

Angular: Getting Started

Introduction



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata

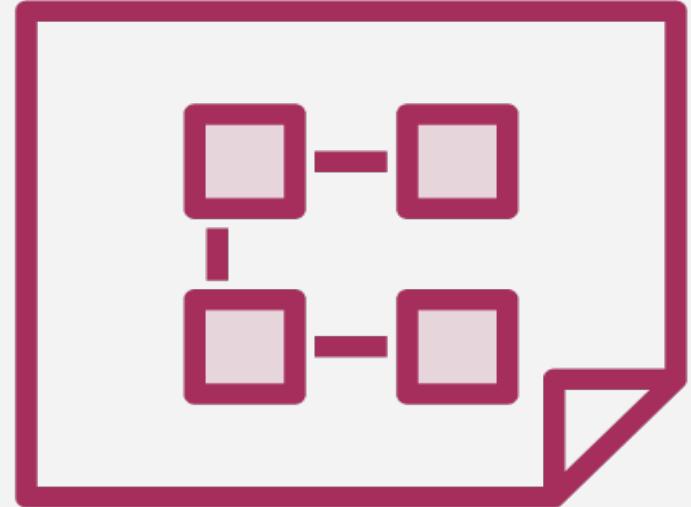
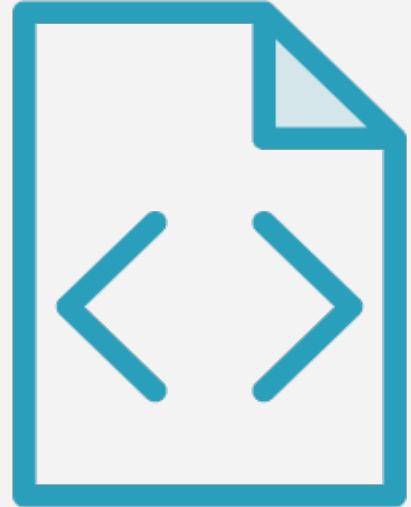


Angular Is ...



**A JavaScript framework
For building client-side applications
Using HTML, CSS and TypeScript**

Why Angular?



Expressive HTML

**Powerful Data
Binding**

**Modular By
Design**

**Built-in Back-End
Integration**

Module Overview



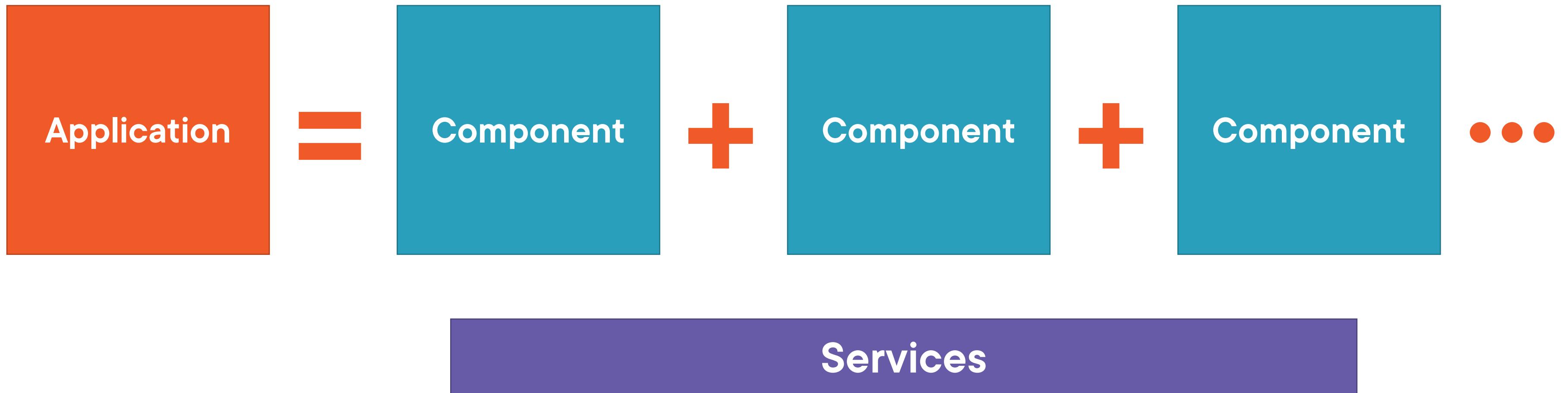
Anatomy of an Angular application

Getting the most from this course

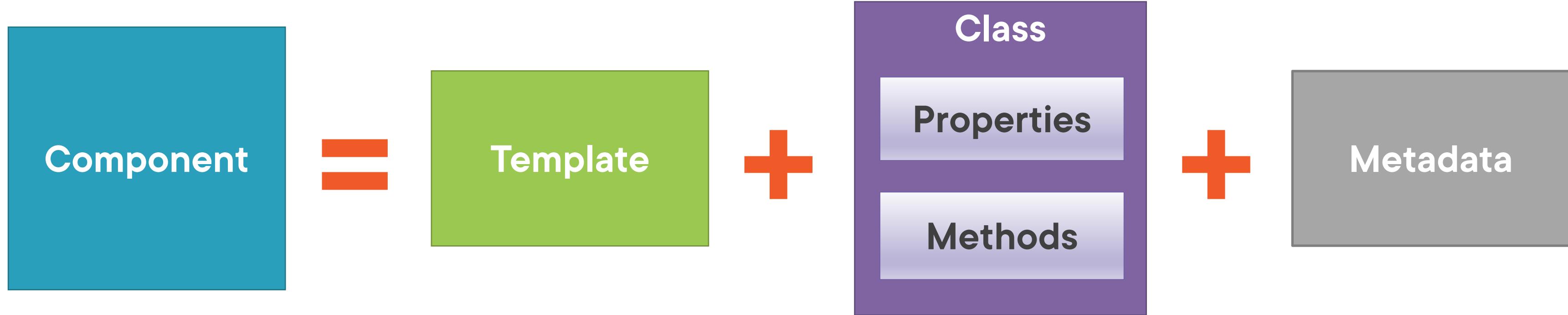
Sample application

Course outline

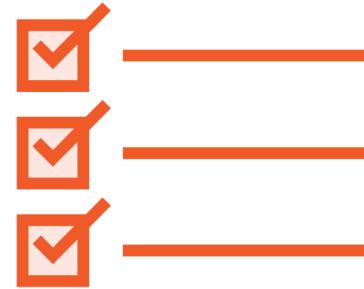
Anatomy of an Angular Application



Component

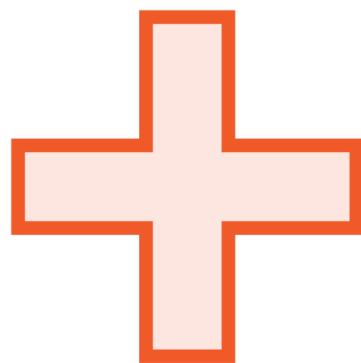


Prerequisites



Required

- JavaScript basics
- HTML basics
- CSS basics



Helpful

- Object-oriented programming (OOP) concepts
- C++, C#, Java, ...



Not required

- Prior knowledge of Angular
- Prior knowledge of TypeScript

Thoughts? Comments? Questions?

Angular: Getting Started

by Deborah Kurata

Angular is one of the fastest, most popular open source web app frameworks today, and knowing how to use it is essential for developers. You'll learn how to create components and user interfaces, data-binding, retrieving data using HTTP, and more.

Namespaces Modules Code Organization

Angular 1 Modules TypeScript Modules

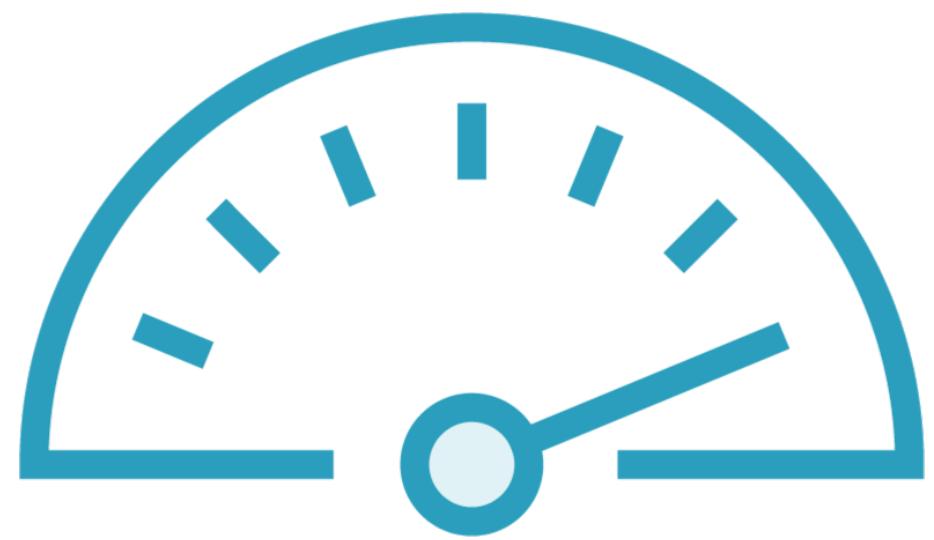
ES 2015

Resume Course Bookmarked Add to Channel Download Course Schedule Reminder

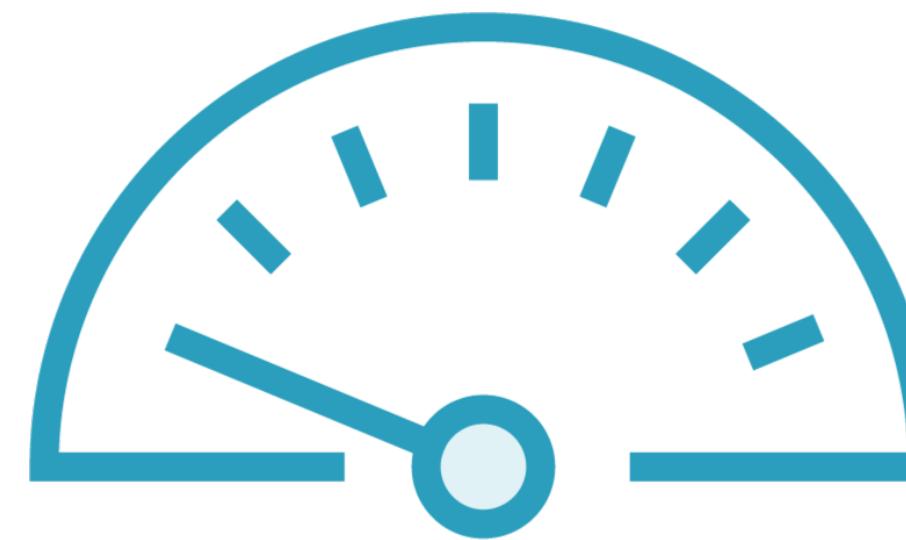
Table of contents Description Transcript Exercise files Discussion Learning Check Related Courses

@deborahkurata

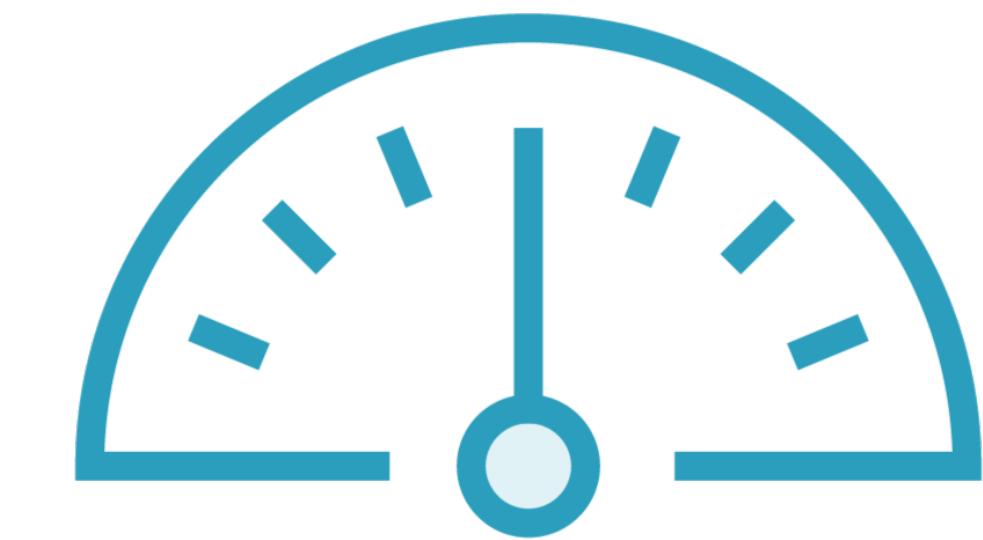
Adjust the Speed



Too fast?



Or too slow?



Just Right

Checklist

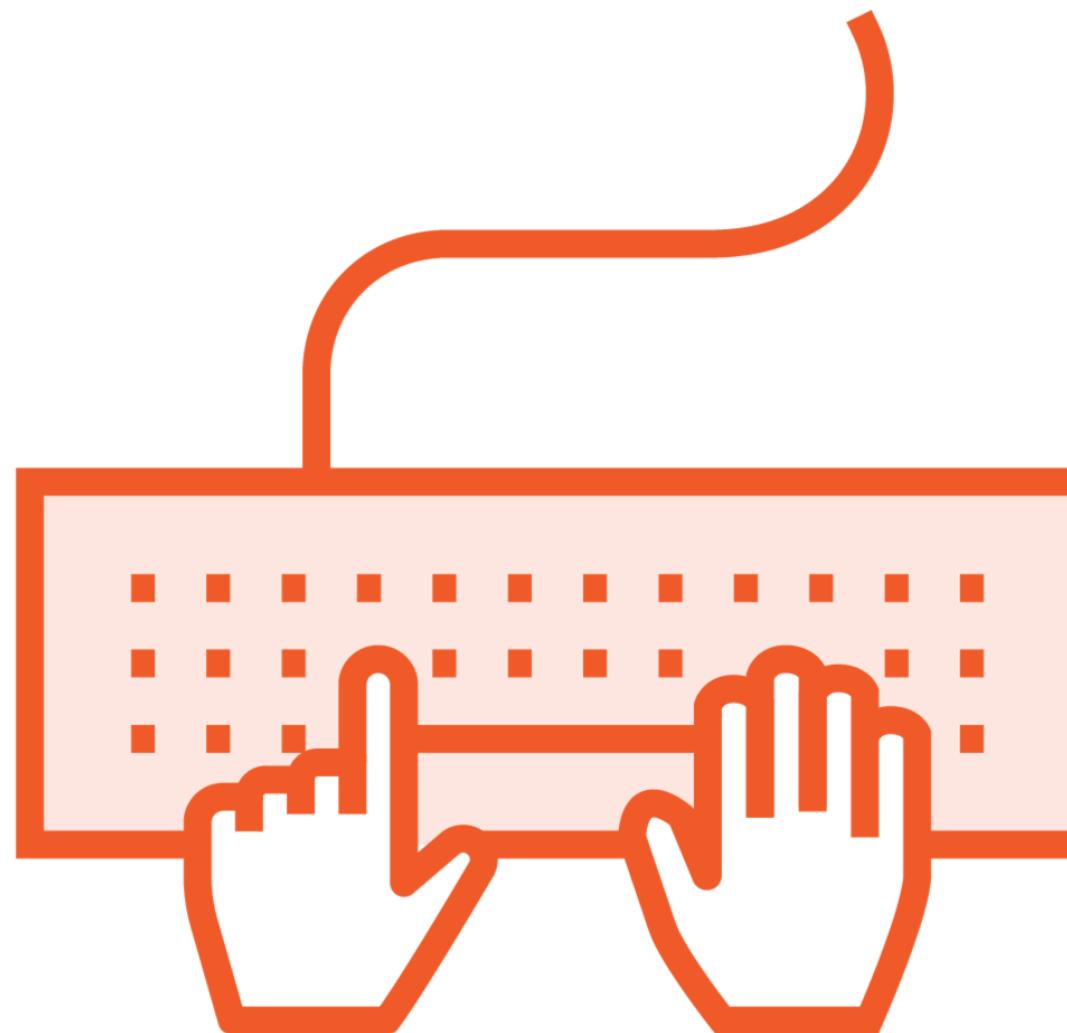


Review module concepts

Code along assistance

Revisit as you build

Coding Along



Code along with the demos throughout this course

Not required

Helpful for practicing what you learn

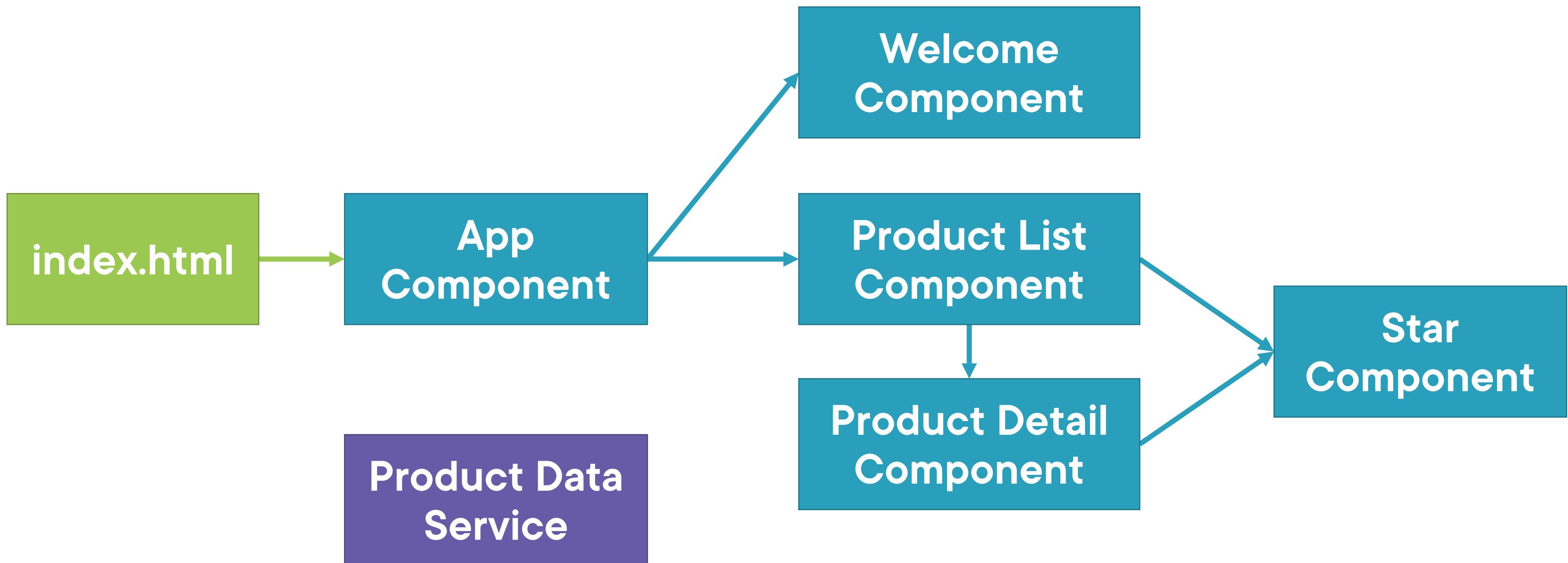
You'll have a working sample application

Demo



Sample application in action

Sample Application Architecture



Course Outline



First Things First

Introduction to Components

Templates, Interpolation, and Directives

Data Binding & Pipes

More on Components

Building Nested Components

Services and Dependency Injection

Retrieving Data Using HTTP

Navigation and Routing

Angular Modules

Building, Testing and Deploying with the CLI



Coming up next ...

First Things First

First Things First



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview



Introduce TypeScript

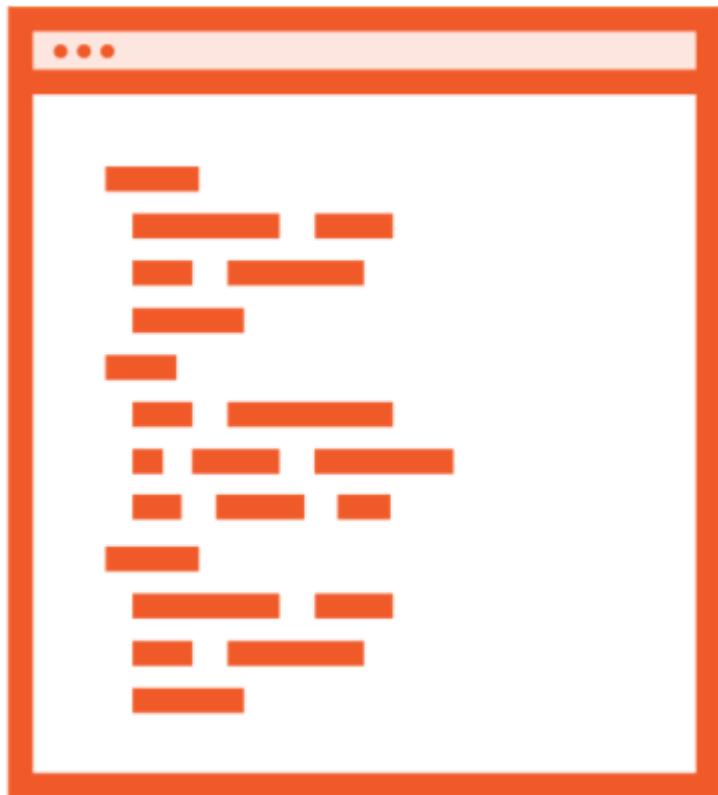
Install what we need

Set up and run our Angular application

**Create an Angular application with the
Angular CLI**

TypeScript
is the programming language
we use
when building
Angular applications

JavaScript



The language for the Web

ECMAScript (ES)

ES 3/5

ES 2015 (formerly known as ES 6)

- Classes**
- Arrow functions**

**Newer JavaScript transpiled to older
JavaScript**

TypeScript



- Open-source language**
- Superset of JavaScript**
- Transpiles to plain JavaScript**
- Strongly typed**
- Class-based object-orientation**

Learning More



TypeScript Playground
<https://www.typescriptlang.org/play>

Pluralsight Course
"TypeScript: Getting Started"

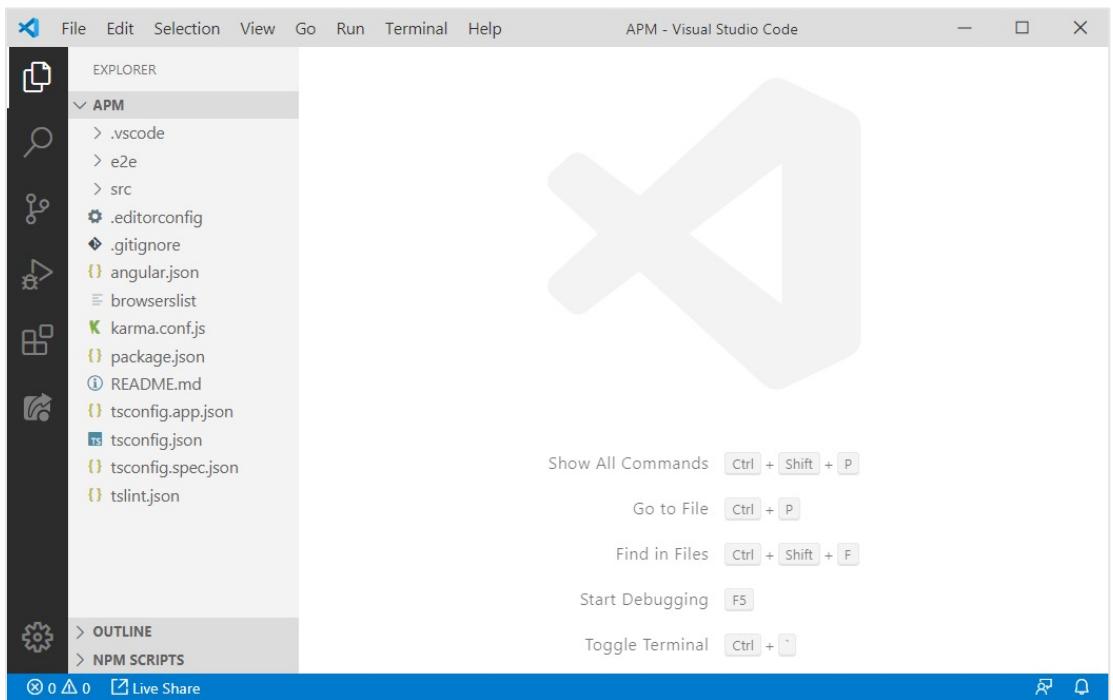
Installing What We Need



Editor

- **Visual Studio (VS) Code**

VS Code



Created by Microsoft

Runs in Linux, Windows, and OS X

**Has numerous features that support
TypeScript**

- Auto-completion**
- Intellisense**
- Syntax checking**
- Refactorings**

It's FREE

Visual Studio Code

The screenshot shows the official Visual Studio Code website. The top navigation bar includes links for Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, Learn, a search bar labeled "Search Docs", and a blue "Download" button. The main content area features a large heading "Code editing. Redefined." with the subtitle "Free. Built on open source. Runs everywhere." Below this are download links for Windows, Linux (deb/rpm), and OS X, along with a "Zip archive" link. A note states: "By using VS Code, you agree to its license and privacy statement." To the right, a code editor window displays a file named "serviceWorker.js" from a "create-react-app" project. The editor shows code related to service workers and navigation. The bottom of the page includes a terminal window showing "1:node", a message about viewing the app in the browser, local and network URLs, and status information like "Ln 43, Col 19".

<https://code.visualstudio.com/>

Learning More



Visual Studio Code Site
<https://code.visualstudio.com/>

Pluralsight Course
"Visual Studio Code"

Installing What We Need



✓ **Editor**
– **VS Code**

Installing What We Need



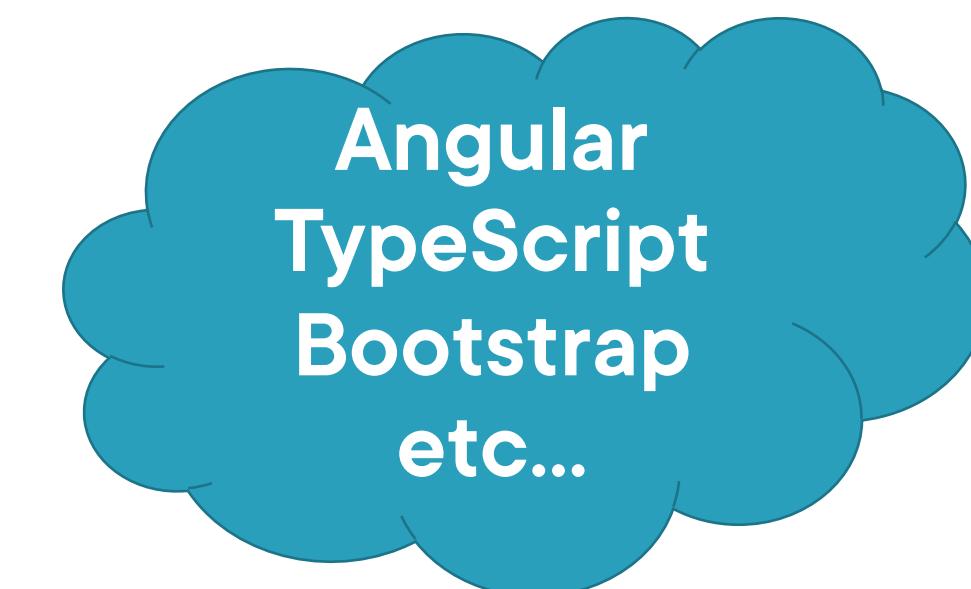
Editor

- **VS Code**

npm

- **Node package manager**

npm



Open-source
repository

Angular
TypeScript
Bootstrap
etc...

Command-line utility for
interacting with the repository

The image shows two Command Prompt windows. The top window displays the command `npm install abc`. The bottom window shows a directory listing for `c:\Users\Deborah`, including `newv9app`, `node_modules`, and `.nuget`.

```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Deborah>npm install abc

Directory of c:\Users\Deborah

02/07/2020  05:58 PM    <DIR>          newv9app
03/25/2020  04:53 PM    <DIR>          node_modules
01/04/2019  05:33 PM    <DIR>          .nuget
                                         0 File(s)           0 bytes
                                         3 Dir(s)  1,371,293,900,800 bytes free

C:\Users\Deborah>
```

npm



Installs libraries, packages, and applications

Executes scripts

<https://nodejs.org/en/download>

Installing What We Need



- ✓ **Editor**
– **VS Code**

- ✓ **npm**
– **Node package manager**

What Else Do We Need?



Angular

- Framework and libraries

Angular CLI

- Command line interface for Angular

TypeScript

- Programming language

Testing tools, linters, etc.

package.json

dependencies

- Packages required for development and deployment

devDependencies

- Packages only required for development

```
12 "dependencies": {
13   "@angular/animations": "~12.0.0",
14   "@angular/common": "~12.0.0",
15   "@angular/compiler": "~12.0.0",
16   "@angular/core": "~12.0.0",
17   "@angular/forms": "~12.0.0",
18   "@angular/platform-browser": "~12.0.0",
19   "@angular/platform-browser-dynamic": "~12.0.0",
20   "@angular/router": "~12.0.0",
21   "rxjs": "~6.6.0",
22   "tslib": "^2.1.0",
23   "zone.js": "~0.11.4"
24 },
25 "devDependencies": {
26   "@angular-devkit/build-angular": "~12.0.0",
27   "@angular/cli": "~12.0.0",
28   "@angular/compiler-cli": "~12.0.0",
29   "@types/jasmine": "~3.6.0",
30   "@types/node": "^12.11.1",
31   "jasmine-core": "~3.7.0",
32   "karma": "~6.3.0",
33   "karma-chrome-launcher": "~3.1.0",
34   "karma-coverage": "~2.0.3",
35   "karma-jasmine": "~4.0.0",
36   "karma-jasmine-html-reporter": "^1.5.0",
37   "typescript": "~4.2.3"
38 }
```

When Setting Up Existing Angular Code

#1

Navigate down to the project folder

The project folder contains the `package.json` file

#2

Run: `npm install`

To install the packages defined in the `package.json` file

#3

Run: `npm start`

To start the installed Angular application

Setting up Our Angular Application



Starter files

[https://github.com/DeborahK/Angular-
GettingStarted](https://github.com/DeborahK/Angular-GettingStarted)

C:\Users\Deborah\Pluralsight\Angular Getting Started\APM

	Name	Date modified	Type	Size
Q	.vscode	5/17/2021 1:23 PM	File folder	
D	src	5/17/2021 1:23 PM	File folder	
D	.browserslistrc	5/17/2021 1:17 PM	BROWSERSLISTRC...	1 KB
	.editorconfig	5/17/2021 1:17 PM	Editor Config Sour...	1 KB
O	.gitignore	5/17/2021 1:17 PM	Text Document	1 KB
T	angular.json	5/17/2021 1:32 PM	JSON Source File	4 KB
	karma.conf.js	5/17/2021 1:17 PM	JavaScript File	2 KB
E	package.json	5/17/2021 1:28 PM	JSON Source File	2 KB
N	README.md	5/17/2021 1:17 PM	Markdown Source...	1 KB
	tsconfig.app.json	5/17/2021 1:17 PM	JSON Source File	1 KB
	tsconfig.json	5/17/2021 1:17 PM	JSON Source File	1 KB
	tsconfig.spec.json	5/17/2021 1:17 PM	JSON Source File	1 KB

12 items

When Setting Up Existing Angular Code

#1

Navigate down to the project folder

The project folder contains the `package.json` file

#2

Run: `npm install`

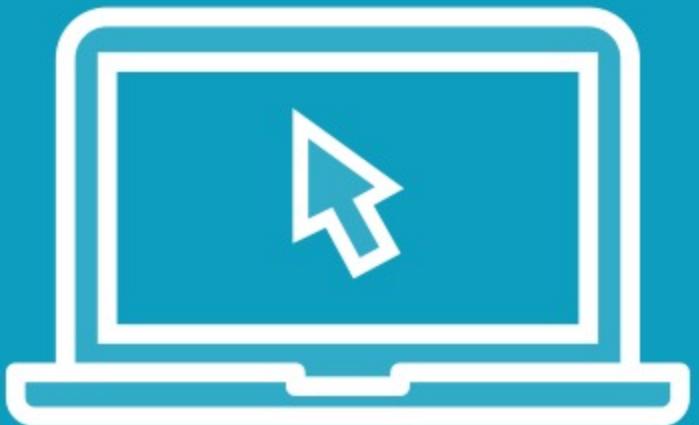
To install the packages defined in the `package.json` file

#3

Run: `npm start`

To start the installed Angular application

Demo



**Create an Angular app using the
Angular CLI**

Summary



Introduced TypeScript

Installed what we need

- Editor (VS Code)
- npm (node)

Set up and ran our Angular application

Used the Angular CLI to create a new Angular application

Angular CLI

Global Angular CLI

projects/apm

projects/hello-world

projects/cms

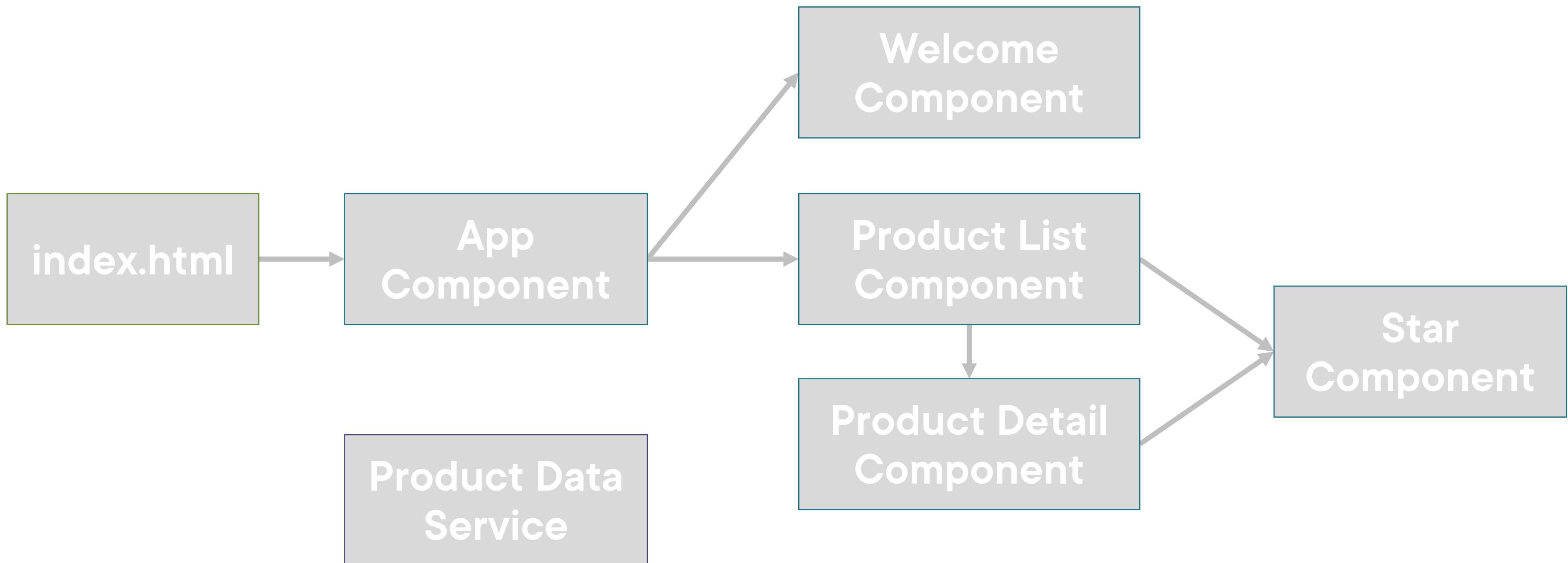
Local Angular CLI

Local Angular CLI

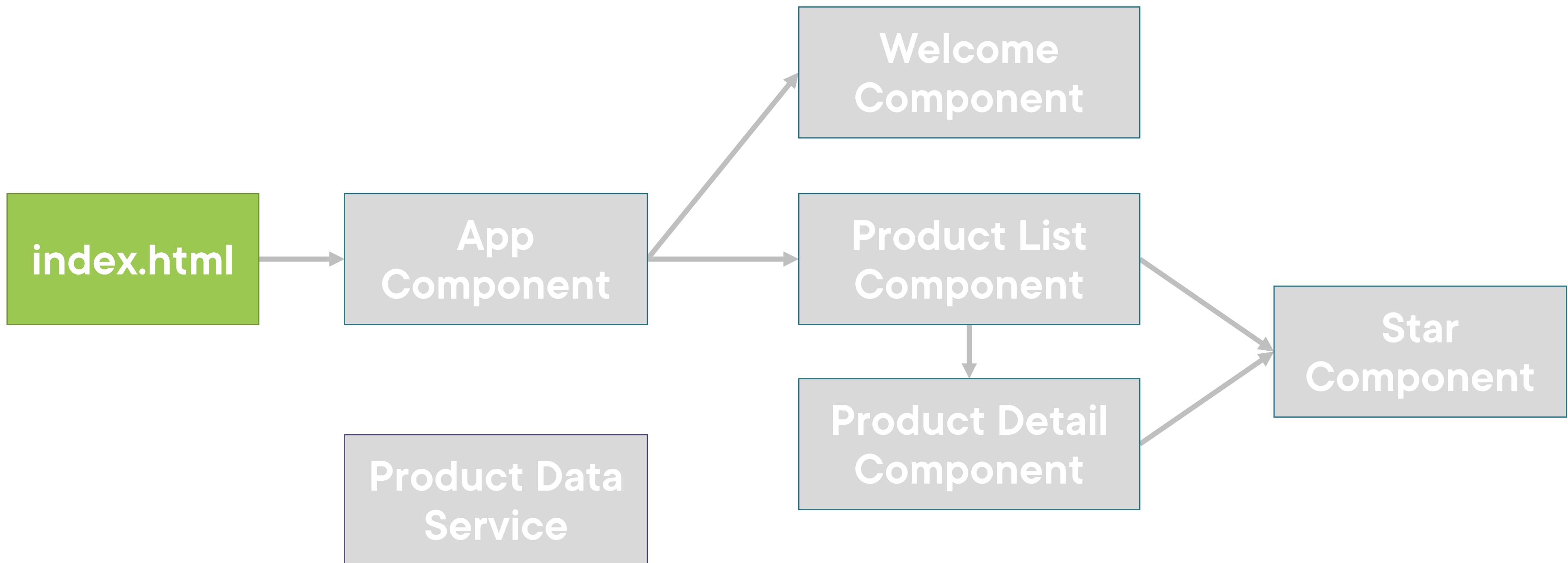
Local Angular CLI



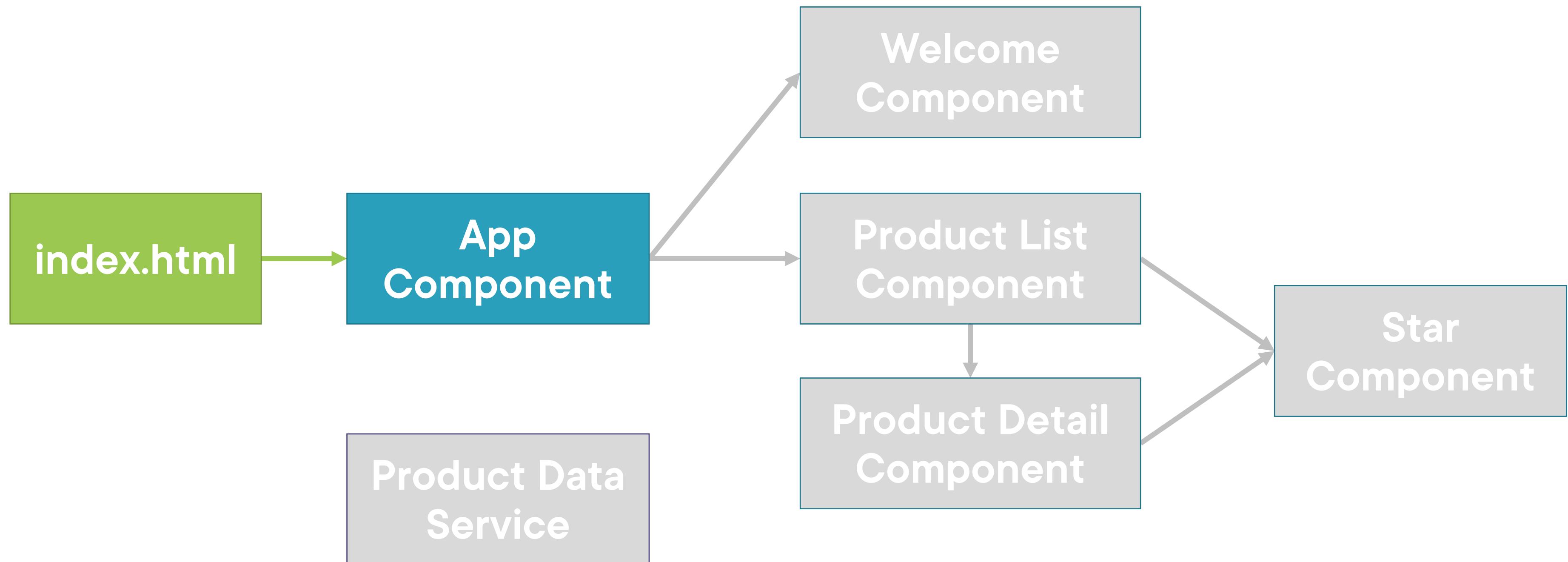
Application Architecture



Application Architecture



Application Architecture





Coming up next ...

Introduction to Components

Introduction to Components



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview



What is a component?

Creating the component class

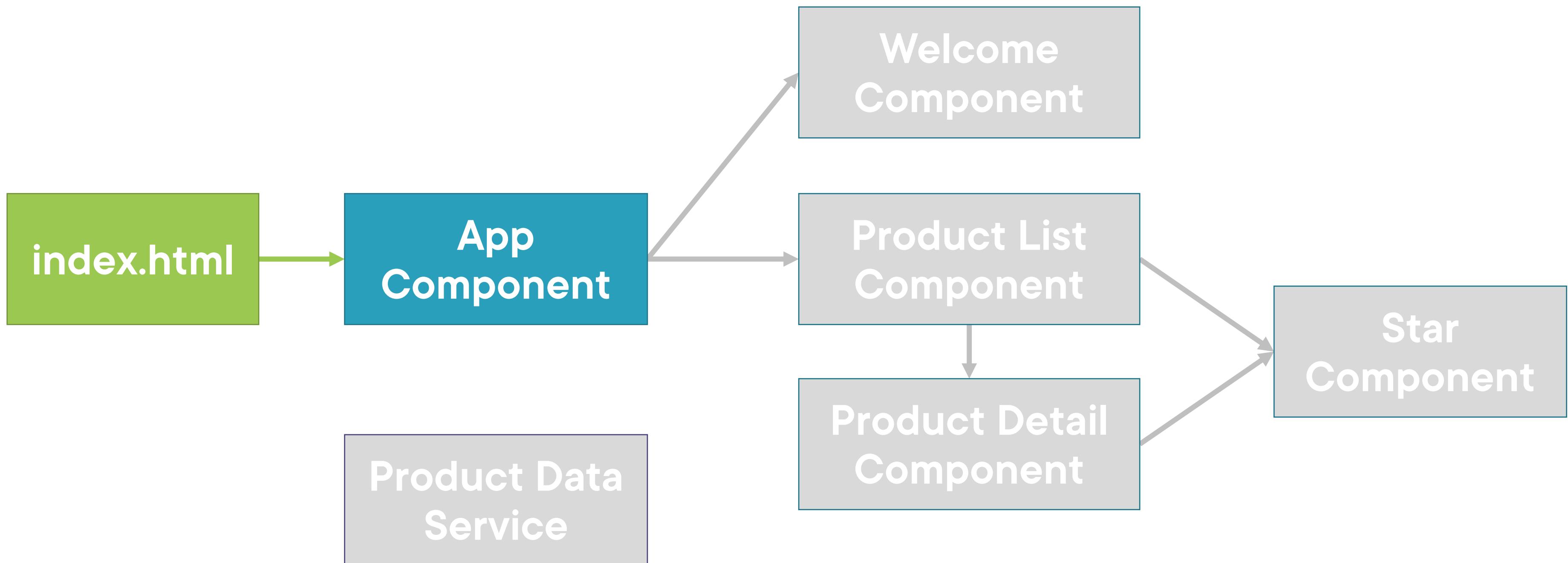
Defining the metadata with a decorator

Importing what we need

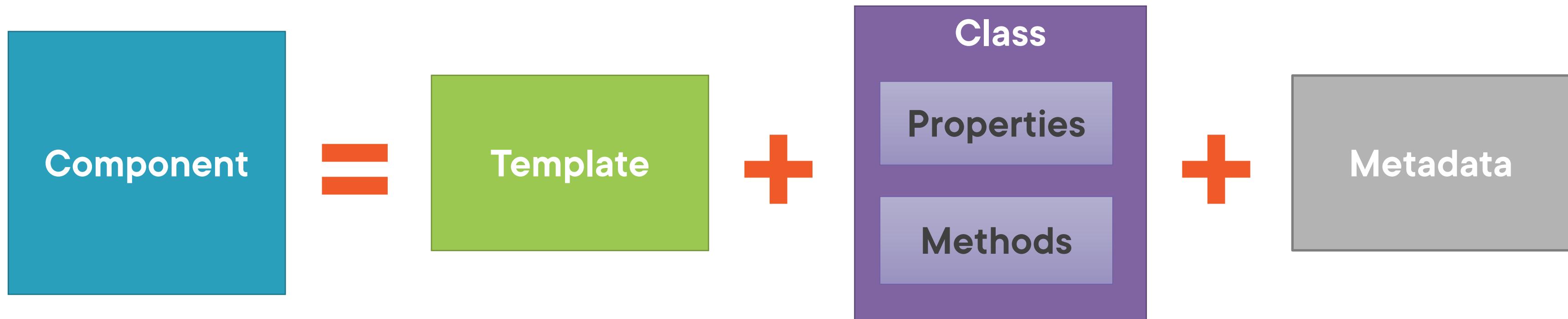
Bootstrapping our app component

Something's wrong!

