Angular: Getting Started -by Deborah Kurata

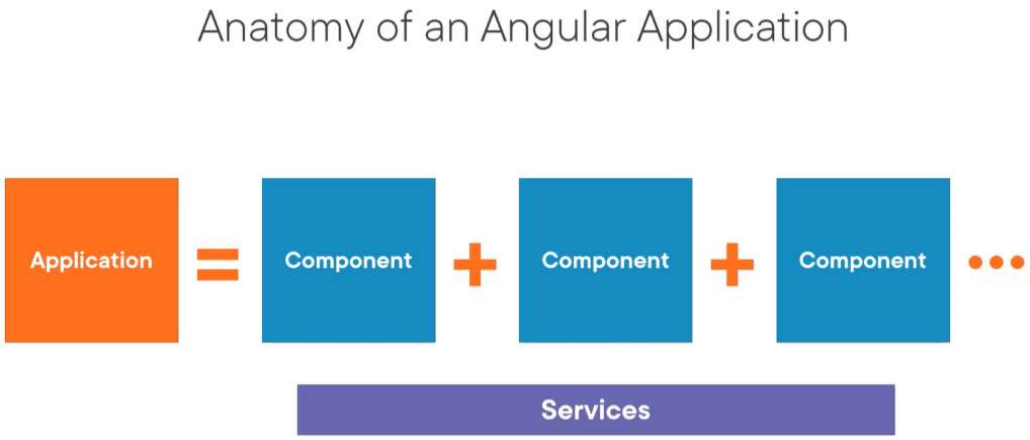
05 September 2021 21:29

Chapter 1 to 3:

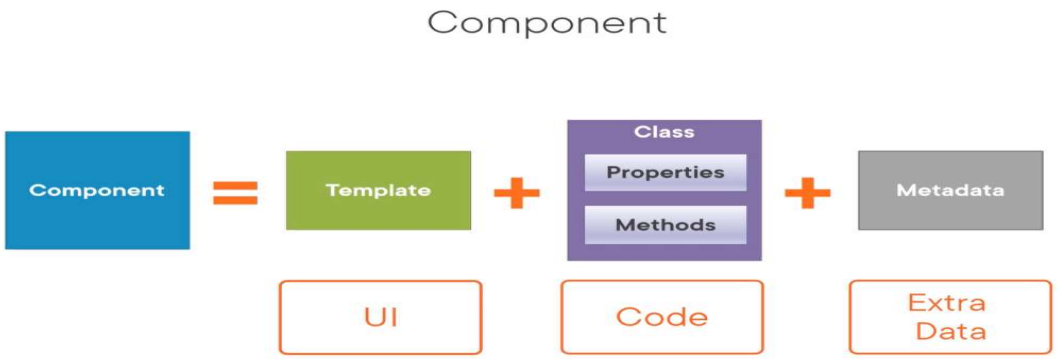
Why Angular?

1. Expressive HTML.
2. Powerful Data Binding
3. Modular By Design
4. Built-in Back-End Integration

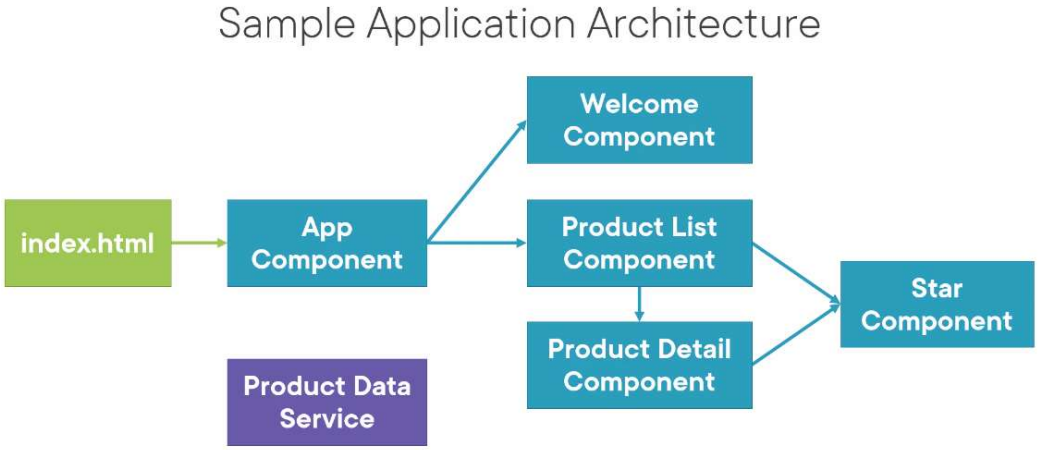
Angular Anatomy?



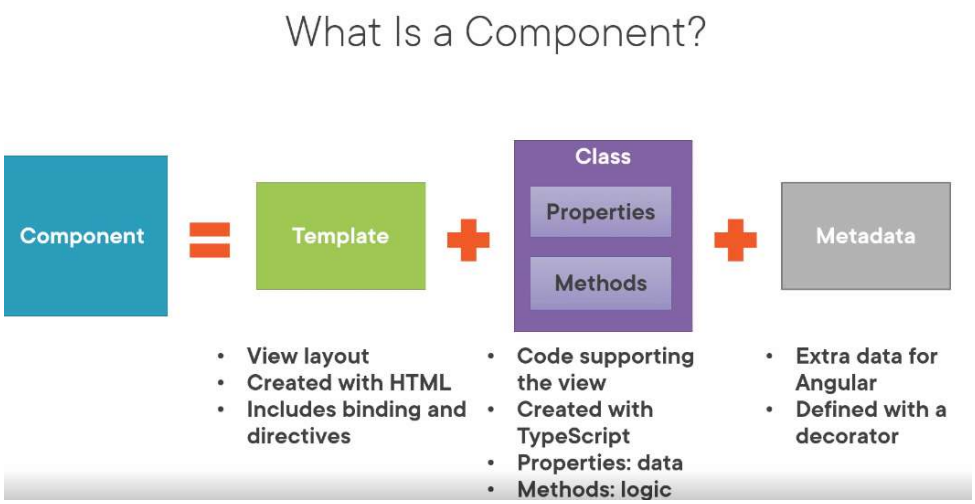
Application = Component/s + Services



Component = Template (UI) + Class {properties/methods} (Code) + MetaData (Extra Data)

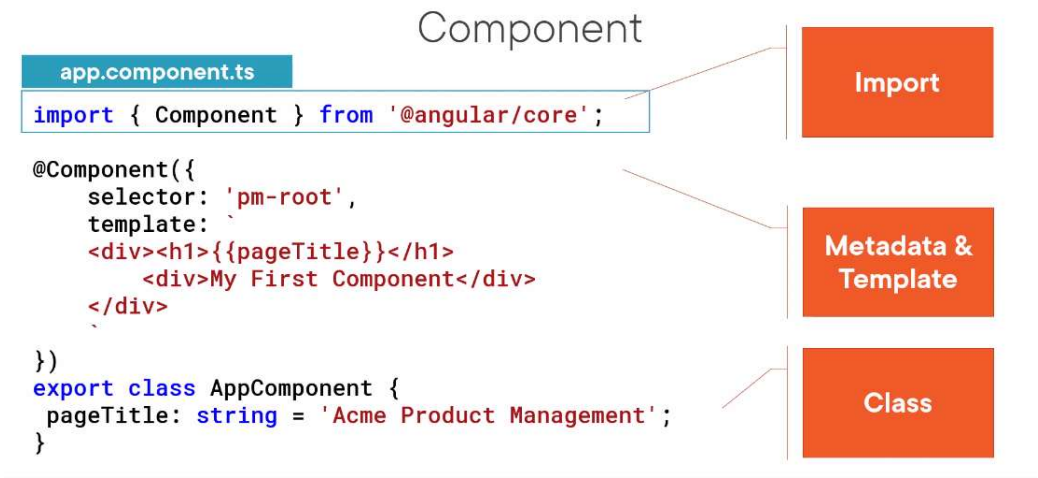


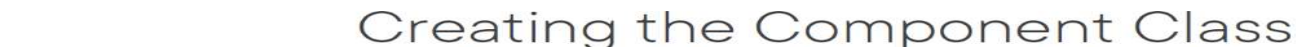
What is Component?



- Npm => Is a open source repository for Angular, TypeScript, Bootstrap etc. AND also a command line utility for interacting with the repository.
- Npm => Package manager for Javascript applications.

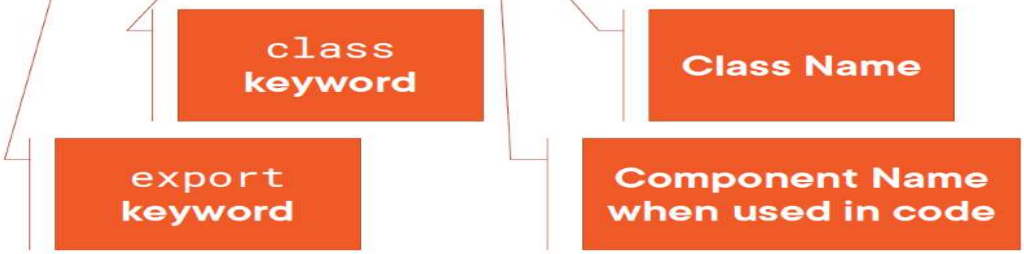
Chapter 4:





app.component.ts

```
export class AppComponent {
    pageTitle: string = 'Acme Product Management';
}
```



Creating the Component Class

app.component.ts

```
export class AppComponent {
    pageTitle: string = 'Acme Product Management';
}
```

Defining the Metadata

app.component.ts

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

```
  }
```

Decorator

A function that adds metadata to a class, its members, or its method arguments.

Prefixed with an @.

Angular provides built-in decorators.

@Component()

Defining the Metadata

app.component.ts

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

- Component decorator
- Directive Name used in HTML
- View Layout
- Binding

Importing What We Need

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 = 'Acme Product Management';
}
```

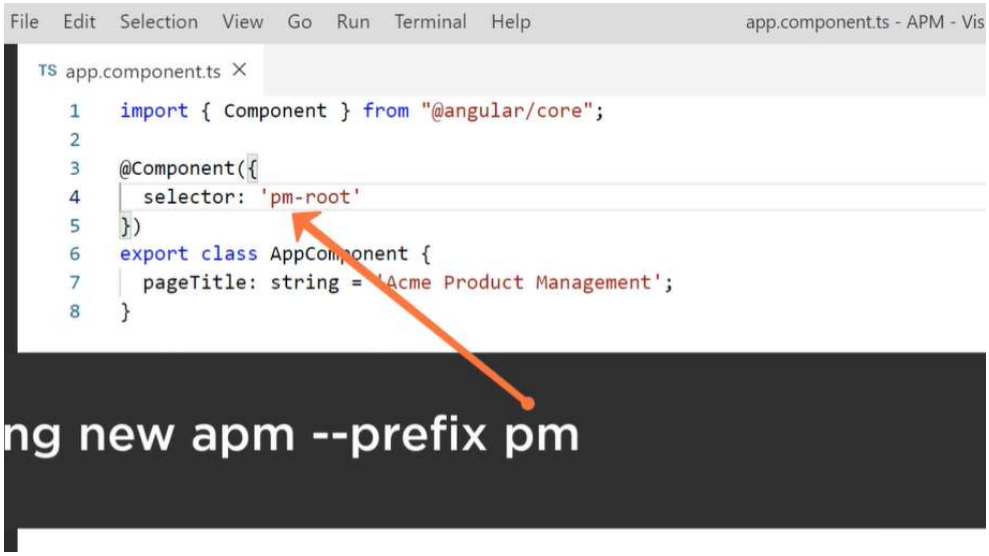
- import keyword
- Angular library module name
- Member name

Completed Component

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 = 'Acme Product Management';
}
```



Single Page Application (SPA)

index.html contains the main page for the application

This is often the only Web page of the application

Hence an Angular application is often called a Single Page Application (SPA)

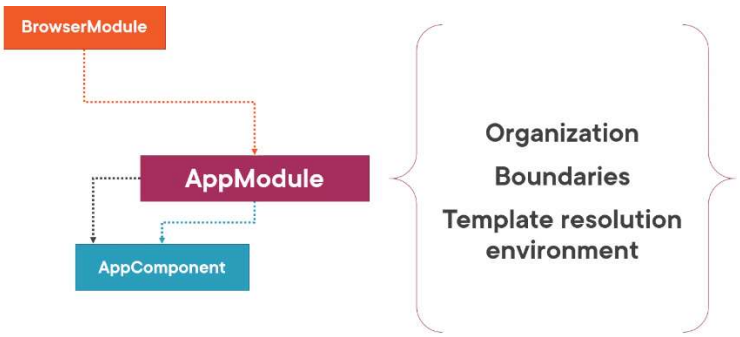
Hosting the Application

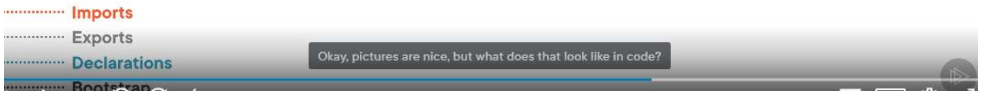
index.html

```
<body>
 <pm-root></pm-root>
</body>
```

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 = 'Acme Product Management';
}
```





Defining the Angular Module

app.module.ts

```
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 { }
```

Something's Wrong!

Compiler displays syntax errors

```
export class AppComponent {
  pageTitle = 'Acme PM';
  PageTitle = 'Something else';
}
```

Casing matters

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

Components must be declared in an Angular module

Start with Your Code Editor

Check for squiggly lines

Open the terminal

https://angular.io/errors

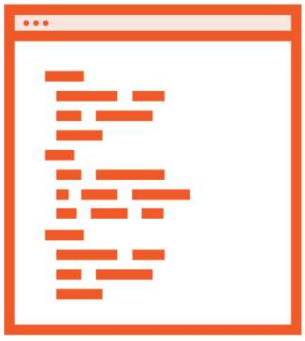
Start with Your Code Editor

Check for squiggly lines

Open the terminal

Stop (Ctrl+C) and Restart

Recheck Your Code



- HTML
  - Close tags
  - Angular directives are case sensitive
- TypeScript
  - Close braces
  - TypeScript is case sensitive

Component Checklist

Class -> Code

Decorator -> Template and Metadata

Import what we need

Component Checklist:

Class

Clear name
- Use PascalCasing
- Append "Component" to the name

`export` keyword

```
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

Component Checklist: Members

Data in properties
- Appropriate data type
- Appropriate default value
- camelCase with first letter lowercase

Logic in methods
- camelCase with first letter lowercase

```
export class AppComponent {
  showImage: boolean =  false;    // property

  toggleImage(): void {           // method
    this.showImage = true;
  }
}
```

Use camelCase for method names with the first letter lowercase.

Component Checklist: Metadata

Component decorator
- Prefix with @; Suffix with ()

`selector`: Component name in HTML
- Prefix for clarity

`template`: View's HTML
- Correct HTML syntax

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

Component Checklist: Import Statement

Defines where to find the members that this component needs

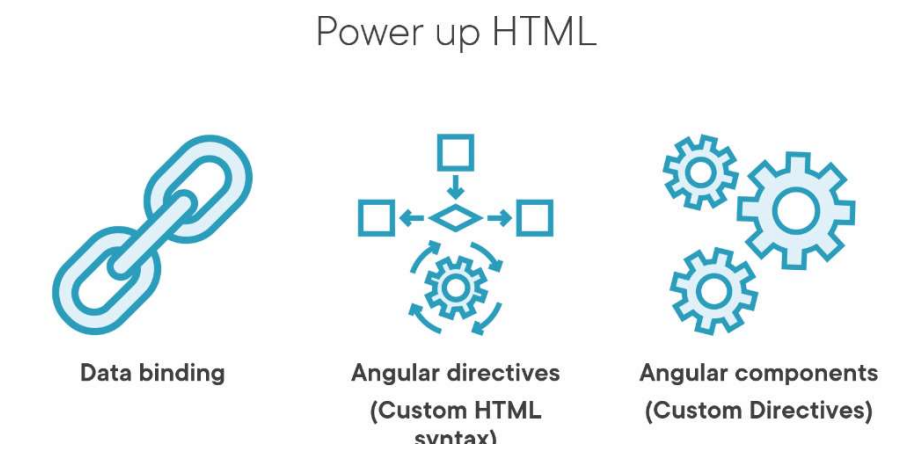
`import` keyword

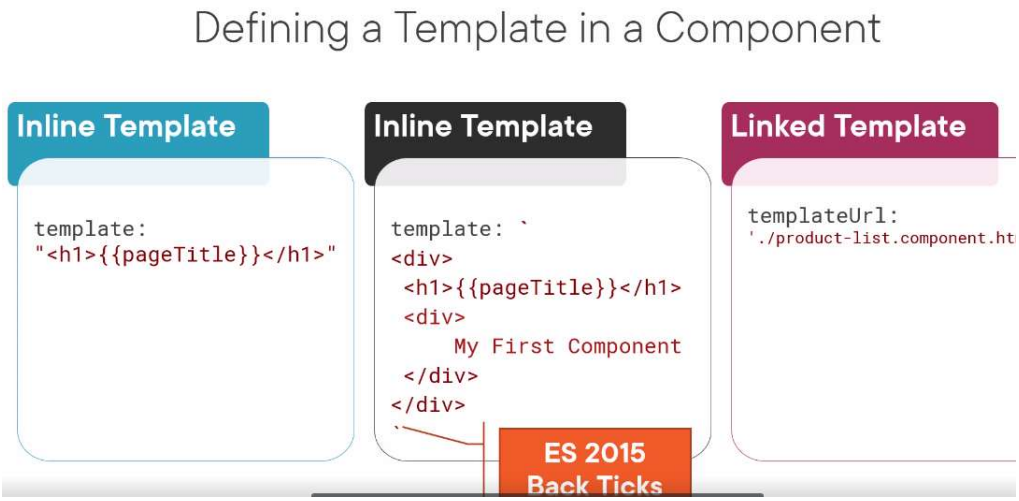
Member name
- Correct spelling/casing

Path
- Enclose in quotes
- Correct spelling/casing

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

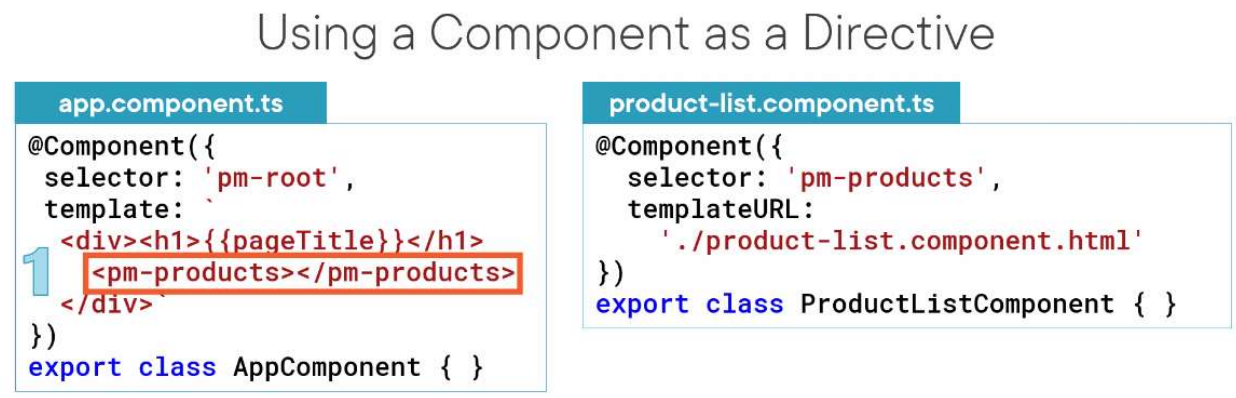
Chapter 5:

