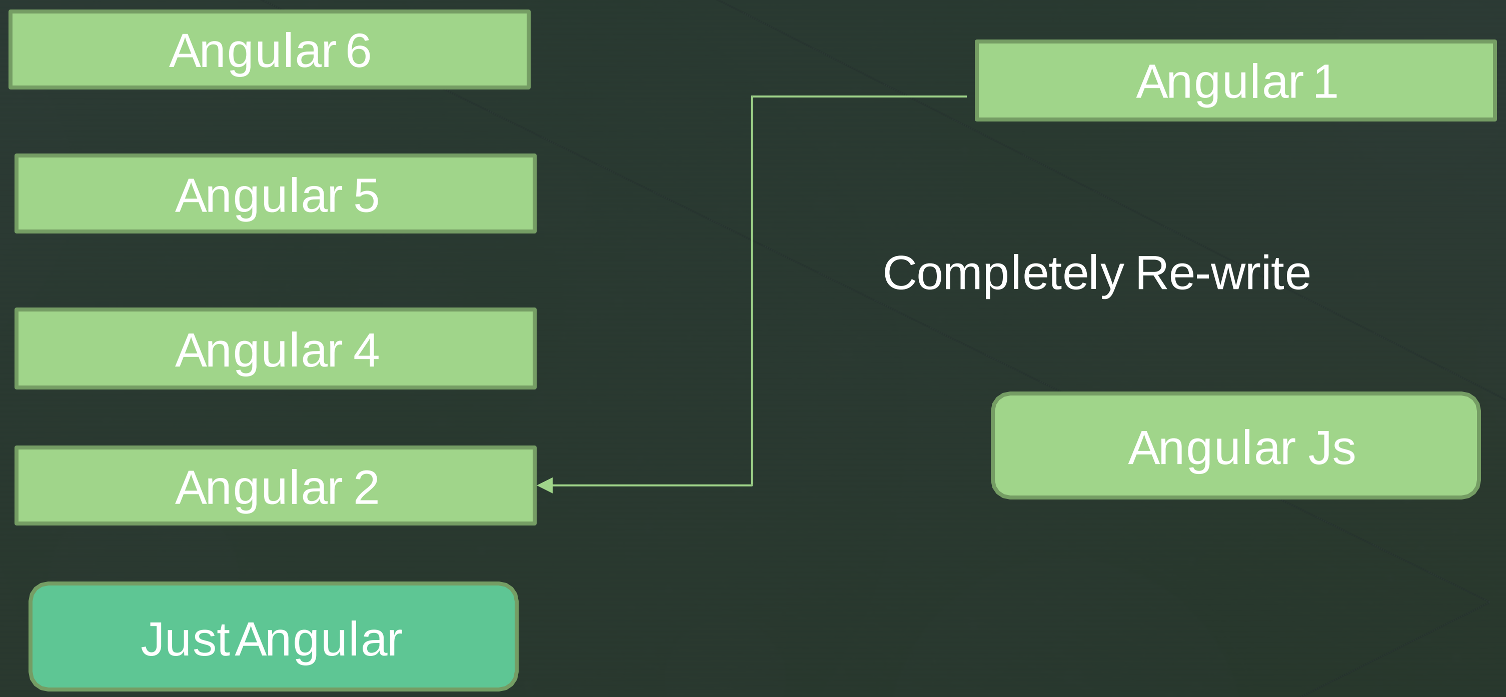
Angular
Course Introduction

What is Angular?

- Angular is a JavaScript Framework which allows you to create reactive Single-Page-Applications (SPA s).

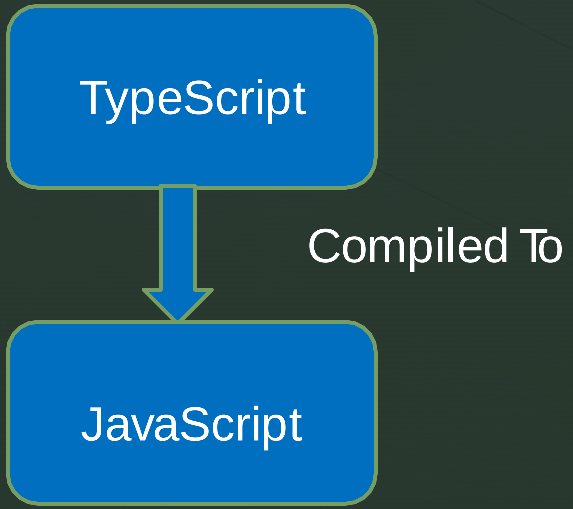
Angular 6 vs Angular 5 vs Angular2vs Angular1



