More on Components

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

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;
   }
```

Interface

A specification identifying a related set of properties and methods.

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

Use the interface as a data type.

Development time only!

Interface Is a Specification

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

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

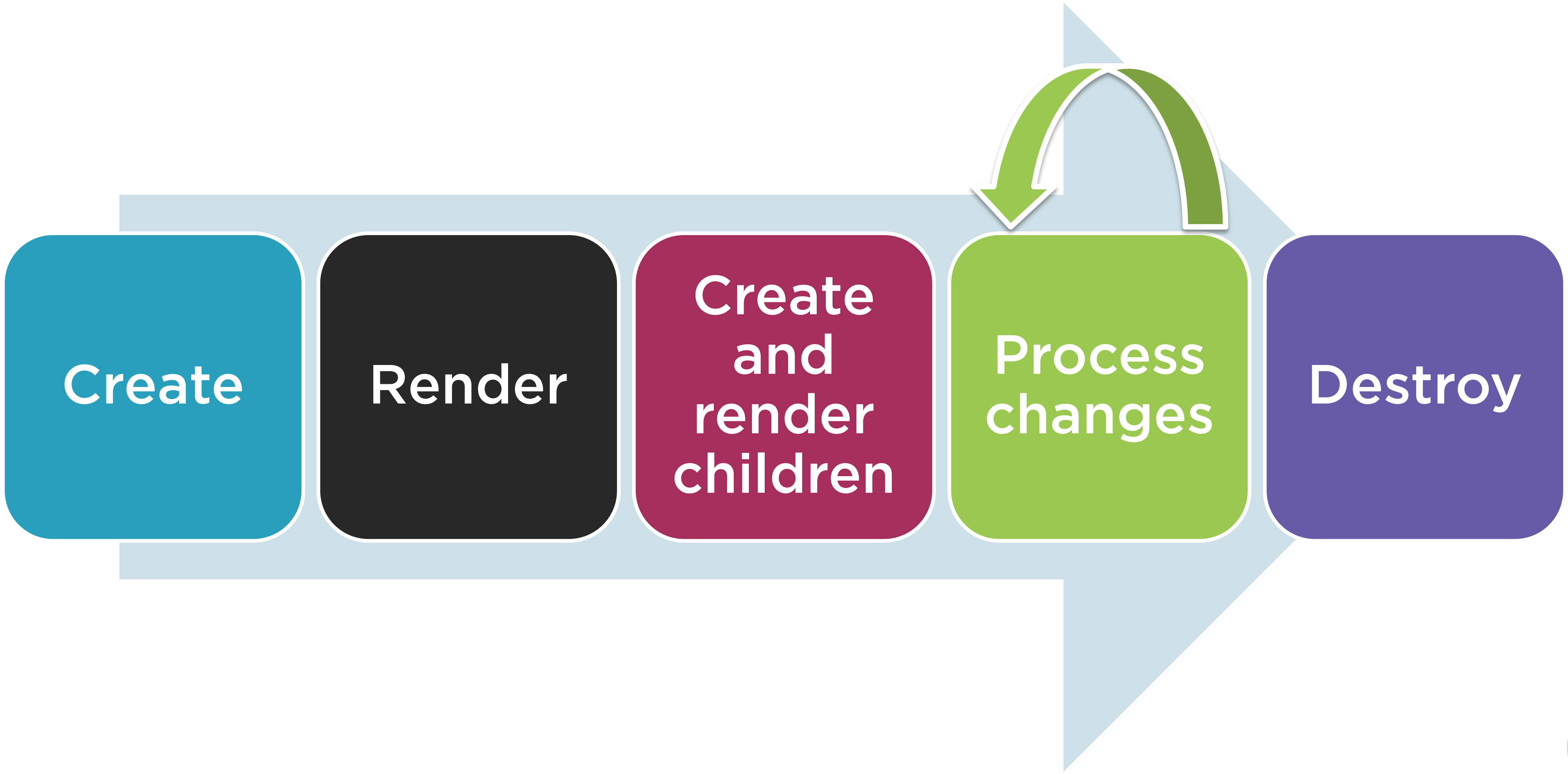
styles