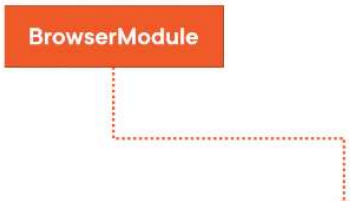


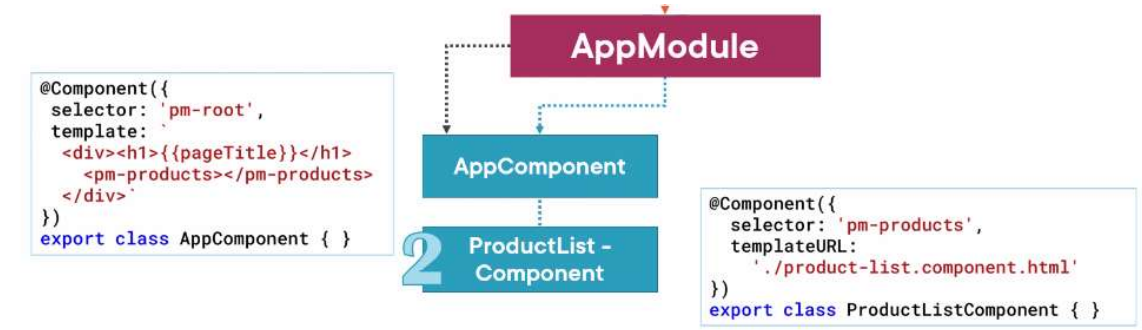


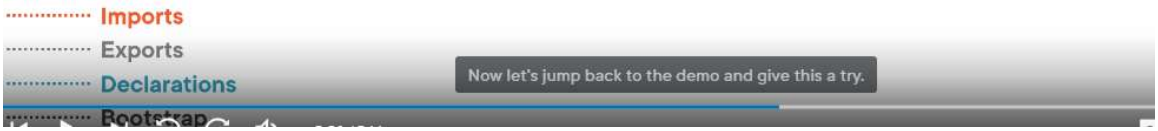
Install command for bootstrap and font-awesome:

npm install bootstrap font-awesome



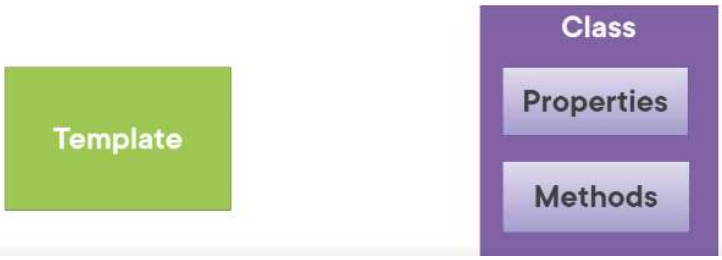






Binding

Coordinates communication between the component's class and its template and often involves passing data.



Interpolation

Template

```
<h1>{{pageTitle}}</h1>

{{'Title: ' + pageTitle}}

{{2*20+1}}

<h1 innerText={{pageTitle}}></h1>
```

Class

```
export class AppComponent {
 pageTitle: string =
     'Acme Product Management';
}
```

Interpolation

Template

```
<h1>{{pageTitle}}</h1>
```

Class

```
export class AppComponent {
```

```
pageTitle: string =
    'Acme Product Management';
}
```

{{' tle}}

{{2

<h1 Title}}></h1>

Template Expression

Directive

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

- **Custom**
- **Built-In**

Angular Built-in Directives

Structural Directives

- ***ngIf: If logic**
- ***ngFor: For loops**

*ngFor Built-In Directive

```
<tr *ngFor='let product of products'>
    <td></td>
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
    <td>{{ product.releaseDate }}</td>
    <td>{{ product.price }}</td>
    <td>{{ product.starRating }}</td>
</tr>
```

for...of vs for...in

for...of

- Iterates over iterable objects, such as an array.
- Result: di, boo, punkeye

```
let nicknames= ['di', 'boo', 'punkeye'];

for (let nickname of nicknames) {
  console.log(nickname);
}
```

for...in

- Iterates over the properties of an object.
- Result: 0, 1, 2

```
let nicknames= ['di', 'boo', 'punkeye'];

for (let nickname in nicknames) {
  console.log(nickname);
}
```

So we see each array index logged to the console. To help remember

Template Checklist: Inline Template

For short templates

Specify the `template` property

Use the ES 2015 back ticks for multiple lines

Watch syntax

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

Template Checklist: Linked Template

For longer templates

Specify the `templateUrl` property

Define the path to the HTML file

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

Checklist: Component as a Directive

product-list.component.ts

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

app.component.ts

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

1

app.module.ts

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

2

NgModule decorator of the Angular module.

Template Checklist: Interpolation

One way binding
- From component class property to an element property

Defined with double curly braces
- Contains a template expression
- No quote needed

```
<td>{{ product.productName }}</td>
```

Template Checklist: Structural Directives

*ngIf and *ngFor
- Prefix with an asterisk
- Assign to a quoted string expression

Template Checklist:

*ngIf

Add or remove an element and its children from the DOM

Expression is evaluated:
- true: elements added to the DOM
- false: elements removed from the DOM

```
<table *ngIf='products'>
```

Template Checklist: *ngFor

Repeat an element and its children in the DOM
- For each object in an iterable list

Define the local variable with `let`

Specify `'of'`

```
<tr *ngFor='let product of products'>
  ...
</tr>
```

Chapter 6:

Module Overview

Property Binding

Handling Events with Event Binding

Handling Input with Two-way Binding

Transforming Data with Pipes

Property Binding

Element Property

Template Expression

```
<img [src]='product.imageUrl'>

<img src={{product.imageUrl}}>

<input type='text' [disabled]='isDisabled'/>
```

```
<img src='http://myImages.org/{{product.imageUrl}}'>
```

Event Binding

Template

```
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
<button (click)='toggleImage()'>
```

Event

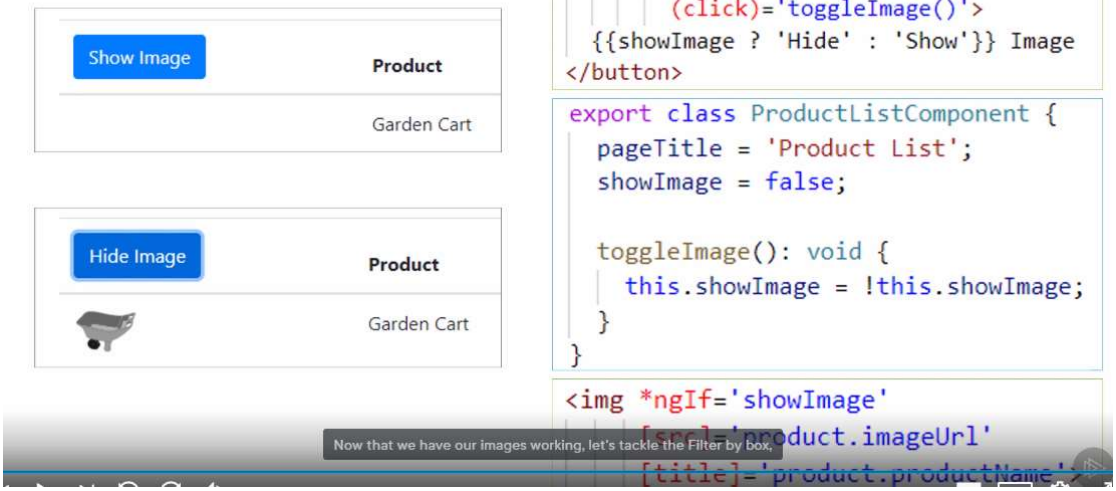
Method

Component

```
export class ListComponent {
 pageTitle: string = 'Product List';
 products: any[] = […];
 toggleImage(): void {…}
}
```

Web Events reference:
https://developer.mozilla.org/en-US/docs/Web/Events

Change Detection




```
<button class='btn btn-primary'
        (click)='toggleImage()'>
  {{showImage ? 'Hide' : 'Show'}} Image
</button>
```

```
export class ProductListComponent {
  pageTitle = 'Product List';
  showImage = false;

  toggleImage(): void {
    this.showImage = !this.showImage;
  }
}
```

```
<img *ngIf='showImage'
     [src]='product.imageUrl'
     [title]='product.productName'>
```

Two-way Binding

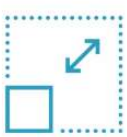
Template

```
<div class='col-md-2'>Filter by:</div>
<div class='col-md-4'>
   <input type='text'
          [(ngModel)]='listFilter'/>
</div>
```

Component

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

Transforming Data with Pipes



Transform bound properties before display

Built-in pipes: date, number, decimal, percent, currency, json, etc.

Custom pipes

Pipe Examples

```
{{ product.productCode | lowercase }}

<img [src]='product.imageUrl'
     [title]='product.productName | uppercase'>

{{ product.price | currency | lowercase }}

{{ product.price | currency:'USD':'symbol':'1.2-2' }}
```

Data Binding

Template

```
<div class='card-header'>
  {{pageTitle}}
</div>
```

```
<img *ngIf='showImage'
     src={{product.imageUrl}}
     [title]='product.productName'
     [style.width.px]='imageWidth'
     [style.margin.px]='imageMargin'>
```

```
<button class='btn btn-primary'
        (click)='toggleImage()'>
  {{showImage ? 'Hide' : 'Show'}} Image
</button>
```

```
<div class='col-md-2'>Filter by:</div>
```

Component

```
@Component({
  selector: 'pm-products',
  templateURL: './product-list.component.html'
})
export class ProductListComponent {
  pageTitle: string = 'Product List';
  imageWidth = 50;
  imageMargin = 2;
  showImage = false;

  listFilter: string = 'cart';
  products: any[] = […];

  toggleImage(): void {…}
}
```

```
<div class='col-md-4'>
  <input type='text'
         [(ngModel)]='listFilte
</div>
```

we can process user entry for an interactive experience.

Data Binding

Data Binding Checklist: ngModel

product-list.component.html

```
<div class='col-md-4'>
  <input type='text'
     [(ngModel)]='listFilter' />
</div>
```

app.module.ts

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

We'll talk more about Angular modules later in this course.

Data Binding Checklist: Pipes

Pipe character |

Pipe name

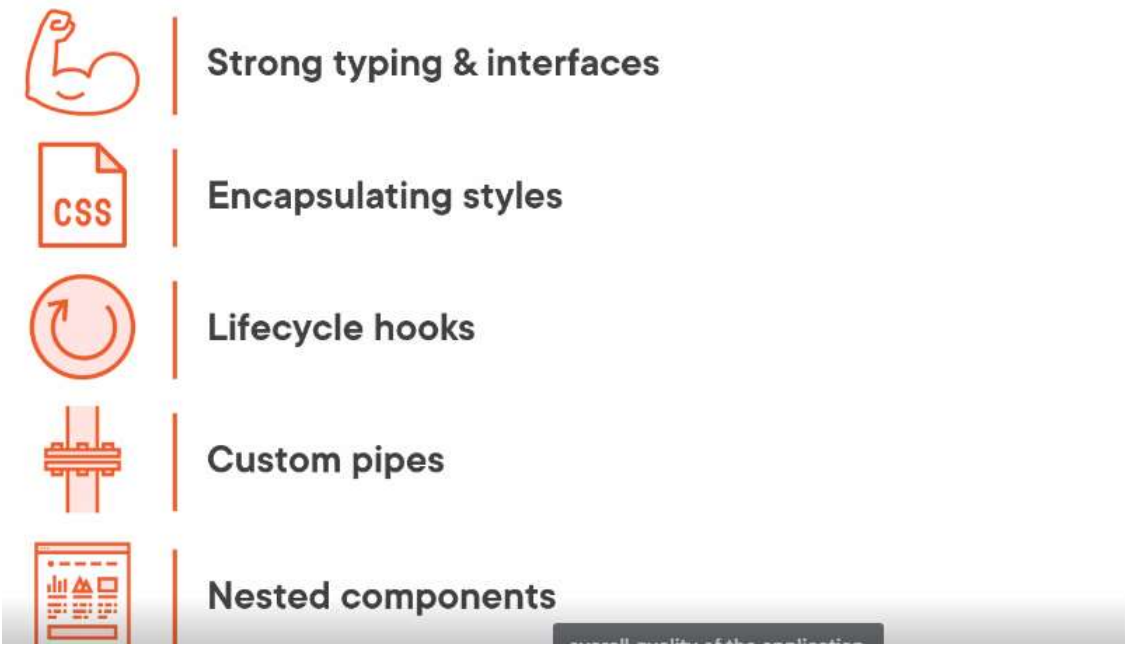
Pipe parameters
- Separated with colons

Example

```
{{ product.price | currency:'USD':'symbol':'1.2-2' }}
```

Chapter 7:

Improving Our Components



Strong Typing

```
export class ProductListComponent {
    pageTitle: string = 'Product List';
    showImage: boolean = false;
    listFilter: string = 'cart';
    message: string;

    products: any[] = […];

    toggleImage(): void {
        this.showImage = !this.showImage;
    }

    onRatingClicked(message: string): void {
        this.message = message;
    }
}
```

which negates the benefits of strong typing.

An interface is a specification identifying a related set of properties and methods.

Two Ways to Use an Interface

```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  releaseDate: string;
  price: number;
  description: string;
  starRating: number;
  imageUrl: string;
}
```

As a type

```
products: IProduct[] = [];
```

```
export interface DoTiming {
   count: number;
   start(index: number): void;
   stop(): void;
}
```

As a feature set

```
export class myComponent
          implements DoTiming {
  count: number = 0;
  start(index: number): void {
    ...
  }
  stop(): void {
    ...
  }
}
```

For now, let's focus on using an interface as a data type.

Two Ways to Use an Interface


```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  releaseDate: string;
  price: number;
  description: string;
  starRating: number;
  imageUrl: string;
}
```

As a type

```
products: IProduct[] = [];
```

```
export interface DoTiming {
   count: number;
   start(index: number): void;
   stop(): void;
}
```

As a feature set

```
export class myComponent
          implements DoTiming {
  count: number = 0;
  start(index: number): void {
    ...
  }
  stop(): void {
    ...
  }
}
```

For now, let's focus on using an interface as a data type.

Declaring an Interface as a Data Type

```
export interface IProduct {
    productId: number;
    productName: string;
    productCode: string;
    releaseDate: Date;
    price: number;
    description: string;
    starRating: number;
    imageUrl: string;
}
```

- export keyword
- Interface name
- interface keyword

Using an Interface as a Data Type

```
import { IProduct } from './product';

export class ProductListComponent {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';

  products: IProduct[] = […];

  toggleImage(): void {
      this.showImage = !this.showImage;
   }
}
```

Handling Unique Component Styles

Templates sometimes require unique styles

We can inline the styles directly into the HTML

We can build an external stylesheet and link it in index.html

There is a better way!

Encapsulating Component Styles

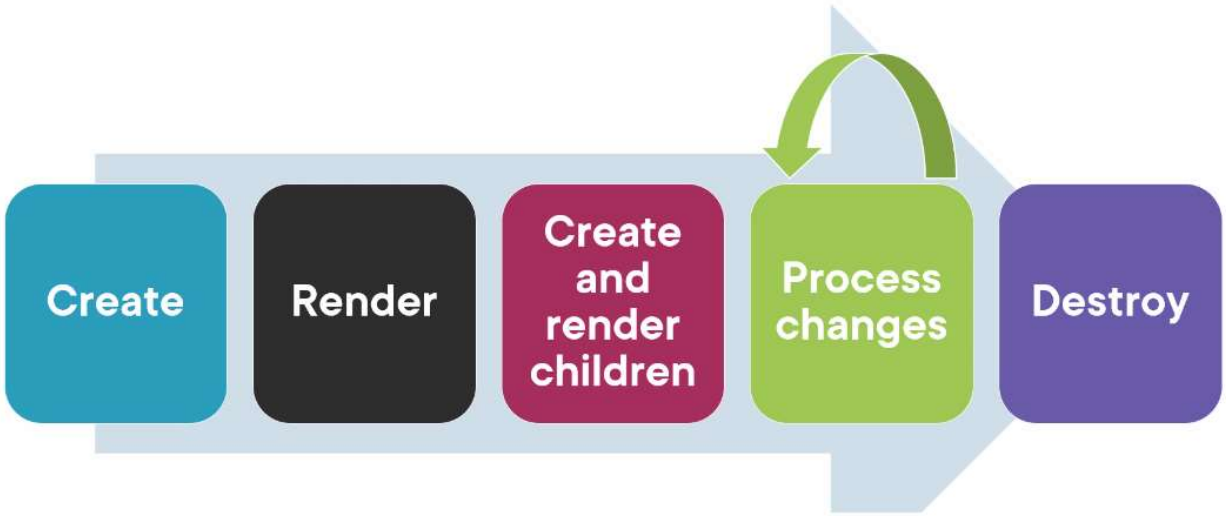
styles

```
@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



A lifecycle hook is an interface we implement to write code when a component lifecycle event occurs.

Component Lifecycle Hooks

`OnInit`: **Perform component initialization, retrieve data**

`OnChanges`: **Perform action after change to input properties**

`OnDestroy`: **Perform cleanup**

Using a Lifecycle Hook

2
```
import { Component, OnInit } from '@angular/core';
```

1
```
export class ProductListComponent implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
```

```
    products: IProduct[] = […];

3   ngOnInit(): void {
        console.log('In OnInit');
    }
}
```

Building a Custom Pipe

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

@Pipe({
    name: 'convertToSpaces'
})
export class ConvertToSpacesPipe implements PipeTransform {

  transform(value: string,
            character: string): string {
  }
}
```

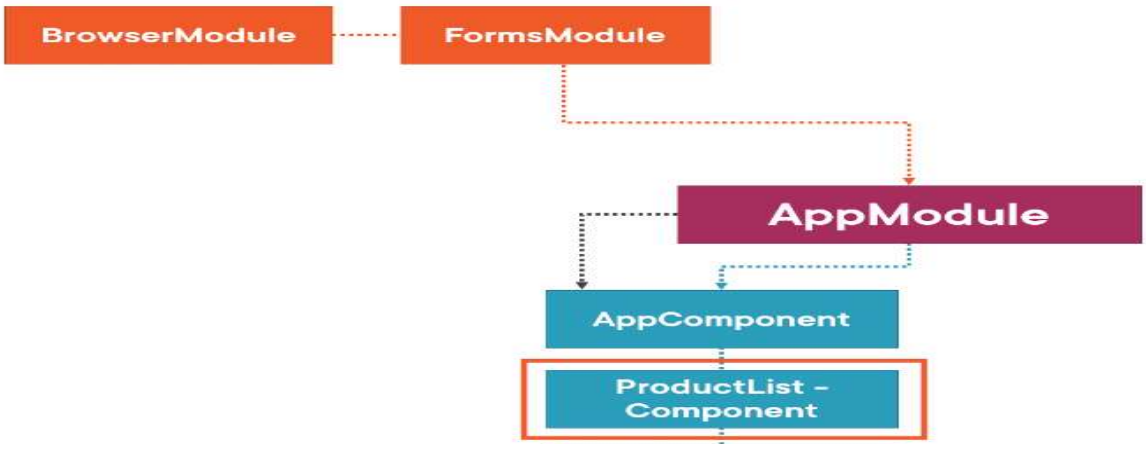
Using a Custom Pipe

Template

```
<td>{{ product.productCode | convertToSpaces:'-'}}</td>
```

Pipe

```
transform(value: string, character: string): string {

}
```



ConvertToSpaces-Pipe

Imports
Exports
Declarations

that declares the ProductListComponent.

