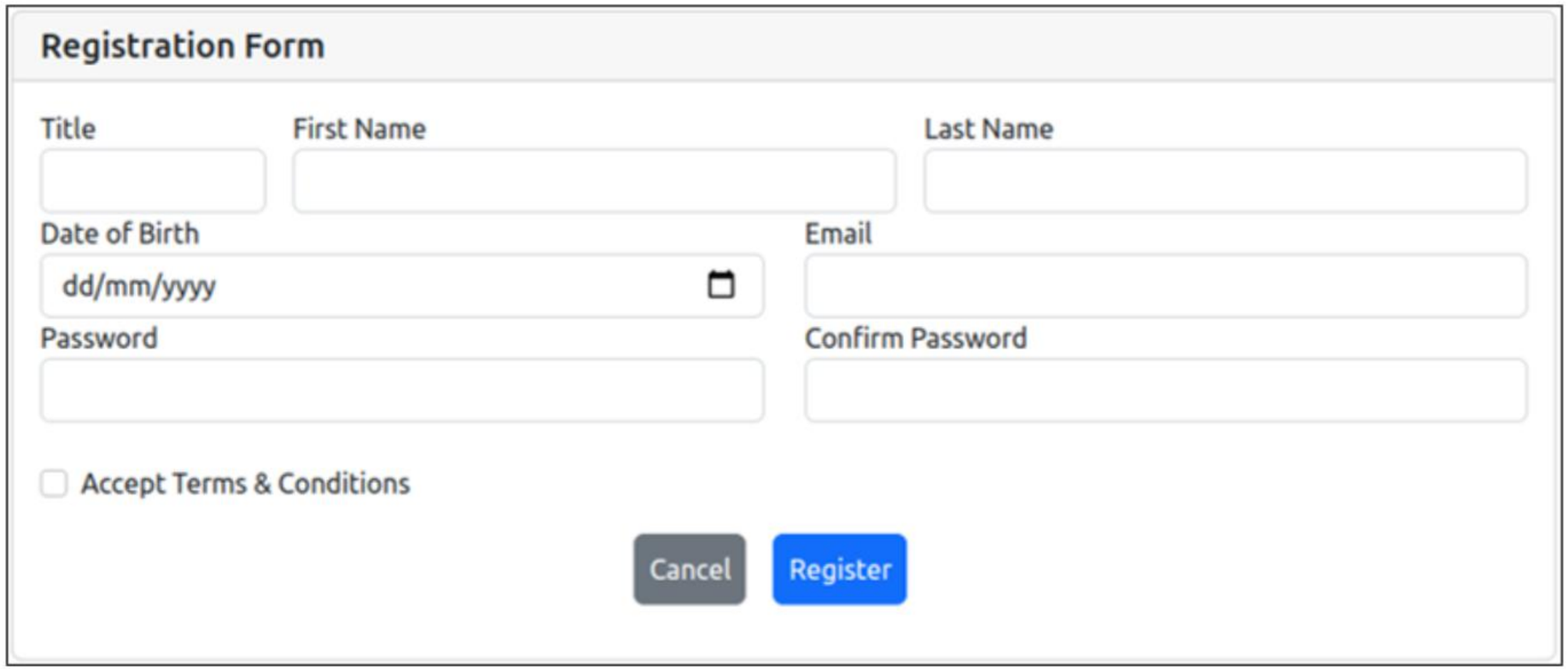
Let’s Observe Some Common Forms

Slide Note

User Registration Form



Menu

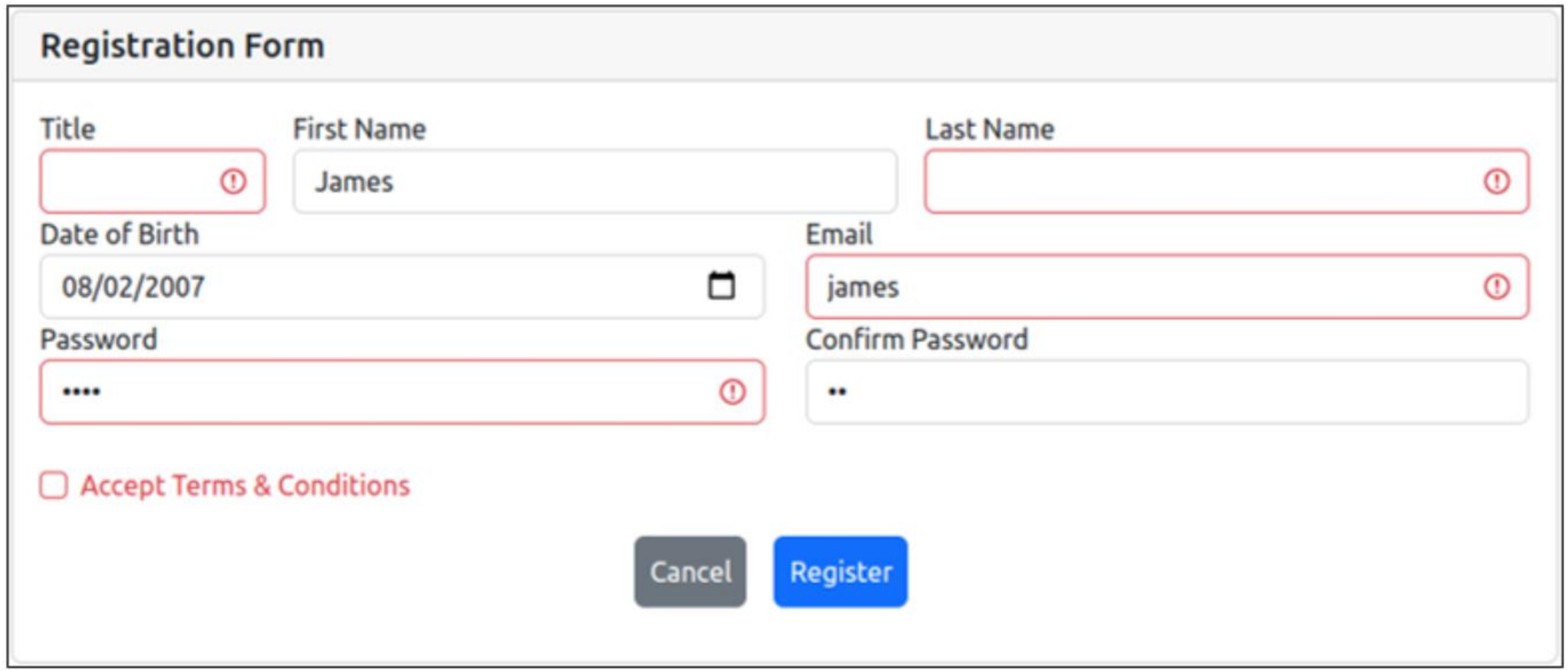
00:02 00:00:41 02/ 41

Slide Note

What if the user learns about the incorrect entries only after hitting the submit button?

Slide Note

Registration Form: Improper Error Messages

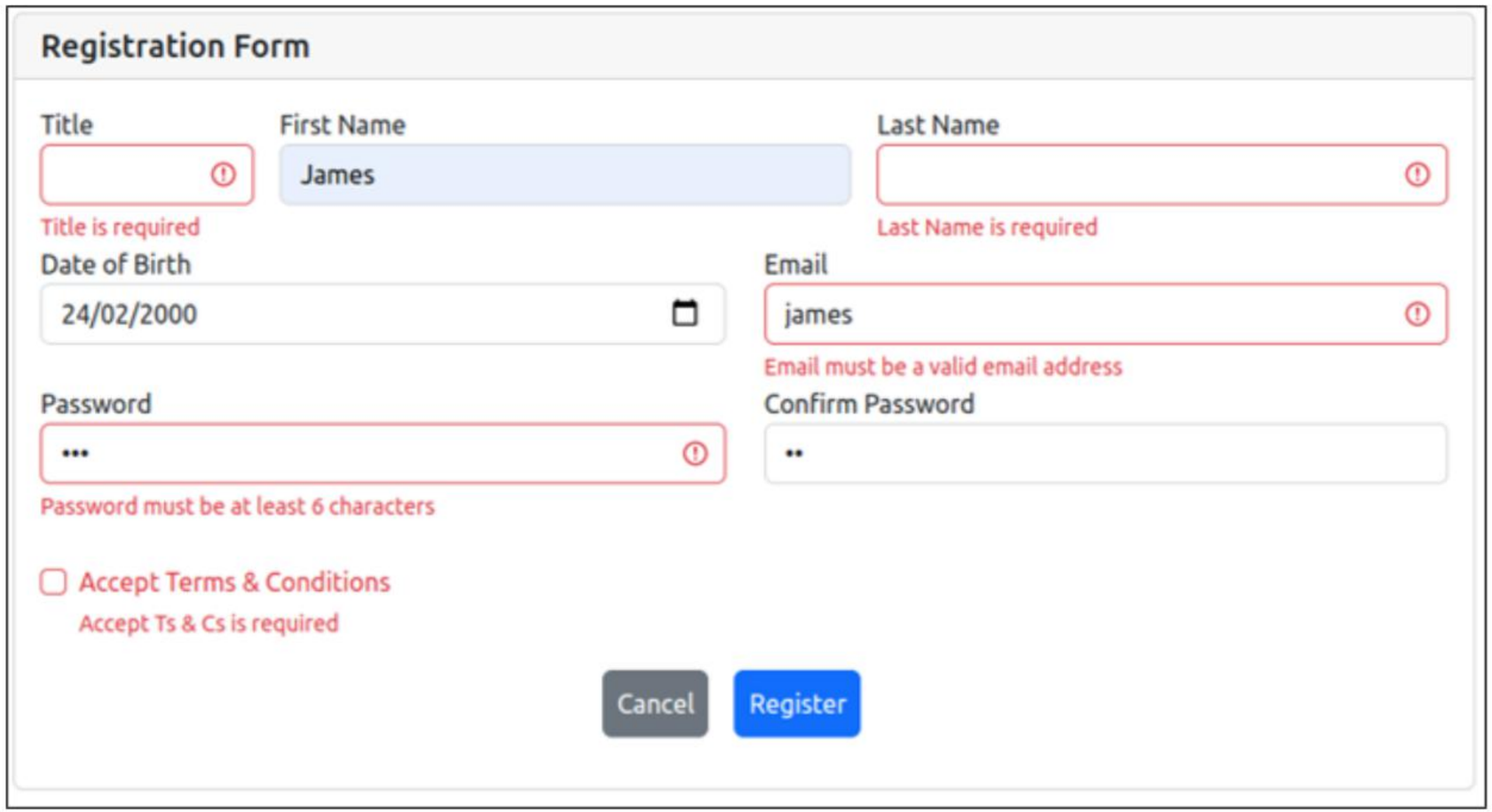


Slide Note

Does displaying proper error messages to the user improve the user experience?

Slide Note

User Registration Form: Proper Validation Error Messages



Slide Note

Checking the valid values for form entries can be done using JavaScript methods. But a lot of code has to be written to achieve this.

