Caltech | Center for Technology & Management Education

Full Stack Java Developer

simplilearn

Angular

Angular Components

Learning Objectives

By the end of this lesson, you will be able to:

- Understand components, decorators and core directives, and forms in Angular
- Learn Angular components and how to build them
- Describe Events and Events Emitter
- Explain how component communication and core directives work
- Illustrate forms and built-in pipes

A Day in the Life of a Full Stack Developer

ABC is an organization that creates E-commerce websites. You have been assigned the project and have been asked to develop this website. The website consists of multiple components, events, forms, and much more.

To develop this, you need to understand components, decorators and core directives, and forms in Angular. You also need to work with Events and Events Emitter.

TECHNOLOGY

Angular Components

Angular Components

Components are the building blocks of any Angular application.

1 They include a definition of the view and the data that describes how the view looks and works.

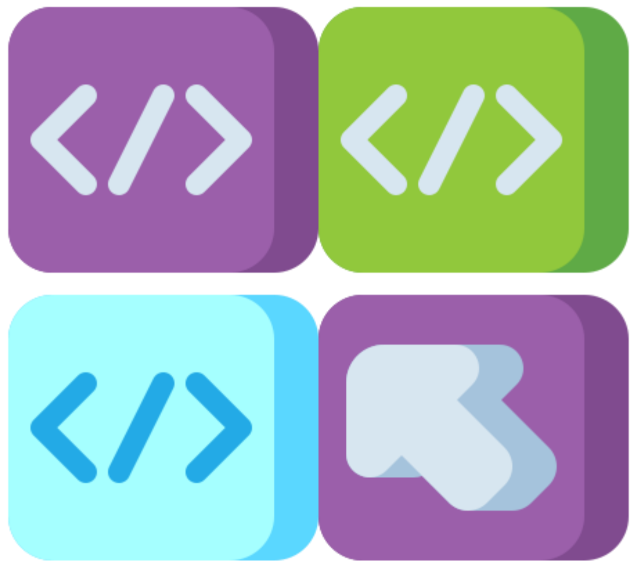
2 They are defined using @component decorator.

3 They help to complete the user interface of an application.

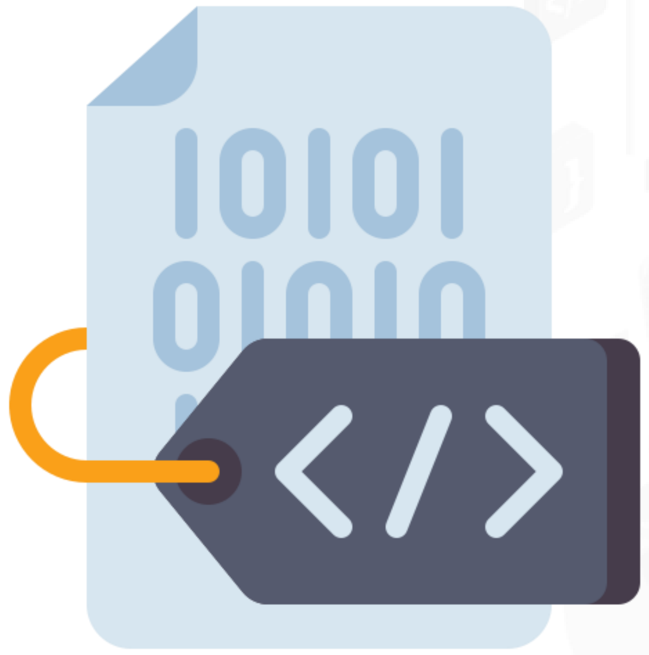
Angular Components

Components are the building blocks of any Angular application.

Template



Class



MetaData

Angular Components: Template

A template is a form of HTML that tells Angular how to render the component.

Template

Angular Template syntax

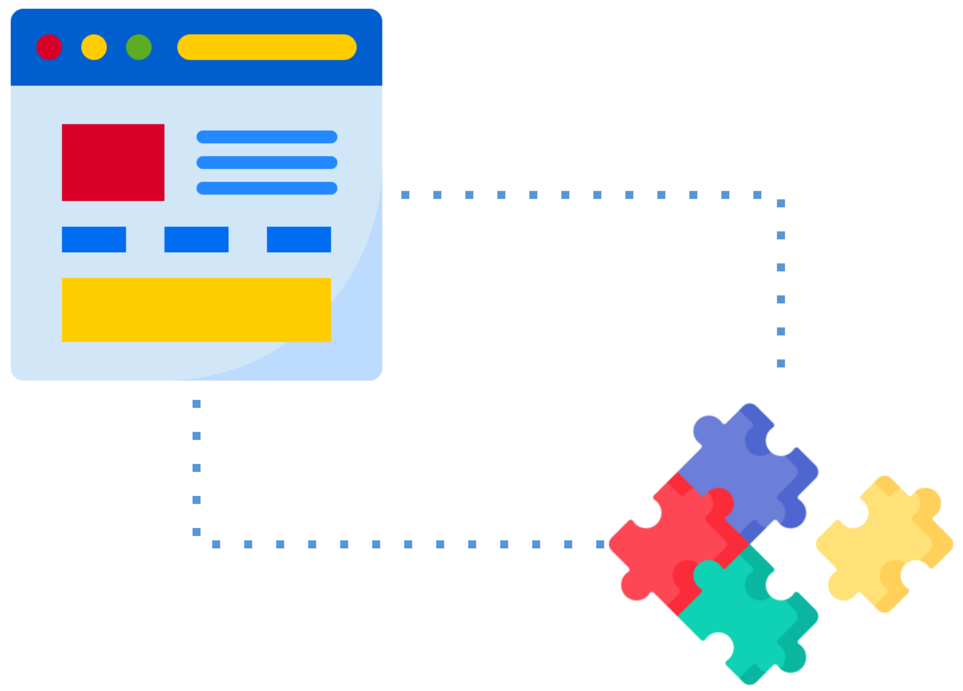
Angular Components

The following can be added to the template:

1. Angular directives
2. Angular pipes
3. Additional Angular components

Angular Components

The data to the template proceeds from the component, which in turn prepares it from an Angular Service.



Template is synced with component using data binding methods.

Template use Event Binding inform the component when a user changes the view.

Angular Components

In Angular, templates are listed in two ways:

Define the template inline

Provide an external template

Angular Components: Class

A class helps to implement data and logic to the view.

1. It contains a JavaScript code associated with the template.
2. It is created using TypeScript.
3. It contains Properties and Methods. Properties are attached to view using Data Binding.

Angular Components: Class

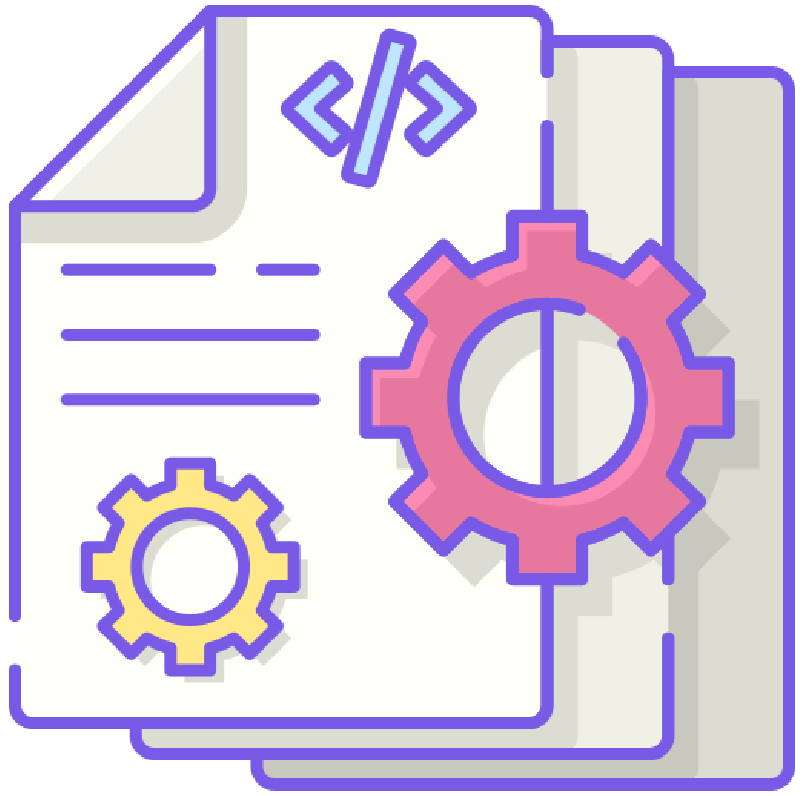
Simple Angular class : Example

```
export class AppComponent
{
      title : string = "app"
```

By practice, the prefix Component class is added with the component to clearly distinguish them.

Angular Components: Metadata

Metadata provides additional information about the component of Angular.

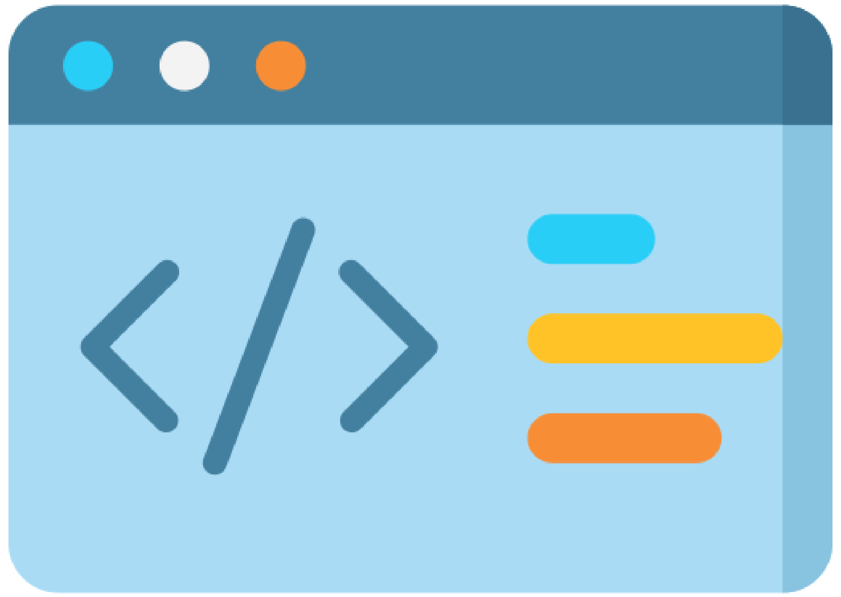


@Component decorator is used to providing metadata to the component.

Angular Components: Metadata

A decorator is a function which adds metadata to a class, its methods and its properties.

Components are marked with a @Component class decorator.



Angular treats the class as a component if it is marked with @Component decorator.

Angular Components: Metadata

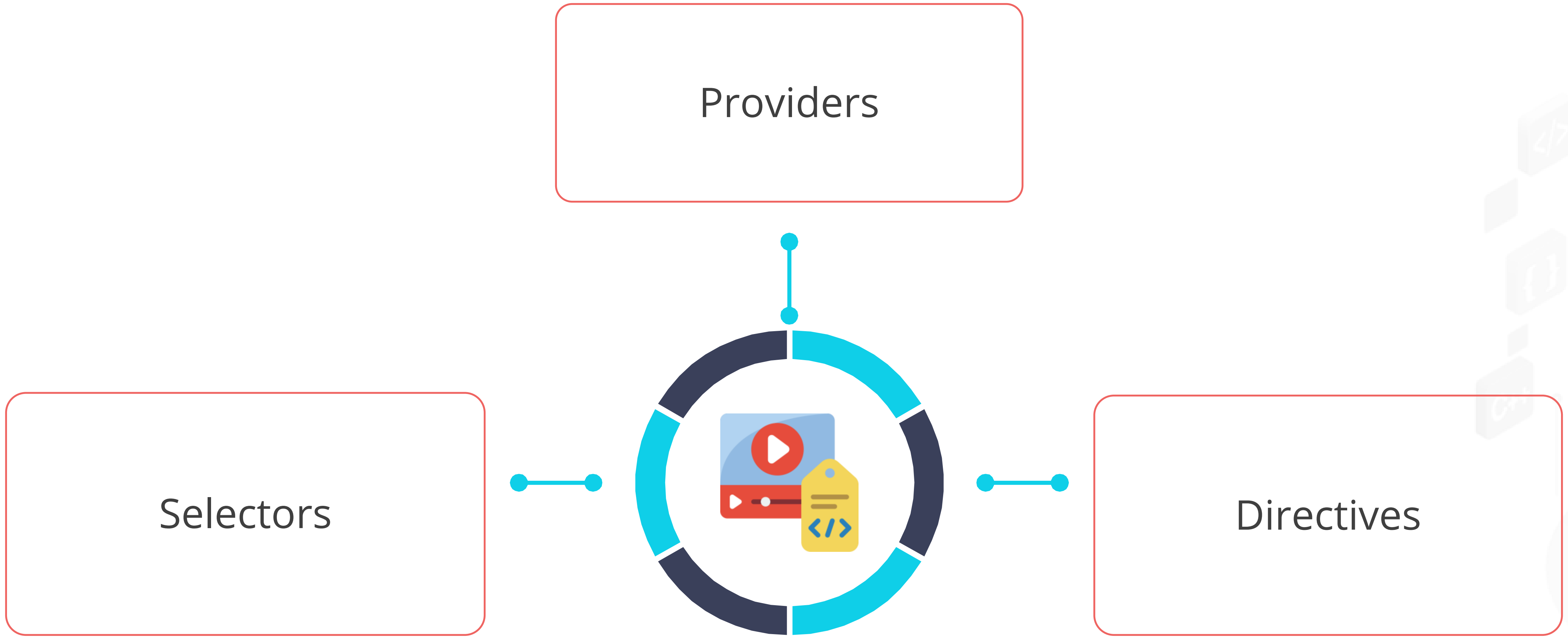
Characteristics of a decorator are:

Angular Decorator

- It is prefixed with @.
- It is placed before the class.
- It can be built as per user needs.
- It has similar attributes to C#.

Metadata Properties

Important properties of Component metadata are:



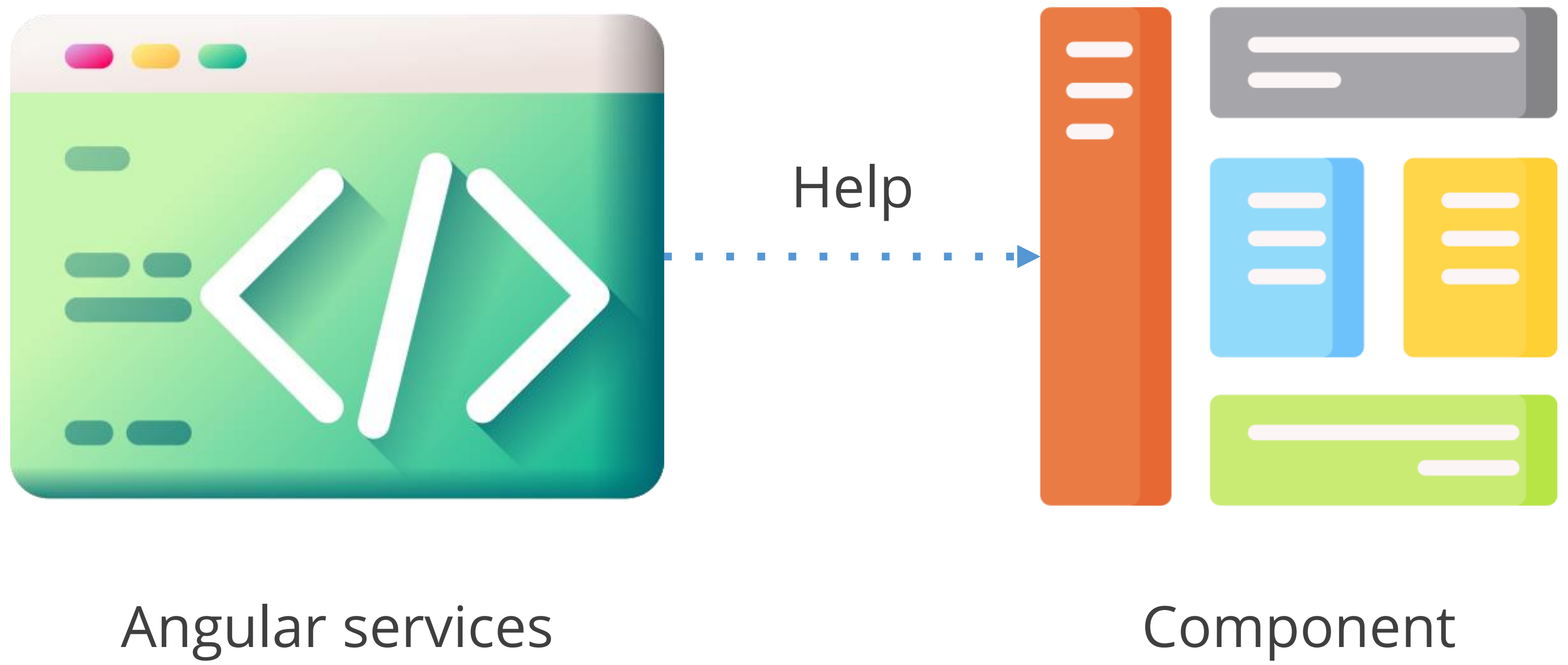
Metadata Properties: Selectors

Selectors in metadata specify the simple CSS selector.

Angular views for the CSS selector and renders the component in CSS selectors.

Metadata Properties: Providers

Providers are the Angular Services that the Component will use.



Metadata Properties: Directives

The directives used in components are:

Styles/StyleUrls

- It refers to the CSS styles.
- It employs both external stylesheets using styleUrls or inline styles using Styles.

Template/ TemplateUrl

- It directs Angular on how to render the Component's view.
- It utilizes an external template using a templateUrl.

