

Client Side Foundation Assessment

Date: Friday September 30 2022

Assessment Time: 0900 - 1715 (including meal breaks)

Overview

There are **7 tasks** in this assessment. Complete all tasks.

Passing mark is **65% (81 marks)**. Total marks is **125**.

Read this entire document before attempting the assessment. There are 10 pages in this document.

Assessment Setup

You will be given an assessment project template (ZIP file) for this assessment. Unzip this project template; a directory called `vttp2022_csf_assessment` will be created. In the said directory, you will find a partially completed Angular and a partially completed SpringBoot application. They are under the `pizza-storefront` and `order-backend` directory respectively.

All the necessary dependencies have been added to the respective projects. You are free to add additional libraries if you wish to do so.

Create a git repository for the assessment. This repository must initially be a **PRIVATE** repository. Click on the 'Private' radio button when you create the repository.

Once you have set up your repository, you should commit and push your assessment directory (`vttp2022_csf_assessment`). Do not wait until the end of the assessment.

Make your repository **PUBLIC** after **1715 Friday Sept 30 2022** so that the instructors can access your work.

IMPORTANT: your assessment repository is PRIVATE and should only be accessible to yourself and nobody during the duration of the assessment and is only public **AFTER 1715 Friday Sept 30 2022**. 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.

Assessment

In this assessment, you will complete an online Pizza ordering application. The application consists of an Angular frontend (pizza-storefront) and a SpringBoot backend (order-backend).

The Angular frontend is a SPA (Single Page Application) consisting of 2 views. The flow of the views is shown in the following diagram

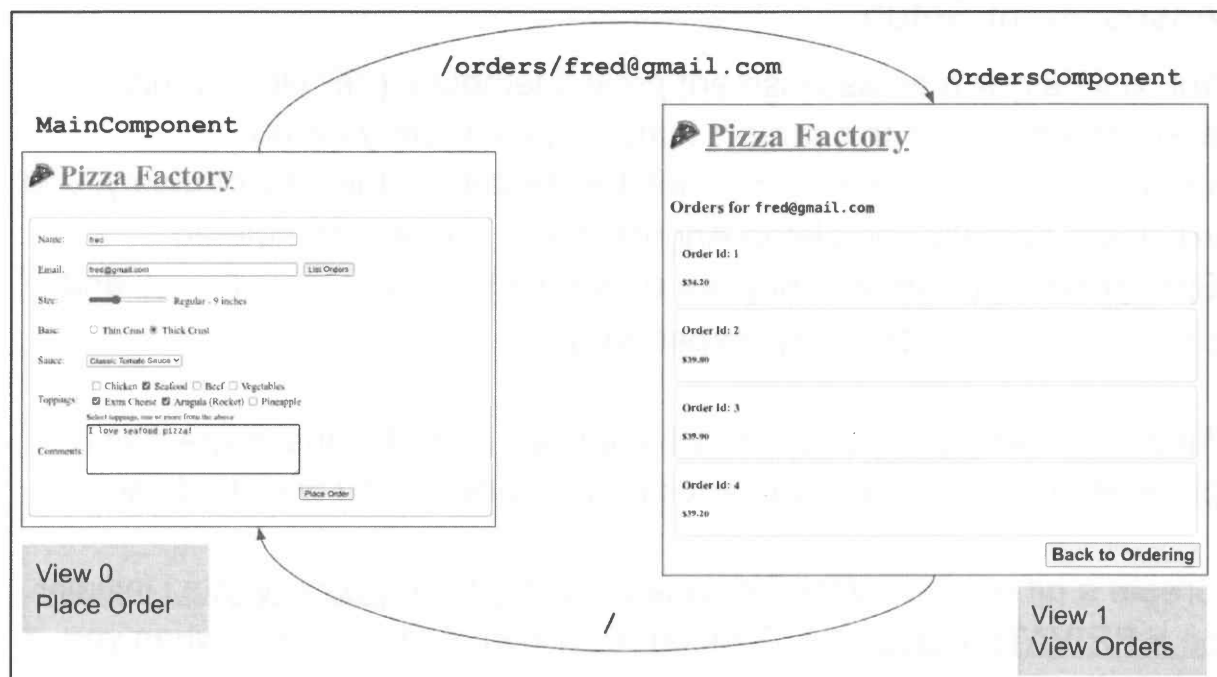


Figure 1 Frontend views

Users can navigate between the 2 views. View 0 is used to place orders for pizza and View 1 allows a user to see his/her orders.

