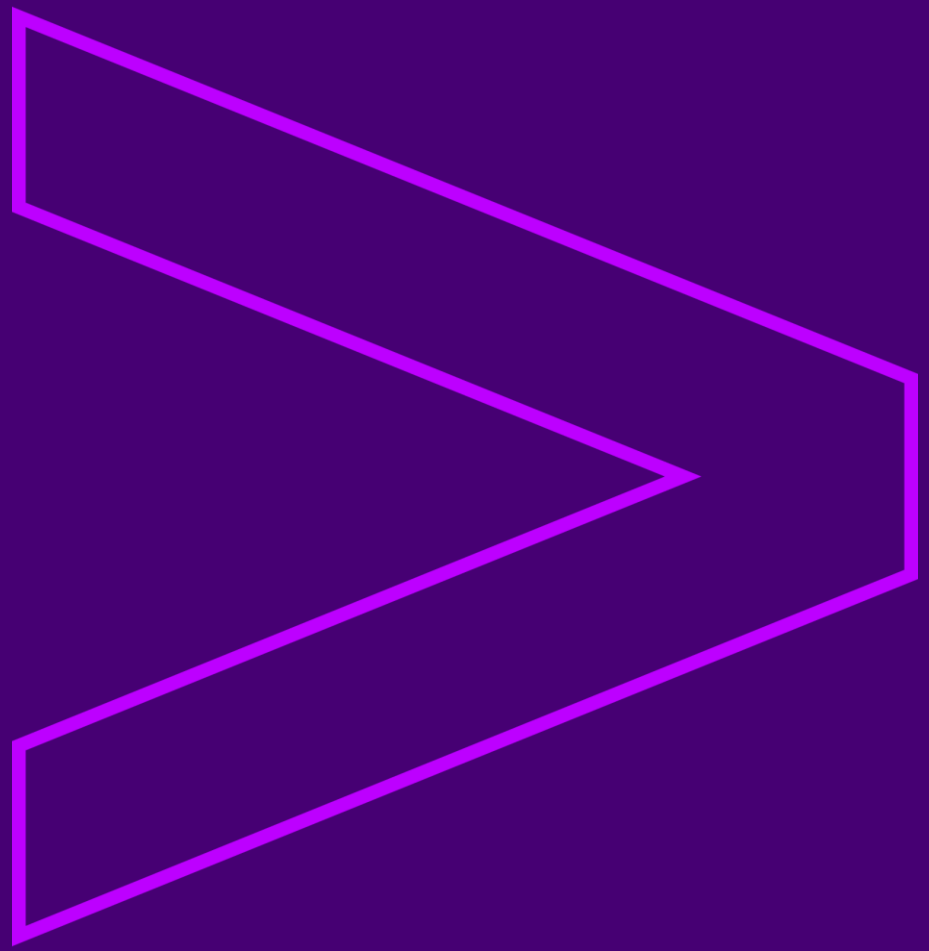
Learning and Knowledge Management

ANGULAR - FORMS



accenture

MODULE OBJECTIVES

At the end of this module, you should be able to:

- Understand the types of Forms in Angular
- Create a Template driven form and do validations
- Create a Model driven form and do validations

AGENDA

- Introduction to Forms
- Types of Forms
- Template Driven Forms (TDF)
  - Building a TDF
  - Validations in a TDF
- Model Driven Forms (MDF)
  - Building a MDF
  - Validations in a MDF

Forms in Angular

- Forms are the most important part of any business application
- Forms are used to log in, submit a request, place an order, book a flight, schedule a meeting, and perform countless other data-entry tasks
- Developing forms requires expertise, as well as framework support for two-way data binding, change tracking, validation, and error handling

Types of Forms in Angular

In Angular, there are mainly two types of forms

1. Template-Driven Forms (TDF)
2. Model Driven Forms (MDF)

Template Driven Forms

Template Driven Forms (TDF)

- In Angular Applications, we can build forms by writing templates in the Angular template syntax with the form-specific directives and techniques to implement form functionality.
- It uses the directives in the template to create and manipulate the underlying form object model.
- Best suited for simple forms
- But not as scalable as model-driven forms

Template Driven Forms (TDF)