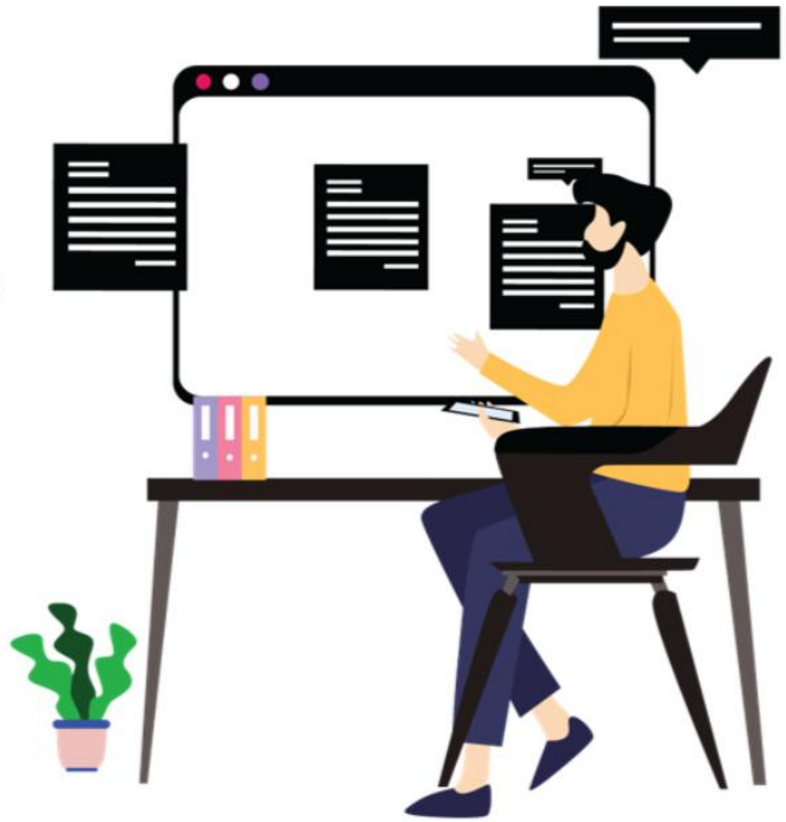
Is it possible to check whether the entries made in the form fields are valid prior to the form submission?

Slide Note

An Angular form helps keep track of the form data and displays proper validation error messages as and when the user enters data.

Slide Note

Developing Interactive Template-Driven Forms Inside SPA



Slide Note

Learning Objectives

- Explain template-driven forms in Angular
- Set up the form model in template-driven forms
- Track forms and form control status
- Analyze how the data flows in template-driven forms
- Generate and display error messages to the user when the form fields are invalid

Slide Note

Angular Forms

- Most of the applications have forms to deal with user input. The input can be in the form of registration, login, an application, profile details, etc.
- Angular forms:
  - capture user inputs from the view
  - validate user input
  - create a form model and data model to update
  - provide a way to track changes
- Angular provides two different approaches to dealing with user input through forms.
  - **Template-driven Forms:** Creates a form data model using directives in templates.
  - **Reactive Forms:** Provide explicit, direct access to the form data model

