Table of Contents

[01 Install CLI tools locally and create Angular application 2](#_Toc167186219)

[02 Create a New Angular Component 3](#_Toc167186220)

[03 Communication Between Angular Components 4](#_Toc167186221)

[04 Create Angular Template 6](#_Toc167186222)

[05 Angular Template Control Flow Demo 9](#_Toc167186223)

[06 Create Angular Custom Pipe 11](#_Toc167186224)

[07 Create Angular Custom Directive 12](#_Toc167186225)

[08 Angular Dependency Injection Demo 14](#_Toc167186226)

[09 Angular Signals Demo 18](#_Toc167186227)

[10 Angular Router Demo 21](#_Toc167186228)

[11 Angular Reactive Form Demo 25](#_Toc167186229)

[12 Angular Template-driven Form Demo 31](#_Toc167186230)

[13 Angular HttpClient Demo 35](#_Toc167186231)

# 01 Install CLI tools locally and create Angular application

## 1. Setup Project

### 1.1 Create Workspace called “calab” with local CLI tools instance.

1. Type the following inside newly opened terminal window:

* npx -p @angular/cli ng new calab
* *You should see be asked to configure your workspace.*

1. Select the following options:

* Need to install the following packages:  
  @angular/cli@17.3.4  
  Ok to proceed? (y) y  
  ? Which stylesheet format would you like to use? CSS [https://developer.mozilla.org/docs/Web/CSS]  
  ? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? Yes

1. Change directory to “calab”:

* cd calab

1. Start your newly created Angular application using local Angular CLI instance.

* npx -p @angular/cli ng serve --host 0.0.0.0
* *–host 0.0.0.0 means that IP address 0.0.0.0 is used on servers to designate a service may bind to all network interfaces. It tells a server to “listen” for and accept connections from any IP address.*

# 02 Create a New Angular Component

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Component

### 2.1 Create Angular Component

1. Create a new component using CLI:

* npx -p @angular/cli ng generate component my-component

1. Import MyComponentComponent into src/app/app.component.ts:

* import { MyComponentComponent } from './my-component/my-component.component';

1. Inside src/app/app.component.ts update imports to include MyComponentComponent:

* imports: [RouterOutlet, MyComponentComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...

* <app-my-component></app-my-component>

### 2.2 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 03 Communication Between Angular Components

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Child and Parent Components

### 2.1 Create Child Component

1. Create a new Child component using CLI:

* npx -p @angular/cli ng generate component child

1. Open src/app/child/child.component.ts and add the following code:

* import { Component, EventEmitter, Input, Output } from '@angular/core';  
  @Component({  
   selector: 'app-child',  
   standalone: true,  
   imports: [],  
   templateUrl: './child.component.html',  
   styleUrl: './child.component.css'  
  })  
  export class ChildComponent {  
   @Input() message: string | undefined;  
   @Output() messageEvent = new EventEmitter<string>();  
    
   sendMessage() {  
   this.messageEvent.emit('Message from child to parent');  
   }  
  }

1. Open src/app/child/child.component.html and replace current html content with the following. Mention that we will cover Templates in more depth in next Topic:

* <h3>Child Component</h3>  
  <p>{{ message }}</p>  
  <button (click)="sendMessage()">Send Message to Parent</button>

### 2.2 Create Prent Component

1. Create a new Parent component using CLI:

* npx -p @angular/cli ng generate component parent

1. Open src/app/parent/paret.component.ts and add the following code:

* import { Component } from '@angular/core';  
  import { ChildComponent } from '../child/child.component';  
    
  @Component({  
   selector: 'app-parent',  
   standalone: true,  
   imports: [ChildComponent],  
   templateUrl: './parent.component.html',  
   styleUrl: './parent.component.css'  
  })  
  export class ParentComponent {  
   parentMessage = "Message from parent to child";  
   childMessage: string | undefined;  
    
   receiveMessage($event: any) {  
   this.childMessage = $event;  
   }  
  }

1. Open src/app/parent/parent.component.html and replace current html content with the following:

* <h2>Parent Component</h2>  
  <app-child-component [message]="parentMessage" (messageEvent)="receiveMessage($event)"></app-child-component>  
  <p>{{ childMessage }}</p>

### 2.3 Inject Parent component into AppComponent

1. Inside src/app/app.component.ts update imports to include ParentComponent:

* imports: [RouterOutlet, ParentComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...

