Angular 2 Training

By Gopi

Consolidated from multiple source

Module Overview

- What is Angular
- Why Angular 2
- Language Selection
- IDE Selection
- About Modules

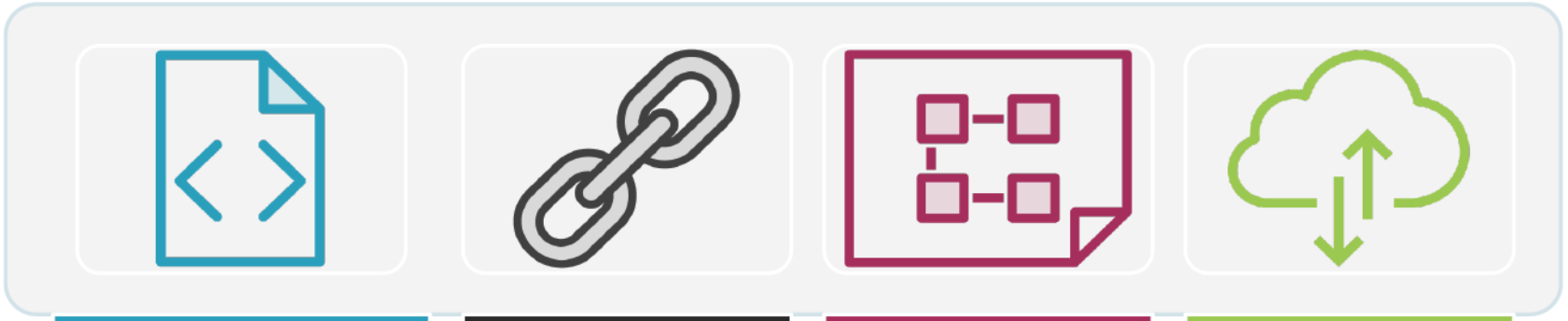
Angular Is ...

A JavaScript framework

For building client-side applications

Using HTML, CSS and JavaScript

Why Angular?



Expressive HTML

Powerful Data Binding

Modular By Design

Built-in Back-End Integration

Why Angular 2?

Built for Speed

Modern

Simplified API

Enhances Productivity

Angular 2 - Start

- Selecting a Language
- IDE Editor
- Setting up environment
- About Modules

JavaScript Language Specification

ECMAScript (ES)

ES 3

ES 5

ES 2015 (formerly known as ES 6)

- Must be transpiled

Selecting a Language

ES 5	ES 2015	TypeScript	Dart
• Runs in the browser • No compile required	• Lots of new features (classes, let, arrow, etc.)	• Superset of JavaScript • Strong typing • Great IDE tooling	• No JavaScript

What Is TypeScript?

Open source language

Superset of JavaScript

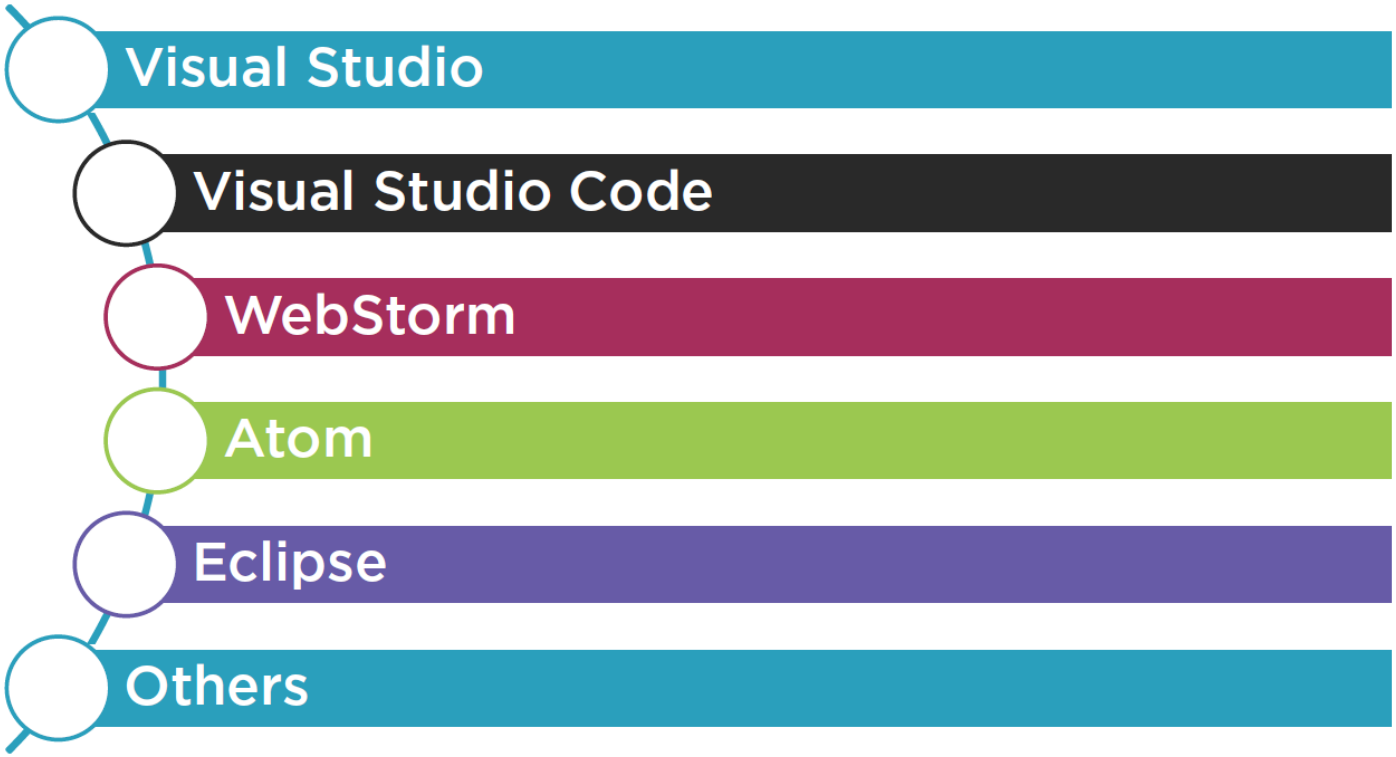
Transpiles to plain JavaScript

Strongly typed

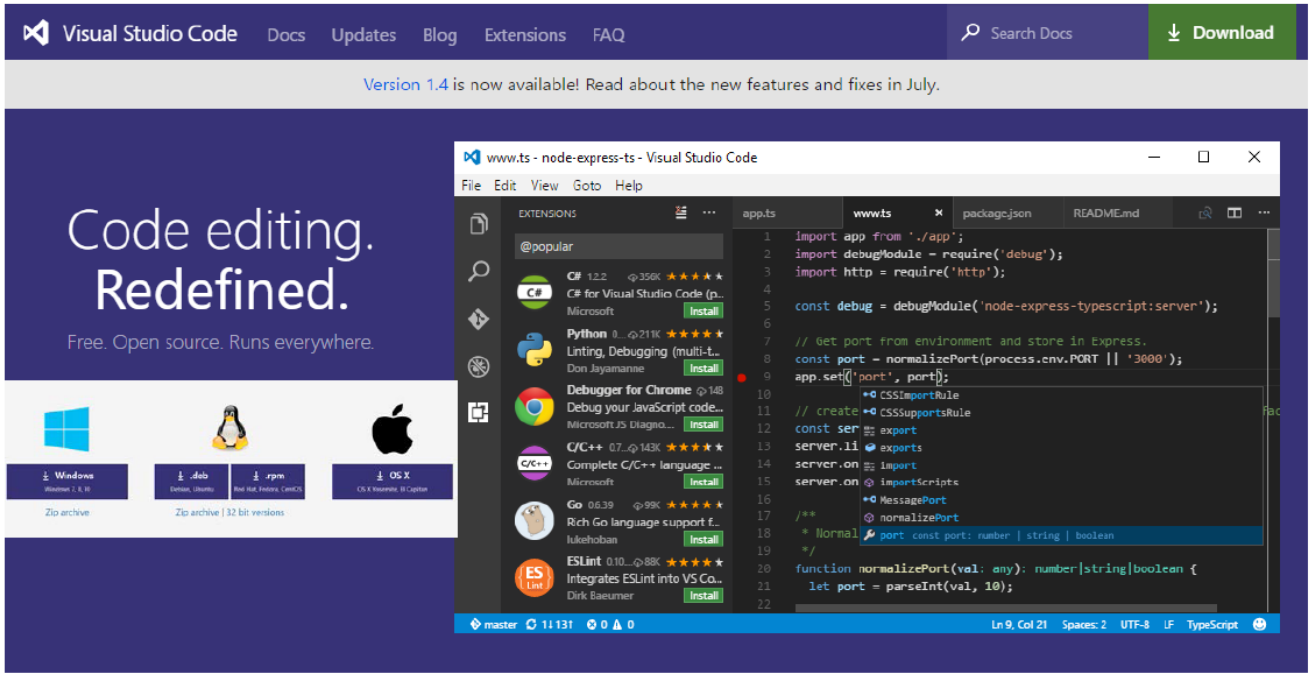
- TypeScript type definition files (*.d.ts)

Class-based object-orientation

TypeScript Editors



Visual Studio Code



https://code.visualstudio.com/

Setting up Our Environment

npm

Set up the Angular 2 application

npm

Node Package Manager

Command line utility

Installs libraries, packages, and applications

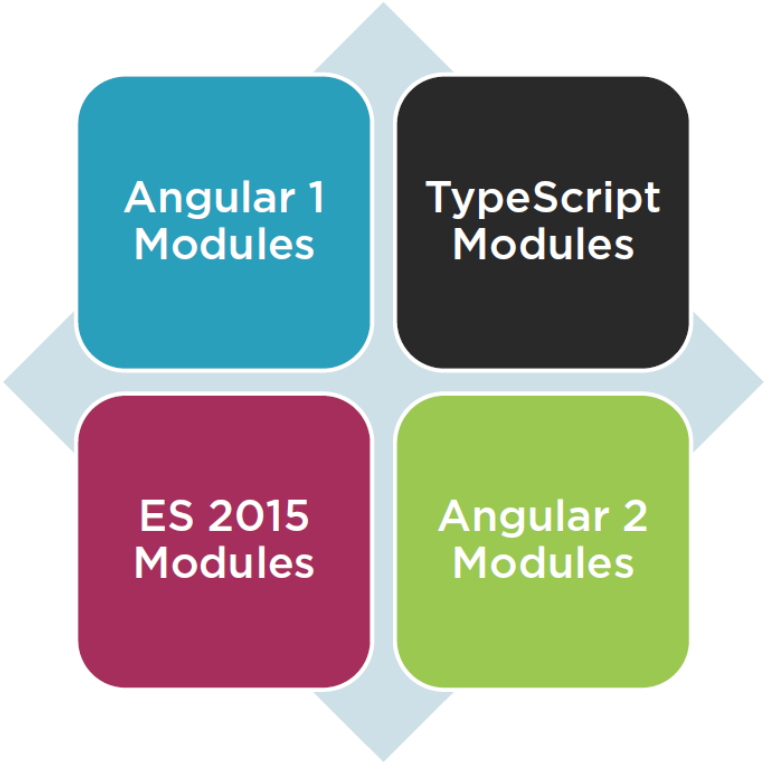
https://www.npmjs.com/

About Modules

Modules

Namespaces

Code Organization



ES 2015 Modules

Export

product.ts

```
export class Product{
}
```

product.js

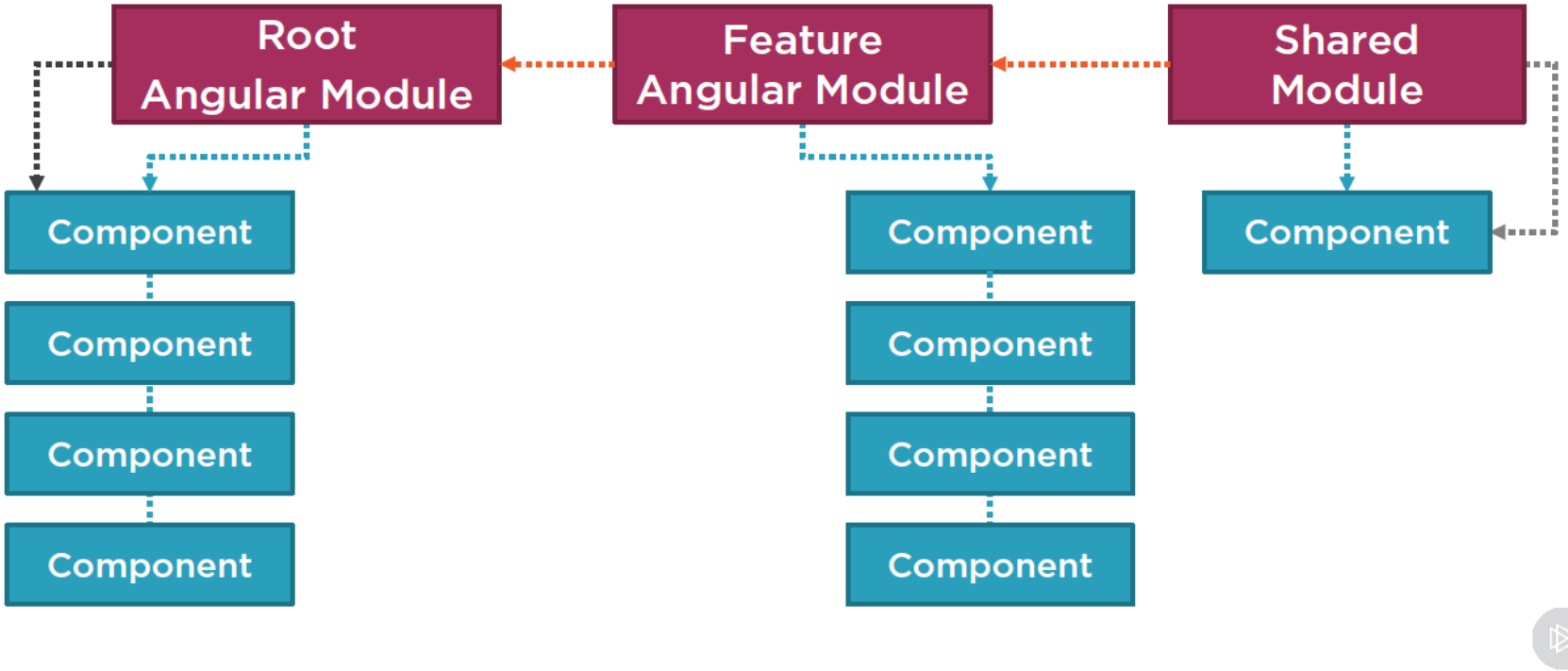
```
function Product() {
}
```

Import

product-list.ts

```
import { Product } from
'./product'
```

Angular Modules



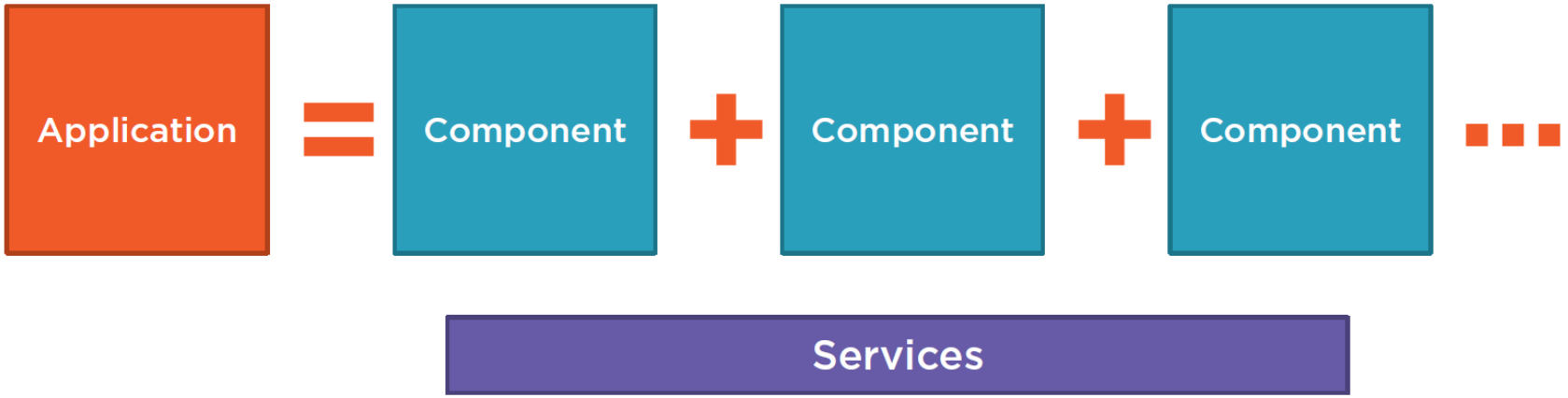
Modules

ES Modules	Angular Modules
Code files that import or export something	Code files that organize the application into cohesive blocks of functionality
Organize our code files	Organize our application
Modularize our code	Modularize our application
Promote code reuse	Promote application boundaries

