

Cognizant Academy

Mini Shopper

Case Study Specification

Contents

[1.0 Problem statement 2](#_Toc113530925)

[2.0 Architecture Diagram for the Problem Statement 3](#_Toc113530926)

[3.0 Database Diagram for the Problem Statement 5](#_Toc113530927)

[4.0 Technical Scope 6](#_Toc113530928)

[5.0 Use case details 6](#_Toc113530929)

[6.0 Functional/Non-Functional Requirement of the Problem Statement 8](#_Toc113530930)

[7.0 Skills to develop the project 13](#_Toc113530931)

[8.0 Deployment Architecture 14](#_Toc113530932)

[9.0 Implementation Notes 14](#_Toc113530933)

[10.0 Evaluation rubrics 15](#_Toc113530934)

# Problem statement

The purpose of the requirements document is to systematically capture requirements for the project and the system “Mini Shop” to be developed. The application should be cloud native architecture with microservices. Both functional and non-functional requirements are captured in this document. It also serves as the input for the scope of the project.

**About the System**

The client would like to develop an independent application, **Mini Shop** application to place customers purchase orders. The portal provides facilities like customer registration, login, and create purchase orders using web portal, and client (shopkeeper) will fulfill the purchase orders. Shopkeeper can also able to Reject or Send back to customer for review the order if items are not available at shop at that moment. Finally shopkeeper should able to upload the bill against the purchase order which can be seen by customer.

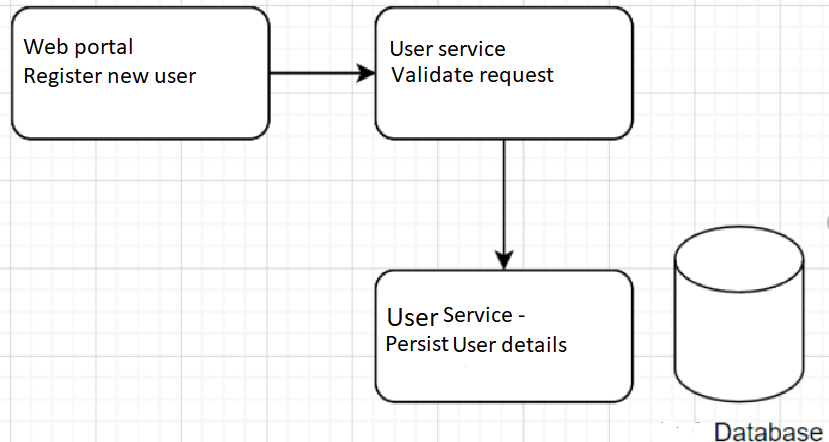
**Scope of the System**

The scope of the system is explained through its modules as follows,

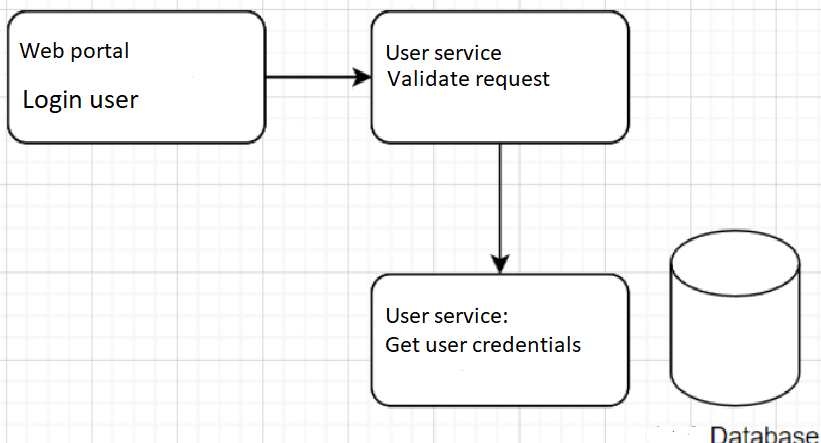
* User Registration – used by portal user to register the details of self-information and delivery address into the system. The system stores the details in the database
* Login – System should allow registered users to login to the portal.
* Purchase Order Place – Registered customer should be able to place his/her orders either using purchase form or by uploading excel sheet.
* Listing – User should be able to view list of orders with various status i.e. “Pending”, “Fulfilled”, “Rejected” and “Send for Review”.
* Editing and Replace Order - User should be able to modify and place those orders which are “Pending”, “Rejected” and “Send for Review”. Fulfilled orders should not be editable.
* Shopper can see each user’s orders and update those orders once Full-filled or rejected.
* Once fulfilled, money receipt should be generated which user can download