* <app-parent></app-parent>

### 2.4 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0

# 04 Create Angular Template

## 1. Setup Project

### Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Component and Temlate

### 2.1 Create a new Component and a Template

1. Create a new CustomInputComponent component using CLI:

* npx -p @angular/cli ng generate component custom-input

### 2.2 Inject CustomInput Component into AppComponent

1. Inside src/app/app.component.ts update imports to include CustomInputComponent:

* imports: [RouterOutlet, CustomInputComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...

* <app-custom-input></app-custom-input>

### 2.3 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 3. Data Binding Examples

### 3.1 Text Interpolation Example

1. Open src/app/custom-input/custom-input.component.html and replace current html content with the following:

* <p>{{value}}</p>

1. Open src/app/custom-input/custom-input.component.ts and add the following code:

* export class CustomInputComponent {  
   value: string = "My Default Value"  
  }
* *Open your Angular application in Browser and see the result.*

### 3.2 Event Binding Example

1. Open src/app/custom-input/custom-input.component.html and add the following line just above <p>:

* <input type="text" (input)="onInputChange($event)">

1. Open src/app/custom-input/custom-input.component.ts and add the following code inside CustomInputComponent class:

* onInputChange(event: any) {  
   this.value = event.target.value;  
  }
* *Open your Angular application in Browser and see the result. Enter any value into the TextInput box and see how value chages on a screen.*

### 3.3 Property Binding Example

1. Open src/app/custom-input/custom-input.component.html and update current HTML:
   * add value property to <input> element:
   * <input type="text" [value]="value" (input)="onInputChange($event)">

* *Open your Angular application in Browser (refresh page if needed) and see “My Default Value” being pre populated in TextInput box.*

### 3.4 Two Way Binding Example

1. Open src/app/custom-input/custom-input.component.html and remove current <p>{{value}}</p> from Template.:
2. Open src/app/custom-input/custom-input.component.ts and update to the following code:

* import { Component, EventEmitter, Input, Output } from '@angular/core';  
   @Component({  
   selector: 'app-custom-input',  
   standalone: true,  
   imports: [],  
   templateUrl: './custom-input.component.html',  
   styleUrl: './custom-input.component.css'  
   })  
   export class CustomInputComponent {  
   @Input() value: string | undefined;  
   @Output() valueChange = new EventEmitter<string>();  
    
   onInputChange(event: any) {  
   this.value = event.target.value;  
   this.valueChange.emit(this.value);  
   }  
   }

1. Open src/app/app.component.ts and just below title variable add the following:

* inputValue: string = 'initial value';

1. Open src/app/app.component.html template and update/add the following element below the <div class="divider"...

* <app-custom-input [(value)]="inputValue"></app-custom-input>  
  <p>Input value: {{ inputValue }}</p>
* *Open your Angular application in Browser (refresh page if needed) and see how default value is set initialy, but later updated when new value is typed into TextInput box.*

### 3.5 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 05 Angular Template Control Flow Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Component and Template

### 2.1 Create a new Component and Template

1. Create a new CustomInputComponent component using CLI:

* npx -p @angular/cli ng generate component custom-input

### 2.2 Inject CustomInput Component into AppComponent

1. Inside src/app/app.component.ts update imports to include CustomInputComponent:

* imports: [RouterOutlet, CustomInputComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...

* <app-custom-input></app-custom-input>

### 2.3 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 3. Capture User Input and Update View Based On Users Value

### 3.1 Capture User Input

1. Open src/app/custom-input/custom-input.component.html and replace current HTML with the following:

* <input type="text" (input)="onInputChange($event)">

1. Open src/app/custom-input/custom-input.component.ts and add the following code inside CustomInputComponent class:

* import { Component } from '@angular/core';  
    
  @Component({  
   selector: 'app-custom-input',  
   standalone: true,  
   imports: [],  
   templateUrl: './custom-input.component.html',  
   styleUrl: './custom-input.component.css'  
  })  
  export class CustomInputComponent {  
   value: string = ''  
    
   onInputChange(event: any) {  
   this.value = event.target.value;  
   }  
  }

### 3.2 Add @if built-in control flow block

1. Open src/app/custom-input/custom-input.component.html and add the following code just below the <input> element:

* <div>  
   @if (!value || value.length < 5) {  
   Type at least 5 characters  
   } @else {  
   Perfect :)  
   }  
  </div>

### 3.3 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 06 Create Angular Custom Pipe

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Pipe

### 2.1 Create a new Pipe and Implement it’s logic

1. Create a new Reverse pipe using CLI:

* npx -p @angular/cli ng generate pipe reverse

1. Open src/app/reverse.pipe.ts file and replace current transform function with the following code:

* transform(value: string, ...args: unknown[]): unknown {  
   return value.split("").reverse().join("");;  
  }

