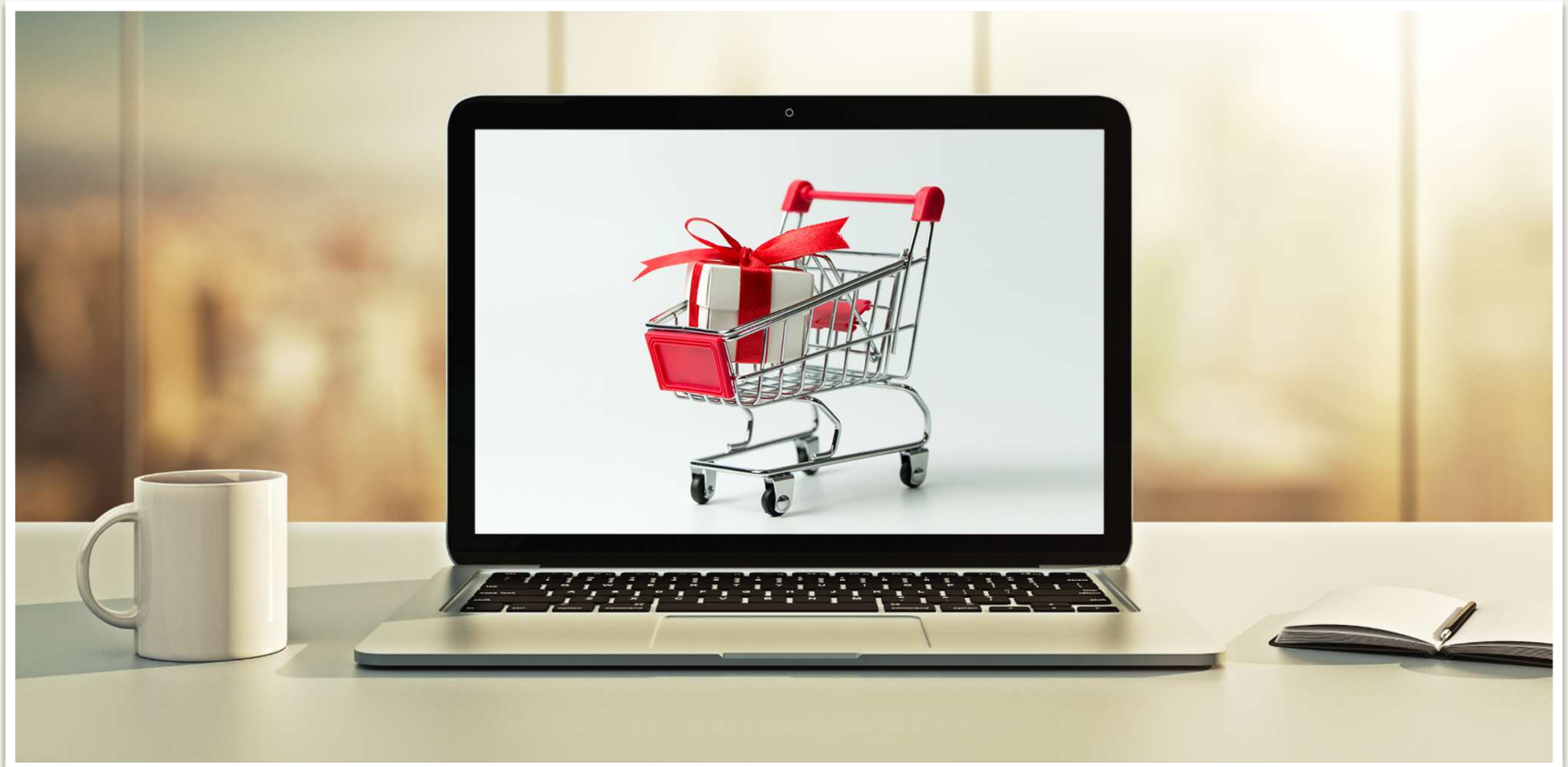
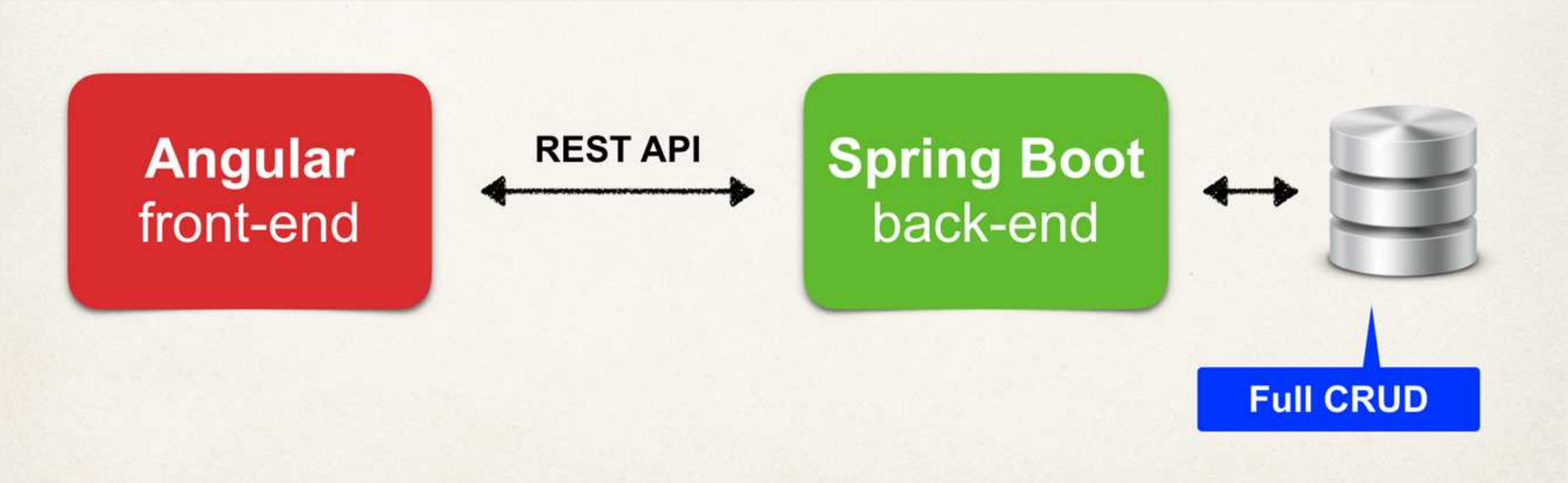
Full Stack Project Overview



