Try it: Manage data

Contents ›

At the end of In-app Navigation, 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 view 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` 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.

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 view and set up a cart service to store information about products in the cart.

> A later part of this tutorial, Use forms for user input, guides you through accessing this cart service from the view where the user checks out.

Define a cart service

1. To generate a cart service, right click on the `app` folder, choose `Angular Generator`, and choose `Service`. Name the new service `cart`.

   src/app/cart.service.ts

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

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

     constructor() {}

   }
   ```

   Introduction to Services and Dependency Injection.

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

   src/app/cart.service.ts

   ```
   export class CartService {
     items = [];
   }
   ```

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

   src/app/cart.service.ts

   ```
   export class CartService {
     items = [];

     addToCart(product) {
       this.items.push(product);
     }

     getItems() {
       return this.items;
     }

     clearCart() {
   ```

```
      this.items = [];
      return this.items;
    }
  }
```

- The `addToCart()` method appends a product to an array of `items`.
- The `getItems()` method collects the items users add to the cart and returns each item with its associated quantity.
- 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.
   a. Import the cart service.

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

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

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

   b. Inject the cart service by adding it to the `constructor()`.

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

   ```
   export class ProductDetailsComponent implements OnInit {
     constructor(
       private route: ActivatedRoute,
       private cartService: CartService
     ) { }
   }
   ```

3. Define the `addToCart()` method, which adds the current product to the cart.
   The `addToCart()` method does the following three things:
   - Receives the current `product`.
   - Uses the cart service's `addToCart()` method to add the product the cart.

- Displays a message that you've added a product to the cart.

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

```
export class ProductDetailsComponent implements OnInit {
  addToCart(product) {
    this.cartService.addToCart(product);
    window.alert('Your product has been added to the cart!');
  }
}
```

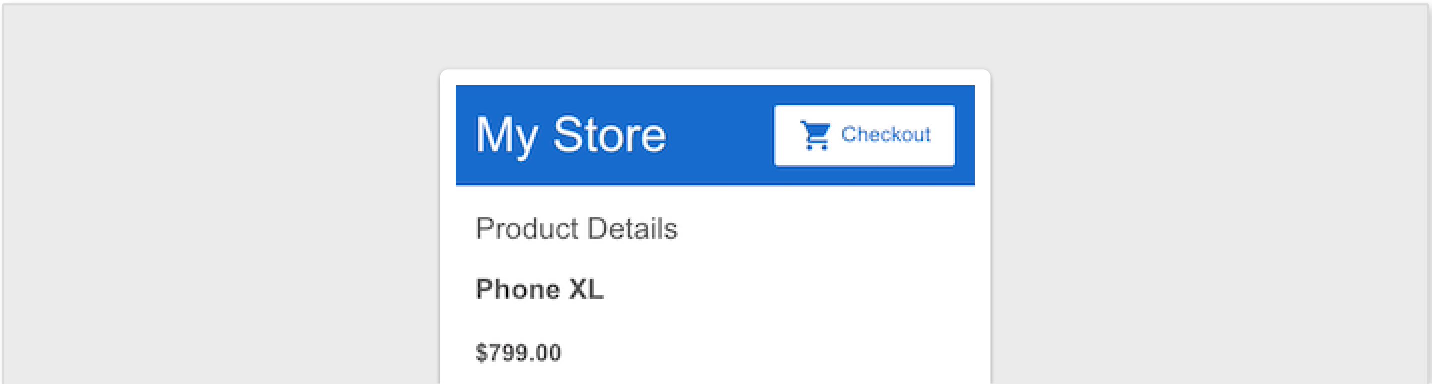
4. Update the product details template with a "Buy" button that adds the current product to the cart.
   a. Open `product-details.component.html`.
   b. Add a button with the label "Buy", and bind the `click()` event to the `addToCart()` method:

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

```
<h2>Product Details</h2>

<div *ngIf="product">
  <h3>{{ product.name }}</h3>
  <h4>{{ product.price | currency }}</h4>
  <p>{{ product.description }}</p>

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

> The line, `<h4>{{ product.price | currency }}</h4>` uses the `currency` pipe to transform `product.price` from a number to a currency string. A pipe is a way you can transform data in your HTML template. For more information about Angular pipes, see Pipes.

