# [Getting](https://docs.angularjs.org/guide) Started

## Getting Started with Angular: Managing Data

At the end of [Routing](https://angular.io/start/start-routing), the online store application has a product catalog with two views: a product list and product details. Users can click on a product name from the list to see details in a new view, with a distinct URL, or route.

This page guides you through creating the shopping cart in three phases:

* Update the product details page to include a "Buy" button, which adds the current product to a list of products that a cart service manages.
* Add a cart component, which displays the items in the cart.
* Add a shipping component, which retrieves shipping prices for the items in the cart by using Angular's [HttpClient](https://angular.io/api/common/http/HttpClient) to retrieve shipping data from a .json file.

## Services

Services are an integral part of Angular applications. In Angular, a service is an instance of a class that you can make available to any part of your application using Angular's [dependency injection system](https://angular.io/guide/glossary#dependency-injection-di).

Services are the place where you share data between parts of your application. For the online store, the cart service is where you store your cart data and methods.

## Create the shopping cart service

Up to this point, users can view product information, and simulate sharing and being notified about product changes. They cannot, however, buy products.

In this section, you add a "Buy" button to the product details page and set up a cart service to store information about products in the cart.

Later, the [Forms](https://angular.io/start/start-forms) part of this tutorial guides you through accessing this cart service from the page where the user checks out.

### Define a cart service

1. Generate a cart service.
   1. Right click on the app folder, choose Angular Generator, and choose Service. Name the new service cart.

src/app/cart.service.ts

content\_copyimport { [Injectable](https://angular.io/api/core/Injectable) } from '@angular/core';

@[Injectable](https://angular.io/api/core/Injectable)({

[providedIn](https://angular.io/api/core/Injectable#providedIn): 'root'

})

export class CartService {

constructor() {}

}

* 1. StackBlitz might generate the @[Injectable](https://angular.io/api/core/Injectable)() decorator without the { [providedIn](https://angular.io/api/core/Injectable#providedIn): 'root' } statement as above. Instead, the generator provides the cart service in app.module.ts by default. For the purposes of this tutorial, either way works. The @[Injectable](https://angular.io/api/core/Injectable)() { [providedIn](https://angular.io/api/core/Injectable#providedIn): 'root' } syntax allows [tree shaking](https://angular.io/guide/dependency-injection-providers#tree-shakable-providers), which is beyond the scope of this guide.

1. In the CartService class, define an items property to store the array of the current products in the cart.

src/app/cart.service.ts

content\_copyexport class CartService {

items = [];

}

1. Define methods to add items to the cart, return cart items, and clear the cart items:

src/app/cart.service.ts

content\_copyexport class CartService {

items = [];

addToCart(product) {

this.items.push(product);

}

getItems() {

return this.items;

}

clearCart() {

this.items = [];

return this.items;

}

}

* 1. The addToCart() method appends a product to an array of items.
  2. The getItems() method collects the items users add to the cart and returns each item with its associated quantity.
  3. The clearCart() method returns an empty array of items.

### Use the cart service

This section walks you through using the cart service to add a product to the cart with a "Buy" button.

1. Open product-details.component.ts.
2. Configure the component to use the cart service.
   1. Import the cart service.

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

content\_copyimport { [Component](https://angular.io/api/core/Component), [OnInit](https://angular.io/api/core/OnInit) } from '@angular/core';

import { [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) } from '@angular/router';

import { products } from '../products';

import { CartService } from '../cart.service';

* 1. Inject the cart service by adding it to the constructor().

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

content\_copyexport class ProductDetailsComponent implements [OnInit](https://angular.io/api/core/OnInit) {

constructor(

private route: [ActivatedRoute](https://angular.io/api/router/ActivatedRoute),

private cartService: CartService

) { }

}

1. Define the addToCart() method, which adds the current product to the cart.

The addToCart() method does the following three things:

* 1. Receives the current product.
  2. Uses the cart service's addToCart() method to add the product the cart.
  3. Displays a message that you've added a product to the cart.