Application Architecture



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';
```

Import

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

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

Metadata &
Template

Class

Creating the Component Class

app.component.ts

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

class
keyword

Class Name

export
keyword

Component Name
when used in code

Creating the Component Class

app.component.ts

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

Property
Name

Data Type

Default Value

Methods

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



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

import statement

Allows us to use exported members from:

- Other files in our application**
- Angular framework**
- External JavaScript libraries**

Importing What We Need

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';
}
```

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 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';
}
```

Demo



Creating the App component

Bootstrapping Our App Component

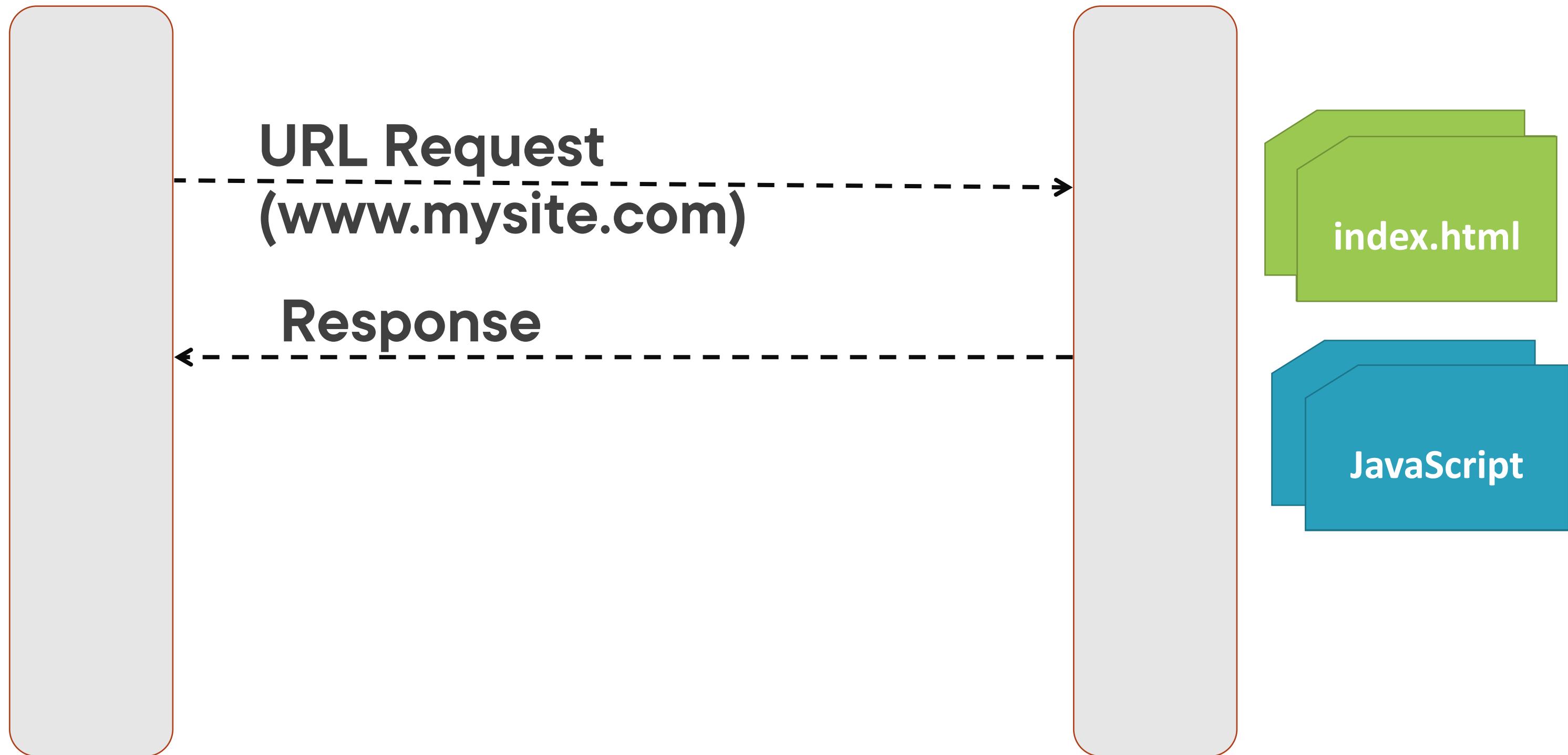


Host the application
Defining the Angular module



Web Browser

Web Server



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

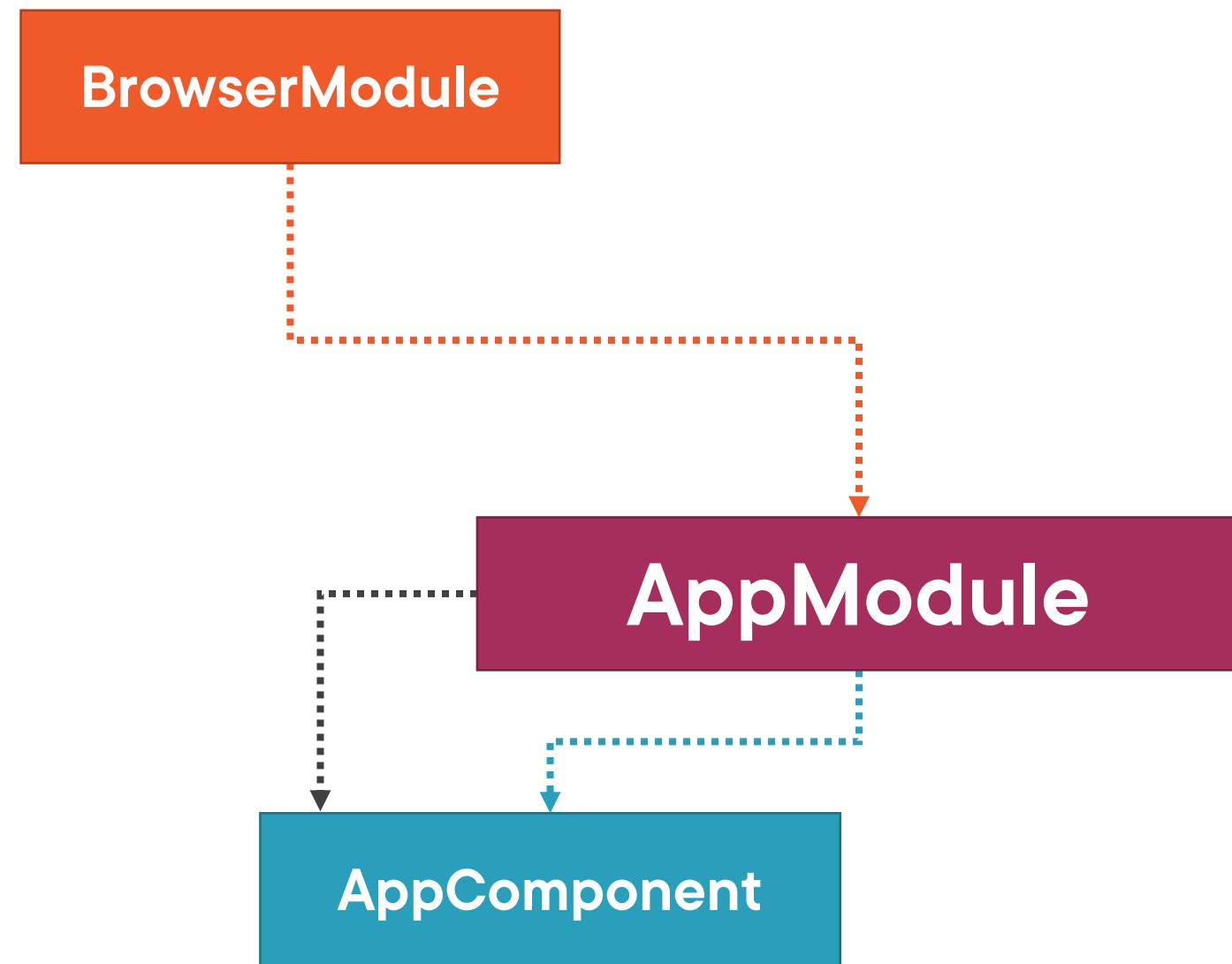
index.html

```
<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';
}
```



Organization
Boundaries
Template resolution
environment

- Imports
- Exports
- Declarations
- Bootstrap

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 { }
```

Demo



Bootstrapping our App component

Angular compiles our
HTML templates and
TypeScript components
to JavaScript

Something's Wrong!



A screenshot of a code editor's terminal tab. It shows a build log with a timestamp, hash, and time taken. Below that is an error message from TypeScript: "Error: src/app/app.component.ts:12:14 - error TS2304: Cannot find name 'strin'." A red box highlights the misspelled variable name "strin".

```
export class AppComponent {  
  pageTitle = 'Acme PM';  
  PageTitle = 'Something else';  
}  
  
12  pageTitle: strin = 'Acme Product Management';  
~~~~~
```

**Compiler displays
syntax errors**

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



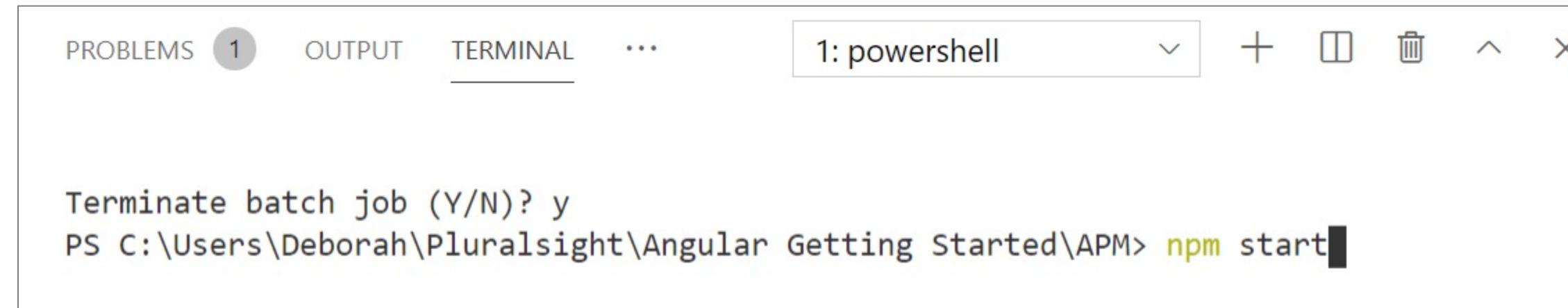
The screenshot shows a code editor window for `app.component.ts`. The code contains a syntax error: `@Component({ })`. The word `Component` is underlined with a red squiggle, indicating a spelling mistake or a reference to an undefined symbol. A tooltip appears over the word, stating "Cannot find name 'Component'. ts(2304)". Below the code, there are two options: "Peek Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".

Open the
terminal



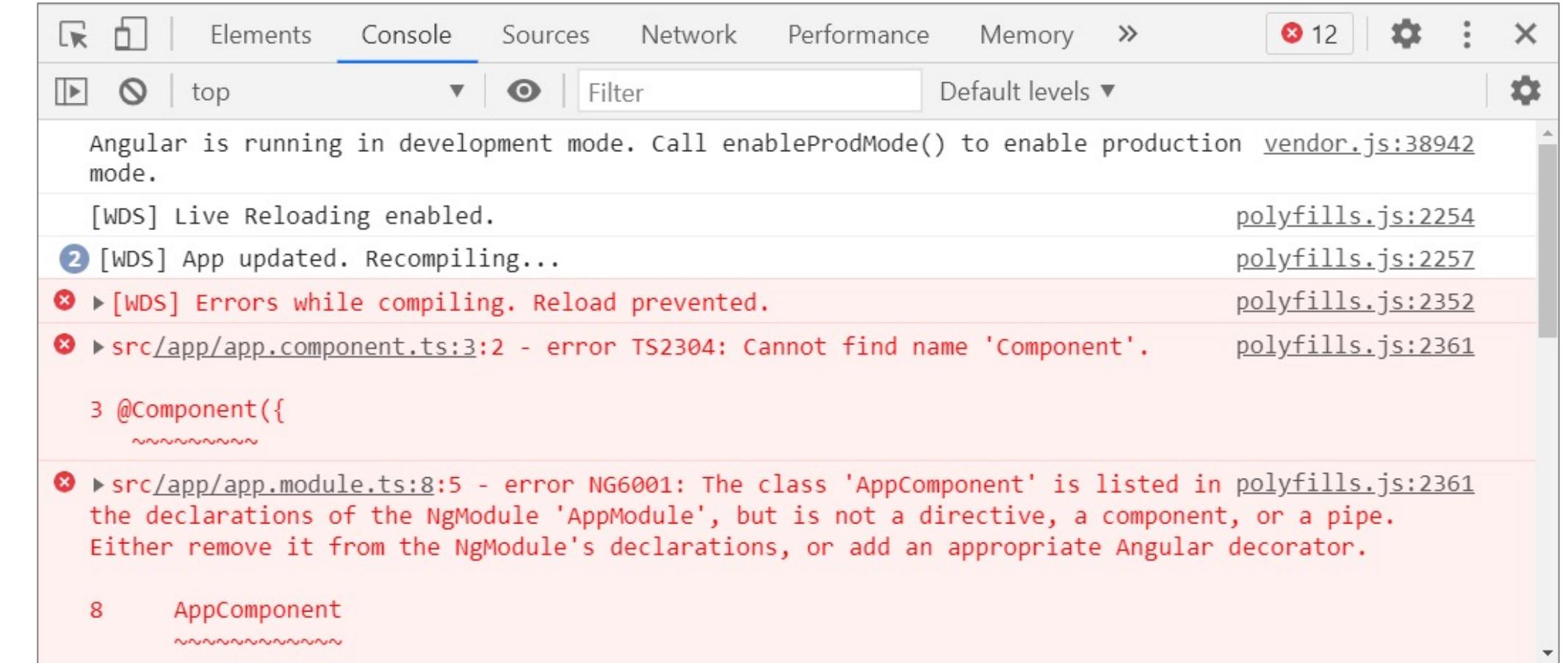
The screenshot shows the VS Code terminal tab. It displays a single error message: "Error: src/app/app.component.ts:1:2 - error TS2304: Cannot find name 'Component'." Below the error message, the first few lines of the component code are shown, with the problematic `Component` identifier underlined with a red squiggle.

Stop (Ctrl+C)
and Restart



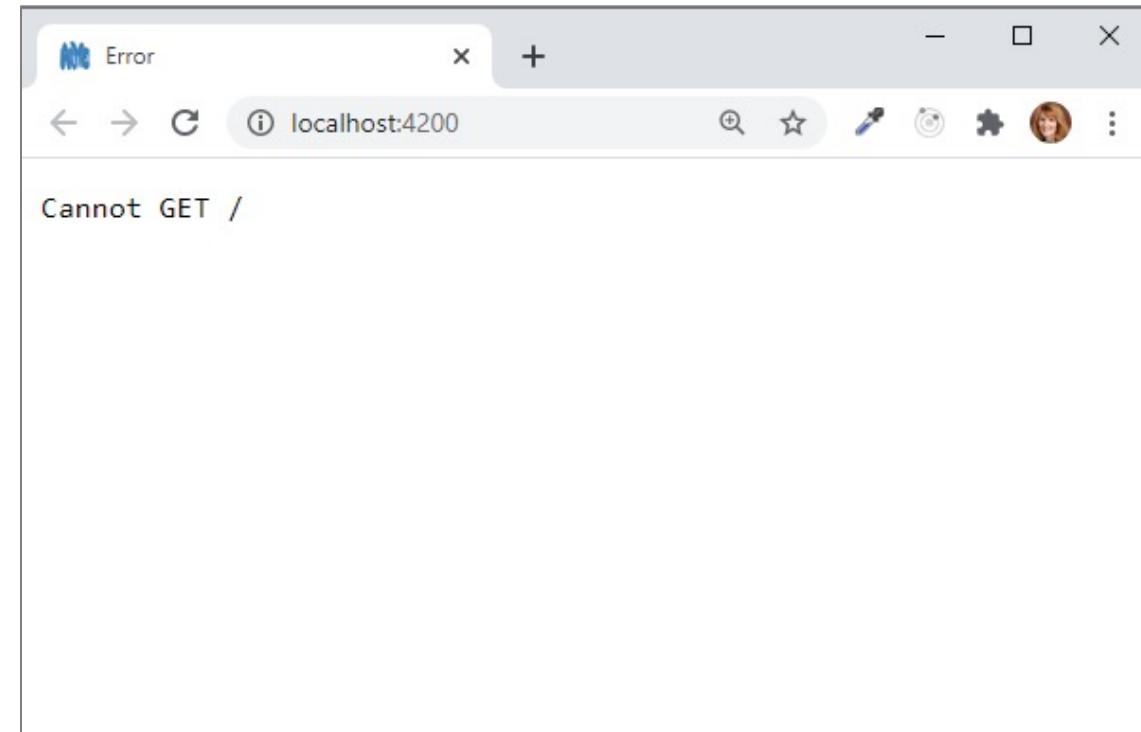
The screenshot shows the VS Code terminal tab. It displays the command `npm start` being run in a PowerShell session. The output shows the terminal prompt "PS C:\Users\Deborah\Pluralsight\Angular Getting Started\APM>" followed by the command `npm start`. A question mark at the end of the command indicates it is waiting for user input.

Use the browser's developer tools



The screenshot shows the Chrome DevTools Console tab. The tabs at the top are Elements, Console (which is selected), Sources, Network, Performance, and Memory. There are 12 errors indicated by a red circle with the number 12. The console output includes:

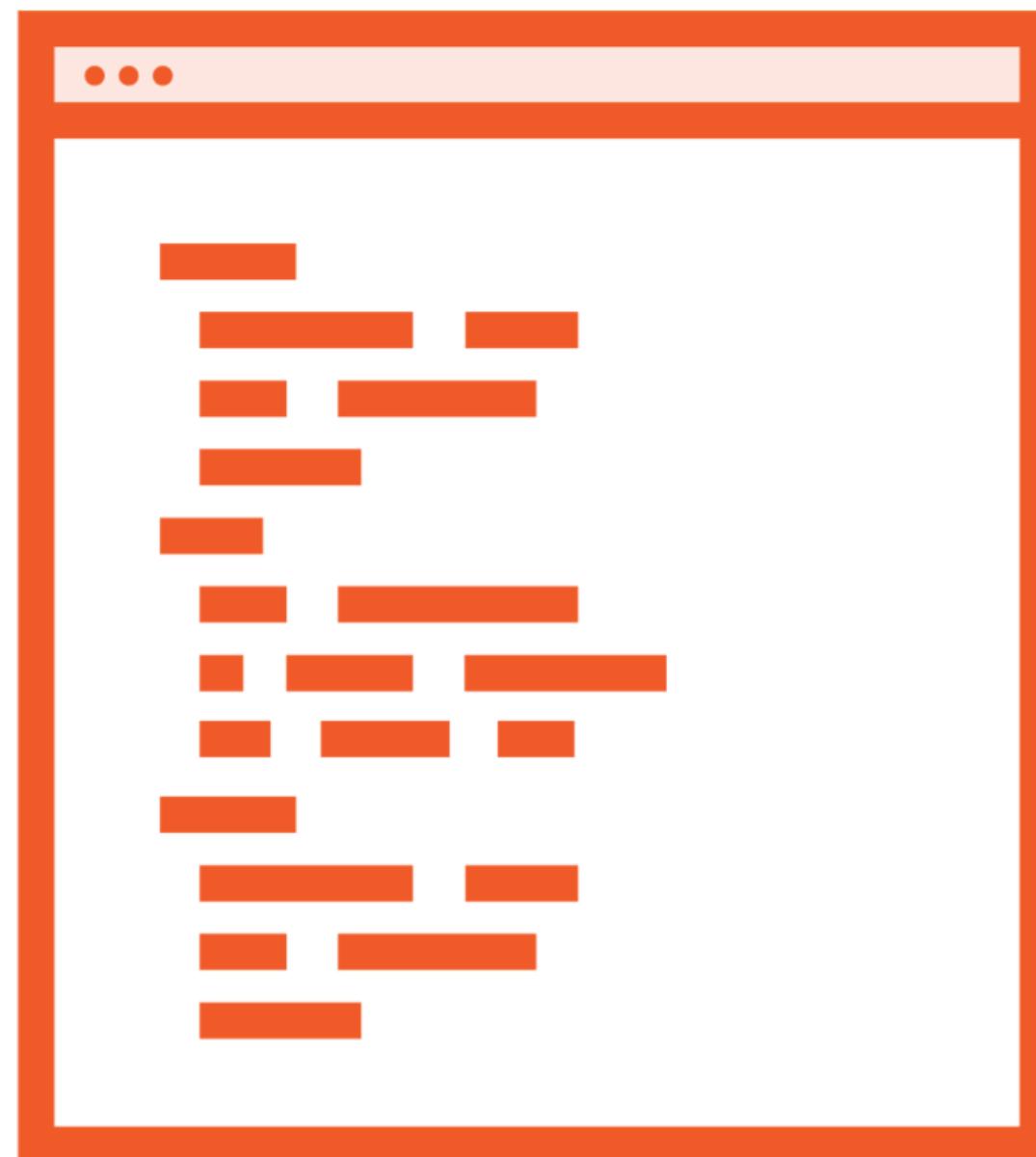
- Angular is running in development mode. Call enableProdMode() to enable production vendor.js:38942 mode.
- [WDS] Live Reloading enabled. polyfills.js:2254
- 2 [WDS] App updated. Recompiling... polyfills.js:2257
- ✖ ▶ [WDS] Errors while compiling. Reload prevented. polyfills.js:2352
- ✖ ▶ src/app/app.component.ts:3:2 - error TS2304: Cannot find name 'Component'. polyfills.js:2361
- 3 @Component({
~~~~~}
- ✖ ▶ src/app/app.module.ts:8:5 - error NG6001: The class 'AppComponent' is listed in polyfills.js:2361 the declarations of the NgModule 'AppModule', but is not a directive, a component, or a pipe. Either remove it from the NgModule's declarations, or add an appropriate Angular decorator.
- 8 AppComponent  
~~~~~



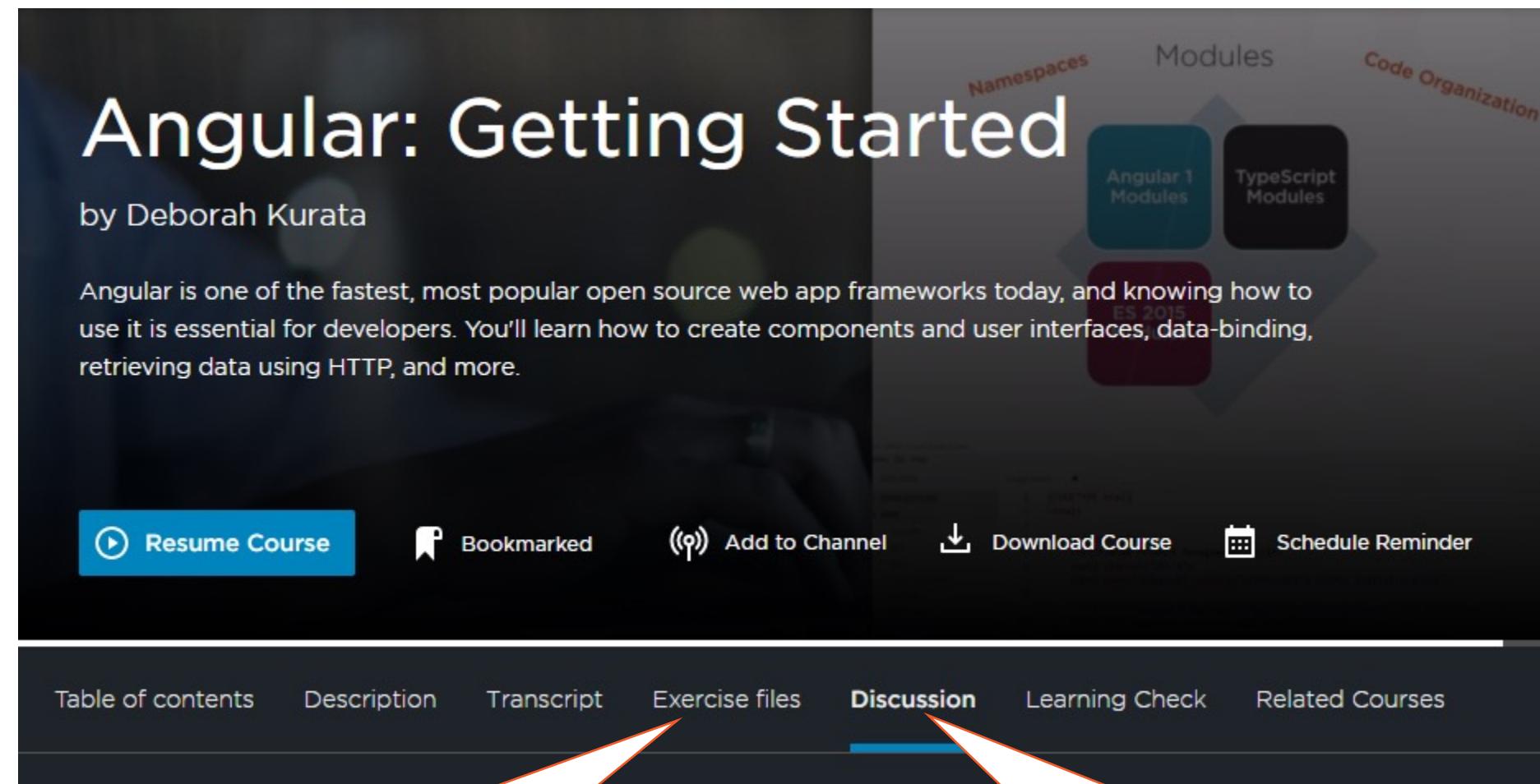
Often means an error prevented the compiler from completing

Use the VS Code terminal to view the errors

Recheck Your Code



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



- Code “as of” the end of each module
- Copy of the slides

- Review posted issues
- Post your own issues

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;  
  }  
}
```

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

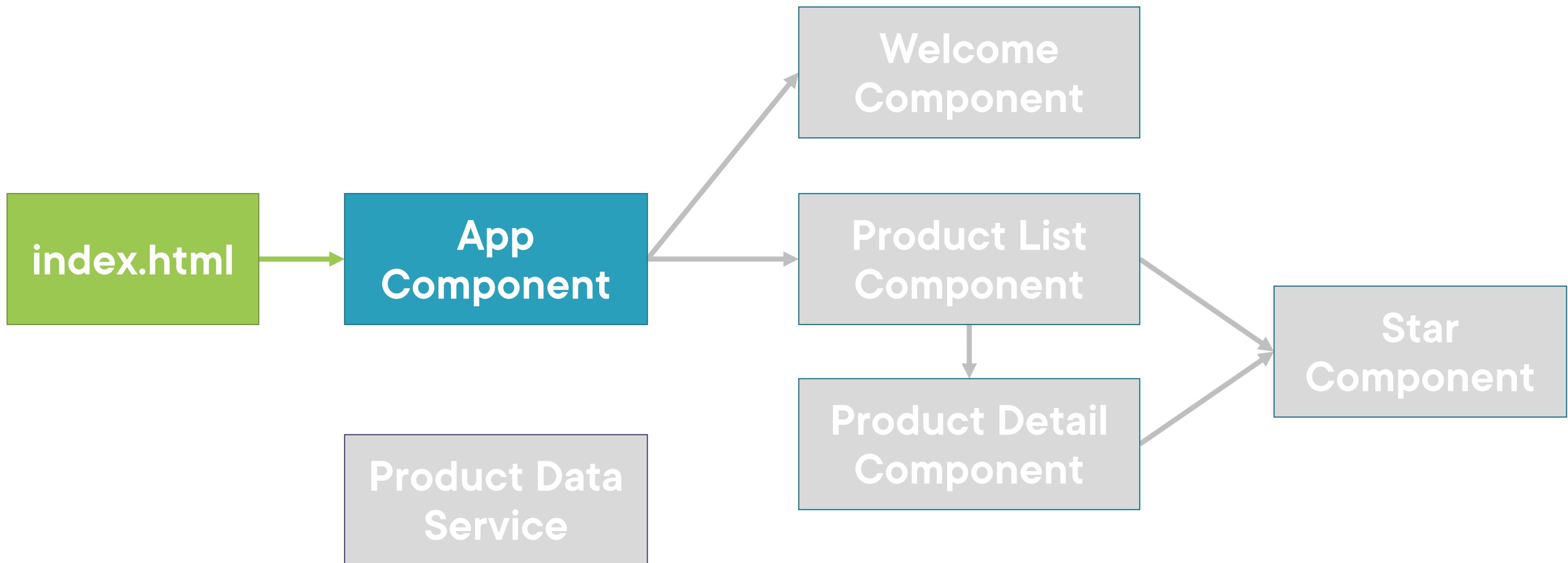
- **Correct spelling/casing**

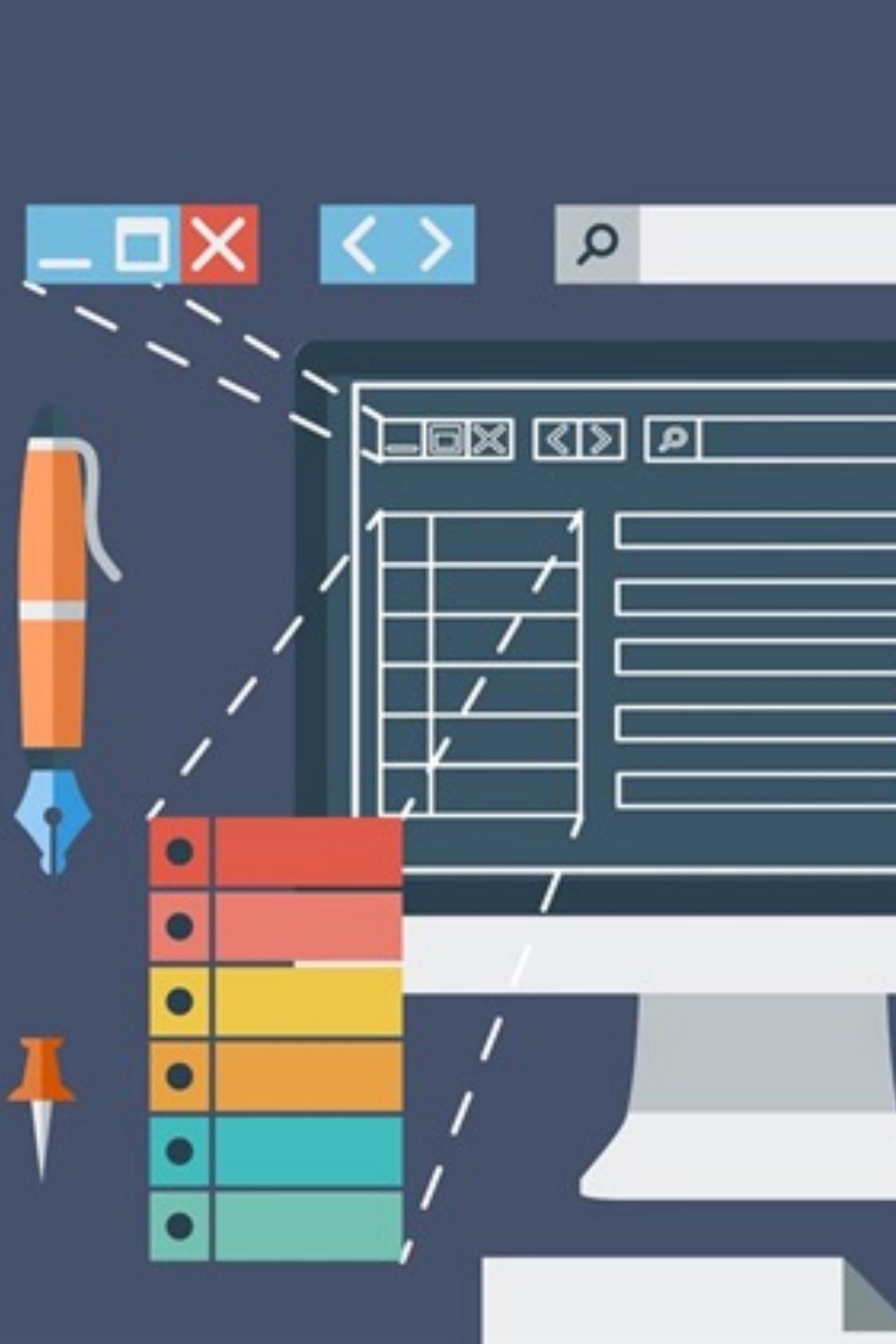
Path

- **Enclose in quotes**
- **Correct spelling/casing**

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

Application Architecture





Coming up next ...

**Templates, Interpolation, and
Directives**

Templates, Interpolation, and Directives



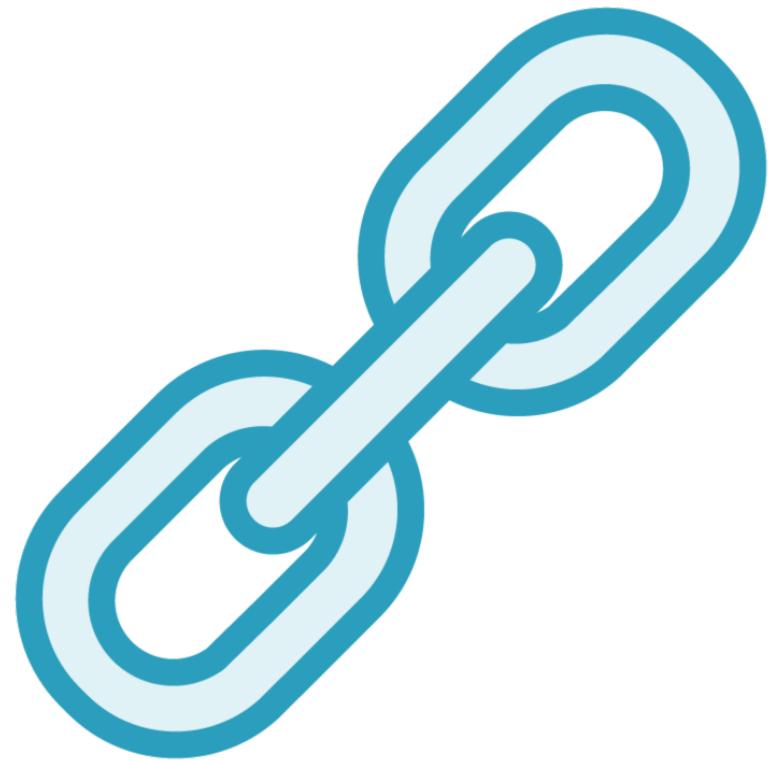
Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

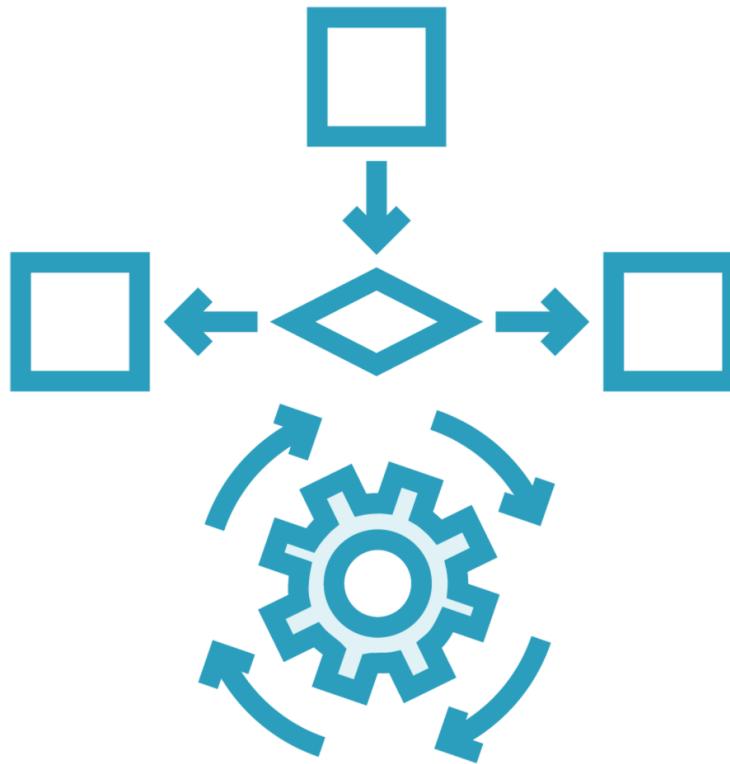
@deborahkurata



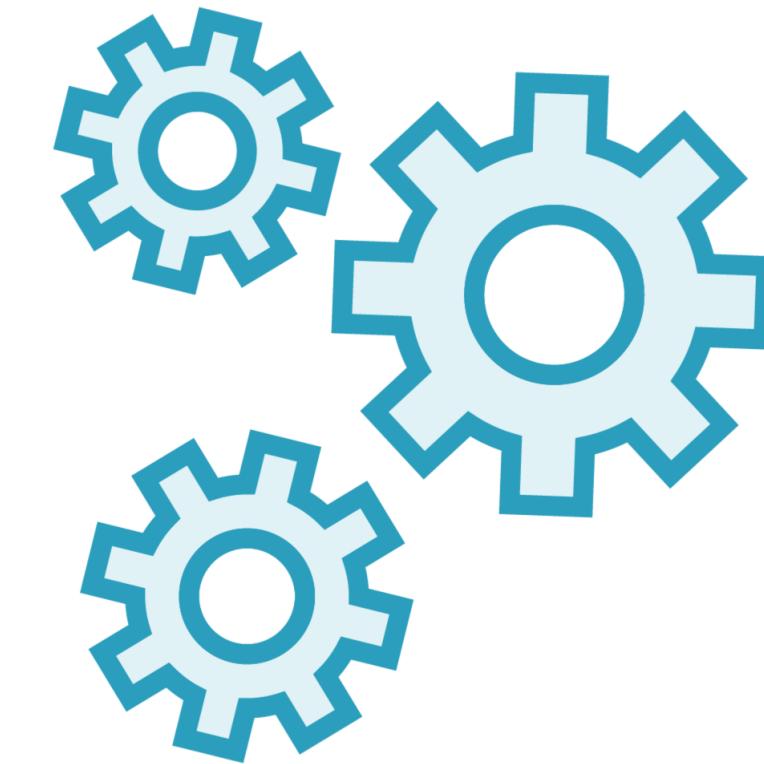
Power up HTML



Data binding

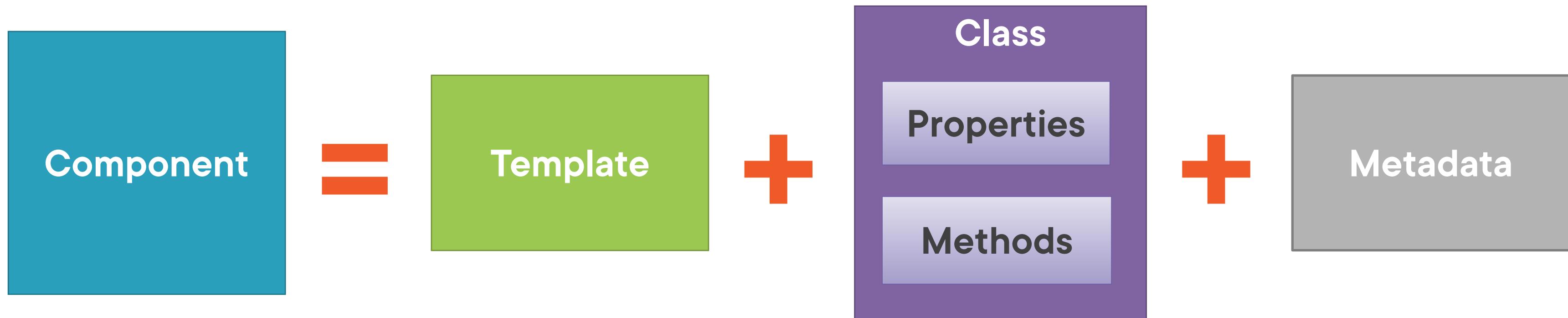


Angular directives
**(Custom HTML
syntax)**



Angular components
(Custom Directives)

Component



Module Overview



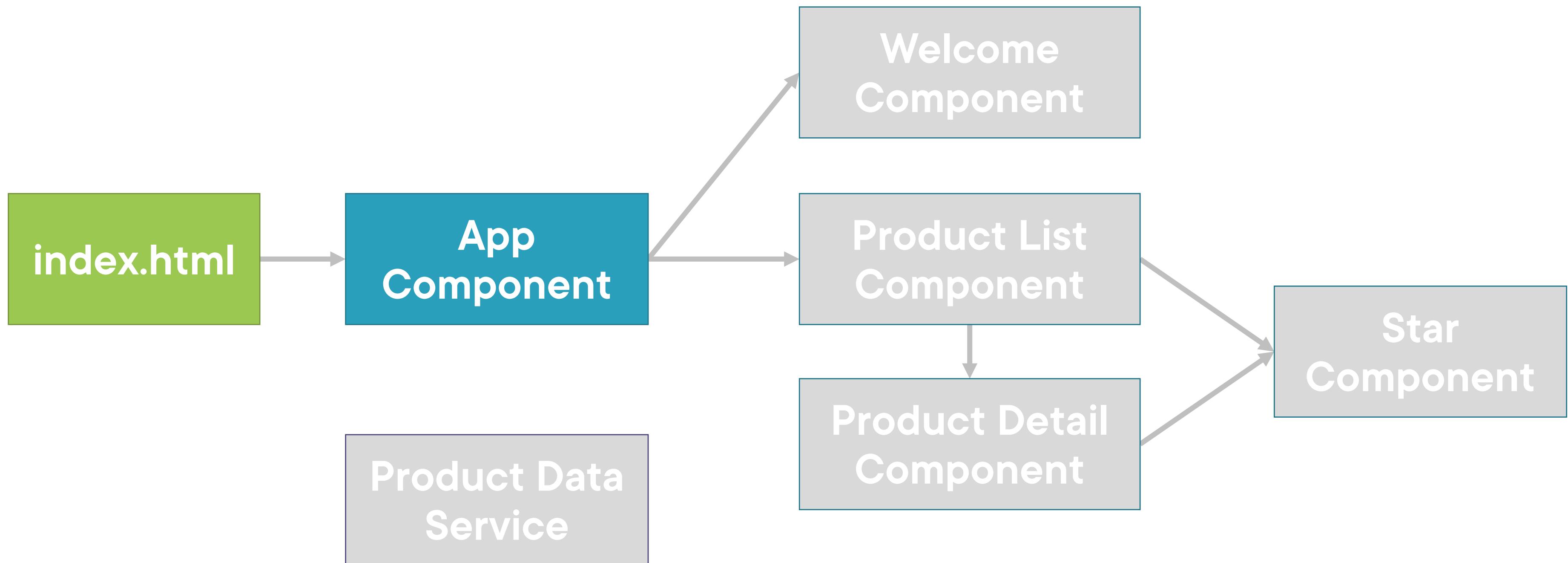
Building a template

Using a component as a directive

Binding with interpolation

Adding logic with directives

Application Architecture



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';
}
```

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>  
`'
```

Linked Template

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

ES 2015
Back Ticks

Product List View

Product List					
Filter by: <input type="text"/>					
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	
	Hammer	tbx 0048	May 21, 2021	\$8.90	
	Saw	tbx 0022	May 15, 2021	\$11.55	
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	

Product List View

Product List					
Filter by:	<input type="text" value="am"/>				
Filtered by: am					
Show Image	Product	Code	Available	Price	5 Star Rating
	Hammer	tbx 0048	May 21, 2021	\$8.90	★★★★★
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	★★★★★

Product List View

Product List					
Filter by:	<input type="text" value="am"/>				
Filtered by: am					
<button>Hide Image</button>	Product	Code	Available	Price	5 Star Rating
	Hammer	tbx 0048	May 21, 2021	\$8.90	
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	

Product List View

Product List					
Filter by: <input type="text"/>					
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	
	Hammer	tbx 0048	May 21, 2021	\$8.90	
	Saw	tbx 0022	May 15, 2021	\$11.55	
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	

<https://getbootstrap.com/>

<https://fontawesome.com>

Building the Component

product-list.component.ts

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

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

Using a Component as a Directive

app.component.ts

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

product-list.component.ts

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

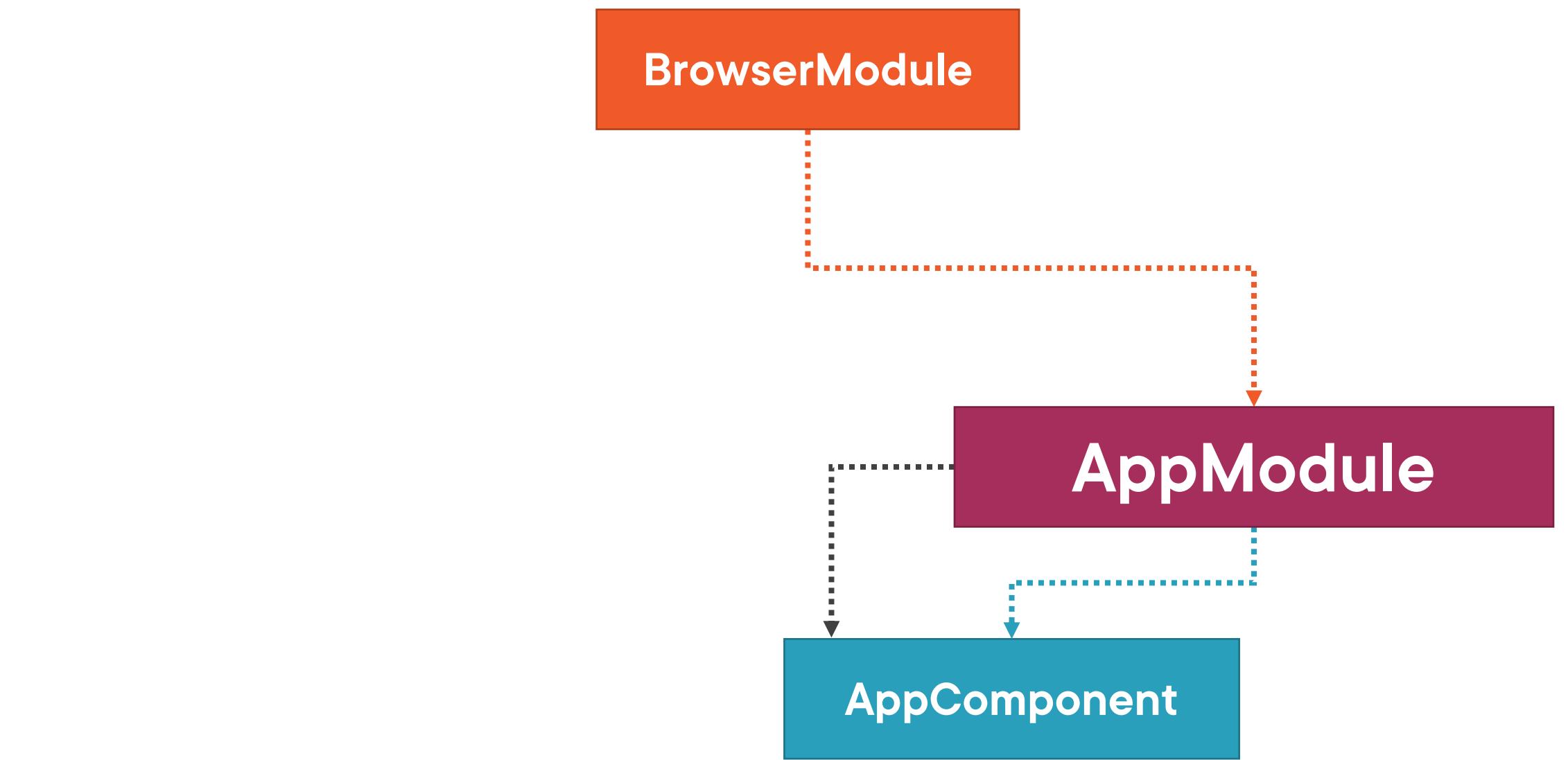
Using a Component as a Directive

app.component.ts

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

product-list.component.ts

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

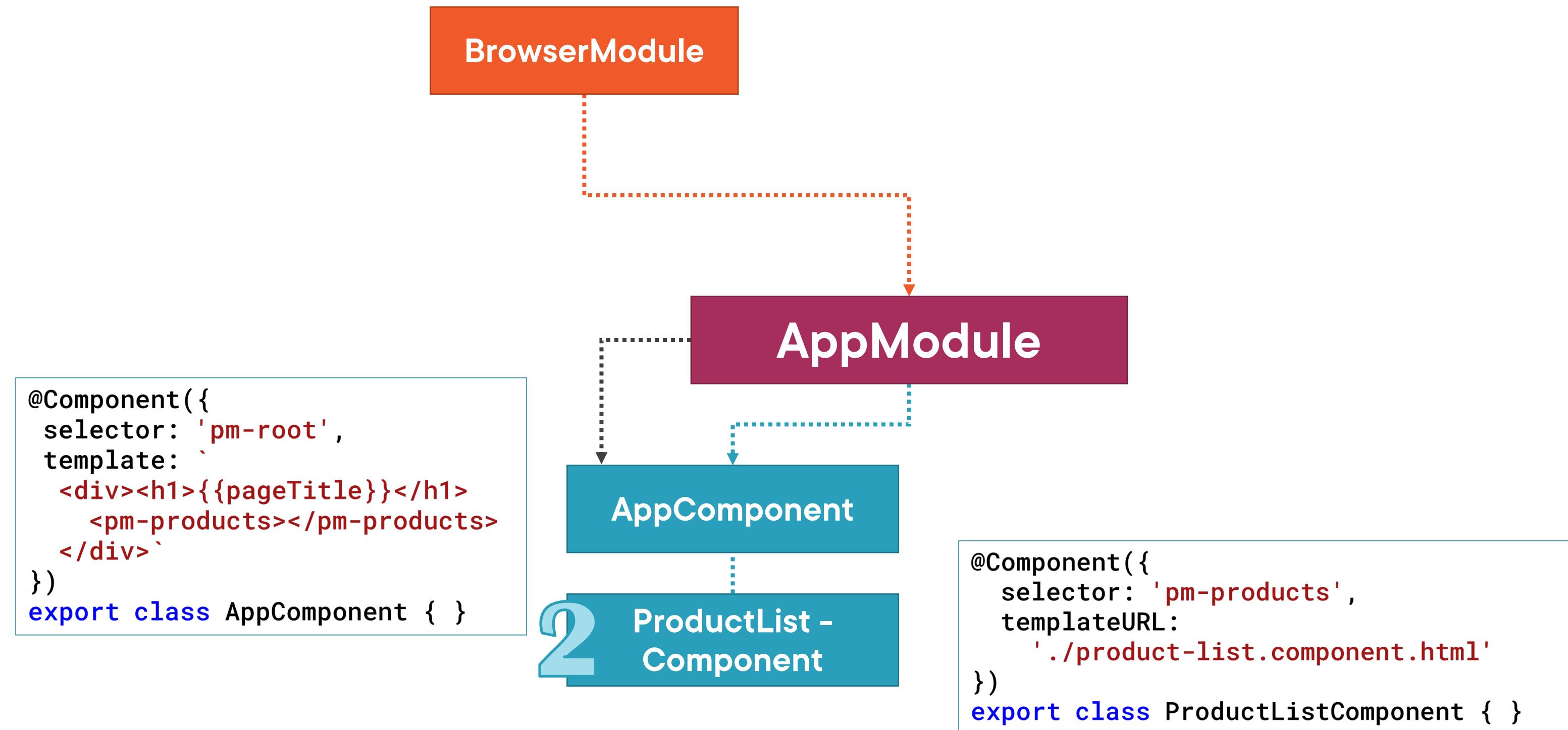


..... **Imports**

..... **Exports**

..... **Declarations**

..... **Bootstrap**



..... Imports

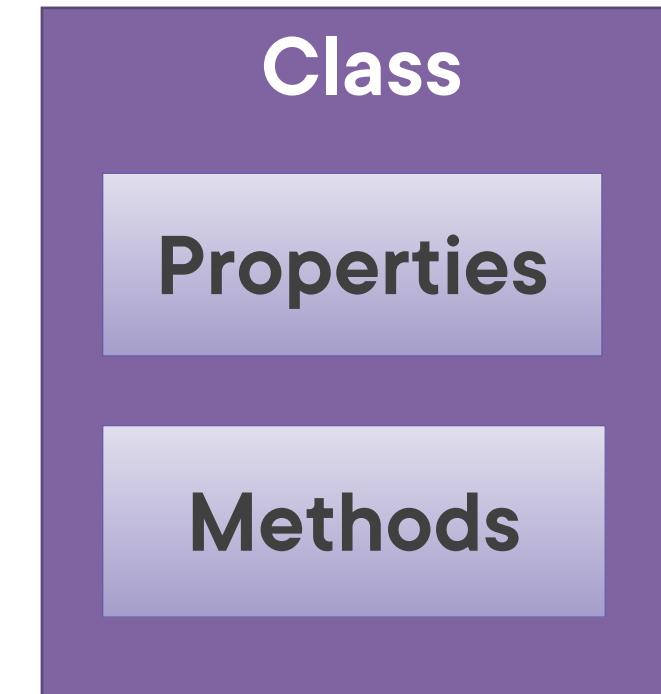
..... Exports

..... Declarations

..... Bootstrap

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';
}
```

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

product-list.component.ts

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

Angular Built-in Directives

Structural Directives



`*ngIf: If logic`

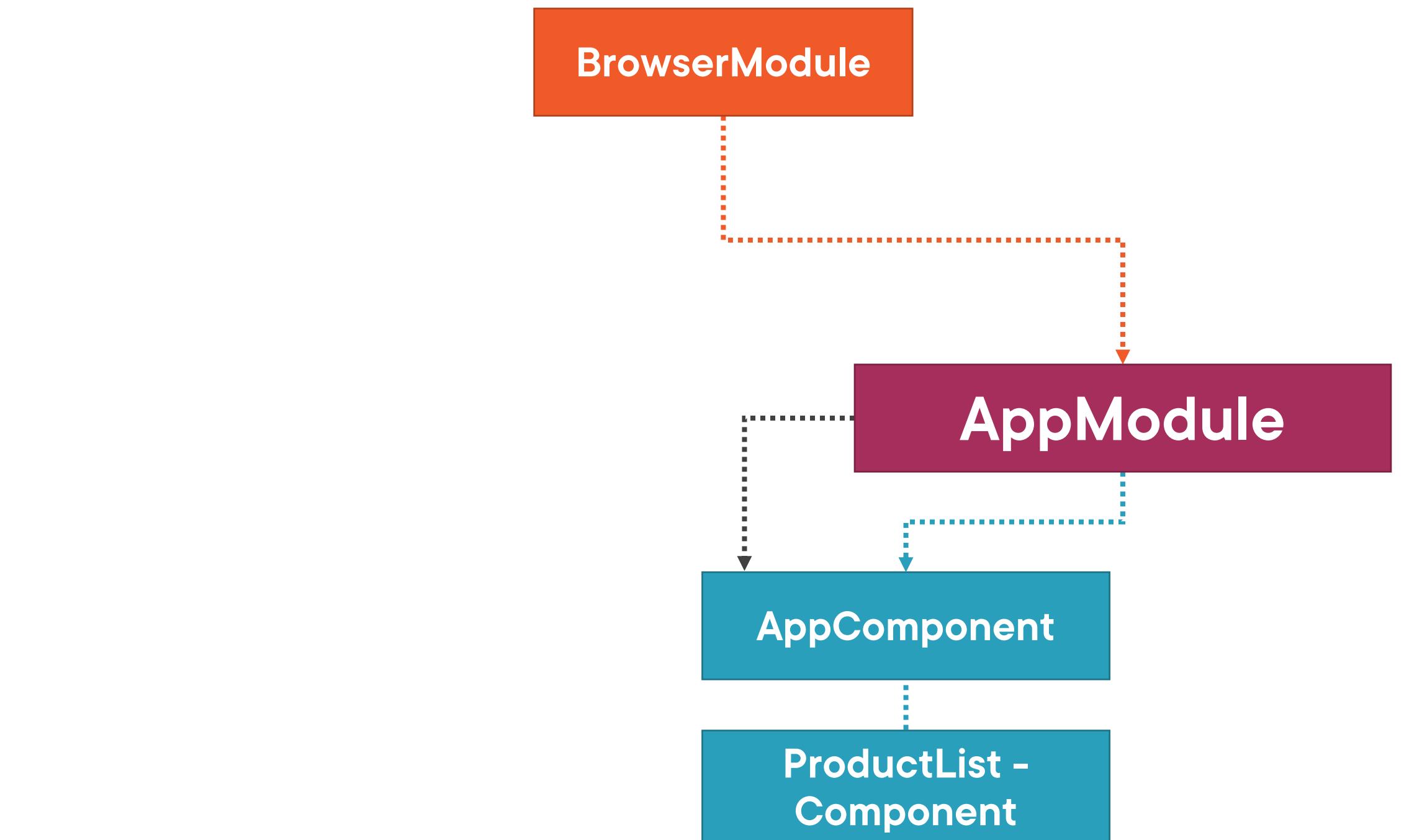
`*ngFor: For loops`

*ngIf Built-In Directive

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

Product List					
Filter by: <input type="text"/>					
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2016	\$19.95	★★★
	Garden Cart	gdn 0023	March 18, 2016	\$32.99	★★★★
	Hammer	tbx 0048	May 21, 2016	\$8.90	★★★★★
	Saw	tbx 0022	May 15, 2016	\$11.55	★★★★
	Video Game Controller	gmg 0042	October 15, 2015	\$35.95	★★★★★

Product List					
Filter by: <input type="text"/>					



..... Imports

..... Exports

..... Declarations

..... Bootstrap

*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);
}
```