These 2 views make HTTP requests to the SpringBoot application to place a new order and to get a list of existing orders.

In order to minimise cost, the deployment strategy is to have the Angular application served by the backend SpringBoot application (same origin).

Read all the required tasks first before attempting the assessment.

Task 1 (10 Marks)

The current Angular frontend is not a SPA. Refactor the application to a SPA by adding a new component called `OrdersComponent` in the `components` directory.

The following table specifies the SPA routes

Route	Component
The first view when the frontend is loaded	MainComponent
/orders/<email>, where <email> is dynamic	OrdersComponent
Any other routes besides the above 2	MainComponent

Task 2 (30 marks)

The HTML for View 0 has been provided (see `main.components.html`). You cannot modify the provided HTML except when you are adding Angular's form directives. See Figure 2.

View 0 has the following 2 use cases:

1. Placing a new order by filling the form and clicking on the 'Place Order' button
2. List all orders by entering the email and clicking on the 'List Orders' button.


Implement the `MainComponent` according to the following requirements.

Use case 1: Placing a New Order

To place a new order, fill in the form according to the below requirements

- Name - **mandatory**, your name as it appears in your NRIC. Your assessment submission will be marked with Name and Email field
- Email - **mandatory**, your email as it appears in LumiNUS
- Pizza size - **mandatory**
- Base - **mandatory**, pizza base
- Sauce - **mandatory**, select one from the drop down list
- Toppings - **mandatory**, select one or more for the provided list of toppings
- Comments - **optional**, any comment accompanying the order



 **Pizza Factory.**

Name:

Email:

Size: ☒ Personal - 6 inches

Base: ☐ Thin Crust ☐ Thick Crust

Sauce:

Toppings: ☐ Chicken ☐ Seafood ☐ Beef ☐ Vegetables
☐ Extra Cheese ☐ Arugula (Rocket) ☐ Pineapple } *FormArray?*

Select toppings, one or more from the above

Comments:

Figure 2 View 0 - Place an Order

The 'Place Order' button should be disabled and only enabled when all the mandatory fields are filled.

what qualifies as successful? record successfully inserted into DB?

If the pizza order is created successfully, transition to View 1 to list the new order and other previous orders. Use the email from the order you have just created for routing; for example, if you have just placed an order with fred@gmail.com, then you should transition to View 1 with fred@gmail.com.

New orders are created with the `PizzaService.createOrder()`. See **Task 3** for more details.

Note that when you create an order, you must use your name as it appears in NRIC and your LumiNUS registered email.

Use case 2: List all Orders

The form can also be used to query a list of orders. When a valid email (format) is entered in the Email field, the 'List Orders' button should be enabled.

Transition to View 1 with the entered email when the 'List Orders' button is pressed.

The 'List Orders' button should be disabled when the Email field is empty or contains an invalid email (format).



Task 3 (20 marks)

The function for interacting with the SpringBoot is implemented in `PizzaService (pizza.service.ts)` class. Implement the 2 given methods

`createOrder()` - creates a new order. Call this method when the 'Place Order' button (View 0) is pressed; makes the following HTTP request to the SpringBoot backend to fulfil the order

POST /api/order

Select an appropriate payload encoding (media type) for the order details.

`getOrders()` - Returns a list of order summaries for a given email. Call this method with then 'List Orders' button (View 0) is pressed; makes the following HTTP request to the SpringBoot backend to retrieve the order summaries

GET /api/order/<email>/all

where <email> is all the orders that were created under that email.

Task 4 (15 marks)



 **Pizza Factory**

Orders for fred@gmail.com

Order Id: 1
\$34.20

Order Id: 2
\$39.80

Order Id: 3
\$39.90

Order Id: 4
\$39.20

Back to Ordering

Figure 3 View 1 - List Orders

Use the `OrderComponent` (created in **Task 1**) for View 1 (List Orders).

