

Getting started with Angular

Welcome to Angular!

This tutorial introduces you to the essentials of Angular by walking you through building an e-commerce site with a catalog, shopping cart, and check-out form.

To help you get started right away, this tutorial uses a ready-made application that you can examine and modify interactively on StackBlitz—without having to set up a local work environment. StackBlitz is a browser-based development environment where you can create, save, and share projects using a variety of technologies.

Prerequisites

To get the most out of this tutorial you should already have a basic understanding of the following.

- HTML
- JavaScript
- TypeScript

{@a components} ## Take a tour of the example application

You build Angular applications with components. Components define areas of responsibility in the UI that let you reuse sets of UI functionality.

A component consists of three things:

- **A component class** that handles data and functionality.
- **An HTML template** that determines the UI.
- **Component-specific styles** that define the look and feel.

This guide demonstrates building an application with the following components.

- **<app-root>**—the first component to load and the container for the other components.
- **<app-top-bar>**—the store name and checkout button.
- **<app-product-list>**—the product list.
- **<app-product-alerts>**—a component that contains the application’s alerts.

For more information about components, see Introduction to Components.

{@a new-project}

Create the sample project

To create the sample project, generate the ready-made sample project in StackBlitz. To save your work:

1. Log into StackBlitz.

2. Fork the project you generated.
3. Save periodically.

In StackBlitz, the preview pane on the right shows the starting state of the example application. The preview features two areas:

- a top bar with the store name, *My Store*, and a checkout button
- a header for a product list, *Products*

The project section on the left shows the source files that make up the application, including the infrastructure and configuration files.

When you generate the StackBlitz example applications that accompany the tutorials, StackBlitz creates the starter files and mock data for you. The files you use throughout the tutorial are in the `src` folder.

For more information on how to use StackBlitz, see the StackBlitz documentation.

```
{@a product-list} ## Create the product list
```

In this section, you'll update the application to display a list of products. You'll use predefined product data from the `products.ts` file and methods from the `product-list.component.ts` file. This section guides you through editing the HTML, also known as the template.

1. In the `product-list` folder, open the template file `product-list.component.html`.
2. Add an `*ngFor` structural directive on a `<div>`, as follows.

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

Structural directives shape or reshape the DOM's structure, by adding, removing, and manipulating elements. For more information about structural directives, see Structural directives.

1. Inside the `<div>`, add an `<h3>` and `{{ product.name }}`. The `{{ product.name }}` statement is an example of Angular's interpolation syntax. Interpolation `{{ }}` lets you render the property value as text.

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

```

```

1. To make each product name a link to product details, add the `<a>` element around `{{ product.name }}`.
2. Set the title to be the product's name by using the property binding `[]` syntax, as follows:

In the preview pane, hover over a product name to see the bound name property value, which is the product name plus the word “details”. Property binding `[]` lets you use the property value in a template expression.

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

The application now displays the name and description of each product in the list. Notice that the final product does not have a description paragraph. Angular doesn't create the `<p>` element because the product's description property is empty.

4. Add a button so users can share a product. 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 `(click)` event on the `<button>` element.

Each product now has a **Share** button.

Clicking the **Share** button triggers an alert that states, "The product has been shared!"

In editing the template, you have explored some of the most popular features of Angular templates. For more information, see Introduction to components and templates.

```
{@a passing-data-in}
```

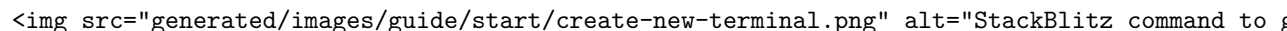
Pass data to a child component

Currently, the product list displays the name and description of each product. The `ProductListComponent` 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 uses product data from the `ProductListComponent`. 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.

This section walks you through creating a child component, `ProductAlertsComponent` that can receive data from its parent component, `ProductListComponent`.

1. Click on the plus sign above the current terminal to create a new terminal to run the command to generate the component.

The image shows a terminal window with the command `ng generate component product-alerts` entered. The terminal has a dark background with light-colored text.

1. In the new terminal, generate a new component named `product-alerts` by running the following command.

```
ng generate component product-alerts
```

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

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

* `product-alerts.component.css``

1. Open `product-alerts.component.ts`. The `@Component()` decorator indicates that the following class is a component. `@Component()` also provides metadata about the component, including its selector, templates, and styles.

Key features in the `@Component()` are as follows:

* The ``selector``, ``app-product-alerts``, identifies the component.

By convention, Angular component selectors begin with the prefix ``app-``, followed by the c

* The template and style filenames reference the component's HTML and CSS.

* The ``@Component()`` definition also exports the class, ``ProductAlertsComponent``, which han

1. To set up `ProductAlertsComponent` to receive product data, first import `Input` from `@angular/core`.
1. 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, `ProductListComponent`.
1. Open `product-alerts.component.html` and replace the placeholder paragraph with a **Notify Me** button that appears if the product price is over \$700.
1. The generator automatically added the `ProductAlertsComponent` to the `AppModule` to make it available to other components in the application.
1. Finally, to display `ProductAlertsComponent` as a child of `ProductListComponent`, add the `<app-product-alerts>` element to `product-list.component.html`. Pass the current product as input to the component using property binding.

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.

{@a output}

Pass data to a parent component

To make the **Notify Me** button work, the child component needs to notify and pass the data to the parent component. The `ProductAlertsComponent` needs to emit an event when the user clicks **Notify Me** and the `ProductListComponent` needs to respond to the event.

In new components, the Angular Generator includes an empty `constructor()`, the `OnInit` interface, and the `ngOnInit()` method. Since these steps don't use them, the following code examples omit them for brevity.

1. In `product-alerts.component.ts`, import `Output` and `EventEmitter` from `@angular/core`.
1. In the component class, define a property named `notify` with an `@Output()` decorator and an instance of `EventEmitter()`. Configuring `ProductAlertsComponent` with an `@Output()` allows the `ProductAlertsComponent` to emit an event when the value of the `notify` property changes.
1. In `product-alerts.component.html`, update the **Notify Me** button with an event binding to call the `notify.emit()` method.
2. Define the behavior that happens when the user clicks the button. The parent, `ProductListComponent`—not the `ProductAlertsComponent`—acts when the child raises the event. In `product-list.component.ts`, define an `onNotify()` method, similar to the `share()` method.
1. Update the `ProductListComponent` to receive data from the `ProductAlertsComponent`.

In `product-list.component.html`, bind `<app-product-alerts>` to the `onNotify()` method of the product list component. `<app-product-alerts>` is what displays the **Notify Me** button.

`<code-example header="src/app/product-list/product-list.component.html" path="getting-started">`

1. Click the **Notify Me** button to trigger an alert which reads, “You will be notified when the product goes on sale”.

``

For more information on communication between components, see [Component Interaction](#).

{@a whats-next}

What’s next

In this section, you’ve created an application that iterates through data and features components that communicate with each other.

To continue exploring Angular and developing this application:

- Continue to [In-app navigation](#) to create a product details page.
- Skip ahead to [Deployment](#) to move to local development, or deploy your application to [Firebase](#) or your own server.

@reviewed 2021-09-15