*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 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

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

app.module.ts

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

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

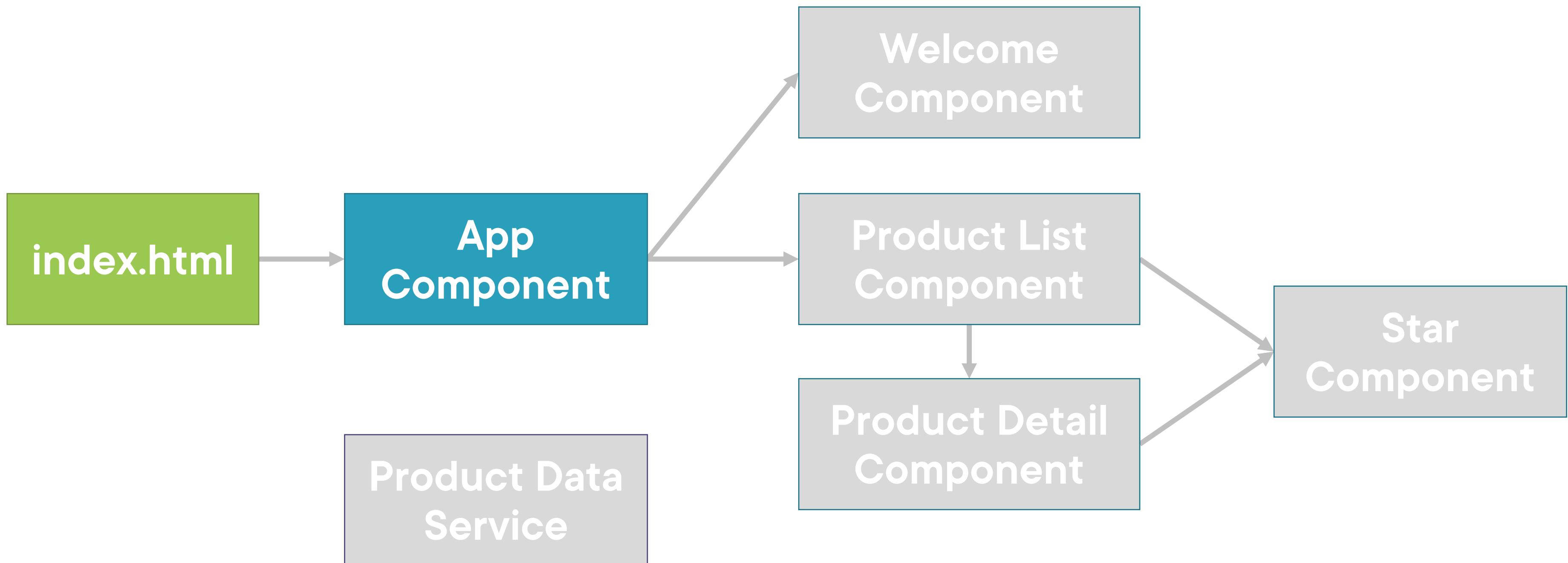
- For each object in an iterable list

Define the local variable with let

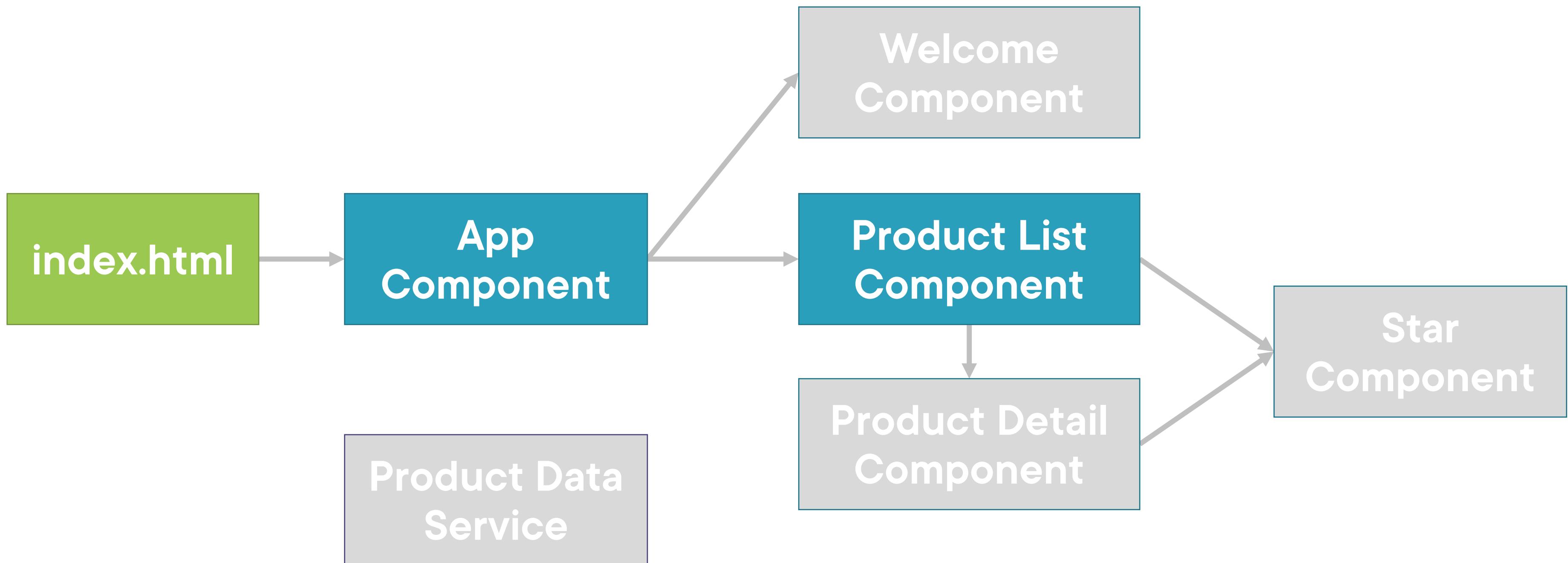
Specify 'of'

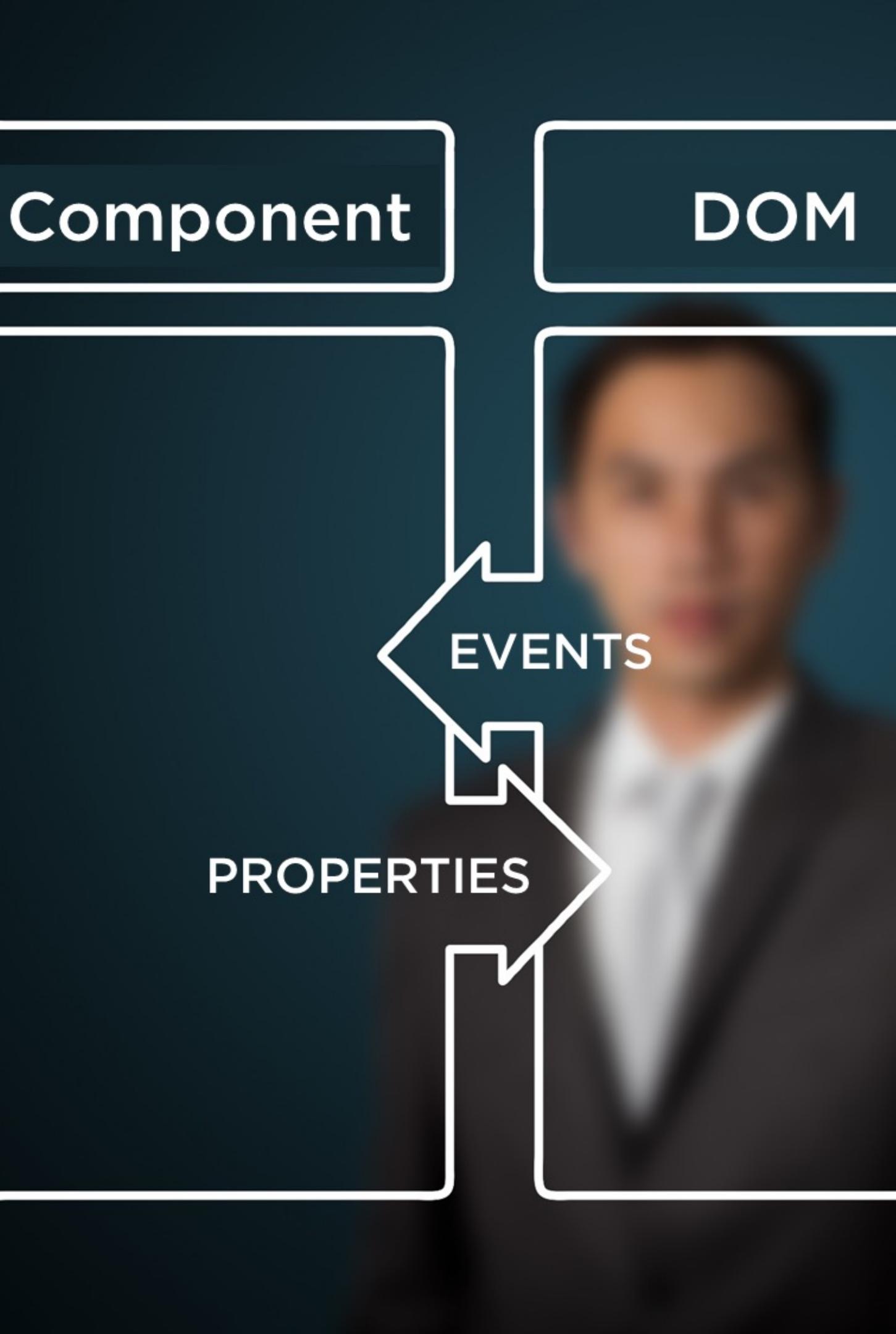
```
<tr *ngFor='let product of products'>  
  ...  
</tr>
```

Application Architecture



Application Architecture





Coming up next ...

Data Binding & Pipes

Data Binding & Pipes



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata

Component

DOM

EVENTS

PROPERTIES



Module Overview



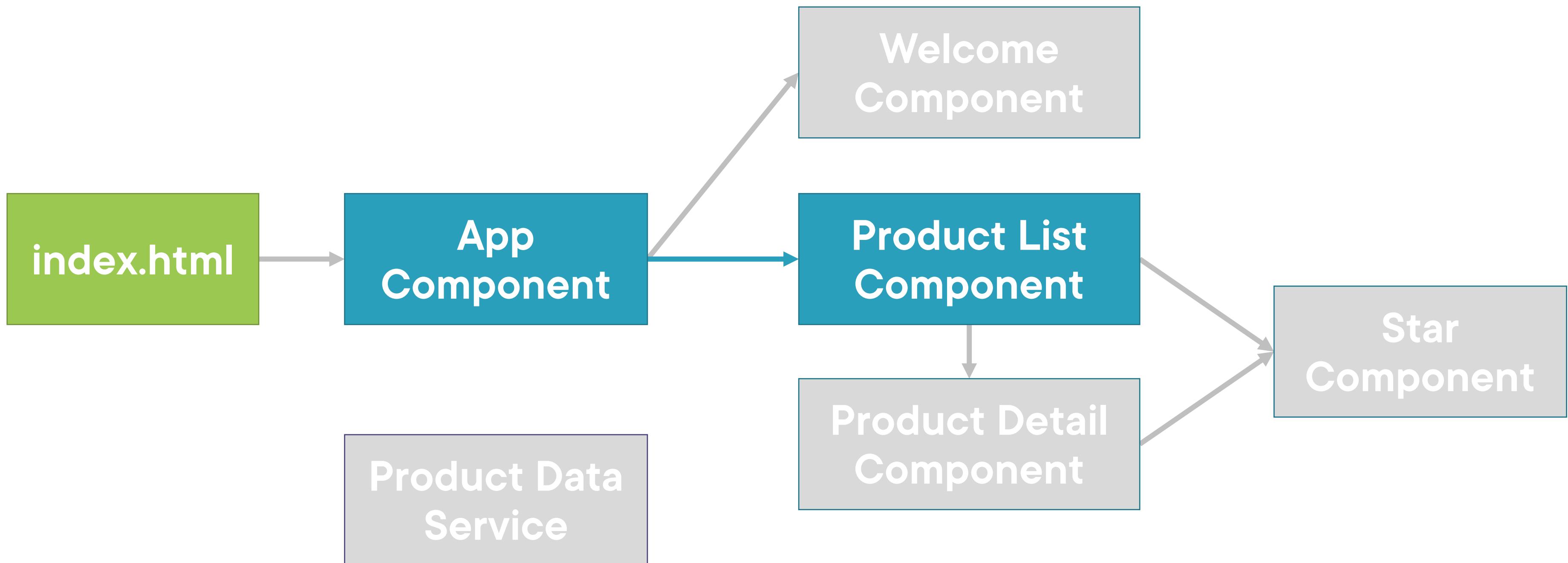
Property binding

Handling events with event binding

Handling input with two-way binding

Transforming data with pipes

Application Architecture



Property Binding

Element Property

Template Expression

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

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

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

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

Property Binding

Template

```
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
```

Component

```
export class ListComponent {
  pageTitle: string = 'Product List';
  products: any[] = [
    {
      productId: 2,
      productName: "Garden Cart",
      productCode: "GDN-0023",
      releaseDate: "March 18, 2021",
      description: "Rolling garden cart",
      price: 32.99,
      starRating: 4.2,
      imageUrl: "assets/images/cart.png"
    }, ...
  ];
}
```

Event Binding

Template

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

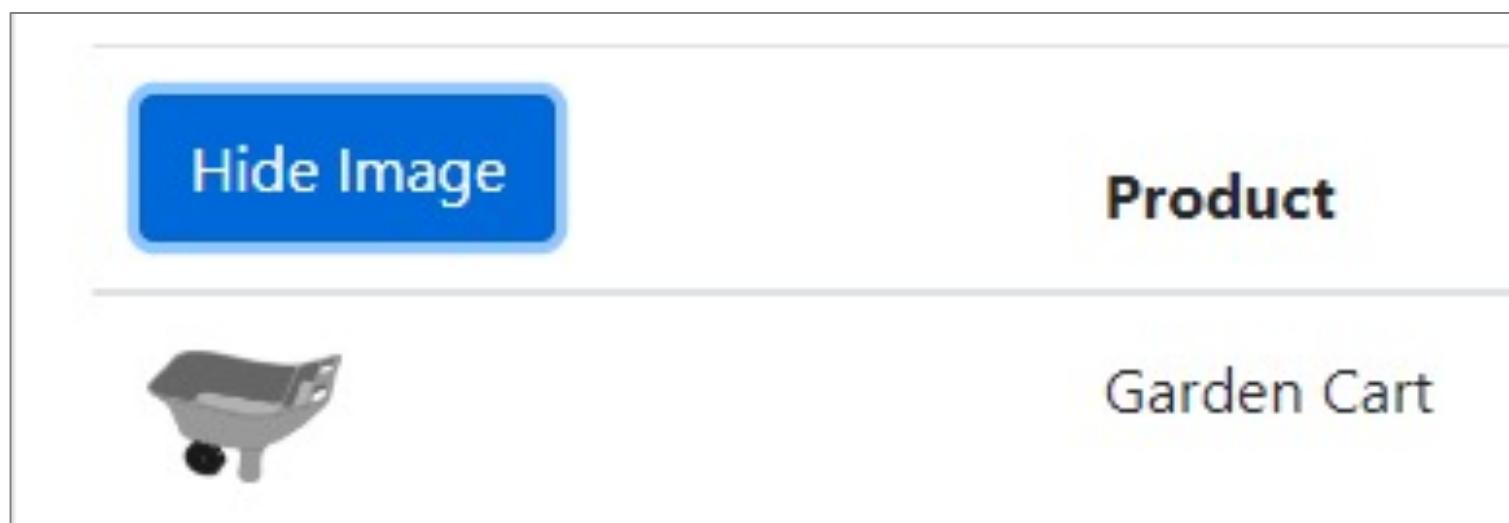
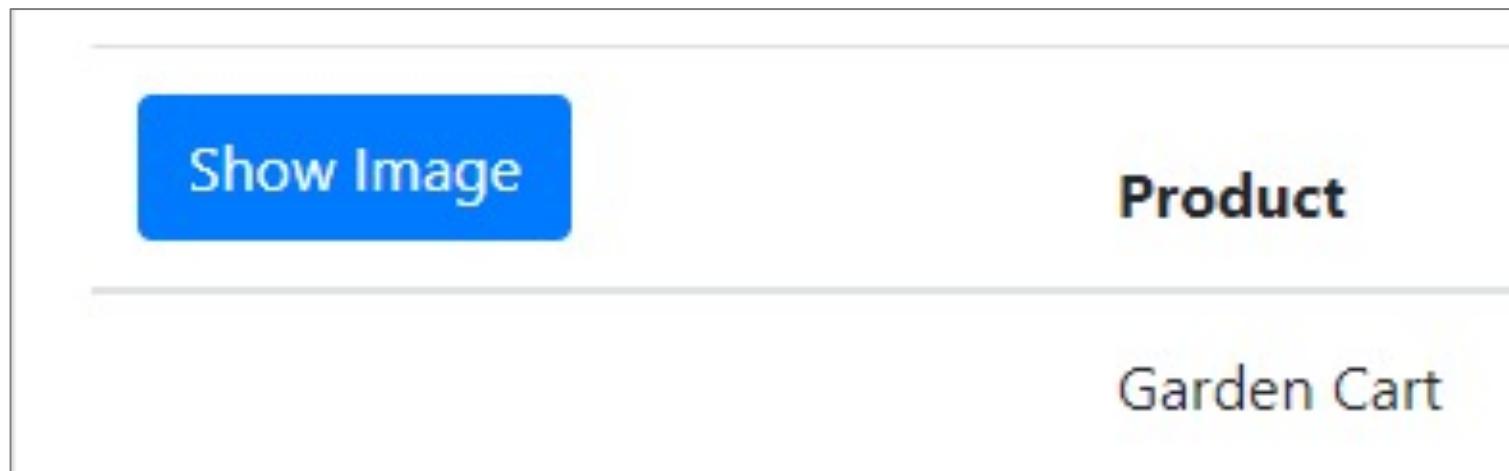
Event

Component

```
export class ListComponent {
  pageTitle: string = 'Product List';
  products: any[] = [...];
}
```

Method

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' />
</div>
```

Filter by:

rake

Component

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

```
10 export class ProductListComponent
11   "rake" :le = 'Product List';
12
13   listFilter = 'cart'
14
```

Two-way Binding

Template

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

Filter by:

rake

Component

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

```
10 export class ProductListComponent
11   "rake":le = 'Product List';
12
13   listFilter = 'cart'
14
```

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';
}
```

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';
}
```

Two-way Binding

Template

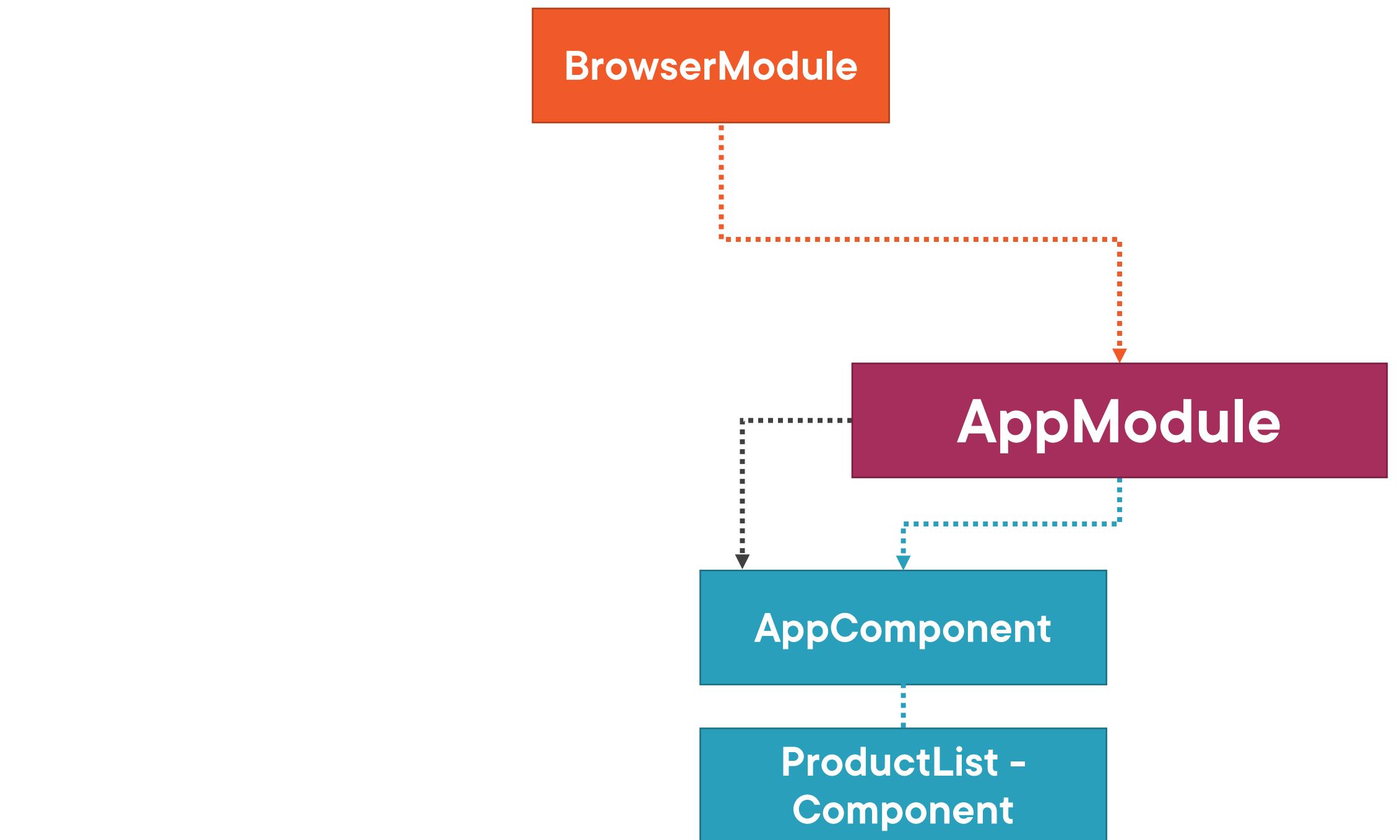
```
<div class='col-md-2'>Filter by:</div>
<div class='col-md-4'>
  <input type='text'
    [(ngModel)]=listFilter' />
</div>
```

Component

```
export classListComponent {
  listFilter: string = 'cart';
}
```

[()]

Banana in a Box

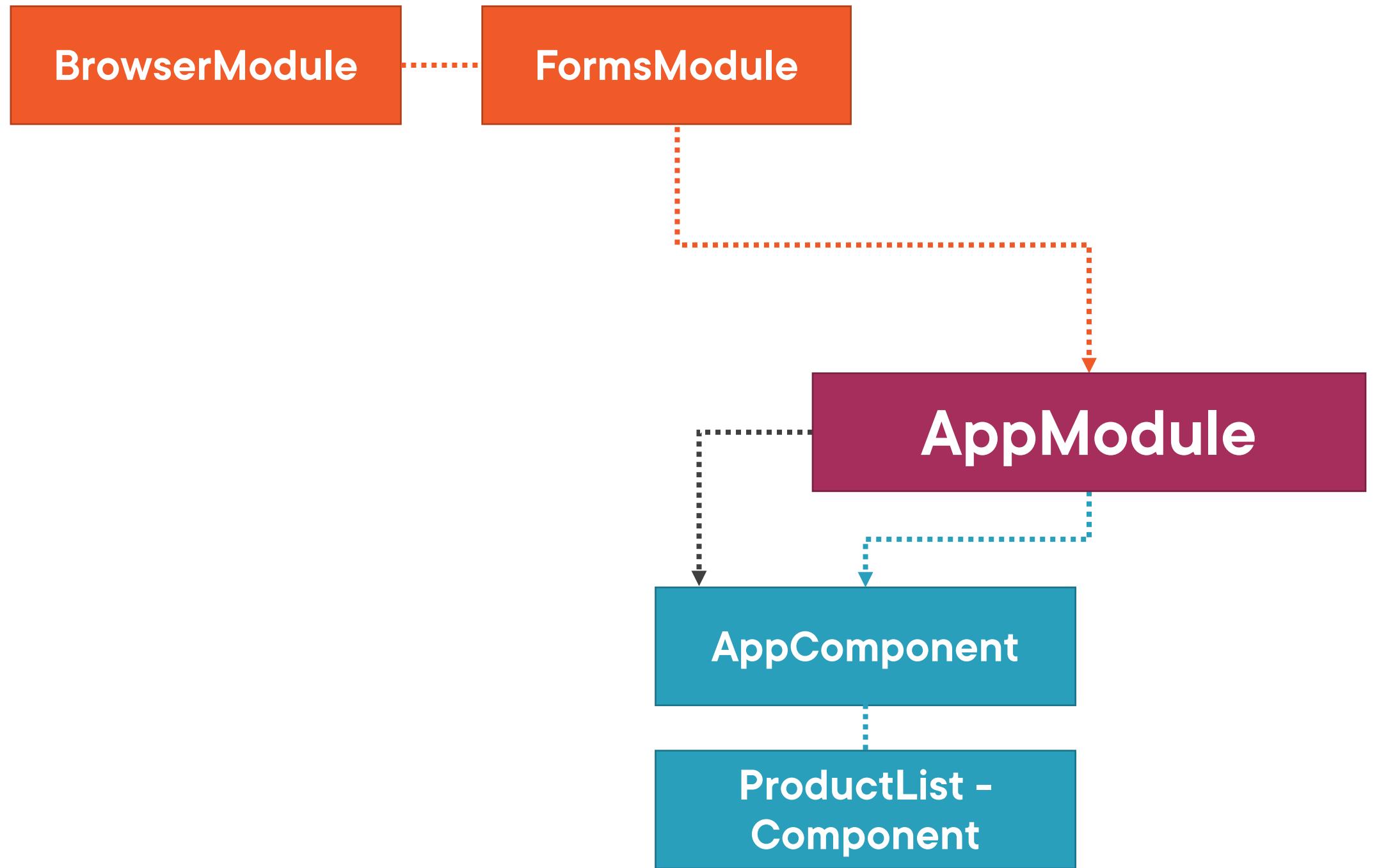


..... Imports

..... Exports

..... Declarations

..... Bootstrap



..... Imports

..... Exports

..... Declarations

..... Bootstrap

Displaying Data with Binding

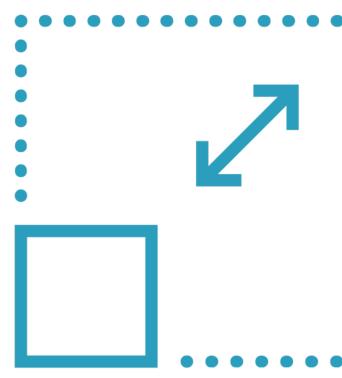
Template

```
<tbody>
  <tr *ngFor='let product of products'>
    <td>
      <img *ngIf='showImage'
          [src]='product.imageUrl'
          [title]='product.productName'
          [style.width.px]='imageWidth'
          [style.margin.px]='imageMargin'>
    </td>
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>
    <td>{{ product.releaseDate }}</td>
    <td>{{ product.price }}</td>
    <td>{{ product.starRating }}</td>
  </tr>
</tbody>
```

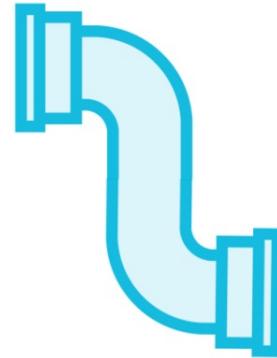
Component

```
export class ListComponent {
  pageTitle: string = 'Product List';
  imageWidth: number = 50;
  imageMargin: number = 2;
  showImage: boolean = false;
  products: any[] = [
    {
      productId: 2,
      productName: "Garden Cart",
      productCode: "GDN-0023",
      releaseDate: "March 18, 2021",
      description: "Rolling garden cart",
      price: 32.99,
      starRating: 4.2,
      imageUrl: "assets/images/cart.png"
    },
    ...
  ];
}
```

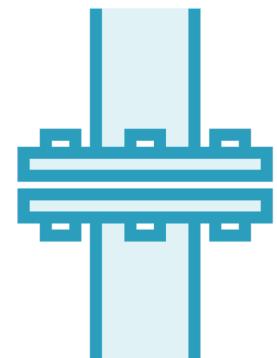
Transforming Data with Pipes



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

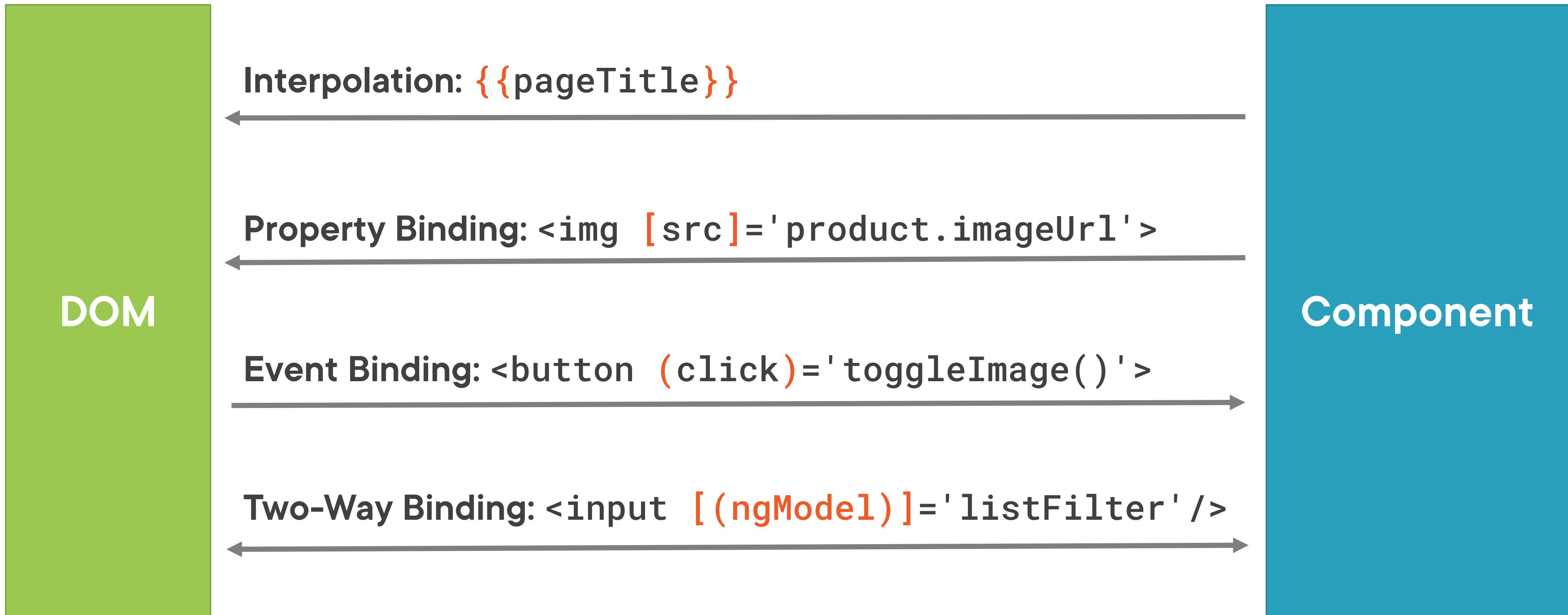
Template

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

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 { }
```

Data Binding Checklist: Pipes



Pipe character |

Pipe name

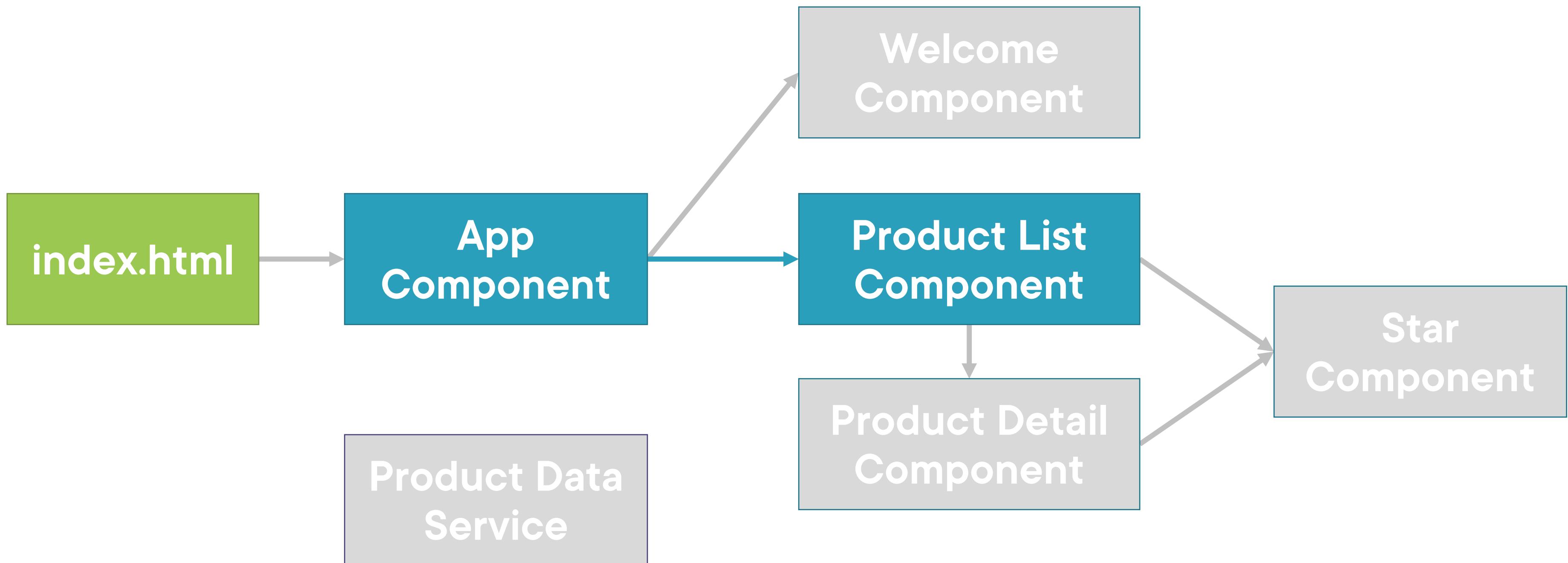
Pipe parameters

- Separated with colons

Example

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

Application Architecture





Coming up next ...

More on Components

More on Components

Introduction



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



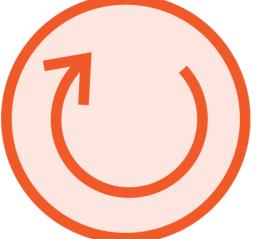
Improving Our Components



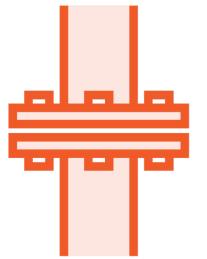
Strong typing & interfaces



Encapsulating styles



Lifecycle hooks



Custom pipes



Nested components

Module Overview



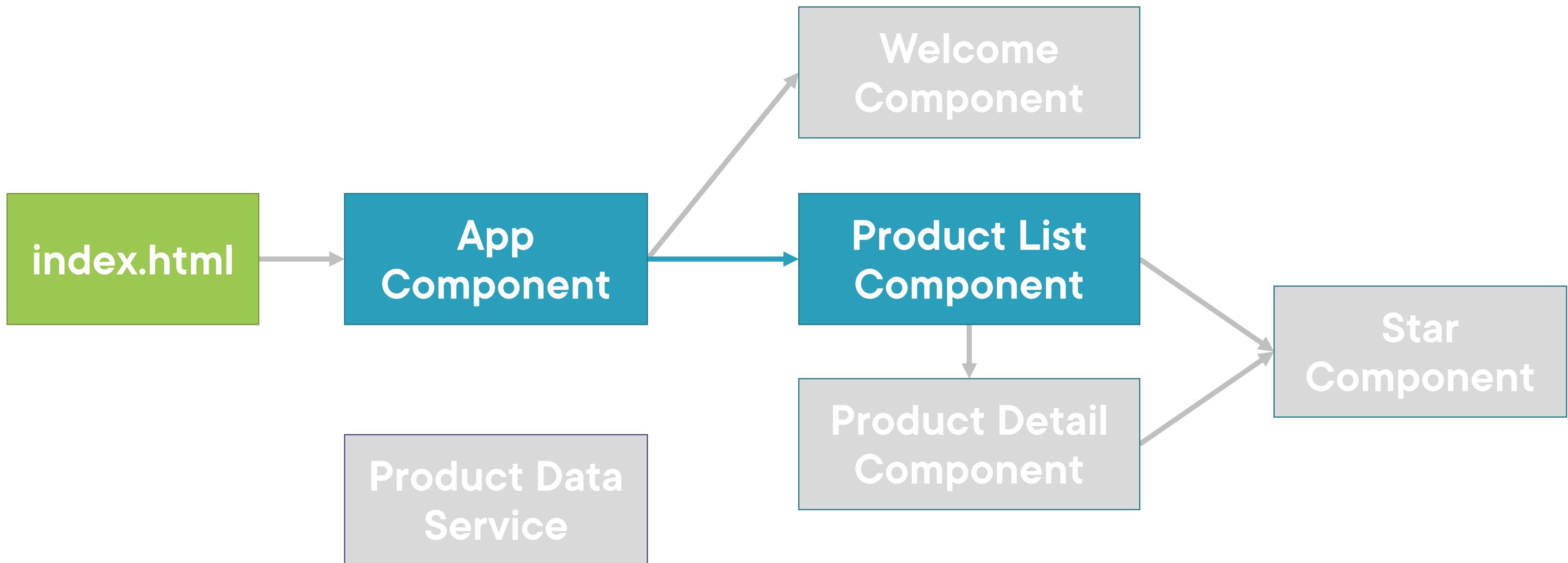
Defining an interface

Encapsulating component styles

Using lifecycle hooks

Building a custom pipe

Application Architecture



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;
    }
}
```

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 {  
        ...  
    }  
}
```

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

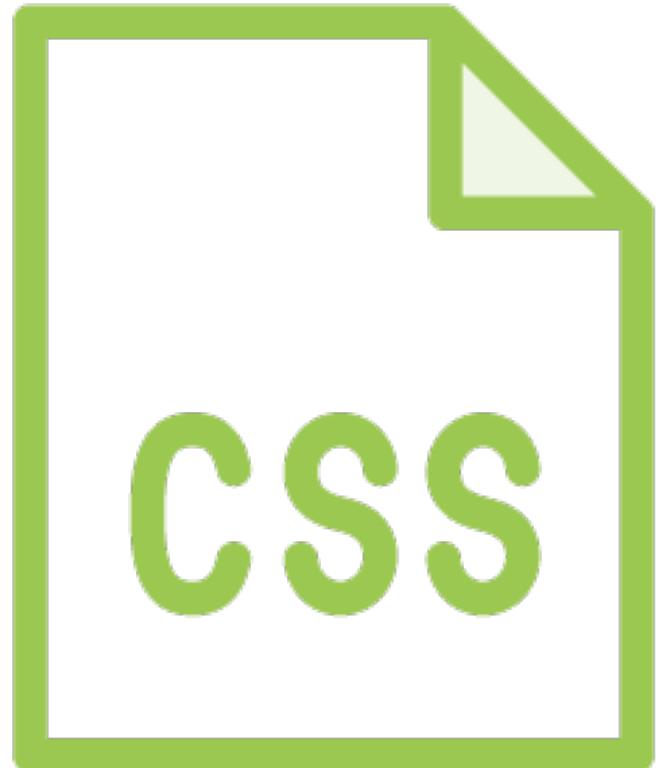
```
import { IProduct } from './product';

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

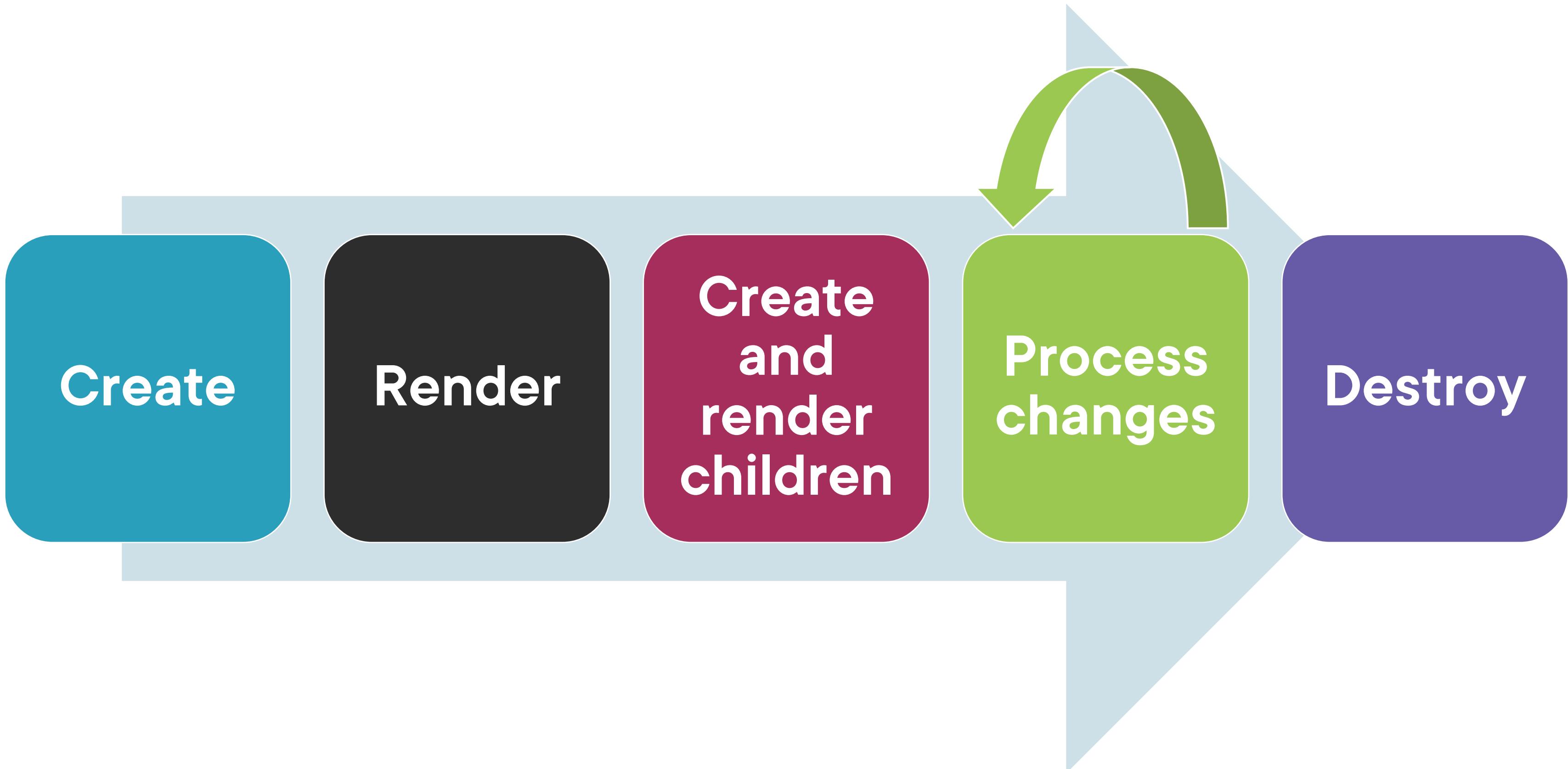
styles

