Client Side Foundation Assessment

Date: Friday February 24 2023
Assessment Time: 0900 - 1700 (including meal breaks)

Overview

In this assessment, you will be writing a 'full stack' application to view and comment on restaurants from a Mongo database.

The backend is written in Spring Boot and the front end is written in Angular.

There are **6 tasks** in this assessment. Complete all tasks.

Passing mark is **65%** (**162 marks**). Total marks is **250**.

Read this entire document before attempting the assessment. There are 14 pages in this document.

Application Overview

This application allows users to search restaurants based on the type of cuisine. It also allows users to rate a restaurant.

The restaurant data is queried from a Mongo database.

The Angular application consists of 3 views/pages. They are

1. **View 1** - list all the cuisines from the database.
2. **View 2** - list of all the restaurant for a specific cuisine
3. **View 3** - details for a restaurant including a map. This view also allows users to rate the restaurant.

The following figure shows the application flow

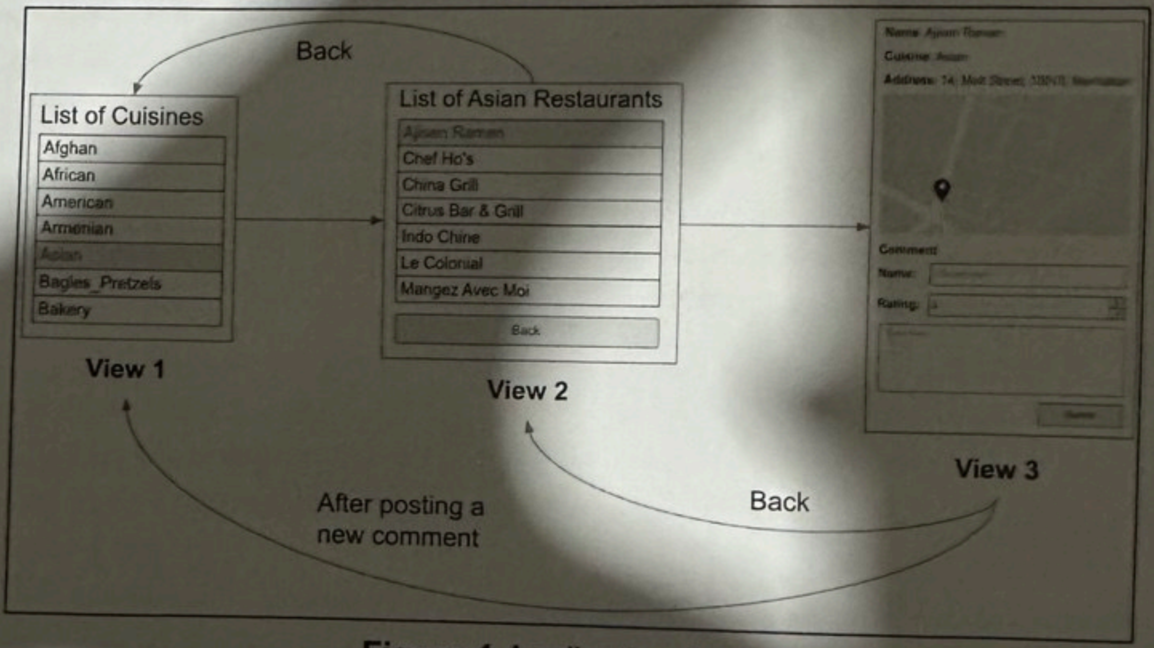


Figure 1 Application flow

Details of individual views will be provided in subsequent tasks.

Note: the terms client and frontend are used interchangeably to refer to the Angular application. Similarly the terms server and backend are used interchangeably to refer to the Spring Boot application.