Using a Custom Pipe

Template

```
<td>{{ product.productCode | convertToSpaces:'-'}}</td>
```

Module

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

Now let's build our custom pipe.

Getters and Setters

```
amount: number = 0;
```

```
private _amount: number = 0;
```

```
get amount(): number {
    // process the amount
    // return amount from private storage
    return this._amount;
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
    this._amount = value;
}
```

Getters and Setters

```
private _amount: number = 0;
```

```
get amount(): number {
```

```
    // process the amount
    // return amount from private storage
    return this._amount;
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
    this._amount = value;
}
```

```
this.amount = 200;
console.log(this.amount);
```

Filtering a List

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {
    return this.products.filter();
}
```

An arrow function is compact syntax for defining a function.

Arrow Functions

Classic named function (method)

```
capitalizeName(product: IProduct): string {
    return product.productName.toUpperCase();
}
```

Arrow function

```
(product: IProduct) => product.productName.toUpperCase();
```

Multi-statement arrow function

```
(product: IProduct) => {
    console.log(product.productName);
    return product.productName.toUpperCase();
}
```

Filtering a List

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {
    return this.products.filter((product: IProduct) =>
          product.productName.includes(this.listFilter));
}
```

Interface Checklist: Interface as a Type

interface keyword

Properties and their types

Export it

```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  ...
}
```

Use the interface as a data type

```
products: IProduct[] = [...];
```

Interface Checklist: Interface as a Feature Set

Implementing interfaces:
- implements keyword & interface name
- Write code for each property & method

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

export class ProductComponent implements OnInit {
  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Then be sure to write code for every property and method declared

Styles Checklist: Encapsulating Styles

styles property
- Specify an array of style strings

styleUrls property
- Specify an array of stylesheet paths

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

Lifecycle Hook Checklist: Using Lifecycle Hooks

Import the lifecycle hook interface

Implement the lifecycle hook interface

Write code for the hook method

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

export class ProductComponent implements OnInit {
  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Pipe Checklist: Building a Custom Pipe

Create a class that implements PipeTransform

Write code for the Transform method

Decorate the class with the Pipe decorator

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

@Pipe({
    name: 'spacePipe'
})
export class SpacePipe implements PipeTransform {
  transform(value: string,
            character: string): string { ... }
}
```

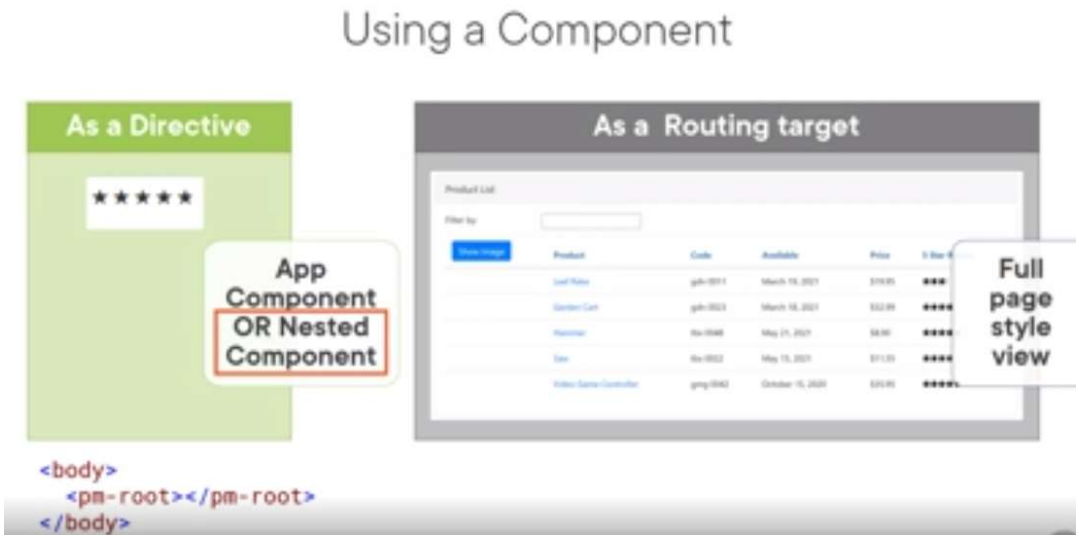
and decorate the class with the @Pipe decorator.

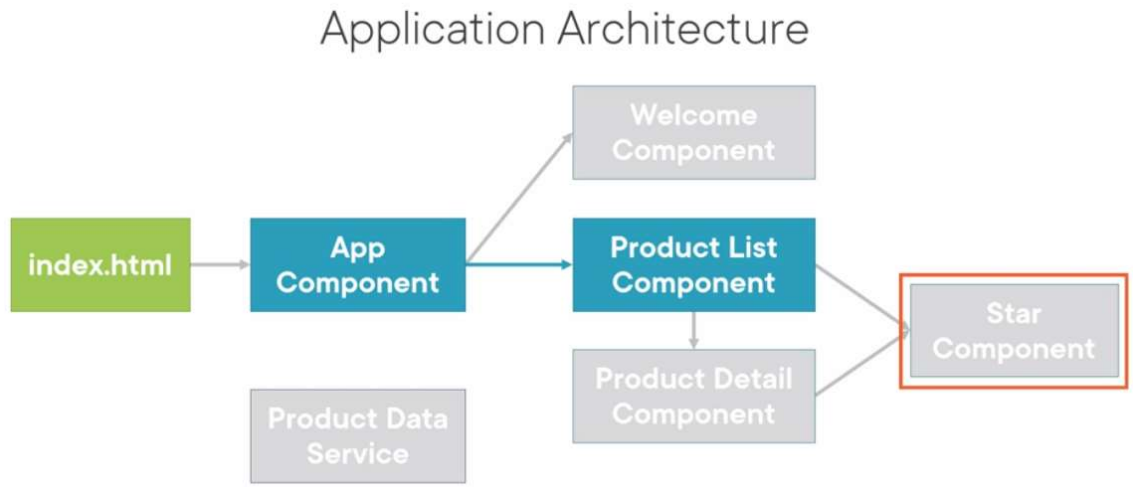
Pipe Checklist: Using a Custom Pipe

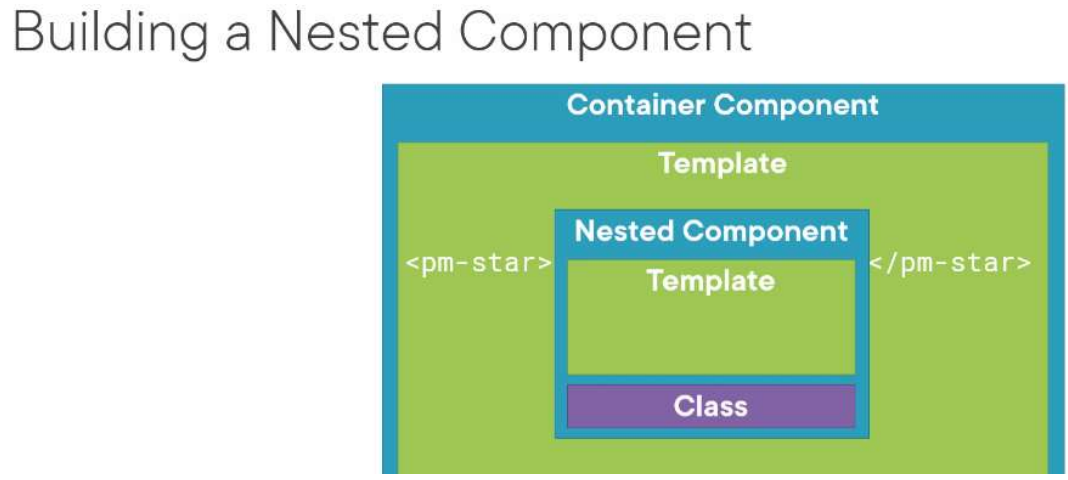
Add the pipe to the declarations array of an Angular module

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

Chapter 8:
Building Nested Components:







Using a Nested Component as a Directive

product-list.component.ts

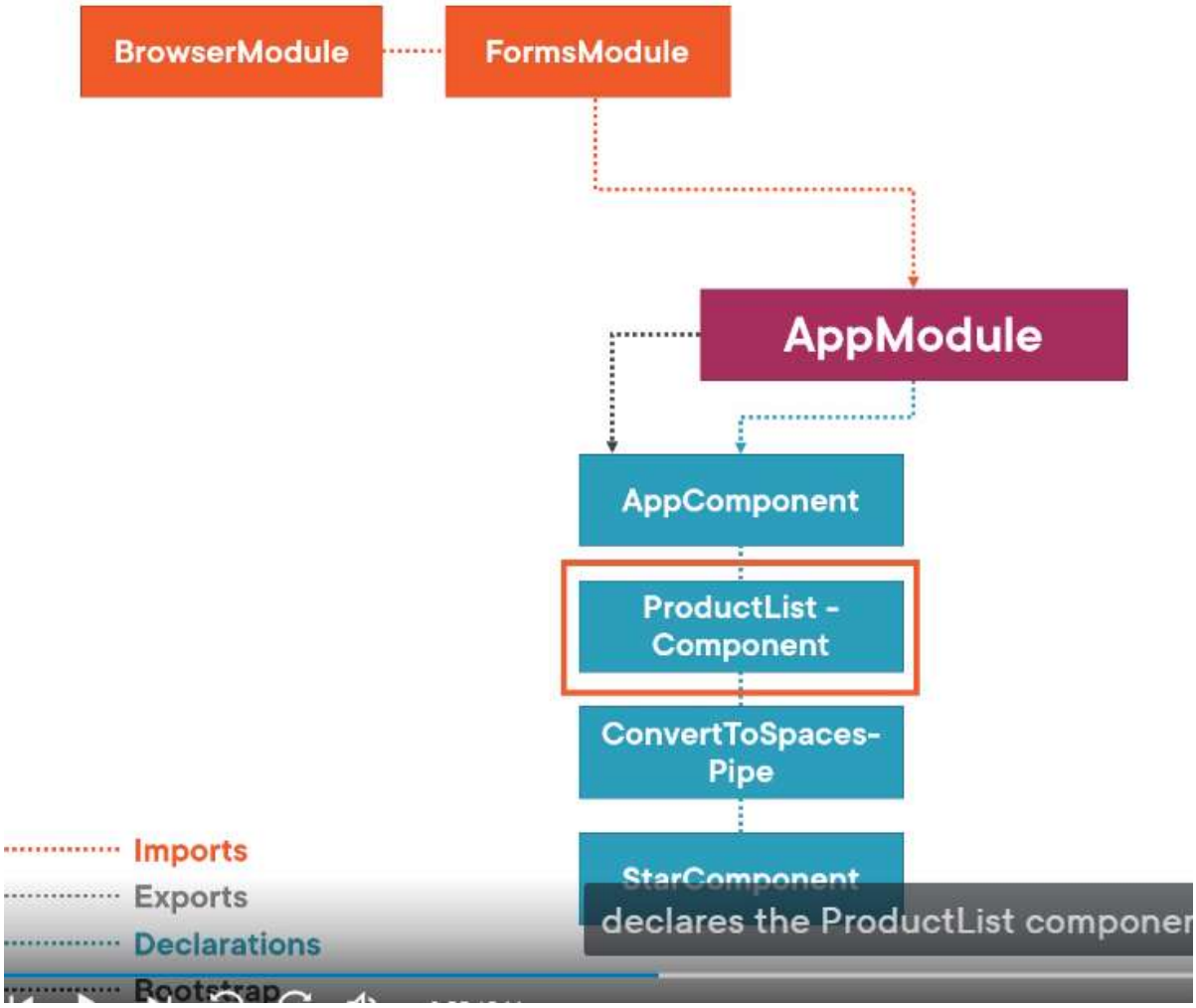
```
@Component({
  selector: 'pm-products',
  templateURL: './product-list.component.html'
})
export class ProductListComponent { }
```

star.component.ts

```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
  rating: number;
  cropWidth: number;
}
```

product-list.component.html

```
<td>
    <pm-star></pm-star>
</td>
```



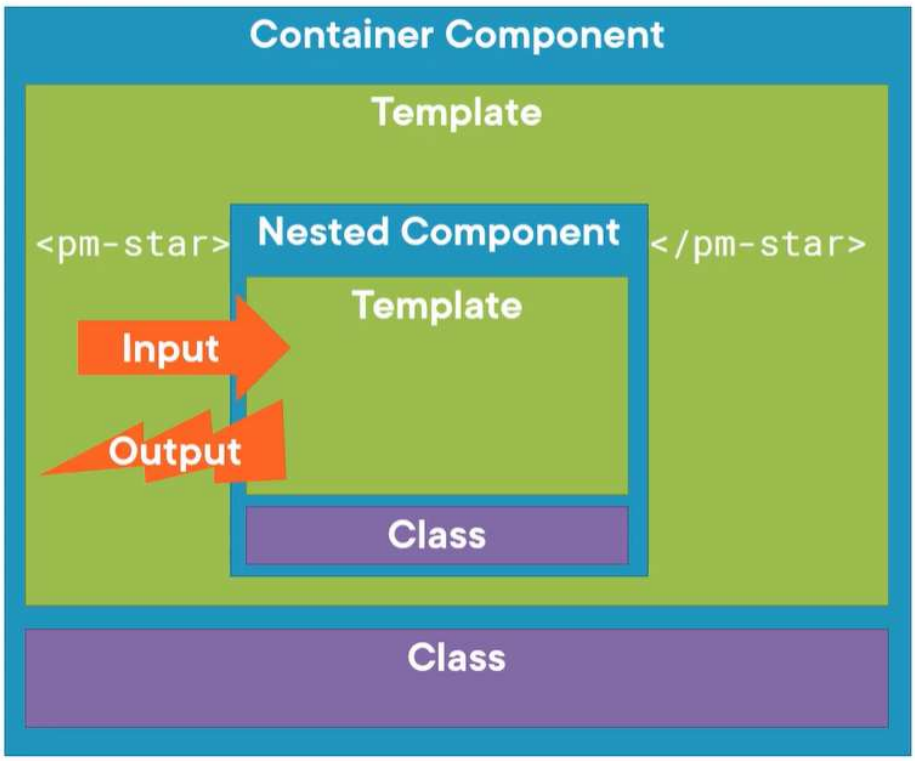
Telling Angular About Our Component

app.module.ts

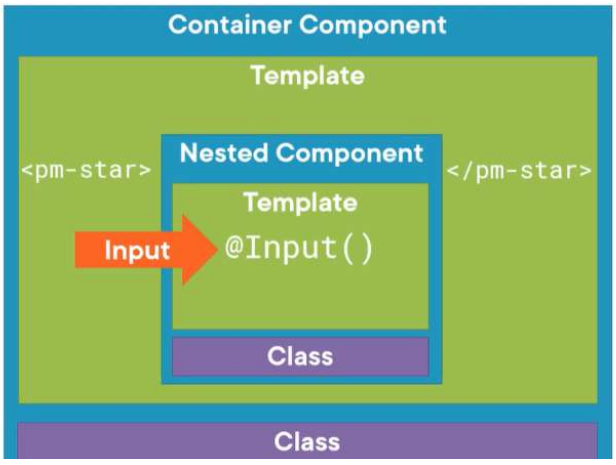
```
...
```

```
import { StarComponent } from './shared/star.component';

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

Building a Nested Component



Passing Data to a Nested Component (@Input)



Passing Data to a Nested Component (@Input)

product-list.component.ts

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

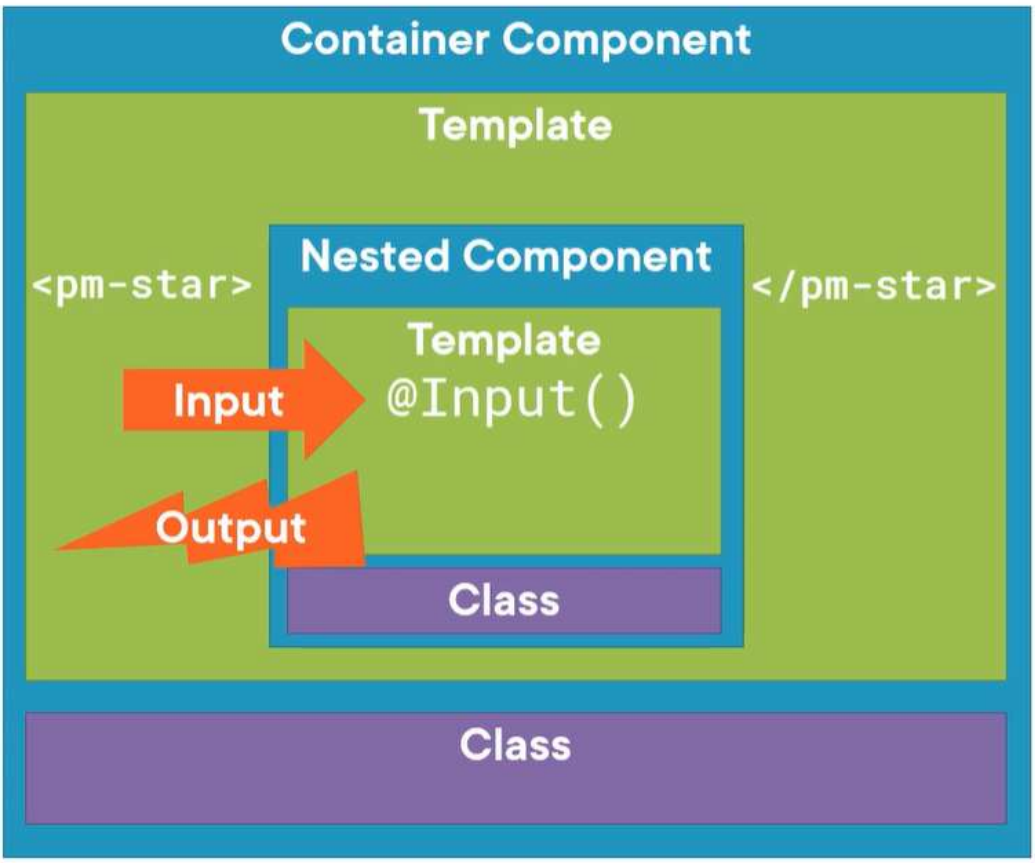
product-list.component.html

```
<td>
 <pm-star [rating]='product.starRating'>
 </pm-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  cropWidth: number;
}
```

Emitting an Event (@Output)



Emitting an Event (@Output)

product-list.component.ts

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

star.component.ts

```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
```

```
  onNotify(message: string): void { }
}
```

```
@Input() rating: number;
cropWidth: number;
@Output() notify: EventEmitter<string> =
                  new EventEmitter<string>();
onClick() {
  this.notify.emit('clicked!');
}
}
```

