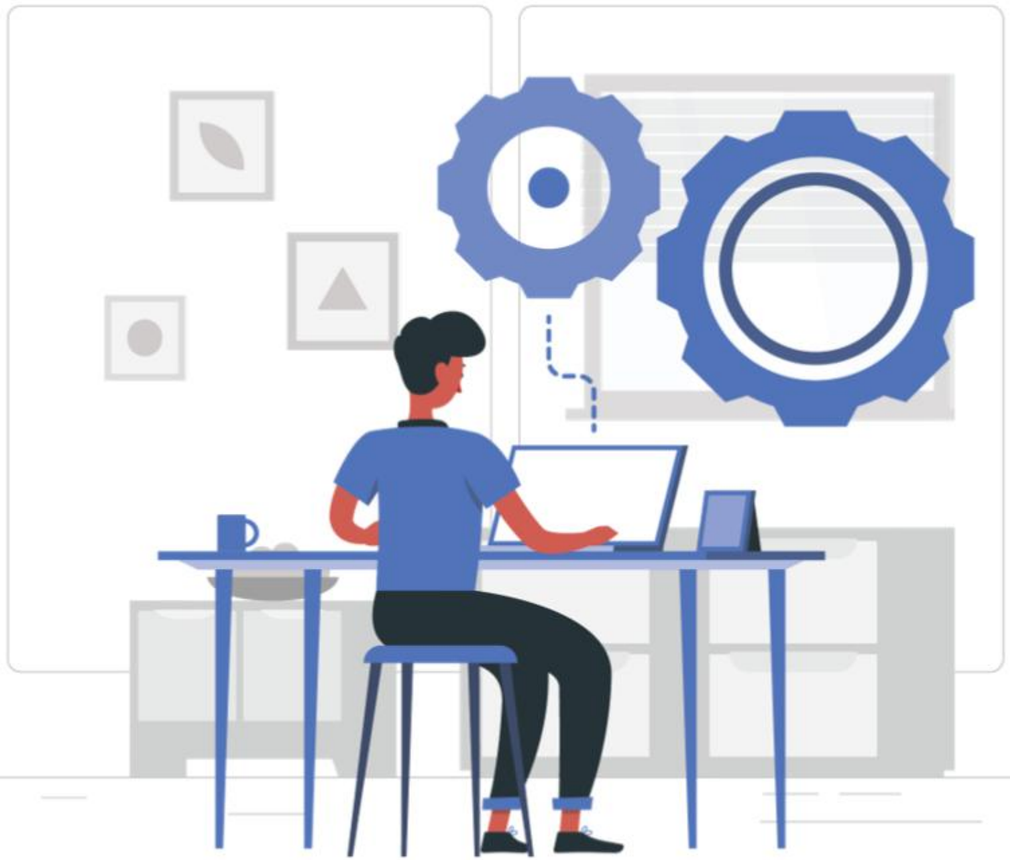


Practice
Developing Interactive Template-Driven Forms Inside SPA

Practice Exercise

- Practice: Develop a feedback form using Template-driven forms

Points to Remember

- Angular Material components should only be used for styling the feedback form.
- The HTML **`pattern`** attribute should be used to specify the pattern for the phone number and the email form fields to validate them.
- Text for validation error messages should be given as stated in the expected output. Failing to do so will result in failure of the test.
- Run the `json-server` to add the feedback data to the `feedbacks.json` file in the **`feedback-data`** folder.

Instructions for Practice

- Click here for the boilerplate.
- Please read the README.md file provided in the boilerplate for further instructions about the practice exercise.
- Fork the boilerplate into your own workspace.
- Clone the boilerplate into your local system.
- Open command terminal and set the path to the folder containing the cloned boilerplate code.
- Run the command `npm install` to install the dependencies.
- Open the folder(**fe-c5-s2-template-forms-practice**) containing the boilerplate code in VS Code.
- Create a template-driven form using Angular Material components and generate the output as shown in the image files present in the boilerplate.

Notes:

1. The solution of this practice will undergo an automated evaluation on the `CodeReview` platform. (Local testing is recommended prior to `CodeReview` testing)
2. The test cases are available in the boilerplate.