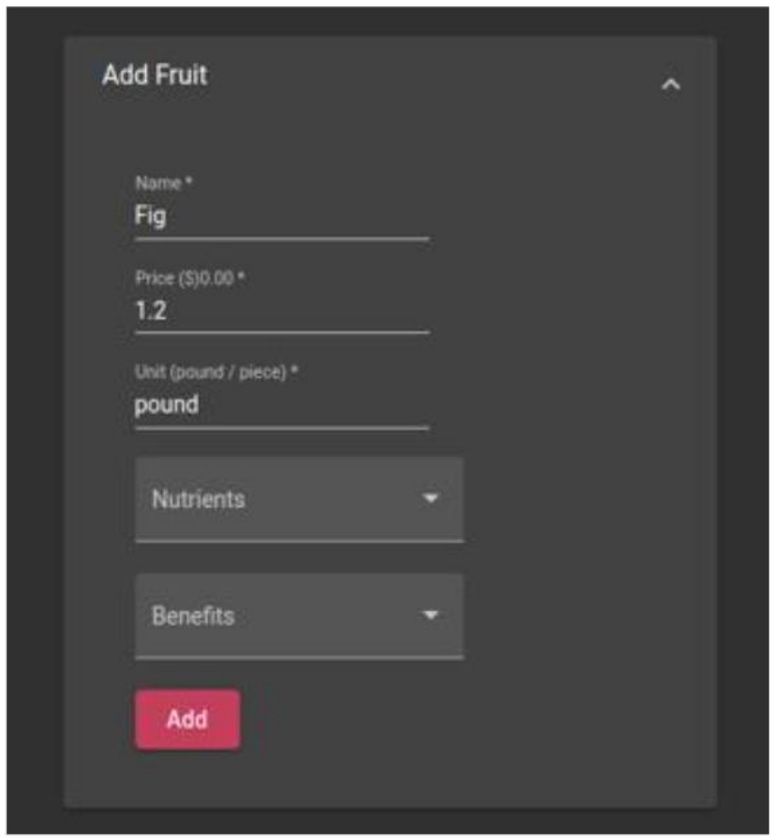
Slide Note

Template-Driven Forms

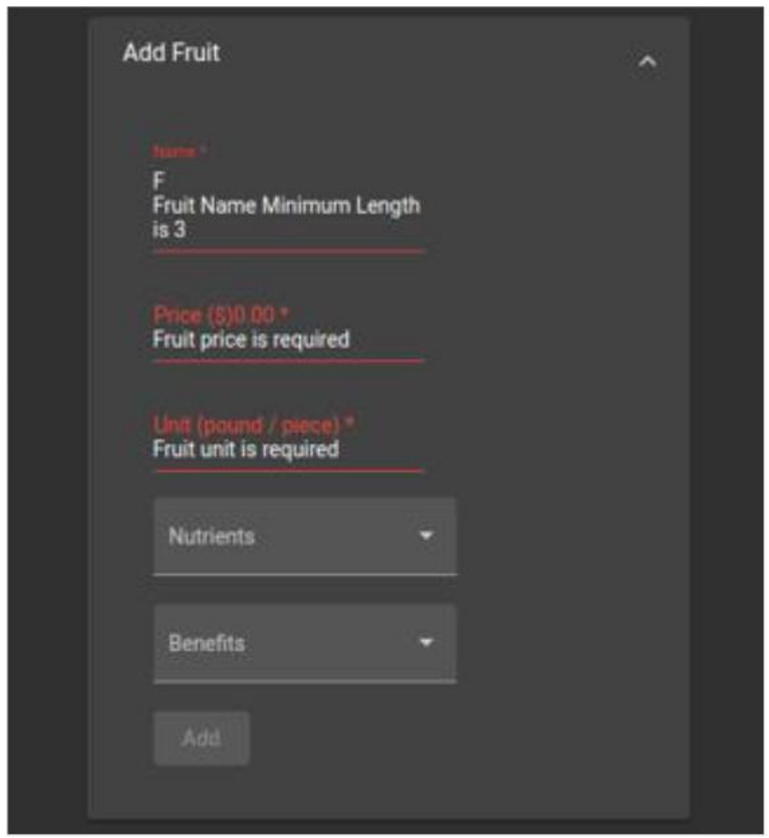
- Template-driven forms depend on directives in the template to create and manipulate the form's data model.
- In Template-driven forms, the behaviors and validations are specified using directives and attributes in the template. The rest is taken care of by the Angular framework working behind the scenes.
- Forms with basic form requirements can be easily created using template-driven form approach.
- This approach is useful for adding a simple form to an app, such as an email signup form, feedback form, contact form, etc.

Slide Note

Template-Driven Form Preview



Form With Valid Values

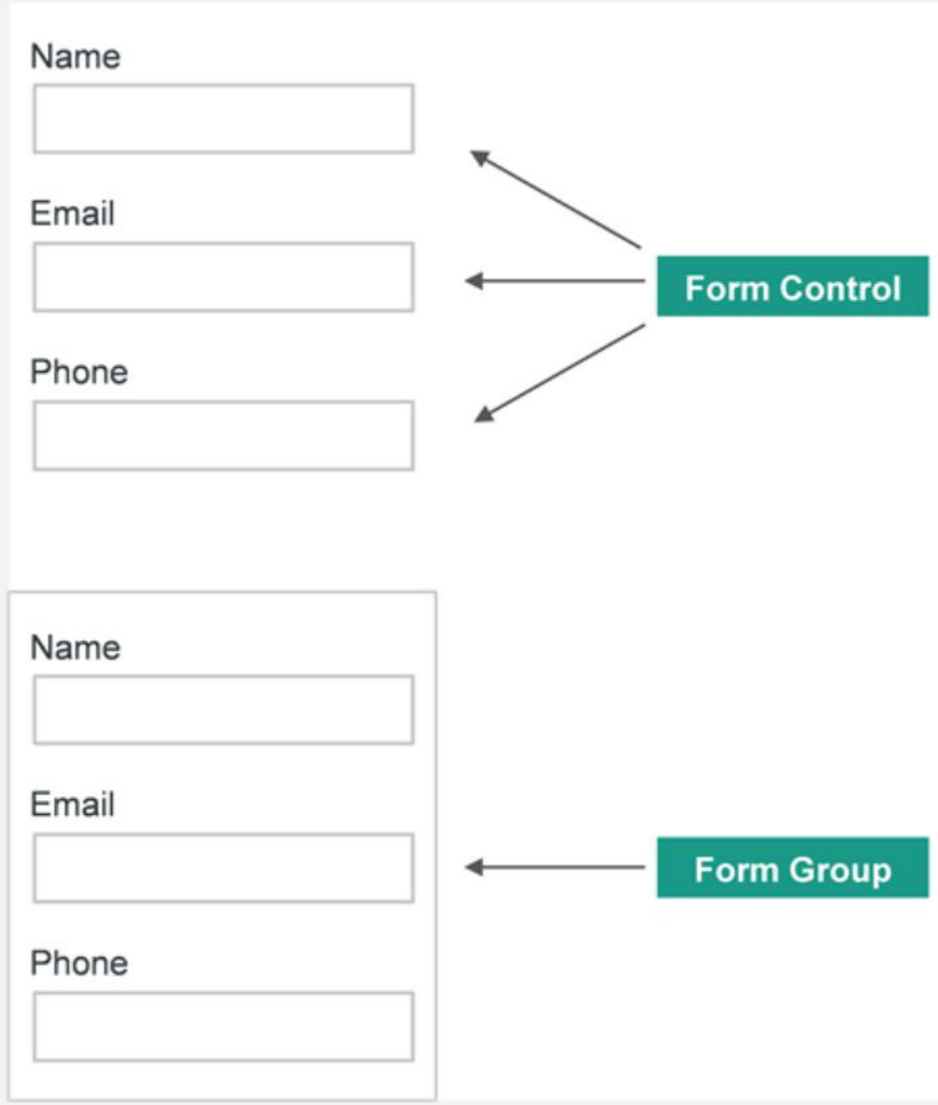


Form With Validation Error Messages

Slide Note

Angular Form Building Blocks

- **FormControl** represents a single input field in an Angular form and tracks its value.
- **FormGroup** is a collection of FormControls.
  Each FormControl is a property in a FormGroup with the control name as the key.



Slide Note

Directives Used in Template-Driven Forms

- **The NgForm** directive is used to set up the form:
  - It automatically detects the <form> tag with form elements and converts it into a template-driven form since it has a selector that matches the <form> tag.
  - It binds it to a form to track aggregate form value and validation status.
  - It creates a top-level FormGroup
- **The NgModel** directive is used to set up the form controls:
  - It creates a FormControl instance from a domain model and binds it to a form control element.
  - The FormControl instance tracks the value, user interaction, and validation status of the control.
- **The NgModelGroup** directive is used to set up the form group:
  - It creates and binds a FormGroup instance to a DOM element.
  - It can only be used as a child of NgForm to validate a sub-group of your form separately from the rest of the form.

