**127. Navigating with Router Links:**

* -: In the last lecture, we made our routing work.
* We added routes and we gave Angular a place to load our currently active component.
* But one issue we have is right now we can only navigate around by manually typing it here in the address bar.
* Not super convenient.
* And while it is important to be able to load a route directly from the address bar, it would also be nice to have some working links here in our navigation to have these tabs work.
* So let's add some links to our application.
* And if we go to our app component, we see, here's our navigation.
* The first thing you might think about is, "Hey, we can or we already have this href attribute here to be precise.
* " So here we could have href, we'll just slash load our root route.
* Here we could have /servers, And here we could have /users.
* And if we now save this, we added all the routes, the paths we set up in our routes configuration to our navigation here.

Text

Description automatically generated

* And hence if this reloads and we click the tabs, Hey, does this indeed working? Now we are correctly loading the route you want to load.
* *Okay, this tab is not updated but I will come back to this later.*
* *But the reloading is working.*
* And that's the issue.
* It's reloading the app.
* Watch this reload icon here.
* *Do you see this? We refresh the app with every link we click.*
* *And that's the natural behavior because with every link we click, a new request gets sent to the server and it returns as a new page.*
* *And since this page is still our Angular app with the routes registered on it, it is able to give us the correct route.*
* So, the same thing happens as if we entered this manually here.
* *That, however, is not the best behavior, because it means it restarts our app on every navigation.*
* ***That of course means our whole application state will be lost and it might not really be the behavior we want to offer to the user.***
* So this is not how you should implement navigation.
* ***How should you implement it then?*** There is a special directive Angular gives us.

**routerLink**

* Let's get rid of all these href attributes here.
* So this is not how we will navigate around.
* Instead, let's use this special directive.
* It's called "routerLink.
* " *Now, routerLink like this simply is able to parse a string where we could pass just slash.*
* *So just this string, slash to routerLink here.*
* ***Now this will tell Angular that this element on which routerLink is placed here, this anchor tag here, will serve as a link in the end, but it will handle a click differently as you will see in a second.***
* Let's now also add this to the Servers link.
* So routerLink equals /servers.
* And I will come back to how you may write the path here in the next lecture.
* Another way of using routerLink is with property binding, so you can enclose it in square brackets.
* And now of course you can't just pass /users here because this would now search for a property with that name which would even be an invalid name in JavaScript.
* So now you have to pass a string here with single quotation marks -: -or, better--an array which gives you a more fine-grain control over routerLink.
* And I will come back to this later, when this is especially useful.

Text

Description automatically generated

* Here, in this array, you now specify all these segments of your path as elements in this array.
* So, the first segment in this case is only a string: /users.
* and if you had /users, /something, you would have a second element here which is then just something without a slash.
* The leading slash here is only needed to make this an absolute path and I will come back to this in the next lecture.
* For now, we'll get rid of this.
* Now, this array notation is not super convenient, but soon you will see when it really gives you an advantage or when you need this notation.
* It allows you to construct more complex paths very easily.
* So with this, we set up our three links using routerLink, either by passing a string or this array which allows us to define our individual path elements.
* **OutPut in Application browser:**
* ***With that, if we save this and have a look at our application, now you see it's still reloads or it still gives us the components, but it doesn't reload the page.***
* If you watch this reload icon, nothing is happening there.
* *Because routerLink catches the click on the element, prevents the default which would be to send a request, and instead analyzes what we passed here to the routerLink directive.*
* *So, this path or this array of path elements.*
* And then parses it and checks if it finds a fitting route in our configuration, which it of course does because we defined all the paths we're using here.
* And this is how we can navigate around with the routerLink.
* And this actually is how we should navigate around, because it gives us the better user experience.
* It doesn't restart our app, therefore it keeps the app state and it's much faster than reloading the page all the time.
* You can of course still type something here manually so that will still work, But if you're inside of the app using routerLink, it's much better.