**Customer Management Microservice**

**POC**  
**Low Level Design (LLD)**



**Warner Bros. Pictures**

Date: 11/02/2022

Current Document Version: [*1.0*]

DOCUMENT APPROVAL

**Approvers of this document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Department** | **Role** | **Signature** | **Date** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Document Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Version #** | **Author** | **Date** | **Description** |
| 1.0 | Utsav Kumar | 11/02/2022 | Customer Management Microservice LLD |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1.0 Document Purpose 4](#_Toc94636300)

[2.0 Intended Audience 4](#_Toc94636301)

[3.0 Project Background, Objective(s) 4](#_Toc94636302)

[3.1 Project Background 4](#_Toc94636303)

[3.2 Project Objective 4](#_Toc94636304)

[4.0 Design Pattern 5](#_Toc94636305)

[5.0 Solution Diagram 5](#_Toc94636306)

[6.0 Solution Steps 5](#_Toc94636307)

[7.0 Classes/function name 6](#_Toc94636308)

[8.0 Validations 6](#_Toc94636309)

[9.0 Data model/Tables 6](#_Toc94636310)

[10.0 API Canvas 6](#_Toc94636311)

[11.0 Env variables 6](#_Toc94636312)

[12.0 Integrations 7](#_Toc94636313)

[13.0 AWS Role 7](#_Toc94636314)

[14.0 HTTP Status Code 7](#_Toc94636314)

[15.0 Unit Testing 7](#_Toc94636314)

[16.0 Request 7](#_Toc94636314)

[17.0 Response 7](#_Toc94636314)

# Document Purpose

This document describes the solution architecture for Customer management microservice

# Intended Audience

This document is intended as a reference for the following roles and stakeholders who are interested in the Customer Management Microservice technical architecture.

|  |  |
| --- | --- |
| Role | Nature of Engagement in WB Classics Portal Technical Architecture |
| Product Owners/SME | Key stakeholder to ensure that the architecture is aligned with business goals. |
| Business Analysts | Business analysts are one of the stakeholders who are informed with the key architectural decisions. |
| Enterprise Architects | To enforce Customer management Platform Architecture is aligned to business goals and architecture, architectural guidelines. |
| Solution Architects | To ensure solution design and architecture is aligned to business requirements, architectural guidelines. |
| Developers | Use Technical Architecture Document as the guiding document for detail design and implantation approach to align with Customer management Microservice |

# Project Background, Objective(s)

## Project Background

Customer management microservice leads to perform

Management of Customers details where one can register themselves and perform various operations

## Project Objective

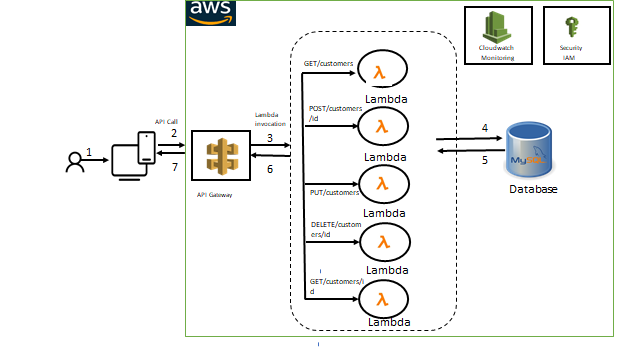
Customer Management Microservice​ will perform various operations like listing, creation, updation and deletion of Customer Details.

Customers can first register themselves and then they can perform all the operations

# Design Pattern

|  |  |  |
| --- | --- | --- |
| # | Name | Description |
| 1 | API | Using HTTP requests, we will use the respective action to trigger various operations |

# Solution Diagram



# 6.0 Solution Steps

**Customer Registration**

1. User will enter the required details such as firstName, lastNname, email, phone number, gender, age and click submit button browser directs the request to customer registration API
2. call reaches the api gateway
3. API gateway does the routing and forwards the request to registerCustomerHandler.handle. And this handle function will call the doProcess ()
4. doProcess () will call the customerschemaValidator.dovalidate() function to do the input validation it will have the customerValidator.validateCustomerRegistration as argument to perform the validation
5. If validation fails, then it will return the error code and error description. with status code
6. If validation is successful, then the handler will call the registerCustomerService.registerCustomer() which will call the registerCustomerRepository.registerCustomer() to store the data in database
7. It sends a response body with HTTP Success response to registerCustomerHandler.
8. registerCustomerHandler returns JSON Response
9. Success JSON response and HTTP status code 200 with corresponding success message.

