**Air Cargo Service Automation**

**Fully Integrated Case study for Full Stack Java Developer**

**Time required to complete – 42 hours**

**What is the project about?**

This project captures the various concepts, techniques and skills learnt and helps to put them into practice using a full stack problem that a software engineer must solve. Admittedly, this would be at a scaled-down level since the purpose is to let the associate experience the various stages of full-stack development as an individual. The individual associate is expected to carry out the project and complete it within a period of 42 working hours.

**Problem Specification**

TekSmart Logistics Pvt Ltd is a leading logistics company. The company offers air cargo services from Mumbai Airport office to ten international locations and vice versa. The company can ship cargo having a weight per item not exceeding 22 kilograms. A single consignment can have a maximum weight of 88 kilograms. The following are the kinds of services provided by TekSmart Logistics: *Normal* - Normal Service consists of consignment delivery using air cargo services offered by scheduled airlines. *Express* - Express Delivery consists of consignment delivery using cargo aircraft of DHL. *Platinum* - Platinum service is a premium service which involves the usage of the exclusive Boeing cargo aircraft of TekSmart Logistics Pvt ltd. The consumer must pay for the consignment based on the weight of the package and the distance.

TekSmart Logistics has the following business activities that must be automated.

1. Register Customer details
2. Update Customer Profile Information
3. Book consignment
4. View booked consignment
5. Reports

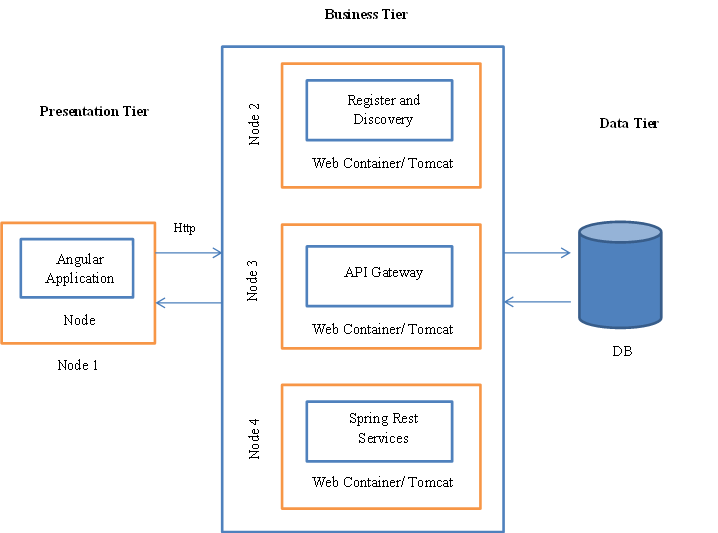
**Architecture diagram:**

The Three-tier Architecture is a software design pattern in which the functional process logic, data access, computer data storage, and user interface are developed and maintained as independent modules on separate platforms.

**Presentation Tier**- The Presentation tier is the front-end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user.

**Business Tier**- The application tier contains the functional business logic which drives an application’s core capabilities.

**Data Tier**- The data tier comprises of the database/data storage system and data access layer.

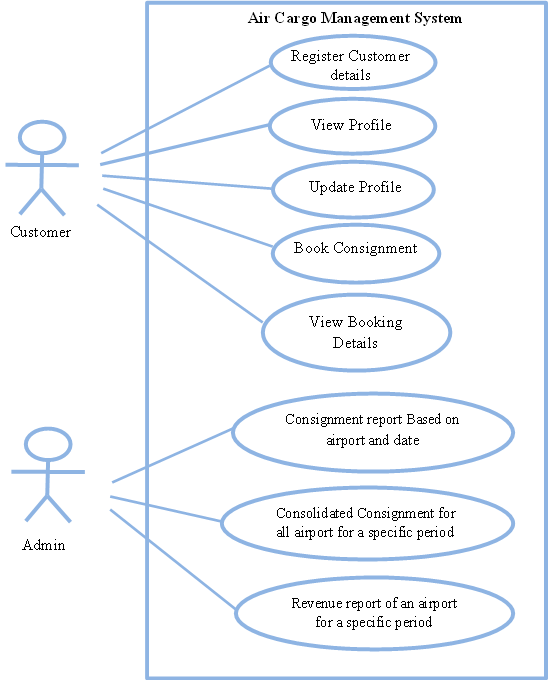


**Software Requirements:**

Kindly refer to the table below for Skill set to be used to implement the requirements for Air Cargo Management System.

