Skip to main content

Managing data

This guide builds on the second step of the Getting started with a basic Angular application tutorial, Adding navigation. At this stage of development, the 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 step of the tutorial guides you through creating a shopping cart in the following 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

---

Create the shopping cart service

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.

Currently, users can view product information, and the application can simulate sharing and notifications about product changes.

The next step is to build a way for users to add products to a cart. This section walks you through adding a **Buy** button and setting up a cart service to store information about products in the cart.

Skip to main content

ervice

This section walks you through creating the `CartService` that tracks products added to shopping cart.

Skip to main content

enerate a new `cart` service by running the and:

```
ng generate service cart
```

2. Import the `Product` interface from `./products.ts` into the `cart.service.ts` file, and in the `CartService` class, define an `items` property to store the array of the current products in the cart.

src/app/cart.service.ts

```
import { Product } from './products';
import { Injectable } from '@angular/core';
/* . . . */
@Injectable({
  providedIn: 'root'
})
export class CartService {
  items: Product[] = [];
/* . . . */
}
```

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

Skip to main content

service.ts

```
@Injectable({
  providedIn: 'root'
})
export class CartService {
  items: Product[] = [];
/* . . . */

  addToCart(product: 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, which empties the cart

Use the cart service

This section walks you through using the `CartService` to add a product to the cart.

Skip to main content

ails.component.ts, import the cart service.

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

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

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

2. 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.

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

```
export class ProductDetailsComponent implements
OnInit {

  addToCart(product: Product) {
    this.cartService.addToCart(product);
    window.alert('Your product has been added to
the cart!');
  }
}
```

The `addToCart()` method does the following:

Skip to main content

current `product` as an argument

`CartService` `addToCart()` method to add the product to the cart

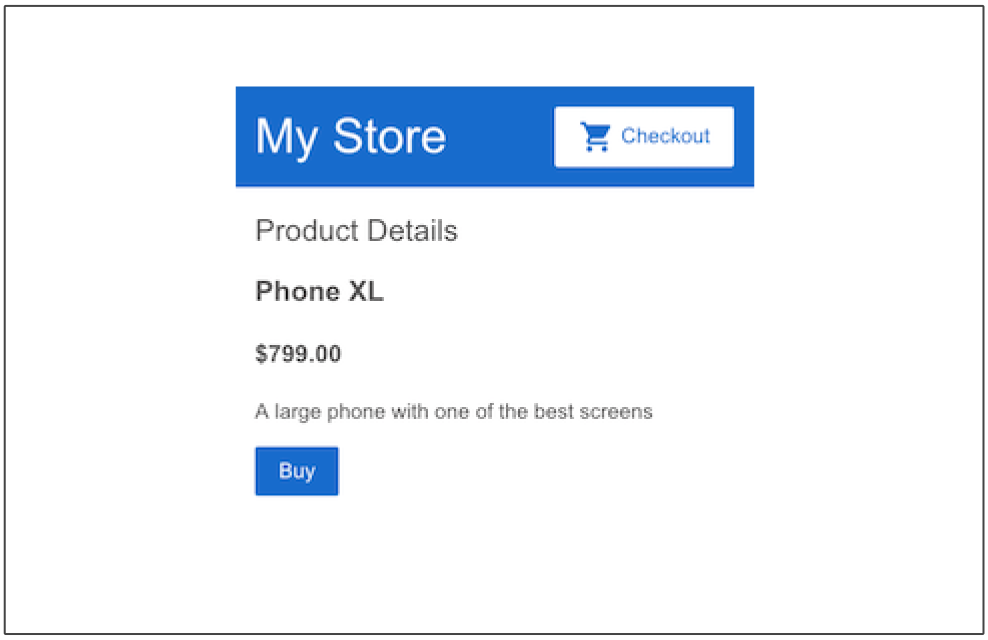
- Displays a message that you've added a product to the cart

4. In `product-details.component.html`, add a button with the label **Buy**, and bind the `click()` event to the `addToCart()` method. This code updates the product details template with a **Buy** button that adds the current product to the cart.

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

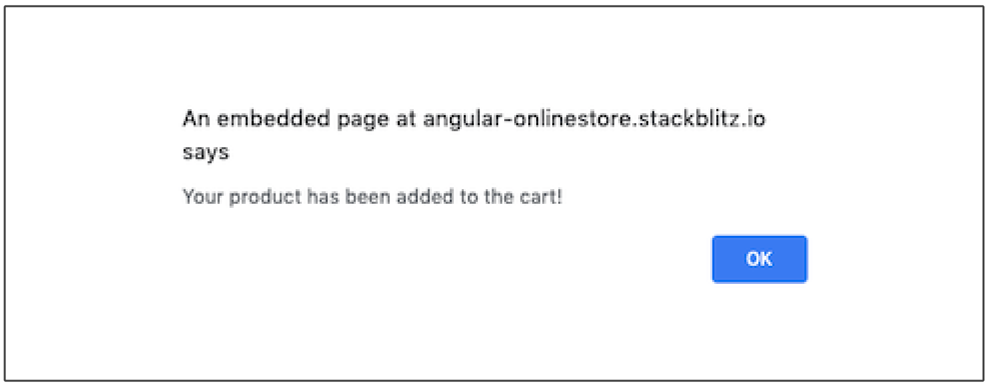
5. Verify that the new **Buy** button appears as expected by refreshing the application and clicking on a product's name to display its details.



Skip to main content

tton to add the product to the stored list of items in

play a confirmation message.


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

Your product has been added to the cart!

Create the cart view

For customers to see their cart, you can create the cart view in two steps:

1. Create a cart component and configure routing to the new component.
2. Display the cart items.

Set up the cart component

To create the cart view, follow the same steps you did to create the `ProductDetailsComponent` and configure routing for the new component.

Skip to main content

component named `cart` in the terminal by running

mmand:

```
ng generate component cart
```

This command will generate the `cart.component.ts` file and its associated template and styles files.

src/app/cart/cart.component.ts

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

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


}
```

2. Notice that the newly created `CartComponent` is added to the module's `declarations` in `app.module.ts`.

Skip to main content

module.ts

```
import { CartComponent } from
'./cart/cart.component';

@NgModule({
  declarations: [
    AppComponent,
    TopBarComponent,
    ProductListComponent,
    ProductAlertsComponent,
    ProductDetailsComponent,
    CartComponent,
  ],
```

3. Still in `app.module.ts`, 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 },
    ])
  ],
```

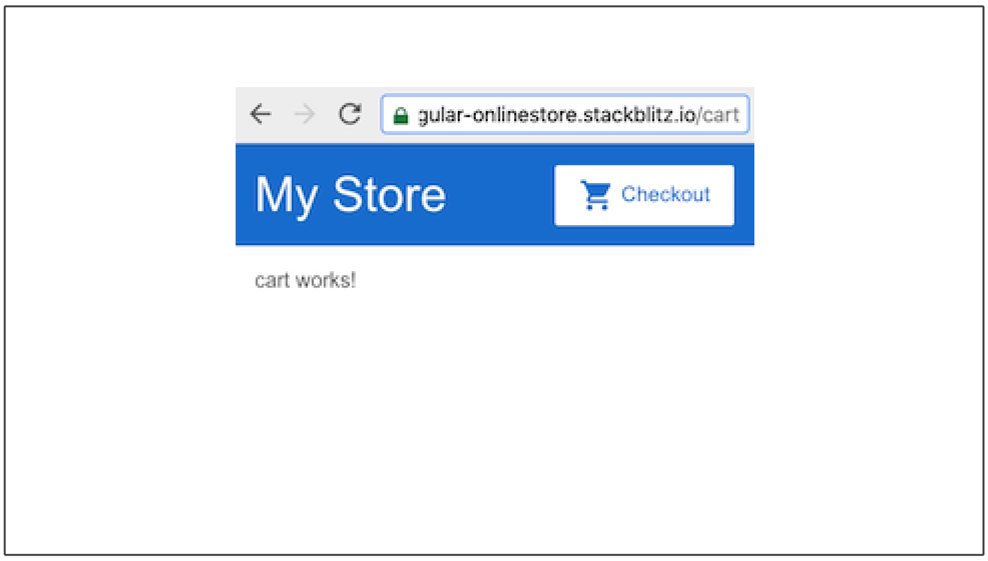
4. Update the **Checkout** button so that it routes to the `/cart` URL. In `top-bar.component.html`, add a `routerLink` directive pointing to `/cart`.

Skip to main content

ar/top-bar.component.html

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

5. Verify the new `CartComponent` works as expected by clicking 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

This section shows you how to use the cart service to display the products in the cart.

Skip to main content

ent.ts, import the `CartService` from the ts file.

src/app/cart/cart.component.ts

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

2. Inject the `CartService` so that the `CartComponent` can use it by adding it to the `constructor()`.

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 = this.cartService.getItems();

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

This code sets the items using the `CartService` `getItems()` method. You defined this method when you created `cart.service.ts`.

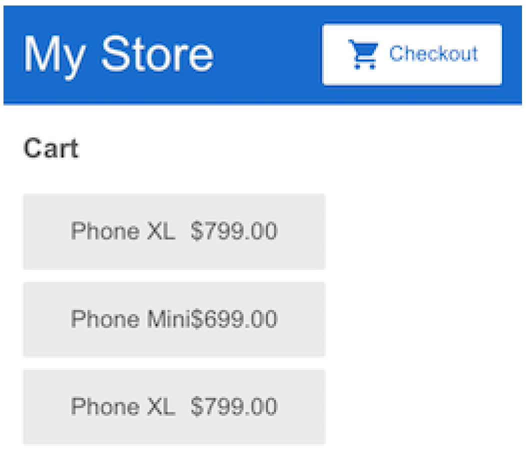
Skip to main content template with a header, and use a `<div>` with an
lay each of the cart items with its name and price.

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>
```

