Server Side Foundation Assessment

Date: Friday March 3 2023
Assessment Time: 0900 - 1700 (including meal breaks)

Overview

In this assessment you will be writing a Spring Boot application to calculate the total price of a purchase order.

There are **5 tasks** in this assessment. Complete all tasks.

Passing mark is **65%** (**90 marks**). Total marks is **138**.

Read this entire document before attempting the assessment. There are 11 pages in this document.

Application Overview

This simple e-commerce application allows users to create purchase orders and to place purchase orders into the ordering system. The ordering system will then lookup the prices of the item from an external quotation system, calculate the total cost and return the invoice of the order to the user.

The following diagram shows the client side flow.

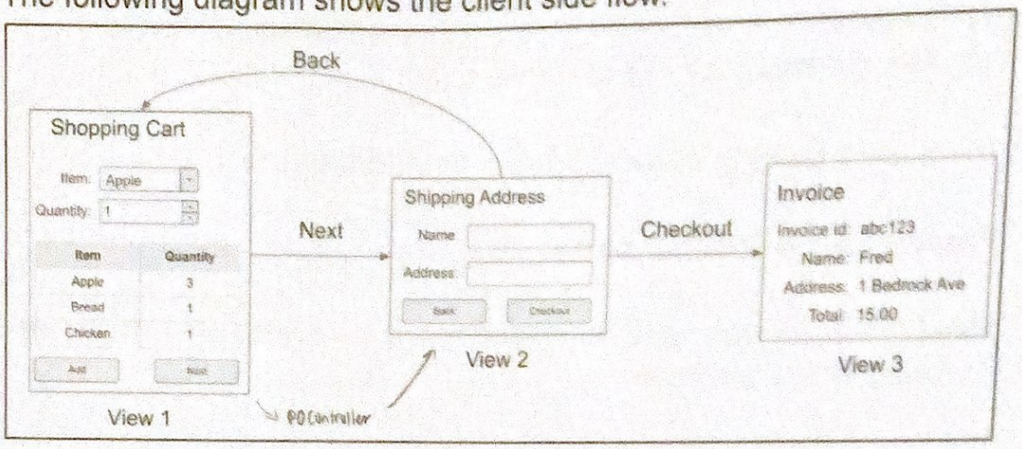

Figure 1 Application flow

The client consists of 3 views/pages. They are

1. **View 1** - add items to an order; when the Next button is pressed, View 2 will be displayed
2. **View 2** - allows the customer to enter the order's shipping address; pressing the Checkout button, will complete the processing and the invoice will be displayed in View 3
3. **View 3** - displays the order's invoice

Further details of the views will be provided in their respective tasks.

