Microservices Developer Exercise

The following exercise has been designed to assess the candidate on his/her approach to develop a small application using Spring Boot, REST, J2EE and UI (HTML, CSS, JSP and Javascript) knowledge. Below is the summary of the tasks required to complete this exercise. Please note that the scenario below is fictional and does not reflect the API of any real-world business.

**Instructions**

Contents

[1 Problem Description 2](#_Toc484523602)

[1.1 Estimated Exercise Duration 2](#_Toc484523603)

[2 Version Control and Submission 2](#_Toc484523604)

[3 Sequence Diagram 3](#_Toc484523605)

[4 Example Environment 3](#_Toc484523606)

[5 Expected Outcomes 4](#_Toc484523607)

[5.1 UI Screens 4](#_Toc484523608)

[5.1.1 Login Screen 4](#_Toc484523609)

[5.1.2 Retrieve Discounts Screen 5](#_Toc484523610)

[5.2 If Total Exp <3.5 6](#_Toc484523611)

[5.3 If Total Exp >3.5 and <7 6](#_Toc484523612)

[5.4 If Total Exp >=7 6](#_Toc484523613)

[6 Unit Testing 6](#_Toc484523614)

[7 Non-functional Considerations 6](#_Toc484523615)

[8 Backend API Documentation 7](#_Toc484523616)

[8.1 Get Customer API 7](#_Toc484523617)

# Problem Description

A new online retailer is working on a small application for catering to his products. Given the short timeframe for the assignment, your task is to build a small web application that will cater to this requirement and display the discount details offered on each product

The task will include setting up a new Java Maven project to the specification below. Ideally, we would prefer the application to be developed using Spring Boot or NodeJS and UI functionality implemented using HTML, CSS and Javascript & you need to make a service call out to the *GetCustomerAPI* to retrieve a list of discounts and then make a transformation on the data into the required format for displaying on UI Layer.

Develop the application preferably using Spring Boot or Play Framework. API’s to be built as lightweight container (Docker or any other) microservice Restful/JSON API’s. For backend no-sql databases such as MongoDB (open source) is preferred. Expose the service using API gateway such as APIGEE.

If you have CICD experience, use continuous build & Continuous deployment tools of your choice to create automated CICD. Otherwise manual build and deploy is ok for running this API.

## Estimated Exercise Duration

Expected duration is somewhere between 6-8 hours be spent on this exercise. Feel free to use TODO comments (e.g. // TODO) where there might be scenarios or features that could be implemented in the future. Alternatively, if you run out of time, please use TODO comments to indicate the pieces of work that weren't able to be finished. This way, we can understand what you were hoping to achieve, even if it couldn't be completed in the timeframe.

# Version Control and Submission

1. Open InfyGIT portal: <http://infosysgit/gitblit/>
2. Login with your Infosys credentials for your User ID to be created in GIT.
3. Raise AHD for InfyGIT access and create a repository name TEOAOCP1\_*<EMP ID>\_*Stack1

Attached the AHD request format.



1. Once the request is handled by Tools team. You will receive the clone url in email from tools team (Charan/Raghavendra)
2. Please run below commands to clone and code check-in.

* Clone the Repository: Run the command from command prompt.

\>git clone <<clone\_url\_shared\_by\_tools\_team>>

* You will see a subfolder (Stack1\_<<your\_employeeId>>) created in the path you ran the command. This is your repository.
* Copy your completed assignment project folder / source code in the repository.

Sample GIT commands

* Run below command to see the new / modified files

\> git status

* Run below command to add the files to stage

\> git add <<filesnames>>

* Run below command to commit the changes locally

\>git commit –m “<<commit comments describing the changes you did to the code for reference/tracking purposes.>>”

* Run below command to push the changes to central/origin repository.

\>git push origin master

**Questions?**

Please email or get in touch with the respective PM/TArch/TL who has given you the assignment

# Sequence Diagram



# Example Environment

A running example of all the discounts available can be seen at the following endpoint. The task is to consume this service and display the relevant details on the UI layer.

