Web Programming

3rd Activity - Intro to Angular

2020/2021

1. Data flow between components

@Input() and @Output() allow Angular to share data between the parent context and child directives or components. An @Input() property is writable while an @Output() property is observable.

@Input() and @Output() act as the API, or application programming interface, of the child component in that they allow the child to communicate with the parent. Think of @Input() and @Output() like ports or doorways—@Input() is the doorway into the component allowing data to flow in while @Output() is the doorway out of the component, allowing the child component to send data out.

2. Adding a model

a) Creating a model

We are storing data, so we want to have a model to structure our information, a blueprint or contract about the structure of our data. For that we need a model which is a file that we are going to create inside the posts folder.

post.model.ts

Our model is defined using an interface, which oblies that everyone who uses this model, has to follow these rules. An interface can't be instantiated.

post.model.ts

```
export interface Post {
  title: string;
  content: string;
}
```

Wherever we use this model, we'll have to import it, and that's why we need to make it exportable.

b) Using our model within our components

- To use our model we need to import it into every file that uses it.
- We will also sign every data that follows our model interface. In post-create we have two situations:
    - The EventEmitter type will be obliged to send a post model data
    - For this to happen we need to create an object that follows the model. For this we only have to say that our const post, is a Post

post-create.component.ts

```
import { Component, EventEmitter, Output } from
"@angular/core";
import { Post } from "../post.model";

@Component({
  selector: "app-post-create",
  templateUrl: "./post-create.component.html",
  styleUrls: ["./post-create.component.css"]
})
export class PostCreateComponent {
```

```
  enteredTitle = "";
  enteredContent = "";

  @Output() postCreated = new EventEmitter<Post>();

  onAddPost() {
    const post: Post = {
      title: this.enteredTitle,
      content: this.enteredContent
    };
    this.postCreated.emit(post);
  }
}
```

app.component.ts

```
import { Component } from "@angular/core";

import { Post } from "./posts/post.model";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  storedPosts: Post[] = [];

  onPostAdded(post) {
    this.storedPosts.push(post);
  }
}
```

post-list.component.ts

```
import { Component, Input } from "@angular/core";

import { Post } from "../post.model";

@Component({
  selector: "app-post-list",
  templateUrl: "./post-list.component.html",
  styleUrls: ["./post-list.component.css"]
})
export class PostListComponent {
  @Input() posts: Post[] = [];
}
```

Now we assure that we are working on the structure of data we want.

3. Form

a) Creating a Template-Driven Forms

If we analyse our post-create we have everything to use a form. A form is a special type of structure that contextualizes the user entered data. And angular has special features to work with forms. There are different ways to deal with forms,and we are going to begin with Template Driven Forms.

a) Creating Template-drive form

- Each field is a **ngModel** which is like to say that this input field is a control
- In the button we'll use **type=submit**, because now we have a form and angular knows that we have an object that will be sent, so now we don't need the click event. When submitting it will trigger a special event **(submit)** that we'll use in form to call out the **onAddPost()** function.
- In order to have access to the values of user data input we have to create a variable **#postform** that has access to the object that angular creates when

we are submitting. But, we need to say that to angular via #postFrom=”ngForm” and then pass it to **onAddPost(postform)** .

post-create.component.html

```
<mat-card>
  <form (submit)="onAddPost(postForm)" #postForm=”ngForm”>
    <mat-form-field>
      <input
matInput
name="title"
type="text"
ngModel />
    </mat-form-field>
    <mat-form-field>
      <textarea
matInput
name="content"
rows="6"
ngModel>
</textarea>
    </mat-form-field>
    <button
mat-raised-button
type="submit"
color="accent">
      Save Post</button>
  </form>
</mat-card>
```

b) Receiving data from form in our component ts file

Our **ngForm** is now sent via **onAddPost** and we should indicate the type of the form we have access to a lot of information, namely form.values.

post-create.component.ts

```
import { Component, EventEmitter, Output } from "@angular/core";
import { NgForm } from "@angular/forms";

import { Post } from "../post.model";

@Component({
  selector: "app-post-create",
  templateUrl: "./post-create.component.html",
  styleUrls: ["./post-create.component.css"]
})

export class PostCreateComponent {
  enteredTitle = "";
  enteredContent = "";

  @Output() postCreated = new EventEmitter<Post>();

  onAddPost(form: NgForm) {
    const post: Post = {
      title: form.value.title,
      content: form.value.content
    };
    this.postCreated.emit(post);
  }
}
```