Using Angular Template, We can,

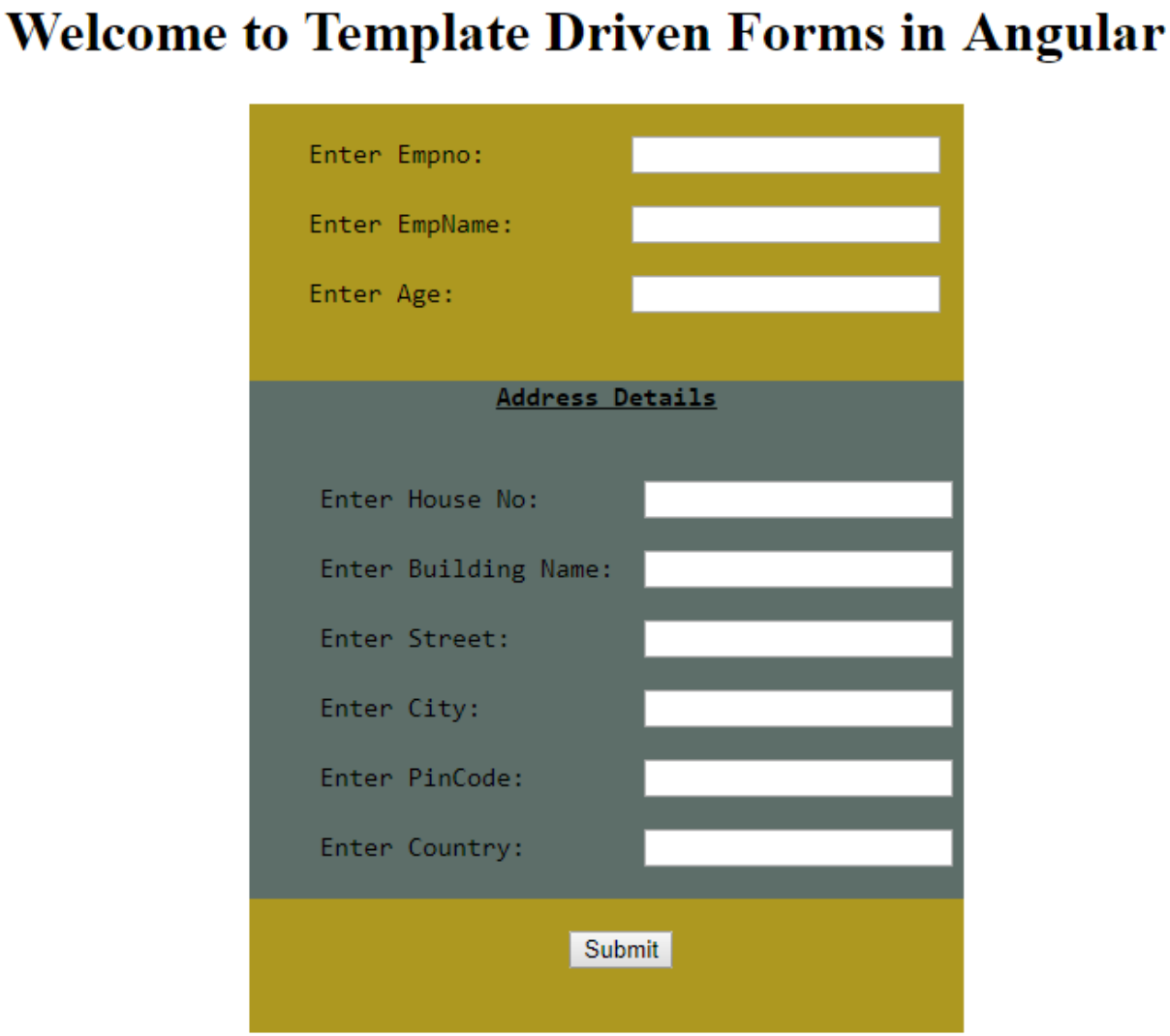
1. Use controls creatively
2. Bind them to data
3. Specify validation rules
4. Display validation errors
5. Conditionally enable or disable specific controls

Angular makes all the above tasks very simple.

Building a Template Driven Form (TDF)

Lets us build a forms with the following controls.

This forms accepts all the values and when the button is clicked, it Displays the values on to the console.



Building a Template Driven Form (TDF)

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {FormsModule} from '@angular/forms'

import { AppComponent } from './app.component';

@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule,FormsModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Import FormsModule from '@angular/forms

Include FormsModule in the imports array

accenture

Building a Template Driven Form (TDF)

While Creating TDF , we make use of the following three Directives

1. **NgForm Directive** Creates a top-level FormGroup instance and binds it to a form to track aggregate form value and validation status.

2. **NgModelGroup Directive** Creates and binds a FormGroup instance to a DOM element (used for subgroups)

3. **NgModel Directive** Creates a FormControl instance from a domain model and binds it to a form control element.

The FormControl instance will track the value, user interaction, and validation status of the control and keep the view synced with the model.

If used within a parent form, the directive will also register itself with the form as a child control.

Building a Template Driven Form (TDF)

```
 <h1 align="center">Welcome to Template Driven Forms in Angular</h1>
  <div align="center" >
<div align="left"  style="background-color:#AD9820 ; width:400px;height:520px;font-size:15px">
<form #EmployeeDetailsForm = "ngForm"   (ngSubmit)="DisplayOnConsole(EmployeeDetailsForm.value)">
<pre>
    <label>Enter Empno:</label>          <input type="text" name="empno" ngModel>
    <label>Enter EmpName:</label>        <input type="text" name="empname" ngModel>
    <label>Enter Age:</label>           <input type="text" name="age" ngModel>
</pre>
<div align="center"  ngModelGroup="AddressDetails" style="background-color:#5D6E69 ;
width:400px;height:290px">
 <pre>
<h4 align="center"><u>Address Details</u></h4>
    <label>Enter House No:</label>       <input type="text" name="houseno" ngModel>
    <label>Enter Building Name:</label>  <input type="text" name="buildingname" ngModel>
    <label>Enter Street:</label>         <input type="text" name="street" ngModel>
    <label>Enter City:</label>           <input type="text" name="city" ngModel>
    <label>Enter PinCode:</label>        <input type="text" name="pincode" ngModel>
    <label>Enter Country:</label>        <input type="text" name="country" ngModel>
  <input type="submit">
  </pre>
</div>
</form>
</div>
</div>
```