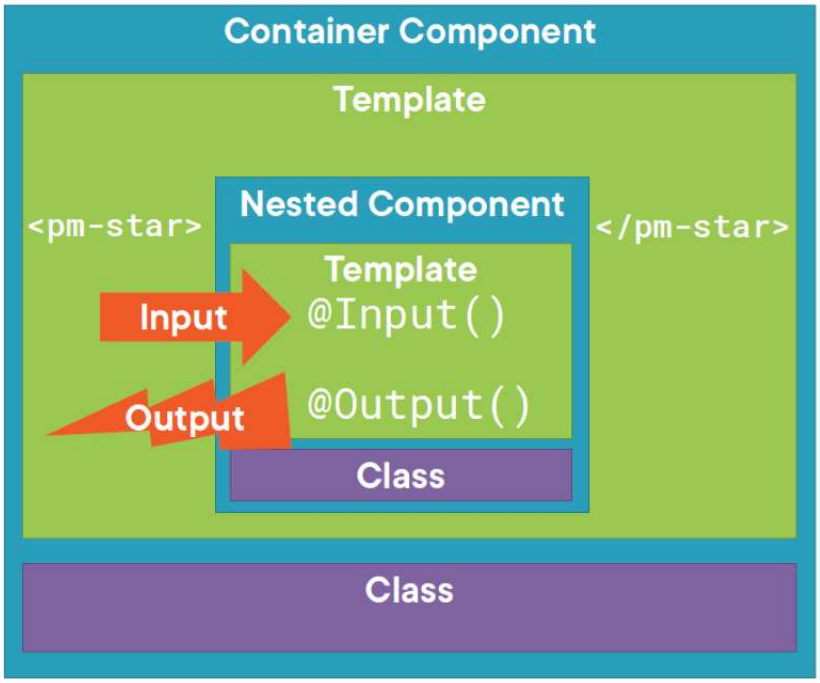
product-list.component.html

```
<td>
 <pm-star [rating]='product.starRating'
          (notify)='onNotify($event)'>
 </pm-star>
</td>
```

star.component.html

```
<div (click)='onClick()'>
  ... stars ...
</div>
```

Nest-able Component's Public API



Nesting Checklist: Input Properties

Input decorator

Attach to a property of any type

Prefix with @; Suffix with ()

```
export class StarComponent {
 @Input() rating: number;
}
```

Nesting Checklist: Output Properties

`Output` decorator

Attached to a property declared as an EventEmitter

Use the generic argument to define the event data type

Use the new keyword to create an instance of the EventEmitter

Prefix with @; Suffix with ()

```
export class StarComponent {
 @Output() notify: EventEmitter<string> =
                          new EventEmitter<string>();
}
```

Nesting Checklist: Container Component

Use the directive
- Directive name -> nested component's selector

Use property binding to pass data to the nested component

Use event binding to respond to events from the nested component
- Use $event to access the event data passed from the nested component

```
<pm-star [rating]='product.starRating'
         (notify)='onNotify($event)'>
</pm-star>
```

Chapter 9

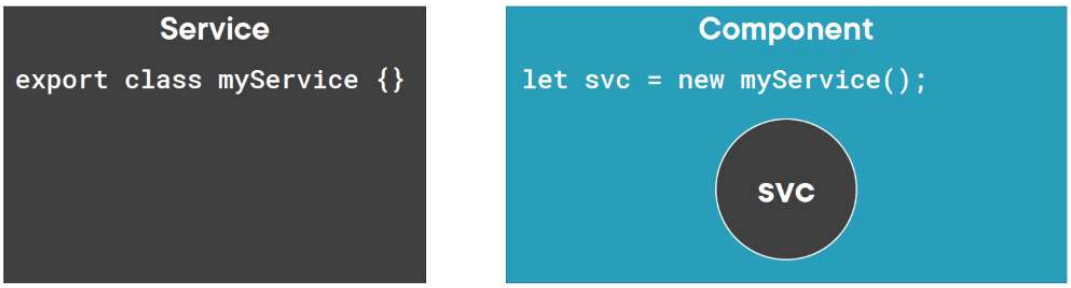
Service

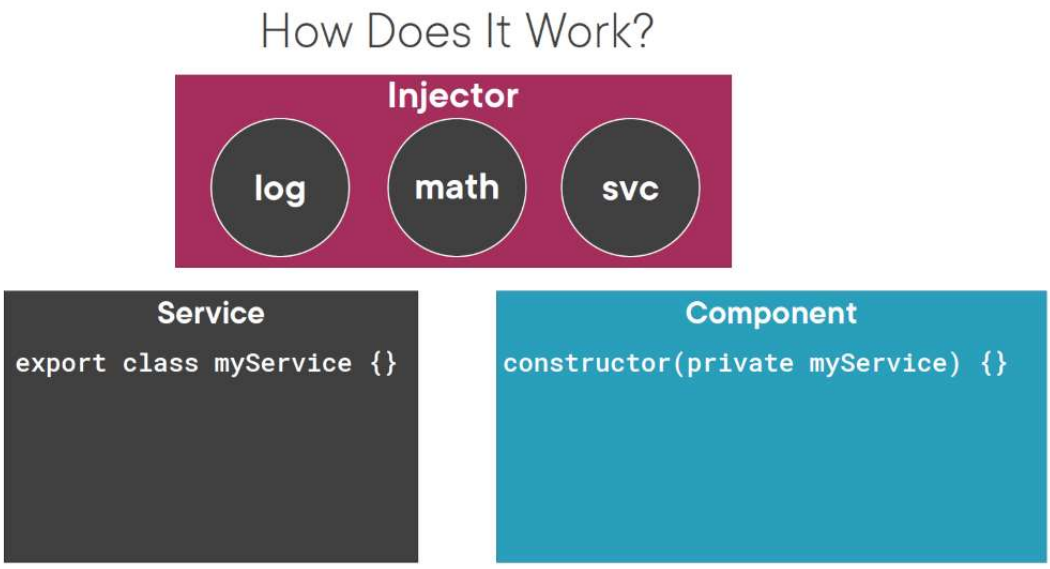
A class with a focused purpose.

Used for features that:

- **Are independent from any particular component**
- **Provide shared data or logic across components**
- **Encapsulate external interactions**

How Does It Work?





Dependency Injection

A coding pattern in which a class receives the instances of objects it needs (called dependencies) from an external source rather than creating them itself.

Building a Service

Create the service class

Define the metadata with a decorator

Define the metadata with a decorator

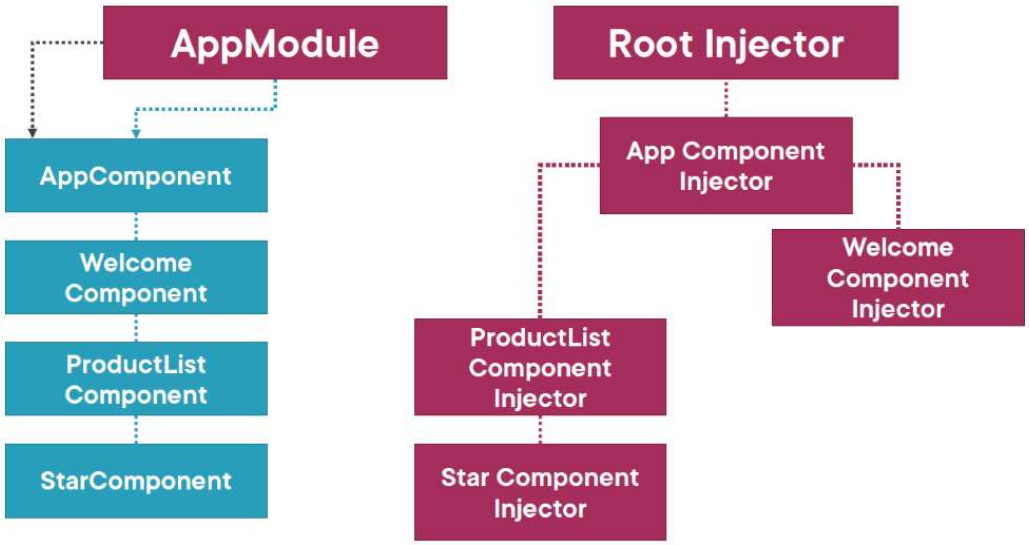
Import what we need

Building a Service

product.service.ts

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

@Injectable()
export class ProductService {

  getProducts(): IProduct[] {
  }

}
```

Angular Injectors



Registering a Service

Root Injector	Component Injector
Service is available throughout the application	Service is available ONLY to that component and its child (nested) components

Recommended for most scenarios

Isolates a service used by only one component

Provides multiple instances of the service

Registering a Service - Root Application

product.service.ts

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

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

  getProducts(): IProduct[] {
  }

}
```

OR

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 { }
```

Injecting the Service

Injector

Service

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

Component

```
constructor(private myService) {}
```

Injecting the Service

product-list.component.ts

```
...
import { ProductService } from './product.service';

@Component({
  selector: 'pm-products',
  templateUrl: './product-list.component.html'
})
export class ProductListComponent {
 private _productService;
 constructor(productService: ProductService) {
   this._productService = productService;
 }

}
```

Injecting the Service

product-list.component.ts

```
...
import { ProductService } from './product.service';

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

 constructor(private productService: ProductService) { }

}
```

Service Checklist: Building a Service

Service class
- Clear name
- Use PascalCasing

- Append "Service" to the name
- export keyword

Service decorator
- Use Injectable
- Prefix with @; Suffix with ()

Import what we need

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class ProductService {...}
```

Service Checklist: Registering a Service

Select the appropriate level in the hierarchy
- Root application injector if the service is used throughout the application
- Specific component's injector if only that component uses the service

Service Injectable decorator
- Set the providedIn property to 'root'

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

Component decorator
- Set the providers property to the service

Service Checklist: Dependency Injection

Specify the service as a dependency

Use a constructor parameter

Service is injected when component is instantiated

```
constructor(private productService: ProductService) { }
```

Chapter 10

Observables

Reactive Extensions (RxJS)

A library for composing data using observable sequences

And transforming that data using operators
- Similar to .NET LINQ operators

Angular uses Reactive Extensions for working with data
- Especially asynchronous data

Synchronous vs. Asynchronous

Synchronous: real time

Asynchronous: No immediate response

HTTP requests are asynchronous: request and response

Getting Data

Observable

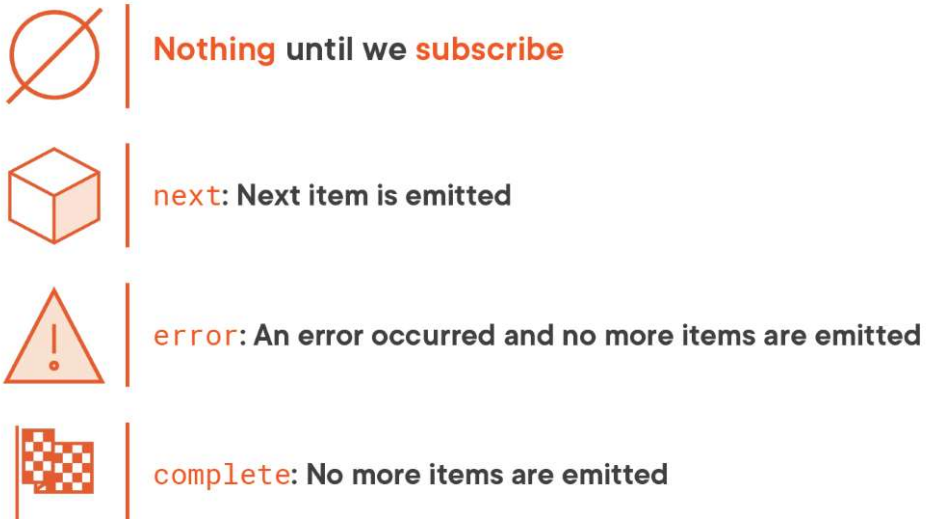
A collection of items over time

- **Unlike an array, it doesn't retain items**
- **Emitted items can be observed over time**

Array: [A, P, P, L, E]

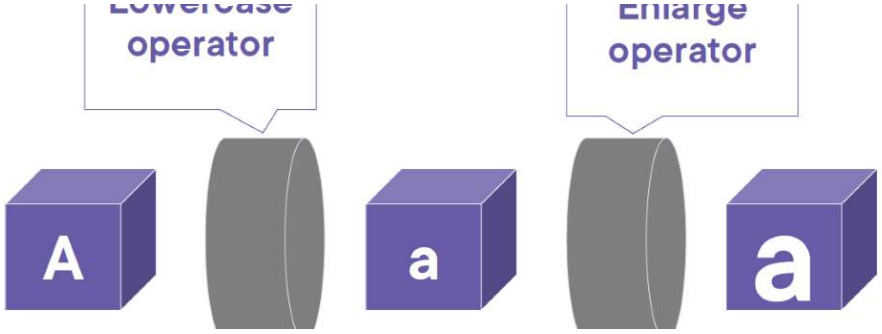
Observable:

What Does an Observable Do?



Getting Data

Observable Pipe



Common Observable Usage

Start the Observable (subscribe)

Pipe emitted items through a set of operators

Process notifications: next, error, complete

Stop the Observable (unsubscribe)

Example

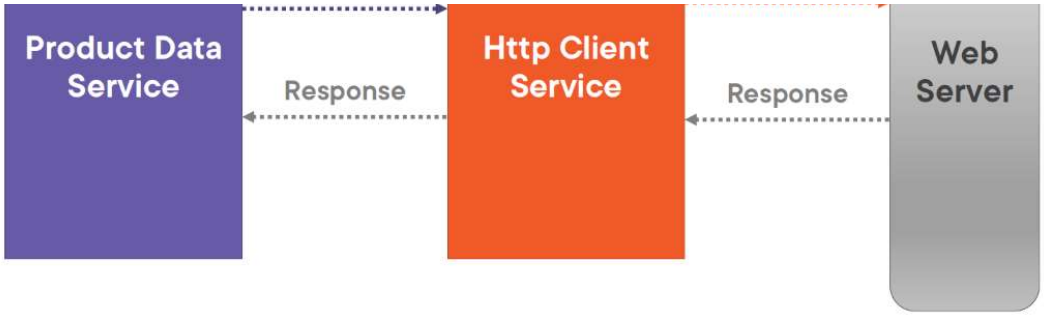
Example

```
import { Observable, range, map, filter } from 'rxjs';

const source$: Observable<number> = range(0, 10);

source$.pipe(
   map(x => x * 3),
   filter(x => x % 2 === 0)
).subscribe(x => console.log(x));
```

Result

```
0
6
12
18
24
```

Sending an HTTP Request



Setting up an HTTP Request

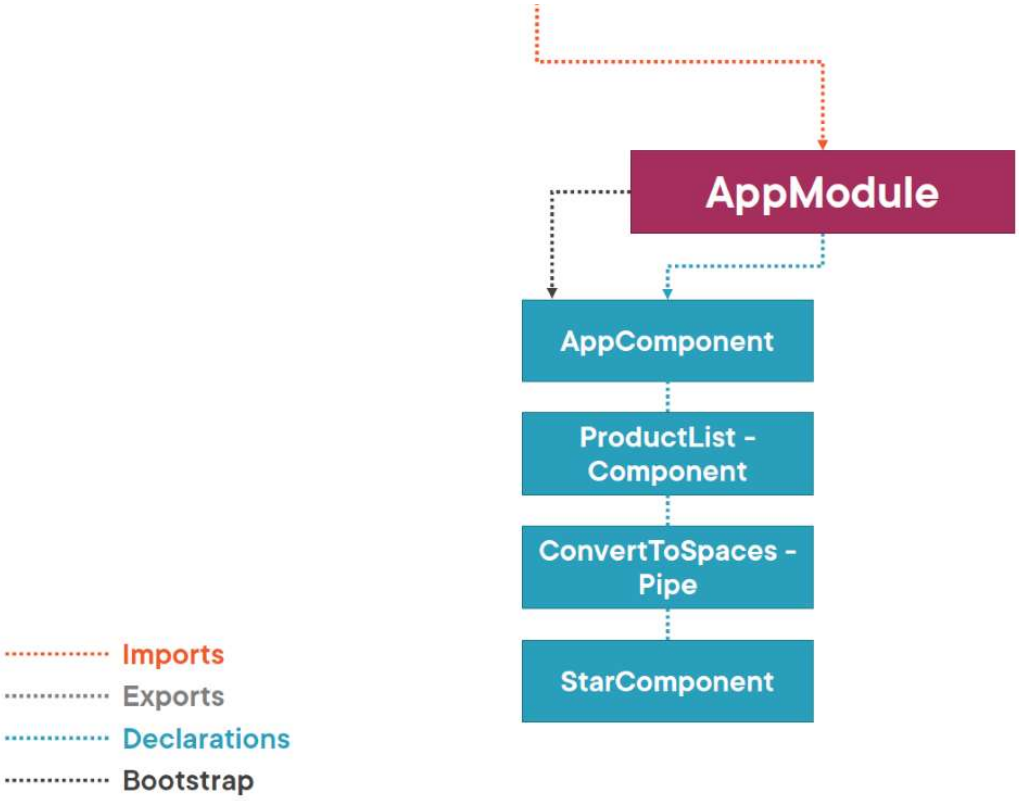
product.service.ts

```
...
import { HttpClient } from '@angular/common/http';

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts() {
   return this.http.get(this.productUrl);

  }
}
```

Registering the HTTP Service Provider

app.module.ts

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

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



Setting up an HTTP Request

product.service.ts

```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
   providedIn: 'root'
})
export class ProductService {
  private productUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl);

  }
}
```

Exception Handling

product.service.ts

```
...
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, catchError, tap } from 'rxjs';
...

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl).pipe(
     tap(data => console.log('All: ', JSON.stringify(data))),
     catchError(this.handleError)
```

```
      catchError(this.handleError)
    );
  }

  private handleError(err: HttpErrorResponse) {
  }
```

Subscribing to an Observable

```
x.subscribe()

x.subscribe(Observer)

x.subscribe({
    nextFn,
    errorFn,
    completeFn
})

const sub = x.subscribe({
     nextFn,
     errorFn,
     completeFn
})
```

Subscribing to an Observable

product.service.ts

```
getProducts(): Observable<IProduct[]> {
 return this.http.get<IProduct[]>(this.productUrl).pipe(
   tap(data => console.log('All: ', JSON.stringify(data))),
   catchError(this.handleError)
 );
}
```

product-list.component.ts

```
ngOnInit(): void {
   this.productService.getProducts().subscribe({
      next: products => this.products = products,
      error: err => this.errorMessage = err
   });
}
```

Asynchronous Operation flow,

```
})
export class ProductService {
  private productUrl = 'api/products/products.json';

  constructor(pr  3  http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
    return this.http.get<IProduct[]>(this.productUrl).pipe(
      tap(data => console.log('All', JSON.stringify(data))),
      catchError(this.handleError)
    );
  }
```

```
    return this.products.filter((product: IProduct) =>
      product.productName.toLocaleLowerCase().includes(f
  }

  toggleImage(): void {
    this.showImage = !this.showImage;
  }  1
  ngOnInit(): void {          2    4
    this.productService.getProducts().subscribe({
      next: products => this.products = products,
      error: err  => this.errorMes      = err
    });                          []
```

```
  private handleError(err: HttpErrorResponse) {
    // in a real world app, we may send the server to some r
    // instead of just logging it to the console
```

```
    this.filteredProducts = this.products;
  }
```

Asynchronous Response received flow,

```
@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private productUrl = 'api/products/products.json';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
    return this.http.get<IProduct[]>(this.productUrl).pipe(
      tap(data => console.log('All', JSON.stringify(data))),
      catchError(this.handleError)
    );
  }

  private handleError(err: HttpErrorResponse) {
    // in a real world app, we may send the server to some r
    // instead of just logging it to the console
```

```
performFilter(filterBy: string): IProduct[] {
  filterBy = filterBy.toLocaleLowerCase();
  return this.products.filter((product: IProduct) =>
    product.productName.toLocaleLowerCase().includes(filt
}

toggleImage(): void {
  this.showImage = !this.showImage;
}

ngOnInit(): void {
  this.productService.getProd
    .subscribe({
    next: products => this.products = products,
    error: err => this.errorMessage = err
  });
  this.filteredProducts = this.products;
}
```

