Human Computer Interaction

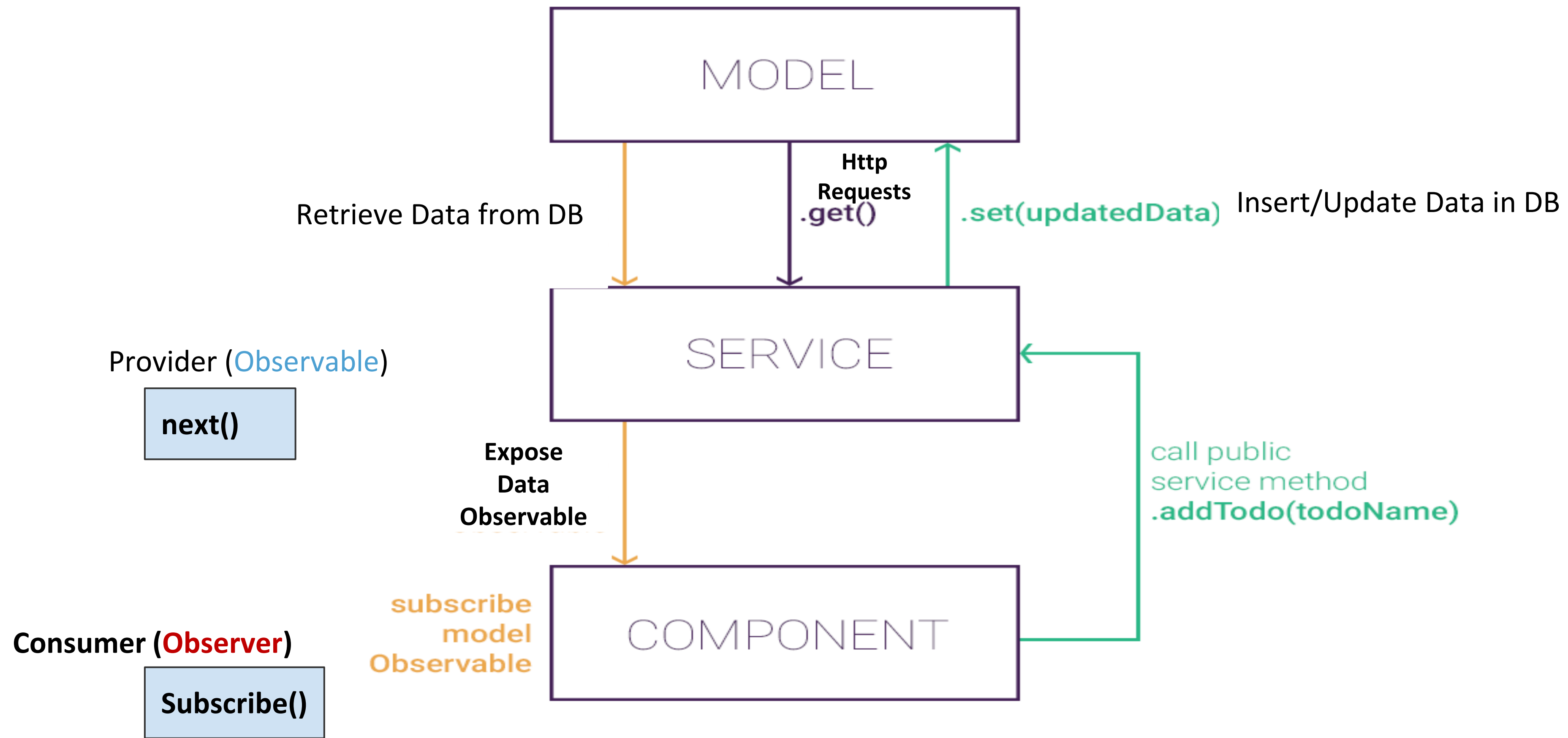
Lab 7 – Angular

Agenda

- Review Angular Services
- Forms & Validation
- Routing
- Task 4

Review- Angular Service

Subjects can do both Observable and observer

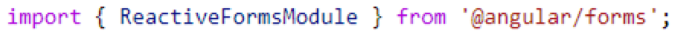


Reactive Forms

- Forms have multiple approaches such as template-driven form and reactive form.