The following shows the entire application architecture

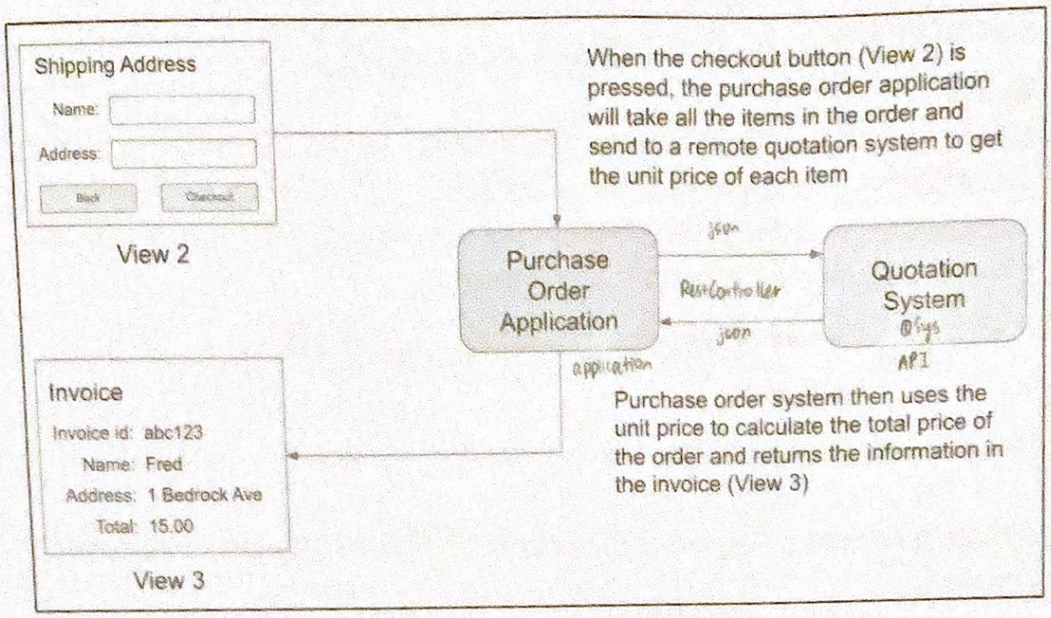

Figure 2

1. After filling in the purchase order (View 1, not shown in Figure 2) and the shipping address (View 2), the order will be submitted to the Purchase Order Application (POApp). This is the application (POApp) that you will be developing in this assessment.
2. The POApp extracts all the items from the submitted purchase order and forwards them to an external Quotation System (QSys) to get the unit price of the items in the purchase order
3. When the quotes are returned from QSys, POApp uses the information to calculate the total cost of the purchase order.
4. POApp then returns the invoice back to the client (View 3)

Assessment

Setup

Generate a Spring Boot Maven project using Java 19. In addition to the usual Spring Boot dependencies associated with a Spring Boot web application, you should also include the following dependencies and any other dependencies you think is necessary

- JSOP-P
- Validation

You are free to add additional dependencies to the `pom.xml` file.

Create a remote git repository. Add/link the Maven project to a remote git repository.

You should now perform an initial commit and push of your project to the remote repository. Do not wait until the end of the assessment.

Your remote Github repository must be a **PRIVATE** repository. Make your repository **PUBLIC** after **1700 Friday Mar 3 2023** so that the instructors can access your work.

IMPORTANT: your assessment repository is PRIVATE and should only be accessible to yourself and nobody else during the duration of the assessment. It should only be public **AFTER 1700 Friday Mar 3 2023**. If your work is plagiarised by others before the end of the assessment, you will be considered as a willing party in the aiding and abetting of the dishonest act.

Task 1 (56 marks)

For this task use the provided `view1.html` file. Feel free to refactor the file to perform this task. However, do not add or remove any items from `options`.

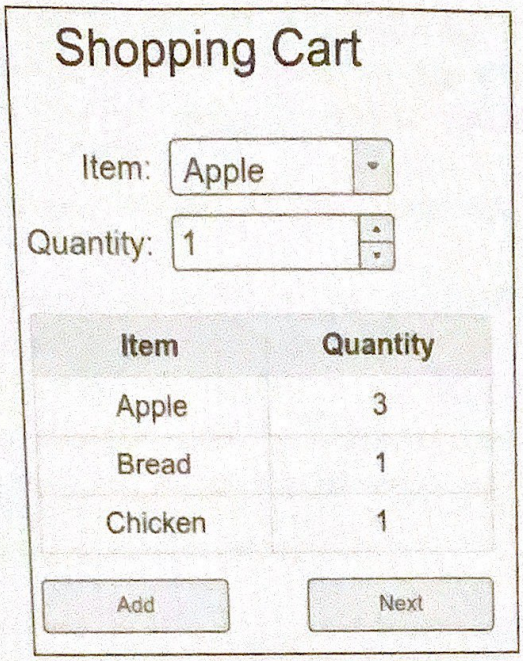

Figure 3 View 1

View 1 is the POApp's application 'landing page', viz. the first view when a customer opens the site.

View 1 allows a customer to add items into their cart cart by selecting the item and the quantity from the input boxes.

A table shows the current items and quantities in a cart. For simplicity, items cannot be removed from the cart.

You are to implement the following behaviour for View 1.

Adding an item into the cart

A customer will select an item from the pull down list and enter a quantity. The Add button will add the item to the customer's cart. After adding the item to the cart, the controller redisplays View 1 with the added item in the table.

Similar items should be aggregated; for example if a customer adds apples twice, the first time the quantity is 2 and the second, the quantity is 3, then View 1 should only show 1 apple entry with a quantity of 5.

The controller should also perform the following checks when an item is added into the cart

- The item must be one of those in the `option` (`view1.html`). If it is not, do not add the item to the cart; display the message '`We do not stock <item name>`' in View 1.
- The quantity must be specified and must be greater than 0. If it is not, do not add the item to the cart; display the message '`You must add at least 1 item`' in View 1.

