Getting Started with Angular: Your First App

Welcome to Angular!

This tutorial introduces you to the essentials of Angular by walking you through building a simple e-commerce site with a catalog, shopping cart, and check-out form. It uses the StackBlitz online development environment so you can get started right away.

> This guide uses the StackBlitz Generator to show you a ready-made, simple application that you can examine and play with interactively. In actual development you will typically use the Angular CLI, a powerful command-line tool that lets you generate and modify applications. For more information, see the CLI Overview.

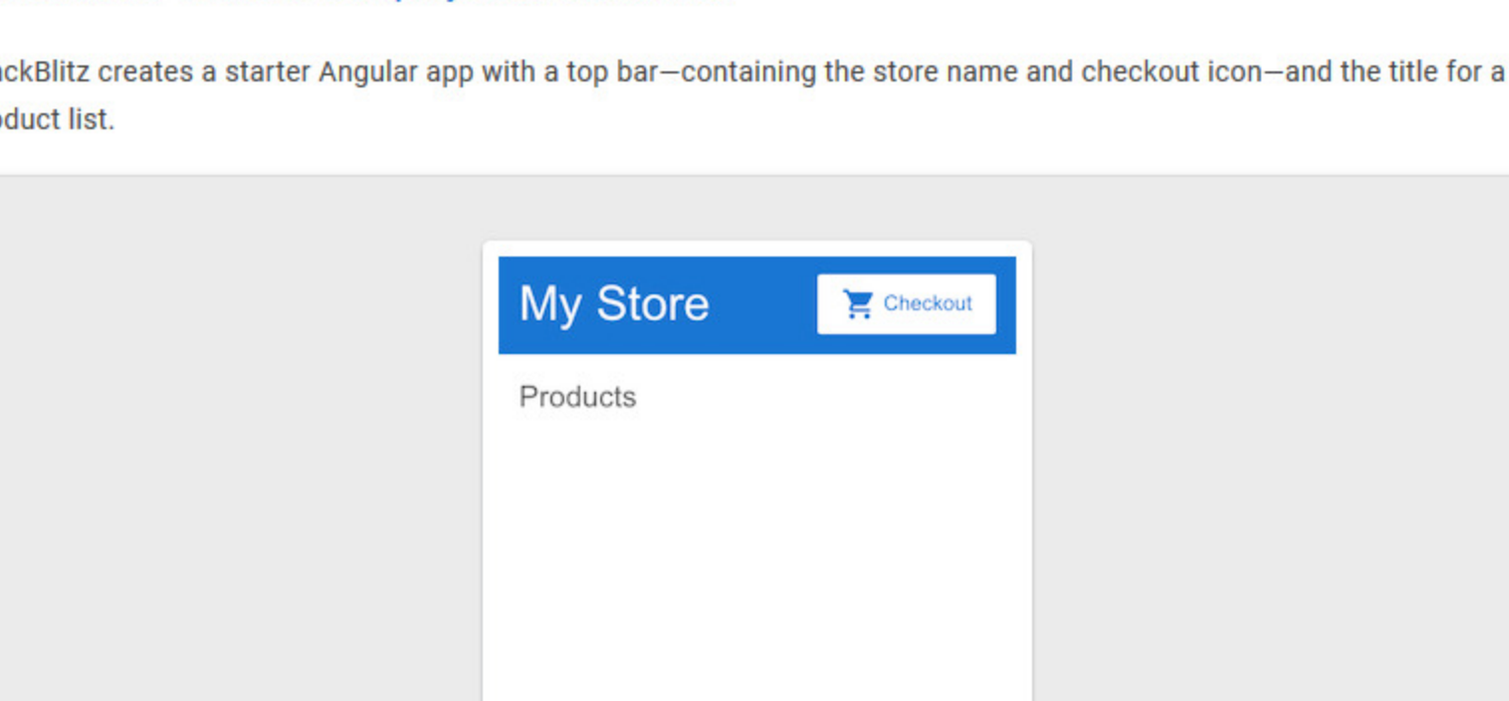
NEW TO WEB DEVELOPMENT?

There are many resources to complement the Angular docs. Mozilla's MDN docs include both HTML and JavaScript introductions. TypeScript's docs include a 5-minute tutorial. Various online course platforms, such as Udemy and Codecademy, also cover web development basics.

Create a new project

Click here to create a new project in StackBlitz.

StackBlitz creates a starter Angular app with a top bar—containing the store name and checkout icon—and the title for a product list.



STACKBLITZ TIPS

- Log into StackBlitz so you can save and resume your work. If you have a GitHub account, you can log into StackBlitz with that account. In order to save your progress, first fork the project using the Fork button at the top left, then you'll be able to save your work to your own StackBlitz account by clicking the Save button.
- To copy a code example from this tutorial, click the icon at the top right of the code example box, and then paste the code snippet from the clipboard into StackBlitz.
- If the StackBlitz preview pane isn't showing what you expect, save and then click the refresh button.
- StackBlitz is continually improving, so there may be slight differences in generated code, but the app's behavior will be the same.

Template syntax

Angular's template syntax extends HTML and JavaScript. This section introduces template syntax by enhancing the "Products" area.

> To help you get going, the following steps use predefined product data and methods from the `product-list.component.ts` file.

