic-outbound/build.gradle**

**/moic-outbound/gradlew**

**/moic-outbound/settings.gradle**

### Package Details

The package detail for Outbound is same as Inbound. Please refer section 4.1.2

## Download & Upload data through Excel / CSV files

One way of data communication between MOIC and external application is through RestFul webservices as detailed in earlier section.

Upload and download of data through Excel / CSV files is the second way in which data goes out and comes into MOIC system.

There will be a specific directory into which all the uploaded files will be stored. REST APIs for uploading and downloading files will be written which will be exposed to MOIC front end layer. The below diagram depicts the component level details towards this module.

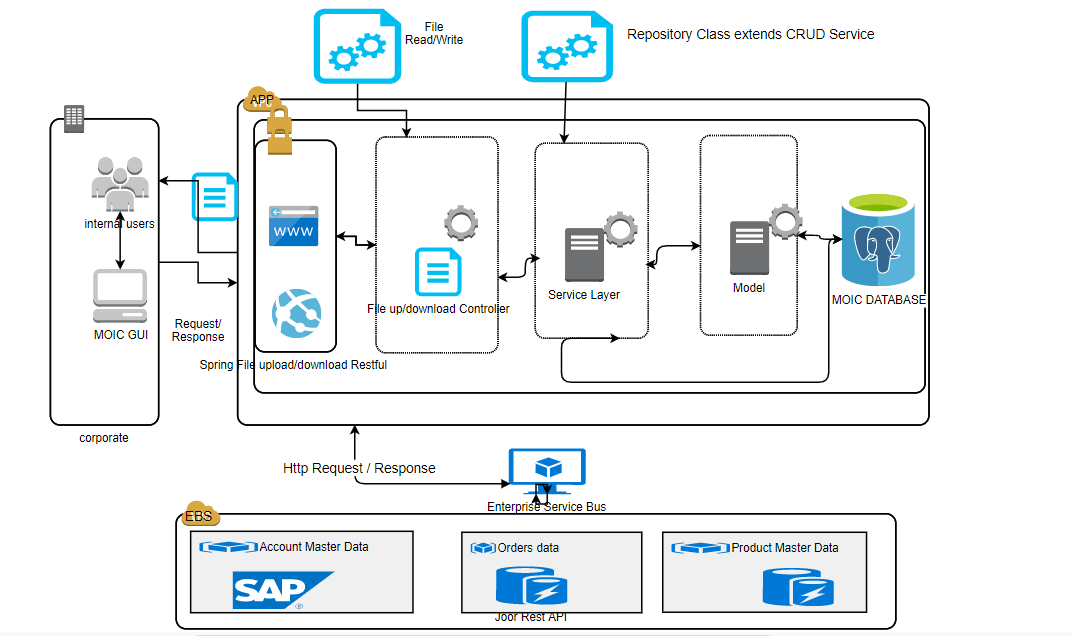


Fig 9.0 Download / Upload Mechanism

* The download excel file format will be as detailed in the SRS document -> Appendix A, Appendix B and Appendix C.

## Scheduler job for CSV files process

Spring Batch creates a Job using Step and step is configured using reader, processor and writer. To read CSV file we will create reader using FlatFileItemReader and to write data in database we will create writer using JdbcBatchItemWriter. To create a processor we need to create a Spring component by implementing ItemProcessor.