Project

- Build Real-time eCommerce App

Full Stack



© luv2code LLC

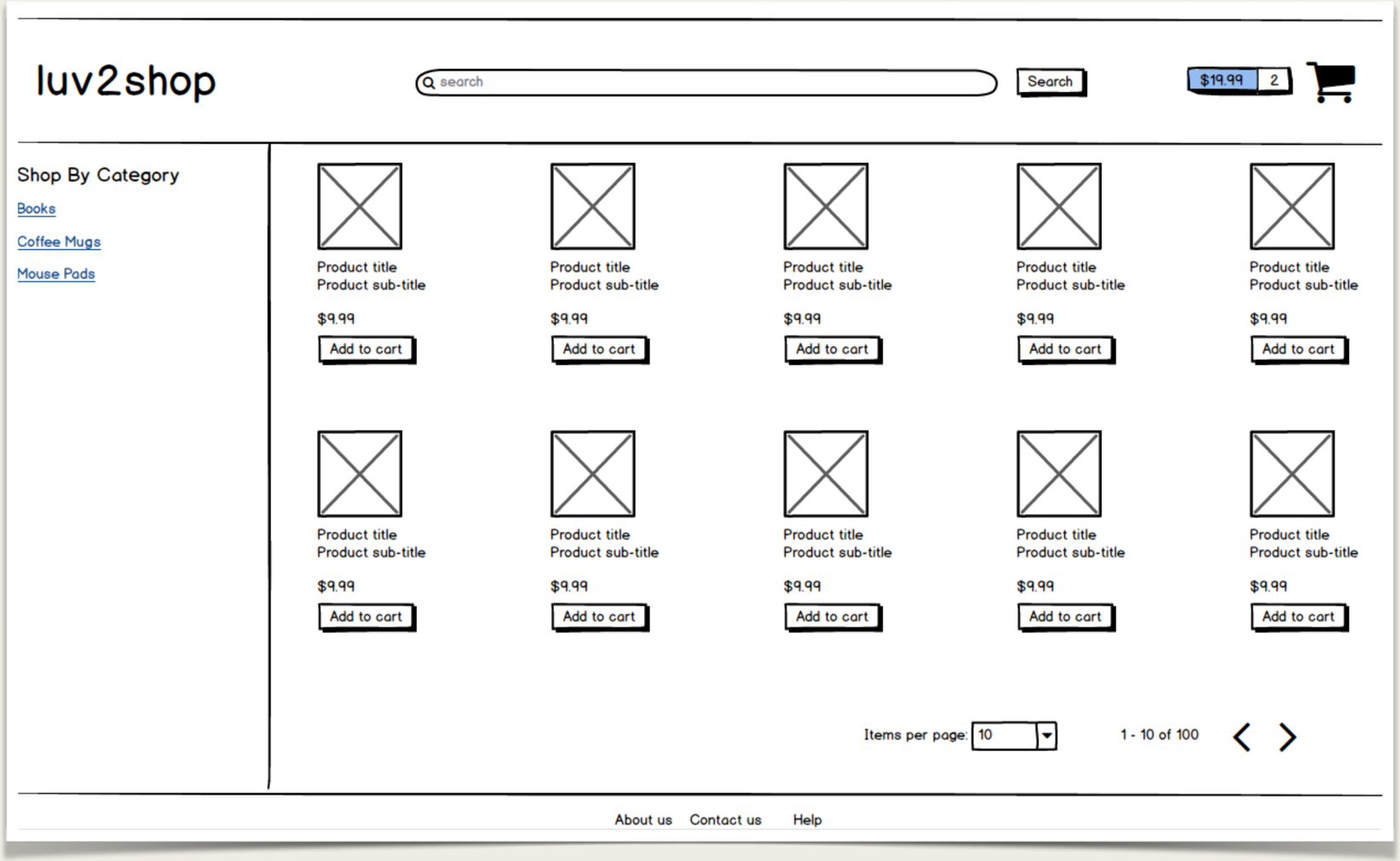
Requirements

From: The Boss

- Show a list of products
- Add products to shopping cart (CRUD)
- Shopping cart check out
- User login/logout security
- Track previous orders for logged in users