```
@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

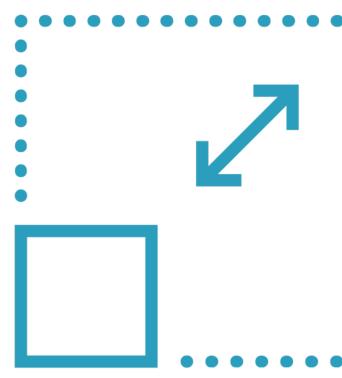
1

```
export class ProductListComponent implements OnInit {  
  pageTitle: string = 'Product List';  
  showImage: boolean = false;  
  listFilter: string = 'cart';  
  products: IProduct[] = [ ...];
```

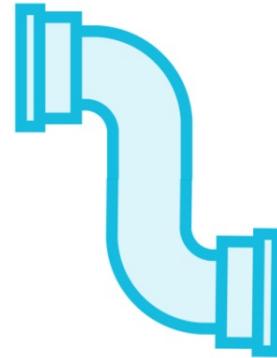
3

```
}
```

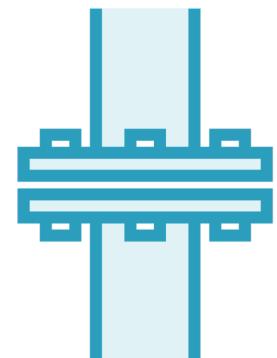
Transforming Data with Pipes



Transform bound properties before display



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



Custom pipes

Building a Custom Pipe

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

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

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

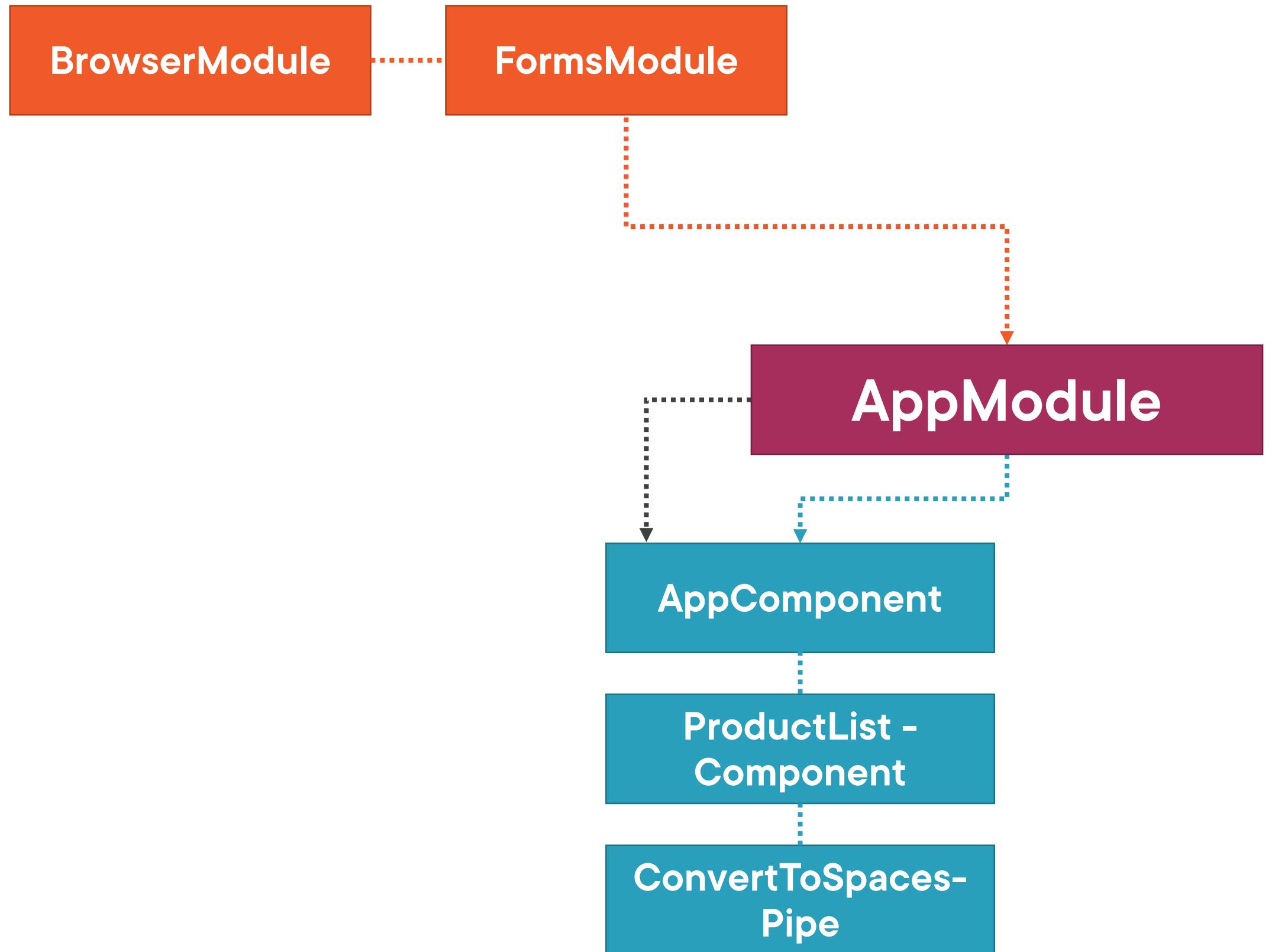
Using a Custom Pipe

Template

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

Pipe

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



..... Imports

..... Exports

..... Declarations

..... Bootstrap

Using a Custom Pipe

Template

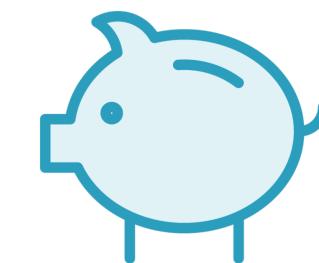
```
<td>{{ product.productCode | convertToSpaces:'-' }}</td>
```

Module

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

Getters and Setters

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

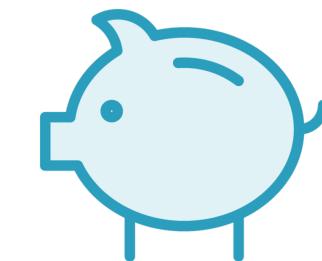


```
get amount(): number {
    // process the amount
    // return amount from private storage
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
}
```

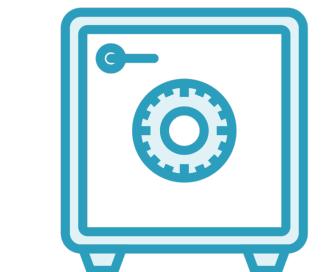


Getters and Setters

```
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.

Filtering a List

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

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

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();  
}
```

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');
  }
}
```

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 { ... }
}
```

Pipe Checklist: Using a Custom Pipe



Add the pipe to the declarations array of an Angular module

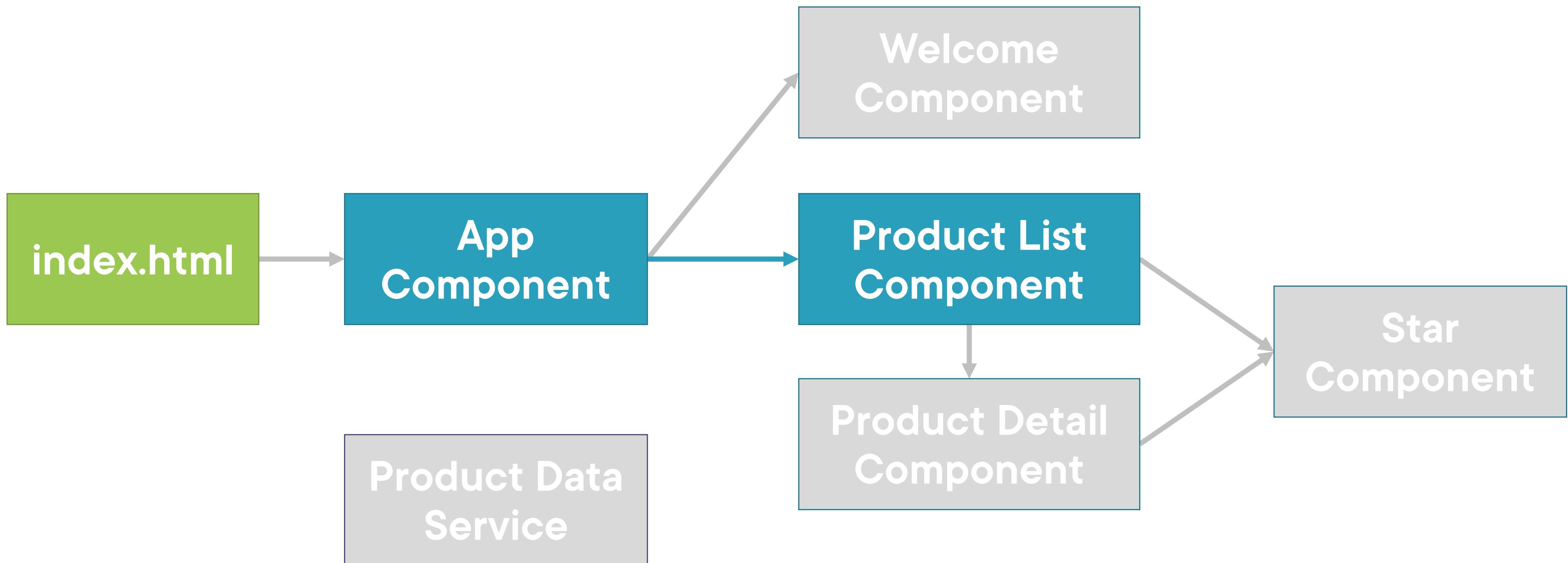
```
@NgModule({  
  imports: [...],  
  declarations: [  
    AppComponent,  
    ProductListComponent,  
    SpacePipe ],  
  bootstrap: [ AppComponent ]  
})  
export class AppModule { }
```

Use the pipe in a template

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

```
{{ product.productCode | spacePipe:'-' }}
```

Application Architecture





Coming up next ...

Building Nested Components

Product List					
Filter by: <input type="text"/>					
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	★★★1
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	★★★★1
	Hammer	tbx 0048	May 21, 2021	\$8.90	★★★★★
	Saw	tbx 0022	May 15, 2021	\$11.55	★★★★
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	★★★★★

Building Nested Components



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



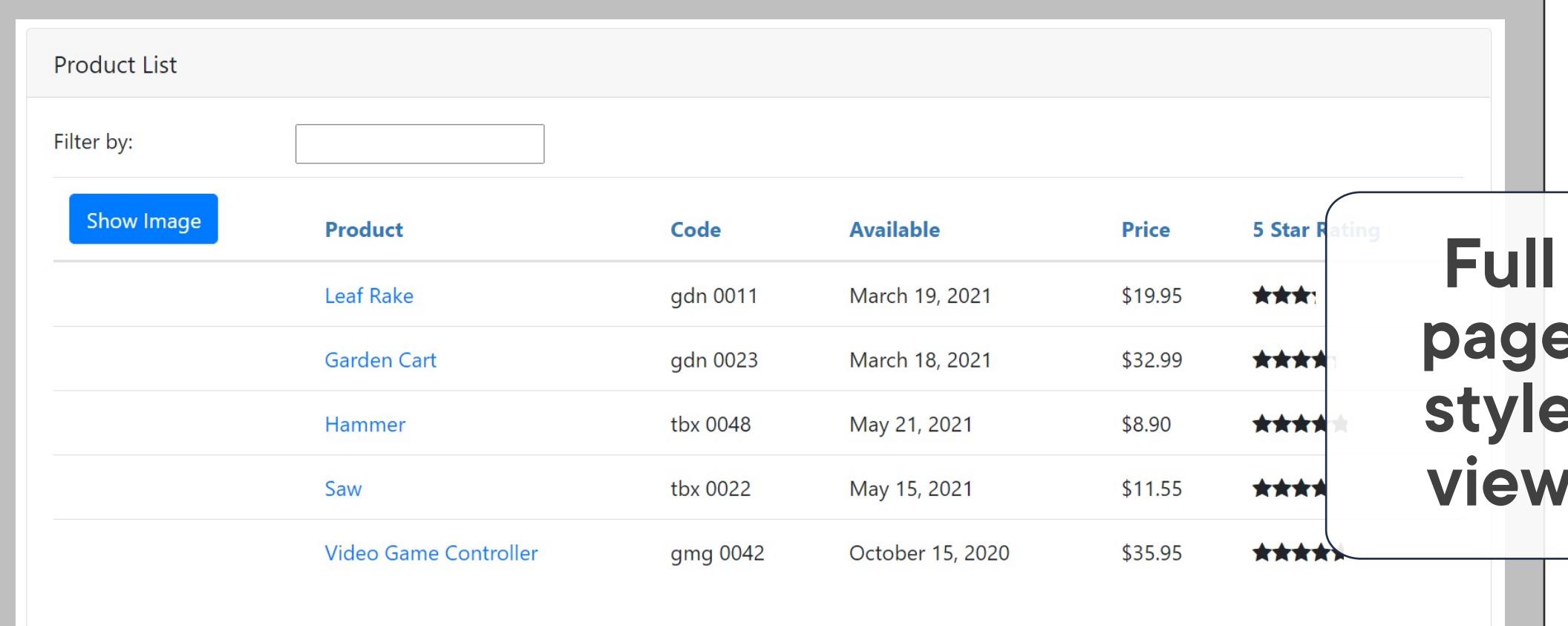
Using a Component

As a Directive



App Component OR Nested Component

As a Routing target



Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	★★★½
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	★★★★½
	Hammer	tbx 0048	May 21, 2021	\$8.90	★★★★★
	Saw	tbx 0022	May 15, 2021	\$11.55	★★★★★
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	★★★★★

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

What Makes a Component Nest-able?



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

Module Overview



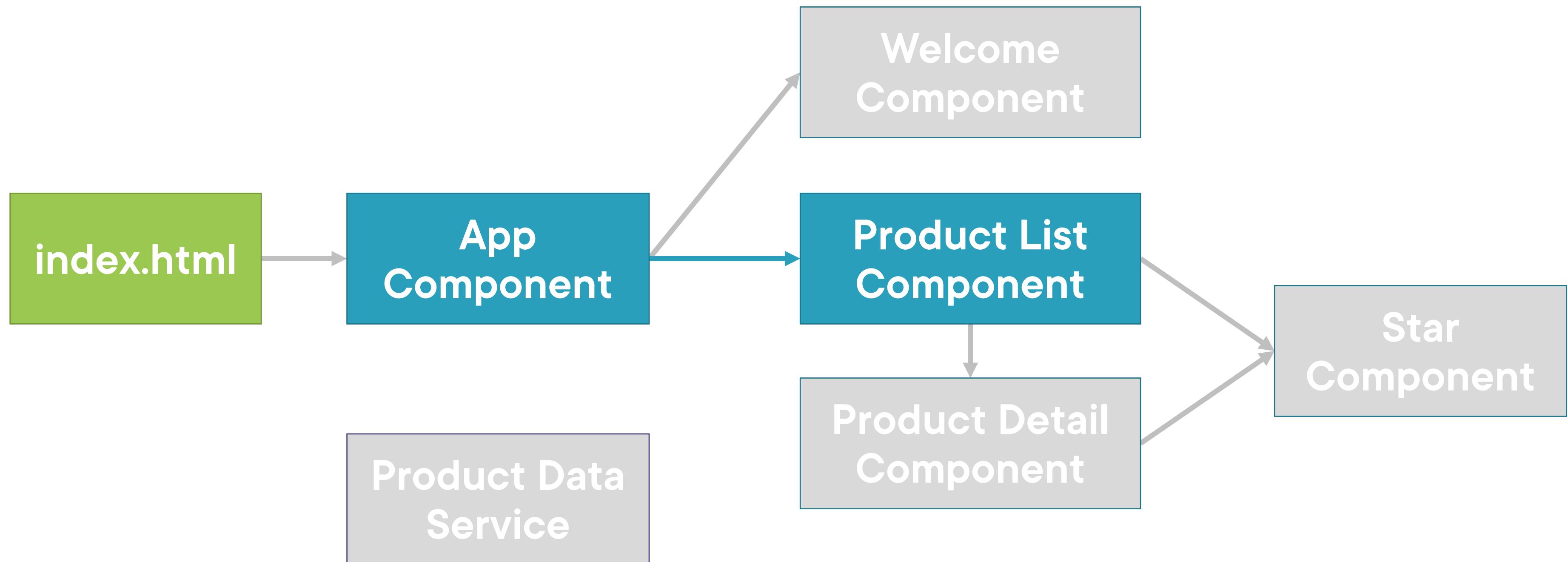
Building a nested component

Using a nested component

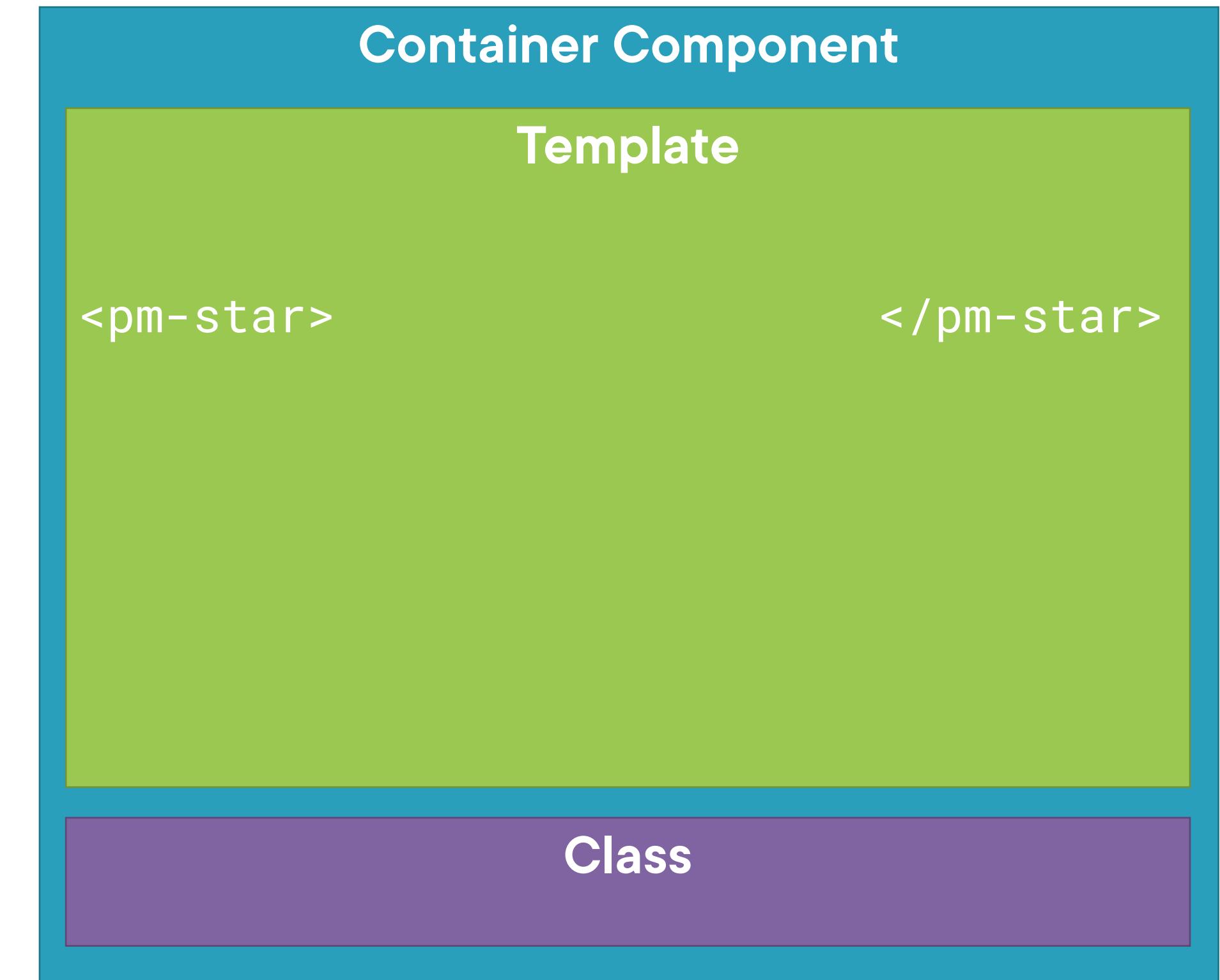
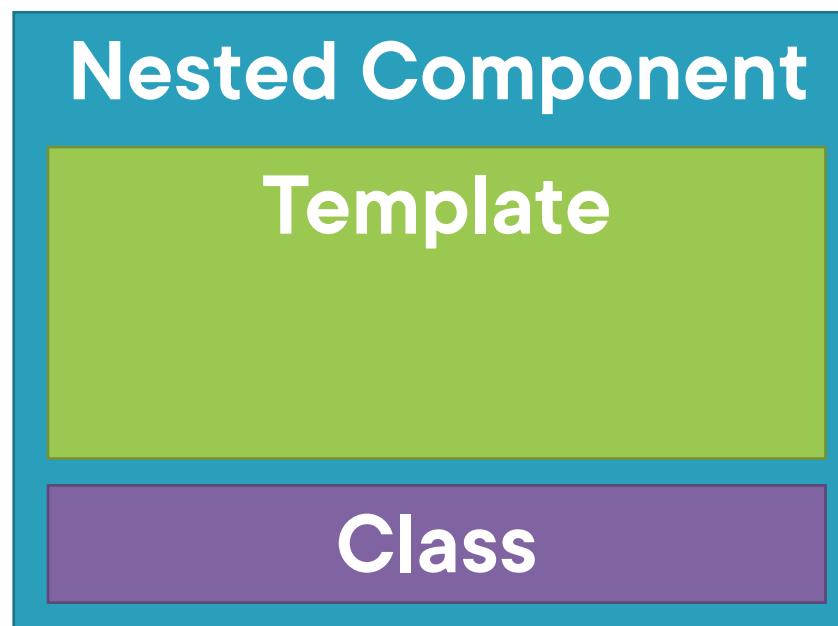
**Passing data to a nested component using
@Input**

**Raising an event from a nested component
using @Output**

Application Architecture



Building a Nested Component



Product List View

Product List					
Filter by:	<input type="text"/>				
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	3.2
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	4.2
	Hammer	tbx 0048	May 21, 2021	\$8.90	4.8
	Saw	tbx 0022	May 15, 2021	\$11.55	3.7
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	4.6

Product List View

Product List

Filter by:

Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	★★★★1
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	★★★★★1
	Hammer	tbx 0048	May 21, 2021	\$8.90	★★★★★
	Saw	tbx 0022	May 15, 2021	\$11.55	★★★★★
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	★★★★★

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

Using a Nested Component as a Directive

product-list.component.ts

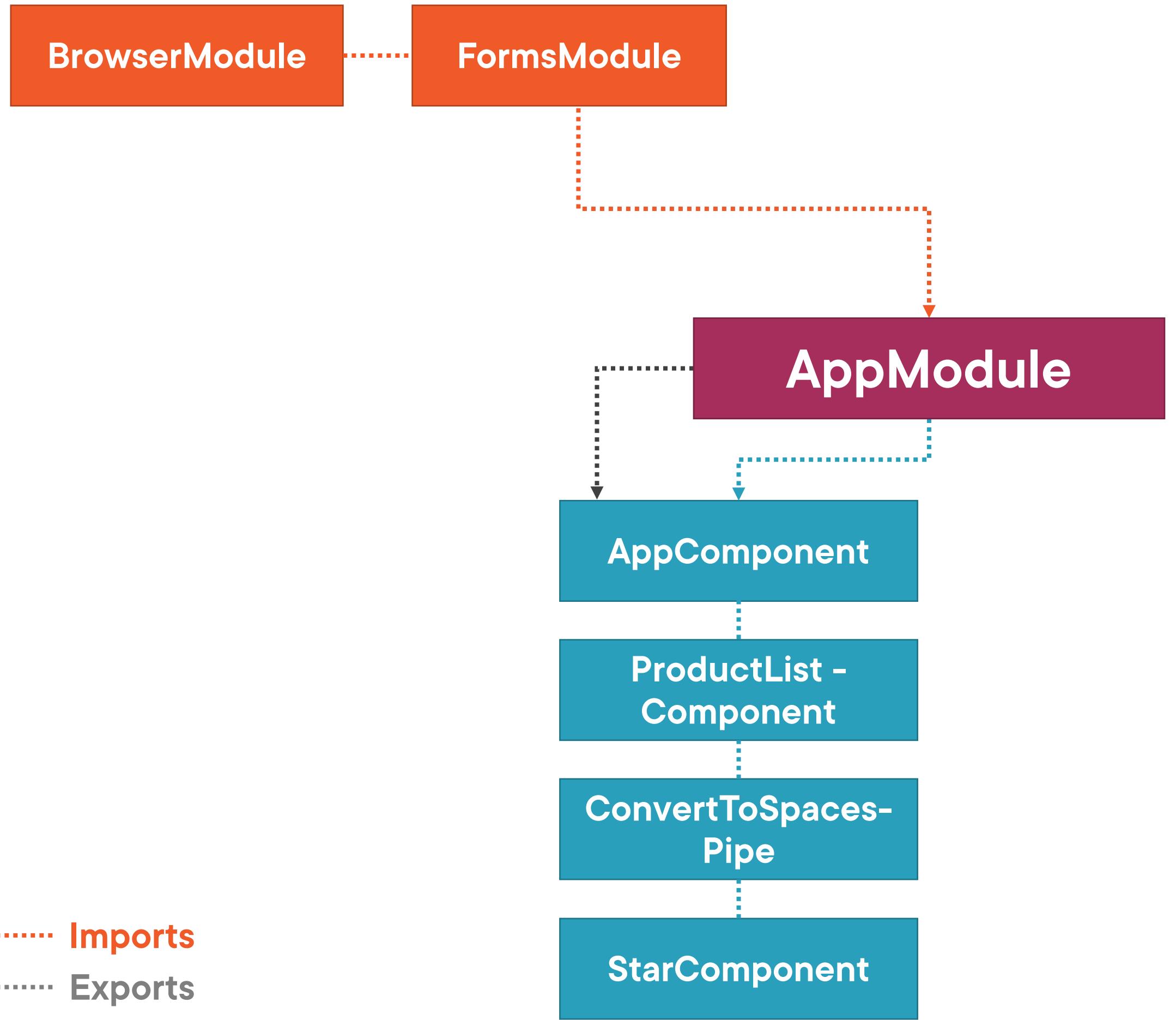
```
@Component({
  selector: 'pm-products',
  templateUrl: './product-list.component.html'
})
export class ProductListComponent { }
```

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>
```



..... Imports

..... Exports

..... Declarations

..... Bootstrap

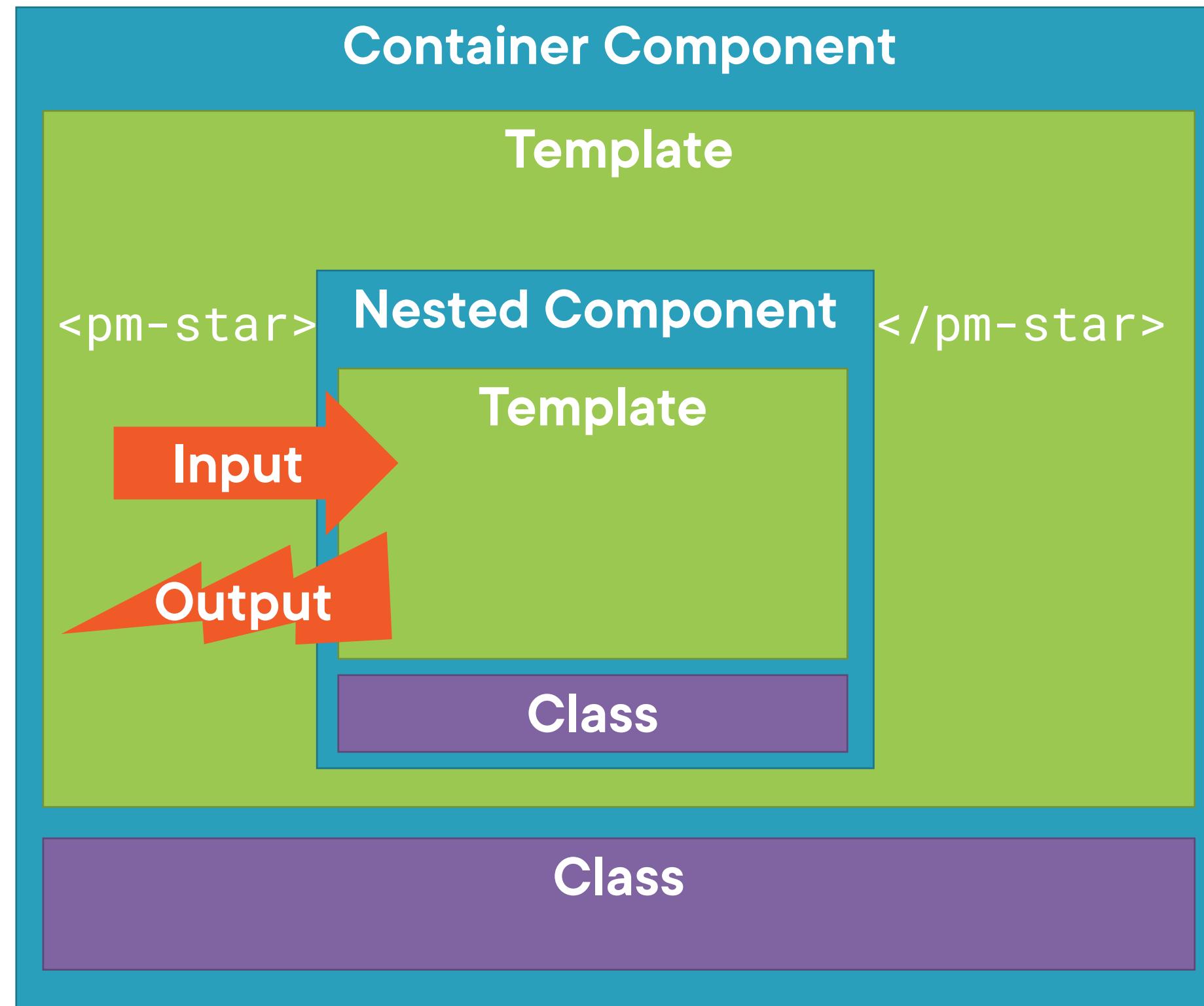
Telling Angular About Our Component

app.module.ts

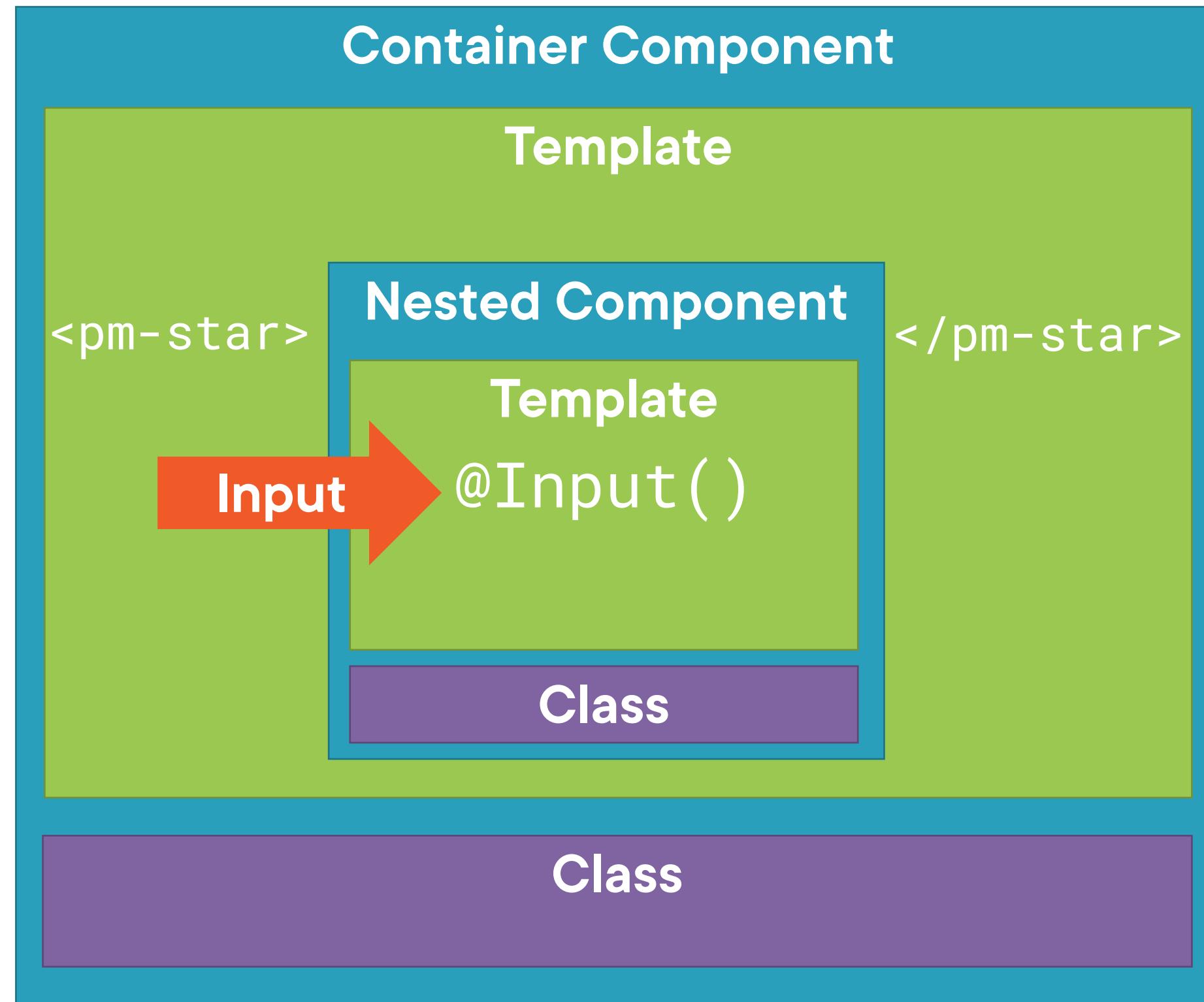
```
...
import { StarComponent } from './shared/star.component';

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

star.component.ts

```
@Component({  
  selector: 'pm-star',  
  templateUrl: './star.component.html'  
})  
export class StarComponent {  
  @Input() rating: number;  
  cropWidth: 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>  
  <pm-star [rating]='product.starRating'>  
  </pm-star>  
</td>
```

star.component.ts

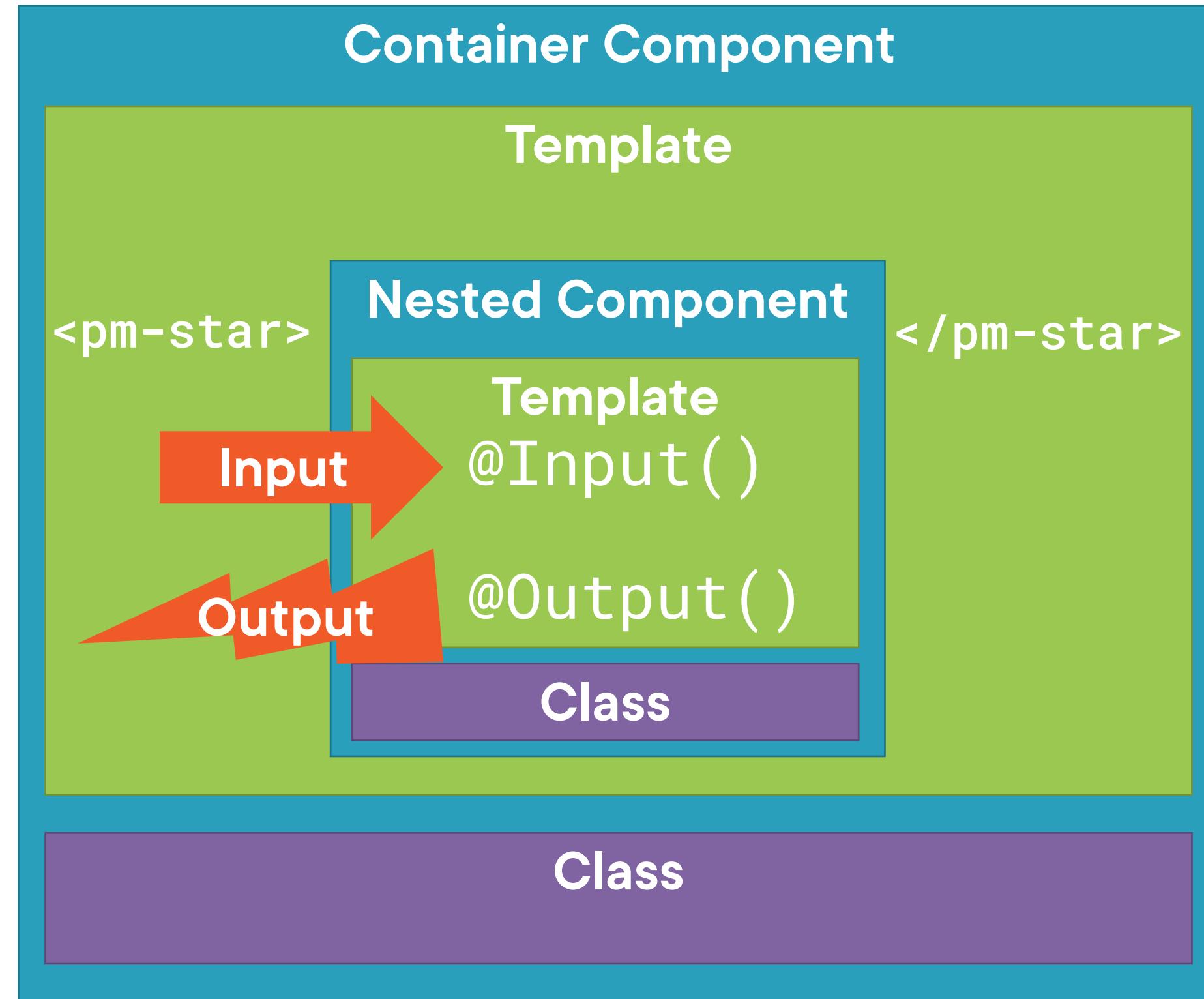
```
@Component({  
  selector: 'pm-star',  
  templateUrl: './star.component.html'  
})  
export class StarComponent {  
  @Input() rating: number;  
  cropWidth: number;  
}
```

Demo



Handling events

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 {  
  @Input() rating: number;  
  cropWidth: number;  
  @Output() notify: EventEmitter<string> =  
    new EventEmitter<string>();  
}
```

product-list.component.html

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

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 {  
  @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'>  
  </pm-star>  
</td>
```

star.component.html

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

Emitting an Event (@Output)

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'  
           (notify)='onNotify($event)'>  
  </pm-star>  
</td>
```

star.component.ts

```
@Component({  
  selector: 'pm-star',  
  templateUrl: './star.component.html'  
})  
export class StarComponent {  
  @Input() rating: number;  
  cropWidth: number;  
  @Output() notify: EventEmitter<string> =  
    new EventEmitter<string>();  
  onClick() {  
    this.notify.emit('clicked!');  
  }  
}
```

star.component.html

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

Emitting an Event (@Output)

product-list.component.ts

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

product-list.component.html

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

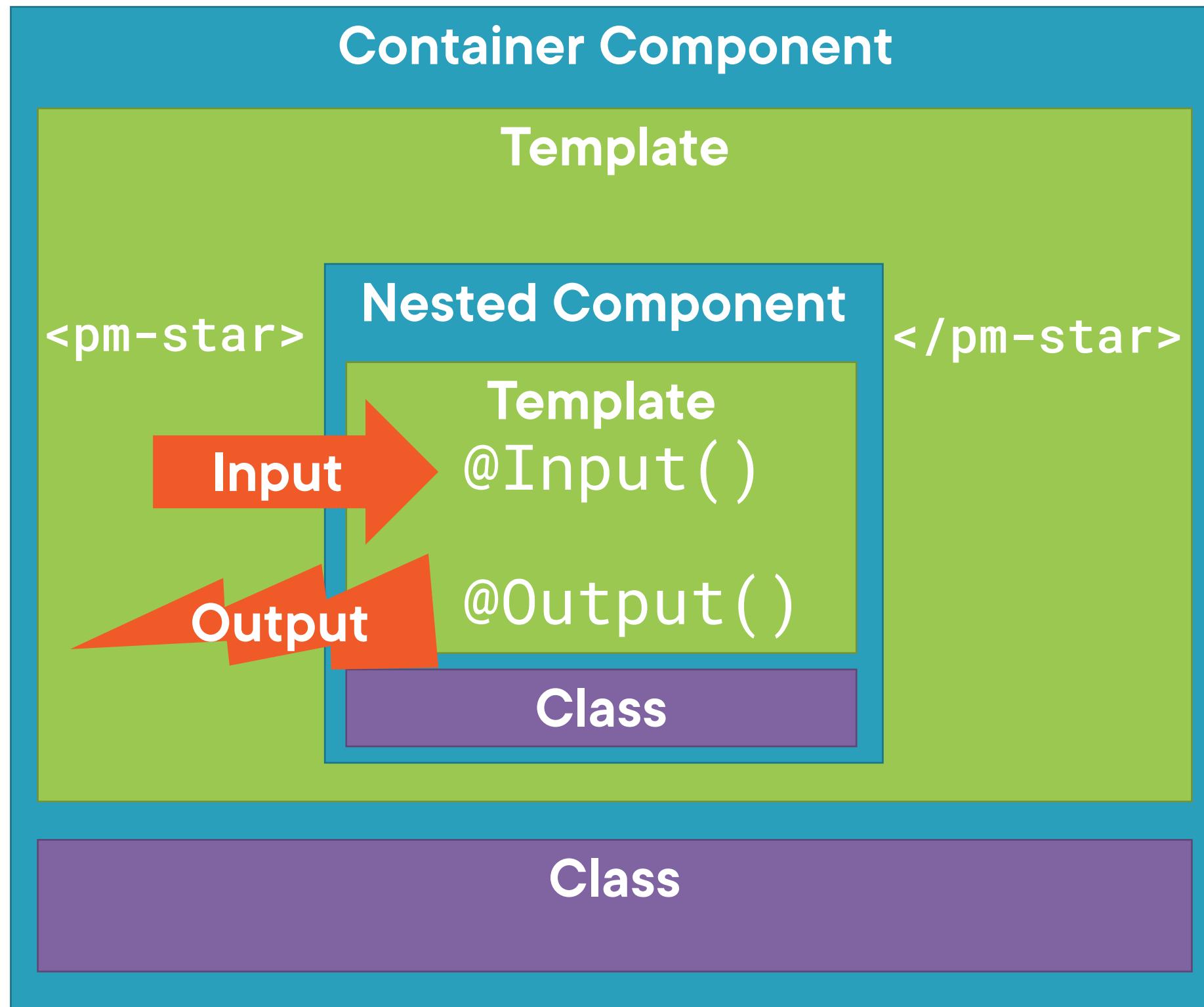
star.component.ts

```
@Component({  
  selector: 'pm-star',  
  templateUrl: './star.component.html'  
})  
export class StarComponent {  
  @Input() rating: number;  
  cropWidth: number;  
  @Output() notify: EventEmitter<string> =  
    new EventEmitter<string>();  
  onClick() {  
    this.notify.emit('clicked!');  
  }  
}
```

star.component.html

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

Nestable 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

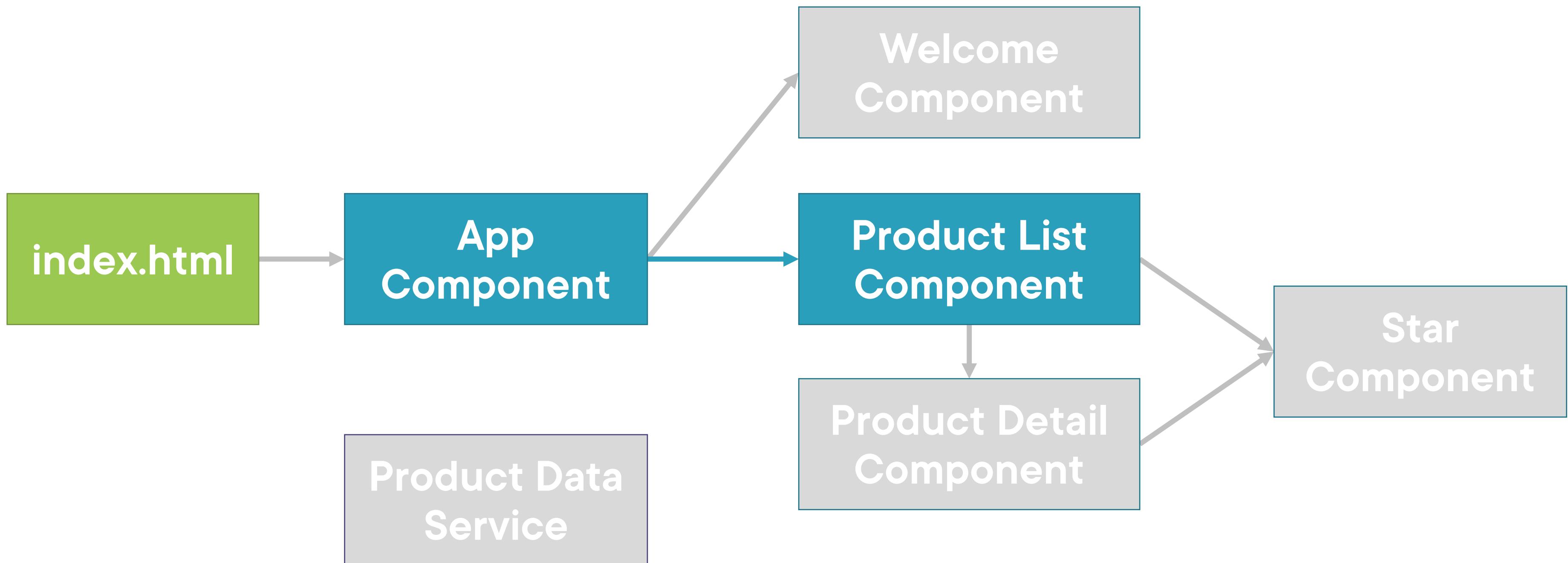
```
<pm-star [rating]='product.starRating'  
         (notify)='onNotify($event)'>  
</pm-star>
```

Learning More

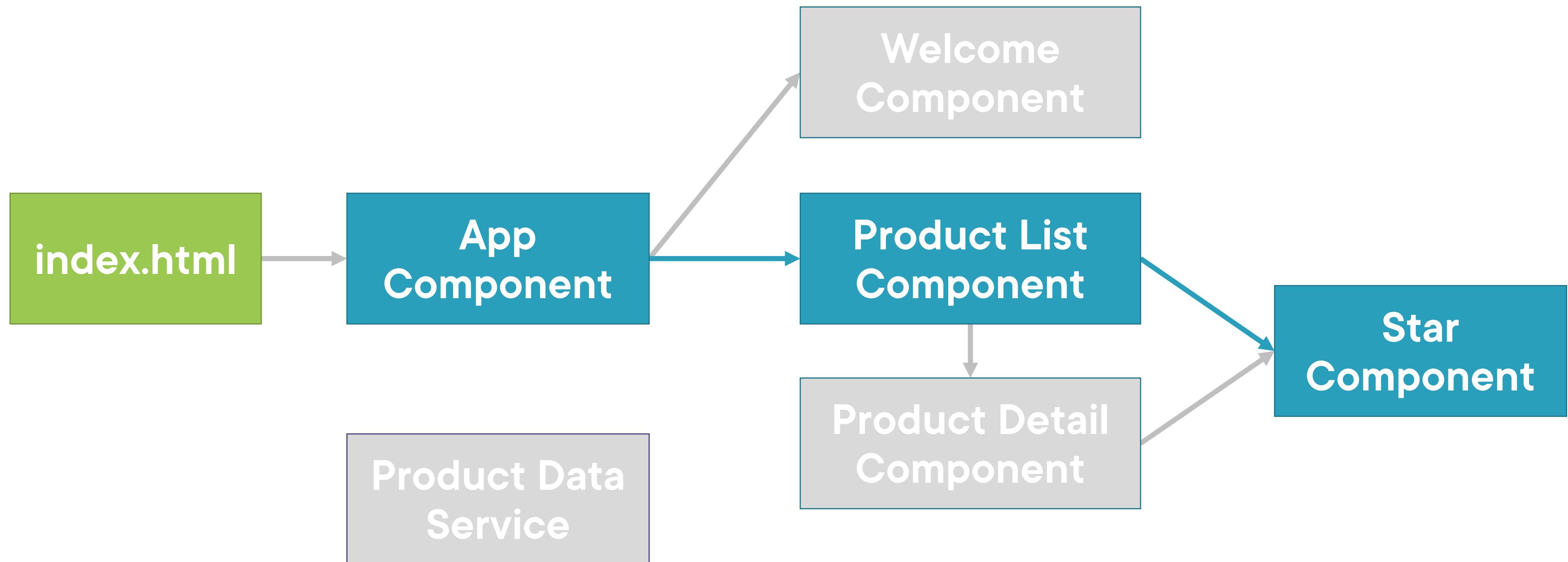


Pluralsight Course
– “Angular Component Communication”

Application Architecture



Application Architecture





Products

Logging

Coming up next ...

**Services and
Dependency Injection**

Services and Dependency Injection



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata

Products

Logging

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**

Module Overview



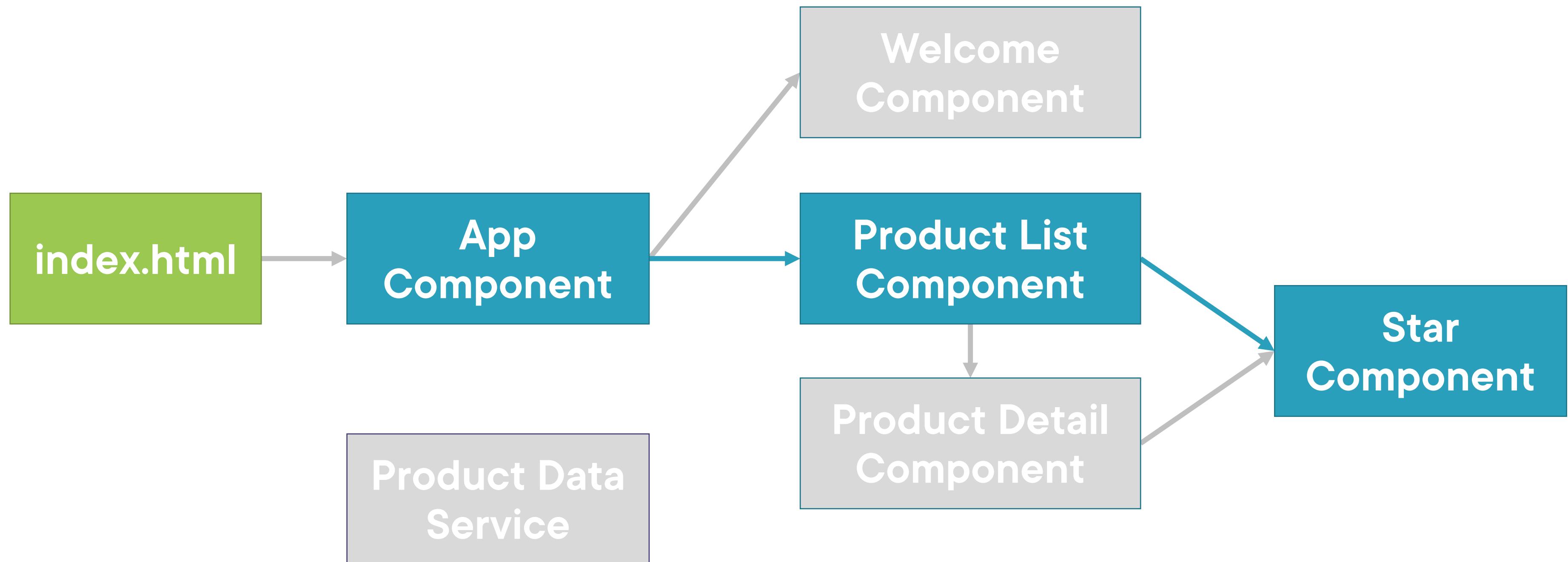
How does it work?

Building a service

Registering the service

Injecting the service

Application Architecture



How Does It Work?

Service