**Customer Listing**

1. customer wants to get the customer details. enters the id in parameter for which customer wants to see the details. browser directs the request to Customer List API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to listCustomerHandler.handle this handle function calls the doProcess ()
4. doProcess () will call the listCustomerService.listCustomer() which calls the listCustomerRepository.listCustomer() to fetch the data from database.
5. It sends response body with HTTP Success response code to listCustomerHandler.
6. listCustomerHandler returns JSON Response
7. Success JSON response and HTTP status code 200 with corresponding success message.

**Customer Updation**

1. customer wants to update the details enters the id and the details which customer wants to update the details. browser directs the request to customer update API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to updateCustomerHandler.handle this handle function calls the doProcess ()
4. doProcess () will call the customerschemaValidator.dovalidate() function to do the input validation.
5. If validation fails, then it will return the error code and error description. with status code
6. If validation is successful, then the handler will call the updateCustomerService.updateCustomer() which will call the updateCustomerRepository.updateCustomer() to update the data in database
7. It sends response body with HTTP Success response code to updateCustomerHandler.
8. updateCustomerHandler returns JSON Response
9. Success JSON response and status HTTP code 200 with corresponding success message.

**Customer Deletion**

1. customer eneters the id in parameter for which customer wants to delete the details. browser directs the request to customer deletion API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to removeCustomerHandler.handle this handle function calls the doProcess ()
4. doProcess () will call the deleteCustomerService.deleteCustomer() which calls the removeCustomerRepository.removeCustomer() to delete the data from database.
5. It sends response body with HTTP Success response code to removeCustomerHandler.
6. removeCustomerHandler returns JSON Response
7. Success JSON response and HTTP status code 200 with corresponding success message.

**Particular Customer Listing**

1. User wants to get the customer details. enters the id in parameter for which customer wants to see the details. browser directs the request to Customer List API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to listOneCustomerHandler.handle this handle function calls the doProcess ()
4. doProcess () will call the listOneCustomerService.listCustomer() which calls the listCustomerRepository.listParticularCustomer() to fetch the data from database.
5. It sends response body with HTTP Success response code to listOneCustomerHandler.
6. listCustomerHandler returns JSON Response
7. Success JSON response and HTTP status code 200 with corresponding success message.

# 7.0 Classes/function

|  |  |  |
| --- | --- | --- |
| **#** | **Class** | **Description** |
| 1 | Customer.js | Model holds the customers schema details |
| 2 | registerCustomerHandler.js | The handler to handle the registration of customers which calls the registerCustomerService class |
| 3 | registerCustomerService.js | It contains the core business logic for the registration of customers. Which calls the registrationCustomerRepository class to create the customer in database |
| 4 | registerCustomerRepository.js | This class deals with the data accessibility for customer registration |
| 5 | listCustomerHandler.js | The handler to handle the listing of customers. which calls the listCustomerService class |
| 6 | listCustomerService.js | It contains the core business logic for the registration of customers. Which calls the listCustomerRepository class to list the customer from database |
| 7 | listCustomerRepository.js | This class deals with data accessibility for customer list |
| 8 | removeCustomerHandler.js | The handler to handle the deletion of customers. which calls the removeCustomerService class |
| 9 | removeCustomerService.js | It contains the core business logic for the registration of customers. Which calls the removeCustomerRepository class to remove the customer from database |
| 10 | removeCustomerRepository.js | This class deals with data accessibility for customer deletion |
| 11 | updateCustomerHandler.js | The handler to handle the updation of customer details in database which calls the updateCustomerService class |
| 12 | updateCustomerService.js | It contains the core business logic for the registration of customers. Which calls the updateCustomerRepository class to update the customer data in database |
| 13 | updateCustomerRepository.js | This class deals with data accessibility for customer deletion |
| 14 | customerValidator.js | It deals with the validation of the inputs provided by the customers |
| 15 | listOneCustomerHandler.js | The handler to handle the updation of customer details in database which calls the updateCustomerService class |
| 16 | listOneCustomerService.js | It contains the core business logic for the registration of customers. Which calls the updateCustomerRepository class to update the customer data in database |
| 17 | listoneCustomerRepository.js | This class deals with data accessibility for customer deletion |
| 18 | customerSchemaValidator.js | It contains the logic to validate schema |