- Forms that rely on conditional logic or custom validation requirements are easier to implement with a reactive form approach.

- Reactive Form is synchronous (more predictable event flow) and utilizes the ReactiveFormsModule.



```
import { ReactiveFormsModule } from '@angular/forms';
```

- Three primary form building tools:
    1) FormControl - Controls a single form element like input text.
    2) FormGroup - Controls a defined group of FormControls.

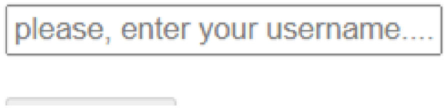
- The FormControl instantiated on the controller acts as the source of truth for each respective form element. Any data manipulation or validation is attached to the FormControl

How to use Reactive Forms?

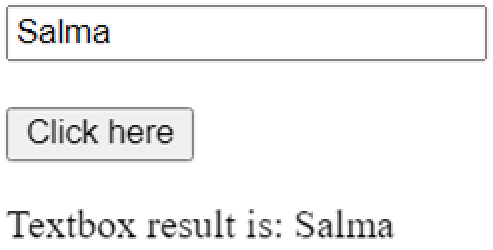
1. Import ReactiveFormsModule in appModule.
2. Create Form Model in component class using Form Group & Form Control or USE Form Builder
3. Create the HTML Form resembling the Form Model.
4. Bind the HTML Form to the Form Model.

```
public formdata:FormGroup;
 ngOnInit() {
 this.formdata = new FormGroup({
Title: new FormControl("Angular Lab",
            Validators.required )
    });
 }
```

```
formdata: FormGroup;
constructor(
private formBuilder:FormBuilder){}
ngOnInit() {
this.formdata = this.formBuilder.group({
  Title: ['', [Validators.required]    });
   }
```



Example- Form Control & Group

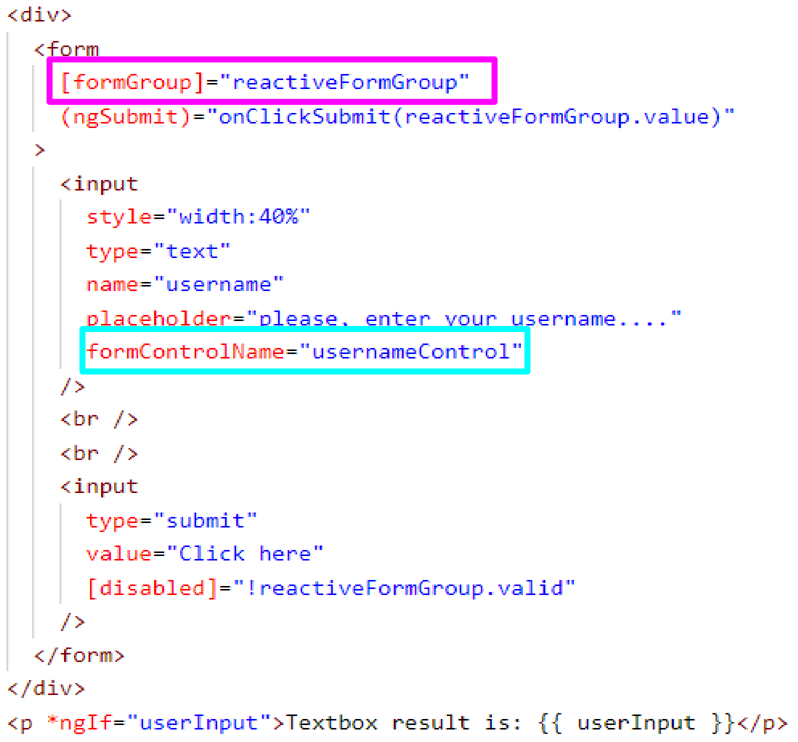


Textbox result is: Salma

- formGroup: The form will be treated as a FormGroup in the component class, so the formGroup directive allows to give a name to the form group.

