**Section 11: Changing Pages with Routing**

**Section 11: Lecture 112//Module Introduction**

1. We already learnt a lot about angular, this far we really built a single page application. We changed directives and so on to change the potions of this page – but in the end we always were on the same page. If we look at the URL on the top it was always localhost: 4200.
2. Now, what if we have page where we want to display several pages – like e have /users or we have /accounts page for this we indeed need multiple pages i.e. multiple index HTML files – but really we don’t need because angular ships with its own router which allows you to change the URL – in the URL bar and still only use one page but then exchange major parts or lot of parts of that page.
3. To the user it looks like a new page was loaded may because only the headed is the same. But behind the scenes it is still JavaScript changing lot of parts in your DOM/in your page, making it look like new page was rendered but you are still in the angular world i.e. in your single page application
4. Let’s take a closer look in this module.

**Section 11: Lecture 113// Why do we need a Router?**

1. Attached to the section we have an example project. We can notice in that project that we cannot switch the pages.
2. To work in this module, we need angular router. We must know which routs our project application has.
3. We want to split all the given tabs in the application into multiple pages i.e. multiple page looking components as angular creates a single page application.
4. Here we have users, servers, and Home on the same page – so that is something we can improve. It would be nice to have separate URLs for these 3 tabs and for this we need angular router.
5. Angular router knows which route our front end application has. Notice pattern from a full stack application where you also register some routes. Now, we can do the same thing in angular.

**Section 11: Lecture 114//Understanding the example project**

1. In our app, we got three sections:

* Home
* Servers
  + View and Edit Servers
  + A Service is used to load and update Servers
* Users
  + View Users

This app will be improved by adding routing but feel free to play around with it - besides routing, everything should be working fine.

**Section 11: Lecture 115//Setting up and loading routes**

1. So, here I am in the source code responsible for the app we saw in the last lecture. Now we can see that we have the 3 major components and then some sub-components of these components.
2. In the app.component.html I am loading them all i.e. my home component, server’s component, and users’ component. For now, we will ignore the sub components.
3. Now, where do we register such routes? – these routes are responsible for our whole app.
4. Now, if we enter /users in the URL then we want to load the users’ component as our main pagelet. Now, this is the core part of our app, so, where should we register it? - well, app.module.ts sounds like a good place, because here is where we configure our app. Here only we add our components and so on.
5. Now, here we will add a new constant called appRoutes; now, this constant should be of a type the routes type which must be imported from @angular/router. This constant should hold an array because our application has multiple routes.
6. We add all the routes and want to setup this array for now. Each route for now is just a JavaScript object in this array.
7. Now, the question is that how does such a rout should be configured in an angular app? - It must follow a pattern and a structure for angular to be able to use it. And this structure always needs a path.
8. Just give the name of the path in the object in the constant and define what should happen when this path is reached. The action typically is the component. Now, we will inform angular here that when the certain path here is reached then this component should be reached. And this component would then be the page which gets loaded.
9. We must make sure we are configuring the components in our app such that they look like pages.
10. Now, just by adding that constant angular won’t know what to do. How would angular know that you want to use this constant? – so, we somehow must register these routes in our app and we do this by adding a new import here i.e. we must add the router module.
11. Using RouteModule.forRoot(appRoutes) we would be able to register our routing functionality in our app. And now angular knows our routes.
12. The missing piece is to render the currently selected component.
13. app.module.ts
14. import { BrowserModule } from '@angular/platform-browser';
15. import { NgModule } from '@angular/core';
16. import { FormsModule } from '@angular/forms';
17. import { HttpModule } from '@angular/http';
18. import { AppComponent } from './app.component';
19. import { HomeComponent } from './home/home.component';
20. import { UsersComponent } from './users/users.component';
21. import { ServersComponent } from './servers/servers.component';
22. import { UserComponent } from './users/user/user.component';
23. import { EditServerComponent } from './servers/edit-server/edit-server.component';
24. import { ServerComponent } from './servers/server/server.component';
25. import { ServersService } from './servers/servers.service';
26. import { Routes, RouterModule } from '@angular/router';
28. const appRoutes: Routes = [
29. { path: '', component: HomeComponent },
30. { path: 'users', component: UserComponent },
31. { path: 'servers', component: ServerComponent }
32. ];
33. @NgModule({
34. declarations: [
35. AppComponent,
36. HomeComponent,
37. UsersComponent,
38. ServersComponent,
39. UserComponent,
40. EditServerComponent,
41. ServerComponent
42. ],
43. imports: [
44. BrowserModule,
45. FormsModule,
46. HttpModule,
47. RouterModule.forRoot(appRoutes)
48. ],
49. providers: [ServersService],
50. bootstrap: [AppComponent]
51. })
52. export class AppModule { }

14. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<ul class="nav nav-tabs">

<li role="presentation" class="active"><a href="#">Home</a></li>

<li role="presentation"><a href="#">Servers</a></li>

<li role="presentation"><a href="#">Users</a></li>

</ul>

</div>

</div>

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<router-outlet></router-outlet>

</div>

</div>

</div>

**Section 11: Lecture 116// Navigating with Router Links**

1. As we can see that right now, we can change pages or make routing work by typing the URLs not by just clicking the buttons or menu items. It would also be nice to make some links work in our application here.
2. So, lets add some links to our application. Now, when we give link paths for routing in the app.component.html we see that now when we click on the menu item buttons the page reloads.
3. Every link we click the page reloads. This however is not the best behavior because restarts our app on every click to the button. That means our whole application state will be lost. It might not be the behavior we want to offer to the user.
4. So, this not how we should implement navigation. How should we implement then? - there is a special directive angular gives us i.e. routerLink.
5. Another way of using routerLink is by using Property Binding.
6. app.component.html:
7. <div class="container">
8. <div class="row">
9. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
10. <ul class="nav nav-tabs">
11. <li role="presentation" class="active"><a routerLink="/">Home</a></li>
12. <li role="presentation"><a routerLink="/servers">Servers</a></li>
13. <li role="presentation"><a [routerLink]="['/users']">Users</a></li>
14. </ul>
15. </div>
16. </div>
17. <div class="row">
18. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
19. <router-outlet></router-outlet>
20. </div>
21. </div>
22. </div>

**Section 11: Lecture 117//Understanding Navigation Paths**

1. In the last lecture we added this navigation, we also mentioned that we can change the way in which we use the paths here.
2. Now, in the servers.component.html we added this router link which was earlier only present at app.component.html.
3. servers.component.html:
4. <div class="row">
5. <div class="col-xs-12 col-sm-4">
6. <div class="list-group">
7. <a
8. href="#"
9. class="list-group-item"
10. \*ngFor="let server of servers">
11. {{ server.name }}
12. </a>
13. </div>
14. </div>
15. <div class="col-xs-12 col-sm-4">
16. <a routerLink="../server">Reload Page</a>
17. <app-edit-server></app-edit-server>
18. <hr>
19. <app-server></app-server>
20. </div>
21. </div>

**Section 11: Lecture 118//Styling active router links**

1. One thing is that we are not getting any indication of what the currently active route is, this is purely CSS thing.
2. Now, home page here is set to active by default. But we can set it dynamically to know which route is currently executing. Angular gives us a directive for this i.e. routerLinkActive directive.

**Section 11: Lecture 119//Navigating Programmatically**

1. Till now we have learnt how to add routs and how to load them. Either by navigation bar or by clicking some links. What now if we want to load a route programmatically.
2. Say, we finished some operation or the user clicked some button or the user clicked some button and then we want to trigger the navigation from our typescript code.
3. Let’s say in our home component here we add a new button to it and on this button I simply want to load the server.
4. Now, when the button is clicked and the click method is called we need to tell this to angular router that please navigate somewhere else.
5. [home.component.html](http://home.component.html):
6. <h4>Welcome to Server Manager 4.0</h4>
7. <p>Manage your Servers and Users.</p>
8. <button class ="btn btn-primay" (click) = "onLoadServers()">Load Servers</button>
10. [Home.component.ts](http://Home.component.ts):
11. import { Component, OnInit } from '@angular/core';
12. import { Router } from '@angular/router';
13. @Component({
14. selector: 'app-home',
15. templateUrl: './home.component.html',
16. styleUrls: ['./home.component.css']
17. })
18. export class HomeComponent implements OnInit {
19. constructor(private router: Router) { }
20. ngOnInit() {
21. }
22. onLoadServers(){
23. //complex calculation
24. this.router.navigate(['/servers']);
25. }
26. }