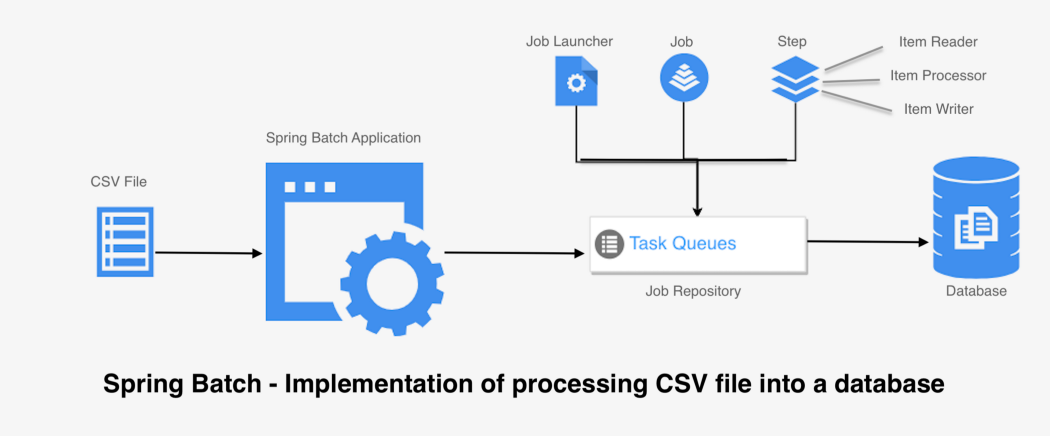


Fig 10.0 Batch process for CSV files

# User Interface - MOIC

MOIC Application will use Spring Boots for the backend and Angular 9 for the frontend.

In this Angular 9 feature, the bundle sizes are likely to decrease by 25-40 percent based on the app size. Small apps benefit the most from the tree shaking feature of IVY as they have to generate less code for the angular components.

Advantages of using Angular for front-end applications are:

* easier to debug,
* faster compilation,
* faster test execution and
* easier dynamic component loading

From a development perspective, Angular will communicate with a server over the HTTP protocol, in order to download or upload data and access other back-end services. Angular provides a simplified client HTTP API for Angular applications, the HttpClient service class in angular/common/http.

## Angular app component structure

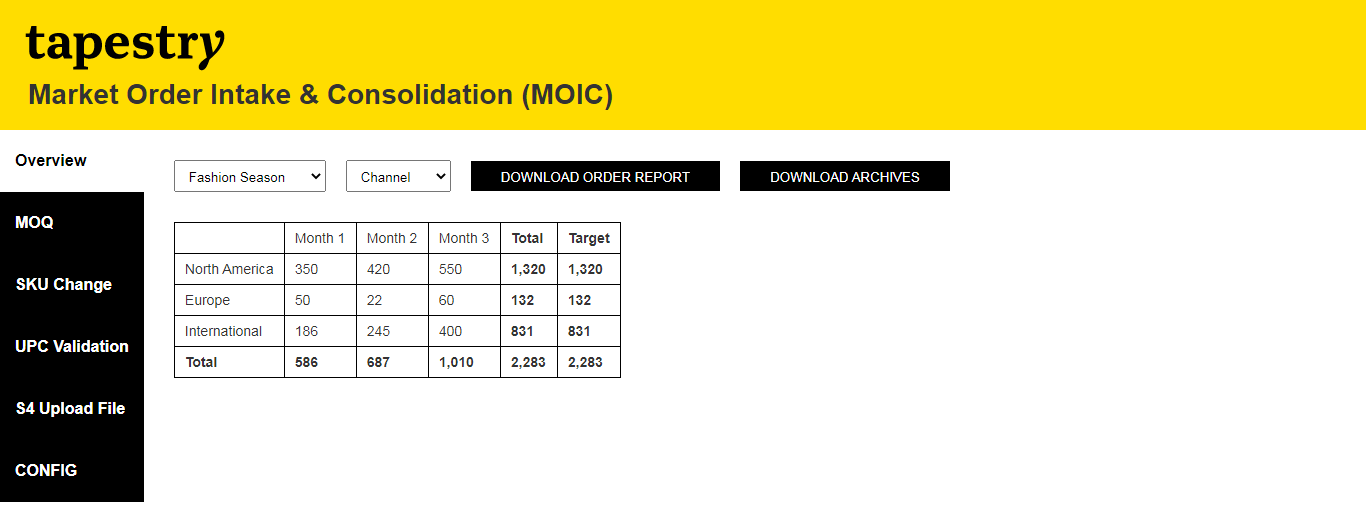
### Common / shared components:

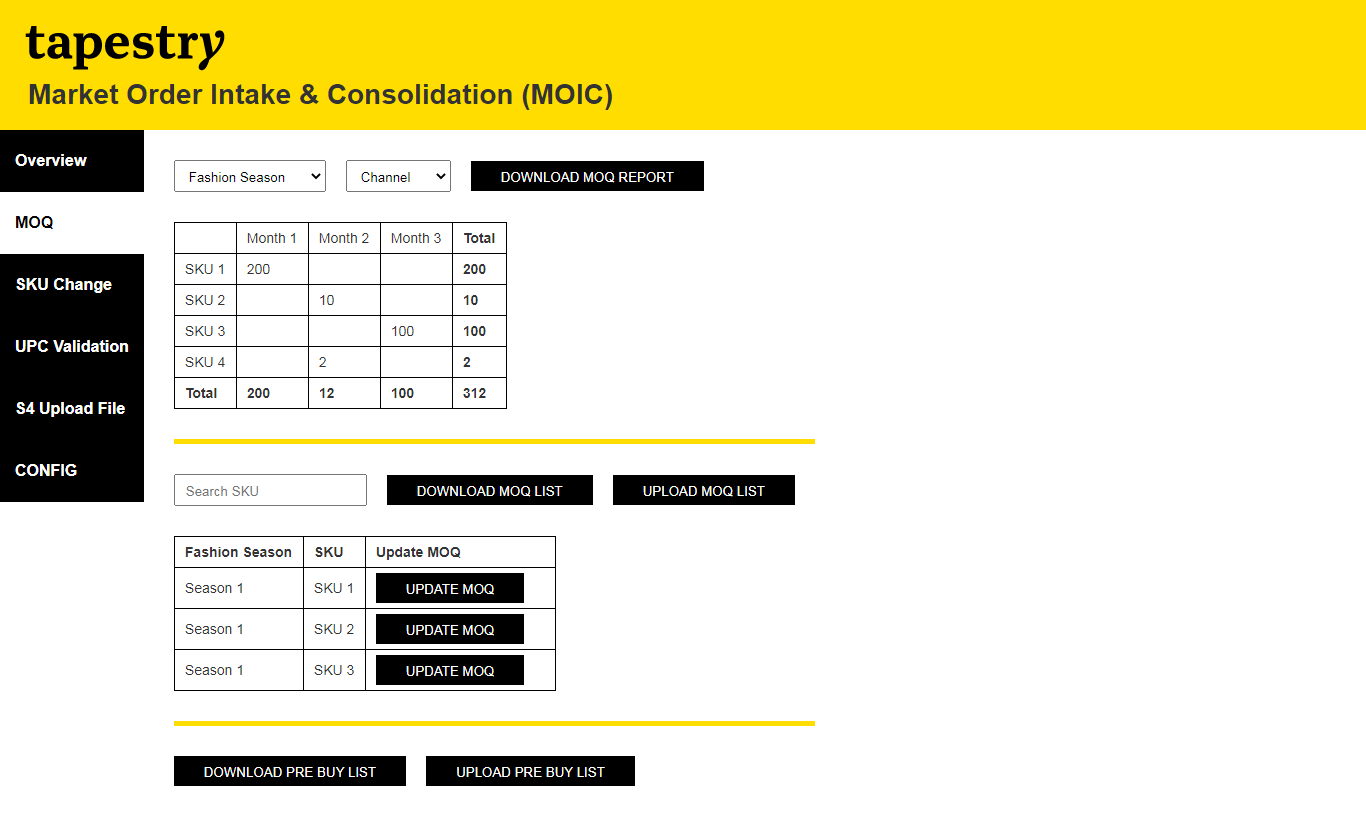
* 1. Header component
  2. Left navigation component

