Class 20 - Course Project (Forms) && Using Pipes to Transform Output

Class 20 Course Content

Lesson Outline

Today we will learn:

1. How to create and use a custom "Pipe".

And solidify our knowledge of:

1. Template-Driven Forms.
2. Form Validation.
3. Ternary Operators.
4. The Angular Router.

Lesson Notes

- **Congratulations**: A *congratulations* is in order for making it this far in the course. Keep it up!
- **Pipes:** In Angular, *Pipes* are simple functions you can use in HTML template expressions, that accept an initial value and return a transformed value.

Course Project Steps

STEP 1: Adding the Bookshelf-Editor Form HTML

bookshelf/bookshelf-editor/bookshelf-editor.component.html:

- Create an HTML form tag with multiple "form-group" div's inside... each with a "label" and "input" for every field we want to capture. (One for "title", "author", "price", "genre", and "image").
- *Note*: Every input should have a "class", "type", "id", "name", and a "formControlName".
- Create a row beneath the last "form-group" div that displays the image from the URL above.
- Add a `#bookFormRef="ngForm"` and `(ngSubmit)="onSubmit(bookFormRef)"` properties on the "form" tag.
- *Note*: Make sure you have "FormsModule" imported in *app.module.ts* file and have restarted the server.
- Create two buttons: `<button>Save</button>` and `<button>Cancel</button>`.
- Have the "Save" button be of type submit, and it should be disabled if the "bookForm" isn't valid.

- The "Cancel" button should call a function `onCancel()` when clicked.

```
<form #bookFormRef="ngForm" (ngSubmit)="onSubmit(bookFormRef)">
  <!-- TITLE -->
  <div class="form-group">
    <label for="title">Title</label>
    <input type="text" class="form-control" id="title" name="title"
ngModel />
  </div>

  <!-- AUTHOR -->
  <div class="form-group">
    <label for="author">Author</label>
    <input type="text" class="form-control" id="author" name="author"
ngModel />
  </div>

  <!-- GENRE -->
  <div class="form-group">
    <label for="genre">Genre</label>
    <select name="genre" id="genre" class="form-control" ngModel>
      <option disabled selected value>Select a Genre</option>
      <option value="Mystery">Mystery</option>
      <option value="Science">Science</option>
      <option value="Non-Fiction">Non-Fiction</option>
      <option value="Fiction">Fiction</option>
    </select>
  </div>

  <!-- IMAGE INPUT -->
  <div class="form-group">
    <label for="coverImagePath">Image Path</label>
    <input
      type="text"
      class="form-control"
      id="coverImagePath"
      name="coverImagePath"
      ngModel
    />
  </div>

  <!-- IMAGE DISPLAY -->
  <div class="row my-2">
    <div class="col-xs-10 mx-auto">
      <img
        class="img-responsive"
        [src]="bookFormRef.value['coverImagePath']"
        [alt]="bookFormRef.value['title']"
      />
    </div>
  </div>

  <!-- BUTTONS -->
```

```
    <button
      type="submit"
      class="btn btn-success mr-3"
      [disabled]="!bookFormRef.valid"
    >
      Save
    </button>
    <button type="button" class="btn btn-danger" (click)="onResetForm()">
      Cancel
    </button>
  </form>
```

STEP 2: Adding the Bookshelf-Editor Form Typescript

bookshelf/bookshelf-editor/bookshelf-editor.component.ts:

- Create a new variable "bookDetails" that has empty strings or null for each input on the form.
- Create the `onSubmit(formObj: NgForm)` function and print to the console the "formObj" variable.
- Create the `onResetForm()` function setup and add code that resets our form. (Inject the Router from Angular and navigate to the previous "/".)
- Inside `ngOnInit()`, whenever the route params change, we want to check if we are in "isEditMode". If we are, the "bookDetails" variable should be set to the `bookshelfService.getBook(this.idx)` functions result.
- The `onSubmit(formObj: NgForm)` will populate the "this.bookDetails" variable with a `new Book(formObj.value.title, . . .)`.
- Create a conditional under where we populated the "this.bookDetails" variable and either call the "bookshelfService" method `updateBook(this.idx, this.bookDetails)` or `addBook(this.bookDetails)`... depending on what "mode" we are currently in.
-

```
import { BookshelfService } from "./../bookshelf.service";
import { Book } from "./../../shared/book/book.model";
import { Component, OnInit } from "@angular/core";
import { NgForm } from "@angular/forms";
import { ActivatedRoute, Params, Router } from "@angular/router";

@Component({
  selector: "app-bookshelf-editor",
  templateUrl: "./bookshelf-editor.component.html",
  styleUrls: ["./bookshelf-editor.component.css"],
})
export class BookshelfEditorComponent implements OnInit {
  idx: number;
  isEditMode = false;
```

```
  bookDetails: Book = {
    title: "",
    author: "",
    genre: "",
    coverImagePath: "",
  };

  constructor(
    private router: Router,
    private route: ActivatedRoute,
    private bookshelfService: BookshelfService
  ) {}

  ngOnInit(): void {
    this.route.params.subscribe((params: Params) => {
      this.idx = +params["id"];
      this.isEditMode = params["id"] != null;

      if (this.isEditMode) {
        this.bookDetails = this.bookshelfService.getBook(this.idx);
      }
    });
  }

  onSubmit(formObj: NgForm) {
    // Destructor book properties
    const { title, author, genre, coverImagePath } = formObj.value;

    // Set local bookDetails to formObj values
    this.bookDetails = new Book(
      book.title,
      book.author,
      book.genre,
      book.coverImagePath
    );

    // Conditional statement to call different methods/functions depending
on what "mode" we are in
    if (this.isEditMode) {
      // Edit book
      this.bookshelfService.updateBook(this.idx, this.bookDetails);
    } else {
      // Create new book
      this.bookshelfService.addBook(this.bookDetails);
    }

    // Reset Form
    this.onResetForm();
  }

  onResetForm() {
    this.router.navigate(["../"], { relativeTo: this.route });
  }
}
```

bookshelf/bookshelf-editor/bookshelf-editor.component.html:

- Bind to the `[ngModel]` properties for each input and set them equal to "bookDetails.VALUE-HERE".

```
<!-- . . . -->
<input name="title" [ngModel]="bookDetails.title" />
<!-- . . . -->
<input name="author" [ngModel]="bookDetails.author" />
<!-- . . . -->
<select name="genre" [ngModel]="bookDetails.genre"></select>
<!-- . . . -->
<input name="coverImagePath" [ngModel]="bookDetails.coverImagePath" />
```

STEP 3: Adding Two Bookshelf-Service Methods

bookshelf/bookshelf-service.component.ts:

- Create a method `updateBook(idx: number, updatedBook: Book)` that takes in the index to edit and the new information.
- Create a method `addBook(book: Book)` and adds a new book to the end of the "myBooks" array.
- *Note*: Make sure to emit the "bookListChanged" event to all methods that update the array.

```
addBook(book: Book) {
    this.myBooks.push(book);
    this.bookListChanged.next(this.myBooks.slice());
}

updateBook(idx: number, updatedBook: Book) {
    this.myBooks[idx] = updatedBook;
    this.bookListChanged.next(this.myBooks.slice());
}
```

STEP 4: Adding the Bookshelf-Editor Form Validation

bookshelf/bookshelf-editor/bookshelf-editor.component.html:

- Add "required" as an attribute for all fields you definitely want to have every time. (We will place it on the "title", "author", "genre", and "coverImagePath" inputs.)
- Add a span tag below each input that displays an appropriate error message. This should only be shown if the input is invalid and has been touched. We do this by adding an element referrence on each element tag and set it equal to "ngModel".

```
<!-- . . . -->
<input #titleRef="ngModel" [ngModel]="bookDetails.title" required />
<span class="help-block text-danger" *ngIf="!titleRef.valid &&
titleRef.touched"
  >Please enter a valid title!</span
>
<!--  . . . -->
<input #authorRef="ngModel" [ngModel]="bookDetails.author" required />
<span
  class="help-block text-danger"
  *ngIf="!authorRef.valid && authorRef.touched"
  >Please enter a valid author!</span
>
<!--  . . . -->
<select #genreRef="ngModel" [ngModel]="bookDetails.genre" required>
</select>
<span class="help-block text-danger" *ngIf="!genreRef.valid &&
genreRef.touched"
  >Please enter a valid genre!</span
>
<!-- . . . -->
<input
  #coverImagePathRef="ngModel"
  [ngModel]="bookDetails.coverImagePath"
  required
/>
<span
  class="help-block text-danger"
  *ngIf="!coverImagePathRef.valid && coverImagePathRef.touched"
  >Please enter a valid image path!</span
>
<!-- . . . -->
```

bookshelf/bookshelf-editor/bookshelf-editor.component.css:

- Create a class selector called `form .ng-invalid.ng-touched` that sets the "border" property to be "1px solid red". (This will put a red border on any element inside a form that is invalid and that the user has touched.)

```
form .ng-invalid.ng-touched {
  border: 1px solid crimson;
}
```

STEP 5: Creating a Custom Pipe

shared/pipes/sortBooks.pipe.ts file:

- Create a new folder called "pipes" inside the "shared" folder.

- Create a new pipe called "sortBooks.pipe.ts" inside the "shared/pipes" folder.
- Use the `@Pipe({})` Decorator to setup our pipe. Pass in the name "sortBooks". (This is how we will select it in our components.)
- Export a class called "SortBooksPipe" that "implements PipeTransform".
- Create a `transform(array: Book[], field: string): number | Book[] {` function. We will take in an array to sort and a field to sort that array by.
- Use the Javascript built in `sort((a,b))` function to sort through our array and return different values depenedent on their values.

```
import { Book } from "./../book/book.model";
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({
  name: "sortBooks",
})
export class SortBooksPipe implements PipeTransform {
  transform(array: Book[], field: string): number | Book[] {
    array.sort((a: Book, b: Book) => {
      if (a[field] < b[field]) {
        return -1;
      } else if (a[field] > b[field]) {
        return 1;
      } else {
        return 0;
      }
    });
    return array;
  }
}
```

app.module.ts file:

- Import and add the newly created "SortBooksPipe" in the "declarations" array.

```
declarations: [
    AppComponent,
    // . . .
    SortBooksPipe,
],
```

STEP 6: Using the SortBooksPipe

bookshelf/book-list/book-list.component.html:

- Next to the `*ngFor` loop, use the sort pipe and pass in a sortField variable we will create shortly.
- At the bottom of the screen, add a new button that calls an `onSort()` function that we will create on click.
- Use the "ternary" operator to dynamically change the text inside this button to either be "Title" or "Author". (These are the two fields we will be able to sort by.)
- Add some margin to the right of the first button so they are spaced out.

```
<div
  class="col-md-12"
  *ngFor="let bookElement of myBooks | sortBooks: sortField; let i =
index"
>
  <!-- . . . -->
</div>
<!-- . . . -->
<button class="btn btn-primary mr-3" (click)="onNewBook()">Add New
Book</button>
<button class="btn btn-primary" (click)="onSort()">
  Sort By {{ sortField === "author" ? "Title" : "Author" }}
</button>
```

bookshelf/book-list/book-list.component.ts:

- Create two new variables: `sortSwitcher = true` and `sortField = 'author'`.
- Create the `onSort()` function. This funtion should switch the "sortSwitcher" variable to the opposite of it's current value. After that it should conditionally change the "sortField" variable to either "author" or "title" depending on the state of the "sortSwitcher" variable.

```
  sortSwitcher = true;
  sortField = 'author';
//   . . .
onSort() {
    this.sortSwitcher = !this.sortSwitcher;

    if (this.sortSwitcher) {
      this.sortField = 'author';
    } else {
      this.sortField = 'title';
    }
  }
```

Additional Notes

Resources