Angular

Ramesh Maharaddi

What is Angular

- Angular is a JavaScript framework that helps developers build applications.
- Angular is a platform that makes it easy to build applications with the web.
- Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.

Setting up development environment

- The Angular CLI is a command line interface tool that can create a project, add files, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

- Step 1. Set up the Development Environment
  - Install Node.js® and npm if they are not already on your machine.
  - node -v and npm -v in a terminal/console window
  - Then install the Angular CLI globally.
  - npm install -g @angular/cli

- Step 2. Create a new project
  - **ng new my-app**

- Step 3: Serve the application
  - cd my-app
  - **ng serve**

How angular app gets loaded and started

- Executes the main.ts file, is the file which get executed first and in this reads bootstrap module.
- Executes the app.module.ts and register the AppComponent in bootstrap
- Executes the app.module.ts, inside this reads the selector and template/templateUrl property
- Finds the app-root tag in index.html, file is served by the server
- Insert template url as per app.component.ts file. i.e.,app.component.html into body tag.

What Is an Angular Module?

- A class with an NgModule decorator.
- Modules are used to put logical boundaries in your application.
- To build separate modules to separate the functionality of your application.
- NgModule decorator is used define the imports, declarations and bootstrapping options.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

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

Imports Array

- Import array can be used to import the functionality from other Angular JS modules.
- Importing a module makes available any exported components, directives, and pipes from that module.
- Only import what this module needs.

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

imports: [
  BrowserModule,
  HttpModule,
  FormsModule
],
```

Declarations Array

- Every component, directive, and pipe we create must belong to one and only one Angular module.
- Only declare components, directives, services and pipes.
- Never re-declare components, directives, or pipes that belong to another module.
- All declared components, directives, and pipes are private by default.

```
declarations: [
  AppComponent,
  TestComponent,
  MyComponentComponent
],
```

Bootstrap Array

- Root component that Angular creates and inserts into the index.html host web page.
- The application launches by bootstrapping the root AppModule.
- Which is also referred to as an entryComponent.
- Among other things, the bootstrapping process creates the component(s) listed in the bootstrap array and inserts each one into the browser DOM.

```
bootstrap: [AppComponent]
```

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

Providers Array

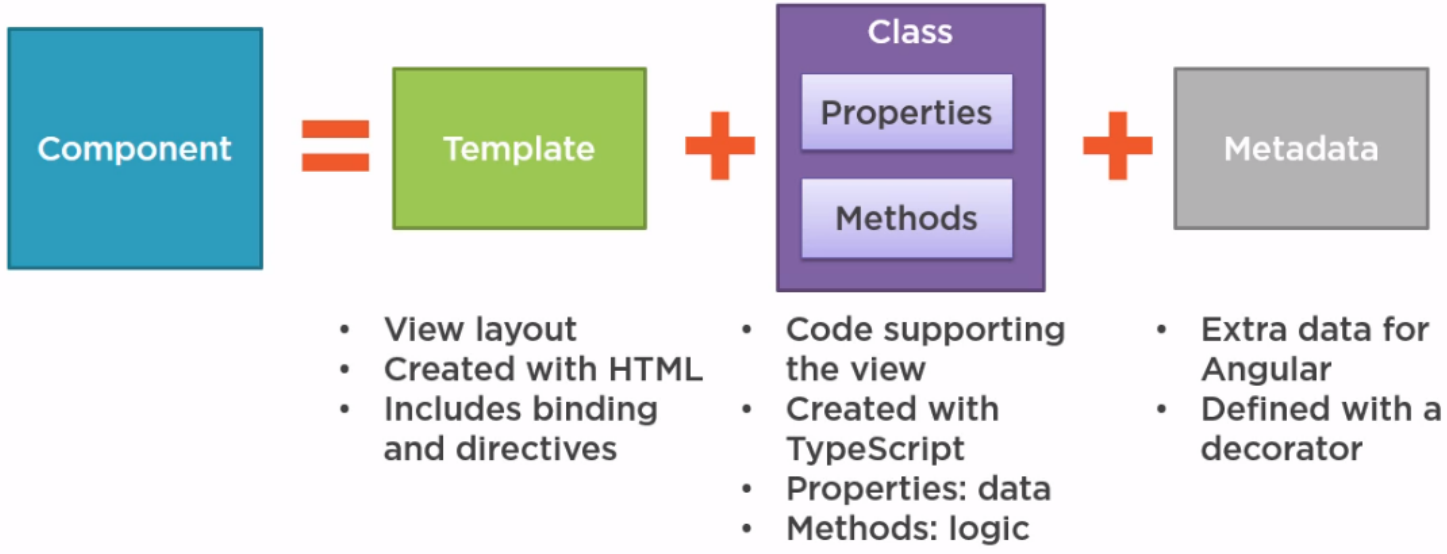
- Any service provider added to the providers array is registered at the root of the application.
- Don't add services to the providers array of a shared module.
- Consider building a CoreModule for services and importing it once again in the AppModule.

```
import { UserService } from './user.service';


@NgModule({
  providers: [UserService],
})
```

Component

What is Component?

- Components are a logical piece of code for Angular JS application.
- Component consist of these items



How component looks

app.component.ts

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

@Component({
    selector: 'pm-root',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = '      Product Management';
}
```

Import

Metadata & Template

Class

Creating new component

How to create an component?

Manually

- Using the @Component() decorator which can be imported from @angular/core
- @Component() decorator that takes information about the HTML view to use for the component and the CSS styles
- Update declarations in app.module

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

@Component({
    selector: 'app-test',
    template: 'Hello from test component'
})
export class TestComponent {

}
```

Using CLI

- Generate a component with the Angular CLI
- ng generate component MyComponent

```
C:\Users\Ramesh\Desktop\ppt\Lab\app1>ng generate component MyComponent
CREATE src/app/my-component/my-component.component.html (31 bytes)
CREATE src/app/my-component/my-component.component.spec.ts (664 bytes)
CREATE src/app/my-component/my-component.component.ts (292 bytes)
CREATE src/app/my-component/my-component.component.css (0 bytes)
UPDATE src/app/app.module.ts (487 bytes)
```

Defining a template in a component

Inline Template

```
template:
"<h1>{{pageTitle}}</h1>"
```

Inline Template

```
template: `
<div>
 <h1>{{pageTitle}}</h1>
 <div>
     My First Component
 </div>
</div>
`
```

ES 2015 Back Ticks

Linked Template

