**Section 13: Understanding Observables**

**Section 13: Lecture 158//Module Introduction**

1. In the last module about routing we had first contact with observables. Now, we will dive deeper into observables in this section. We have a separate course on observables so we will not dive that deep into them here.
2. Here we will understand what observables are and where we can use them for, and why angular uses them.
3. What is an observable?

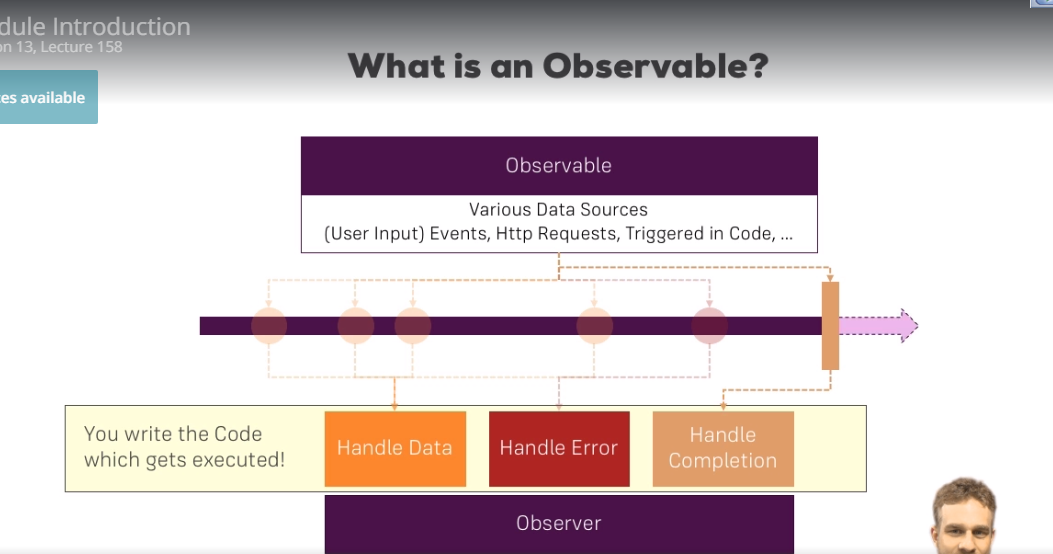
**An observable can be thought of as a data source**, now, in our angular project an observable basically is an object we import from a 3rd party package i.e. RXJS

1. The observable is implemented here such that it follows the observable pattern. So, we have an observable and we have an observer - in between we have a steam i.e. a timeline and on this timeline we have multiple events emitted by the observable or data packages you could say - emitted by observable depending on the data source of that observable.
2. So, an observable could emit data because you triggered it to do so – we can also do that programmatically i.e. it can be linked to a button - therefore when the button is clicked an event or a data package is emitted automatically or as the angular Http service does it - its connected to Http requests. So, when the response returns - the response is emitted as a data package.
3. There are dozens of other data sources too, we will have a look – where to find more soon.
4. So, as we observed - the other part is observer 🡪 this actually is your code, you could say. Its, the subscribe function we saw earlier or at least it has something to do with that.
5. Using observer, we have three ways of handling data packages

Handle Data

Handle error

Handle Completion

1. As shown above – by using observer we can handle normal data, we can handle the error or we can handle the completion of the observable. Because these are the 3 types of the data packages we can receive.
2. In these boxes our code gets executed, so, we can determine what shall we if we receive a new data package, what should happen if we receive an error , what should happen when the observable eventually completes - Note: An Observable doesn’t have to complete.
3. There are observables, for example, hooked up to normal button which neve completes – how would you know, when it completes?
4. Other observables such as http have a clear end and will complete eventually. Because once the response is there, what else should happen? - it’s done
5. This is how the observable pattern generally works and of course we use it to handle asynchronous tasks - because all these data sources here, user event triggered in your code or a HTTP request are asynchronous tasks - you don’t know when they will happen and you don’t know how long they will take. So if you execute your normal application code - you don’t want to wait for such an event and you don’t want to wait for such a HTTP requests, because that would block your program/block your logic.
6. Therefore we need methods for handling such an asynchronous tasks, historically you might have used callbacks and promises - it’s not necessarily bad to use them. Observable is just a different approach of handling that different alternative and angular embraces observables which is why I chose to explain these as angular uses them a lot.
7. Observables have one major advantage i.e. their operators which I will show later in this section too.
8. Back to this slide, we have got our observable and our observer with our free buckets or we can say with our free hooks where our different code is executed depending on the type of package we receive.
9. And then an observable may emit a couple of normal data packages, It might emit an error or it might get completed and the respective code is then executed.
10. 

**Section 13: Lecture 159// Analysing a Built-in Angular Observable**

1. This is a very simple application which we have prepared, which has 3 links. There is one home and 2 user links. If we click user one, then it would open at /user/1 and 2 at /user/2.
2. ID here in the routes gets updated depending on the route that we are using.
3. And of course this is one and the same user component we are using here in the background and we are using an observable. The parents observable Angular or ships within the router package to handle the change of these router parameters because just as a refresher angular doesn’t re-render the whole component just because the parameter changed.
4. Therefore, it uses observables to still give us a chance to react to this updated ID and this is what we do with the observable which angular offers us here now and in the last section we implemented this code and I think it works kind of intuitive because you can basically accept that route params – are some kind of strange asynchronous object and you subscribe to any events that might send you, in this case, updated parameters and then use simple handle these parameters here in those callback we pass to the Subscribe method – keep in mind that this is just argument passed to the subscribe method here this anonymous function where we handle the parameters and then we simply extract the ID in this case .
5. Now, as mentioned we are using observable here – so, which part of the slide, with this code you are basically be the sending part or the receiving part - while the receiving part ofcourse because here we handle the data.
6. The data is sent by as angular, we don’t do this. You could argue we kind of do it by implementing our links here, but that will only kind of trigger it.
7. The emitting of the data will be done by angular which recognizes that we clicked such a router link.
8. We will extract the data from this rattling directive and then will basically navigate us.
9. And at this point of time it will then also push this new data package through this parent observable.
10. So here in subscribe method this actually is our observer part to be precise and to stay in our X chase terms.
11. This is our subscriber – but subscriber here basically is the observer. Therefore, we could implement more callbacks then just one we only implement the handle the normal data call back but the subscribe method always takes the 3 arguments and our ID already shows in there.
12. We could also implement an anonymous function which gets executed in the case of error and one which gets executed if the observable gets completed.
13. Now on these routing observable here that’s not very practical, it doesn’t make much sense because the parents here will not fail and they will not complete. So, the first call here make the most sense.
14. We will soon dive into creating our own observables so that we can actually also see the other possible hooks.
15. We are subscribing to such an observable which wraps that data source being code in angular - which you could say emits a new parameter whenever we click a new link.
16. So, in the end our click is the triggering event but again angular sets up an observable and then pushes it to give us a new piece or a new data package and then we use it here in subscribe function with our subscriber; the subscriber basically just is this free method we pass through subscribe set up which is the Observer.
17. So that is how we use it already as angular does what happens behind the scenes.
18. Now, let’s implement our own observable to understand a bit better how it really works.

**Section 13: Lecture 160//Building and using a first sample Observable**

1. In the last lecture we had a look on this built in observable, now let’s build our own one