Angular formarray

To build an **Angular Reactive Form** we use three fundamental building blocks

- FormControl
- FormGroup
- FormArray

A FormArray as the name implies is an array. It can contain an array of

- FormControls
- FormGroups
- Nested FormArrays

We usually use an array to hold like items, but a FormArray can contain unlike items as well, i.e a few elements in a given array can be FormControls, a few of them in that same array can be FormGroups and the rest of them can be FormArrays.

In the example below, we have a FormArray with

- one FormControl
- one FormGroup
- one FormArray

We usually use the following properties to determine the state of a FormControl or a FormGroup. These status properties are also available on a FormArray. For example, if one of the controls in a FormArray is touched, the entire array becomes touched. Similarly, if one of the controls is invalid, the entire array becomes invalid.

- touched
- untouched
- dirty
- pristine
- valid

- invalid

Useful FormArray methods

Method	Purpose
push	Inserts the control at the end of the array
insert	Inserts the control at the specified index in the array
removeAt	Removes the control at the specified index in the array
setControl	Replace an existing control at the specified index in the array
at	Return the control at the specified index in the array

Creating a FormArray of FormGroup objects

This is preparation for dynamically creating FormGroups at runtime. Every time we click, "Add Skill" button on the "Employee Form" below, we want to dynamically generate a new set of skill related form fields. So in this video, we will do all the preparation required for that.

Angular dynamic forms

We will discuss **generating FormGroups and FormControls dynamically at runtime**.
Step 1 : Include Add Skill button

Place the following HTML inside the element with class well. Notice the click event is bound to addSkillButtonClick() method. We will create this method in the component class next.

Step 2 Include addSkillButtonClick() method in the component class

- From the root FormGroup "employeeForm" get a reference to the skills FormArray. Notice we have passed the name of the FormArray (skills) as a parameter to the get() method.

- The get() method returns the FormArray as an AbstractControl. We know it's a FormArray so we are type casting it to FormArray.
- We are then calling the push() method of the FormArray to push a new FormGroup into the FormArray
- The push() method calls addSkillFormGroup() method which returns an instance of the FormGroup with the 3 skill related form controls (skill, experience & proficiency)

Angular reactive forms edit example

We will discuss **Implementing EDIT operation in a reactive form**. We will use **Create Employee Form** for both creating a new employee as well as editing an existing employee details.

Changes in app-routing.module.ts

- Include a new route for editing an existing employee details in **app-routing.module.ts**.
- We will use **"create"** route to create a new employee.
- The new **"edit"** route will be for editing an existing employee details.
- Notice to the **"edit"** route we are passing the **id** of the employee we want to edit.

Changes in list-employees.component.html : Include click event binding on the **Edit** button.

```
<td>
  <button class="btn btn-primary" (click)="editButtonClick(employee.id)">
    Edit
  </button>
</td>
```

Changes in list-employees.component.ts : Import Angular Router

```
import { Router } from '@angular/router';
```

Inject it into the component class using the constructor

```
constructor(private _employeeService: EmployeeService,
            private _router: Router) { }
```

Include editButtonClick() event handler method in the component class. When the Edit button is clicked, the user will be redirected to the "edit" route, passing it the id of the employee we want to edit.

```
editButtonClick(employeeId: number) {
  this._router.navigate(['/edit', employeeId]);
}
```

We will discuss what **Angular modules are and why we need them in an Angular project. An Angular Module is a mechanism to group components, directives, pipes and services that are related to a feature area of an angular application.**

For example, if you are building an application to manage employees, you might have the following features in your application.

Application Feature	Description
Employee Feature	Deals with creating, reading, updating and deleting employees
Login Feature	Deals with login, logout, authenticate and authorize users
Report Feature	Deals with generating employee reports like total number of employees by department, top 10 best employees etc

To group the components, directives, pipes and services related to a specific feature area, we create a module for each feature area. These modules are called feature modules.

In addition to feature modules, an Angular application also contains the following modules.

Module Type	Description

Root Module	**Every Angular application has at least one module, the root module. By default, this root application module is called AppModule. We bootstrap this root module to launch the application. If the application that you are building is a simple application with a few components, then all you need is the root module. As the application starts to grow and become complex, in addition to the root module, we may add several feature modules. We then import these feature modules into the root module. We will discuss creating feature modules in our upcoming videos**
Core Module	**The most important use of this module is to include the providers of http services. Services in Angular are usually singletons. So to ensure that, only one instance of a given service is created across the entire application, we include all our singleton service providers in the core module. In most cases, a CoreModule is a pure services module with no declarations. The core module is then imported into the root module (AppModule) only. CoreModule should never be imported in any other module. We will discuss creating a core module in our upcoming videos**
Shared Module	**This module contains reusable components, directives, and pipes that we want to use across our application. The Shared module is then imported into specific Feature Modules as needed. The Shared module might also export the commonly used Angular modules like CommonModule, FormsModule etc. so they can be easily used across your application, without importing them in every Feature Module. We will discuss creating a shared module in our upcoming videos**
Routing Modules	**An angular application may also have one or more routing modules for application level routes and feature module routes**

What are the advantages of splitting an angular application into multiple Angular Modules

Well, there are several benefits of Angular Modules.

Benefit	**Description**
Organizing Angular Application	**First of all, Modules are a great way to organise an angular application. Every feature area is present in it's own feature module. All Shared pieces (like components, directives & pipes) are present in a Shared module. All Singleton services are present in a core module. As we clearly know what is present in each module, it's easier to understand, find and change code if**

	required
Code Reuse	Modules are great way to reuse code. For example, if you have components, directives or pipes that you want to reuse, you include them in a Shared module and import it into the module where you need them rather than duplicating code. Code duplication is just plain wrong, and results in unmaintainable and error prone code. We will discuss creating a Shared module and how it can help us reuse code in our upcoming videos
Code Maintenance	Since Angular Modules promote code reuse and separation of concerns, they are essential for writing maintainable code in angular projects
Performance	Another great reason to refactor your application into modules is performance. Angular modules can be loaded either eagerly when the application starts or lazily on demand when they are actually needed or in the background. Lazy loading angular modules can significantly boost the application start up time. We will discuss lazy loading modules in our upcoming videos

@NgModule Decorator

As we have already discussed an Angular module is a class that is decorated with @NgModule decorator. The @NgModule decorator has the following properties.

- **declarations**
- **bootstrap**
- **providers**
- **imports**
- **exports**

We will discuss these properties in detail when we discuss creating Feature, Shared and Core modules in our upcoming videos.

In preparation for refactoring our application into multiple modules, let's create the following 2 components

- **HomeComponent**
- **PageNotFoundComponent**

```
ng g m shared/shared --flat -m employee/employee
json-server --watch db.json
```