**Section 11: Lecture 120 //Using Relative Paths in Programmatic navigation**

1. In the last lecture we learnt how to navigate programmatically, now, we will look at how to use relative paths there.
2. Now, let’s say in our servers.component.htm, we add a button for the reload of the page. We will also inject the router in the services.component.ts
3. Let’s have the reload button this time instead of the reload link as we did in the last lecture. Here I will have my click listener on the reload page maybe.
4. Now, we are already injecting the service in the servers.component.ts, we will also import router in the similar way. So, here we added the router in the Constructor.
5. When we click this reload page then the request will never be sent to server and the page will not show any signs of being reloaded but internally it still does reload.
6. Unlike the router link the navigate method does not know on which route you are currently on.
7. The router link always knows in which component it sits and in which component is the template is, and therefore it knows where the currently loaded route is.
8. Now, to tell the navigate method where we are at present in the routing i.e. second perimeter in the navigate method which would be a JavaScript object. We will come to this later as there are more things we can add. One configuration is **relativeTo** property – meaning- relative to which route this link will be loaded, by default this should always be the link to main. Here we `must give a route though, we don’t have to give a string here.
9. The route which we intend to add in the navigate method can be also injected in the constructor, which is of the type ActivatedRoute.
10. ActivatedRoute injects the currently active routes for the component we loaded. Route is a complex object which keeps a lot of meta information about the currently active route. Now we can set this value for this route for the relative to property.
11. With this extra piece of information angular knows what the currently active route is.
12. So, here we learnt how to use the relative paths in the navigate method. With the second perameter we would be able to define the relative path to which we want to navigate the application.
13. servers.component.ts:
14. import { Component, OnInit } from '@angular/core';
15. import { ServersService } from './servers.service';
16. import { Router, ActivatedRoute } from '@angular/router';
17. @Component({
18. selector: 'app-servers',
19. templateUrl: './servers.component.html',
20. styleUrls: ['./servers.component.css']
21. })
22. export class ServersComponent implements OnInit {
23. private servers: {id: number, name: string, status: string}[] = [];
24. constructor(private serversService: ServersService, private router: Router, private route: ActivatedRoute) { }
25. ngOnInit() {
26. this.servers = this.serversService.getServers();
27. }
28. onReload(){
29. this.router.navigate(['servers'], {relativeTo:this.route});
30. }
31. }

**Section 11: Lecture 121//Passing Parameters to Routes**

1. We are now adding some other routes to app.module.ts; now; now let’s say besides our users route here we should be able to load the single. Now we would need to give path for the user inside the constant we declared inside the app.module.ts.
2. Now, here we will provide path as