View 1 must display the following information

- Email of the orders you are listing
- List of all the orders for a given email. Every order should display the following detail
 - Order id
 - Total cost of the order; see **Task 6**
- 'Back' button to return to View 0

An example of View 1 is shown in Figure 3. You are free to layout/style View 1.

Use `PizzaService.getOrders()` from **Task 3** to retrieve the list of orders from the backend and populate View 1.

If there are no orders for a particular email, you should display an appropriate message

Task 5 (5 marks)

A MySQL database has been deployed. Use the below details to configure the SpringBoot backend to use the database:

Server	157.245.202.70 (Do not use NUS_Guest to connect. Use the wireless provided by the lecturer)
Port	3306
Database name	pizzafactory
User	fred
Password	yabadabadoo

Remember not to include the database password in `application.properties` file or hard coded in your application. All properties in `application.properties` can be set with environment variables.

Also note that user `fred` has been given permission to only perform select and insert operations.

The database contains 1 table called `orders`. The `orders` table has the following columns

Column Name	Description
<code>order_id</code>	Unique identifier representing the order id
<code>name</code> <code>string</code>	Value from the name control, View 0
<code>email</code> <code>string</code>	Value from the email control, View 0
<code>pizza_size</code> <code>integer</code>	Value from the size control, View 0
<code>thick_crust</code> <code>boolean</code>	true if the Thick Crust is selected; false otherwise, View 0
<code>sauce</code> <code>string</code>	Value from the size control, View 0
<code>toppings</code> <code>List<string></code>	A comma separated list of all selected toppings, View 0. Eg if seafood, cheese and arugula was selected, then form a comma separated string of the 3 values 'seafood,cheese,arugula' (without the quotes)
<code>comments</code> <code>string</code>	Value from comments control, View 0

The `orders` schema can be found in `schema.sql` in the assessment project template directory.

Task 6 (35 marks)

Use the provided class `OrderRestController` to process the REST request from the Angular frontend. Write request handlers to handle the following HTTP requests:

`Order []`

POST /api/order

Create a new order from the payload. The new order should be inserted into the `orders` table; implement the create order logic in `OrderService.createOrder()`. Use appropriate status code for the response.

GET /api/order/<email>/all

Return a list of orders from the given email. Implement the logic to retrieve the orders in `OrderService.getOrdersByEmail()`. Use appropriate status code for the response.

To calculate the total cost of an order, use the provided `PricingService`.

Task 7 (10 marks)

Deploy your assessment to Heroku. The Angular frontend should be served from the SpringBoot backend.

The database password should not be exposed either in `application.properties` or hard coded in the source code. Marks will be deducted if they are exposed.

Do not undeploy your application from Heroku until after **0000 (12AM) Saturday Oct 15 2022**.

Submission

You must submit your assessment by pushing it to your repository at GitHub.

Only commits on or before **1715 Sept 30 2022** will be accepted. Any commits after **1715 Sept 30 2022** will not be accepted. No other form of submission will be accepted (eg. ZIP file).

Your Heroku deployment (**Task 7**) must be from a commit of your repo on or before **1715 Sept 30 2022**.

Remember to **make your repository public after 1715 Sept 30 2022** so the instructors can review your submission.

After committing your work, post the following information to Slack channel #04-csf-submission

1. Your name (as it is shown in your NRIC)
2. Your email (as your LumiNUS)
3. Git repository URL. Remember to make it public after **1715 Sept 30 2022**.
4. Heroku deployment URL. The Heroku deployment URL ends with herokuapp.com. Please do not undeploy your application from Heroku until after **0000 (12AM) Saturday Oct 15 2022**

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 from #04-csf-submission channel, and/or
2. Your information did not comply with the submission requirements eg. not providing your full name as per your NRIC, and/or
3. Your repository is not publicly accessible **after 1715 Sept 30 2022**

You should post the submission information to the Slack channel #04-csf-submission **no later than 1730 Sept 30 2022**.

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. This will result in an automatic failure.

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.