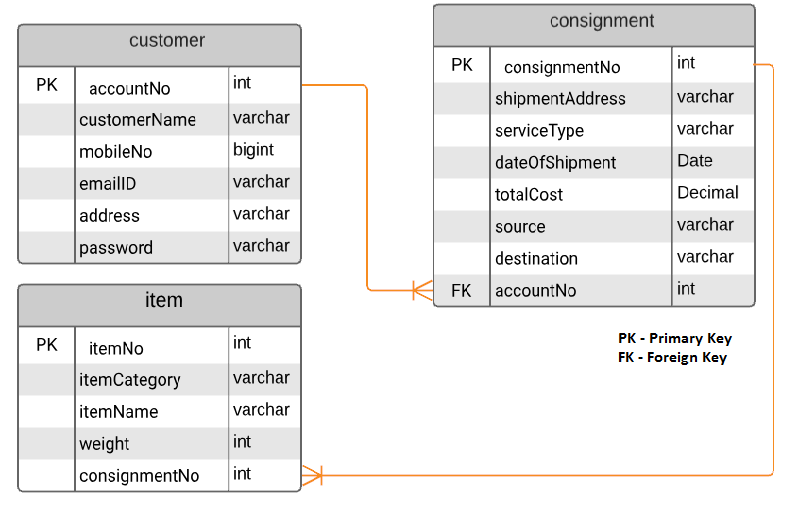
|  |
| --- |
| Skill set to be used |
| * HTML5 * CSS3/bootstrap * JQUERY * Angular * REST Restful Web Service (JAX – WS) * JSON * Spring boot * spring data JPA * MYSQL |

**Use case Diagram:**



**Database Design:**

1. **E-R Diagram**



1. **Table Structure**

**Table Name: customer**

|  |  |  |
| --- | --- | --- |
| Column Name | Column Type | Description |
| customerName | varchar | Name of the customer |
| mobileNo | bigint | Mobile no of the customer |
| emailId | varchar | Email of the customer |
| address | varchar | Address of the customer |
| password | varchar | Password of the customer |
| accountNo | int | Primary Key |

**Table Name: consignment**

| Column Name | Column Type | Constraints | Description |
| --- | --- | --- | --- |
| consignmentNo | int | Primary Key | Unique number generated for a consignment |
| shipmentAddress | varchar |  | Address to be shipped |
| serviceType | varchar |  | Service Type – normal / Express / Platinum |
| dateofshipment | date |  | Shipment Date |
| totalCost | decimal |  | Accumulated cost of all the items (max of 4) booked in the consignment |
| source | varchar |  | Source from where the consignment is shipped |
| destination | varchar |  | Destination of the shipment |
| accountNo | int | Foreign Key | Unique number for the customer |

**Table Name: item**

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Column Type | Constraint | Description |
| itemNo | int | Primary Key | Unique number generated for the item |
| itemCategory | varchar |  | Category of the item |
| itemName | varchar |  | Name of the item |
| weight | int |  | Weight of the item |
| consignmentNo | int | Foreign Key | Consignment number for which the item is added |

**Design Constraints:**

1. Use MYSQL database to store the data. The database name should be **“cargo”.**
2. The above tables should be created by the application automatically or you need to provide the SQL script to create the tables. The script should be placed in the file “script.sql”.
3. Establish proper relationships between the tables.
4. The table names and the columns names should be the same as is specified in the table structure.
5. Database connections should be configurable; it should not be hard coded. The database information should be specified in the “**application.properties**” file.