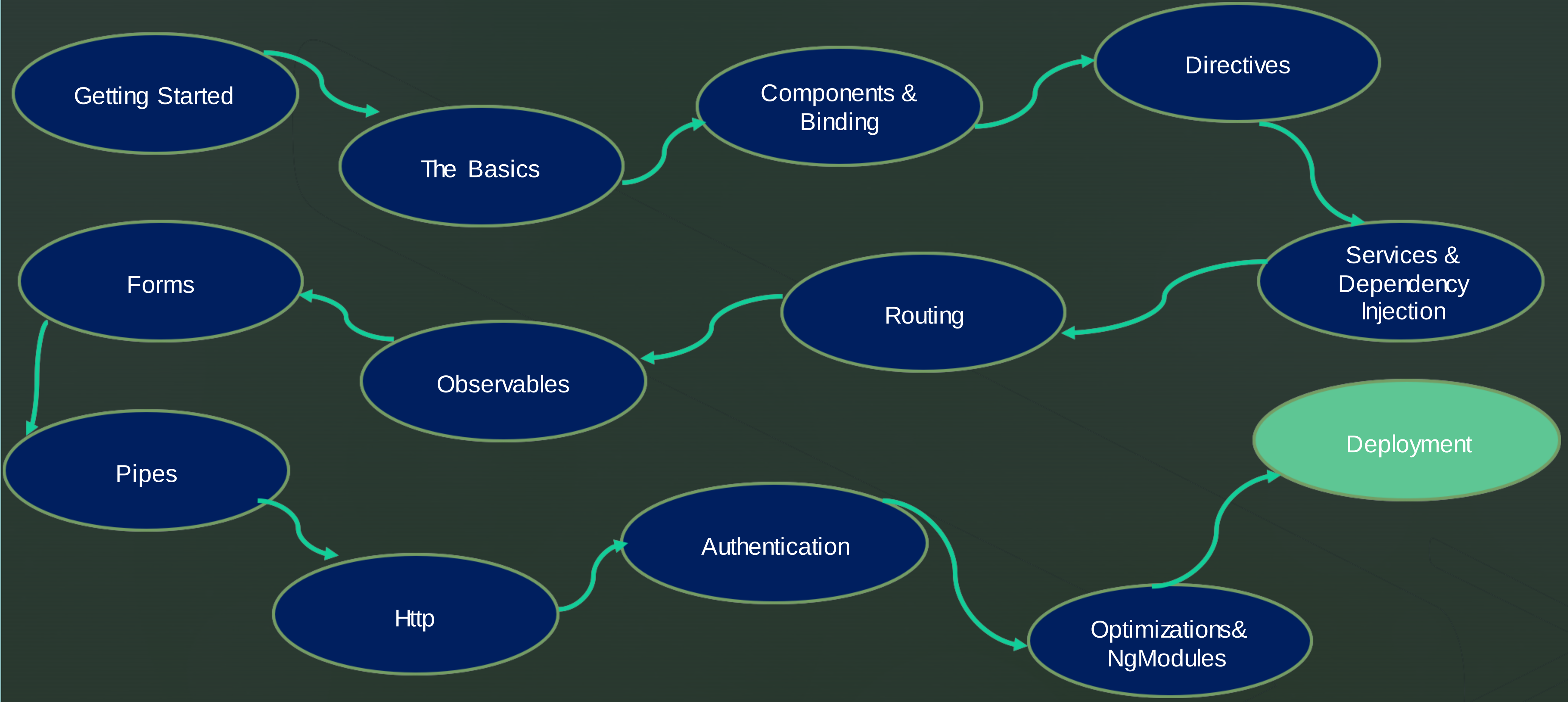
z

Type Script

- TypeScript is a superset of JavaScript.
- Define in TypeScript if a certain variable is a number, a string.
- TypeScript doesn't run in the browser, It is compiled to JavaScript in the end. This Compilation is handled by CLI. This compilation is really fast and in the end, in the browser JavaScript is going to run.



Course Structure



z

Course Overview

- Getting Started : Built and Edit Angular Application
- Basics : Components, Two way binding and how does that all work?
- Components & Data Binding: Output Data to display in DOM and also react to user events.
- Directives: Angular has another feature, directives. ngModel, which we used for two-way data binding and also built own custom directives
- Services& Dependency Injection: A core feature of Angular which we makes it really easy to have different pieces in app to communicate with each other, to centralize code, to manage the state of your application.
- Routing: Routing basically means navigation between pages. Its look like we are switching pages even though technically, but still remain on that single page.
- Observables :It is a concept to allow with asynchronous code.