1. In the `product-list` folder, open the template file `product-list.component.html`.
2. Modify the product list template to display a list of product names.
   a. Each product in the list displays the same way, one after another on the page. To iterate over the predefined list of products, put the `*ngFor` directive on a `<div>`, as follows:

src/app/product-list/product-list.component.html

```html
<h2>Products</h2>

<div *ngFor="let product of products">
</div>
```

With `*ngFor`, the `<div>` repeats for each product in the list.

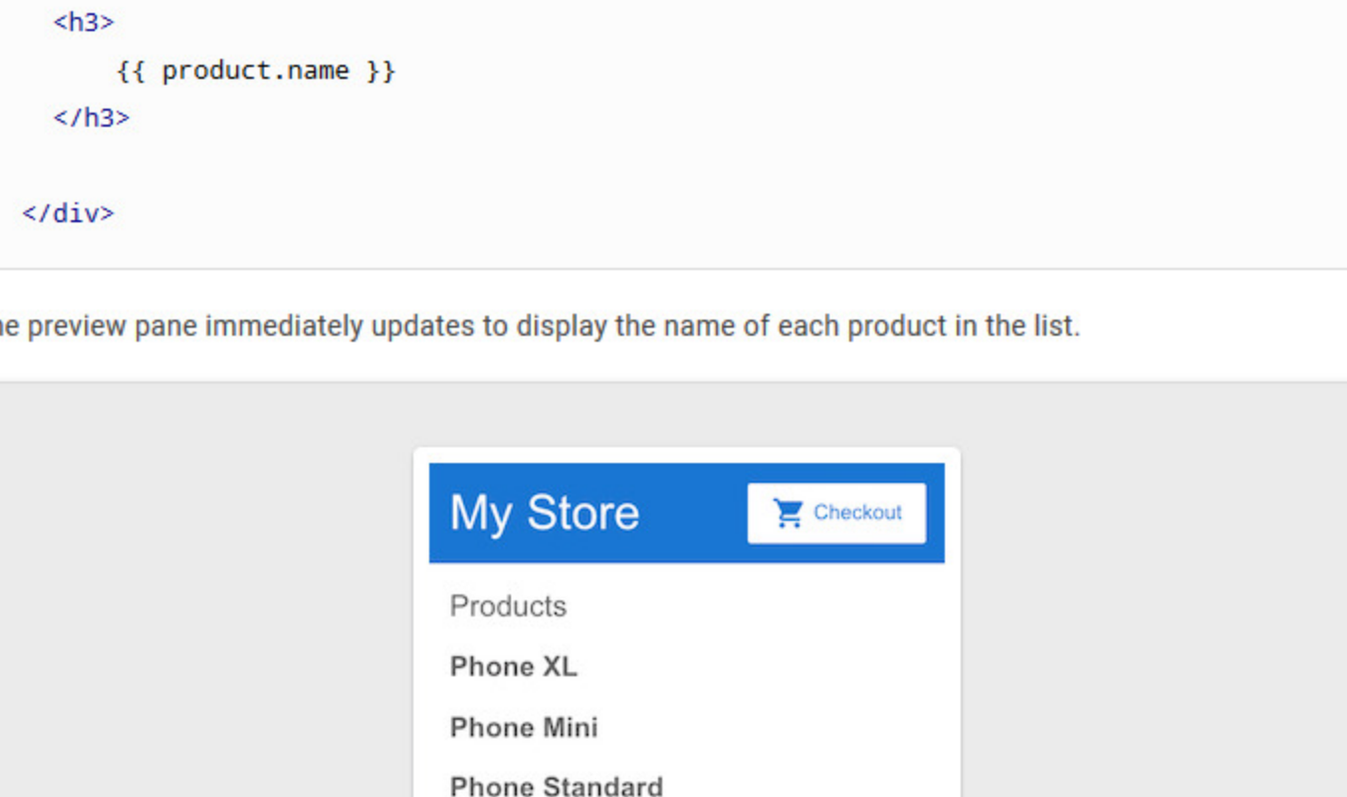
> `*ngFor` is a "structural directive". Structural directives shape or reshape the DOM's structure, typically by adding, removing, and manipulating the elements to which they are attached. Any directive with an asterisk, `*`, is a structural directive.

   b. To display the names of the products, use the interpolation syntax `{{ }}`. Interpolation renders a property's value as text. Inside the `<div>`, add an `<h3>` to display the interpolation of the product's name property:

src/app/product-list/product-list.component.html

```html
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
      {{ product.name }}
  </h3>

</div>
```

The preview pane immediately updates to display the name of each product in the list.