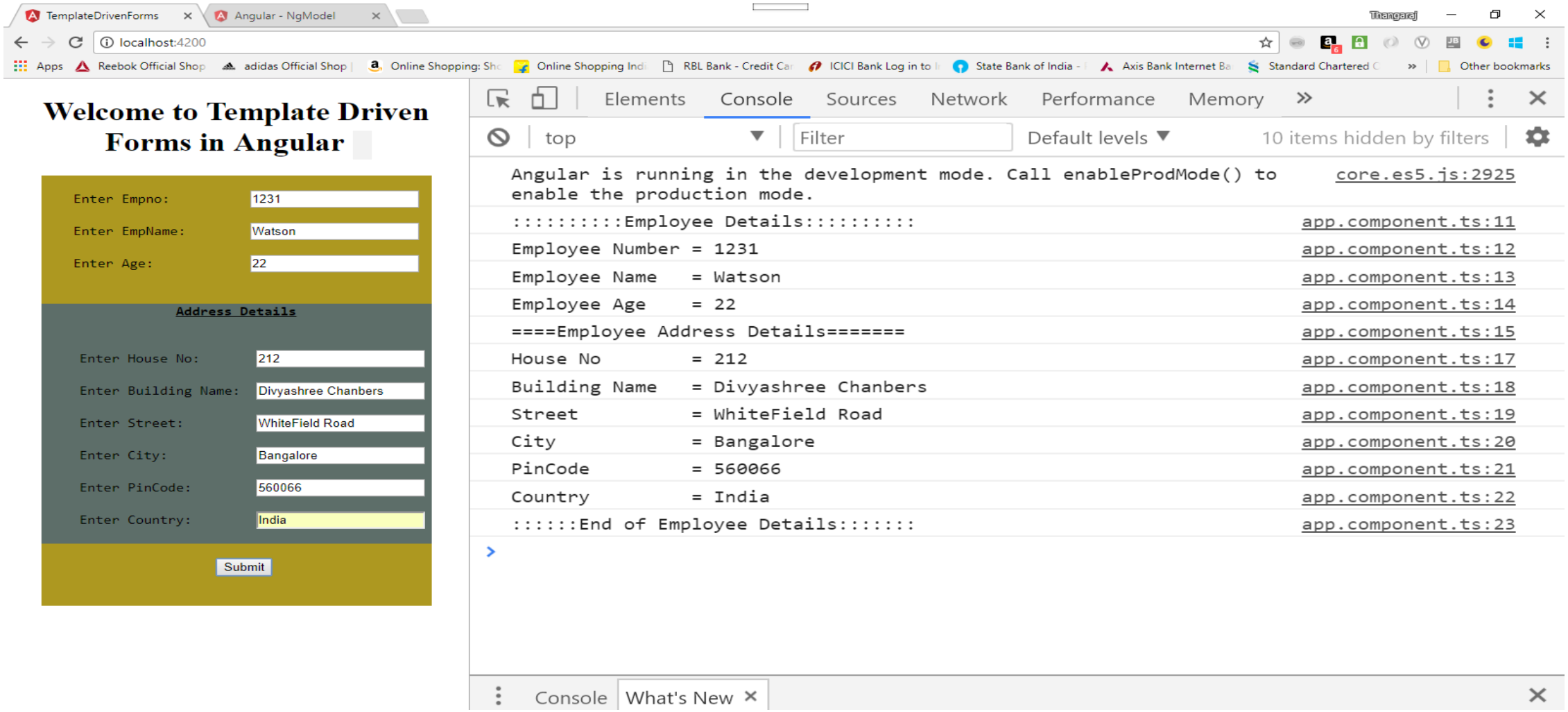
app.component.html

Building a Template Driven Form (TDF)

app.component.ts

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
DisplayOnConsole(formdata:any)
{
console.log("::::::::::Employee Details::::::::::")
console.log("Employee Number = "+formdata.empno);
console.log("Employee Name = "+formdata.empname);
console.log("Employee Age = "+formdata.age);
console.log("====Employee Address Details======= ");
console.log("House No = "+formdata.AddressDetails.houseno);
console.log("Building Name =
"+formdata.AddressDetails.buildingname);
console.log("Street = "+formdata.AddressDetails.street);
console.log("City = "+formdata.AddressDetails.city);
console.log("PinCode = "+formdata.AddressDetails.pincode)
console.log("Country = "+formdata.AddressDetails.country);
console.log("::::::End of Employee Details:::::::")
}
}
```

Building a Template Driven Form (TDF) - Output



Form Validation in Angular

- Form validation in Angular is based on HTML validation attributes.
- HTML validation attribute are used in input elements.

Angular Validation CSS Classes

For all forms in an Angular Application, Angular automatically attaches certain CSS classes to the input elements depending on the state of the control. At any point, Angular will apply three of the below CSS classes to the form depending on the 'state' of the form/field. The following are the CSS class names and its description.

CSS Class	Description	Action
.touched	Control has been visited	True if the form field has been visited, false if some fields have not been visited.
.untouched	Control has not been visited	True if the form field has not been visited, false if some fields have been visited.
.pristine	Control's value hasn't been changed	True if the form has not been changed (no form fields has changed), false if some fields have been changed.
.dirty	Control's value has been changed	The reverse of pristine - false if the form has not been changed - true if it has.
.valid	Control's value is valid	True if the form field (or the whole form = all form fields) is valid. False if not.
.invalid	Control's value isn't valid	The reverse of the valid - false if the field (or all fields in the form) is valid, true if the field is invalid.

accenture

Applying Validation rules to a TDF

Now, Let us apply *required* validation rule to Empname field and , *required* , *maxlength* and *minlength* validation rules to Street Field and also display appropriate error messages when validation rules are violated.

To achieve this, we make use of angular classes **.pristine .valid .touched .dirty.**

This is shown in the next two slides.

Applying Validation to Template Driven Forms(TDF)

Note that , In TDF , all the validation rules are written in the html template.

app.component.html