z

Course Overview

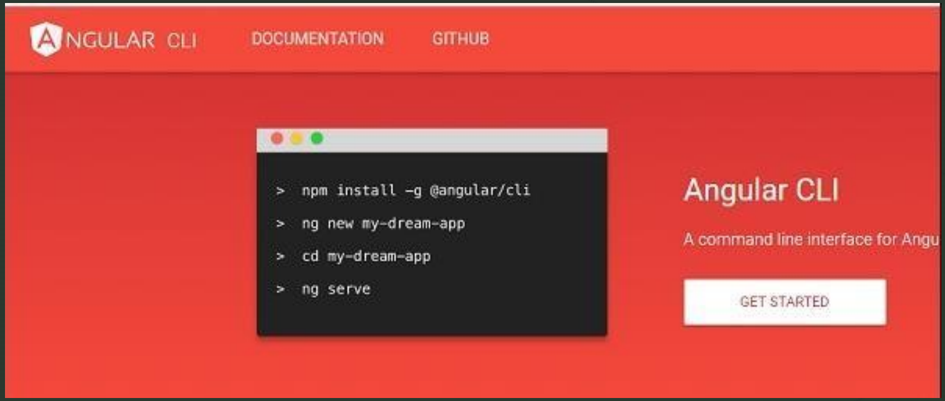
- Forms: Handling Forms
- Pipes: The | character is used to transform data.to display the on the template at run time.
- Http: What if you need to reach out to a web server?. Angular cant connect to a database directly, but it can connect to a server which is able to do in HTTP section.
- Authentication: How to implement authentication is Angular application?
- Optimizations & ng Modules – To Manage different modules in application.
- Deployment –How can we get Angular application from local machine to a place in the internet.

Environment & Project Setup

- Environment Setup required for Angular 4. To install Angular 4, we require the following −
  - Nodejs
  - Npm
  - Angular CLI
  - IDE for writing code (Visual Studio Code)
- AngularJS is based on the model view controller, whereas Angular 2 is based on the components structure. Angular 4 works on the same structure as Angular2 but is faster when compared to Angular2.

Angular Official Website : https://angular.io/

For Project Creation:



```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Angular CLI

A command line interface for Angu

z

Basic Project Setup with Bootstrap styling

- Create Project : ng new [project-name]
- >cd [project-name]
- >ng server – It will serve our application on http://localhost:4200 in default
- To install bootstrap – In CLI, npm install bootstrap@latest --save

    -- save – It will maintain the project version at package.json

To Import Bootstrap in angular application:

Open styles.css add the following line:

```
@import "~bootstrap/dist/css/bootstrap.min.css";
```

Components

Components are basically classes that interact with the .html file of the component, which gets displayed on the browser.

App Component is the root component which holds our entire application. So We can nested our other component inside the app component.

Each component has its own styling and own template.

It allows to split up complex web page into reusuable parts.

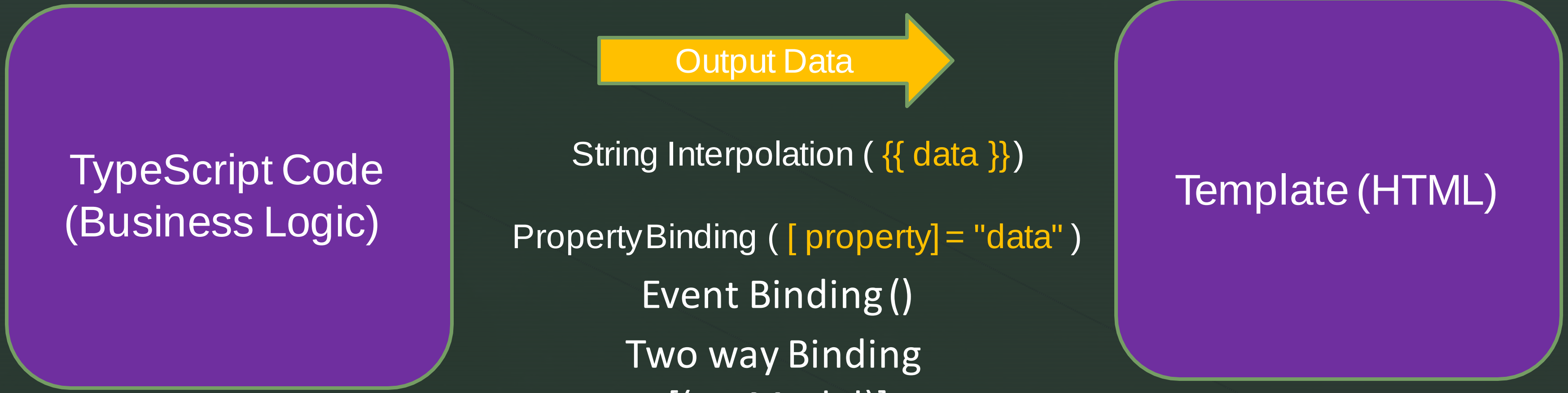
z

Create a New Component

- >ng generate component [your-component-name]
- Or
- >ng g c [your-component-name]
- Or
- You can manually create your own component and add it in app component.

Data Binding

Databinding = communication



String Interpolation ({{ data }})

Property Binding ([property] = "data")

Event Binding ()

Two way Binding
[(ng Model)]

Basically translate data binding with communication communication between types of code of your component your business logic and the template.

Data Binding

String InterPolation:

- use curly braces for data binding - {{}}; this process is called interpolation.

Property Binding:

- This allows you to bind values to properties of an element to modify their behavior or appearance.- [ ]

Event Binding:

Event bindings listen for DOM events such as `click` and `keypress` on standard HTML elements such as `button` and `input`. - ()

Two Way Binding:

Two-way data binding explains how the view updates when the model changes **and vice-versa**. This happens immediately which means that the model and view are always in-sync.

A.String Interpolation:

HTML Template:

```
<p>   {{ title }}  </p>
```

TS file:

```
export class MyApp {
  title: string = 'Welcome';
}
```

B.Property Binding:

HTML Template:

```
<button (click)="test = !test">{{test ? 'hide' : 'show'}}</button>
```

TS File

```
export class Myapp {

test: boolean = false;
}
```

C. Event Binding:

Html Template:

```
<h1>{{firstname}}</h1>
Name: <input type="text" [(ngModel)]="firstname">
<button (click)="changeName()">Change Name</button>
```

Ts file:

```
export class MyApp {
  firstname: string = 'Jimmy';

  changeName() {
    this.firstname = 'Houssein';
  }
}
```

D. Two Way Binding:

HTML Template:

```
<h1>{{firstname}}</h1>
Name: <input type="text" [(ngModel)]="firstname">
```

Ts File:

```
export class MyApp {
  firstname: string = 'Jimmy';
}
```

*ngIf

*ng If :

ngIf directive allows us to simply toggle content based on a conditions.

<u>HTML templates:</u>

```
<div *ngIf="isLoggedIn"> Welcome back. Congrats. </div>
```

<u>Ts File</u>

```
export class AppComponent {
        isLoggedIn = true;
}
```

ng-For

ngFor :

Angular **ngFor** is a built-in Directive that allows us to **iterate** over a collection. This collection is typically an array,

HTML Template:

```
<div *ngFor = "let item of items;">
 <div> {{item.name}}  </div>
</div>
```

TS File:

```
items=[
  {
      id:1,
      name: "Jennifer Cohen"
  },
  {
       id:2,
       name: "alex"
   }
];
```

*ng-template

`<ng-template>` is an angular element for rendering HTML.

HTML Template:

```
<div class="lessons-list" *ngIf="lessons else loading">
...
</div>
<ng-template #loading>
<div>Loading...</div>
</ng-template>
```

ngStyle and ngclass

NgStyle and NgClass directives can be used to conditionally set the look and feel of your application.

```
<div [ngStyle]="{'background-color':'green'}"></<div>
This sets the background color of the div to green.
```

