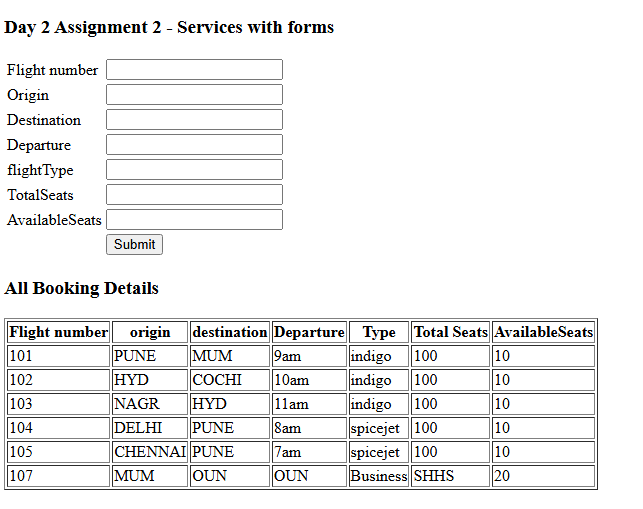
<https://codeshare.io/29XVRg>

1)Sevice as injection  
  
use service to store records On submit add new record to table below

Component.ts  
@Component({

  selector: 'app-new-flight-list',

  templateUrl: './new-flight-list.component.html',

  styleUrl: './new-flight-list.component.css'

})

export class NewFlightListComponent {

  flightService : flightService | undefined;

  flights : flight[] | undefined;

  public model : any ={};

  public bookingList : Array<any> = [];

  constructor(){

    this.flightService = new flightService();

    }

  ngOnInit(){

    this.getFlights();

  }

  getFlights() : any {

     this.flights = this.flightService?.getFlights();

  }

  bookflight():void{

   this.flights?.push(this.model);

   this.model = {}

  }

  getAllBookingDetails() : any{

    return this.flights;

 }

}

//service

import { flight } from "./flight";

export class flightService{

     public getFlights(){

            let flights : flight[];

            flights = [new flight(101,'PUNE','MUM','9am','indigo',100,10),

            new flight(102,'HYD','COCHI','10am','indigo',100,10),

            new flight(103,'NAGR','HYD','11am','indigo',100,10),

            new flight(104,'DELHI','PUNE','8am','spicejet',100,10),

            new flight(105,'CHENNAI','PUNE','7am','spicejet',100,10),

            ]

            return flights;

     }

}

//html code

<h3> Day 2  Assignment 2 - Services with forms</h3>

<div>

<table>

    <thead>

        <tr>

            <td>Flight number </td> <td><input type="text" [(ngModel)]="model.flightNumber"></td>

        </tr>

        <tr>

            <td>Origin</td> <td><input type="text" [(ngModel)]="model.origin"></td>

        </tr>

        <tr>

            <td>Destination</td> <td><input type="text" [(ngModel)]="model.destination"></td>

        </tr>

        <tr>

            <td>Departure</td> <td><input type="text" [(ngModel)]="model.deptTime"></td>

        </tr>

        <tr>

            <td>flightType</td> <td><input type="text" [(ngModel)]="model.flightType"></td>

        </tr>

        <tr>

            <td>TotalSeats</td> <td><input type="text" [(ngModel)]="model.totalSeats"></td>

        </tr>

        <tr>

            <td>AvailableSeats</td> <td><input type="text" [(ngModel)]="model.availableSeats"></td>

        </tr>

        </thead>

        <tbody>

            <tr>

                <td></td>

                <td>

                <button type="button" value="Submit" (click)="bookflight()">Submit</button>

                </td>

            </tr>

        </tbody>

</table>

<h3>All Booking Details</h3>

<div>

  <table border="1">

    <thead>

      <tr>

        <th><span>Flight number</span></th>

        <th><span>origin</span></th>

        <th><span>destination</span></th>

        <th><span>Departure</span></th>

        <th><span>Type</span></th>

        <th><span>Total Seats </span></th>

        <th><span>AvailableSeats</span></th>

      </tr>

    </thead>

    <tbody>

      <tr \*ngFor="let f of flights">

        <td >{{f.flightNumber}}</td>

        <td>{{f.origin}}</td>

        <td>{{f.destination}}</td>

        <td>{{f.deptTime}}</td>

        <td>{{f.flightType}}</td>

        <td>{{f.totalSeats}}</td>

        <td >{{f.availableSeats}}</td>

      </tr>

    </tbody>

  </table>

</div>

</div>

===== Simple Form ===================

component.html:

<p>contact-form works!</p>

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

<div>

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

<input type="text" id="name" name="name" [(ngModel)]="contact.name" required/>

</div>

<div>

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

<input type="text" id="email" name="email" [(ngModel)]="contact.email" required/>

</div>

<div>

<label for="message"> message </label>

<textarea id="message" name="message" [(ngModel)]="contact.message" required>

</textarea>

</div>

<button type="submit" [disabled]="!contactForm.valid"> Submit </button>

</form>

<div \*ngIf="submitted">

<h3> Form Submitted </h3>

<p> Name : {{contact.name}}</p>

<p> Email : {{contact.email}}</p>

<p> Message : {{contact.message}}</p>

</div>

component.ts:

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

@Component({

selector: 'app-contact-form',

templateUrl: './contact-form.component.html',

styleUrl: './contact-form.component.css'

})

export class ContactFormComponent {

contact = {

name: '',

email :'',

message:''

};

submitted= false

onSubmit(form:any) :void {

if(form.valid){

this.submitted = true;

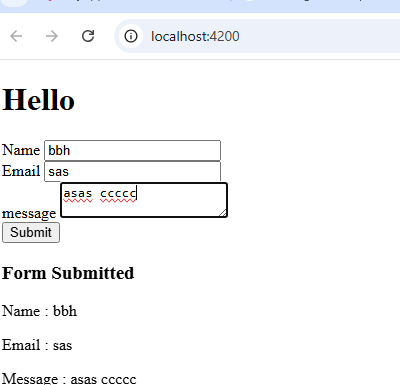
console.log(' Form data' , this.contact)

}

}

}

O/p



2) Form with formgroup AND customized validation using ngModel in for

<p>product-form works!</p>

<form #productForm="ngForm" (ngSubmit)="onSubmit(productForm)" novalidate>

<div>

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

<input type="text" id="name" name="name" class="form-control" [(ngModel)]="product.name" required minlength="3" #name="ngModel"/>

<div \*ngIf="name.invalid && name.touched" class="text-danger">

<div \*ngIf="name.errors?.['required']">

Product Name is required

</div>

<div \*ngIf="name.errors?.['minlength']">

Minimum 3 chracters is required

</div>

</div>

</div>

<div>

<label for="price"> Product price </label>

<input type="number" id="price" name="price" class="form-control" [(ngModel)]="product.price" required min="5" max="100" #price="ngModel"/>

<div \*ngIf="price.invalid && price.touched" class="text-danger">

<div \*ngIf="price.errors?.['required']" >

Price is required

</div>

<div \*ngIf="price.errors?.['min']" >

minimum price is 10.00

</div>

</div>

</div>

<div>

<label for="category"> Product Category</label>

<select type="text" id="category" name="category" class="form-select" [(ngModel)]="product.category" required #category="ngModel">

<option value="" > Select Category </option>

<option value="Electronics"> Electronics </option>

<option value="Clothing"> Clothing</option>

<option value="Home"> Home Decor </option>

</select>

<div \*ngIf="category.invalid && category.touched" class="text-danger">

<div \*ngIf="category.errors?.['required']">

Category is required

</div>

</div>

</div>

<div>

<label for="description"> Description </label>

<textarea id="description" name="description" class="form-control" [(ngModel)]="product.description" required row="3">

</textarea>

</div>

<button type="submit" [disabled]="productForm.invalid"> Submit </button>

</form>

=========================

form.component.ts

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

import { Product } from './product.model';

@Component({

selector: 'app-product-form',

templateUrl: './product-form.component.html',

styleUrl: './product-form.component.css'

})

export class ProductFormComponent {

product = new Product();

onSubmit(form:any) {

if(form.valid){

console.log(' product is submitted', this.product)

//form.resetForm()

}

}

}

==========

//product.model.ts – model class

export class Product {

constructor(

public name:string=' ',

public price: number=0,

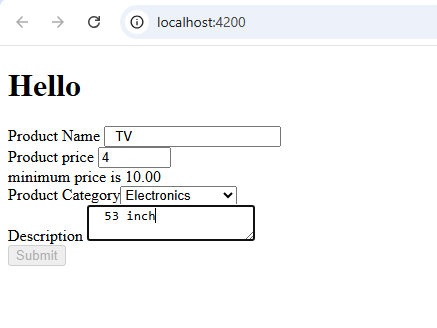
public category:string =' ',

public description:string =' '

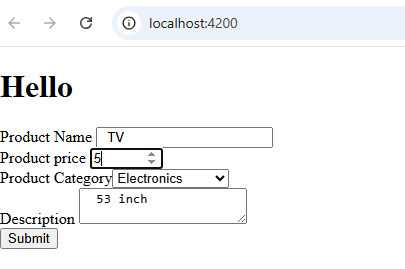
){}

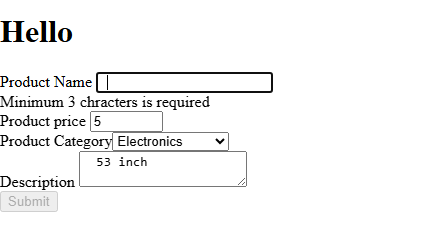
}

o/p



Min price is 5 so submit button is disabled here. If we enter 5 button will be enabled.