{ path: 'users/:id', component: UserComponent },

1. Colon simply tells angular that this the dynamic part of the path
2. app.module.ts:
3. import { BrowserModule } from '@angular/platform-browser';
4. import { NgModule } from '@angular/core';
5. import { FormsModule } from '@angular/forms';
6. import { HttpModule } from '@angular/http';
7. import { AppComponent } from './app.component';
8. import { HomeComponent } from './home/home.component';
9. import { ServersComponent } from './servers/servers.component';
10. import { UserComponent } from './users/user/user.component';
11. import { UsersComponent } from './users/users.component';
12. import { EditServerComponent } from './servers/edit-server/edit-server.component';
13. import { ServerComponent } from './servers/server/server.component';
14. import { ServersService } from './servers/servers.service';
15. import { Routes, RouterModule } from '@angular/router';
17. const appRoutes: Routes = [
18. { path: '', component: HomeComponent },
19. { path: 'users', component: UsersComponent },
20. { path: 'users/:id', component: UserComponent },
21. { path: 'servers', component: ServerComponent }
22. ];
23. @NgModule({
24. declarations: [
25. AppComponent,
26. HomeComponent,
27. UsersComponent,
28. ServersComponent,
29. UserComponent,
30. EditServerComponent,
31. ServerComponent
32. ],
33. imports: [
34. BrowserModule,
35. FormsModule,
36. HttpModule,
37. RouterModule.forRoot(appRoutes)
38. ],
39. providers: [ServersService],
40. bootstrap: [AppComponent]
41. })
42. export class AppModule { }

**Section 11: Lecture 122//Fetching Route Parameters**

1. In the last lecture we created our route with dynamic path segment, now we want to have access of the data which user sent or which is encoded in the URL I say.
2. So, we noted that we will load the user component here and we know that there will be some data in the URL as well. How could we get access to it - now it is the typescript file in which we can get access of it.
3. Now, we need to inject the same things through the constructor that we injected earlier i.e. the Active route i.e. ActivatedRoute.
4. Now, we know that that the currently loaded route is the JavaScript object with the metadata about the currently loaded route. One of the important information is the currently active user.
5. We can see in the app.component.user file that we have defined a user in the file and for now it should have the following structure and it’s not used right now.
6. Now, after getting the parameters from the URL, we can output those using the string interpolation
7. app.module.ts:
8. import { BrowserModule } from '@angular/platform-browser';
9. import { NgModule } from '@angular/core';
10. import { FormsModule } from '@angular/forms';
11. import { HttpModule } from '@angular/http';
12. import { AppComponent } from './app.component';
13. import { HomeComponent } from './home/home.component';
14. import { ServersComponent } from './servers/servers.component';
15. import { UserComponent } from './users/user/user.component';
16. import { UsersComponent } from './users/users.component';
17. import { EditServerComponent } from './servers/edit-server/edit-server.component';
18. import { ServerComponent } from './servers/server/server.component';
19. import { ServersService } from './servers/servers.service';
20. import { Routes, RouterModule } from '@angular/router';
22. const appRoutes: Routes = [
23. { path: '', component: HomeComponent },
24. { path: 'users', component: UsersComponent },
25. { path: 'users/:id/:name', component: UserComponent },
26. { path: 'servers', component: ServerComponent }
27. ];
28. @NgModule({
29. declarations: [
30. AppComponent,
31. HomeComponent,
32. UsersComponent,
33. ServersComponent,
34. UserComponent,
35. EditServerComponent,
36. ServerComponent
37. ],
38. imports: [
39. BrowserModule,
40. FormsModule,
41. HttpModule,
42. RouterModule.forRoot(appRoutes)
43. ],
44. providers: [ServersService],
45. bootstrap: [AppComponent]
46. })
47. export class AppModule { }

8. user.component.ts:

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-user',

templateUrl: './user.component.html',

styleUrls: ['./user.component.css']

})

export class UserComponent implements OnInit {

user: {id: number, name: string};

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user = { id: this.route.snapshot.params['id'],

name:this.route.snapshot.params['name'] };

}

}

1. user.component.html:
2. <p>User with {{ user.id }}loaded.</p>
3. <p>User name is {{ user.name }}</p>