```
templateUrl:
'./product-list.component.html'
```

Defining a styles in a component

styles

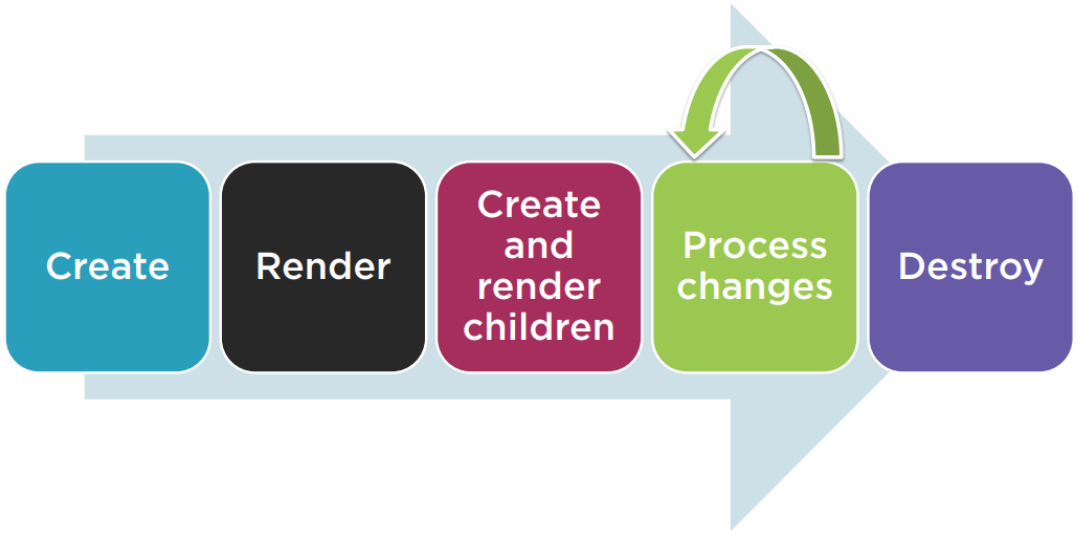
```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styleUrls: ['./product-list.component.css']})
```

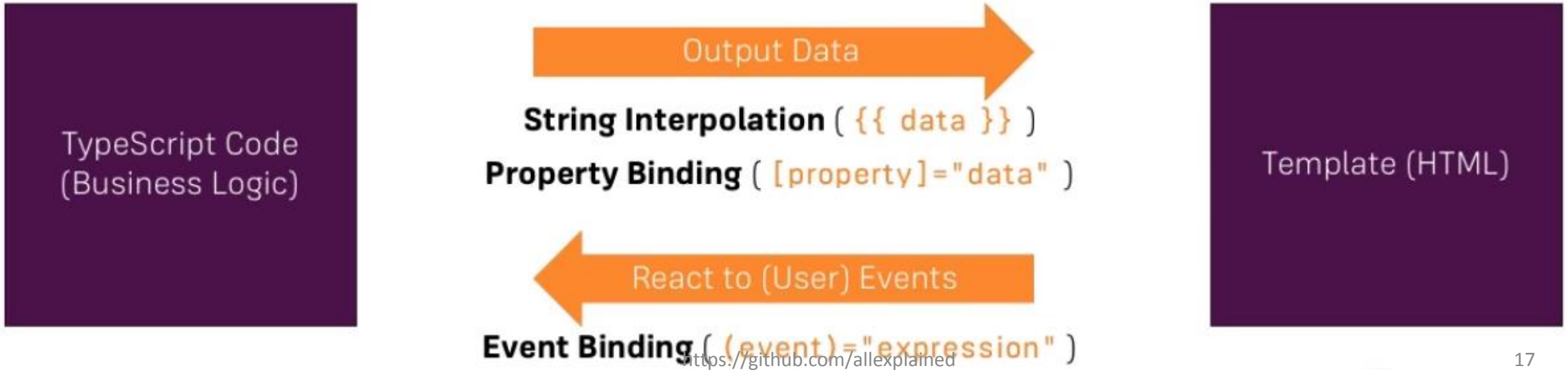
Component Lifecycle Hooks

- Constructor: This is invoked when Angular creates a component or directive by calling new on the class.
- OnInit: Perform component initialization, retrieve data
- OnChanges: Perform action after change to input properties
- OnDestroy: Perform cleanup



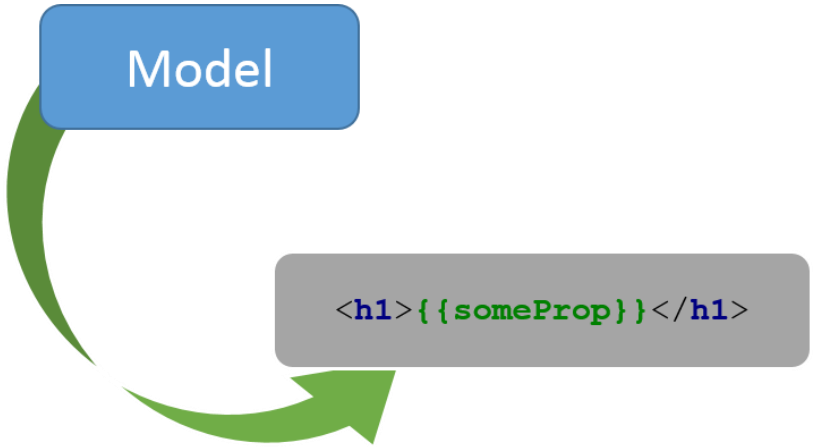
What is Databinding?

- Data binding is a core concept in Angular .
  - One-way data binding
  - Two-way data binding
- Allows to define communication between a component and the DOM and vice versa.
- Data binding making it very easy to define interactive applications without worrying about pushing and pulling data.



String Interpolation

- From the Component to the DOM
- This adds the value of a property from the component
- Syntax : {{data}}
- Any expression which can be resolved as string, that is the only condition.
- You can call property, method and ternary expression.
- You cannot write multiline, block, condition statements



- Examples:

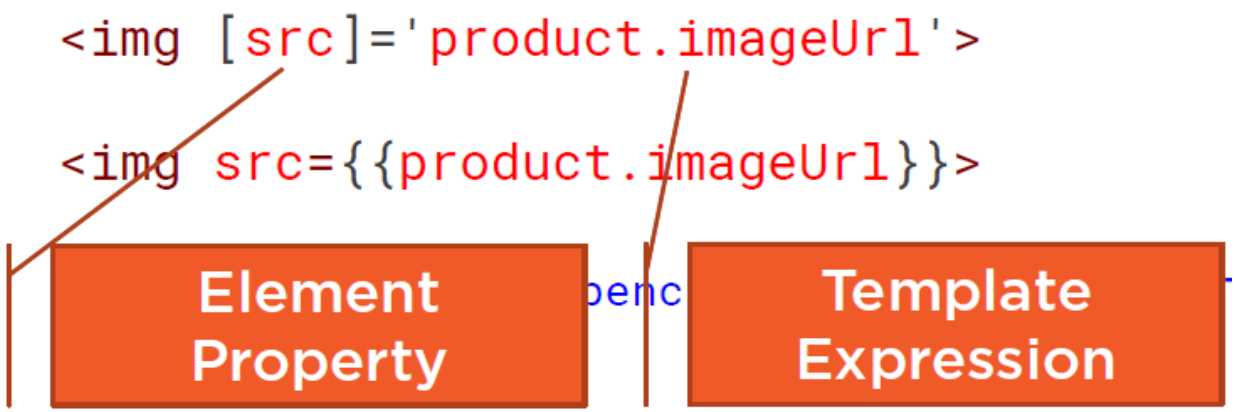
```
{{pageTitle}}
{{getTitle()}}
{{'Title: ' + pageTitle}}
{{'Title: ' + getTitle()}}
{{10*20+1}}
<h1 innerText={{PateTItle}}></h1>
```

Property binding

- Component to the DOM
- With property binding, the value is passed from the component to the specified property, which can often be a simple html attribute:
- Syntax : [property]=”value”
- Examples, one that applies a background-color from the value of selectedColor in the component and one that applies a class name if isSelected evaluates to true:

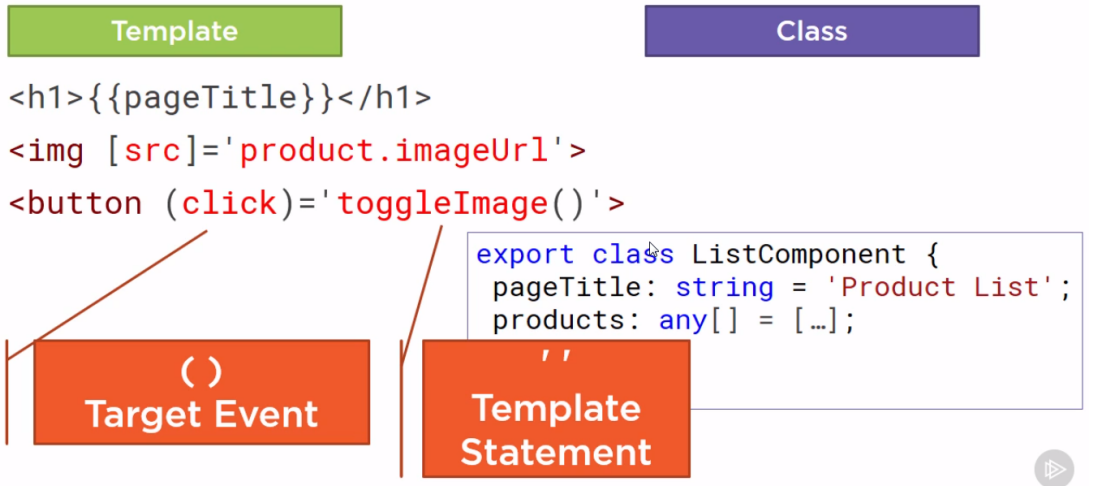
<div [style.background-color]="selectedColor">

- <div [class.selected]="isSelected">



Event binding

- From the DOM to the Component.
- Syntax: (event)=”function()”
- When a specific DOM event happens (eg.: click, change, keyup), call the specified method in the component.
- In the example below, the toggleImage() method from the component is called when the button is clicked:
- <button (click)=“toggleImage()"></button>



Two-way Data Binding

- Two way data binding where we are able to react event and output something at the same time.
- Syntax: [(ngModel)]=”value”
- You need to import **FormsModule** for two way data binding from forms library
- Component to the DOM and DOM to the Component
- Using the banana in a box syntax, two-way data binding allows to have the data flow both ways.
- The email data property is used as the value for the input. If the user changes the value, the component property gets updated automatically to the new value
- <input type="email" [(ngModel)]=“email">

Template