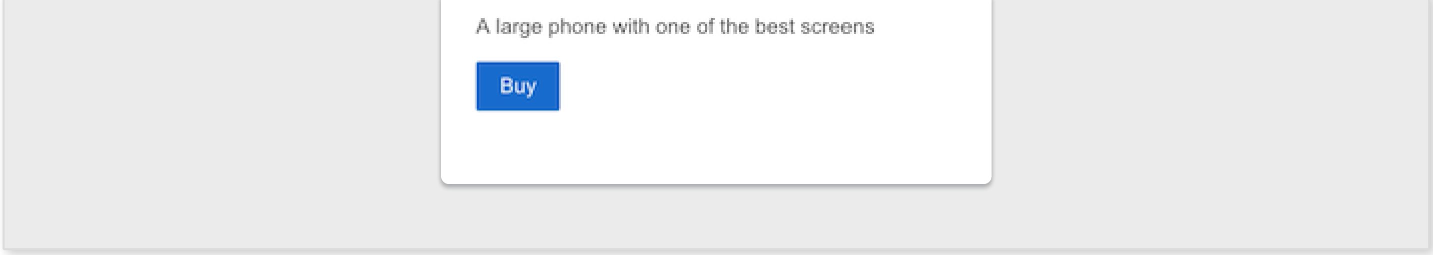
5. To see the new "Buy" button, refresh the application and click on a product's name to display its details.



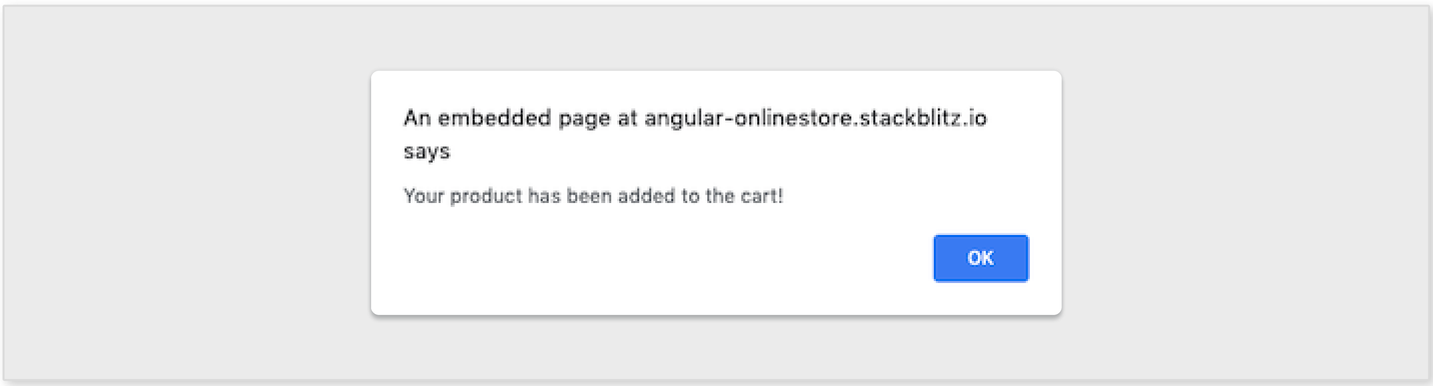
Product Details

Phone XL

$799.00


A large phone with one of the best screens

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


An embedded page at angular-onlinestore.stackblitz.io says

Your product has been added to the cart!