Assessment

Task 0 (0 marks)

Unzip the given assessment source. Once you have unzipped the file, you will find the following

- `client` directory - a partially completed Angular application ←
- `server` directory - a partially completed Spring Boot application with ← all the required dependencies for this assessment. You can add additional dependencies to the `pom.xml` file
- `restaurants.json` - a JSON file containing restaurant data

You should now initialise the directory as a Git repository and perform a commit and push it to Github. 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 Feb 24 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 Feb 24 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 (4 marks)

Each document in `restaurants.json` has the following field

Field	Description
`_id`	Mongo's document id
`restaurant_id`	Restaurant's id
`name`	Restaurant's name
`address`	Address field is an embedded document which consists of the following fields

Field	Description
	• `building` • `street` • `zipcode` • `coord` - an array containing the longitude and latitude of the restaurant. The first element is the longitude and the second is the latitude
`borough`	District
`cuisine`	Type of food that the restaurant serves
`grades`	An array containing the grades received. Each element of the array is a document consisting of the following fields • `date` - when the grade was given • `grade` - grade consist of the following ○ `A, B, C, P, Z, "Not Yet Graded"` • `score`

Table 1 Description of the restaurant document

Import local first

Import `restaurants.json` to your remote (cloud based eg Mongo Atlas, Railway, etc) Mongo database. You may call the database and collection any name you wish.

Write the command you used to import `restaurant.json` into your Mongo database in the file called `restaurant_import.txt` of your repository.

Hint: during development you should work with a local instance of the restaurant database for better productivity and performance.

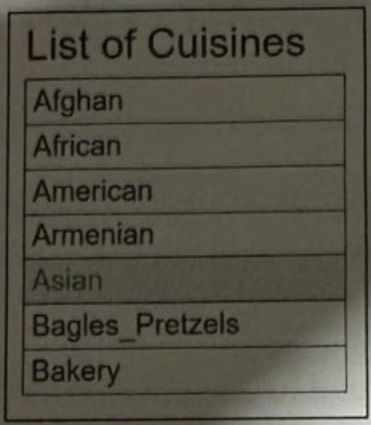
Task 2 (58 marks) ✓

Implement the interaction between frontend View 1 and the backend server; View 1 makes the following HTTP request to retrieve a list of cuisines available from the Mongo database

```
GET /api/cuisines
Accept: application/json
```

1 ✓

Figure 2 shows View 1 displaying the list of cuisines returned from the backend.



List of Cuisines

Afghan
African
American
Armenian
Asian
Bagles_Pretzels
Bakery

← db.restaurants.distinct("cuisine")

✓

Figure 2 View 1

Use the following provided classes for Task 2.

Frontend (Angular)

- `CuisineListComponent` ✓
- `RestaurantService` ✓

Backend (Spring Boot)

- `RestaurantService` ✓
- `RestaurantRepository` ✓

You can create additional classes; you can modify any of the above classes unless you are specifically prohibited. If you modify classes that you are not supposed, 10 marks will be deducted from Task 2. See the comments in the classes.

Write the native Mongo query in the comment above the method in the repository class that returns the list of cuisine names. For example, if your repository method `findRestaurantByName()` returns restaurant by name, then you should write the native Mongo query above `findRestaurantByName()` as follows assuming `restaurants` is the name of the collection

```
// db.restaurants.find(
//        { name: "a restaurant name" })
public Optional<Restaurant> findRestaurantByName(...
```

Marks will be awarded for the native Mongo query.

Some of the cuisine names returned from the restaurant collection contains a slash (/). Convert the slash to an underscore (_) before returning the result to the client (Angular). You can perform this in Java.

You can choose the frontend to be deployed either as standalone (from a different origin) or as part of the backend (from the same origin). Whichever method you choose, create the necessary configuration files or classes.

Task 3 (48 marks)

When a cuisine is selected (eg. Asian), the client sends the following HTTP request to retrieve a list of all the restaurant serving the cuisine

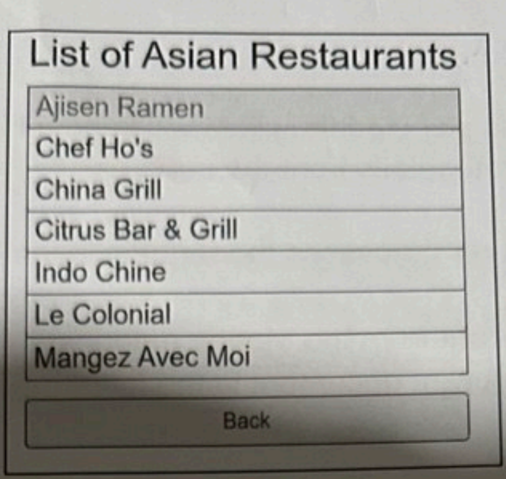
```
GET /api/<cuisine>/restaurants
Accept: application/json
```

where `<cuisine>` is the selected cuisine.