```
<input [(ngModel)]='listFilter'>
```

Class

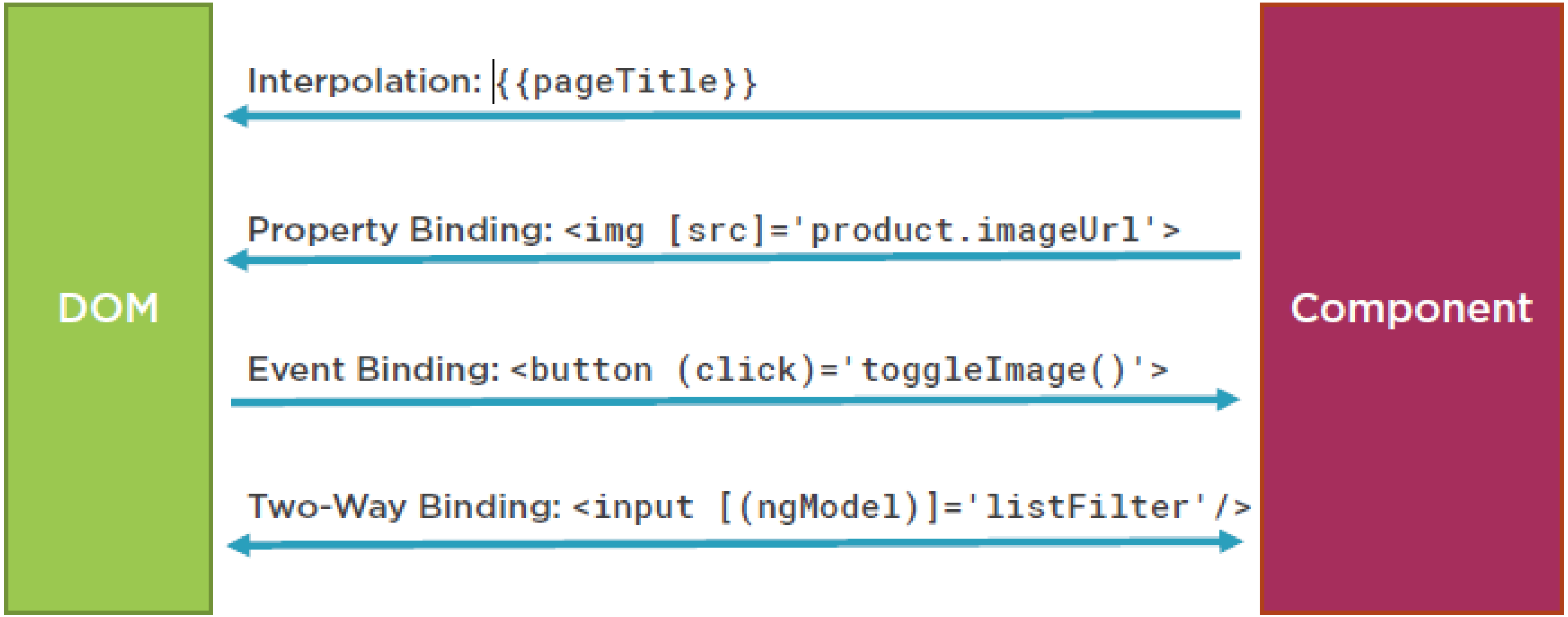
```
export class ListComponent {
 listFilter: string = 'cart';
}
```

[()]
Banana in a Box

app.module.ts

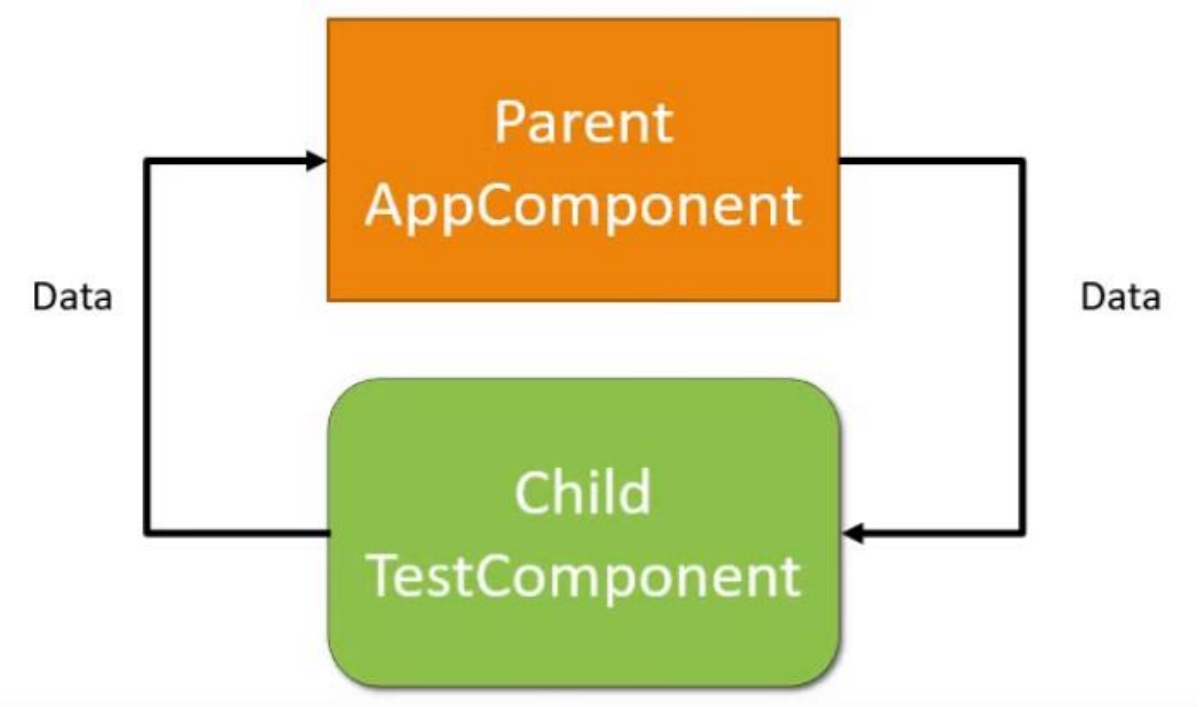
```
@NgModule({
  imports: [
      BrowserModule,
      FormsModule ],
  declarations: [
      AppComponent,
      ProductListComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Data binding - Flow



Component Interaction

- @input() – Pass data to child component
- @output() – Pass data to Parent



Directives

Instructions in the DOM

Custom HTML element or attribute used to power up and extend our HTML.