```
<h1 align="center">Welcome to Template Driven Forms in Angular </h1>
<div align="center" >
<div align="left" style="background-color:#AD9820 ; width:400px;height:520px;font-size:15px">
<form #EmployeeDetailsForm = "ngForm" (ngSubmit)="DisplayOnConsole(EmployeeDetailsForm.value)">
<pre>

<label>Enter Empno:</label> <input type="text" name="empno" ngModel>
<label>Enter EmpName:</label> <input type="text" #nameRef="ngModel" required name="empname"
ngModel>
<div [hidden]="nameRef.valid || nameRef.pristine" style="color:red">
Name Cannot be Blank..
</div>
<label>Enter Age:</label> <input type="text" name="age" ngModel>
</pre>
<div align="center" ngModelGroup="AddressDetails" style="background-color:#5D6E69 ;
width:400px;height:290px">
<pre>
```

accenture

Applying Validation to Template Driven Forms(TDF) .. Contd

```
<div align="center" ngModelGroup="AddressDetails" style="background-color:#5D6E69 ; width:400px;height:290px">
<pre>
<h4 align="center"><u>Address Details</u></h4>
<label>Enter House No:</label> <input type="text" name="houseno" ngModel>
<label>Enter Building Name:</label> <input type="text" name="buildingname" ngModel>
<label>Enter Street:</label> <input type="text" #streetRef="ngModel" maxlength="7" minlength="3" required name="street"
ngModel>
<div *ngIf="streetRef.errors && (streetRef.dirty || streetRef.touched)" style="color:red">
<div [hidden]="!streetRef.errors.required">
Street Name cannot be Blank
</div>
<div [hidden]="!streetRef.errors.minlength">
