Angular

Routing

https://www.techiediaries.com/routing-angular-router/

Directives:

https://www.freecodecamp.org/news/angular-directives-learn-how-to-use-or-create-custom-directives-in-angular-c9b133c24442/

forms:

https://edupala.com/how-to-implement-angular-template-driven-form/ - important

https://edupala.com/angular-reactive-form-validation/

https://www.itsolutionstuff.com/post/angular-13-template-driven-forms-with-validation-exampleexample.html

Services and HTTP

https://www.itsolutionstuff.com/post/angular-13-service-tutorial-with-exampleexample.html

Reactive forms

https://codecraft.tv/courses/angular/forms/model-driven/

custom validation

https://codecraft.tv/courses/angular/advanced-topics/basic-custom-validators/

Custom Directives

https://www.freecodecamp.org/news/angular-directives-learn-how-to-use-or-create-custom-directives-in-angular-c9b133c24442/#:~:text=Creating%20a%20custom%20directive%20is,Component%20decorator%20with%20%40Directive%20decorator.

Pipes

https://www.positronx.io/how-to-create-custom-pipe-in-angular-application/

@Input and @Output

https://dzone.com/articles/component-communication-in-angular-day-1

npm config set legacy-peer-deps true

ng add @ng-bootstrap/ng-bootstrap

Angular forms

Angular provides two approaches to building forms: template-driven forms and reactive forms. In this answer, I'll focus on template-driven forms.

Template-driven forms are built using Angular's built-in ngForm directive. To create a template-driven form in Angular, you typically follow these steps:

Import the FormsModule module into your module.

typescript

Copy code

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { CommonModule } from '@angular/common';

@NgModule({

imports: [

CommonModule,

FormsModule

],

declarations: [ ... ],

exports: [ ... ]

})

export class MyModule { }

Add a form element to your template and bind it to an instance of ngForm.

html

Copy code

<form #myForm="ngForm">

<!-- form controls go here -->

</form>

Add form controls to your template using the ngModel directive.

html

Copy code

<form #myForm="ngForm">

<label for="name">Name</label>

<input type="text" id="name" name="name" [(ngModel)]="myModel.name">

<label for="email">Email</label>

<input type="email" id="email" name="email" [(ngModel)]="myModel.email">

</form>

Handle form submission by adding a submit event handler to your form.

html

Copy code

<form #myForm="ngForm" (ngSubmit)="onSubmit()">

<!-- form controls go here -->

<button type="submit">Submit</button>

</form>

typescript

Copy code

export class MyComponent {

myModel: any = {};

onSubmit() {

// handle form submission here

}

}

Reactive forms are built using the ReactiveFormsModule module, which provides a set of classes and directives that allow you to build and manage forms programmatically in your component code. To create a reactive form in Angular, you typically follow these steps:

Import the ReactiveFormsModule module into your module.

typescript

Copy code

import { NgModule } from '@angular/core';

import { ReactiveFormsModule } from '@angular/forms';

import { CommonModule } from '@angular/common';

@NgModule({

imports: [

CommonModule,

ReactiveFormsModule

],

declarations: [ ... ],

exports: [ ... ]

})

export class MyModule { }

Create a FormGroup object in your component code.

typescript

Copy code

import { Component } from '@angular/core';

import { FormGroup, FormControl } from '@angular/forms';

@Component({

selector: 'my-component',

templateUrl: 'my-component.html'

})

export class MyComponent {

myForm: FormGroup;

constructor() {

this.myForm = new FormGroup({

name: new FormControl(),

email: new FormControl()

});

}

}

Bind form controls to your template using the formControlName directive.

html

Copy code

<form [formGroup]="myForm">

<label for="name">Name</label>

<input type="text" id="name" formControlName="name">

<label for="email">Email</label>

<input type="email" id="email" formControlName="email">

</form>

Handle form submission by adding a submit event handler to your form.

html

Copy code

<form [formGroup]="myForm" (ngSubmit)="onSubmit()">

<!-- form controls go here -->

<button type="submit">Submit</button>

</form>

typescript

Copy code

export class MyComponent {

myForm: FormGroup;

constructor() {

this.myForm = new FormGroup({

name: new FormControl(),

email: new FormControl()

});

}

onSubmit() {

// handle form submission here

}

}

In this example, we have a simple form with two form controls: a name input and an email input. We create a FormGroup object in our component code, and we bind the values of the form controls to this group using the formControlName directive. We handle form submission using the (ngSubmit) event.

Note that with reactive forms, you have more fine-grained control over the state of the form and its controls. You can access and modify this state information using the FormGroup and FormControl classes provided by the ReactiveFormsModule module.

Angular Library

https://javascript.plainenglish.io/create-angular-library-2022-3965beee6dc6

https://indepth.dev/posts/1238/complete-beginner-guide-to-publishing-an-angular-library-to-npm

ng new my-library --create-application=false

ng generate library my-library --prefix=myLib

**ng g module new-lib**

**ng g c new-lib --project my-library**

**ng g application testing**

ng build my-library --configuration production

**npm login**

**npm publish**

**npm install my-library-sudha**

**How to use in a project.**

**In app.module**

import { MyLibraryModule } from 'my-library';

in component.html

<my-library-component></my-library-component>

ng new angular-pwa

ng add @angular/pwa

npm install --global http-server

ng build --configuration production

http-server -p 8080 -c-1 dist/project name

Interceptors

ng generate interceptor my-interceptor

Remember to import the interceptor in your module's providers array to make it available for use in your application.

Animation:

ngular provides a powerful animation module that allows you to create various animations within your application. These animations can be used to enhance user experience, create visually appealing transitions, and bring life to the user interface.

To use Angular animations, you need to import the **BrowserAnimationsModule** in your root module (**AppModule**) and inject it using the **imports** array.

Here's an example:

1. First, import the required animation-related modules:

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { NgModule } from '@angular/core';

Add the **BrowserAnimationsModule** in the **imports** array of your root module (**AppModule**):

@NgModule({

declarations: [

// Components, directives, etc.

],

imports: [

// Other modules

BrowserAnimationsModule // Import the BrowserAnimationsModule here

],

providers: [],

bootstrap: [AppComponent] // Your root component

})

export class AppModule { }

Now that the **BrowserAnimationsModule** is set up, you can create animations in your components or other parts of your application.

Angular provides various ways to create animations:

* **CSS Animations:** These animations are defined using CSS transitions or keyframes and can be triggered within Angular components using Angular's animation hooks.
* **JavaScript Animations:** You can use the Angular animation API to define animations programmatically in TypeScript/JavaScript.

Here's a simple example of creating a fade-in animation using Angular's animation API in a component:

import { Component, OnInit } from '@angular/core';

import { trigger, transition, style, animate } from '@angular/animations';

@Component({

selector: 'app-my-component',

template: `

<div [@fadeIn]>This element will fade in</div>

`,

animations: [

trigger('fadeIn', [

transition(':enter', [

style({ opacity: 0 }),

animate('500ms', style({ opacity: 1 }))

])

])

]

})

export class MyComponent implements OnInit {

constructor() { }

}