Truck Scheduling Application System

Contents

[1.0 Problem statement 2](#_Toc52981935)

[2.0 Skills to develop the project 3](#_Toc52981936)

[3.0 Use Case Diagram for the Problem Statement 3](#_Toc52981937)

[4.0 User Stories 4](#_Toc52981938)

[5.0 Expected Deliverables 6](#_Toc52981939)

[6.0 Milestone and duration 6](#_Toc52981940)

[7.0 Implementation Notes 7](#_Toc52981941)

[8.0 Evaluation rubrics 8](#_Toc52981942)

# Problem statement

The purpose of this requirement document is to systematically schedule truck delivery in distribution centers and receive the orders from the vendors of a retailer. The application should be built on Java Spring Boot, Kafka and deployed to Azure Cloud. Both functional and non-functional requirements are captured in this document.

**About the System**

The client would like to develop an independent application ‘Truck Scheduling Application System’ application in order to maintain DCs, DC schedules or slots, truck maintenance, vendor maintenance, truck map to appointment slots in DCs, integrate with PO system or a 3rd party system, send the appointment schedule to 3rd party systems and dashboard.

All the below functionality will be served as a Service via REST end point.

The following section will cover aspects related to Application.

1. DCs
2. Truck maintenance
3. Vendor maintenance
4. DC slots or schedules
5. Appointment scheduling

The below functionality has the integration using Apache Kafka with Spring Boot

1. PO system (to download the PO information)
2. Send appointment information (to send the information 3rd party systems)

**Scope of the System**

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

* DC – Add, update, delete and search DC.
* Truck Maintenance - Add, update, delete and search trucks.
* Vendor Maintenance - Add, update, delete and search vendors.
* DC slots or schedules – Add, update, delete and search DC slots or working schedules.
* Appointment Scheduling - Add, update, delete and search appointment schedules.
* Integration with PO information to download POs
* Integration with 3rd party to send the appointment information.

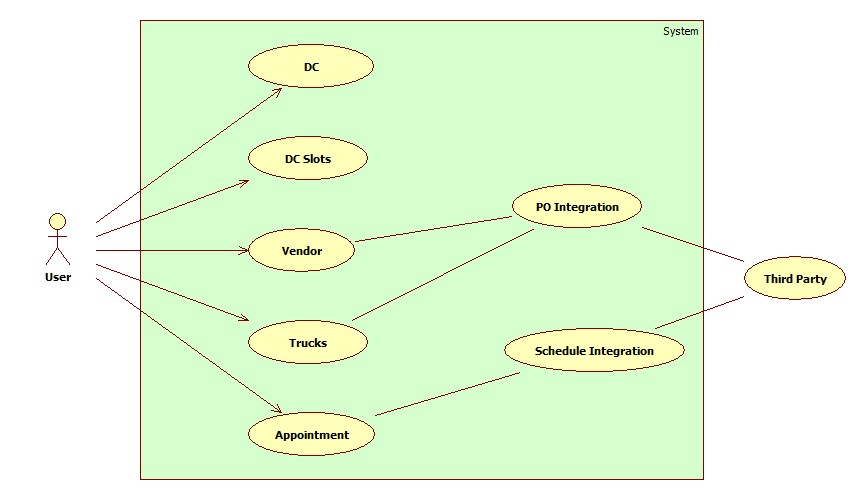
# Skills to develop the project

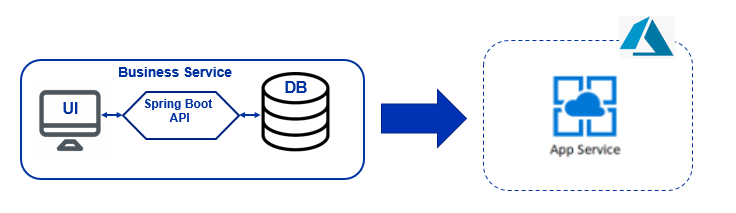
Associate will implement skills from Backend and Azure Cloud platform to develop the application.

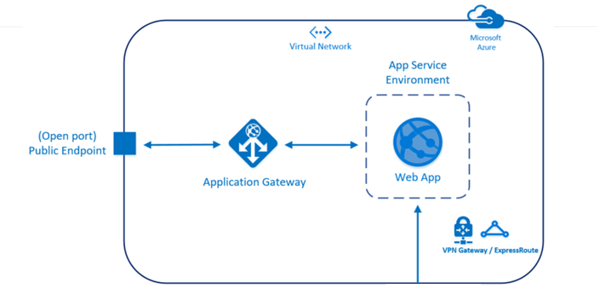
Below are the skill details.

|  |  |
| --- | --- |
| **Tower Name** | **Topics** |
| Backed - Java | Core Java 8, JUnit  Spring Boot with Hibernate  Microservices  Junit  Kafka  MySQL |
| Cloud - Azure | Azure App services |