Create the cart view

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

Create the cart view in two steps:

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

Set up the component

To create the cart view, 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`.

src/app/cart/cart.component.ts

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

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

  ngOnInit() {
  }

}
```

2. 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

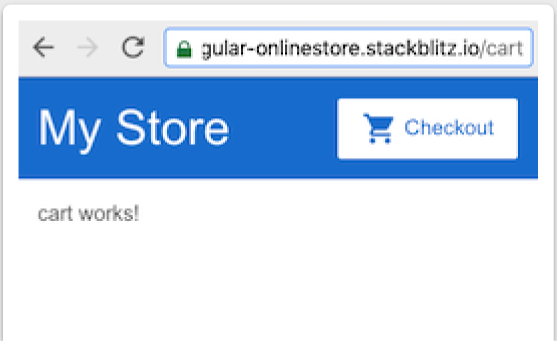
```
@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    RouterModule.forRoot([
      { path: '', component: ProductListComponent },
      { path: 'products/:productId', component: ProductDetailsComponent },
      { path: 'cart', component: CartComponent },
    ])
  ],
```

3. Update the "Checkout" button so that it routes to the `/cart` url.
   Open `top-bar.component.html` and add a `routerLink` directive pointing to `/cart`.

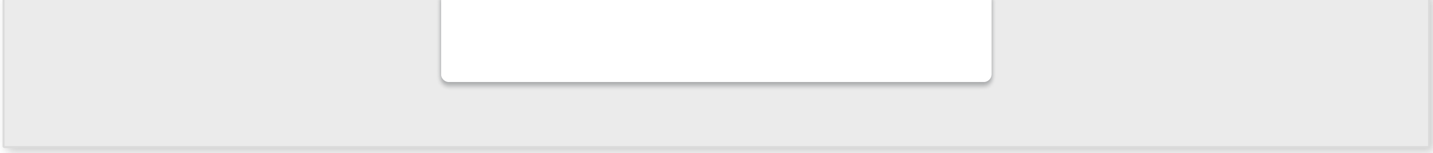
src/app/top-bar/top-bar.component.html

```
<a routerLink="/cart" class="button fancy-button">
  <i class="material-icons">shopping_cart</i>Checkout
</a>
```

4. 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.



cart works!



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.
   a. Import the `CartService` from the `cart.service.ts` file.

src/app/cart/cart.component.ts

```
import { Component } from '@angular/core';
import { CartService } from '../cart.service';
```

   b. Inject the `CartService` so that the cart component can use it.

src/app/cart/cart.component.ts

```
export class CartComponent {

  constructor(
    private cartService: CartService
  ) { }
}
```

3. Define the `items` property to store the products in the cart.

src/app/cart/cart.component.ts

```
export class CartComponent {
  items;

  constructor(
    private cartService: CartService
  ) { }
}
```

4. Set the items using the cart service's `getItems()` method. Recall that you defined this method when you generated `cart.service.ts`.

   The resulting `CartComponent` class is as follows:

src/app/cart/cart.component.ts

```
export class CartComponent implements OnInit {
  items;