Context

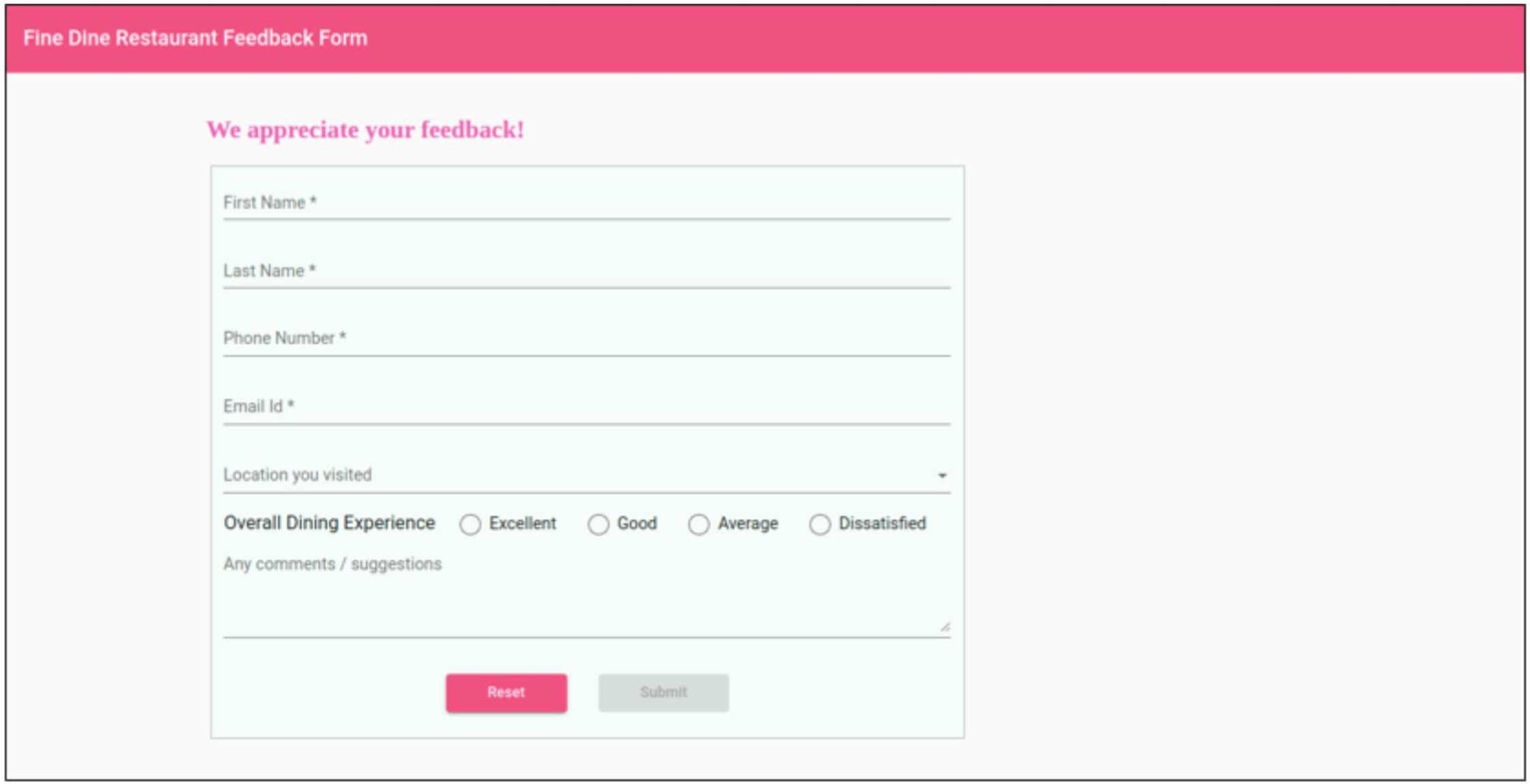
Fine Dine Restaurant is a popular restaurant with branches located in various parts of the United States. Its website provides various details about the location of its branches, the process of making reservations, and the various items offered by the restaurant.

The restaurant's management wants to collect customer feedback to improve the customer experience.

