Client Side Foundation Assessment

Date: Wed Apr 05 2023
Assessment Time: 0900 - 1700 (including meal breaks)

Overview

In this assessment, you will be writing an application to search and comment on restaurants from a Mongo database.

There are **9 tasks** in this assessment. Complete all tasks.

Passing mark is **65%** (**78 marks**). Total marks is **120**.

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

Application Overview

This application allows users to search for movie reviews and to add their comments and ratings to those reviews.

The following figure shows the application flow

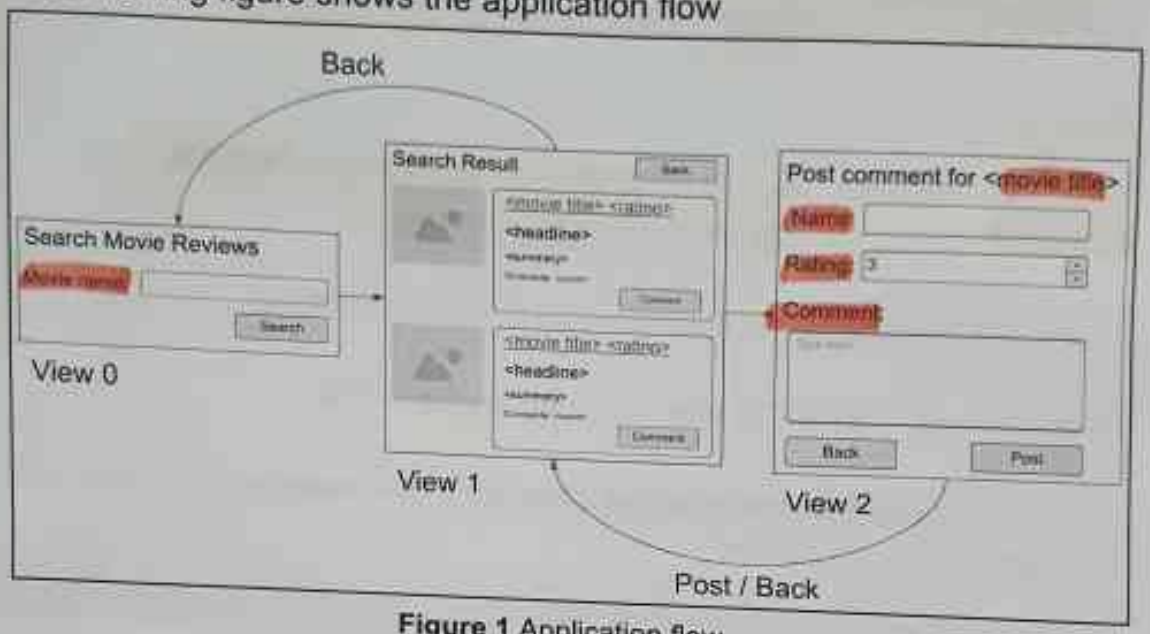

Figure 1 Application flow

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

1. **View 0** - Search movie review
2. **View 1** - Shows the list of reviews that matches the search criteria
3. **View 2** - A comment form for user post their thoughts on a movie

Details of individual views will be provided in subsequent tasks.

Assessment

Setup

Create an empty Git repository. Unzip the given assessment template ZIP file into your repository. This should create a directory called `server` with a partially completed Spring Boot application. The `pom.xml` contains all the required dependencies; however you are free to add any additional dependencies you may need.

Generate an Angular application inside your git repository. After this you should have 2 directories in your repository; an Angular application which you have generated and a Spring Boot application under `server` directory.

You should now 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 Wed Apr 05 2023** so that the instructors can access your work.

Provision an instance of Mongo database in the cloud either in Railway, Mongo Atlas or any cloud provider of your choice. The database, you may give it any name, will be used to store user comments. Hint: for efficiency and productivity, you should work with a local (on your notebook) instance of Mongo rather than the cloud during the assessment.

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 Wed Apr 05 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 (6 marks)

Create an account with The New York Times Developer Network (https://developer.nytimes.com/). After you have created the account, get an API key; the key should be enabled for Movie Reviews.

This assessment will be using the Movie Reviews API (https://developer.nytimes.com/docs/movie-reviews-api/1/overview). The specific endpoint will be the `search.json` endpoint (https://api.nytimes.com/svc/movies/v2/reviews/search.json) where you will be searching movie reviews by movie title/name.

