To create a new angular application command is:

ng new projectName

routing:yes, stylesheet : css

Folder structure :

Configuration files

Node\_modules : all modules (dev, testing, deploying)

Src: source code :

Main.ts : main file from which the application will start

Styles.css: global style

Index.html (only HTML document – it is loaded first when we start the application)

Assets folder: static content (images, data…)

App : app folder defines 2 files modules and 4 files which refer to the component

Modularization is implemented in Angular : modules : independent section, which can be reused in other projects.

Here we have two modules: app.module.ts class AppModule, app-routing.module.ts (routing)

============= running the application ===============

Cd projectname

Ng serve -o

Localhost:4200

============ component =================

Every component has :

Css : local style sheet

Html : “**template”** (UI – html code)

.ts : class Component – “**component”**

.spec.ts : unit testing code for the component

Every component defines the metadata using @Component

It defines Selector, templateUrl, styleUrls

--- > add bootstrap

CDN links in index.html

Added navbar in app.component.html

============ component creation ==================

1. Ng generate component home

Ng g c home

Angular will create the Home folder and within that there will be 4 different files

Ng g c product-list (class ProductListComponent)

Ng g c login

Ng g c register

============== routing ==============

For navigation there are 3 steps:

1. Set up the link within the navbar (url/ route)

e.g

<li class="nav-item">

                <a class="nav-link" routerLink="/products">Products</a>

 </li>

1. Add the routes in app-routing module

const routes: Routes = [

  { path:'', component:HomeComponent },

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

  { path:'login', component:LoginComponent},

  { path:'register', component:RegisterComponent}

];

1. Define the placeholder (place, in which the child component will be rendered)

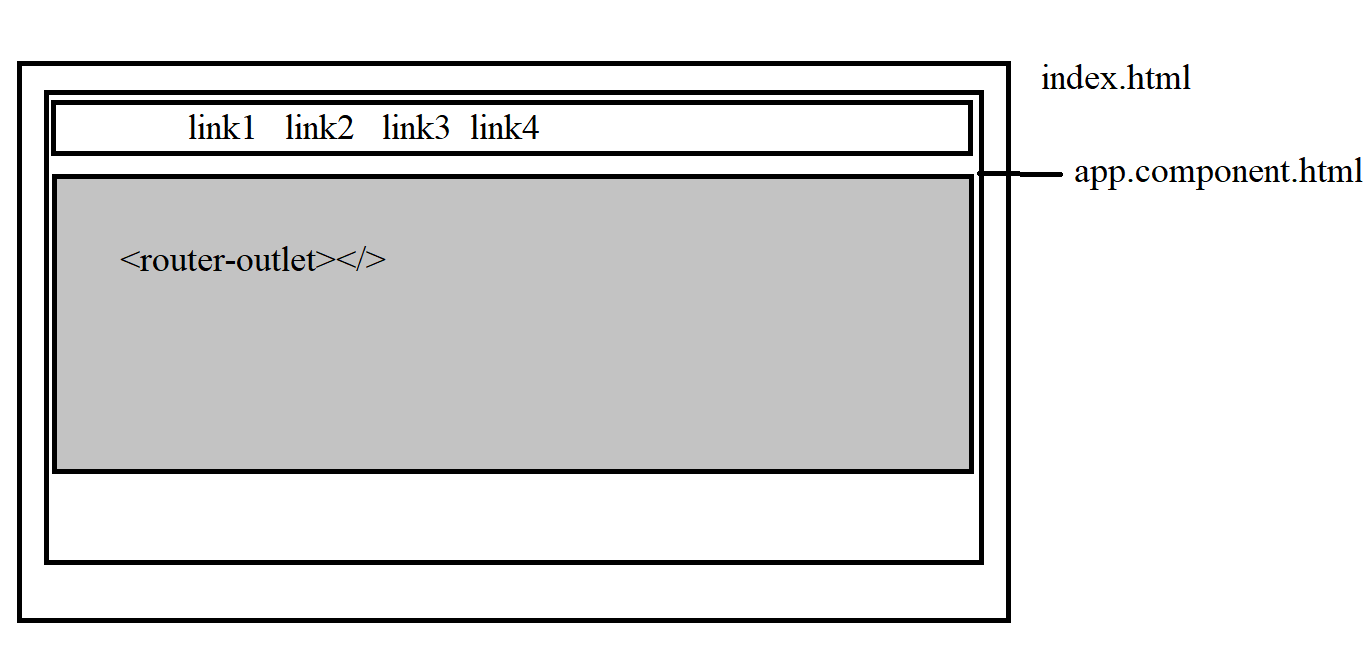
App.component.html

<div>

<nav></nav>

<router-outlet></router-outlet>

</div>



================= Form Handling ===============

Data binding: sharing data/ codes within the component. (template and component communication)

.ts that defines the component class, can declare variables, methods, constructors (additional methods - - lifecycle hooks).

As we need to display the data on template.

|  |  |
| --- | --- |
| Component to template | String interpolation {{ }} |
| Component to template | Property binding [] |
| Template to Component | Event Binding () |
| Two way data binding | [( ngModel )] |

Css classes for input text boxes:

|  |  |  |
| --- | --- | --- |
|  | False | True |
| Input control touched? | Ng-untouched | Ng-touched |
| Data entered? (input is clean or dirty) | Ng-pristine | Ng-dirty |
| Valid? | Ng-invalid | Ng-valid |

Form types:

<https://v16.angular.io/guide/forms-overview>

1. Template driven form
2. Reactive Form

Template Driven Form validations:

1. Set up the validations on form control – template

<input type="email" class="form-control"

                [(ngModel)]="username"

                required email

                >

1. We need the template variable – template

 <input type="email" class="form-control"

                [(ngModel)]="username"

                required email

                #usernameInput="ngModel"

                >

1. CSS manipulation – CSS

.ng-touched.ng-invalid {

    border: 2px solid red;

}

1. Displaying the suitable error messages – template

<div class="alert alert-danger"

                \*ngIf="usernameInput.touched && usernameInput.invalid"

                 rolw="alert">

                    <span \*ngIf="usernameInput.errors?.['required']">Username is compulsory</span>

                    <span \*ngIf="usernameInput.errors?.['email']">Please enter the valid email</span>

                </div>

Reactive Forms Validations:

1. Set up the validations on form control – within the ts (form control)

  registerForm = new FormGroup({

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

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

      password: new FormControl(),

      confirmPassword: new FormControl()

      // ,address: new FormGroup({

      //   city: new FormControl(),

      //   state: new FormControl()

      // })

  });

1. CSS manipulation – CSS

.ng-touched.ng-invalid:not(form) {

    border:2px solid red;

}

1. Access the formCotrol individually

  get fullNameCtrl() {

    return this.registerForm.get("fullName") as FormControl;

  }

  get emailCtrl(){

    return this.registerForm.get("email") as FormControl;

  }

1. Displaying the suitable error messages – template

<!-- {{emailCtrl.errors|json}} -->

                <div class="alert alert-danger" role="alert"

                    \*ngIf="emailCtrl.touched && emailCtrl.invalid">

                    <span \*ngIf="emailCtrl.errors?.['required']">

                        Please enter your Email </span>

                    <span \*ngIf="emailCtrl.errors?.['email']">

                        Please enter the valid email</span>

                </div>