5. Verify that your cart works as expected:
   a. Click **My Store**.
   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.



For more information about services, see Introduction to Services and Dependency Injection.

Skip to main content

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. Angular `HttpClient` is a built-in way to fetch data from external APIs and provide them to your application as a stream.

This section shows you how to use `HttpClient` to retrieve shipping prices from an external file.

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
  }
]
```

Configure `AppModule` to use `HttpClient`

To use Angular's `HttpClient`, you must configure your application to use `HttpClientModule`.

Skip to main content

`tModule` registers the providers your application

`pClient` service throughout your application.

Skip to main content

ts, import `HttpClientModule` from the on/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';
```

2. To register Angular's `HttpClient` providers globally, add `HttpClientModule` to the `AppModule` `@NgModule()` `imports` array.

Skip to main content

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 { }
```

Configure `CartService` to use `HttpClient`

The next step is to inject the `HttpClient` service into your service so your application can fetch data and interact with external APIs and resources.

Skip to main content

e.ts, import HttpClient from the on/http package.

src/app/cart.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Product } from './products';
import { Injectable } from '@angular/core';
```

2. Inject HttpClient into the CartService constructor().

src/app/cart.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class CartService {
  items: Product[] = [];

  constructor(
    private http: HttpClient
  ) {}
/* . . . */
}
```

Configure CartService to get shipping prices

To get shipping data, from shipping.json, You can use the HttpClient get() method.

Skip to main content

e.ts, below the clearCart() method, define a ngPrices() method that uses the HttpClient get() method.

src/app/cart.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class CartService {
/* . . . */
  getShippingPrices() {
    return this.http.get<{type: string, price:
number}[]>('/assets/shipping.json');
  }
}
```

For more information about Angular's HttpClient, see the Client-Server Interaction guide.

Create a shipping component

Now that you've configured your application to retrieve shipping data, you can create a place to render that data.

Skip to main content

component named `shipping` in the terminal by wing command:

```
ng generate component shipping
```

This command will generate the `shipping.component.ts` file and it associated template and styles files.

src/app/shipping/shipping.component.ts

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

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

}
```

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

Skip to main content

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://angular-ynqttp--4200.local.webcontainer.io/shipping` where the `angular-ynqttp--4200.local.webcontainer.io` part may be different for your StackBlitz project.

Skip to main content

e `ShippingComponent` to use

This section guides you through modifying the `ShippingComponent` to retrieve shipping data via HTTP from the `shipping.json` file.

Skip to main content

mponent.ts, import CartService.

src/app/shipping/shipping.component.ts

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

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

2. Inject the cart service in the `ShippingComponent` `constructor()`.

src/app/shipping/shipping.component.ts

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

3. Define a `shippingCosts` property that sets the `shippingCosts` property using the `getShippingPrices()` method from the `CartService`. Initialize the `shippingCosts` property inside `ngOnInit()` method.

src/app/shipping/shipping.component.ts

```
export class ShippingComponent implements OnInit {

  shippingCosts!: Observable<{ type: string,
price: number }[]>;

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

}
```

Skip to main content

ppingComponent template to display the shipping 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>
```