```
@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



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

  export class ProductListComponent
1                         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, slice
- 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

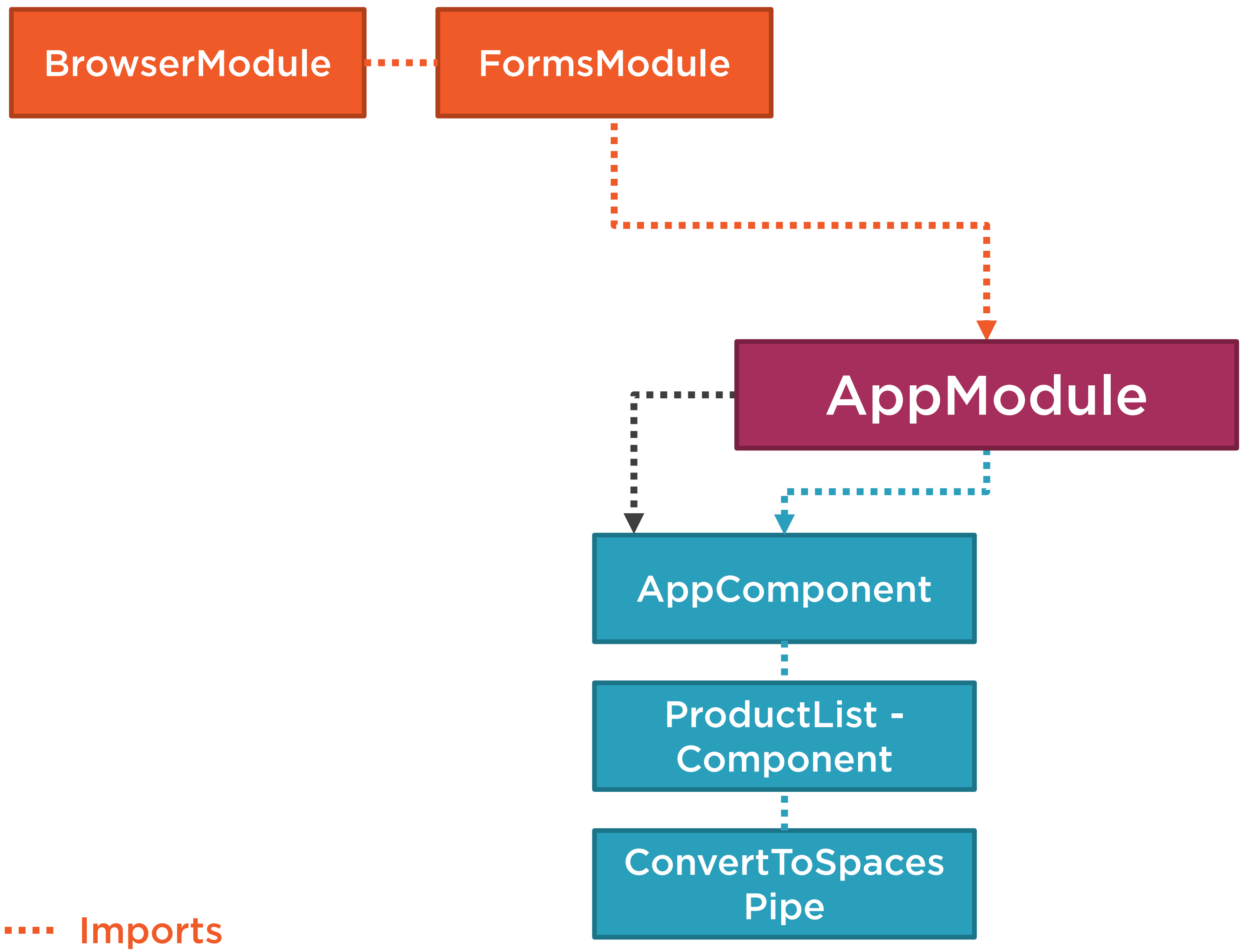
Template

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

Pipe

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

  }
```



⋯⋯⋯	Imports
⋯⋯⋯	Exports
⋯⋯⋯	Declarations
⋯⋯⋯	Providers
⋯⋯⋯	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 { }
```