**UI Design Document**

**Functional Requirements**

1. **Register Customer Details**

Customer can have his own logistics account by registering himself in the TekSmart Logistics portal. In this way he is eligible to send a consignment and track the same. The following details are to be entered in the Web based form by the Customer for registration: -

* Name of the Customer
* Customer Address
* Mobile Number
* Email Id

**Pre-Condition:** The required tables should be created at the backend.

**Post- Condition**: Customer details must be stored in the database along with the generated account no and password. Initially the mobile will be the password.

**Success Scenario**: On success, the customer object should be returned. The UI application should accept JSON response and display the “Account no and initial password”.

**Failure Scenario**: The application should display an appropriate error message if any of the validation fails. The below client-side validations should be implemented.

**Business Rules:**

1. All fields are mandatory.
2. Customer name should contain only alphabets and space.
3. Mobile no should hold only 10 digits, it should start with 7/8/9.
4. Email id should be a valid email id. It should contain ‘@’ and ‘.’ special characters only.
5. All the above validations should be performed using HTML5/Angular JS.

**Component Specification (DAO and Model)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package name** | **Class/Interface** | **Attributes** | **Method** |
| com.cargo.model | Customer(class) | int accountNo  String customerName  long mobileNo  String emailId  String address  String password  List<Consignment> bookingList | Include getters and setter method for all the attributes. |
| com.cargo.dao | CustomerDAOImpl(class) |  | Customer registerCustomer(Customer customer ) |
| com.cargo.dao | CustomerDAO(Interface) |  | Customer registerCustomer(Customer customer ) |

**Service Specification**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Package**  **Name** | **Class/Interface** | **Attributes** | **Request**  **Method** | **URI** | **Content type** | **Input** | **Output** |
| com.cargo.service | CargoService(class) | CustomerDAO – Should be injected using annotation | POST | /register | JSON | Customer Object | Customer Object |

**Microservice Specification:**

Spring boot application containing the REST services-

Application name = **“aircargoservice”** (to be specified in properties file)

**Eureka server client port** to be specified in **application.properties** file.

API gateway server-

Must contain a routing information for /register URI that is the mapping to the actual implementation running at port 8080.

The API gate way server port (8085) MUST be specified in the “**application.properties**” file.

*The service must be accessible through API gateway using small-case url.*

**UI Components Specification**

Associated screen no:1

**Flow of events**

View

API Gateway

Service

DAOImpl

JSON Object

Route Customer JSON Object

Persist the Customer Object

Customer Object with generated AccNo and Password

Customer Object as JSON

Customer Object as JSON

Enter the Customer details

Display the AccNo and Password

1. **Update Profile**

Once the customer clicks the update profile, the customer profile details will be auto-populated in the screen. Customer has the privilege to update their profile details such as Address, Mobile number, password or Email Id.

**Pre-Condition:** The required tables should contain the required data at the backend.

**Post- Condition**: Customer details will be updated in the database.

**Success Scenario**: On success, the updated customer object should be returned. The UI application should accept JSON response and display the “updated customer information”.

**Failure Scenario**: The application should display an appropriate error message if any of the validation fails. The below client-side validations should be implemented.

**Business Rules:**

1. All fields are mandatory.
2. Mobile no should hold only 10 digits, it should start with 7/8/9.
3. Email id should be a valid email id. It should contain ‘@’ and ‘.’ special characters only.
4. All the above validations should be performed using HTML5/Angular JS.

**Component Specification (DAO and Model)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package name** | **Class/Interface** | **Attributes** | **Method** |
| com.cargo.model | Customer(class) | int accountNo  String customerName  long mobileNo  String emailId  String address  String password  List<Consignment> bookingList | Include getters and setter method for all the attributes. |
| com.cargo.dao | CustomerDAOImpl(class) |  | Customer updateCustomer(Customer customer ) |
| com.cargo.dao | CustomerDAO(Interface) |  | Customer updateCustomer(Customer customer ) |

**Service Specification**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Package**  **Name** | **Class/Interface** | **Attributes** | **Request**  **Method** | **URI** | **Content type** | **Input** | **Output** |
| com.cargo.service | CargoService(class) | CustomerDAO – Should be injected using annotation | PUT | /update | JSON | Customer Object | Customer Object |

**Microservice Specification:**

Spring boot application containing the REST services-

Application name = **“aircargoservice”** (to be specified in properties file)

**Eureka server client port** to be specified in **application.properties** file.

API gateway server-

Must contain a routing information for /update URI that is the mapping to the actual implementation running at port 8080.