```
export class myService {}
```

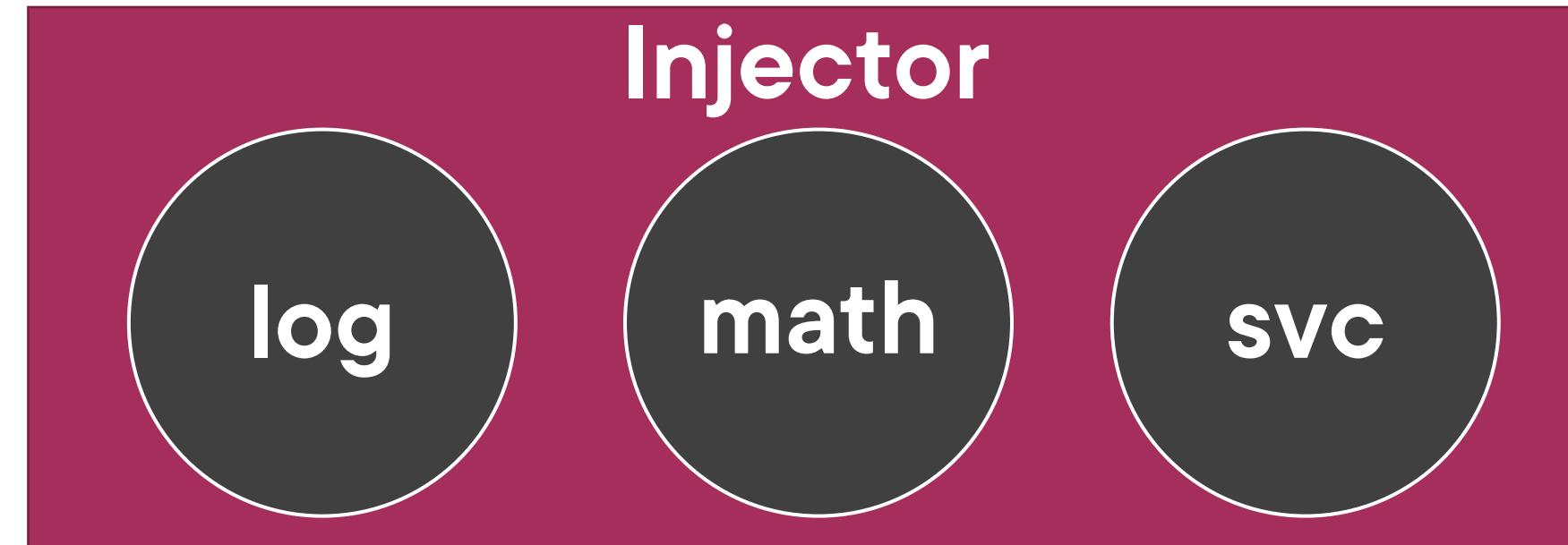
Component

```
let svc = new myService();
```



svc

How Does It Work?



Service

```
export class myService {}
```

Component

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

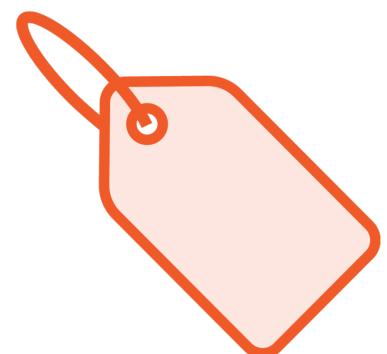
Dependency Injection

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



Import what we need

Building a Service

product.service.ts

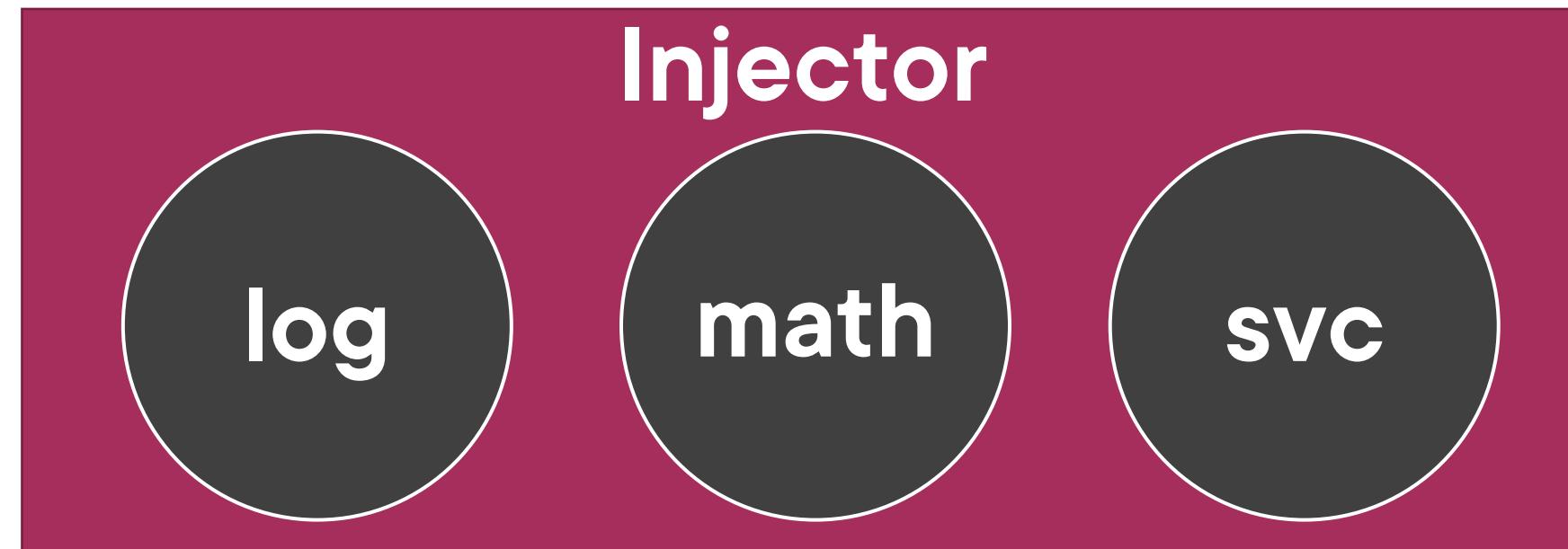
```
import { Injectable } from '@angular/core'

@Injectable()
export class ProductService {

  getProducts(): IProduct[] {
    }

}
```

Registering a Service



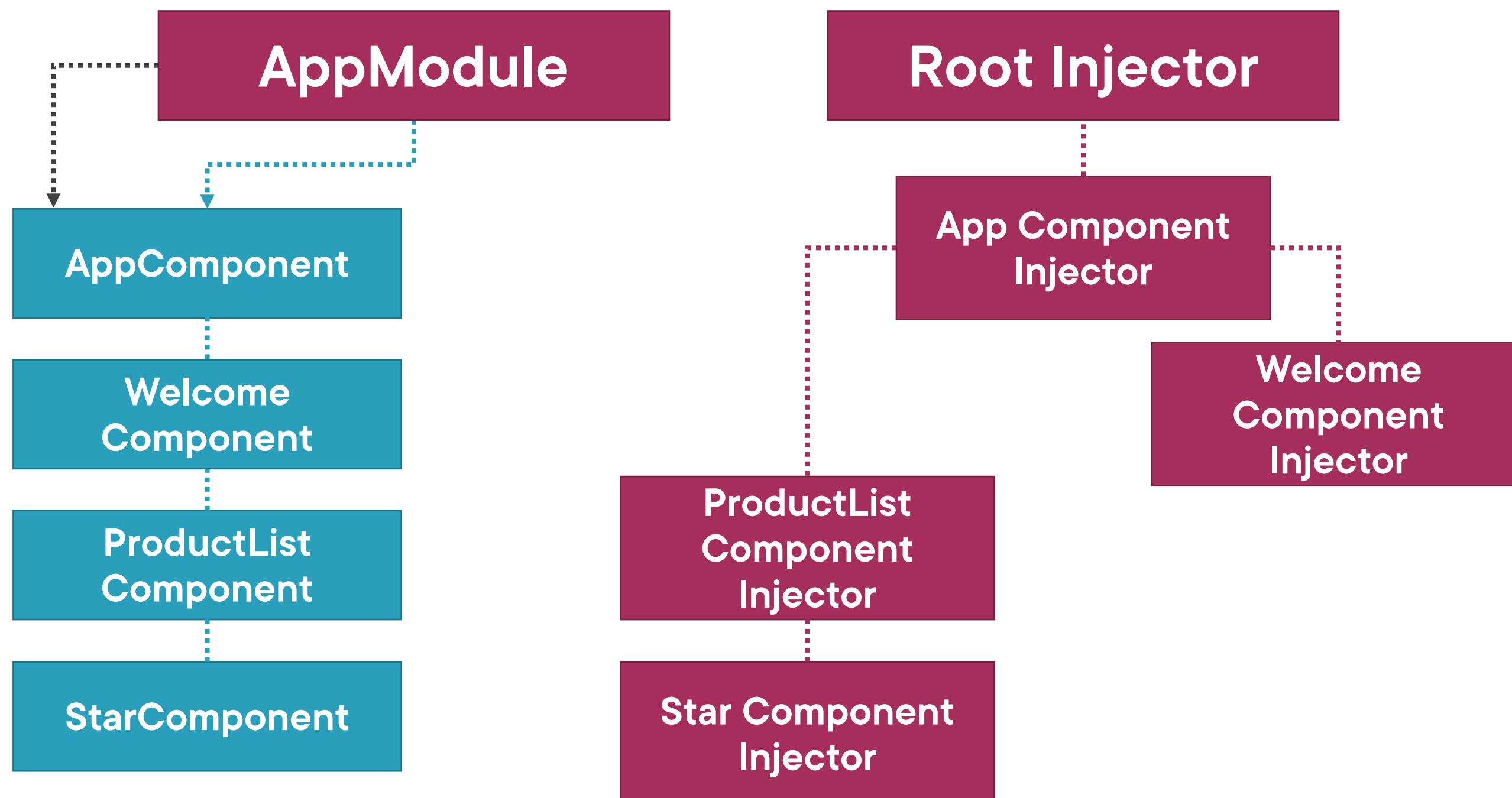
Service

```
export class myService {}
```

Component

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

Angular Injectors



Registering a Service

Root Injector

Service is available throughout the application

Recommended for most scenarios

Component Injector

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

Isolates a service used by only one component

Provides multiple instances of the service

Registering a Service - Root Application

product.service.ts

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

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

  getProducts(): IProduct[] {
  }

}
```

product.service.ts

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

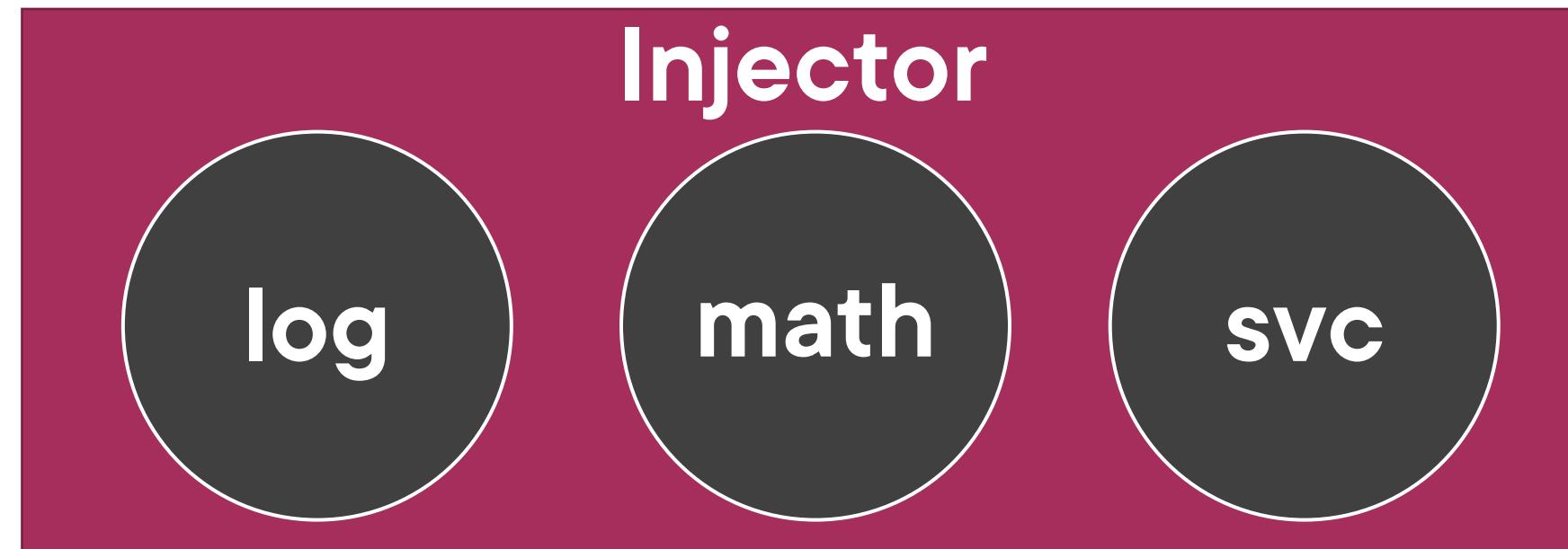
product-list.component.ts

```
@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



Service

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

Component

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

Injecting the Service

product-list.component.ts

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

  constructor() {
  }

}
```

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



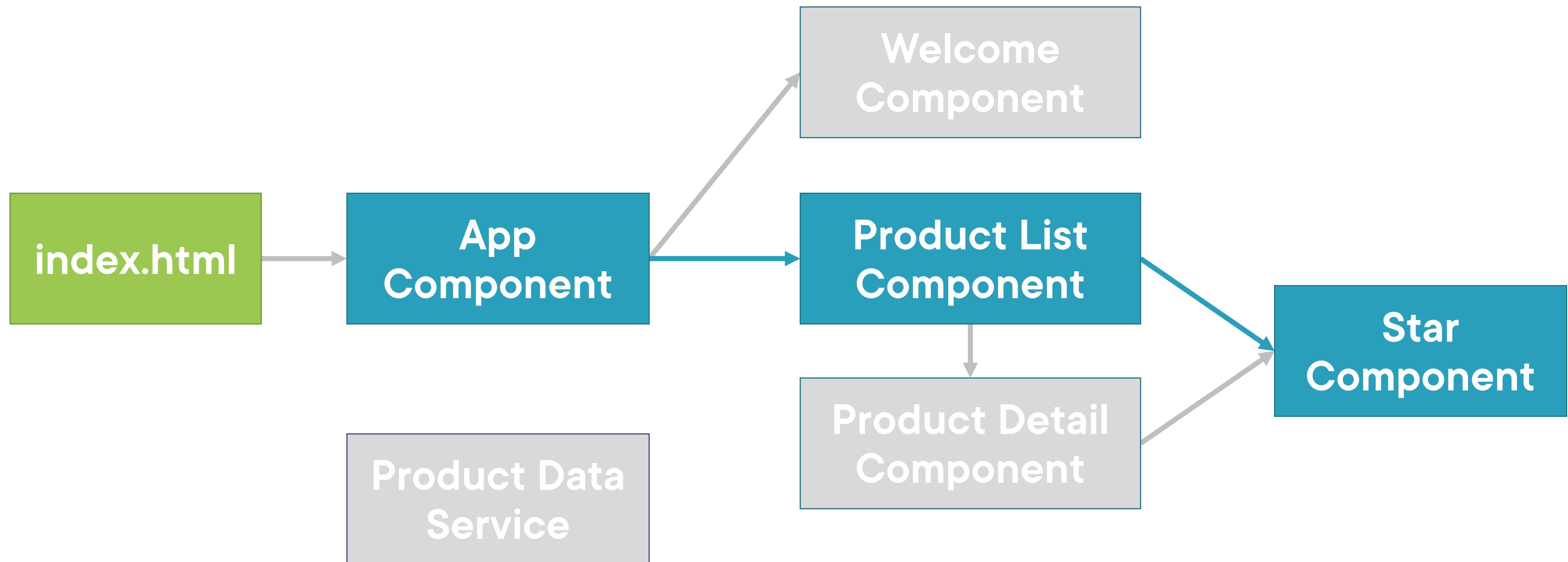
Specify the service as a dependency

Use a constructor parameter

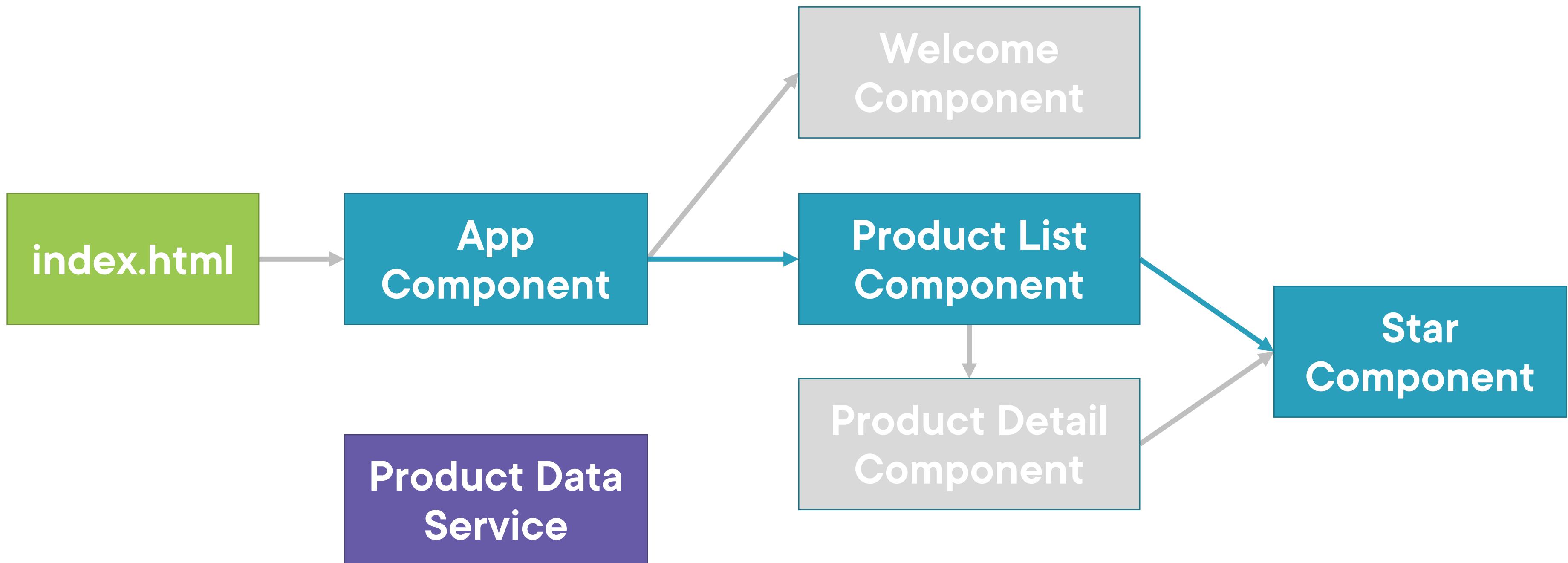
Service is injected when component is instantiated

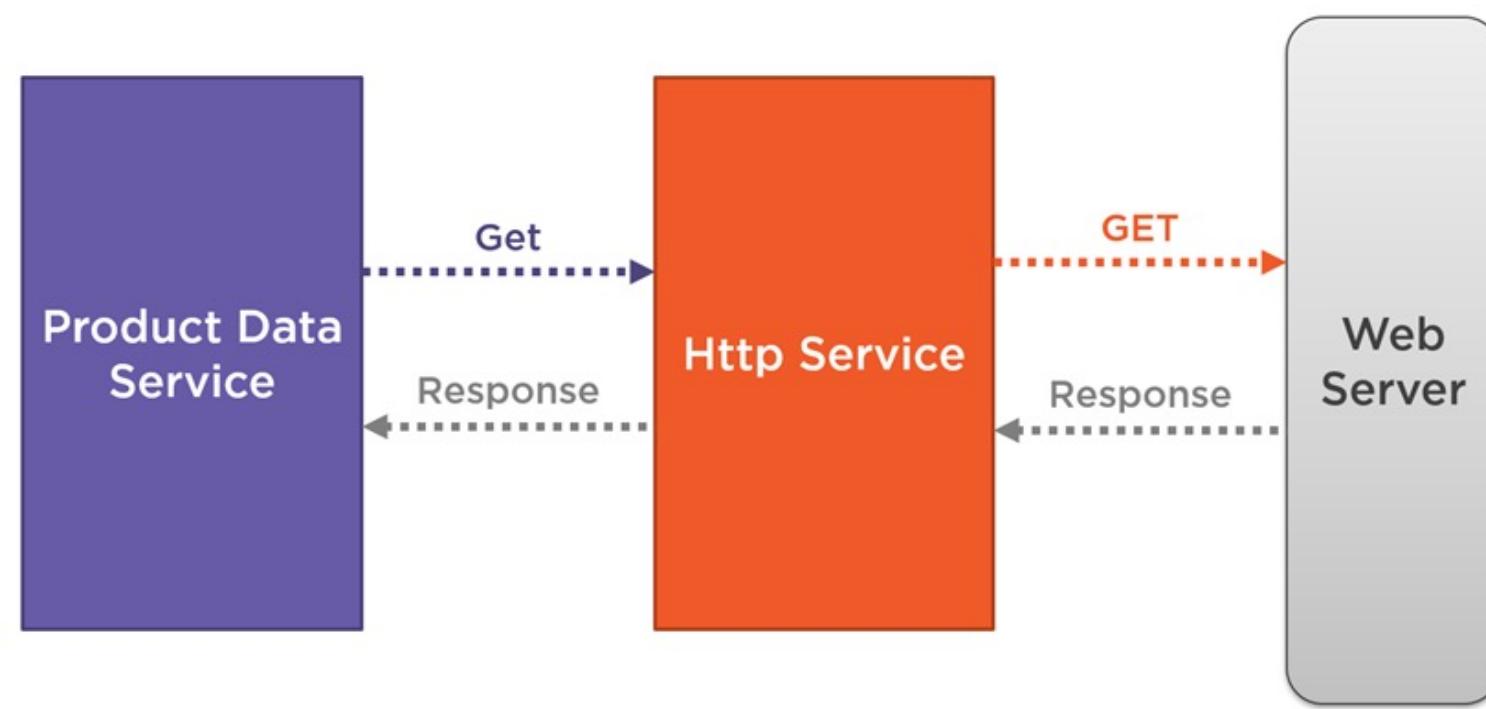
```
constructor(private productService: ProductService) { }
```

Application Architecture



Application Architecture





Coming up next ...

Retrieving Data Using HTTP

Retrieving Data Using HTTP



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

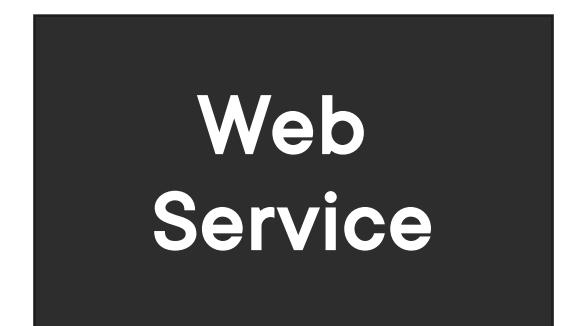
@deborahkurata



Web Browser

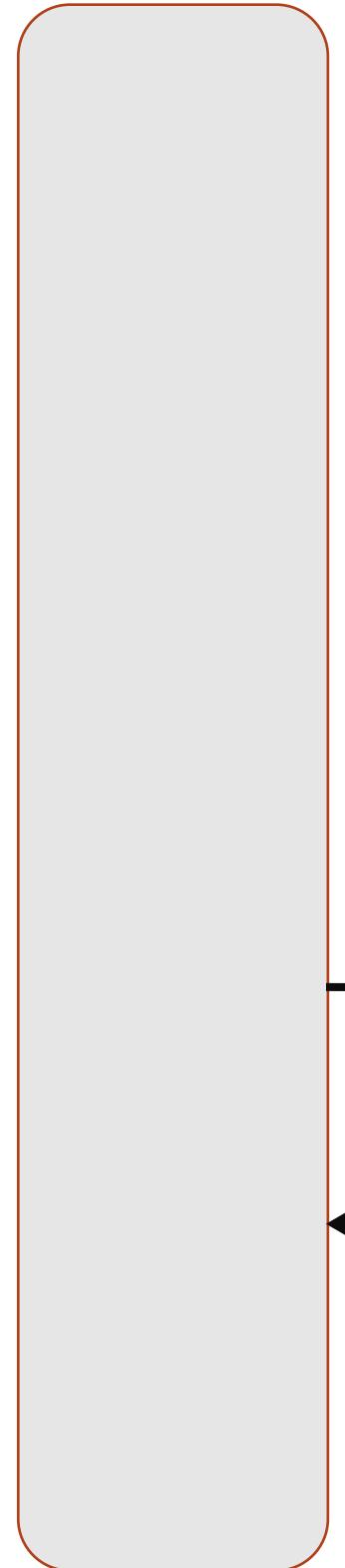


Web Server



`http://mysite/api/products/5`

Response



Module Overview



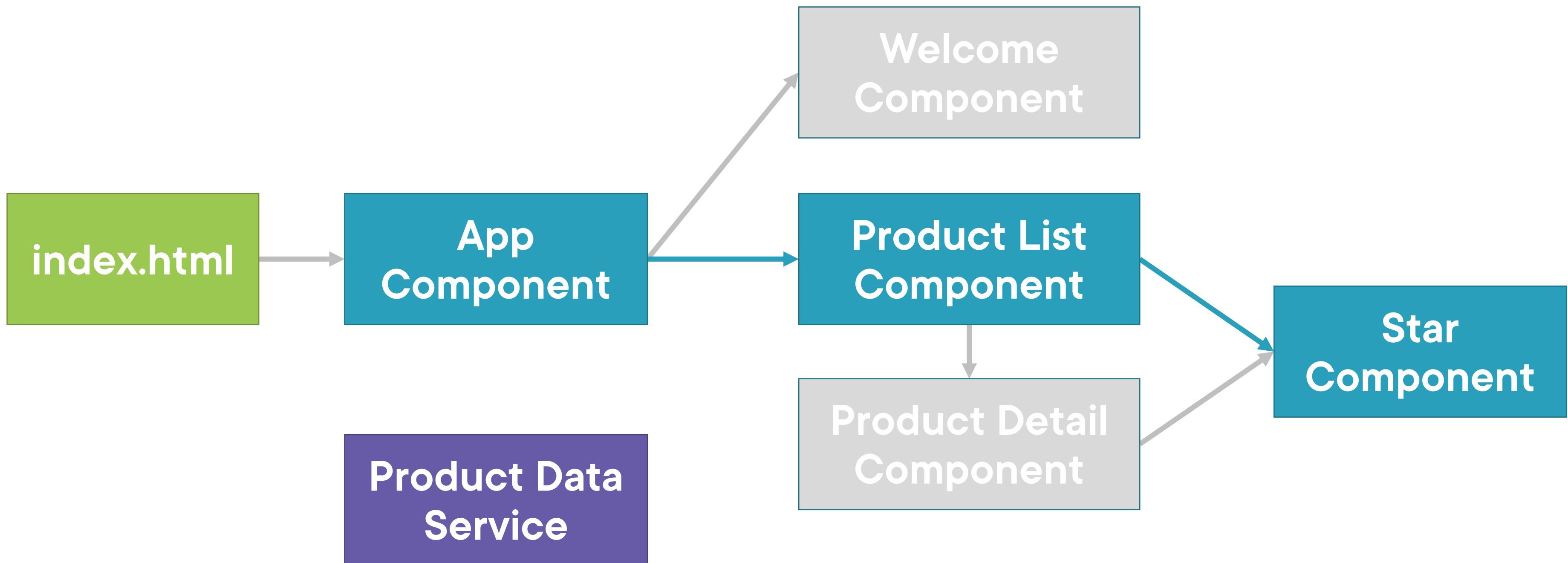
Observables and Reactive Extensions

Sending an HTTP request

Exception handling

Subscribing to an Observable

Application Architecture



To understand the HTTP code,
it's important to understand
Reactive Extensions and
Observables

Reactive Extensions (RxJS)



A library for composing data using observable sequences

And transforming that data using operators

- Similar to .NET LINQ operators

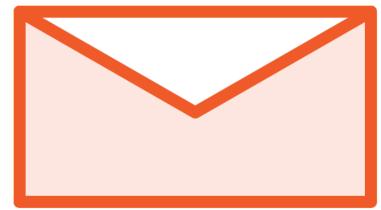
Angular uses Reactive Extensions for working with data

- Especially asynchronous data

Synchronous vs. Asynchronous



Synchronous: real time



Asynchronous: No immediate response



HTTP requests are asynchronous: request and response

Getting Data

Application

- Get me a list of products
- Notify me when the response arrives
- I'll continue along

Get me products

Web Server

At some later point in time...

Application

- "Hey, your data arrived"
- OK, I'll process it. Thanks!

Here are the products

Web Server

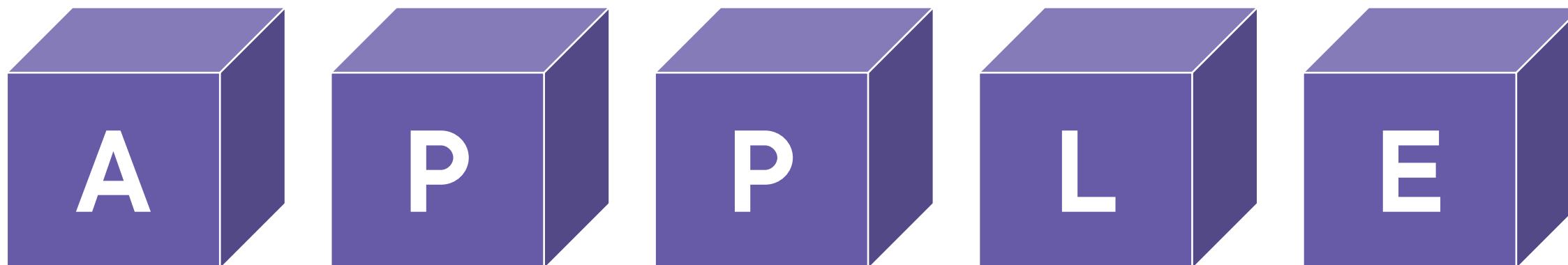
Observable

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]

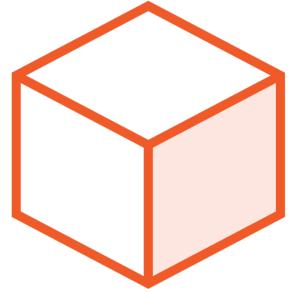
Observable:



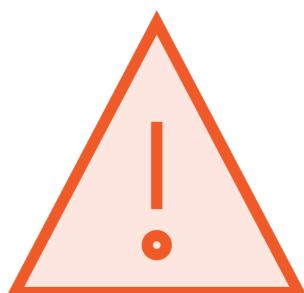
What Does an Observable Do?



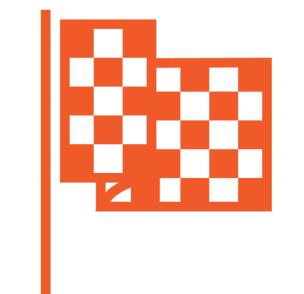
Nothing until we subscribe



next: Next item is emitted



error: An error occurred and no more items are emitted



complete: No more items are emitted

Getting Data

Application

- Call http get
- http get returns an Observable, which will emit notifications
- Subscribe to start the Observable and the get request is sent
- Code continues along

Get me products

Web Server

At some later point in time...

Application

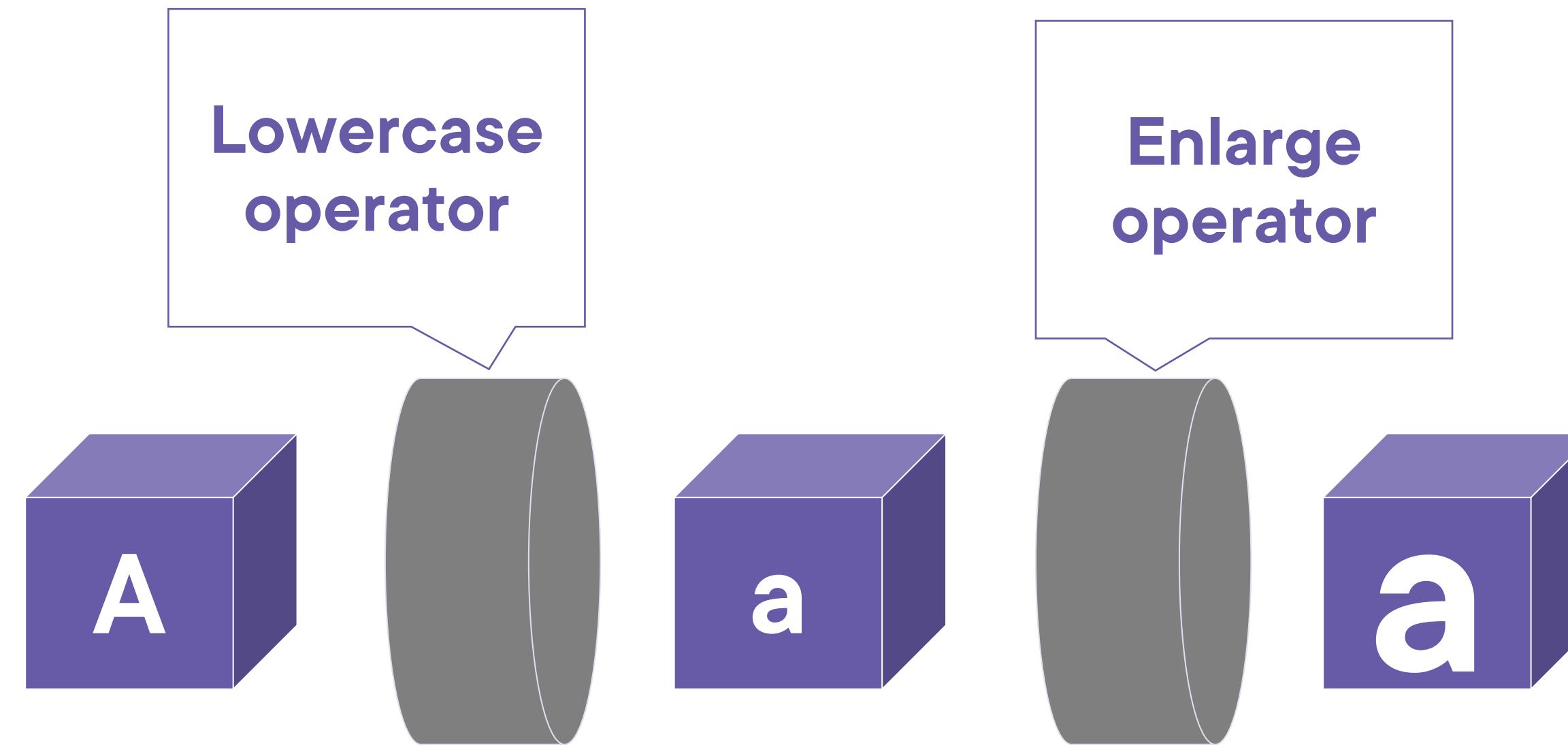
- The response is returned
- The Observable emits a **next** notification
- We process the emitted response

Here are the products

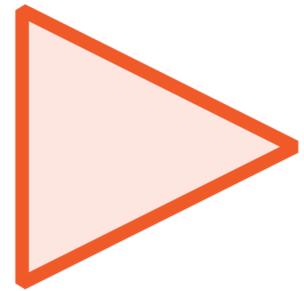
[{cart},{hammer},{saw}]

Web Server

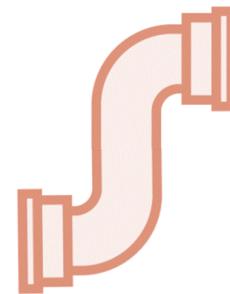
Observable Pipe



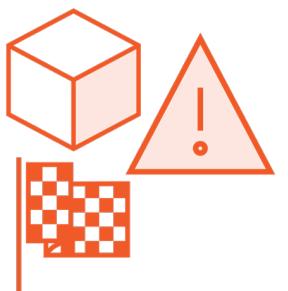
Common Observable Usage



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 } from 'rxjs';
import { map, filter } from 'rxjs/operators';

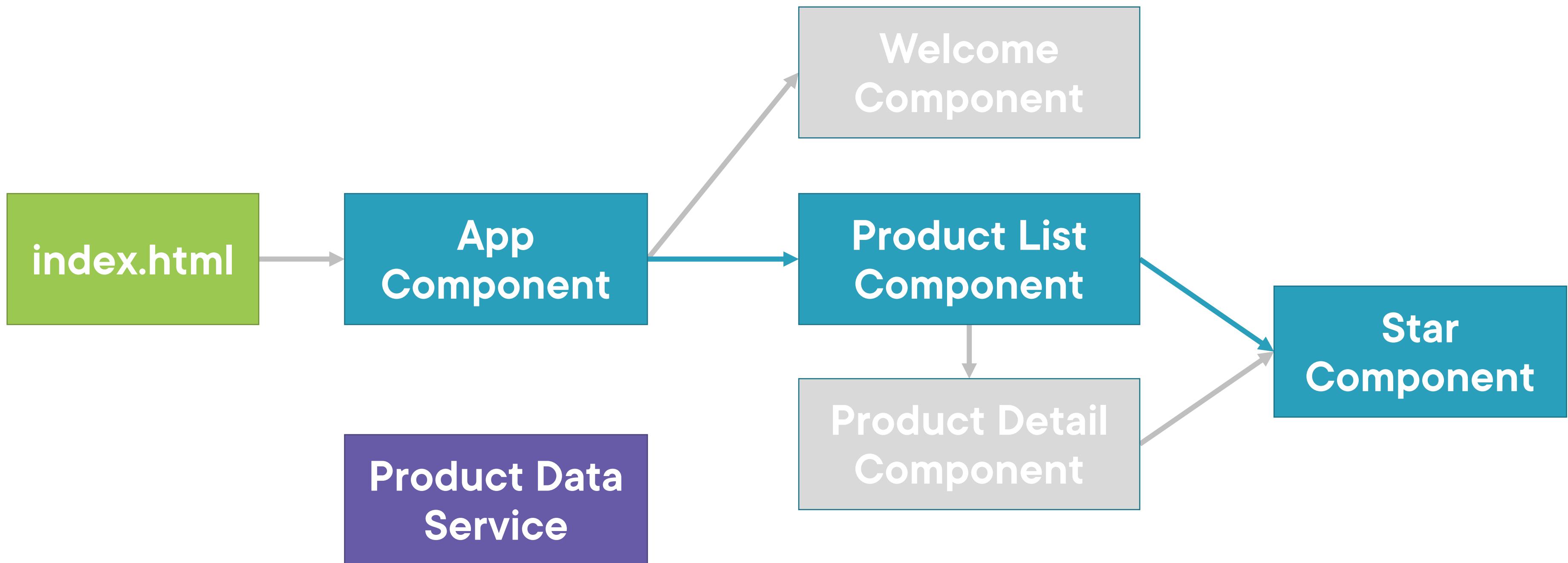
const source$: Observable<number> = range(0, 10);

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

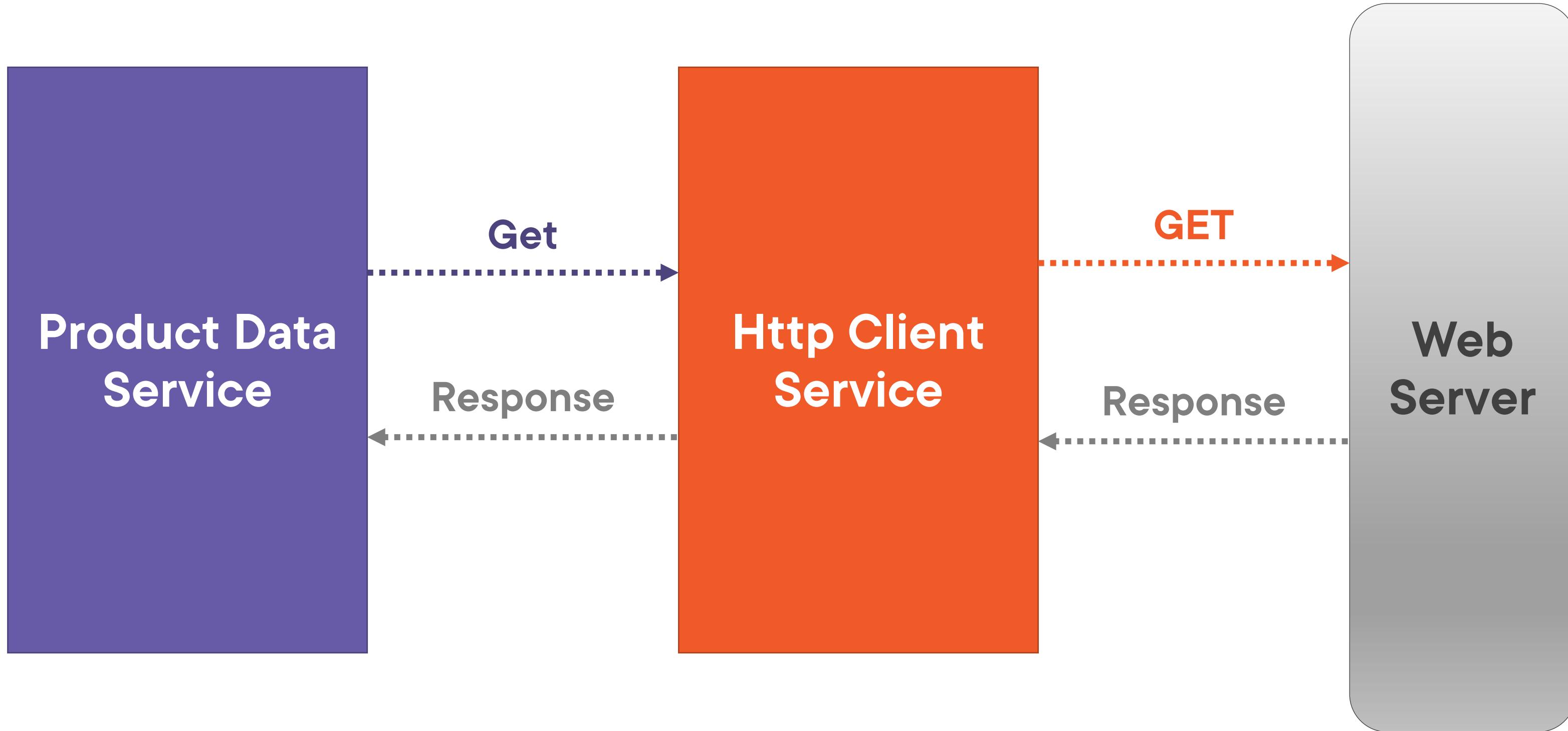
Result

```
0
6
12
18
24
```

Application Architecture



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) { }

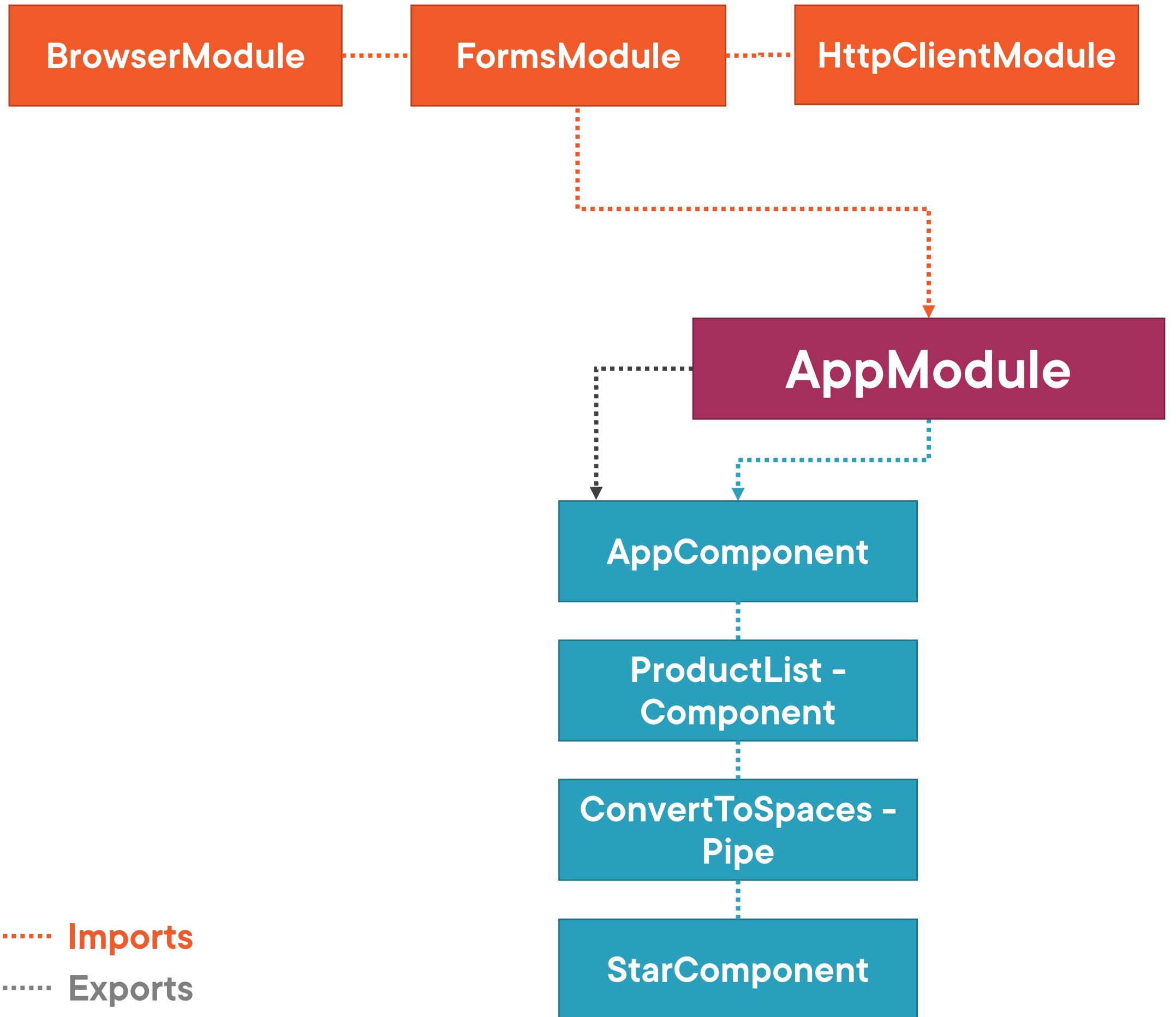
  getProducts() {
    return this.http.get(this.productUrl);
  }
}
```

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 { }
```