**Section 11: Lecture 123//Fetching Route Parameters Reactively**

1. Now, in our user component we saw that we passed the user ID and URL, lets quickly add a router link in the user.component.html.
2. Now we will add an array in the router link as shown below which will act as /user/10/Anna
3. Once we click load Anna here; routing will take us to this link - /user/10/Anna; here we load our data using the snapshot object of the route.
4. Now, if we load a new route what will happen - Angular will look into our app.module.ts – finds the fitting route here – loads the component – initializes the component and gives us the data by accessing the snapshot here.
5. This all only happens if we haven’t been on this component before but if we click this link which is on the user component 🡪 the URL still changes but we are on the component already. Here angular cleverly doesn’t instantiate this component – that would only cost the performance by re-rendering the component we already are on.
6. By default, angular will not destroy the current component and re-create the same component again if we are on that component already.
7. Its fine to use the snapshot for the first initialization but to be able to react to the subsequent changes we need a different approach. In our user.component.ts
8. Now, we can use our route object i.e. the params property of the route object itself, we didn’t use that before we have the snapshot in between.
9. Params here is an observable; observable is something on which we will come to right after this section.
10. Basically, **observables** are the features added by a third party package and not by angular but its heavily used by angular which allows you to work easily with **Asynchronous tasks**.
11. Observable is used to subscribe some event which might happen in future to then execute this code when it happens.
12. Now, as the name describes observable is something which we can observe. So, we will call subscribe method on it and this is still called on params. Whenever the parameters change in this case the observable will be executed.
13. Now, we must pass arguments in the subscribe method – subscribe can take 3 functions as arguments – the 1st one is the most important here, this would be fired when new data will be passed through the observable. To put in the easier words whenever the parameters change in this use case then this first argument which should be a function here is executed.
14. We will use the ES6 arrow function which will take prams as an argument and we can set this as the type param.
15. Now, inside the function, which is the first parameter in the subscribe function we can fetch the user object and pass id from the params.
16. So, this will now update our user object whenever the parameter changes.
17. This code will not be executed when ngOnInit runs through and the subscription will be set up; now only if then parameters change then the inside part of the subscribe function will run.
18. If you know that the component you are working on might not reloaded from within the component, then we might not need this.
19. user.component.ts:
20. import { Component, OnInit } from '@angular/core';
21. import { ActivatedRoute, Params } from '@angular/router';
22. import { paramKey } from 'blocking-proxy/built/lib/webdriver\_commands';
23. @Component({
24. selector: 'app-user',
25. templateUrl: './user.component.html',
26. styleUrls: ['./user.component.css']
27. })
28. export class UserComponent implements OnInit {
29. user: {id: number, name: string};
30. constructor(private route: ActivatedRoute) { }
31. ngOnInit() {
32. this.user = { id: this.route.snapshot.params['id'],
33. name:this.route.snapshot.params['name'] };
34. this.route.params
35. .subscribe(
36. (params: Params) => {
37. this.user.id = params['id'];
38. this.user.name = params['name'];
39. }
40. );
41. }
42. }

20. user.component.html:

] <p>User with {{ user.id }}loaded.</p>

<p>User name is {{ user.name }}</p>

<hr>

<a [routerLink]="['/users', 10, 'Anna']">Load Anna (10)</a>

**Section 11: Lecture 124 //An important Note about Route Observables**

1. In the last lecture we learnt that how we can subscribe to our params to update them or to react to any changes and update our page if we already are on that page then we set up this subscription using observables and you don’t have to change anything here.
2. Here something is there which needs our attention though, the fact that we don’t need to add anything to this component is that angular does something for us here in the background which is super important for us. It cleans up the subscription that we created whenever this component is destroyed because if wouldn’t do it what are we doing here - we are subscribing to the parameter changes and then leave this component and later we come back. So, once we come back to this component then the same component will be created again and the subscription will remain in some memory. As, the component is destroyed the subscription won’t.
3. It will be there as the angular handles the destroying of the subscription for us but theoretically we want to implement on destroy lifecycle hook from @Core
4. And then we can store this subscription in some property. rxjs is the package offering all this observables functionality and as we missioned it is not shipping with angular. Now we can bound the subscription with this property and later when the component is destroyed then we can unsubscribe the subscription using that property.
5. It’s important to know that we don’t have to do this and angular will do it for us, but even if we does this manually then also it will not heart and will not add anything bad to our app.
6. User.component.ts
7. import { Component, OnInit, OnDestroy } from '@angular/core';
8. import { ActivatedRoute, Params } from '@angular/router';
9. import { paramKey } from 'blocking-proxy/built/lib/webdriver\_commands';
10. import { Subscription } from 'rxjs/Subscription';
11. @Component({
12. selector: 'app-user',
13. templateUrl: './user.component.html',
14. styleUrls: ['./user.component.css']
15. })
16. export class UserComponent implements OnInit, OnDestroy {
17. user: {id: number, name: string};
18. paramsSubsription: Subscription;
19. constructor(private route: ActivatedRoute) { }
20. ngOnInit() {
21. this.user = { id: this.route.snapshot.params['id'],
22. name:this.route.snapshot.params['name'] };
23. this.paramsSubsription = this.route.params
24. .subscribe(
25. (params: Params) => {
26. this.user.id = params['id'];
27. this.user.name = params['name'];
28. }
29. );
30. }
31. ngOnDestroy(){
32. this.paramsSubsription.unsubscribe();
33. }
34. }