Minimum Length is 3 Characters
</div>
<div [hidden]="!streetRef.errors.maxlength">
Maximum Length is 7 Characters
</div>
</div>
<label>Enter City:</label> <input type="text" name="city" ngModel>
<label>Enter PinCode:</label> <input type="number" name="pincode" ngModel>
<label>Enter Country:</label> <input type="text" name="country" ngModel>
<input type="submit">
</pre></div></form></div></div>
```

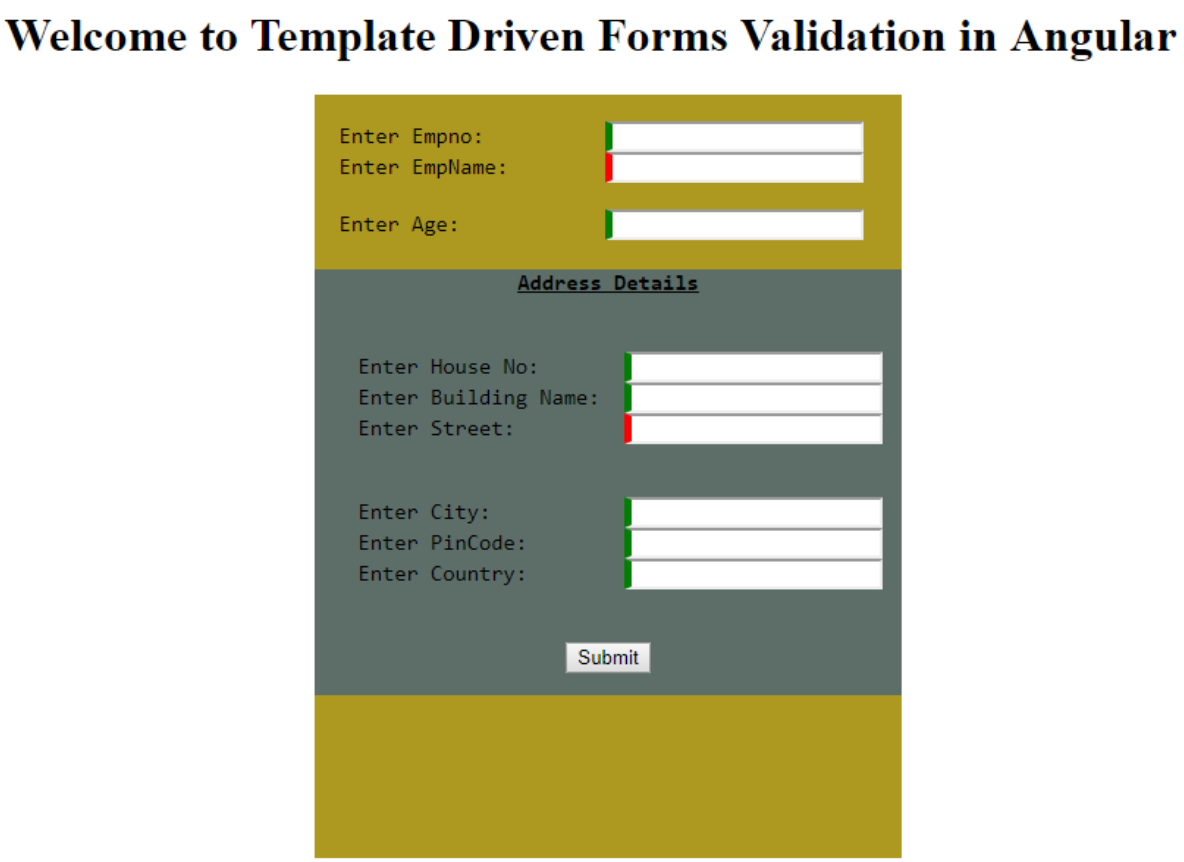
This ‘div’ will be displayed only if the conditions are met

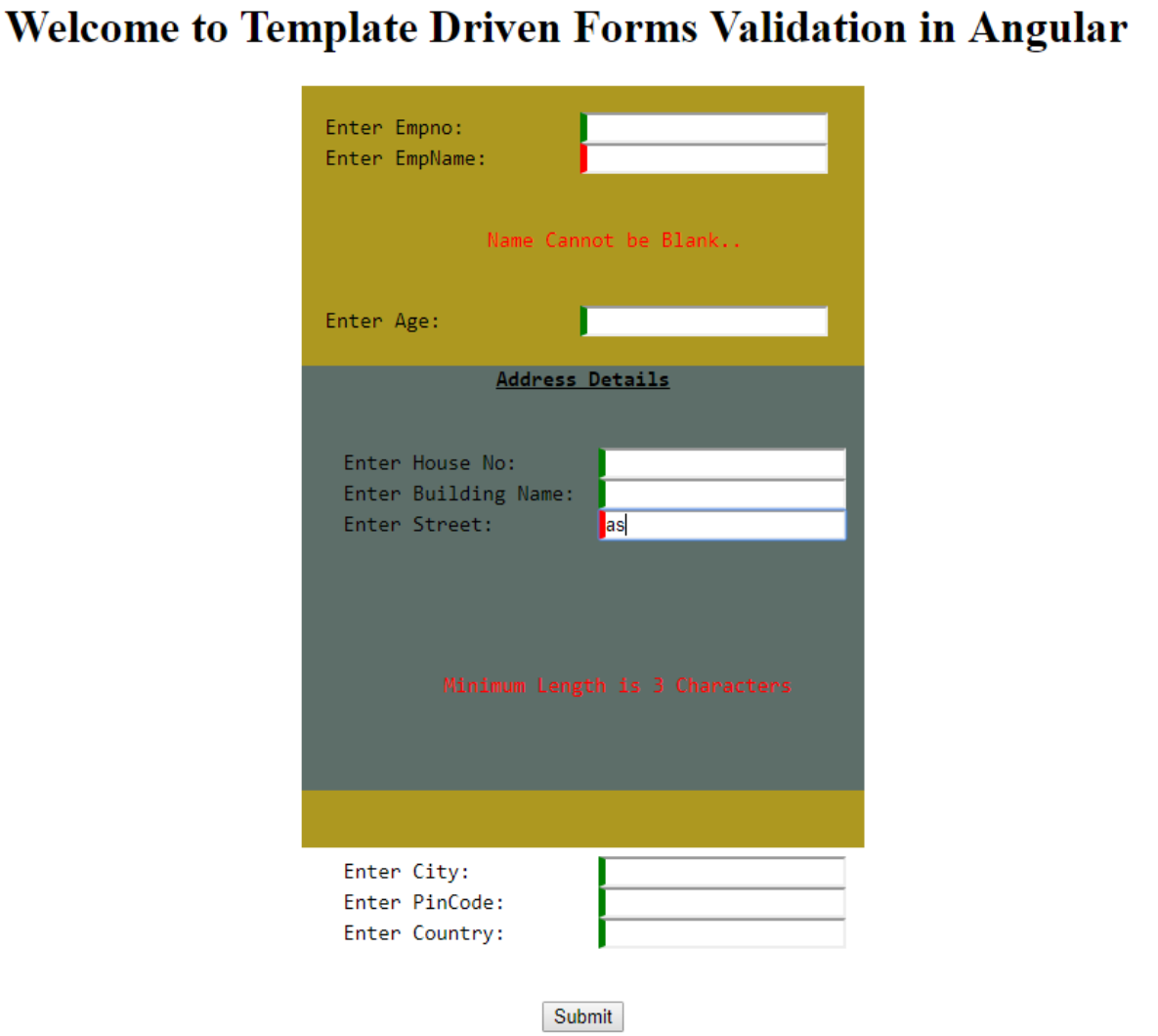
Applying Validation to Template Driven Forms(TDF) .. Contd

app.component.ts

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
 styles:[`input.ng-invalid{border-left:5px solid red;}
          input.ng-valid{border-left:5px solid green;}`      ]
})
export class AppComponent {
DisplayOnConsole(formdata:any)
{
console.log("::::::::::Employee Details::::::::::")
console.log("Employee Number = "+formdata.empno);
console.log("Employee Name = "+formdata.empname);
console.log("Employee Age = "+formdata.age);
console.log("====Employee Address Details======= ");
console.log("House No = "+formdata.AddressDetails.houseno);
console.log("Building Name =
"+formdata.AddressDetails.buildingname);
console.log("Street = "+formdata.AddressDetails.street);
console.log("City = "+formdata.AddressDetails.city);
console.log("PinCode = "+formdata.AddressDetails.pincode)
console.log("Country = "+formdata.AddressDetails.country);
console.log("::::::End of Employee Details::::::") }}
```

