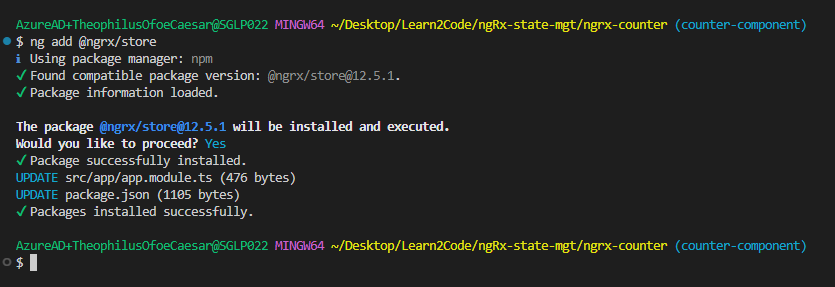
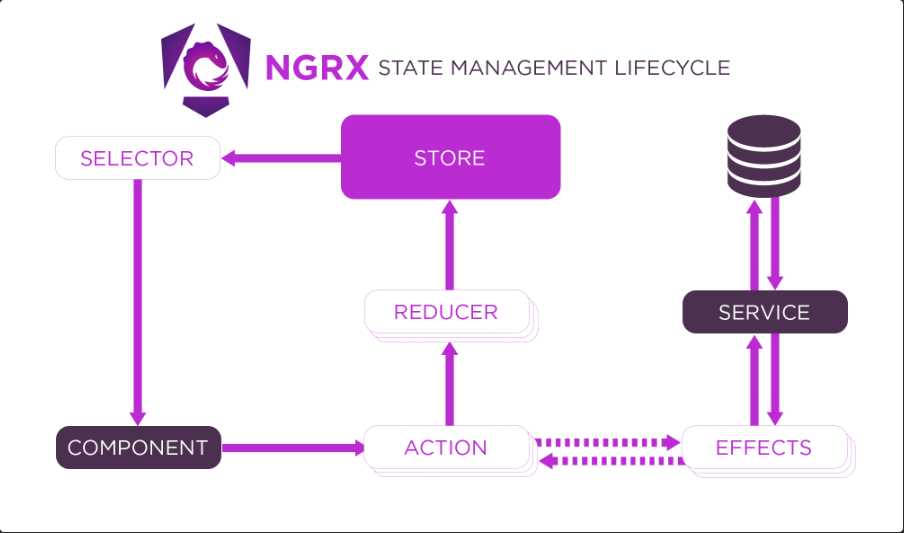


## Install ngrx



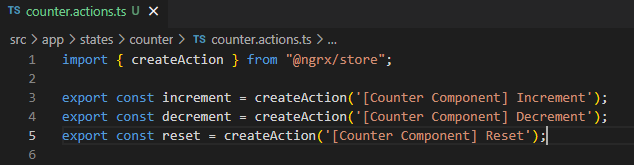
## Getting Started

The first thing we need to create is “actions”.

To segregate all that has to do with state management, dude has us create a folder named states for this purpose.

In there, we will have all to do with state management.

1. Create folder “states”. Dude goes ahead to create subfolders for our component and any additional components we might have in future.

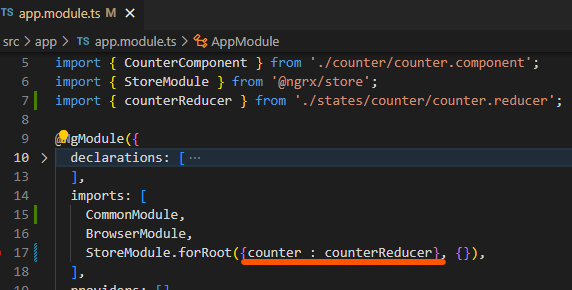


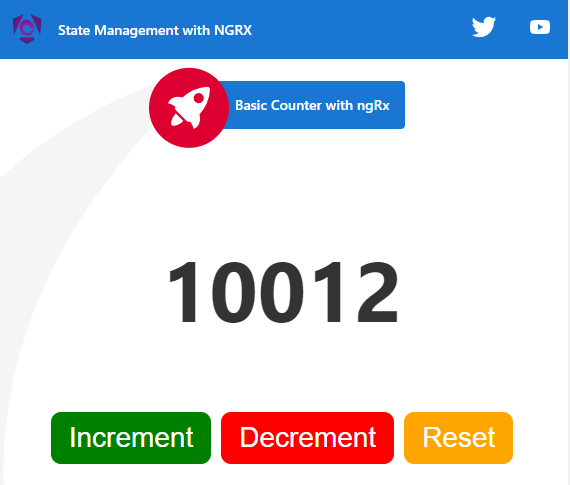
1. Create file for actions – counter.actions.ts. as it stands now, this is more of a description of shit.
2. A computer screen shot of a program code

   Description automatically generatedWe will then have our logic for the created actions in a reducer – counter.reducer.ts. In here we first have an interface that’s going to depict the properties of our resulting object.

We init our object, based on which we will be applying logic to.  
in lines 12 to 16, we have the createReducer method perform some actions on our init object by just spreading the props and then altering the \_count\_ prop.

1. A screenshot of a computer

   Description automatically generatedWith the logic done, the resulting data will now have to be accessible within our global store state. We create an app.state.ts file within which we will define the structure of our store state with an interface. Pay attention to the key value pairs in our interface, we will need it in our app module to make the values accessible app wide.
2. We create a selector which is going to help us access pieces of data from our store state (app-state). Line 4 in our snippet basically takes our store state and then returns value being pointed to by “counter\_” key – which is our reducer’s initial object.   
   Lines 6-7 basically take our selectCounterState (which at its lowest base, is our reducer’s init object – init counter state) and uses it as an argument in our second parameter (function), then the function accesses the \_count\_ property and ultimately returns the value.
3. Finally, our configuration in app module, as an extension of step 4 is done last. We add our key from for appState (counter\_) and tag it with our reducer function (counterReducer). Import common module so we can use asyncPipe to unsubscribe to the observable result of the counter.
4. we can now use it in our component and template.

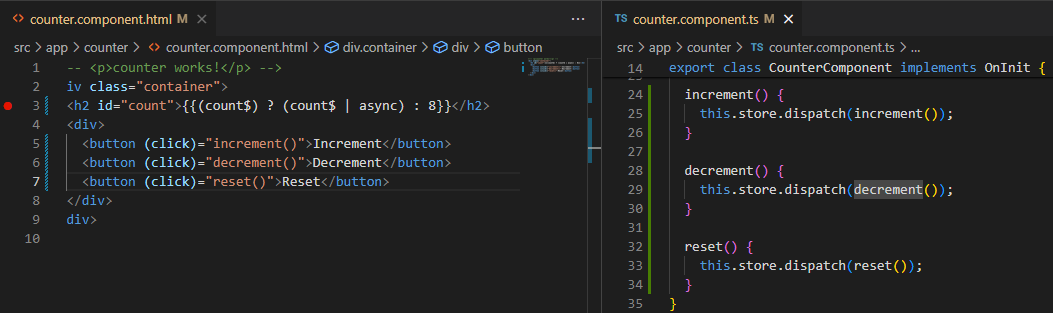
In our component’s constructor we get our defined selector passed as an argument to the select method of our store instance and store the value in a variable for our template.

Success.

A screenshot of a computer

Description automatically generated

## Click Functionality

Now to illustrate we are going to get our interpolation from the service rendered in our app component as it is the only other component we’ve got.

# A screen shot of a computer code Description automatically generatedAccess State in Other Component

We came up with our own constructor and did the needful we see our rendering successful in the snippet captured.