  constructor(
    private cartService: CartService
  ) { }

  ngOnInit() {
    this.items = this.cartService.getItems();
  }
}
```

5. Update the template with a header, and use a `<div>` with an `*ngFor` to display each of the cart items with its name and price.

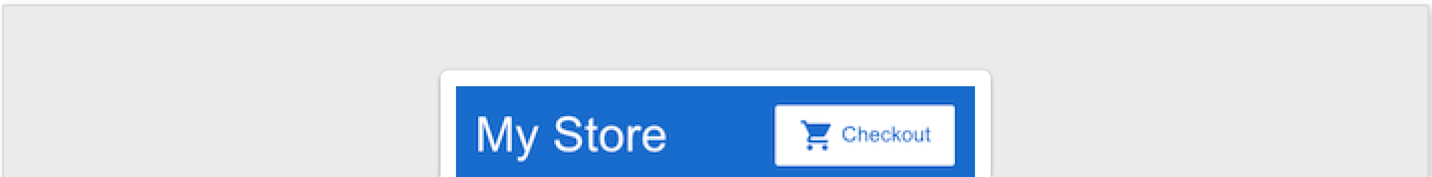
   The resulting `CartComponent` template is as follows:

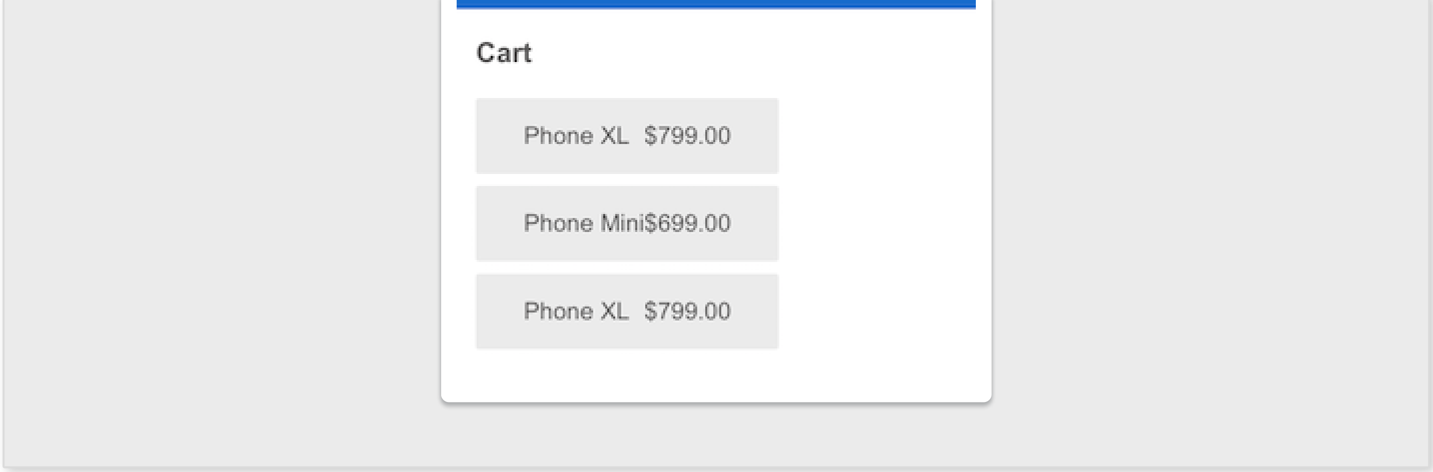
src/app/cart/cart.component.html

```
<h3>Cart</h3>

<div class="cart-item" *ngFor="let item of items">
  <span>{{ item.name }}</span>
  <span>{{ item.price | currency }}</span>
</div>
```

6. Test your cart component.
   a. Click on "My Store" to go to the product list view.
   b. Click on a product name to display its details.
   c. Click "Buy" to add the product to the cart.
   d. Click "Checkout" to see the cart.
   e. To add another product, click "My Store" to return to the product list.

   Repeat to add more items to the cart.




Cart

Phone XL $799.00

Phone Mini$699.00

Phone XL $799.00

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.

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`, 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

```
[
  {
    "type": "Overnight",
    "price": 25.99
  },
  {
    "type": "2-Day",
    "price": 9.99
  },
  {
    "type": "Postal",
    "price": 2.99
    }
  ]
```

Use HttpClient in the AppModule

Before you can use Angular's HTTP client, you must configure your app to use `HttpClientModule`.

Angular's `HttpClientModule` registers the providers your app needs to use a single instance of the `HttpClient` service throughout your app.

1. Open `app.module.ts`.
   This file contains imports and functionality that is available to the entire app.

2. Import `HttpClientModule` from the `@angular/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

   ```
   import { HttpClientModule } from '@angular/common/http';
   ```

3. Add `HttpClientModule` to the `AppModule` `@NgModule()` `imports` array to register Angular's `HttpClient` providers globally.

   src/app/app.module.ts

   ```
   @NgModule({
     imports: [
       BrowserModule,
       HttpClientModule,
       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` in the cart service

Now that the `AppModule` imports the `HttpClientModule`, the next step is to inject the `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` from the `@angular/common/http` package.

src/app/cart.service.ts

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

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

3. Inject `HttpClient` into the `CartService` constructor:

src/app/cart.service.ts

```
export class CartService {
  items = [];

  constructor(
    private 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` `get()` method to retrieve the shipping data.

src/app/cart.service.ts

```
export class CartService {
    items = [];

    constructor(
      private 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`, see the Client-Server Interaction guide.

Define the shipping view

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`.

src/app/shipping/shipping.component.ts

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

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

  ngOnInit() {
  }

}
```

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

src/app/app.module.ts

```
@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule,
    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.