Customized validated msg for product name :  


4) Data sharing from parent to child , child to parent  
use case : Pass flight details to child through @input and booking event (just simple msg) back to parent using @Output

//Parent.ts

export class FlightDetailsSharingComponent {

 flightService : flightService | undefined

 flights!: flight[] | undefined;

 flightDetails : flight[] |undefined

 bookingMsg : string =''

  constructor(){

    this.flightService = new flightService();

    this.bookingMsg = '';

  }

  ngOnInit(){

    this.getFlights();

  }

  bookflight(){

      this.flightDetails = this.flights ;

  }

  getFlights() : any {

    this.flights = this.flightService?.getFlights();

 }

 receiveBooking(event :any){

  alert("received")

  this.bookingMsg = event;

 }

}

//parent.html

<h3>All Booking Details - Parent</h3>

<div>

  <table border="1">

    <thead>

      <tr>

        <th><span>Flight number</span></th>

        <th><span>Origin</span></th>

        <th><span>Destination</span></th>

        <th><span>Departure</span></th>

        <th><span>Type</span></th>

        <th><span>Total Seats </span></th>

        <th><span>AvailableSeats</span></th>

      </tr>

    </thead>

    <tbody>

      <tr \*ngFor="let f of flights">

        <td >{{f.flightNumber}}</td>

        <td>{{f.origin}}</td>

        <td>{{f.destination}}</td>

        <td>{{f.deptTime}}</td>

        <td>{{f.flightType}}</td>

        <td>{{f.totalSeats}}</td>

        <td >{{f.availableSeats}}</td>

      </tr>

    </tbody>

  </table>

  <button type="button" value="Submit" (click)="bookflight()"> Send flight details</button>

</div>

<div>{{bookingMsg}}</div>

<app-flight-details-show [flightDetailsToChild]="flightDetails" (childevent)="receiveBooking($event)"></app-flight-details-show>

//child compont.ts

export class FlightDetailsShowComponent {

  @Input() // get data from parent

  flightDetailsToChild : flight[] |undefined

  @Output() //send event to parent

  childevent = new EventEmitter();

  bookflight(){

    this.childevent.emit("Booking is successful");

  }

}

//child component.html

<h2>This is child component - Display flights details received from parent</h2>

<div>

    <table border="1">

      <thead>

        <tr>

          <th><span>Flight number</span></th>

          <th><span>Origin</span></th>

          <th><span>Destination</span></th>

          <th><span>Departure</span></th>

          <th><span>Type</span></th>

          <th><span>Total Seats </span></th>

          <th><span>AvailableSeats</span></th>

        </tr>

      </thead>

      <tbody>

        <tr \*ngFor="let f of flightDetailsToChild">

          <td >{{f.flightNumber}}</td>

          <td>{{f.origin}}</td>

          <td>{{f.destination}}</td>

          <td>{{f.deptTime}}</td>

          <td>{{f.flightType}}</td>

          <td>{{f.totalSeats}}</td>

          <td >{{f.availableSeats}}</td>

        </tr>

      </tbody>

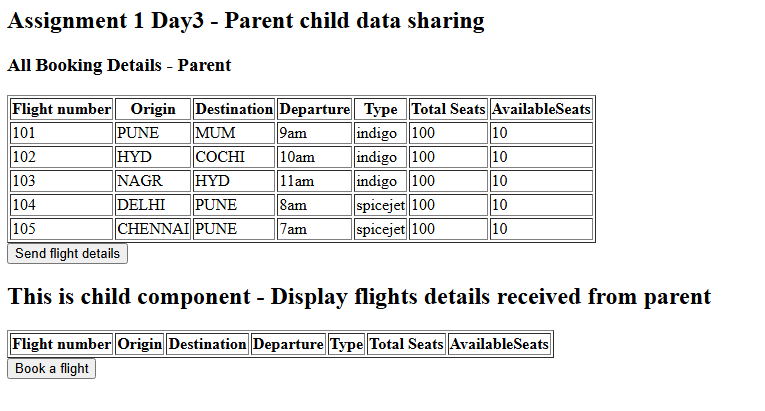
    </table>

    <button type="button" value="Submit" (click)="bookflight()"> Book a flight</button>

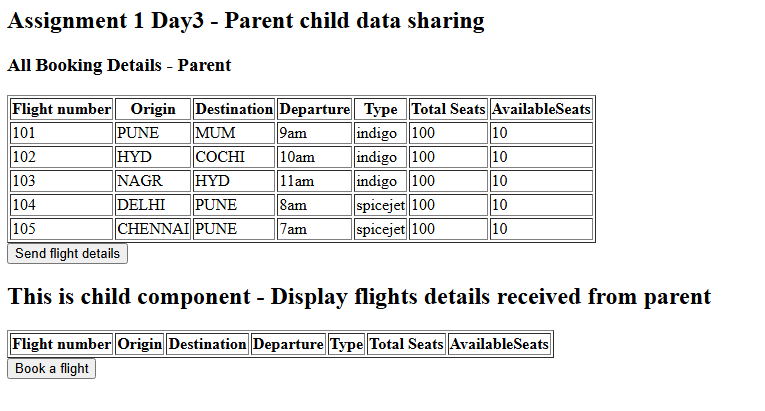
Alson entry in app.component.ts for parent compoenent