**Host: 172.21.115.8**

**Port: 8080**

**Retrieve Discounts (no filters):** *http://172.21.115.8:8080/getcustomer/discounts.json*

# Expected Outcomes

1. Follow the required coding standards during implementation.
2. Ensure error handing is done properly and all the scenarios are covered.
3. Ensure testing coverage > 80%, write proper JUnit test cases to cover the required scenarios.
4. Include application logging - DEBUG,INFO,ERROR
5. Ensure your code is properly committed and pushed to central GIT repository with proper comments. Provide the required access as mentioned under Section 2

## UI Screens

### Login Screen

Build a basic login screen. A valid user will navigate to the retrieve discounts screen and is allowed to search discounts based on Product Id.

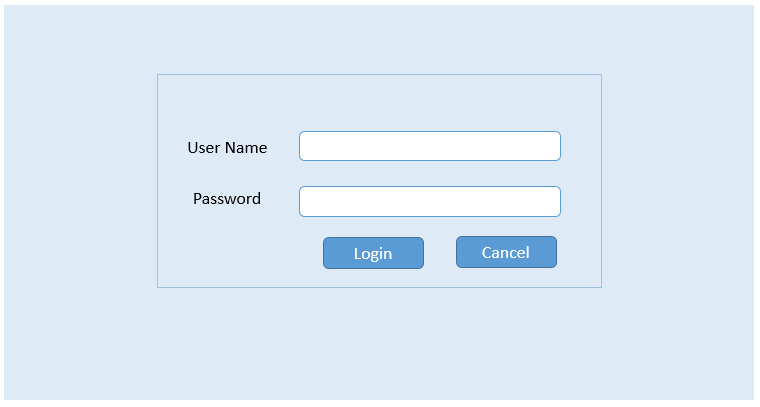
Implement the following validations on User name & Password fields

* 1. User name & password should not empty or null value
  2. Check a user name allows characters and special character hyphen '-'
  3. Check a password between 8 to 15 characters which contain at least one lowercase letter, One uppercase letter, one numeric digit, and one special character
  4. Use a default password and for an Invalid user display ‘Invalid User’ on the login screen itself.

Example:

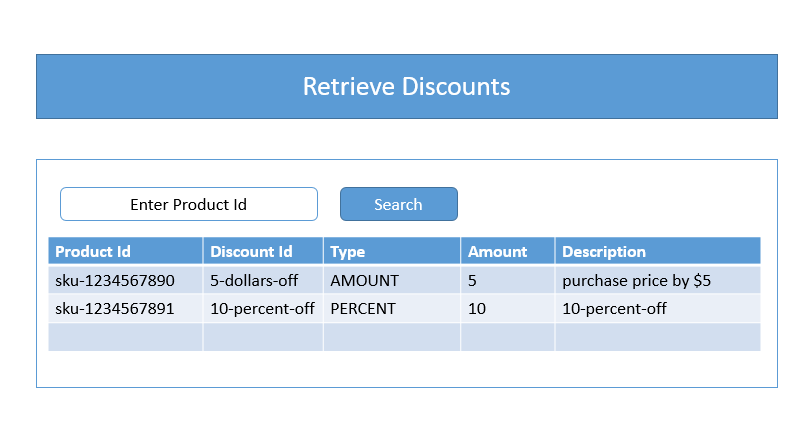
**Valid user:** qa-test-user

**Invalid user:** bad-user



### Retrieve Discounts Screen

Valid user will navigate to this screen and by default discounts on all the products are displayed. It also allows the user to search the discounts based on a Product Id.



Implement the following validations on Search Text box field.