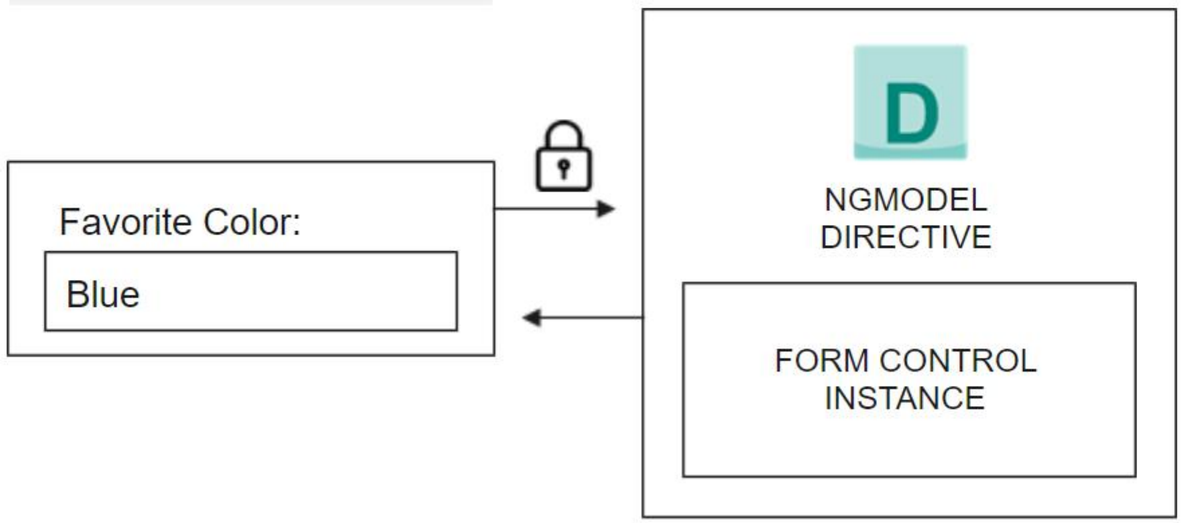
Slide Note

Setting up the Form Model

- In a template-driven form, the source of truth for the form Model is the template.
- The directive NgModel creates and manages a FormControl instance for a given form element.
- The FormControl instances can only be accessed via the NgModel directive.

You can only access the FormControl instance via the NgModel directive

TEMPLATE-DRIVEN FORMS



Slide Note

Steps for Creating Template-Driven Forms

- Following tasks need to be performed to create Template-driven forms:

Add FormsModule to application root module.

Add ngModel directive to each form control, which binds the form control with the corresponding HTML element.

Add name attribute to each form control to register it with the NgForm directive attached to the parent <form> element.

Slide Note

Create a User Profile Form

Create a simple user profile form using Angular template-driven forms to add a new user with details like username, email address, and phone number.

Click here for the demo solution.

DEMO

Menu

myrepos.stackroute.niit.com/frontend-workspace-nd-v3/demonstrations/.../user-profile

00:18:00:00:41

18/ 41

Slide Note

```
import { NgModule } from '@angular/core';
import { BrowserModule } from
'@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-
routing.module';
import { AppComponent } from
'./app.component';
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

FormsModule

- Template-driven forms are enabled by adding the FormsModule to the application root module.
- FormsModule provides the directives to build template-driven forms, making them available for import by NgModules that import this module.
- Import FormsModule in app.module.ts and add it to the NgModule imports array.
- Once added, Angular will apply a NgForm directive to every <form> HTML template element implicitly.

Slide Note

An alternative way to create an object is using a constructor.

A constructor is a function that specifies name, properties, and methods of object.

Inside the function, use keyword *this* to assign values to the object's properties based on the values passed to the function.

Create an instance of the object with new.

In the code below:

Employee is name of object type

firstName, lastName, email are its properties

```
function Employee(firstName, lastName, email) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
}
const employee = new Employee('Nancy','Davilio','nancy.d@gma
```

Creating a Template-Driven Form

- In the example, a user profile form is created using the `<form>` tag.
- Export the `ngForm` directive into a local template variable (ex: `#userForm="ngForm"`)
- The template variable `userForm` can be used to access the properties and methods of `ngForm`.
- The `ngModel` directive is used to bind the form element with the `FormControl` instance.
- The `name` attribute is used to build the form tree. The value given to the name attribute is the property name inside the template model, with the form control instance as the value.
- The `ngSubmit` event is used to submit the form data to the component class.