# Validations



# Data model/Table

|  |  |  |
| --- | --- | --- |
|  | T\_CUSTOMERS |  |
| PK | customer\_id | INT |
|  | customer\_first\_name | VARCHAR(25) |
|  | customer\_last\_name | VARCHAR(25) |
|  | customer\_address | VARCHAR(255) |
|  | age | INT |
|  | email | VARCHAR(255) |
|  | customer\_phone\_number | VARCHAR(10) |
|  | gender | VARCHAR (255) |
|  | created\_on | Timestamp with time zone |
|  | created\_by | VARCHAR(255) |
|  | modified\_on | Timestamp with time zone |
|  | modified\_by | VARCHAR(255) |

|  |  |  |
| --- | --- | --- |
|  | T\_ADDRESS |  |
| PK | customer\_id | INT |
| FK | Address\_id | INT |
|  | country | VARCHAR (255) |
|  | state | VARCHAR (255) |
|  | city | VARCHAR (255) |
|  | pin code | VARCHAR (255) |
|  | Address line 1 | VARCHAR (255) |
|  | Address line 2 | VARCHAR (255) |
|  | created\_on | Timestamp with time zone |
|  | created\_by | VARCHAR(255) |
|  | modified\_on | Timestamp with time zone |
|  | modified\_by | VARCHAR(255) |

# 10.0 API Canvas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Micro Service | Path | Verb | API Description | Role | Auth |
| customer-mgmt-ms | /customers | POST | Customer registration | No | True |
| customer-mgmt-ms | /customers | GET | To get customer list | Admin | True |
| customer-mgmt-ms | /customers | PUT | To update customer details | No | True |
| customer-mgmt-ms | /customers/id | GET | To get details of a particular customer | No | True |
| customer-mgmt-ms | /customers/id | DELETE | To remove the customer | Admin | True |

# 11.0 ENV variables

/${self: provider.stage}/DB\_NAME

/${self:provider.stage}/DB\_USERNAME

/${self:provider.stage}/DB\_PORT

/${self:provider.stage}/DB\_PASSWORD

/${self:provider.stage}/DB\_HOST

Stage should be passed as an argument (serverless deploy $stage) stage dev/qa/prod

**12.0 Integration**

N/A

# 13.0 AWS Role

LAMBDA – RDS – Cloud Watch

LAMBDA Invocation

AWS API Gateway

# 14.0 HTTP Status Code

201 – Customer Registered

200 - Request succeeded

400 – Inputs are invalid

404 – Customer Not found

502 – Bad gateway

# 15.0 Unit Testing

|  |  |
| --- | --- |
| Project Name | Customer Management Microservice |
| Created by | Utsav Kumar |
| Date of Creation | 05/02/22 |
| Date of review | 07/02/22 |

**For Registration of customers**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test CASE ID | TEST CASE SCENARIO | TEST CASE | PRE CONDITION | TEST STEPS | TEST DATA | EXPECTED RESULT | Actual Result |
| TC\_o1 | Customer registration | Enter the valid data to get registered | Customer needs to enter all the valid details | 1) Enter  customer\_first\_name: John  customer\_last\_name:doe  emai:customer@gmail.com  Phone:0123456789  age:22  address:delhi  gender:male  2) Enter Submit | <Valid Details | Successful registration | Successful registration |
| TC\_o2 | Customer registration | Enter invalid data to get registered | Customer needs to enter the valid details with wrong phone number type | 1) Enter  customer\_first\_name: John  customer\_last\_name:do  emai:customer@gmail.com  Phone:01234567  age:22  address:delhi  gender:male  2) Enter Submit | <invalid phone number> | Phone number should be of 10 digits | Phone number should be 10 digits |
| TC\_o3 | Customer registration | Enter all the required fields to get registered | Customer must enter all the required field as per specified in schema | 1) Enter  customer\_first\_name: John  customer\_last\_name:doe  emai:customer@gmail.com  Phone:0123456789  age:22  address:delhi  gender:male  2) Enter Submit | <All the Required fields are available> | Successful registration | Successful registration |
| TC\_o4 | Customer registration | Enter all the required fields to get registered | If customer misses one of the fields during registration which is marked as required in schema | 1) Enter  customer\_first\_name: John  customer\_last\_name:doe  Phone:0123456789  age:22  address:delhi  gender:male  2) Enter Submit | <email is missing> | You need to enter email | You need to enter email |

