



# Angular Advanced - Dynamic Forms - FormArray



Peter Kassenaar –  
[info@kassenaar.com](mailto:info@kassenaar.com)

## Goal in this module

*Learn how to build **dynamic Angular Forms**  
with `FormArray`  
by **adding or removing form controls at  
runtime.***

*Project: build an in-place editable data table*

# What are we going to build?

A 'Playlist editor' with which you can create a (dummy) Playlist and add or remove songs at runtime.

### Playlist Component

Title \*

All Time Classics

Date \*

10-10-2022

Description \*

A small list containing all time classic songs

Artist	Title	Duration	
Madonna	Into The Groove	03:40	
Beatles	I wanna hold your ha	02:55	
AC/DC	Highway to Hell	03:27	
artist	title	00:00	

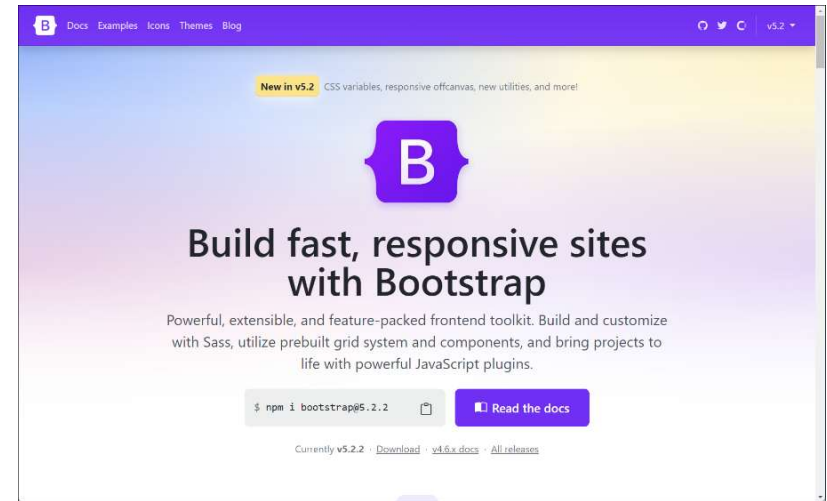
+ Add Song

Save Playlist

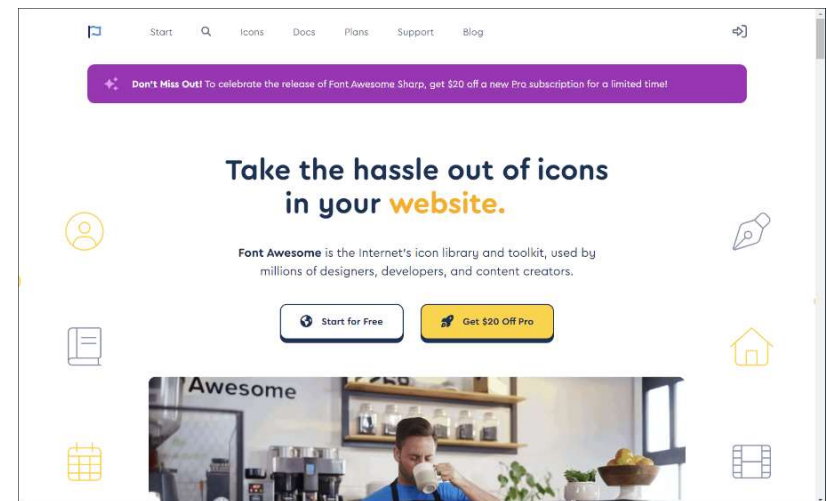
../examples/785-dynamic-form-array

# We are using:

- Bootstrap for UI classes
  - <https://getbootstrap.com/>
- FontAwesome for icons
  - <https://fontawesome.com/>
- Add them to your `styles.css`



```
npm install bootstrap@5.2.2
```



```
npm install @fortawesome/fontawesome-free
```

```
/* You can add global styles to this file, and also import other style files */  
@import "../node_modules/bootstrap/dist/css/bootstrap.min.css";  
@import "../node_modules/@fortawesome/fontawesome-free/css/all.min.css";
```

# Contents

- What is a `FormArray`?
- Difference between `FormArray` and `FormGroup`
- The `FormArray` API
- Using `FormArray` in real code
- Using the `formArrayName` directive
- Workshop