..... Imports

..... Exports

..... Declarations

..... Bootstrap

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);
  }
}
```

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<IPrduct[]>(this.productUrl);
  }
}
```

Setting up an HTTP Request

product.service.ts

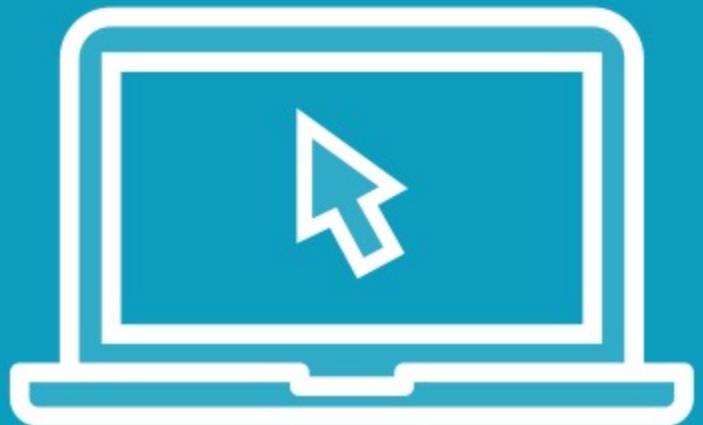
```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@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);
  }
}
```

Demo



Setting up an HTTP request

Exception Handling

product.service.ts

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

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) {
}
```

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)
  );
}
```

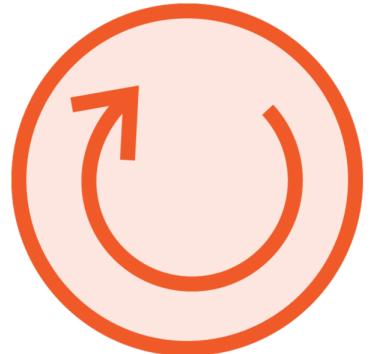
product-list.component.ts

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

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();
}
```

Demo



Subscribing to an Observable
Unsubscribing from an Observable

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

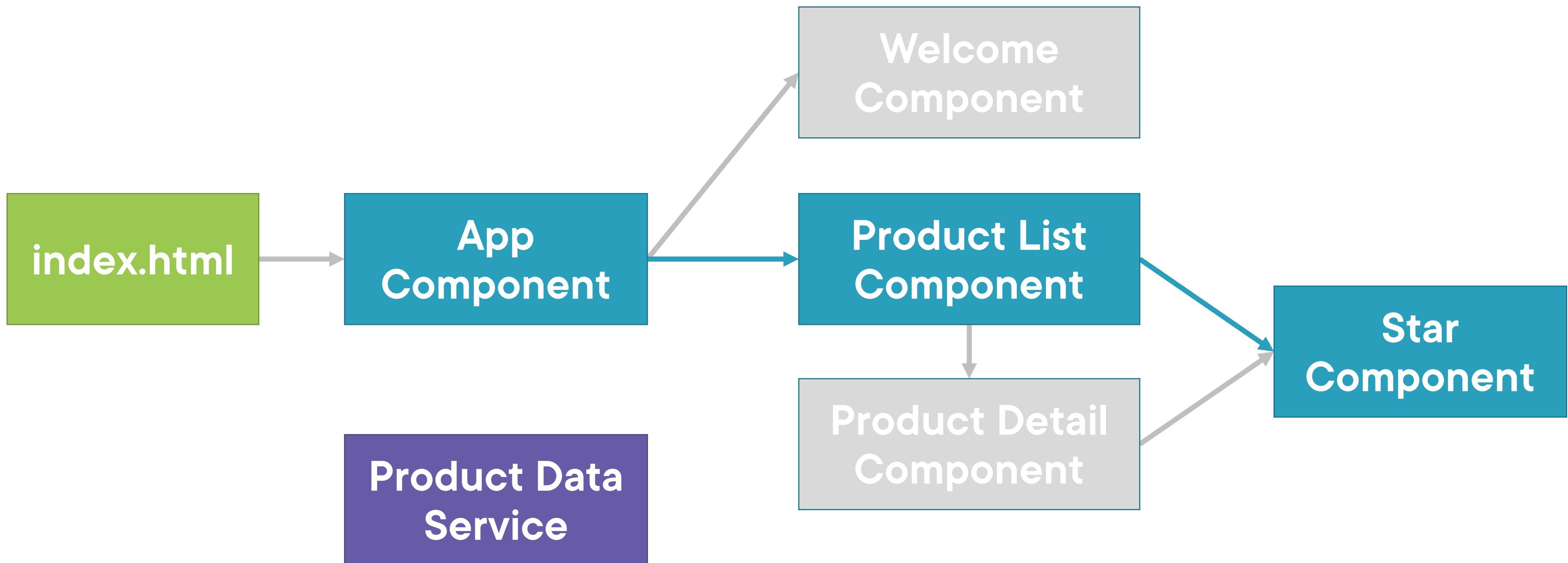
"RxJS in Angular: Reactive Development"

RxJS and Observables

"Angular HTTP Communication"

Intermediate HTTP Techniques

Application Architecture





Coming up next ...

Navigation and Routing Basics

Navigation and Routing Basics



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview



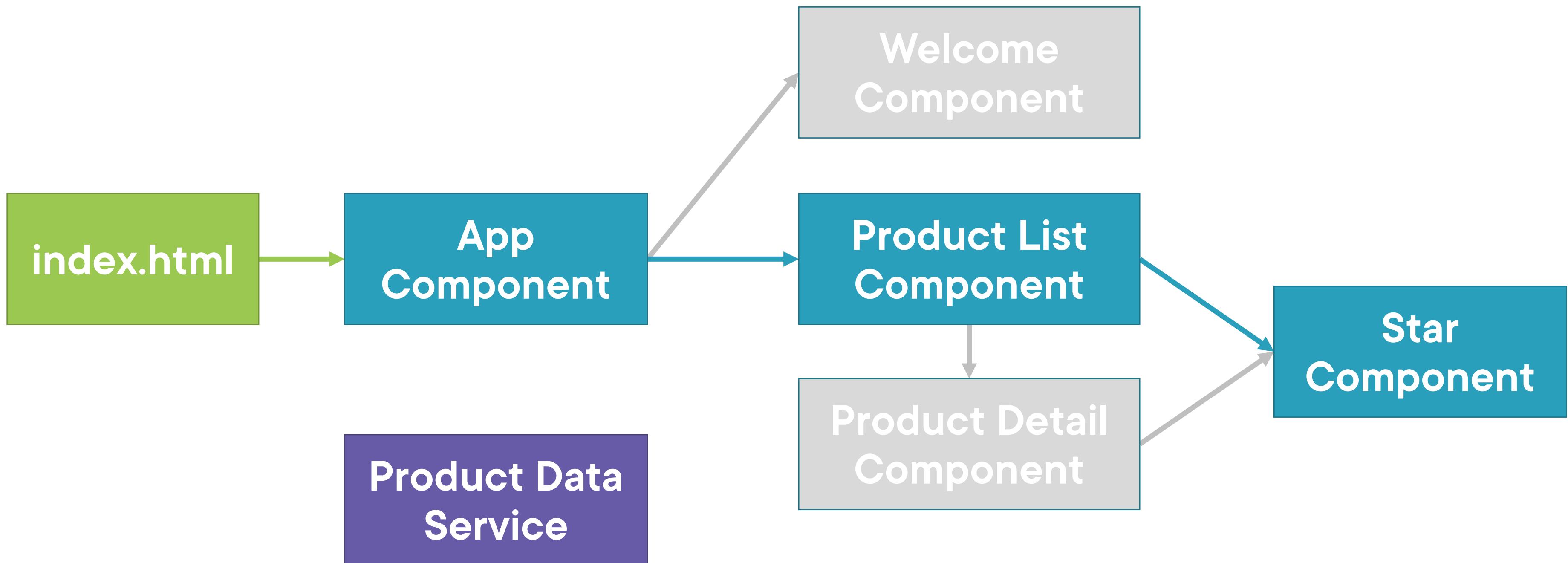
How does routing work?

Configuring routes

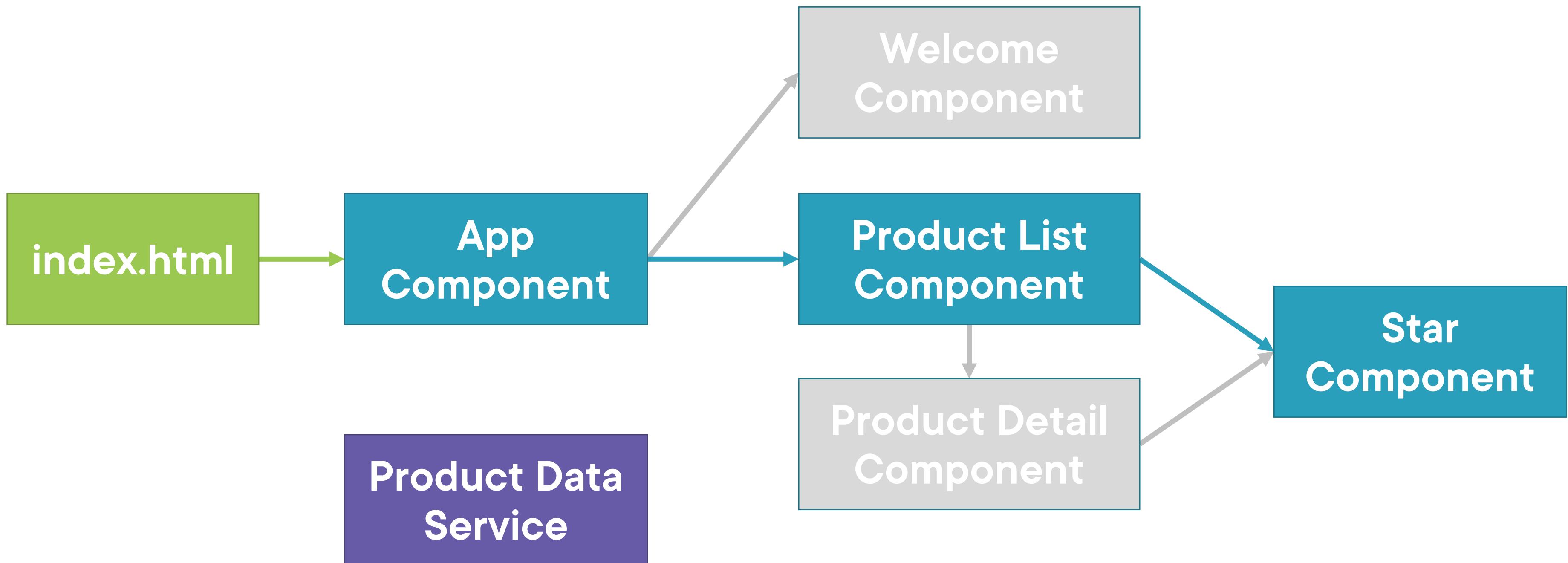
Tying routes to actions

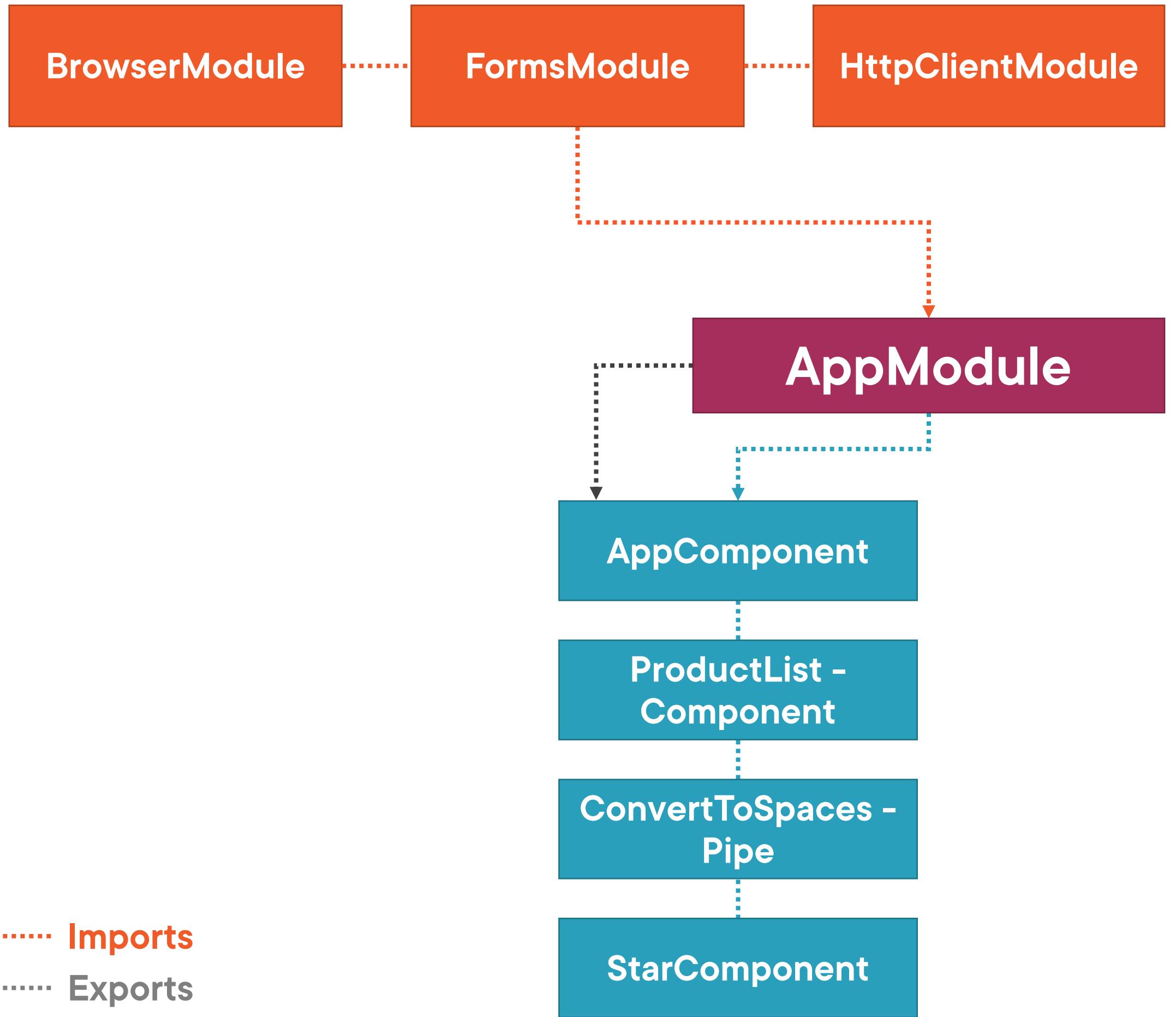
Placing the views

Application Architecture



Application Architecture



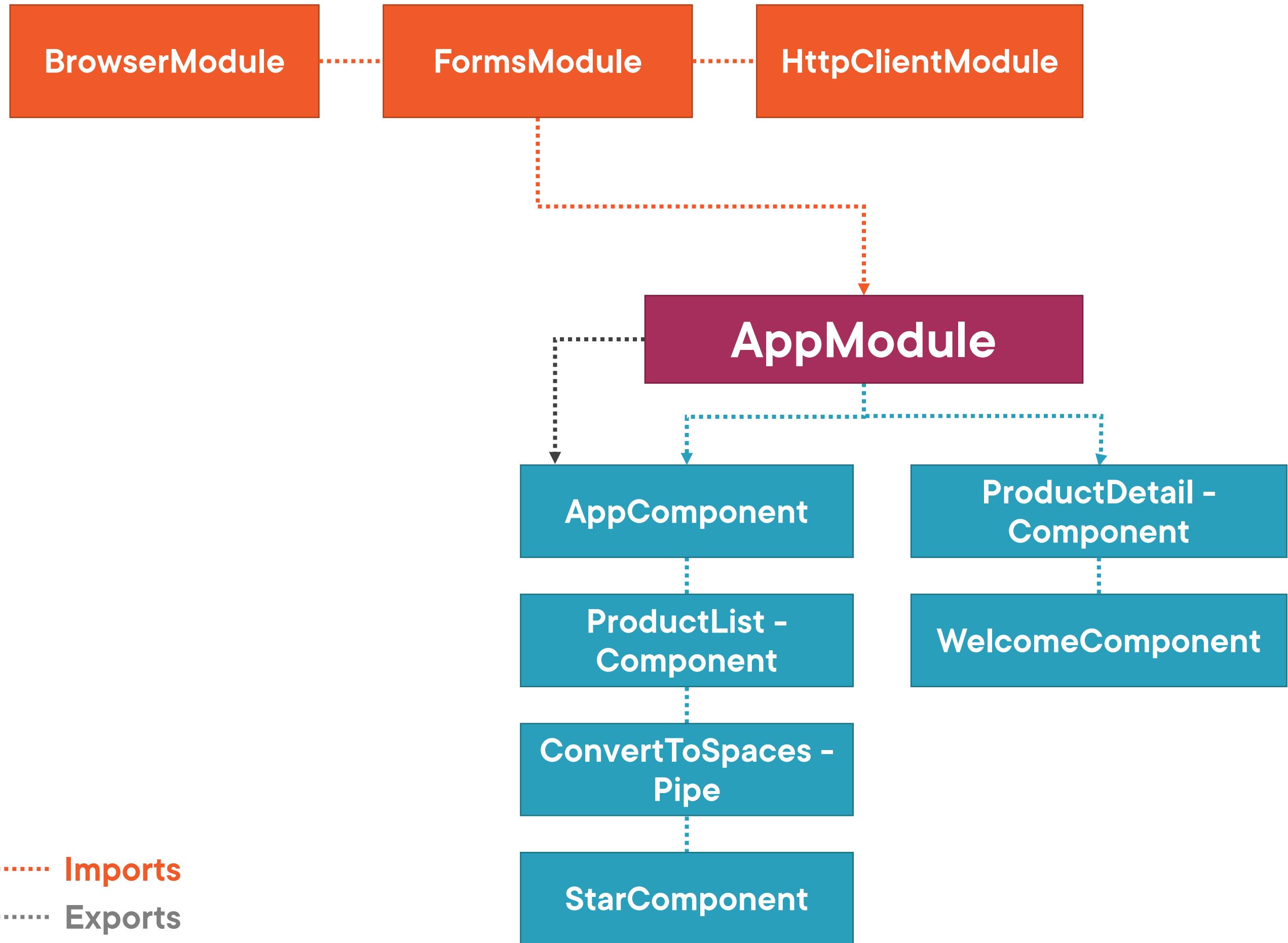


..... Imports

..... Exports

..... Declarations

..... Bootstrap



..... Imports

..... Exports

..... Declarations

..... Bootstrap

How Routing Works

```
▼<pm-root ng-version="12.0.2">
  ▶<nav class="navbar navbar-expand navbar-light bg-light">...</nav>
  <routing></routing>
  ▼<ng-component _nghost-jvs-c47>
    ▼<div _ngcontent-jvs-c47 class="card">
      <div _ngcontent-jvs-c47 class="card-header"> Product List </div>
      ▼<div _ngcontent-jvs-c47 class="card-body">
        ▶<div _ngcontent-jvs-c47 class="row">...</div>
        ▶<div _ngcontent-jvs-c47 class="row">...</div>
        ▶<div _ngcontent-jvs-c47 class="table-responsive">...</div>
      </div>
    </div>
  </ng-component>
  <!--container-->
</pm-root>
```

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

Acme Product Management Home Product List

Product List

{ 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 {}
```



'products', ProductListComponent

'products/:id', ProductDetailComponent

'welcome', WelcomeComponent

Configuring Routes

app.module.ts

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

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

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

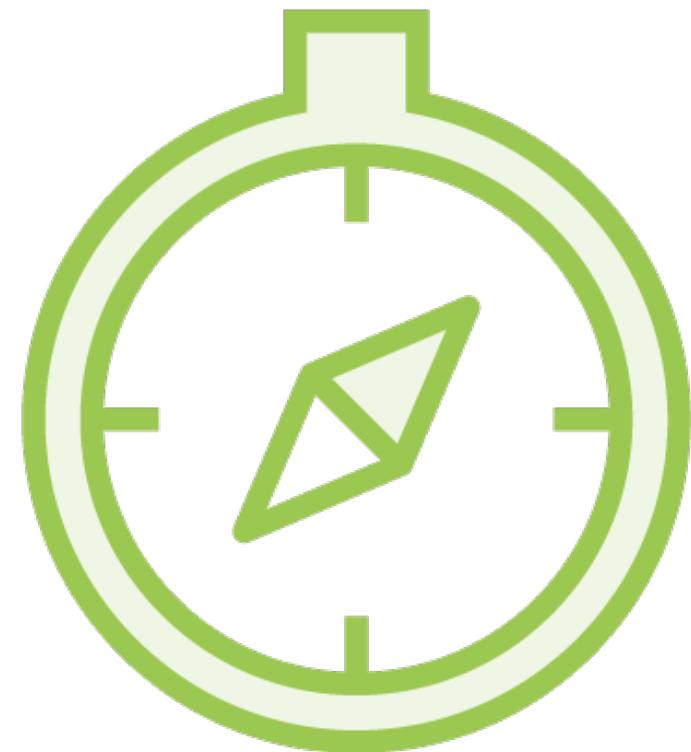
```
[  
  { path: 'products', component: ProductListComponent },  
  { path: 'products/:id', component: ProductDetailComponent },  
  { path: 'welcome', component: WelcomeComponent },  
  { path: '', redirectTo: 'welcome', pathMatch: 'full' },  
  { path: '**', component: PageNotFoundComponent }]  
]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

Demo



Configuring routes

Navigating the Application Routes

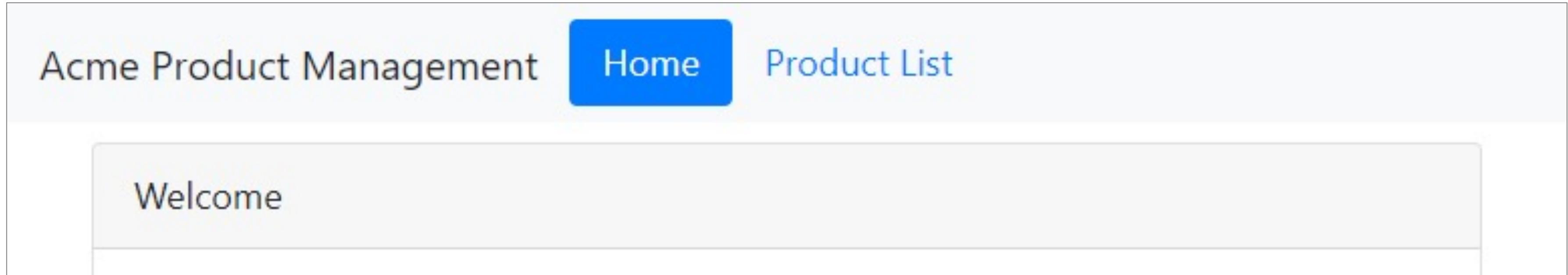


Menu option, link, image or button that activates a route

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

```
@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>
  `
})
```

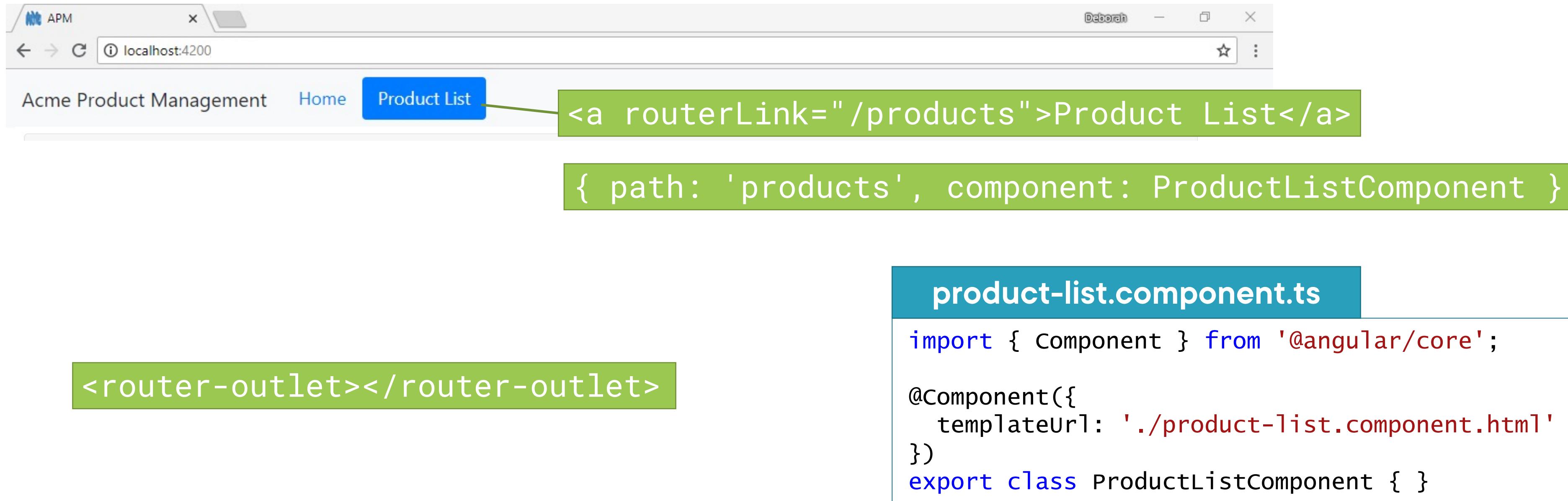
```
<li><a routerLink="/welcome">Home</a></li>
<li><a routerLink="/products">Product List</a></li>
```

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



Acme Product Management Home Product List

Product List

{ 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 {}
```

Routing Checklist: Displaying Components



Nestable 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

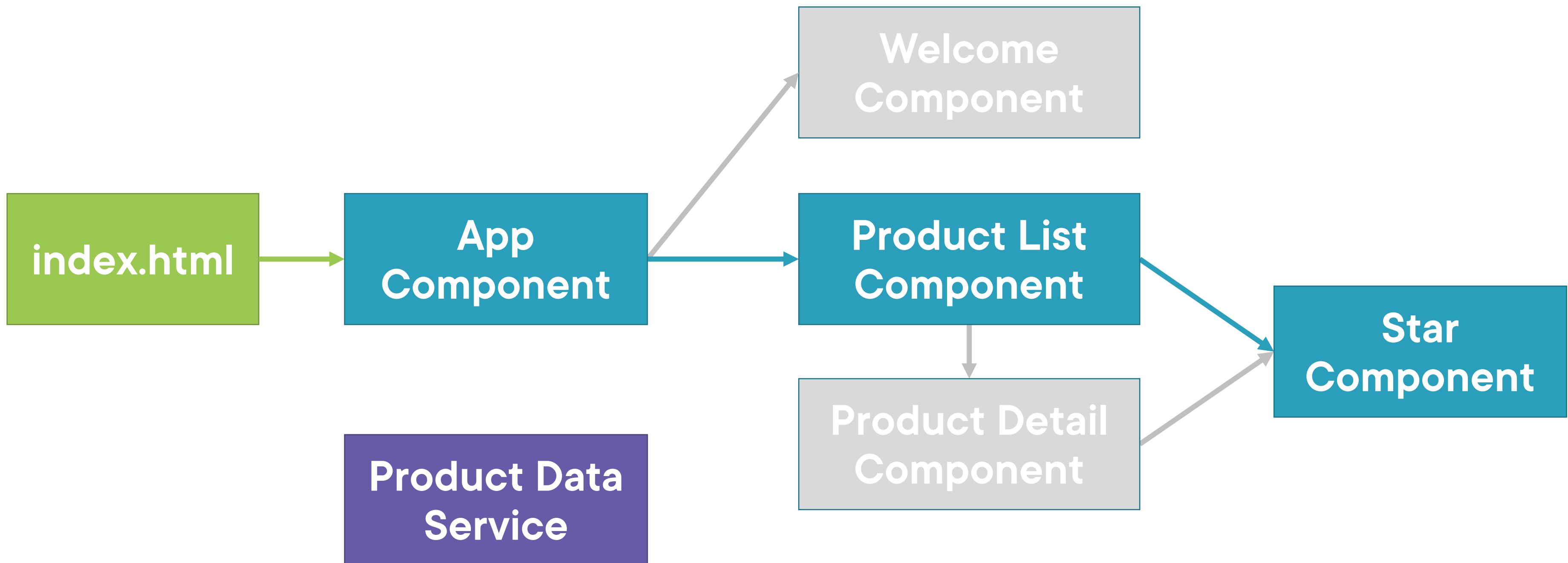


Add the RouterOutlet directive

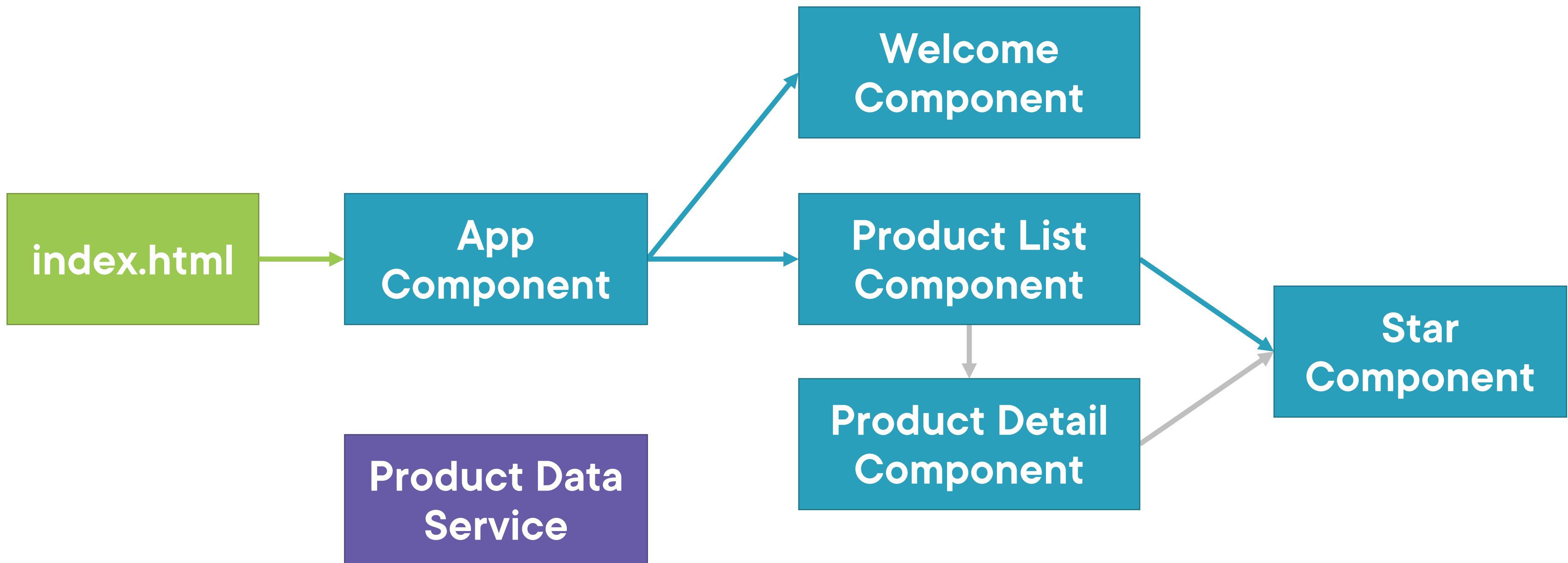
- Identifies where to display the routed component's view
- Specified in the host component's template

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

Application Architecture



Application Architecture





Coming up next ...

Navigation and Routing Additional Techniques

Navigation and Routing Additional Techniques



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview

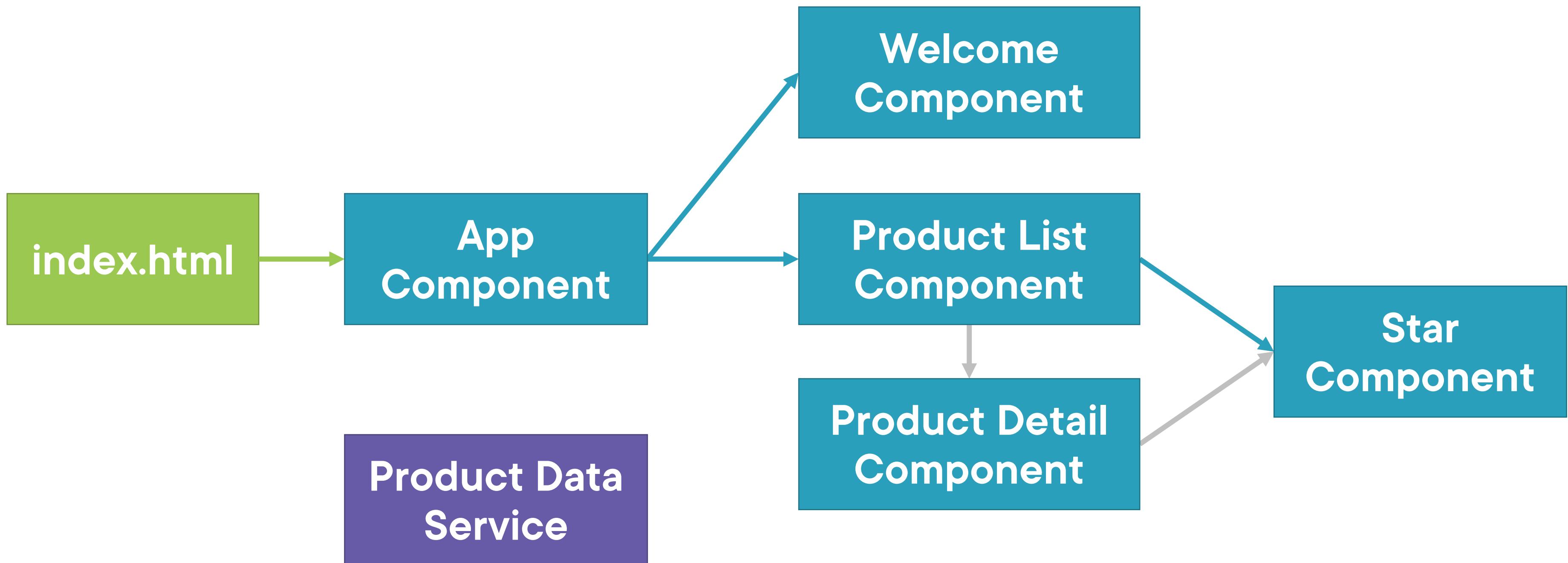


Passing parameters to a route

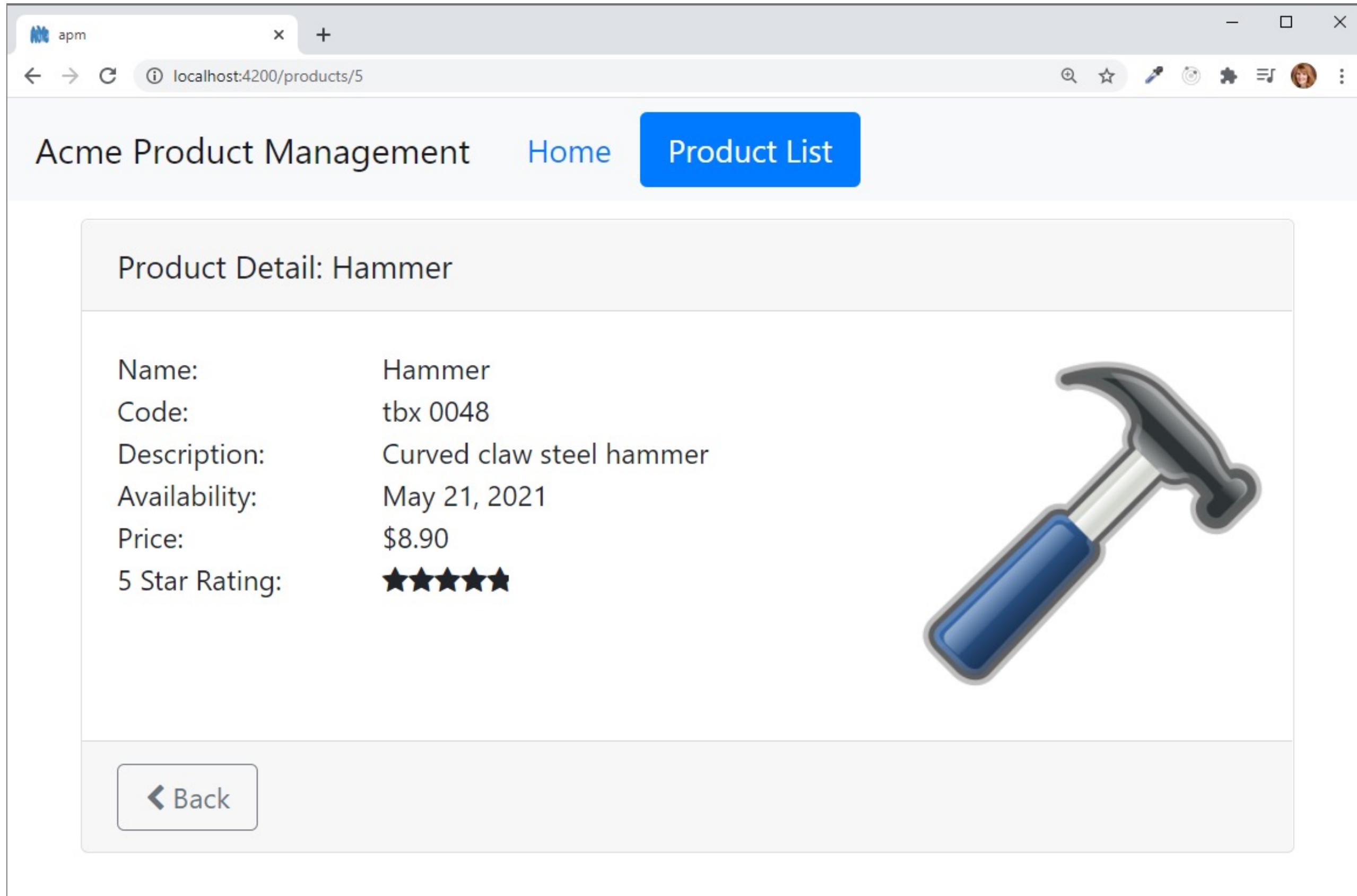
Activating a route with code

Protecting routes with guards

Application Architecture



Passing Parameters to a Route



A screenshot of a web browser window titled "apm". The address bar shows "localhost:4200/products/5". The page content is from "Acme Product Management". It features a navigation bar with "Home" and "Product List" buttons. The main area displays "Product Detail: Hammer". A table lists product information:

Name:	Hammer
Code:	tbx 0048
Description:	Curved claw steel hammer
Availability:	May 21, 2021
Price:	\$8.90
5 Star Rating:	★★★★★

To the right of the table is an image of a hammer. At the bottom left is a "Back" button.

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

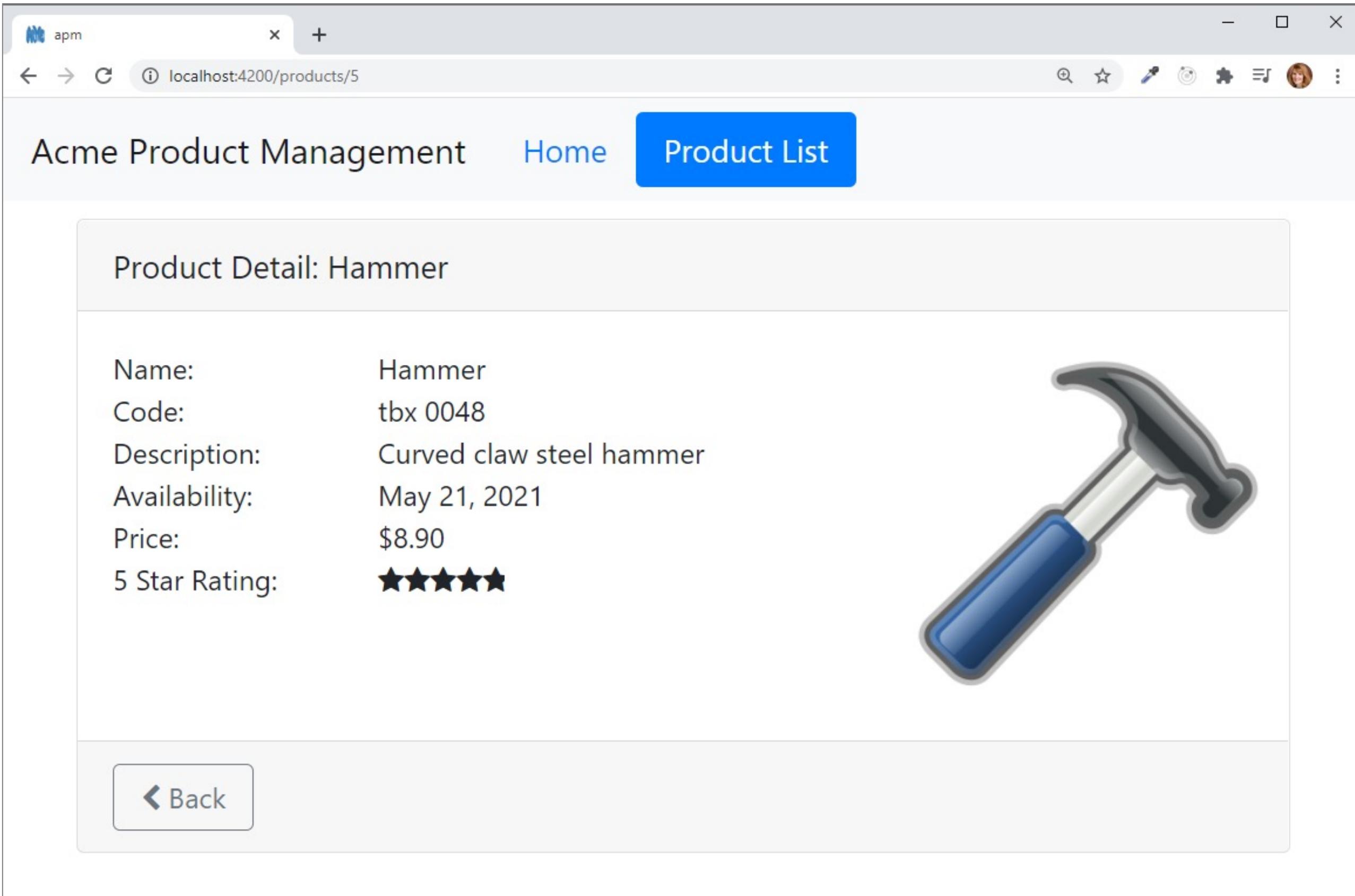
product-list.component.html

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

app.module.ts

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

Reading Parameters from a Route



A screenshot of a web browser window titled "apm". The address bar shows "localhost:4200/products/5". The page content is from "Acme Product Management". It features a navigation bar with "Home" and "Product List" buttons. The main area displays "Product Detail: Hammer". A table lists product information:

Name:	Hammer
Code:	tbx 0048
Description:	Curved claw steel hammer
Availability:	May 21, 2021
Price:	\$8.90
5 Star Rating:	★★★★★

To the right of the table is an image of a hammer. At the bottom left is a "Back" button.