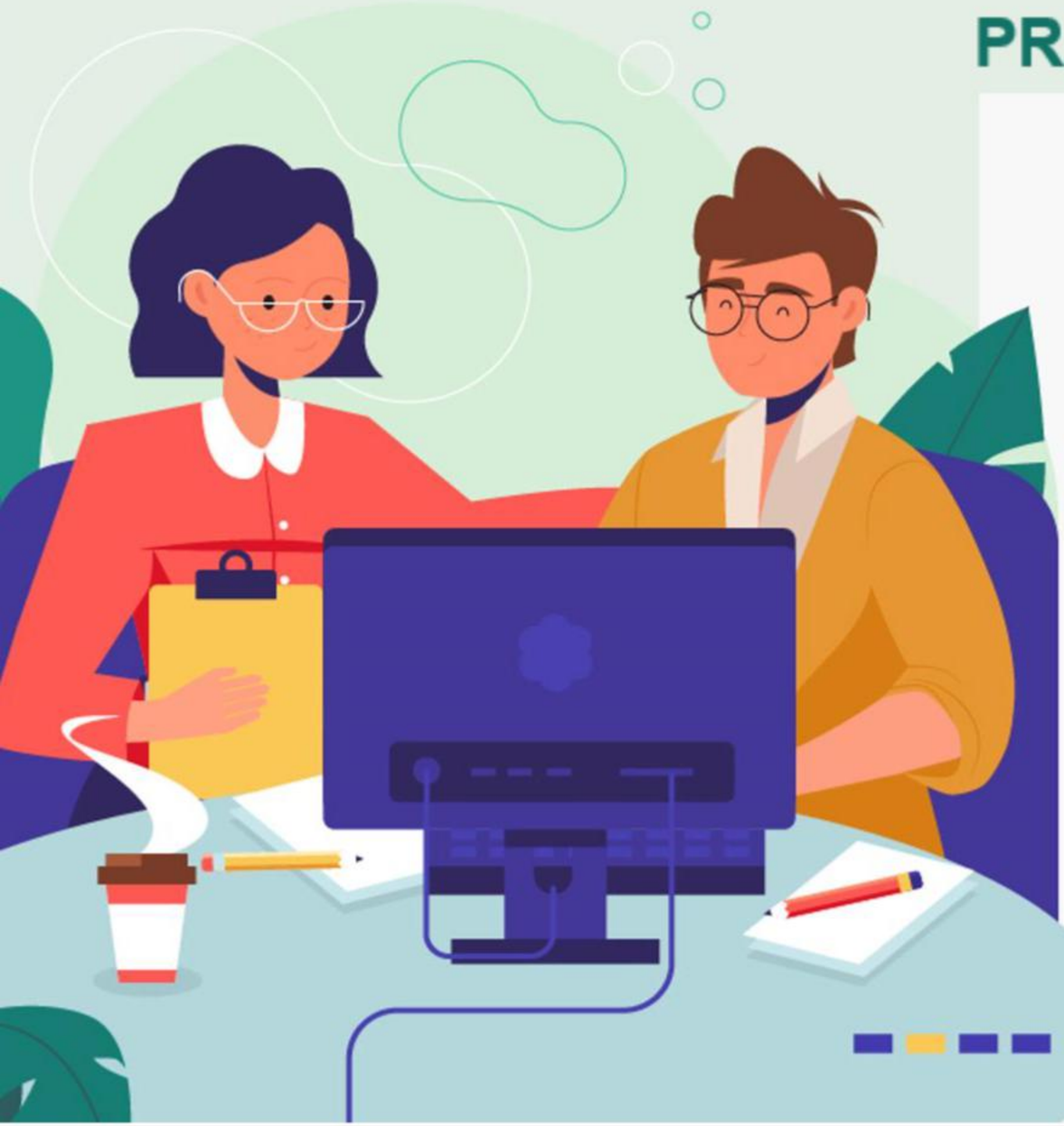
A restaurant feedback form is a tool that customers are asked to fill out to provide feedback about their dining experience. This form will ask for basic details such as the customer's name, its contact details, the location they visited, etc., along with the overall rating for the dining experience. The form will also seek suggestions for improvement.