- Custom
- Built-In

Structural Directives

app.component.ts

```
@Component({
  selector: 'pm-root',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    `
})
export class AppComponent { }
```

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateURL:
    './product-list.component.html'
})
export class ProductListComponent { }
```

Custom Directives

app.component.ts

```
@Component({
  selector: 'pm-root',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    `
})
export class AppComponent { }
```

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateURL:
    './product-list.component.html'
})
export class ProductListComponent { }
```

Built-in Directive - *ngIf

Expression is evaluated as a true or false

Conditionally includes a template based on the value of an expression.

ngIf evaluates the expression and then renders the then or else template in its place.

Then template is the inline template of ngIf unless bound to a different value else template is blank unless it is bound.

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

show = {{show}}

<div *ngIf="show">Text to show</div>
```

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

show = {{show}}

<div *ngIf="show; else elseBlock">Text to show</div>

<ng-template #elseBlock>Alternate text while primary text is hidden</ng-template>
```

Built-in Directive - *ngFor

Makes it easy to iterate over something like an array or an object.

ngFor provides several exported values that can be aliased to local variables

```
<ul>
  <li *ngFor="let user of users">{{ user.name }}</li>
</ul>
```

index: number: The index of the current item

first: boolean: True when the item is the first item

last: boolean: True when the item is the last item

even: boolean: True when the item has an even

odd: boolean: True when the item has an odd

```
<ul>
  <li *ngFor="let user of users; let i = index; let odd = odd"
      [class.odd]="odd">
    {{i + 1}}. {{ user.name }}
  </li>
</ul>
```

Built-in Directive - *ngSwitch

Like ngFor and ngIf, ngSwitch is a built-in template directive. It behaves in a similar way as a JavaScript switch statement. Use it to include one of multiple possible element trees in the DOM.

```
<div [ngSwitch]="dietSelection">
  <p *ngSwitchCase="'gf'">Gluten-free</p>
  <p *ngSwitchCase="'veg'">Vegetarian / Vegan</p>
  <p *ngSwitchDefault>Standard diet</p>
</div>
```

```
<ul [ngSwitch]="person">
  <li *ngSwitchCase="'Mohan'">Hello Mohan</li>
  <li *ngSwitchCase="'Sohan'">Hello Sohan</li>
  <li *ngSwitchCase="'Vijay'">Hello Vijay</li>
  <li *ngSwitchDefault>Bye Bye</li>
</ul>
```

ngStyle

The NgStyle directive lets you set a given DOM elements style properties.

ngStyle becomes much more useful when the value is dynamic.

```
<div [ngStyle]="{'background-color':'green'}"></<div>

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

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

ngClass

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

```
[ngClass]="{'text-success':true}"
[ngClass]="{'text-success':person.country === 'UK'}“
```

```
<p [ngClass]="['warning', 'big']">
   warning', 'big‘
</p>
```

```
          <p [ngClass]="{ card: true, dark: false, flat: flat }">
    <ng-content></ng-content>
    <br>
    <button type="button" (click)="flat=!flat">Toggle Flat</button>
   </p>
```

ViewChild

To access child component's, directive or a DOM element from a parent component use ViewChild decorator.

ViewChild returns the first element that matches a given component, directive or template reference selector.

```
<input #someInput placeholder="Your pizza toping">
@ViewChild('someInput') someInput: ElementRef;
  ngAfterViewInit() {
    this.someInput.nativeElement.value = "Anchovies!";
  }


import { Directive, ElementRef, Renderer2 } from
'@angular/core';
@Directive({ selector: '[appBacon]' })
export class BaconDirective {
  ingredient = 'mayo';
  constructor(elem: ElementRef, renderer: Renderer2) {
    let bacon = renderer.createText(' bacon');
    renderer.appendChild(elem.nativeElement, bacon);
  }
}
<span appBacon>sandwich!</span>
```

Handling User input

```
template: `
  <input (keyup)="onKey($event)">
  <p>{{values}}</p>
`
```

```
export class KeyUpComponent_v1 {
  values = '';

  onKey(event: any) { // without type info
    this.values += event.target.value + ' | ';
  }
}
```

```
onKey(event: KeyboardEvent) { // with type info
  this.values += (<HTMLInputElement>event.target).value + ' | ';
}
```

Promises

Promises are a new feature in the ES6 (ES2015) JavaScript.

That allow you to deal with asynchronous code without resolving to multiple levels of callback functions.

With a Promise you can only handle one event

Promises always return only one value.

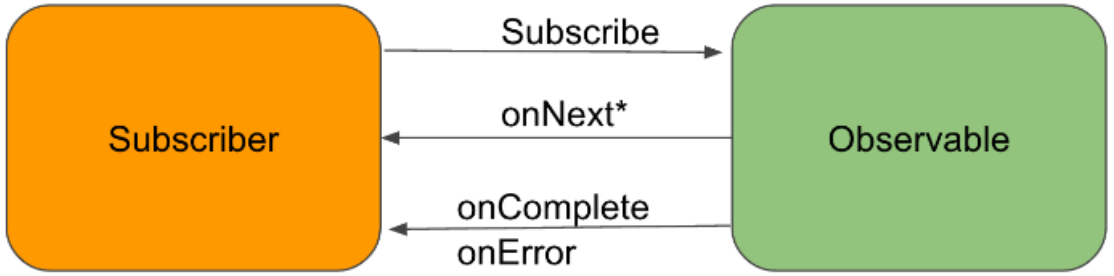
Another thing is that promises are not cancellable.

```
let myPromise = new Promise((resolve, reject) => {
  let data;
  setTimeout(() => {
    data = "Some payload";

    if (data) {
      resolve(data);
    } else {
      reject();
    }
  }, 2000);
});
```

```
myPromise.then(data => {
  console.log('Received: ' + data);
}).catch(() => {
  console.log("There was an error");
});
```

Observables(RxJS)

- Observables are powerful way to compose asynchronous code.
- RxJS is all about unifying the ideas of Promises
- An Observable is an sequence of events over time.
- It is not executed until a consumer subscribes
- It has at least two participants.
  - The creator (the data source)
  - subscriber
- Many Angular APIs use Observables
  - The EventEmitter.
  - The HTTP module uses observables to handle AJAX requests and responses.
  - The Router and Forms modules use observables to listen for and respond to user-input events.



Promise	Observable
Provides a single future value	Emits multiple values over time
Not lazy	Lazy
Not cancellable	Cancellable
	Supports map, filter, reduce and similar operators

Defining observers

A handler for receiving observable notifications implements the Observer interface.

It is an object that defines callback methods to handle the three types of notifications that an observable can send.

An Observable instance begins publishing values only when someone subscribes to it.

You subscribe by calling the subscribe() method of the instance, passing an observer object to receive the notifications.

Function	Description
Next	A handler for each delivered value
Error	A handler for an error notification
complete	A handler for the execution - complete notification

Example

Create Observer

Subscribe

Unsubscribe