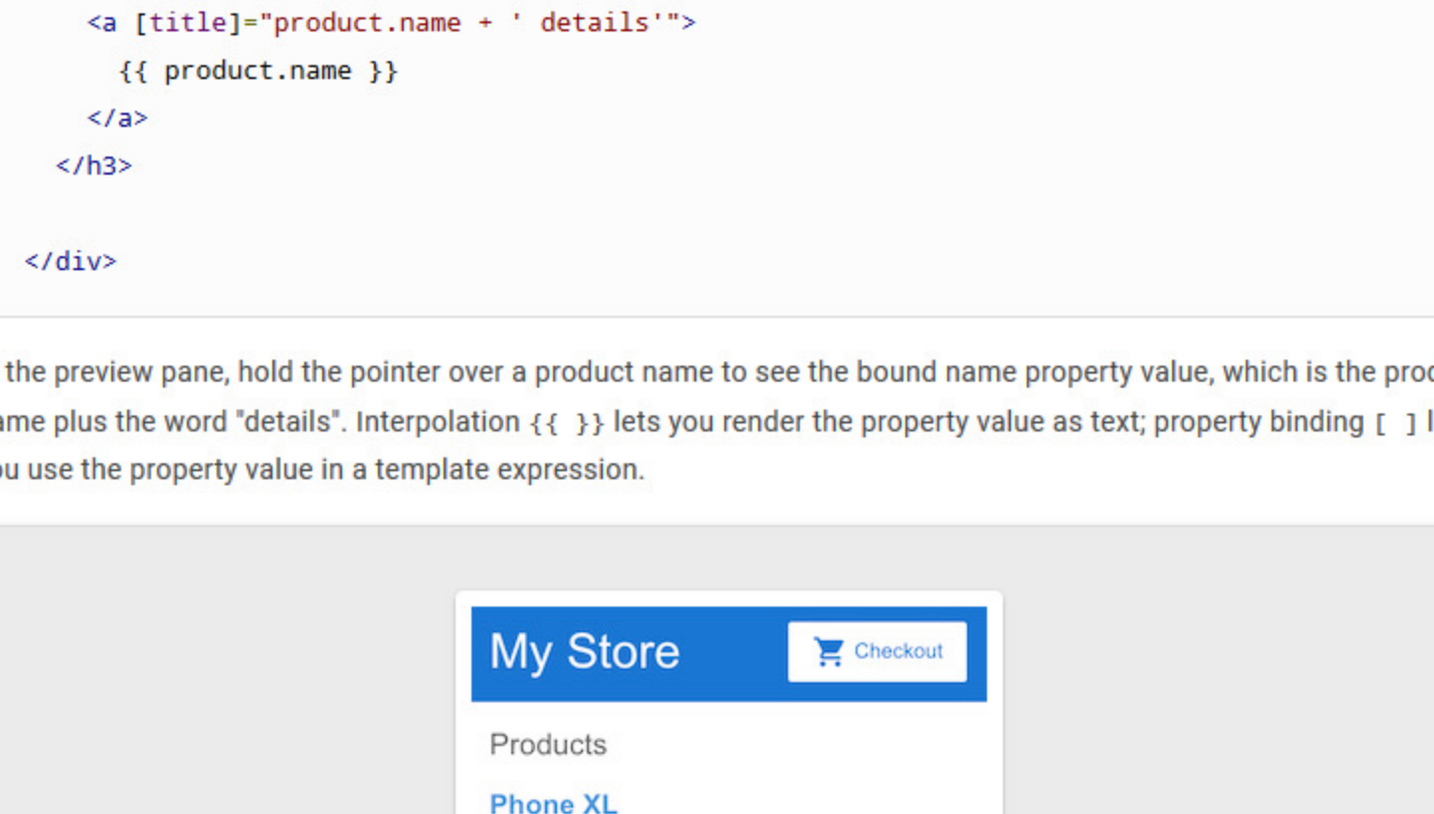


3. To make each product name a link to product details, add the `<a>` element and set its title to be the product's name by using the property binding `[ ]` syntax, as follows:

src/app/product-list/product-list.component.html

```html
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

</div>
```

In the preview pane, hold the pointer over a product name to see the bound name property value, which is the product name plus the word "details". Interpolation `{{ }}` lets you render the property value as text; property binding `[ ]` lets you use the property value in a template expression.