The API gate way server port (8085) MUST be specified in the “**application.properties**” file.

*The service must be accessible through API gateway using small-case url.*

**UI Components Specification**

Associated screen no:2

**Flow of events**

View

API Gateway

Service

DAOImpl

Display Customer Profile

Route Customer JSON Object

Update the Customer Object

Updated Customer Object

Updated Customer Object as JSON

Customer Object as JSON

Change Customer details & update

Display updated Customer Details

1. **Book Consignment**

The Customer can book one or more consignments. Each consignment can have a maximum of 4 items in it. Customer must provide the following details to book:

1. Source
2. Destination
3. Shipment Address
4. Item to be Shipped
5. Item Category
6. Weight in kilograms
7. Service Type

The customer can add up to 4 items by clicking “Add more items”. Source, destination, shipment address, and service type will remain same for all items in a consignment. On clicking “Add consignment” he/she will be able to add the consignment by calculating the consignment cost.

The TekSmart Logistics should calculate the consignment cost based on the following: -

1. An item can contain a maximum of 22 kilograms to be shipped and a consignment’s weight can be a maximum of 88 kilograms.
2. Type of Service
3. Source and destination places where the consignment must be taken from and delivered to.

The item cost is calculated based on the following: -

* The invoice amount for every 100 air miles in the normal service is Rs 20 per kilogram. Express service will cost 2 times the normal rate and the Platinum service will cost 3 times the normal rate.
* The below table contains the total miles for every source and destination.

|  |  |  |
| --- | --- | --- |
| SOURCE | DESTINATION | AIR MILES |
| Mumbai | London | 4500 |
| Mumbai | Singapore | 4000 |
| Mumbai | Sydney | 11000 |
| Delhi | London | 6700 |
| Delhi | Singapore | 4200 |
| Delhi | Sydney | 6500 |
| Bangalore | London | 8050 |
| Bangalore | Singapore | 3200 |
| Bangalore | Sydney | 9360 |

Similarly, the customer can add any number of consignments. Every time after adding the consignment, consignment details like source, destination, shipment address, service type, items details (item name, category and weight), total cost of the consignment should be displayed below. The accumulated cost of all the consignments should be calculated in UI and displayed.

Once adding all the consignment(s), customer clicks on “Confirm Consignment(s)” button which will generate unique ids for each consignment. “Confirm Consignment(s)” button will be enabled only when at least 1 consignment is added.

**Pre-Condition:** The required tables should be created at the backend.

**Post- Condition**: Consignment details will be stored in the database along with the generated consignment number for each consignment, shipment date (current date) and calculated total cost for each consignment.

**Success Scenario**: On success, the customer object containing the list of consignments booked should be returned. The UI application should accept JSON response and display the message “Consignment(s) Booked”, with a table listing the details of the consignment(s) – Source, Destination, Service Type, Shipment Address, Item Details (item name, category and weight), total cost of the consignment. The overall cost of all the consignments should also be displayed.

**Failure Scenario**: The application should display an appropriate error message if any of the validation fails. The below client-side validations should be implemented.

**Business Rules:**

1. All fields are mandatory.
2. The weight in kilograms should have a minimum of 10 kilograms and a maximum of 22 kilograms for an item
3. Only 4 items can be added to a consignment.
4. Customer can add any number of consignments.
5. All the above validations should be performed using HTML5/Angular JS.