//Output

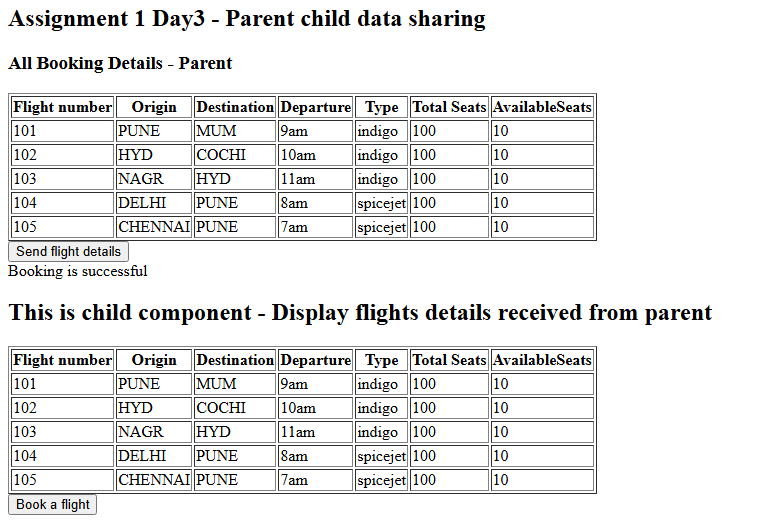
Page load()



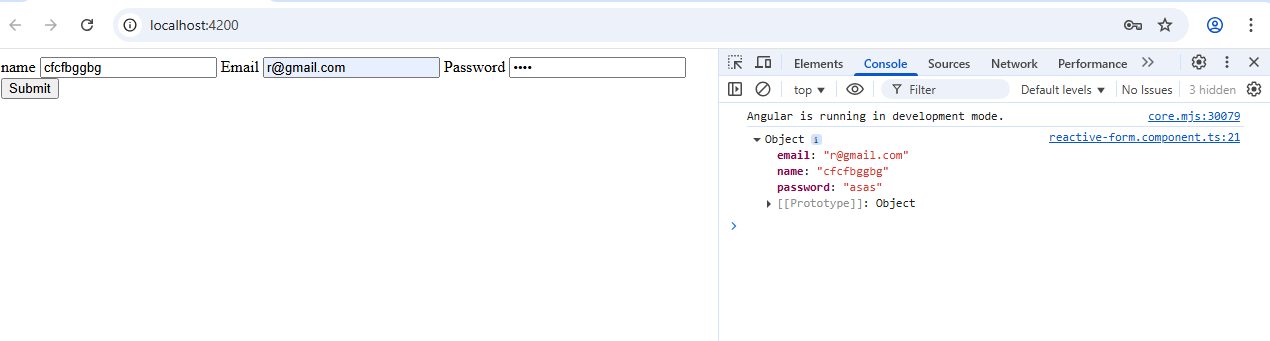
Send file button clicked



On book flight click , send booking msg back to parent



5) reactive simple form.



App.module.ts

@NgModule({

  declarations: [

    AppComponent,

    FlightListComponent,

    NewFlightListComponent,

    FlightFormComponent,

    FlightDetailsSharingComponent,

    FlightDetailsShowComponent,

    ReactiveFormComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule,

    ReactiveFormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

component.ts

export class ReactiveFormComponent {

  myForm!: FormGroup; // ! means no initialization required for this variable

  ngOnInit(): void { // here separate control for each UI field

    this.myForm = new FormGroup({

      name : new FormControl('', Validators.required),

      email: new FormControl('', [Validators.required, Validators.email]),

      password: new FormControl('', Validators.required)

    })

  }

  onSubmit(){

    console.log(this.myForm.value)

  }

}

//html- with FormControl a separate control for each UI field. We are using predefined validator in ts file.

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

    <label> name </label>

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

    <label> Email </label>

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

    <label> Password </label>

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

    <button type="submit" [disabled]="myForm.invalid"> Submit </button>

</form>

6) Reactive form with FormGroup and for builder – formgroup is to manage all field in one group , no separate control is required like previous example . Add some custom validation in HTML file

//html file – No separate control defined here for any input field

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

<label> Product name </label>

<input formControlName="name">

<div \*ngIf="f.name.touched && f.name.invalid">

<small \*ngIf="f.name.errors?.required"> Nmae is required</small>

<small \*ngIf="f.name.errors?.minlength"> Minimum 3 characters are required</small>

</div>

<label> Price </label>

<input type="number" formControlName="price">

<div \*ngIf="f.price.touched && f.price.invalid">

<small \*ngIf="f.price.errors?.required"> Price is required</small>

<small \*ngIf="f.price.errors?.minlength"> Minimum price should be 10.00</small>

</div>

<label> Description </label>

<textarea formControlName="description"></textarea>

<label> Category </label>

<select formControlName="category">

<option value=""> Select </option>

<option value="electronics"> Electronics </option>

<option value="books"> Books </option>

<option value="clothing"> Clothing </option>

</select>

<div \*ngIf="f.category.touched && f.category.invalid">

<small> category is required</small>

</div>

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

</form>

//ts file

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

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

@Component({

selector: 'app-reactive-form',

templateUrl: './reactive-form.component.html',

styleUrl: './reactive-form.component.css'

})

export class ReactiveFormComponent {

productForm!: FormGroup;

constructor(private fb:FormBuilder)

{

this.productForm = this.fb.group({

name: ['', [Validators.required, Validators.minLength(3)]],

price: [null,[Validators.required,Validators.min(10.00)]],

description:[''],

category:['',Validators.required]

})

}

onSubmit(){

if(this.productForm.valid){

console.log(' Product data is ', this.productForm.value)

alert(' Product saved !!')