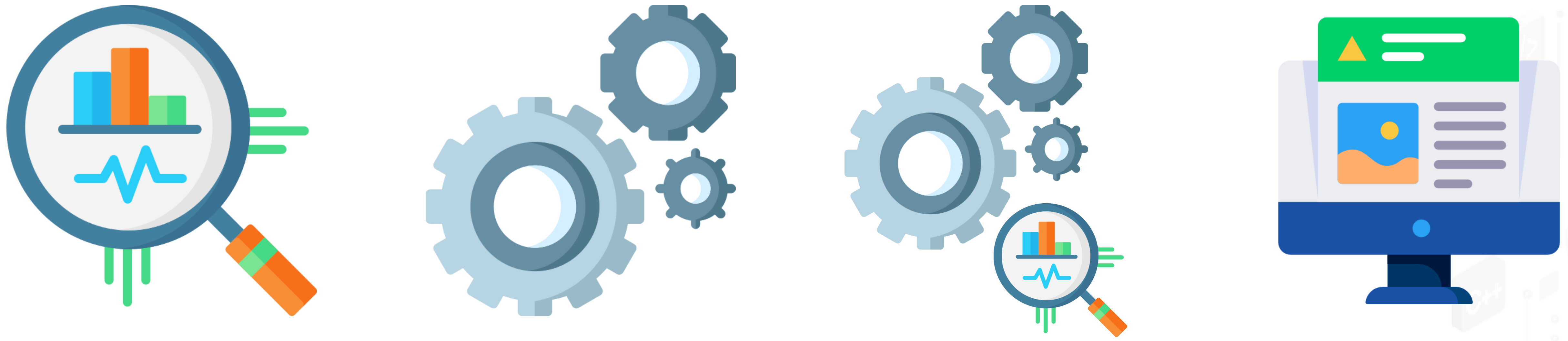
The Component can have only one template, hence either an inline template or an external template can be used, but not both.

TECHNOLOGY

Data Binding

Data Binding

In data binding, the statistics remain in sync along with the factors and the view.



Statistics are updated

Angular updates the factor

Factor receives new statistics

Angular updates the view

Data Binding

Uses of Data or Statistics binding are:

Display model to the user

Dynamically change detail

Style

Reply

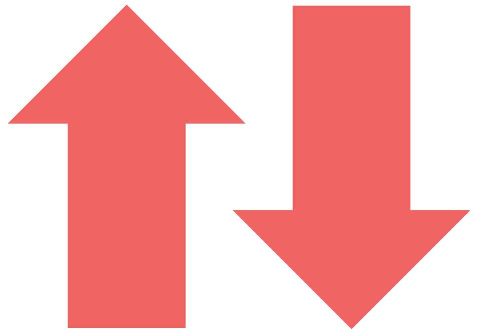
Events

Data Binding

The statistics binding in Angular can be categorized into two groups:

One-way binding

Two-way binding

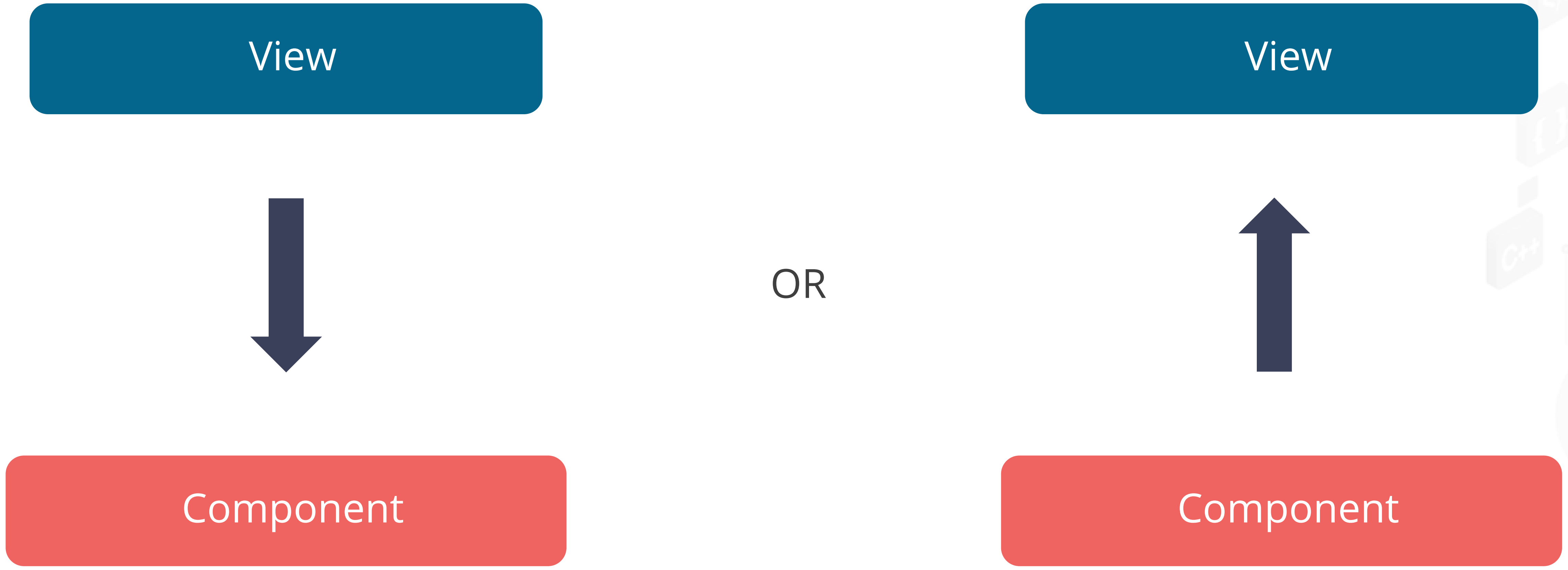


TECHNOLOGY

One-Way Binding

One-Way Binding

In one-way binding, statistics flow from one direction.



One-Way Binding: Interpolation Binding

Interpolation allows having expressions as a part of any string literal that can be used in HTML.

One-Way Binding: Interpolation Binding

Angular makes use of the (double curly braces) inside the template to indicate the interpolation and it.

- Evaluates the expressions right into a string
- Replaces it with the automatic string
- Updates the view

{{ }}

Interpolation can also be used at a place where the user wants to operate a string literal inside the view.

One-Way Binding: Interpolation Binding

Template Expression is the content material in the double braces.

Syntax:

```
{{ Template Expression }}
```

Angular evaluates a template expression

↓

Converts it into a string

↓

Replaces the template expression with an authentic string

↓

Updates the authentic string again

One-Way Binding: Interpolation Binding

Interpolation Binding: Example

Syntax:

```
Welcome, {{firstName}} {{lastName}}
import { Component ) from ‘ @angular/core’ ;
@Component ( {
      selector: 'app—root’,
      templateUr1 : ‘./app.component.html’,
      styleUr1s: [‘./app.component.css’]
})
export class AppComponent {
      firstName=Jack;
      lastNamr=“Martin”
```

One-Way Binding: Interpolation Binding

Interpolation Binding: Example

Syntax:

```
Welcome, Jack Martin
```

Angular replaces both {{firstName}} & {{lastName}} with the values of firstName and lastName variables from the issue.

Angular updates the view as and when the values of the firstName and lastName change.

One-Way Binding: Property Binding

Property binding permits the user to bind HTML element property to the property inside the component.

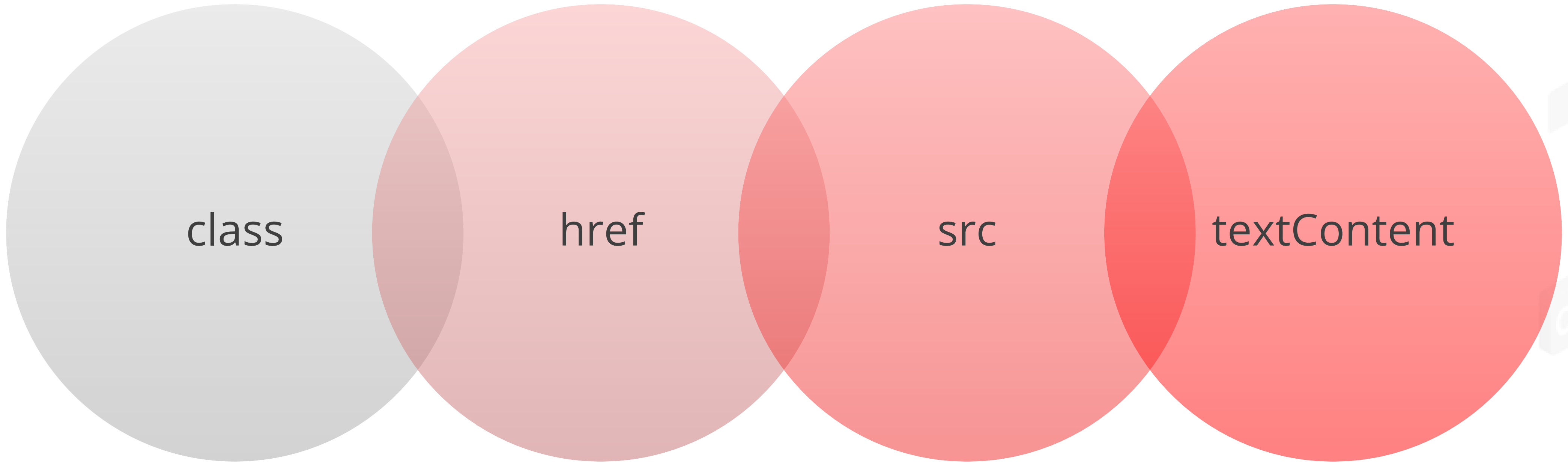
Angular waits for the value of the component to change

$\downarrow$

Angular updates the element property

One-Way Binding: Property Binding

Properties include:



One-Way Binding: Property Binding

The Property Binding makes use of the subsequent syntax as shown:

Syntax:

```
[ binding-target ] = '@binding-supply';
```

The Binding goal (or goal belongings) is enclosed in a rectangular bracket [].

One-Way Binding

Binding-supply is enclosed in value and can assign to the binding goal.

Syntax:

```
[binding-target] = "binding-supply"
```

Belongings inside the issue

Technique in issue

Template reference variable

Expression

Whenever the value of the binding supply changes, Angular updates the view.

One-Way Binding: Property Binding

Syntax:

```
app.component.html
<h1 [innerText]=“title”></h1>
<h2>First Example</h2>
<button [disabled]="isdisabled</button>
app.component.ts
import { Component } from '@angular/core'
@Component ( {
        selector: ‘app-root’,
        templateUrl: ‘./app.component.html’,
        styleUrls: [‘./app.component.css’]
})
export class AppComponent {
        title=“Angular Property Binding Example”

//First Example
isDisabled= true ;
```

One-Way Binding: Class Binding

Class binding binds the data from the component to the HTML class property.

Data from component
HTML class property

Syntax:

```
<HTMLTag [class]=“class name holds component variable”>
```

One-Way Binding: Style Binding

Style binding binds the data from the component to the HTML style property.

Syntax:

```
<HTMLTag [style.STYLE]=“component data”>
```

One-Way Binding: Attribute Binding

Attribute binding is used when there are no HTML detail belongings to bind to.

The attribute syntax begins with attr followed by a dot and ends with the name of the attribute.

Syntax:

```
<button [attr.aria-label]="Label"
(oneclick)="closed()">X</button>
```

Angular Components: Event Binding

Event binding is used for the view to component and helps to bind events, such as:

- Keystrokes
- Clicks
- Hower
- Touch

Syntax:

```
<target-event> = "TemplateStatement"
```

Angular Components: Event Binding

Example :

```
<button  (click)=“onHover()”>Hower</button>
```

Whenever the user clicks on the button, Angular invokes the onHover() method.

Two-Way Binding

Two-Way Binding

In two-way binding, adjustments made to the version inside the aspect are propagated to the view.

Any adjustments made in the view are updated in the underlying aspect.

Two-Way Binding

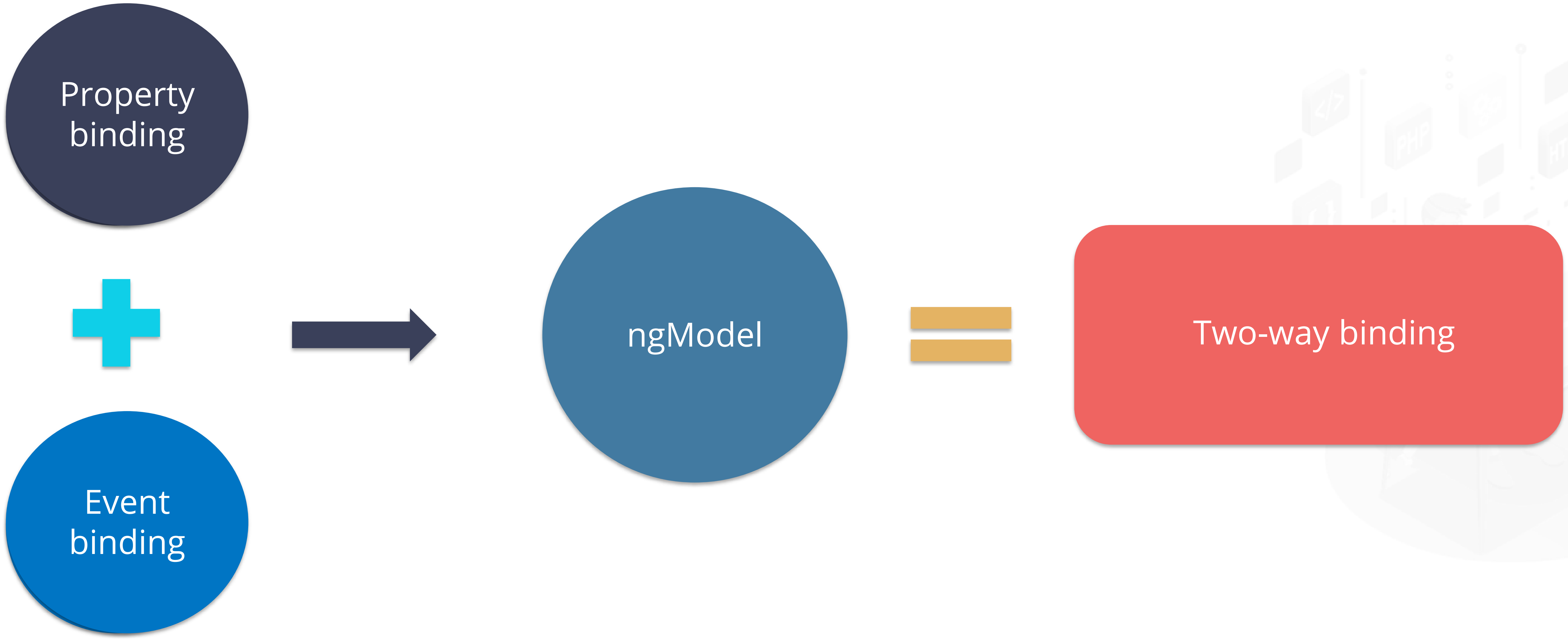
Two-way binding is beneficial in information access forms.

Whenever a user makes adjustments to a shape field, the version also needs to be replaced.

Similarly, if the version is replaced with new information, then the view is replaced as well.

Two-Way Binding

Two-way binding makes use of the unique syntax called a banana in a box [()].



Angular binds them to a form detail like input, select, select area, and more.

Two-Way Binding

The ngModel directive is a part of the @angular/forms.

Import the FormsModule bundle into the Angular module.

Syntax:

```
import from '@angular/forms';
```

TECHNOLOGY

Build Your First Angular Component

Build Your First Angular Component

The whole Angular application is developed by using various components.

Components make the application into reusable parts.

1. Create a new component:

```
ng generate component component_name
Or
ng g component_name
```

Build Your First Angular Component

A new component contains four files:

- component_name.component.ts
- component_name.component.html
- component_name.component.spec.ts
- component_name.component.ts.css

Build Your First Angular Component

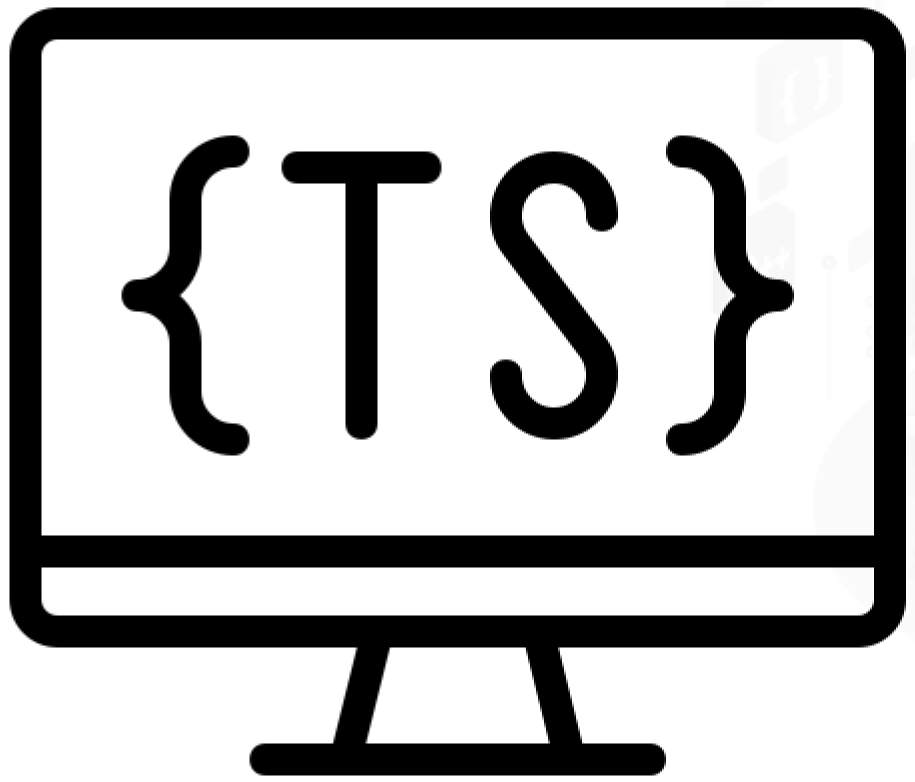
Component_name.component.ts

Component_name.component.html

Component_name.component.spec.ts

Component_name.component.ts.css

It contains all TypeScript code.



Build Your First Angular Component

Component_name.component.ts

Component_name.component.html

Component_name.component.spec.ts

Component_name.component.ts.css

It contains all HTML code.