Empty cart

If the customer's cart is empty, display the message '`Your cart is currently empty`' instead of the table.

Create a class called `PurchaseOrderController` in the package of your choice to serve as the controller for View 1.

Task 2 (37 marks)

When the Next button is pressed in View 1, it will make the following HTTP request to transition to View 2

```
GET /shippingaddress
```

View 2 (see Figure 4) allows the customer to enter the delivery details.

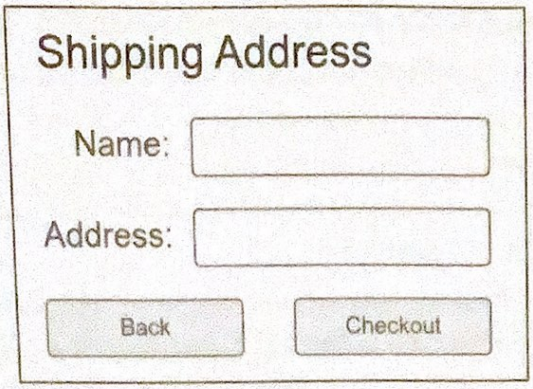

Figure 4 View 2

After filling the delivery details, the customer will proceed to checkout (View 3) by pressing the Checkout button.

All the input fields in View 2 are mandatory. The name must be at least 2 characters long. If the conditions are not met, display an appropriate error message back in View 2.

validation

If a customer attempts to navigate to View 2 without a valid cart (either the cart is missing or there are no items in the cart), the controller should redisplay View 1.

A Back button redisplays View 1 with all the existing cart's contents.

Write View 2 in a file called `view2.html`.

Reuse the `PurchaseOrderController` to implement the controller behaviour described in Task 2.

Task 3 (22 marks)

When the Checkout button (View 2) is pressed, the controller will make a HTTP call to a REST endpoint (QSys) to get a quotation id and the unit price of all the items in the customer's cart.

The call to QSys should be abstracted in a Spring Boot service. Create a class called `QuotationService`; add a method with the following signature

```
public Quotation getQuotations(List<String> items)
        throws Exception
```

where the `items` parameter is the list of items from the customer's cart. Use the `Quotation` class provided with the assessment assets ZIP file.

`getQuotations()` makes a HTTP call to QSys with the items in a JSON array as payload to the Qsys REST endpoint at `https://quotation.chuklee.com`. Figure 5 shows the HTTP exchange between `getQuotations()` and QSys

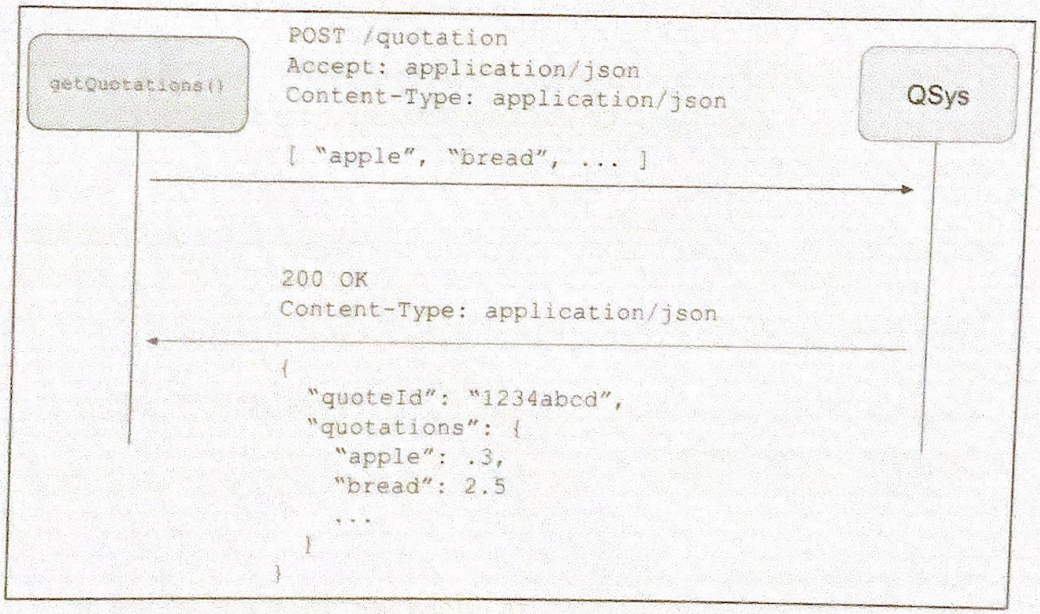

Figure 5 HTTP exchange

If the request is valid, QSys will return a JSON object with the following attributes