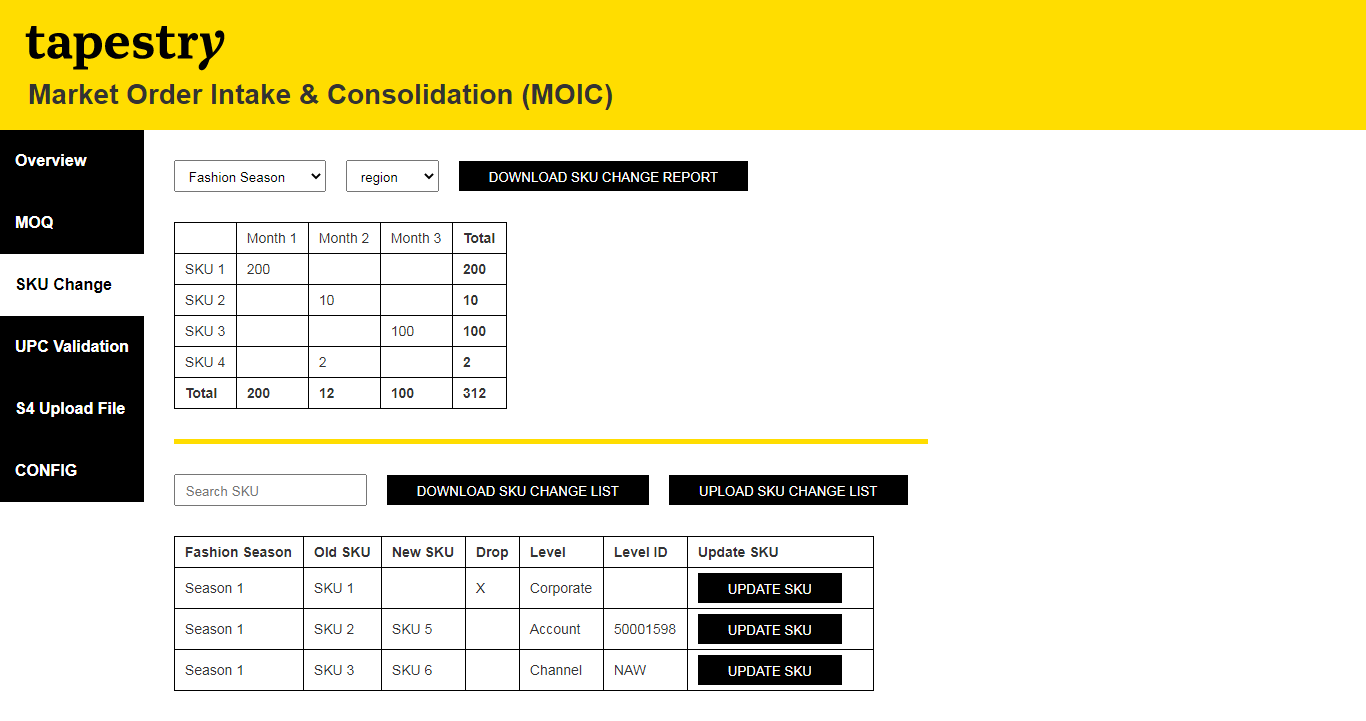
### Page level components:

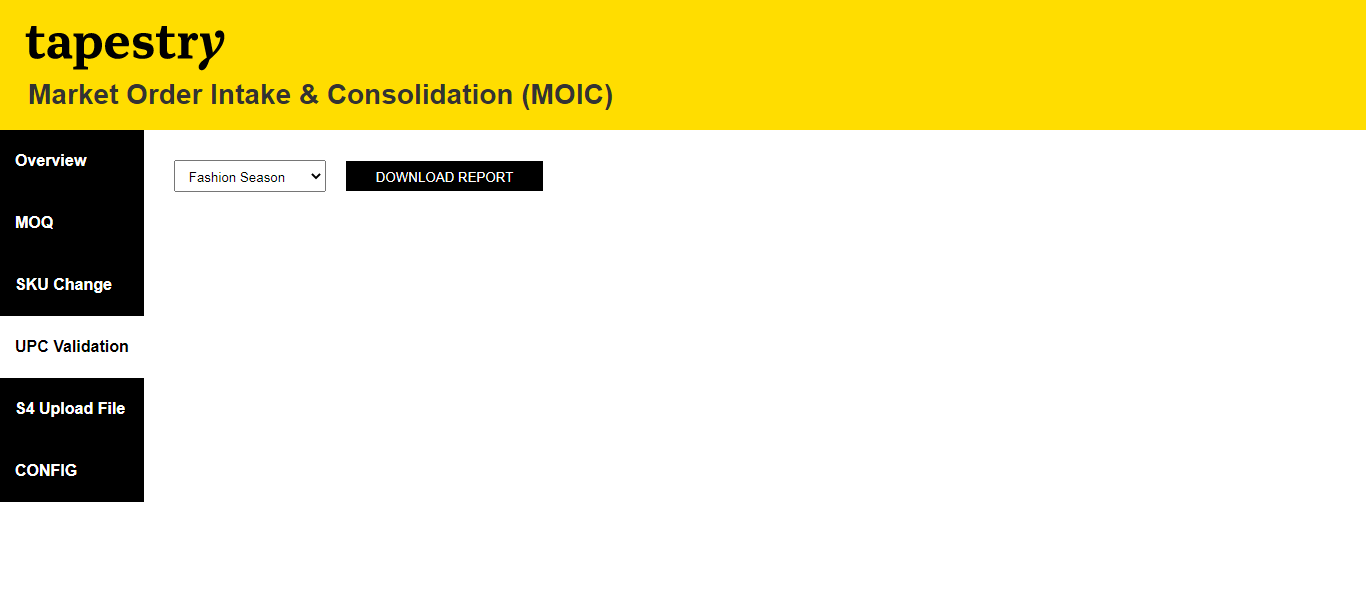
1. Overview
2. MOQ
3. SKU Change
4. UPC Validation
5. S4 Upload
6. CONFIG