}

else{

this.productForm.markAllAsTouched(); // predefined method

}

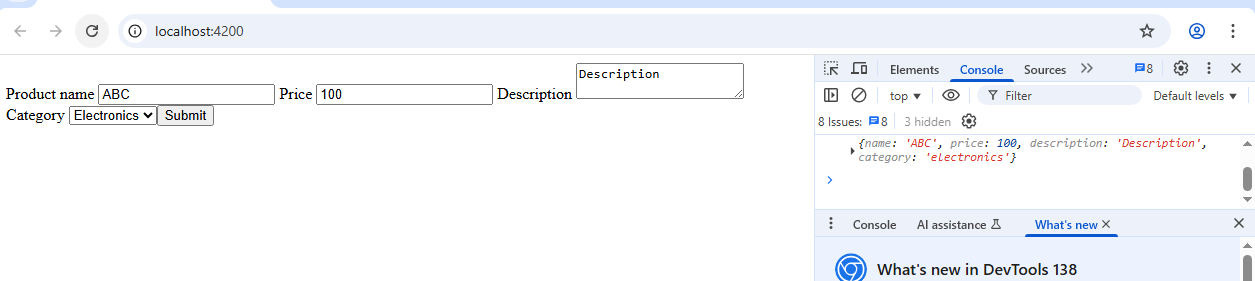
}

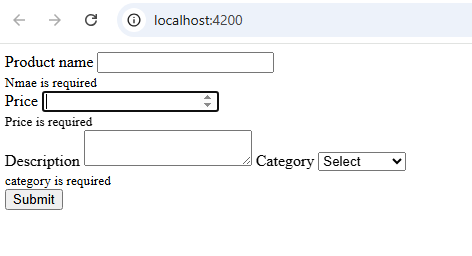
get f(){

return this.productForm.controls;

}

}

  
All validation errors we can see all in one go .



7) Routing   
Step 2: Setup Routes in app-routing.module.ts

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

import { RouterModule, Routes } from '@angular/router';

import { ProductsComponent } from './products/products.component';

import { ProductDetailComponent } from './product-detail/product-detail.component';

const routes: Routes = [

{ path: 'products', component: ProductsComponent },

{ path: 'products/:id', component: ProductDetailComponent },

{ path: '', redirectTo: '/products', pathMatch: 'full' },

{ path: '\*\*', redirectTo: '/products' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule {}

\*\*Explanation:

/products: Static route to display all products using ProductsComponent.

/products/:id: Dynamic route that uses :id as a route parameter (e.g., /products/2) and shows product details.

'': Default path redirects to /products.

\*\*: Wildcard for undefined routes → redirects to /products.

Step 3: Product Data in a Service

Create service: ---> ng generate service product

3.1) Go to --> product.service.ts:

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

@Injectable({

providedIn: 'root'

})

export class ProductService {

private products = [

{ id: 1, name: 'Laptop', price: 999 },

{ id: 2, name: 'Smartphone', price: 699 },

{ id: 3, name: 'Headphones', price: 199 }

];

getProducts() {

return this.products;

}

getProductById(id: number) {

return this.products.find(p => p.id === id);

}

}

Step 4: Show Product List

4.1) Go to ---> products.component.ts

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

import { ProductService } from '../product.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-products',

templateUrl: './products.component.html'

})

export class ProductsComponent implements OnInit {

products: any[] = [];

constructor(private productService: ProductService, private router: Router) {}

ngOnInit(): void {

this.products = this.productService.getProducts(); // this will loading all the predefined products from the service class

}

goToDetails(id: number) {

this.router.navigate(['/products', id]);

}

}

\*\*Explanation:

goToDetails() uses Angular's Router to navigate to dynamic detail route

4.2) Go to --> products.component.html:

<h2>Product List</h2>

<ul>

<li \*ngFor="let product of products">

{{ product.name }} - ${{ product.price }}

<button (click)="goToDetails(product.id)">View Details</button>

</li>

</ul>

Step 5: Show Product Details

5.1) Go to ---> product-detail.component.ts:

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

import { ActivatedRoute } from '@angular/router';

import { ProductService } from '../product.service';

@Component({

selector: 'app-product-detail',

templateUrl: './product-detail.component.html'

})

export class ProductDetailComponent implements OnInit {

product: any;

constructor(

private route: Activated, prRoute,

private productService: ProductService

) {}

ngOnInit(): void {

const id = Number(this.route.snapshot.paramMap.get('id'));

this.product = this.productService.getProductById(id);

}

}

5.2) Go to --> product-detail.component.html:

<h2>Product Detail</h2>

<div \*ngIf="product">

<p><strong>Name:</strong> {{ product.name }}</p>

<p><strong>Price:</strong> ${{ product.price }}</p>

</div>

<div \*ngIf="!product">

<p>Product not found.</p>

</div>

Step 6 : Go to app.component.html

<router-outlet></router-outlet>

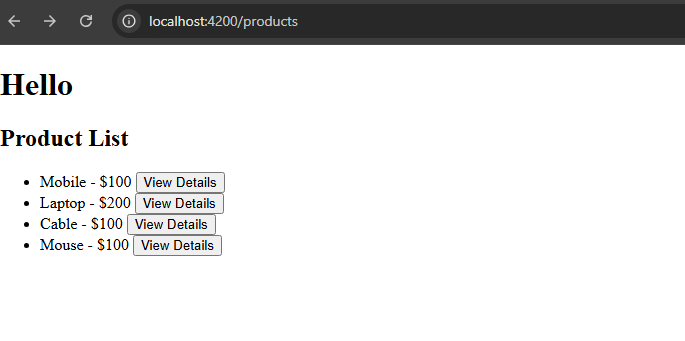
Step 7: Run the App

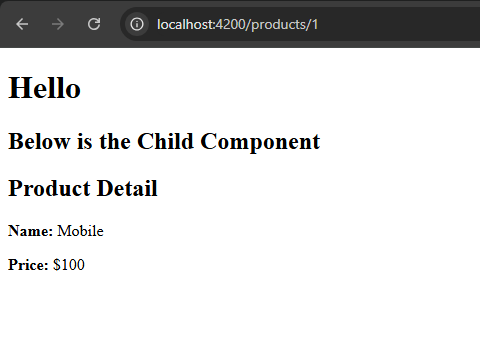
ng serve

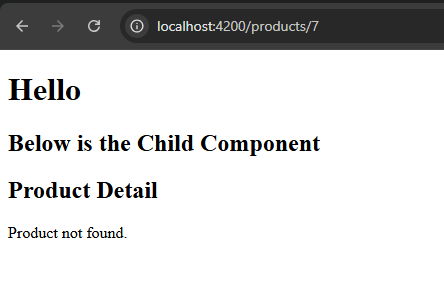
/products --> List of products

/products/2 --> View Smartphone detail (dynamically routed)

/products/4 --> Product not found







7)Routing – Nested Dynamic routing

Step 1: create project

Step 2: Generate Components

ng generate component product

ng generate component product/overview

ng generate component product/specs

Step 3: Set Up Routing

Open src/app/app-routing.module.ts and set up the routes:

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

import { RouterModule, Routes } from '@angular/router';

import { ProductComponent } from './product/product.component';

import { ProductOverviewComponent } from './product/overview/overview.component';

import { ProductSpecsComponent } from './product/specs/specs.component';

import { ProductReviewsComponent } from './product/reviews/reviews.component';

const routes: Routes = [

{

path: 'product/:id',

component: ProductComponent,

children: [

{ path: 'overview', component: ProductOverviewComponent },

{ path: 'specs', component: ProductSpecsComponent },

{ path: 'reviews', component: ProductReviewsComponent },

{ path: '', redirectTo: 'overview', pathMatch: 'full' }

]

},

{ path: '', redirectTo: '/product/1', pathMatch: 'full' }, // default route

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

------------------------------------------------------------------

Step 4: ProductComponent Template

src/app/product/product.component.html

<h2>Product Details (ID: {{ productId }})</h2>

<nav>

<a [routerLink]="['overview']">Overview</a> |

<a [routerLink]="['specs']">Specs</a> |

<a [routerLink]="['reviews']">Reviews</a>

</nav>

<router-outlet></router-outlet>

Step 5: ProductComponent

Go to --> src/app/product/product.component.ts

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

import { ActivatedRoute } from '@angular/router';

@Component({

selector: 'app-product',

templateUrl: './product.component.html'

})

export class ProductComponent implements OnInit {

productId!: string;

constructor(private route: ActivatedRoute) {}

ngOnInit() {

this.productId = this.route.snapshot.paramMap.get('id')!;

}

}

Step 6: Child Component Templates

6.1) overview.component.html

<p>This is the product overview.</p>

6.2) specs.component.html

<p>Here are the product specifications.</p>

6.3)reviews.component.html

<p>These are customer reviews.</p>

Step 7 : app.component.html

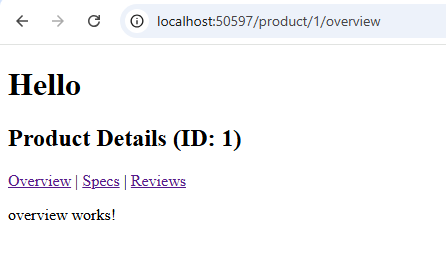
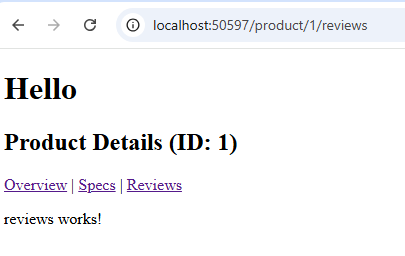
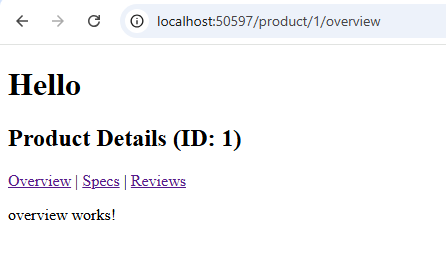
<h1>My E-Commerce Store</h1>

