Search By Category

Release 2.0 - Plan

From: The Boss

© luv2code LLC

Release 2.0 - Plan

From: The Boss

- Online Shop Template Integration
- Search for products by category
- Search for products by text box
- Master / detail view of products
- Pagination support for products
- Add products to shopping cart (CRUD)
- Shopping cart check out

© luv2code LLC

Release 2.0 - Plan

From: The Boss

- Online Shop Template Integration
- Search for products by category
- Search for products by text box
- Master / detail view of products
- Pagination support for products
- Add products to shopping cart (CRUD)
- Shopping cart check out

Current task

© luv2code LLC

Search for Products by Category

© luv2code LLC

Search for Products by Category

© luv2code LLC

DEMO

© luv2code LLC

Angular Routing

© luv2code LLC

Angular Routing

- In Angular, you can add links in your application

© luv2code LLC

Angular Routing

- In Angular, you can add links in your application
- The links will route to other components in your application

© luv2code LLC

Angular Routing

- In Angular, you can add links in your application
- The links will route to other components in your application
- Angular routing will handle updating a view of your application

© luv2code LLC

Angular Routing

- In Angular, you can add links in your application
- The links will route to other components in your application
- Angular routing will handle updating a view of your application

© luv2code LLC

Angular Routing

- In Angular, you can add links in your application
- The links will route to other components in your application
- Angular routing will handle updating a view of your application

Only updates a section of your page

Doesn't reload entire page

© luv2code LLC

Key Components

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.
RouterOutlet	Acts as a placeholder. Renders the desired component based on route.

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.
RouterOutlet	Acts as a placeholder. Renders the desired component based on route.
RouterLink	Link to specific routes in your application.

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.
RouterOutlet	Acts as a placeholder. Renders the desired component based on route.
RouterLink	Link to specific routes in your application.
ActivatedRoute	The current active route that loaded the component. Useful for accessing route parameters.

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.
RouterOutlet	Acts as a placeholder. Renders the desired component based on route.
RouterLink	Link to specific routes in your application.
ActivatedRoute	The current active route that loaded the component. Useful for accessing route parameters.

We will see code examples in upcoming slides

© luv2code LLC

Key Components

Name	Description
Router	Main routing service. Enables navigation between views based on user actions.
Route	Maps a URL path to a component.
RouterOutlet	Acts as a placeholder. Renders the desired component based on route.
RouterLink	Link to specific routes in your application.
ActivatedRoute	The current active route that loaded the component. Useful for accessing route parameters.
more ….	

We will see code examples in upcoming slides

http://angular.io/guide/router

© luv2code LLC

Development Process

Step-By-Step

© luv2code LLC

Development Process

Step-By-Step

1. Define routes

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param
5. Enhance ProductListComponent to read category id param

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param
5. Enhance ProductListComponent to read category id param
6. Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param
5. Enhance ProductListComponent to read category id param
6. Modify Spring Boot app - REST Repository needs new method
7. Update Angular Service to call new URL on Spring Boot app

© luv2code LLC

Step 1: Define Routes

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path
  - Angular will create a new instance of component

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path
  - Angular will create a new instance of component

```
const routes: Routes = [

  {path: 'products', component: ProductListComponent}

];
```


© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path
  - Angular will create a new instance of component

```
const routes: Routes = [

  {path: 'products', component: ProductListComponent}

];
```

Path to match

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path
  - Angular will create a new instance of component

```
const routes: Routes = [

  {path: 'products', component: ProductListComponent}

];
```

When path matches,
create new instance of component

Path to match

© luv2code LLC

Step 1: Define Routes

- A route has a **path** and a reference to a **component**
- When the user selects the link for the route path
  - Angular will create a new instance of component

```
const routes: Routes = [

  {path: 'products', component: ProductListComponent}

];
```

When path matches,
create new instance of component

Path to match

Note: The path has no leading slashes!

© luv2code LLC

Step 1: Define Routes

© luv2code LLC

Step 1: Define Routes

- Add route to show products for a given category **id**

© luv2code LLC

Step 1: Define Routes

- Add route to show products for a given category **id**

```
const routes: Routes = [

  {path: 'category/:id', component: ProductListComponent},
  {path: 'products', component: ProductListComponent}

];
```


© luv2code LLC

Step 1: Define Routes

- Add route to show products for a given category **id**

```
const routes: Routes = [

  {path: 'category/:id', component: ProductListComponent},
  {path: 'products', component: ProductListComponent}

];
```

Category `id` parameter

The component can read this later and show products for this category

© luv2code LLC

Step 1: Define Routes

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```


© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

No category id, internally the component will use default category id

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

No path given, redirect to: /products

This is an exception to the rule about "no leading slashes"

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListCompon
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

`full` means match on this exactly.