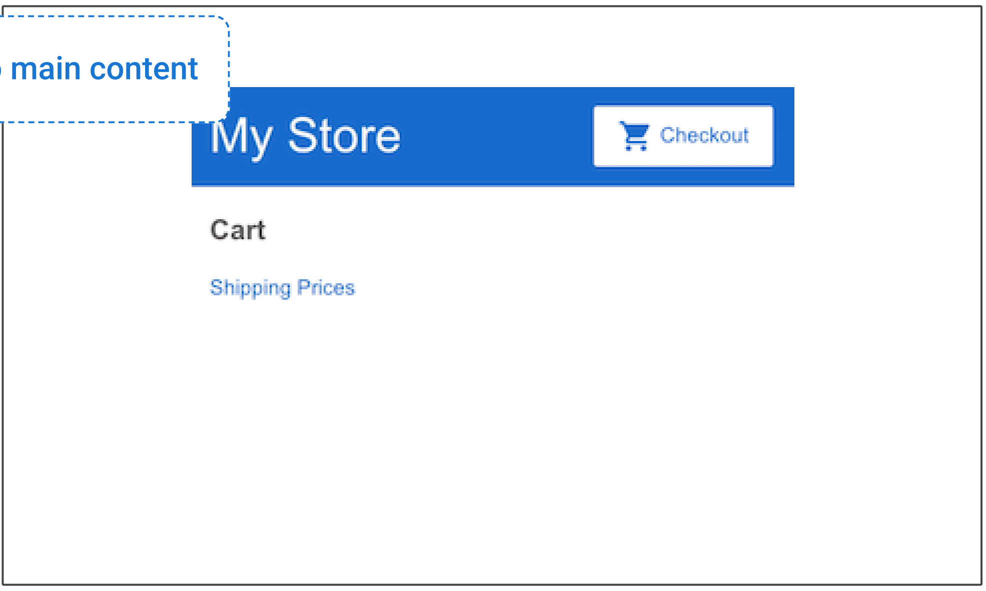
The `async` pipe returns the latest value from a stream of data and continues to do so for the life of a given component. When Angular destroys that component, the `async` pipe automatically stops. For detailed information about the `async` pipe, see the AsyncPipe API documentation.

5. Add a link from the `CartComponent` view to the `ShippingComponent` view.

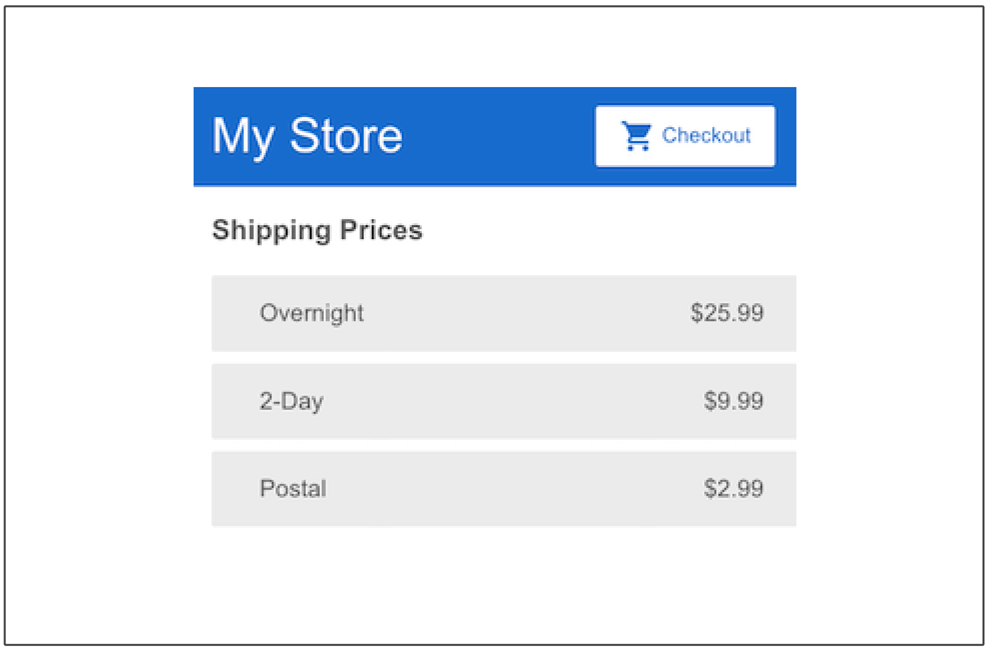
src/app/cart/cart.component.html

```
<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. Click the **Checkout** button to see the updated cart. Remember that changing the application 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

What's next

You now have a store application with a product catalog, a shopping cart, and you can look up shipping prices.

To continue exploring Angular:

- Continue to Forms for User Input to finish the application by adding the shopping cart view and a checkout form
- Skip ahead to Deployment to move to local development, or deploy your application to Firebase or your own server

Last reviewed on Mon Feb 28 2022