### 2.2 Inject ReversePipe into AppComponent

1. Inside src/app/app.component.ts update imports to include Reverse:

* import {ReversePipe} from './reverse.pipe'  
  ...  
  imports: [RouterOutlet, ReversePipe],

### 2.3 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

1. Inspect the Rendered Screen, you should see title calab being reversed.

# 07 Create Angular Custom Directive

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create Directive

### 2.1 Create a new Directive and Implement it’s logic

1. Create a new IsAuth directive using CLI:

* npx -p @angular/cli ng generate directive is-auth

1. Open src/app/is-auth.directive.ts file and do the following:
   * Import TemplateRef, and ViewContainerRef from @angular/core.
   * Update current constructor with following parameters:
   * constructor(private templateRef: TemplateRef<any>, private viewContainer: ViewContainerRef) {   
     }

### 2.2 Implement @Input property

1. Open src/app/is-auth.directive.ts file and do the following:
   * Import Input from @angular/core.
   * Define an input property appIsAuth with a setter. This allows you to pass a condition to the directive from the template.
   * @Input() set appIsAuth(condition: boolean) {  
      if (condition) {  
      this.viewContainer.createEmbeddedView(this.templateRef);  
      } else {  
      this.viewContainer.clear();  
      }  
     }
   * *Use the setter to conditionally create or clear the embedded view in the ViewContainerRef based on the provided condition.*

## 3. Use Directive

### 3.1 Apply the Directive using Long-form syntax

1. Open src/app/is-auth.directive.html file and do the following:
   * Inside <div class="content"> insert the following:
   * <!-- Long-form syntax: -->  
     <ng-template [appIsAuth]="true">  
      <p>You have access to privileged information.</p>  
     </ng-template>
   * *In this example, the content within the <p> will only be rendered if the appIsAuth property in the component is true. Otherwise, the content will not be displayed.*

### 3.2 Apply the Directive using Short-form syntax

1. Open src/app/is-auth.directive.html file and do the following:
   * Inside <div class="content"> insert the following
   * <!-- Shorthand syntax: -->  
     <p \*appIsAuth="true">You have access to privileged information.</p>
   * *In this example, the content within the <p> will only be rendered if the appIsAuth property in the component is true. Otherwise, the content will not be displayed.*

### 3.3 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

1. Inspect the Rendered Screen, you should see title calab being reversed.

# 08 Angular Dependency Injection Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Creating Injectable Service

### 2.1 Create a new Course Service and Implement it’s logic

1. Create a new CourseService using CLI:

* npx -p @angular/cli ng generate service course/course

1. Open src/app/course/course.service.ts file and do the following:
   * Import Mock course data from the following file:
   * import COURSES from './MOCK\_COURSE\_DATA.json';
   * Add the following method just after the constructor:
   * getCourses(){  
      return COURSES;  
     }

### 2.2 Implement Course Model Class

1. Inside src/app/course folder create a new file called course.ts.
2. Open course.ts and add the following code:

* export class Course{  
   course\_id: number | undefined;  
   course\_name: string | undefined;  
   course\_description: string | undefined;  
   course\_duration: number | undefined;  
  }

### 2.3 Inject Course Service Into AppComponent

1. Open src/app/app.component.ts file and do the following:
   * Import Course model:
   * import { Course } from './course/course';
   * Inside AppComponent class declare variable called courses with type of list of Courses:
   * courses: Course[] = [];
   * Add the CourseService as a parameter into the constructor.
   * constructor(private courseService: CourseService){}
   * Fetch list of courses by calling getCourses() method from courseService and assign responce to variable declared above.
   * this.courses = courseService.getCourses();

### 2.4 Render List of Courses

1. Open src/app/app.component.html file and just below the <div class="divider"> add the following code that loops list of courses and renders each course in a list:

* <ul>  
   @for (course of courses; track course.course\_id) {  
   <li>  
   <div>  
   <p><strong>Id</strong> {{ course.course\_id }}</p>  
   <p><strong>Title: </strong>{{ course.course\_name }}</p>  
   <p><strong>Description: </strong>{{ course.course\_description }}</p>  
   <p><strong>Duration: </strong>{{ course.course\_duration }} weeks</p>  
    