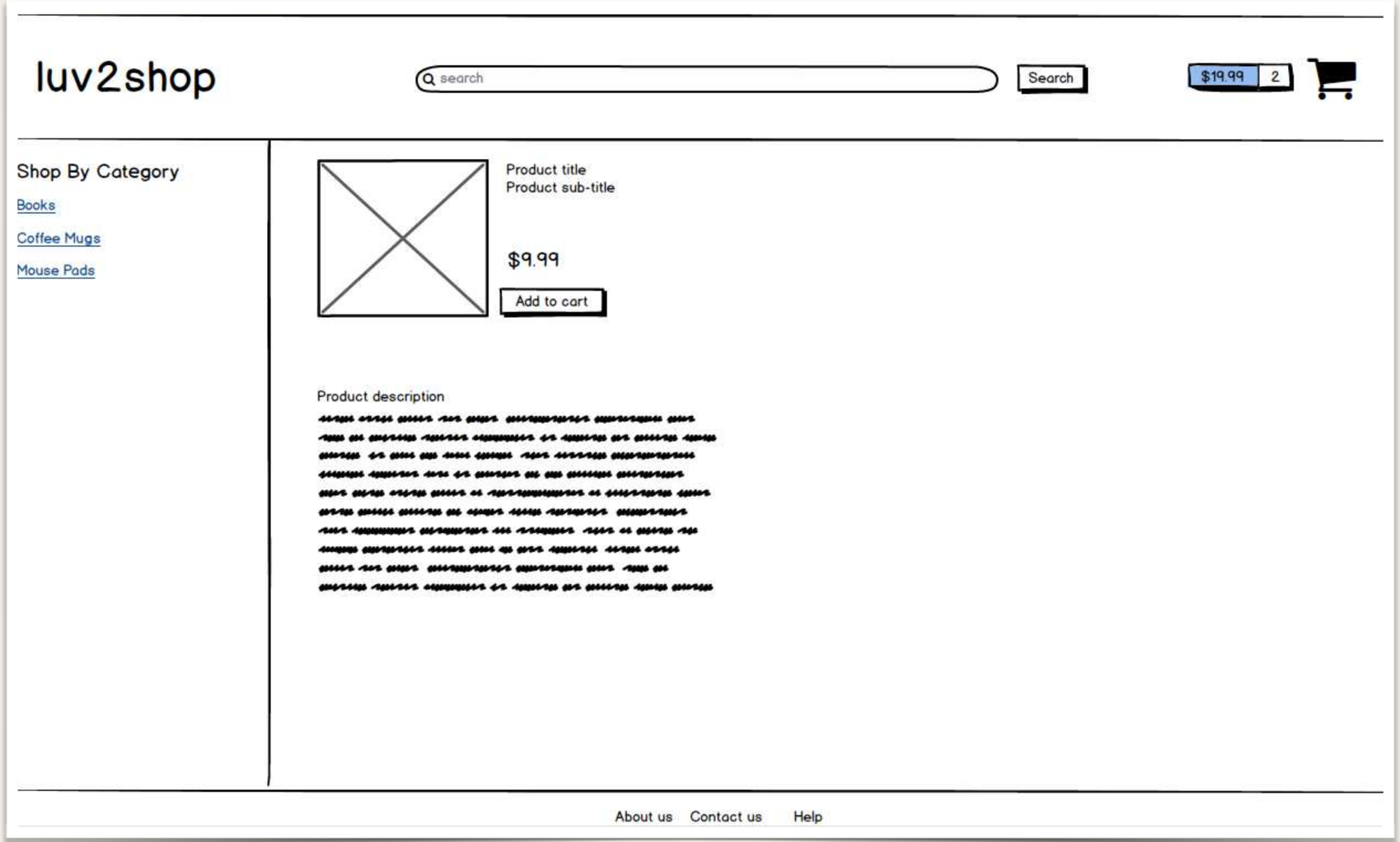
© luv2code LLC

Wireframes - Home Page