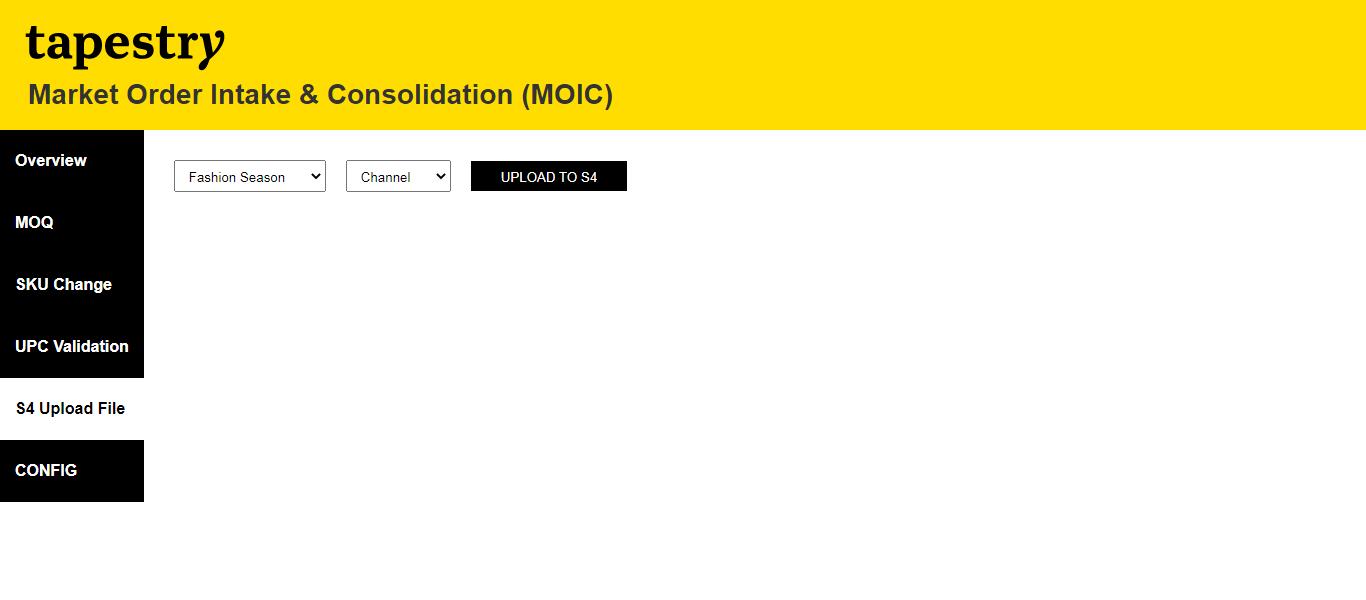
## Non Functional Screens – (Developed in Angular)