Build Your First Angular Component

Component_name.component.ts

Component_name.component.html

Component_name.component.spec.ts

Component_name.component.ts.css

It contains all stylesheets for the same component.

Build Your First Angular Component

Component_name.component.ts

Component_name.component.html

Component_name.component.spec.ts

Component_name.component.ts.css

It is used for testing.

Build Your First Angular Component: Steps

Step 1 Open the terminal

Step 2 Navigate to the folder where the project has to be created

Step 3 Write the command: "ng new app_name"

Build Your First Angular Component: Steps

The command helps create the project with the following files:

src folder

App folder

app.component.css

app.component.html

app.component.spec.ts

app.component.module.ts

The folder contains the main code files associated with the Angular app.

Build Your First Angular Component: Steps

The command helps create the project with the following files:

- src folder
- App folder
- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.module.ts

The folder contains the files created for app components.

Build Your First Angular Component: Steps

The command helps create the project with the following files:

src folder

App folder

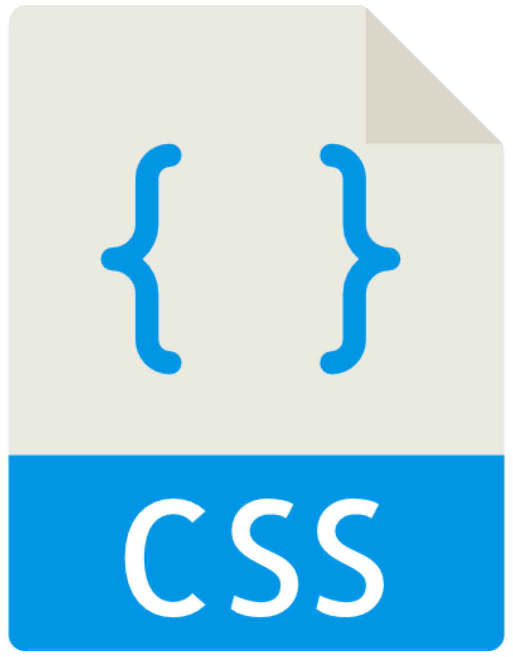
app.component.css

app.component.html

app.component.spec.ts

app.component.module.ts

The folder contains the style sheets code for the app.



Build Your First Angular Component: Steps

The command helps create the project with the following files:

src folder	App folder	app.component.css	app.component.html	app.component.spec.ts	app.component.module.ts

The folder contains the HTML code associated with the app component.

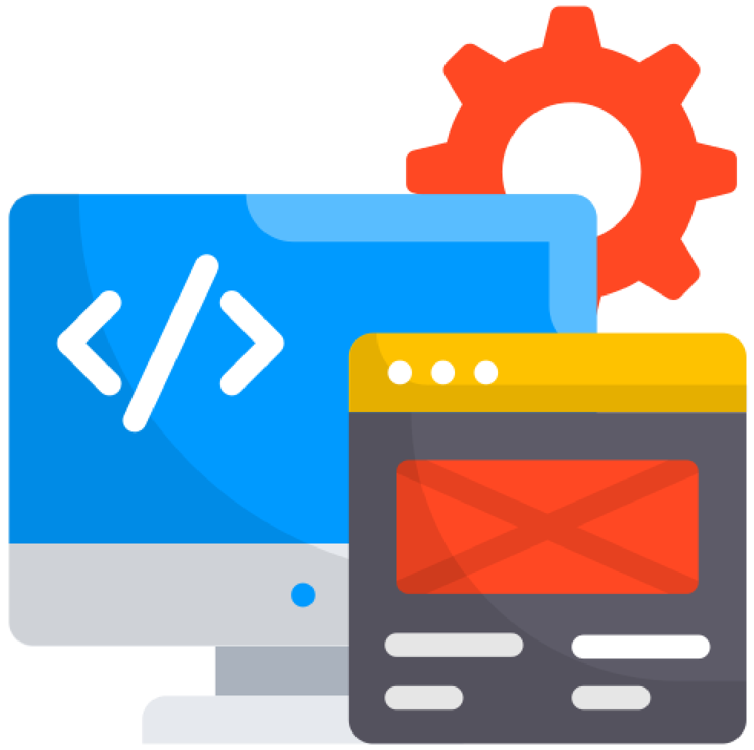
The template file is used by Angular to perform the data binding.

Build Your First Angular Component: Steps

The command helps create the project with the following files:

- src folder
- App folder
- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.module.ts

This file is used for unit testing associated with the app component.



Build Your First Angular Component: Steps

Two-way binding is beneficial in information access forms.

src folder

App folder

app.compo nent.css

app.compo nent.html

app.compo nent.spec.ts

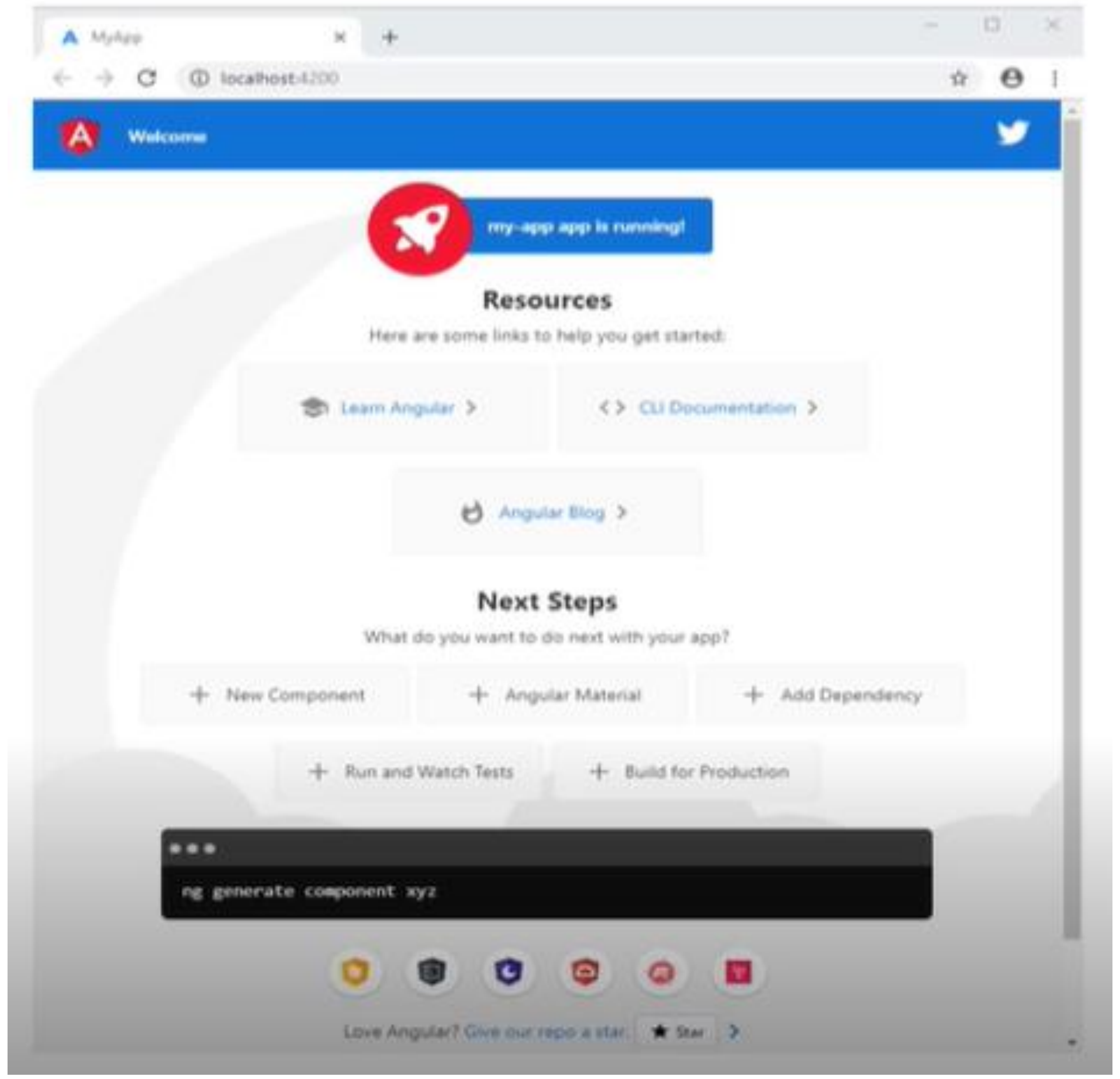
app.compone nt.module.ts

This is a TypeScript file that contains all the dependencies for the application.

Defines imported modules and components.

Build Your First Angular Component: Steps

Step 4: Run ng serve command in a terminal



Resources

Here are some links to help you get started:

Next Steps

What do you want to do next with your app?

```
ng generate component xyz
```

Love Angular? Give our repo a star.

TECHNOLOGY

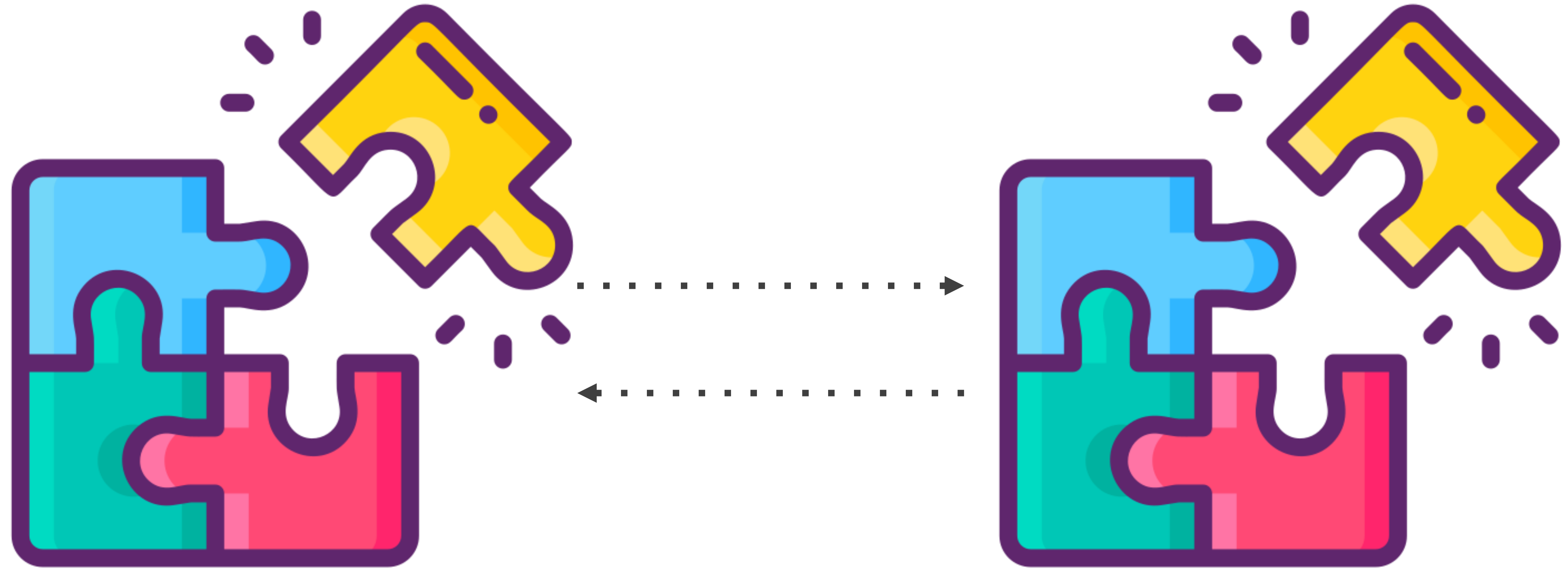
Decorators in Angular

simplilearn

Decorators in Angular

Angular uses components that interact with one another.

Angular uses decorators and the event emitter to change the component's data.



Decorators in Angular

A decorator in Angular is a function, which is used to attach metadata to a:

- Class
- Accessor
- Property
- Method
- Partner

The Decorator is applied in the form @expression.

Note

Decorators are features of TypeScript and are still not part of JavaScript.

Decorators in Angular

To enable the decorators in Angular, add the experimentalDecorators in the tsconfig.json file to enable the decorator.

The ng new command enables the decorators in angular automatically.

Syntax:

```
{
  “compilerOptions”: {
“target”: “ES5”,
“experimentalDecorators”: true
    }
}
```

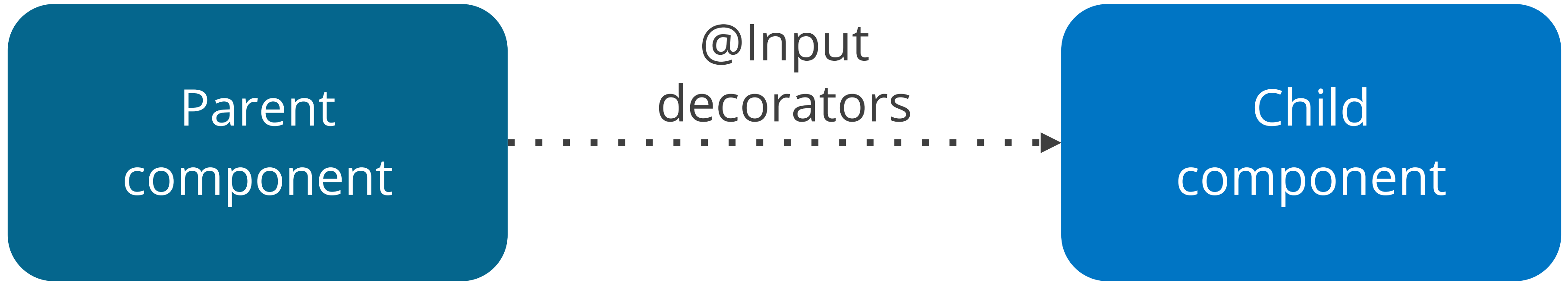
Decorators in Angular

Decorators in Angular are categorized into:

Input decorators

Output decorators

@Input decorator links the child component with a value that was given by the parent component.



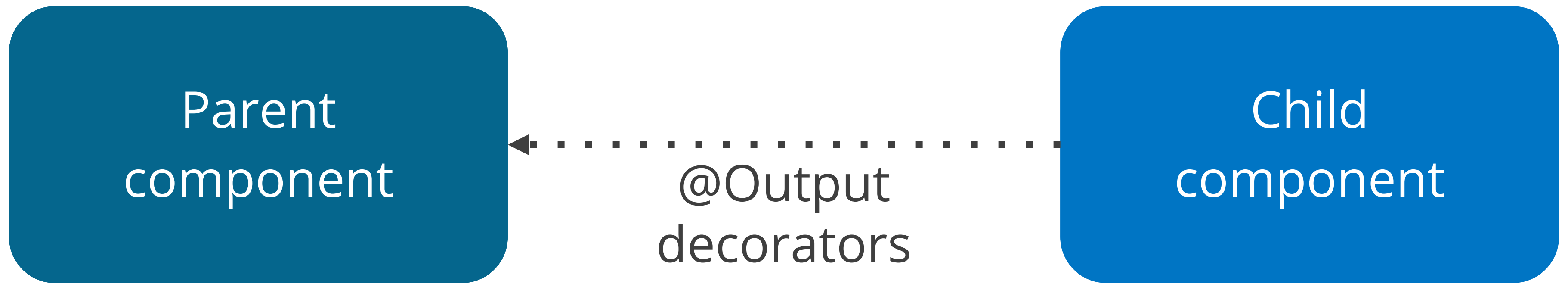
Decorators in Angular

Decorators in Angular are categorized into:

Input decorators

Output decorators

@Output decorator links child component property with a parent component.