You, as a web developer, have been assigned the task of designing and creating the feedback form for the Fine Dine restaurant.

Expected Output: Feedback Form



PRACTICE



Develop a Feedback Form Using Template-Driven Forms

Design and create a template-driven feedback form for the Angular application that is given in the boilerplate.

Note: The tasks to design and create the form are given in the upcoming slides.

Tasks

- To develop the solution for the Fine Dine restaurant application, following tasks need to be completed:
  - Task 1: Create a data model
  - Task 2: Include required modules
  - Task 3: Create the feedback component
  - Task 4: Define the form layout inside the template
  - Task 5: Add validation attributes to form controls
  - Task 6: Handle form validation
  - Task 7: Display a notification message upon successful form submission

Note: Task details for the practice are provided in the upcoming slides.

Task 1: Create a Data Model

- Create a feedback data model that corresponds to the form data model.
- Create a type called `Feedback` in the `feedback.ts` file in the **`models`** folder with the following type properties:
  - id (number)
  - firstName (string)
  - lastName (string)
  - email (string)
  - phone (number)
  - location (string)
  - rating (number)
  - comments (string)

Note: The value of the property **`id`** will be auto generated while saving the data in the `json` file.

Task 2: Include the Required Modules

- Add the `FormsModule` in the application root module to enable the forms feature.
- Add the following modules in the application root module to create the form styled with Angular Material components.
  - MatFormFieldModule
  - MatSelectModule
  - MatInputModule
  - MatRadioModule
  - MatSnackBarModule
  - MatButtonModule
  - MatToolBarModule
- Import all the above-mentioned modules in the import list of the @NgModule decorator.

Task 3: Create the Feedback Component

- Create the `FeedbackComponent` inside the Fine-Dine Angular application using the command:

```
ng generate component feedback
```

- The command creates an Angular component with the name `FeedbackComponent` and updates the import statements in the `app.module.ts` file.
- Do the following inside the `FeedbackComponent` (as given in the code below):
  - Define a `feedback` property that reflects the form data model with empty values.
  - Add a string array for the `location` property with values as mentioned in the code below.
  - Define the constructor to create a `MatSnackBar` instance.

```
feedback: Feedback = {};
location = ['Huntsville', 'Springdale', 'Orlando',
'Augusta', 'New York'];
constructor(private _snackBar: MatSnackBar) { }
```

Note: The component name should be kept as "FeedbackComponent" as it is used in testing.

Task 4: Define the Form Layout Inside the Template

- Define a layout for the form with form controls that correspond to form model properties: `firstName`, `lastName`, `phone`, `email`, `location`, `rating` and `comments`. Style the form by adding custom styles inside `feedback.component.css`.
- Following are the steps to define the form layout:
  - Step 1: Add a template reference variable to the <form> tag to access its values upon form submission (as given in the code below).

```
<form class="form" (submit)="onSubmit(feedbackForm)" #feedbackForm="ngForm">
```

Task 4: Define the Form Layout Inside the Template (Cont'd.)

Use `<mat-form-field>` within the `<form>` tag to create input form controls for `firstName`, `lastName`, `phone`, `email`, `location`, and `comments` properties, and `<mat-radio-group>` for `rating` property.

- Step 2: Create a form control element for the `firstName` property as explained below.
  - Add the `ngModel` directive to the form control element to bind the control with the data model property using two-way binding syntax.
  - Add the **`name`** attribute with value matching to the model property to the form element, which Angular uses to register the element with the parent `<form>`. The code below shows how to add a `<mat-form-field>` for the `firstName` property .

```
<mat-form-field appearance="legacy">
  <input type="text" name="firstName" [(ngModel)]="feedback.firstName" />
<mat-form-field/>
```

Notes:

1. Repeat the same steps for all other form control elements.
2. The form controls with "name" properties are used in testing, so they must have the same names while coding.

Task 4: Define the Form Layout Inside the Template (Cont'd.)