   <div class="divider" role="separator" aria-label="Divider"></div>  
   </div>  
    
   </li>  
   }  
   </ul>

### 2.5 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

1. Inspect the Rendered Screen, you should see list of courses on your screen.

## 3. Injecting services in other services

### 3.1 Create a new Logger and Implement it’s logic

1. Create a new Logger using CLI:

* npx -p @angular/cli ng generate service logger/logger

1. Open src/app/logger/logger.service.ts file and do the following:
   * Create logging methods just below the constructor :
   * log(msg: unknown) { console.log(msg); }  
     error(msg: unknown) { console.error(msg); }  
     warn(msg: unknown) { console.warn(msg); }

### 3.2 Inject Logger Service Into Course Service and Log When Courses are Fetched

1. Open src/app/course/course.service.ts file and do the following:
   * Import Logger service:
   * import { LoggerService } from '../logger/logger.service';
   * Add the CourseService as a parameter into the constructor.
   * constructor(private logger: LoggerService) { }
   * Inside getCourses() method log that courses are getting fetched.
   * this.logger.log('Fetching Courses');

## 4. Configuring Dependency Providers

### 4.1 Creating Enhanced Logger

1. Create a new TimedLoggerService using CLI:

* npx -p @angular/cli ng generate service logger/timed-logger

1. Open src/app/logger/timed-logger.service.ts file and do the following:
   * Extend LoggerService with TimedLoggerService:
   * export class TimeLoggerService extends LoggerService {...}
   * Override logging methods just below the constructor:
   * constructor() {  
      super()  
     }  
     override log(msg: unknown) {   
      const date = Date.now();  
      console.log(`${date}: ${msg}`);   
     }  
     override error(msg: unknown) {   
      const date = Date.now();  
      console.error(`${date}: ${msg}`);   
     }  
     override warn(msg: unknown) {   
      const date = Date.now();  
      console.warn(`${date}: ${msg}`);   
     }

### 4.2 Configure an app-wide provider in the ApplicationConfig of bootstrapApplication, it overrides one configured for root in the @Injectable() metadata.

1. Open app.config.ts file and add the following:
   * Update providers with the following:

* providers: [provideRouter(routes), provideClientHydration(), {provide: LoggerService, useClass: TimeLoggerService}]

### 4.3 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

1. Inspect console and see whether your application logs with new Enhanced Timed Logger.

* 1714649534570: Fetching Courses

# 09 Angular Signals Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Creating a Signal

### 2.1 Create new Signal

1. Open src/app/app.component.ts file and do the following:
   * Import signal form '@angular/core'.
   * Inside constructor of AppComponent create a new signal and assign it to a variable:
   * const quantity = signal(0);

### 2.2 Read value from the Signal

1. Open src/app/app.component.ts file and do the following:
   * Import effect form '@angular/core'.
   * Inside constructor of AppComponent, just below Signal, call an effect and log the Signal value:
   * effect(() => {  
      console.log(`The current quantity is: ${quantity()}`);  
     });

### 2.3 Update Signal value

1. Open src/app/app.component.ts file and do the following:
   * Inside constructor of AppComponent, just below the effect, set new value to the Signal:
   * quantity.set(1);
   * Now use alternative approach to change value in Signal called update:
   * quantity.update(value => value + 1);

## 3. toSignal Example

### 3.1 Create an Observable

1. Open src/app/app.component.ts file and do the following:
   * Import Observable form rxjs.
   * Replace code inside constructor of AppComponentto the following:
   * let increasingQuantity = new Observable<number>(observer => {  
      let curVal = 0;  
      setInterval(() => observer.next(curVal +=1), 5000);  
     });

### 3.2 Convert Observable to Signal

1. Continue working with src/app/app.component.ts file:
   * Import toSignal form @angular/core/rxjs-interop.
   * Just below previously declared Observable, add the following:
   * let increasingQuantitySignal = toSignal(increasingQuantity, {initialValue: 0});

### 3.3 Read value from the newly converted Signal

1. Continue working with src/app/app.component.ts file:
   * Import effect form '@angular/core'.
   * Inside constructor of AppComponent, just below Signal, call an effect and log the Signal value:
   * effect(() => {  
      console.log(`The current quantity is: ${increasingQuantitySignal()}`);  
     });

## 4. toObservable Example

### 4.1 Create a Signal

1. Open src/app/app.component.ts file and do the following:
   * Import signal form @angular/core.
   * Replace code inside constructor of AppComponentto the following:
   * let quantity = signal(0);

### 4.2 Convert Signal to Observable

1. Continue working with src/app/app.component.ts file:
   * Import toObservable form @angular/core/rxjs-interop.
   * Just below previously declared Signal, add the following:
   * let quantityObservable = toObservable(quantity);

