CaseStudyNotes-Angular-ShoppingCart:

1. Set up Dev environments: VSC NPM package manager.
2. Install Angular CLI: npm install -g @angular/cli@latest
3. Create a new Angular Application. Ng new shopping-cart; ng serve -o to check if the app is running correctly.
4. Module design:

/app

/admin

/components

/admin-orders

/admin-products

/product-form

/services

/admin-auth-guard.service.ts

Admin.module.ts

/core

/components

/bs-navbar

/Home

/Login

Core.module.ts

/shared

/components

/product-card

/product-quantity

/models

/services

Shared.module.ts

/shopping

/components

/check-out

/my-orders

/order-success

/products

/shipping-form

/shopping-cart

/shopping-cart-summary

Shopping.module.ts

App.components.css

App.components.html

App.components.spec.ts

App.components.ts

/assets

Some set up data like: category.json, shopping-cart.json

Index.html

1. Communication between the modules:

=>Shared the router service:

In the Root AppModule:

RouterModule.forRoot([routes: path and components combo]);

In the sub feature module:

RouterModule.forChild() to get the same instance injection.

private populateProducts() {

    this.productService

      .getAll()

      .switchMap(products => {

        this.products = products;

        return this.route.queryParamMap;

      })

      .subscribe(params => {

        this.category = params.get('category');

        this.applyFilter();

      });

  }

this.router.navigate(['/order-success', result.key]);

=>Imports: [ … ] and export class FeatureModule {}

=> Passing data with @input @output

@input @output @eventEmitter decorators: parent html:

<h2>Shipping</h2>

<div class="row" \*ngIf="cart$ | async as cart">

  <div class="col-6">

    <shipping-form [cart]="cart"></shipping-form>

  </div>

  <div class="col-6">

    <shopping-cart-summary [cart]="cart"></shopping-cart-summary>

  </div>

</div>

Shopping-cart-summary.components.ts:

export class ShoppingCartSummaryComponent {

@Input('cart') cart: ShoppingCart;

}

Example of @output and eventEmitter:

https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/

1. fdsdfsa
2. fdsfds

### **angular parammap vs queryparammap**

<https://www.digitalocean.com/community/tutorials/angular-query-parameters>

Query parameters in Angular allow for passing optional parameters across any route in the application. Query parameters are different from regular route parameters, which are only available on one route and are not optional (e.g., /product/:id). In this article, we will reference an example of an application that displays a list of products.

use paramMap observable of ActivatedRoute

Query parameters mean in your URL whatever comes from after question mark(?). We can pass the query params using the routerLink directive as well as programmatically.

you used different examples to set and get query parameters in Angular. You were introduced to queryParams and queryParamsHandling with Router.navigate and RouterLink. You were also introduced to queryParams and queryParamMap with ActivatedRoute.

QueryParams: Using Query Parameters with Router.navigate

Set query param:

goProducts() {

this.router.navigate(

['/products'],

{ queryParams: { order: 'popular', 'price-range': 'not-cheap' } }

);

}

Preserve the query parameters:

goUsers() {

this.router.navigate(

['/users'],

{ queryParamsHandling: 'preserve' }

);

}

Merge route parameters:

goUsers() {

this.router.navigate(

['/users'],

{

queryParams: { filter: 'new' },

queryParamsHandling: 'merge' }

);

}

We can also set query parameters with RouterLink:

<a

[routerLink]="['/products']"

[queryParams]="{ order: 'popular'}"

>

Products

</a>

<a

[routerLink]="['/users']"

[queryParams]="{ filter: 'new' }"

queryParamsHandling="merge"

>

Users

</a>

How to access Query Parameter Values:

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

import 'rxjs/add/operator/filter';

@Component({ ... })

export class ProductComponent implements OnInit {

order: string;

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.route.queryParams

.filter(params => params.order)

.subscribe(params => {

console.log(params); // { order: "popular" }

this.order = params.order;

console.log(this.order); // popular

}

);

}

}

Access the merged parameter list:

this.route.queryParamMap

.subscribe((params) => {

this.orderObj = { ...params.keys, ...params };

}

);

{

"0": "order",

"1": "filter",

"params": {

"order": "popular",

"filter": "new"

}

}

Learning With Mosh:

<https://codewithmosh.com/courses/206545/lectures/3196484>

DataSource Json API:

<https://jsonplaceholder.typicode.com/>

<https://reqres.in/api/users>

Router Module:

Configure the routes: map routes to components

Add a router outlet

Add a link: routerLink : routerLinkActive=”active current”

In app.module.ts file:

Import RouterModule

At imports array: more specific routes on top!!!!!!

imports: [

RouterModule.forRoot([

{ path: ‘ ‘, component: homeComponent },

{ path: ‘followers’, component: GitHubFollowersComponent},

{ path: ‘profile/:username’, component: GithubProfileComponent}

])

]

Life Cycle hooks:

OnInit

OnChanges

DoCheck

AfterContentInit;

Error Handling:

Two imports: rxjs/add/operator/catch rxjs/add/observable/throw

Or use ErrorHandler:

In the app.module.ts file:

Providers: [

PostService,

{ provide: ErrorHandler, useClass: AppErrorHandler }

]

In Component class, use catch and error but in service class, use application specific error classes.

(error: Response) => {

If ( error.status === 400)

This.form.setErrors(error.json());

Else

Something else.