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

content\_copyexport class ProductDetailsComponent implements [OnInit](https://angular.io/api/core/OnInit) {

addToCart(product) {

this.cartService.addToCart(product);

window.alert('Your product has been added to the cart!');

}

}

1. Update the product details template with a "Buy" button that adds the current product to the cart.
   1. Open product-details.component.html.
   2. Add a button with the label "Buy", and bind the click() event to the addToCart() method:

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

content\_copy<h2>Product Details</h2>

<div \*[ngIf](https://angular.io/api/common/NgIf)="product">

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

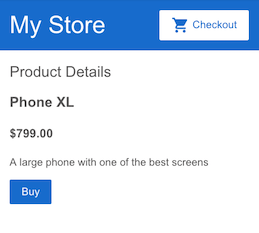
<h4>{{ product.price | [currency](https://angular.io/api/common/CurrencyPipe) }}</h4>

<p>{{ product.description }}</p>

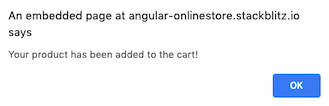
<button (click)="addToCart(product)">Buy</button>

</div>

1. To see the new "Buy" button, refresh the application and click on a product's name to display its details.



1. Click the "Buy" button to add the product to the stored list of items in the cart and display a confirmation message.



## Create the cart page

At this point, users can put items in the cart by clicking "Buy", but they can't yet see their cart.

Create the cart page in two steps:

1. Create a cart component and configure routing to the new component. At this point, the cart page will only have default text.
2. Display the cart items.

### Set up the component

To create the cart page, begin by following the same steps you did to create the product details component and configure routing for the new component.

1. Generate a cart component, named cart.

Reminder: In the file list, right-click the app folder, choose Angular Generator and [Component](https://angular.io/api/core/Component).

src/app/cart/cart.component.ts

content\_copyimport { [Component](https://angular.io/api/core/Component), [OnInit](https://angular.io/api/core/OnInit) } from '@angular/core';

@[Component](https://angular.io/api/core/Component)({

selector: 'app-cart',

[templateUrl](https://angular.io/api/core/Component#templateUrl): './cart.component.html',

[styleUrls](https://angular.io/api/core/Component#styleUrls): ['./cart.component.css']

})

export class CartComponent implements [OnInit](https://angular.io/api/core/OnInit) {

constructor() { }

ngOnInit() {

}

}

1. Add routing (a URL pattern) for the cart component.

Open app.module.ts and add a route for the component CartComponent, with a path of cart:

src/app/app.module.ts

content\_copy@[NgModule](https://angular.io/api/core/NgModule)({

imports: [

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule),

[ReactiveFormsModule](https://angular.io/api/forms/ReactiveFormsModule),

RouterModule.forRoot([

{ path: '', component: ProductListComponent },

{ path: 'products/:productId', component: ProductDetailsComponent },

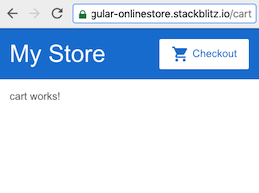
{ path: 'cart', component: CartComponent },

])

],

1. To see the new cart component, click the "Checkout" button. You can see the "cart works!" default text, and the URL has the pattern https://getting-started.stackblitz.io/cart, where getting-started.stackblitz.io may be different for your StackBlitz project.

The starter code for the "Checkout" button already includes a [routerLink](https://angular.io/api/router/RouterLink) for /cart the top-bar component.



### Display the cart items

You can use services to share data across components:

* The product details component already uses the cart service to add products to the cart.
* This section shows you how to use the cart service to display the products in the cart.

1. Open cart.component.ts.
2. Configure the component to use the cart service.
   1. Import the CartService from the cart.service.ts file.

src/app/cart/cart.component.ts

content\_copyimport { [Component](https://angular.io/api/core/Component) } from '@angular/core';

import { CartService } from '../cart.service';

* 1. Inject the CartService so that the cart component can use it.

src/app/cart/cart.component.ts

content\_copyexport class CartComponent {

constructor(

private cartService: CartService

) { }

}

1. Define the items property to store the products in the cart.

src/app/cart/cart.component.ts

content\_copyexport class CartComponent {

items;

constructor(

private cartService: CartService

) { }

}

1. Set the items using the cart service's getItems() method. Recall that you defined this method [when you generated cart.service.ts](https://angular.io/start/start-data#generate-cart-service).