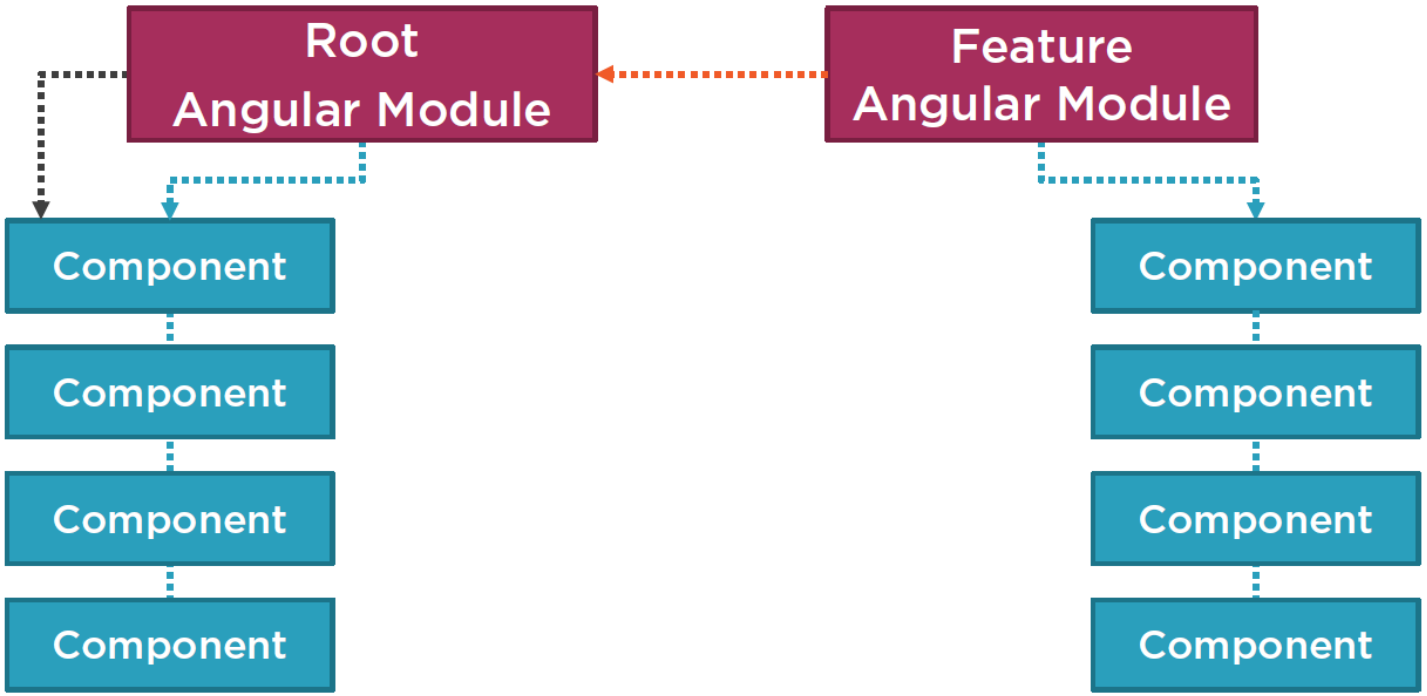
Angular Concepts

- Introduction to Angular Application
- Introduction to Angular Components
- Introduction to Angular Modules
- Templates, Interpolation, Directives
- Databinding, Pipes, NG Style, NG Class
- Nested Components
- Services and Dependency Injection
- Retrieving data using HTTP
- Navigation and Routing,
- Forms and Validations,
- Angular Guards

Anatomy of an Angular 2 Application



Angular Modules



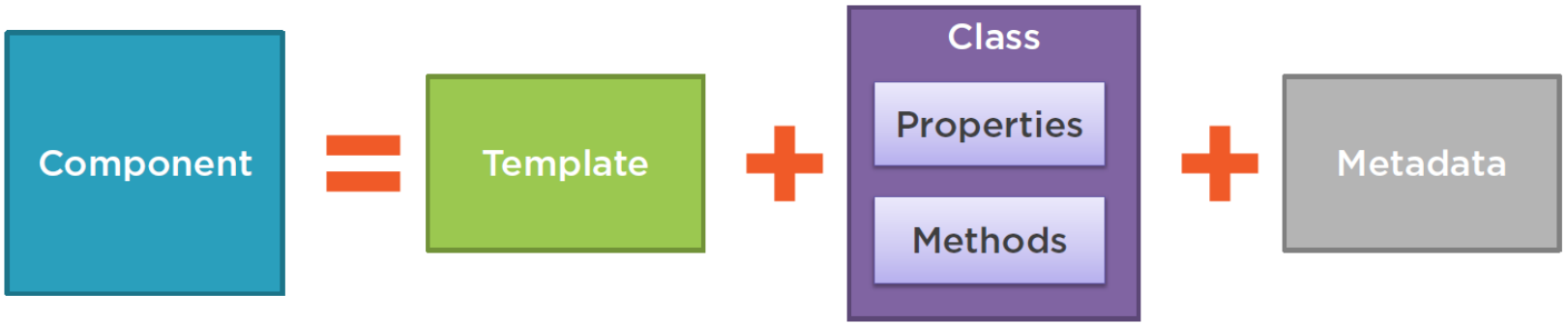
Setting up an Angular 2 Application

1. Create an application folder
2. Add package definition and configuration files
3. Install the packages
4. Create the app's Angular Module
5. Create the main.ts file
6. Create the host Web page (index.html)

Module Overview

- What is Component
- Creating the Component Class
- Defining Metadata With Decorator
- Importing what we need
- Bootstrapping our app component.

What Is a Component?



- View layout
- Created with HTML
- Includes binding and directives

- Code supporting the view
- Created with TypeScript
- Properties: data
- Methods: logic

- Extra data for Angular
- Defined with a decorator

Component

app.component.ts

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

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

Import

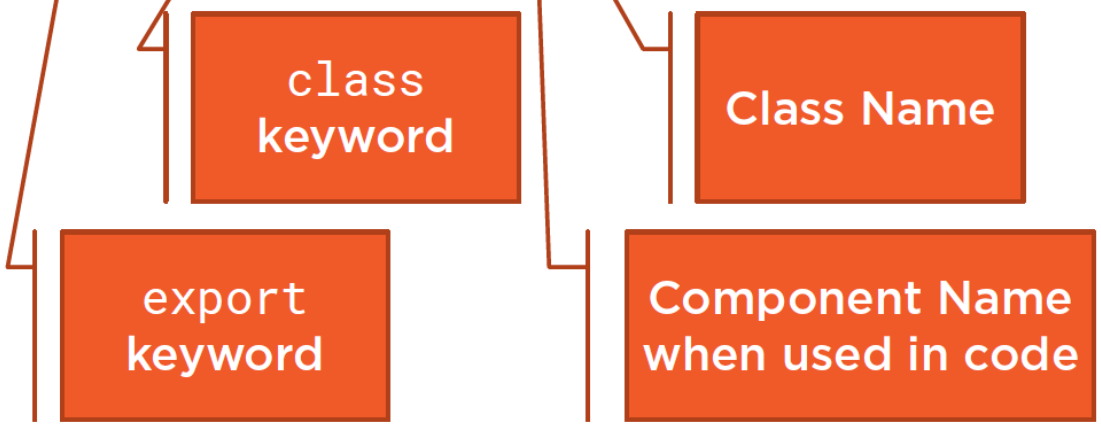
Metadata & Template

Class

Creating the Component Class

app.component.ts

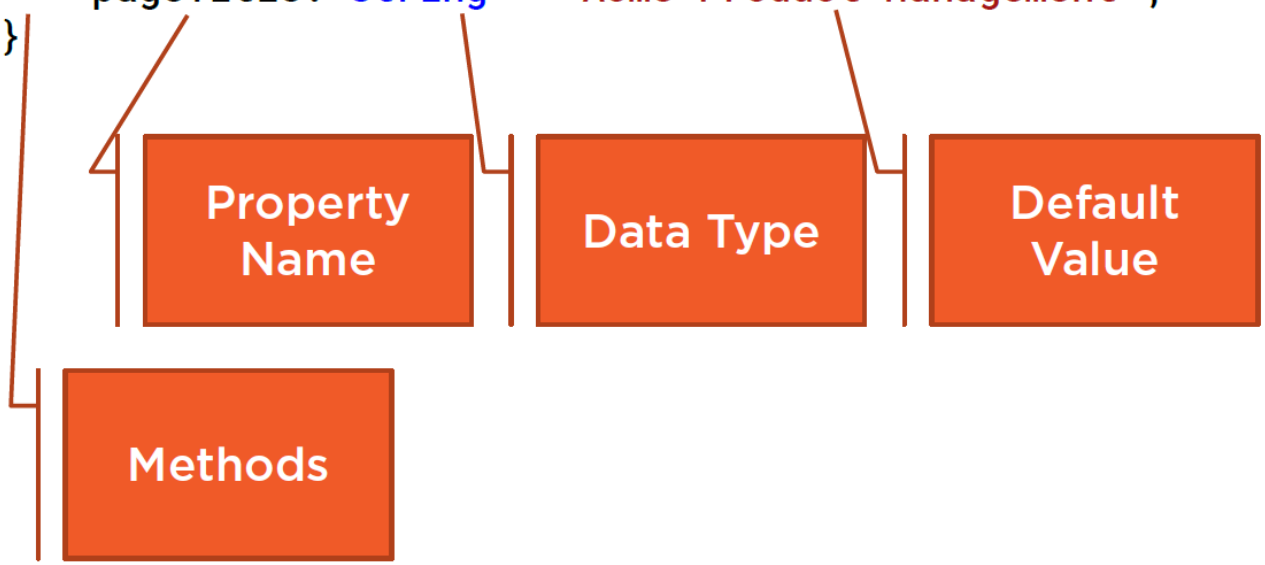
```
export class AppComponent {
    pageTitle: string = 'Acme Product Management';
}
```



Creating the Component Class

app.component.ts

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



Defining the Metadata

app.component.ts

```
@Component({
    selector: 'pm-app',
    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 @.

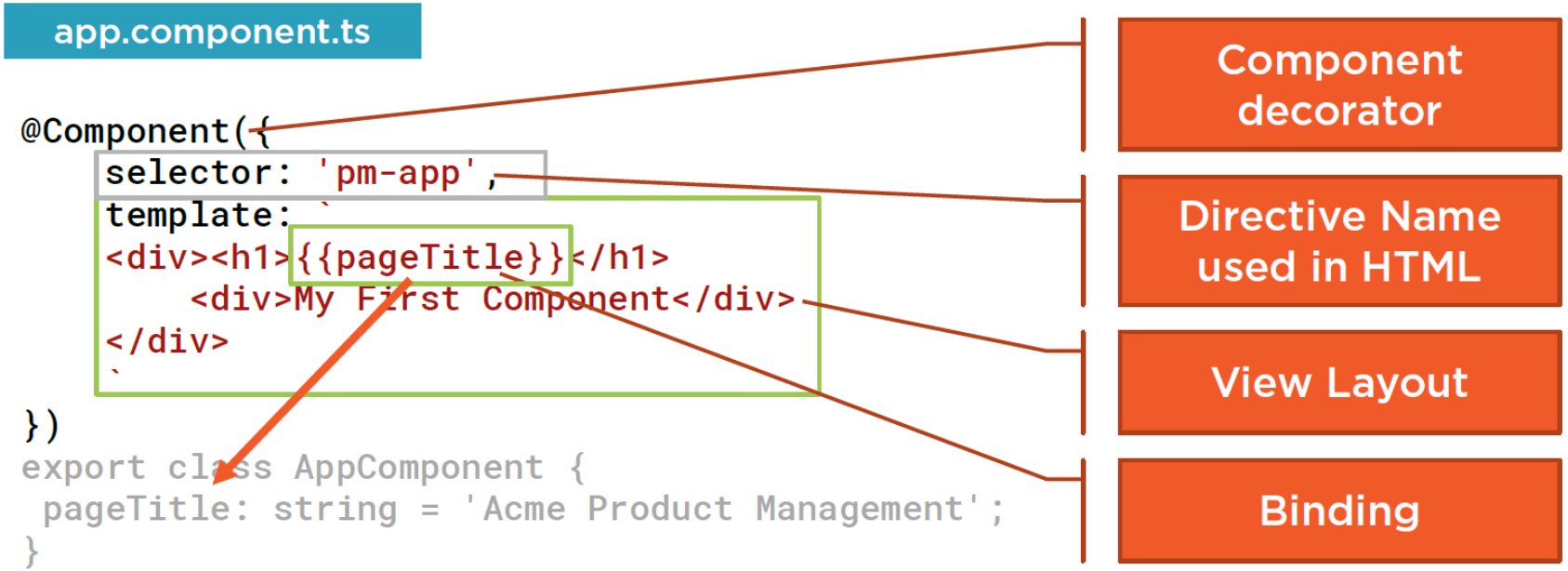
Angular provides built-in decorators.

```
@Component()
```

Defining the Metadata

app.component.ts

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



Importing What We Need

Before we use an external function or class, we define where to find it

`import` statement

`import` allows us to use exported members from external ES modules

Import from a third-party library, our own ES modules, or from Angular

Angular Is Modular

https://www.npmjs.com/~angular

Importing What We Need

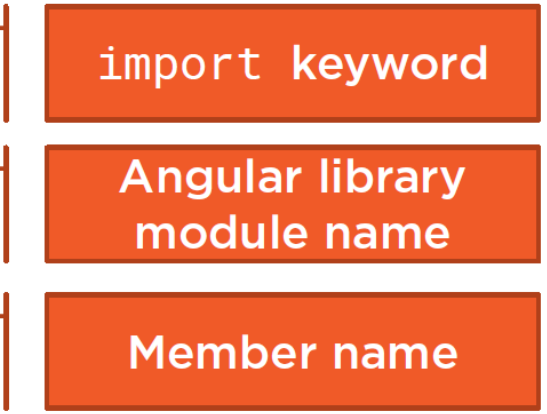
app.component.ts

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

Importing What We Need

app.component.ts

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

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



Completed Component

app.component.ts

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

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

Demo

Creating the App Component

Bootstrapping Our App Component

Load the root component (bootstrapping)

Host the application

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-app>Loading App ...</pm-app>
</body>
```

app.component.ts

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

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

Angular Application Startup

index.html

```
System.import('app')...;
```

```
<body>
  <pm-app>Loading App ...
  </pm-app>
</body>
```

Systemjs.config.js

```
packages: {
 app: {
  main: './main.js',
  defaultExtension: 'js'
 },
...
```

main.ts

```
import { platformBrowserDynamic }
 from '@angular/platform-browser-dynamic';
import { AppModule }
 from './app.module';

platformBrowserDynamic().
 bootstrapModule(AppModule);
```

app.component.ts

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

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 { }
```

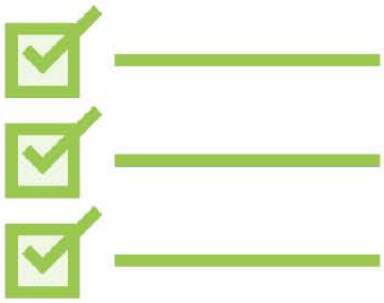
Component Checklist

Class -> Code

Decorator -> Metadata

Import what we need

Component Checklist: Class



Clear name

- Use PascalCasing
- Append "Component" to the name

`export` **keyword**

Data in properties

- Appropriate data type
- Appropriate default value
- camelCase with first letter lowercase

Logic in methods

- camelCase with 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 Checklist: Import

Defines where to find the members that this component needs

`import` **keyword**

Member name

- Correct spelling/casing

Module path

- Enclose in quotes
- Correct spelling/casing

Something's Wrong! Checklist

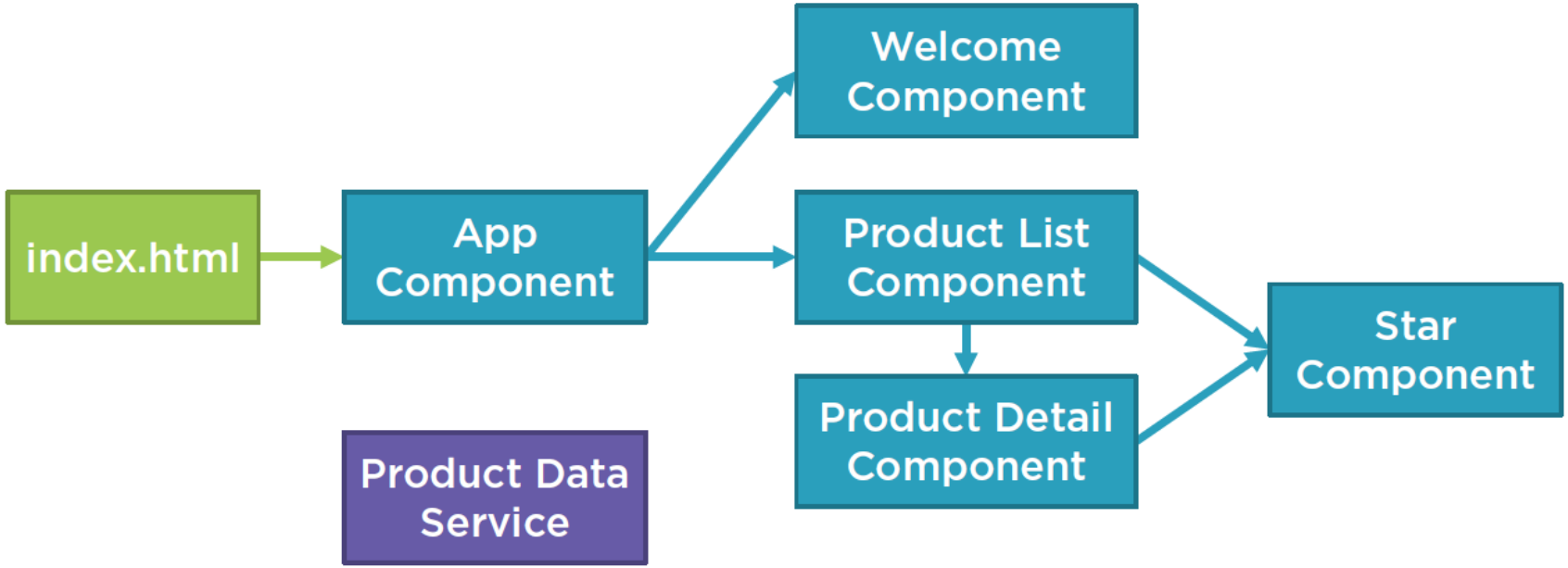
F12 is your friend

Recheck your code

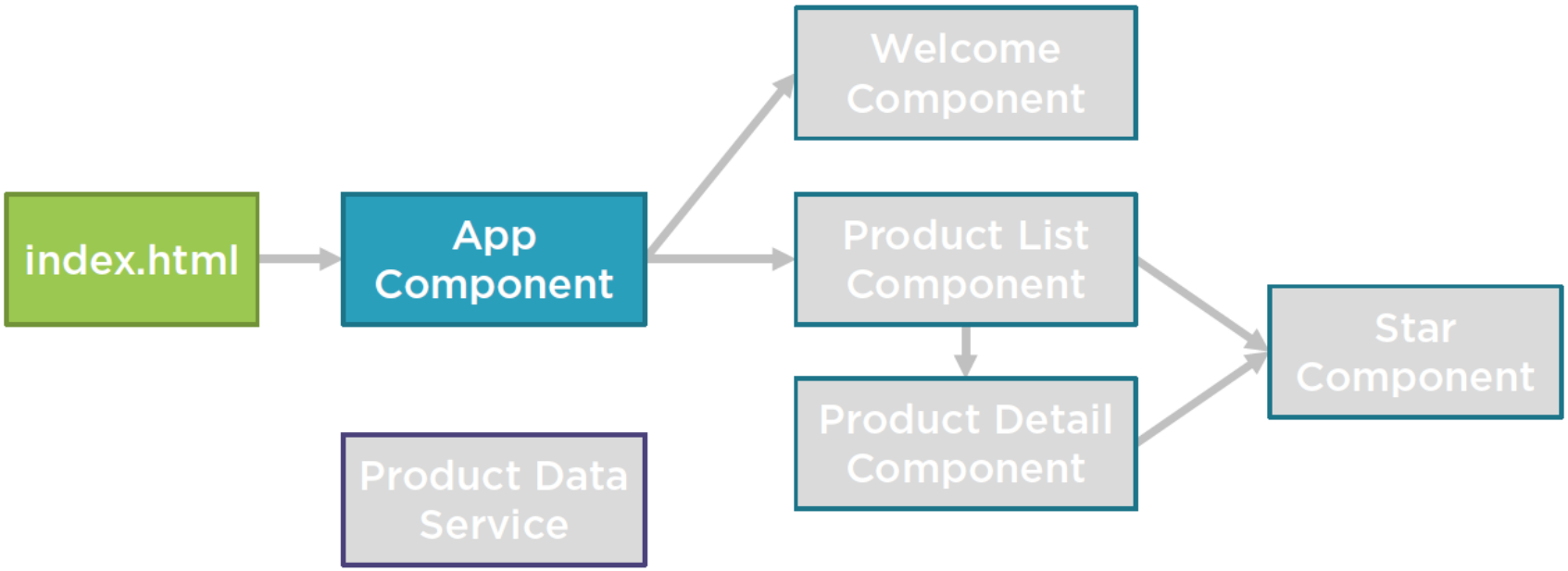
- HTML
  - Close tags
  - Angular directives are case sensitive
- TypeScript
  - Close braces
  - TypeScript is case sensitive

Lets go to Sample Application

Sample Application Architecture



Application Architecture



Typescript Basics

Typescript Concepts

- Basic Types : Boolean, Number, String, Array, Tuple, Void, Null and Undefined,
- Type Assertions, Variable Declarations, Var, Let, Const, Optional Param
- Interfaces, Classes, Inheritances, functions, this, Generics, enum,
- For—of , for –in
- Modules,
- Export interface, objects, class, Export with alias, export *
- Import , Import alias , import *

TypeScript Features

- Static Typing
- Interfaces
- Class Properties
- Public/Private Accessibility