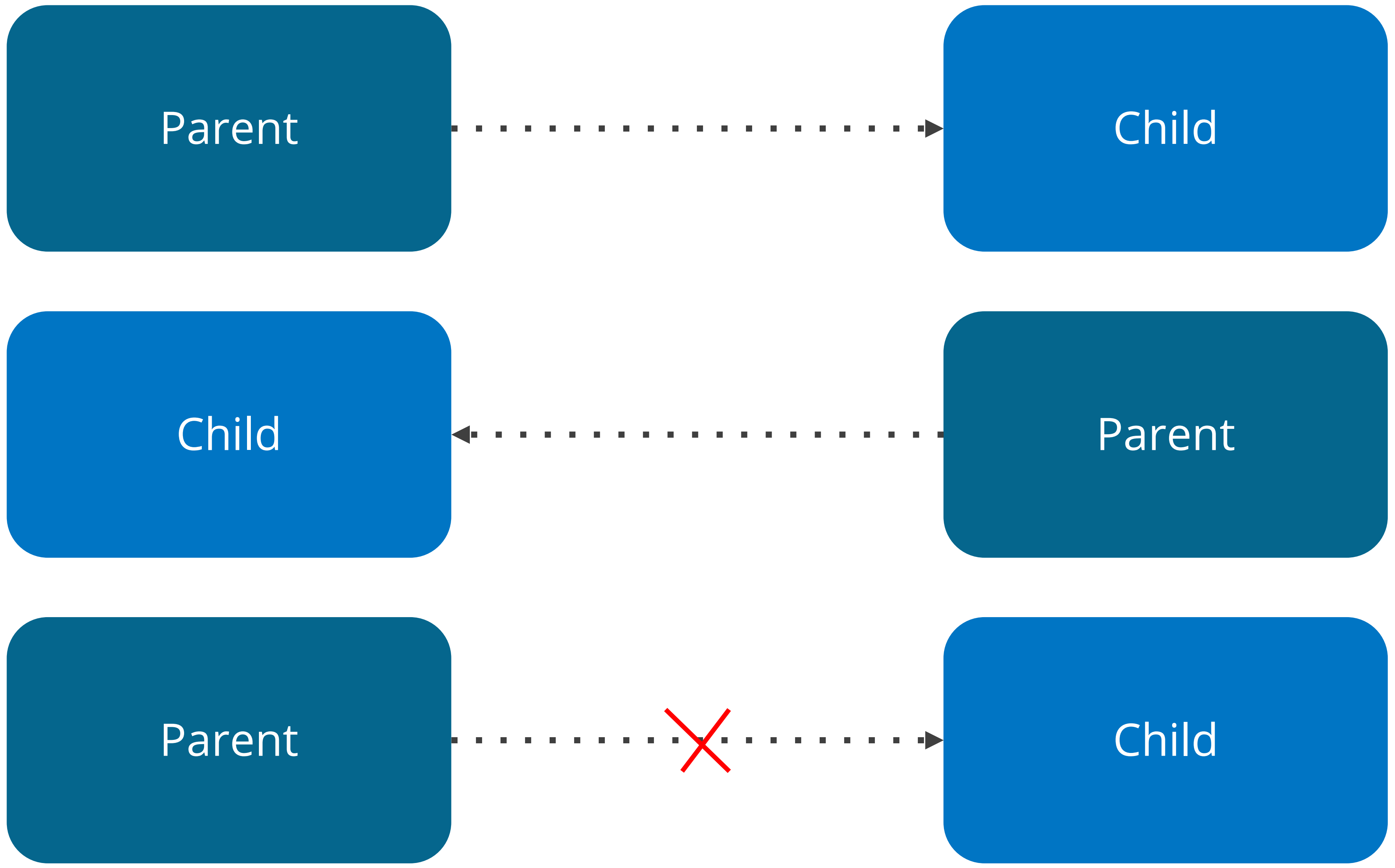


TECHNOLOGY

Angular Component Communication

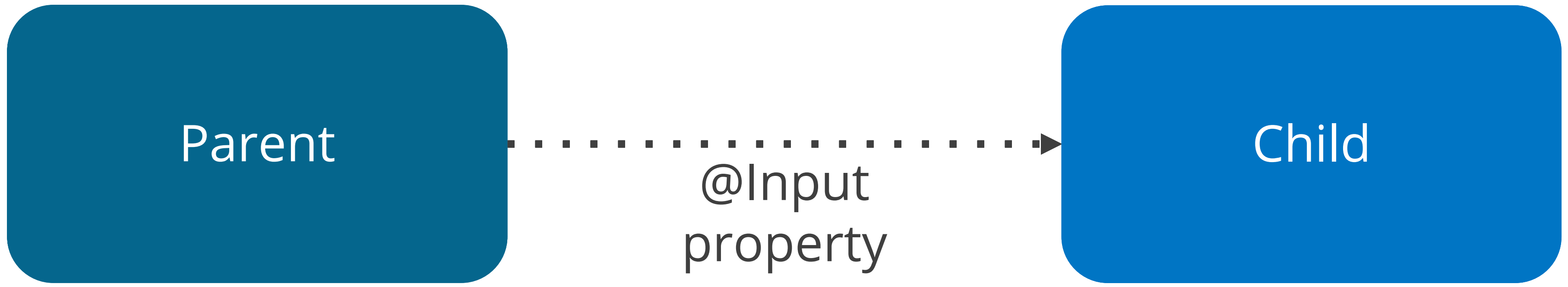
Angular Component Communication

Angular supports a few methods that facilitate communication across components and sharing of data among them.



Parent to Child Communication

If the Components possess a parent-child relationship then the parent component can pass the data to the child using the @input Property.



Syntax:

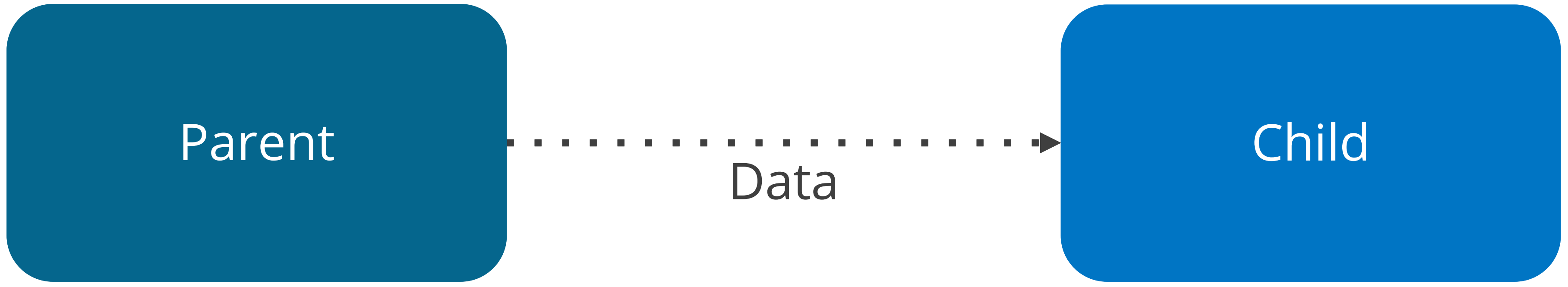
```
export class ChildComponent {
      @Input() someProperty:number;
}
```

Parent to Child Communication

As the parent component represents the child component, the value can be passed to some property using the property bind syntax as shown:

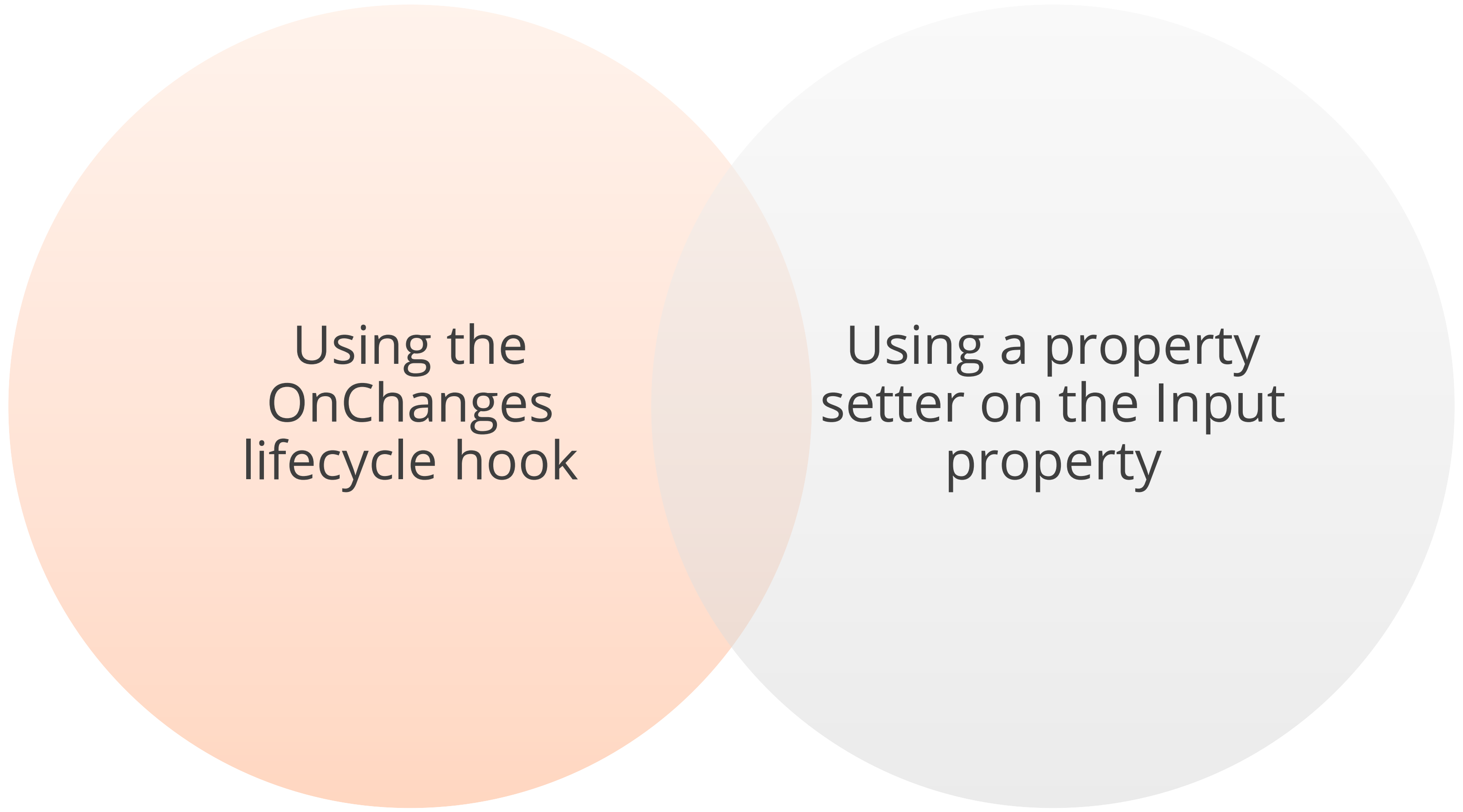
Syntax:

```
<child-component {
[someProperty]=value></child-component>
```



Parent to Child Communication

The child component must get a notification when the values change.


Using the OnChanges lifecycle hook
Using a property setter on the Input property

Child to Parent Communication

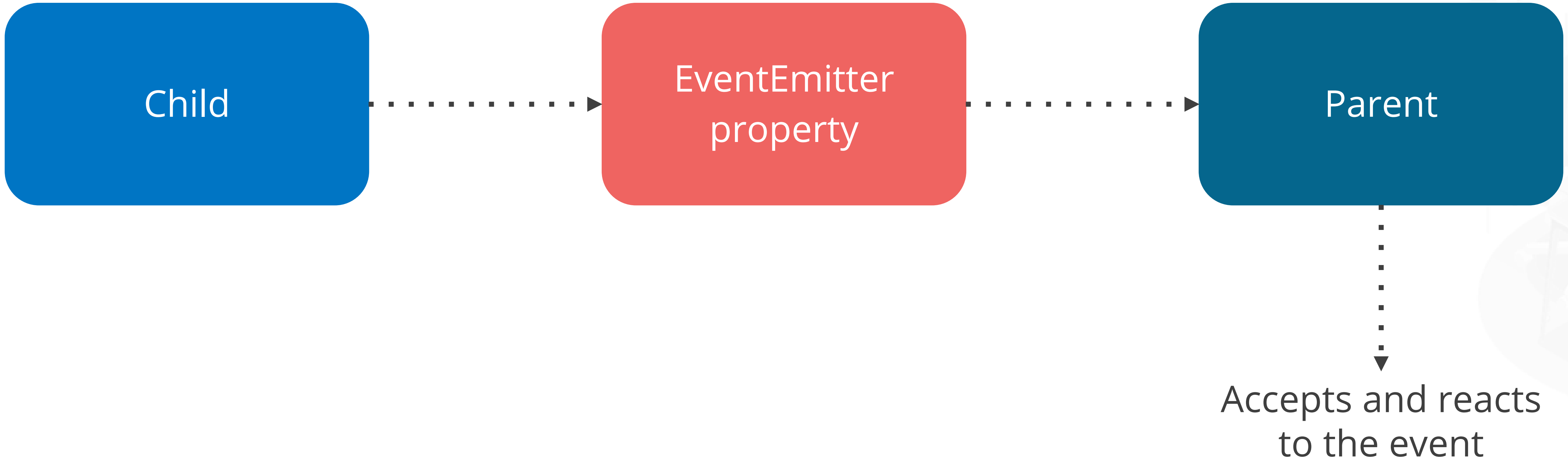
The child to parent communication can progress in many ways, that include:

- Listening to events from child
- Using local variable to access the child in the template
- Using a @ViewChild to get a reference to the child component

Child to Parent Communication: Listening to Child Events

The child component listens to the child events by exposing an EventEmitter property.

This property can be decorated with an @Output decorator.



Child to Parent Communication: Using Local Variable

A local variable is used to refer to the child component as another technique.

Syntax:

```
<child-component #child></child-component>
```

To access a property of the child component, a user can use the child.

Child to Parent Communication: Listening to Child Events

Assume the code below illustrates the count of the child component:

Syntax:

```
<p> current count is {{child.count}}</p>
```

Use @ViewChild

<child-component></child-component>

Child to Parent Communication: Using @ViewChild

Get the reference to the child component by using the ViewChild query as shown:

```
@ViewChild (ChildComponent) child:
ChildComponent;
```

Users can call any method in the child component using:

```
increment() {
  this.child.increment();
}
```

Communication When There Is No Relation

This enables users to:

Share data within various components

Use observables that can notify components when the data changes

Communication When There Is No Relation

Create a service and an Angular observable in service by using BehaviorSubject or subject as shown:

```
export class TodoService {

private_todo = new
BehaviorSubject<Todo[]>([]);
Readonly todos$ =
this._todo.asObserveable();
```

The_todo observable emits data: this._todo.next(Object.assign(_, this.todos));

Communication When There Is No Relation

The component class listens to the changes just by subscribing to the observable, as shown:

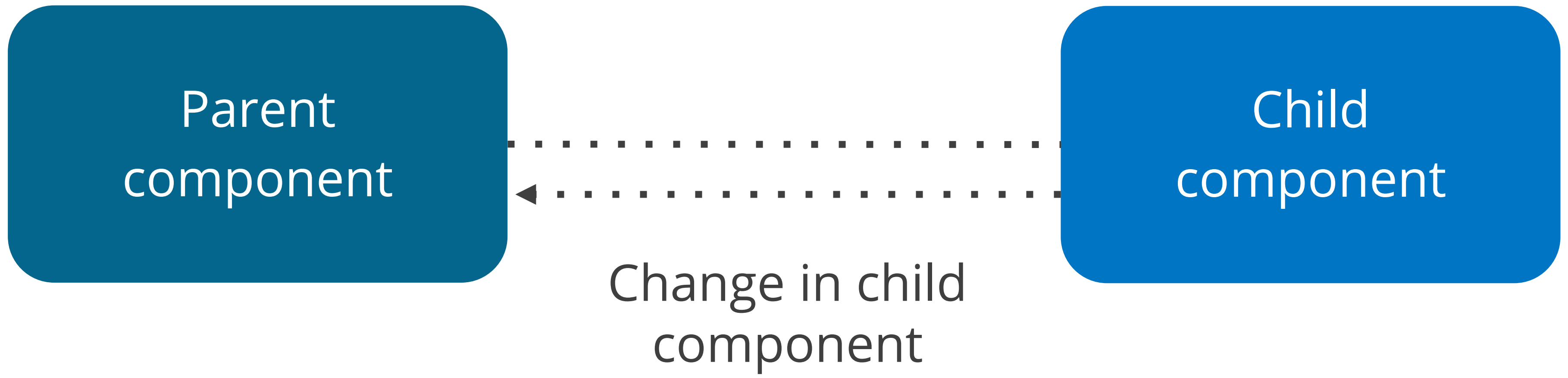
```
This.todoService.todos$.subscribe(val=>
{
      this.data=val;
      //do whatever you want to with it.
})
```

TECHNOLOGY

Angular Custom Events

Angular Custom Events

Custom event binding is a way to make communication between parent and child components.



Angular Custom Events

The parent component shows the event binding for "onProductSelected" in child component.

Statement in parent component:

```
<app-product
(onProductSelected)="showProduct($event)">
```

Property in child component:

```
@Output onProductSelected:
EventEmitter<Product>;
}
```

Angular Custom Events

When performing custom event binding in angular, there are three things one will come across:

- @Output decorator
- EventEmitter class
- $event object

Angular Custom Events

When performing custom event binding in angular, there are three things one will come across:

@Output decorator

EventEmitter class

$event object

It is used in components with the @Output Decorator to emit custom events asynchronously or synchronously.

EventEmitter registers handles for an event and handles a list of subscribing instances.

Angular Custom Events

When performing custom event binding in angular, there are three things one will come across:

- @Output decorator
- EventEmitter class
- $event object

emit (value?: T)

Emits an event with a given value, T shows the value to emit

subscribe()

Handles for events emitted

Subscription of instances is automatically performed by Angular at the time of assigning an EventEmitter to output.

Angular Custom Events

When performing custom event binding in angular, there are three things one will come across:

- @Output decorator
- EventEmitter class
- $event object

Angular sets up an event handler for the target event during event binding.

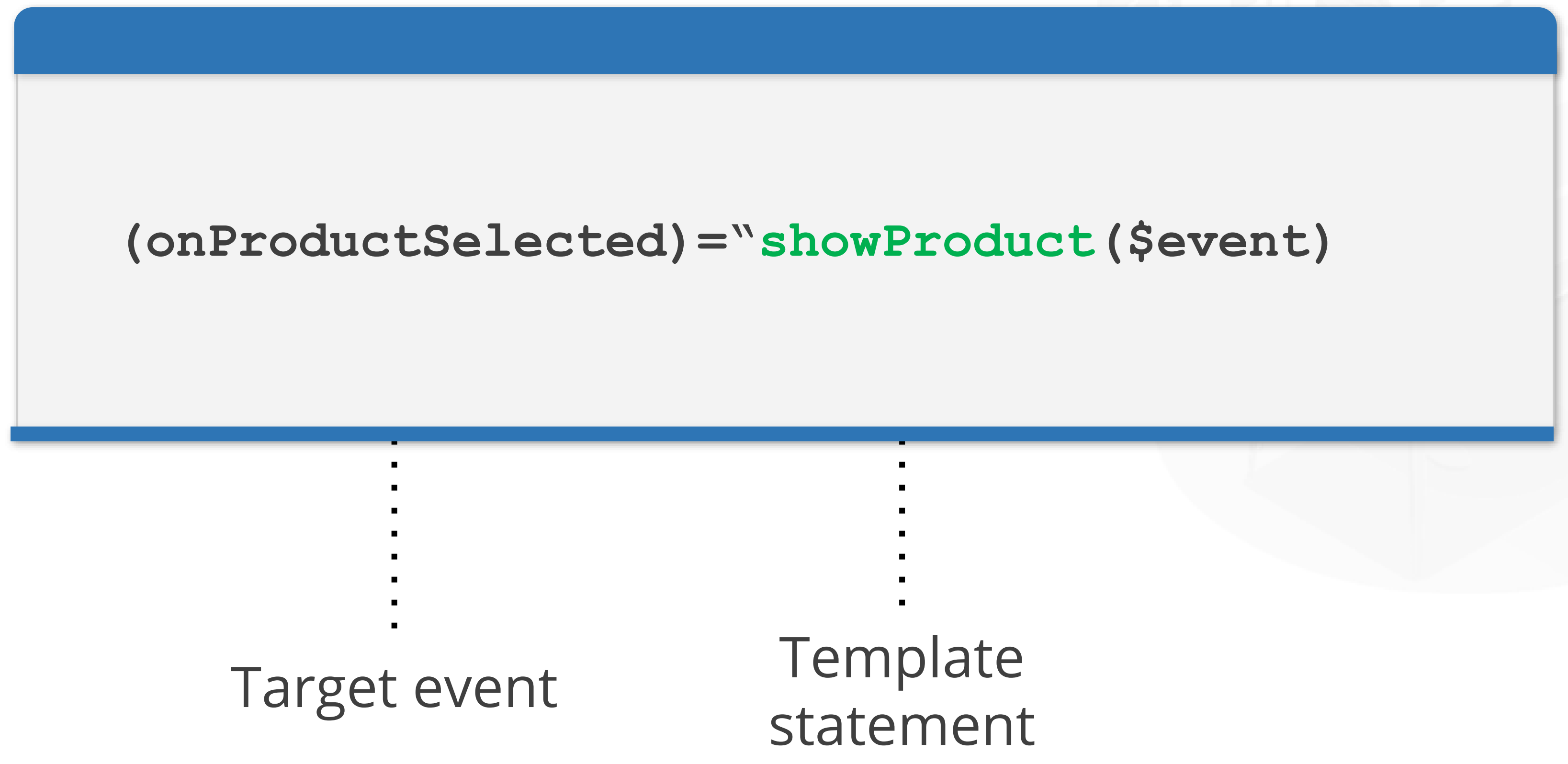
Information is stored in $event and includes data values, such as an object, event, number, or string.

