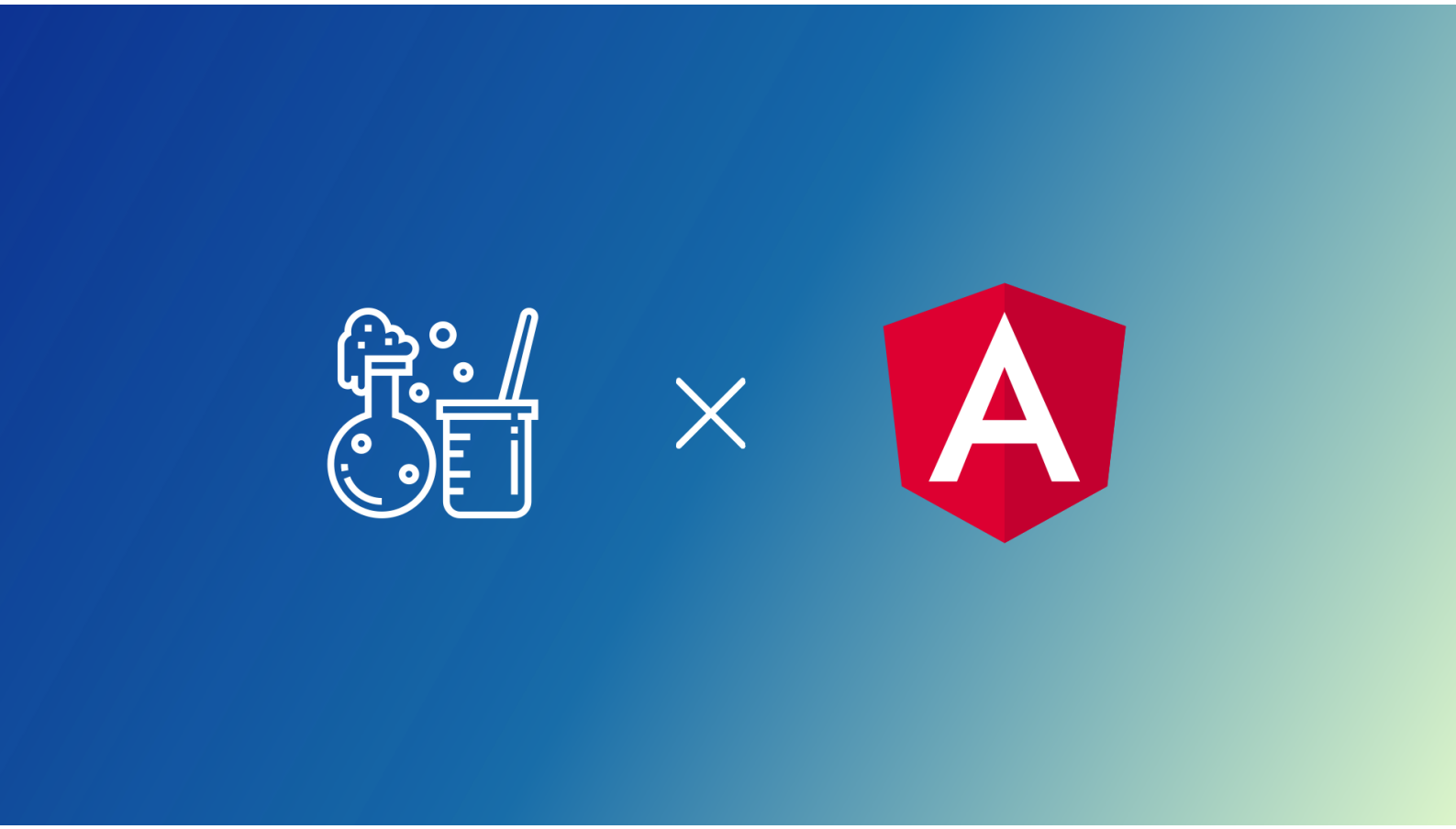
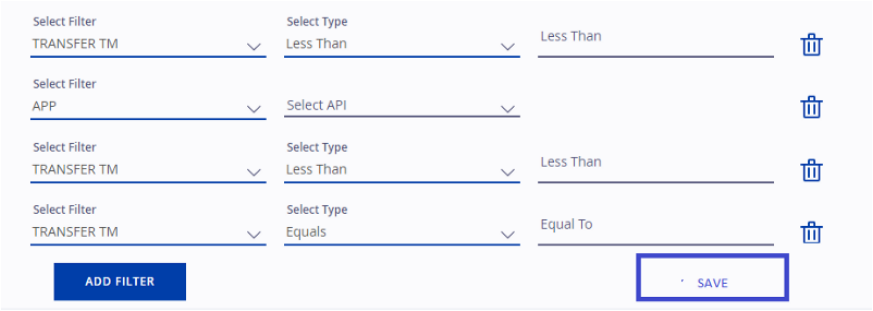