```
let name : string
let age : number
let birthDate : date
```

Static Typing

```
interface ICat {
 name:string
 age:number
}
```

TypeScript Interfaces

```
interface ICat {
 name:string
 age:number
}

let fluffy:ICat
```

```
fluffy =
  {
    name: 'Fluffy',
    age: 'seven'
  }
```

TypeScript Interfaces

```
interface ICat {
 name:string
 age:number
}

let fluffy:ICat
```

```
fluffy =
  {
    name: 'Fluffy'
  }
```

TypeScript Interfaces

```
interface ICat {
 name:string
 age?:number
}

let fluffy:ICat
```

```
fluffy =
  {
    name: 'Fluffy'
  }
```

TypeScript Interfaces

```
class Cat {
  name:string
  color:string
  constructor (name) {
    this.name = name;
  }
}
```

TypeScript Class Properties

```
class Cat {
  name:string
  color:string
  constructor (name) {
    this.name = name;
  }
  speak() { console.log('My name is: ' + this.name) }
}
```

Public and Private Accessibility

```
class Cat {
  name:string
  speak() { console.log('My name is: ' + this.name) }
}

let fluffy = new Cat()
console.log(fluffy.name)
fluffy.speak()
```

Public and Private Accessibility

```
class Cat {
  private name:string
  private speak() { console.log('My name is: ' + this.name) }
}

let fluffy = new Cat()
console.log(fluffy.name)
fluffy.speak()
```

Public and Private Accessibility

```
class Cat {
  private name:string
  private speak() { console.log('My name is: ' + this.name) }
}

let fluffy = new Cat()
console.log(fluffy.name) //compile-time error
fluffy.speak() // compile-time error
```

Public and Private Accessibility

convenient enumeration type thrown in to help things alor

Boolean

The most basic datatype is the simple true/false value, whi

```
let isDone: boolean = false;
```

Number

As in JavaScript, all numbers in TypeScript are floating poi
addition to hexadecimal and decimal literals, TypeScript al
2015.

```
let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
```

String

Another fundamental part of creating programs in JavaScr
in other languages, we use the type `string` to refer to th
double quotes (`"`) or single quotes (`'`) to surround strin

```
let color: string = "blue";
color = 'red';
```

https://www.typescriptlang.org/docs/handbook/basic-types.html

```
class Cat {
  private name:string
  private color:string
  constructor(name, color) {
    this.name = name
    this.color = color
  }
}
```

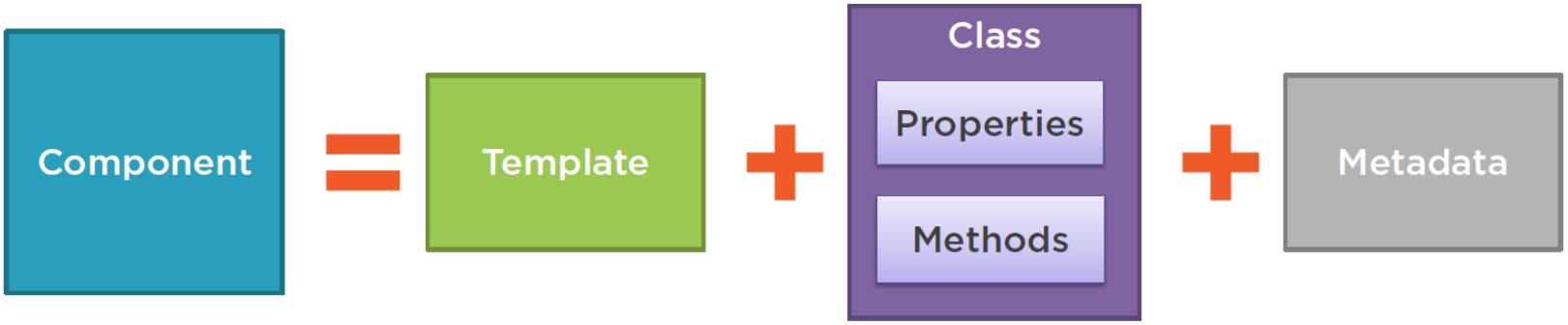
```
class Cat {
  constructor(private name, private color) {

  }
}

let fluffy = new Cat('Fluffy', 'White')
```

Public and Private Accessibility

Templates, Interpolation, and Directives

- https://angular.io/docs/ts/latest/guide/template-syntax.html

Component



Module Overview

Building a Template

Using a Component as a Directive

Binding with Interpolation

Adding Logic with Directives

Component

app.component.ts

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

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

Defining a Template in a Component

Inline Template

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

Inline Template

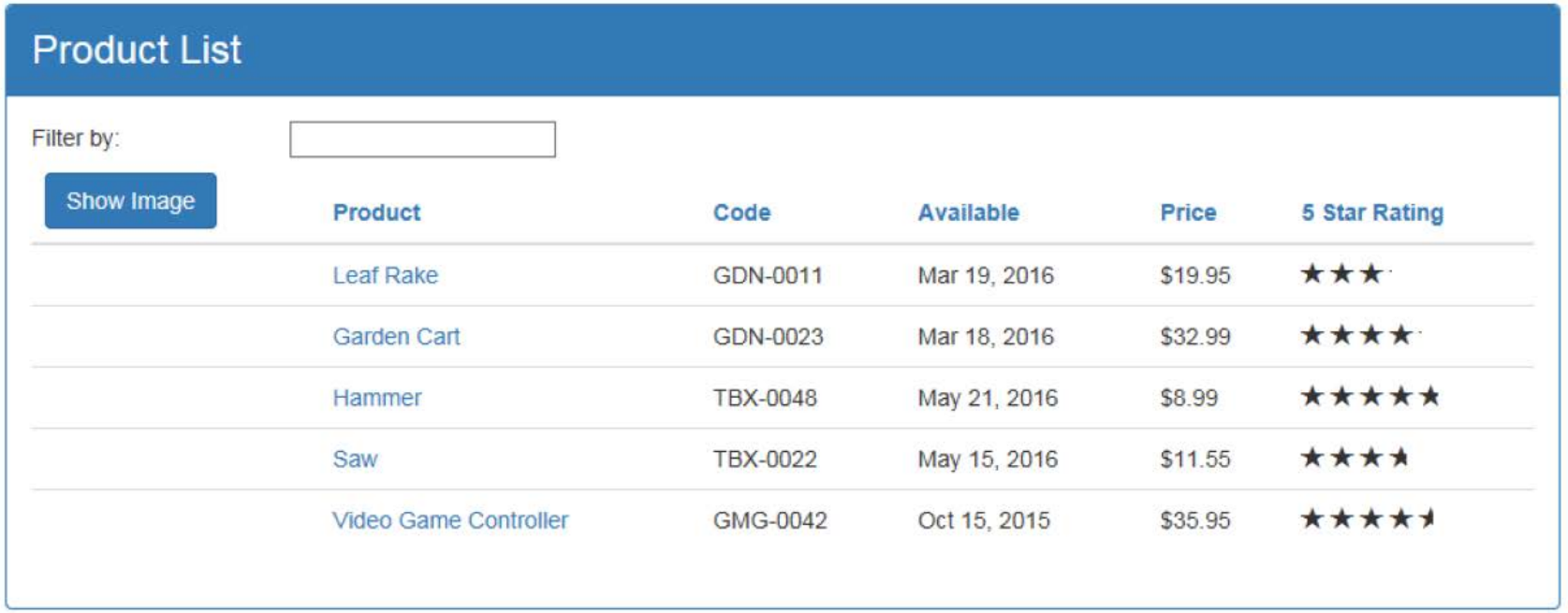
```
template: `
<div>
 <h1>{{pageTitle}}</h1>
 <div>
     My First Component
 </div>
</div>
`
```

ES 2015
Back Ticks

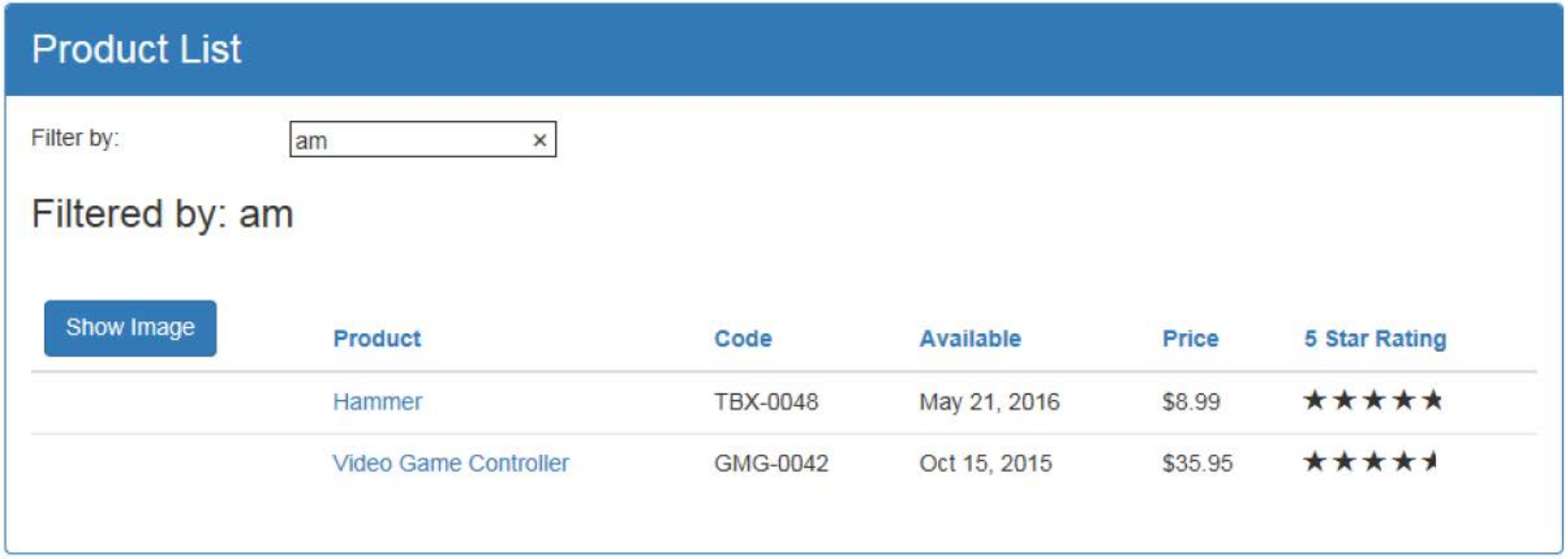
Linked Template

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

Product List View



Product List View



Product List View

Product List

Filter by: am

Filtered by: am

Hide Image	Product	Code	Available	Price	5 Star Rating
	Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
	Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★½

Product List View

Product List

Filter by: am

Filtered by: am

Hide Image	Product	Code	Available	Price	5 Star Rating
	Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
	Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★½

Building the Component

product-list.component.ts

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

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

Using a Component as a Directive

app.component.ts

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

product-list.component.ts

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

Using a Component as a Directive

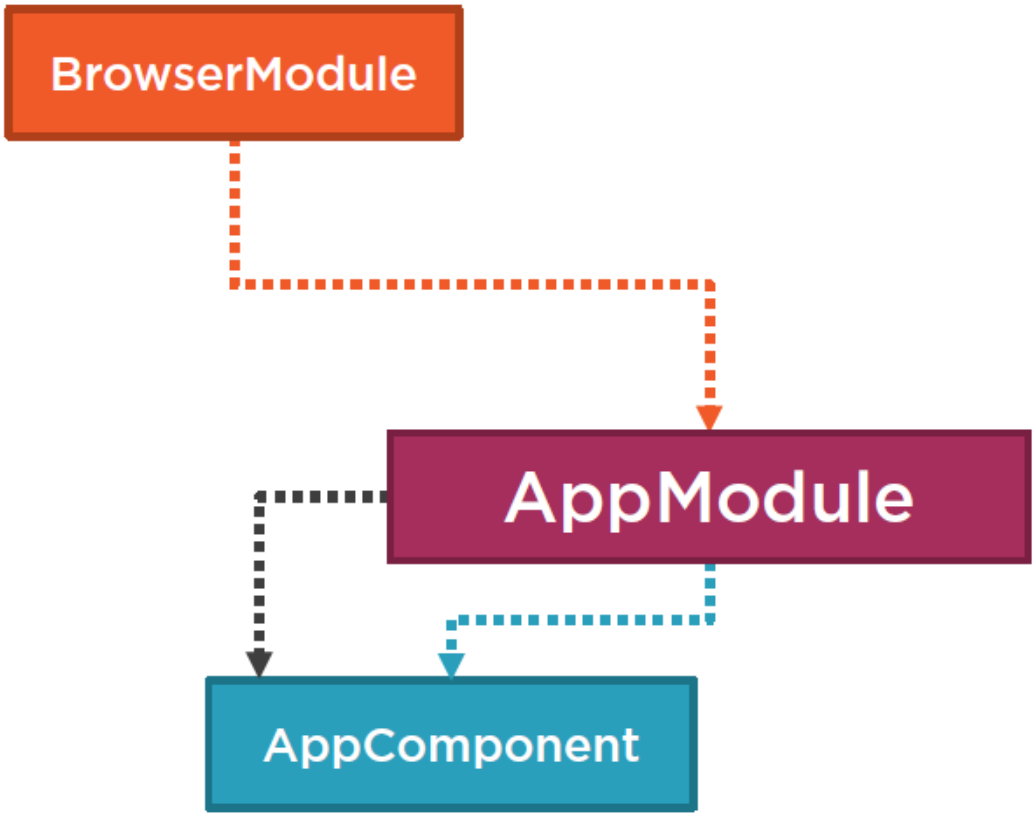
app.component.ts

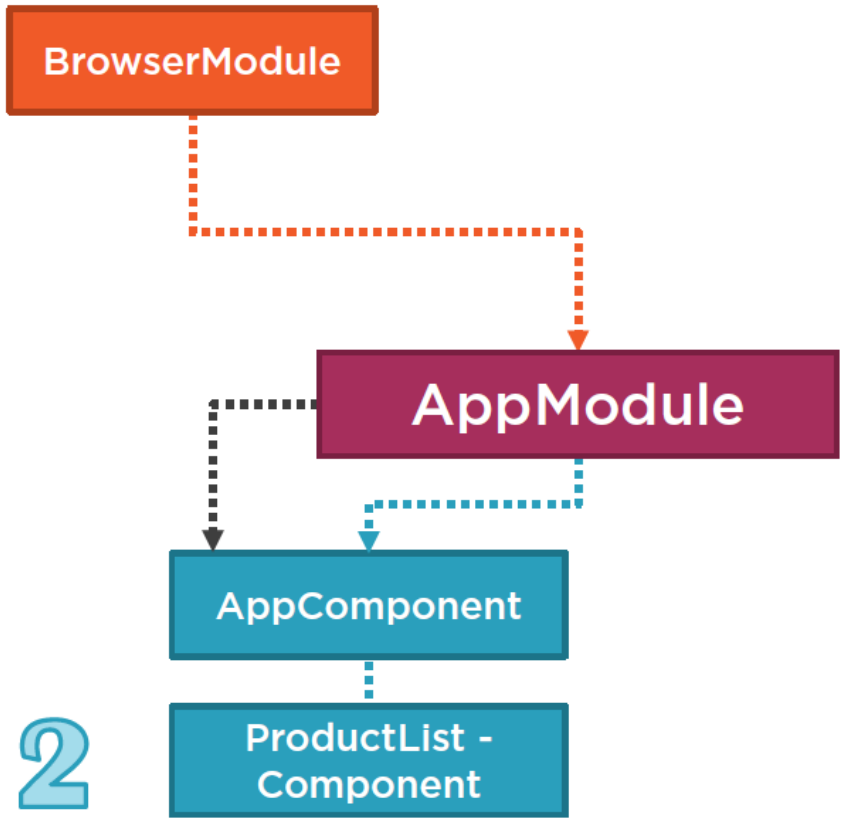
```
@Component({
 selector: 'pm-app',
 template: `
   <div><h1>{{pageTitle}}</h1>
     <pm-products></pm-products>
   </div>`
})
export class AppComponent { }
```

1

product-list.component.ts

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


BrowserModule
AppModule
AppComponent


BrowserModule
AppModule
AppComponent
2
ProductList -
Component

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}}

{{'Title: ' + getTitle()}}

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

Class