```
myObservableSubscription: Subscription;
myObservable = interval(1000);
```

```
this.myObservableSubscription = this.myObservable.subscribe(
  x => console.log('Observer got a next value: ' + x),
  err => console.error('Observer got an error: ' + err),
  () => console.log('Observer got a complete notification')
);
```

```
this.myObservableSubscription.unsubscribe();
```

RxJs Operator

switchMap	Cancels the current subscription/request and can cause race condition **Use for get requests or cancelable requests like searches**
concatMap	Runs subscriptions/requests in order and is less performant **Use for get, post and put requests when order is important**
mergeMap	Runs subscriptions/requests in parallel **Use for put, post and delete methods when order is not important**
exhaustMap	Ignores all subsequent subscriptions/requests until it completes **Use for login when you do not want more requests until the initial one is complete**

Services and Dependency Injection

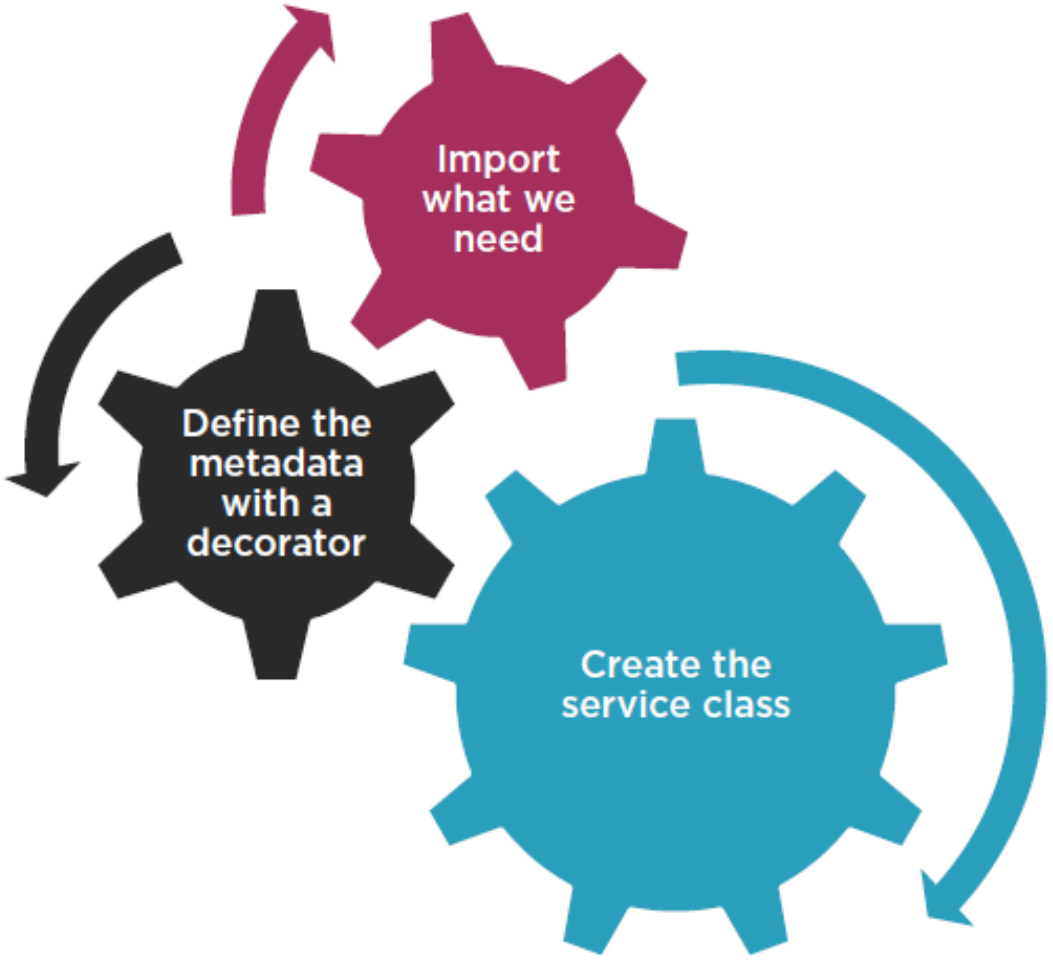
A class with a focused purpose.

Are independent from any particular component

Provide shared data or logic across components

Loosely coupled code

More flexible code



Creating service

Create service using CLI

ng generate service calculator

ng g s calculator

Angular CLI command ng generate service

Registers a provider with the root injector

Including provider metadata in the @Injectable() decorator.

```
C:\Users\Ramesh\Desktop\ppt\Lab\app1>ng g s calculator
CREATE src/app/calculator.service.spec.ts (398 bytes)
CREATE src/app/calculator.service.ts (139 bytes)
```

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

@Injectable({
  providedIn: 'root'
})
export class CalculatorService {

  constructor() { }

  getSum(n1: number, n2: number): number {
    return n1 + n2;
  }

}
```

Registering a Service

Root Injector

Service is available throughout the application

Recommended for most scenarios

Component Injector

Service is available ONLY to that component and its child (nested) components

Isolates a service used by only one component

Provides multiple instances of the service

product.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class ProductService { }
```

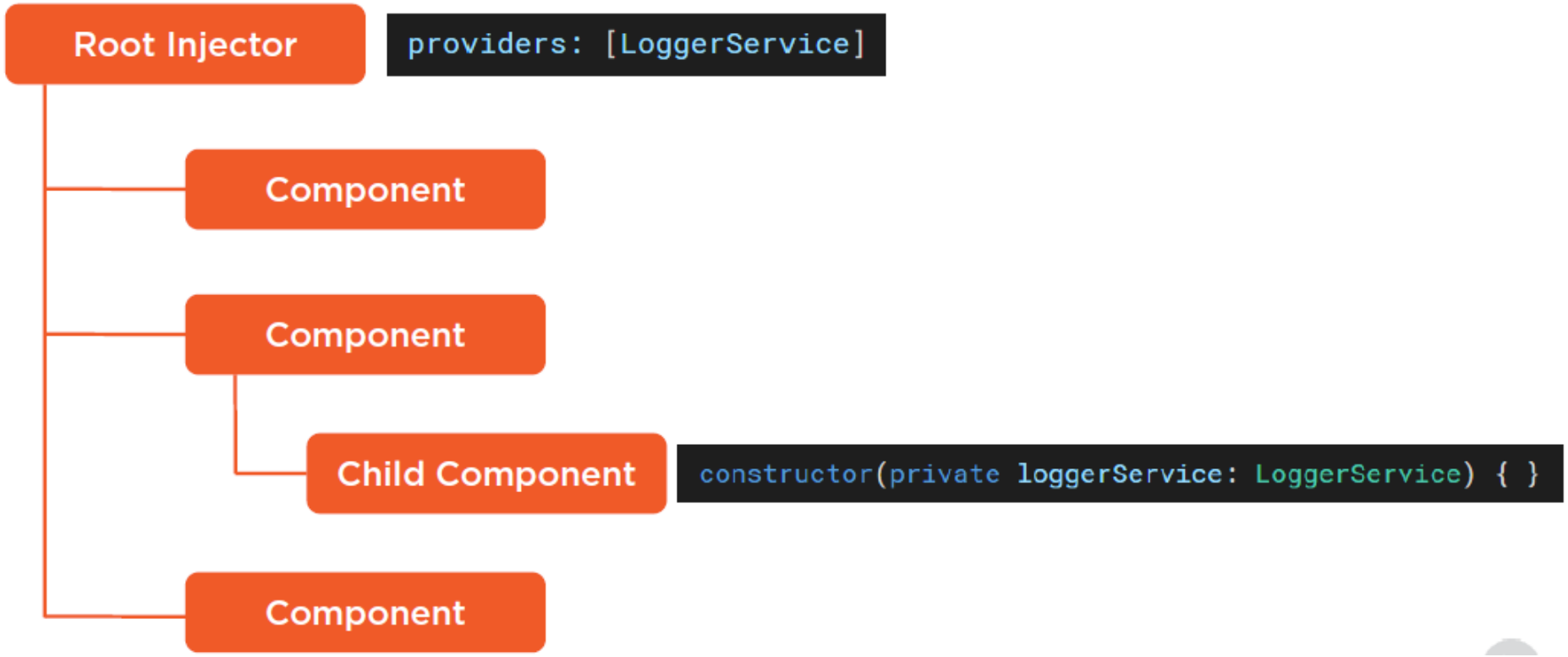
product-list.component.ts