Unleashing the Power of forms with Angular's Reactive Forms | Icon Courtesy: chemical by Eucalyp, and Angular by Simon04

Unleash the power 💪of Forms with Angular's Reactive Forms

Consider a use case where you need to build a form that is dynamic enough to add sections. In these sections, the user could also add controls dynamically. To help you understand what I'm talking about, here's an image of a sample:

Siddharth Ajmera Follow
Feb 6 · 5 min read ★



A sample form that we're going to create

🐼TL;DR:

1. We'll create a Reactive Form to add/remove Filters.
2. Each filter will have two or three fields depending on the value of the `type`

. . .

The inspiration to write this article came from a Stack Overflow *question*, the title of which read:

> *"Dynamically adding deleting and getting data of select box and text box based on select box selected"*

I did eventually answer the question. But then I thought why not write a detailed medium article about it. So here I am, doing just that.

. . .

✓Prerequisites:

1. Basics of Angular. If you're new to Angular, I have a free YouTube Playlist on Angular.
2. Familiarity with Reactive Forms in Angular. If you're not comfortable with Reactive Forms, I'd highly recommend this video.
3. This article will not go into details on how to use Angular Material. So if you're new to Angular Material, going through this getting started guide would help.

. . .

Let's get down to business now. Just by looking at the form, I can understand that:

1. I'll have to create a `FormGroup`, with a filters `FormArray`.
2. Each filter in the `FormArray` would be another `FormGroup`.
3. This `filter` `FormGroup` would initially have two `FormControls` one for `filterType` and the other for `apiType`.
4. I'll add a new `FormControl` to the filter `FormGroup` if the user selects a value for `apiType`.
5. I can use `.push` method on the `filters` `FormArray` to push a new `FormGroup` for a filter.
6. I can use the `.removeAt` method on the `filters` `FormArray` to remove a specific `FormGroup` at a particular `index`.
7. I can also use the `.addControl` method on a `FormGroup` in order to add a `FormControl` to a `FormGroup`.

And that pretty much sums up what we'll be doing in order to create the form.

. . .

Enough talk. Show me the code:

I'll start with the Component Class first. That's where one in a sane mind would start while working on Reactive Forms.

1. Setup 🚀:

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, FormArray } from '@an

@Component({...})
export class AppComponent  {

  filterTypes = [ 'TRANSFER TM', 'APP', ... ];
  apiTypes = [ 'Less Than', 'Equals', 'Greater Than',
  dynamicForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.dynamicForm = this.fb.group({
      filters: this.fb.array([])
    });
```

Setting up the Reactive Form by declaring and initializing the properties and the form itself.

Nothing fancy here, I just added the imports. `filterTypes` and `apiTypes` would be used to populate the select options. I also injected the `FormBuilder` as a dependency as that's what I'll use to quickly bootstrap the form.

2. Add + / Remove − FormGroup(s) to the filters FormArray :

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, FormArray } from '@an

@Component({...})
export class AppComponent  {

  ...

  get filtersFormArray() {
    return (<FormArray>this.dynamicForm.get('filters')
  }

  createFilterGroup() {
    return this.fb.group({
      filterType: [],
      apiType: []
    });
  }

  addFilterToFiltersFormArray() {
```

Adding/Removing a Filter FormGroup to the filters FormArray.

I've created a `get`ter for `filters` `FormArray` named **`filtersFormArray`**. This is something that I'll be using quite frequently in this as well as in further parts of the code. The **`addFilterToFiltersFormArray`** method will be called on the Add Filter button click. This `pushes` new `filter` `FormGroup` to the `filters` `FormArray` which is returned by the **`createFilterGroup`** method. The **`removeFilterFromFiltersFormArray`** method will do the exact opposite of the `addFilterToFiltersFormArray` method. It accepts an index and will remove the element at that index from the `filters` `FormArray`. The important point to note here is, we're adding the **`push`** method to add a `FormGroup` and **`removeAt`** method to remove a `FormGroup` from the filters `FormArray`.

3. Dynamically adding the `value` `FormControl` to a `filter` `FormGroup` ⛈:

The addition of the `value` `FormControl` would depend on the value of the `apiType` `FormControl`. Once its value is set, we can then add the value `FormControl` to the filter `FormGroup`. We can do this by leveraging the **`addControl`** method on the `filter` `FormGroup`.

```
...

@Component({...})
export class AppComponent  {

  ...

  getFilterGroupAtIndex(index) {
    return (<FormGroup>this.filtersFormArray.at(index)
  }

  getFormControl() {
    return this.fb.control(null);
  }

  selectedAPIChanged(i) {
```

Adding a value FormControl to a filter FormGroup on selection change.