Unsubscribing from an Observable

Store the subscription in a variable

Implement the OnDestroy lifecycle hook

Use the subscription variable to unsubscribe

Unsubscribing from an Observable

product-list.component.ts

```
ngOnInit(): void {
   this.sub = this.productService.getProducts().subscribe({
      next: products => this.products = products,
      error: err => this.errorMessage = err
   });
}
```

```
ngOnDestroy(): void {
   this.sub.unsubscribe();
}
```

HTTP

Checklist: Setup

Add HttpClientModule to the imports array of one of the application's Angular Modules

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

HTTP Checklist: Calling HTTP Get

Define a dependency for the Http Client Service in the constructor

Create a method for each HTTP request

Call the desired HTTP method, such as get

Use generics to specify the returned type

```
export class ProductService {
  private productUrl = 'www.myService.com/api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<IProduct[]> {
   return this.http.get<IProduct[]>(this.productUrl);
  }
}
```

HTTP Checklist: Exception Handling

Add error handling

```
getProducts(): Observable<IProduct[]> {
  return this.http.get<IProduct[]>(this.productUrl).pipe(
    tap(data => console.log(JSON.stringify(data))),
    catchError(this.handleError)
  );
}

private handleError(err: HttpErrorResponse) {
}
```

HTTP Checklist:

Subscribing

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

Provide a function to handle any returned errors

```
ngOnInit(): void {
   this.productService.getProducts().subscribe({
      next: products => this.products = products,
      error: err => this.errorMessage = err
   });
}
```

HTTP Checklist: Unsubscribing

Store the subscription in a variable

```
this.sub = this.ps.getProducts().subscribe(…)
```

Implement the OnDestroy lifecycle hook

```
export class PLComponent implements OnInit, OnDestroy
```

Use the subscription variable to unsubscribe

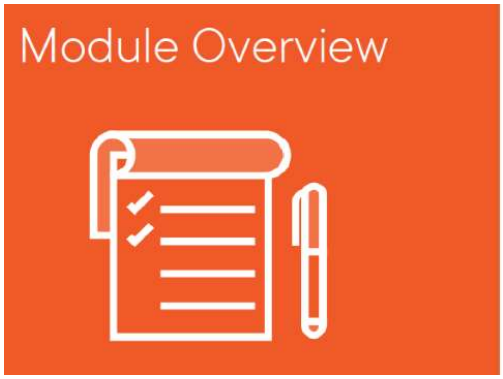
```
ngOnDestroy(): void {
   this.sub.unsubscribe();
}
```

Learning More

Pluralsight Courses

- "Angular: Reactive Forms"
  - HTTP and CRUD
- "RxJS in Angular: Reactive Development"
  - RxJS and Observables
- "Angular HTTP Communication"
  - Intermediate HTTP Techniques

Chapter 11:



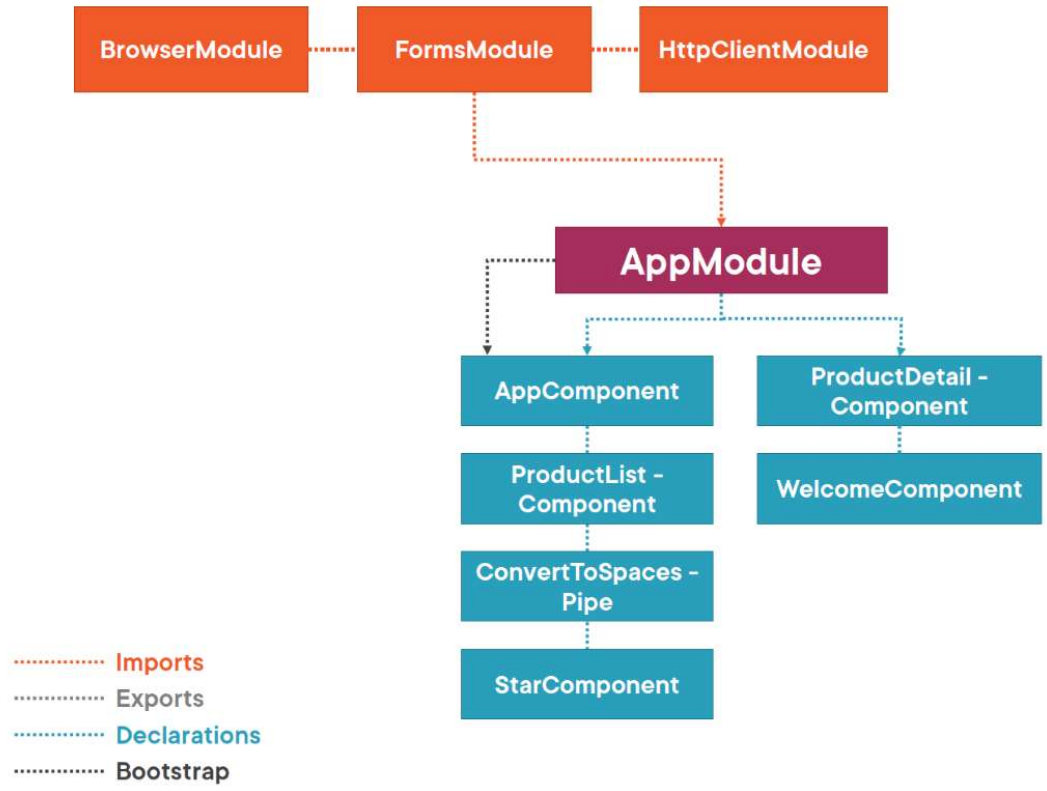
How does routing work?

Configuring routes

Tying routes to actions

Placing the views

CMD: ng g c products/product-detail –flat



How Routing Works

Configure a route for each component

Define options/actions

Tie a route to each option/action

Activate the route based on user action

Activating a route displays the component's view

How Routing Works


```
<a routerLink="/products">Product List</a>
```

```
{ path: 'products', component: ProductListComponent }
```

```
<router-outlet></router-outlet>
```

product-list.component.ts

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

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

Configuring Routes

app.module.ts

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

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

Registers the router service
Declares the router directives
Exposes configured routes

Configuring Routes

app.module.ts

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

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

Configuring Routes

www.myWebService.com

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

Order Matters as below, "FIRST-MATCH-WINS STRATEGY"

Configuring Routes

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

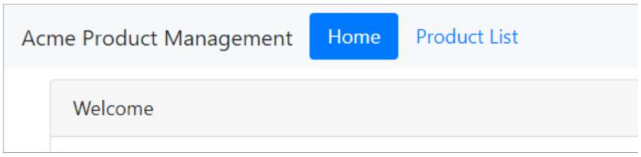
Navigating the Application Routes

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

Tying Routes to Actions



Tying Routes to Actions

app.component.ts

app.component.ts

```
@Component({
    selector: 'pm-root',
    template: `
     <ul class='nav navbar-nav'>
       <li><a>Home</a></li>
       <li><a>Product List</a></li>
     </ul>
    `
})
```

Tying Routes to Actions

app.component.ts

```
@Component({
    selector: 'pm-root',
    template: `
     <ul class='nav navbar-nav'>
       <li><a [routerLink]="['/welcome']">Home</a></li>
       <li><a [routerLink]="['/products']">Product List</a></li>
     </ul>
    `
})
```

```
       <li><a routerLink="/welcome">Home</a></li>
       <li><a routerLink="/products">Product List</a></li>
```

Placing the Views

app.component.ts

```
@Component({
    selector: 'pm-root',
    template: `
     <ul class='nav navbar-nav'>
       <li><a [routerLink]="['/welcome']">Home</a></li>
       <li><a [routerLink]="['/products']">Product List</a></li>
     </ul>
     <router-outlet></router-outlet>
    `
})
```

How Routing Works

```
})
export class ProductListComponent { }
```

Routing Checklist: Displaying Components

Nest-able components

- Define a selector

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

- Nest in another component

```
<div><h1>{{pageTitle}}</h1>
  <pm-products></pm-products>
</div>
```

- No route

Routed components

- No selector or nesting
- Configure routes

```
[
 { path: 'products', component: ProductListComponent }
]
```

- Tie routes to actions

Routing Checklist: Doing Routing

Configure routes

Tie routes to actions

Place the view

Routing Checklist: Configuring Routes

Define the base element

```
<head>
  ...
  <base href="/">
</head>
```

Add RouterModule

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

Routing Checklist: Configuring Routes

Add each route to forRoot array
- Order matters

path: Url segment for the route
- No leading slash
- '' for default route
- '**' for wildcard route

component
- Not string name; not enclosed in quotes

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

Routing Checklist: Tying Routes to Actions

Add the RouterLink directive as an attribute
- Clickable element
- Enclose in square brackets

Bind to a link parameters array
- First element is the path
- All other elements are route parameters

```
<ul>
  <li><a [routerLink]="['/welcome']">Home</a></li>
```

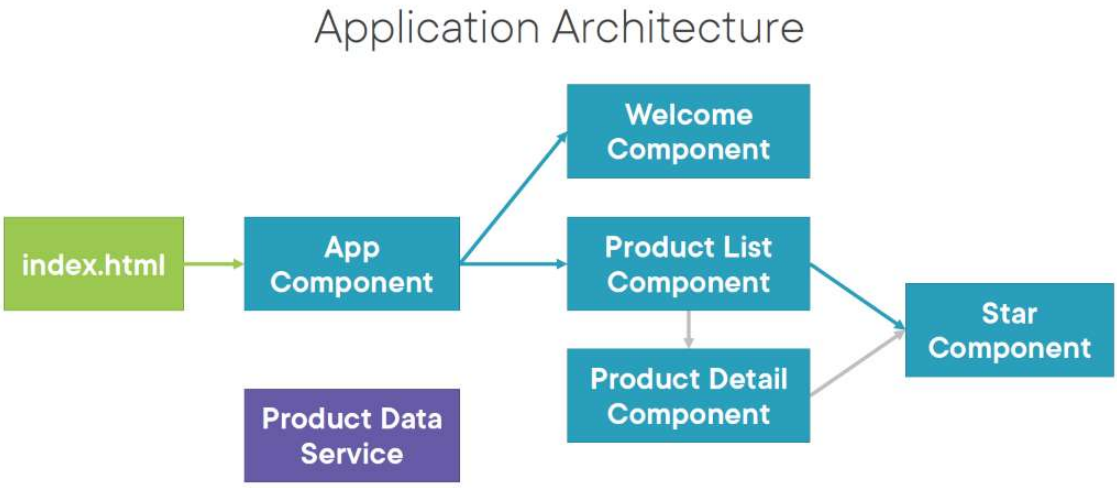
```
  <li><a [routerLink]="['/products']">Product List</a><
/li>
</ul>
```

Routing Checklist: Placing the View

Add the RouterOutlet directive

- Identifies where to display the routed component's view
- Specified in the host component's template

```
<ul>
  <li><a [routerLink]="['/welcome']">Home</a></li>
  <li><a [routerLink]="['/products']">Product List</a><
/li>
</ul>
<router-outlet></router-outlet>
```



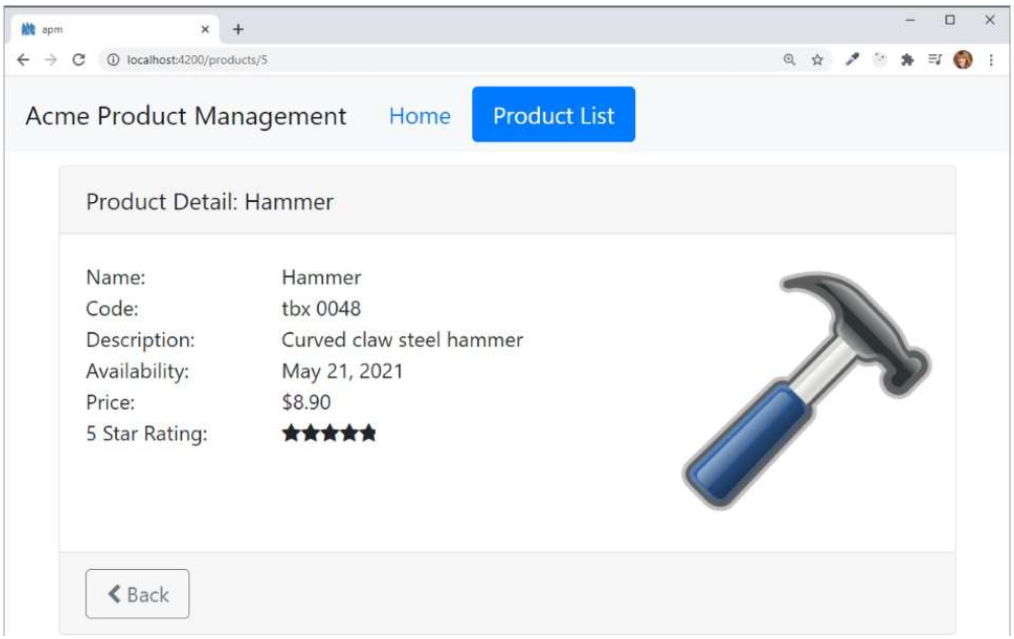
Chapter 12:

Passing parameters to a route

Activating a route with code

Protecting routes with guards

Passing Parameters to a Route



Passing Parameters to a Route

app.module.ts

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

-

Passing Parameters to a Route

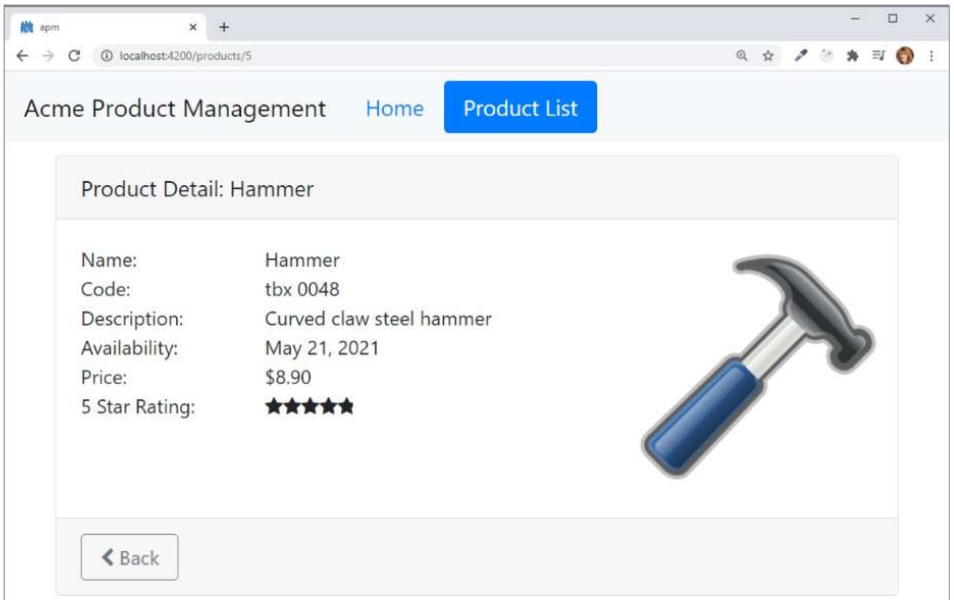
product-list.component.html

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

app.module.ts

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

Reading Parameters from a Route



Reading Parameters from a Route

product-detail.component.ts

```
import { ActivatedRoute } from '@angular/router';

    constructor(private route: ActivatedRoute) { }
```

app.module.ts

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

Reading Parameters from a Route

Snapshot : Read the parameter one time

```
this.route.snapshot.paramMap.get('id');
```

Observable: Read emitted parameters as they change

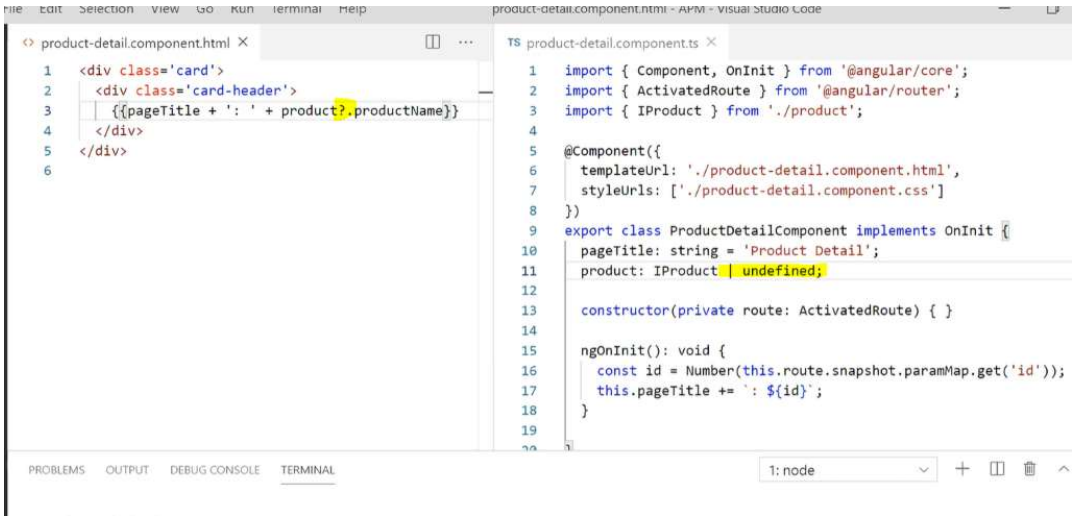
```
this.route.paramMap.subscribe(
  params => console.log(params.get('id'))
);
```

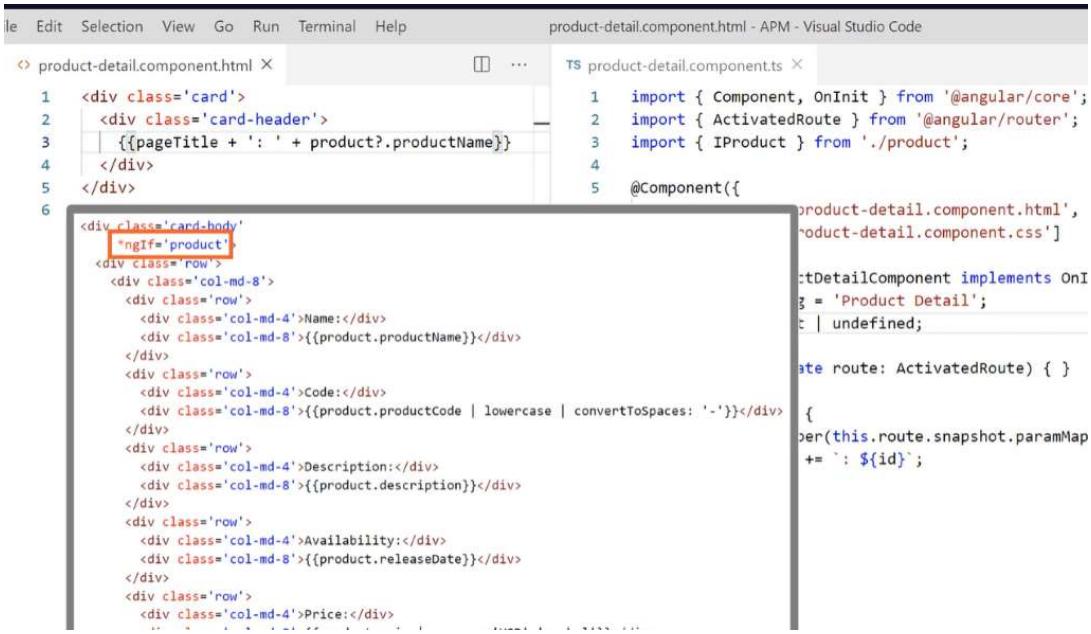
Specified string is the route parameter name

```
{ path: 'products/:id',
```