# Architecture Diagram for the Problem Statement

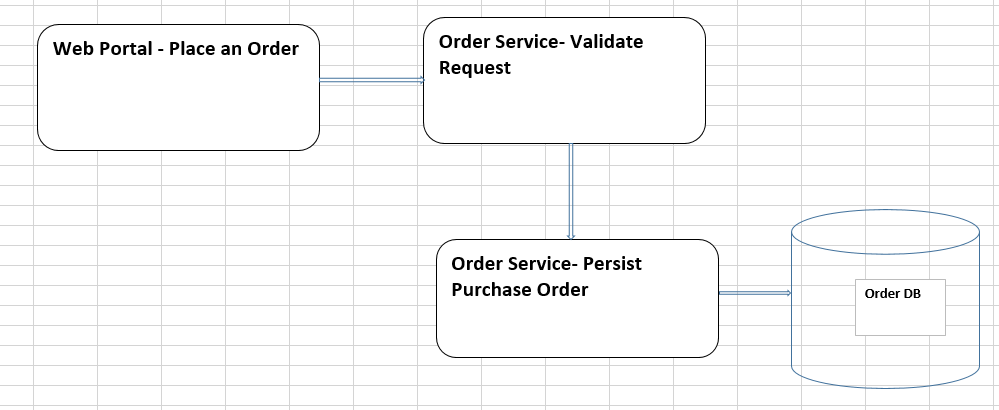
**US\_01 User/Customer Registration**



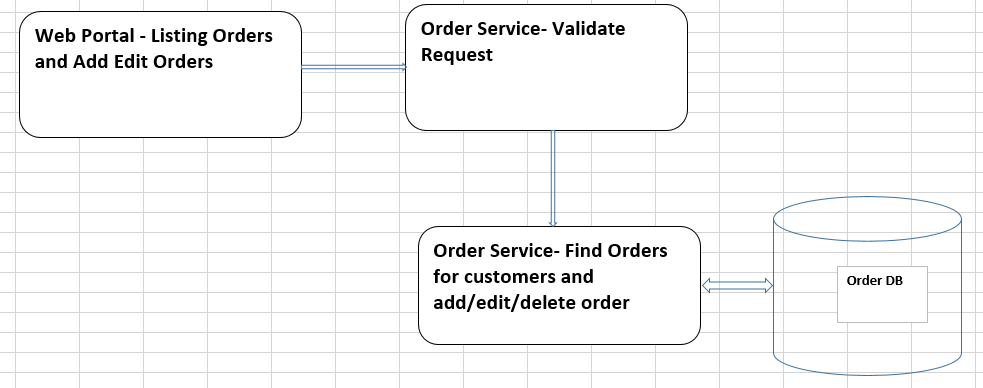
**US\_02 Customer Login**



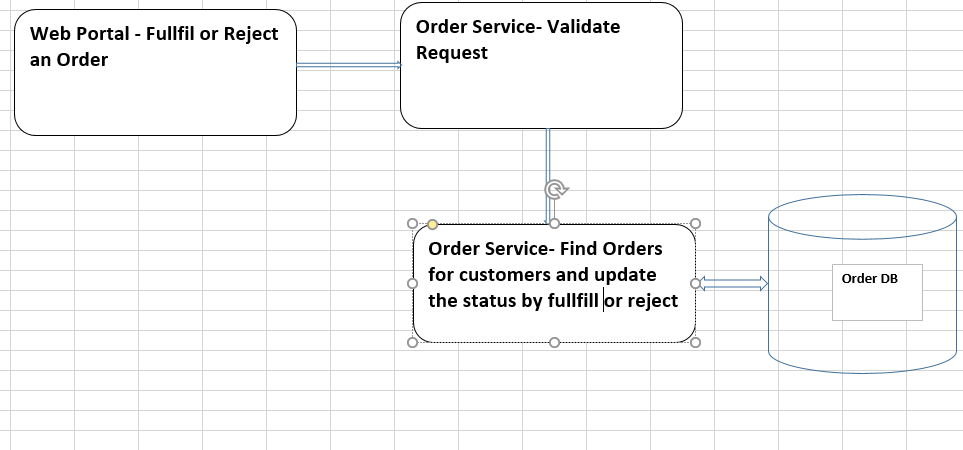
**US\_03 Place Purchase Order**



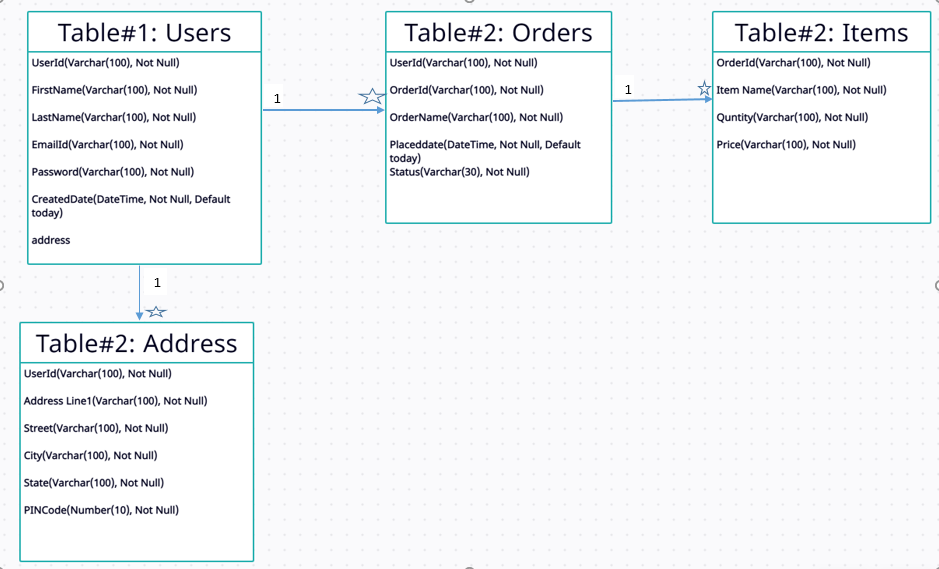
**US\_04 List Orders and View/Edit Items**



**US\_04 Fulfill Purchase Order**



# Database Diagram for the Problem Statement



# Technical Scope

|  |  |
| --- | --- |
| Compute & Integration | As an application developer, develop the application as a microservice architecture. And implement as follows:   1. Follow the Single Data Store per microservice practice 2. Document REST endpoints with OpenAPI or Swagger |
| Security & Identity | As an application developer:   1. Restrict the access over all write operation (secured operations) by adding authentication |
| Database & Storage | As an application developer:   1. Ensure Application is implementing ORM with Java JPA 2. Restructure the same as per microservice architecture 3. Use SQL SERVER for maintaining data of the microservice 4. Optimize existing implementation for Transactions |
| Governance & tooling | As an application developer:   1. Containerize the complete application 2. Perform unit testing of your application and do proper CI/CD |
| Code Quality/Optimizations | 1. Optimize the DB implementation using index search 2. Use SonarQube to scan the backend application for security vulnerabilities 3. Should have written clean code that is readable 4. Should have written testable code |

