Angular Form Typings Support that Every Dev Needs - JavaScript in Plain English

DevJo

Have you ever wondered why Angular reactive form does not have **IntelliSense** support? Here is how you can set them up for better form typings in your next Angular form.

The standard way of form setup in Angular 🌤

To begin with, here is how a typical reactive form is being set up in a project

```html
<form [formGroup]="form">
  <div class="form-control">
    <label for="username">Username</label>
    <input type="text" formControlName="username" />
  </div>
  <div class="form-control">
    <label for="email">Email</label>
    <input type="email" formControlName="email" />
  </div>
  <div class="form-control">
    <label for="password">Password</label>
    <input type="password" formControlName="passsword" />
  </div>
  <div class="form-control">
    <label for="firstname">First Name</label>
    <input type="text" formControlName="firstname" />
  </div>
  <div class="form-control">
    <label for="lastname">Last Name</label>
    <input type="text" formControlName="lastname" />
  </div>
</form>
```

form.html

```typescript
export class ReactiveFormComponent implements OnInit {
  form!: FormGroup;

  constructor() {
    this.form = this.initForm();
  }

  ngOnInit(): void {}

  private initForm(): FormGroup {
    return new FormGroup({
      username: new FormControl(null),
      email: new FormControl(null),
      password: new FormControl(null),
      firstname: new FormControl(null),
      lastname: new FormControl(null),
    });
  }
}
```

form.ts

Did you spot a typo that I have accidentally made while matching the formControlName to my form in the template? The `password` field is set up with an extra `s` in the formControlName 😢. Bugs like this can take us hours to spot especially if your form is huge 👹.

A better way of form setup in Angular 😎

To resolve this kind of template typo issue that we all face in time past, we can implement a pair of enum and const to standardise the naming to avoid typo error.

```
enum FormFieldName {
  firstName = 'firstName',
  lastName = 'lastName',
  email = 'email',
  password = 'password',
  username = 'username'
}

const FormFieldLabel = {
  [FormFieldName.firstName]: 'First Name',
  [FormFieldName.lastName]: 'Last Name',
  [FormFieldName.email]: 'Email',
  [FormFieldName.password]: 'Password',
  [FormFieldName.username]: 'Username'
}
```

Add this enum and const to your model file or in your TS file to standardise the form label and form field name.

```
Go to component
1  <form [formGroup]="form">
2    <div class="form-control">
3      <label for="username">{{formFieldLabel.}}</label>
4  (property) [FormFieldName.email]: string
5                                                  email
6    <div class="form-control">                    firstName
7      <label for="email">Email</label>            lastName
8      <input type="email" formControlName="em     password
9    </div>                                        username
```

Notice that immediately after you declare the model of your form, you can use it in your form label and enjoy the assistance of the **IntelliSense** from your IDE.

You can also enjoy **IntelliSense** support for passing your formControl to your `[formControlName]` binding in the template automatically using the enum that you have declared in your TS file.

```
<form [formGroup]="form" (submit)="handleSubmit()">
  <div class="form-control">
    <label for="username">{{ formFieldLabel.username }}</label>
    <input type="text" [formControlName]="formFieldName.username" />
  </div>
  <div class="form-control">
    <label for="email">{{ formFieldLabel.email }}</label>
    <input type="email" [formControlName]="formFieldName.email" />
  </div>
  <div class="form-control">
    <label for="password">{{ formFieldLabel.password }}</label>
    <input type="password" [formControlName]="formFieldName.password" />
  </div>
  <div class="form-control">
    <label for="firstname">{{ formFieldLabel.firstName }}</label>
    <input type="text" [formControlName]="formFieldName.firstName" />
  </div>
  <div class="form-control">
    <label for="lastname">{{ formFieldLabel.lastName }}</label>
    <input type="text" [formControlName]="formFieldName.lastName" />
  </div>

  <button type="submit">Submit</button>
</form>
```