# What is a FormArray?

- FormArray is a part of **Reactive Forms**
  - So, **not available** in Template Driven Forms
  - `import ReactiveFormsModule from "@angular/forms"`
- FormArray is – as its name implies – a **dynamic array** of FormControls
- By default, a Reactive Form is **static**, using the FormGroup feature where all controls are **known upfront**.
  - For instance:

```
this.form = this.formBuilder.group({
  email    : ``,
  password: ``,
  customer: this.formBuilder.group({
    prefix   : ``,
    firstName: ``,
    lastName : ``
  })
});
```

# What is a FormArray?

- In a `FormArray` we **do not know** all form fields or rows at design time
- The form is a **result of a user, interacting with the form.**
- Examples include:

- Order forms with variable number of items to order
- Tax return forms
- Travel and Expenses forms
- Playlists ;-)
- And many, many more.



The screenshot shows a Dutch tax form (3a) titled 'Intracommunautaire leveringen en diensten'. It features a table with columns for 'Regel' (Rule), 'Landcode' (Country code), 'Nummer' (Number), 'Leveringen' (Deliveries), and 'Diensten' (Services). The table has 5 rows, each with a 'Regel' number and a 'Landcode' dropdown. The 'Leveringen' and 'Diensten' columns contain input fields for amounts in Euros (€). A link '5 regels toevoegen' (Add 5 rules) is visible at the bottom of the table. The form also includes a sidebar with options like 'Ondertekenen en verzenden' (Sign and send), 'Opslaan en afdrukken' (Save and print), 'Help', 'Afdrukken' (Print), and 'Afsluiten' (Finish).


- With a `FormGroup` this would be impossible –  
since we *don't know* the number of items upfront
  - The end user can dynamically add or delete rows at runtime

# Inside a FormArray

A Reactive form can combine:

- **static fields** (like: id, title, date and so on) of type `FormControl`
- **dynamic fields**, of type `FormArray`

```
export class PlaylistComponent {  
  
  ...  
  constructor(private fb: FormBuilder) {  
    }  
  
  // The reactive form  
  playlist = this.fb.group({  
    title: ['', Validators.required],  
    date: [new Date(), Validators.required],  
    description: [''],  
    songs: this.fb.array([])  
  })  
  ...  
}
```





## Inside a FormArray

*A FormArray can have an **undetermined number of form controls**, starting at zero! The controls can then be dynamically added and removed depending on **how the user interacts** with the UI.*

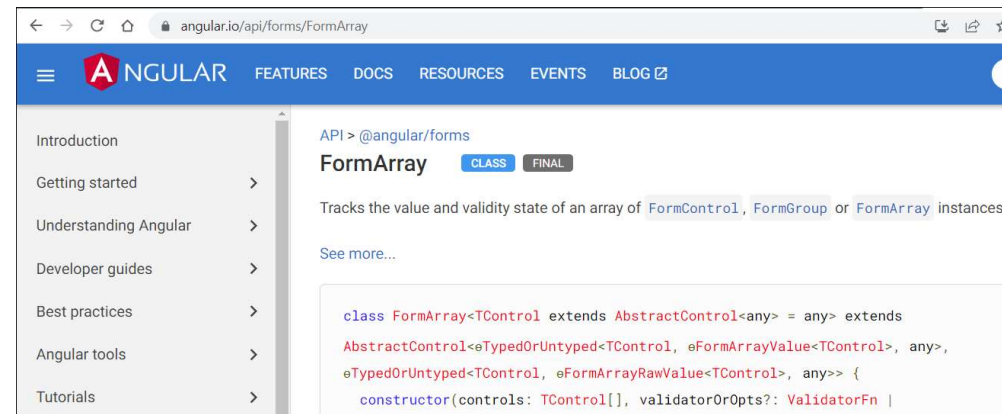
## Inside a FormArray

*Each control will then have a **numeric position** in the form controls array, instead of a unique name.*

*Form controls can be added or removed from the form model anytime at runtime using the **FormArray API**.*

# The FormArray API

- We can control the `FormArray` via its API.
  - Official documentation: <https://angular.io/api/forms/FormArray>
- Most used methods:
  - `controls`: an array containing **all the controls** that are part of the array
  - `length`: the **total length** of the array
  - `at(index)`: returns the **form control at a given array position**
  - `push(control)`: **adds a new control** to the end of the array
  - `removeAt(index)`: **removes a control** at a given position of the array
  - `getRawValue()`: Gets the values of **all form controls**, via the `control.value` property of each control
- We are using this API in our Playlist-example





