|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Return Management System**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Seshadri M R |  |  | | **Role** | Solution Designer |  |  | | **Signature** |  |  |  | | **Date** |  |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc46477152)

[2.0 Introduction 4](#_Toc46477153)

[2.1 Purpose of this document 4](#_Toc46477154)

[2.2 Project Overview 4](#_Toc46477155)

[2.3 Scope 4](#_Toc46477156)

[2.4 Hardware and Software Requirement 5](#_Toc46477157)

[2.5 System Architecture Diagram 6](#_Toc46477158)

[3.0 System Requirements 6](#_Toc46477159)

[**3.1.1** **Functional Requirements – Component processing** **Microservice** 6](#_Toc46477160)

[**3.1.2** **Functional Requirements – PackagingAndDelivery Microservice** 8](#_Toc46477161)

[**3.1.3** **Functional Requirements – Payment Microservice** 9](#_Toc46477162)

[**3.1.4** **Functional Requirements – Authorization Microservice** 9](#_Toc46477163)

[**3.1.5** **Functional Requirements – Return order portal** 9](#_Toc46477164)

[4.0 Cloud Deployment requirements 10](#_Toc46477165)

[5.0 Design Considerations 10](#_Toc46477166)

[6.0 Reference learning 10](#_Toc46477167)

[7.0 Change Log 11](#_Toc46477168)

# Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Return Order Management System” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

A leading Supply chain Management Organization wants to automate the return orders, by classifying them to repair or replacement. Repair is for all main or integral part of their product. Replacement is for accessories.

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | Component processing | Component processing Module is a Middleware Microservice that performs following operations:   * Determine if the request is for Repair or Replacement * Determine the repair or replacement cost along with the consideration if it’s a priority request or not. Determine the date of process completion * Invoke Packaging and Delivery service to determine the cost and date of delivery * Return the Processing response detail object |
| REQ\_02 | Packaging and Delivery module | Packaging and Delivery Module is a Middleware Microservice that performs the following operations:   * Determine the packaging and delivery charge for the item based on a pre-defined logic * Provide the expected date of delivery |
| REQ\_03 | Payment module | Payment Module is a Middleware Microservice that performs the following operations:   * Gets the processing charge and the Credit card detail as input. Deducts the amount and provides the result message if the deduction succeeded or not |
| REQ\_04 | Authorization service | This microservice is used with anonymous access to Generate JWT |
| REQ\_05 | Return order portal | A Web Portal that allows a member to Login and allows to do following operations:   * Login * Provide detail for Return order * View the processing detail * Confirm processing |

## Hardware and Software Requirement

1. Hardware Requirement:
   1. Developer Desktop PC with 8GB RAM
2. Software Requirement (Java)
3. Spring Tool Suite (STS) Or any Latest Eclipse
4. Have PMD Plugin, EclEmma Code Coverage Plugin and AWS Code Commit Enabled
5. Configure Maven in Eclipse
6. Maven
7. Docker (Optional)
8. Postman Client in Chrome
9. Software Requirement (Dotnet)
   1. Visual studio 2017 enterprise edition
   2. SQL Server 2014
   3. Postman Client in Chrome
   4. Azure cloud access

## System Architecture Diagram



# System Requirements

### **Functional Requirements – Component processing** **Microservice**

|  |  |
| --- | --- |
| Return Order Management System | ComponentProcessing Microservice |
| **Functional Requirements**  The intent of this Microservice is to determine the Component processing detail. It interacts with packaging and delivery microservice to get the consolidated cost for the processing.  Post Authorization using JWT, based upon the type of component – Integral or Accessory, repair or replacement workflow respectively, should determine the processing details. | |
| **Entities**  **ProcessRequest**   1. **Name**   <User name>   1. **ContactNumber**   <User contact number>   1. **CreditCardNumber**   <User credit card number>   1. **DefectiveComponentDetail**    1. ComponentType – Integral / Accessory    2. ComponentName    3. Quantity   <Details of defective component>   1. **IsPriorityRequest**   <Determine if this is a high priority request or not>  **ProcessResponse**   1. **RequestId**   <A random number generated to identify the process detail>   1. **ProcessingCharge**   <Total charge for the processing>   1. **PackagingAndDeliveryCharge**   <Packaging and delivery charge>   1. **DateOfDelivery**   <Date on which the product would be delivered to the user>  **REST End Points**  **ComponentProcessing Microservice**   * + GET: /ProcessDetail (Input: ProcessRequest | Output: ProcessResponse)   + POST: /CompleteProcessing (Input: RequestId, CreditCardNumber, CreditLimit, ProcessingCharge) | Output: string response of the success of operation) | |
| **Trigger** – Should be invoked from Return order Portal (local MVC app) | |
| **Steps and Actions**   1. Return order Portal should be the front-end application where a user provides the detail of the defective component for servicing. An instance of the ProcessRequest object should be created to fill the request detail. 2. The portal should invoke the Authentication Microservice to get the JWT. 3. On receiving the token, the web portal should invoke the ComponentProcessing Microservice GET action method with the ProcessRequest object. JWT should be added to the request header for authorization. 4. The microservice design should have an Interface with a method declaration for Processing the service    * + There should be two Classes implementing the Interface. It should contain        - Repair for Integral part workflow – This workflow should cater to the logic for Integral part servicing. The default processing duration should be 5 days. If it’s a priority request, the processing can be done in 2 days with an additional charge of INR 200. Default processing charge is INR 500        - Replacement for Accessory part workflow – This workflow should cater to the logic for accessory part servicing. Is priority request doesn’t apply for replacement. Default processing charge is INR 300 5. The Process response should be displayed to the user. 6. Upon confirmation from the user to proceed, the CompleteProcessing action method should be invoked, which internally invokes the Payment microservice to complete the processing. 7. The complete processing detail(process request and response) should be saved in the database. | |
| **Non-Functional Requirement:**   * Only Authorized requests can access these REST End Points | |