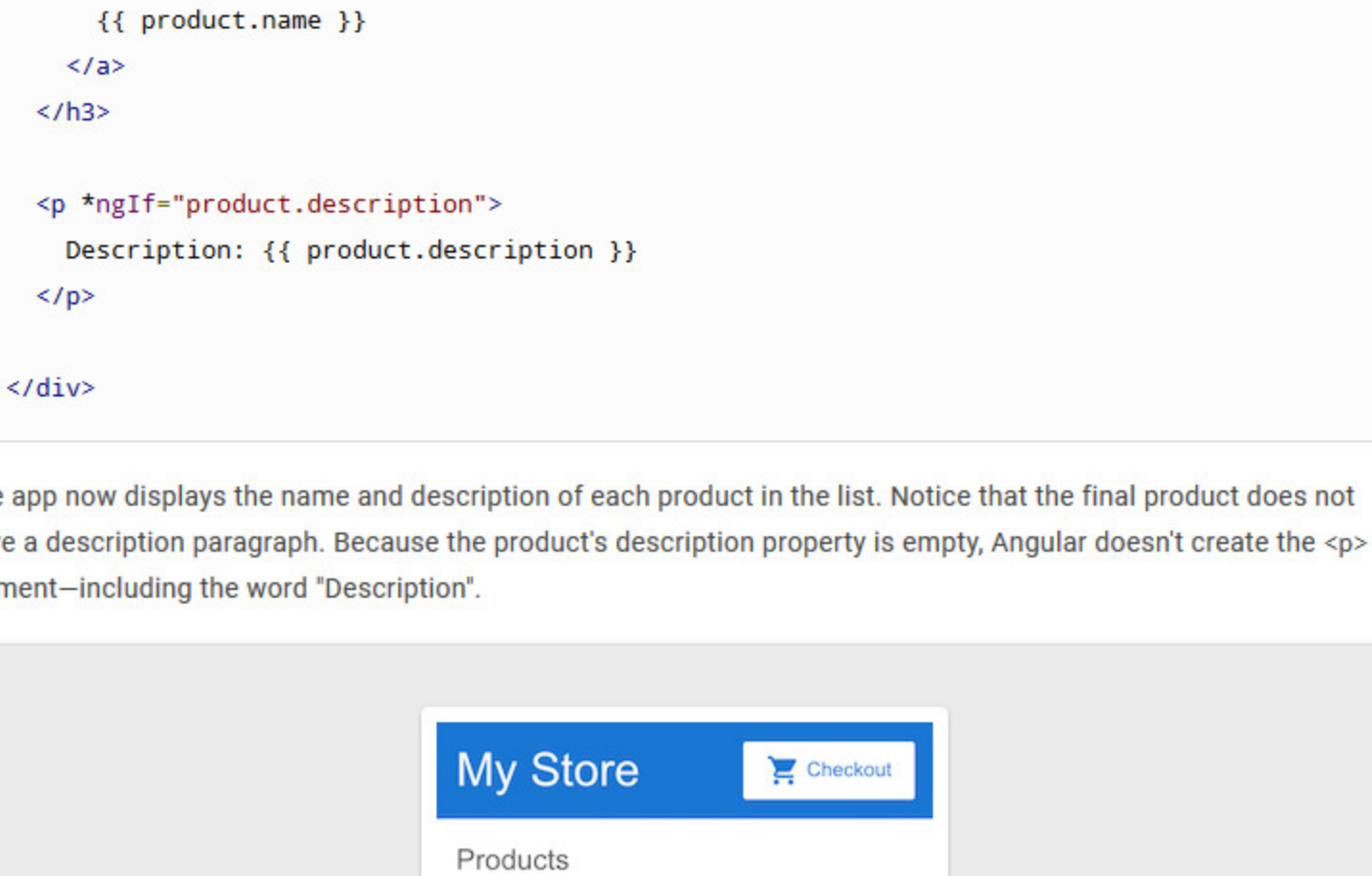


4. Add the product descriptions. On the `<p>` element, use an `*ngIf` directive so that Angular only creates the `<p>` element if the current product has a description.

src/app/product-list/product-list.component.html

```html
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>

</div>
```

The app now displays the name and description of each product in the list. Notice that the final product does not have a description paragraph. Because the product's description property is empty, Angular doesn't create the `<p>` element—including the word "Description".

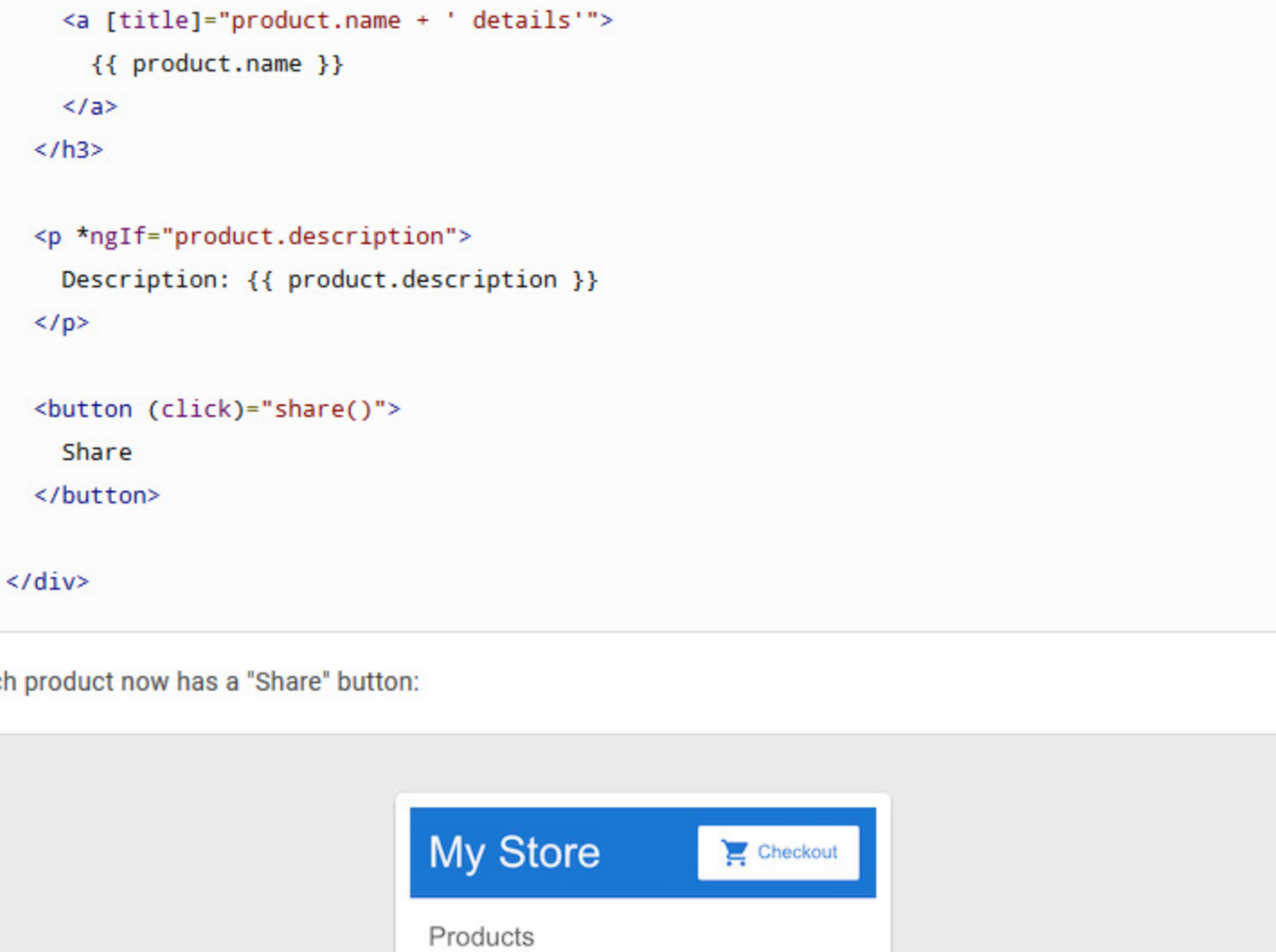


5. Add a button so users can share a product with friends. Bind the button's `click` event to the `share()` method (in `product-list.component.ts`). Event binding uses a set of parentheses, `( )`, around the event, as in the following `<button>` element:

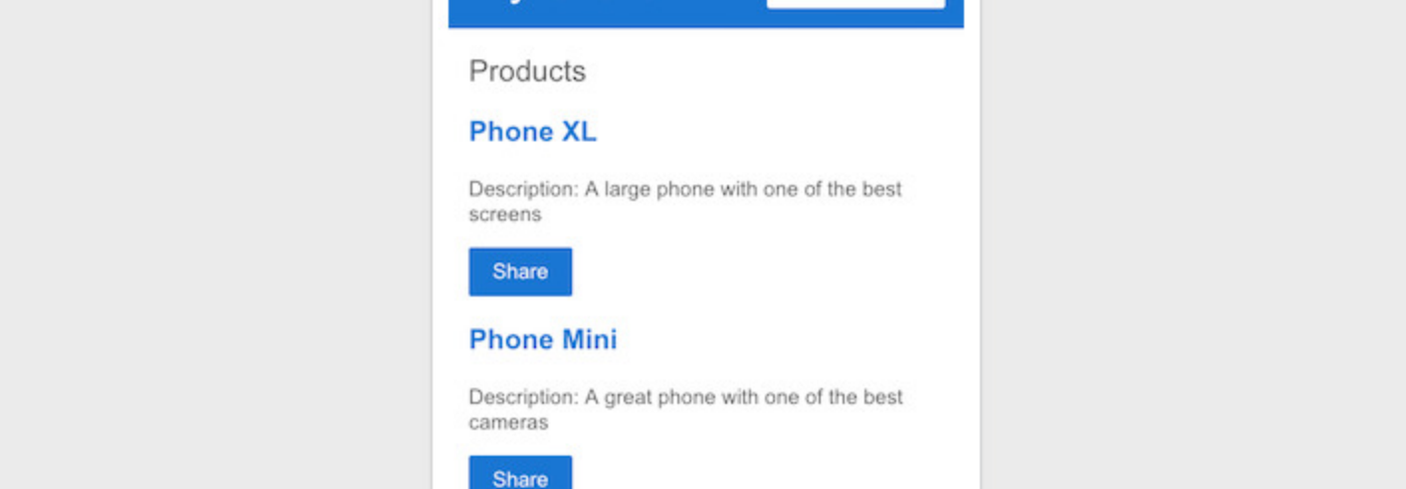
src/app/product-list/product-list.component.html

```html
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>

  <button (click)="share()">
    Share
  </button>

</div>
```

Each product now has a "Share" button:



Test the "Share" button:



The app now has a product list and sharing feature. In the process, you've learned to use five common features of Angular's template syntax:

- `*ngFor`
- `*ngIf`
- Interpolation `{{ }}`
- Property binding `[ ]`
- Event binding `( )`

> For more information about the full capabilities of Angular's template syntax, see Template Syntax.