The resulting CartComponent class is as follows:

src/app/cart/cart.component.ts

content\_copyexport class CartComponent implements [OnInit](https://angular.io/api/core/OnInit) {

items;

constructor(

private cartService: CartService

) { }

ngOnInit() {

this.items = this.cartService.getItems();

}

}

1. Update the template with a header, and use a <div> with an \*[ngFor](https://angular.io/api/common/NgForOf) to display each of the cart items with its name and price.

The resulting CartComponent template is as follows:

src/app/cart/cart.component.html

content\_copy<h3>Cart</h3>

<div class="cart-item" \*[ngFor](https://angular.io/api/common/NgForOf)="let [item](https://angular.io/api/core/IterableChangeRecord#item) of items">

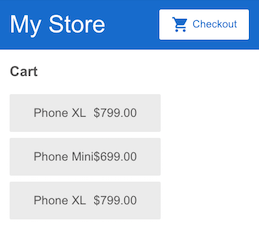
<span>{{ item.name }}</span>

<span>{{ item.price | [currency](https://angular.io/api/common/CurrencyPipe) }}</span>

</div>

1. Test your cart component.
   1. Click on "My Store" to go to the product list page.
   2. Click on a product name to display its details.
   3. Click "Buy" to add the product to the cart.
   4. Click "Checkout" to see the cart.
   5. To add another product, click "My Store" to return to the product list.

Repeat to add more items to the cart.



StackBlitz tip: Any time the preview refreshes, the cart is cleared. If you make changes to the app, the page refreshes, so you'll need to buy products again to populate the cart.

For more information about services, see [Introduction to Services and Dependency Injection](https://angular.io/guide/architecture-services).

## Retrieve shipping prices

Servers often return data in the form of a stream. Streams are useful because they make it easy to transform the returned data and make modifications to the way you request that data. The Angular HTTP client, [HttpClient](https://angular.io/api/common/http/HttpClient), is a built-in way to fetch data from external APIs and provide them to your app as a stream.

This section shows you how to use the HTTP client to retrieve shipping prices from an external file.

### Predefined shipping data

The application that StackBlitz generates for this guide comes with predefined shipping data in assets/shipping.json. Use this data to add shipping prices for items in the cart.

src/assets/shipping.json

content\_copy[

{

"type": "Overnight",

"price": 25.99

},

{

"type": "2-Day",

"price": 9.99

},

{

"type": "Postal",

"price": 2.99

}

]

### Use [HttpClient](https://angular.io/api/common/http/HttpClient) in the AppModule

Before you can use Angular's HTTP client, you must configure your app to use [HttpClientModule](https://angular.io/api/common/http/HttpClientModule).

Angular's [HttpClientModule](https://angular.io/api/common/http/HttpClientModule) registers the providers your app needs to use a single instance of the [HttpClient](https://angular.io/api/common/http/HttpClient) service throughout your app.

1. Open app.module.ts.

This file contains imports and functionality that is available to the entire app.

1. Import [HttpClientModule](https://angular.io/api/common/http/HttpClientModule) from the @angular/common/[http](https://angular.io/api/common/http) package at the top of the file with the other imports. As there are a number of other imports, this code snippet omits them for brevity. Be sure to leave the existing imports in place.

src/app/app.module.ts

content\_copyimport { [HttpClientModule](https://angular.io/api/common/http/HttpClientModule) } from '@angular/common/[http](https://angular.io/api/common/http)';

1. Add [HttpClientModule](https://angular.io/api/common/http/HttpClientModule) to the AppModule @[NgModule](https://angular.io/api/core/NgModule)() imports array to register Angular's [HttpClient](https://angular.io/api/common/http/HttpClient) providers globally.