```
@Component({
  templateUrl: './product-list.component.html',
  providers: [ProductService]
})
export class ProductListComponent { }
```

app.module.ts

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

Hierarchical Injectors



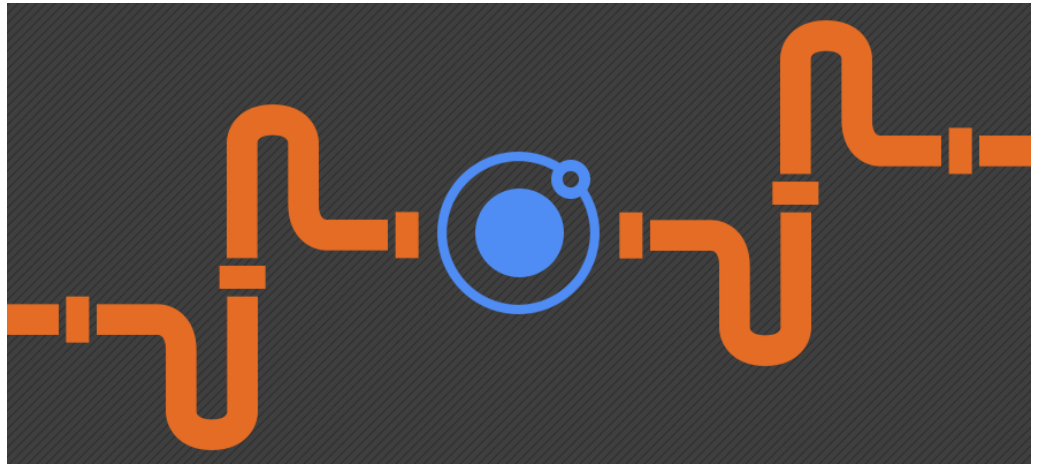
Pipes

Great form and format data right from your templates

Transform bound properties before display

Built-in pipes

Custom pipes



Async pipes

The Async pipe automatically subscribes to an Observable or a Promise and returns the emitted values as they come in

```
<p>
    {{ observable | async }}
</p>
```

```
observable: Observable<number> = interval(1000);
```

Date pipes

Format date values with the Date pipe:

```
{{ someDate | date:'medium' }}
{{ someDate | date:'fullDate' }}
{{ someDate | date:'yy' }}
{{ someDate | date:'Hm' }}
```

You can use a number of symbols to define a custom format, or you can also use a number of predefined keywords.

The available keywords are the following:

'medium', 'short', 'fullDate', 'longDate', 'mediumDate', 'shortDate', 'mediumTime' and 'shortTime'.

Currency pipes

The Currency pipe allows to format numbers in different currencies:

```
{{ price | currency:'CAD' }}
{{ price | currency:'USD':true }}
{{ price | currency:'EUR':false:3.2-2 }}
```

The first argument is a string with the local currency code.

The second possible argument is a Boolean to show the that will show either the currency symbol, or the currency code. The default is false and shows the currency code.

Others

Decimal

{{ decimalValue | number:'4.3-5' }}

LowerCase & UpperCase

{{ user.name | uppercase }}

{{ user.name | lowercase }}

Percent

{{ decimalValue | percent }}

{{ decimalValue | percent:'3.2-3' }}

Slice

{{ 'welcome to pipes' | slice:3:6 }}

Custom pipes

Create pipe using cli command

ng generate pipe pipes/symbol --spec false

Ng g p pipes/symbol --spec false

Calling this pipe

{{'Ramesh' | symbol:'*'}}

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'symbol'
})
export class SymbolPipe implements PipeTransform {
  transform(value: string, symbol?: string): string {
    return symbol + ' ' + value;
  }
}
```

Bootstrap with Angular

angular.json

```
"styles": [
"node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.scss"
]
```

Or

src/style.css:

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

```
C:\Users\Ramesh\Desktop\ppt\Lab\app1>npm install bootstrap
npm WARN rm not removing C:\Users\Ramesh\Desktop\ppt\Lab\app1\nod
\ppt\Lab\app1\node_modules\semver
npm WARN rm not removing C:\Users\Ramesh\Desktop\ppt\Lab\app1\nod
\Lab\app1\node_modules\semver
```

```
C:\Users\Ramesh\Desktop\ppt\Lab\app1>yarn add bootstrap
yarn add v1.9.4
[1/4] Resolving packages...
[2/4] Fetching packages...
info fsevents@1.2.4: The platform "win32" is incompatible with this module.
info "fsevents@1.2.4" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
warning " > bootstrap@4.1.3" has unmet peer dependency "jquery@1.9.1 - 3".
warning " > bootstrap@4.1.3" has unmet peer dependency "popper.js@^1.14.3".
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└─ bootstrap@4.1.3
info All dependencies
└─ bootstrap@4.1.3
Done in 18.70s.
```

Http/HttpClient

HttpClient provides some well-awaited features.

It exists as a separate module called @angular/common/http to ensure retro compatibility

Http client, living in the module @angular/http.

ng g i http/customer

CREATE src/app/http/customer.ts

```
imports: [
  BrowserModule,
  // HttpModule,
  FormsModule,
  HttpClientModule
],
```

```
// import { HttpModule } from '@angular/http';
import {HttpClientModule } from '@angular/common/http';
```

Http Get Methods

```
/** GET customer from the server */
getCustomer(): Observable<Customer[]> {
  return this.http.get<Customer[]>(this.url)
    .pipe(
      tap(customers => this.log('fetched customers')),
      catchError(this.handleError('getCustomer', []))
    );
}
```

```
<div class="row">
    <div class="col">
        <ul class="list-group">
            <li class="list-group-item list-group-item-action" *ngFor="let customer of customers">
                <span class="badge">{{customer.id}}</span> {{customer.name}}
            </li>
        </ul>
    </div>
</div>
```