- Step 3: Use `<mat-select>` to create a drop-down list for location values.
  - Use the `*ngFor` directive with `<mat-option>` to load the list of location values as shown in the code below.

```
<mat-form-field appearance="legacy">
  <mat-label>Location you visited</mat-label>
  <mat-select id="location" name="location"
[(ngModel)]="feedback.location">
      <mat-option *ngFor="let loc of location"
[value]="loc">{{loc}}</mat-option>
  </mat-select>
</mat-form-field>
```

  - Step 4: Use the <mat-radio-group> to allow users to select only one value at a time for feedback ratings (as given in the code below).

```
<mat-radio-group aria-label=Select an option" name="rating"
[(ngModel)]="feeedback.rating">
<mat-radio-button value="5">Excellent</mat-radio-button>
<!-- Add for other options-->
</mat-radio-group>
```

Task 4: Define the Form Layout Inside the Template (Cont'd.)

- Step 5: Use the HTML element `<textarea>` that allows users to input comments in multiple lines (as given in the code below).

```
<textarea rows="4" cols="45" placeholder="Any comments / suggestions"
type="text" matInput name="comments" id="comments"
[(ngModel)]="feedback.comments"></textarea>
```

- Step 6: Add `<button>` of type `submit` which will submit the form when clicked (as given in the code below).

```
<button color="primary" type="submit" mat-raised-button>Submit</button>
```

- Step 7: Add `<button>` of type `"reset"` which resets the form with empty values (as given in the code below).

```
<button color="accent" mat-raised-button type="reset">Reset</button>
```

Task 5: Add Validation Attributes to Form Controls

- Following are the form controls with their validation criteria.

Form Control	Validation	Error Messages
First Name	Should not be blank and have minimum length of 2 characters	- First name is required - First name minimum length is 2 characters
Last Name	No validation	-Nil-
Phone	Should not be blank and accepts only 10-digited numbers starting with 7, 8 or 9	- Phone number is required - Enter valid phone number containing 10 digits starting with 7/8/9
Email	Should not be left blank and accepts valid email value	- Email id is required - Enter valid email id
Location	Should select one value from the given set of drop-down values	-Nil-
Rating	Should select one of the radio buttons from the group	-Nil-
Comments	No validation (optional to type some text content)	-Nil-

Note: The error message text should be used as mentioned above, as these texts are used in testing.

Task 5: Add Validation Attributes to Form Controls (Cont'd.)

- Step 1: Add HTML5 attributes like `required`, `minlength`, and `pattern` to validate the form control input values. The code below shows how to add attributes to the `firstName` property.

```
  <input type="text" name="firstName" required minlength="2"
[(ngModel)]= "feedback.firstName" />
```

- Step 2: Similarly, add necessary attributes for all other form control elements.

Task 6: Handle Form Validation

- Step 1: Add a template reference variable to each form control element with the same name as the value of the `name` attribute. The template reference variables will be used to check for validation errors (as given in the code below).

```
  <input type="text" name="firstName" #firstName="ngModel" required
minlength="2" [(ngModel)]= "feedback.firstName" />
```

- Step 2: Use `<mat-error>` for displaying the validation error message. To determine which validation error has occurred, `*ngIf` should be used. The code below is shown for the `firstName` property .

```
  <mat-error *ngIf="firstName.errors?.['required']">
    First name is required
  </mat-error>
  <mat-error *ngIf="firstName.errors?.['minlength']">
    First Name minimum length is
{{firstName.errors?.['minlength']?.requiredLength}}
  </mat-error>
```

- Step 3: Repeat the same steps for all other form control elements.