# Using FormArray

Creating an in-place editable PlaylistComponent

# #0 – Creating the model

- We created `interfaces` for playlists and songs
  - Not mandatory, but convenient and gives you intellisense/type safety

```
export interface Playlist {  
  title: string;  
  date: Date;  
  songs: Song[];  
  description?: string;  
}  
  
export interface Song {  
  artist: Artist;  
  title: string;  
  duration: string;  
}  
  
export interface Artist {  
  name: string;  
}
```

# #1. Creating the reactive form model

```
export class PlaylistComponent {  
  ...  
  
  constructor(private fb: FormBuilder) {  
  }  
  
  // The reactive form model  
  playlist = this.fb.group({  
    title: ['', Validators.required],  
    date: [new Date(), Validators.required],  
    description: [''],  
    songs: this.fb.array([])  
  })  
  
  // A getter, so we get easily type-safe access  
  // to the FormArray for adding and removing entries.  
  get songs(): FormArray {  
    return this.playlist.controls['songs'] as FormArray  
  }  
  ...  
}
```

Our playlist has a title, a date, a description and an array of songs.

Initially the array is empty, meaning our playlist doesn't contain any songs

## #2. Dynamically adding controls

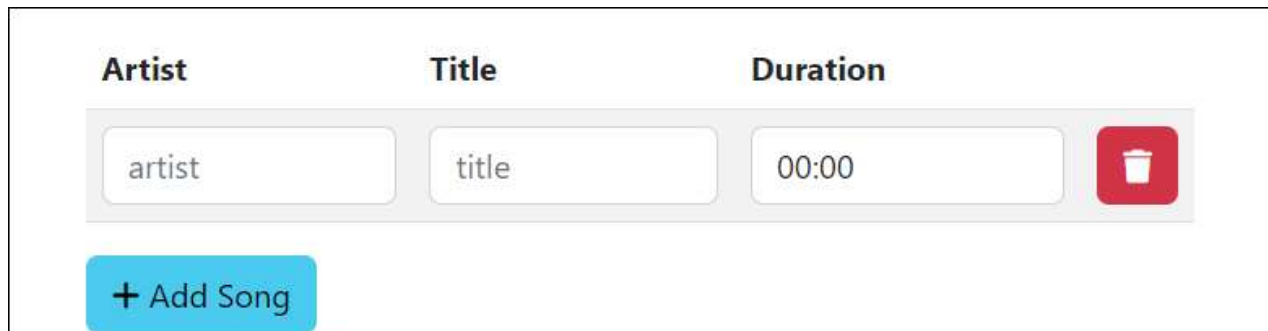
- In the component we have a button
  - UI classes are from Bootstrap and do not add functionality
- The button triggers the following code:


```
<button class="btn btn-info" (click)="addSong()">  
  <i class="fa fa-add"></i>  
  Add Song  
</button>
```

```
// Adding a new song by pushing a new song item to the formArray  
addSong() {  
  const newSong = this.fb.group({  
    artist: ['', Validators.required],  
    title: ['', Validators.required],  
    duration: ['00:00', Validators.required]  
  })  
  this.songs.push(newSong);  
}
```

# What's going on here?

- In order for our form validation to work, we add a control which is a `fb.group()` in itself!
  - Remember, `FormGroups` are also `FormControls`, so they are perfectly valid inside the `FormArray`.
- Every `FormGroup` has an `artist`, a `title` and a `duration`.
- We will build the User Interface in a minute, but it will look like this:



Artist	Title	Duration	
<input type="text" value="artist"/>	<input type="text" value="title"/>	<input type="text" value="00:00"/>	



## #3 Deleting a song

Every song has an associated `Delete` button:

```
<button class="btn btn-danger"
        title="Delete this song"
        (click)="deleteSong(i)">
  <i class="fa fa-trash"></i>
</button>
```

```
// delete a song
deleteSong(index: number) {
  this.songs.removeAt(index);
}
```

## #4 – Template – the static fields

```
<h4>Playlist Component</h4>
<form [formGroup]="playlist">
  <input type="text" class="form-control" placeholder="Playlist title"
    FormControlName="title">
  <input class="form-control" type="date" placeholder="date"
    FormControlName="date">
  <input class="form-control" type="text" placeholder="Description"
    FormControlName="description">
  ...
```