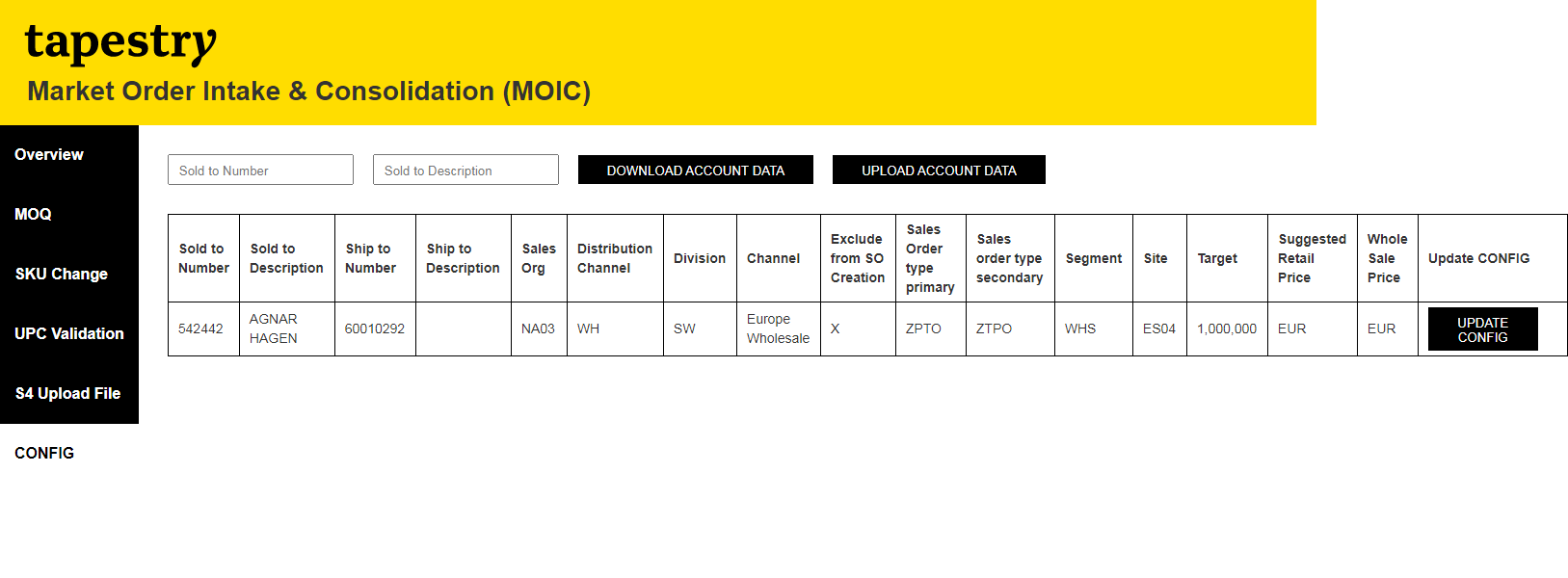












# Error Handling

An exception hierarchy will be implemented within the MOIC Application. There will be enough information available in the ‘Exception’ object such that the client (Portal front end in this case (Angular Application) and ESB which will invoke our APIs) will receive at a minimum, the same ‘Business Exception’ thrown by the Service. The exception layer will add its own exception information on top of the actual Exception and re-throw it to the client.

Business Exception, along with the error code will be written to the Log file. This may help to derive exception-based metrics at a later phase of the MOIC Application.

MOIC Application API layer will handle both a) Business Exceptions like ‘Database exceptions’ etc. and also b) Infrastructure related exceptions like ‘Application is down’, etc.

MOIC Application API layer will maintain a set of exceptions that maps to the business exception. This will be thrown as ‘System Exception’ or 'Database Exceptions' to the client layer (View) or API response to ESB.

Below are examples of few exceptions.

|  |  |  |
| --- | --- | --- |
| # | Exception | Comments |
| 1 | SystemException | If anything goes wrong while accessing APIs |
| 2 | DatabaseException | If anything goes wrong while persisting or fetching data |
| 3 | InvalidParameterException | If any parameters are invalid |

### Naming Conventions

We will be using Camel Case as a practice for writing names of methods, variables, classes, packages and constants in our MOIC application development.

* It consists of compound words or phrases such that each word or abbreviation begins with a capital letter or first word with a lowercase letter, rest all with capital.

#### Classes and Interface

Class names would be nouns in mixed case with the first letter of each internal word capitalised. Interfaces name should also be capitalised just like class names.

Use whole words and must avoid acronyms and abbreviations.

#### Methods

Methods would be verbs in mixed case with the first letter lowercase and with the first letter of each internal word capitalised.

#### Variables:

Variable names would be short yet meaningful. Variables can also start with either underscore (‘\_’) or dollar sign ‘$’ characters. Names would be mnemonic i.e, designed to indicate to the casual observer the intent of its use. One-character variable names will not be used even for temporary variables.

#### Constant variables:

Constants will all be uppercase with words separated by underscores (“\_”). There are various constants used in predefined classes like Float, Long, String etc.

#### Packages:

The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, like com, edu, gov, mil, net, org. Subsequent components of the package name will vary.

### Deployment Design

* The deployment details will be captured in a separate deployment and configuration document.

### Build Tool & Code Repository

* Gradle will be used for build, compile and deploy.
* SVN (Subversion) will be used for configuration management.

### Coding standard and compliance

* + Cignex will be internally using Sonar for checking code standards and compliance.

### Backup

* + Tapestry IT team will define and manage hot and cold backup.