Answer Clarification Document

Author: Senali Perera

Date: 18/12/2023

Assignment 1

The provided band object contains a bunch of JavaScript objects of members of the band. The member object can be structured under the following type.

```
type Member = {
   name: string,
   age: number,
   plays: Array<string>
}
```

Each Band object contains a property called members which then categorised under current and past where current and past are arrays of Member objects.

```
type Band = {
   members: {
      current: Array<Member>,
      past: Array<Member>
   }
}
```

The type for the plays attribute in the expected variable looks like the following.

```
type PlayType = {
   [index: string]: Array<string>
}
```

Finally the expected output after processing the band object (FormattedBand) looks like the following type.

```
type FormattedBand = {
   members: {
      current: Array<Member>,
      past: Array<Member>,
      all: Array<string>
   },
   plays: PlayType
}
```

all property contains an array of strings while plays contains an object with strings as keys and Array of strings as values.

In order to create **all** band members, the **getAll** function is used. Since it’s an array manipulation. The array util methods are used to provide an optimal solution. **getAll** function expects an array of members.

First it sorts the members array. The way it sorts is by using the **sort** Array method which takes a function with 2 parameters (2 member parameters in this case). The 2 members' names are converted to lowercase. Then check if 2 people are the same age. If the age is similar, a local comparison of the names is done to sort 2 people in ascending order. If not the difference between 2 people’s ages are returned ultimately sorting the members array from the age as well as the names. Finally the newly returned array is mapped to the member’s name using the array **map** method.

In order to get the **plays** Array, the **getPlays** function is used. It also accepts an array of members. Array method **forEach** is used to traverse the members array and then the plays array of each member is traversed.

Finally inside the **getFormattedBand** function, both above functions are called to get the properly formatted FormattedBand object.

Assignment 2 (Reusable dialog component)

Implementation

A service based implementation is used to achieve the best reusability of the Dialog Component.

Dialog Component

Component Name	`DialogComponent`

Inputs	`title: string`
	`buttonText: string`
	`customButtonHandler: Function`
	`cancelOnBackgroundClick: boolean`
	`hideCustomButton: boolean`

Methods	**`onCancelPressed()`**`: removes the dialog from the view`
	`customButtonPressed()``: Executes custom button click listener if provided. Calls onCancelPressed() to remove the view.`

Implementation	- The component displays the custom/default title. - The component displays the custom template passed as TemplateRef. - The dialog component closes when the “x” button is pressed. - The developer should be able to pass a click handler (customButtonPressed) for the custom button and that function should be executed on custom button click along with the general dialog closing logic. - The developer should have the ability to hide the custom button. - The developer should be able to set a custom name for the custom button. - The developer should have the option to

	choose if the dialog should be closed when clicked in the background.

Dialog Service

Service Name	`DialogService`
Methods	`open(...args)`
open(): inputs	- content: TemplateRef<any> - options: OptionType
Implementation	open method creates the view from the the template reference. Then it injects the view into the DialogComponent. Finally the options passed are assigned and updates the view.

OptionType

Type Name	`OptionType`
Attributes	title?: string buttonText?: string cancelOnBackgroundClick?: boolean hideCustomButton?: boolean customButtonHandler?: function

AppComponent

Component Name	`AppComponent`
Inputs	-
Methods	**openDialog(...args):** invokes the service function to display the DialogComponent. **dummyFunction():** This function is used to replicate the form submission. This is passed as the customButtonHandler.
Implementation	- The DialogService is initialised from the constructor. - **openDialog()** invokes the service method which causes the DialogComponent to show up. - **dummyFunction()** is a mock function which

	gets executed in the Dialog box custom button.

How to use the Dialog Component

Add **`DialogService`** as a provider in app configs (This is already configured for this project).

```
export const appConfig: ApplicationConfig = {
      providers: [provideRouter(routes), DialogService]
};
```

Create a new template

```
ng generate component component1
```

This will generate a template HTML file and component.ts file.
The DialogService should be initialised in order to trigger the DialogComponent. It can be done as follows.

1. Create the constructor and add the Dialog service to get initialised by itself.

   ```
   constructor(private dialogService: DialogService) {}
   ```

2. It's recommended to create a method which calls **`dialogService`** functions in order to make the code more readable.