```
export class AppComponent {
 pageTitle: string =
     'Acme Product Management';
}getTitle(): string {...};
}
```

Expression Restrictions

- Assignments (=, +=, ++, etc)
- new Keyword
- Expression Chaining With ;
- Global Namespace

Expression Recommendations

- No Side-Effects
- Fast
- Simple
- Idempotent

```
...
@Component({
  template: `
    <h2>{{user.name}}</h2>
    <img [src]="user.imageUrl"/>
    <button (click)="doSomething()"></button>`
})
export class ProfileComponent {
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}

  doSomething() {
  }
}
```

Interpolation, Expressions, Events, and Statements

Statement Restrictions

- Assignments Except = (+=, ++, etc)
- new Keyword
- Expression Chaining With ;
- Global Namespace

Statement Restrictions

Assignments Except = (+=, ++, etc)

new Keyword

~~Expression Chaining With ;~~

Global Namespace

Statement Restrictions

Assignments Except = (+=, ++, etc)

new Keyword

Global Namespace

Statement Restrictions

~~No Side-Effects~~

~~Fast~~

Simple

~~Idempotent~~

Directive

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

- Custom
- Built-In

Custom Directives

app.component.ts

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

product-list.component.ts

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

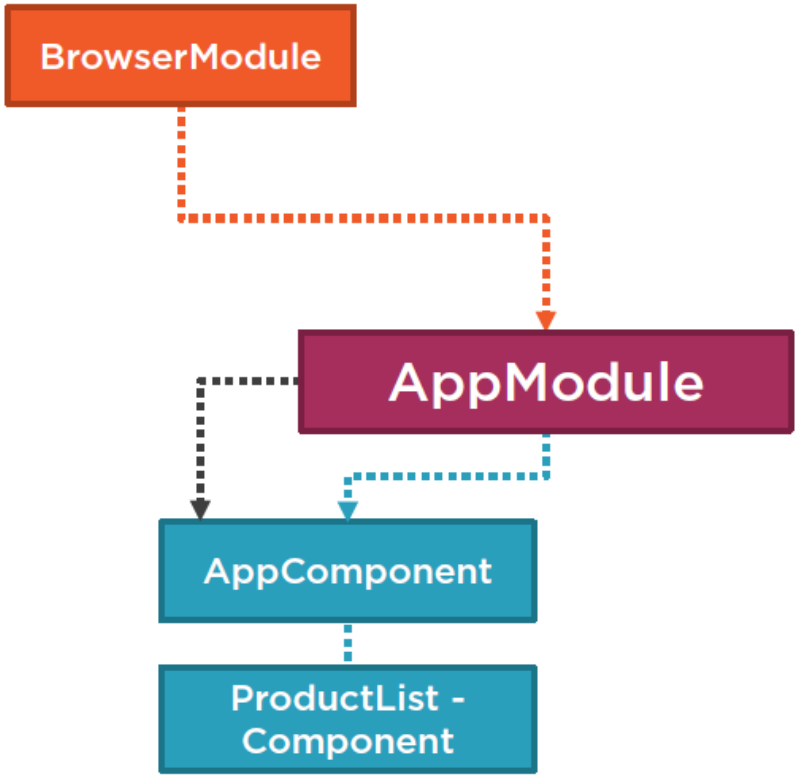
Angular Built-in Directives

Structural Directives

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

*ngIf Built-In Directive

```
<div class='table-responsive'>
 <table class='table' *ngIf='products && products.length'>
  <thead> ...
  </thead>
  <tbody> ...
  </tbody>
 </table>
</div>
```


BrowserModule
AppModule
AppComponent
ProductList -
Component

*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>
```

Template input variable

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);
}
```

Checklist: Template

Inline template

- For short templates
- Specify the `template` property
- Use the ES 2015 back ticks for multiple lines
- Watch syntax

Linked template

- For longer templates
- Specify the `templateUrl` property
- Define the path to the HTML file

Checklist: Component as a Directive

product-list.component.ts

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

app.component.ts

```
@Component({
 selector: 'pm-app',
 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

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

Checklist: Interpolation

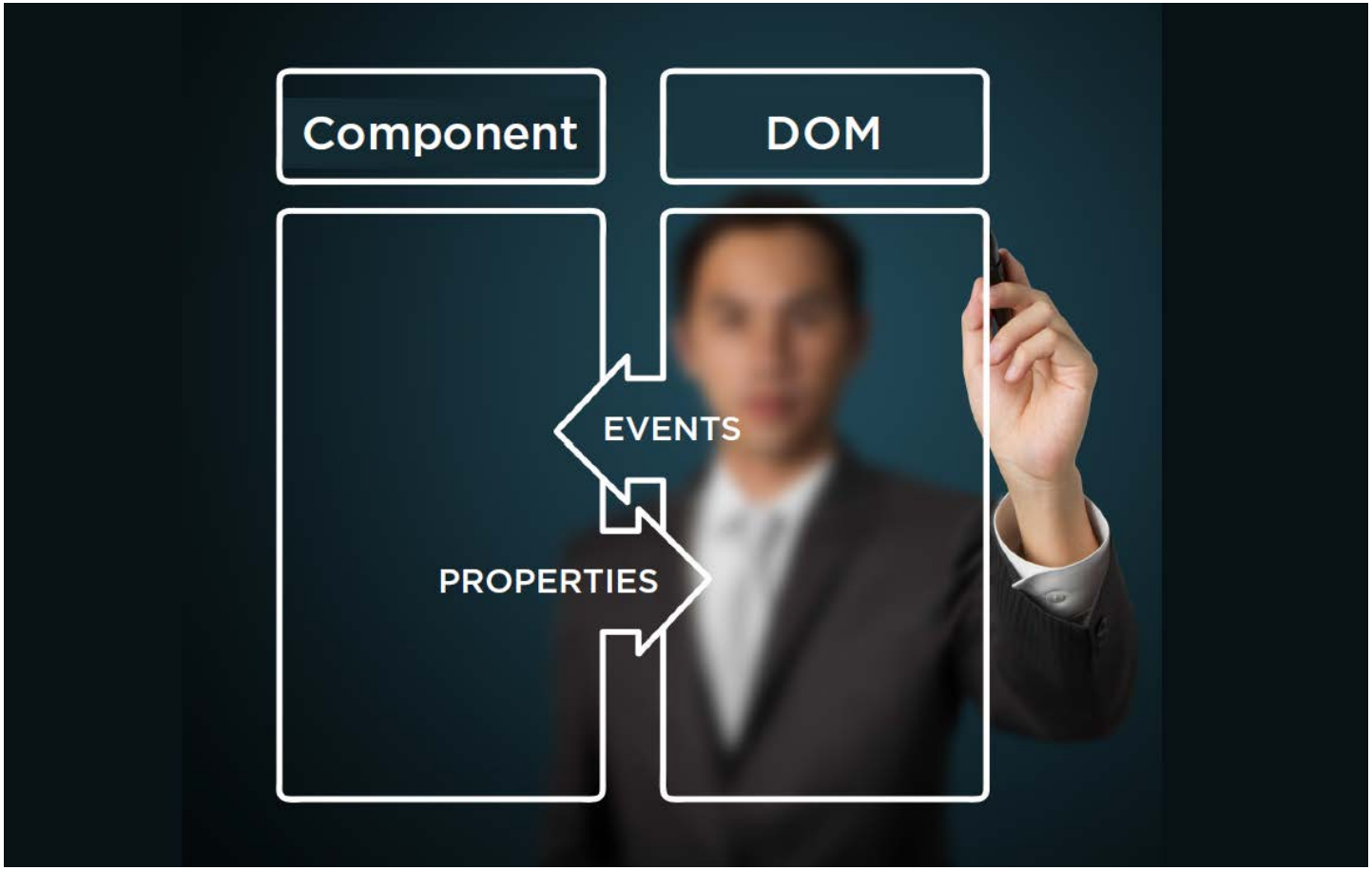
One way binding

- From component class property to an element property.

Defined with double curly braces

- Contains a template expression
- No quote needed

Data Binding & Pipes


Component
DOM
EVENTS
PROPERTIES

Module Overview

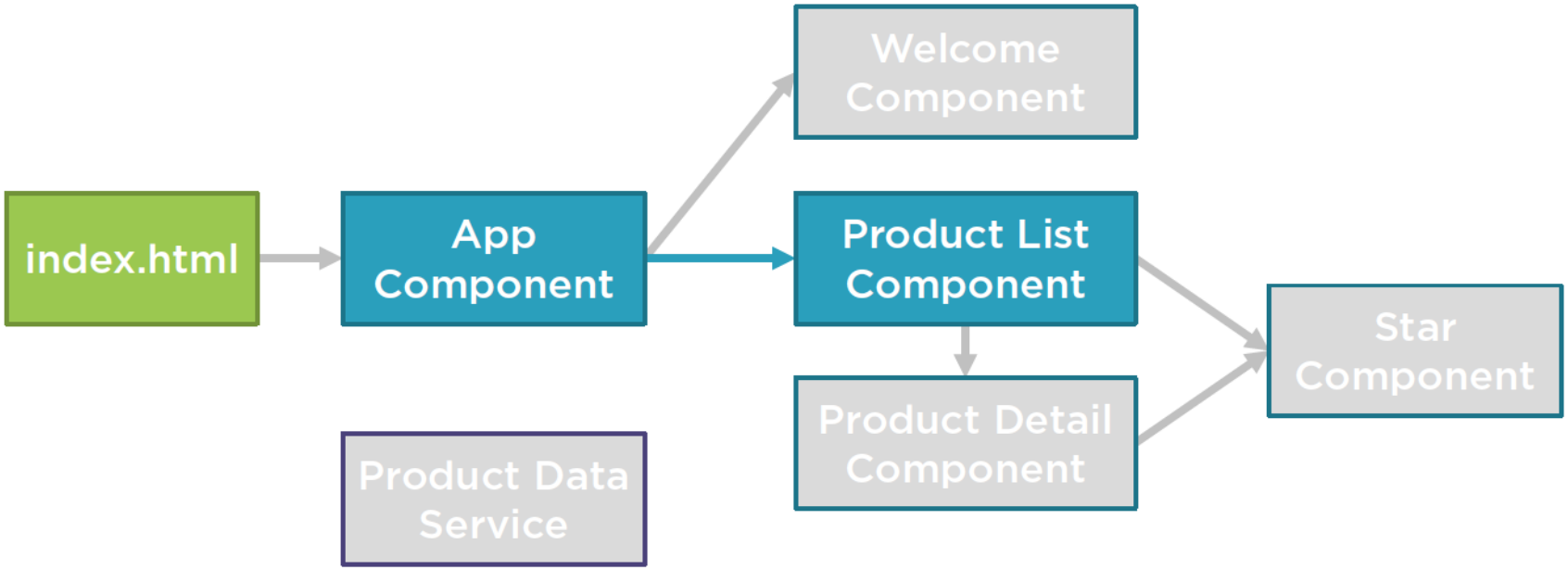
Property Binding

Handling Events with Event Binding

Handling Input with Two-way Binding

Transforming Data with Pipes

Application Architecture



Property Binding

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

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

Event Binding

Template

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

Class

```
export class ListComponent {
```

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

( )
Target Event

' '
Template Statement

Two-way Binding

Template

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

Class

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

Two-way Binding

Template

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

Class

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

Two-way Binding

Template

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

Class

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

Two-way Binding

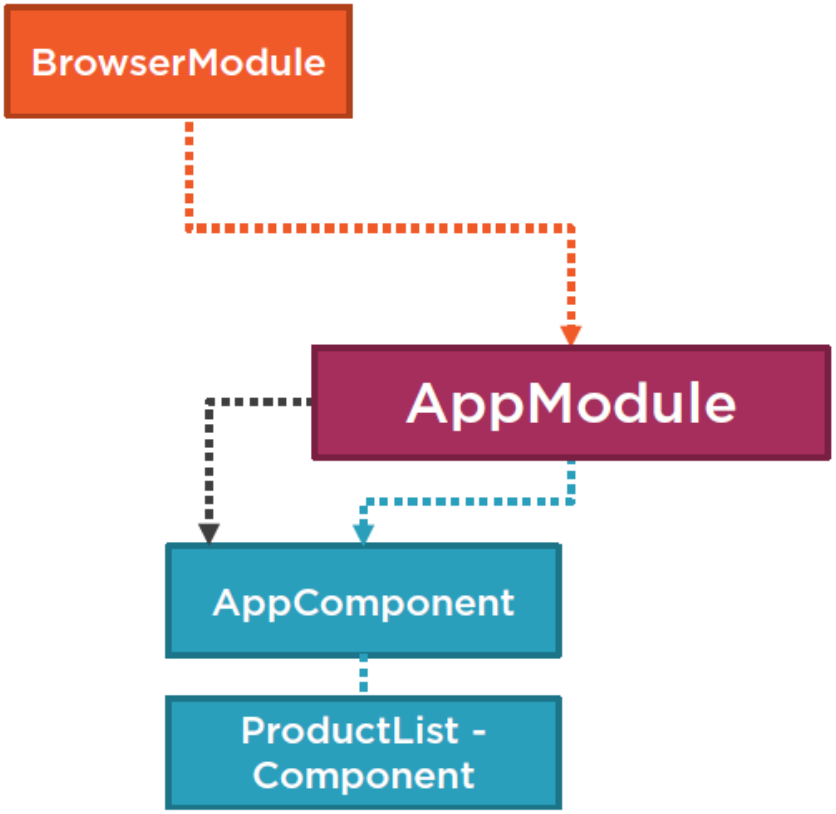
Template

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

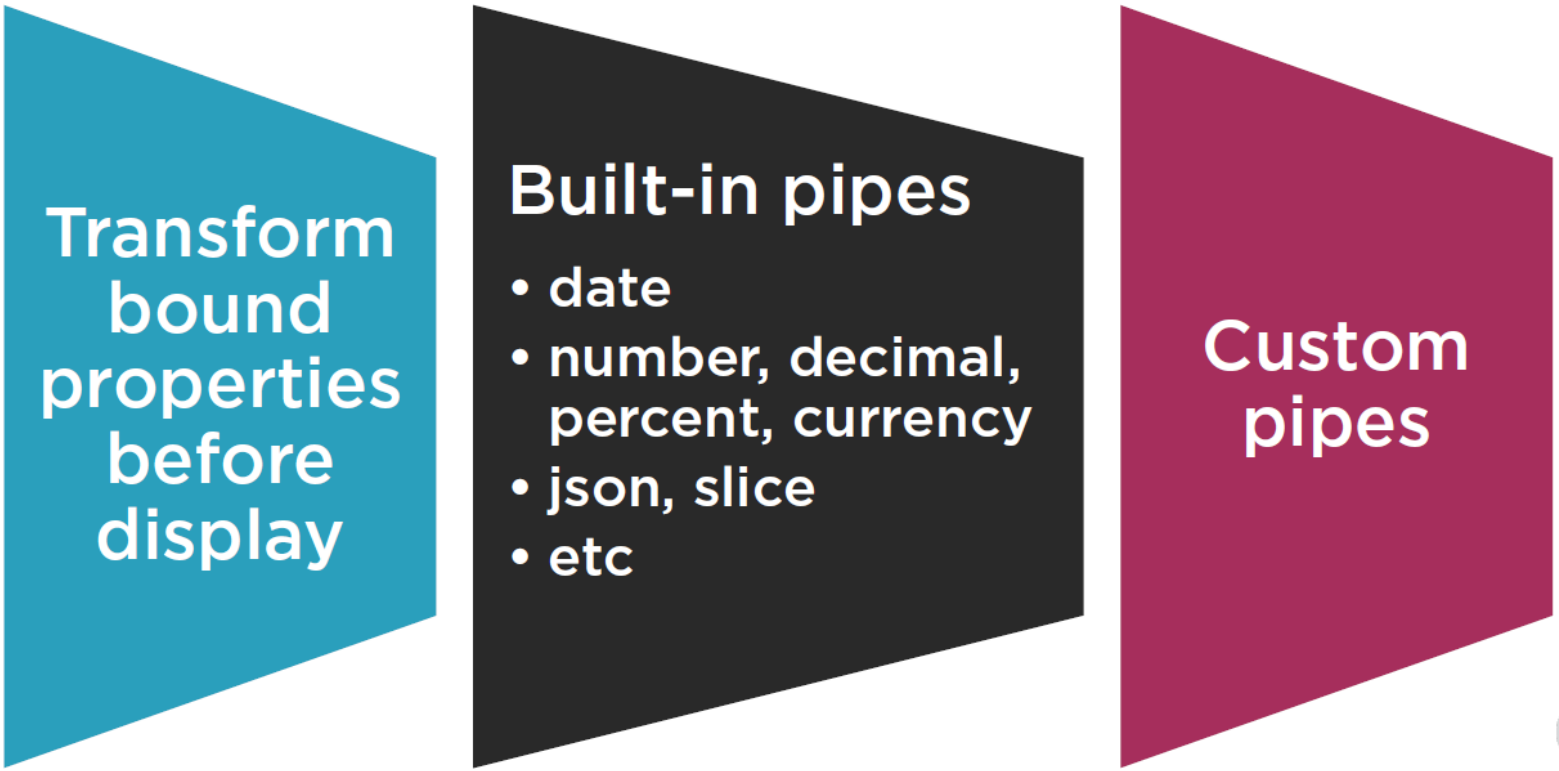
Class

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

[()]
Banana in a Box


BrowserModule
AppModule
AppComponent
ProductList -
Component

Transforming Data with Pipes



Built In Pipes

- Mostly used pipes : Date, Currency, Decimal, Lowercase, uppercase, PercentPipe, Async Pipe, Json
- List of Pipes :
- https://angular.io/docs/ts/latest/api/#!?query=pipe