<router-outlet></router-outlet>

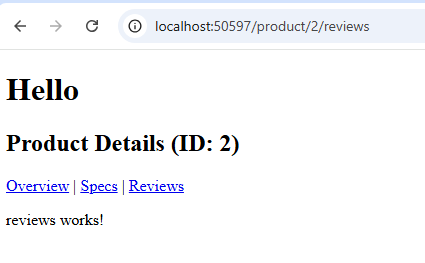
Step 8: Run the App

ng serve

Open: <http://localhost:4200/product/1>



If we pass id 2 in URL



8) http service

<div \*ngIf="(data$ | async) as data">

<h2> Below are the user details</h2>

<div \*ngFor="let user of data.users">

<div>

<img [src]="user.image" alt="User Image">

<p> Name: {{user.firstName}} {{user.lastName}}</p>

<p> Age: {{user.age}}</p>

<p> Email : {{user.email}}</p>

<p> Phone: {{user.phone}} </p>

</div>

</div>

</div>

=========

import { HttpClient } from '@angular/common/http';

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

import { Observable } from 'rxjs';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrl: './app.component.css'

})

export class AppComponent {

title = 'second-app';

data$: Observable<any>;

constructor(private http: HttpClient){

this.data$ = this.http.get('https://dummyjson.com/users')

}

}



9)http service CRUD

import { HttpClient } from '@angular/common/http';

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

import { Employee } from './employee/employee.model';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

private apiUrl = 'https://jsonplaceholder.typicode.com/users'

constructor(private http: HttpClient) { }

getEmployees() : Observable<Employee[]>{

return this.http.get<Employee[]>(this.apiUrl)

}

addEmployee(employee: Employee) :Observable<Employee>{

return this.http.post<Employee>(this.apiUrl, employee)

}

}

==============

export interface Employee{

id?:number;

name: string;

email:string;

}

===========

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

import { Employee } from './employee.model';

import { EmployeeService } from '../employee.service';

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrl: './employee.component.css'

})

export class EmployeeComponent implements OnInit{

employees: Employee[] = [

{id:1, name:'Memory Card', email:'abc@gmailcl.com'},

{id:2, name:'Headphone', email:'abc@gmailcl.com'},

{id:3, name:'LED TV', email:'abc@gmailcl.com'}

]

newEmployee : Employee = { name :' ' , email: ' '}

constructor(private employeeService: EmployeeService){}

ngOnInit(): void {

this.fetchEmployees();

}

fetchEmployees():void{

this.employeeService.getEmployees().subscribe(data => {

this.employees = data;

})

}

createEmployee(): void{

if(!this.newEmployee.name ||!this.newEmployee.email){

console.log("name and position are required")

return

}

this.employeeService.addEmployee(this.newEmployee).subscribe(response =>{

const add = {...this.newEmployee, id:101}

this.employees.push(add)

this.newEmployee = {name: '', email: ''}

})

}

}

===========

<h2> Employee List</h2>

<ul>

<li \*ngFor="let emp of employees">

{{emp.name}} has position as {{emp.email}}

</li>

</ul>

<h2> Add New Employee</h2>

<form (ngSubmit)="createEmployee()">

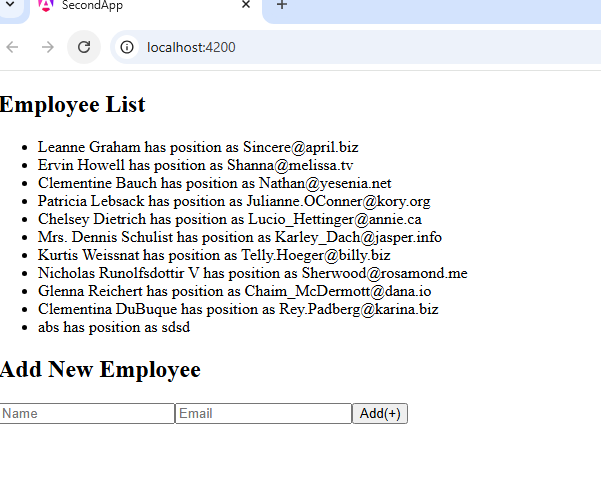
<input [(ngModel)]="newEmployee.name" name="name" placeholder="Name">

<input [(ngModel)]="newEmployee.email" name="email" placeholder="Email">

<button type="submit"> Add(+)</button>

</form>

UI



9) Edit and delete records from above list

import { HttpClient } from '@angular/common/http';

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

import { Employee } from './employee/employee.model';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

private apiUrl = 'https://jsonplaceholder.typicode.com/users'

constructor(private http: HttpClient) { }

getEmployees() : Observable<Employee[]>{

return this.http.get<Employee[]>(this.apiUrl)

}

addEmployee(employee: Employee) :Observable<Employee>{

return this.http.post<Employee>(this.apiUrl, employee)

}

updateEmployee(id:number, employee: Employee) : Observable<Employee>{

console.log(`------- ${this.apiUrl}/${id}`)

console.log(employee)

return this.http.put<Employee>(`${this.apiUrl}/${id}`, employee)

}

deleteEmployee(id:number) : Observable<void>{

return this.http.delete<void>(`${this.apiUrl}/${id}`)

}

}

=====

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

import { Employee } from './employee.model';

import { EmployeeService } from '../employee.service';

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrl: './employee.component.css'

})

export class EmployeeComponent implements OnInit{

editing :boolean = false;

editEmployee: Employee = {id:0, name:'', email:''}

startEdit(emp: Employee): void {

this.editing = true;

this.editEmployee = {...emp}

}

employees: Employee[] = [

{id:1, name:'Memory Card', email:'abc@gmailcl.com'},

{id:2, name:'Headphone', email:'abc@gmailcl.com'},

{id:3, name:'LED TV', email:'abc@gmailcl.com'}

]

newEmployee : Employee = { name :' ' , email: ' '}

constructor(private employeeService: EmployeeService){}

ngOnInit(): void {

this.fetchEmployees();

