# Instructions – Exercise 4.3 – Handling Form Events with Observables

composer-app, part 4

**Instructions**

* Make a copy of the di-composer-application from Exercise 4.2 and add it to your week-4 directory
* Rename the application to reactive-composer-app
* Delete the node\_modules directory
* Delete the package-lock.json file
* Open the angular.json file and find and replace all “di-composer-app” entries with “reactive-composer-app”
* Open the package.json file and change the name to “reactive-composer-app”
* Run npm install and ng serve
  + You are doing this to test the application and confirm there are no errors
* app.component.html
  + Change the exercises name to Exercise 4.3 - Handling Events with Observables
* app.module.ts
  + Add an import statement for Angular’s built-in FormsModule and ReactiveFormsModule
    - import { FormsModule, ReactiveFormsModule } from ‘@angular/forms’;
  + Add the FormsModule and ReactiveFormsModule to the imports array
* composer-list.component.html
  + Add a Bootstrap input field above the table and float it to the right.
    - The input field should be above the table and to the right, but it must be align with the table
    - Give the input field a width of 30% and margin-bottom of 5px
* composer-list.component.html
  + Add an import statement for FormControl
    - This is component is part of Angulars built-in FormsModule
    - Import { FormControl } from ‘@angular/forms’;
  + Add a variable named txtSearchControl and assign it a new instance of the FormControl object
    - txtSearchControl = new FormControl(‘’);
  + Add an import statement for rxjs debounceTime component
    - import { debounceTime } from ‘rxjs/operators’;
  + Add a new function and name it filterComposers(name: string) and in the body of the function call the alert() function and pass-in the name parameter
    - alert(name)
  + In the components constructor and underneath the getComposers() call add a subscribe() method that listens for valueChanges and calls the filterComposers() function.  Also, make sure you add a debounceTime of 500.
    - this.txtSearchControl.valueChanges.pipe(debounceTime(500)).subscribe(val => this.filterComposer(val));
    - Note: the debounceTime function is used to “slow down” the number of times the filterComposer function is called.  If we do not add this, each time a value is entered into the txtSearchControl the filterComposers() function would be called.  Obviously, if this were a real API we would not want to hit the server after each word is entered into the field.  Instead, we would want to wait nth number of milliseconds before calling the filterComposers() function.  The debounceTime function serves this purpose.  Setting it to 500 is effectively saying we will wait 500ms before the first call to our filterComposers() function and each time the user stops typing we will wait 500ms before a subsequent call.
* composer-list.component.html
  + Add the FormControl directive to the txtSearch input field and pass-in our txtSearchControl variable
    - [formContorl]=”txtSearchControl”
* Run and test the application by entering values into the txtSearchControl.  Notice how there is a delay from when you enter the values and when the alert box is triggered.  Also, notice that if you begin typing a name and stop the alert box is triggered after a 500ms delay.  This is the whole idea behind rxjs debounceTime function.  Delay the execution of a function call.

Note: the goal of this exercise is to demonstrate Observables and how they are able to “listen” for changes in the state of a field.  The Observer pattern, as we learned in WEB 330, is a class that “observes” and “reacts” to changes in an object.  In our case, we are using an Observable, which is the FormControl variable we created and are “listening” for changes to the input field.  Each time the user enters or removes text from the input field, our Observable is reacting to those changes and calling the filterComposer() function.