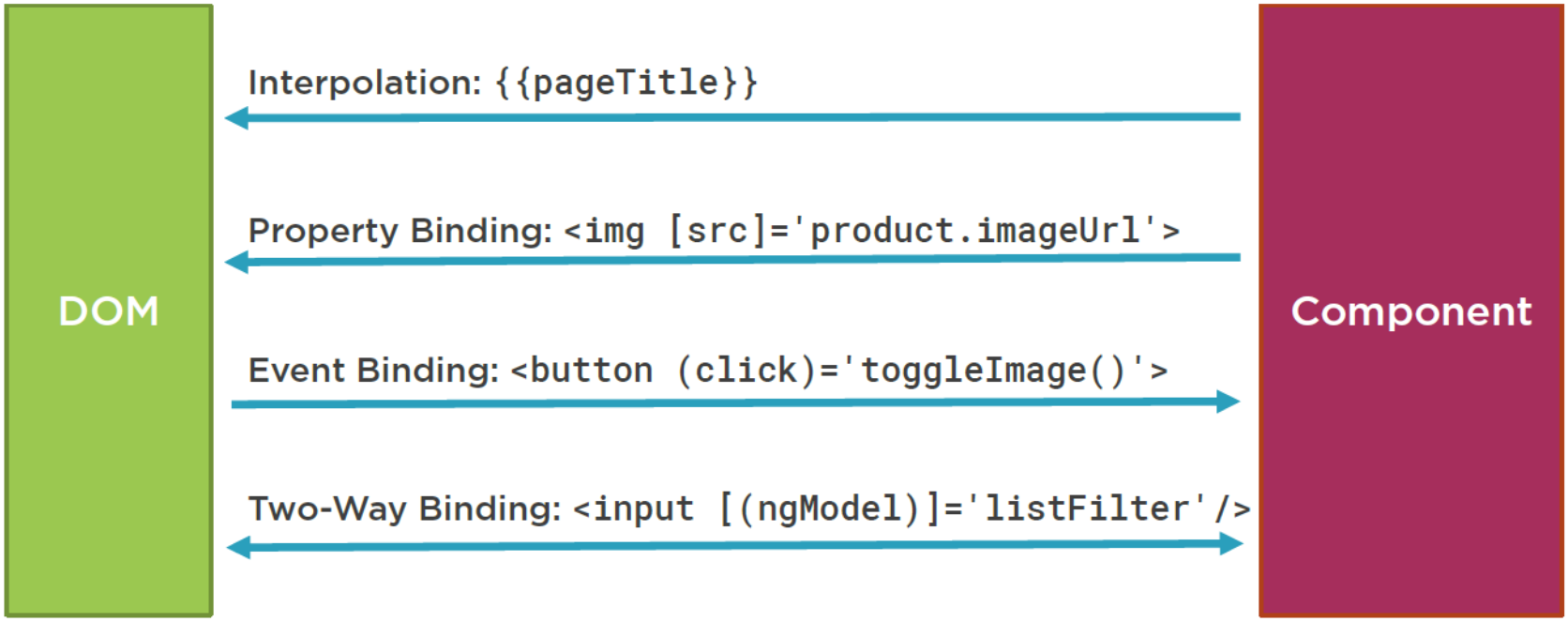
- Ref : https://angular.io/docs/ts/latest/guide/pipes.html

- Example : https://embed.plnkr.co/?show=preview

Pipe Examples

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

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

{{ product.price | currency | lowercase }}

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

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 { }
```

Checklist: Pipes

Pipe character |

Pipe name

Pipe parameters

- Separated with colons

Example

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

More on Components

Module Overview

Defining an Interface

Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe

Defining Relative Paths with Module Id

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;
    }
```

Interface

A specification identifying a related set of properties and methods.

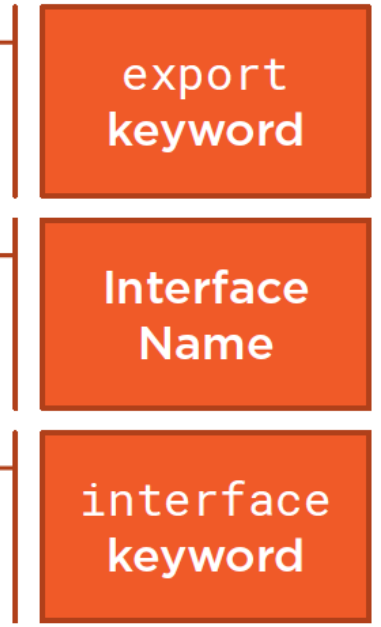
A class commits to supporting the specification by implementing the interface.

Use the interface as a data type.

Development time only!

Interface Is a Specification

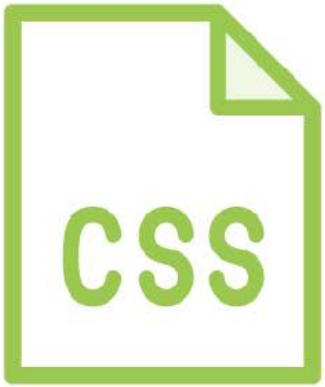
```
export interface IProduct {
    productId: number;
    productName: string;
    productCode: string;
    releaseDate: Date;
    price: number;
    description: string;
    starRating: number;
    imageUrl: string;
    calculateDiscount(percent: number): number;
}
```



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!

NGClass

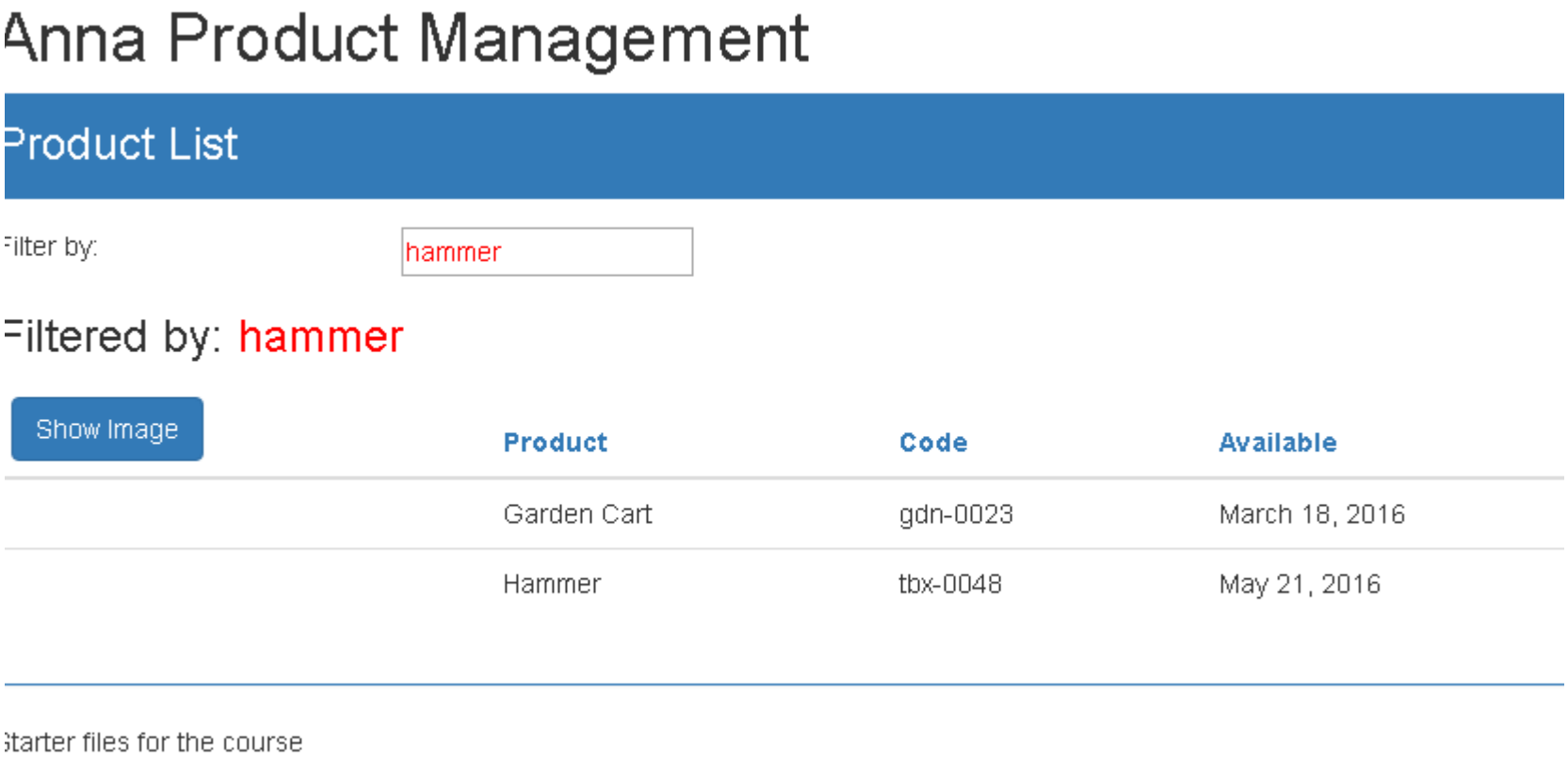
```
<div [ngClass]="{green: event?.time === '8:00 am', bold: event?
  .time === '8:00 am'}" [ngSwitch]="event?.time">
  Time: {{event?.time}}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

```
getStartTimeClass() {
  const isEarlyStart = this.event && this.event.time === '8:00 am'
  return {green: isEarlyStart, bold: isEarlyStart}
}
```

NG Style

```
<div [ngStyle]="{'color': event?.time === '8:00 am' ?
  '#003300' : '#bbb', 'font-weight': event?.time === '8:00 am'
  ? 'bold' : 'normal'}" [ngSwitch]="event?.time">
  Time: {{event?.time}}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

After NGClass applied



Encapsulating Component Styles

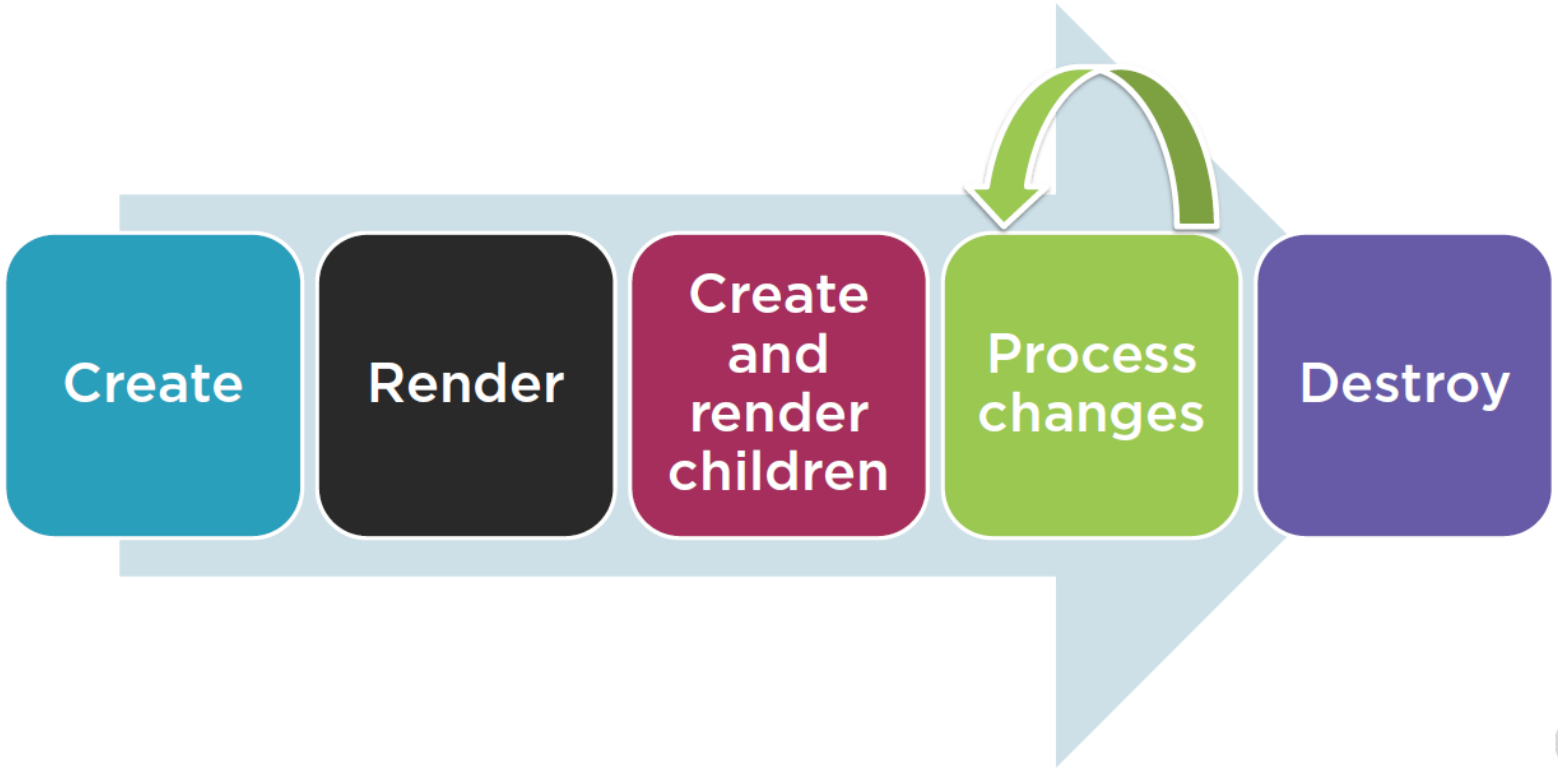
styles

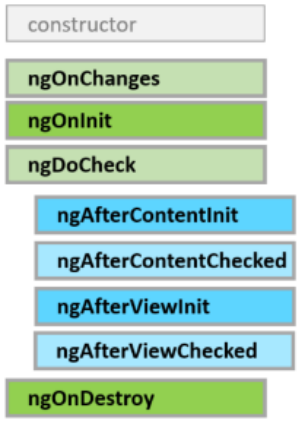
```
@Component({
    selector: 'pm-products',
    templateUrl: 'app/products/product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls

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

Component Lifecycle





A component has a lifecycle managed by Angular itself.

Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.

A directive has the same set of lifecycle hooks, minus the hooks that are specific to component content and views.

https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

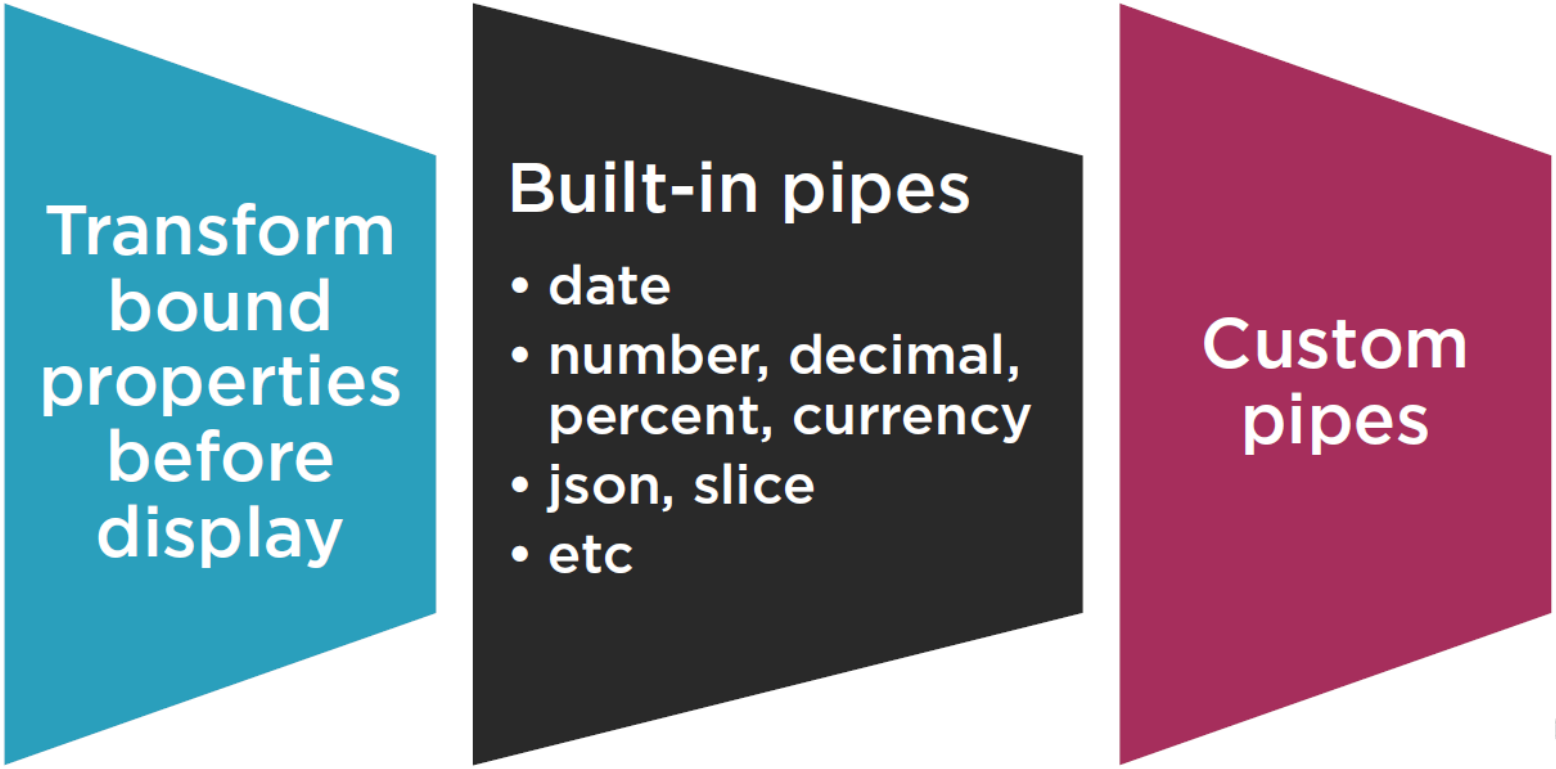
Component Lifecycle Hooks

OnInit: Perform component initialization, retrieve data

OnChanges: Perform action after change to input properties

OnDestroy: Perform cleanup

Transforming Data with Pipes



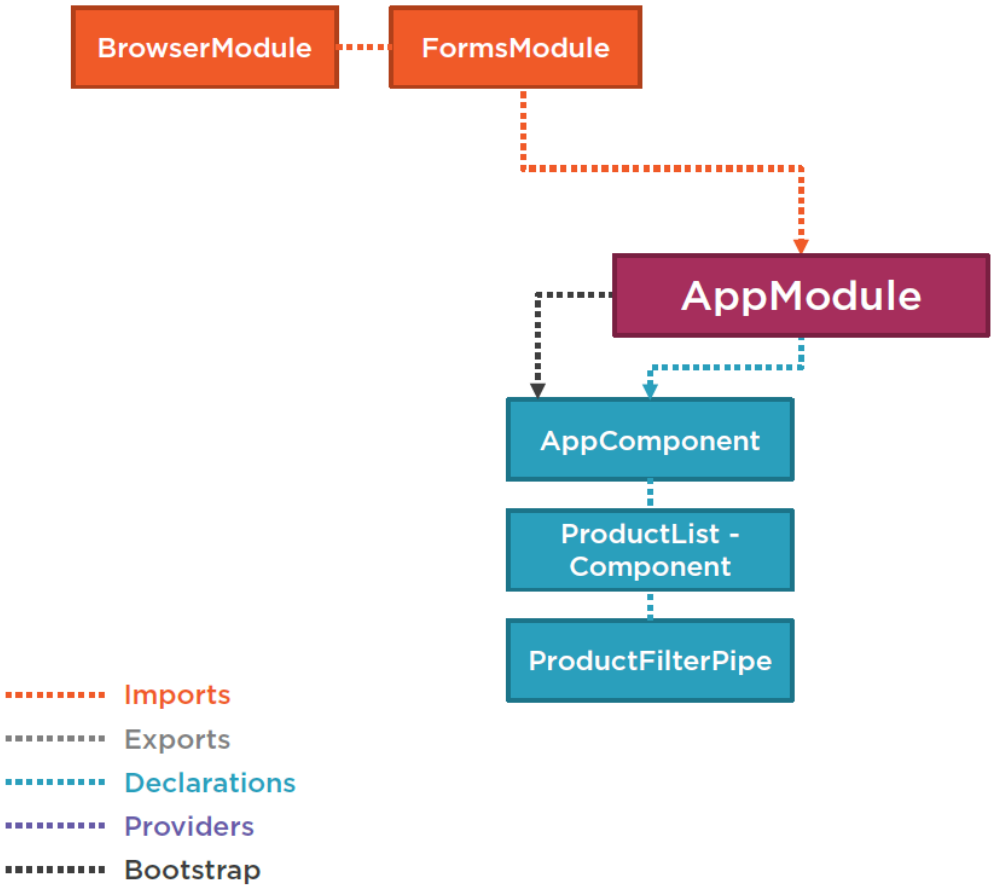
Building a Custom Pipe

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

@Pipe({
    name:'productFilter'
})
export class ProductFilterPipe
                    implements PipeTransform {

  transform(value: IProduct[],
            filterBy: string): IProduct[]{
  }
}
```