# Use case details

|  |  |  |
| --- | --- | --- |
| **User Story #** | **User Story Name** | **User Story** |
| US\_01 | Customer Registration | As a customer, I should be able to register my details in the system.  Acceptance criteria:   * Customer should be able register the details in the system, and it should be saved in the database * Capture the details like First Name, Last Name, Email Address and Password * Email address should be User ID |
| US\_02 | Login | As a customer, I should be able to login the system with valid credentials.  Acceptance criteria:   * Customer should be able login with their registered User Id aka Email Id and Password * When a user tries to login with incorrect credentials, valid error message should be shown |
| US\_03 | Place Purchase Order | As a customer, I should be able to submit purchase order through the Mini Shopper App portal.  Acceptance criteria:   * Customer should be able to place their purchase order in the Portal * System should have option to add order or upload all items orders by uploading an excel sheet. * User should validate delivery address before placed submit the order. * If the required fields are empty then appropriate validation error message must be shown. * User can add multiple items in a single purchase order. * Malicious text should be validated to prevent from CSRF and XSS attacks |
| US\_04 | View Purchase Orders and Listing item added to that purchase order | As a user, I should be able to view the purchase order and its item details.  Acceptance criteria:  User should be able add/delete/edit items to purchase order.   * This will only be possible till the purchase order pending status. * Malicious text should be validated to prevent from CSRF and XSS attacks |
| US\_05 | Fulfill and generate money receipt for purchase orders. | As a shopkeeper, I should be able to list all the purchase orders from different customers & fulfil their orders by checking each items availability.  Acceptance criteria:   * If all items available then without any modification it should be fulfilled and upload the money receipt with that order. * The status of the order should be changed to fulfilled. |
| US\_06 | Reject purchase order or Send for modification. | As a shopkeeper, I should be able to list all the purchase orders from different customers & fulfil their orders by checking each items availability.  Acceptance criteria:   * If some or no items available shop keeper can either send back to customer for modification or Rejected whole purchase order. |

# Functional/Non-Functional Requirement of the Problem Statement

|  |  |
| --- | --- |
| US\_01 | Customer Registration |
| Description  Customer should be able to register with valid details | |
| Input Parameters   * First Name * Last Name * Email ID * Password * Address | |
| Business Rules & Validations   * User Email ID should be considered as User ID. System should show success message with User ID * The user's first and last name should contain only letters and spaces * All fields should be mandatory * Email ID should contain “@” and”.” symbols and should be in valid Email format * Validation error messages should be displayed near the respective label * Email ID should be unique and should not be used previously for registration purposes * Password should contain at least one lower case letter, one upper case letter, one number, one special character and total length should be 10 * The password should be stored as an encrypted text in database * Address is the mandatory field which can be used as a delivery address. | |
| Non-Functional Requirement   * Implement the caching to get requests saved in memory and use the same for further needs. * Implement docker and containerize your service. * Use the SonarQube for code quality purpose. * Write the MS unit testcase and all the test case should be passed. * Code coverage should be 80% and above. * Write an Integration test case for the project. * Only Authorized requests can access these REST End Points | |

|  |  |
| --- | --- |
| US\_02 | Customer Login |
| Description  Users should be able to login with valid details | |
| Input Parameters   * User ID * Password | |
| Business Rules & Validations   * Shouldn’t allow submitting with empty password or empty User/Customer ID * Should show valid error message | |
| Non-Functional Requirement   * Implement the caching to get requests saved in memory and use the same for further needs. * Implement docker and containerize your service. * Use the SonarQube for code quality purpose. * Write the MS unit testcase and all the test case should be passed. * Code coverage should be 80% and above. * Write an Integration test case for the project. * Only Authorized requests can access these REST End Points | |

|  |  |
| --- | --- |
| US\_03 | Place Purchase Order |
| Description  The logged-in customer should be able to place purchase orders in the “Mini Shop” portal and fulfill their orders by checking each items availability in his/her shop. | |
| Input Parameters   * Name of the order * List of item details * User token | |
| Business Rules & Validations   * All fields are mandatory * User Id in token should be valid and not null * A purchase order must have at least 1 item and at most 50 items. * Each items must have item name and quantity. | |
| Non-Functional Requirement   * Implement the caching to get requests saved in memory and use the same for further needs. * Implement docker and containerize your service. * Use the SonarQube for code quality purpose. * Write the MS unit testcase and all the test case should be passed. * Code coverage should be 80% and above. * Write an Integration test case for the project. * Only Authorized requests can access these REST End Points | |