### Playlist Component

Title \*

All Time Classics

Date \*

10-10-2022



Description \*

A small list containing all time classic songs

## #4 Template – dynamic fields

```
<!-- Dynamic fields in the form, in an array-->
<ng-container formArrayName="songs">
  <hr>
  <table class="table table-striped">
    <thead>
      <tr><th>Artist</th><th>Title</th><th>Duration</th><th></th></tr>
    </thead>
    <tbody>
      <ng-container *ngFor="let songForm of songs.controls; let i = index">
        <tr [formGroupName]="i">
          <td>
            <input type="text" class="form-control" formControlName="artist" placeholder="artist">
          </td>
          <td>
            <input type="text" class="form-control" formControlName="title" placeholder="title">
          </td>
          <td>
            <input type="text" class="form-control" formControlName="duration" placeholder="duration">
          </td>
          <td>
            <button class="btn btn-danger" (click)="deleteSong(i)">
              <i class="fa fa-trash"></i>
            </button>
          </td>
        </tr>
      </ng-container>
    </tbody>
  </table>
</ng-container>
</form>
<button class="btn btn-info" (click)="addSong()">
  <i class="fa fa-add"></i>
  Add Song
</button>
```

playlist.component.html

# This is what's going on:

- We add the `[formGroup]` directive to a `<form>` element and assign it the `playlist` variable
- Then, we first add/show the static fields `title`, `date` and `description`
- We then use `<ng-container>` to render the `FormArray` `songs`
  - But only if there *are* any songs
- Inside that container we create *another* `<ng-container>` to render the individual song fields `artist`, `title` and `duration`.
  - We use `*ngFor` to loop over the collection of controls.
  - The directive `[formGroupName]="i"` is used to assign a unique value to each `FormGroup` created this way.

## Cont'd

- We use the `formControlName` directive to bind each individual form field to a field in the form model
- So, the editable table form is simply a form with a **list of nested child forms**, one form per table row.
  - Each table row form contains three controls inside it.
- The user can freely **add and remove** song rows to the form.
- The parent form will only be considered valid once every row added by the user is filled in with valid values.
  - See the `Validators` values in the class/form model.

```
formControlName="title"
```

## #5 Posting the form

- In Real Life: post form to backend, using `this.http.post()` or similar
- For now: displaying the values in the UI
  - We use `.getRawValue()` to get the (nested) values of controls in the form.
  - If using simply `.value` we would get `[Object object]` or a circular json-dependency.
- For instance:



```
<button class="btn btn-success"
        [disabled]="!playlist.valid"
        (click)="save()">
  <i class="fa fa-floppy-disk"></i>
  Save Playlist
</button>
```

```
// For displaying in the UI
playlistRaw?: Playlist;
save() {
  this.playlistRaw = this.playlist.getRawValue()
  console.log('TODO: Post to database: ',
              this.playlist.getRawValue());
}
```

```

<div *ngIf="playlistRaw">
  <p>TODO: Post to database, playlist:</p>
  <pre>
    {{playlistRaw | json }}
  </pre>
</div>

```

Beatles	Blackbird	02:19	
AC/DC	Highway to Hell	3:27	

+ Add Song

 Save Playlist

TODO: Post to database, playlist:

```

{
  "title": "All Time Classics",
  "date": "2022-10-10",
  "description": "A small list containing all time classic songs",
  "songs": [
    {
      "artist": "Madonna",
      "title": "Into the Groove",
      "duration": "04:44"
    },
    {
      "artist": "Beatles",
      "title": "Blackbird",
      "duration": "02:19"
    },
    {
      "artist": "AC/DC",
      "title": "Highway to Hell",
      "duration": "3:27"
    }
  ]
}

```





# Workshop

## 1. **Work from the example code** `../785-dynamic-form-array`

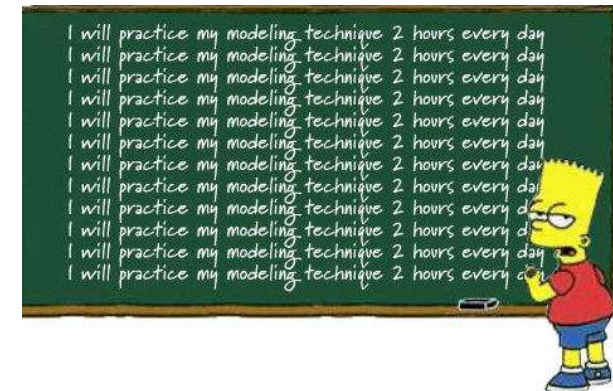
- Study the data flow and inner workings of this form example
- Add a new property `year` to the `song` instance
- Make sure users can enter a `year` for every song in the playlist.