```
<section>
 <h2>Enter User Details</h2>
 <form (ngSubmit)="onSubmit(userForm)" #userForm="
ngForm">
  <mat-form-field>
  <mat-label>Username </mat-label>
  <input type="text" [(ngModel)]="user.username"
name="username" #username="ngModel" id="username">
  </mat-form-field>
  <mat-form-field>
  <label>Email </label>
  <input type="text" name="email" [(ngModel)]="use
r.email" #email ="ngModel" id="email">
  </mat-form-field>
  <mat-form-field>
  <label>Phone Number </label>
  <input type="text" name="phone" [(ngModel)]="use
r.phone" #phone="ngModel" id="phone">
  </mat-form-field>
  <button type="submit">Submit</button>
 </form>
</section>
```

Slide Note

Pristine means that the user hasn't changed the value since it was displayed in this form.

Tracking Form Status

- The template local variable assigned to the NgForm directive allows us to check the status of the form like (whether the form is valid, submitted, and to get the values of the form elements.)

Property	Description
Value	Returns the object containing the value of every FormControl
Valid	Returns true if the form is valid, otherwise returns false.
Touched	True if the user has entered a value in at least one field.
Submitted	Returns true if the form is submitted, otherwise returns false.

Slide Note

Tracking Form Control Status

- The NgModel directive on a form control tracks the state of that control.
- It tells us if the user touched the control, if the value changed, or if the value became invalid.
- Angular sets special CSS classes on the control element to reflect the state.

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

- Angular also applies the ng-submitted class to form elements upon submission, but not to the controls inside the form element.

Slide Note

Track Form and Form Control Status in the Profile Form

Access the ngForm instance using the local template variable to retrieve the form status.

Provide visual feedback for invalid status of form controls by accessing them using the template reference variable and changing their appearance.

Click here for the demo solution.

DEMO

Slide Note

Form Control Status Values

#name="ngModel"

The control name input is valid	name.valid	name.invalid
The control name input value has changed	name.dirty	name.pristine
The control name has been visited	name.touched	name.untouched

The values in green boxes return true
The values in the red boxes return false

Control is **dirty**: User changes the value in the field
Control is **touched**: User blurs the form control element

Slide Note

Angular framework dynamically adds CSS class to template-driven form elements. Using these classes, visual feedback can be provided by applying styles to these form elements based on their validation status.

Visual Feedback for Invalid Status

- Angular's framework dynamically adds CSS classes to template-driven form elements. Using these classes, visual feedback can be provided by applying styles to these form elements based on their validation status.

```
.ng-valid[required],
.ng-valid.required {
border - left: 3px solid green;
}
.ng-invalid: not(form) {
border-left: 3px solid red;
}
```

- Messages can be made visible/hidden using the [hidden] attribute based on the form control's valid status.

```
<input matInput placeholder="Enter your phone" required name="phone"
#phone="ngModel" [(ngModel)]="user.phone" id="phone">
<mat-hint [hidden]="phone.valid || phone.pristine">
Phone number is required
</mat-hint>
```

Slide Note

Submit the Form Using ngSubmit

The Submit button at the bottom of the form triggers a form-submit event.

To respond to this event, take the following steps:

1. Bind the form's `ngSubmit` event property to the form components `onSubmit()` method.

```
<form (ngSubmit)="onSubmit()" #userForm="ngForm">
```

2. Use the template reference variable, `#userForm`, to access the form that contains the Submit button and create an event binding. You will bind the form's overall-validity property to the submit button's disabled property.