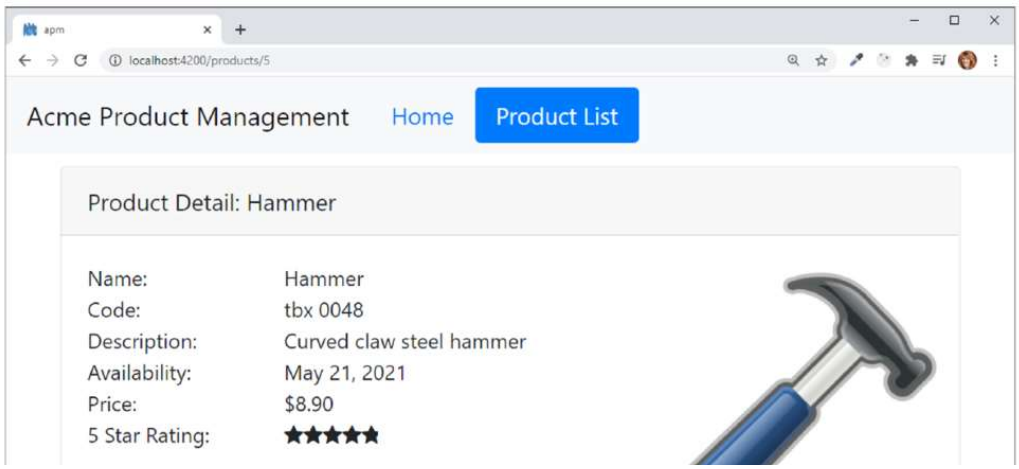
```
component: ProductDetailComponent }
```

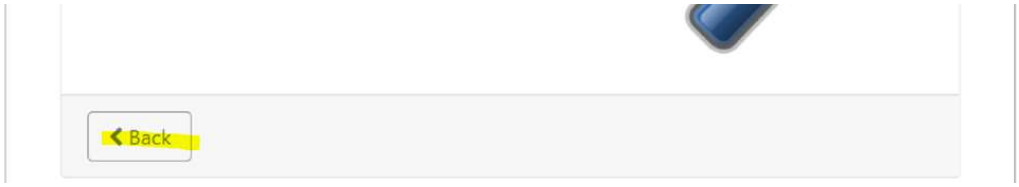
Handling Null & Undefined:





Activating a Route with Code

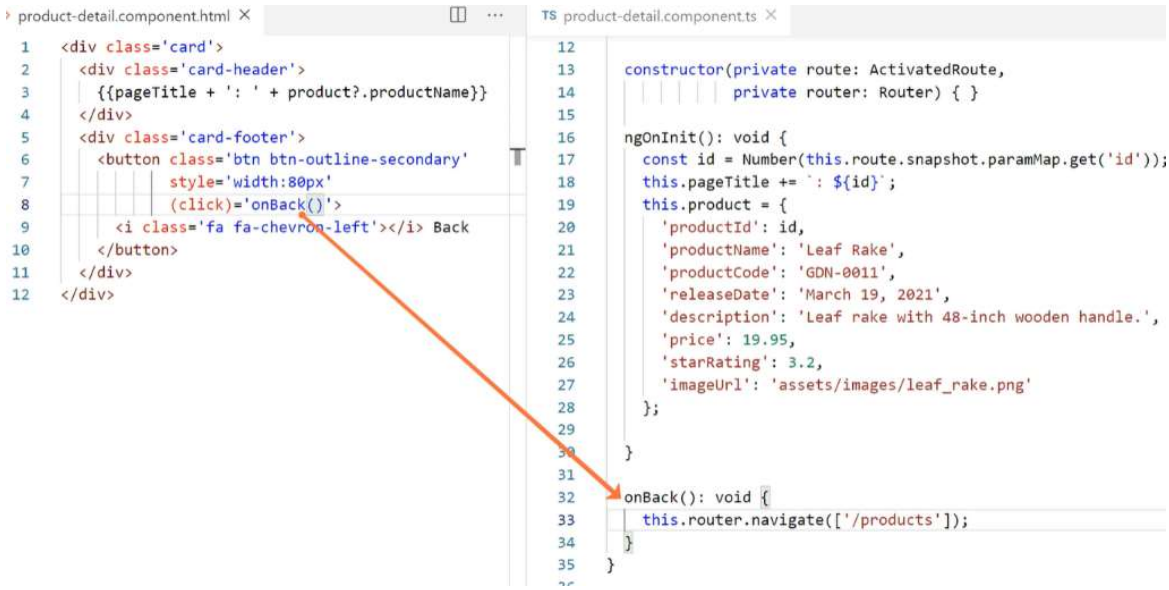




Activating a Route with Code

product-detail.component.ts

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

    onBack(): void {
      this.router.navigate(['/products']);
    }
```



Protecting Routes with Guards

Limit access to a route

Restrict access to only certain users

Require confirmation before navigating away

Require confirmation before navigating away

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

Building a Guard

product-detail.guard.ts

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

@Injectable({
  providedIn: 'root'
})
export class ProductDetailGuard implements CanActivate {

    canActivate(): boolean {
      ...
    }
}
```

Using a Guard

app.module.ts

```
@NgModule({
  imports: [
    ...,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },
      { path: 'products/:id',
        canActivate: [ ProductDetailGuard ],
        component: ProductDetailComponent },
      ...])
  ],
  declarations: [...],
  bootstrap: [ AppComponent ]
})
```

```
export class AppModule { }
```

Ng CLI Command to build a guard:

ng g g products/product-detail

product-detail.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ProductDetailGuard implements CanActivate {

  constructor(private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
      const id = Number(route.paramMap.get('id'));
      if (isNaN(id) || id < 1) {
        alert('Invalid product id');
        this.router.navigate(['/products']);
        return false;
      }
    return true;
  }

}
```

General Checklist: Null and Undefined

Prevent null or undefined errors in a template

- Safe navigation operator (?)

```
{{product?.productName}}
```

```
{{product?.supplier?.companyName}}
```

- *ngIf directive

```
<div *ngIf='product'>
  <div>Name:</div>
  <div>{{product.productName}}</div>
  <div>Description:</div>
  <div>{{product.description}}</div>
</div>
```

Routing Checklist: Passing Parameters

app.module.ts

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

product-list.component.html

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

product-detail.component.ts

```
import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {
   console.log(this.route.snapshot.paramMap.get('id'));
}
```

Routing Checklist: Activate a Route with Code

Use the Router service
- Define it as a dependency

Create a method that calls the navigate method of the Router service
- Pass in the link parameters array

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

    onBack(): void {
      this.router.navigate(['/products']);
    }
```

Add a user interface element
- Use event binding to bind to the created method

```
<button (click)='onBack()'>Back</button>
```

Routing Checklist: Protecting Routes with Guards

Build a guard service
- Implement the guard type (CanActivate)
- Create the method (canActivate())

Register the guard service provider
- Use the providedIn property

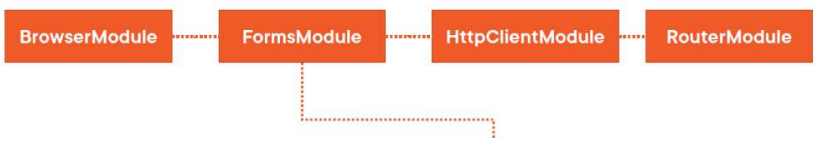
```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';

@Injectable({ providedIn: 'root' })
export class ProductDetailGuard implements CanActivate {
    canActivate(): boolean { ... }
}
```

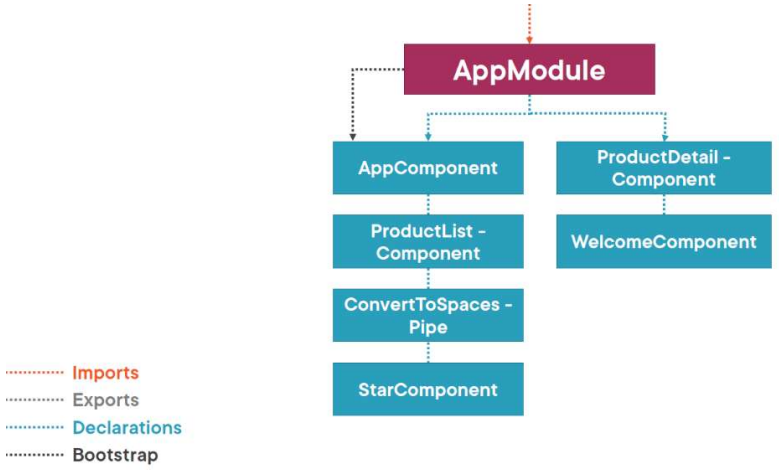
Add the guard to the desired route

```
{ path: 'products/:id', canActivate: [ ProductDetailGuard ],
  component: ProductDetailComponent },
```

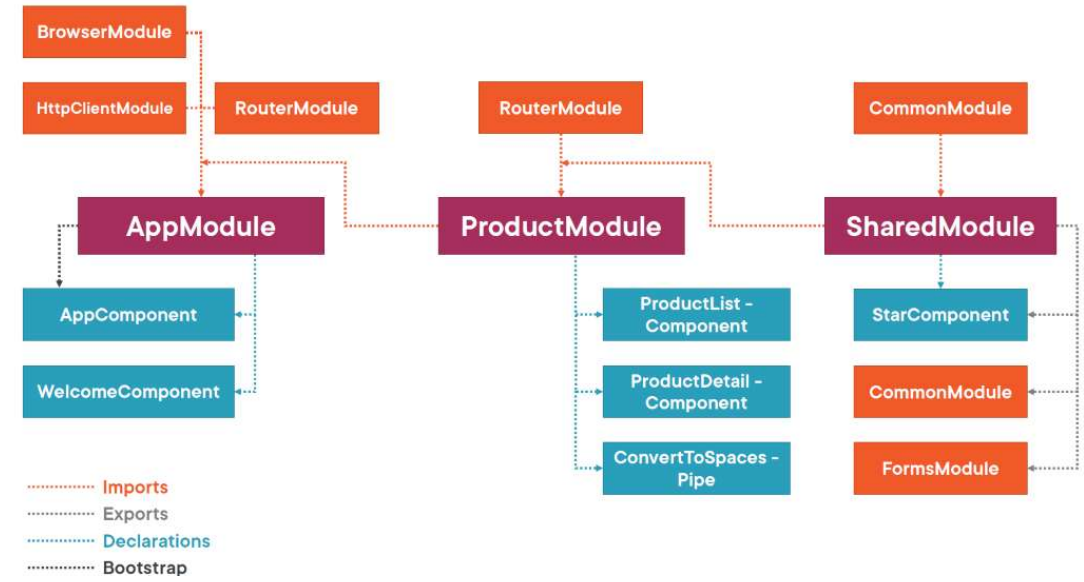
Learning More

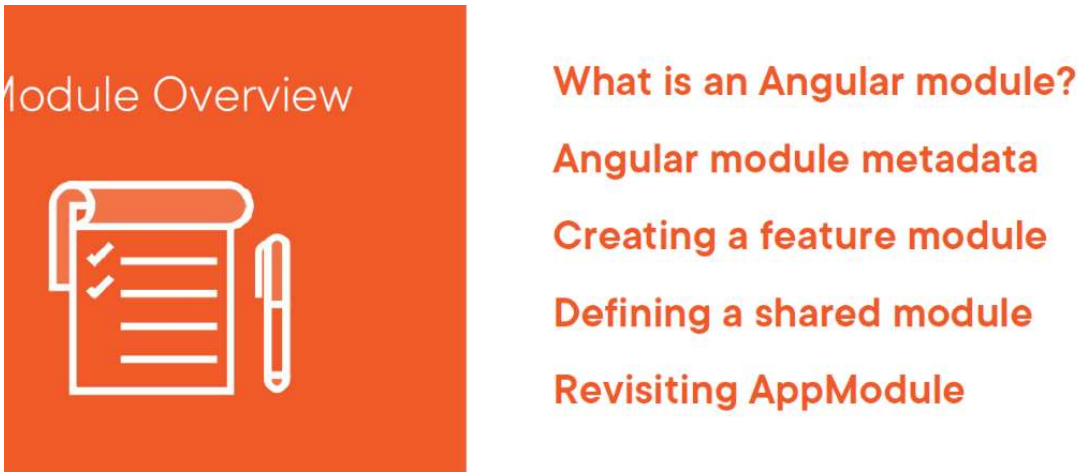
"Angular Routing"
- Required, optional, & query parameters
- Prefetching with route resolvers
- Child and secondary router outlets
- Router guards
- Lazy loading





Chapter 13:





What Is an Angular Module?

A class with an NgModule decorator

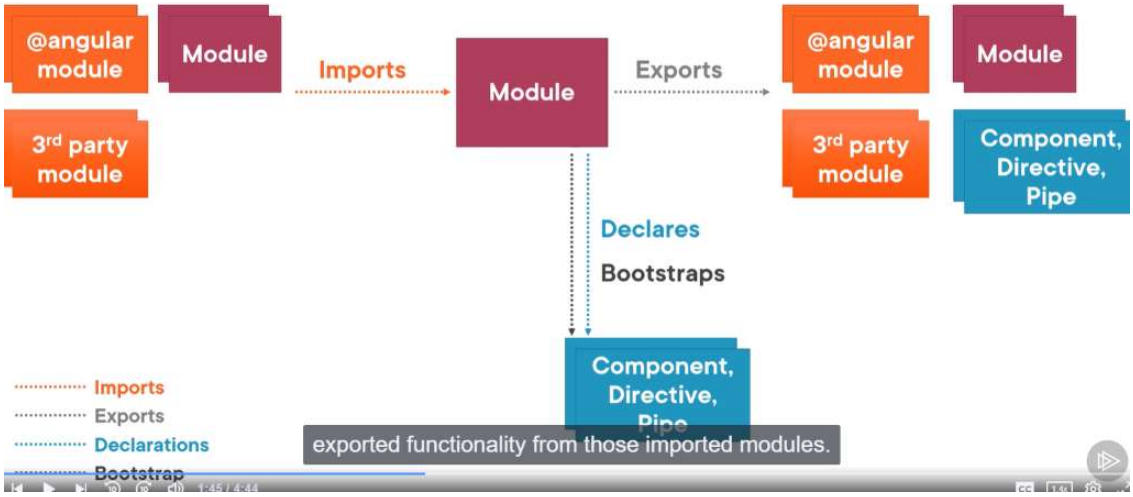
Its purpose:

- Organize the pieces of our application

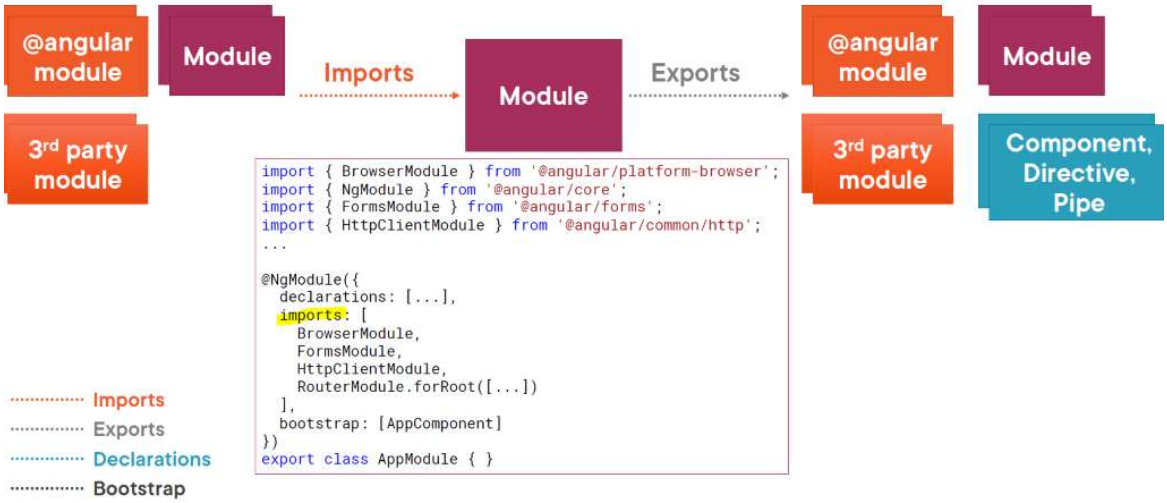
Module

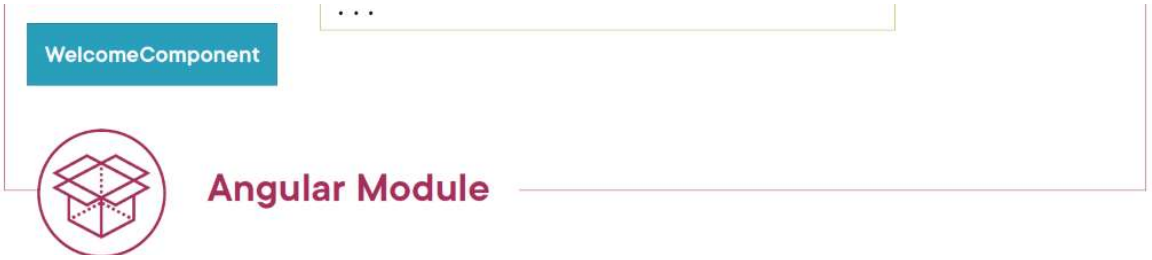
- Arrange them into blocks
- Extend our application with capabilities from external libraries
- Provide a template resolution environment
- Aggregate and re-export