});

In Service class:

( error: AppError) => {

If ( error instanceof BadInput) {

} else {

}

#############################################################

**Unit Test and Integration Test:**

**Testing Service:**

[**https://danielk.tech/home/how-to-test-angular-http-services**](https://danielk.tech/home/how-to-test-angular-http-services)

**Testing Component with service:**

[**https://shashankvivek-7.medium.com/testing-a-component-with-stub-services-and-spies-in-jasmine-1428d4242a49**](https://shashankvivek-7.medium.com/testing-a-component-with-stub-services-and-spies-in-jasmine-1428d4242a49)

When using Stub Services and Spies in Jasmine, the data created in stub need to match the data

That get from api source in service call exactly, or will get weird errors.

<https://kentcdodds.com/blog/static-vs-unit-vs-integration-vs-e2e-tests>

**#############################################################**

<https://medium.com/codex/how-to-share-data-between-components-in-angular-a-shopping-cart-example-b86ce8254965>

<https://www.baeldung.com/spring-boot-angular-web>

1. our sample data will come from/assets/data.json
2. create a shopping cart service that will hold the basic state of the shopping cart shoppingCart$
3. Use BehaviorSubject is a type of observable we can use to get “Snapshots” of its latest value via its .getValue() method or .value property without the need of a subscription.
4. We can use the observable to create new projections with RxJS operators.

The Observer that we pass in, is an object, and it contains three properties: next, error, and complete. Each one represents a function that will be triggered automatically from the observable.

<img class="img-fluid" src="https://via.placeholder.com/175" alt="sample image">

|  |
| --- |
| private getShoppingCart() { |
|  | this.httpClient.get<ShoppingCart>('/assets/data.json').subscribe((  shoppingCart) => { |
|  | this.setShoppingCart(shoppingCart); |
|  | }, |
|  | () => { |
|  | console.error('Shopping cart data could not be loaded.'); |
|  | }); |
|  | } |
|  |  |
|  | private setShoppingCart(shoppingCart: ShoppingCart) { |
|  | this.shoppingCart$.next(shoppingCart); |
|  | } |

Module.ts file for separate module peeled off from app.module.ts:

The module.ts file has @NgModule annotation and three important parts:

Imports: [ ] to import all the needed modules;

Declarations: [ ] to declared all the internal used components;

Exports: [ ] to export all the componets are needed by the other modules.

@NgModule({

imports: [

CommonModule,

FormsModule,

CustomFormsModule,

DataTableModule,

AngularFireDatabaseModule,

AngularFireAuthModule,

NgbModule.forRoot(),

],

declarations: [

ProductCardComponent,

ProductQuantityComponent,

],

exports: [

ProductCardComponent,

ProductQuantityComponent,

CommonModule,

FormsModule,

CustomFormsModule,

DataTableModule,

AngularFireDatabaseModule,

AngularFireAuthModule,

NgbModule.forRoot().ngModule,

],

providers: [

AuthService,

AuthGuard,

UserService,

CategoryService,

ProductService,

ShoppingCartService,

OrderService

]

})

export class SharedModule { }

<https://blog.angular-university.io/introduction-to-angular-2-forms-template-driven-vs-model-driven/>

Everything about the forms:

**Enabling Template Driven Forms is close to AngularJS**

**ngModel; ngForm. ngSubmit**

## Angular Reactive Forms: more preferred form for Angular?

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

HTML reactive form:

<section class="sample-app-content">

<h1>Reactive Form Example:</h1>

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

<p>

<label>First Name:</label>

<input type="text" formControlName="firstName">

</p>

<p>

<label>Password:</label>

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

</p>

<p>

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

</p>

</form>

</section>

1. there is a formGroup directive applied to the whole form, binding it to a component variable named form.
2. the required validator attribute is not applied to the form controls. This means the validation logic must be somewhere in the component class, where it can be more easilly unit tested.

**import { FormGroup, FormControl, Validators, FormBuilder }**

**from '@angular/forms';**

**@Component({**

**selector: "reactive-form",**

**templateUrl: 'reactive-form.html'**

**})**

**export class ReactiveFormExample {**

**form = new FormGroup({**

**"firstName": new FormControl("", Validators.required),**

**"password": new FormControl("", Validators.required),**

**});**

**onSubmitModelBased() {**

**console.log("reactive form submitted");**

**console.log(this.form);**

**}**

**}**

**FormGroup will keep track of the global form value and the validity state.**

**The controls themselves can be instantiated individually using the FormControl**

**Constructor.**

The end result is a programmatic definition of our form model with all of its controls and validity rules, that is created programmatically at the level of the component class, and not the template.

### The FormBuilder API: the built-in FormBuilder service

import { FormGroup, FormControl, Validators, FormBuilder }

from '@angular/forms';

@Component({

selector: "reactive-form",

templateUrl: 'reactive-form.html'

})

export class ReactiveFormExample {

form = fb.group({

"firstName": ["", Validators.required],

"password":["", Validators.required]

});

constructor(fb: FormBuilder) {

}

onSubmitModelBased() {

console.log("reactive form submitted");

console.log(this.form);

}

}

Reactive Forms are a much better choice!

Main Points: share the data between components using the observables.e Mainly BehaviorSubjects. Also add backend webservice support.

1. Create a new angular project.
2. Create all the components.
3. Create routing module.
4. Create services.