## 2. **OR: Create a new dynamic `FormArray` application yourself, for instance:**

- A shopping/groceries list
- A Travel & Expenses form
- A form containing employees from your company
- Etc...

## 3. **Optional:** expand the working example so you can save a Playlist to `localStorage`.

- If the form is loaded, the app looks in `localStorage` if there is already data available. If yes – it displays it.



# More info



<https://blog.angular-university.io/angular-form-array/>

# <https://angular.io/api/forms/FormArray>

The screenshot shows the Angular API documentation for `FormArray`. The top navigation bar includes the Angular logo and links to FEATURES, DOCS, RESOURCES, EVENTS, and BLOG. A search bar on the right contains the text "formarray". The left sidebar lists various documentation sections, with "Reference" expanded to show "API reference". The main content area displays the `FormArray` class, marked as "CLASS" and "FINAL". It provides a description: "Tracks the value and validity state of an array of `FormControl`, `FormGroup` or `FormArray` instances." Below this, the full TypeScript class definition is shown, including its inheritance from `AbstractControl` and its methods: `at()`, `push()`, `insert()`, `removeAt()`, `setControl()`, `setValue()`, `patchValue()`, `reset()`, `getRawValue()`, `clear()`, and `Usage notes`. The right sidebar lists the methods and their descriptions.

API > @angular/forms

## FormArray

CLASS FINAL

Tracks the value and validity state of an array of `FormControl`, `FormGroup` or `FormArray` instances.

See more...

```
class FormArray<TControl> extends AbstractControl<any> = any> extends
  AbstractControl<eTypedOrUntyped<TControl, eFormArrayValue<TControl>, any>,
  eTypedOrUntyped<TControl, eFormArrayRawValue<TControl>, any>> {
  constructor(controls: TControl[], validatorOrOpts?: ValidatorFn |
    AbstractControlOptions | ValidatorFn[], asyncValidator?: AsyncValidatorFn |
    AsyncValidatorFn[])
    controls: eTypedOrUntyped<TControl, Array<TControl>, Array<AbstractControl<any>>>
    length: number
    at(index: number): eTypedOrUntyped<TControl, TControl, AbstractControl<any>>
    push(control: TControl, options: { emitEvent?: boolean; } = {}): void
    insert(index: number, control: TControl, options: { emitEvent?: boolean; } = {}):
    void
    removeAt(index: number, options: { emitEvent?: boolean; } = {}): void
    setControl(index: number, control: TControl, options: { emitEvent?: boolean; } =
    {}): void
    setValue(value: eIsAny<TControl, any[], eRawValue<TControl>[]>, options: {
    onlySelf?: boolean; emitEvent?: boolean; } = {}): void
    patchValue(value: eIsAny<TControl, any[], eValue<TControl>[]>, options: { onlySelf?:
    boolean; emitEvent?: boolean; } = {}): void
    reset(value: eIsAny<TControl, any, eIsAny<TControl, any[], eValue<TControl>[]>> =
```

FormArray

- Description
- Constructor
- Properties
- Methods
  - `at()`
  - `push()`
  - `insert()`
  - `removeAt()`
  - `setControl()`
  - `setValue()`
  - `patchValue()`
  - `reset()`
  - `getRawValue()`
  - `clear()`
- Usage notes
  - Create an array of form controls
  - Create a form array with array-level validators

<https://www.youtube.com/watch?v=By0TTKuxWI>

The screenshot displays a YouTube video player interface. The video title is "What is Form Array | Reactive Forms | Angular 13+", and the channel is "Reactive Forms - The Basics". The video has 5,950 views and 110 likes. The video player shows a VS Code editor window with the following code in `app.component.ts`:

```
src > app > TS app.component.ts > AppComponent > ngOnInit > skills
1 import { Component, OnInit } from '@angular/core';
2 import { FormArray, FormControl, FormGroup, Validators } from '@angular/forms';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css']
8 })
9 export class AppComponent implements OnInit {
10   title = 'ReactiveForms';
11   reactiveForm = new FormArray<AbstractControl>([
12     new FormControl<string>(''),
13     new FormControl<string>(''),
14     new FormControl<string>('')
15   ]);
16   ngOnInit() {
17     this.reactiveForm = new FormArray([
18       new FormControl<string>(''),
19       new FormControl<string>(''),
20       new FormControl<string>('')
21     ]);
22   }
23   onSubmit() {
24     // ...
25   }
26 }
```

The video player controls show the video is at 5:43 / 14:48. The video description includes "Complete Angular 13 Course Step by Step".