**Component Specification (DAO and Model)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package name** | **Class/Interface** | **Attributes** | **Method** |
| com.cargo.model | Consignment(class) | int consignmentNo  String shipmentAddress  String serviceType  Date dateofshipment  double cost  String source  String destination  List<Item> itemList | Include getters and setter method for all the attributes. |
| com.cargo.model | Customer(class) | int accountNo  String customerName  long mobileNo  String emailId  String address  String password  List<Consignment> consignmentList | Include getters and setter method for all the attributes. |
| com.cargo.model | Item (class) | int itemNo  String itemName  String itemCategory  int weight | Include getters and setter method for all the attributes. |
| com.cargo.dao | ConsignmentDAOImpl(class) |  | Customer bookConsignment(Customer customer ) |
| com.cargo.dao | ConsignmentDAO(Interface) |  | Customer bookConsignment (Customer customer ) |

**Service Specification**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Package**  **name** | **Class/Interface** | **Attributes** | **Request**  **Method** | **URI** | **Content type** | **Input** | **Output** |
| com.cargo.service | CargoService(class) | ConsignmentDAO – Should be injected using annotation | POST | /bookconsignment | JSON | Customer Object | Customer Object |

**Microservice Specification:**

Spring boot application containing the REST services-

Application name = **“aircargoservice”** (to be specified in properties file)

**Eureka server client port** to be specified in **application.properties** file.

API gateway server-

Must contain a routing information for /bookconsignment URI that is the mapping to the actual implementation running at port 8080.

The API gate way server port (8085) MUST be specified in the “**application.properties**” file.

*The service must be accessible through API gateway using small-case url.*

**UI Components Specification**

Associated screen no:3

**Flow of events**

View

API Gateway

Service

DAOImpl

Submit Customer JSON Object containing list of Consignment details

Route Customer JSON Object containing list of Consignment details

Book Consignment

Customer Object with Consignment along with consignmentNo

Customer Object with Consignment along with consignmentNo as JSON

Customer Object with Consignment along with consignmentNo as JSON

Enter the Consignment details

Display the Consignment details

1. **View booked consignment**

Once the customer clicks the view consignment details, the customer can view the all the consignments booked by him/her.

**Note:** Since it is a single page application, the consignment details already obtained through booking functionality will be iterated and displayed in the client side.

**Pre-Condition:** Nil

**Post- Condition**: Nil

**Success Scenario**: After the booking is done successfully by the customer, the customer can view the list of consignments booked by him/her.

**Failure Scenario**: Nil

**UI Components Specification**

Associated screen no:4

1. **Customer Login**

Once the customer registers themselves with the TekSmart Logistics, they can login into the application by providing the username (i.e accountno) and password. If the credentials are valid, the customer can book the consignment, update his/her profile and view the consignment booked by the customer.

**Pre-Condition:** The required tables should contain the required data at the backend.

**Post- Condition**: Nil

**Success Scenario**: On success, the customer object is returned. The UI application accepts JSON response and displays the “Customer Home Page”.

**Failure Scenario**: If the username and password does not exist, then a user defined exception “InvalidCredentialException” is thrown and the message “Login unsuccessful” is displayed. The application should display an appropriate error message if any of the validation fails. The below client-side validations should be implemented.

**Business Rules:**

1. All fields are mandatory.
2. All the above validation should be performed using HTML5/Angular JS

**Component Specification (DAO and Model)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package name** | **Class/Interface** | **Attributes** | **Method** |
| com.cargo.model | Customer(class) | int accountNo  String customerName  long mobileNo  String emailId  String address  String password  List<Consignment> bookingList | Include getters and setter method for all the attributes. |
| com.cargo.dao | CustomerDAOImpl(class) |  | Customer validateCustomer(Customer customer ) throws InvalidCredentialException |
| com.cargo.dao | CustomerDAO(Interface) |  | Customer validateCustomer(Customer customer )throws InvalidCredentialException |

**Service Specification**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Package**  **name** | **Class/Interface** | **Attributes** | **Request**  **Method** | **URI** | **Content type** | **Input** | **Output** |
| com.cargo.service | CargoService(class) | CustomerDAO – Should be injected using annotation | POST | /login | JSON | Customer Object | Customer Object |

**Microservice Specification:**

Spring boot application containing the REST services-

Application name = **“aircargoservice”** (to be specified in properties file)

**Eureka server client port** to be specified in **application.properties** file.

API gateway server-

Must contain a routing information for /login URI that is the mapping to the actual implementation running at port 8080.

The API gate way server port (8085) MUST be specified in the “**application.properties**” file.