* Search text should not empty or null value
* It should not allow special characters except hyphen ‘-’

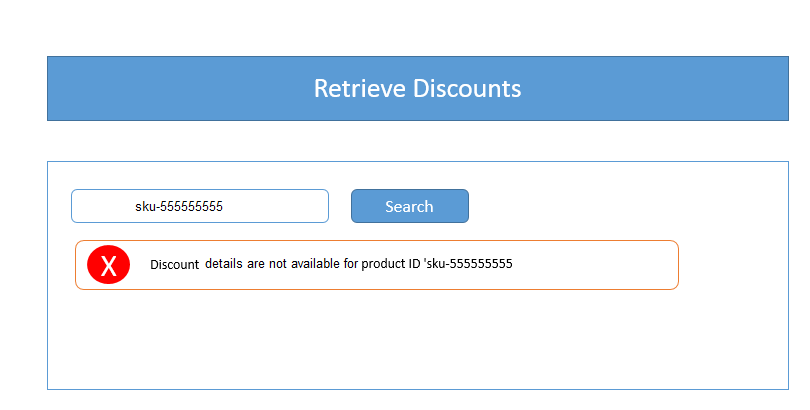
#### Error screen:

User trying to retrieve the discounts details using invalid product Id, then it will show the following error message to user.

Example:

**Product Id:** sku-555555555

**Error message:** Discount details not available for Product ID 'sku-555555555’



## If Total Exp <3.5

In addition to the above following items need to be covered

1. Handle multiple submits i.e. search button should not submit the request twice.
2. Use an AJAX request to asynchronously load and display system time somewhere at the top of the results screen.

Do not be limited by your experience level and if you are aware and confident of implementing the below items within the stipulated timeframe. *Please do so!!!*

## If Total Exp >3.5 and <7

In addition to the above following items need to be covered

1. Use a properties file for validation of user name and password. This file will have admin and user roles. Ex: “super-user” ,”normal-user”
2. Use the standard Spring Security features for managing users and roles.

Your registered users should have at least one of these roles: user and administrator.

## If Total Exp >=7

1. Implement a file upload feature that would add 3 more records to the original response and display on the screen.
2. Implement pagination by displaying only 5 product details per page in default view.

# Unit Testing

Unit tests are to be written using JUnit, pulled in as a Maven dependency. Regarding test coverage, we would be happy with the following:

* A unit test showing the discounts on all the products.
* A unit test showing discounts for a valid productid.
* A unit test showing an error condition, for example a productid not found in the system.
* A unit test showing an error condition, for example a user not found in the system.

# Non-functional Considerations

Consideration should be given to the following areas:

* Configuration - How will you store configuration that might vary in different environments, for example the endpoint for the backend API call. Are there frameworks that make this easy for us out of the box?
* How often would we call the backend API? Would we hit it every time, or would we think about caching the response for a period of time? *NB: We don't expect you to build a caching solution, simply think about it and we can discuss afterwards.*

# Backend API Documentation

## Get Customer API

The Get customer API is an existing service that has been deployed into the cloud. This service will provide all of the information about a given customer, including their discounts. This API must be consumed and the eligible discounts information must be extracted, transformed, filtered (if necessary) and returned.

Example - Successful customer retrieval: <http://172.21.115.8:8080/getcustomer/discounts.json>

**Example Response:**

{

"uuid": "qa-test-user",

"name": "Test User",

"address": "1 Main St",

"eligibleDiscounts": [

{

"discountId": "10-percent-off",

"type": "PERCENT",

"description": "Reduce the purchase price by 10%",

"amount": 10

},

{

"discountId": "5-dollars-off",

"type": "AMOUNT",

"description": "Reduce the purchase price by $5",

"amount": 5,

"productId": "sku-1234567890"

}

],

"products": [

{

"productId": "sku-1234567890",

"description": "12 month subscription to 'Horse and Hound'",

"originalPrice": 100,

"finalPrice": 50,

"discountInformation": {

"discountId": "50-percent-off",

"type": "PERCENT",

"description": "Reduce the purchase price by 50%",

"amount": 50,

"productId": "sku-1234567890" }

}

]

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Mandatory?** | **Example Value** | **Comments** |
| uuid | String | Y | qa-test-user | Represents the ID of a user in the system. |
| name | String | Y | Test User | The customer's full name. |
| address | String | Y | 1 Main St | The customer's home address. |
| eligibleDiscounts | List<Discount> | N | See Discount. | An optional list of discounts stored against the customer in the database |
| products | List<PurchasedProduct> | N | See PurchasedProduct | An optional list of purchased products stored against the customer in the database. See below for PurchasedProduct field definition. |