   ```
   openDialog(dialogTemplate: TemplateRef<any>, options?: OptionType) {
         this.dialogService.open(dialogTemplate, options);
   }
   ```

Make sure to import **`OptionType`** from **./components/dialog/utils/types**

3. Invoke the above function from the HTML file with a defined template for the content.

   ```
   <button id="btn-form-alert"
         (click)="openDialog(
         dialogTemplateWithForm,
         { title: 'Sign In',
           cancelOnBackgroundClick: true,
   ```

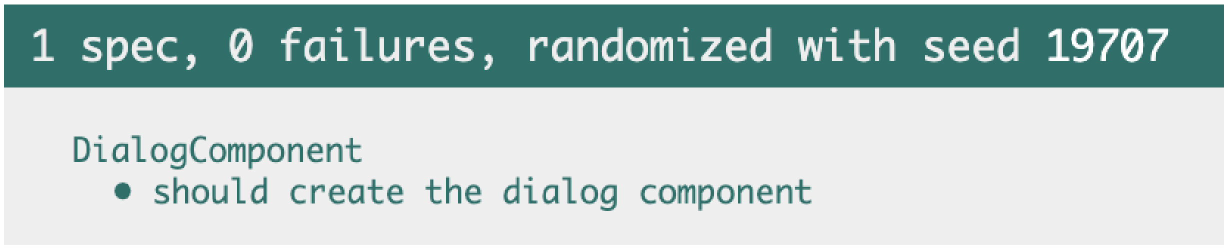
```
        customButtonHandler: buttonClickHandler,
        buttonText: 'Sign In' }
      )"
>
      Dialog with Form
</button>

<ng-template #dialogTemplateWithForm>
 <div>
  <form>
      <label id="fName-lbl" for="fNameInput">First name:</label><br>
      <input type="text" id="fNameInput" class="input"><br>
      <label id="lName-lbl" for="lNameInput">Last name:</label><br>
      <input type="text" id="lNameInput" class="input"><br><br>
  </form>
 </div>
</ng-template>
```

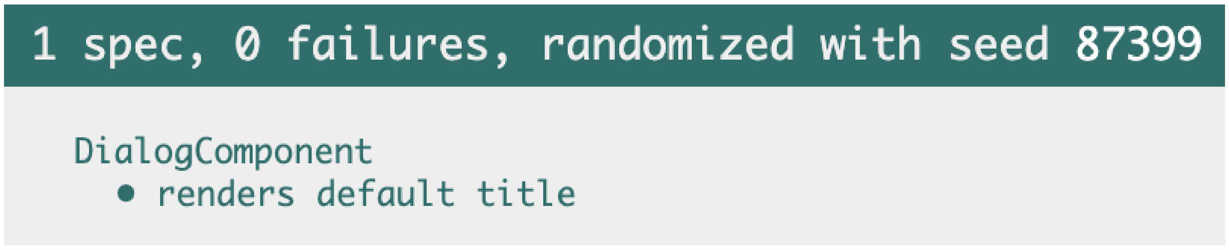
Unit Testing

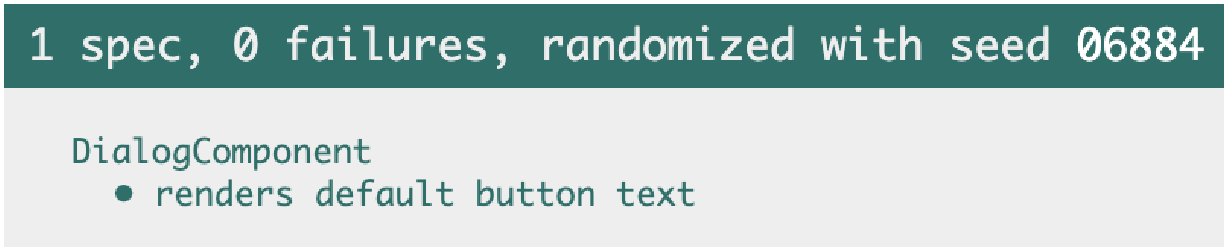
The unit tests were conducted using Angular default testing library Karma. The following unit tests were conducted.

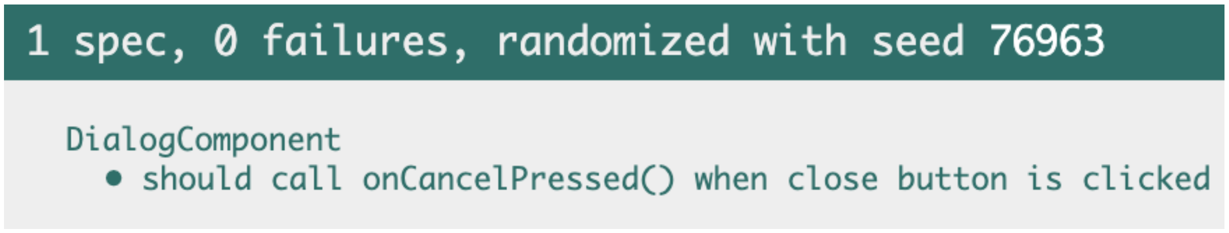
Dialog Component Testing

Action	Dialog Component creation
Expected	The Dialog Component to be created successfully
Result	The Dialog Component created successfully
Status	**SUCCESS**
 1 spec, 0 failures, randomized with seed 19707 DialogComponent • should create the dialog component	