**Section 11: Lecture 125//Passing Query Parameters and Fragments**

1. In the last lecture we learnt how we can retrieve our route params that awesome and this is super important, we will probably use it a lot.
2. Now, there are more things that we can add to our url; you might have query parameters separated by a question mark

Ex: localhost:4200/users/10/Anna?mode=editing

1. We can have multiple query parameters separated by the add i.e. & signs, question is that ow you can pass them using angular links/router links and how can you then retrieve them. You might also have # fragments
2. We can us the # sign to jump to the places in our app or maybe you just want to use it to retrieve some extra information from the URL. Let’s have a closer look at how we can pass the extra information in our links both when using our router link method and navigate method, and how we can then retrieve the information.
3. Let’s start by passing it first; lets say in app.module.ts we want to add more routes; lets say that would be the route to allow us to add certain server.
4. 'servers/:id/edit'
5. Here we have added edit in the end to describe what will happen if we access this component.
6. To be able to load this route in my servers component; now you say we also want to have some query parameter which defines whether we are allow to edit the parameter or not so and we will not add the ? in the routerLink array. But we will bind this to the [queryParams] property.
7. queryParams is not a new directive but it is the bind able property of the routerLink directive and here we must pass the JavaScript object and here in the queryParams we can add the name value pairs in the object in this.
8. Like queryParams property we also have the fragment property. This is named fragment as we ca only have 1 fragment
9. Now, here we will make the id dynamic as we were hard codding it earlier.
10. Now, in the [home.component.ts](http://home.component.ts) we can add parameter 2 which would be an object in the navigate method i.e. the queryParams
11. Like the queryParams we can also add the third parameter that would be the fragment
12. [Home.component.ts](http://Home.component.ts):
13. import { Component, OnInit } from '@angular/core';
14. import { Router } from '@angular/router';
15. @Component({
16. selector: 'app-home',
17. templateUrl: './home.component.html',
18. styleUrls: ['./home.component.css']
19. })
20. export class HomeComponent implements OnInit {
21. constructor(private router: Router) { }
22. ngOnInit() {
23. }
24. onLoadServers(id: number){
25. //complex calculation
26. this.router.navigate(['/servers',id, 'edit'], {queryParams: {allowEdit: '1'}, fragment:'loading'});
27. }
28. }
29. servers.component.html:
30. <div class="row">
31. <div class="col-xs-12 col-sm-4">
32. <div class="list-group">
33. <a
34. [routerLink]="['/server', 5, 'edit']"
35. [queryParams]="{allowEdit: '1'}"
36. fragment="loading"
37. href="#"
38. class="list-group-item"
39. \*ngFor="let server of servers">
40. {{ server.name }}
41. </a>
42. </div>
43. </div>
44. <div class="col-xs-12 col-sm-4">
45. <!-- <a routerLink="../server">Reload Page</a> -->
46. <button class="btn btn-primary" (click)="onReload()">Reload Page</button>
47. <app-edit-server></app-edit-server>
48. <hr>
49. <app-server></app-server>
50. </div>
51. </div>

**Section 11: Lecture 126 //Retrieving Query Parameters and Fragments**

1.