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)

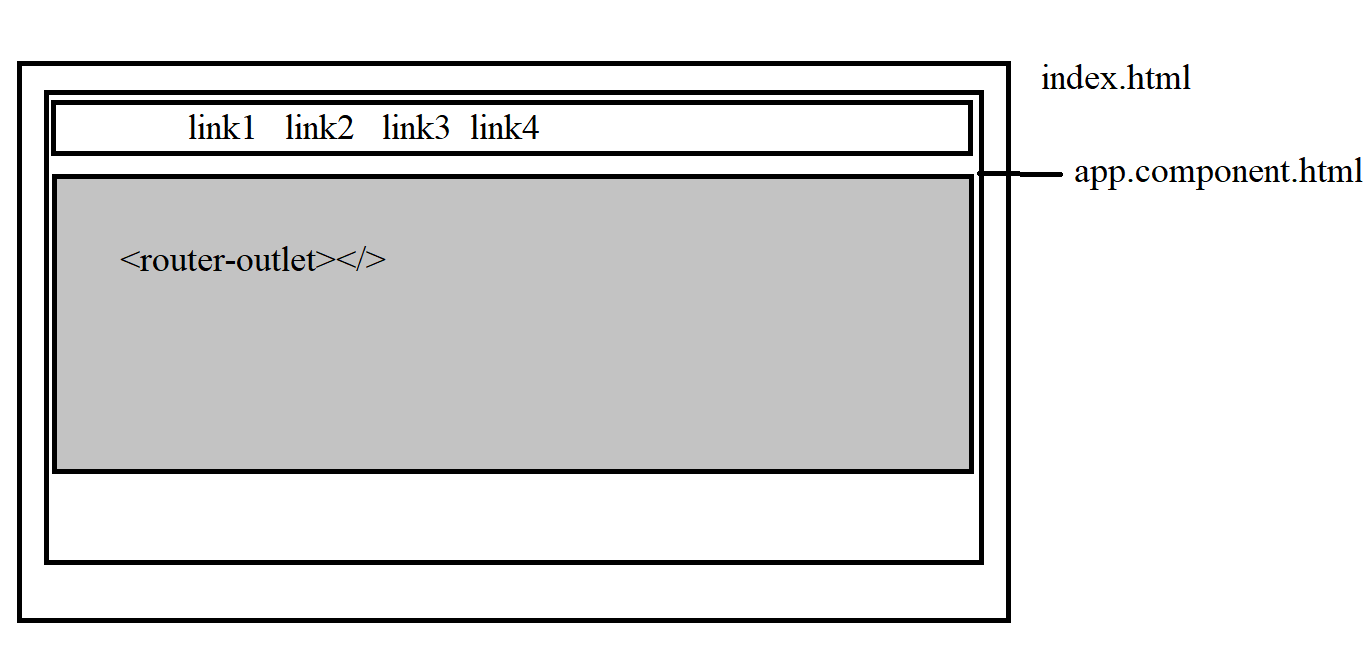
App.component.html

<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>

================ data binding, pipes and directives ==================

Communication between the component class and the template

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

Pipes: are used for transformation of data

Component Communication:

Data Binding is the data sharing among the template and class of a specific component.

Communication between different components

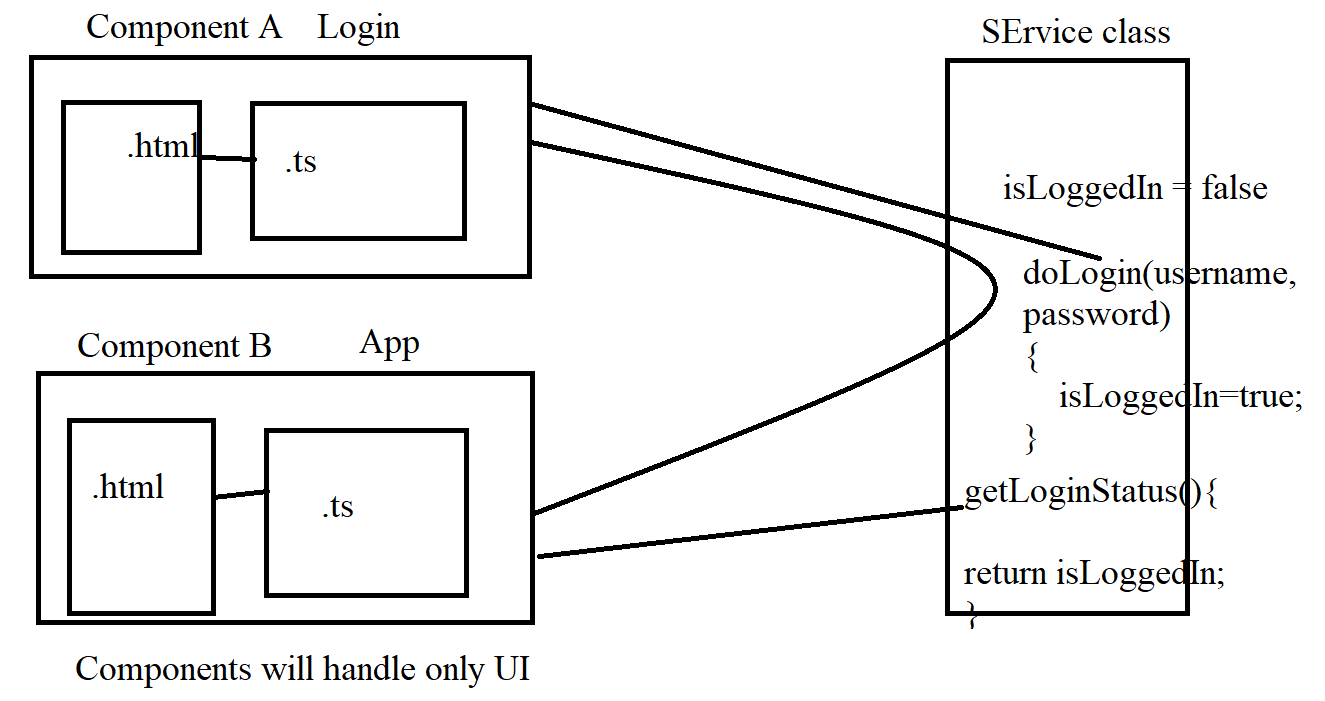
e.g. Login Activity:

Login Component 🡪 login details -🡪 verify (hardcoded admin , admin123)

AppComponent 🡪 navbar

Ng generate service user

Ng g s user 🡪 This will create the UserService class



**Dependency Injection –**

Laptop has the dependency: motherboard, RAM

Angular Component must be fully configured (with all required objects)

Class LoginComponent 🡪 object is created by Angular framework.

( IOC - inversion of control – creation of the object and maintain the lifecycle of the object is completely handled by the framework. )

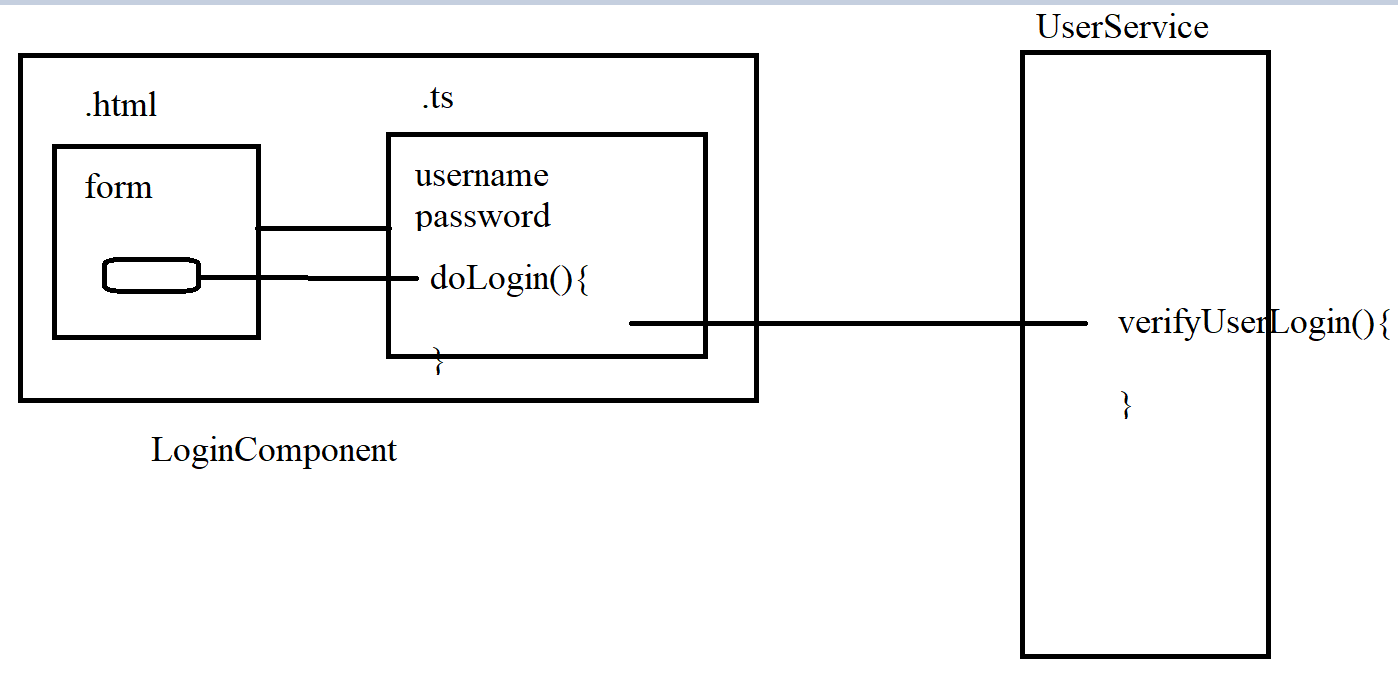
When the object is getting created by framework, list all the requirements - DEPENDENCY

LOginComponent 🡪 service class 🡪 doLogin() . Here we need the object of UserService class inside the LoginComponent

LoginComponent class has the dependency of UserService. And we need to use the DI (Dependency Injection). In Angular DI is implemented through constructor.