- ngSubmit: This is the event that will be triggered upon submission of the form.

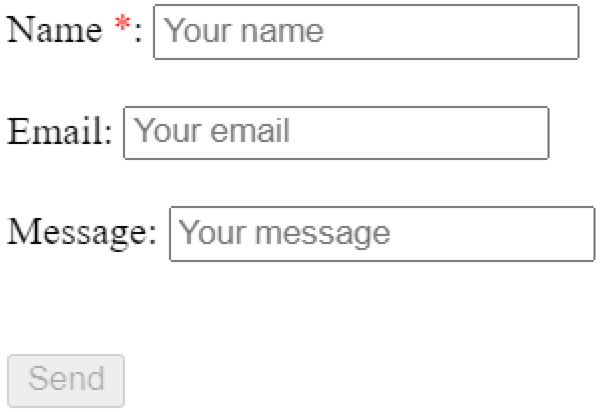
- formControlName: Each form field should have a formControlName directive with a value that will be the name used in the component class.



```
<div>
  <form
    [formGroup]="reactiveFormGroup"
    (ngSubmit)="onClickSubmit(reactiveFormGroup.value)"
  >
    <input
      style="width:40%"
      type="text"
      name="username"
      placeholder="please, enter your username...."
      formControlName="usernameControl"
    />
    <br />
    <br />
    <input
      type="submit"
      value="Click here"
      [disabled]="!reactiveFormGroup.valid"
    />
  </form>
</div>
<p *ngIf="userInput">Textbox result is: {{ userInput }}</p>
```