Components

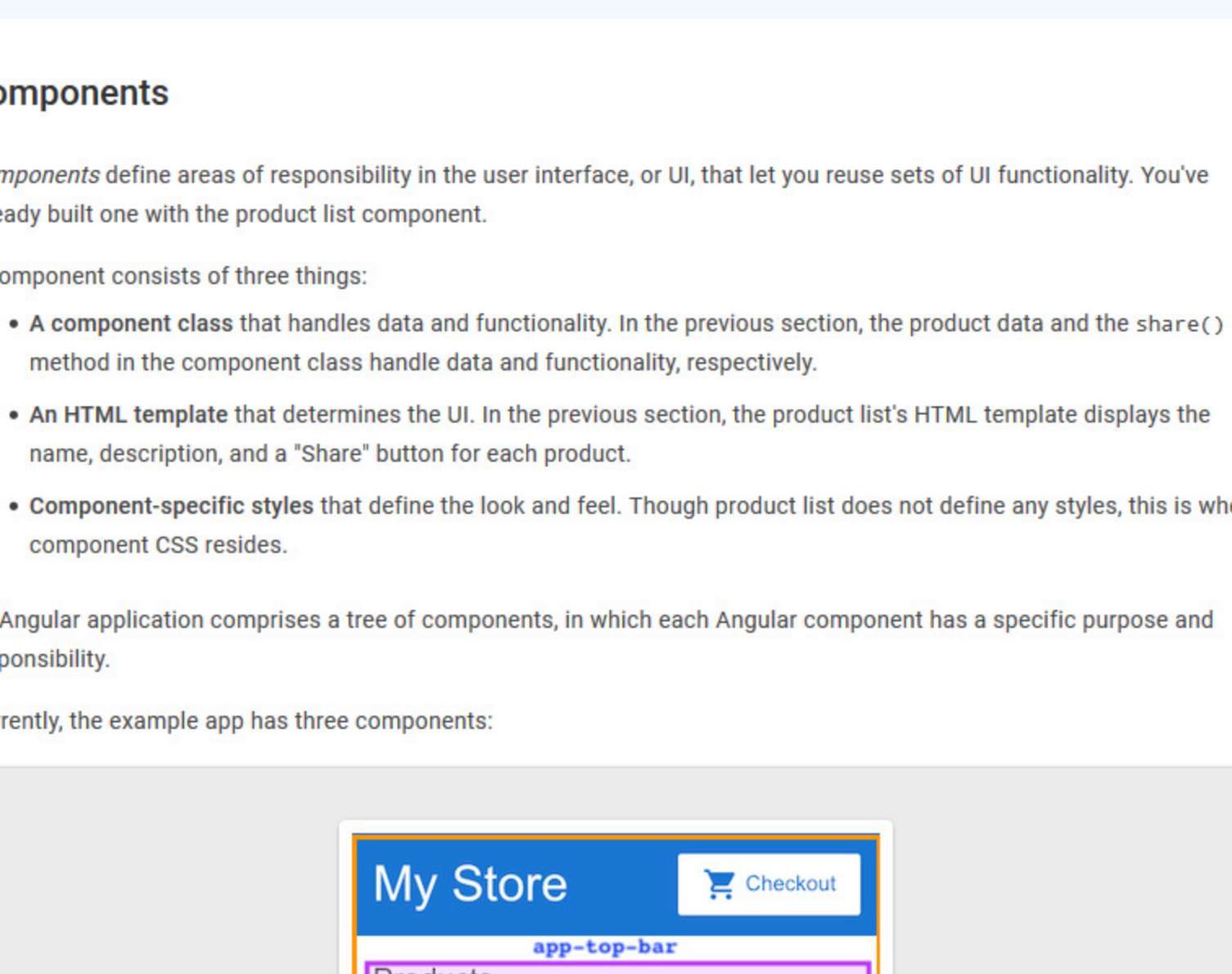
Components define areas of responsibility in the user interface, or UI, that let you reuse sets of UI functionality. You've already built one with the product list component.

A component consists of three things:

- **A component class** that handles data and functionality. In the previous section, the product data and the `share()` method in the component class handle data and functionality, respectively.
- **An HTML template** that determines the UI. In the previous section, the product list's HTML template displays the name, description, and a "Share" button for each product.
- **Component-specific styles** that define the look and feel. Though product list does not define any styles, this is where component CSS resides.

An Angular application comprises a tree of components, in which each Angular component has a specific purpose and responsibility.

Currently, the example app has three components:



- `app-root` (orange box) is the application shell. This is the first component to load and the parent of all other components. You can think of it as the base page.
- `app-top-bar` (blue background) is the store name and checkout button.
- `app-product-list` (purple box) is the product list that you modified in the previous section.

The next section expands the app's capabilities by adding a new component—a product alert—as a child of the product list component.

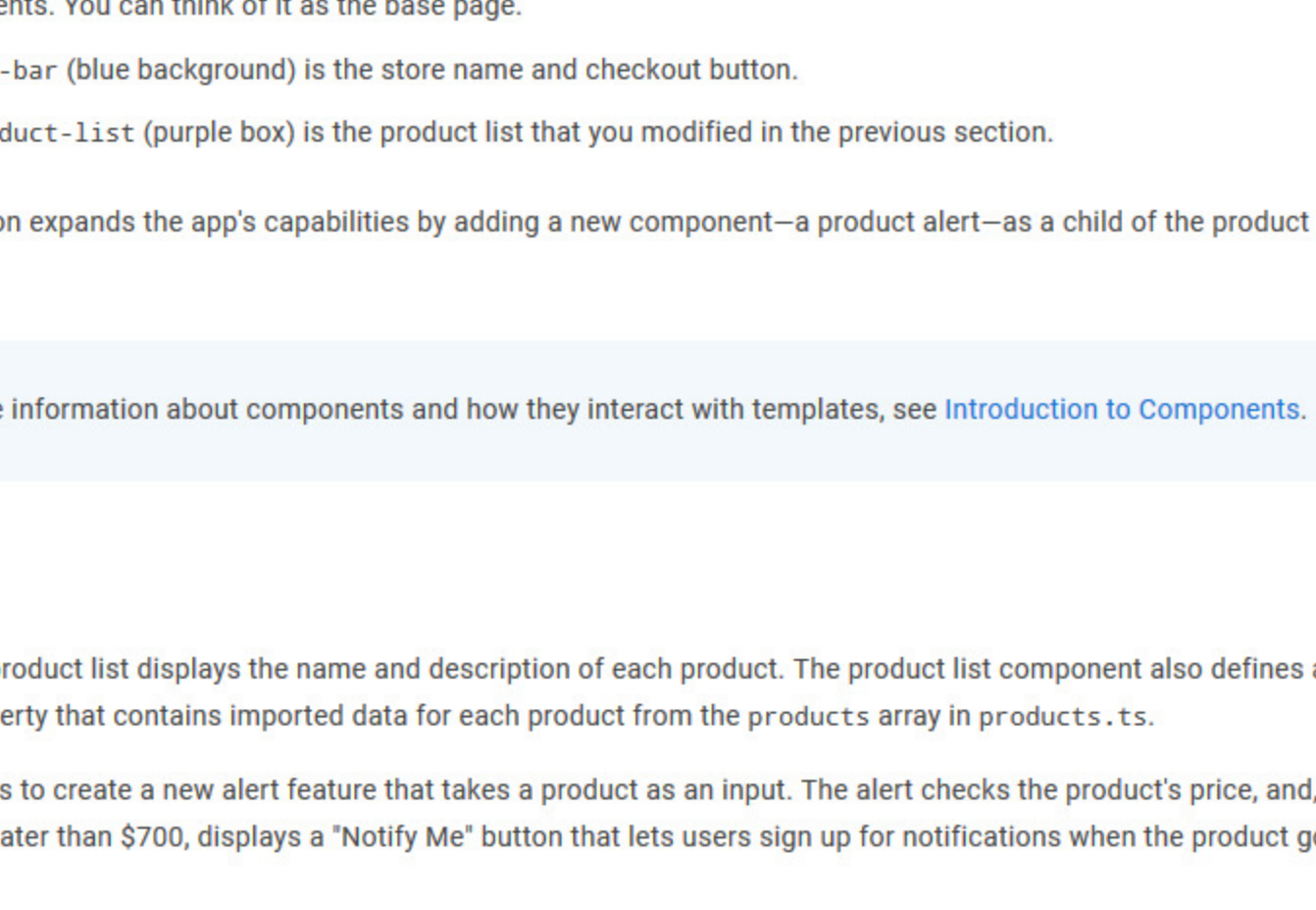
> For more information about components and how they interact with templates, see Introduction to Components.