Default option is `prefix` ...
match if path starts with a given value

No path given, redirect to: /products

This is an exception to the rule about "no leading slashes"

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

This is the generic wildcard. It will match on anything that didn't match above routes.

© luv2code LLC

Step 1: Define Routes

- Add more routes to handle for other cases ...

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

Order of routes is important.
First match wins.
Start from most specific to generic.

This is the generic wildcard. It will match on anything that didn't match above routes.

© luv2code LLC

Step 1: Define Routes

© luv2code LLC

Step 1: Define Routes

- Can add a custom PageNotFoundComponent ... for 404s

© luv2code LLC

Step 1: Define Routes

- Can add a custom PageNotFoundComponent ... for 404s

```
const routes: Routes = [
 ...
 {path: '**', component: PageNotFoundComponent}
];
```


© luv2code LLC

Step 1: Define Routes

- Can add a custom PageNotFoundComponent ... for 404s

```
const routes: Routes = [
 ...
 {path: '**', component: PageNotFoundComponent}
];
```

Custom component that you create

Can give any name and provide your own custom view

© luv2code LLC

Step 1: Define Routes

- Can add a custom PageNotFoundComponent ... for 404s

```
const routes: Routes = [
 ...
 {path: '**', component: PageNotFoundComponent}
];
```

Be sure to place this last.

Order of routes is important.
First match wins.
Start from most specific to generic.

Custom component that you create

Can give any name and
provide your own custom view

© luv2code LLC

Step 1: Define Routes

- Can add a custom PageNotFoundComponent ... for 404s

```
const routes: Routes = [
 ...
 {path: '**', component: PageNotFoundComponent}
];
```

Be sure to place this last.

Order of routes is important.
First match wins.
Start from most specific to generic.

Custom component that you create

Can give any name and
provide your own custom view

https://angular.io/guide/router#define-a-wildcard-route

© luv2code LLC

Step 2: Configure Router based on our routes

© luv2code LLC

Step 2: Configure Router based on our routes

- Configure the router in the application module

© luv2code LLC

Step 2: Configure Router based on our routes

- Configure the router in the application module

File: app.module.ts

```
const routes: Routes = [ ... ];


@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent
  ],
  imports: [
    RouterModule.forRoot(routes),
    BrowserModule,
    HttpClientModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```


© luv2code LLC

Step 2: Configure Router based on our routes

- Configure the router in the application module

File: app.module.ts

```
const routes: Routes = [ ... ];


@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent
  ],
  imports: [
    RouterModule.forRoot(routes),
    BrowserModule,
    HttpClientModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```


© luv2code LLC

Step 2: Configure Router based on our routes

- Configure the router in the application module

File: app.module.ts

```
const routes: Routes = [ ... ];


@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent
  ],
  imports: [
    RouterModule.forRoot(routes),
    BrowserModule,
    HttpClientModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```


© luv2code LLC

Step 3: Define the Router Outlet

© luv2code LLC

Step 3: Define the Router Outlet

- **Router Outlet** acts as a placeholder.

© luv2code LLC

Step 3: Define the Router Outlet

- **Router Outlet** acts as a placeholder.
- Renders the desired component based on route.

© luv2code LLC

Step 3: Define the Router Outlet

- **Router Outlet** acts as a placeholder.
- Renders the desired component based on route.

© luv2code LLC

Step 3: Define the Router Outlet

- **Router Outlet** acts as a placeholder.
- Renders the desired component based on route.

© luv2code LLC

Step 3: Define the Router Outlet

- **Router Outlet** acts as a placeholder.
- Renders the desired component based on route.

© luv2code LLC

Step 3: Define the Router Outlet

© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

File: app.component.html

```
...

    <!-- MAIN CONTENT -->
    <router-outlet></router-outlet>

    <!-- <app-product-list></app-product-list> -->

...
```


© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

File: app.component.html

```
...

    <!-- MAIN CONTENT -->
    <router-outlet></router-outlet>

    <!-- <app-product-list></app-product-list> -->

...
```

Based on Router configuration
Display appropriate component here

© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

File: app.component.html

```
...