**For customers listing**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test CASE ID | TEST CASE SCENARIO | TEST CASE | PRE-CONDITION | TEST STEPS | TEST DATA | EXPECTED RESULT | ACTUAL RESULTS |
| TC\_o1 | Customer List | enter the valid Customer Id of customer in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | Customer Details | Customer Details |
| TC\_o2 | Customer List | enters the wrong Customer Id which is not there in database to get customer details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | Customer Details with this id is not present | Customer Details with this id is not present |

**For customers Deletion**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test CASE ID | TEST CASE SCENARIO | TEST CASE | PRE-CONDITION | TEST STEPS | TEST DATA | EXPECTED RESULT | ACTUAL RESULT |
| TC\_o1 | Customer Deletion | enter the valid id of customer in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | Customer Deleted successfully | Customer Deleted successfully |
| TC\_o2 | Customer deletion | enters the wrong id which is not there in database to get customer details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | Customer Details with this id you want to delete is not present | Customer Details with this id you want to delete is not present |

**For customers Updation**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test CASE ID | TEST CASE SCENARIO | TEST CASE | PRE-CONDITION | TEST STEPS | TEST DATA | EXPECTED RESULT | ACTUAL RESULT | STATUS(PASS/FAIL) |
| TC\_o1 | Customer Updation | enter the valid id of customer in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | Customer Update  Successfully | Customer Update  Successfully | pass |
| TC\_o2 | Customer Updation | enters the wrong id which is not there in database to get customer details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | Customer Details with this  Customer Id you want to update is not present | Customer Details with this id you want to update is not present | pass |

# 16.0 Request

**Register Customer**

/customers

{

“customer\_first\_name”: “John”,

“customer\_last\_name” :”Doe”,

“email”:”customer@gmail.com”

“age” : 22,

“customer\_phone\_number” :” 0123456789”,

“gender”: “Male”,

“customer\_address”:”Jamshedpur”,

}

**Update Customer**

/customers/

{

“customer\_first\_name”: “Michael”,

“customer\_last\_name” :”Clarke”,

“email”:”customer@gmail.com”

“age” : 22,

“customer\_phone\_number” : 0123456789,

“gender”: “Male”,

“customer\_address”:”Jamshedpur”,

}

**17.0 Response:**

**Register Customer**

**If valid details**

{

"message": "Customer registered successfully"

}

status code: 201

**If invalid details**

{

“message”: “Inputs are not valid”

}, status code: 400

**If server encounters unexpected error**

{

“message” :” Internal server error ”

}, status code: 500

**List customers**

**If valid ID**

{

“customer\_first\_name”: “Michael”,

“customer\_last\_name” :”Clarke”,

“email”:”customer@gmail.com”

“age”: 22,

“customer\_phone\_number” : 0123456789,

“gender”: “Male”,

“customer\_address”:”Jamshedpur”,

}

status code: 200

**If invalid ID**

{

“message”: “Customer not found”

}, status code:404

**If server encounters unexpected error**

{

“message” :” Internal server error ”

}, status code: 500

**Update customer**

**If valid ID**

{

“Customer updated successfully”

}

status code: 200

**If invalid ID**

{

“message”: “Customer not found”

}, status code:404

**If server encounters unexpected error**

{

“message” :” Internal server error”

}, status code: 500

**Delete customer**

**If valid ID**

{

“Customer Deleted successfully”

}

status code: 200

**If invalid ID**

{

“message”: “Customer not found”

}, status code:404

**If server encounters unexpected error**

{

“message” :” Internal server error ”

}, status code: 500