*The service must be accessible through API gateway using small-case url.*

**UI Components Specification**

Associated screen no:5

**Flow of events**

View

API Gateway

Service

DAOImpl

Submit Customer details as JSON

Route Customer details as JSON

Validate Customer

Customer Object

Customer Object as JSON

Customer Object as JSON

Enter AccNo. Password & Click login

Display homepage of Customer

1. **Reports**

Once the admin logs into the application, the admin can view the following reports:

* 1. Consignment report based on airport and date
  2. Consolidated Consignment for all airport for specific period
  3. Revenue report of an airport for a specific period

Admin username and password will not be persisted in the database, the username and password for admin will be “admin” and “Teksmartadmin” respectively. The validation for the login will be taken care at the UI level. On successful login admin will be redirected to the admin home page, where the 3 options for report generation will be available.

**Pre-Condition:** The required tables should contain the required data at the backend.

**Post- Condition**: Should return the list of consignment for last two years.

**Success Scenario**: On success, the list of consignment object should be returned. The UI application should accept JSON response and display the “consignment details”.

**Failure Scenario:** The application should display an appropriate error message if any of the validation fails. The below client-side validations should be implemented.

**Business Rules:**

1. All fields are mandatory (for individual reports).
2. All the above validation should be performed using HTML5/Angular JS.

**Component Specification (DAO and Model)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package name** | **Class/Interface** | **Attributes** | **Method** |
| com.cargo.model | Consignment(class) | int consignmentNo  String shipmentAddress  String serviceType  Date dateofshipment  double cost  String source  String destination  List<Item> itemList | Include getters and setter method for all the attributes. |
| com.cargo.dao | ConsignmentDAOImpl(class) |  | List<Consignment> genarateReport( ) |
| com.cargo.dao | ConsignmentDAO(Interface) |  | List<Consignment> genarateReport( ) |

**Service Specification**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Package**  **name** | **Class/Interface** | **Attributes** | **Request**  **Method** | **URI** | **Content type** | **Input** | **Output** |
| com.cargo.service | CargoService(class) | CosignmentDAO – Should be injected using annotation | GET | /reports | JSON | Nil | List<Consignment> |

**Microservice Specification:**

Spring boot application containing the REST services-

Application name = **“aircargoservice”** (to be specified in properties file)

**Eureka server client port** to be specified in **application.properties** file.

API gateway server-

Must contain a routing information for /reports URI that is the mapping to the actual implementation running at port 8080.

The API gate way server port (8085) MUST be specified in the “**application.properties**” file.

**Note:** The list of consignments is already retrieved by clicking on the link “View Reports”. Since it is a single page application, the individual report (a,b,c) should fetch the corresponding details from the already retrieved consignment list based on the specific inputs given for each report.

*The service must be accessible through API gateway using small-case url.*

**UI Components Specification**

Associated screen no:6

**Flow of events**

View

API Gateway

Service

DAOImpl

Submit Request

Route Request

Generate Report Data

Consignment list

Consignment list as JSON

Consignment list as JSON

Check the View Report link

Report Page

**Unit Testing**

* Developer is expected to perform unit test for all the business scenarios.
* Use Spring Test framework for testing the application.

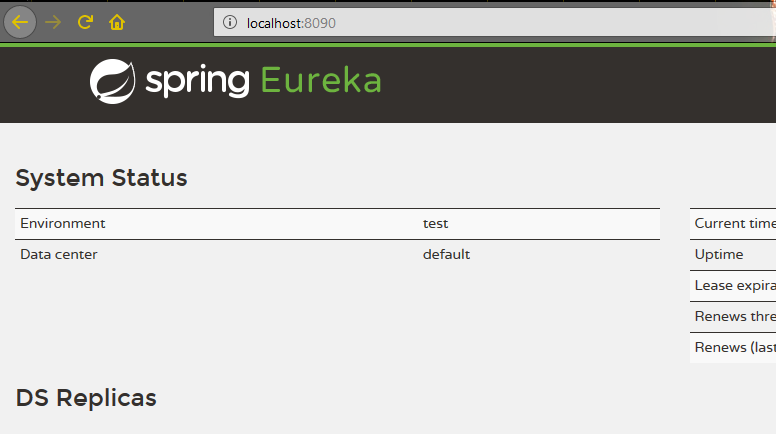
**Microservices implementation Guidelines**

For Microservices implementation, we will be implementing 3 projects.