|  |  |
| --- | --- |
| US\_04 | Listing Purchase Orders and Listing items for Edit/Delete |
| Description  The logged-in customer should be able to list down all purchase orders placed and able to add/dele items to purchase order. Also able to edit the item quantity and then placed to the shopkeeper. | |
| Input Parameters   * Name of the order * List of item details * User token | |
| Business Rules & Validations   * All fields are mandatory * User Id in token should be valid and not null * Address should have PIN and at least 100 characters. | |
| Non-Functional Requirement   * Implement the caching to get requests saved in memory and use the same for further needs. * Implement docker and containerize your service. * Use the SonarQube for code quality purpose. * Write the MS unit testcase and all the test case should be passed. * Code coverage should be 80% and above. * Write an Integration test case for the project. * Only Authorized requests can access these REST End Points | |

|  |  |
| --- | --- |
| US\_05 | Fulfill/Reject Order and Generate Money Receipt. |
| Description  The shopkeeper should be able to see the listed orders from various customers in ‘Mini Shop’ portal. | |
| Input Parameters   * Orders * Customers   **Output Parameters**   * Orders with status update. | |
| Business Rules & Validations   * Item quantity must have a valid quantity. | |
| Non-Functional Requirement   * Implement docker and containerize your service. * Use the SonarQube for code quality purpose. * Write the MS unit testcase and all the test case should be passed. * Code coverage should be 80% and above. * Write an Integration test case for the project. * Only Authorized requests can access these REST End Points | |

**Service Requirements**

**US\_01 Customer Registration**

Once the user enters the details, they should be sent to the POST method and saved in the db. Mandatory fields should be validated as mentioned in the rules above and 400 status code exception, response should be sent with the missing field details. When the details are saved successfully, the service should response 200 ok along with success message. If there are any exceptions while connecting/saving to DB, the service should throw corresponding error with error status as 500.

Header: NA

Body:

* FirstName
* LastName
* EmailID
* Password
* Address

**US\_02 Customer Login**

Once the user enters the details, they should be sent to the POST method. Mandatory fields should be validated as mentioned in the rules above and 400 status code exception, response should be sent with the missing field details. When the details are authenticated successfully, the service should response 200 ok along with success message. If there are any exceptions while connecting/saving to DB, the service should throw corresponding error with error status as 500.

Header: NA

Body:

* UserID
* Password

**US\_03 Placing Purchase Orders**

Once the customer place an order in ‘Mini Shop’ portal, portal will make a call to Place Order service with following details,

Header:

* Token

Body:

* PurchaseOrder

**POST */minishop/placeorder***

If the user id is not valid or the provided token is expired, then 401 will be returned. Request will be validated if model validation fails then service will return 400 as response. If the request is valid, then the Order details will be persisted to database and return 200 as response.

**US\_04 Listing Purchase Orders for a customer**

Service should able to fetch all the placed purchase orders from database for all customers.

Header:

* Token

**GET */minishop/orders/{userid}***

If the user id is not valid or the provided token is expired, then 401 will be returned. Request will be validated if model validation fails then service will return 400 as response. If the request is valid, then the order will be retrieved from database and return 200 as response.

**US\_05 Listing All Purchase Orders**

Once shopkeeper logged in, the Portal will make a call to ***orders*** service with following details,

Header:

* token

**GET */minishop/orders***

If the user id is not valid or the provided token is expired, then 401 will be returned. Request will be validated if model validation fails then service will return 400 as response. If the request is valid, then the orders will be retrieved from database and return 200 as response.