Angular Custom Events

When performing custom event binding in angular, there are three things one will come across:

- @Output decorator
- EventEmitter class
- $event object

The handler executes the template statement at the time the event is raised.



```
(onProductSelected)="showProduct($event)
```

Angular Custom Events

To generate the custom events, the user needs to create a model TypeScript class.

In this model class, there are three fields: name, colour, and expiryDate.

Syntax:

```
export class Product {
 name: string;
 quantity: number;
 expiryDate: Date;

Constructor(name: string, quantity: number, expiryDate: Date)
{      this.name = name;
       this.quantity = quantity;
       this.expiryDate = expiryDate;
}
```

Angular Custom Events

The app.component.ts is also called the parent component.

In constructor:

- An array of type products is defined
- Product instances are added to the array

The property selectedProduct stores the value of the product which is selected.

Angular Custom Events

The HTML template used for app.component.ts or the parent component TypeScript is App.component.html, as shown:

Syntax:

```
<h3>Click on Product to get the details</h3>

<app-product *ngFor="let product of products" [usr]="product";
(onProductSelected) = "showProduct($event)">

</app-user>

<product-data *ngIf="selectedProduct"
[product]="selectedProduct"></product-data>
```

Angular Custom Events

The Product.component.ts is also a child component, as shown:

Syntax:

```
import { Component, Input, Output, EventEmitter } from
@angular/core';
import { Product } from './product.model';
@Component({
      selector: 'app-product',
      templateUrl: './product.component.html'
})
export class ProductComponent {
      @Input() usr: Product;
      @Output() onProductSelected: EventEmitter<Product>;
      constructor(){
        this.onProductSelected: new EventEmitter();
}
productClicked() : void{
      this.onProductSelected.emit(this.usr);
      }
}
```

Angular Custom Events

The HTML template used for product.component.ts or the child component TypeScript is Product.component.html, as shown:

Syntax:

```
<div class="container">
  <div class="row">
    <div class="col-xs-6">
      <label>Name:</label><span (click)='productClicked()'>{{
usr.name}}</span>
         </div>
     </div>
</div>
```

Angular Custom Events

productData.component.ts child component:

Syntax:

```
import { Component, Input, Output, EventEmitter } from
@angular/core';
import { Product } from './product.model';
@Component({
      selector: 'product-data',
      templateUrl: './productData.component.html'
})
export class ProductDataComponent {
      @Input() product: Product;
}
```

In this property, the selected product is assigned and then used to display product details.

Angular Custom Events

The HTML template used for product.component.ts or the child component TypeScript is Product.component.html, as shown:

Syntax:

```
<div class="jumbotron">
  <div class="container">
    <h2>Product Details</h2>
      <div class>"row">
        <div class>"col-xs-5 px-3">
          <label>Quantity (in ml):</label> {{ product.quantity }}
        </div>
        <div class="col-xs-4 px-3">
          <label>Expiry Date: </label> {{ user.expiryDate |
Date:'dd/MM/yyyy'}}
        </div>
      </div>
    </div>
</div>
```

TECHNOLOGY

Angular ngFor Core Directives

Angular ngFor Core Directives

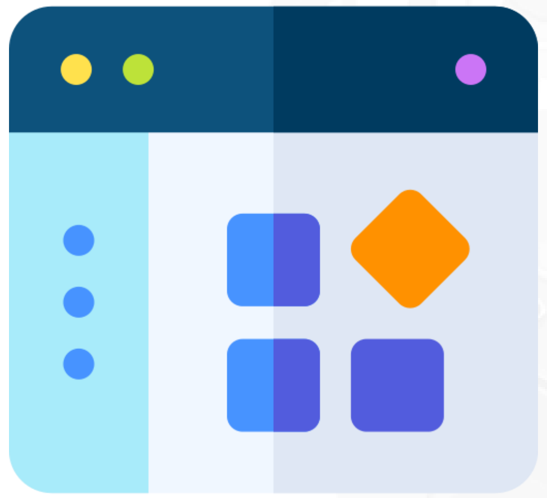
Directives are classes that provide additional behavior to elements in Angular applications.

Angular contains built-in directives that are used to manage:

Lists

Forms

Styles



What users see

Types of Directives

Components

These are the most common directives; that contain a template.

Structural directives

These are the directives that change the layout of the DOM by removing and adding DOM elements.

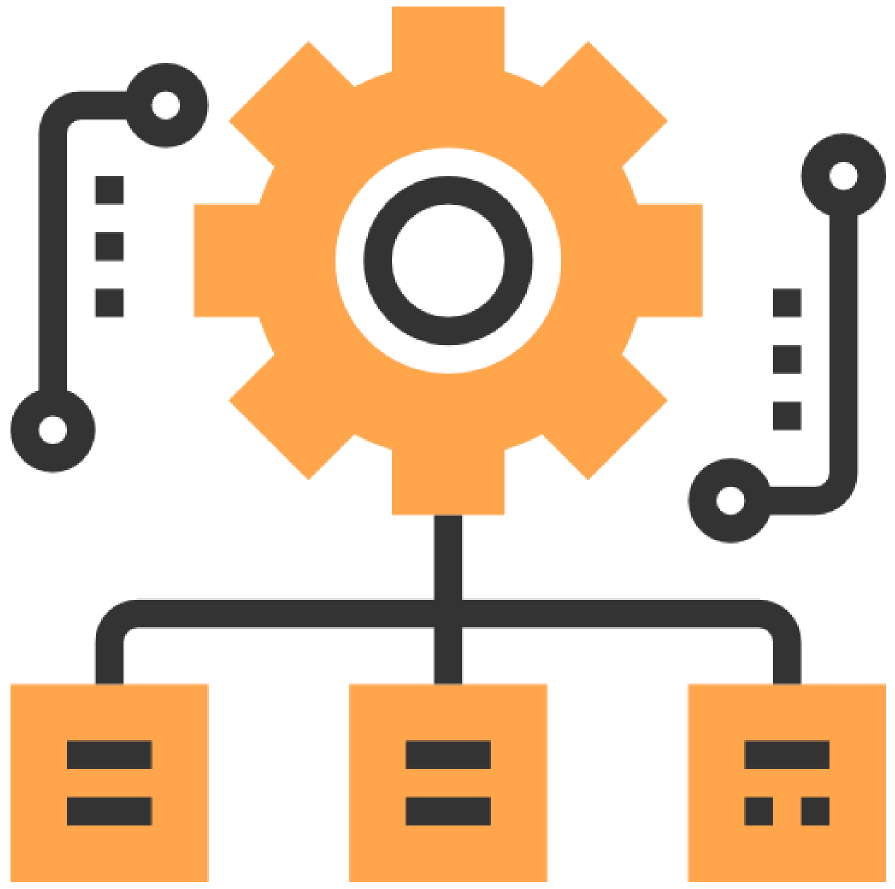
Attribute directives

These directives change the behavior or appearance of a component, element, or another directive.

Structural and Attribute Directives

Structural directives are responsible for the HTML layout.

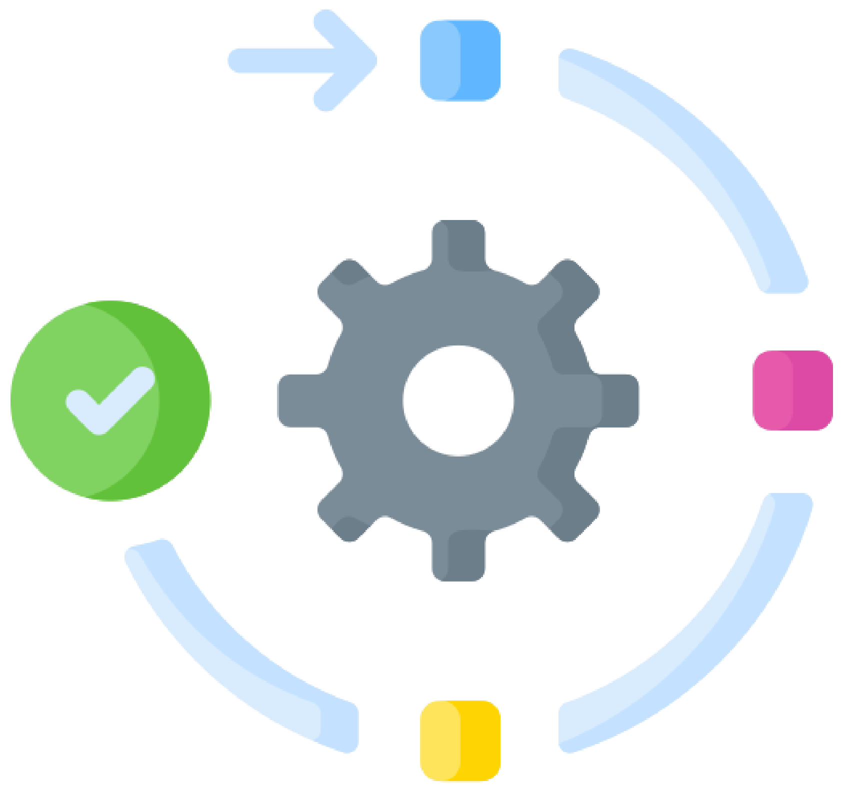
They reshape DOM by removing, adding, or changing host elements.



Built-in Structural Directives

ngFor:

It is a looping directive which iterates over every item in a list or any type of collection and renders the element for every item.



Built-in Structural Directives

Here is an example where ngFor is used in a li element:

```
<li *ngFor="let product of products">
….
</li>
```

The asterisk (*) in *ngFor depicts that it is a structural directive.

Built-in Structural Directives

Here is an example to iterate through an array of product instances using ngFor:

Create a model TypeScript class as:

```
Product.model.ts

export class Product {
 name: string;
 quantity: number;
 expiryDate: Date;

Constructor(name: string, quantity: number, expiryDate: Date)
{      this.name = name;
       this.quantity = quantity;
       this.expiryDate = expiryDate;
}
```

Built-in Structural Directives

Create the component class also called the parent component, as shown:

```
import { Component } from @angular/core';
import { Product } from './product/product.model';
@Component({
       selector: 'app—root',
       templateUrl: './app.component.html'
       styleUrls: ['./app.component.css']
})
export class AppComponent {
        Product : Product[];
        Constructor(){
        //Adding Products instances to products array
         this.users: [new Product('soap', 100, new Date('2022-03-21')),
                 new User('Toothpaste', 200, new Date('2022-05-09')),
                 new User('Shampoo', 150, new Date('2022-10-21'))];
}
showProduct(product: Product): void{
       // Setting selected product
       this.selectedProduct = product;
  }
}
```

Built-in Structural Directives

Instances can also be added to an array by using the JSON format.

Built-in Structural Directives

ngFor is used to iterate over an array of products.

```
<div class=“container”>
    <h2>Product Details</h2>
    <table class>“table table-sm table-bordered m-t-4”>
        <tr>
          <th>Name</th>
          <th>Quantity (in ml)</th>
          <th>Expiry Date</th>
        </tr>
        <tr *ngFor=“let product of products”>
          <td>{{product.name}}</td>
          <td>{{product.age}}</td>
          <td>{{product.joinDate | date:’dd/MM/yyyy’}}</td>
        </tr>
     </table>
</div>
```

ngFor Core Directives Values

ngFor Core Directive Values

ngFor gives exported values which can be assigned to local variables and can be used in the element.

Index: number

Gives an index of the item in iterable

Count: number

Gives the length of the iterable

First: boolean

Gives true when the item is first in the iterable

ngFor Core Directive Values

ngFor gives exported values which can be assigned to local variables and can be used in the element.

Last: boolean

Gives true when the item is last in the iterable

Even: boolean

Gives true when the item has an even index in the iterable

Odd: boolean

Gives true when the item has an odd index in the iterable

ngFor Core Directive Values

Here is an example for ngFor index value to provide an S.no as a column in a table:

```
<div class="container">
    <h2>Product Details</h2>
    <table class>"table table-sm table-bordered m-t-4">
        <tr>
          <th>Name</th>
          <th>Quantity (in ml)</th>
          <th>Expiry Date</th>
        </tr>
        <tr *ngFor="let product of products; index as i;">
          <td>{{i+1}}</td>
          <td>{{user.name}}</td>
          <td>{{product.quantity}}</td>
          <td>{{product.expiryDate | date:'dd/MM/yyyy'}}</td>
        </tr>
     </table>
</div>
```

ngFor Core Directive Values

Here is an example for ngFor index value to provide an S.no as a column in a table:

```
<div class="container">
    <h2>Product Details</h2>
    <table class>"table table-sm table-bordered m-t-4">
        <tr>
          <th>S. No</th>
          <th>Name</th>
          <th>Quantity (in ml)</th>
          <th>Expiry Date</th>
        </tr>
        <tr *ngFor="let product of products; index as i; let odd=odd">
          <td>{{i+1}}</td>
          <td>{{user.name}}</td>
          <td>{{product.quantity}}</td>
          <td>{{product.expiryDate | date:'dd/MM/yyyy'}}</td>
        </tr>
     </table>
</div>
```

TECHNOLOGY

Angular ngIf Directive

Angular ngIf Directive

ngIf directive is used to remove or include an element in the HTML document.

If the expression used in ngIf is true, an element is included; otherwise, the expression is removed from the DOM.

Angular ngIf Directive

Asterisk(*) in *ngIf directive shows that it is a structural directive.

```
<div *ngIf="product"
class="name">{{product.name}}</div>
```

These are responsible for changes in HTML, and DOM's structure, mainly removing, adding or changing elements.

Angular ngIf Directive

In component, the visibility of the element depends on the expression, whether the expression is true or false.

TypeScript file for ngIf as App.component.ts:

```
import { Component } from ‘@angular/core’;
@Component({
      selector: ‘app-root’,
      templateUrl: ‘./app.component.html’,
      styleUrls:[‘./app.component.css’]
})
export class AppComponent {
      toggle = Boolean = true;
}
```

Angular ngIf Directive

The HTML template used for ngIf is app.component.html as shown:

```
<div class="container">
  <div class>"row">
     <div class>"col-xs-12">
          <p *ngIf="toggle"> Hello ngIf </p>
        </div>
    </div>
</div>
```

Angular ngIf Directive

The value of the toggle variable is not changed once it is assigned.

Add the toggle button to display by adding a code snippet

```
onClickToggle(){
        this.toggle = !this.toggle;
}
```

Angular ngIf Directive

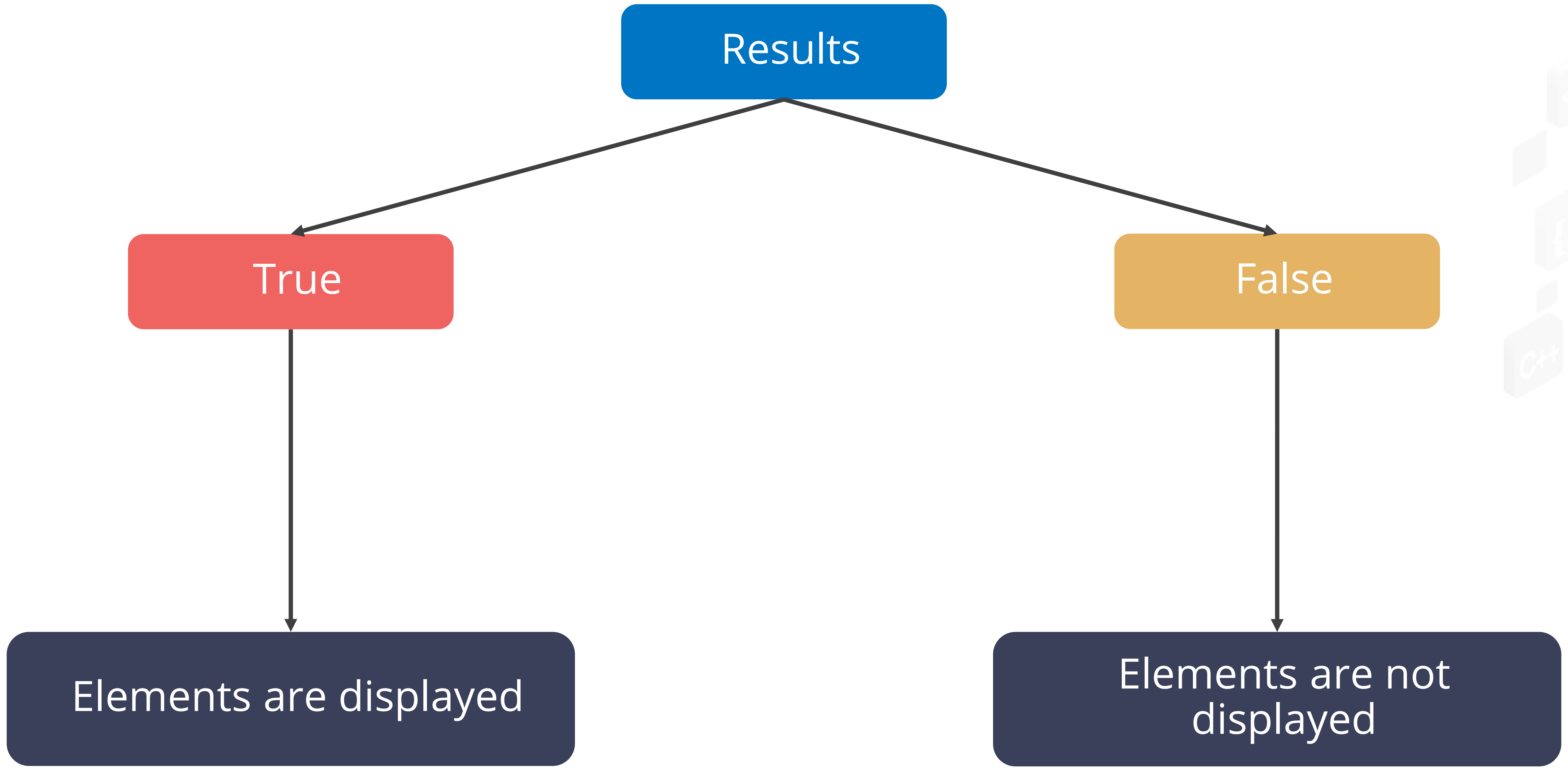
onClickToggle() method reverses the value of the toggle.