Using a Custom Pipe

Template

```
<tr *ngFor ='let product of products | productFilter: listFilter'>
```


BrowserModule
FormsModule
AppModule
AppComponent
ProductList - Component
ProductFilterPipe
Imports
Exports
Declarations
Providers
Bootstrap

Using a Custom Pipe

Template

```
<tr *ngFor ='let product of products | productFilter: listFilter'>
```

Module

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

Checklist: Interfaces

Defines custom types

Creating interfaces:

- `interface` keyword
- export it

Implementing interfaces:

- `implements` keyword & interface name
- Write code for each property & method

Checklist: Encapsulating Styles

`styles` **property**

- Specify an array of style strings

`styleUrls` **property**

- Specify an array of stylesheet paths

Checklist: Using Lifecycle Hooks

- Import the lifecycle hook interface
- Implement the lifecycle hook interface
- Write code for the hook method

Checklist: Building a Custom Pipe

Import Pipe and PipeTransform

Create a class that implements PipeTransform

- **`export`** the class

Write code for the Transform method

Decorate the class with the Pipe decorator

Checklist: Using a Custom Pipe

Import the custom pipe

Add the pipe to the declarations array of an Angular module

Any template associated with a component that is also declared in that Angular module can use that pipe

Use the Pipe in the template

- Pipe character
- Pipe name
- Pipe arguments (separated with colons)

Module Overview

Building a Nested Component

Using a Nested Component

Passing Data to a Nested Component Using @Input

Raising an Event from a Nested Component Using @Output

What Makes a Component Nest-able?

- Its template only manages a fragment of a larger view
- It has a selector
- It optionally communicates with its container

Module Overview

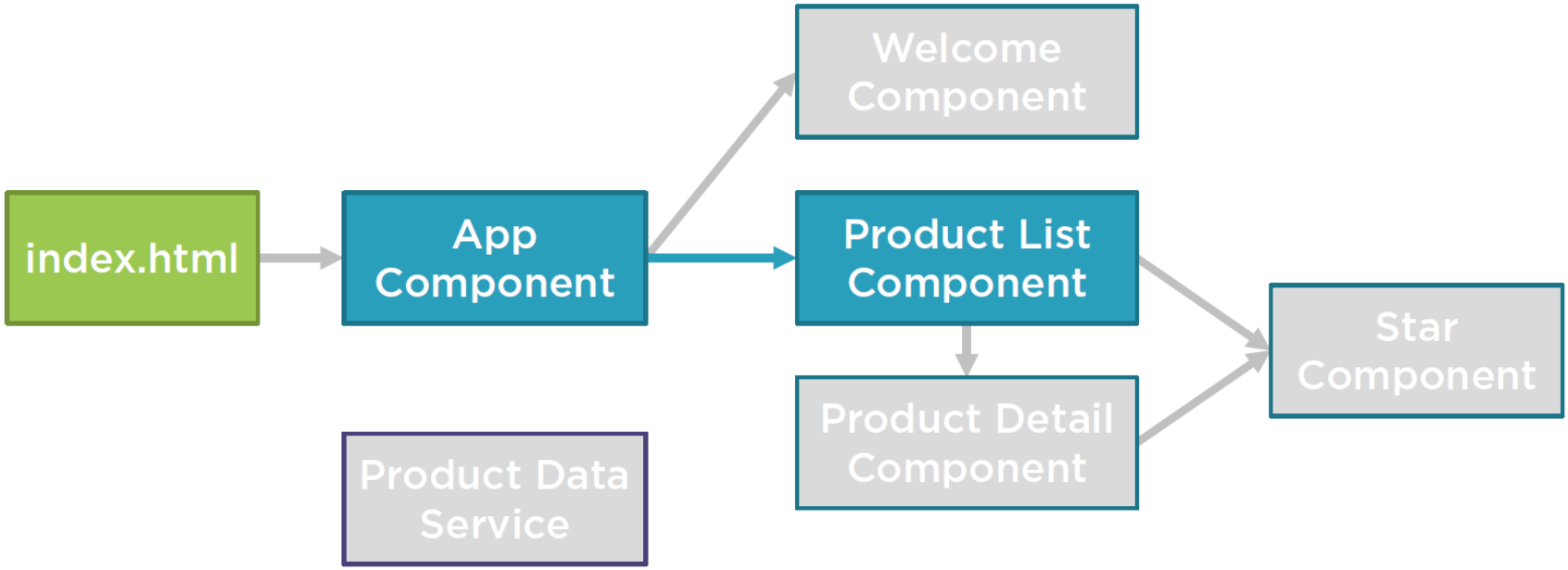
Building a Nested Component

Using a Nested Component

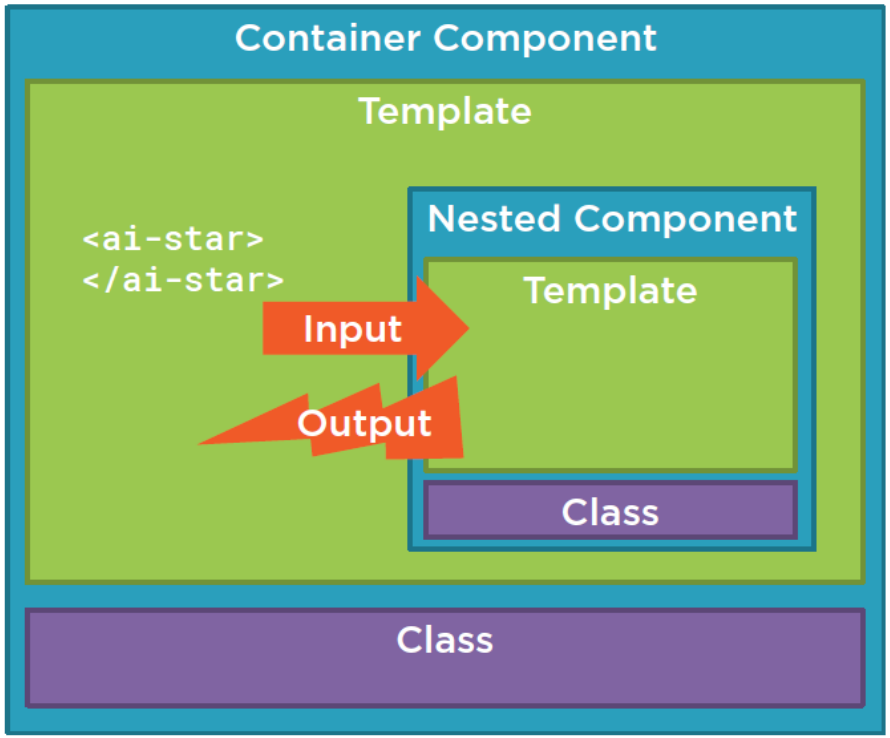
Passing Data to a Nested Component Using @Input

Raising an Event from a Nested Component Using @Output

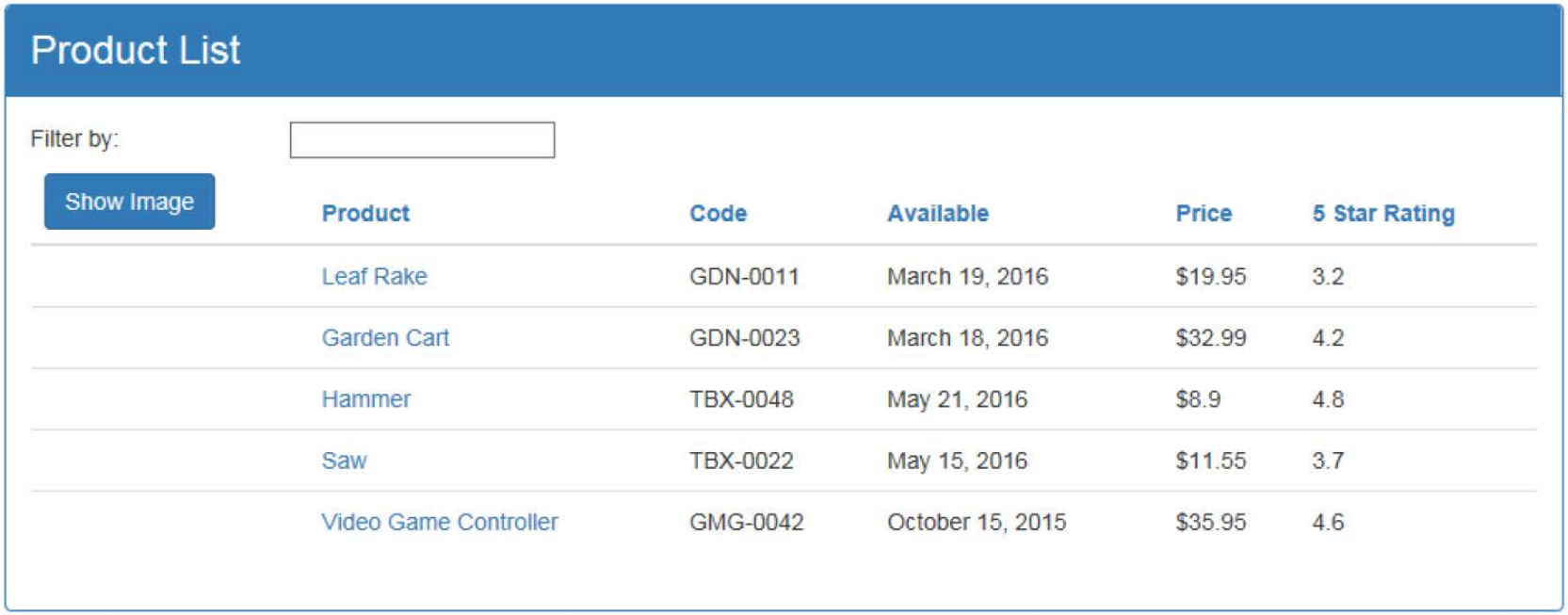
Application Architecture



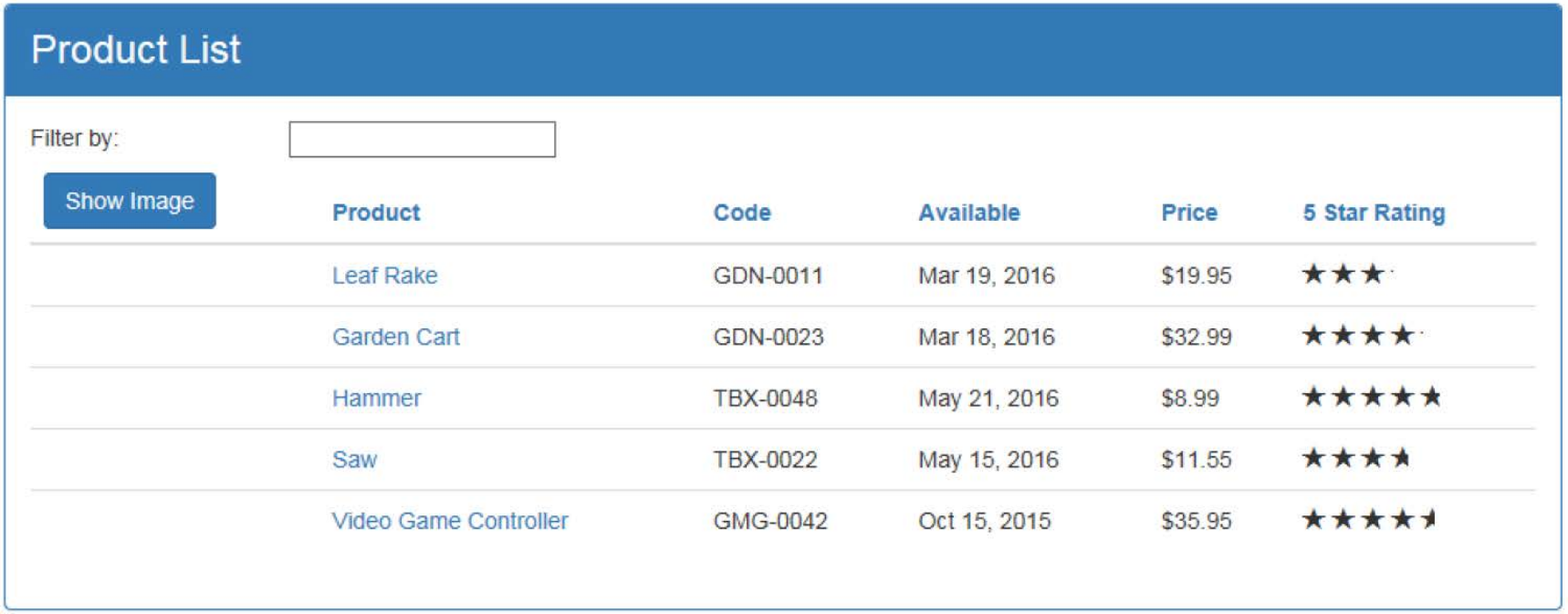
Building a Nested Component



Product List View



Product List View



Using a Nested Component as a Directive

product-list.component.ts

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

product-list.component.html

```
<td>
   {{ product.starRating | number }}
</td>
```

star.component.ts

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

Using a Nested Component as a Directive

product-list.component.ts

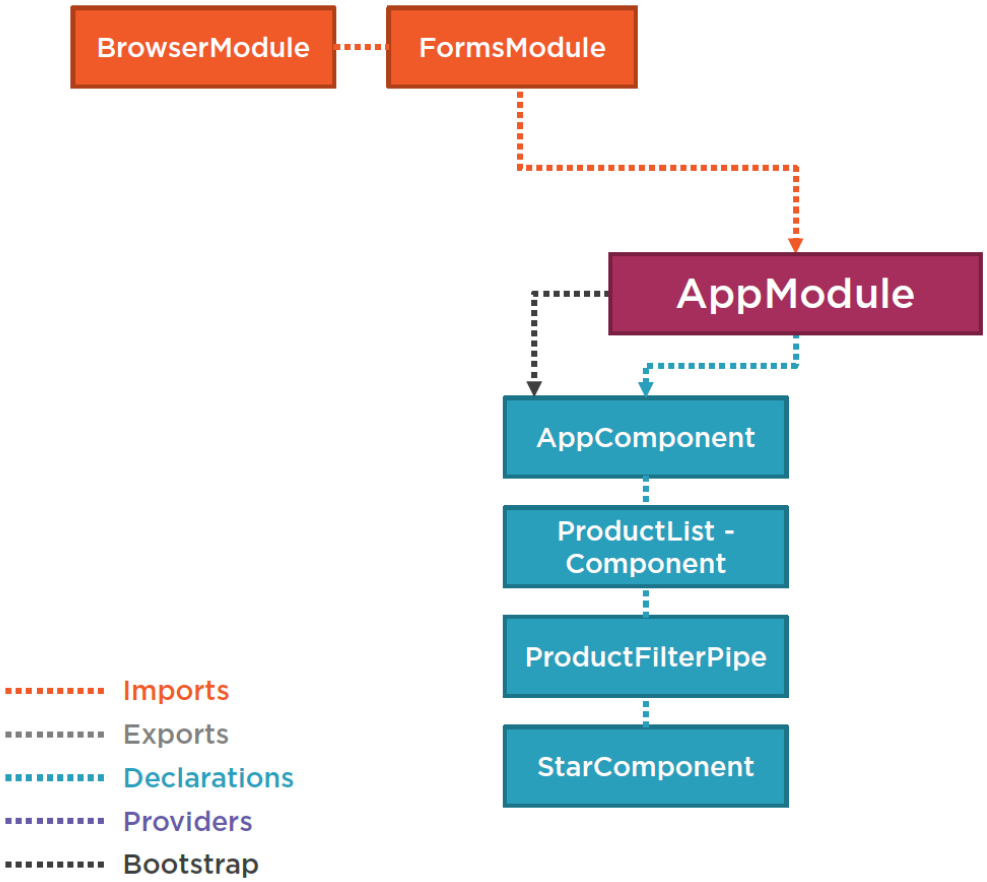
```
@Component({
  selector: 'pm-products',
  templateURL: 'product-list.component.html'
})
export class ProductListComponent { }
```

product-list.component.html

```
<td>
   <ai-star></ai-star>
</td>
```

star.component.ts

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

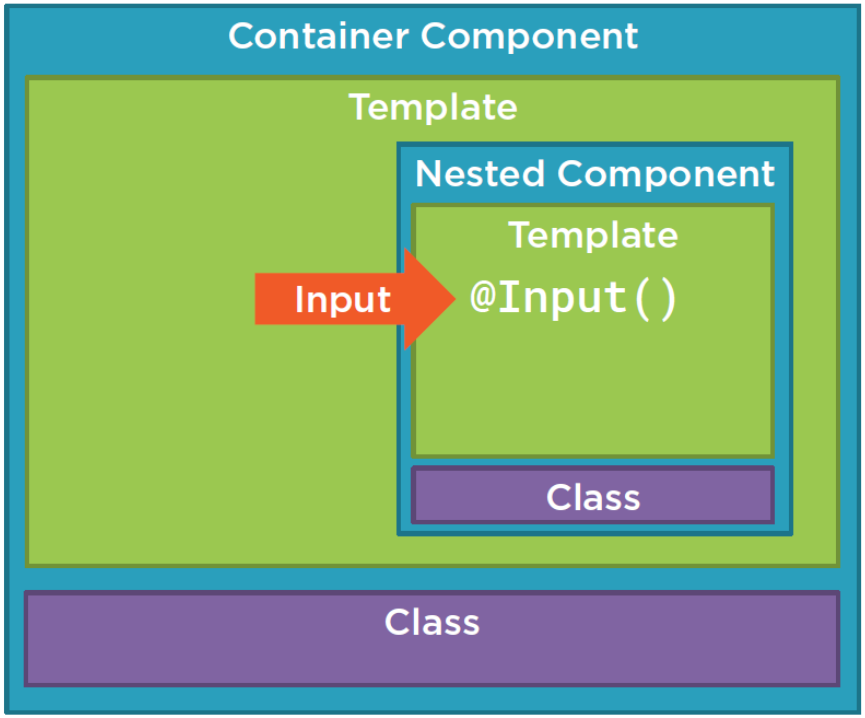

BrowserModule
FormsModule
AppModule
AppComponent
ProductList - Component
ProductFilterPipe
StarComponent
Imports
Exports
Declarations
Providers
Bootstrap