### 4.3 Subscribe to observable to get async updates

1. Continue working with src/app/app.component.ts file:
   * Just below previously converted signal into observable add the following:
   * quantityObservable.subscribe({  
      next: (v) => console.log(`Subscriber: The current quantity is: ${v}`),  
      error: (e) => console.error(e),  
      complete: () => console.info('complete')   
     });

### 4.4 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 10 Angular Router Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Setup Application for Routing

### 2.1 Create Two New Components

1. Create a new component using CLI and name it First:

* npx -p @angular/cli ng generate component components/first

1. Create another component using CLI and name it Second:

* npx -p @angular/cli ng generate component components/second

## 3. Define and Use Your Routes

### 3.1 Define Routes In Routes Array

1. Open src/app/app.routes.ts file and do the following:
   * Import previously created components.
   * import {FirstComponent} from './components/first/first.component';  
     import {SecondComponent} from './components/second/second.component';
   * Define each route as an JavaScript object and add it to Routes array:
   * { path: 'first-component', component: FirstComponent},  
     { path: 'second-component', component: SecondComponent}

### 3.2 Use Defined Routs In an Application

1. Open src/app/app.component.ts file and do the following:
   * Import RouterLink from the @angular/router.
2. Open src/app/app.component.html file and do the following:
   * Just below <div class="divider"... Use a routerLink attributes to add routes to selected elements:
   * <nav>  
      <ul>  
      <li><a routerLink="/first-component/MyName" >First Component</a></li>  
      <li><a routerLink="/second-component">Second Component</a></li>  
      </ul>  
     </nav>

### 3.3 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 4. Bind Route Info to Component Inputs

### 4.1 Enable Route Processing

1. Open src/app/app.config.ts file and do the following:
   * Import withComponentInputBinding from the @angular/router.
   * update provideRouter method with the following:
   * providers: [provideRouter(routes, withComponentInputBinding())]

### 4.2 Update the component to have an Input matching the name of the parameter

1. Open src/app/components/first/first.component.ts and add the following code:

* @Component({  
   selector: 'app-first',  
   standalone: true,  
   imports: [],  
   templateUrl: './first.component.html',  
   styleUrl: './first.component.css'  
  })  
  export class FirstComponent {  
   @Input() name = '';  
  }

### 4.3 Update path in router array to include path parameter or query string.

1. Open src/app/app.routes.ts file and do the following:
   * Update first-component path with the filowing:
   * { path: 'first-component/:name', component: FirstComponent},  
     { path: 'second-component', component: SecondComponent}
2. Open src/app/app.component.html file and do the following:
   * Update a First Component’s routerLink attribute to include name path parameter:
   * <li><a routerLink="/first-component/John" >First Component</a></li>

### 4.3 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 5. Nesting Routes

### 5.1 Create child components

1. Create a new component using CLI and name it FirstChild:

* npx -p @angular/cli ng generate component components/first-child

1. Add Styling to first child component:
   * Open src/app/components/first-child/first-child.component.ts and add replace current code with the following:
   * <div class="container">  
      <p>first-child works!</p>  
     </div>
   * Open src/app/components/first-child/first-child.component.html and add the following css:
   * .container {  
      background-color: bisque;  
      height: 100px;  
     }
2. Create another component using CLI and name it SecondChild:

* npx -p @angular/cli ng generate component components/second-child

1. Add Styling to second child component:
   * Open src/app/components/second-child/second-child.component.ts and add replace current code with the following:
   * <div class="container">  
      <p>second-child works!</p>  
     </div>
   * Open src/app/components/second-child/second-child.component.html and add the following css:
   * .container {  
      background-color:cadetblue;  
      height: 100px;  
     }

### 5.2 Set Up Child Routes

1. Open src/app/app.routes.ts file and do the following:
   * Import FirstChildComponent and SecondChildComponent.
   * Place child routes in a children array within the parent route:
   * { path: 'second-component', component: SecondComponent, children: [  
      {path: 'first-child', component: FirstChildComponent},  
      {path: 'second-child', component: SecondChildComponent}  
      ]   
     },

### 5.3 Update Parent

1. Open src/app/components/second/second.component.ts and add do the following:
   * import RouterOutlet and RouterLink from @angular/router.
2. Open src/app/components/second/second.component.html and add do the following:
   * Use a routerLink attributes to add routes to selected elements:
   * <nav>  
      <ul>  
      <li><a routerLink="first-child">First Child</a></li>  
      <li><a routerLink="second-child">Second Child</a></li>  
      </ul>  
     </nav>  
     <router-outlet></router-outlet>