Task 2 (18 marks)

Create a component called `SeachReviewComponent` which will be view 0.

The component should have a single input text box and a button labelled '**Search**'. An example of this component is shown in Figure 2

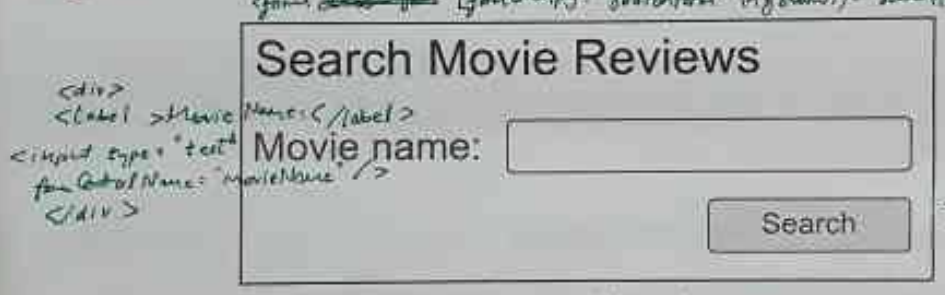

Figure 2 View 0

Configure this component to be the first component (View 0) that a user sees when the Angular (frontend) application is opened.

The Search button should be disabled until at least 2 characters are entered into the input box. Leading and trailing blank spaces are not considered characters.

You may layout View 0 according to your preference but all the requirement elements must be present.

Decide on a deployment model for your Angular and Spring Boot application either 'same origin' (serving Angular from Spring Boot) or

'cross origin' (Angular and Spring Boot are on 2 separate domains) and setup the required configuration.

Task 3 (14 marks)

When the Search button in View 0 is pressed, it will trigger the following HTTP to Spring Boot (backend) application

```
GET /api/search?query=<movie name>
Accept: application/json
```

where `<movie name>` is the name entered into the input box in View 1. Note that the HTTP request may not necessarily be made from View 0. You are free to decide when is the best time to make this request.

Write a request handler in `MovieController` class to process the above request.

Task 4 (12 marks)

Implement a search for movie reviews from the following endpoint

https://api.nytimes.com/svc/movies/v2/reviews/search.json

Map the result to the `Review` class; select the following attributes JSON attributes from the result

- `display_title` title
- `mpaa_rating` rating
- `byline` byline
- `headline` headline
- `summary_short` summary
- `link.url` reviewURL
- `multimedia.src` image

and marshall these into the `Review` class. See the `Review` class source file for mapping between the JSON attributes and the `Review`'s members.

Write this implementation in `MovieService.searchReviews()`. If there are any errors to the REST call of the New York Time's `search.json` endpoint, `searchReviews()` should return an empty list.

Task 5 (12 marks)

Find the number of comments that have been posted for all the movie reviews returned by `MovieService.searchReview()`.

The comments are stored in a collection called `comments` in your Mongo database.

Write your implementation for getting the comment count for the movies in `MovieRepository.countComments()`. You may modify `countComments()` parameters (viz use any number of parameters of any type) but not the return type.

You must also write the equivalent native Mongo query in the comments above `countComments()`. Native Mongo query is the query that you write in the Mongo shell when querying Mongo; an example of native Mongo query is `db.books.findOne()`. Marks will be given for this query.

Update the Review instances with the count returned from `commentCount()`.

The following is the pseudo code that summarises **Task 4** and **Task 5**.

```
movie_review_results = search_reviews()
for each movie m, in movie_review_results
    comment_count = get_comments(m)
    update m with comment_count
```

Note: initially your `comments` collection will be empty.

Integrate **Task 4** and **Task 5** into the request handler from **Task 2**. Return the response back to the client (Angular).

Task 6 (16 marks)

Create a component called `MovieReviewsListComponent` (View 1) to display the review results (returned from **Task 5**). View 1 is shown in the following Figure 3

For every movie review display the following information

- Movie title
- Rating
- Review summary
- Review article URL should be a clickable link (`<a>`) in the movie title viz. when a person clicks on the movie title, it will open the article in a separate browser tab.
- Number of comments for that movie
- Movie image. If the endpoint does not provide any image, display a placeholder image instead. You can use the provided `placeholder.jpg` image as the placeholder image or any decent image of your choice.