Example - Form Builder

Let's check it: here



Name *: Your name

Email: Your email

Message: Your message

Form Validation

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
}
```

Numbers

Any

Boolean -> Mandatory Check box

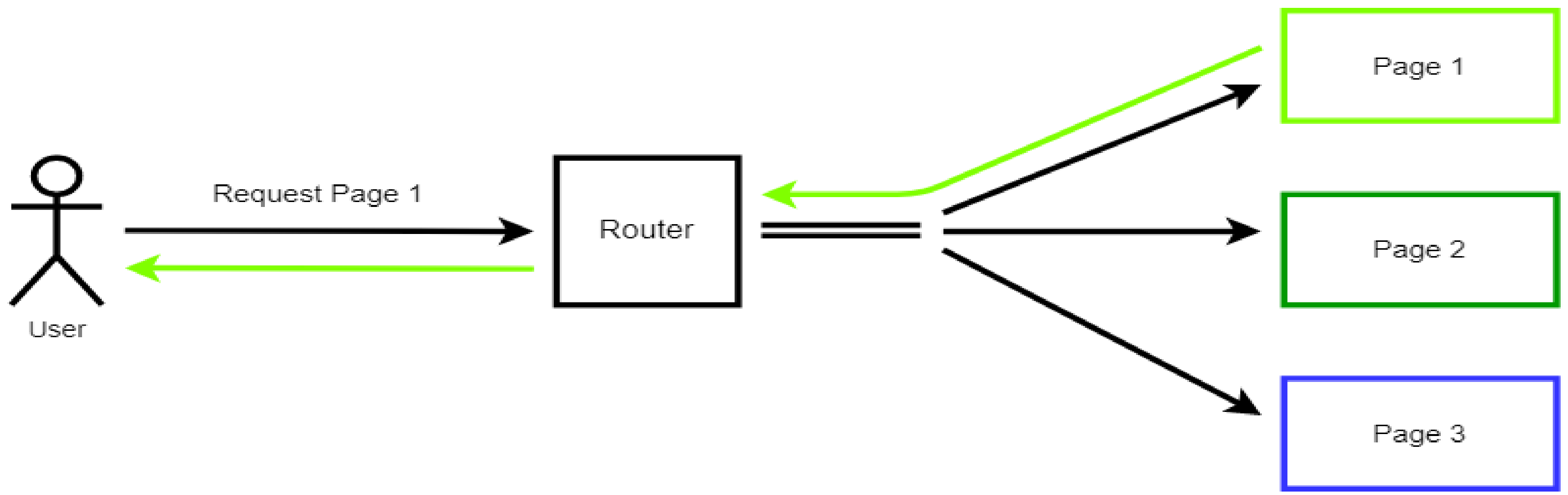
Email Regex example: email@gmail.com

String length

Any Regex for Example: ^[0-9]*

Routing

The Angular router enables developers to build Single Page Application with multiple views and allow navigation between these views.



Angular Routing

- Routing allows you to **navigate within the app** from View to another, also **protect the routes** from unauthorized users using **Guards**.

- The **Router** is a module in a library package called @angular/router which also provides the necessary service providers and directives for navigation and should be imported in appModule.

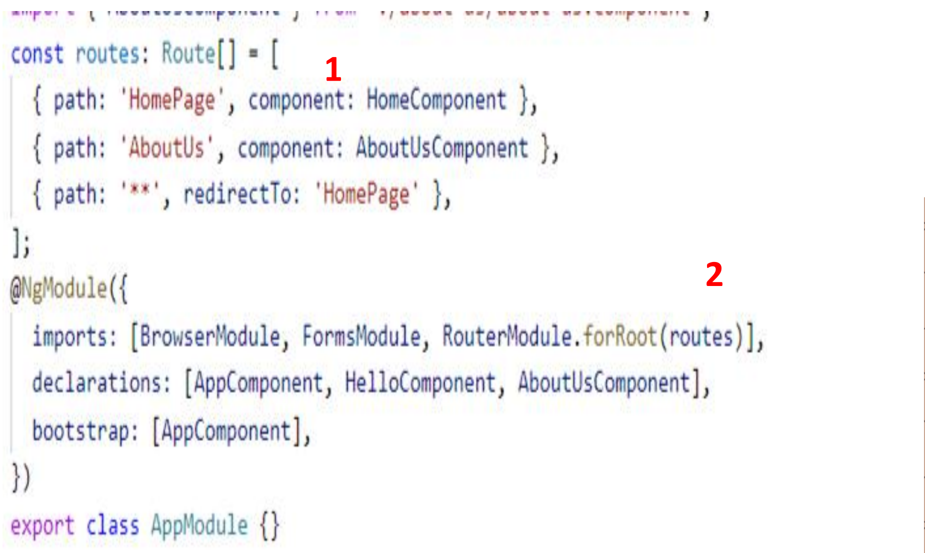
- Essentials in Routing:
  a. **RouterOutlet:** a directive (<router-outlet>) that serves as a placeholder, where the Router should display the view.
  b. **Route (The Path):** tells the Angular Router which view (Component) to display when a user clicks a link or pastes a URL.
  c. **RouterLink:** a directive that binds the HTML element to a Route.
  d. **Router:** an object that enables navigation from one component to the next component.
  e. **ActivatedRoute:** an object that represents the currently activated route associated with the loaded Component (Pass and Get Parameters).

Angular Routing Code Example

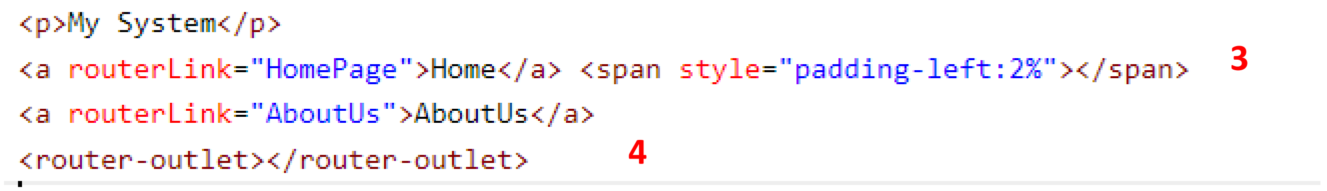
To apply routing:

1. Define the routes and the views that will be loaded for these routes in an array.
2. Register these routes inside the **app.module.ts**.
3. Map action to routes.
4. Choose where it should be displayed as we are using single page.

Note: ***RouterModule.forRoot(appRoutes)*** returns a RouterModule that is configured with your app routes.