There is also a nice feature about ngForm, we can perform some validations:

post-create.component.html

```
<mat-card>
  <form  #postForm="ngForm" (submit)="onAddPost(postForm)">
    <mat-form-field>
      <input matInput name="title" type="text" ngModel required minlength="3"
#title="ngModel">
<mat-error *ngIf="title.invalid">Please enter a valid post title</mat-error>
    </mat-form-field>
    <mat-form-field>
      <textarea matInput name="content" rows="6" ngModel required>
</textarea>
    </mat-form-field>
    <button mat-raised-button type="submit" color="accent">Save Post</button>
  </form>
</mat-card>
```

post-create.component.ts

```
import { Component, EventEmitter, Output } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Post } from '../post.model';

@Component({
selector: 'app-post-create',
templateUrl: './post-create.component.html',
styleUrls: [ './post-create.component.css' ]
})
export class PostCreateComponent {
enteredTitle = '';
enteredContent = '';

@Output() postCreated = new EventEmitter<Post>();

onAddPost(form: NgForm) {
  if (form.invalid) {
  return;
  }
  const post: Post = {
        title: form.value.title,
        content: form.value.content
  };
  this.postCreated.emit(post);
```

```
    }
  }
```

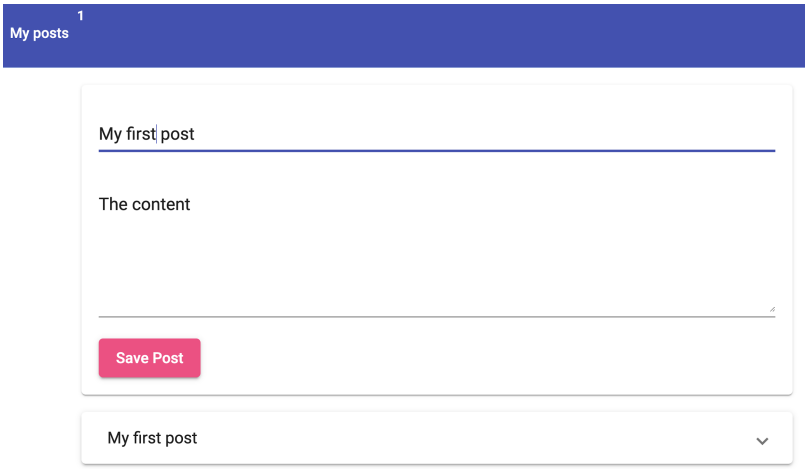
Here we validate if the form is valid, and if not, it will not proceed.

4. Exercise

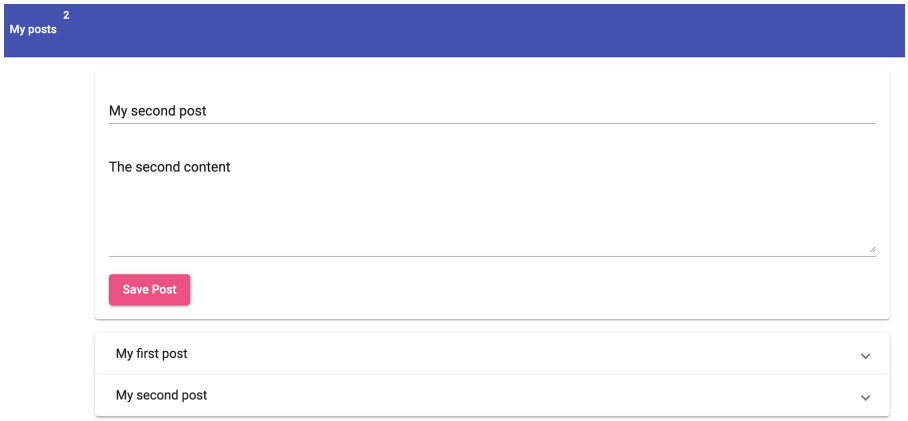
Create a badge in the header component that informs us the number of posts we've just added. Like in the following figures

Posts=0

Posts = 1


1
My posts
My first post
The content
Save Post
My first post

Posts = 2


2
My posts
My second post
The second content
Save Post
My first post
My second post