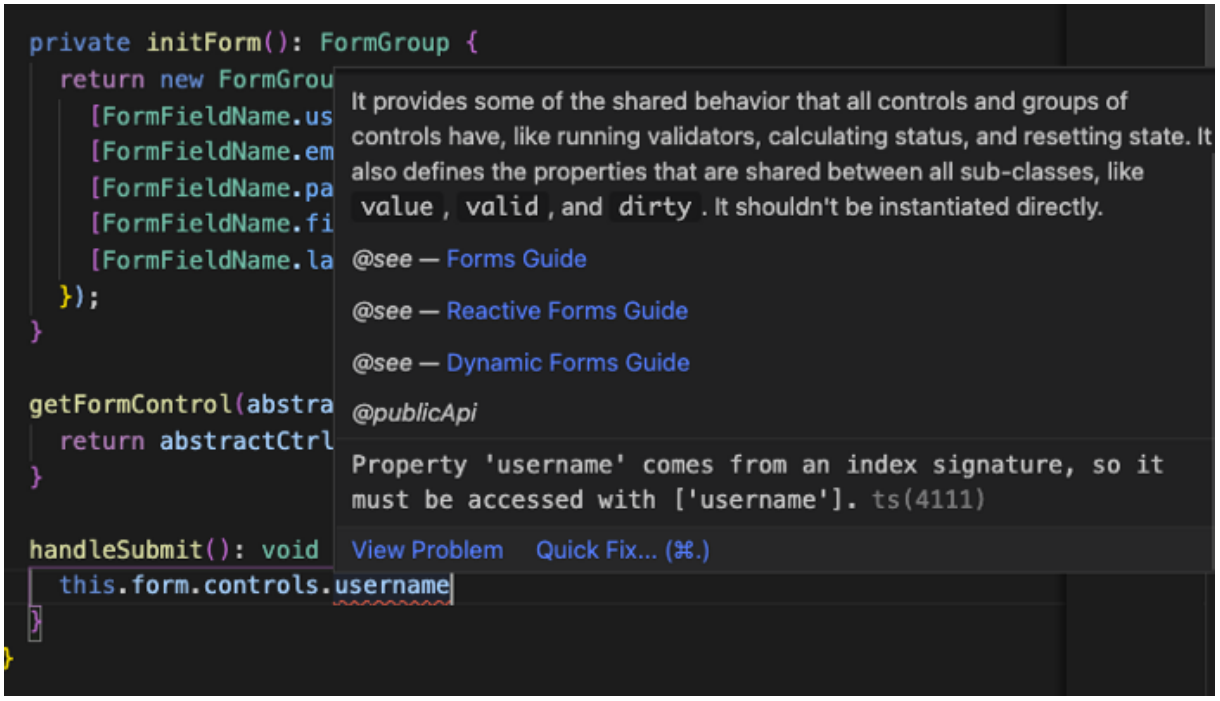
form.html

```
export class ReactiveFormComponent implements OnInit {
  form!: FormGroup;
  formFieldName = FormFieldName;
  formFieldLabel = FormFieldLabel;

  constructor() {
    this.form = this.initForm();
  }

  ngOnInit(): void {}

  private initForm(): FormGroup {
    return new FormGroup({
      [FormFieldName.username]: new FormControl(null),
      [FormFieldName.email]: new FormControl(null),
      [FormFieldName.password]: new FormControl(null),
      [FormFieldName.firstName]: new FormControl(null),
      [FormFieldName.lastName]: new FormControl(null),
    });
  }
}

enum FormFieldName {
  firstName = 'firstName',
  lastName = 'lastName',
  email = 'email',
  password = 'password',
  username = 'username',
}

const FormFieldLabel = {
  [FormFieldName.firstName]: 'First Name',
  [FormFieldName.lastName]: 'Last Name',
  [FormFieldName.email]: 'Email',
  [FormFieldName.password]: 'Password',
  [FormFieldName.username]: 'Username',
};
```

form.ts

With this implementation, you can surely avoid typos and bugs in your template, and whenever you want to make amends to your form field label or field name, you just need to change your model and everything will still work automatically.

