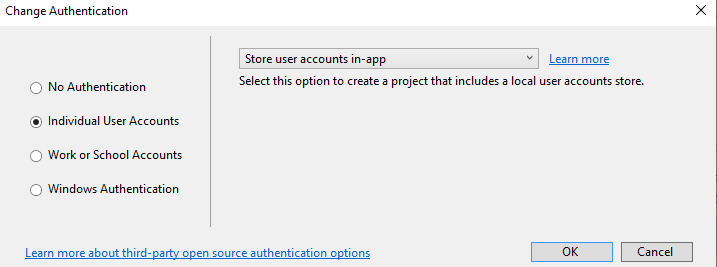
1. Read assignment requirements and play with the sample code.

2. Create a web application with “Individual User Accounts” authentication.



3. Added the items.cs class and the Sales.cs class and scaffolded it with entity

framework, using the existing security context, then added the migration and

updated the database, then add the controller and action names to \_Layout.cshtml.

4. Edit action ‘index’ in items controller in a way that the index function take a string as its parameter, and it compares the string with each item name from ‘\_context.items’ within the function, then it returns the matched results.

5. Scaffold identity into an MVC project with authorization. To add ‘identity’ scaffold to the project, select the existing layout page and data context, then select 1 file to override.

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-3.1&tabs=visual-studio#scaffold-identity-into-an-mvc-project-with-authorization>

**Angular**:

Install bootstrap

Add a component folder with standard component files: “ng g c --skipImport=true add-item”

Pass data from parent component(item-card) to child component(item-list)

Use ‘ngFor’ to iterate a list in html file

Use Property Binding:

@Input() item : MyItem

<app-item-card [item]="item"></app-item-card>

Configure “item-list.component.ts” to make http call to get data from a json file

Inject dependency: “constructor(private http:HttpClient) { }”

Observable functions are not executed until a consumer subscribe to them

Angular services are singleton object that get instantiated only once during the lifecycle of an application, the main objective of the service is to organize and shared business logic and data and function with different conponents.

Start creating a service to fetch data from the json file

A service can be used for the whole applicaiton

Create ‘add-item’ component, register it and assign a path to it in ‘app.module’

The most dangerous side effect of reloading the whole page: data passed in navigation will get reset

[routerLinkActiveOptions]=”exact:true”(this set the nav active when it is exact match with the url )

'item-detail/:id' '/:id' allows passing a value into the url

<a routerLink="/item-detail/**{{**item.Id**}}**"> passing item id into the url using string interporlation

using activated route to get the value of 'id' from the active route and assign it to the ‘itemId’

using event binding to bind 'click' event with 'onback' function

ts file

onBack() {

this.router.navigate(['/']);

}

<button (click)="onBack()" class="btn btn-primary">Back</button>

// inject activatedRoute and Router

// using activated route to ensure right the right nav link is highlighted

// using router to navigate the click event binded with the next page button

constructor(private route: ActivatedRoute, private router: Router) { }

ngOnInit() {

//get the value of 'id' from the active route and assign it to the itemId

this.itemId = this.route.snapshot.params['id'];

}

onNextPage() {

this.itemId += 1;

//navigate to the url with itemId increased by 1

this.router.navigate(['item-detail', this.itemId]);

}

<button (click)="onNextPage" class="btn btn-primary "> Next Page</button>

//get updated route value within the same component

this.route.params.subscribe(

(params) => {

this.itemId = +params['id'];

}

)

**Add image to each item card**:

{

"Id": 6,

"SellRent": 1,

"Type": "fire",

"Price": 10,

"Name": "Fire",

"Image": "iphoneXR"

}

// '?' define optional field

Image?: string;

**Determine which items belong to which page: (useful for showing items based on account currently logged in)**

In Item-list.ts:

export class ItemListComponent implements OnInit {

//initialize variable SellRent with value 1, it will be used to determine where an item should be displayed

SellRent = 1;

ngOnInit(): void {

// if the page url is "selling-items", SellRent = 2, which is the value of the SellRent property of selling items.