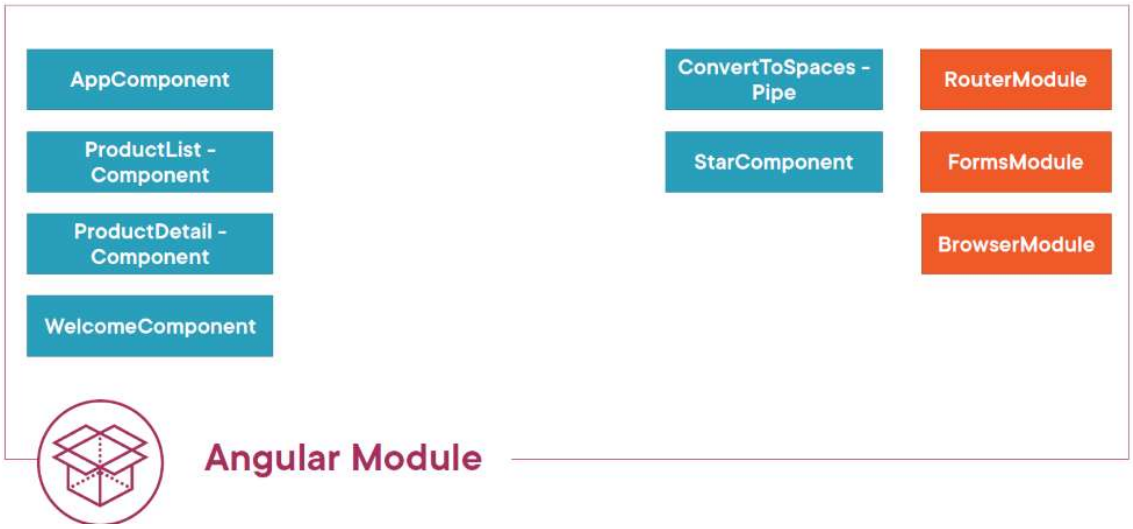
Angular Module



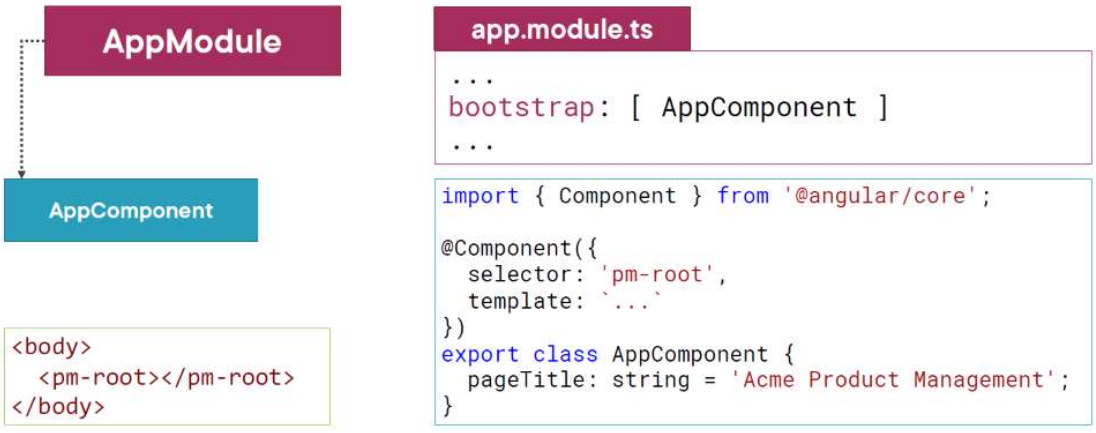
Angular Module







Bootstrap Array



Bootstrap

Bootstrap Array Truth #1

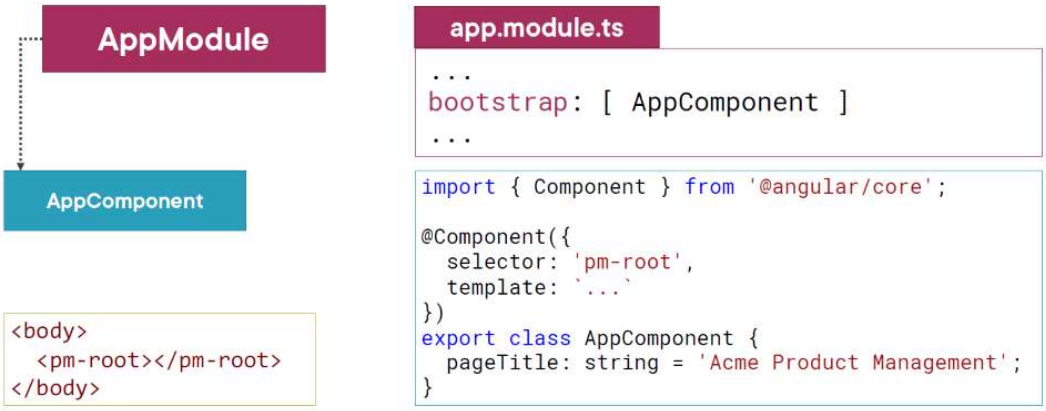
Every application must bootstrap at least one component, the root application component.

Bootstrap Array Truth #2

The bootstrap array should only be used in the root application module,

AppModule.

Bootstrap Array



Declarations Array

Declarations Array Truth #1

Every component, directive, and pipe we create must belong to one and only one Angular module.

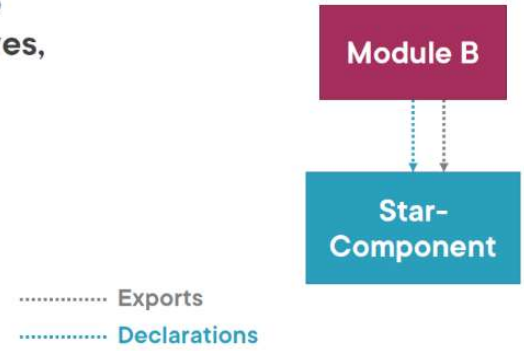
Declarations Array Truth #2

Only declare components, directives and pipes.

Declarations Array Truth #3

All declared components, directives, and pipes are private by default.

They are only accessible to other components, directives, and pipes declared in the same module.



Declarations Array Truth #4

The Angular module provides the template resolution environment for its component templates.

product-list.component.html

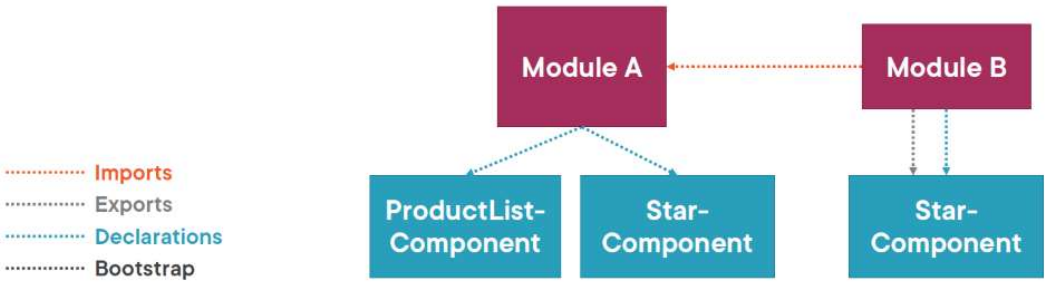
```
<pm-star ...>
</pm-star>
```

star.component.ts

```
...
@Component({
    selector: 'pm-star',
    template: ...
})
...
```

Declarations Array Truth #4

The Angular module provides the template resolution environment for its component templates.



Exports Array


```
...
exports: [
   StarComponent,
   FormsModule
]
...
```

Imports
Exports
Declarations
Bootstrap

Exports Array Truth #1

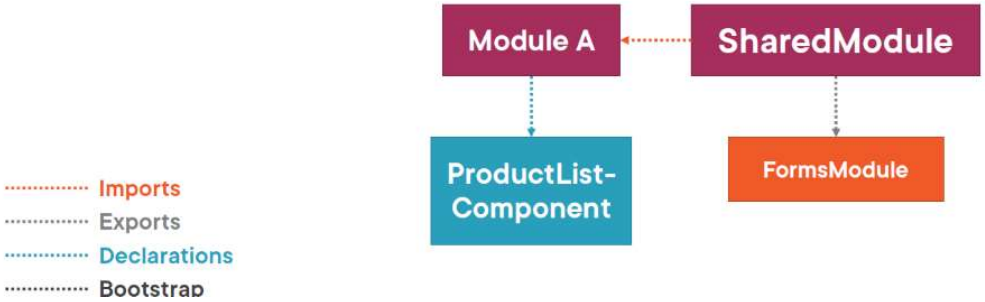
Export any component, directive, or pipe if other components need it.

Exports Array Truth #2

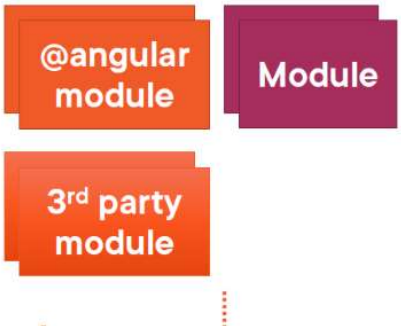
Re-export modules to re-export their components, directives, and pipes.

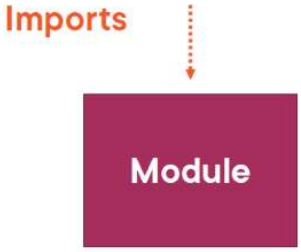
Exports Array Truth #3

We can export something without including it in the imports array.



Imports Array






```
...
imports: [
 BrowserModule,
 FormsModule,
 HttpClientModule,
 RouterModule.forRoot([...])
]
...
```

Imports Array Truth #1

Adding a module to the imports array makes available any components, directives, and pipes defined in that module's exports array.

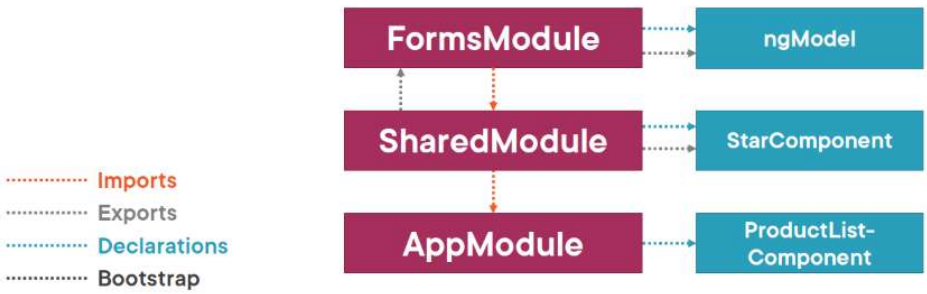
Imports Array Truth #2

Only import what this module needs.

For each component declared in the module, add to the imports array what is needed by the component's template.

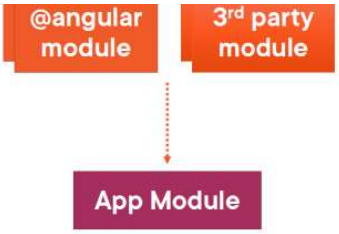
Imports Array Truth #3

Importing a module does NOT provide access to its imported modules



Imports Array Truth #4

Use the imports array to register services

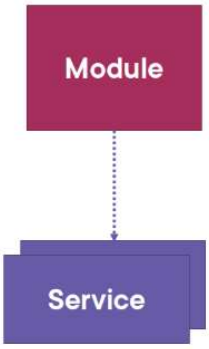


provided by Angular or third-party modules

Import the module in the AppModule to ensure its services are registered one time

Imports

Providers Array




```
...
providers: [ ProductService ]
...
```

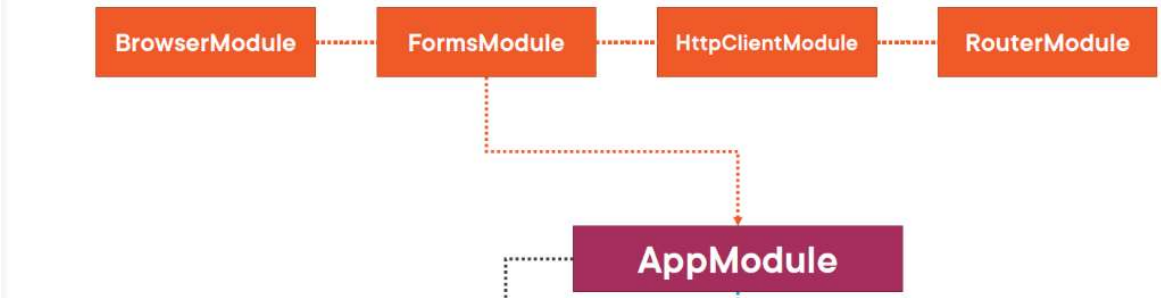
```
...
@Injectable({
  providedIn: 'root'
})
export class ProductService {
...
```

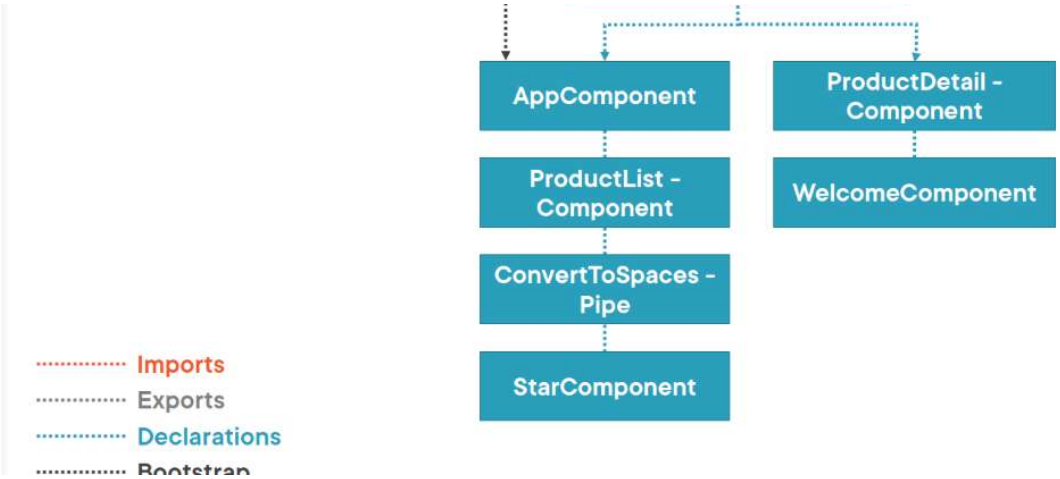
Providers

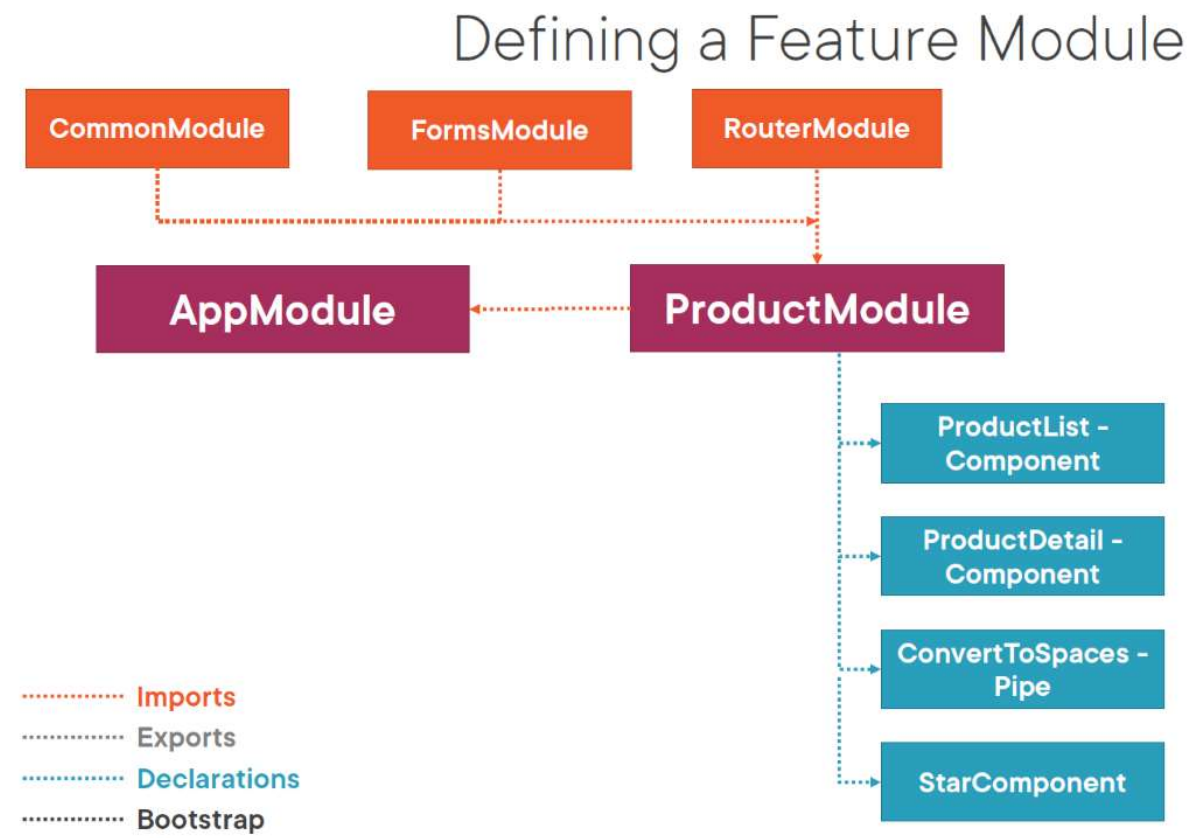
NgModule Decorator

```
@NgModule({
  declarations: [
    AppComponent,
    WelcomeComponent,
    ProductListComponent,
    ConvertToSpacesPipe,
    ProductDetailComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot([...]),
    SharedModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

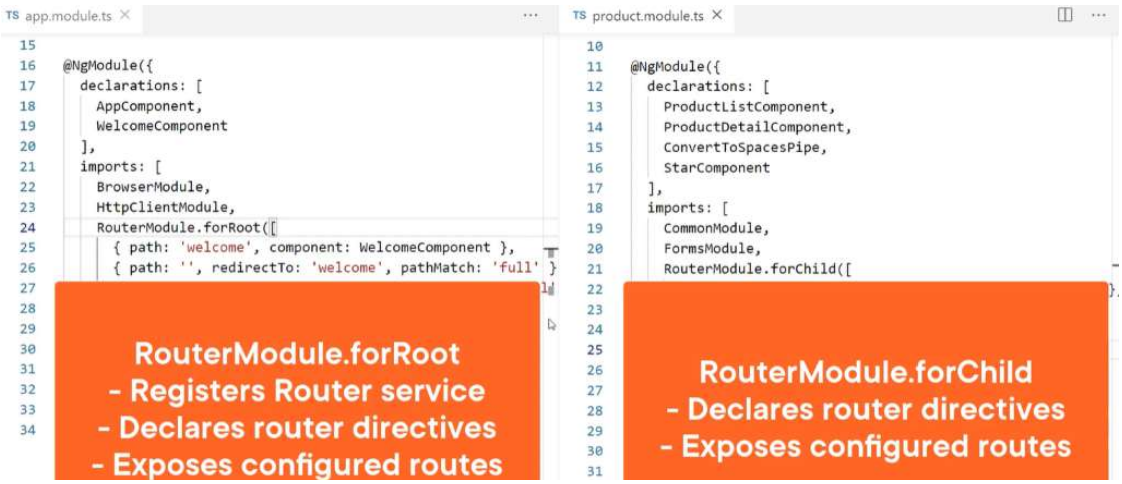
```
@NgModule({
  declarations: [
    StarComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    CommonModule,
    FormsModule,
    StarComponent
  ]
})
export class SharedModule { }
```







Cmd: ng g m products/product --flat –m app



32

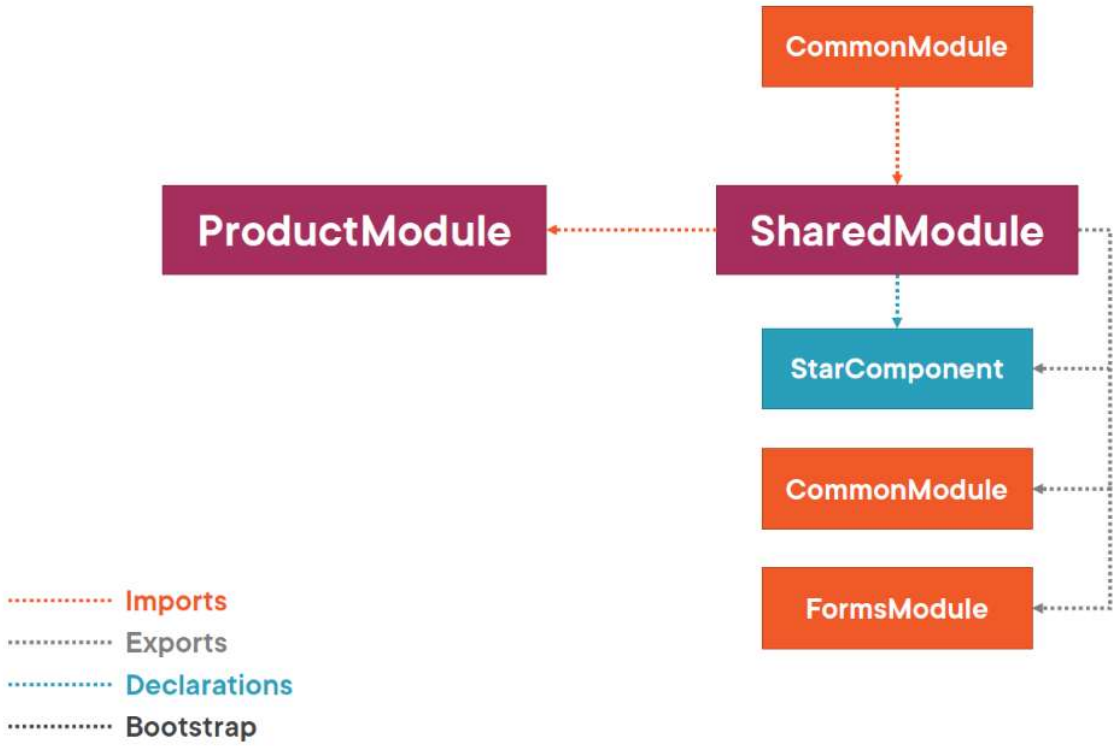
Shared Module

SharedModule

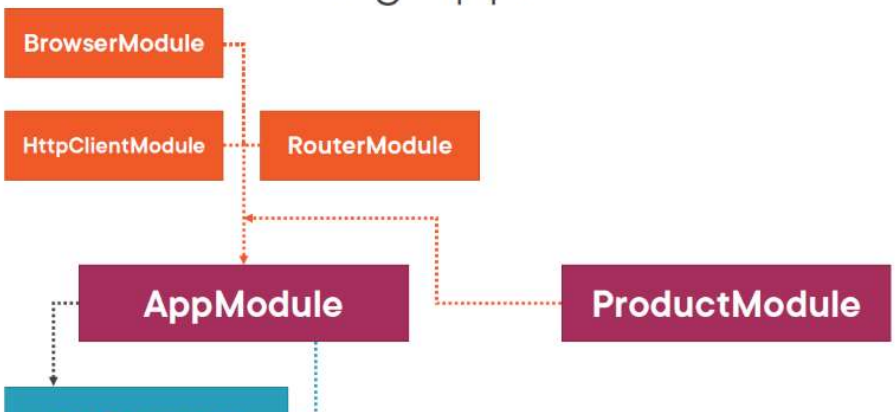
Organize commonly used pieces of our application

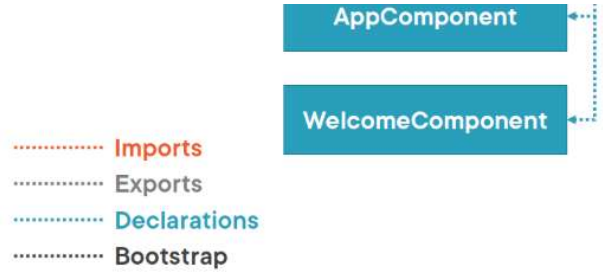
Export those pieces to share them

Defining a Shared Module



Revisiting AppModule





Angular Module Checklist: NgModule