To intercept the change in the `apiType` select list, we're binding to the **`selectionChange`** event on the `mat-select` . Something like this: `(selectionChange)="selectedAPIChanged(i)"`

The **`selectedAPIChanged`** method accepts an index. It looks for the filter `FormGroup` at that index by calling the **`getFilterGroupAtIndex`** method, and then calls the **`addControl`** method on it. It accepts the first argument as the name of the `FormControl` to be created and the second argument as the `FormControl` instance which is returned by the `getFormControl` method.

There's also our humble **`save`** method which does nothing apart from logging the value of the form to the console.

4. Binding the FormGroup to the Template 🤗:

```
<form [formGroup]="dynamicForm">
  <div formArrayName="filters">
    <div
      *ngFor="let filter of filtersFormArray.controls;

      <div
        [formGroupName]="i"
        class="container">

        <mat-form-field>
          <mat-select
            placeholder="Select Filter"
            formControlName="filterType">
            <mat-option
              *ngFor="let filterType of filterTypes"
              [value]="filterType">
              {{ filterType }}
            </mat-option>
          </mat-select>
        </mat-form-field>

        <mat-form-field>
          <mat-select
            (selectionChange)="selectedAPIChanged(i)"
            placeholder="Select API"
            #apiField
            formControlName="apiType">
            <mat-option
              *ngFor="let apiType of apiTypes"
              [value]="apiType">
              {{ apiType }}
            </mat-option>
          </mat-select>
        </mat-form-field>

        <mat-form-field
          *ngIf="filter.get('value')">
          <input
            matInput
            formControlName="value"
            [placeholder]="apiField.value">
        </mat-form-field>

        <mat-icon
```

```
45                (click)="removeFilterFromFiltersFormArray(i)
```

Binding the FormGroup to the Template

I think this is pretty straight-forward. We've just used directives like `[formGroup]` , `formArrayName` , `[formGroupName]` , and `formControlName` to bind different parts of our FormGroup to the template. One important thing to note down is on Line 37 where I have used `*ngIf="filter.get('value')"` . This is done to check if the value `FormControl` exists on the `filter` `FormGroup` , and only render this `matInput` if it does. Leave a comment down below and I'll be happy to answer any questions.

5. Bringing it all together 👫:

Doing all this results in a Form that looks something like this:

Demo of the Form that We've just created.

. . .

6. Wait! What if I had some seed data? 😬

No worries. Angular's got you covered. I can use some JavaScript Array magic to be done with it in no time. Let's see how:

```
...

@Component({...})
export class AppComponent  {

  ...

  seedData = [
    { filterType: 'TRANSFER TM', apiType: 'Less Than',
    { filterType: 'TRANSFER TM' },
    { filterType: 'TRANSFER TM', apiType: 'Equals', va
    { filterType: 'TRANSFER TM', apiType: 'Equals' },
    { filterType: 'TRANSFER TM', apiType: 'Greater Tha
    { filterType: 'APP', apiType: 'Less Than', value:
    { filterType: 'APP', apiType: 'Equals', value: '50
    { filterType: 'APP' },
    { filterType: 'APP', apiType: 'Greater Than' },
  ];

  ...

  ngOnInit() {

    ...

    // Uncomment the line below If you want to seed th
    this.seedFiltersFormArray();
  }
```

Populating the form with some seed data

Along with some `seedData`, we've added a function named `seedFiltersFormArray()`. This function does 4️⃣ things:

1. Iterates over the `seedData`, and create a filter `FormGroup` for each datum.
2. Checks if the `seedDatum` has `apiType` and adds control for `value` based on that.
3. Populates this `FormGroup` with the `seedDatum` by calling the `patchValue` method on it.
4. Pushes this `FormGroup` to the `filters` `FormArray`.

And there you have your Dynamic Form prepopulated with some seed data. 🔥

. . .

Pretty neat 😍 , right? Now that you have seen how you can dynamically add/remove groups and controls from FormArrays and FormGroups, I hope you can literally achieve anything related to Forms using Angular.

. . .

Where do I get the code? 🤔

I'm attaching the StackBlitz I created for this article. Fork it, edit it, create it, share it. Here it is:

dynamic-add-remove-and-change-filter-form-in-angular-material
By SiddAjmera

Run Project

StackBlitz Sample for the App we just built.

Closing notes 🎉🎆🎇:

On that note, I'd like to conclude this article.

Special thanks to Sander Elias for the suggestion on adding seed data as a feature. And to rajat badjatya and Murli Prajapati for proof-reading this article. 🙌🏻

I hope this article has taught you something new related to Angular and Reactive Forms. Share this article with your friends who are new to Angular and want to achieve something similar. Also, comment below on what you'd like to read about next. **Until next time then.** 👋🏻