LoginComponent(LC) needs the object of UserService class. That means LC has a dependency of UserService object. We need to inject this dependency through the constructor.

Login activity flow:



isLoggedIn = true

The communication is established, but its not real time !!!

====================== Signal ======================

Signals : they are

Writable Signals: it is reactive primitive that tracks the state change within the application.

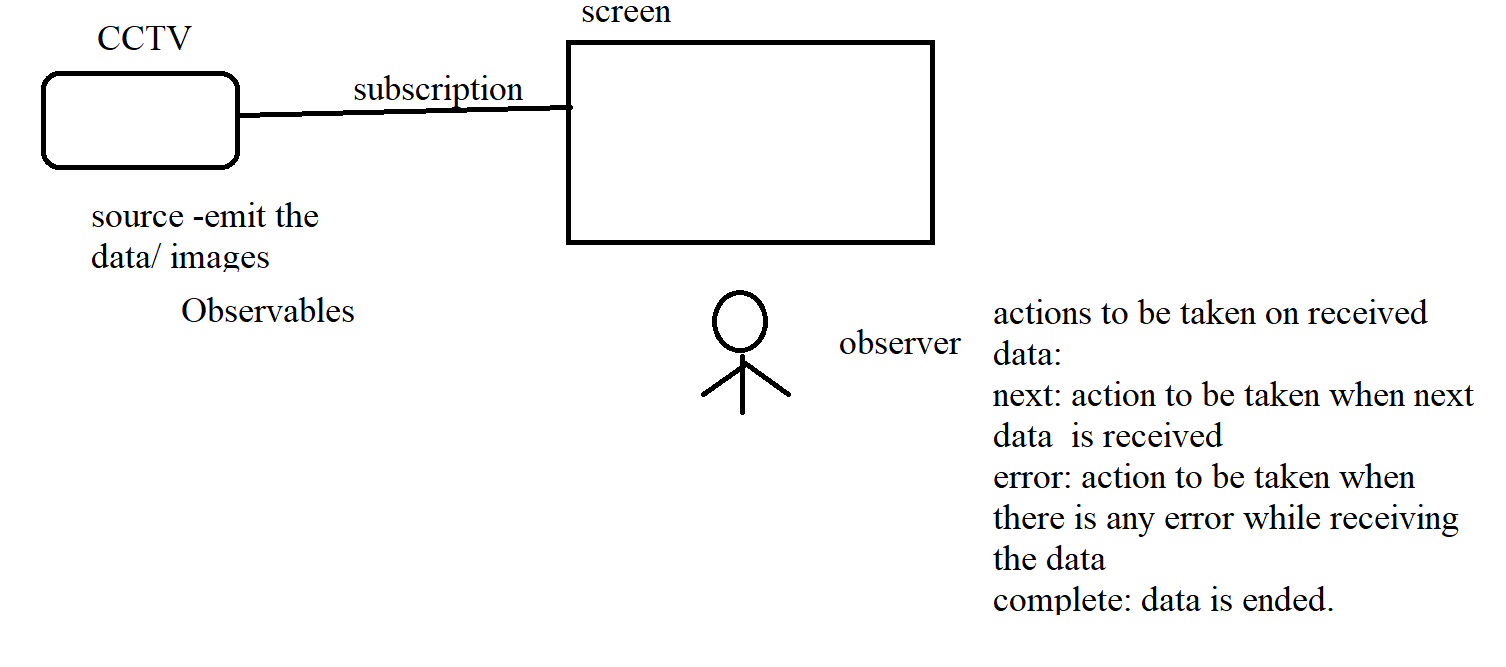
We are using signals at AppComponent (signal is status) and UserService (signal is isLoggedIn) . The signal ‘status’ is a copy of ‘isLoggedIn’. Whenever isLoggedIn is updated, immediately status will be updated. Thus we can control the navbar based on ‘status’ signal.

=========================== Observable and api communication ==============

Observables : RxJS – real time data sharing, to get access to the future events.

<https://rxjs.dev/guide/overview>

Observable, Subscription and Observer



We need to use dummy api (json-server) – we can perform basic CRUD operations

Npm install -g json-server

Define db.json in assets folder and set the json object

Start the json-server:

json-server --watch ./src/assets/db.json

**Connecting to the api**

HttpClientModule is required in Angular application (we need to add this in AppModule imports)

We need the service class – ProductService

Ng g s product