After calling the onClickToggle() method, the paragraph in app.component.html will not be visible, and it is visible again when the method is called again.

```
onClickToggle(){
        this.toggle = !this.toggle;
}
```

Angular ngIf Directive

For comparing variables, ngIf can be used.



Angular ngIf Directive

Scenario: In a component, a string variable and element are displayed based on whether the string variable has some value.

The TypeScript file app.component.ts is as shown:

```
import { Component } from '@angular/core';
@Component({
      selector: 'app-root',
      templateUrl: './app.component.html',
      styleUrls:['./app.component.css']
})
export class AppComponent {
      show: String;
      constructor(){
      this.show = "something";
   }
}
```

Angular ngIf Directive

The HTML file for the parent component is app.component.html, as shown:

```
<div class=“container”>
  <div class>“row”>
     <div class>“col-xs-12”>
          <p *ngIf=“ show === ‘something’”>
              Hello from ngIf </p>
        </div>
    </div>
</div>
```

Angular ngIf Directive

ngIf helps to check whether the object exists or not before trying to access its fields.

Model class Product.model.ts is as shown:

```
export class Product {
 name: string;
 quantity: number;
 expiryDate: Date;

Constructor(name: string, quantity: number, expiryDate: Date)
{      this.name = name;
       this.quantity = quantity;
       this.expiryDate = expiryDate;
}
```

Angular ngIf Directive

The TypeScript file is app.component.ts, as shown:

```
import { Component } from '@angular/core';
import { Product } from './product/product.model';
@Component({
      selector: 'app-root',
      templateUrl: './app.component.html',
      styleUrls:['./app.component.css']
})
export class AppComponent {
      product: Product;
      constructor(){
      //creating product instance
      this.show = new Product("hand wash",200,new Date('2022-03-25'));
   }
}
```

Angular ngIf Directive

The HTML file for the app.component.ts is:

```
<div class="“container”>
  <div class>“row”>
     <div class>“col-xs-12”>
          <p *ngIf=“ product ”><label>
              Name: </label>
{{product.name}}</p>
        </div>
    </div>
</div>
```

Note

ngIf will not hide the elements with CSS. It removes or adds them physically from DOM.

ngIf with Else in Angular: Example

An else block can be used in this code:

```
<div class="container">
  <div class>"row">
     <div class>"col-xs-12">
          <div *ngIf=" show;else elseBlock "> This text is
displayed when condition is true </div>
          <ng-template #elseBlock> This text is displayed when
condition is false </ng-template >
            </div>
      </div>
</div>
```

TECHNOLOGY

Angular ngClass Core Directive

Angular ngClass Core Directive

The ngClass directive is used to remove or add CSS classes for an HTML element.

There are a couple of ways by which the CSS classes can be provided.

String

The classes listed in the string (space delimited) are added as shown:

```
<element [ngClass] =“first second”>
….</element>
```

Angular ngClass Core Directive

There are a couple of ways by which the CSS classes can be provided.

Array

The classes declared as array elements as shown:

```
<element [ngClass] =[‘first’, ‘second’]>
….</element>
```

Angular ngClass Core Directive

There are a couple of ways by which the CSS classes can be provided.

Object

The object is a key-value pair in which the keys are CSS classes and values are the conditions.

Classes get added as and when the expression satisfies the condition as shown:

```
<element [ngClass] ="{'first' : true,
'second' : true, 'third' : false}">
….</element>
```

Angular ngClass Core Directive: Example

An example for the ngClass directive to show the usage of ngClass.

1. Add a CSS class

```
app.component.css

.text-border {
color: black;
border: 1px solid blue;
background-color: green;
}
```

Angular ngClass Core Directive: Example

2. In the HTML file, add two elements:
One with the value being actual and another with a value as false

```
<div class="container">
  <div [ngClass]="{'text-border': false}">
      This text has no border
  </div>
  <div [ngClass]="{'text-border': true}">
      This text has border
  </div>
</div>
```

Angular ngClass Core Directive: Example

An example of ngClass directive dynamic class assignment:

CSS used is as shown:

```
.text-border {
color: white;
border: 1px solid black;
background-color: red;
}
```

In this example, a message is sent for network transfer.

A Status button shows the current status of the network transfer.

Angular ngClass Core Directive: Example

The Component class is written as:

```
import { Component } from '@angular/core';
@Component({
      selector: 'app-root',
      templateUrl: './app.component.html',
      styleUrls:['./app.component.css']
})
export class AppComponent {
      temp: [];
      onClick(){
      this.temp.push('still transferring');
   }
}
```

The onClick() method adds one more message to the array.

Angular ngClass Core Directive: Example

The Template class is as shown:

```
<div class="container">
  <button class ="btn btn-primary" (click)="onButtonClick()">
   status
  </button>
  <div *ngFor="let t of temp; let i = index">
      <span [ngClass]="{'text-border': i > 5}">({t})</span>
  </div>
</div>
```

TECHNOLOGY

Angular ngStyle Core Directive

Angular ngStyle Core Directive

ngStyle directive helps the user set the CSS properties for the containing HTML element.

These properties are defined as colon-separated key-value pairs.

- Key refers to the style name, with an optional suffix (such as 'top.px', 'font-style.em).
- Value is an expression to be evaluated.

Angular ngStyle Core Directive

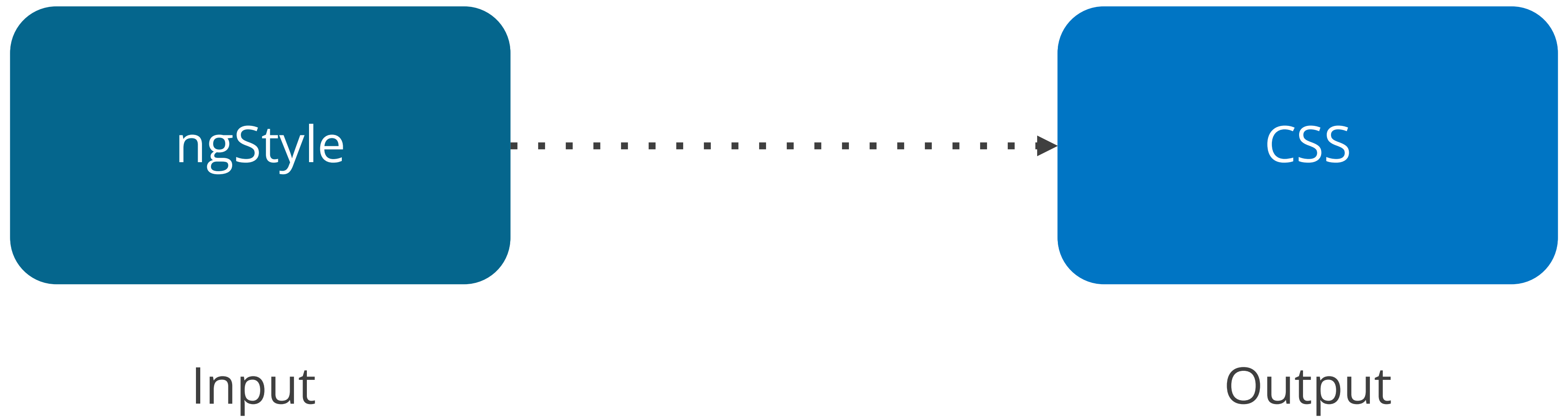
Example:

```
<div class="container">
  <div [ngStyle="{'font-style': 'italic', 'color': 'white',
'background-color': 'blue'}">
      Text style using ngStyle
  </div>
</div>
Here the color, font-style, and background-color CSS
properties are set for the containing div using
ngStyle.
```

The color, font-style, and background-color CSS properties are set for the containing div using ngStyle.

Angular ngStyle Core Directive: Example

ngStyle is best used when the value is dynamic.



Angular ngStyle Core Directive: Example

Condition: A model class user with three marked fields is given, and the color of the mark is changed according to the conditions defined within a function.

User.model.ts is shown:

```
export class User {
      name: sring;
      userId: number;
      m1: number;
      m2: number;
      m3: number;
      constructor(name: sring, userId: number, m1: number,
m2: number, m3: number){
      this.name: name;
      this.userId: userId;
      this.m1: m1;
      this.m2: m2;
      this.m3: m3;
    }
}
```

Angular ngStyle Core Directive: Example

The component class is shown:

```
import { Component } from '@angular/core';
import { User } from './user/user.model';
@Component({
      selector: 'app-root',
      templateUrl: './app.component.html',
      styleUrls:['./app.component.css']
})
export class AppComponent {
      users: User[];
```

Angular ngStyle Core Directive: Example

The component class is shown:

```
constructor () {
      // Adding Student instances to users array
      this.Users [
      { name:'John', userId: 12, ml:55, m2:79, m3:85 },
      { name:'Peter', userId: 25, ml:35, m2:50, m3:65 },
      { name:'Brij', userId: 27, ml:85, m2:90, m3:92 },
   ];
}
```

Angular ngStyle Core Directive: Example

The component class is shown:

```
getMarkColor(mark) {
      if(m >= 75)
            return ‘green’;
      else if(m >= 50 && m < 75)
            //color Amber
            return ‘#FFBF00’;
      else
            return ‘red’;

   }
}
```

Angular ngStyle Core Directive: Example

In the component class, an array of type users are given, and user instances are added to that array in the constructor.

getMarkColor() method has statements for returning the color of the marks.

```
<div class="container">
    <h2 [ngStyle]={'font-size.px':30}>User Details</h2>
    <table class>"table table-sm table-bordered m-t-4">
        <tr>
           <th>Name</th>
          <th>User Id</th>
           <th>M1</th>
           <th>M2</th>
           <th>M3</th>
         </tr>
         <tr *ngFor="let user of users">
           <td>{{user.name}}</td>
           <td>{{user.userId}}</td>
           <td[ngStyle]="{'color':getMarkColor(user.m1)}">(user.m1)}</td>
           <td[ngStyle]="{'color':getMarkColor(user.m1)}">(user.m2)}</td>
           <td[ngStyle]="{'color':getMarkColor(user.m1)}">(user.m3)}</td>
         </tr>
     </table>
</div>
```

Angular ngStyle Core Directive

The ngStyle directive is used with the tag to set the font size.

ngStyle is to change the color of the text.

getMarkColor() method is used to get the value for the color property.

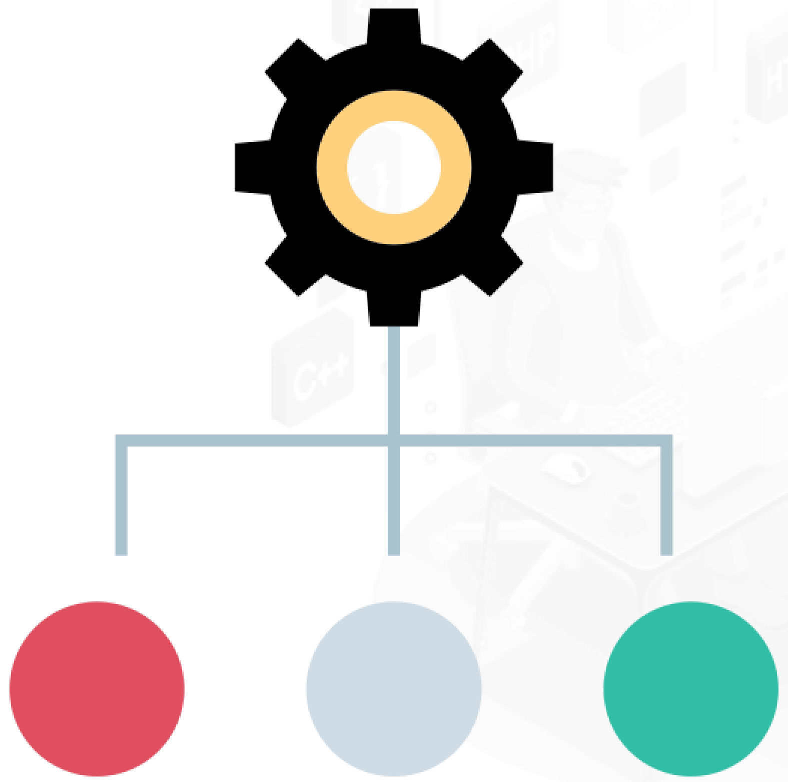
TECHNOLOGY

Angular ngSwitch Core Directive

Angular ngSwitch Core Directive

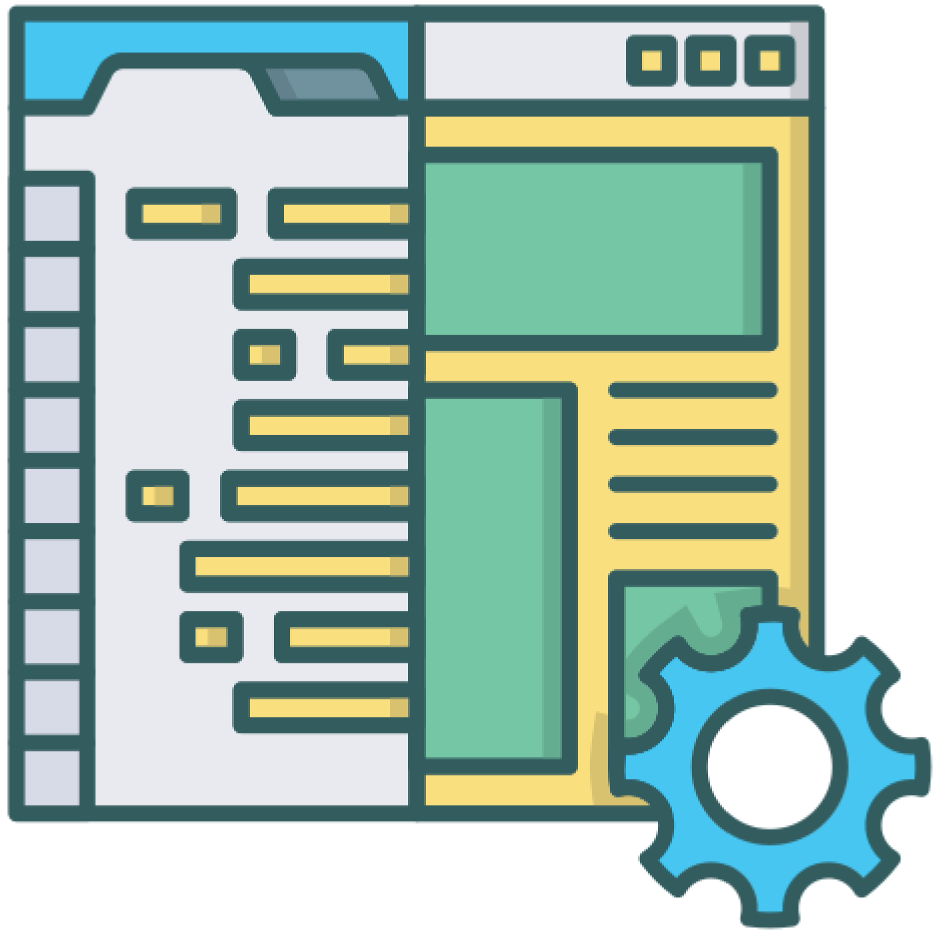
Use Angular ngSwitch directive to execute different elements with different conditions

```
<container-element [ngSwitch]=“switch_expression”>
      <element
*ngSwitchCase="match_expression_1”>…</element>
      <element
*ngSwitchCase="match_expression_2”>…</element>

      ....

   <element *ngswitchDefault>...</element>
</container-element>
```



Angular ngSwitch Core Directive

By using the ngSwitch directive, one can observe that:



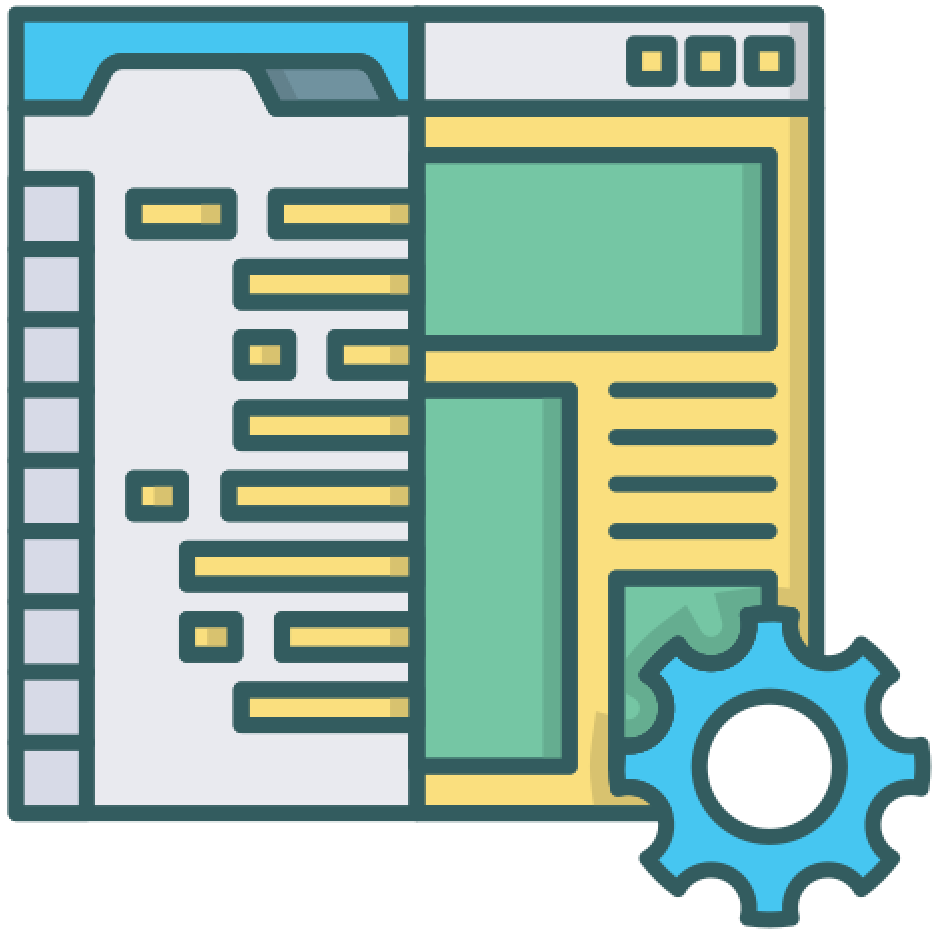
Every matched view is accomplished.