Applying Validation to Template Driven Forms(TDF) .. Output





Model driven forms

Model Driven / Reactive Forms

Reactive forms is an Angular technique for creating forms in a *reactive* style

While Creating Model Driven Forms / Reactive Forms, we need to use classes like FormGroup , FormControl etc,

FormControl tracks the value and validity status of an individual form control. It corresponds to an HTML form control such as an input box or selector.

FormGroup tracks the value and validity state of a group of FormControls. The group's properties include its child controls.

The top-level form in the component is a FormGroup.

To Understand Model Driven Forms, let us consider the same example discussed previously in TDF.

Building a Model Driven Form (MDF) – Step 1

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {ReactiveFormsModule} from '@angular/forms'

import { AppComponent } from './app.component';

@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, ReactiveFormsModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Import ReactiveFormsModule from '@angular/forms

Include ReactiveFormsModule in the imports array

accenture

Building a Model Driven Form (MDF) – Step 2

In the app.component.ts , create an instance of FormGroup.

Example EmployeeDetails.

This FormGroup instance manages the data from the form. i.e. it contains all the other FormControls.

Each FormControl defined in the FormGroup receives Data from a form element from html template .

A FormGroup can contain other FormGroups.

Building a Model Driven Form (TDF) – Step 2 Contd..

