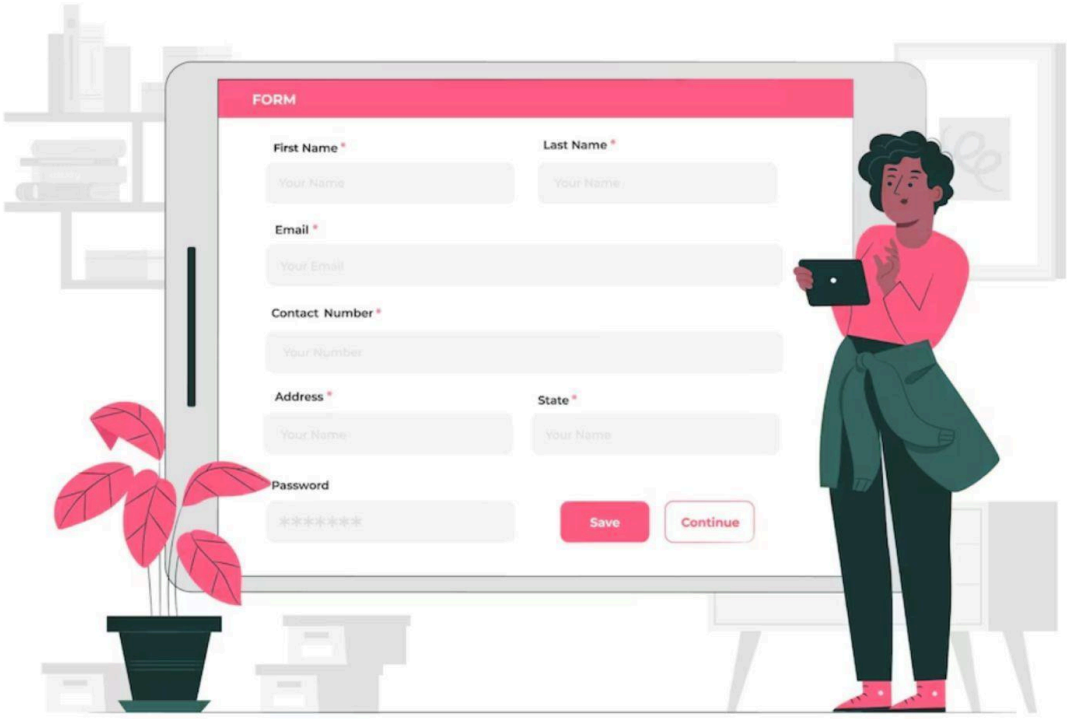
Angular Reactive Forms

Everything you need to know



Abdelfattah Ragab

Angular Reactive Forms

Everything you need to know

Abdelfattah Ragab

Introduction

Welcome to the book “Angular Reactive Forms”.
In this book, I explain everything you need to know about reactive forms.
With reactive forms, you define the form model directly in the component class. This is an elegant way to deal with form fields that gives you full control over all aspects of reading and updating values and performing all kinds of validations.
This way you can easily create and edit dynamic forms from your component class.
In this book, I will show you how to create forms, update values, perform validations, display error messages, work with hierarchical groups and form arrays, upload files, preview images, and much more.
By the end of this book, you will be confident working with forms in your Angular application and be able to handle all kinds of scenarios.
Let us get started.

Overview of reactive forms

Reactive forms provide a model-driven approach to handling form inputs whose values change over time. Reactive forms use an explicit and immutable approach to manage the state of a form at a given time. Each change to the form state results in a new state, preserving the integrity of the model between changes. Reactive forms are based on observable streams, where the inputs and values of the form are provided as streams of input values that can be accessed synchronously.

Import the `ReactiveFormsModule`

To use reactive form controls, import `ReactiveFormsModule` from the `@angular/forms` package and add it to your imports array.

Simple Form

Let's start with a simple login form where you ask for the user's email and password.
In the imports array of the standalone component, import the `ReactiveFormsModule` as follows:

```
import { Component } from
'@angular/core';
```

```
import { ReactiveFormsModule } from
'@angular/forms';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css',
})
export class LoginComponent {}
```

Now we declare a form with email and password fields:

```
import { Component, OnInit } from
'@angular/core';
import { FormBuilder, FormGroup,
ReactiveFormsModule } from
'@angular/forms';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css',
})
export class LoginComponent implements
OnInit {
  loginForm!: FormGroup;
```

```
  constructor(private readonly fb:
FormBuilder) {}

  ngOnInit(): void {
    this.loginForm = this.fb.group({
      email: this.fb.control(''),
      password: this.fb.control(''),
    });
  }
}
```

We use the FormBuilder service to create instances of our form such as groups, controls and so on. We inject it into the constructor as follows:

```
  constructor(private readonly fb:
FormBuilder) {}
```

Now let's create the form in html. Here it is:

```
<form (submit)="onSubmit()" class="form"
[formGroup]="loginForm">
  <div class="title">Login</div>
  <div class="container">
    <div class="group">
      <label for="email"
class="label"></label>
      <input
        formControlName="email"
        placeholder="Email"
        type="text"
        class="input"
      />
    </div>
    <div class="group">
      <label for=""
class="label"></label>
      <input
        formControlName="password"
        type="password"
        class="input"
        placeholder="Password"
      />
    </div>
    <div class="submit-group">
      <button class="login"
type="submit">Login</button>
    </div>
  </div>
</form>
```

To connect your form in the component class with the form in the HTML template, you must do the following:
Set the `formGroup` in the tag, as we have already done.
Next, connect each form field to its input field by setting the `formControlName` as we have already done.

Now you can read the data entered by the user at any time by accessing the value `this.loginForm.value`. It will return an object

```
{
    "email": "emilys",
    "password": "emilyspass"
}
```

I have attached a handler to the `submit` event of the form. We have set the type of the login button to `"submit"` so that every time it is clicked, the form is submitted and the `submit` event is triggered and the event handler which is the `onSubmit` method is executed.

Here is the component class now:

```
import { Component, OnInit } from
'@angular/core';
import { FormBuilder, FormGroup,
ReactiveFormsModule } from
'@angular/forms';

@Component({
```

```
  selector: 'app-login',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css',
})
export class LoginComponent implements
OnInit {
  loginForm!: FormGroup;

  constructor(private readonly fb:
FormBuilder) {}

  ngOnInit(): void {
    this.loginForm = this.fb.group({
      email: this.fb.control(''),
      password: this.fb.control(''),
    });
  }

  onSubmit() {
    console.log(this.loginForm.value);
  }
}
```

Validation

Now that both the email address and password are mandatory, you want to verify that the user has entered them before submitting the form. The second parameter

of the method fb.control is the array validators. So let's add some validators.

```
  ngOnInit(): void {
    this.loginForm = this.fb.group({
      email: this.fb.control('',
[Validators.required, Validators.email]),
      password: this.fb.control('',
[Validators.required]),
    });
  }
```

For the email field, we validate it is not empty and is a valid email address.
For the password, we validate it is not empty.
At any time, we can check if the form is valid by reading the value of `this.loginForm.invalid` or `this.loginForm.valid`, any of them.
For example I will disable the login button if the form is invalid.