Input

Currently, the product list displays the name and description of each product. The product list component also defines a `products` property that contains imported data for each product from the `products` array in `products.ts`.

The next step is to create a new alert feature that takes a product as an input. The alert checks the product's price, and, if the price is greater than $700, displays a "Notify Me" button that lets users sign up for notifications when the product goes on sale.

1. Create a new product alerts component.
   a. Right click on the `app` folder and use the `Angular Generator` to generate a new component named `product-alerts`.



The generator creates starter files for all three parts of the component:

- `product-alerts.component.ts`
- `product-alerts.component.html`
- `product-alerts.component.css`

2. Open `product-alerts.component.ts`.

src/app/product-alerts/product-alerts.component.ts

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

@Component({
  selector: 'app-product-alerts',
  templateUrl: './product-alerts.component.html',
  styleUrls: ['./product-alerts.component.css']
})
export class ProductAlertsComponent implements OnInit {
  constructor() { }

  ngOnInit() {
  }

}
```

   a. Notice the `@Component()` decorator. This indicates that the following class is a component. It provides metadata about the component, including its selector, templates, and styles.
      - The `selector` identifies the component. The selector is the name you give the Angular component when it is rendered as an HTML element on the page. By convention, Angular component selectors begin with the prefix `app-`, followed by the component name.
      - The template and style filenames reference the HTML and CSS files that StackBlitz generates.

   b. The component definition also exports the class, `ProductAlertsComponent`, which handles functionality for the component.

3. Set up the new product alerts component to receive a product as input:
   a. Import `Input` from `@angular/core`.

src/app/product-list/product-alerts.component.ts

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

   b. In the `ProductAlertsComponent` class definition, define a property named `product` with an `@Input()` decorator. The `@Input()` decorator indicates that the property value passes in from the component's parent, the product list component.

src/app/product-list/product-alerts.component.ts

```typescript
export class ProductAlertsComponent implements OnInit
{
  @Input() product;
  constructor() { }