# Use Case Diagram for the Problem Statement

****





# User Stories

|  |  |  |
| --- | --- | --- |
| **User Story #** | **User Story Name** | **User Story** |
| US\_TSAS\_01 | DC | Customers can add, update, delete and search DC.  Acceptance criteria:   * UI to add, update, delete and search DC * Use MySQL DB table to store and retrieve information. * Fields are DC number, DC city, DC type (International, Regional and Imports) * REST APIs to do all the operation in JSON format |
| US\_TSAS\_02 | DC Slots | Customers can add, update, delete and search DC slots.  Acceptance criteria:   * UI to add, update, delete and search DC slots * Use MySQL DB table to store and retrieve information. * Fields are DC number, DC timeslots (Can be like 6.00 to 7.00, 7.00 to 8.00 etc with 1 hour duration), max trucks * REST APIs to do all the operation in JSON format |
| US\_TSAS\_03 | Vendor | Customers can add, update, delete and search vendors.  Acceptance criteria:   * UI to add, update, delete and search vendors * Use MySQL DB table to store and retrieve information. * Fields are vendor name, vendor email, vendor phone number, vendor address. * REST APIs to do all the operation in JSON format |
| US\_TSAS\_04 | Truck | Customers can add, update, delete and search trucks.  Acceptance criteria:   * UI to add, update, delete and search trucks * Use MySQL DB table to store and retrieve information. * Fields are truck number, truck name, truck type (Straight Truck, Flatbed (Flat Bed) Trailers, Conestoga Trailers, Refrigerated (Reefer) Trailers) * REST APIs to do all the operation in JSON format |
| US\_TSAS\_05 | Appointment Scheduling | Customers can create, modify, cancel and search truck appointments.  Acceptance criteria:   * UI to create, modify, cancel and search truck appointments * Use MySQL DB table to store and retrieve information. * Fields are truck number, DC, DC slot, PO numbers (List of PO numbers downloaded in the table), and date of appointment. Need to check if the slot already reached the max trucks in the timeslot. Check if the PO is present in the application DB before scheduling. * REST APIs to do all the operation in JSON format |
| US\_TSAS\_06 | PO Download | System should have the ability to download the PO information through Kafka from 3rd party system.  Acceptance criteria:   * System should have the ability to download the PO information from the 3rd party system * Third party system will send the information in XML format in the Kafka * Use MySQL DB table to store and retrieve information. * Fields are PO number, PO date, PO address, PO line number, UPC number, UPC name/Item name, and ordered quantity. |
| US\_TSAS\_07 | Send Scheduling Information | System should have the ability to send the appointment scheduling information to any third party through Kafka.  Acceptance criteria:   * After successful appointment creation send the information to a Kafka channel. * The format of the data should be in JSON |

# Expected Deliverables

The following deliverables are expected as outcomes

* Application Code base
* API for each service
* Readme document on the complete application
  + Setup of the application
  + How to run the application
  + Any inference
  + Snapshot of any implementation
* Reports:
  + Code Quality Report.
  + Functional Test Report
  + Code Profiling Report

# Milestone and duration

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

|  |  |
| --- | --- |
| Milestone | Topic |
| Milestone -1 | Develop the required APIs using Java, Spring Boot, Hibernate and Kafka |
| Milestone-2 | Deploy the application to Azure  Common web application architectures | Microsoft Docs |

# Implementation Notes

|  |  |
| --- | --- |
| Backend – Java | Milestone-1   * UI is optional or minimal * Use Spring Boot- Rest APIs to develop the services * Use Hibernate or JPA for ORM * Ensure 12 factor app methodology is followed * Use Micro service Architecture and Domain Driven Design * Use Swagger definitions * Use Spring Data JPA to work with database * Use browser / POST Man to invoke APIs * User access security micro service to allow/disallow CRUD operations * Any error message or exception should be logged (and help in refactor) * Unit test the application * All implementation should publish Code Quality Metrics using SonarCloud/SonarQube * Technical Debt – lower-the-better * Code Smell – lower-the-better * Cyclomatic Complexity - lower-the-better * Code Coverage – higher-the-better * Secure coding practices * Follow coding standards * Create publisher and consumer using Java (Spring Boot) * Basic error handler if there is a failure during processing Apache Kafka message * Kafka should be used to publish and subscribe the messages |
| Cloud - Azure | Milestone-2   * Deploy the application to Azure App services   + - Deploy the code to Azure App services |

# Evaluation rubrics

|  |  |
| --- | --- |
| Microservices | * Follow the below basic structure   + API - Controllers   + Domain - Model, Events, Business Services Integration   + Services – API Implementation   + Infrastructure Project * Associate must have designed/developed Microservices as per the requirement * Each of the Microservices need to comprise below functionality, which need to be developed * Entity & Model classes, including appropriate relationship (like One-One, Many-One, etc…) between Entity Classes. (Entity and Model classes have been developed in the Previous Phase) * In case specific Entity or Model classes are required across multiple Microservices, it is recommended to maintain separate copy of Entity or Model classes for each Microservices. * Microservices should interact with corresponding databases it owns. * Microservice need to interact with other Microservice * Usage of Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice * Circuit Breaker, Service Registry, Service Discovery should be implemented |
| Rest API | * Associate must have used REST API for exposing resources * Associate must have used HTTP GET/PUT/POST request method designators for the business methods which is to be exposed * Associate must have customized the request and response formats according to the requirement * Associates must have used appropriate RETURN CODES based on the service outcome * Associates must have extracted query/form/header parameters from the input * Associate must have built a custom response based on the input * Use Swagger UI and test each public method in the service * Implementation of Repository pattern * Implement Service Registration and Discovery |
| Java | Associate should have used at least 5 java 8 features |
| Unit Testing | * Test cases covers the functionality of API with custom inputs * Good test Coverage |
| Common | * Code Smell * Technical Debt * Secured Coding * Coding Standards |
| Azure | * Application should be deployed to Azure App services |