### 5.4 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0

# 11 Angular Reactive Form Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create a Form Component

### 2.1 Create a New Component

1. Create a new component using CLI and name it OrderForm:

* npx -p @angular/cli ng generate component components/order-form

1. Import OrderForm into src/app/app.component.ts:

* import { MyComponentComponent } from './my-component/my-component.component';

1. Inside src/app/app.component.ts update imports to include OrderForm:

* imports: [RouterOutlet, OrderFormComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...:

* <app-order-form></app-order-form>

## 3. Setup Reactive Form in OrderFormComponent

### 3.1 Update Component

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Update imports to include ReactiveFormsModule:
   * @Component({  
      ...  
      imports: [ReactiveFormsModule],  
      ...  
     })
   * Create FormControl instancess inside your OrderFormComponent.
   * product = new FormControl('');  
     quantity = new FormControl('');
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Register controls in the template.
   * <label for="product">Product: </label>  
     <input id="product" type="text" [formControl]="product">  
       
     <label for="quantity">Quantity: </label>  
     <input id="quantity" type="text" [formControl]="quantity">
3. Open src/app/components/order-form/order-form.component.css file and do the following:
   * Add some style.
   * /\* Style inputs \*/  
     input, select {  
      width: 100%;  
      padding: 12px 20px;  
      margin: 8px 0;  
      display: inline-block;  
      border: 1px solid #ccc;  
      border-radius: 4px;  
      box-sizing: border-box;  
     }  
       
     /\* Style the submit button \*/  
     button[type=submit] {  
      width: 100%;  
      background-color: #04AA6D;  
      color: white;  
      padding: 14px 20px;  
      margin: 8px 0;  
      border: none;  
      border-radius: 4px;  
      cursor: pointer;  
     }  
       
     /\* Add a background color to the submit button on mouse-over \*/  
     button[type=submit]:hover {  
      background-color: #45a049;  
     }  
       
     .ng-valid[required], .ng-valid.required {  
      border-left: 5px solid #42A948; /\* green \*/  
     }  
     .ng-invalid:not(form) {  
      border-left: 5px solid #a94442; /\* red \*/  
     }

## 4. Displaying a Form Control Value

### 4.1 Update Component

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Subscribe to the valueChanges observable:
   * ngOnInit() {   
      this.product.valueChanges.subscribe(data => console.log(`Product updated to ${data}`))  
      this.quantity.valueChanges.subscribe(data => console.log(`Quantity updated to ${data}`))  
     }
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Access the current value directly through the value property.
   * <p>Current Product: {{product.value}}</p>  
     <p>Current Quantity: {{quantity.value}}</p>

## 5. Grouping Form Controls

### 5.1 Create A FormGroup Instance

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Create a FormGroup instance. Replace current code in OrderFormComponent class with the following:
   * orderForm = new FormGroup({  
      product: new FormControl(''),  
      quantity: new FormControl(''),  
     });
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Associate The FormGroup Model And View. Replace current HTML code with the following:
   * <form [formGroup]="orderForm">  
      <label for="product">Product: </label>  
      <input id="product" type="text" formControlName="product">  
      <label for="quantity">Quantity: </label>  
      <input id="quantity" type="text" formControlName="quantity">  
     </form>

### 5.2 Save form Data

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * create an onSubmit() callback method, allowing you to process the captured form data as needed:
   * onSubmit() {  
      // TODO: Use EventEmitter with form value  
      console.warn(this.orderForm.value);  
     }
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Add an ngSubmit event listener to the form tag with the onSubmit() callback method:
   * <form [formGroup]="orderForm" (submit)="onSubmit()">  
      ...  
     </form>
   * Use button within
   * element to trigger Submit event:
   * <form [formGroup]="orderForm" (submit)="onSubmit()">  
      ... // form elements  
      <button type="submit">Submit</button>  
     </form>

### 5.3 Use FormBuilder Service

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Inject the FormBuilder service into your component using dependency injection:
   * constructor(private formBuilder: FormBuilder){}
   * Replace FormGroup with FormBuilder:
   * orderForm = this.formBuilder.group({  
      product: [''],  
      quantity: [''],  
     });

### 5.4 Define A FormArray Control

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Add new property to your existing FormGroup:
   * instructions: this.formBuilder.array([this.formBuilder.control('')])
   * Create a getter that provides an efficient way to access the values in your form array instance:
   * get instructions() {  
      return this.orderForm.get('instructions') as FormArray;  
     }
   * Define a method to dynamically add an alias control to your form array:
   * addInstruction(){  
      this.instructions.push(this.formBuilder.control(''));  
     }
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Integrate the form model’s interests into your template:

* <form [formGroup]="orderForm" (submit)="onSubmit()">  
   ... //form controls  
   <div formArrayName="instructions">  
   <h2>Instructions</h2>  
   <button type="button" (click)="addInstruction()">+ Add Instruction</button>  
   @for (instruction of instructions.controls; track instruction; let i = $index) {  
   <div>  
   <label for="instruction-{{ i }}">Instruction:</label>  
   <input id="instruction-{{ i }}" type="text" [formControlName]="i">  
   </div>  
   }  
   </div>  
   ... //submit button  
  </form>

## 6. Reactive Form Validation

### 6.1 Built-in Validators

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Add required validators to product and quantity form controls. Update orderForm with the following:
   * orderForm = this.formBuilder.group({  
      product: ['', Validators.required],  
      quantity: ['', Validators.required],  
      instructions: this.formBuilder.array([this.formBuilder.control('')])  
     });

### 6.2 Custom Validators

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Create a new method that validates agains list of forbidden product names:
   * forbiddenNameValidator(): ValidatorFn {  
      return (control: AbstractControl): ValidationErrors | null => {  
      const list = ['Lightsaber', 'Millennium Falcon'];  
      const forbidden = list.includes(control.value);  
      return forbidden ? {forbiddenName: {value: control.value}} : null;  
      };  
     }
   * Add forbiddenNameValidator to product form controls.
   * product: ['', [Validators.required, this.forbiddenNameValidator()]],

### 6.3 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 12 Angular Template-driven Form Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

## 2. Create a Form Component

### 2.1 Create a New Component

1. Create a new component using CLI and name it OrderForm:

* npx -p @angular/cli ng generate component components/order-form

1. Import OrderForm into src/app/app.component.ts:

* import { MyComponentComponent } from './my-component/my-component.component';

1. Inside src/app/app.component.ts update imports to include OrderForm:

* imports: [RouterOutlet, OrderFormComponent],

1. Open src/app/app.component.html template and add the following element after the <div class="divider"...:

* <app-order-form></app-order-form>

## 3. Setup Template-driven Form in OrderFormComponent

### 3.1 Update Component

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Update imports to include FormsModule:
   * @Component({  
      ...  
      imports: [FormsModule],  
      ...  
     })

### 3.2 Create The Form Template

1. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Create a form template:
   * <form>  
      <label for="product">Product:</label>  
      <input type="text" id="product" name="product" required>  
      <label for="quantity">Quantity:</label>  
      <input type="text" id="quantity" name="quantity" required>  
      <button type="submit">Submit</button>  
     </form>
2. Open src/app/components/order-form/order-form.component.css file and do the following:
   * Add some style:
   * /\* Style inputs \*/  
     input, select {  
      width: 100%;  
      padding: 12px 20px;  
      margin: 8px 0;  
      display: inline-block;  
      border: 1px solid #ccc;  
      border-radius: 4px;  
      box-sizing: border-box;  
     }  
       
     /\* Style the submit button \*/  
     button[type=submit] {  
      width: 100%;  
      background-color: #04AA6D;  
      color: white;  
      padding: 14px 20px;  
      margin: 8px 0;  
      border: none;  
      border-radius: 4px;  
      cursor: pointer;  
     }  
       
     /\* Add a background color to the submit button on mouse-over \*/  
     button[type=submit]:hover {  
      background-color: #45a049;  
     }  
       
     .ng-valid[required], .ng-valid.required {  
      border-left: 5px solid #42A948; /\* green \*/  
     }  
     .ng-invalid:not(form) {  
      border-left: 5px solid #a94442; /\* red \*/  
     }

### 3.3 Data Bind With ngModel

1. Create a new class inside src/app/components/order-form/ representing an Order Model:

* export class Order {  
   constructor(  
   public product: string,  
   public quantity?: number,  
   ) {}  
  }

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Declare a model that you want to bind to the template:
   * order = new Order('');
   * *default product set to empty string.*
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Update a form template with the following:
   * <form>  
      <label for="product">Product:</label>  
      <input type="text" id="product" name="product" required [(ngModel)]="order.product">  
      <label for="quantity">Quantity:</label>  
      <input type="text" id="quantity" name="quantity" required [(ngModel)]="order.quantity">  
      <button type="submit">Submit</button>  
     </form>

### 3.4 Submitting the Form

1. Open src/app/components/order-form/order-form.component.ts file and do the following:
   * Create an onSubmit() callback method, allowing you to process the captured form data as needed:
   * onSubmit(){  
      console.log(this.order);  
     }
2. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Add an ngSubmit event listener to the form tag with the onSubmit() callback method:
   * <form (ngSubmit)="onSubmit()">  
     ...