The client displays the selected cuisine name and a list of all the restaurants serving that cuisine in View 2. See Figure 3.

Add a Back button to return to View 1. You may layout View 2 according to your preference but must meet all the stated requirements for Task 2.

Figure 3 shows an example of View 2.



List of Asian Restaurants

- Ajisen Ramen
- Chef Ho's
- China Grill
- Citrus Bar & Grill
- Indo Chine
- Le Colonial
- Mangez Avec Moi

Figure 3 View 2

Use the following classes to perform Task 3.

Frontend (Angular)

- `RestaurantCuisineComponent`
- `RestaurantService`

Backend (Spring Boot)

- `RestaurantService`
- `RestaurantRepository`

You can create additional classes; you can modify any of the above classes unless you are specifically prohibited. If you modify classes that you are not supposed, 10 marks will be deducted from Task 3. See the comments in the classes.

Write the native Mongo query in a comment above the method in the repository class on the server side that queries and returns the list of restaurant names. The restaurant names should be sorted in ascending order according to their names.

Note: Remember to replace all underscore (_) back to slashes (/) in the cuisine name before performing therestaurant search.

Task 4 (92 marks)

When a restaurant is selected the client will send a HTTP request to retrieve the restaurant details from the server.

The Spring Boot server processes the request in the following manner:

Retrieves the restaurant from the Mongo database. The Mongo query must return the restaurant document in the following structure

```
{
  restaurant_id: <restaurant_id>,
  name: <name>,
  cuisine: <cuisine>,
  address: <concatenation of building, street, zipcode
and borough with a comma separator>
  coordinates: <coord in address>
}
```

Create From Doc

For example, querying Ajisen Ramen should return the following document from the restaurant collection.

```
{
  restaurant_id: "40827287",← get int!                ✓
  name: "Ajisen Ramen", ← get string
  cuisine: "Asian",     ← get string
  address: "14, Mott Street, 10013, Manhattan" ← get string
  coordinates: [ -73.9985052, 40.7141563 ]    ← get array
}
```

Write a native Mongo query in the comment above the method in the repository class on the server side that returns the above mentioned restaurant document.

After you have the restaurant document, use its coordinates from the `coordinates` field, to retrieve the map from http://map.chuklee.com with the following request

```
GET /map?lat=<latitude>&lng=<longitude>
Accept: image/png ←
```

where `lat` is the latitude and `lng` is the longitude. The latitude is the second element of the `coordinates` and longitude is the first element. The http://map.chuklee.com endpoint returns the map image in a byte array (`byte[].class`).

Store the map image in your Spaces/S3 bucket and include the map as an attribute of the restaurant document before returning the result back to the client.

If the map of the restaurant has been cached in your Spaces/S3 bucket, you should use that instead of querying it from http://map.chuklee.com endpoint.