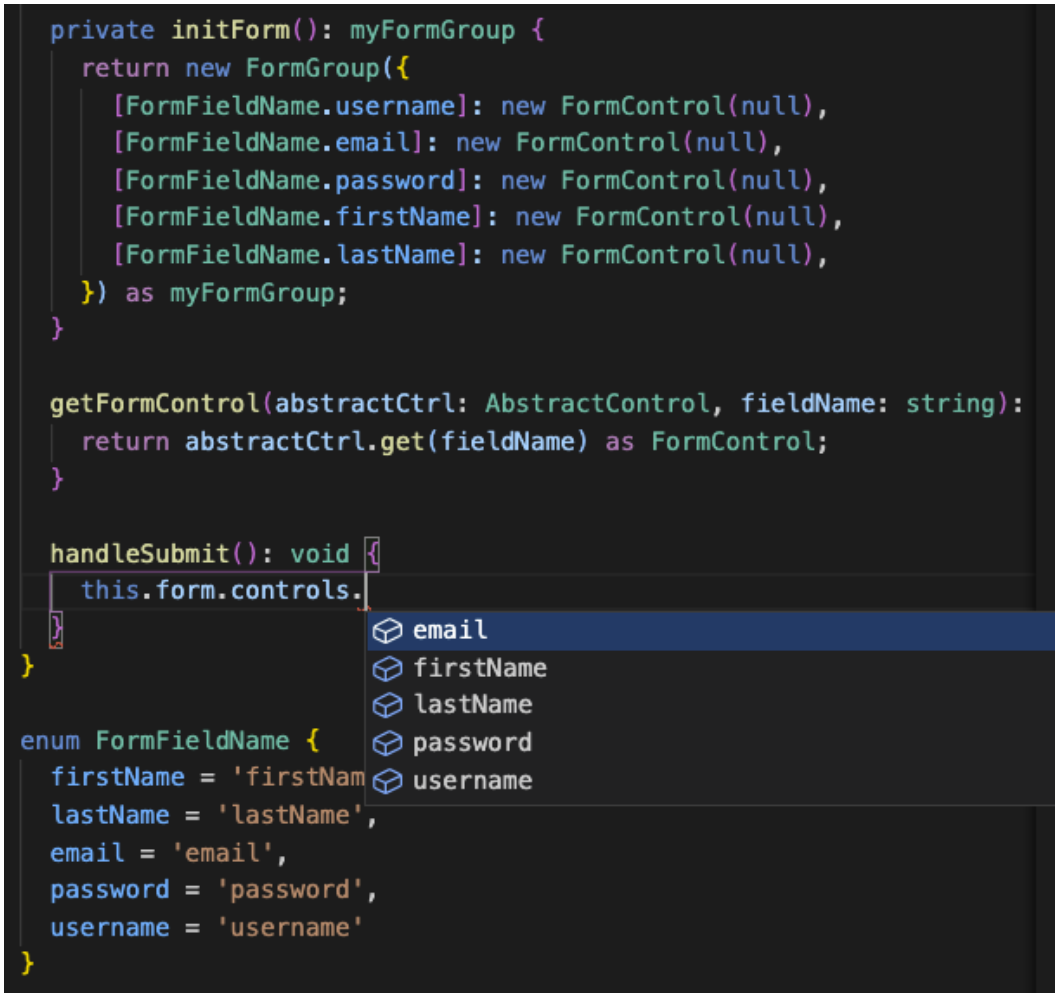
Do we get the same **IntelliSense**/typings support when we try to access our form controls value? Let's take a look at this.



We can see that the form controls does not recognise our field name as its property. Of course, we will use the workaround that is suggested by the IDE to access with `['username']` to get the formControl information.

Accessing form control with square brackets is something that we always try to avoid since we are using Typescript for Angular, we want the best practice for strong typings our implementation in accessing form control. Thankfully, with just a few tweaks, we can set up typing support for form control in our form.

Add this extra interface and type to your model file or TS file. Replace the return type of your `initForm()` and your `form` variable declaration to use this newly created `myFormGroup` and you will be able to see the **IntelliSense** support when accessing your form controls.



Finally, after all the setup, from today onwards, you can have access to bug-free and error-free form templates and strong typings in your TS file for your reactive form. 🎉

```
import { ChangeDetectionStrategy, Component, OnInit } from '@angular/core';
import { AbstractControl, FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-reactive-form',
  templateUrl: './reactive-form.component.html',
  styleUrls: ['./reactive-form.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class ReactiveFormComponent implements OnInit {
  form!: myFormGroup;
  formFieldName = FormFieldName;
  formFieldLabel = FormFieldLabel;

  constructor() {
    this.form = this.initForm();
  }

  ngOnInit(): void {}

  private initForm(): myFormGroup {
    return new FormGroup({
      [FormFieldName.username]: new FormControl(null),
      [FormFieldName.email]: new FormControl(null),
      [FormFieldName.password]: new FormControl(null),
      [FormFieldName.firstName]: new FormControl(null),
      [FormFieldName.lastName]: new FormControl(null),
    }) as myFormGroup;
  }

  handleSubmit(): void {
    console.log(this.form.controls);
  }
}

enum FormFieldName {
  firstName = 'firstName',
  lastName = 'lastName',
  email = 'email',
  password = 'password',
  username = 'username',
}

const FormFieldLabel: Record<keyof MyFormControls, string> = {
  [FormFieldName.firstName]: 'First Name',
  [FormFieldName.lastName]: 'Last Name',
  [FormFieldName.email]: 'Email',
  [FormFieldName.password]: 'Password',
  [FormFieldName.username]: 'Username',
};

interface myFormModel {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
  username: string;
}

type MyFormControls = Record<keyof myFormModel, AbstractControl>;
```

form.ts

Summary

Hope that you feel relieved after reading this article and feel happy about implementing typo or bug-free form with full TypeScript **IntelliSense** support. Let me know in the comments below if you are using another way to strict type your Angular form. I would love to see what other ways we can employ to strict type our Angular application. Cheers!

*More content at **plainenglish.io**. Sign up for our **free weekly newsletter**. Get exclusive access to writing opportunities and advice in our **community Discord**.*