```
<button type="submit"  [disabled]="userForm.valid">Submit</button>
```

Slide Note

Quick Check

Which property of NgModel is set to true when an input value is modified by a user?

A. Pristine

B. Dirty

C. Invalid

D. modified

Slide Note

Quick Check: Solution

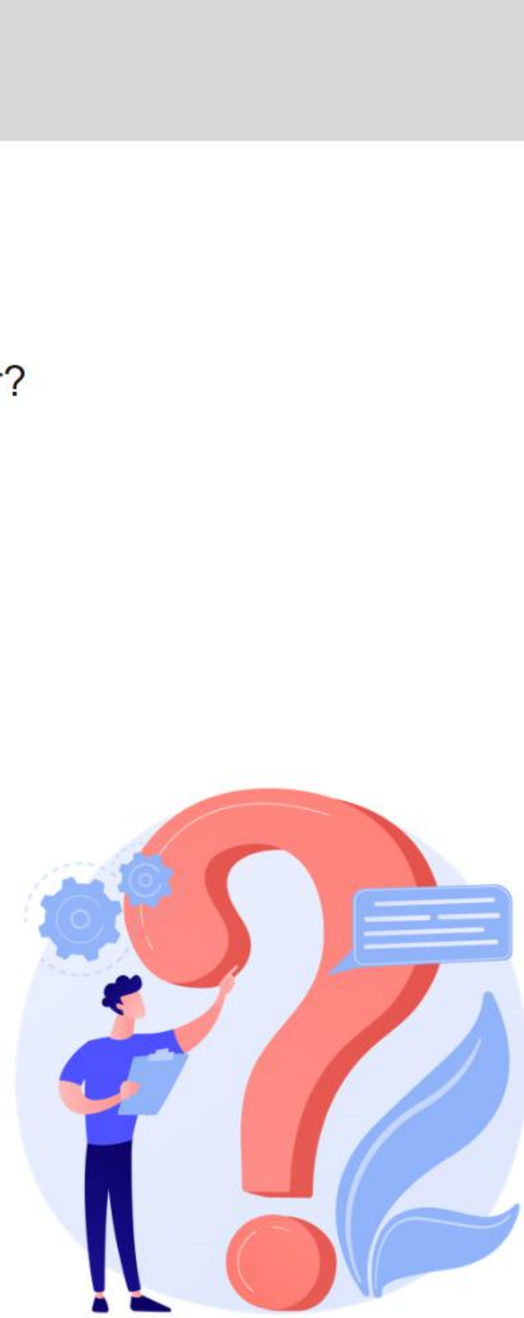
Which property of NgModel is set to true when an input value is modified by a user?

A. Pristine

B. **Dirty**

C. Invalid

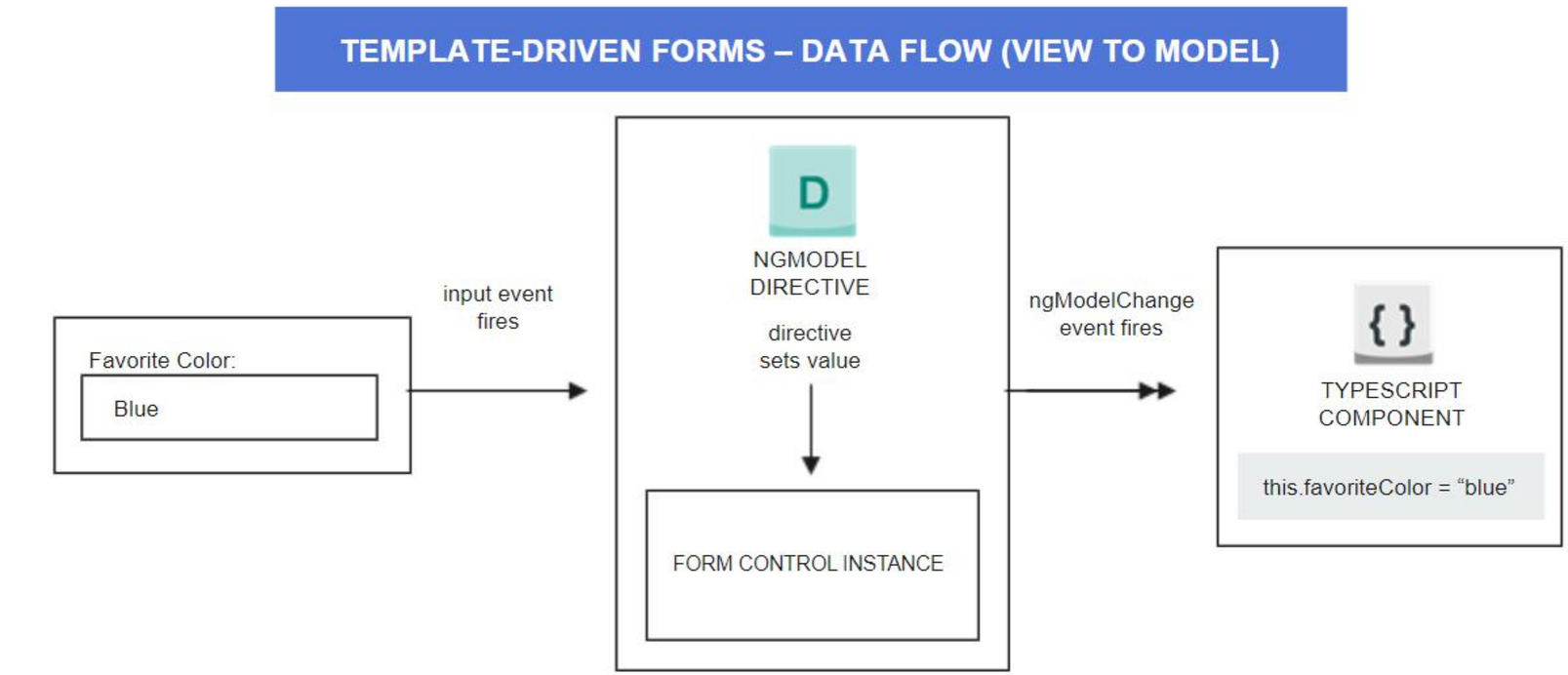
D. modified



Slide Note

The view-to-model diagram shows how data flows when an input field's value is changed from the view, which is explained in the next slide.

View-to-Model in Template-Driven Forms



START	
this.favoriteColor (ngModel)	RED
FormControl instance value	RED
view	BLUE

DIRECTIVE SETS VALUE	
this.favoriteColor (ngModel)	RED
FormControl instance value	BLUE
view	BLUE

END RESULT	
this.favoriteColor (ngModel)	BLUE
FormControl instance value	BLUE
view	BLUE

Slide Note

Data Flow From View-to-Model

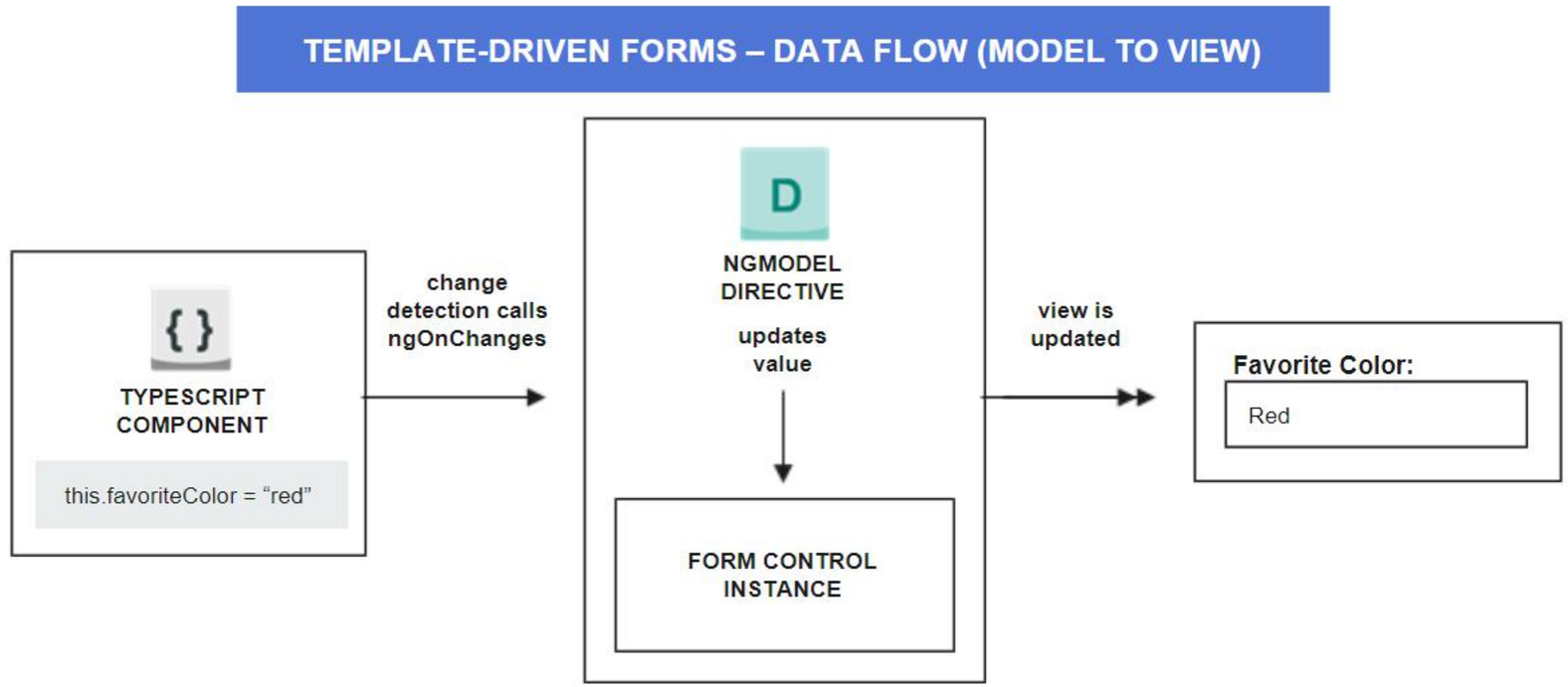
The following scenario shows how data flows when an input field's value is changed from the view:

1. The user types *Blue* into the input element.
2. The input element emits an "input" event with the value *Blue*.
3. Angular framework updates the `FormControl` instance with the new value.
4. The `FormControl` instance emits the new value through the `valueChanges` observable.
5. Any subscribers to the `valueChanges` observable receive the new value.
6. Angular framework also calls the `NgModel.viewToModelUpdate()` method, which emits an `ngModelChange` event.
7. The component template uses two-way data binding for the `favoriteColor` property. Hence, it is updated to the value emitted by the `ngModelChange` event (*Blue*).

Slide Note

The model-to-view diagram shows how data flows from the model to view when the favoriteColor changes from *Blue* to *Red,* which is explained in the next slide

Model-to-View in Template-Driven Forms



START

this.favoriteColor (ngModel)	RED
FormControl instance value	BLUE
view	BLUE

DIRECTIVE UPDATES VALUE

this.favoriteColor (ngModel)	RED
FormControl instance value	RED
view	BLUE

END RESULT

this.favoriteColor (ngModel)	RED
FormControl instance value	RED
view	RED

Slide Note

Data Flow From Model-to-View

The following scenario shows how data flows from model to view when the `favoriteColor` changes from *Blue* to *Red:*

1. The `favoriteColor` value is updated in the component.
2. The `ngOnChanges` lifecycle hook is called on the `NgModel` directive instance because the value of one of its inputs has changed.