Telling Angular About Our Component

app.module.ts

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

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

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>
   <ai-star></ai-star>
</td>
```

star.component.ts

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

Passing Data to a Nested Component (@Input)

product-list.component.ts

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

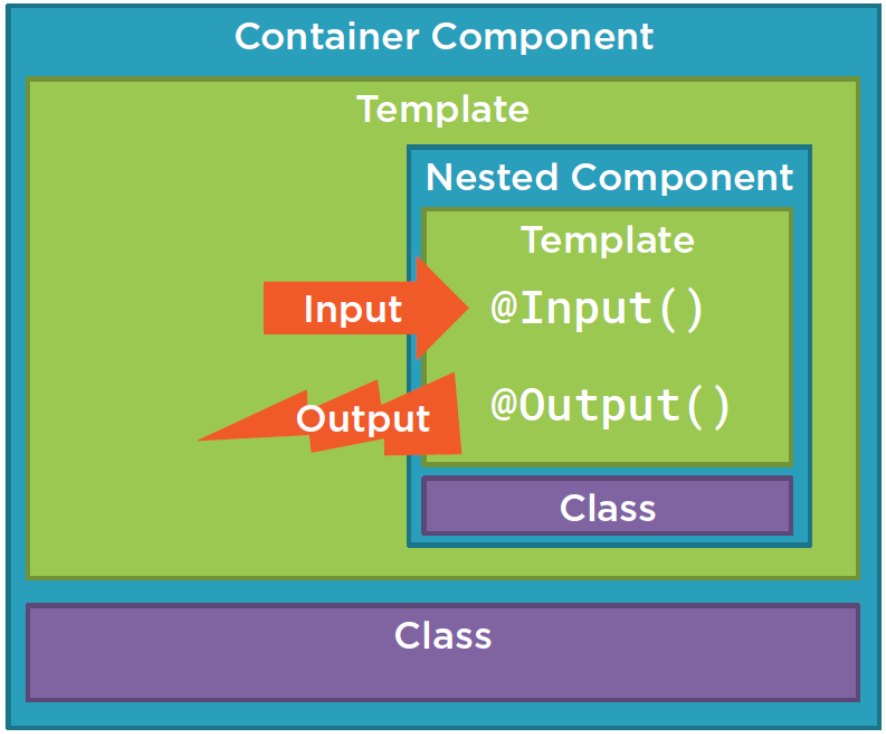
product-list.component.html

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

star.component.ts

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

Raising an Event (@Output)



Raising an Event (@Output)

product-list.component.ts

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

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
              new EventEmitter<string>();
}
```

product-list.component.html

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

Raising an Event (@Output)

product-list.component.ts

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

product-list.component.html

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

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
              new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

star.component.html

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

Raising an Event (@Output)

product-list.component.ts

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

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
              new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

product-list.component.html

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

star.component.html

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

Raising an Event (@Output)

product-list.component.ts

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

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
              new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

product-list.component.html

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

star.component.html

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

Checklist: Nested Component

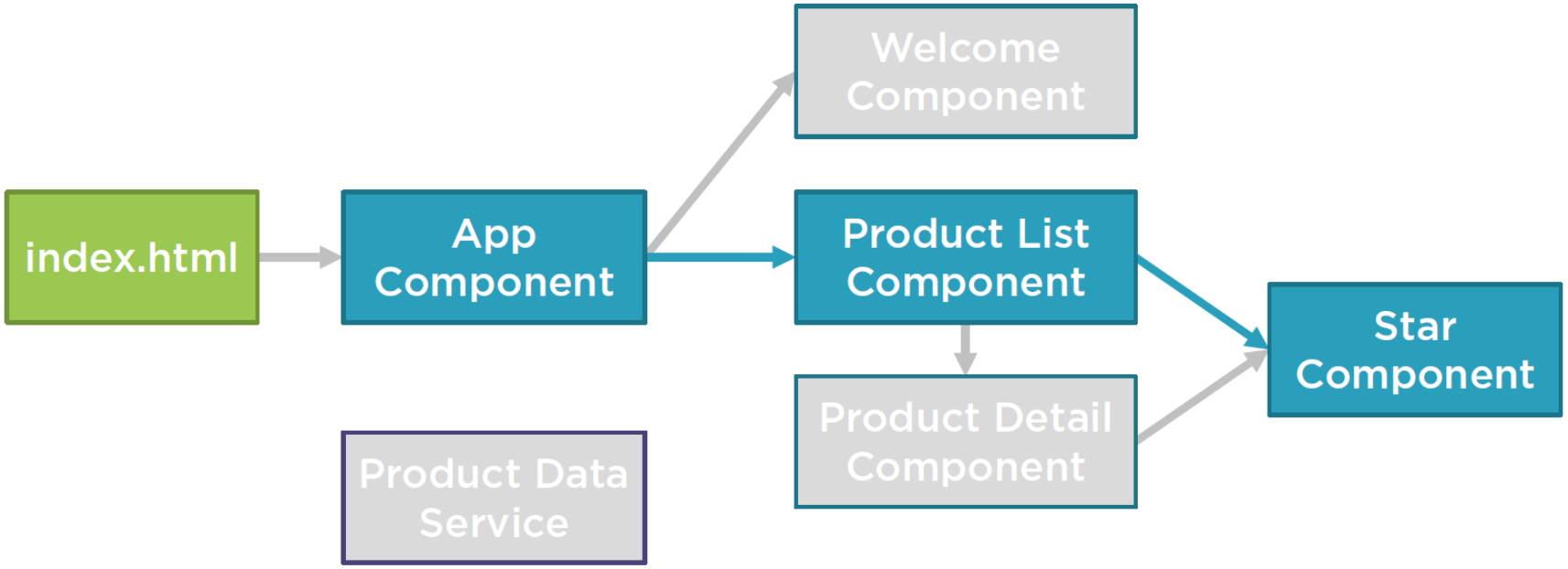
Input decorator

- Attached to a property of any type
- Prefix with @; Suffix with ()

Output decorator

- Attached to a property declared as an EventEmitter
- Use the generic argument to define the event payload type
- Use the new keyword to create an instance of the EventEmitter
- Prefix with @; Suffix with ()

Application Architecture



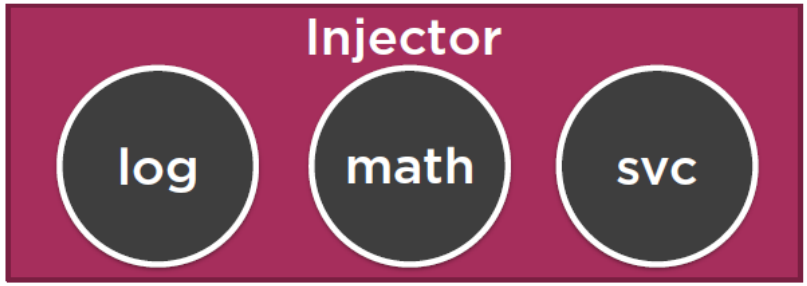
Services and Dependency Injection

Building a Service

product.service.ts

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

@Injectable()
export class ProductService {

  getProducts(): IProduct[] {
  }

}
```

Registering the Service



Service

```
export class myService {}
```

Component

```
constructor(private _myService) {}
```

Registering a Service

Register a provider

- Code that can create or return a service
- Typically the service class itself

Define in component OR Angular module metadata

Registered in component:

- Injectable to component AND its children

Registered in Angular module:

- Injectable everywhere in the application

Registering a Provider

app.component.ts

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

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

Injecting the Service

product-list.component.ts

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

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

}
```

Injecting the Service

product-list.component.ts

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

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

 constructor(private _productService: ProductService) {
 }

}
```

Module Overview

Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable

Observables and Reactive Extensions

Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Are a proposed feature for ES 2016

Use Reactive Extensions (RxJS)

Are used within Angular

Observable Operators

Methods on observables that compose new observables

Transform the source observable in some way

Process each value as it is emitted

Examples: map, filter, take, merge, ...

Promise vs Observable

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

Sending an Http Request

product.service.ts

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



@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts() {
   return this._http.get(this._productUrl);

  }
}
```

Registering the Http Service Provider

app.module.ts

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

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

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';


@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts(): Observable<Response> {
   return this._http.get(this._productUrl);

  }
}
```

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
              .map((response: Response) => <IProduct[]>response.json());
  }
}
```

Exception Handling

product.service.ts

```
...
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/catch';
...

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
              .map((response: Response) => <IProduct[]>response.json())
              .do(data => console.log('All: ' + JSON.stringify(data)))
              .catch(this.handleError);
  }

  private handleError(error: Response) {
  }
```

Subscribing to an Observable

```
x.then(valueFn, errorFn)                    //Promise
x.subscribe(valueFn, errorFn)               //Observable
x.subscribe(valueFn, errorFn, completeFn)   //Observable
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

product-list.component.ts

```
ngOnInit(): void {
    this._productService.getProducts()
            .subscribe(products => this.products = products,
                       error => this.errorMessage = <any>error);
}
```

Module Overview

How Does Routing Work?

Configuring Routes

Tying Routes to Actions

Placing the Views

How Routing Works

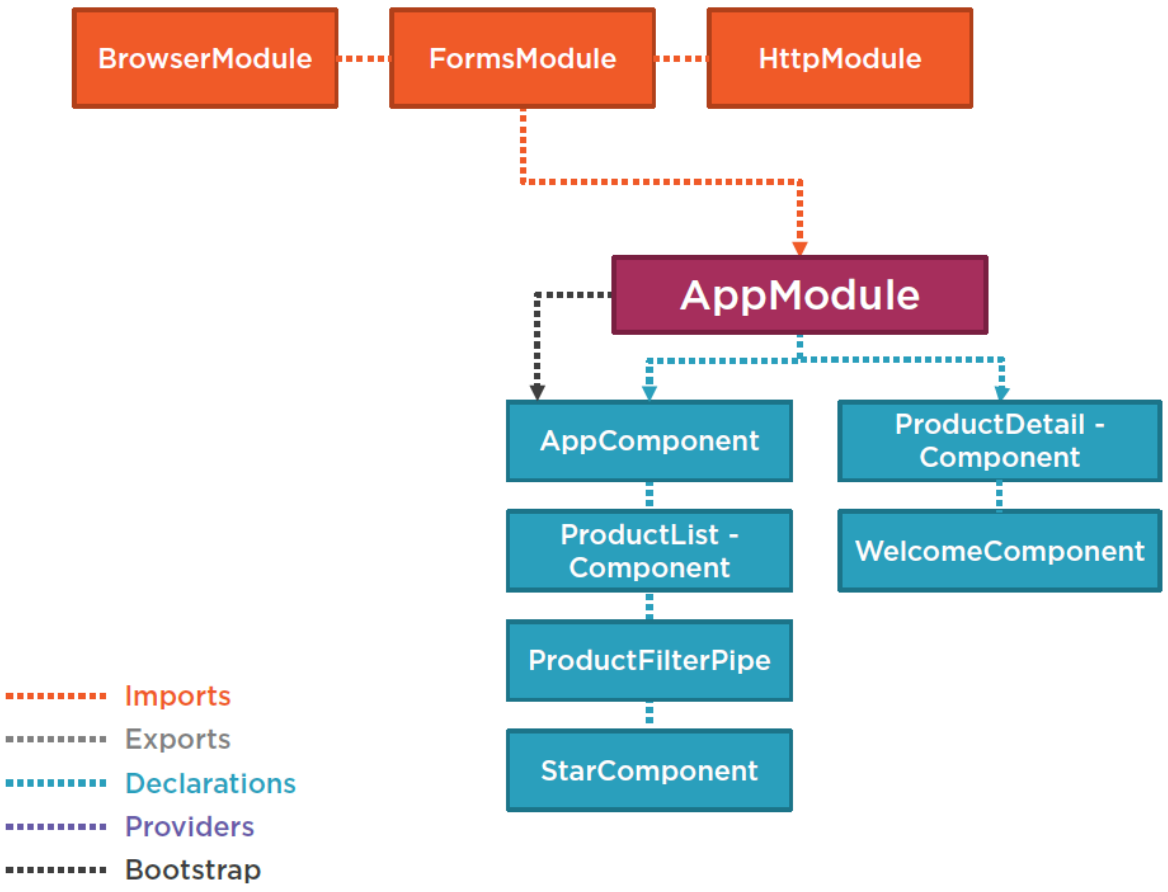

```
<a routerLink="/products">Product List</a>
```

```
{ 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 { }
```


BrowserModule
FormsModule
HttpModule
AppModule
AppComponent
ProductDetail - Component
ProductList - Component
WelcomeComponent
ProductFilterPipe
StarComponent
Imports
Exports
Declarations
Providers
Bootstrap


'products', ProductListComponent
'product/:id', ProductDetailComponent
'welcome', WelcomeComponent

Configuring Routes

app.module.ts

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

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

Configuring Routes

app.module.ts

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

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

Configuring Routes

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

Routes for Root

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

```
const routes: Routes = [
    { path: 'property',  loadChildren: 'app/views/property/property.module#PropertyModule' },
    { path: 'valuation', loadChildren:  'app/views/valuation/valuation.module#ValuationModule'  },
    { path: 'acquisitions',  loadChildren:  'app/views/acquisitions/acquisitions.module#AcquisitionsModule' },
    { path: 'reports',  loadChildren:  'app/views/reports/reports.module#ReportsModule' },
    { path: 'admin', loadChildren:  'app/views/admin/admin.module#AdminModule'  },
    { path: 'operating-capital',  redirectTo: 'operating-capital', pathMatch: 'full'},
    { path: 'sales', redirectTo: 'sales', pathMatch: 'full'},
    { path: '', redirectTo: 'dashboard', pathMatch: 'full'}
];
```

```
import { AppRoutingModule } from './app-routing.module';
import {DashboardModule} from './views/dashboard/dashboard.module';

@NgModule({
  imports: [
    BrowserModule,
    HttpModule,
    CoreModule.forRoot({userName: 'MetLife EMS'}),
    DashboardModule,
    ServicesModule,
    AppRoutingModule,
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Load Routes for Early Load feature Modules

- Import feature routing module(Dashboard Routing Module) to Feature Module(Dashboard Module) and import feature module(DashboardModule) to App Module.