```
<div [ngStyle]="{'background-color':person.country === 'UK' ? 'green' : 'red'
}"></<div>
```

The above code uses the ternary operator to set the background color to green if the persons country is the UK else red.

ngClass

NgClass:

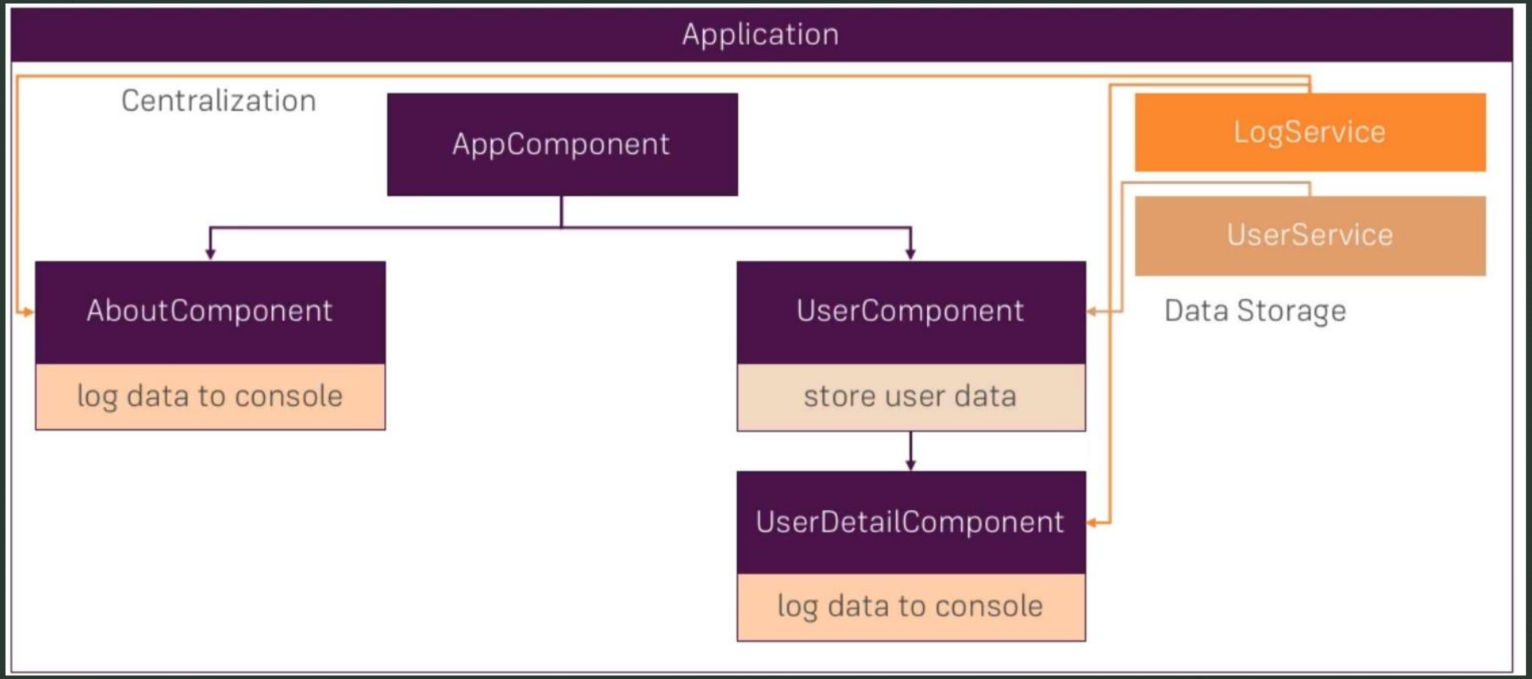
The NgClass directive allows you to set the CSS class dynamically for a DOM element.

```
<button [ngClass]="calculateClasses()"> Btn </button>
```

```
calculateClasses() {
return {
btn: true,
'btn-primary': true};
}
```

Angular Service & Dependency Injection

An angular service is simply a function that allows you to access its' defined properties and methods. It also helps keep your coding organized.



Creating Service

The Angular CLI allows you to quickly generate a service. It takes care of some of the manual labor involved with service creation. To create a service, at the console in the project folder type:

```
> ng g service data
```

Upon running this, your output may look something like:

```
installing service
  create src\app\data.service.spec.ts
  create src\app\data.service.ts
```

Importing Service:

```
import { DataService } from './data.service';

@NgModule({
  // Other properties removed
  providers: [DataService],

})
```

Working with the Service File

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

@Injectable()
export class DataService {

  constructor() { }

  cars = [
    'Ford','Chevrolet','Buick'
  ];


  myData() {
    return 'This is my data, man!';
  }

}
```

Using Services in Components

The first step requires importing the service at the top of the component. So, in **app.component.ts**:

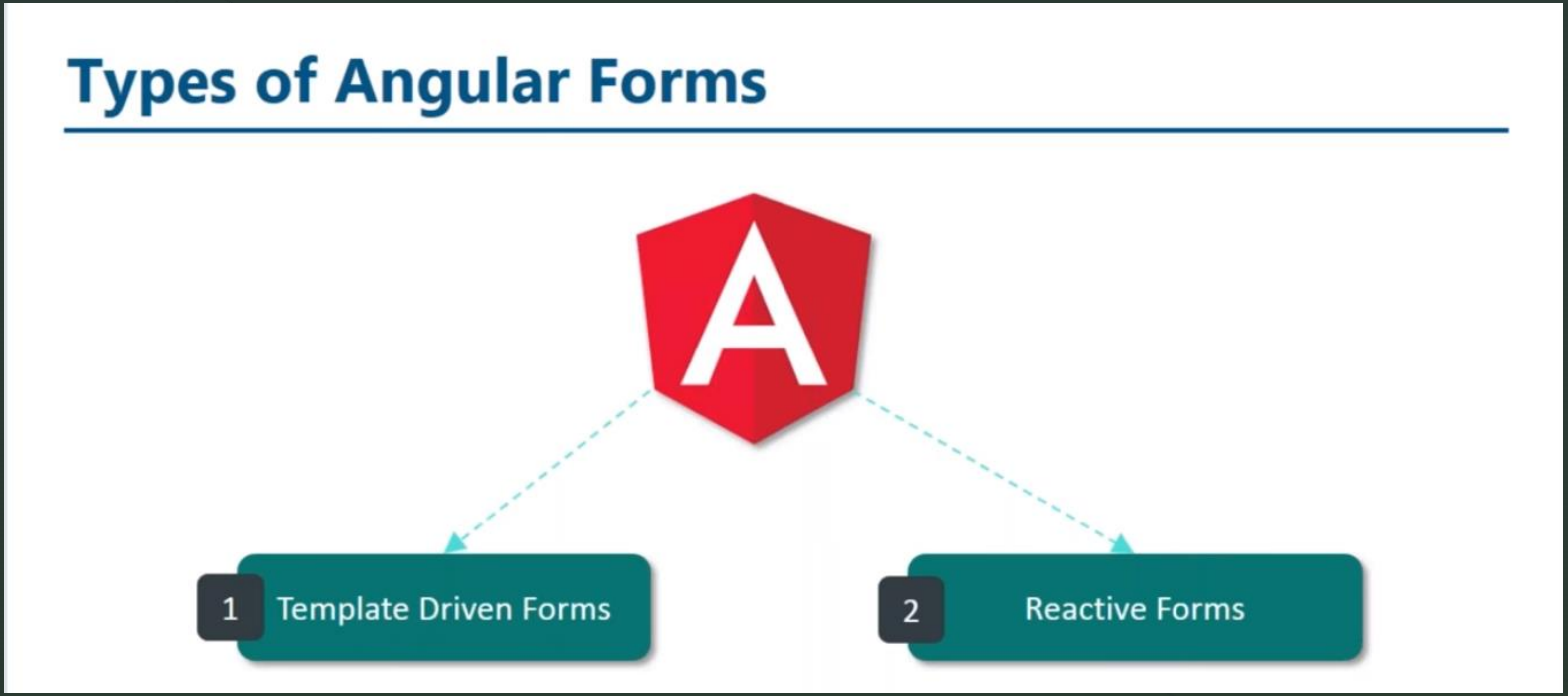
```
import { DataService } from './data.service';
```

Next, within the constructor, we have to import it through dependency injection:

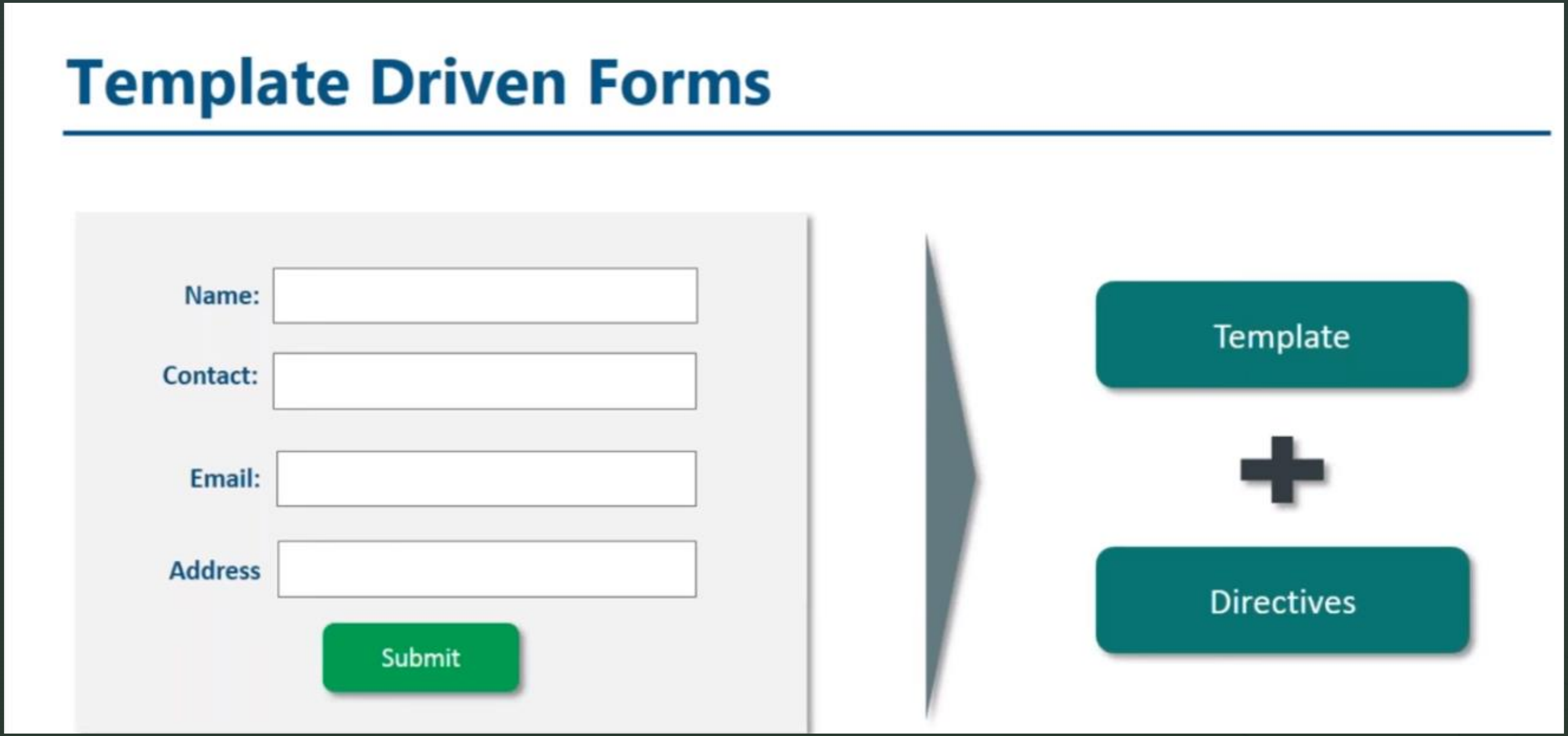
```
export class AppComponent {

  constructor(private dataService:DataService) {

  }

}
```