3. The `ngOnChanges()` method queues an async task to set the value for the internal `FormControl` instance.
4. The task to set the `FormControl` instance value is executed asynchronously.
5. The `FormControl` instance emits the latest value through the `valueChanges` observable.
6. Any subscribers to the valueChanges observable receive the new value.
7. The Angular framework updates the form input element with the latest `favoriteColor` value in the view.

Slide Note

Add Fruit Detail Form in Fruit Fantasy App

Modify the add fruit form created in the previous Sprint to have various form fields like fruit name, price, unit, nutrients, and benefits.
Add ngForm, ngModel directives to the add fruit form to access its properties and understand the data flow in template-driven forms.

Click here for the demo solution.

DEMO

Slide Note

Input Validation in Template-Driven Forms

Slide Note

• Custom validators are covered in the next sprint (Angular Reactive Forms)

Validating Input in Template-Driven Forms

- The HTML validation attributes like required, minlength are used for validating user inputs. These attributes can be used to add validation to template-driven forms.
- Angular uses directives to match HTML native validation attributes with validator functions in the framework.
- Examples of built-in validators include min, max, required, minlength, etc.
- Custom validators can also be defined as validator functions to match business needs. They can be added as a directive to the template later.
- Every time the value of a form control changes, Angular runs validation and generates either a list of validation errors that results in an INVALID status, or null, which results in a VALID status.

Slide Note

Add Validators in the Fruit Fantasy App

Add built-in validators like `required`, `minlength`, and `pattern` for adding fruit form to validate various form control elements. Display custom error messages for each invalid status of form controls.

Click here for the demo solution.

DEMO

Slide Note

Input Validation

- `<input>` element carries the HTML validation attributes like `required` and `minlength`.
- `#name="ngModel"` exports `ngModel` into a local variable called `"name"` to access its state.
- The `*ngIf` on the `<mat-error>` element displays a custom error message for one of the possible validation errors.
- The `<mat-error>` element displays error messages when the control is invalid and either the user has interacted with (touched) the element or the parent form has been submitted.