Reading Parameters from a Route

product-detail.component.ts

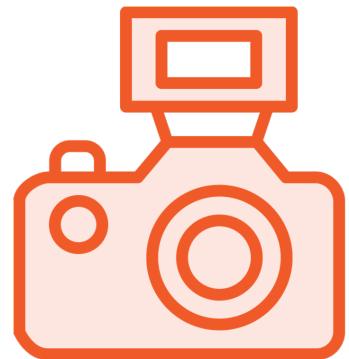
```
import { ActivatedRoute } from '@angular/router';

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

app.module.ts

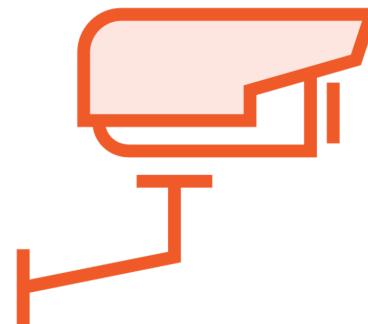
```
{ path: 'products/:id', component: ProductDetailComponent }
```

Reading Parameters from a Route



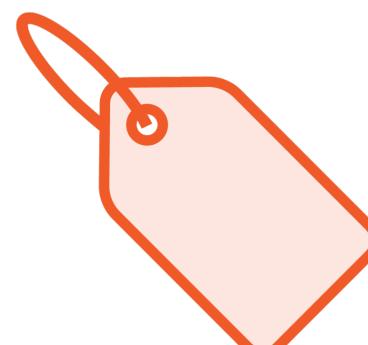
Snapshot : Read the parameter one time

```
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 }
```

Demo



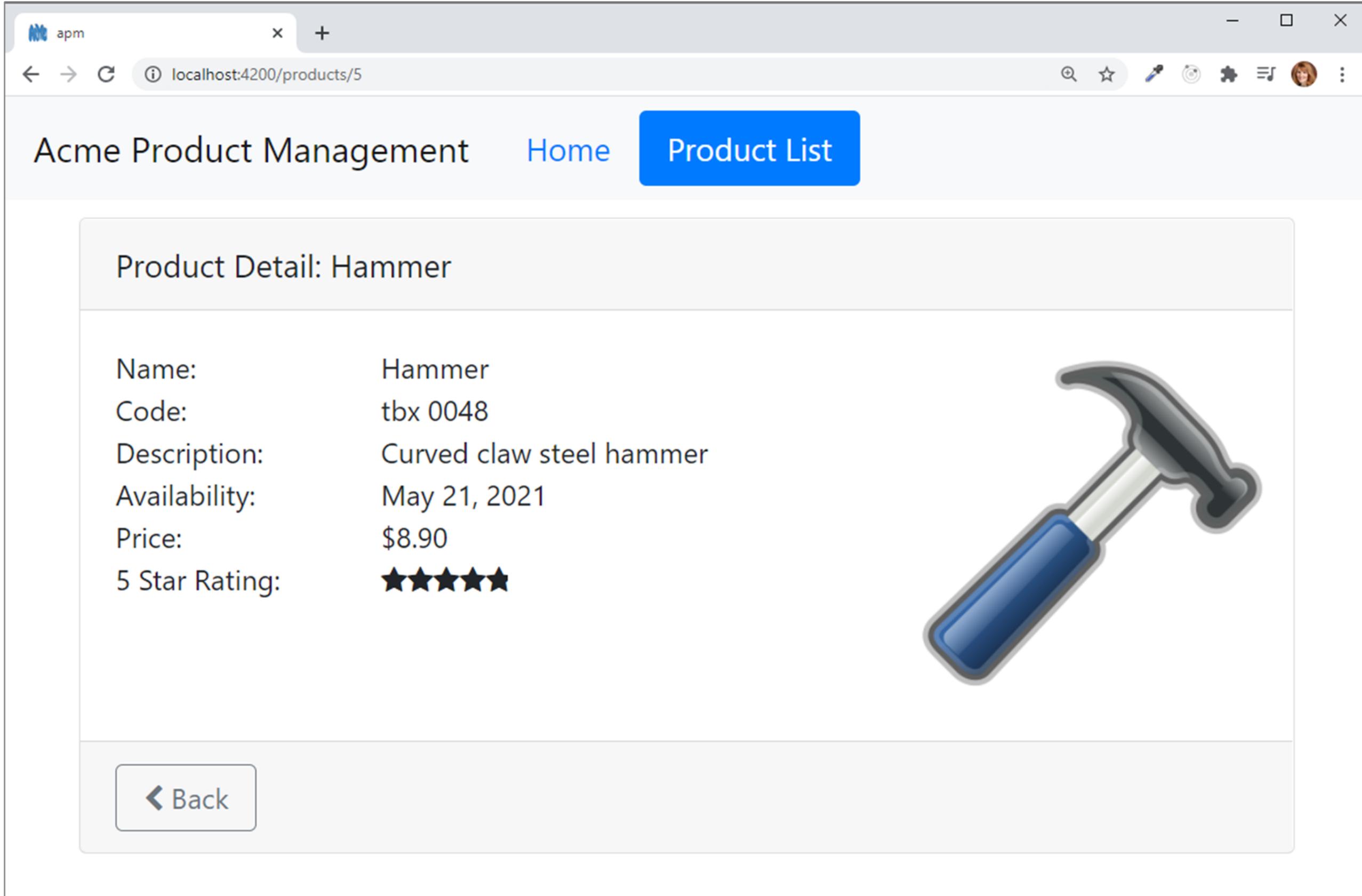
Passing parameters to a route

Demo



Handling null and undefined

Activating a Route with Code



A screenshot of a web browser window titled "apm" showing a product detail page. The URL in the address bar is "localhost:4200/products/5". The page header includes "Acme Product Management", "Home", and "Product List" buttons. The main content area displays "Product Detail: Hammer" and a table of product information:

Name:	Hammer
Code:	tbx 0048
Description:	Curved claw steel hammer
Availability:	May 21, 2021
Price:	\$8.90
5 Star Rating:	★★★★★

To the right of the table is an image of a hammer. At the bottom left is a "Back" button.

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

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 {

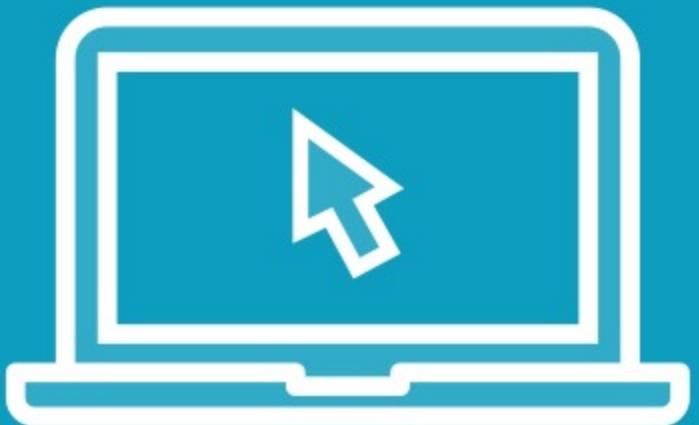
  canActivate(): boolean {
    ...
  }
}
```

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 { }
```

Demo



Protecting routes with guards

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 },
```

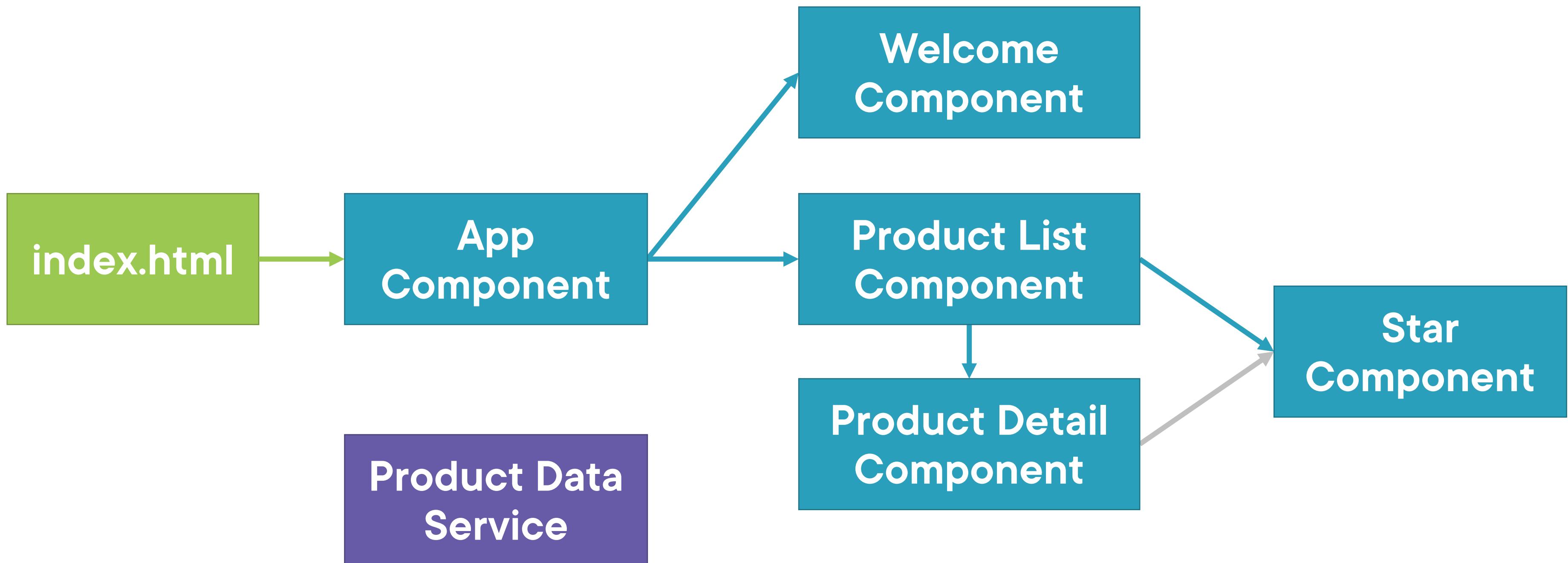
Learning More

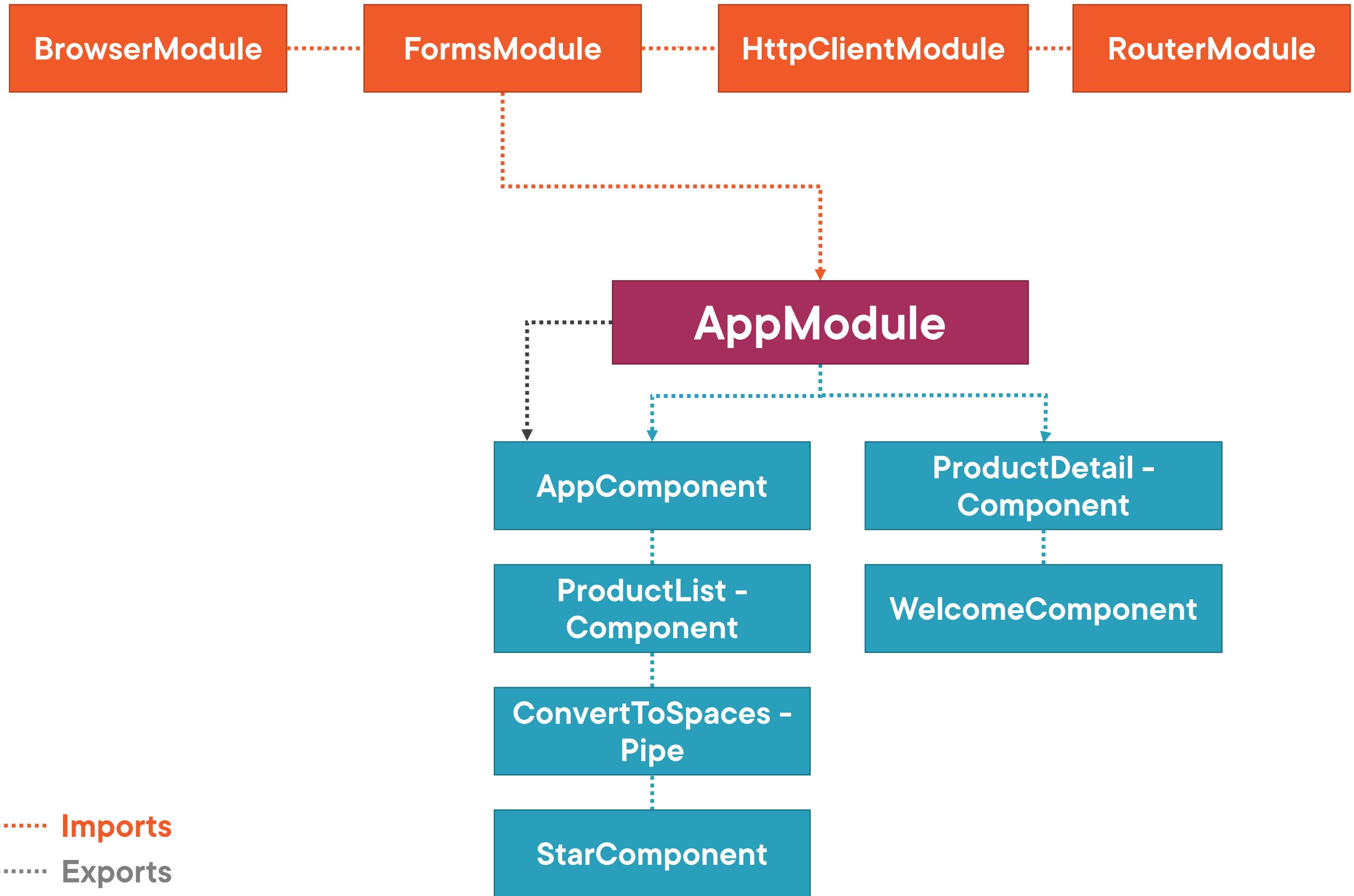


"Angular Routing"

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

Application Architecture



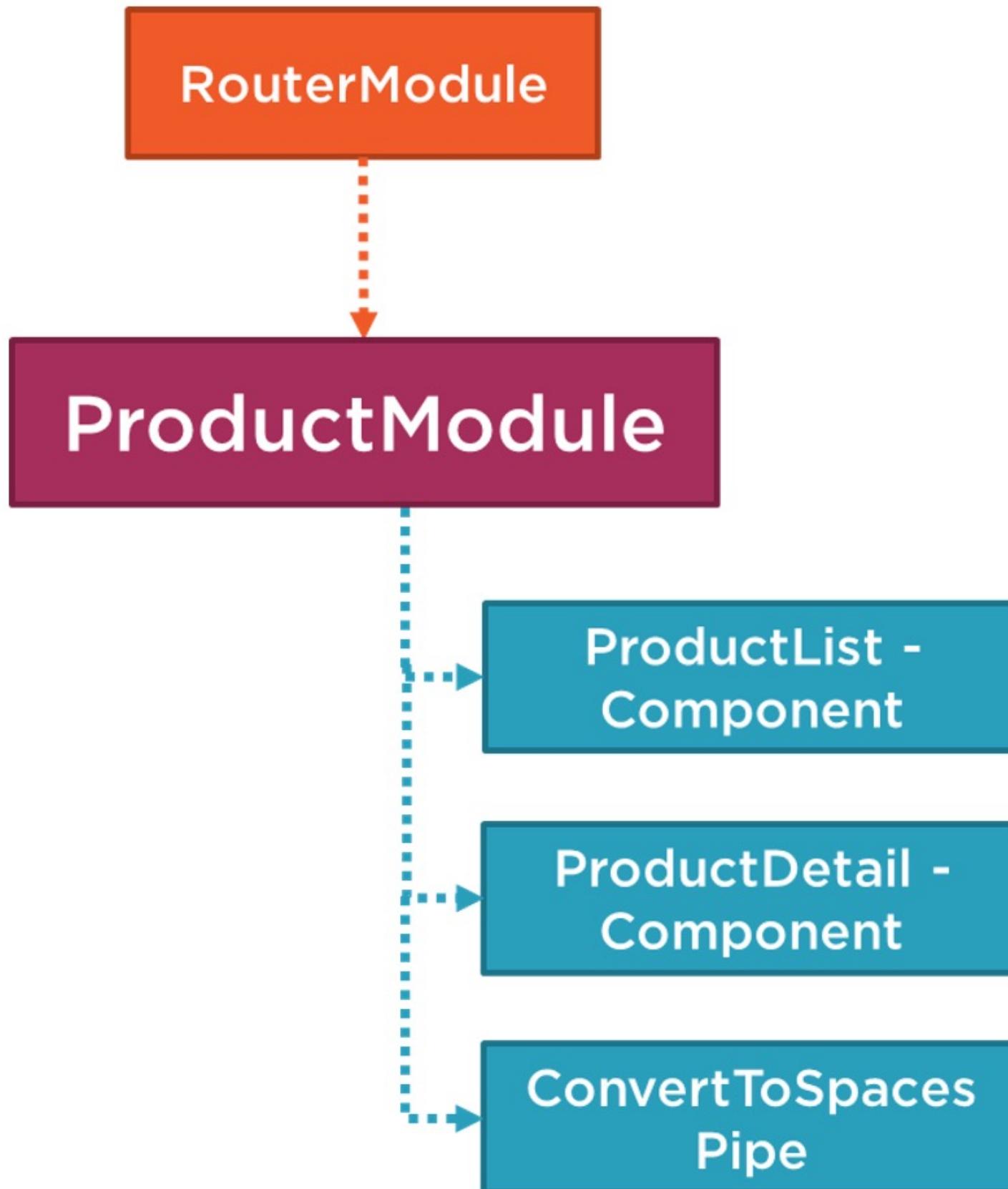


..... Imports

..... Exports

..... Declarations

..... Bootstrap



Coming up next ...

Angular Modules

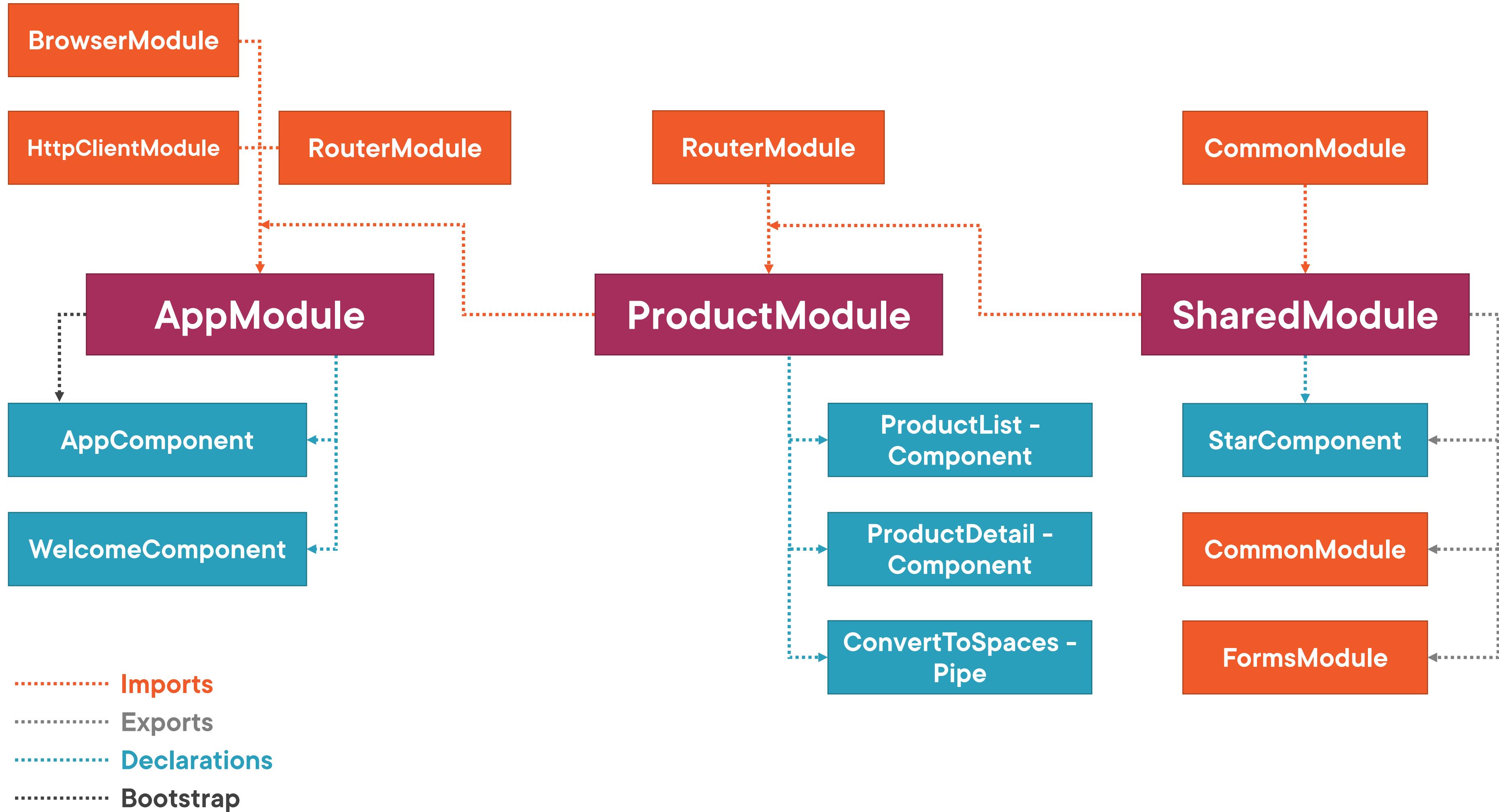
Angular Modules



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview



What is an Angular module?

Angular module metadata

Creating a feature module

Defining a shared module

Revisiting AppModule

What Is an Angular Module?

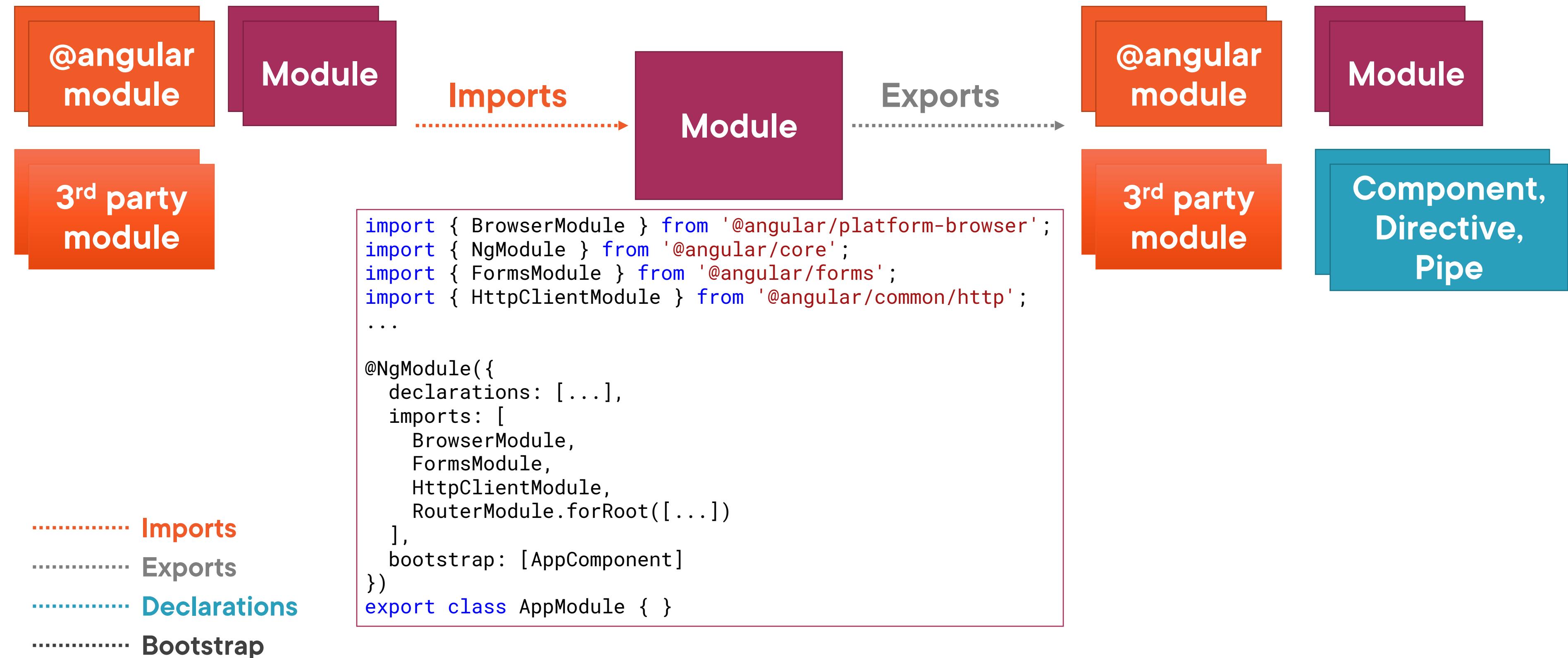
Module

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

Angular Module



AppComponent

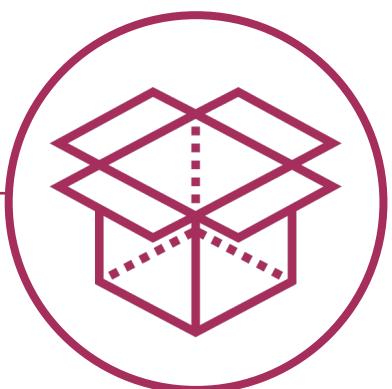
ProductList - Component

ProductDetail - Component

WelcomeComponent

RouterModule

```
...
<li><a routerLink='/welcome'>
  Home</a></li>
<li><a routerLink='/products'>
  Product List</a></li>
...
<router-outlet></router-outlet>
...
```



Angular Module

AppComponent

ProductList -
Component

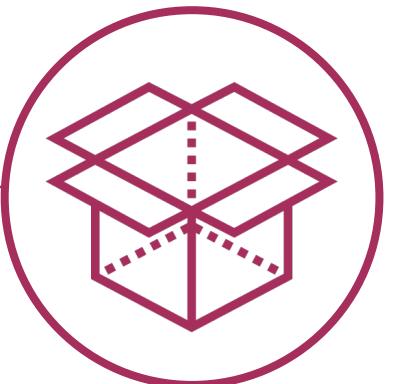
ProductDetail -
Component

WelcomeComponent

RouterModule

FormsModule

```
...  
<input type='text'  
      [(ngModel)]='listFilter' />  
...
```



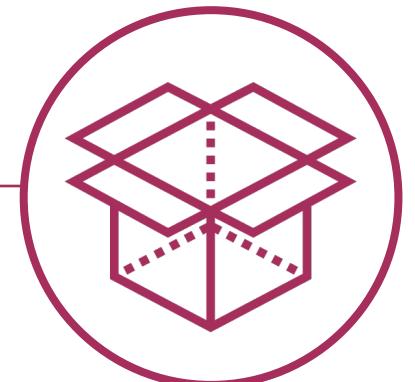
Angular Module

AppComponent

ProductList -
Component

ProductDetail -
Component

WelcomeComponent



```
...  
<tr *ngFor='let product of products'>  
  <td>  
    <img *ngIf='showImage' ...>
```

RouterModule

FormsModule

BrowserModule

Angular Module

AppComponent

ProductList - Component

ProductDetail - Component

WelcomeComponent

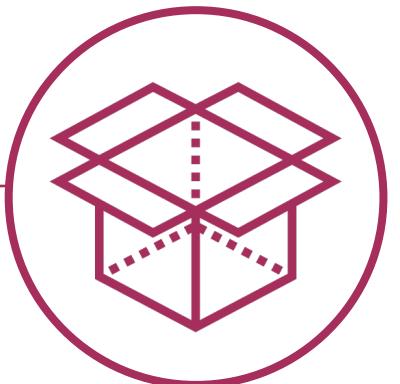
ConvertToSpaces - Pipe

RouterModule

FormsModule

BrowserModule

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



Angular Module

AppComponent

ProductList - Component

ProductDetail - Component

WelcomeComponent

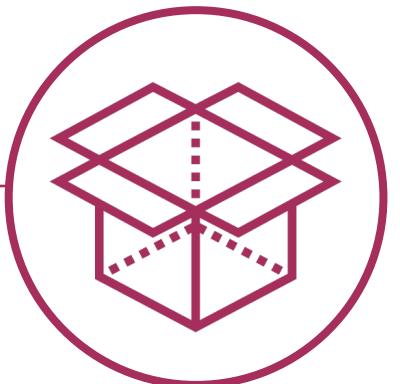
ConvertToSpaces - Pipe

RouterModule

FormsModule

BrowserModule

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



Angular Module

AppComponent

ProductList - Component

ProductDetail - Component

WelcomeComponent

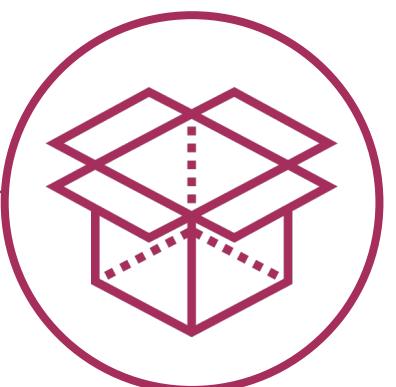
ConvertToSpaces - Pipe

StarComponent

RouterModule

FormsModule

BrowserModule



Angular Module

Bootstrap Array

AppModule

AppComponent

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

app.module.ts

```
...
bootstrap: [ AppComponent ]
...
```

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

@Component({
  selector: 'pm-root',
  template: `...`
})
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

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.

Declarations Array

Module

Declares

Component,
Directive,
Pipe

..... Declarations

```
...
declarations: [
    AppComponent,
    WelcomeComponent,
    ProductListComponent,
    ProductDetailComponent,
    ConvertToSpacesPipe,
    StarComponent
]
...
```

Declarations Array Truth #1

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

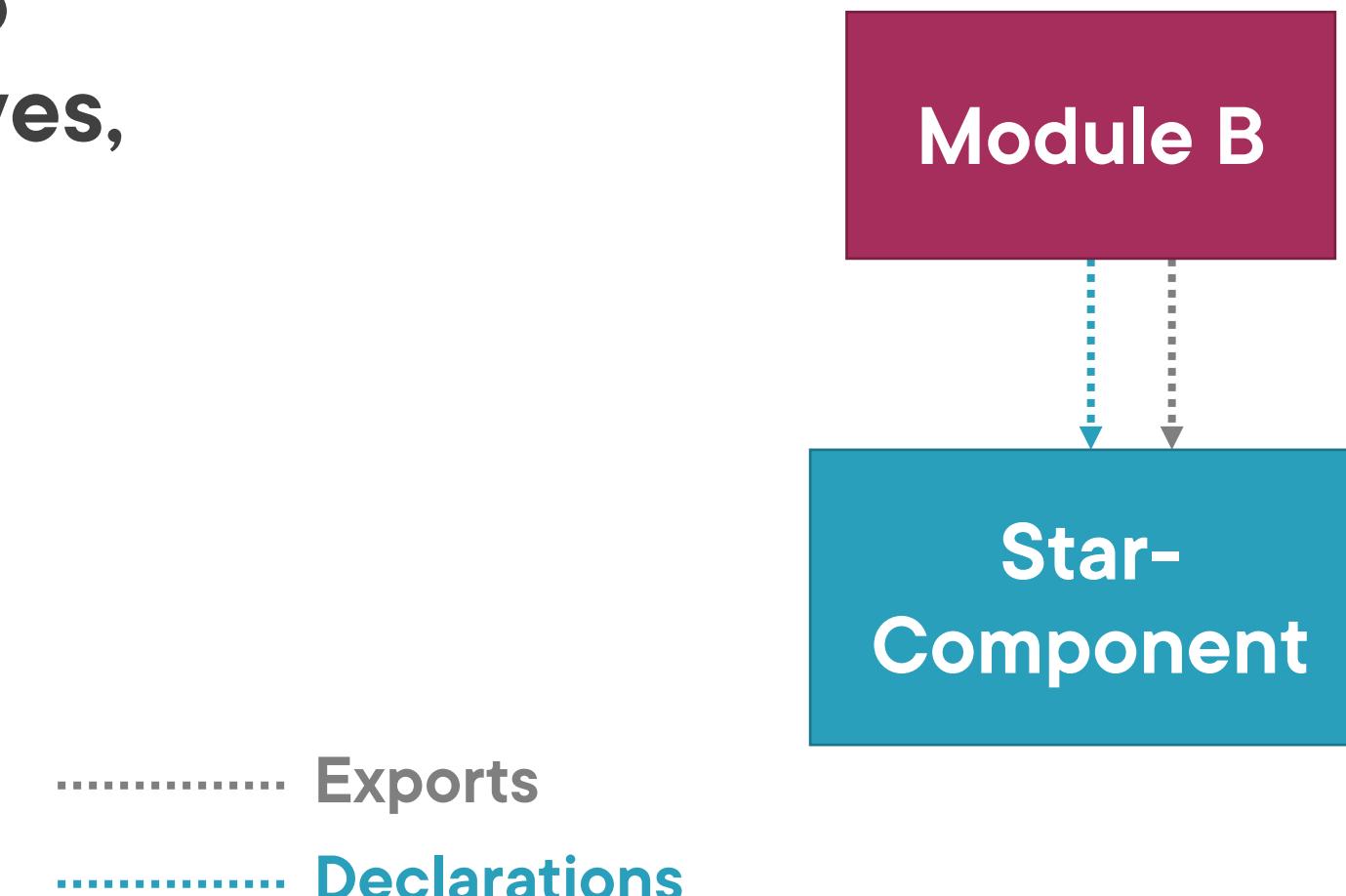
Declarations Array Truth #2

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

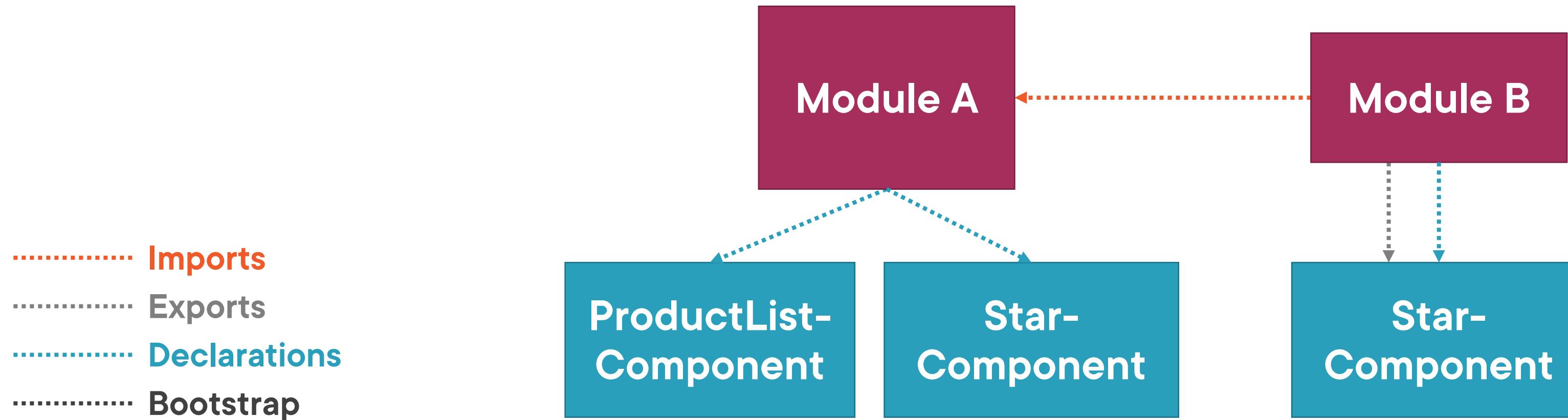
```
<pm-star ...>  
</pm-star>
```

star.component.ts

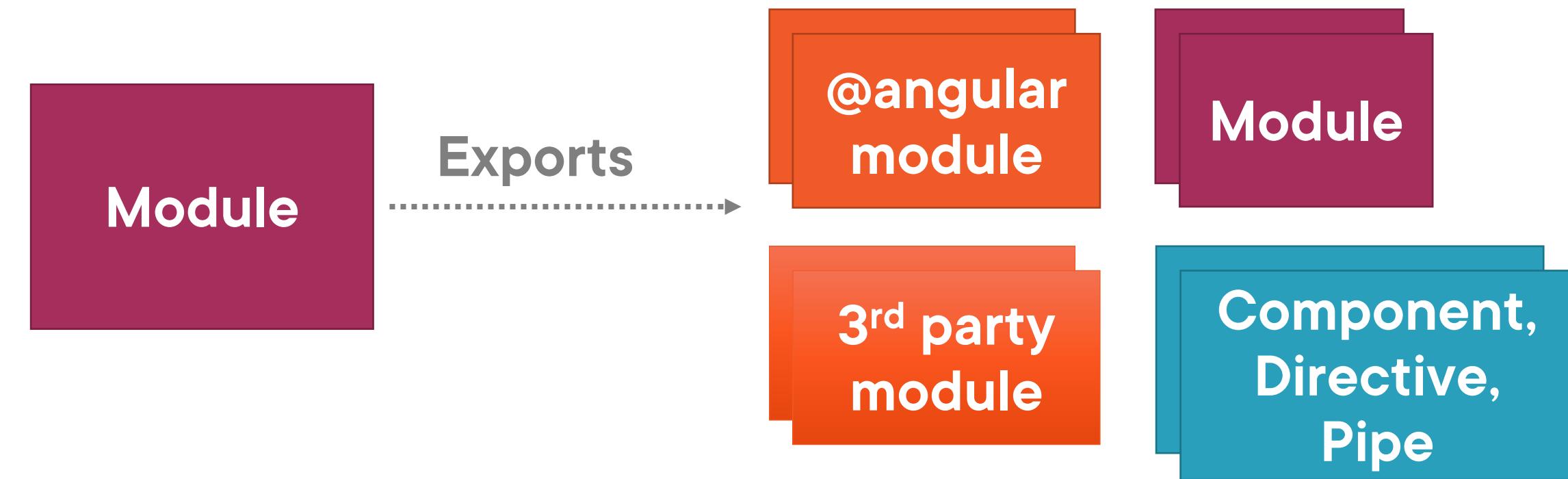
```
...  
@Component({  
  selector: 'pm-star',  
  template: ...  
})  
...
```

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

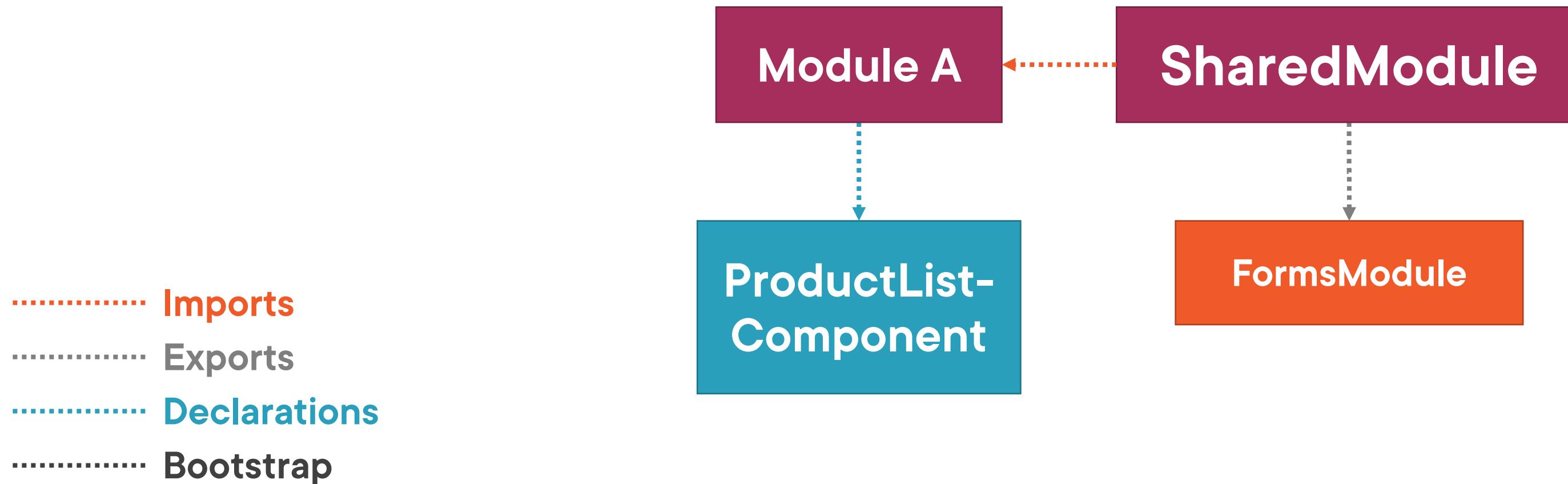
Export any component, directive, or pipe if other components need it.

Exports Array Truth #2

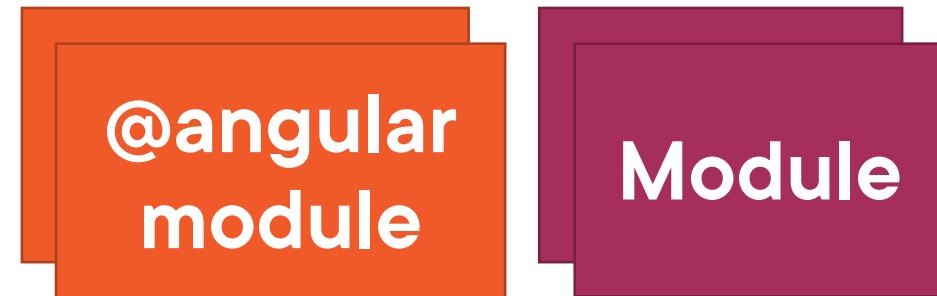
Re-export modules to re-export their components, directives, and pipes.

Exports Array Truth #3

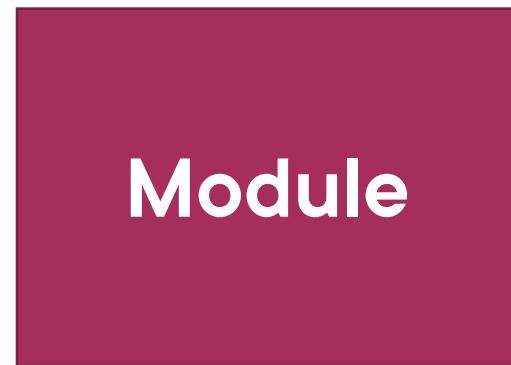
We can export something without including it in the imports array.



Imports Array



Imports



```
...
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**.



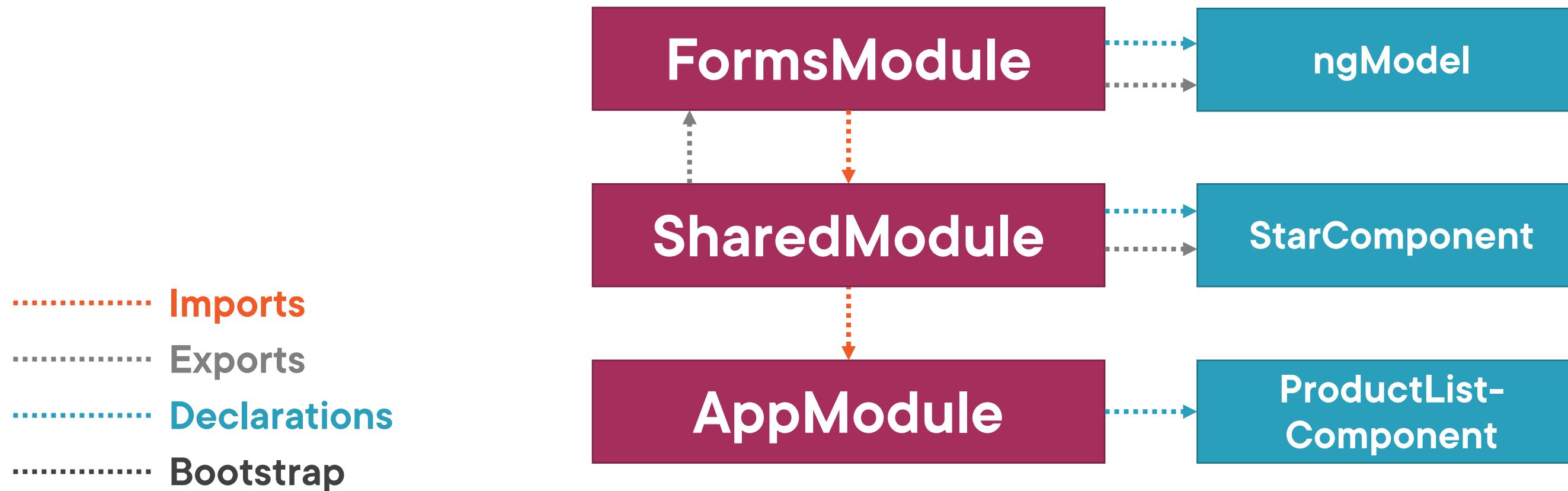
Imports Array Truth #2

Only import what this module needs.

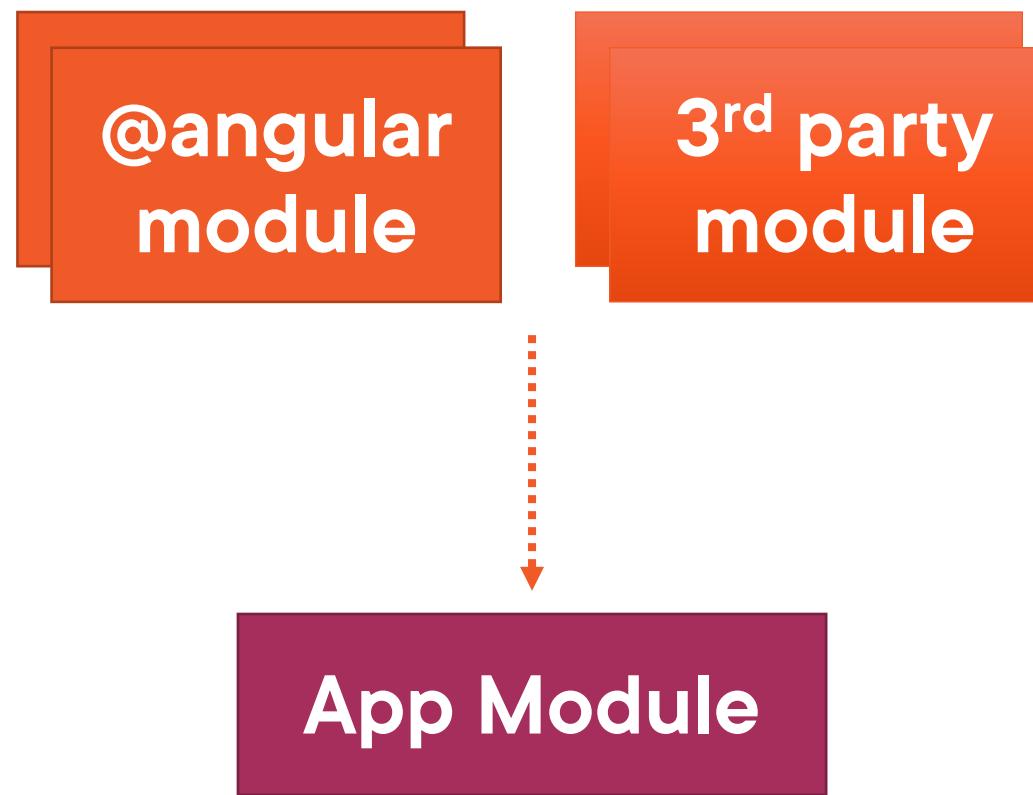
For each component declared in the module, add to the imports array what is needed by the component's template.

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

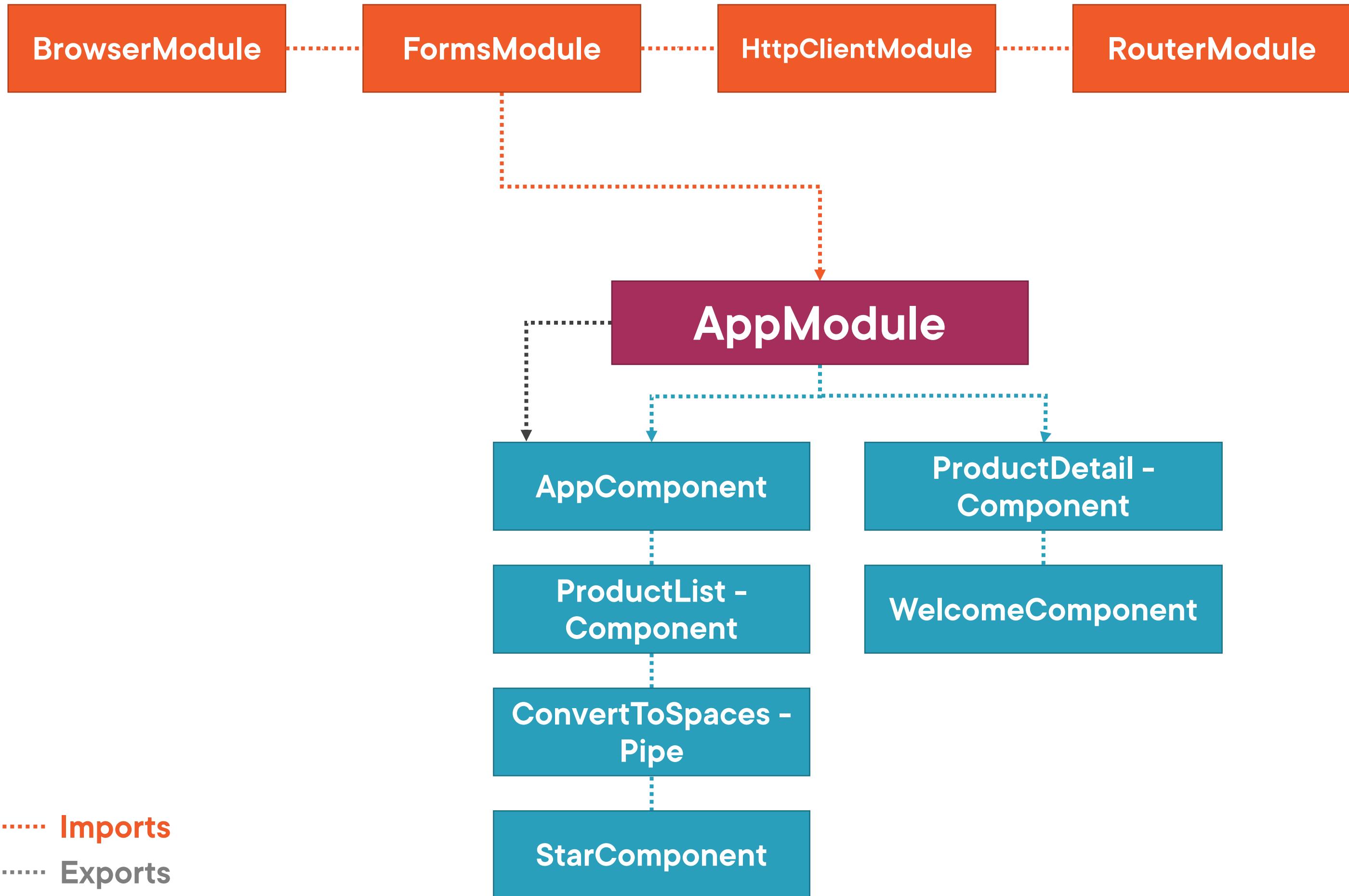
Providers Array



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 { }
```