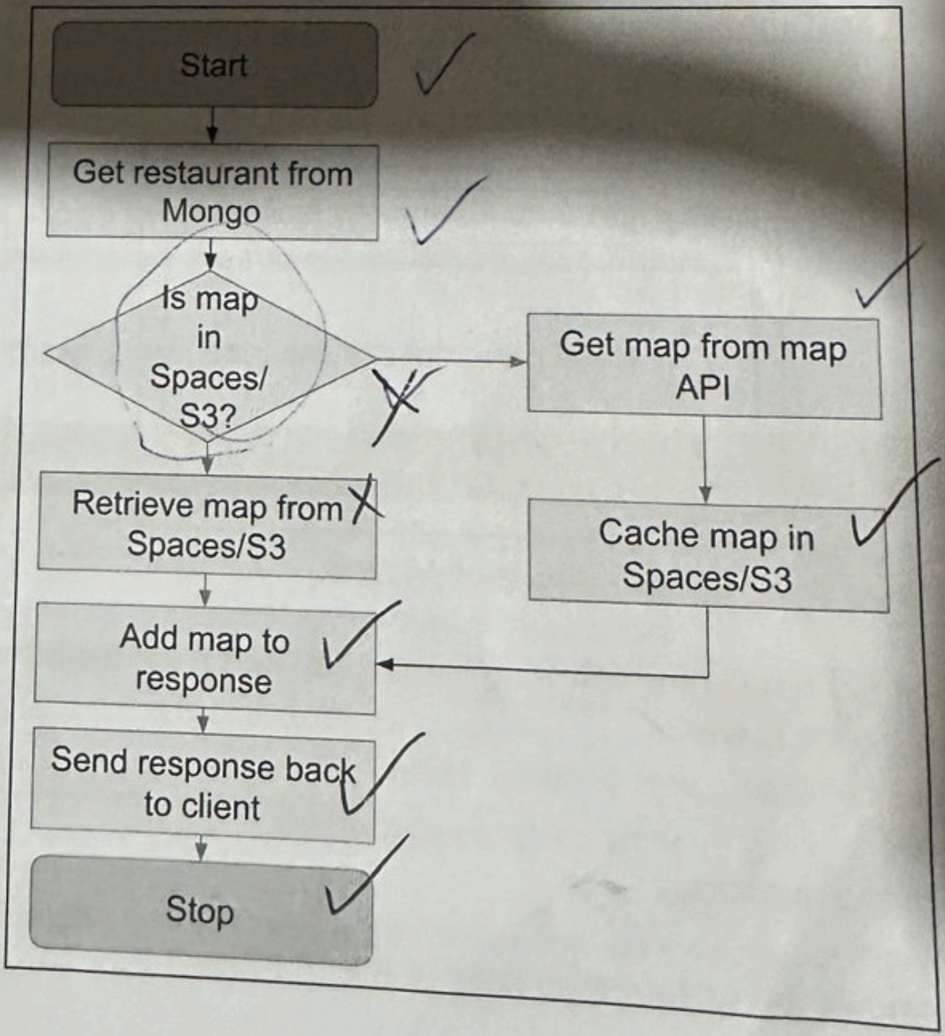
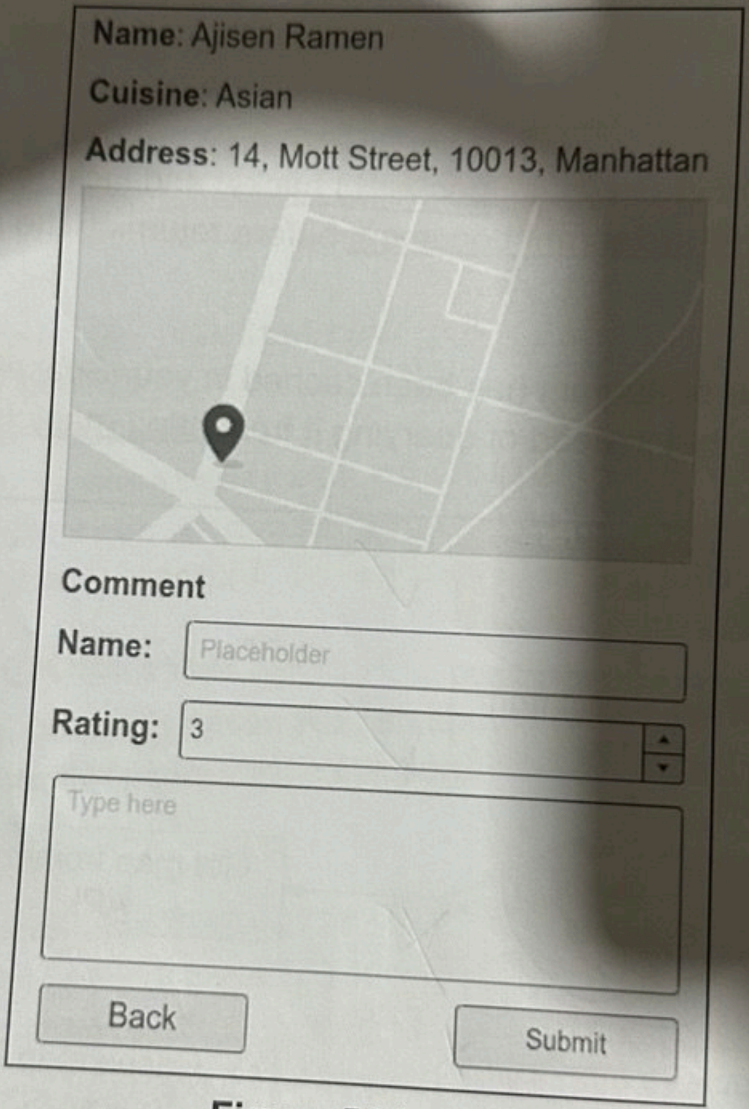