Now we can use **dataService** to access its's associated properties and methods.

Angular Forms



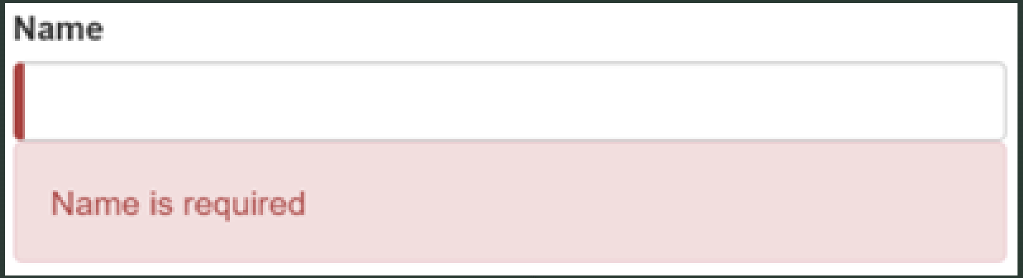
Template Driven Forms



Template Driven Forms

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

import { Hero } from '../hero';

@Component({
          selector: 'app-hero-form',
          templateUrl: './hero-form.component.html',
          styleUrls: ['./hero-form.component.css'] })

export class HeroFormComponent {

         powers = ['Really Smart', 'Super Flexible', 'Super Hot', 'Weather Changer'];
         model = new Hero(18, 'Dr IQ', this.powers[0], 'Chuck Overstreet');

         submitted = false; onSubmit() {
               this.submitted = true;
        } // TODO: Remove this when we're done

        get diagnostic() {

           return JSON.stringify(this.model);