### **Functional Requirements – PackagingAndDelivery Microservice**

|  |  |
| --- | --- |
| Return Order Management System | PackagingAndDelivery Microservice |
| **Functional Requirements**  ComponentProcessing Microservice interacts with PackagingAndDelivery Microservice. PackagingAndDelivery Microservice allows the following operations:  The microservice should contain a list of packaging and delivery cost detail  Packaging items   * Integral item – INR 100 * Accessory – INR 50 * Protective sheath – INR 50   Delivery items   * Delivery charge for Integral item – INR 200 * Delivery charge for Accessory – INR 100   The microservice should get the component type and count to determine the packaging and delivery charge | |
| **REST End Points**  **Claims Microservice**   * + GET: /GetPackagingDeliveryCharge (Input: ComponentType, Count | Output: PacakagingAndDeliveryCharge) | |
| **Trigger** – Can be invoked from ComponentProcessing Microservice | |
| **Steps and Actions**   * + This microservice should have only 1 REST endpoint to calculate the packaging and delivery charge   + The microservice has the detail on the charges as pre-defined values   + Based on the input, the packaging and delivery charge should be calculated and returned to the caller | |
| **Non-Functional Requirement:** | |

### **Functional Requirements – Payment Microservice**

|  |  |
| --- | --- |
| Return Order Management System | Payment Microservice |
| **Functional Requirements**  Payment Microservice should be invoked from Return order portal. it allows the following operations:   * Get the credit card details of the user and the total processing charge. Deducts the processing charge from the current amount and returns the balance amount. | |
| **REST End Points**  **Payment Microservice**   * + GET: /ProcessPayment (Input: CreditCardNumber, CreditLimit, ProcessingCharge | Output (CurrentBalance) | |
| **Trigger** – Can be invoked from ComponentProcessing microservice | |
| **Steps and Actions**   * + Get the credit card details of the user and the total processing charge. Deducts the processing charge from the current amount and returns the balance amount. | |
| **Non-Functional Requirement:** | |

### **Functional Requirements – Authorization Microservice**

|  |  |
| --- | --- |
| Return Order Management System | Authorization Microservice |
| **Security Requirements**   * Create JWT * Have the token expired after specific amount of time say 30 minutes * Has anonymous access to get the token detail | |

### **Functional Requirements – Return order portal**

|  |  |
| --- | --- |
| Return Order Management System | Return order Portal |
| **Client Portal Requirements**   * Return Portal must allow a member to Login. Once successfully logged in, the member do the following operations:   + Provide detail for Component processing   + View processing detail and charge   + Upon user confirmation, processing should complete deducting the total processing charge which should be viewed by a ‘Processed successfully’ message along with the processing detail * The request detail should be saved in the database along with the complete process request and response detail * Each of the above operations should invoke the middleware Microservices that are hosted in cloud. | |

# Cloud Deployment requirements

* All the Microservices must be deployed in Cloud
* All the Microservices must be independently deployable. They have to use In-memory database or data in the application wherever applicable
* The Microservices has to be dockerized and these containers must be hosted in Cloud using CI/CD pipelines
* The containers have to be orchestrated using AWS/Azure Kubernetes Services.
* These services must be consumed from an MVC app running in a local environment.

# Design Considerations

These design specifications, technology features have to be strictly adhered to.



# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Netcore application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on <24-Jul-2020> by <Seshadri M R> | | | |
|  |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
|  |  |  |  |