```
@NgModule({
  declarations: [
    AppComponent,
    WelcomeComponent,
    ProductListComponent,
    ConvertToSpacesPipe,
    ProductDetailComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot([...]),
    SharedModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
@NgModule({
  declarations: [
    StarComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    CommonModule,
    FormsModule,
    StarComponent
  ]
})
export class SharedModule { }
```

Angular Module Checklist: NgModule Declarations Array

- Declarations array: What belongs to this module
- Components owned by the module
- Declare each component in one and only one module
- Directives and pipes used by the declared components

Angular Module Checklist: NgModule Imports Array

- Imports array: Modules this module needs
- Modules that provide components, directives, and pipes needed by templates associated with components declared in the module
  - Example: CommonModule, FormsModule, SharedModule
- Modules that provide system or third-party services
  - Example: HttpClientModule
  - Imported into AppModule
- Feature modules
  - Example: ProductModule, InvoiceModule

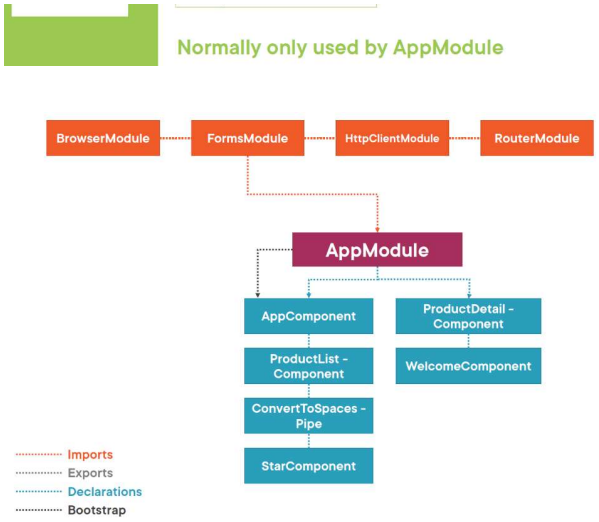
Angular Module Checklist: NgModule Exports Array

- Exports array: Pieces to share
- Components, directives, and pipes
  - Example: StarComponent
- Other modules
  - Example: CommonModule, FormsModule
- Often only used by shared modules

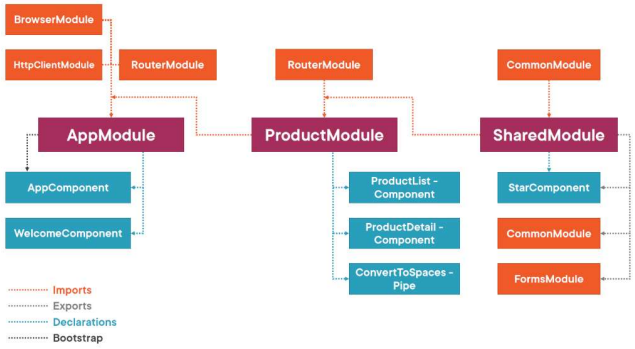
Angular Module Checklist: NgModule Bootstrap Array

- Bootstrap array: What the index.html file needs
- Component with selector used in index.html

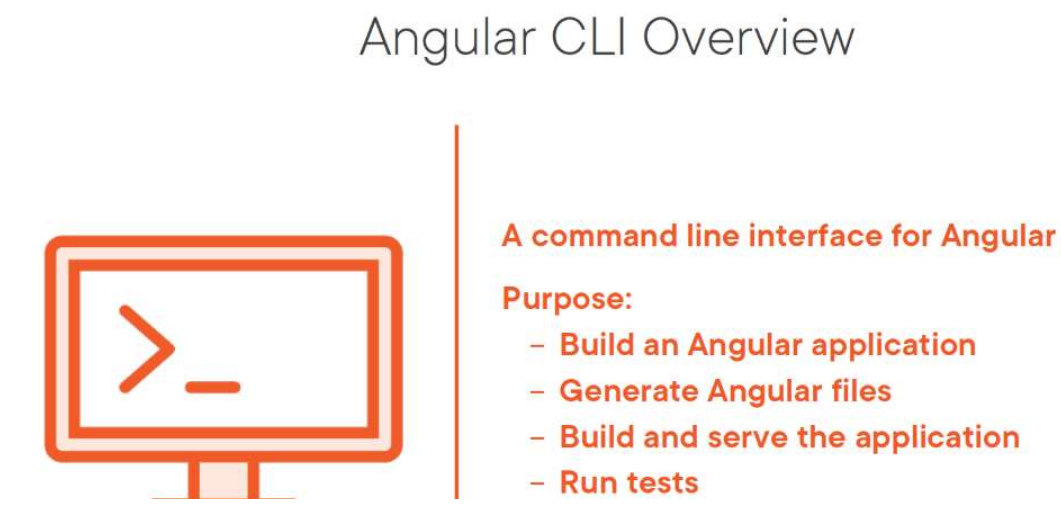
```
<body>
  <pm-root></pm-root>
</body>
```



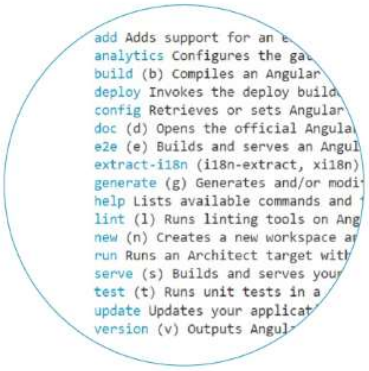
EVOLVED TO BELOW,



Chapter 14: Building, Testing and Deploying with the CLI



- Prepare the application for deployment



Angular CLI Command Syntax

ng <command> <args> --<options>

ng new hello-world --prefix hw

Angular CLI

Global Angular CLI

Installing the Angular CLI

npm install -g @angular/cli

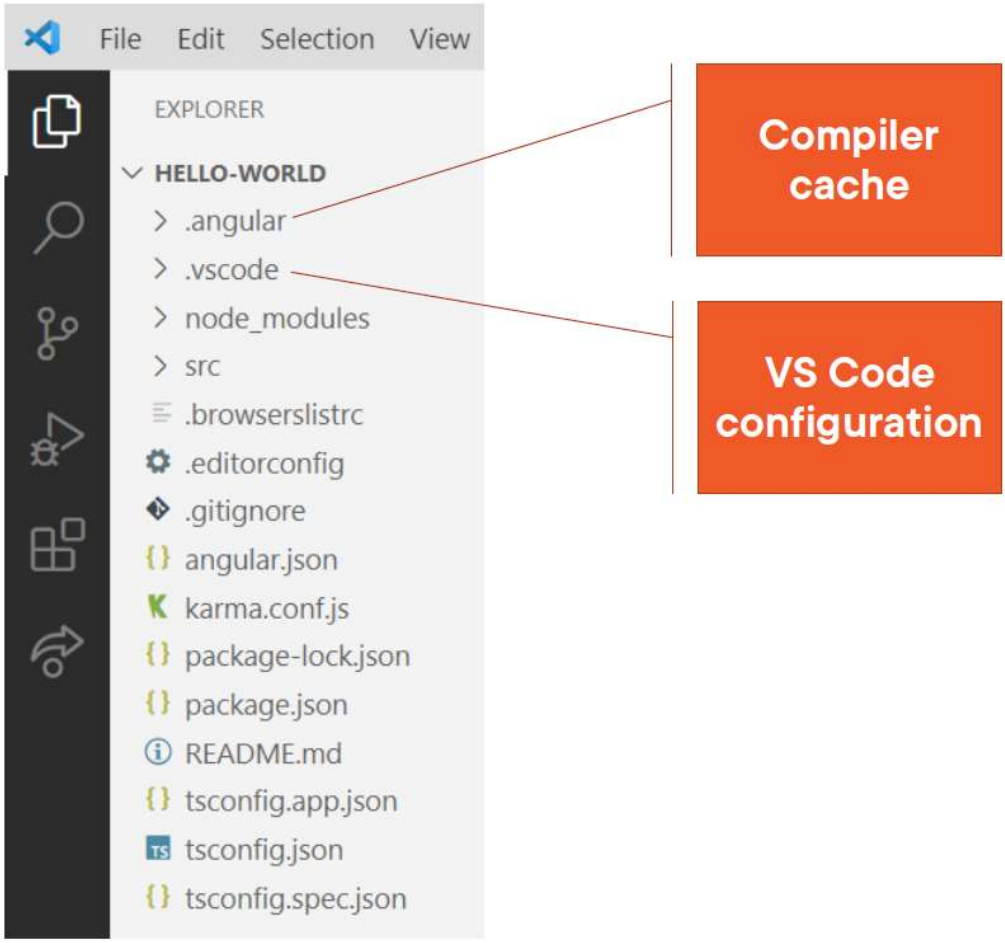
Demo

Generate a new Angular application

(ng new)

Examine the generated files

Addition Folders



Serving the Application (ng serve)

Compiles the application

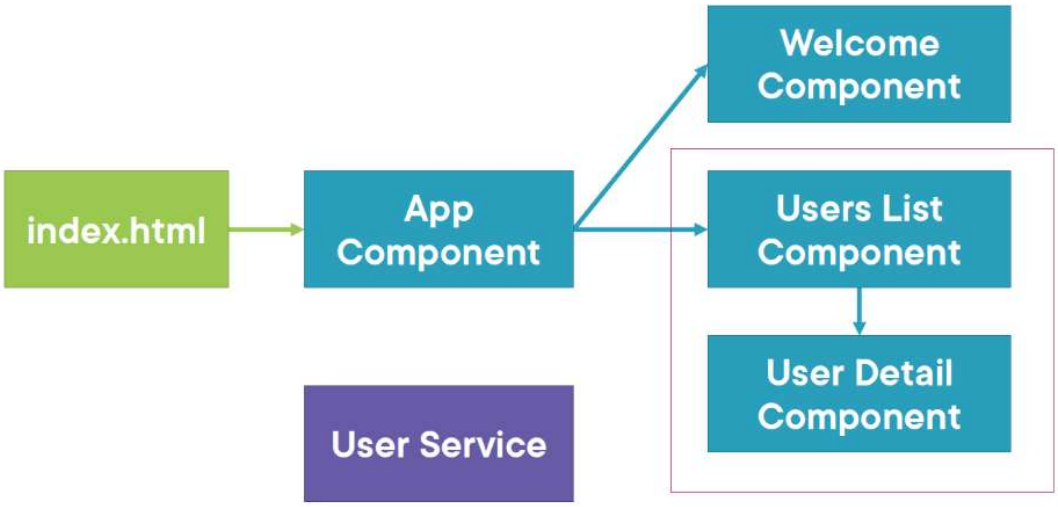
Generates application bundles

Starts a local web server

Serves the application from memory

Rebuilds on file changes

Generating Code (ng generate)

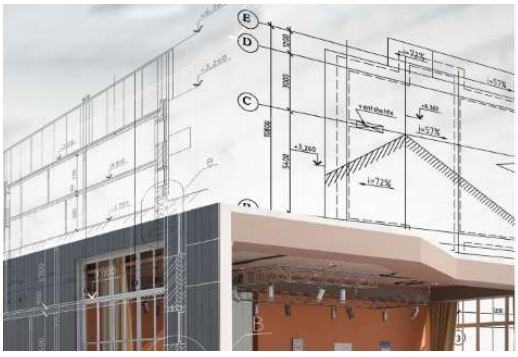


Schematic

A template-based code generator that supports complex logic.

It is a set of instructions for transforming a software project by generating or modifying code.

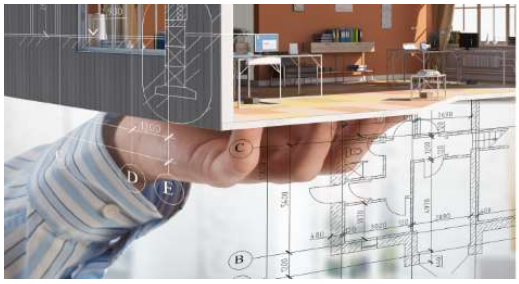
Angular documentation: https://angular.io/guide/schematics



Components (ng g c <name>)

Directives (ng g d <name>)

Route guards (ng g g <name>)



Interfaces (ng g i <name>)

Modules (ng g m <name>)

Pipes (ng g p <name>)

Services (ng g s <name>)

Testing the application (ng test)

Building the application (ng build)

Angular CLI Checklist: Commands

- ng help - Displays available commands
- ng new - Creates a new Angular application
- ng serve - Builds the app and launches a server
- ng generate - Generates code
- ng add - Adds support for an external library to the app
- ng test - Runs unit tests
- ng e2e - Runs end-to-end tests*
- ng build - Compiles into an output directory
- ng deploy - Deploys the application*
- ng update - Updates the Angular version for the app

*Requires adding an external package before using the command

Angular CLI Checklist: ng generate Commands

class	ng g cl
component	ng g c
directive	ng g d
enum	ng g e
guard	ng g g
interface	ng g i
module	ng g m
pipe	ng g p
service	ng g s

Learning More

"Angular CLI"

Chapter 15: Final Words

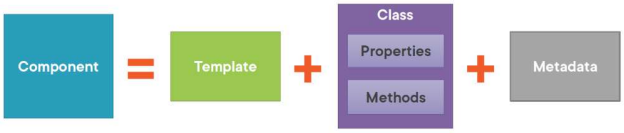
Final Words


Recap of our journey

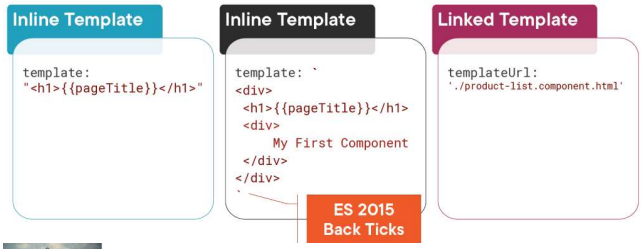

Tips for enhancing your development experience


Pointers to additional information

What Is a Component?



Where Do We Put the HTML?



When Should We Use Data Binding?

Why Do We Need a Service?

Services

How Do We Build?

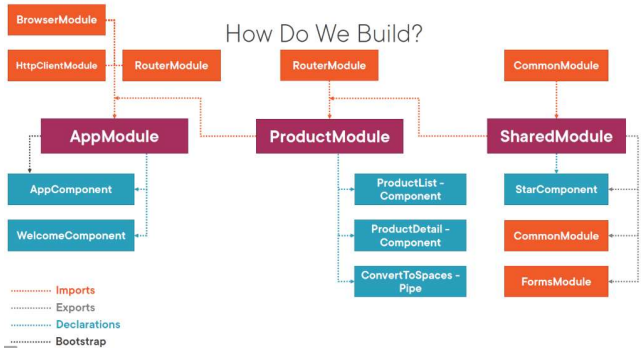
product-list.component.ts

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

@Component({
    templateUrl: './product-list.component.html',
    styleUrls: './product-list.component.css'
})
export class ProductListComponent {
 pageTitle: string = 'Product List';
}
```

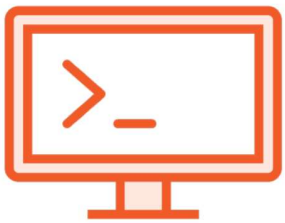
app.module.ts

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

How Do We Build?



Angular CLI



```
ng new
ng serve
ng generate
ng test & ng e2e
ng build
```

Checklists

Steps and tips

Revisit as you build

Download the slides from the Pluralsight page

Enhancing Your Development

Install the Angular Language Service extension for VS Code

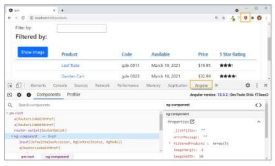
Implement lazy loading for improved load performance

Use Angular forms for building and validating user-entry forms

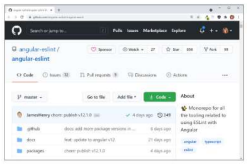
Leverage your Observable pipelines to process multiple datasets

Modify or create schematics to generate code how you want it

Enhancing Your Development



Install the Angular DevTools Chrome extension



Add ESLint using
ng add @angular-eslint/schematics

Learning More

Pluralsight Courses
- "Angular CLI"
- "Angular Forms"
- "Angular Reactive Forms"
- "RxJS in Angular: Reactive Development"
- "Angular Routing"
- "Angular Component Communication"
- "Angular Fundamentals"

Angular Documentation
- Angular.io