```
const routes: Route[] = [
  { path: 'HomePage', component: HomeComponent },
  { path: 'AboutUs', component: AboutUsComponent },
  { path: '**', redirectTo: 'HomePage' },
];
@NgModule({
  imports: [BrowserModule, FormsModule, RouterModule.forRoot(routes)],
  declarations: [AppComponent, HelloComponent, AboutUsComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

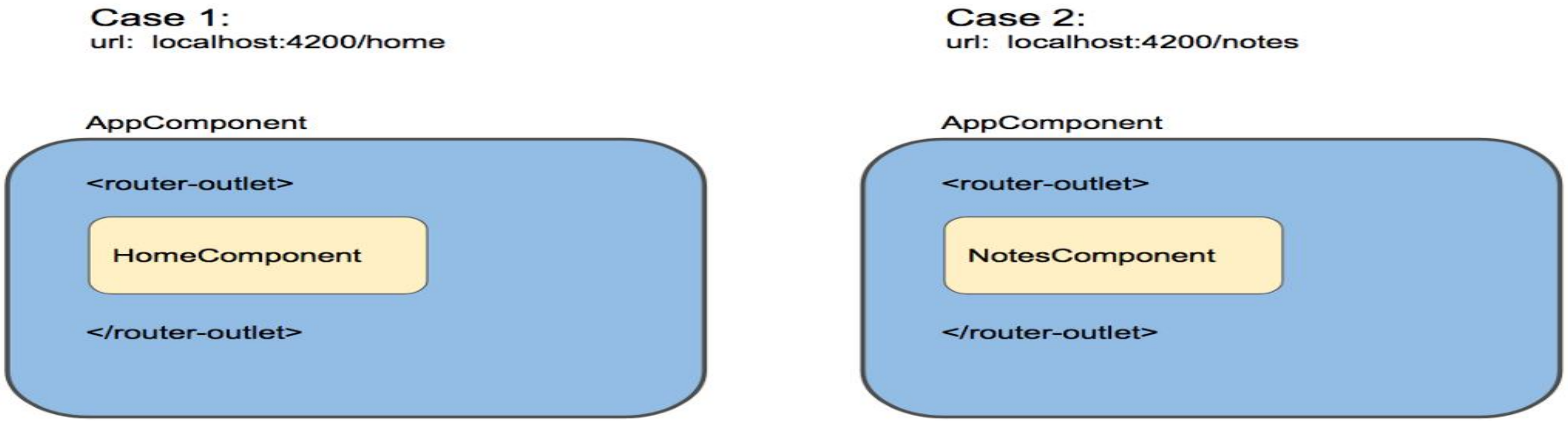


```
<p>My System</p>
<a routerLink="HomePage">Home</a> <span style="padding-left:2%"></span>
<a routerLink="AboutUs">AboutUs</a>
<router-outlet></router-outlet>
```

Routing

The Angular router enables developers to build Single Page Application with multiple views and allow navigation between these views.

The `Router-Outlet` is a directive that's available from the router library where the **Router inserts the component** that gets matched based on the current browser's URL. **Should be Added to Parent Component's template of target links.**