       }
 }
```

Validation Part


Name

Name is required

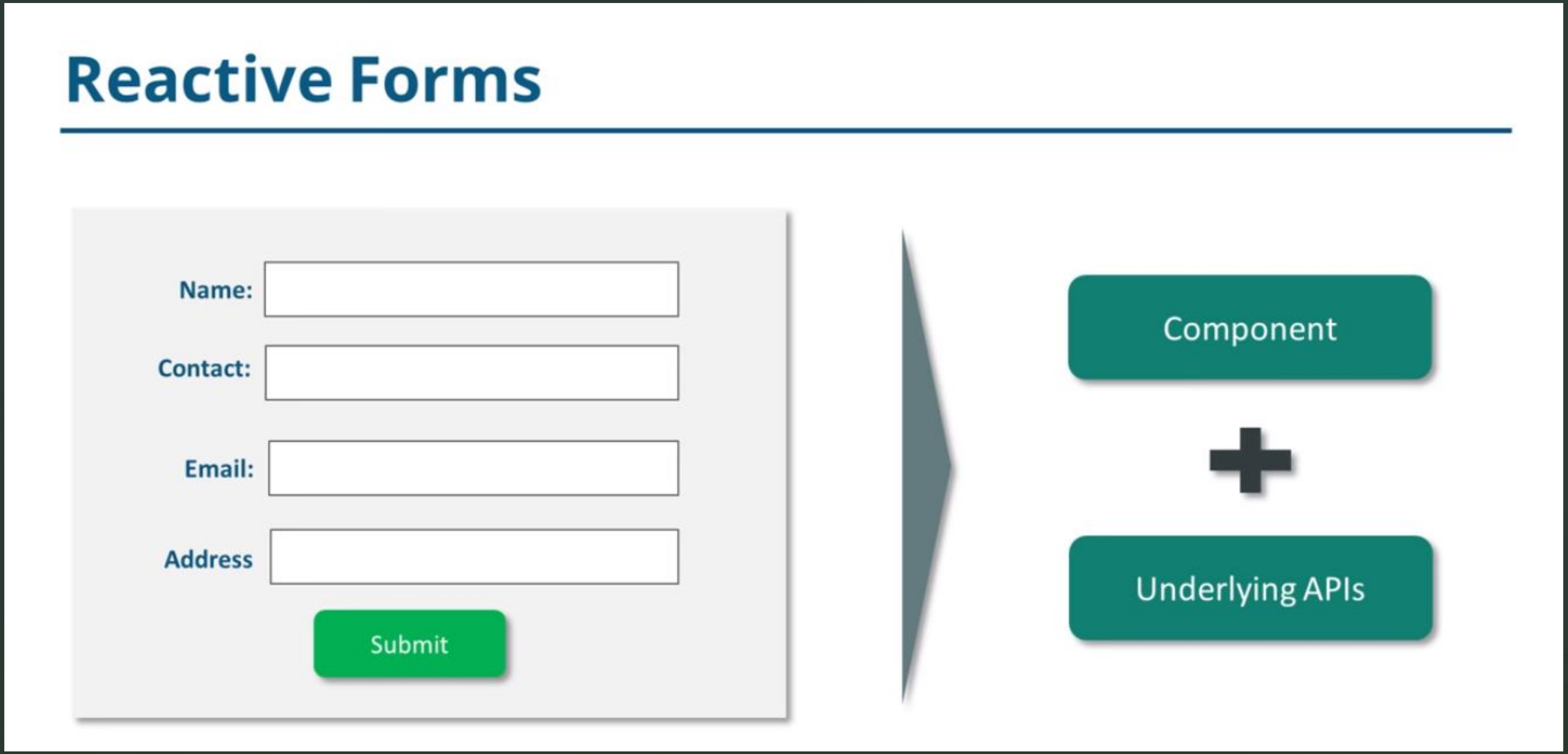
```
<label for="name">Name</label>
<input type="text"  class="form-control"  id="name" required [(ngModel)]="model.name"  name="name"  #name="ngModel">
<div [hidden]="name.valid  || name.pristine"  class="alert alert-danger"> Name is required </div>
```

Pattern Validation

```
<form  #frm="ngForm"   (ngSubmit)="fn(frm)">
    <label> Name </label>
<input
       type=" text"
       name = "name"
        ngModel
        required
         pattern = "[a-zA-Z][a-zA-Z]+"
```

Use Regex Pattern based on the field

Reactive Forms



Reactive Forms

Angular *reactive* forms facilitate a *reactive style* of programming that favors explicit management of the data flowing between a non-UI *data model* (typically retrieved from a server) and a UI-oriented *form model* that retains the states and values of the HTML controls on the screen. Reactive forms offer the ease of using reactive patterns, testing, and validation.

<u>Import Reactive Forms:</u>

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

<u>Import FormGroup and FormControl</u>

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

```
<form [formGroup]="angularForm"  >
    <div class="form-group">
       <label class="center-block">Name:
         <input class="form-control" formControlName="name">  </label>
     </div>
</form>
```

Reactive Forms validators

Insert Validation into the field.

```
// app.component.ts

 import { FormControl, FormGroup, FormBuilder, Validators } from '@angular/forms';


<form [formGroup]="angularForm"  novalidate>
          <div class="form-group">
                 <label>Name:</label>
                         <input class="form-control" formControlName="name" type="text">
           </div>
          <div *ngIf="angularForm.controls['name'].invalid  && (angularForm.controls['name'].dirty  ||
angularForm.controls['name'].touched)"                    class="alert alert-danger">
                  <div *ngIf="angularForm.controls['name'].errors.required">
                           Name is required.
                   </div>
           </div>
          <button type="submit" [disabled]="angularForm.pristine  || angularForm.invalid"  class="btn btn-success">
           Save
         </button>
 </form>
```

Reactive Form Pattern checking

app.component.ts

```
CreateForm() {
     this.angularForm  = this.fb.group({
                name: ['', Validators.required ],
        });
}

ngOnInit(){
    this.form  = new FormGroup  ({
                name :   new Formcontrol  (' ' ,[Validators.required  ,validators.pattern('a-ZA-Z][a-zA-Z]+')])
});
```