Http Post, Delete, Put

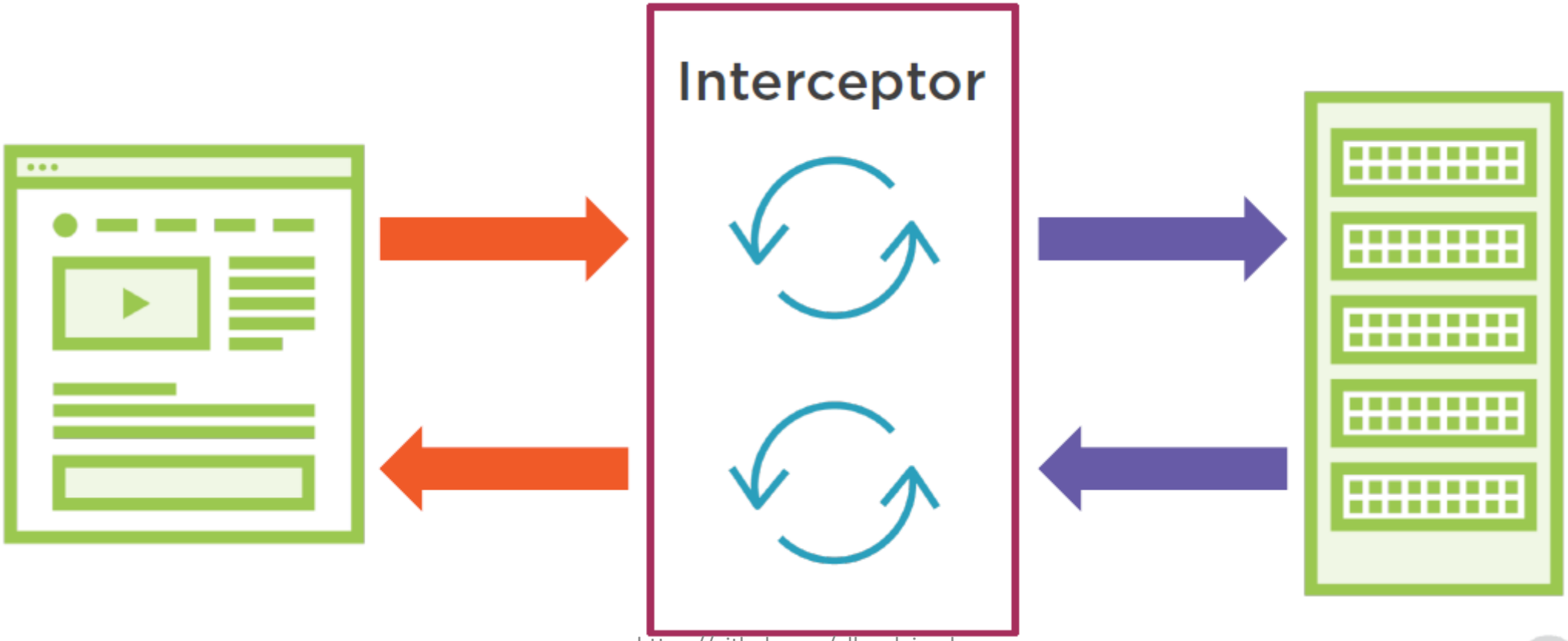
```
/** POST: add a new customer to the server */
saveCustomer(customerData: Customer): Observable<Customer> {
  console.log('Received Data' + customerData);
  return this.http.post<Customer>(this.url, customerData, this.httpOptions).pipe(
    tap((customer: Customer) => this.log(`added customer w/ id=${customerData}`)),
    catchError(this.handleError<Customer>('saveCustomer'))
  );
}

/** DELETE: delete the customer from the server */
deleteCustomer(customer: Customer | number): Observable<Customer> {
  return this.http.delete<Customer>(this.url, this.httpOptions).pipe(
    tap(_ => this.log(`deleted hero id=${customer}`)),
    catchError(this.handleError<Customer>('deleteCustomer'))
  );
}

/** PUT: update the customer on the server */
updateCustomer(customer: Customer): Observable<any> {
  return this.http.put(this.url, customer, this.httpOptions).pipe(
    tap(_ => this.log(`updated hero id=${customer.id}`)),
    catchError(this.handleError<any>('updateCustomer'))
  );
}
```

Interceptor

Create Interceptor

ng g s http/FirstInterceptor --spec false

CREATE src/app/http/first-interceptor.service.ts (145 bytes)



Creating Interceptor

Interceptor are like services, Implements HttpInterceptor

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<any> {
    console.log(`Requested URl ${req.url}`);
    const newReq: HttpRequest<any> = req.clone({
      setHeaders: { 'Content-Type': 'application/json' }
    });
    return next.handle(newReq);
  }
}
```

```
import { FirstInterceptor } from './http/first-interceptor';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';

providers: [
  { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true }
],
```

Routing & Navigation

The Angular Router enables navigation from one view to the next as users perform application tasks.

Navigating the Application Routes

Menu option, link, image or button that activates a route

Typing the Url in the address bar / bookmark

The browser's forward or back buttons

Configuring Routes

app.module.ts

```
...
import { RouterModule } from '@angular/router';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule.forRoot([], { useHash: true })
  ],
  declarations: [
    ...
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Configuring Routes

```
[
 { path: 'products', component: ProductListComponent },
 { path: 'products/:id', component: ProductDetailComponent },
 { path: 'welcome', component: WelcomeComponent },
 { path: '', redirectTo: 'welcome', pathMatch: 'full' },
 { path: '**', component: PageNotFoundComponent }
]
```

```
RouterModule.forRoot([
  { path: 'products', component: ProductListComponent },
  { path: 'products/:id', component: ProductDetailComponent },
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: 'welcome', pathMatch: 'full' },
  { path: '**', redirectTo: 'welcome', pathMatch: 'full' }
])
```

Passing Parameter to Route

product-list.component.html

```
<td>
  <a [routerLink]="['/products', product.productId]">
     {{product.productName}}
  </a>
</td>
```

app.module.ts

```
{ path: 'products/:id', component: ProductDetailComponent }
```

Assignment-Routing – Components generation

- Generate separate module(routing, user, product)
  - ng g m modules/routing --flat --spec false
  - Ng g m modules/user --flat --spec false
  - ng g m modules/product --flat --spec false
- User and product should import RouterModule
- Generate Home component- This will be a landing component
  - ng g c home --spec false
  - Add <app-home></app-home> to app.component.html
- Generate user components (Login, Register) in user modules
  - ng g c user/UserHome -m modules/user --spec false
  - ng g c user/login -m modules/user --spec false
  - ng g c user/register -m modules/user --spec false
  - ng g c user/AllUser -m modules/user --spec false
  - ng g c user/UserProfile -m modules/user --spec false
  - user.module should be updated with
    - declarations: [LoginComponent, RegisterComponent, UserHome]
- Generate product in user modules
  - ng g c products/ProductAdd -m modules/product --spec false
  - product.module should be updated with
  - declarations: [ProductAddComponent]
  - exports: [ProductAddComponent]
- Generate Service
  - ng g s services/User --spec false
  - ng g s services/Auth --spec false

Assignment-Routing – Configuration routing.module.ts

```
import { RouterModule, Routes } from '@angular/router';

@NgModule({
imports: [
CommonModule,
RouterModule.forRoot( [
{path: 'home', component: HomeComponent },
……
……
]) ],
exports: [RouterModule]
})
```

<router-outlet></router-outlet> In app.component.html

Assignment-Routing – Creating all routs routing.module.ts

```
const routes: Routes = [
{path: 'home', component: HomeComponent},
{path: 'login', component: LoginComponent},
{path: 'userHome', component: UserHomeComponent},
{path: 'product', component: ProductComponent},
{path: '', redirectTo: 'home', pathMatch: 'full'},
{path: '**', redirectTo: 'home', pathMatch: 'full'}
];

RouterModule.forRoot(routes) ],
exports: [RouterModule]
})
```

Assignment-Routing – routerLink

```
<a class="flex-sm-fill text-sm-center nav-link" [routerLink]="['/home']">Home</a>
<a class="flex-sm-fill text-sm-center nav-link" [routerLink]="['/login']">Login</a>
<a class="flex-sm-fill text-sm-center nav-link" [routerLink]="['/userHome']">User</a>
<a class="flex-sm-fill text-sm-center nav-link" [routerLink]="['/product']">Product</a>
```