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

import { DashboardComponent }    from './dashboard.component';
import { PropertyPortfolioComponent }    from './property-portfolio.component';
import { LoanMaturitiesComponent }    from './loan-maturities.component';
import { PropertySearchComponent }    from './property-search.component';
import { WorkflowComponent }    from './workflow.component';
```

```
export const routes: Routes = [
  { path: 'dashboard', component: DashboardComponent,
    children: [
      { path: 'property-search', component: PropertySearchComponent },
      { path: 'workflow-dashboard', component: WorkflowComponent },
      { path: 'loan-maturities', component: LoanMaturitiesComponent },
      { path: 'property-portfolio', component: PropertyPortfolioComponent },
       { path: '', component: PropertyPortfolioComponent },
    ]
  }
];
```

```
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})

export class DashboardRoutingModule { }
```

Lazy Load Modules

- Modules will be loaded on demand,
- Modules wont be part of App Module.
- Feature modules will get loaded to system, based on router link action.
- Declare Lazy load routing modules using LoadChildren.
- Define routing compoents as part of Routing Module and Import to Feature Module(Lazy Load Feature Module)

```
const routes: Routes = [
    { path: 'property',  loadChildren: 'app/views/property/property.module#PropertyModule' },
    { path: 'valuation', loadChildren:  'app/views/valuation/valuation.module#ValuationModule'  },
    { path: 'operating-capital',  loadChildren:  'app/views/operating-capital/operating-capital.module#OperatingCapitalModule' },
    { path: 'acquisitions',  loadChildren:  'app/views/acquisitions/acquisitions.module#AcquisitionsModule' },
    { path: 'sales',  loadChildren:  'app/views/sales/sales.module#SalesModule' },
    { path: 'reports',  loadChildren:  'app/views/reports/reports.module#ReportsModule' },
    { path: 'admin', loadChildren:  'app/views/admin/admin.module#AdminModule'  },
    { path: '', redirectTo: 'dashboard/property-portfolio', pathMatch: 'full'}
];
```

Child Routes with Lazy Load Routes

Children routes cant have redirectTo
Children routes can have lazy load routes
For all Routings precedence has to be followed

```
import { PropertyInputComponent } from './property-input.component';
import { PropertySummaryComponent } from './property-summary.component';
import { PropertyComponent } from './property.component';

export const routes: Routes = [
  {  path: 'property', component: PropertyComponent },
  {  path: '', component: PropertyComponent,
     children: [
      { path: '', component: PropertySummaryComponent },
      { path: 'property-summary', component: PropertySummaryComponent },
      { path: 'input', component: PropertyInputComponent },
      { path: 'loan-information', loadChildren: 'app/views/loan-information/loan-information.module#LoanInformationModule' },
      { path: 'book-value', loadChildren: 'app/views/book-value/book-value.module#BookValueModule' },
    ]
  }
];
```

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

Passing Parameters to a Route

app.module.ts

```
@NgModule({
  imports: [
    ...,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },
      { path: 'product/: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]="['/product', product.productId]">
     {{product.productName}}
  </a>
</td>
```

app.module.ts

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

Reading Parameters from a Route

product-detail.component.ts

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

    constructor(private _route: ActivatedRoute) {
       console.log(this._route.snapshot.params['id']);
    }
```

app.module.ts

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

Route Params Recommended option

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

```
public ngOnInit(): void {
    this.propertySk = 0;
    this.route.params.forEach((params: Params) => {
        if (params[this.appConstants.paramspropertySk]) {
            this.propertySk = +params[this.appConstants.paramspropertySk];
```

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

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-guard.service.ts

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

@Injectable()
export class ProductDetailGuard implements CanActivate {

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

Registering a Guard

app.module.ts

```
...
import { ProductDetailGuard } from './products/product-guard.service';

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

Using a Guard

app.module.ts

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

Routing Checklist: Passing Parameters

app.module.ts

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

product-list.component.html

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

product-detail.component.ts

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

constructor(private _route: ActivatedRoute) {
   console.log(this._route.snapshot.params['id']);
}
```

Module Overview

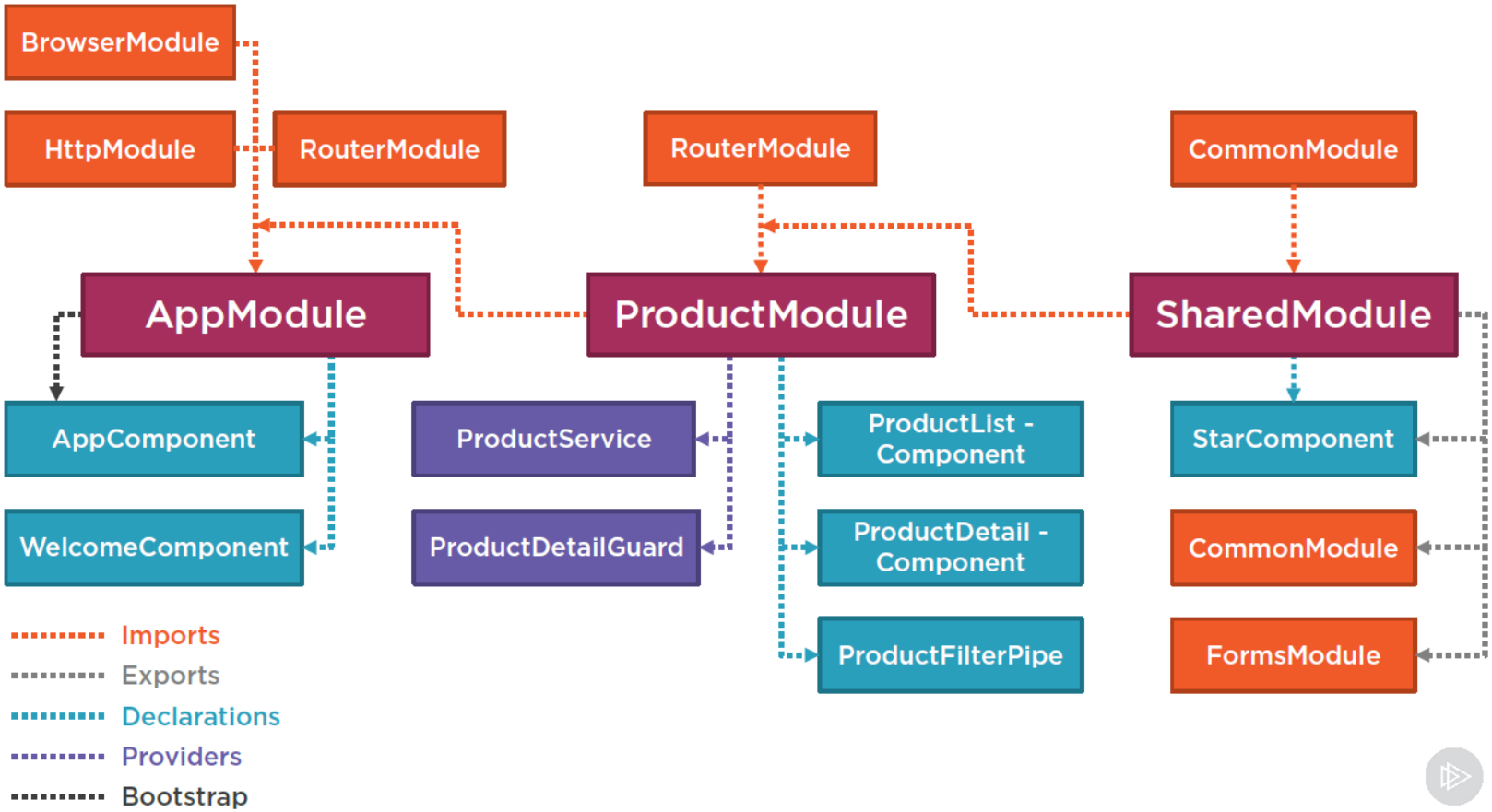
What Is an Angular Module?

Angular Module Metadata

Creating a Feature Module

Defining a Shared Module

Revisiting AppModule


BrowserModule
HttpModule
RouterModule
RouterModule
CommonModule
AppModule
ProductModule
SharedModule
AppComponent
WelcomeComponent
ProductService
ProductDetailGuard
ProductList - Component
ProductDetail - Component
ProductFilterPipe
StarComponent
CommonModule
FormsModule
Imports
Exports
Declarations
Providers
Bootstrap

What Is an Angular Module?

A class with an NgModule decorator

Its purpose:

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

Going to Production

Agenda

- Linting
- Tuning RxJS Requests
- Enabling Production Mode
- Basic Deployment
- AOT Compiler
- Tree Shaking & Bundling
- Webpack
- Optimistic Bundle Download

Preparation Levels

Minimal

Tune RxJS Requests

Enable Prod Mode

Full

Use AOT

Bundle

Minify

Tree Shake

Lazy Loading

Use Lazy Loaded Modules

Minification, Bundling, Code Splitting, and Tree Shaking

Minification

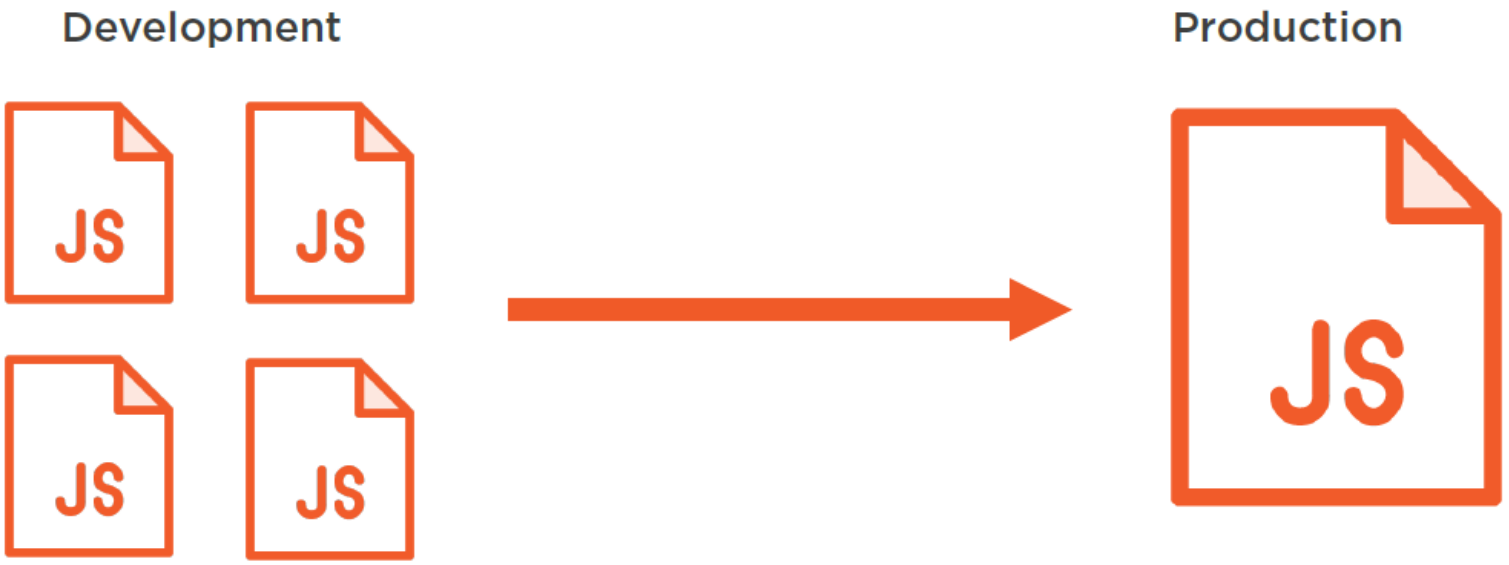
Source

```
export class orderService {

  addOrder(order) {
    ...
  }

  findOrder(id) {
    ...
  }
```

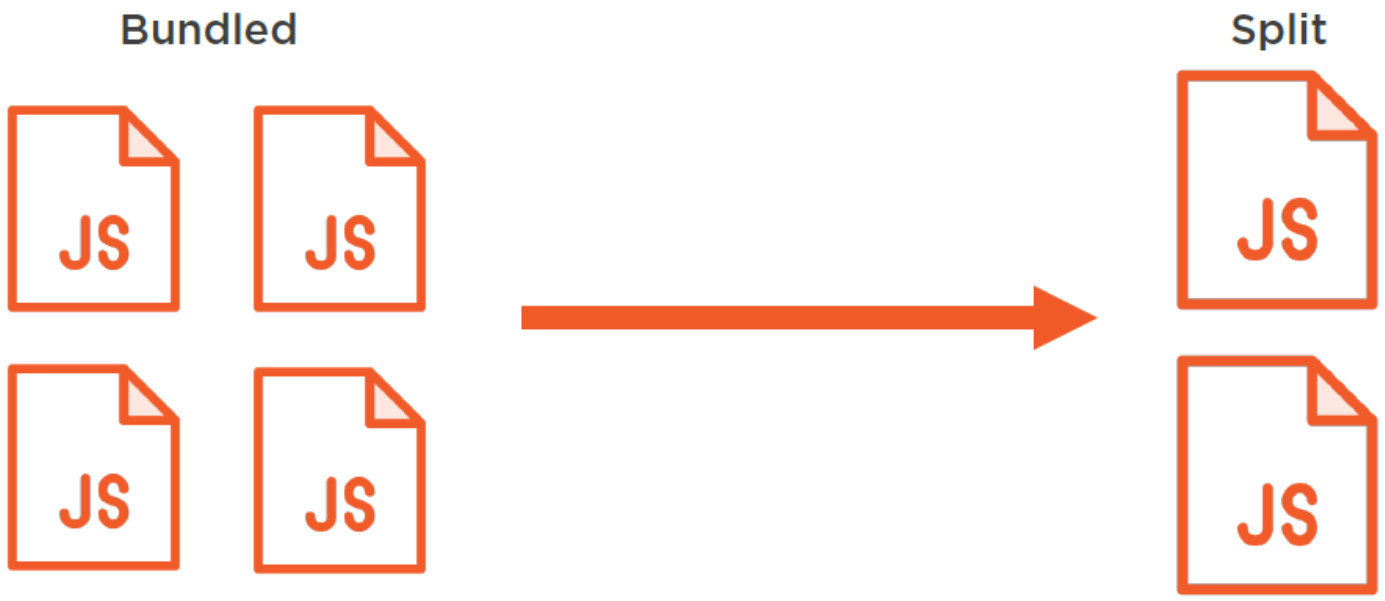
Service

```
export class orderService{add
Order(order){...}findOrder(id){
...}}
```

Bundling



Code Splitting



Tree Shaking

Calling Code

```
...
userService.getUsers()
...
```

Service

```
export class UserService {
  getUsers() {
    ...
  }

  getUser(id) {
    ...
  }
}
```

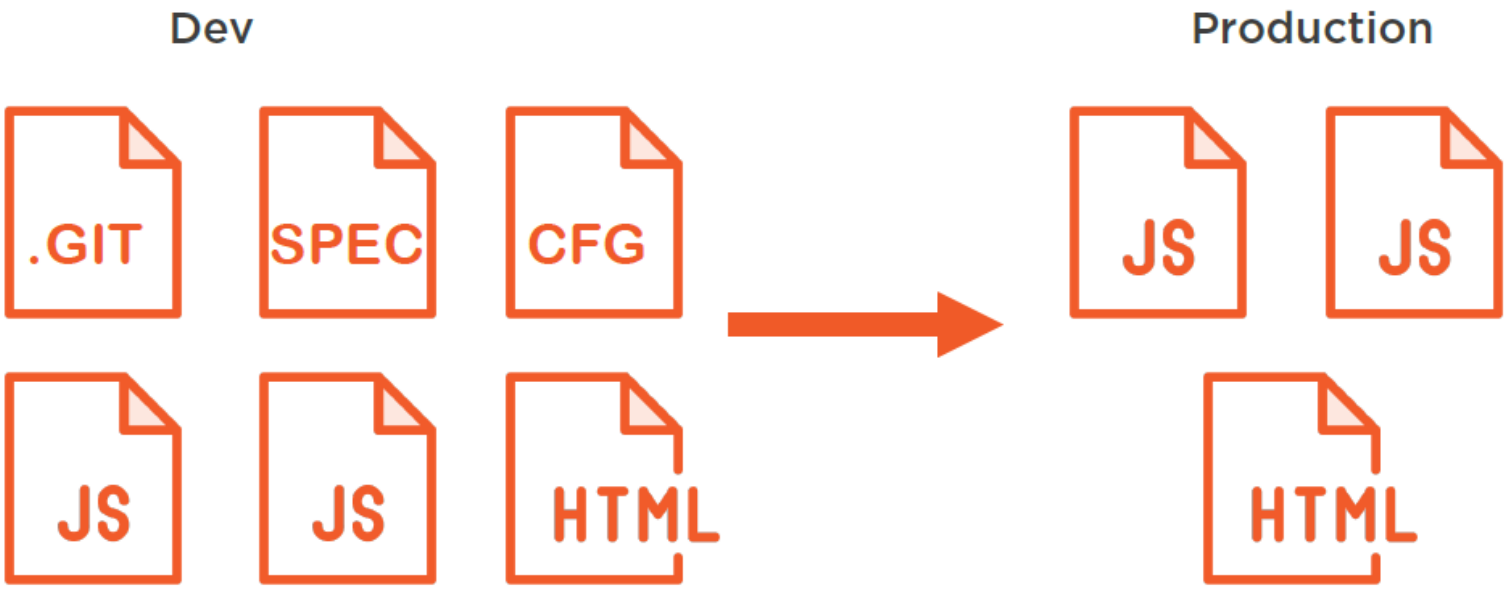
Bundler Options

SystemJS

Rollup

Webpack

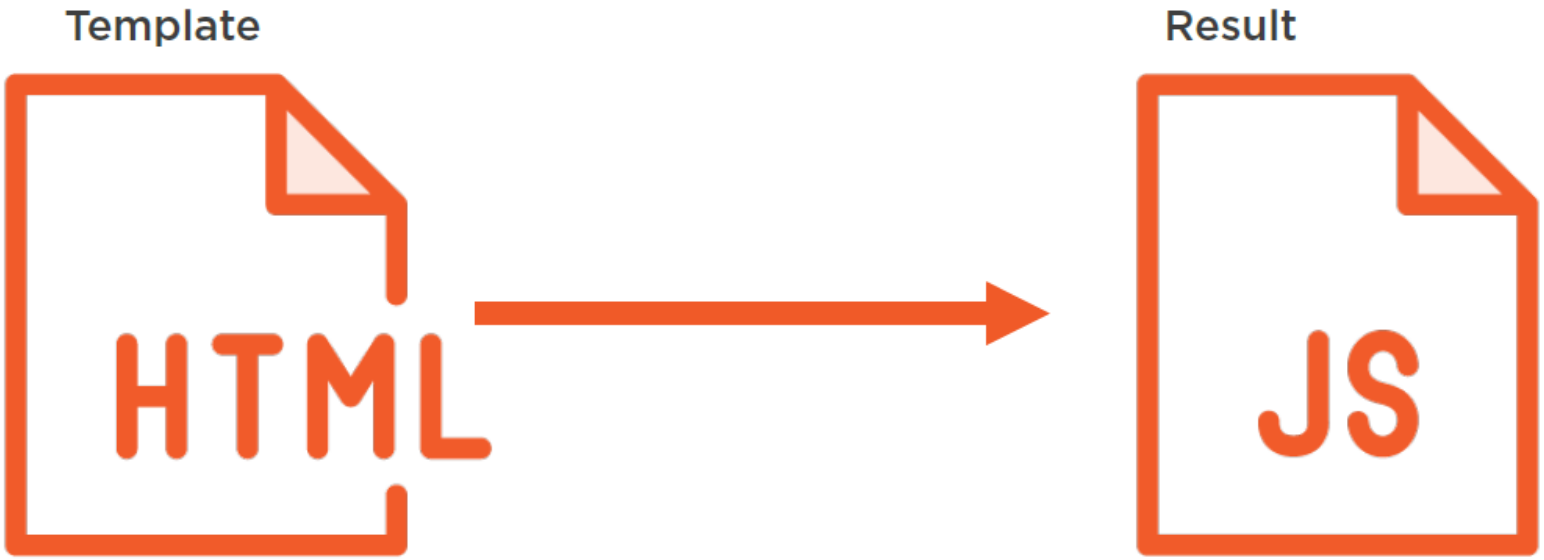
Simple Copy Deployment



AOT Benefits

- Faster Rendering
- Fewer Requests
- Smaller Angular Framework Downlo
- Detect Template Errors
- Better Security

Compiling Templates



```
<div well-title>
    {{session.name}}
    <i *ngIf="session.voters.length > 3“
            class="glyphicon glyphicon-fire"
            style="color:red"></i>
</div>
```

Development = Just In Time Compiler

Much of Angular Needed in Browser is the Compiler

Takes Time to Compile Templates

Production = Ahead of Time Compiler

AOT No-No's

form.controls.controlName

control.errors?.someError

Default exports

Functions in providers, routes or declarations of a module

Any field used in a template, including Inputs, must be public

declare var for globals

Thank You