Relative path Vs Absolute path

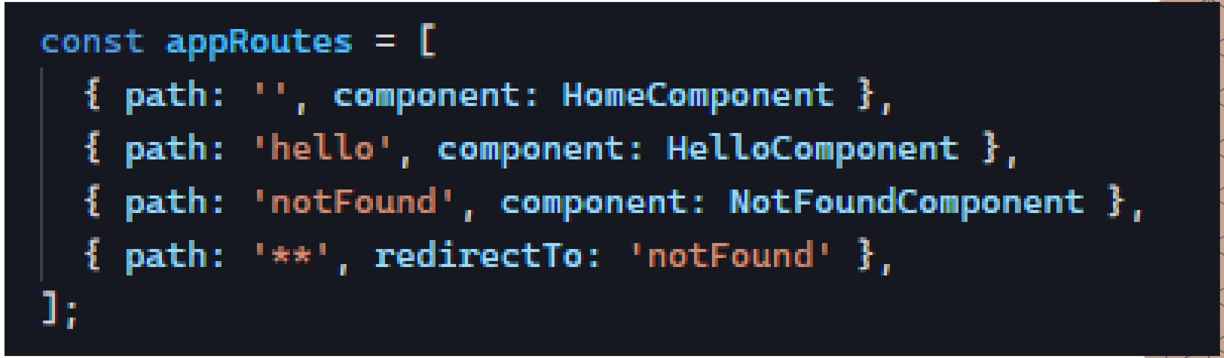
- The difference between **Relative path** and **Absolute path** is that the relative path is a new path that will be added to the path that it is relative to, while the other is the route that will be added to the root path regardless which page you are on now.
  - consider that you are viewing now: **localhost:4200/profile**
    - **Redirecting to '/likes' will give you Absolute path: localhost:4200/likes**
    - **Redirecting to 'likes' will give you Relative path: localhost:4200/profile/likes**

Redirecting and Wildcards

- Wildcard ** lets you catch any unregistered route in the routes array.

Note:

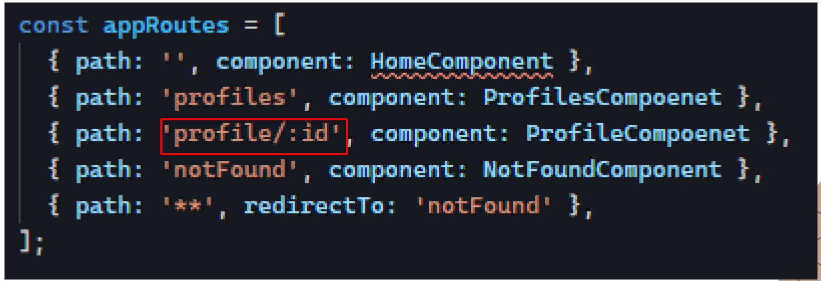
Order matters: First one wins



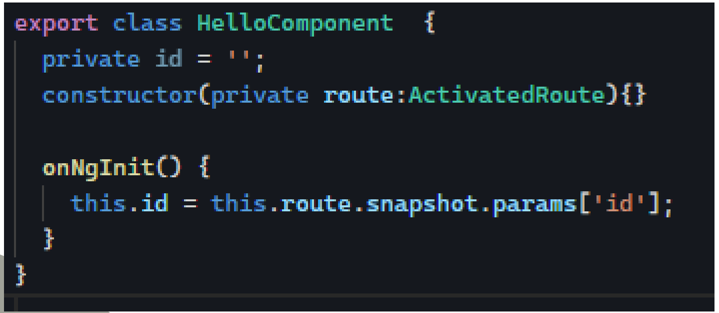
```
const appRoutes = [
  { path: '', component: HomeComponent },
  { path: 'hello', component: HelloComponent },
  { path: 'notFound', component: NotFoundComponent },
  { path: '**', redirectTo: 'notFound' },
];
```

Route (Path) Parameters

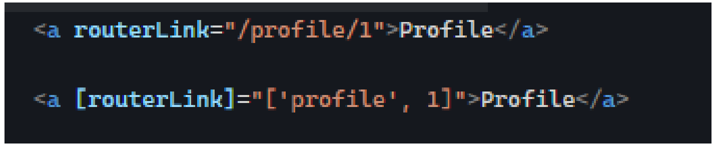
- A placeholder for path params can be registered in the route by using **colon operator** before then the key name **(:key)**
- Then it can be applied on the routerLink directive.
- Finally, you can fetch it within your component file using the **ActivatedRoute** object.



```
const appRoutes = [
  { path: '', component: HomeComponent },
  { path: 'profiles', component: ProfilesCompoenet },
  { path: 'profile/:id', component: ProfileCompoenet },
  { path: 'notFound', component: NotFoundComponent },
  { path: '**', redirectTo: 'notFound' },
];
```