app.component.ts

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
public EmployeeDetails:FormGroup;
constructor(){
this.EmployeeDetails = new FormGroup({
empno: new FormControl(),
empname:new FormControl(),
age: new FormControl(),
AddressDetails: new FormGroup({
houseno: new FormControl(),
buildingname: new FormControl(),
street:new FormControl(),
city:new FormControl(),
pincode:new FormControl(),
country:new FormControl(),
})
});
}
DisplayOnConsole():void
{
console.log("::::::::::Employee Details::::::::::")
console.log(this.EmployeeDetails.value);
Details
}}
```

Import classes FormGroup and FormControl from @angular/forms module

Instance of FormGroup.

A FormGroup Instance inside another FormGroup.

Instances of FormControl.

Building a Model Driven Form (TDF) – Step 3

app.component.html

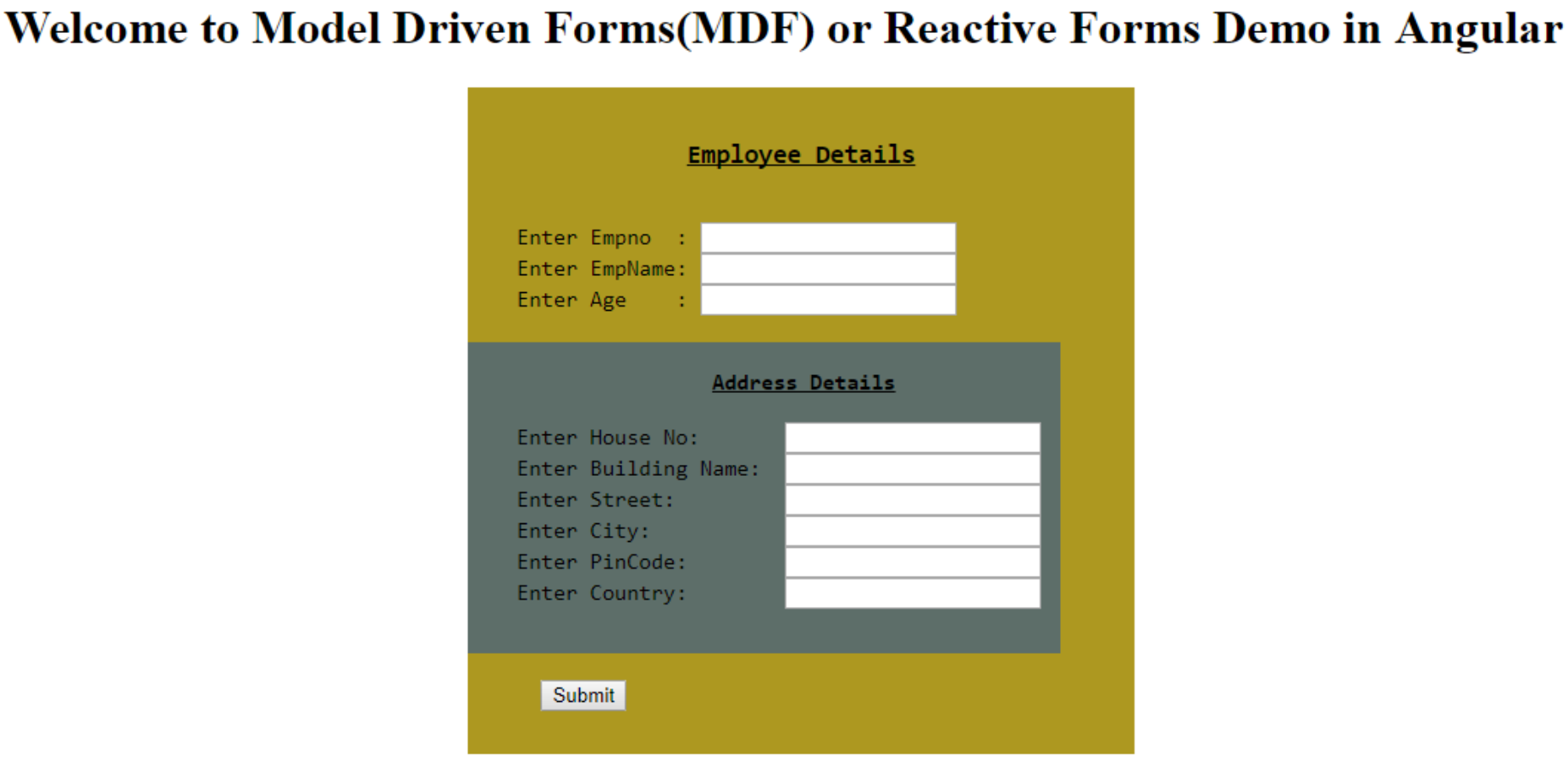
```
<h1 align="center">Welcome to Model Driven Forms(MDF) or Reactive Forms Demo in Angular
</h1>
<div align="center">
<div align="left" style="background-color:#AD9820 ; width:450px;height:450px;font-size:15px">
<form [formGroup]="EmployeeDetails" (ngSubmit)="DisplayOnConsole()" >
<pre>
<h3 align="center"><u>Employee Details</u>
<label>Enter Empno :</label> <input formControlName="empno">
<label>Enter EmpName:</label> <input formControlName="empname">
<label>Enter Age :</label> <input formControlName="age">
<div formGroupName="AddressDetails" style="background-color:#5D6E69 ;
width:400px;height:210px">
<b><u>Address Details</u></b><br>
<label>Enter House No:</label> <input type="text" formControlName="houseno" >
<label>Enter Building Name:</label> <input type="text" formControlName="buildingname">
<label>Enter Street:</label> <input type="text" formControlName="street">
<label>Enter City:</label> <input type="text" formControlName="city">
<label>Enter PinCode:</label> <input type="text" formControlName="pincode" >
<label>Enter Country:</label> <input type="text" formControlName="country" >
</div>
<input type="submit">
</pre>
</form>
</div></div>
```

Every Form contains this FormGroup. It is mapped to FormGroup Instance of Component Class

Every Form control of the html template is mapped to a FormControl Instance of the Component Class

FormGroupName attribute is for a FormGroup inside another FormGroup

Building a Model Driven Form (MDF) - Output



Applying Validation to Model Driven Forms(MDF)

Now, Let us apply required validation rule to Empname field and , required , maxlength and minlength validation rules to Street Field and Also Display appropriate error messages when validation rules are violated.

To Achieve this, Angular requires Validators Class

Applying Validation to Model Driven Forms(MDF)

```
import { Component } from '@angular/core';
import {FormGroup, FormControl, Validators } from '@angular/forms';
@Component({
selector: 'app-root',
templateUrl: './app.component.html'})
export class AppComponent {
public EmployeeDetails:FormGroup;
constructor(){
this.EmployeeDetails = new FormGroup({
empno: new FormControl(),
empname:new FormControl('',Validators.required),
age: new FormControl(),
AddressDetails: new FormGroup({
houseno: new FormControl(),
buildingname: new FormControl(),
street:new
FormControl('',[Validators.required,Validators.minLength(3),Validators.maxLength(7)]),
city:new FormControl(),
pincode:new FormControl(),
country:new FormControl(),
})});}
DisplayOnConsole():void {
console.log("::::::::::Employee Details::::::::::")
console.log(this.EmployeeDetails.value);
console.log("::::::End of Employee Details:::::::")
}}
```

Import Validators Class

Apply required, to ‘empname’ field

Apply required, minlength and maxlength to ‘street’ field

Applying Validation to Model Driven Forms(MDF)

app.component.html

```
<h1 align="center">Welcome to Model Driven Forms(MDF) or Reactive Forms Validation Demo in
Angular </h1>
<div align="center">
<div align="left" style="background-color:#AD9820 ; width:450px;height:450px;font-size:15px">
<form [formGroup]="EmployeeDetails" (ngSubmit)="DisplayOnConsole()" novalidate >

<h3 align="center"><u>Employee Details</u></h3>
<label>Enter Empno :</label> <input formControlName="empno"><br>
<label>Enter EmpName:</label> <input formControlName="empname"><br>
<div *ngIf="EmployeeDetails.controls['empname'].hasError('required')" style="color:red">
Name Cannot be Blank..
</div>