  ngOnInit() {
  }

}
```

4. Define the view for the new product alert component.
   a. Open the `product-alerts.component.html` template and replace the placeholder paragraph with a "Notify Me" button that appears if the product price is over $700.

src/app/product-alerts/product-alerts.component.html

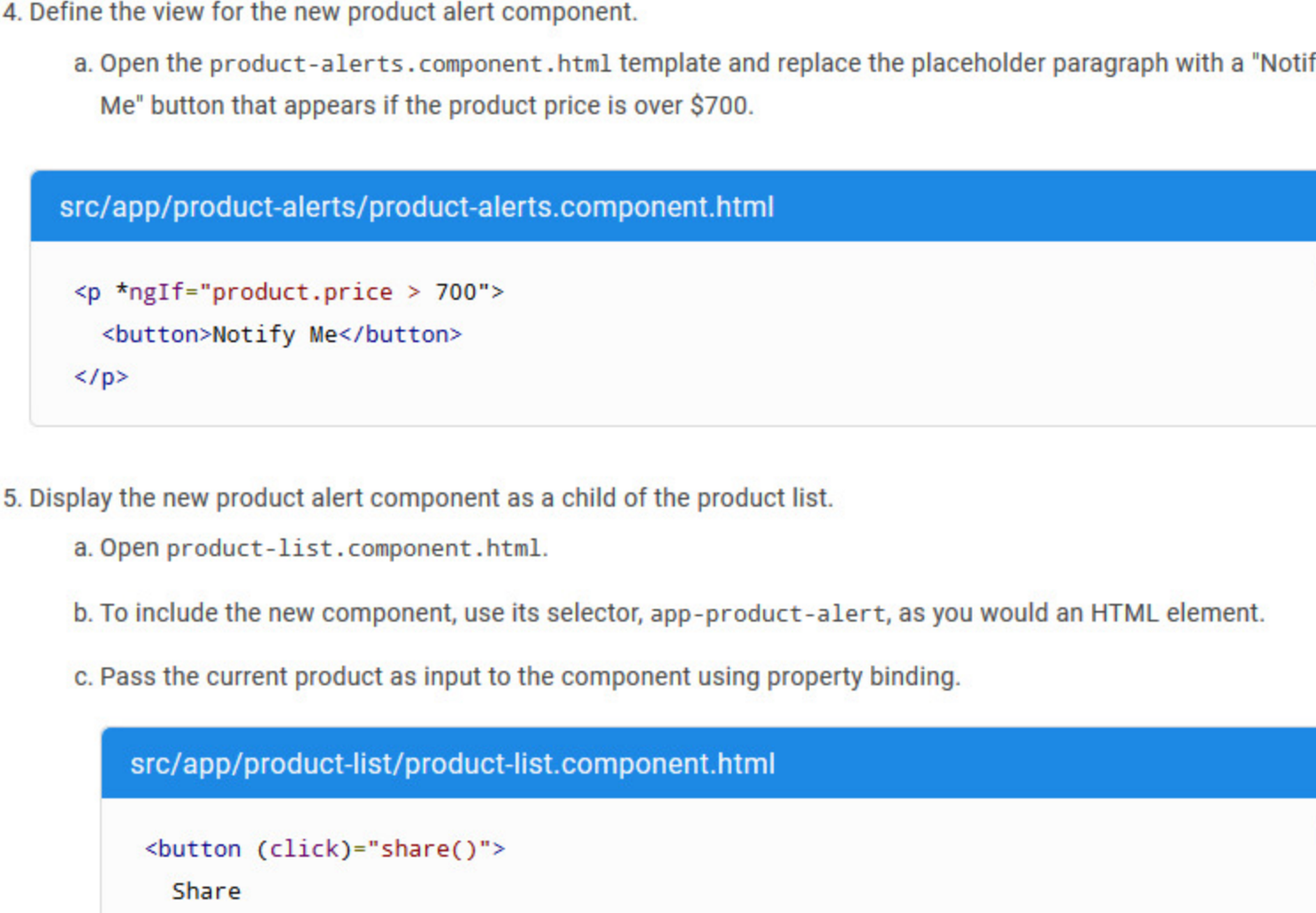
```html
<p *ngIf="product.price > 700">
  <button>Notify Me</button>
</p>
```

5. Display the new product alert component as a child of the product list.
   a. Open `product-list.component.html`.
   b. To include the new component, use its selector, `app-product-alert`, as you would an HTML element.
   c. Pass the current product as input to the component using property binding.

src/app/product-list/product-list.component.html

```html
<button (click)="share()">
  Share
</button>

<app-product-alerts
  [product]="product">
</app-product-alerts>
```

The new product alert component takes a product as input from the product list. With that input, it shows or hides the "Notify Me" button, based on the price of the product. The Phone XL price is over $700, so the "Notify Me" button appears on that product.



> See Component Interaction for more information about passing data from a parent to child component, intercepting and acting upon a value from the parent, and detecting and acting on changes to input property values.

Output

To make the "Notify Me" button work, you need to configure two things:

- the product alert component to emit an event when the user clicks "Notify Me"
- the product list component to act on that event

1. Open `product-alerts.component.ts`.
2. Import `Output` and `EventEmitter` from `@angular/core`:

src/app/product-alerts/product-alerts.component.ts

```typescript
import { Component } from '@angular/core';
import { Input } from '@angular/core';
import { Output, EventEmitter } from '@angular/core';
```

3. In the component class, define a property named `notify` with an `@Output()` decorator and an instance of `EventEmitter()`. This allows the product alert component to emit an event when the value of the notify property changes.

src/app/product-alerts/product-alerts.component.ts

```typescript
export class ProductAlertsComponent {
  @Input() product;
  @Output() notify = new EventEmitter();
}
```

4. In the product alert template, `product-alerts.component.html`, update the "Notify Me" button with an event binding to call the `notify.emit()` method.

src/app/product-alerts/product-alerts.component.html

```html
<p *ngIf="product.price > 700">
  <button (click)="notify.emit()">Notify Me</button>
</p>
```

5. Next, define the behavior that should happen when the user clicks the button. Recall that it's the parent, product list component—not the product alerts component—that acts when the child raises the event. In `product-list.component.ts`, define an `onNotify()` method, similar to the `share()` method:

src/app/product-list/product-list.component.ts

```typescript
export class ProductListComponent {
  products = products;

  share() {
    window.alert('The product has been shared!');
  }

  onNotify() {
    window.alert('You will be notified when the product goes on sale');
  }
}
```

6. Finally, update the product list component to receive output from the product alerts component. In `product-list.component.html`, bind the `app-product-alerts` component (which is what displays the "Notify Me" button) to the `onNotify()` method of the product list component.

src/app/product-list/product-list.component.html

```html
<button (click)="share()">
  Share
</button>

<app-product-alerts
  [product]="product"
  (notify)="onNotify()">
</app-product-alerts>
```

7. Try the "Notify Me" button:

> See Component Interaction for more information about listening for events from child components, reading child properties or invoking child methods, and using a service for bi-directional communication between components.

Next steps

Congratulations! You've completed your first Angular app!

You have a basic online store catalog with a product list, "Share" button, and "Notify Me" button. You've learned about the foundation of Angular: components and template syntax. You've also learned how the component class and template interact, and how components communicate with each other.

To continue exploring Angular, choose either of the following options:

- Continue to the "Routing" section to create a product details page that can be accessed by clicking a product name and that has its own URL pattern.
- Skip ahead to the "Deployment" section to move to local development, or deploy your app to Firebase or your own server.