* Spring Cloud (Eureka-Server) for service registry and discovery
* Spring Boot for REST service creation
* Spring Cloud gateway for API Gateway

**Project 1:**

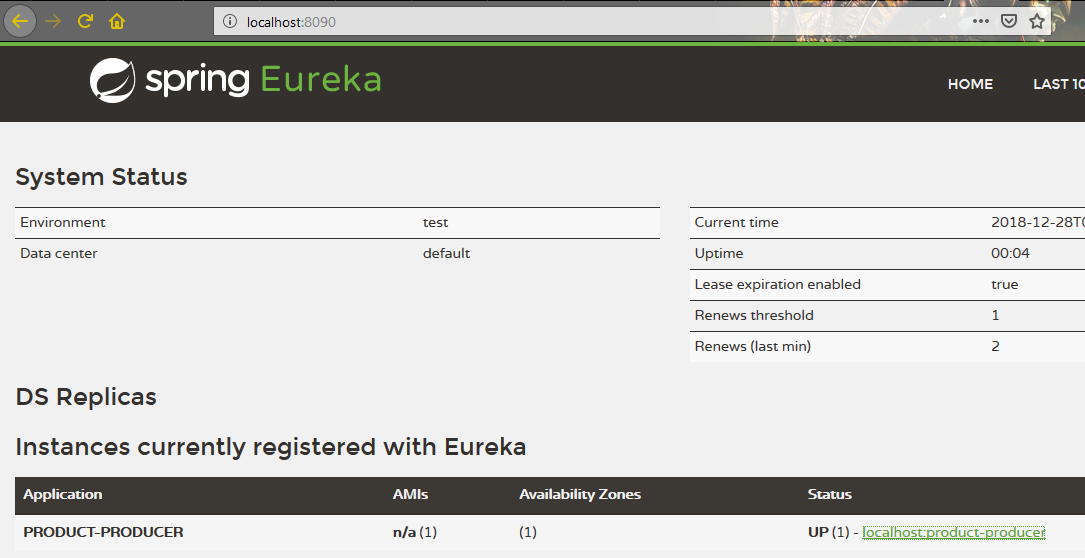
This project should be the Eureka-Server acting as the registry. Run this server in port 8090.



**Project 2:**

This project should be a spring boot application containing all the REST services. These services must be registered in Eureka-Server. Run the services in port 8080.

After the service is registered,



**Project 3:**

This must be a Spring cloud gateway project that contains the routing implementation to the actual services running in port 8080. Run this in port 8085.

port :8085

AirCargoRouter

POST/registerCustomer

port :8080 port: 8090

Registry and Discovery

AirCargoService

**Design Constraints**

1. **The project must be developed from scratch and the project name, file name, package name, class name, along with the methods and attributes, and DB table names must be coded as per the design constraints. On violating the above, the solution should not be considered for auto evaluation.**
2. Follow the coding standards to develop the solution.
3. Use logger API to log the transaction details.
4. The reports will be generated only for last two years.
5. If there are any exceptions, the user-defined exception should be thrown.
6. Use Maven to build the project and create WAR file. Use the correct dependency as specified in the software lab requirement.
7. Technical design of web application has to be carried out by the developer based on the business requirements.
8. Do not change the packagename/attributename/methodname/classname specified in the design document. You can include any new attributes/methods/classes.
9. Developer will create 4 applications:
   1. Angular/HTML5/bootstrap/jQuery (**Name of the application Should be AirCargoClient**)
   2. Spring Boot (**Name of the application Should be AirCargoService**)
   3. Spring Cloud (Eureka-Server, spring-cloud-starter-eureka-server) (**Name of the application Should be AirCargoRegistry**)
   4. Spring Cloud API GateWay(spring-cloud-starter-gateway) **(Name of the application Should be AirCargoRouter**)
10. CORS related issues need to be handled at the application level.

**Lab Requirement**