- `quoteId` a unique id generated by QSys

- `quotations` which contain the unit price of the items you requested. For example, if you pass into `getQuotations()` a list containing the items apple and bread, then quotations will contain the unit prices of these 2 items. The item name will be the key and the unit price will be the value under the `quotations` attribute

Marshal the returned JSON object into an instance of `Quotation` class and return the instance (to the caller).

Note that the HTTP request may fail. In that case, the REST endpoint will return an error object of the following structure

```
{ "error": "error message" }
```

Your `getQuotation()` method should throw an exception with the error message.

Task 4 (18 marks)

Use the `QuotationService.getQuotations()` you have implemented in Task 3 in `PurchaseOrderController` to perform the checkout.

Calculate the total price of the cart using the unit price from `Quotation`. For example if the purchase order contains 3 apples and 1 loaf of bread then the total cost of the purchase order will be

(3 * .3) + (1 * 2.5) = 3.4

assuming that the unit price of apple is .3 and bread is 2.5.

Once you have calculated the total cost, display the information in View 3 (see Figure 6) and clear the contents of the customer's cart.

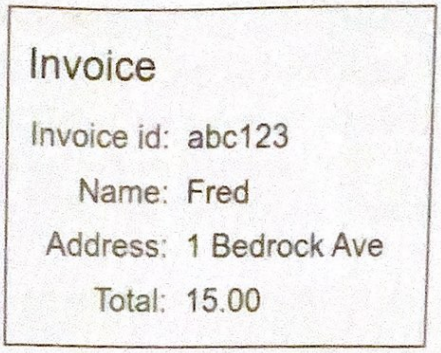

Figure 6 View 3

If the QSys returns a quotation error, then the error should be displayed in View 2 along with the previously filled name and address.

Write View 3 in a file called `view3.html`.

Task 5 (5 marks)
Deploy the application to Railway. The deployment should be based on any commits on or before **1700 Mar 3 2023**. Do not undeploy your application before **2359 Friday Mar 17 2023**.

For JDK19 deployment, add the environment variable `NIXPACKS_JDK_VERSION` to your Railway service; set the value of `NIXPACKS_JDK_VERSION` to 19. See https://nixpacks.com/docs/providers/java

Submission

You must submit your assessment by pushing it to your repository at either GitHub, GitLab or BitBucket.
Only commits on or before **1700 Mar 3 2023** will be accepted. Any commits after **1700 Mar 3 2023** will not be accepted. No other form of submission will be accepted (eg. ZIP file).

Remember to **make your repository public after 1700 Mar 3 2023** so the instructors can review your submission.

After you have committed your work, post the following information to Slack channel `#02-ssf-submission`

1. Your name (as it appears in your NRIC)
2. Your email
3. Your deployed application's URL on Railway
4. Git repository URL

It is your responsibility to ensure that all the above submission requirements are met. Your assessment submission will not be accepted if

1. any of the 4 items mentioned above is missing, or
2. you information did not comply with the submission requirements eg. not providing your full name as per your NRIC, incorrect email subject line, forgetting to post the Railway deployment URL, etc. and/or
3. the repository is not public after **1700 Friday Mar 3 2023**

Academic Integrity

This is an open book assessment. You may search the Internet for resources or use reference books during the assessment. The assessment must be your own work. You cannot ask a third party to write any part of this assessment or use AI tools such as ChatGPT to generate output and submit it as part of your assessment. This will result in an automatic disqualification from the assessment.

The NUS ISS takes a strict view of cheating in any form, deceptive fabrication, plagiarism and violation of intellectual property and copyright laws. Any student who is found to have engaged in such misconduct will be subject to disciplinary action by NUS ISS.

You are to ensure the integrity and working condition of your PC/notebooks (eg. wireless/internet connection, battery, screen, accidents like water spillage) during the assessment. NUS ISS will not accept any of these as a reason for deferring or retaking your assessment. accept any of these as a reason for deferring or retaking your assessment.