}

fetchEmployees():void{

this.employeeService.getEmployees().subscribe(data => {

this.employees = data;

})

}

createEmployee(): void{

if(!this.newEmployee.name ||!this.newEmployee.email){

console.log("name and position are required")

return

}

this.employeeService.addEmployee(this.newEmployee).subscribe(employee =>{

//const add = {...this.newEmployee}

this.employees.push(employee)

this.newEmployee = {name: '', email: ''}

})

}

updateEmployee(): void{

if(!this.editEmployee.id) return

this.employeeService.updateEmployee(this.editEmployee.id, this.editEmployee).subscribe(updated => {

const index = this.employees.findIndex(e => e.id === updated.id)

if( index !== -1)

this.employees[index] = updated;

this.editing= false;

this.editEmployee = {id: 0, name: '', email: ''}

})

}

deleteEmployee(id:number) :void {

this.employeeService.deleteEmployee(id).subscribe( () =>{

this.employees = this.employees.filter(emp => emp.id !== id)

})

}

}

======

<h2> Employee List</h2>

<ul>

<li \*ngFor="let emp of employees">

{{emp.name}} has position as {{emp.email}}

<button (click)="startEdit(emp)"> Edit </button>

<button (click)="deleteEmployee(emp.id!)"> Delete </button>

</li>

</ul>

<h2> Add New Employee</h2>

<form (ngSubmit)="createEmployee()">

<input [(ngModel)]="newEmployee.name" name="name" placeholder="Name">

<input [(ngModel)]="newEmployee.email" name="email" placeholder="Email">

<button type="submit"> Add(+)</button>

</form>

<!-- edit the employeee record-->

<div \*ngIf="editing">

<h2> Edit Employee</h2>

<input [(ngModel)]="editEmployee.name" placeholder="Name">

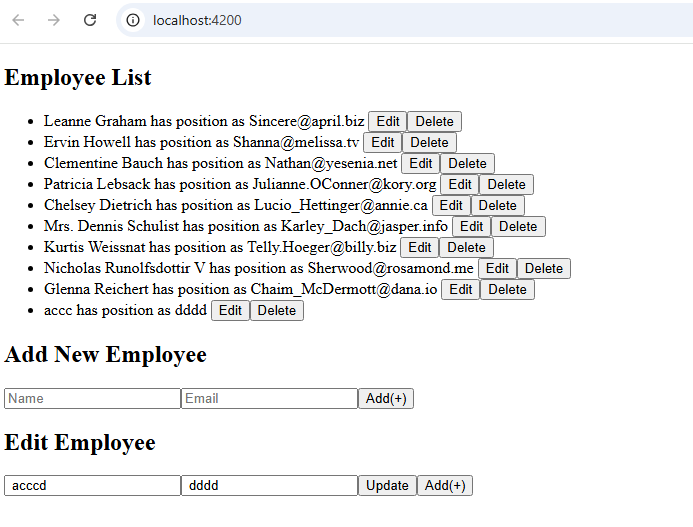
<input [(ngModel)]="editEmployee.email" placeholder="Email">

<button (click)="updateEmployee()"> Update </button>

<button (click)="editing = false"> Add(+)</button>

</div>

//OUTPUT



10) PIPE – to transform input

import { Pipe, PipeTransform } from '@angular/core';

import { Employee } from './employee/employee.model';

@Pipe({

name: 'employeefilter'

})

export class EmployeefilterPipe implements PipeTransform {

transform(employees : Employee[], searchText: string): Employee[] {

if(!employees || !searchText) return employees;

console.log(employees)

return employees.filter(emp =>

emp.name.toLowerCase().includes(searchText.toLowerCase())

)

}

}

=======

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'capitalize'

})

export class CapitalizePipe implements PipeTransform {

transform(value: string): unknown {

if(!value) return ''

return value.charAt(0).toUpperCase() + value.slice(1).toLowerCase()

}

}

=== employee.component.ts ====> add searchText :string = ''

======

<h2> Employee List</h2>

<input [(ngModel)] = "searchText" placeholder="Search BY name">

<ul>

<li \*ngFor="let emp of employees | employeefilter:searchText">

{{emp.name | capitalize}} has position as {{emp.email | capitalize}}

<button (click)="startEdit(emp)"> Edit </button>

<button (click)="deleteEmployee(emp.id!)"> Delete </button>

</li>

</ul>

<h2> Add New Employee</h2>

<form (ngSubmit)="createEmployee()">

<input [(ngModel)]="newEmployee.name" name="name" placeholder="Name">

<input [(ngModel)]="newEmployee.email" name="email" placeholder="Email">

<button type="submit"> Add(+)</button>

</form>

<!-- edit the employeee record-->

<div \*ngIf="editing">

<h2> Edit Employee</h2>

<input [(ngModel)]="editEmployee.name" placeholder="Name">

<input [(ngModel)]="editEmployee.email" placeholder="Email">

<button (click)="updateEmployee()"> Update </button>

<button (click)="editing = false"> Add(+)</button>

</div>