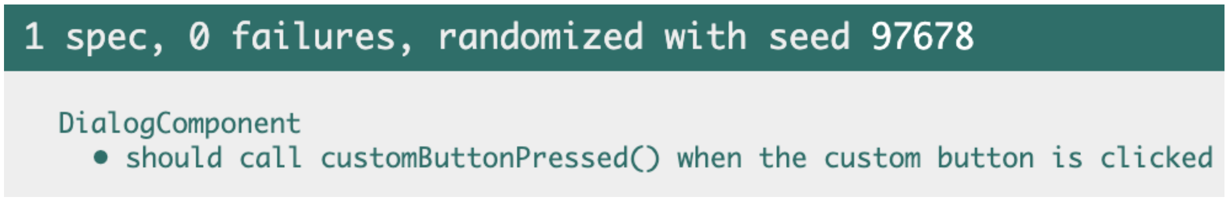
Action	Render default title
Expected	The Dialog Component to render default title if not provided
Result	The Dialog Component rendered the default title successfully

Status	**SUCCESS**
 1 spec, 0 failures, randomized with seed 87399 DialogComponent • renders default title	

Action	Render default button text
Expected	The Dialog Component to render default button text if not provided
Result	The Dialog Component rendered the default button text successfully
Status	**SUCCESS**
 1 spec, 0 failures, randomized with seed 06884 DialogComponent • renders default button text	

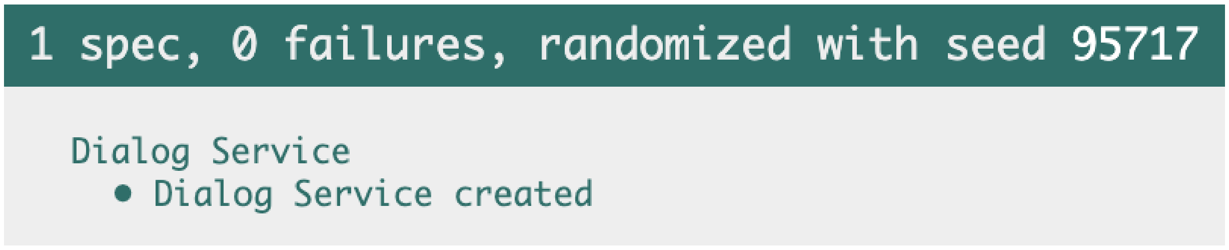
Action	Cancel pressed
Expected	Should call **`onCancelPressed()`** when close button is clicked
Result	The **`onCancelPressed()`** function was called when exiting.
Status	**SUCCESS**
 1 spec, 0 failures, randomized with seed 76963 DialogComponent • should call onCancelPressed() when close button is clicked	

Action	Custom Button Pressed
Expected	Should call `customButtonPressed()` when custom button is clicked
Result	The `customButtonPressed()` function was called when the custom button was clicked.
Status	**SUCCESS**

 `1 spec, 0 failures, randomized with seed 97678`
`DialogComponent`
`• should call customButtonPressed() when the custom button is clicked`	

Dialog Service Testing

Action	Dialog service is created
Expected	Should create the dialog service
Result	Dialog service got created successfully
Status	**SUCCESS**

 `1 spec, 0 failures, randomized with seed 95717`
`Dialog Service`
`• Dialog Service created`	

End to End Testing

E2E tests were conducted using cypress to test the whole flow of the application. Following are the test results.

Test Cases	- User visits the URL - User clicks on the Dialog with Text button - User checks if the default title is applied to the dialog - User checks if the default content is displayed - User clicks on the background - User clicks on the button with form button - User checks if the title is correct - User checks if the label is correct - User enters first name

	- User check if the label is correct - User enters last name - User clicks on the submit button - User checks if the console has the entered names - User clicks on button with component - User checks if the title is correct - User checks if the content is correct - User checks if the custom button is not visible - User clicks on the x (close) button
Expected	The whole workflow to progress successfully
Result	The whole workflow executed successfully
Status	**SUCCESS**

Future Implementation

- Add more styling and support to customise styling.
- Add the resizing for the dialog box component.