Figure 4 Operations in Task 4

Figure 4 shows the process flow in Task 4.

When the client receives the response (restaurant details) it will display the response in View 3 (Figure 5) ✓



Name: Ajisen Ramen

Cuisine: Asian

Address: 14, Mott Street, 10013, Manhattan

Comment

Name: Placeholder

Rating: 3

Figure 5 View 3

View 3 should display the following information about the restaurant

- Restaurant's name ✓
- Type of cuisine ✓
- Address ✓
- Map of the restaurant ✓

Figure 5 shows the restaurant details of Ajisen Ramen

You can layout View 3 according to your preference but all the required fields must be present.

Use the following classes for Task 4 ✓

Frontend (Angular)

- `RestaurantDetailsComponent`
- `RestaurantService`

Backend (Spring Boot) ✓

- `RestaurantService`
- `RestaurantRepository`
- `MapCache`

You can create additional classes; you can modify any of the above classes unless you are specifically prohibited. If you modify classes that you are not supposed to, 10 marks will be deducted from Task 4. See the comments in the classes.

Task 5 (42 marks) ✓

View 3 allows users to post comments on a restaurant. A comment form is available below the restaurant details (see Figure 5).

The comment form contains the following fields

Field name	Description
`name`	The name of the person giving the comment. The name must be longer than 3 characters
`rating`	A value between 1 and 5 (inclusive)
`text`	Must not be empty

Table 2 Comments form

The Submit button should be disabled if the form does not conform to the requirements in Table 2.

The Back button to allow users to return to View 2.

Use the following HTTP

```
POST /api/comments
Content-Type: application/json
Accept: application/json
```

to submit the comments to Spring Boot; comments should be saved in a collection called `comments`. The `comments` collection should be in the same Mongo database as the restaurant collection. The payload of the above `POST` method includes all the comment fields described in Table 2 and the restaurant_id.

The Spring Boot controller should return the following response after successfully inserting the new comment

Done

```
201 Created
Content-Type: application/json

{ "message": "Comment posted" }
```

✓

The client should then navigate back to View 1. See Figure 1.

Use the following classes for Task 5

Frontend (Angular) ✓

- `RestaurantDetailsComponent`
- `RestaurantService`

Backend (Spring Boot) ✓

- `RestaurantService`
- `RestaurantRepository`

You can create additional classes; you can modify any of the above classes unless you are specifically prohibited. If you modify classes

Chang promise

is fine

where you are not supposed to, 10 marks will be deducted from Task 5. See the comments in the classes.

Task 6 (6 marks)

Deploy the backend to Railway or other equivalent service.

If you are deploying the frontend separately from Spring Boot, then deploy your frontend to Vercel or similar service.

Submission

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

Only commits on or before **1700 Friday Feb 24 2023** will be accepted. Any commits **after 1700 Friday Feb 24 2023** will not be accepted. No other form of submission will be accepted (eg. ZIP file).

Remember to **make your repository public after 1700 Friday Feb 24 2023** 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 shown in your NRIC)
2. Your email
3. Batch - 2a or 2b
4. Git repository URL. You should only post 1 Git
5. Railway deployment URL. If you are deploying Angular as a standalone application (cross origin), post the URL as well. Do not undeploy your application and its dependent (resources eg. databases) until after **2359 Friday March 10 2023**

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 5 items mentioned above is missing, and/or