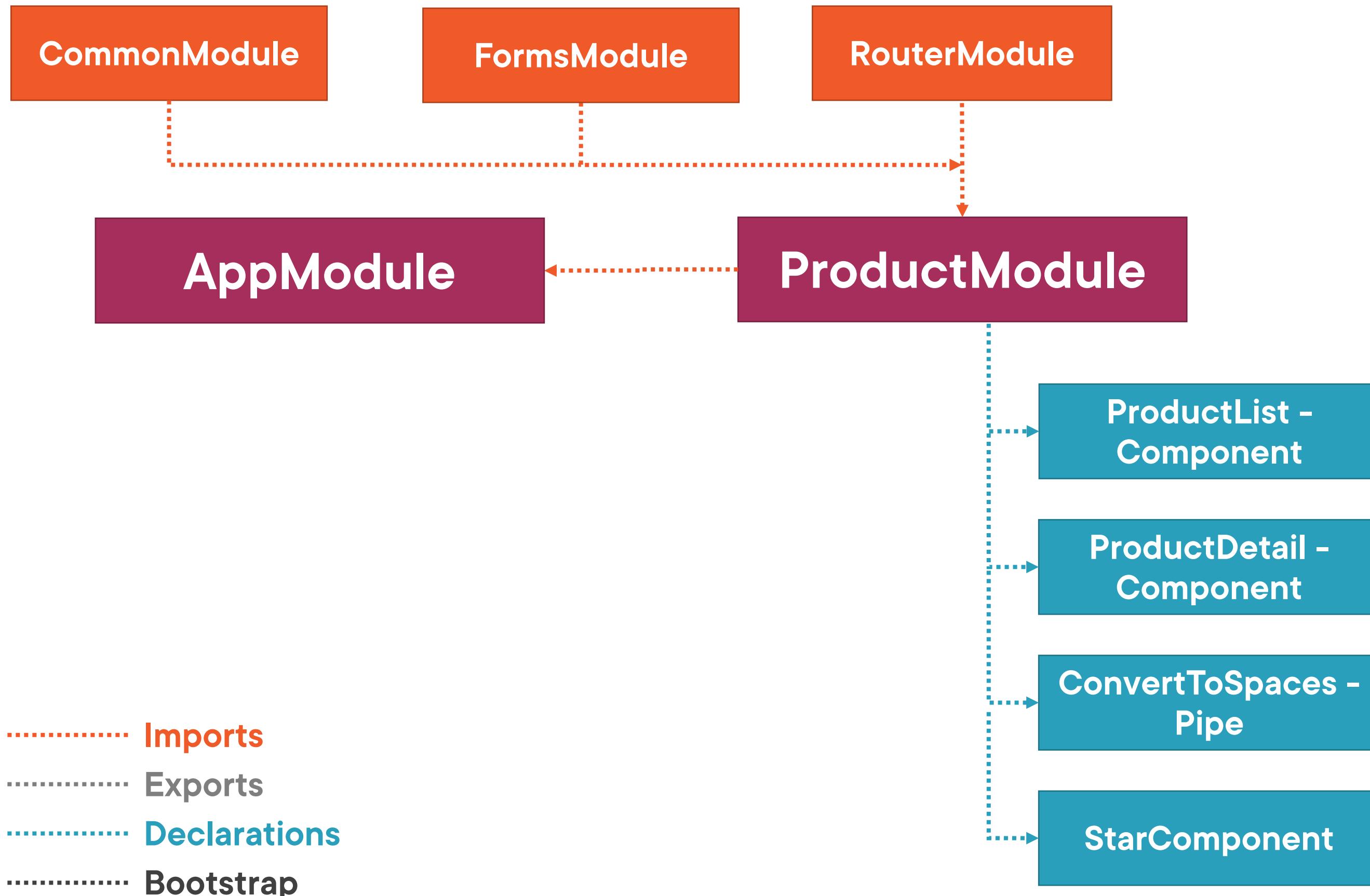
..... Imports

..... Exports

..... Declarations

..... Bootstrap

Defining a Feature Module



Demo



Building a feature module

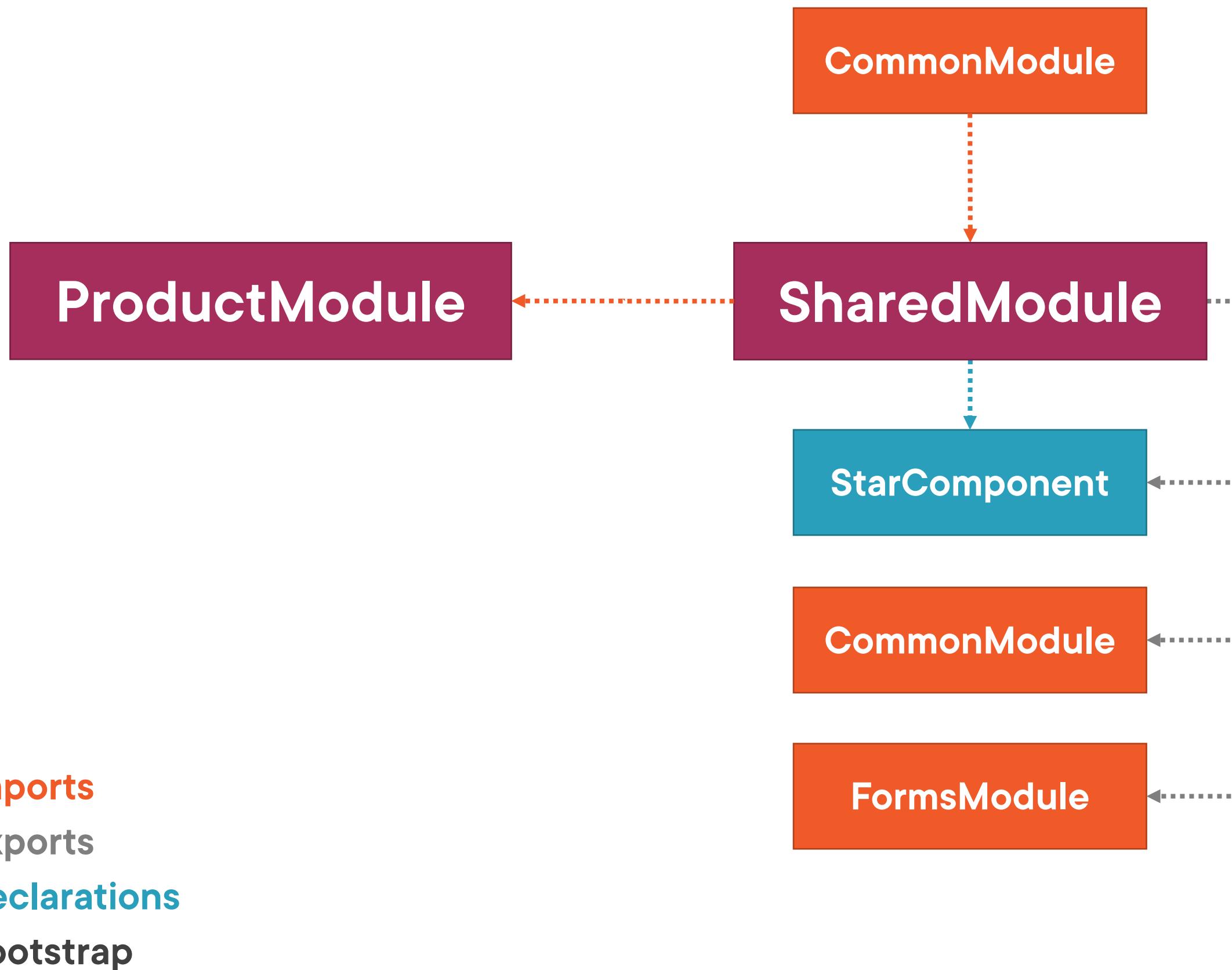
Shared Module

SharedModule

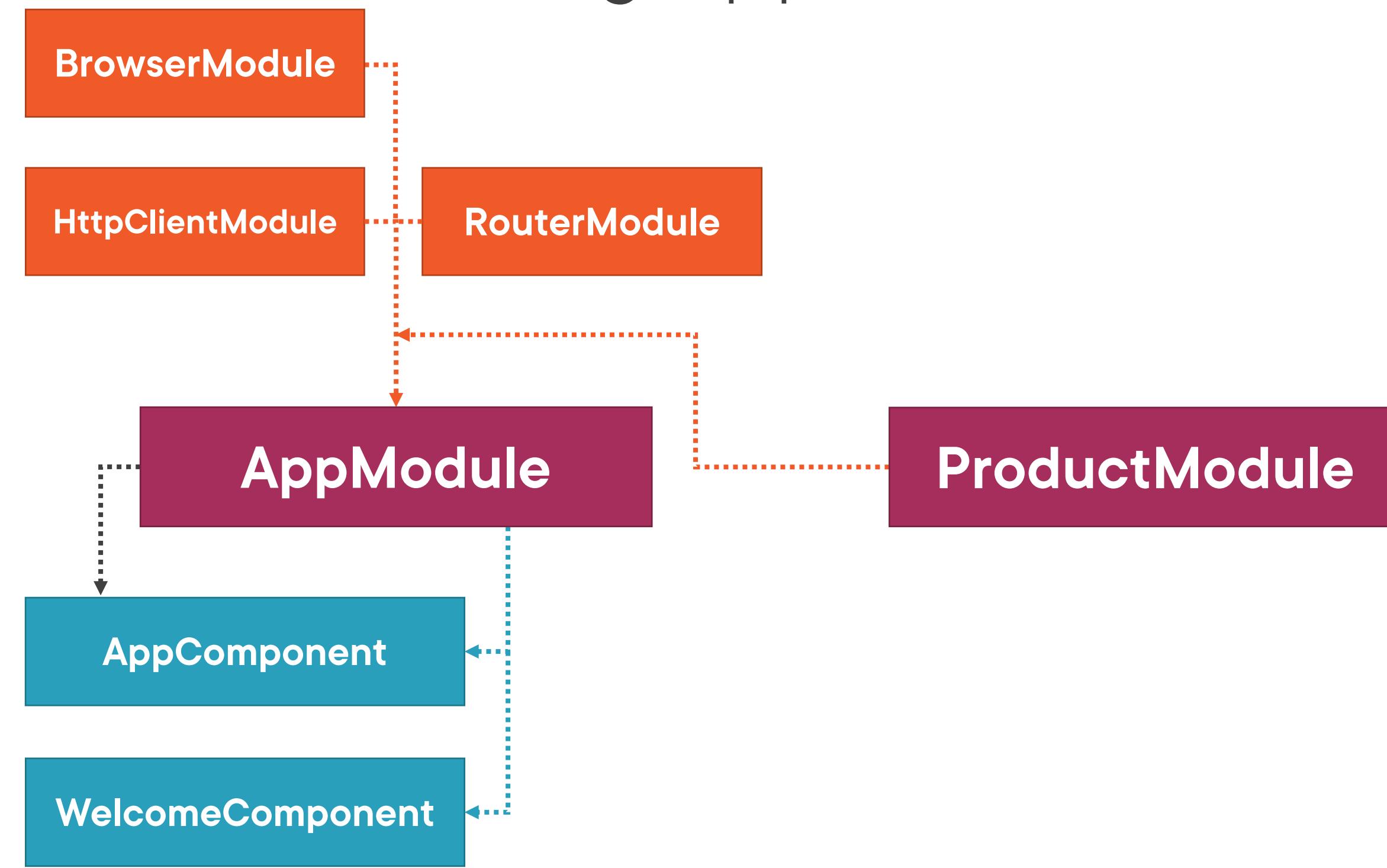
Organize commonly used pieces of our application

Export those pieces to share them

Defining a Shared Module



Revisiting AppModule



..... Imports

..... Exports

..... Declarations

..... Bootstrap

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

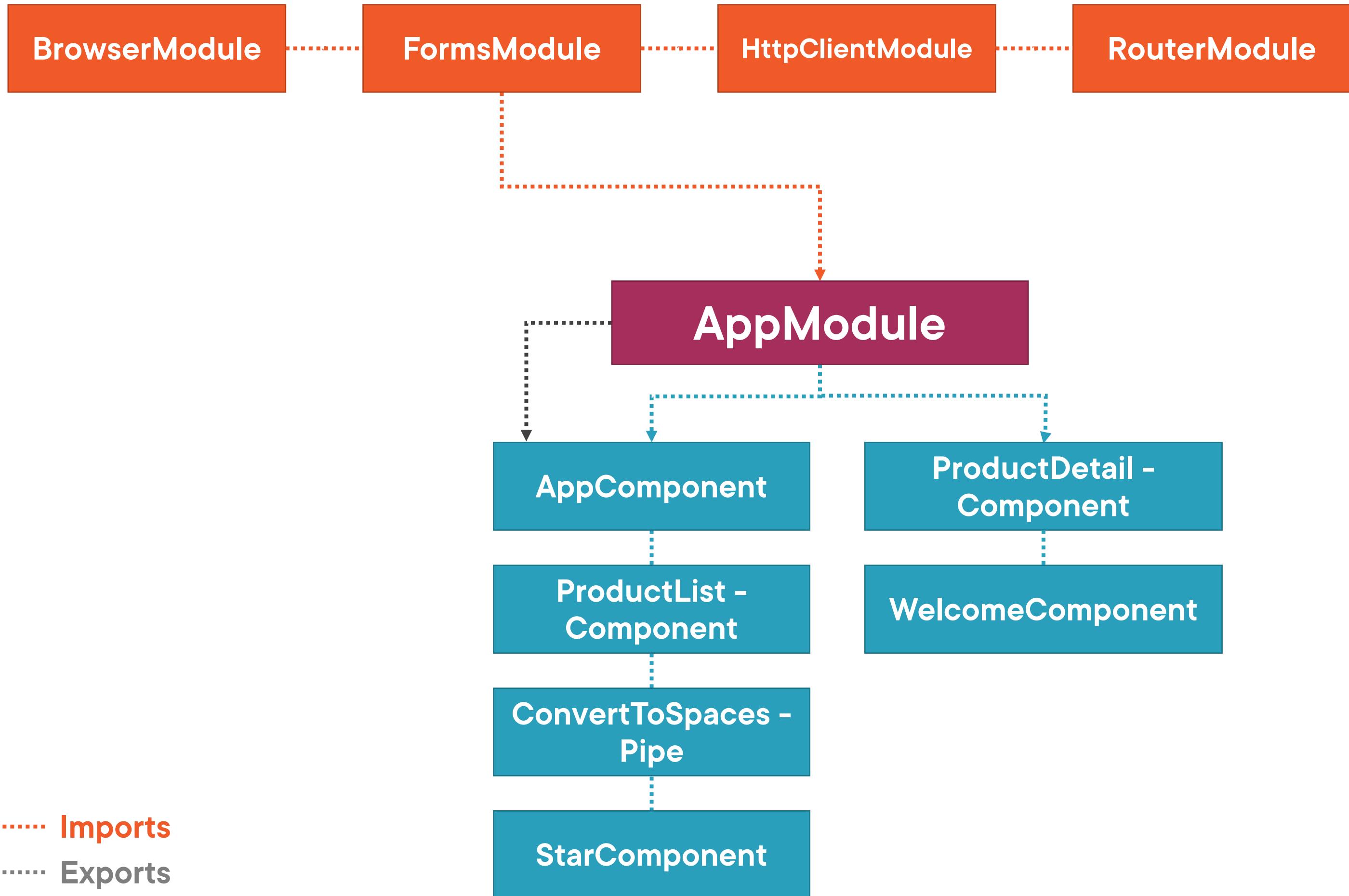


Bootstrap array: What the index.html file needs

Component with selector used in index.html

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

Normally only used by AppModule

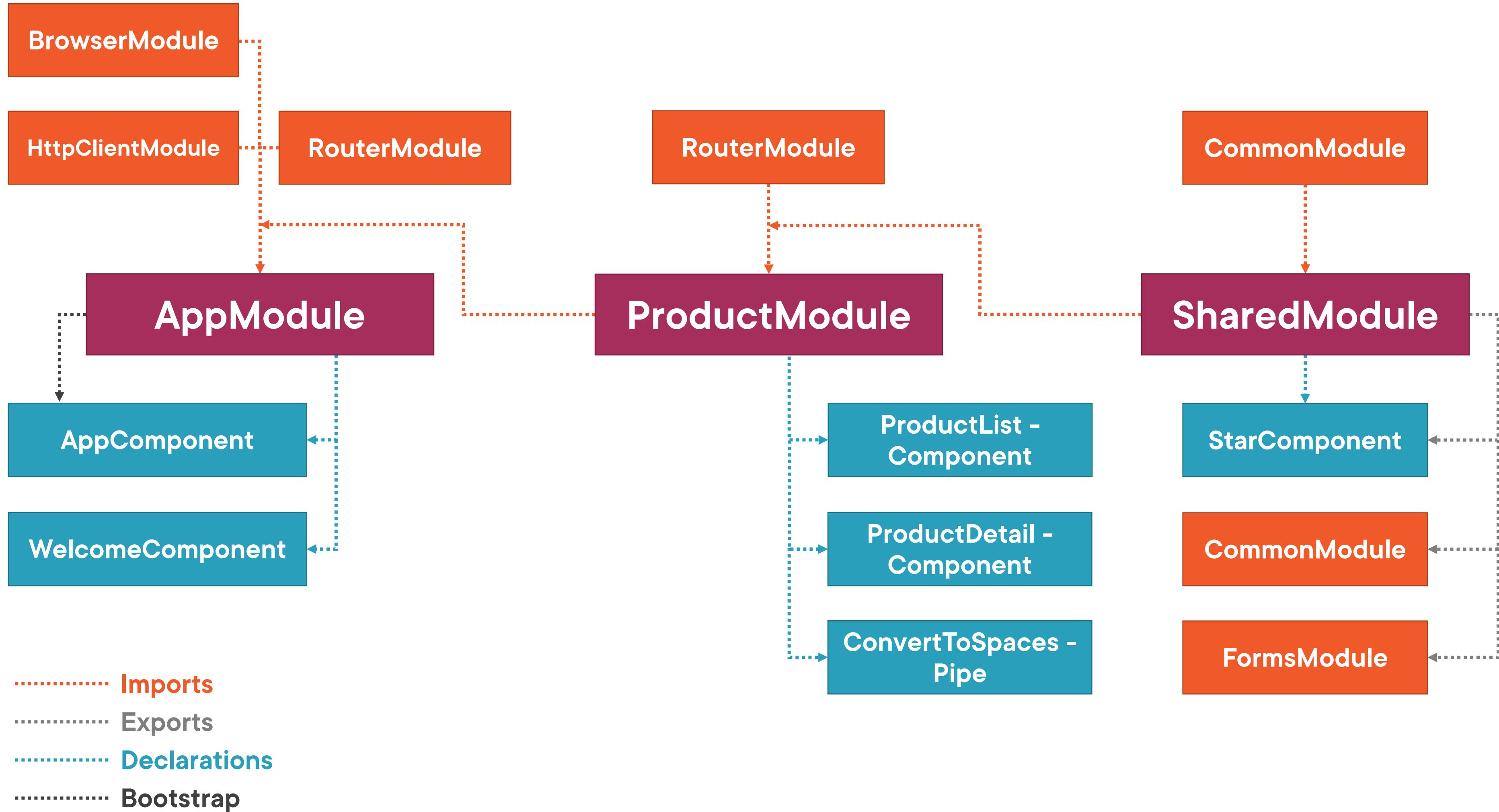


..... Imports

..... Exports

..... Declarations

..... Bootstrap





Coming up next ...

**Building, Testing, and Deploying
with the CLI**

Building, Testing, and Deploying with the CLI



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Module Overview



Overview

Generating a new Angular app (ng new)

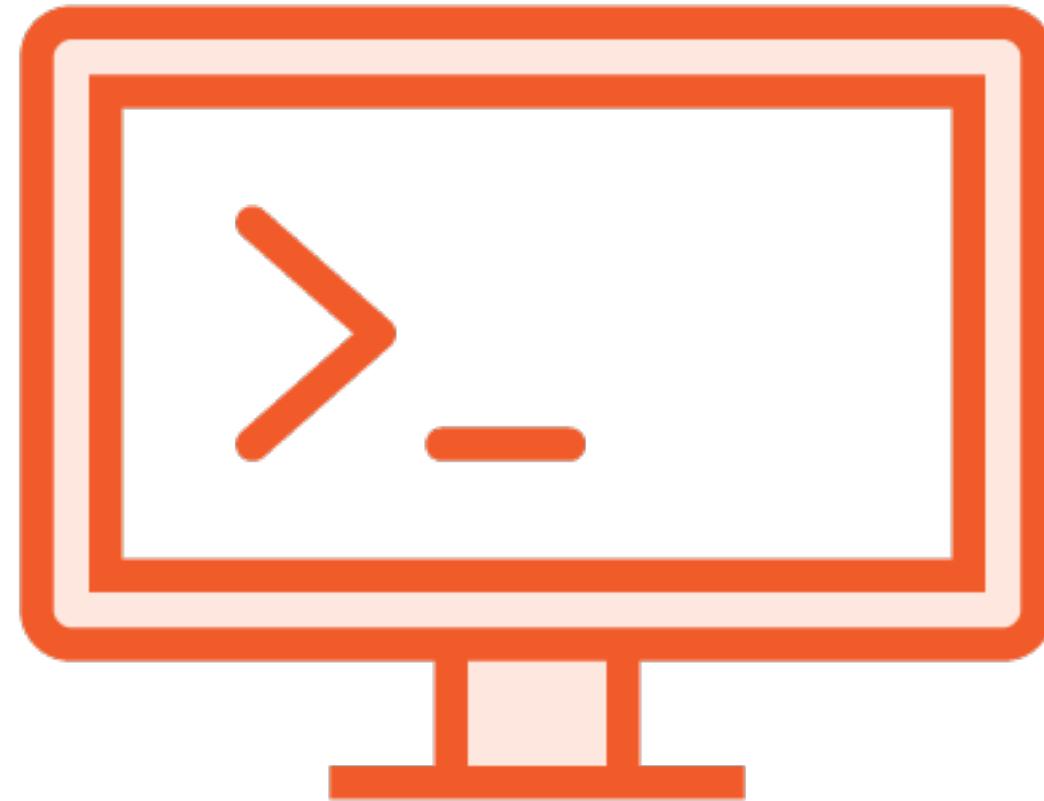
Serving the application (ng serve)

Generating code (ng generate)

Testing the application (ng test)

Building the application (ng build)

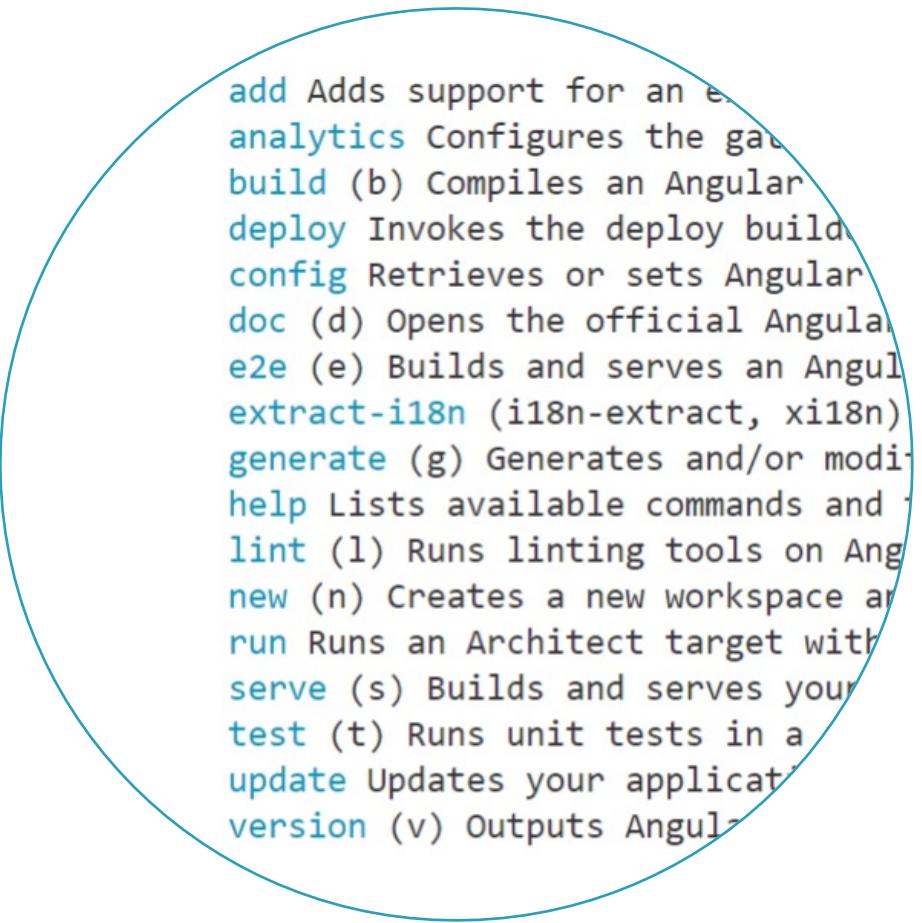
Angular CLI Overview



A command line interface for Angular

Purpose:

- Build an Angular application
- Generate Angular files
- Build and serve the application
- Run tests
- Prepare the application for deployment



```
add Adds support for an external package
analytics Configures the gathering of analytics data
build (b) Compiles an Angular application
deploy Invokes the deploy build hook
config Retrieves or sets Angular configuration
doc (d) Opens the official Angular documentation
e2e (e) Builds and serves an Angular application for end-to-end testing
extract-i18n (i18n-extract, xi18n) Extracts i18n strings from your codebase
generate (g) Generates and/or modifies files
help Lists available commands and usage information
lint (l) Runs linting tools on Angular code
new (n) Creates a new workspace and application
run Runs an Architect target with specified options
serve (s) Builds and serves your application
test (t) Runs unit tests in a workspace
update Updates your application to the latest version
version (v) Outputs Angular CLI version information
```

Angular CLI Command Syntax

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

ng new hello-world --prefix hw

Angular CLI

Global Angular CLI

projects/apm

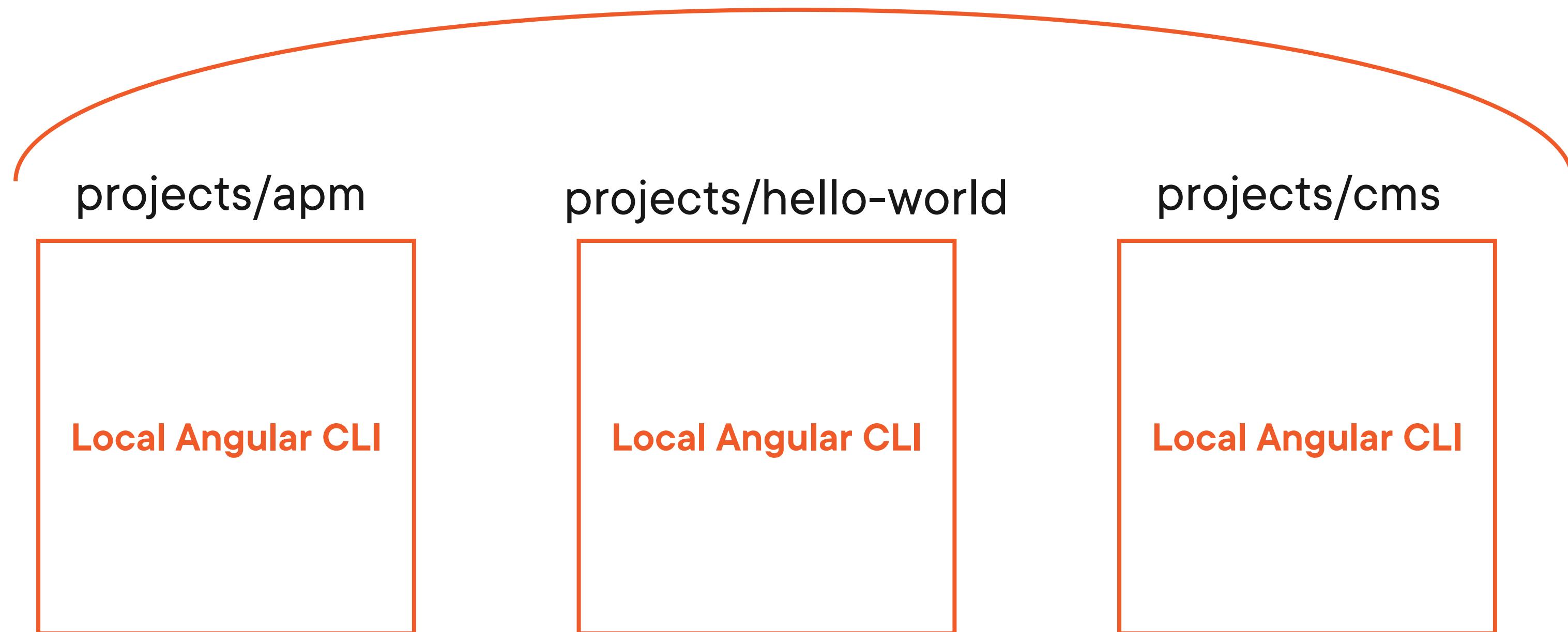
projects/hello-world

projects/cms

Local Angular CLI

Local Angular CLI

Local Angular CLI



Installing the Angular CLI



```
npm install -g @angular/cli
```

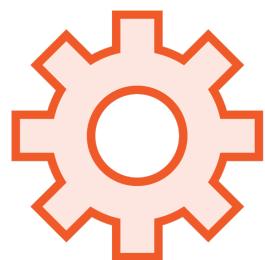
Demo



**Generate a new Angular application
(ng new)**

Examine the generated files

Serving the Application (`ng serve`)



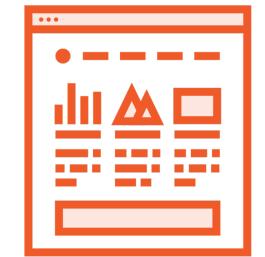
Compiles the application



Generates application bundles



Starts a local web server

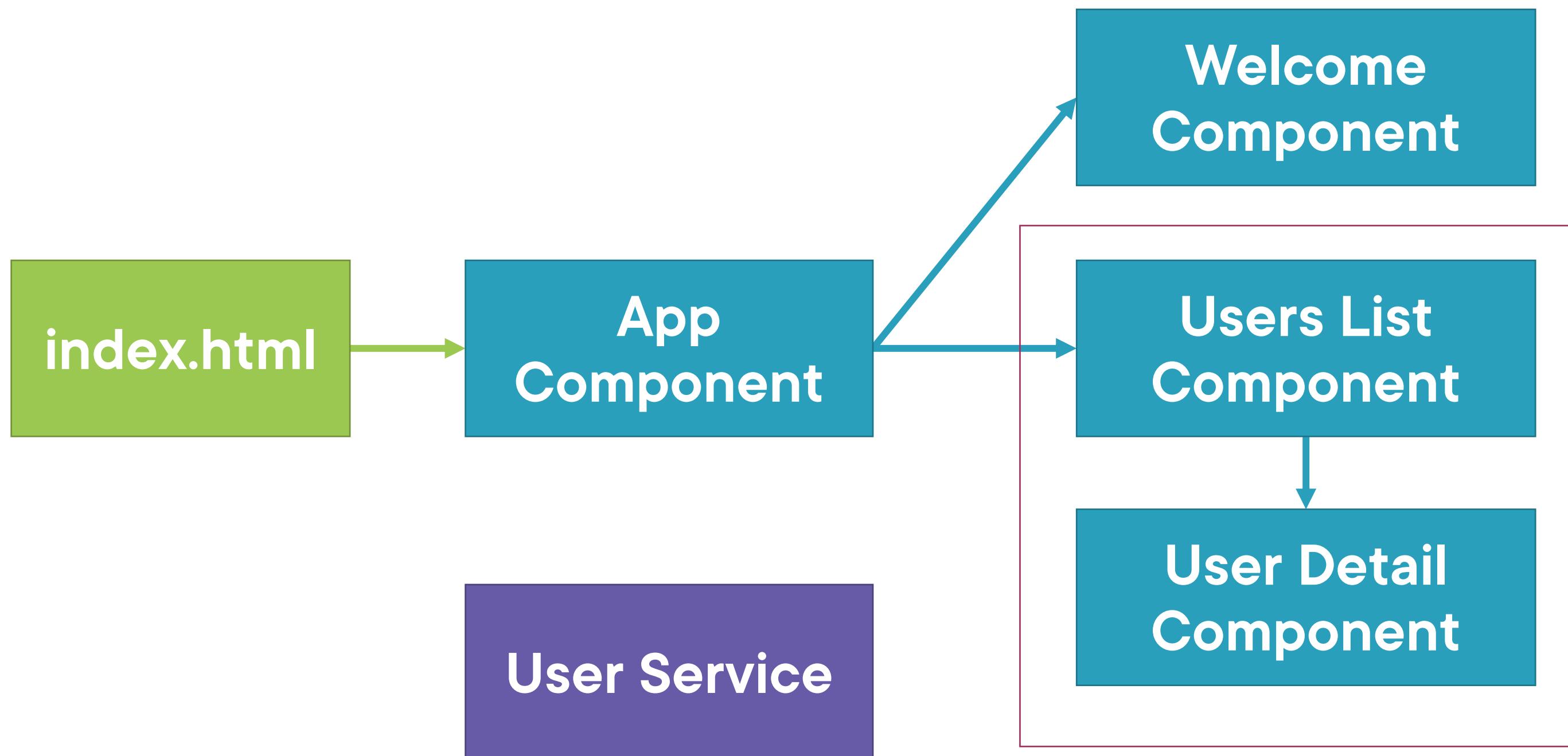


Serves the application from memory



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.



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>)

Demo



Testing the application (ng test)

Demo



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"
– John Papa



Coming up next ...

Final Words

Final Words



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata

Final Words



Recap of our journey



Tips for enhancing your development experience



Pointers to additional information

WHAT?

WHO?

WHERE?

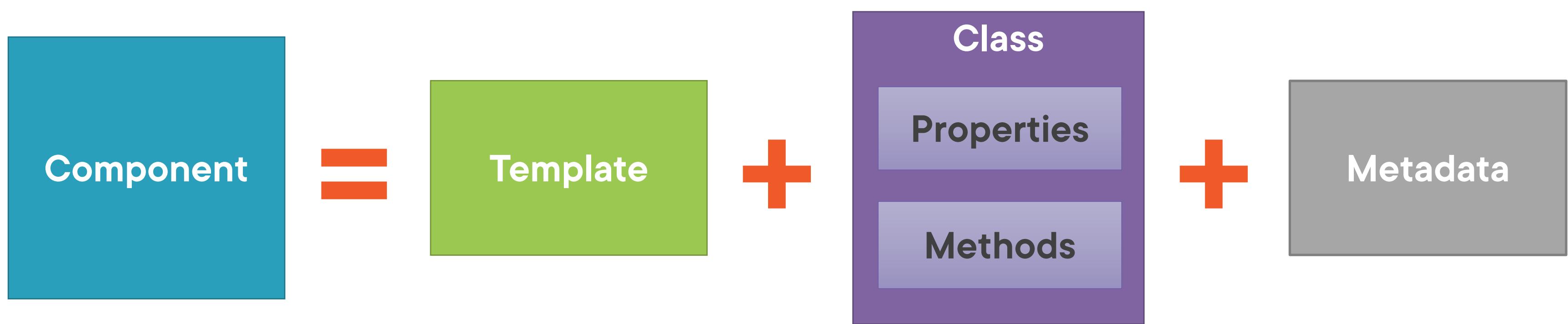
WHEN?

WHY?

HOW?



What Is a Component?





Where Do We Put the HTML?

Inline Template

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

Inline Template

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

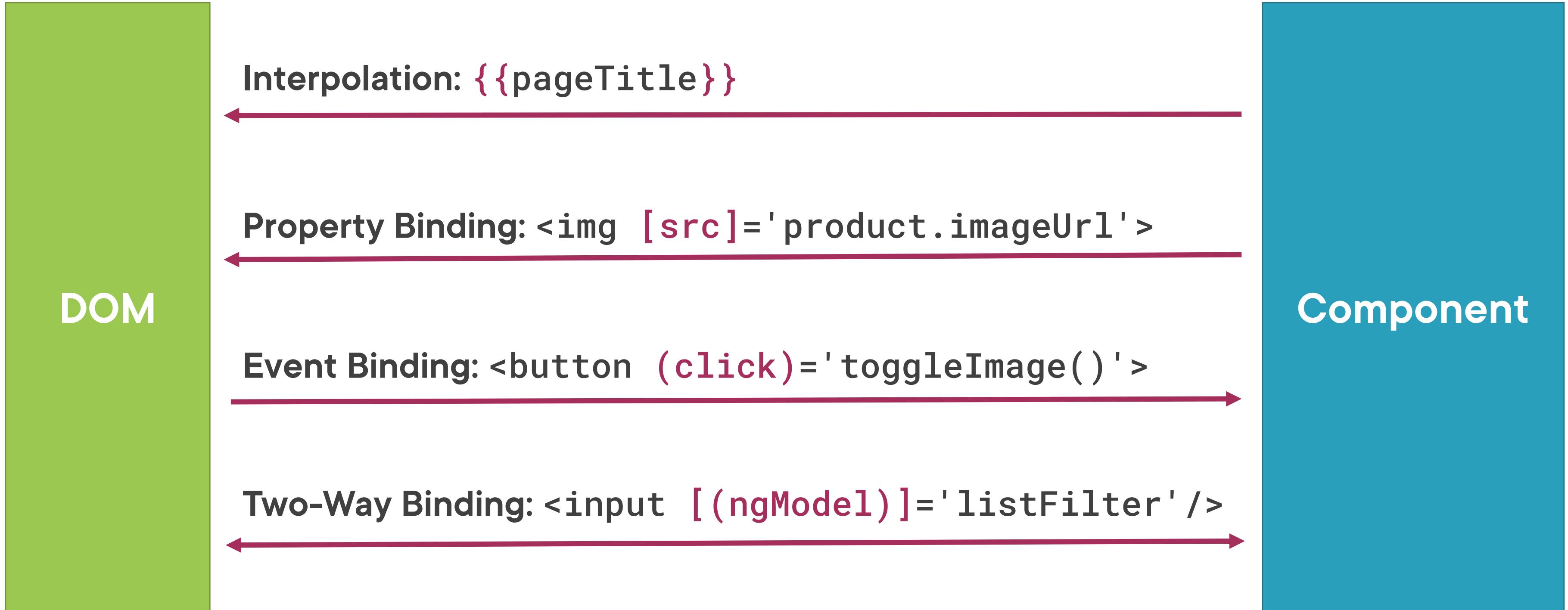
ES 2015
Back Ticks

Linked Template

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

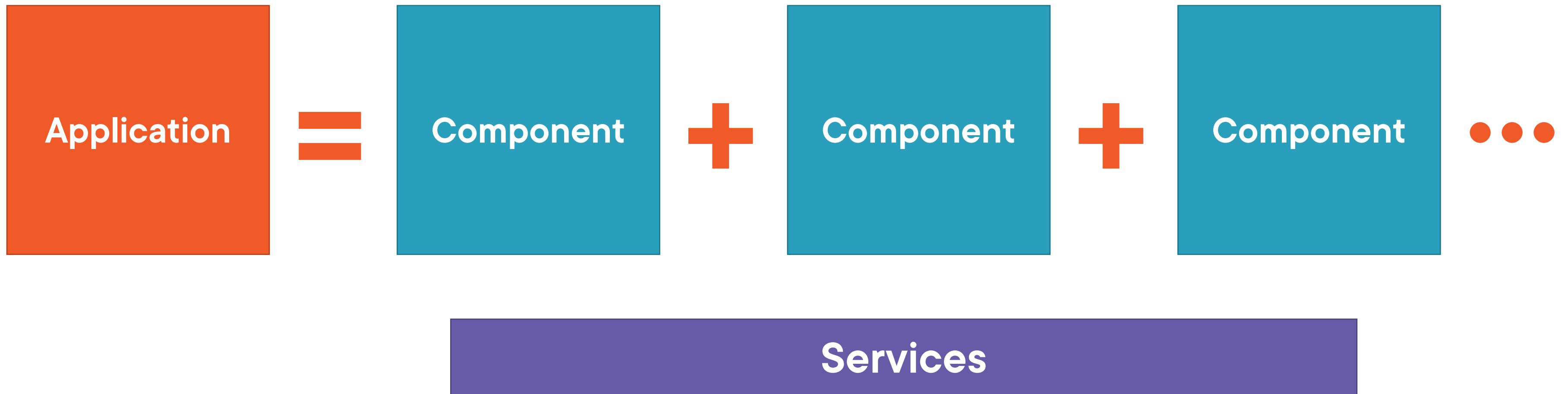


When Should We Use Data Binding?





Why Do We Need a Service?





How Do We Build?

product-list.component.ts

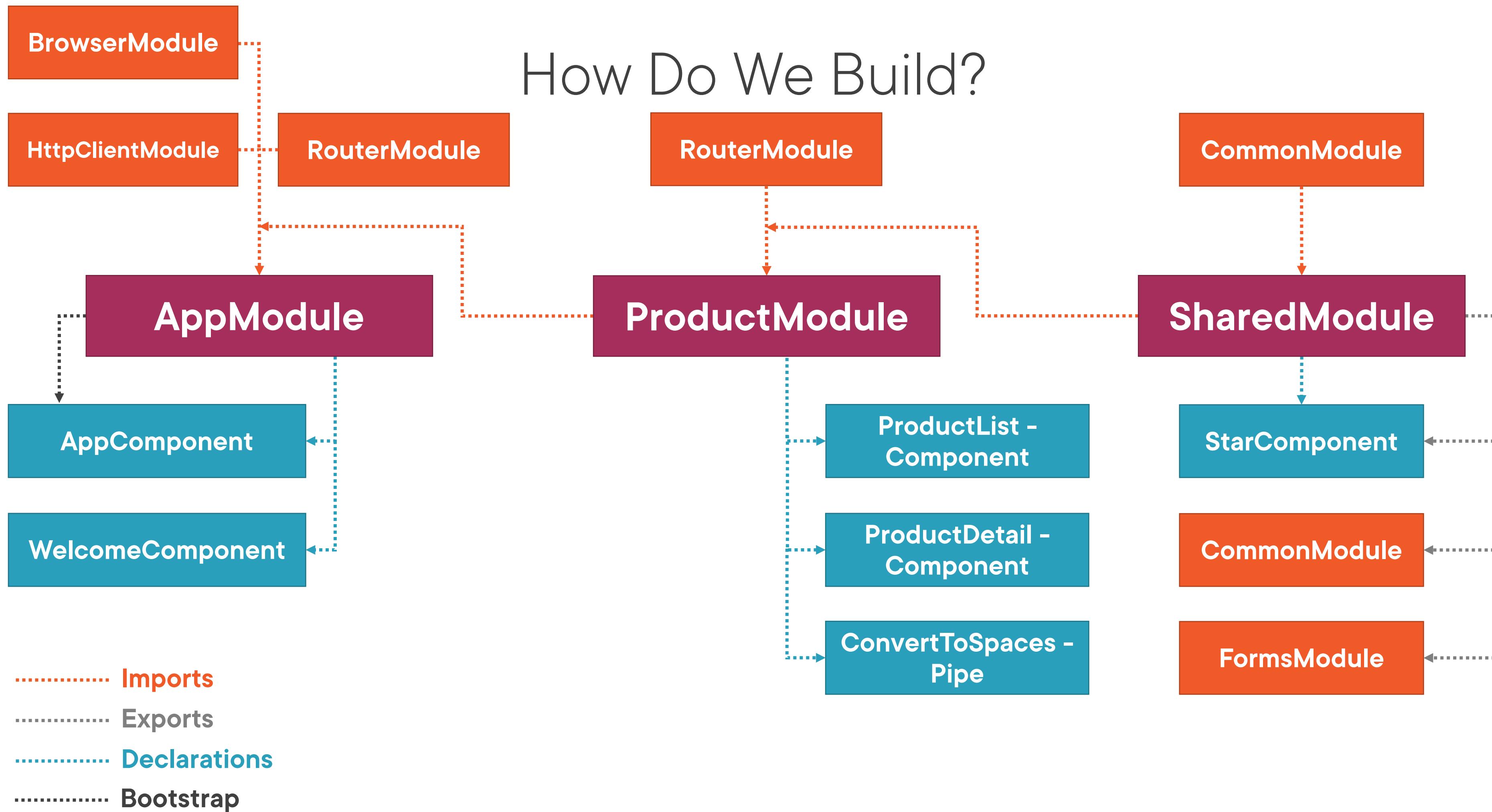
```
import { Component } from '@angular/core';

@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

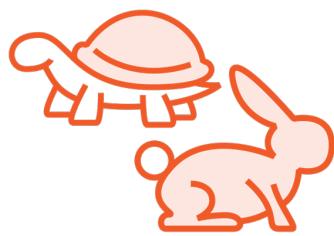
Revisit as you build

Download the slides from the Pluralsight page

Enhancing Your Development



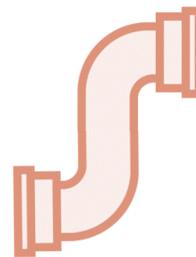
Install the Angular Language Service extension for VS Code



Implement lazy loading for improved load performance



Use Angular forms for building and validating user-entry forms

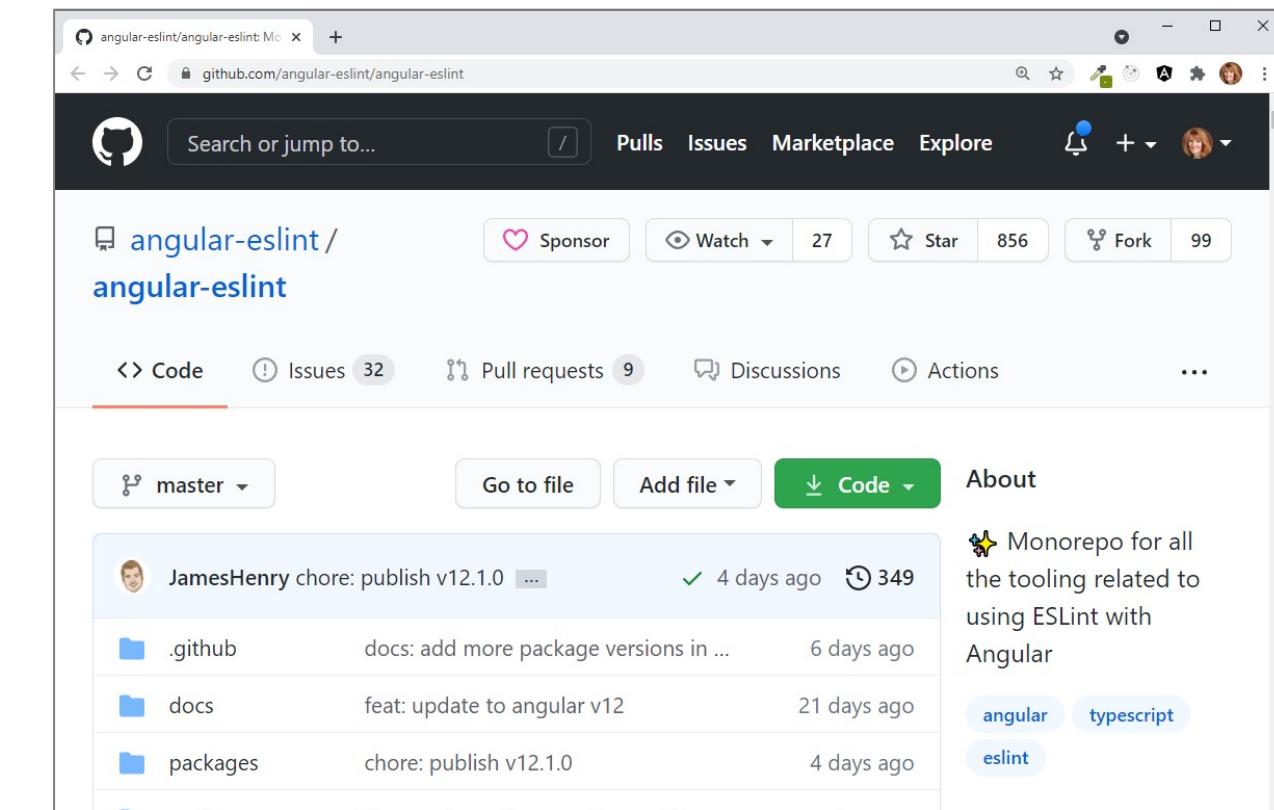
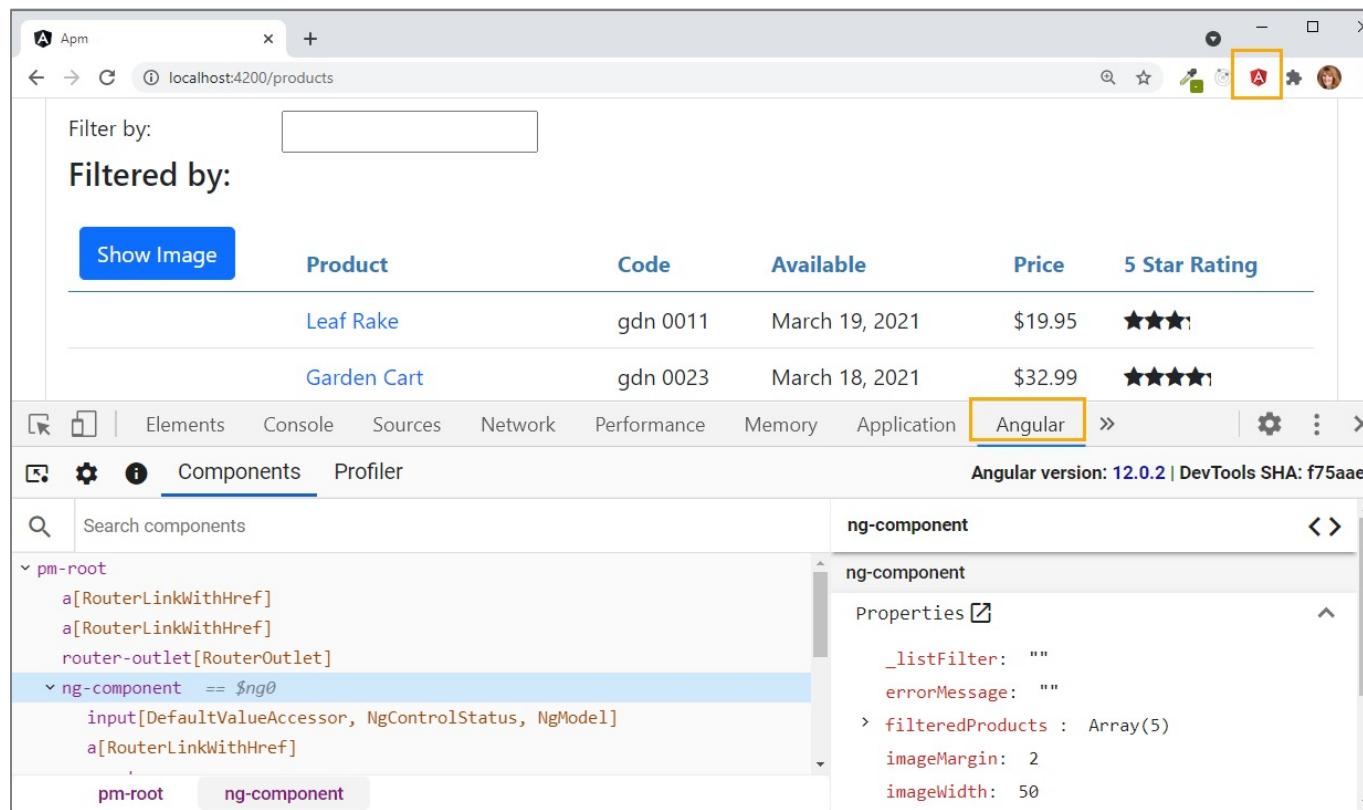


Leverage your Observable pipelines to process multiple datasets



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](#)

A photograph of a hiker from behind, standing on a rocky outcrop. The hiker is wearing a blue t-shirt, dark shorts, a large blue backpack, and yellow running shoes. Their arms are raised in a triumphant pose. The setting is a mountainous landscape at sunset, with the sky transitioning from orange to dark blue. The foreground is dominated by large, textured rocks.

@deborahkurata