    <!-- MAIN CONTENT -->
    <router-outlet></router-outlet>

...
```

Based on Router configuration
Display appropriate component here

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```


© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

File: app.component.html

```
...

    <!-- MAIN CONTENT -->
    <router-outlet></router-outlet>

...
```

Based on Router configuration
Display appropriate component here

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```

Create an instance of ProductListComponent
Display products based on category id

© luv2code LLC

Step 3: Define the Router Outlet

- Update app.component.html to use Router Outlet

File: app.component.html

```
...

    <!-- MAIN CONTENT -->
    <router-outlet></router-outlet>

    <!-- <app-product-list></app-product-list> -->

...
```

Based on Router configuration
Display appropriate component here

Comment out or delete the reference to
<app-product-list></app-product-list>

© luv2code LLC

Step 4: Set up Router Links to pass category id param

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

Based on Routes configuration, use ProductListComponent

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

Based on Routes configuration, use ProductListComponent

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkAct
   </li>

   <li>
      <a routerLink="/category/2" routerLinkAct
   </li>
...
```

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

Based on Routes configuration, use ProductListComponent

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkAct
   </li>

   <li>
                                  erLinkAct
```

The param name is `id`
Pass the value of "1" for this parameter

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent },
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

Once user clicks the link,
then we can apply a custom CSS style

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

Once user clicks the link,
then we can apply a custom CSS style

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```

File: styles.css

```
.active-link {
  font-weight: bold;
}
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

At the moment we are hard-coding the categories.

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```


© luv2code LLC

Step 4: Set up Router Links to pass category id param

- In our HTML page, set up links to our route
- Pass category id as a parameter

At the moment we are hard-coding the categories.

In upcoming videos, we will make this dynamic and read category information from REST API

File: app.component.html

```
...
   <!-- MENU SIDEBAR -->
   <li>
      <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
   </li>

   <li>
      <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
   </li>
...
```

id	category_name
1	Books
2	Coffee Mugs
3	Mouse Pads
4	Luggage Tags

© luv2code LLC

Recall: Router Outlet

© luv2code LLC

Recall: Router Outlet

- When user clicks the link
  - ProductListComponent will appear in the location of `router-outlet`

© luv2code LLC

Recall: Router Outlet

- When user clicks the link
  - ProductListComponent will appear in the location of `router-outlet`

© luv2code LLC

Recall: Router Outlet

- When user clicks the link
  - ProductListComponent will appear in the location of `router-outlet`

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```


© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

Use the activated route

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

State of route at this given moment in time

Use the activated route

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

State of route at this given moment in time

Use the activated route

Map of all the route parameters

© luv2code LLC

Step 5: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

State of route at this given moment in time

Read the `id` parameter

Use the activated route

Map of all the route parameters

© luv2code LLC

Step 5: Enhance Produc

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent}
```

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

State of route at this given moment in time

Read the id parameter

Use the activated route

Map of all the route parameters

© luv2code LLC

Step 5: Enhance Produc

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent}
```

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

State of route at this given moment in time

Read the id parameter

Use the activated route

Map of all the route parameters

© luv2code LLC

Step 5: Enhance Produc

```
const routes: Routes = [
  {path: 'category/:id', component: ProductListComponent},
  {path: 'category', component: ProductListComponent}
```

- Need to read the category id parameter

```
currentCategoryId: number;
...
...
...
this.currentCategoryId = +this.route.snapshot.paramMap.get('id');
```

Read the `id` parameter

State of route at this given moment in time

Use the activated route

Map of all the route parameters

Parameter value is returned as `string`

Use the "+" symbol to convert to `number`

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Currently, the Spring Boot app, returns products regardless of category

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Currently, the Spring Boot app, returns products regardless of category
- Need to modify to only return products for a given category id

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions

Magic!

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions
- Methods starting with: `findBy`, `readBy`, `queryBy`, etc ...

Magic!

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions
- Methods starting with: `findBy`, `readBy`, `queryBy`, etc ...

Magic!

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions
- Methods starting with: `findBy`, `readBy`, `queryBy`, etc ...

Magic!