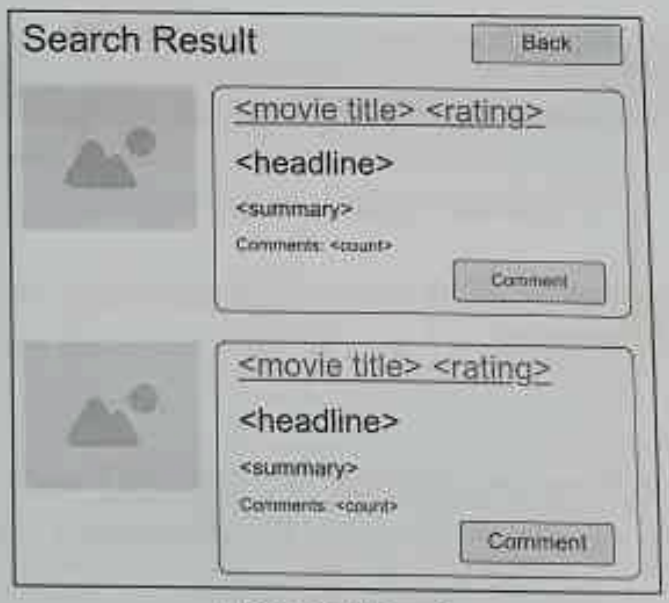

Figure 3 View 1

Every movie review should also have a button to allow users to comment on the movie.

Add a back button to View 1 to allow users to navigate back to View 0. See Figure 1 for the application flow.

If the search produces no result, show the following

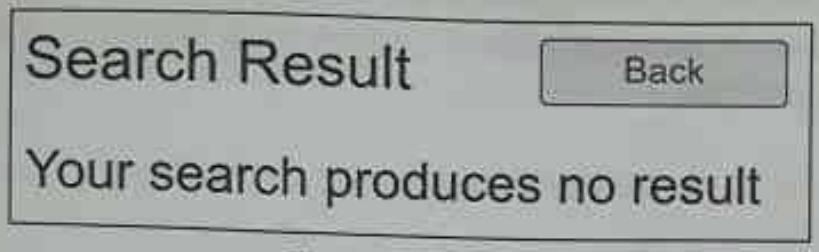

Figure 4 No result

Note: you may layout View 1 according to your preference but all the requirement fields must be present in the view.

Task 7 (24 marks)

When the Comment button is pressed in View 1, navigate to View 2. Create a component called `PostCommentComponent` for View 2; the following is an example of View 2 shown in Figure 5

Figure 5 View 2

View 2 displays the movie title. It has the following mandatory fields to allow users to comment on the movie

- The poster's name viz. the name of the person who is posting the comment. Users must provide their names when posting comments; the names should be at least 3 characters long

- Rating from 1 to 5
- Comment text which should not be empty

The Post button should be disabled until all the above requirements are met.

Angular will send the following HTTP request to Spring Boot when the Post button is pressed

```
POST /api/comment
Content-Type: application/x-www-form-urlencoded
Accept: application/json
```

The HTTP request should contain the following data in the payload

- movie name
- poster's name
- rating as number
- common text

Once the comment has been successfully posted, navigate back to View 1. View 1 should redisplay the same list of movies before transitioning from View 1 to View 2. The movie that the user has just commented on should also have its comment count incremented by 1.

If the user clicks on Back button in View 2, discard all inputs from the comment form and transition back to View 1 redisplaying the same list of movies as before.

You may layout View 2 according to your preference; make sure that all the required fields and buttons are present.

Task 8 (12 marks)

Write a request handler in `MovieController` class to receive and process the movie comments made from View 2.

Insert the comment into the `comments` collection; write your implementation of this insert in `MovieRepository` class. You must also

write the equivalent native Mongo insert above the insert method in `MovieRepository`. Marks will be given for this Mongo statement.

Task 9 (6 marks)
Deploy the backend to Railway or other equivalent service.

Submission

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

Only commits on or before **1700 Fri Mar 24 2023** will be accepted. Any commits **after 1700 Fri Mar 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 Fri Mar 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. Git repository URL. You should only post 1 Git
4. 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 Fri Mar 31 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 4 items mentioned above is missing, and/or
2. your information did not comply with the submission requirements eg. not providing your full name as per your NRIC, incorrect email, forgetting to post the Railway deployment URL, etc, and/or
3. the repository is not public after **1700 Fri Mar 24 2023**