Filtering Data

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

"Angular doesn't offer such pipes because they perform poorly and prevent aggressive minification."

angular.io

"The Angular team and many experienced Angular developers strongly recommend moving filtering and sorting logic into the component itself."

angular.io

Checklist: Interfaces

Defines custom types

Creating interfaces:

- `interface` keyword
- export it

Implementing interfaces:

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

Checklist: Encapsulating Styles

`styles` **property**

- Specify an array of style strings

`styleUrls` **property**

- Specify an array of stylesheet paths

Checklist: Using Lifecycle Hooks

Import the lifecycle hook interface

Implement the lifecycle hook interface

Write code for the hook method

Checklist: Building a Custom Pipe

- Import Pipe and PipeTransform
- Create a class that implements PipeTransform
  - `export` the class
- Write code for the Transform method
- Decorate the class with the Pipe decorator

Checklist: Using a Custom Pipe

Import the custom pipe

Add the pipe to the declarations array of an Angular module

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

Use the Pipe in the template

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

Summary

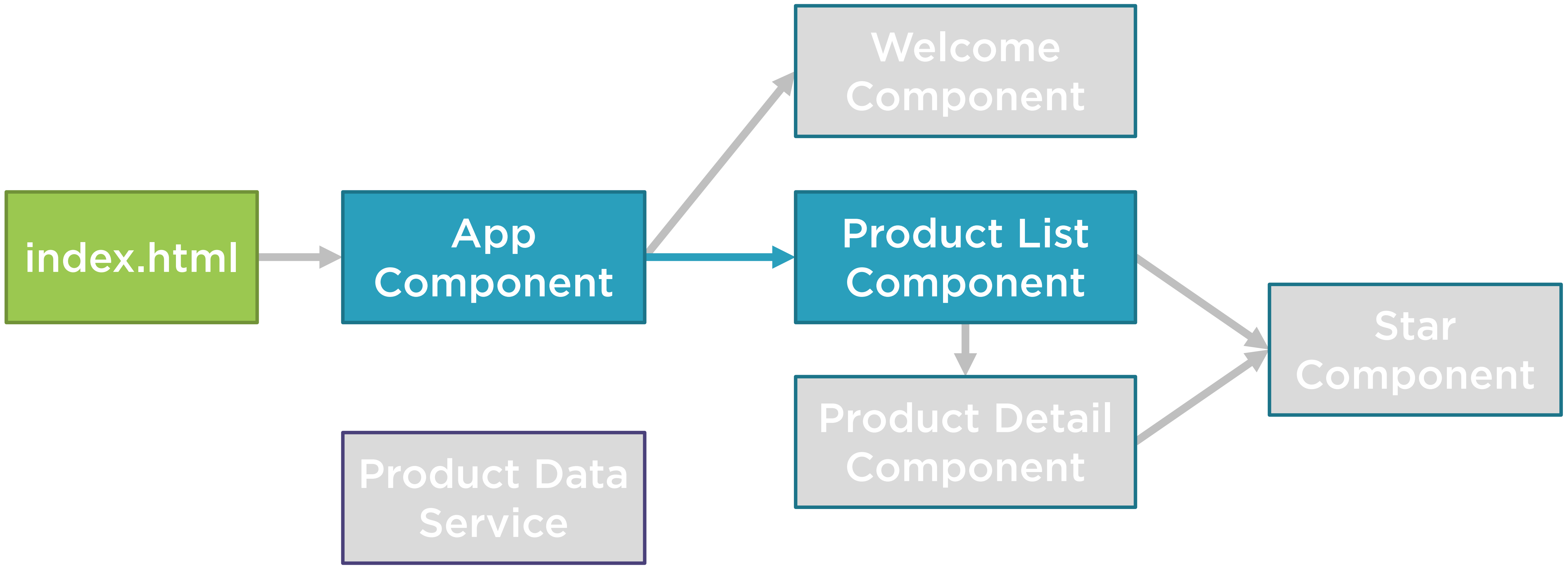
Defining an Interface

Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe

Application Architecture


index.html
App Component
Welcome Component
Product List Component
Product Detail Component
Star Component
Product Data Service