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

Match by category id

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

Match by category id

Use this parameter value

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

Match by category id

Use this parameter value

Behind the scenes, Spring will execute a query similar to this

```
SELECT * FROM product where category_id=?;
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

A query method

Match by category id

Use this parameter value

Behind the scenes, Spring will execute a query similar to this

```
SELECT * FROM product where category_id=?;
```

Magic!

© luv2code LLC

More on Query Methods

© luv2code LLC

More on Query Methods

- You can provide your own custom query using `@Query` annotation

© luv2code LLC

More on Query Methods

- You can provide your own custom query using `@Query` annotation
- Support is available for conditionals: and, or, like, sort etc

© luv2code LLC

More on Query Methods

- You can provide your own custom query using `@Query` annotation
- Support is available for conditionals: and, or, like, sort etc
- For details on this topic

© luv2code LLC

More on Query Methods

- You can provide your own custom query using `@Query` annotation
- Support is available for conditionals: and, or, like, sort etc
- For details on this topic
  - See the Query Method section in the **Spring Data Reference Manual**

© luv2code LLC

More on Query Methods

- You can provide your own custom query using `@Query` annotation
- Support is available for conditionals: and, or, like, sort etc
- For details on this topic
  - See the Query Method section in the **Spring Data Reference Manual**

www.luv2code.com/spring-data-query-methods

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- `Page` and `Pageable` provides support for pagination

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- `Page` and `Pageable` provides support for pagination

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- `Page` and `Pageable` provides support for pagination

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

`Page` is a sublist of a list of objects

Has information such as
totalElements, totalPages, curentPosition etc

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- `Page` and `Pageable` provides support for pagination

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

`Page` is a sublist of a list of objects

Has information such as
totalElements, totalPages, curentPosition etc

`Pageable` represents pagination information

Has information such as
pageNumber, pageSize, previous, next etc

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST automatically expose endpoints for query methods

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST automatically expose endpoints for query methods
- Exposes endpoint: `/search/<<queryMethodName>>`

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST automatically expose endpoints for query methods
- Exposes endpoint: `/search/<<queryMethodName>>`

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST automatically expose endpoints for query methods
- Exposes endpoint: `/search/<<queryMethodName>>`

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

http://localhost:8080/api/products/search/findByCategoryId

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

- Spring Data REST automatically expose endpoints for query methods
- Exposes endpoint: `/search/<<queryMethodName>>`

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

http://localhost:8080/api/products/search/findByCategoryId

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```


© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

http://localhost:8080/api/products/search/findByCategoryId?id=1

© luv2code LLC

Step 6: Modify Spring Boot app - REST Repository needs new method

File: ProductRepository.java

```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

}
```

http://localhost:8080/api/products/search/findByCategoryId?id=1

To pass data to REST API

© luv2code LLC

Step 7: Update Angular Service to call new URL on Spring Boot app

© luv2code LLC

Step 7: Update Angular Service to call new URL on Spring Boot app

File: product.service.ts

```
export class ProductService {

  private baseUrl = 'http://localhost:8080/api/products';

  constructor(private httpClient: HttpClient) { }

  getProductList(theCategoryId: number): Observable<Product[]> {

    const url = `${this.baseUrl}/search/findByCategoryId?id=${theCategoryId}`;

    return this.httpClient.get<GetResponse>(url).pipe(
      map(response => response._embedded.products)
    );
  }
}
```


© luv2code LLC

Step 7: Update Angular Service to call new URL on Spring Boot app

File: product.service.ts

```
export class ProductService {

  private baseUrl = 'http://localhost:8080/api/products';

  constructor(private httpClient: HttpClient) { }

  getProductList(theCategoryId: number): Observable<Product[]> {

    const url = `${this.baseUrl}/search/findByCategoryId?id=${theCategoryId}`;

    return this.httpClient.get<GetResponse>(url).pipe(
      map(response => response._embedded.products)
    );
  }
}
```


© luv2code LLC

Step 7: Update Angular Service to call new URL on Spring Boot app

File: product.service.ts

```
export class ProductService {

  private baseUrl = 'http://localhost:8080/api/products';

  constructor(private httpClient: HttpClient) { }

  getProductList(theCategoryId: number): Observable<Product[]> {

    const url = `${this.baseUrl}/search/findByCategoryId?id=${theCategoryId}`;

    return this.httpClient.get<GetResponse>(url).pipe(
      map(response => response._embedded.products)
    );
  }
}
```

http://localhost:8080/api/products/search/findByCategoryId?id=1

© luv2code LLC

Development Process

Step-By-Step

1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param
5. Enhance ProductListComponent to read category id param
6. Modify Spring Boot app - REST Repository needs new method
7. Update Angular Service to call new URL on Spring Boot app

© luv2code LLC