Assignment-Routing – ChildRoutes

```
{path: 'userHome', component: UserHomeComponent, children: [
        {path: 'register', component: RegisterComponent},
        {path: 'userList', component: AllUserComponent},
        {path: 'profile', component: UserProfileComponent}
]},
```

```
<a [routerLink]="['/userHome/register']">Register</a>
<a [routerLink]="['/userHome/userList']">User List</a>
<a [routerLink]="['/userHome/profile']">User Profile</a>
```

Add <router-outlet></router-outlet>

Assignment-Routing – Navigating to Other Component

```
<button type="button" class="btn btn-link" (click)="existingUserLogin()">
        Existing User? Login Here
</button>

import { Router } from '@angular/router';

constructor(private router: Router) { }

existingUserLogin() {
        this.router.navigate(['/login']);
}
```

Homework: Not existing user, Register Here

Assignment-Routing – Passing and reading route parameter

Passing Parameter:

```
<a[routerLink]="['/userHome/profile/MyUserIdParam']">
        ramId
</a>
{path: 'profile/:userName', component: UserProfileComponent}
```

Reading: Parameter:

```
import { ActivatedRoute } from '@angular/router';
constructor(private activatedRoute: ActivatedRoute) { }

 ngOnInit() {
        this.userName = this.activatedRoute.snapshot.params['userName'];
        this.activatedRoute.params.subscribe((params: Params) => this.userName = params['userName']);

}
```

Assignment-Routing – Passing and reading query parameter

Passing Parameter: - No changes in routing module

```
<a [routerLink]="['/userHome/profile',user.userName]"
    [queryParams]="{fname: user.fname, lname: user.lname, email: user.email}">
            Edit
</a>
```

Reading:

```
profile = { };
// console.log(this.activatedRoute.snapshot.queryParams['fname']);  //Using snapshot
//Using Observables
this.activatedRoute.queryParams.subscribe((params: Params) =>
   this.profile = {
   fname: params['fname'],
   lname: params['lname'],
  email: params['email'],
 }
);
```

Protecting Routes with Guards

CanActivate
: Guard navigation to a route

CanDeactivate
: Guard navigation from a route

Resolve
: Pre-fetch data before activating a route

CanLoad
: Prevent asynchronous routing

Creating Guards : CanActivate and CanActivateChild

How to create guard?
ng g g guards/User --spec false

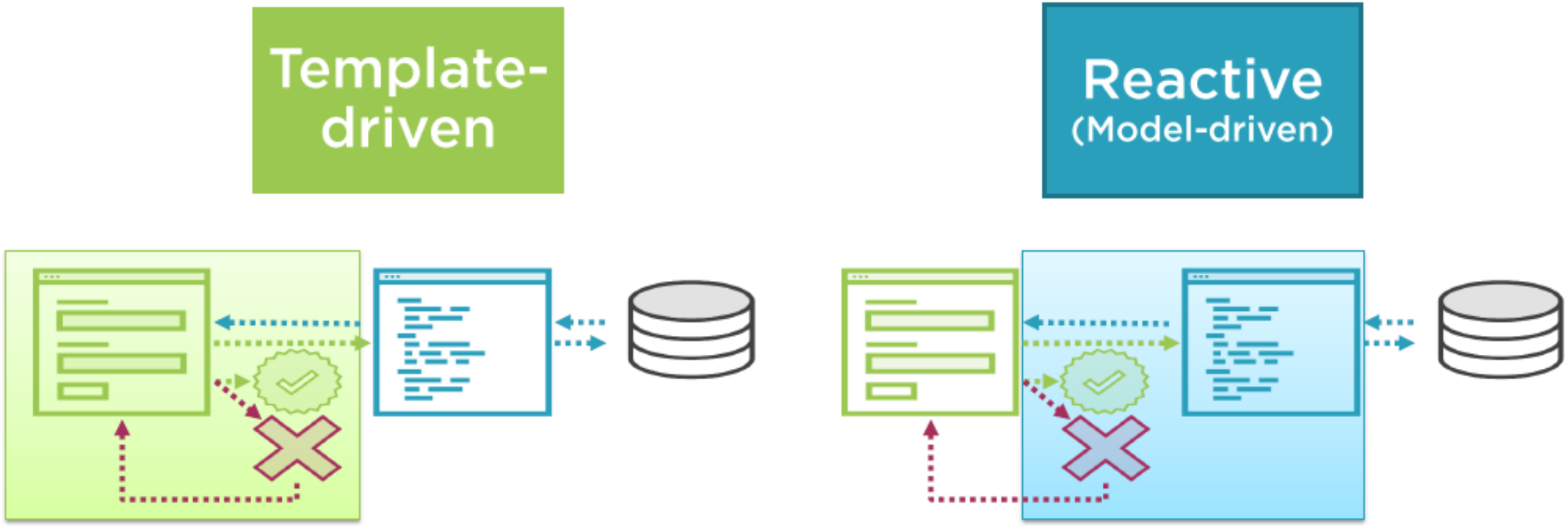
```
export class UserGuard implements CanActivate, CanActivateChild {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
    return this.authService.isAuthenticated()
      .then(
        (authenticated: boolean) => {
          if (authenticated) {
            return true;
          } else {
            this.router.navigate(['/login']);
          }
        }
      );
  }

  canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {
    return this.canActivate(childRoute, state);
  }

}
```

Using Guards : CanActivate

```
const routes: Routes = [
 {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'home', component: HomeComponent},
  {path: 'product', canActivate: [UserGuard], component: ProductComponent},
  {path: 'login', component: LoginComponent},
  {path: 'userHome', canActivate: [UserGuard], component: UserHomeComponent, children: [
    {path: 'register', component: RegisterComponent},
    {path: 'userList', component: AllUserComponent},
    {path: 'profile/:userName', component: UserProfileComponent}
  ]},
  {path: '**', redirectTo: 'home', pathMatch: 'full'}
];
```

```
 const routes: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
   {path: 'home', component: HomeComponent},
   {path: 'product', canActivate: [UserGuard], component: ProductComponent},
   {path: 'login', component: LoginComponent},
   {path: 'userHome', /* canActivate: [UserGuard],*/ canActivateChild: [UserGuard], component: UserHomeComponent, children: [
     {path: 'register', component: RegisterComponent},
     {path: 'userList', component: AllUserComponent},
     {path: 'profile/:userName', component: UserProfileComponent}
   ]},
   {path: '**', redirectTo: 'home', pathMatch: 'full'}
 ];
```

Forms



Template driven vs Reactive or Model driven

Template-driven

Easy to use

Similar to Angular 1

Two-way data binding ->
Minimal component code

Automatically tracks form and
input element state

Reactive

More flexible ->
more complex scenarios

Immutable data model

Easier to perform an action
on a value change

Reactive transformations ->
DebounceTime or DistinctUntilChanged

Easily add input elements dynamically

Easier unit testing

Creating Reactive form – Import ReactiveFormsModule

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


 registrationForm: FormGroup;

 ngOnInit() {
   this.pageTitle = this.activatedRoute.snapshot.data['pageTitle'];
   this.registrationForm = new FormGroup({
     'firstName': new FormControl(null),
     'lastName': new FormControl(null),
     'userName': new FormControl(null),
     'email': new FormControl(null)
   });
 }

 onSubmit() {
   console.log(this.registrationForm);
 }
```

```
<form [formGroup]="registrationForm" class="needs-validation" (ngSubmit)="onSubmit()">
     <input name="firstName" formControlName="firstName" type="text" id="firstName">
     <input name="lirstName" formControlName="lastName" type="text" id="lirstName">
     <input name="userName" type="text" formControlName="userName" id="userName">
     <input name="email" formControlName="email" type="email"  id="email">
     <button class="btn btn-primary btn-lg btn-block" type="submit">Continue to Register</button>
   </form>
```