<label>Enter Age :</label> <input formControlName="age"><br>
```

Apply required to ‘empname’

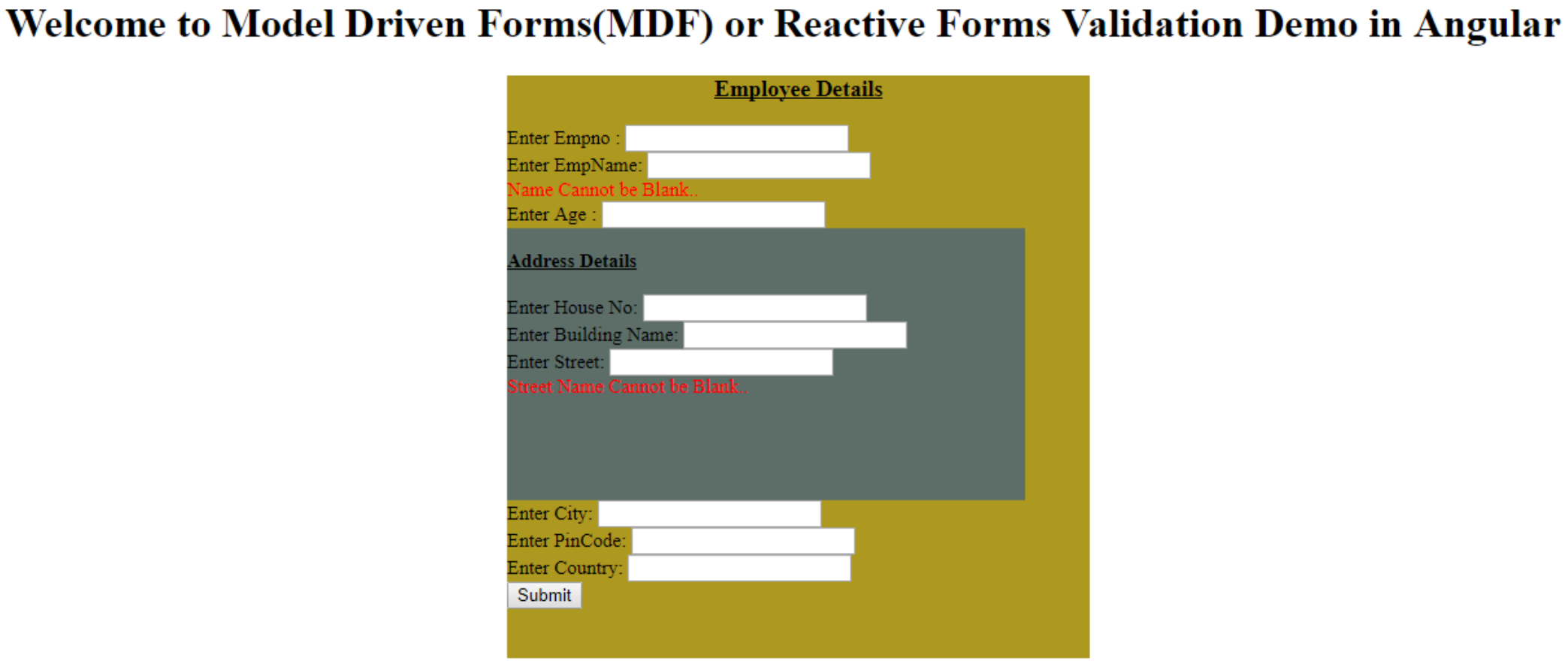
Applying Validation to Model Driven Forms(MDF)

app.component.html

```
<div formGroupName="AddressDetails" style="background-color:#5D6E69 ;
width:400px;height:210px"><br>
<b><u>Address Details</u></b><br><br>
<label>Enter House No:</label> <input type="text" formControlName="houseno" ><br>
<label>Enter Building Name:</label> <input type="text" formControlName="buildingname"><br>
<label>Enter Street:</label> <input type="text" formControlName="street"><br>
<div *ngIf="EmployeeDetails.controls['AddressDetails'].controls['street'].hasError('required')"
style="color:red">
Street Name Cannot be Blank.. </div>
<div *ngIf="EmployeeDetails.controls['AddressDetails'].controls['street'].hasError('minlength')"
style="color:red">
Minimum length is 3 Characters </div>
<div *ngIf="EmployeeDetails.controls['AddressDetails'].controls['street'].hasError('maxlength')"
style="color:red">
Maximum length is 7 Characters </div>
</div>
<label>Enter City:</label> <input type="text" formControlName="city"><br>
<label>Enter PinCode:</label> <input type="text" formControlName="pincode" ><br>
<label>Enter Country:</label> <input type="text" formControlName="country" ><br>
<input type="submit">
</form></div></div>
```

Apply required, minlength and maxlength to ‘street’ field

Applying Validation to Model Driven Forms(MDF) - Output



MODULE SUMMARY

Now, you should be able to:

- Understand the types of Forms in Angular
- Create a Template driven form and do validations
- Create a Model driven form and do validations

QUESTIONS

THANK YOU