src/app/app.module.ts

content\_copy@[NgModule](https://angular.io/api/core/NgModule)({

imports: [

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule),

[HttpClientModule](https://angular.io/api/common/http/HttpClientModule),

[ReactiveFormsModule](https://angular.io/api/forms/ReactiveFormsModule),

RouterModule.forRoot([

{ path: '', component: ProductListComponent },

{ path: 'products/:productId', component: ProductDetailsComponent },

{ path: 'cart', component: CartComponent },

])

],

declarations: [

AppComponent,

TopBarComponent,

ProductListComponent,

ProductAlertsComponent,

ProductDetailsComponent,

CartComponent,

],

bootstrap: [

AppComponent

]

})

export class AppModule { }

### Use [HttpClient](https://angular.io/api/common/http/HttpClient) in the cart service

Now that the AppModule imports the [HttpClientModule](https://angular.io/api/common/http/HttpClientModule), the next step is to inject the [HttpClient](https://angular.io/api/common/http/HttpClient) service into your service so your app can fetch data and interact with external APIs and resources.

1. Open cart.service.ts.
2. Import [HttpClient](https://angular.io/api/common/http/HttpClient) from the @angular/common/[http](https://angular.io/api/common/http) package.

src/app/cart.service.ts

content\_copyimport { [Injectable](https://angular.io/api/core/Injectable) } from '@angular/core';

import { [HttpClient](https://angular.io/api/common/http/HttpClient) } from '@angular/common/[http](https://angular.io/api/common/http)';

1. Inject [HttpClient](https://angular.io/api/common/http/HttpClient) into the CartService constructor:

src/app/cart.service.ts

content\_copyexport class CartService {

items = [];

constructor(

private [http](https://angular.io/api/common/http): [HttpClient](https://angular.io/api/common/http/HttpClient)

) {}

}

### Define the get() method

Multiple components can leverage the same service. Later in this tutorial, the shipping component uses the cart service to retrieve shipping data via HTTP from the shipping.json file. First, define a get() method.

1. Continue working in cart.service.ts.
2. Below the clearCart() method, define a new getShippingPrices() method that uses the [HttpClient](https://angular.io/api/common/http/HttpClient) get() method to retrieve the shipping data.

src/app/cart.service.ts

content\_copyexport class CartService {

items = [];

constructor(

private [http](https://angular.io/api/common/http): [HttpClient](https://angular.io/api/common/http/HttpClient)

) {}

addToCart(product) {

this.items.push(product);

}

getItems() {

return this.items;

}

clearCart() {

this.items = [];

return this.items;

}

getShippingPrices() {

return this.http.get('/assets/shipping.json');

}

}

For more information about Angular's [HttpClient](https://angular.io/api/common/http/HttpClient), see [HttpClient](https://angular.io/guide/http).

## Define the shipping page

Now that your app can retrieve shipping data, create a shipping component and template.

1. Generate a new component named shipping.

Reminder: In the file list, right-click the app folder, choose Angular Generator and [Component](https://angular.io/api/core/Component).

src/app/shipping/shipping.component.ts

content\_copyimport { [Component](https://angular.io/api/core/Component), [OnInit](https://angular.io/api/core/OnInit) } from '@angular/core';

@[Component](https://angular.io/api/core/Component)({

selector: 'app-shipping',

[templateUrl](https://angular.io/api/core/Component#templateUrl): './shipping.component.html',

[styleUrls](https://angular.io/api/core/Component#styleUrls): ['./shipping.component.css']

})

export class ShippingComponent implements [OnInit](https://angular.io/api/core/OnInit) {

constructor() { }

ngOnInit() {

}

}

1. In app.module.ts, add a route for shipping. Specify a path of shipping and a component of ShippingComponent.

src/app/app.module.ts

content\_copy@[NgModule](https://angular.io/api/core/NgModule)({

imports: [

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule),

[HttpClientModule](https://angular.io/api/common/http/HttpClientModule),

[ReactiveFormsModule](https://angular.io/api/forms/ReactiveFormsModule),

RouterModule.forRoot([

{ path: '', component: ProductListComponent },

{ path: 'products/:productId', component: ProductDetailsComponent },

{ path: 'cart', component: CartComponent },

{ path: 'shipping', component: ShippingComponent },

])

],

declarations: [

AppComponent,

TopBarComponent,

ProductListComponent,

ProductAlertsComponent,

ProductDetailsComponent,

CartComponent,

ShippingComponent

],

bootstrap: [

AppComponent

]

})

export class AppModule { }

There's no link to the new shipping component yet, but you can see its template in the preview pane by entering the URL its route specifies. The URL has the pattern: https://getting-started.stackblitz.io/shipping where the getting-started.stackblitz.io part may be different for your StackBlitz project.