// That means selling items will be displayed in this page

if (this.route.snapshot.url.toString() === "selling-items") {

this.SellRent = 2;

}

// pass the value of SellRent when calling the service

this.eshoppingService.getAllItems(this.SellRent).subscribe(

data => {

this.sellingItems = data;

console.log(data);

In shopping.service.ts:

//receive parameter 'SellRent' to determine which items to push

getAllItems(SellRent: number): Observable<MyItem[]> {

//pipe method transform one object to another type of object

return this.http.get('data/items.json').pipe(

map(data => {

const sellingItems: Array<MyItem> = [];

for (const id in data) {

//the items with the same value of SellRent as the function received will be push

if (data.hasOwnProperty(id) && data[id].SellRent === SellRent) {

sellingItems.push(data[id]);

}

}

return sellingItems;

})

);

}

**Create a form to add new items:(template driven form)**

In app.module

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

imports: [

BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),

HttpClientModule,

FormsModule,

ApiAuthorizationModule,

//let the application know the routes exist

RouterModule.forRoot(appRoutes),

FormsModule

],

<!-- create variable #Form and assign a javascript representation "ngForm" to it-->

<form (ngSubmit)='onSubmit()' #Form="ngForm">

<div class="form-group col-12">

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

<!-- define nameCheck variable for the error block to determine if the name input is valid -->

<input type="text" class="form-control" #nameCheck="ngModel" ngModel name="itemName" required minlength="5">

<span \*ngIf="!nameCheck.valid && nameCheck.touched" class="error-block" >

<!-- if no name is entered-->

<span \*ngIf="checkName.errors.required">

Please provide name

</span>

<!-- if name length is smaller than 5 -->

<span \*ngIf="checkName.errors.minlength">

Name must be at least 5 characters long!

</span>

</span>

<!-- When form is not valid, disable the save button-->

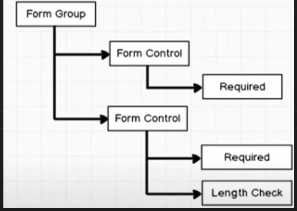
<button [disabled]="!Form.valid" type="submit" class="btn btn-primary mr-2">save</button>

// get a reference of the value of the form before passing it to the submit method

@ViewChild('Form', { static: false }) addItemForm: NgForm;

**Create identity (register using reactive forms and login):**

Form group:



Create register form and add validation:

Html:

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

<div class="form-group col-12">

<label for="name" class="form-label">Name</label>

<input type="text" class="form-control" formControlName="userName">

<!-- userName if from the get userName() method in the .ts file-->

<span \*ngIf="!userName.valid && userName.touched" class="error-block">

Please provide a name!

</span>

</div>

Ts file:

registerationForm: FormGroup;

ngOnInit() {

this.registerationForm = new FormGroup(

{

// apply field validators

// comtrol level validators

userName: new FormControl(null, Validators.required),

get userName() {

return this.registerationForm.get('userName') as FormControl;

}

onSubmit() {

console.log(this.registerationForm);

}

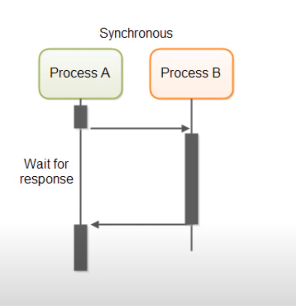
Reactive form:

Better performance

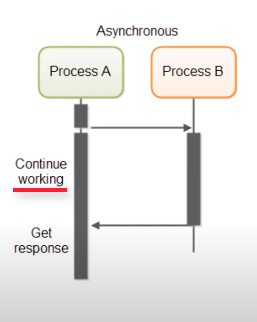
Clean template

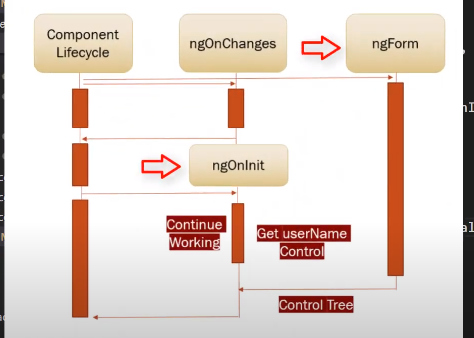
More predictable (Synchronous)

In a Synchronous program, when process B is called during process A, process A will wait until process B is finished



In a Asynchronous program, process A will not wait for process B to complete, it will move on to another task





Building reactive form using FormBuilder

Store account info into local storage

Building user services

Create userAccount interface

use ‘alertifyjs’ for customized alert boxes