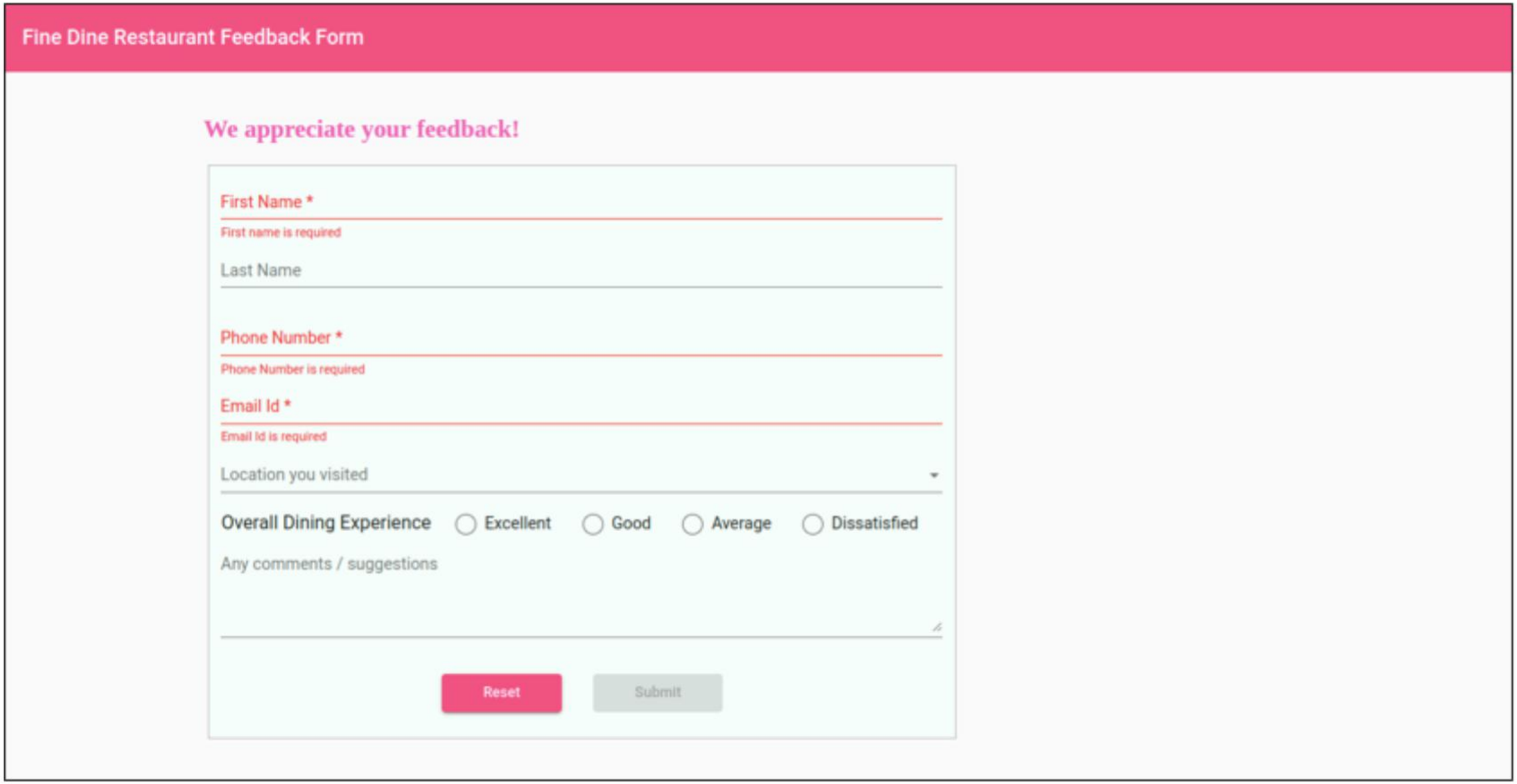
Task 6: Handle Form Validation (Cont'd.)

- Step 4: Add the `[disabled]` attribute to the submit button to make it disabled when the form is in invalid status (as given in the code below).