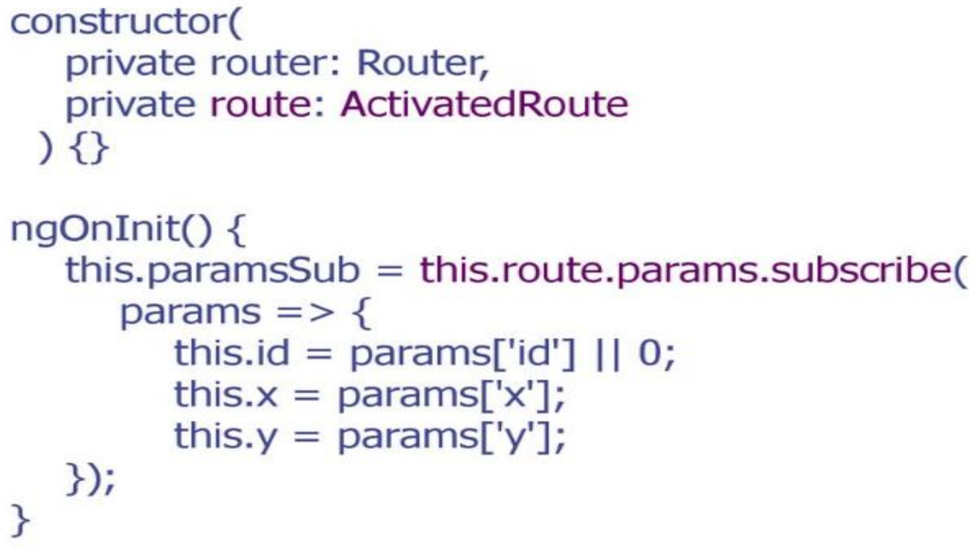
```
export class HelloComponent  {
  private id = '';
  constructor(private route:ActivatedRoute){}

  onNgInit() {
    this.id = this.route.snapshot.params['id'];
  }
}
```



```
<a routerLink="/profile/1">Profile</a>

<a [routerLink]="['profile', 1]">Profile</a>
```

Subscribing to the params Observable

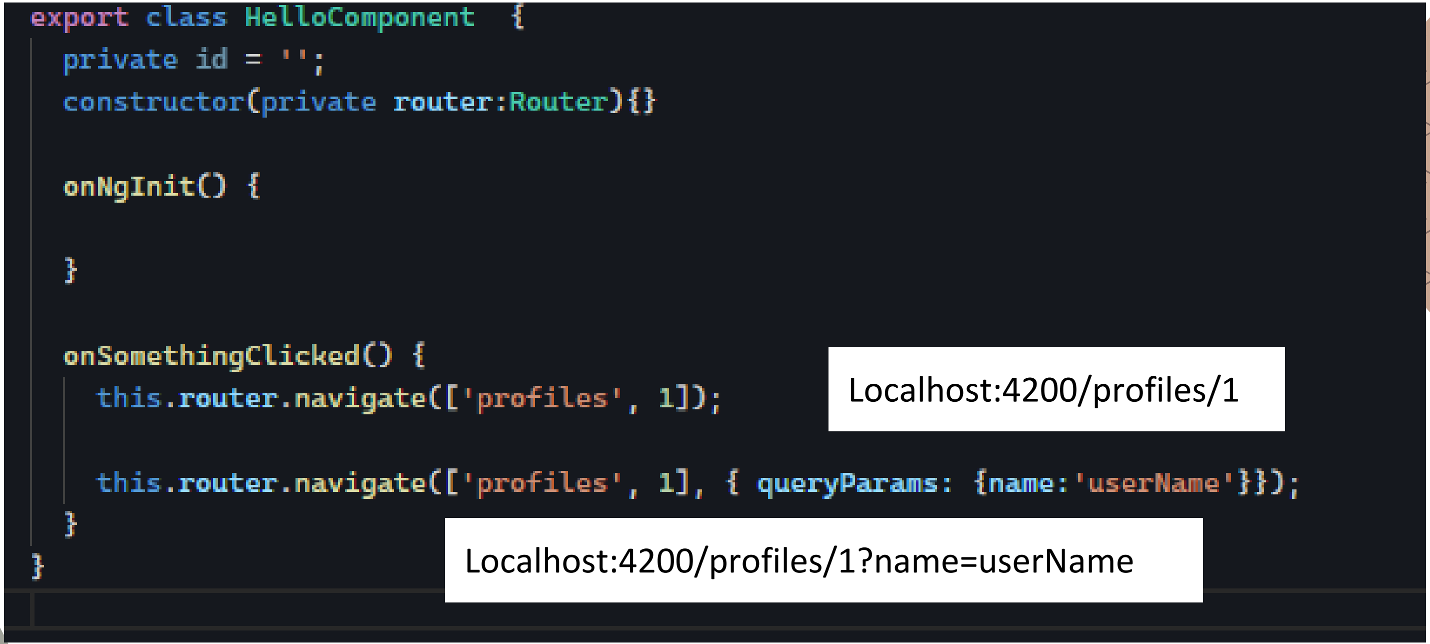
- Angular **reuses components** to improve performance, so the **same component instance** will need to handle the change of params.