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

   a. Import the cart service.

   src/app/shipping/shipping.component.ts

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

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

   b. Define a `shippingCosts` property.

   src/app/shipping/shipping.component.ts

   ```
   export class ShippingComponent implements OnInit {
     shippingCosts;
   }
   ```

   c. Inject the cart service in the `ShippingComponent` constructor:

   src/app/shipping/shipping.component.ts

   ```
   constructor(
     private cartService: CartService
   ) {
   }
   ```

   d. Set the `shippingCosts` property using the `getShippingPrices()` method from the cart service.

   src/app/shipping/shipping.component.ts

   ```
   export class ShippingComponent implements OnInit {
     shippingCosts;

     constructor(
       private cartService: CartService
     ) {
     }

     ngOnInit() {
       this.shippingCosts = this.cartService.getShippingPrices();
     }
   ```

```
}
```

4. Update the shipping component's template to display the shipping types and prices using the `async` pipe:

src/app/shipping/shipping.component.html

```
<h3>Shipping Prices</h3>

<div class="shipping-item" *ngFor="let shipping of shippingCosts | async">
  <span>{{ shipping.type }}</span>
  <span>{{ shipping.price | currency }}</span>
</div>
```

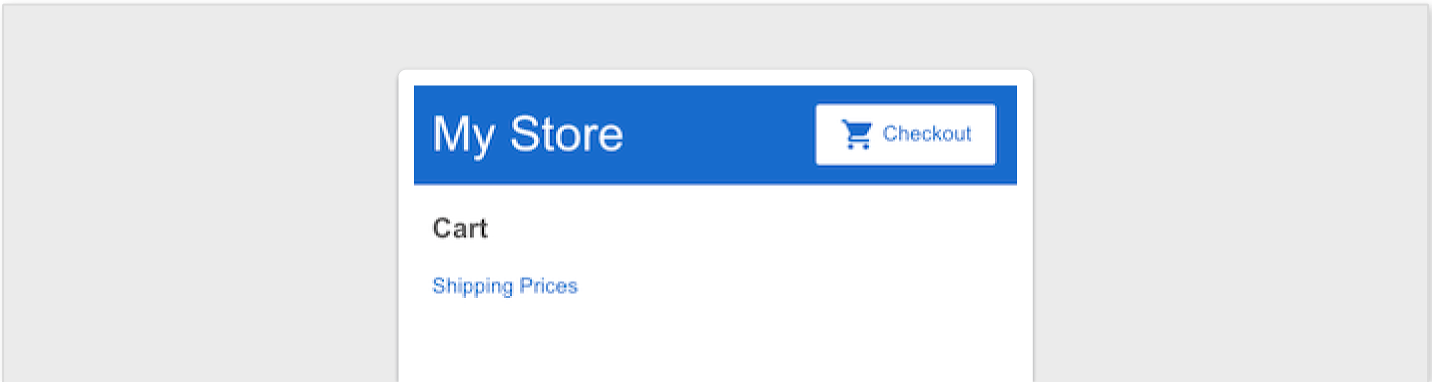
The `async` pipe returns the latest value from a stream of data and continues to do so for the life of a given component. When Angular destroys that component, the `async` pipe automatically stops. For detailed information about the `async` pipe, see the AsyncPipe API documentation.

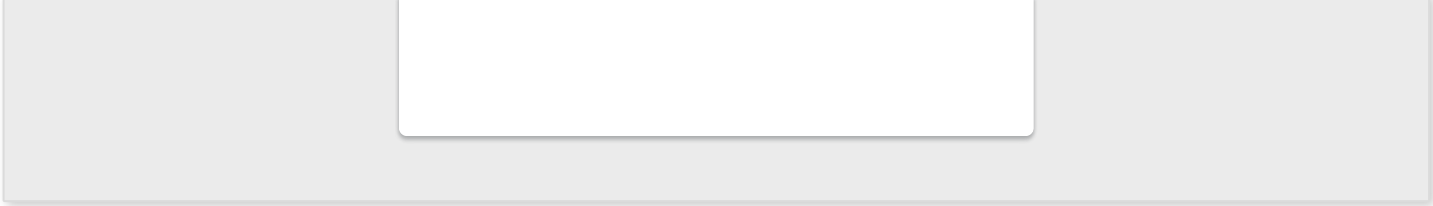
5. Add a link from the cart view to the shipping view:

src/app/cart/cart.component.html

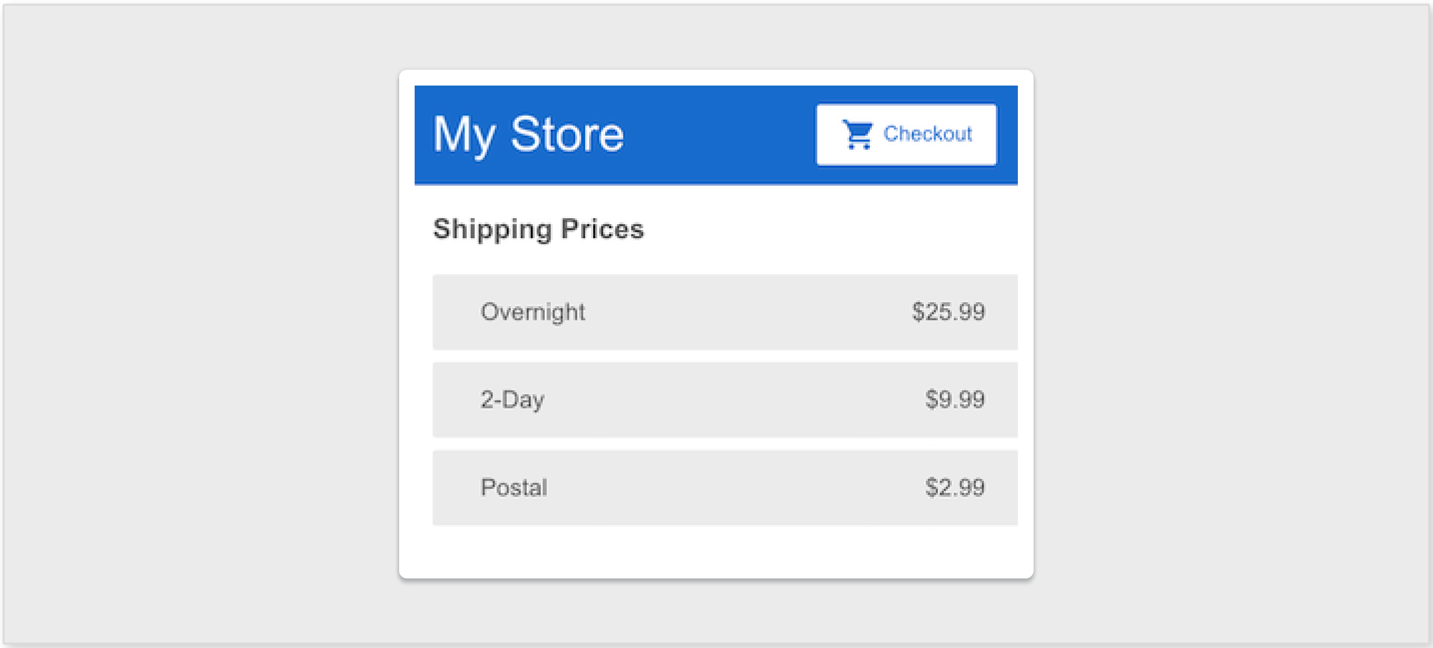
```
<h3>Cart</h3>

<p>
  <a routerLink="/shipping">Shipping Prices</a>
</p>

<div class="cart-item" *ngFor="let item of items">
  <span>{{ item.name }}</span>
  <span>{{ item.price | currency }}</span>
</div>
```

6. 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.



Shipping Prices

Overnight	$25.99
2-Day	$9.99
Postal	$2.99

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 to finish the app by adding the shopping cart view and a checkout form.
- Skip ahead to the "Deployment" section to move to local development, or deploy your app to Firebase or your own server.