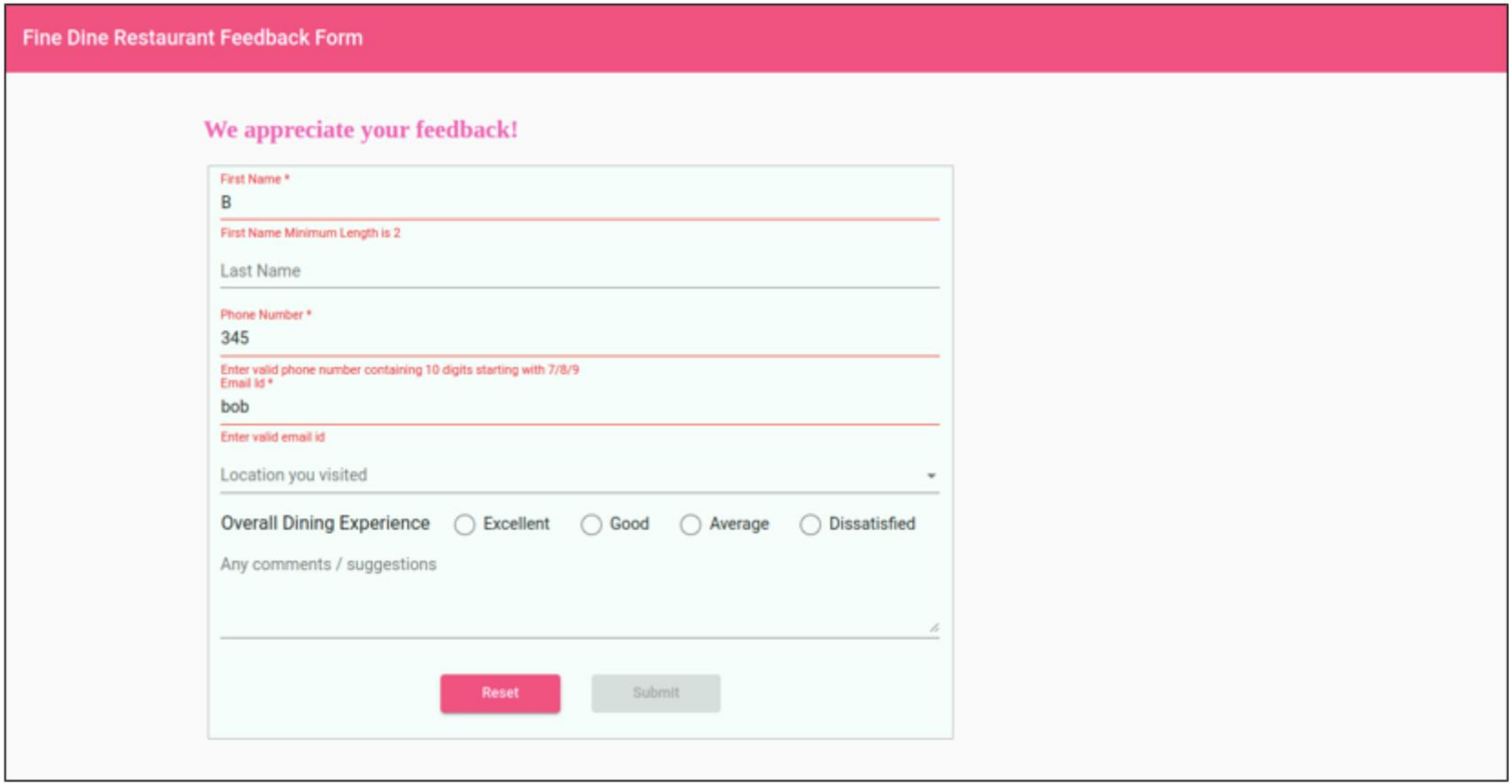
```
<button type="submit" [disabled]="feedbackForm.invalid"  mat-raised-
button >Submit</button>
```

Note: The expected output for the feedback form along with validation error messages is given in the upcoming slides.

Expected Output: Form With Validation Errors



Expected Output: Form With Validation Errors (Cont'd.)



Task 7: Display Notification Message Upon Successful Form Submission

- Step 1: Inside the `FeedbackComponent` class, define the **`onSubmit()`** method, which calls the `Feedback` service method to save the form data in the `json-server`. The application should display a notification message, "Feedback submitted successfully", using a snack bar upon successful form submission.

- Step 2: After a successful form submission, reset the form with empty values by calling the **resetForm()** method on the `feedbackForm` object.

```
onsubmit(feedbackForm:any){
this.feedbackService.saveFeedback(feedbackForm.value).subscribe({
       next: data => {
              this._snackBar.open('Feedback submitted successfully',
{ duration: 5000, panelClass: ['mat-toolbar', 'mat-primary']}
       )},
       error: err => {
             this._snackBar.open('Failure while connecting to server, try
again!!', 'Failure', {duration: 3000,panelClass: ['mat-toolbar', 'mat-warn']}
         )}
  });
}
```

Expected Output: Successful Form Submission