```
<input type="text" id="name" #name="ngModel" requi
red minlength="3"  [(ngModel)]="fruit.name" #name=
"ngModel">
<mat-error  *ngIf="name.errors?.required"> Fruit
Name is required.
</mat-error>
<mat-error *ngIf="name.errors?.minlength"> Fruit
Name should be minimum 3 characters long.
</mat-error>
```

Slide Note

Built-in Validator Functions

Validator Functions	Description
min()	Requires the control's value to be greater than or equal to the provided number.
max()	The control's value should be less than or equal to the provided number.
required()	The control's value should be a non-empty value.
email()	The control's value should pass an email validation test.
minLength()	The length of the control's value should be greater than or equal to the provided minimum length.
maxLength()	The length of the control's value should be less than or equal to the provided maximum length.
pattern()	The control's value must match a regex pattern.
nullValidator()	This validator performs no operation.

Slide Note

Quick Check

Which of the following is not true about a template-driven form in Angular?

A. It is used for two-way binding to update the data model.

B. It allows for the creation of dynamic fields in the form.

C. It is suitable for simple scenarios.

D. It uses an implicit form model rather than an explicit.

Slide Note

Quick Check: Solution

Which of the following is not true about a template-driven form in Angular?

A. It is used for two-way binding to update the data model.

B. **It allows for the creation of dynamic fields in the form.**

C. It is suitable for simple scenarios.

D. It uses an implicit form model rather than an explicit.