Routing

At the end of Your First App, the online store application has a basic product catalog. The app doesn't have any variable states or navigation. There is one URL, and that URL always displays the "My Store" page with a list of products and their descriptions.

In this section, you'll extend the app to display full product details in separate pages, with their own URLs.

To do this, you'll use the Angular *router*. The Angular router enables you to show different components and data to the user based on where the user is in the application. The router enables navigation from one view to the next as users perform application tasks:

- Enter a URL in the address bar, and the browser navigates to a corresponding page.
- Click links on the page, and the browser navigates to a new page.
- Click the browser's back and forward buttons, and the browser navigates backward and forward through the history of pages you've seen.

Registering a route

The app is already set up to use the Angular router and to use routing to navigate to the product list component you modified earlier. Let's define a route to show individual product details.

1. Generate a new component for product details. Give the component the name `product-details`.
   Reminder: In the file list, right-click the `app` folder, choose `Angular Generator` and `Component`.
2. In `app.module.ts`, add a route for product details, with a `path` of `products/:productId` and `ProductDetailsComponent` for the `component`.

   src/app/app.module.ts

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

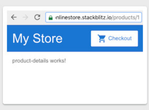
   A route associates one or more URL paths with a component.
3. Define a link using the `RouterLink` directive. The `routerLink` defines how the user navigates to the route (or URL) declaratively in the component template.
   We want the user to click a product name to display the details for that product.
   a. Open `product-list.component.html`.
   b. Update the `*ngFor` directive to assign each index in the `products` array to the `productId` variable when iterating over the list.
   c. Modify the product name anchor to include a `routerLink`.

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

   ```
   <div *ngFor="let product of products; index as productId">

     <h3>
       <a [title]="product.name + ' details'" [routerLink]="['/products', productId]">
         {{ product.name }}
       </a>
     </h3>
   <!-- . . . -->
   </div>
   ```

   The RouterLink directive gives the router control over the anchor element. In this case, the route (URL) contains one fixed segment (`/products`) and the final segment is variable, inserting the id property of the current product. For example, the URL for a product with an `id` of 1 will be similar to `https://getting-started-myfork.stackblitz.io/products/1`.
4. Test the router by clicking a product name. The app displays the product details component, which currently always says "product-details works!" (We'll fix this in the next section.)
   Notice that the URL in the preview window changes. The final segment is `products/1`.



Using route information

The product details component handles the display of each product. The Angular Router displays components based on the browser's URL and your defined routes. You'll use the Angular Router to combine the `products` data and route information to display the specific details for each product.

1. Open `product-details.component.ts`
2. Arrange to use product data from an external file.
   a. Import `ActivatedRoute` from the `@angular/router` package, and the `products` array from `../products`.

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

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

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

   b. Define the `product` property and inject the `ActivatedRoute` into the constructor.

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

   ```
   export class ProductDetailsComponent implements OnInit
   {
     product;

     constructor(
       private route: ActivatedRoute,
     ) { }

   }
   ```

   The `ActivatedRoute` is specific to each routed component loaded by the Angular Router. It contains information about the route, its parameters, and additional data associated with the route.
3. In the `ngOnInit()` method, *subscribe* to route params and fetch the product based on the `productId`.

   ```
   ngOnInit() {
     this.route.paramMap.subscribe(params => {
       this.product = products[+params.get('productId')];
     });
   }
   ```

   Angular calls `ngOnInit()` shortly after creating a component.
   The route parameters correspond to the path variables defined in the route. The `productId` is provided from the URL that was matched to the route. You use the `productId` to display the details for each unique product.

   > For more information on `ngOnInit()`, see Lifecycle hooks.

4. Update the template to display product details information inside an `*ngIf`.

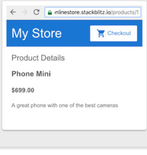
   src/app/product-details/product-details.component.html

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

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

   </div>
   ```

Now, when the user clicks on a name in the product list, the router navigates you to the distinct URL for the product, swaps out the product list component for the product details component, and displays the product details.



> Learn more: See Routing & Navigation for more information about the Angular router.

Next steps

Congratulations! You have integrated routing into your online store.

- Products are linked from the product list page to individual products
- Users can click on a product name from the list to see details in a new view, with a distinct URL (route)

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

- Continue to the "Managing Data" section to add the shopping cart feature, using a service to manage the cart data and using HTTP to retrieve external data for shipping prices.
- Skip ahead to the Deployment section to deploy your app to Firebase or move to local development.