**Expected Deliverables**

The following deliverables are expected as outcomes:

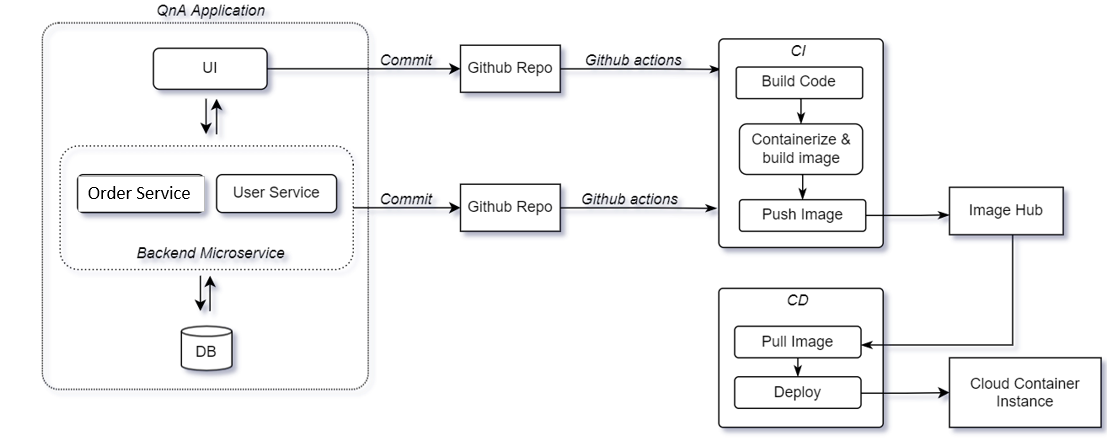
* Application Code base
* Readme document on the complete application
  + Setup of the application
  + How to run the application
  + Any inference
  + Screenshot of UI results
* Reports:
  + Unit/Functional test report

# Skills to develop the project

List the Technology based on your respective technology stack, that will be used to develop the project.

|  |  |
| --- | --- |
| Skill Stack | Core Java, Spring Boot, Hibernate |
| Front end | Angular/ReactJS  CSS,  Typescript/JavaScript,  Karma/ Jasmine |
| Service End | Spring Web and Entity Framework |
| Database | SQL Server |
| Source Control | GIT, GitHub, GitHub Actions for CI & CD. |
| Cloud | Azure/AWS/Google Cloud |
| Unit testing | JUnit & Web Mock framework |

# Deployment Architecture



# Implementation Notes

As per the project requirement modification can be done in the table below.

|  |  |  |
| --- | --- | --- |
| Milestone -1 | 5 days | * Implement user-stories using React. * Design application with minimum backend or mock backend as the focus for milestone-1 in on frontend skills * Implement forms, data binding, validations * Use appropriate unit test framework |
| Milestone -2 | 10 days | **Spring Boot REST API:**  Implement Order service and User service by following below guidelines,  Create Order service using Spring Boot REST Microservices to perform SAVE Operation using POST method.   * Using Microservices architecture * Follow coding standards * Follow standard project structure. * Log all request details. * Log errors. * Message input/output format should be in JSON (read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant * Database connections and web service URLs should be configurable. * Use browser / POST Man to invoke APIs * Run SonarQube for code quality. * Implement MS-Test and MOQ for unit testing. |
| Milestone -3 | 5 - days | * Integrate service layer with UI component. * Setup CI & CD pipelines. * Dockerize the application. |

# 10.0 Evaluation rubrics

|  |  |
| --- | --- |
| Angular | * Associate must have used angular components, modules, data binding, data validation, CLI commands * Associate must have used forms and forms validation * Associate must have used directives * Associate must have developed reusable components * Associate must have followed coding standards |
| REACT | * Associate must have used component, databinding, data validation, CLI commands. * Associate must have used forms and forms validation * Associate must have defined react state * Associate must have followed coding standards |
| Microservices | * REST controller * Follow controller -> service -> Dao model * Entity and model classes * Appropriate logging statements * Exception handling |
| Docker | * Dockerize the application * Build docker containers * Push your docker images to repository with the docker push command |