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

ngOnInit() {
    this.paramsSub = this.route.params.subscribe(
        params => {
            this.id = params['id'] || 0;
            this.x = params['x'];
            this.y = params['y'];
    });
}
```

Navigating from the component.ts

- All that what we've done was routing from the view side.
- We can also do it programmatically from the **component.ts** by using the **Router** object's function that is called **navigate()**.



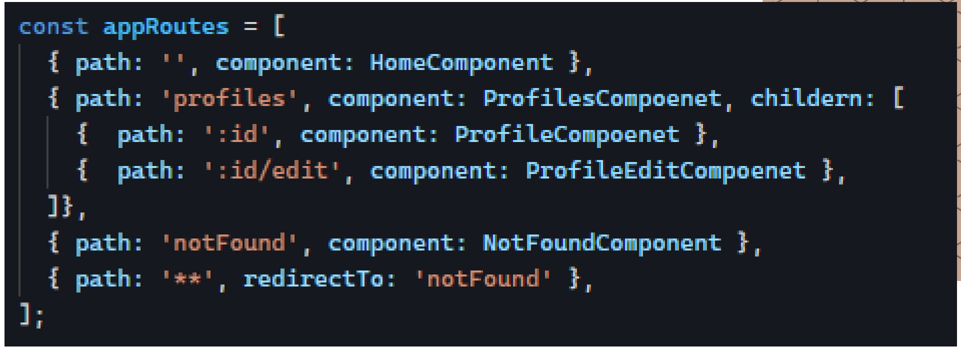
```
export class HelloComponent  {
  private id = '';
  constructor(private router:Router){}

  onNgInit() {

  }

  onSomethingClicked() {
    this.router.navigate(['profiles', 1]);

    this.router.navigate(['profiles', 1], { queryParams: {name:'userName'}});
  }
}
```

Localhost:4200/profiles/1

Localhost:4200/profiles/1?name=userName

Child Routes

- It’s easy to implement the child routes which is basically nesting routes.
- Update the routes array to make a sub-group using **children array**.
- Then nest another **router-outlet** in the parents’ html file**.**
- **Note: check the DOM tree to fully understand what’s the nesting in our case.**



```
const appRoutes = [
  { path: '', component: HomeComponent },
  { path: 'profiles', component: ProfilesCompoenet, childern: [
    {  path: ':id', component: ProfileCompoenet },
    {  path: ':id/edit', component: ProfileEditCompoenet },
  ]},
  { path: 'notFound', component: NotFoundComponent },
  { path: '**', redirectTo: 'notFound' },
];
```

Task 4

- Applying Forms, Validations
- Applying routing

Thank you

Any question?