1. Modify the shipping component so that it uses the cart service to retrieve shipping data via HTTP from the shipping.json file.
   1. Import the cart service.

src/app/shipping/shipping.component.ts

content\_copyimport { [Component](https://angular.io/api/core/Component), [OnInit](https://angular.io/api/core/OnInit) } from '@angular/core';

import { CartService } from '../cart.service';

* 1. Define a shippingCosts property.

src/app/shipping/shipping.component.ts

content\_copyexport class ShippingComponent implements [OnInit](https://angular.io/api/core/OnInit) {

shippingCosts;

}

* 1. Inject the cart service in the ShippingComponent constructor:

src/app/shipping/shipping.component.ts

content\_copyconstructor(

private cartService: CartService

) {

}

* 1. Set the shippingCosts property using the getShippingPrices() method from the cart service.

src/app/shipping/shipping.component.ts

content\_copyexport class ShippingComponent implements [OnInit](https://angular.io/api/core/OnInit) {

shippingCosts;

constructor(

private cartService: CartService

) {

}

ngOnInit() {

this.shippingCosts = this.cartService.getShippingPrices();

}

}

1. Update the shipping component's template to display the shipping types and prices using the [async](https://angular.io/api/core/testing/async) pipe:

src/app/shipping/shipping.component.html

content\_copy<h3>Shipping Prices</h3>

<div class="shipping-item" \*[ngFor](https://angular.io/api/common/NgForOf)="let shipping of shippingCosts | async">

<span>{{ shipping.type }}</span>

<span>{{ shipping.price | [currency](https://angular.io/api/common/CurrencyPipe) }}</span>

</div>

The [async](https://angular.io/api/core/testing/async) pipe returns the latest value from a stream of data and continues to do so for the life of a given component. When Angular destroys that component, the [async](https://angular.io/api/core/testing/async) pipe automatically stops. For detailed information about the [async](https://angular.io/api/core/testing/async) pipe, see the [AsyncPipe API documentation](https://angular.io/api/common/AsyncPipe).

1. Add a link from the cart page to the shipping page:

src/app/cart/cart.component.html

content\_copy<h3>Cart</h3>

<p>

<a [routerLink](https://angular.io/api/router/RouterLink)="/shipping">Shipping Prices</a>

</p>

<div class="cart-item" \*[ngFor](https://angular.io/api/common/NgForOf)="let [item](https://angular.io/api/core/IterableChangeRecord#item) of items">

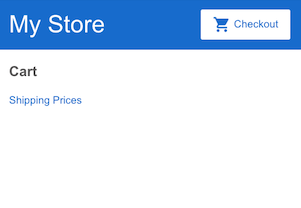
<span>{{ item.name }}</span>

<span>{{ item.price | [currency](https://angular.io/api/common/CurrencyPipe) }}</span>

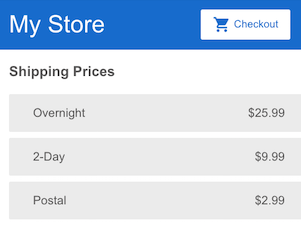
</div>

1. Test your shipping prices feature:

Click the "Checkout" button to see the updated cart. Remember that changing the app causes the preview to refresh, which empties the cart.



Click on the link to navigate to the shipping prices.



## Next steps

Congratulations! You have an online store application with a product catalog and shopping cart. You can also look up and display shipping prices.

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

* [Continue to the "Forms" section](https://angular.io/start/start-forms) to finish the app by adding the shopping cart page and a checkout form.
* [Skip ahead to the "Deployment" section](https://angular.io/start/start-deployment) to move to local development, or deploy your app to Firebase or your own server.