In-app navigation

Contents

At the end of part 1, 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.

This guide shows you how to use Angular routing to give the user in-app navigation. In a single-page app, instead of loading new pages, you show different components and data to the user based on where the user is in the application.

The router lets you display full product details in separate views, each with its own URL. Routing enables navigation from one view to the next (within the same page) as users perform tasks such as the following:

- Entering a URL in the address bar to navigate to a corresponding view.
- Clicking links on the page to navigate to a new view.
- Clicking the browser's back and forward buttons to navigate backward and forward through the browser history.

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. This section shows you how to 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. The directive configures the component template to define how the user navigates to the route or URL. When the user clicks a product name, the app displays 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`.

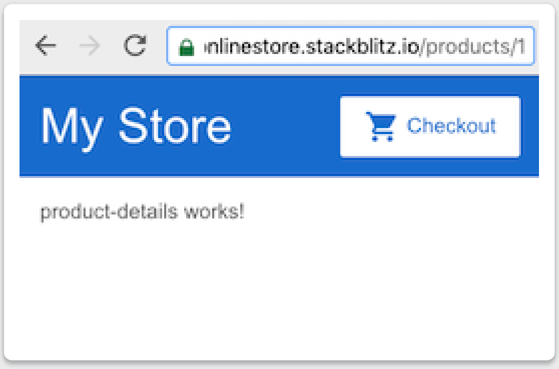
src/app/product-list/product-list.component.html

```
<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, or URL, contains one fixed segment, `/products`, while 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!"
Notice that the URL in the preview window changes. The final segment is `products/#` where `#` is the number of the route you clicked.



product-details works!

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. This section shows you how to 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 by adding it as an argument within the constructor's parentheses.

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 that the Angular Router loads. It contains information about the route, its parameters, and additional data associated with the route.
By injecting the `ActivatedRoute`, you are configuring the component to use a *service*. The Managing Data page covers services in more detail.

1. In the `ngOnInit()` method, subscribe to route parameters and fetch the product based on the `productId`.

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

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

The route parameters correspond to the path variables you define in the route. The URL that matches the route provides the `productId`. Angular uses the `productId` to display the details for each unique product.

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

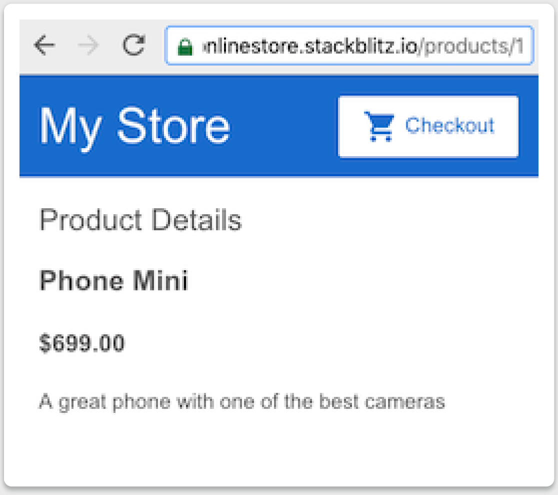
src/app/product-details/product-details.component.html

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

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

</div>
```

Now, when users click on a name in the product list, the router navigates them to the distinct URL for the product, swaps out the product list component for the product details component, and displays the product details.



Product Details

Phone Mini

$699.00

A great phone with one of the best cameras

For more information about the Angular Router, see Routing & Navigation.

Next steps

Congratulations! You have integrated routing into your online store.

- Products are linked from the product list view 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 a shopping cart feature, use a service to manage the cart data and use 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.