A view associated with the ngSwitchDefault directive is rendered.

If ngSwitchDefault is not used, nothing will be rendered.

Angular ngSwitch Core Directive

By using the ngSwitch directive, one can observe that:



The asterisk(*) indicates a structural directive and removes or adds a DOM element.

If match_expression = switch_expression, the corresponding element is added.

The element that the ngSwitch directive is applied to is included in the HTML file.

Angular ngSwitch Core Directive: Example

Array length = Passes number:

```
import { Component } from '@angular/core';
@Component({
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls:['./app.component.css']
})
export class AppComponent {
        array: number[];
        length: number[];

        constructor(){
          this.array = [1,2,3];
        }
        getLength(): number{
          this.length = this.array.length;
          return this.array.length;
    }
}
```

Angular ngSwitch Core Directive: Example

The template is as shown:

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <div> There are {{getLength()}}
numbers.</div>

     <div [ngSwitch]="getLength()">
      <span *ngSwitchCase="0">No number is added</span>
      <span *ngSwitchCase="1">One number is added</span>
      <span *ngSwitchCase="2">Two number is added</span>
      <span *ngSwitchCase="3">Three number is added</span>
    </div>
   </div>
 </div>
</div>
```

ngSwitchDefault: <span *ngSwitchDefault>

Angular ngSwitch with String Values: Example

To match a literal string, the string needs to be enclosed in single quotes.

To match that character, the component can be written as:

```
import { Component } from '@angular/core';
@Component({
       selector: 'app-root',
       templateUrl: './app.component.html',
       styleUrls:['./app.component.css']
})
export class AppComponent {
       array: string[];
       constructor(){
         this.array = ['A', 'B', 'C', 'D'];
       }
       getLength(): String{
         return this.array[0];
    }
}
```

Angular ngSwitch Directives: Example

The template is as shown:

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
     <div [ngSwitch]="getChar()">
      <span *ngSwitchCase="A">This is A from the array</span>
      <span *ngSwitchCase="B">This is B from the array</span>
      <span *ngSwitchCase="C">This is C from the array</span>
      <span *ngSwitchCase="D">This is D from the array</span>
      <span *ngSwitchDefault>Some other alphabet</span>
    </div>
   </div>
 </div>
</div>
```

TECHNOLOGY

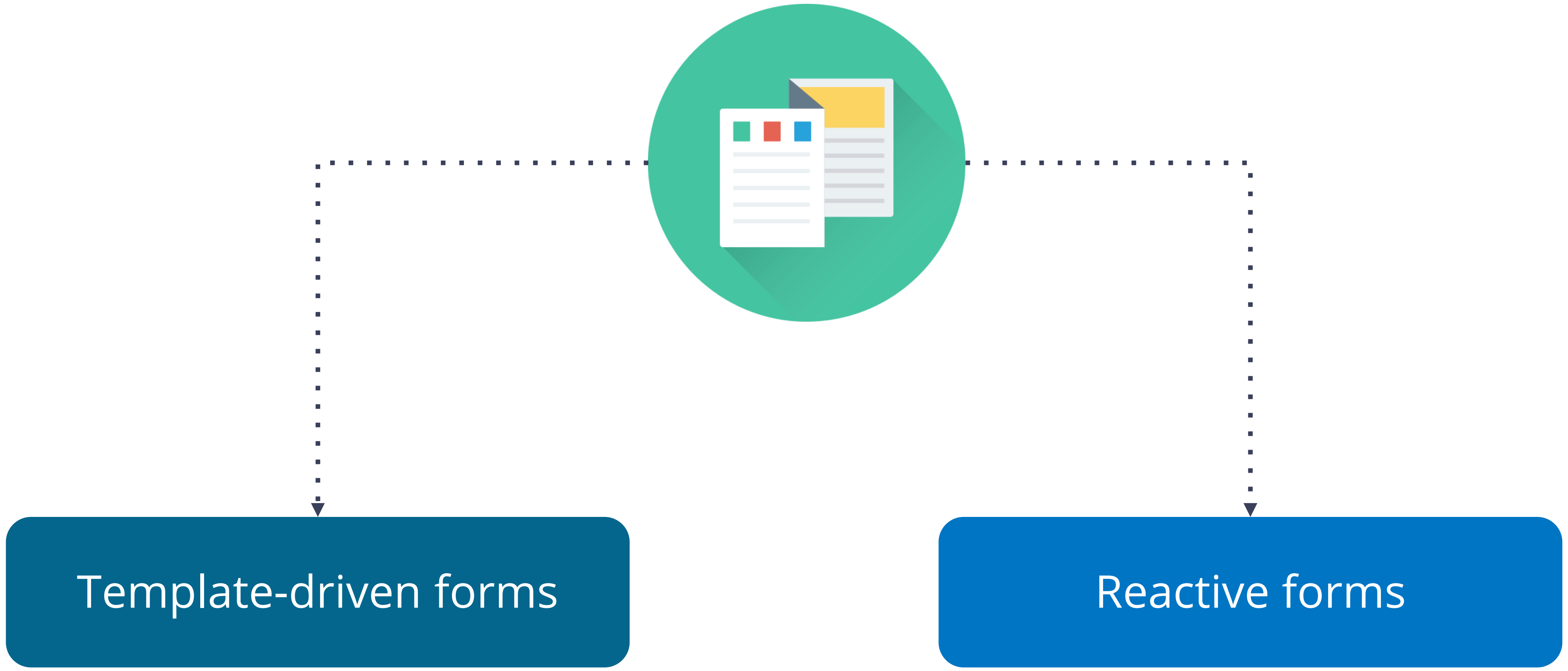
Forms

simplilearn

Forms

User input is an important part of any front-end application.

Angular frameworks provide a very strong API for handling forms.



Forms

Handling forms in Angular provides the functionality to:

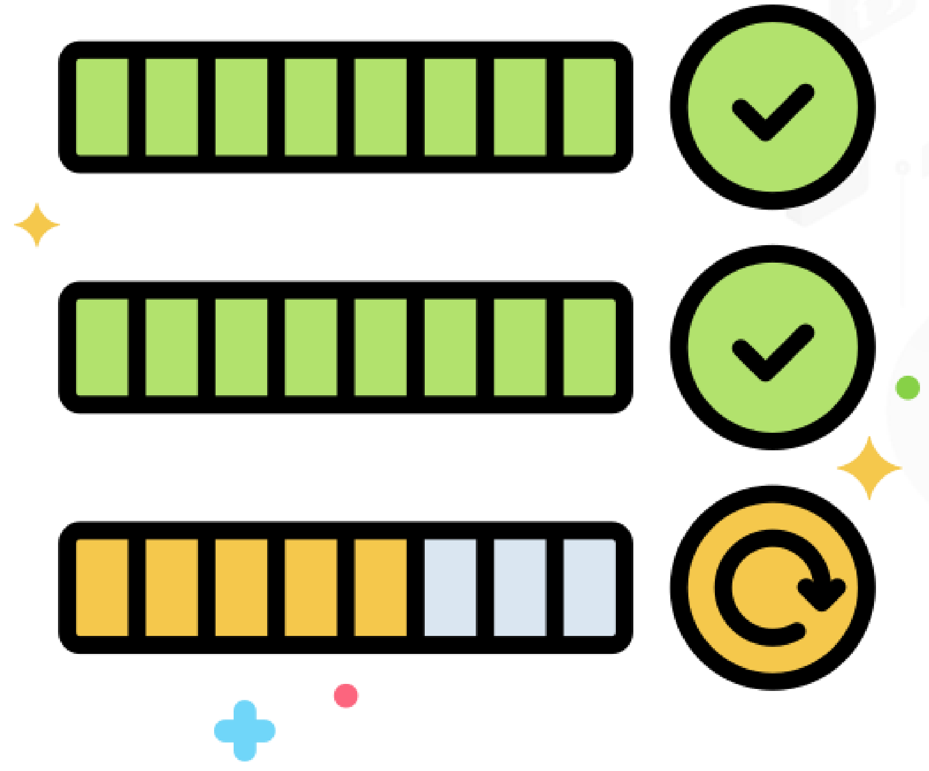
- Capture the inputs by the user from UI
- Validate user input
- Generate a form and data model
- Track the changes in form and update the model accordingly

Forms

Here are some basic classes that are used:

- FormControl
- FormGroup
- FormArray
- ControlvaluerAccessor

It is an individual form control and tracks the value and validation status.

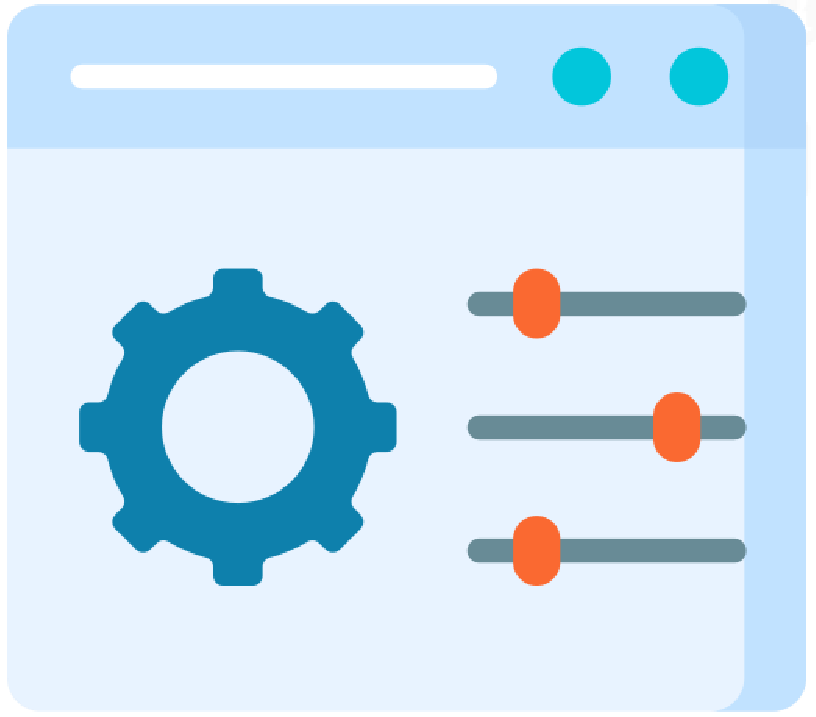


Forms

Here are some basic classes that are used:

- FormControl
- FormGroup
- FormArray
- ControlvaluerAccessor

It is a collection of form controls.



Forms

Here are some basic classes that are used:

- FormControl
- FormGroup
- FormArray
- ControlvaluerAccessor

It tracks the same values and status of an array of form controls.

Forms

Here are some basic classes that are used:

FormControl

FormGroup

FormArray

ControlvaluerAccessor

It is an interface that makes a connection between Angular FormControl instances and native HTML elements.

Forms

Most of the form-related functionality is in the template. Forms rely on directives in the HTML to create and change the underlying object model.

The ngModel directive creates and manages a form control instance for a given form.

Forms

Here is an example of a single control implemented by a template and component using template-driven forms:

```
import { Component } from '@angular/core';
@Component({
       selector: 'app-product',
       template:
       Employee Name:<input type="text"
[(ngModel)]="ProductName";
})
export class ProductComponent{
        ProductName ='';
}
```

Forms

Reactive forms help define the form model directly in the component class.

[FormControl] directive links the form FormControl instance to a particular form element in the template using an internal value accessor.

Forms

Here is an example of a view and component implementing a single control with reactive form:

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

@Component({
       selector: 'app-product',
       template:
       Employee Name:<input type="text"
[(FormControl)]="ProductName";
})

export class ProductComponent{
        ProductName = new FormControl('');
}
```

Forms

Form validation is a crucial part of any web application.

It is used to validate whether the user input is in the correct format.

Validation in template-driven forms can be done with the help of tracking control states.

Forms

Tracking control states is used with form controls to track the state of the control.

There are three things that one need to look for while validating form fields:

- Whether the user touched the control
- Whether the value of the form control is changed
- Whether the entered value is valid

Forms

Angular provides special CSS classes on the control element to reflect the state.

State	Class is true	Class is 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

Forms

The CSS classes help to highlight the required fields and create the error message.

The syntax can be written as:

```
<input type=“text” id=“name” ngmodel=“”
name=“name” required=“” class=“form-control ng-
untouched ng-pristine ng-invalid” ng-reflect-
model=“”
ng-reflect-name=“name” ng-reflect-required=“”(‘’)
```

TECHNOLOGY

Angular Service

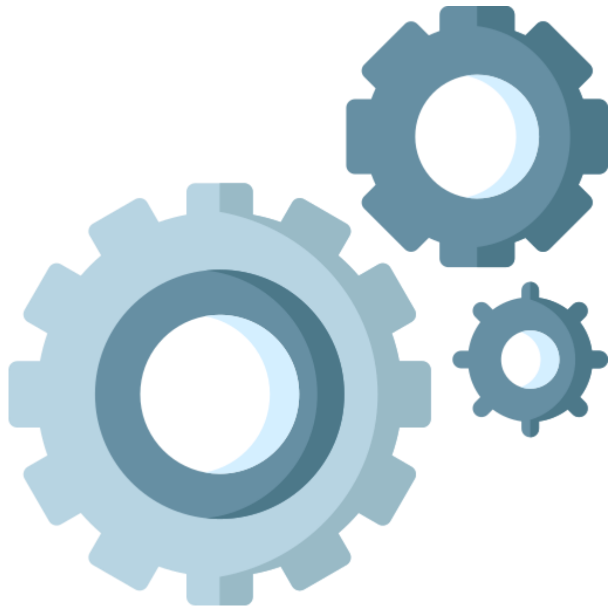
Angular Service

- Angular service is a piece of reusable code with one purpose.
- The elements have to access the information.
- The element should focus on presenting data to the user.

Data to access the code can be written in every Component.

The task of obtaining data from the back-end server should be allotted to another category.

Angular Service: Uses



For features which are independent of elements such as work services

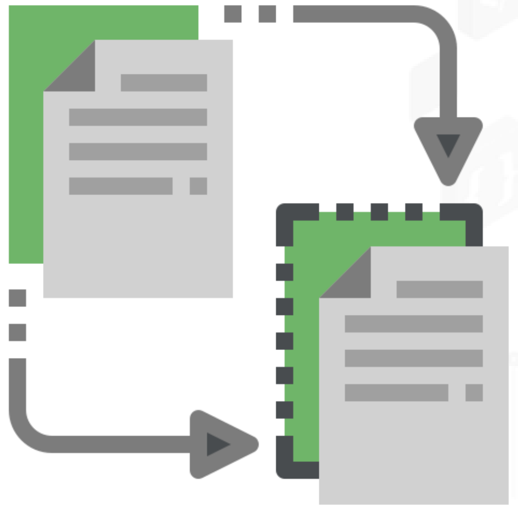
To share logic or data across components

To encapsulate external interactions like knowledge access

Angular Service: Advantages

Easy to check

Easy to debug



Reusable

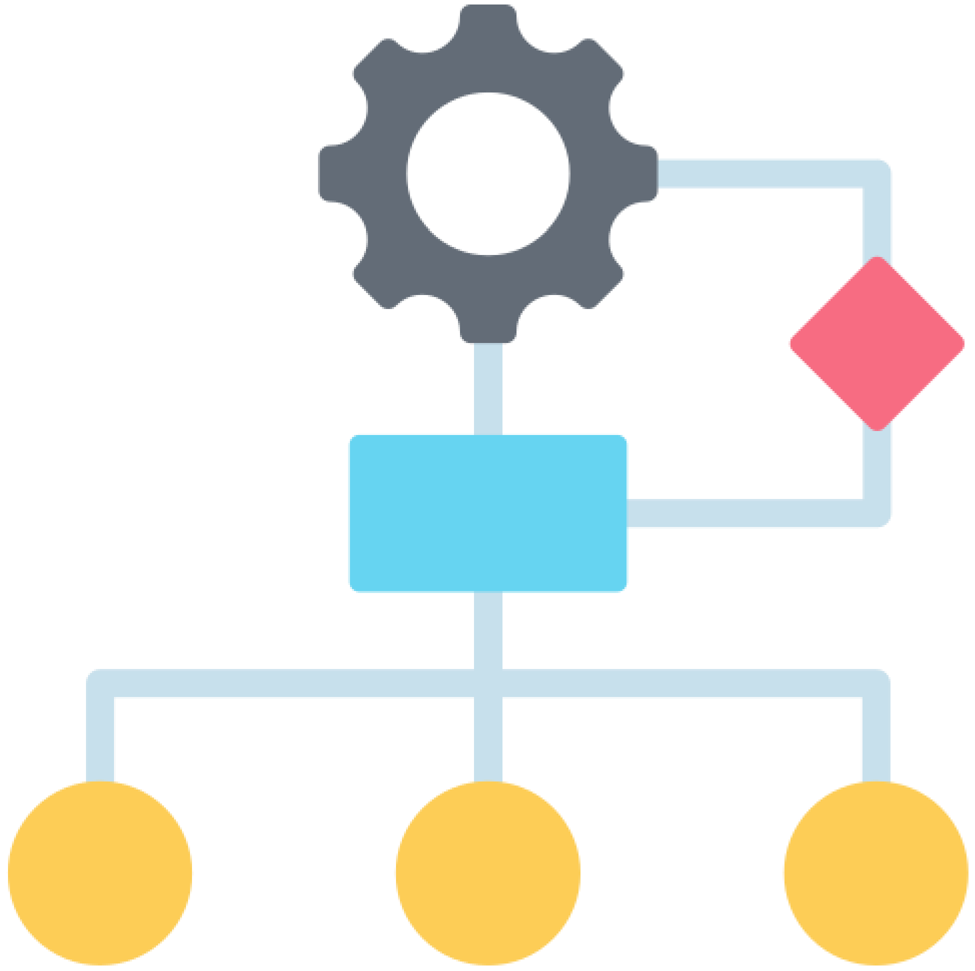
TECHNOLOGY

Angular Dependency Injection

Angular Dependency Injection

The AppComponent depends on the ProductService to display the list of products.

In the Angular Dependency Injection technique, a class gets its dependencies from external sources.



Benefits of Dependency Injection

- The component is loosely coupled to the ProductService.
- AppComponent works with the ProductService that is passed onto it.
- The AppComponent will work with any execution of ProductService that is passed on to it. One can create a mockProductService class and pass it while testing.

Benefits of Dependency Injection

- Reusing the component is easier.
- The dependency injection pattern makes the AppComponent testable, maintainable, etc.

Angular Dependency Injection Framework

Angular dependency injection framework creates and manages dependencies and injects them into components, directives, or services.

- Consumer
- Dependency
- Injection Token (DI Token)
- Provider
- Injector

It is the class (component, directive, or service) which needs the dependency.

Angular Dependency Injection Framework

Angular dependency injection framework creates and manages dependencies and injects them into components, directives, or services.

- Consumer
- **Dependency**
- Injection Token (DI Token)
- Provider
- Injector

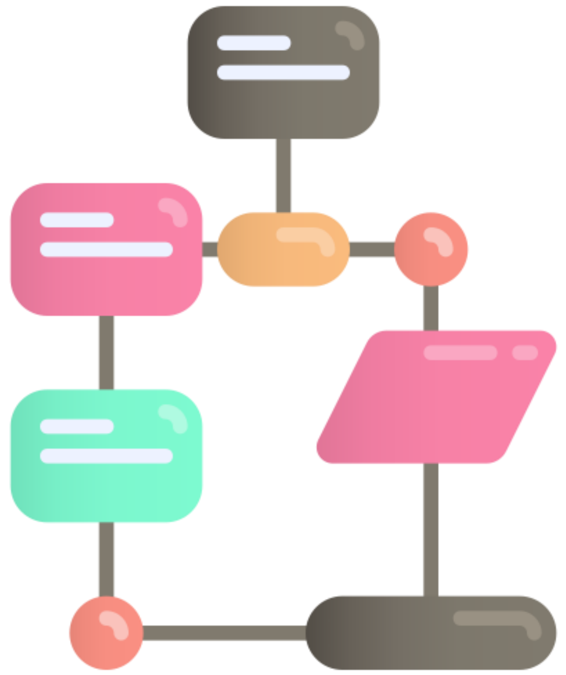
It is the service which one needs from the consumers.

Angular Dependency Injection Framework

Angular dependency injection framework creates and manages dependencies and injects them into components, directives, or services.

- Consumer
- Dependency
- Injection Token (DI Token)
- Provider
- Injector

It identifies a dependency uniquely.



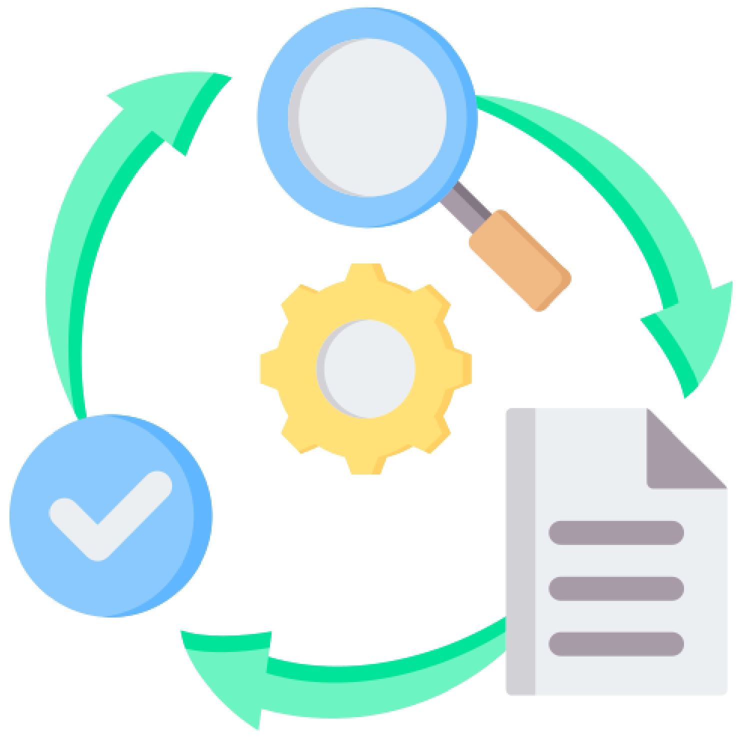
Use DI Token while registering a dependency

Angular Dependency Injection Framework

Angular dependency injection framework creates and manages dependencies and injects them into components, directives, or services.

- Consumer
- Dependency
- Injection Token (DI Token)
- Provider
- Injector

- Maintains the list of dependencies
- Identifies the dependency



Angular Dependency Injection Framework

Angular dependency injection framework creates and manages dependencies and injects them into components, directives, or services.

- Consumer
- Dependency
- Injection Token (DI Token)
- Provider
- **Injector**

- Holds the providers
- Resolves dependencies
- Injects the situation of the dependency

It uses an injection token and creates an instance of the dependency, and injects it into the consumer.

TECHNOLOGY

Built-in Pipes

simplilearn

Built-in Pipes

Pipes help in Angular to transform data like strings, currency amounts, dates, and more.

The (I) symbol is used for angular pipes.

Syntax:

```
Value_expression | Angular pipe
```

Built-in Pipes

LowerCasePipe is used to convert all text to lowercase.

The lowercase keyword is used for this pipe.

Example:

```
{{ name | lowercase }}
```

Built-in Pipes

Angular has various built-in pipes:

- AsyncPipe
- CurrencyPipe
- DatePipe
- KeyValuePipe
- LowerCasePipe

It is used for string concatenation.

Keyword: 'async'

Built-in Pipes

Angular has various built-in pipes:

- AsyncPipe
- CurrencyPipe
- DatePipe
- KeyValuePipe
- LowerCasePipe

It converts a number to a currency string.

Keyword: 'currency'

Built-in Pipes

Angular has various built-in pipes:

- AsyncPipe
- CurrencyPipe
- DatePipe
- KeyValuePipe
- LowerCasePipe

It formats date values according to locale rules.

Keyword: 'date'

Built-in Pipes

Angular has various built-in pipes:

- AsyncPipe
- CurrencyPipe
- DatePipe
- **KeyValuePipe**
- LowerCasePipe

It converts the object or maps into an array of key-value pairs.

Keyword: ‘keyvalue’

$$\begin{tikzcd} \text{Object} \arrow[r] & \text{Array} \end{tikzcd}$$

Built-in Pipes

Angular has various built-in pipes:

- AsyncPipe
- CurrencyPipe
- DatePipe
- KeyValuePipe
- LowerCasePipe

It converts all text to lowercase.

Keyword: ‘lowercase’

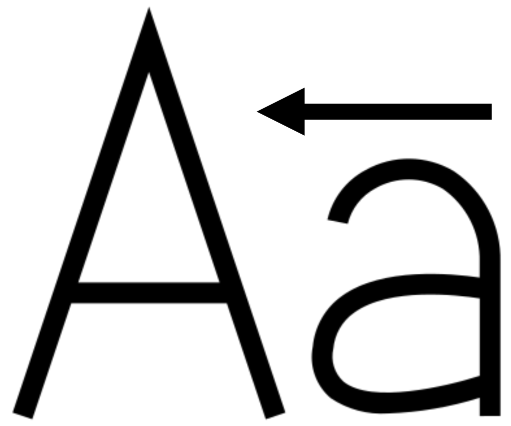
Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It converts all text to uppercase.

Keyword: 'uppercase'



Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It converts a decimal number into a String.

Keyword: ‘decimal’

$$\begin{tikzcd}\text{Decimal} \arrow[r] & \text{String}\end{tikzcd}$$

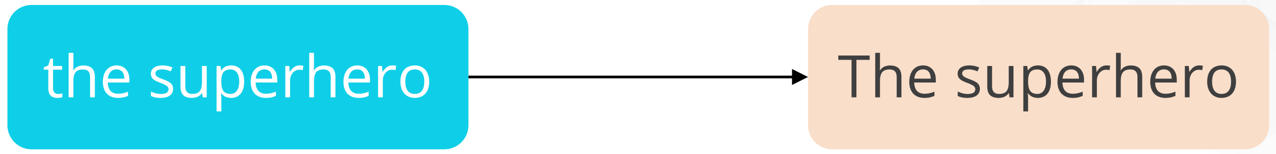
Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It converts text into a title case.

Keyword: 'title case'



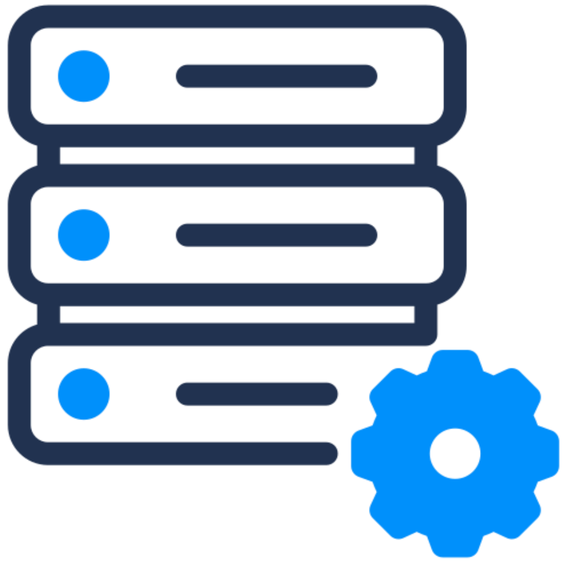
Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It creates an array or string containing a subset of the elements.

Keyword: ‘slice’



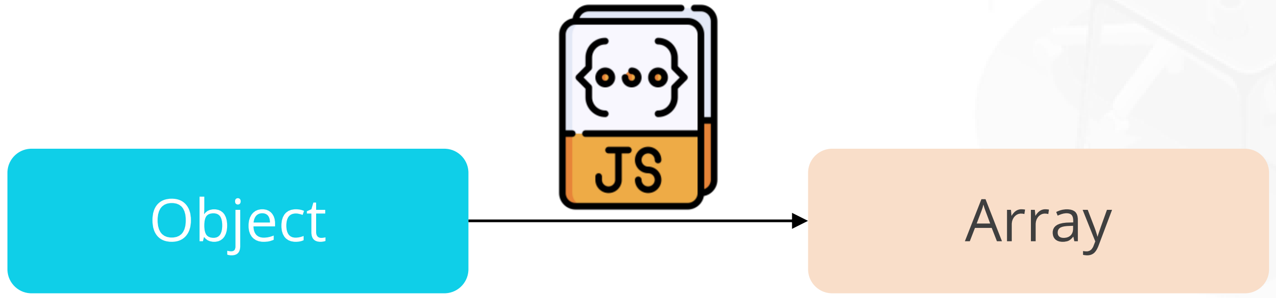
Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It transforms objects or maps into an array of key-value pairs.

Keyword: 'JSON'



Built-in Pipes

Angular has various built-in pipes:

- UppercasePipe
- DecimalPipe
- TitleCasePipe
- SlicePipe
- JSONPipe
- PercentPipe

It converts a number into a formatted percentage string.

Keyword: 'percent'

TECHNOLOGY

Angular Observables

Angular Observables

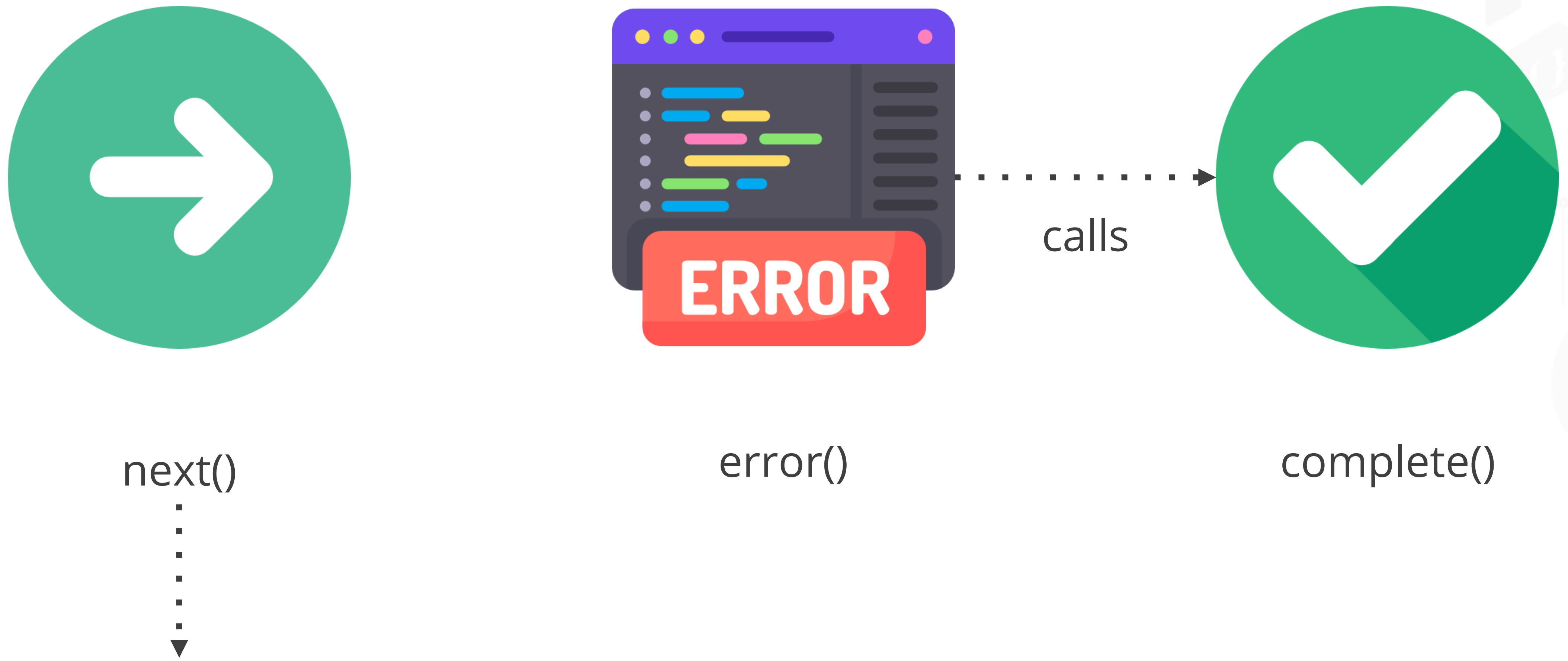
A function which changes the ordinary stream of data into an observable stream is called an observable.

- The observable stream emits values from the stream asynchronously.
- The observable stream emits a complete signal when the stream is completed, or an error signal is emitted.
- Observables are declarative and they can be defined.
- The observable emits values when someone subscribes to it.
- Observers or subscribers are the ones who consume the value emitted by the observable.

Angular Observables

The observer connects with the observable using callback functions.

The observer must subscribe to the observable to get the value from the observable.



Observable calls the next() callback as and when the value arrives

Angular Observables

Import the observable from the rxjs library using this statement:

```
Import { Observable } from ‘rxjs’;
```

Angular Observables

With the observable constructor, it is easy to create observables in Angular. It takes the observer as its argument.

The subscriber gets executed when this observable subscribe() method is called.

Use this code to create an observable:

```
Observable = new Observable((observer) => {
      console.log(“Observable starts”)
      observer.next(“5”)
      observer.next(“4”)
      observer.next(“3”)
      observer.next(“2”)
      observer.next(“1”)
                              })
```

Angular Observables

Subscribe to the observable:

```
ngOnInit() {
      this.observable.subscribe(
        val => { console.log(val) }, //next callback
        error => { console.log("error") }, //error callback
        ()=> { console.log("completed") } //complete callback
      )
}
```

Angular Observables

To unsubscribe, create a variable of type subscription:

```
obsSub = Subscription
```

Store the subscription to observable in the variable as shown:

```
obsSub = this.observable.subscribe(
        value => { console.log("Received" + this.id);
)};
```

Angular Observables

To unsubscribe the subscription, call the unsubscribe method in ngOnDestroy():

```
ngOnDestroy() {
      this.obsSub.unsubscribe();
}
```

Note

When one destroys the component, the observable is unsubscribed and cleaned up.

Key Takeaways

- Components are the building blocks of any Angular application.
- A decorator is a function which adds metadata to a class, its methods and its properties.
- Interpolation allows having expressions as a part of any string literal that can be used in HTML.
- Property binding permits the user to bind HTML element property to the property inside the component.
- Angular service is a piece of reusable code with one purpose.

TECHNOLOGY

Thank You