## 4. Template-driven Form Validation

### 4.1 Show And Hide Validation Error Messages

1. Open src/app/components/order-form/order-form.component.html file and do the following:
   * Add a local reference to the input #product="ngModel":
   * <input type="text" id="product" name="product" required [(ngModel)]="order.product" #product="ngModel">
   * Add a conditional error message to product:
   * <div [hidden]="product.valid || product.pristine">  
      Product name is required  
     </div>

### 4.2 Review Changes

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

# 13 Angular HttpClient Demo

## 1. Setup Project

### 1.1 Install Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

### 1.2 Start JSON Server

1. Start JSON server:

* npx json-server MOCK\_DATA.json -p 3001

### 1.3 Install Angular Dependencies

1. Change directory to calab:

* cd calab

1. Install dependencies by running the following command:

* npm install

### 1.4 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 2. Setup HttpClient

### 2.1 Update App Configuration

1. Open src/app/app.config.ts file and do the following:
   * Provide provideHttpClient helper function:
   * export const appConfig: ApplicationConfig = {  
      providers: [ provideRouter(routes), provideHttpClient() ]  
     };

## 3. Create and configure a new Service

### 3.1 Create a new Service

1. Create a new component using CLI and name it MovieService:

* npx -p @angular/cli ng generate service services/movie

### 3.2 Inject The HttpClient Service

1. Open src/app/services/movie.service.ts file and do the following:
   * Inject HttpClient as a dependency into MovieService constructor.
   * export class MovieService {  
      constructor(private httpClient: HttpClient) { }  
     }

## 4. Create a Movie Model

### 4.1 Create a new class representing a movie model

1. Create a new directory in src/app/ called models.
2. Change directory to src/app/models
3. Create new TypeScript file called movie.ts
4. Add the following code inside movie.ts:

* export class Movie {  
   constructor(  
   public title: string,  
   public genre: string,  
   public release\_date: string,  
   public director: string,  
   public rating: number,  
   public duration\_minutes: number,  
   ) {}  
  }

## 5. Making Http Requests

### 5.1 Create a GET http request

1. Open src/app/services/movie.service.ts file and do the following:
   * Declare a new function called getAllMovies that calls http get() method.
   * getAllMovies(){  
      this.httpClient.get<Movie>('http://localhost:3001/movies').subscribe(data => {  
      console.log(data);  
      });  
     }

### 5.2 Call method containing GET http request

1. Open src/app/app.component.ts file and do the following:
   * Inject MovieService as dependency into AppComponent.
   * export class AppComponent {  
      constructor(private movieService: MovieService){}  
      ...  
     }
   * Inside constructor, make a call to getAllMovies() method.
   * export class AppComponent {  
      constructor(private movieService: MovieService){  
      movieService.getAllMovies();  
      }  
      ...  
     }

### 5.3 Create a POST http request

1. Open src/app/services/movie.service.ts file and do the following:
   * Declare a new function called createMovie that calls http post() method.
   * this.httpClient.post<Movie>('http://localhost:3001/movies', movie).subscribe(res => {  
      console.log('Created movie:', res);  
     });

### 5.4 Call method containing POST http request

1. Open src/app/app.component.ts file and do the following:
   * Inside constructor, create an instance of a movie.
   * const movie = new Movie(   
      "Forrest Gump",  
      "Drama",  
      "1994",  
      "Robert Zemeckis",  
      8.8,  
      142  
     );
   * Inside constructor, make a call to createMovie() method.
   * movieService.createMovie(movie);

### 5.5 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*

## 6. Interceptors

### 6.1 Define an Interceptor

1. Open src/app/app.config.ts file and do the following:
   * Define a loggingInterceptor helper function:
   * export function loggingInterceptor(req: HttpRequest<unknown>, next: HttpHandlerFn): Observable<HttpEvent<unknown>> {  
      console.log(`Request URL is: ${req.url}`);  
      return next(req);  
     }
   * Declare an interceptor inside provideHttpClient helper function:
   * export const appConfig: ApplicationConfig = {  
      providers: [provideRouter(routes), provideHttpClient(  
      withInterceptors([loggingInterceptor]),  
      ),]  
     };

### 6.2 Start The Application

1. Start Angular Development Server if not yet started:

* npx -p @angular/cli ng serve --host 0.0.0.0
* *Otherwise refresh the browser tab to see updated view.*