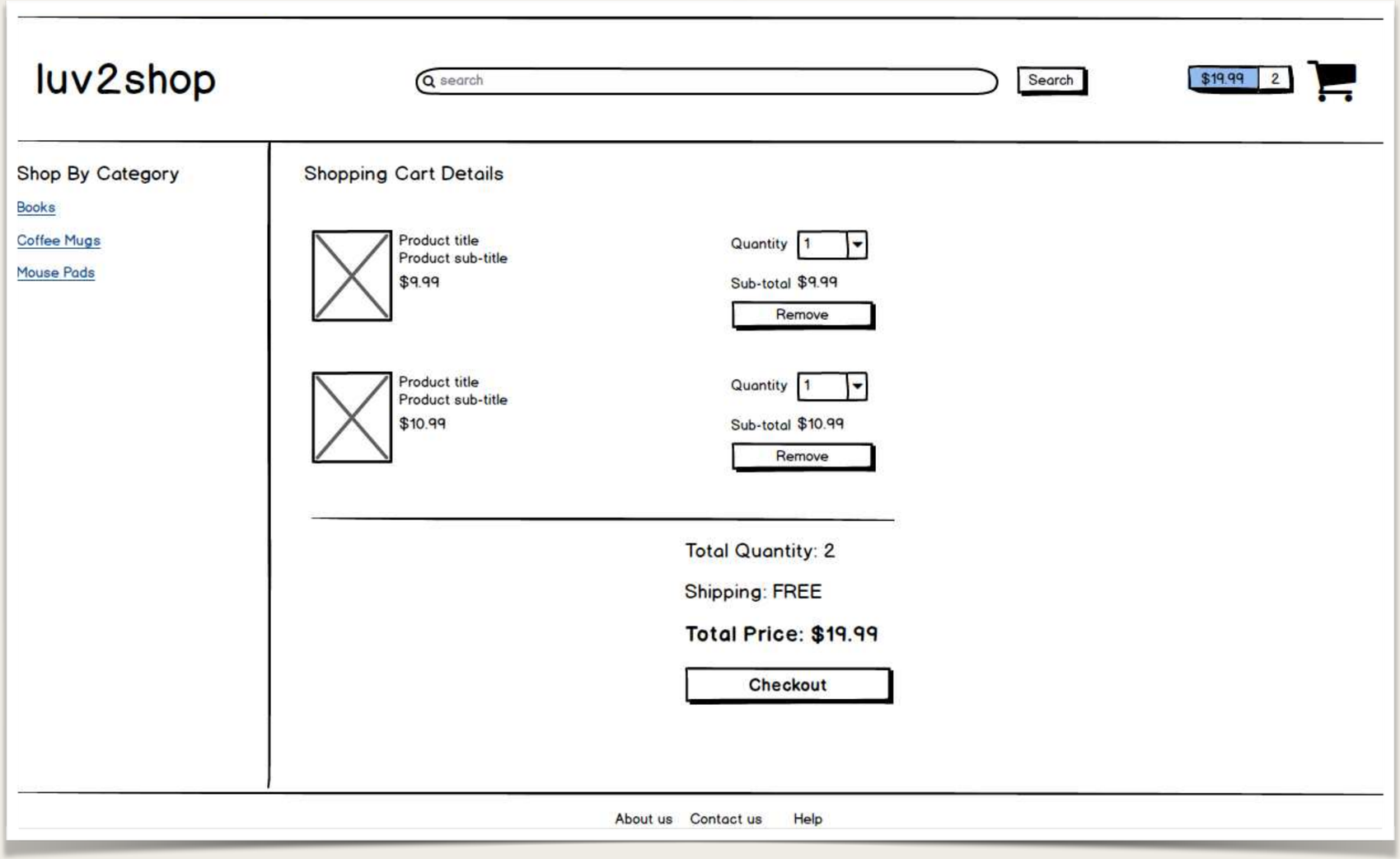
© luv2code LLC

Wireframes - Product Details



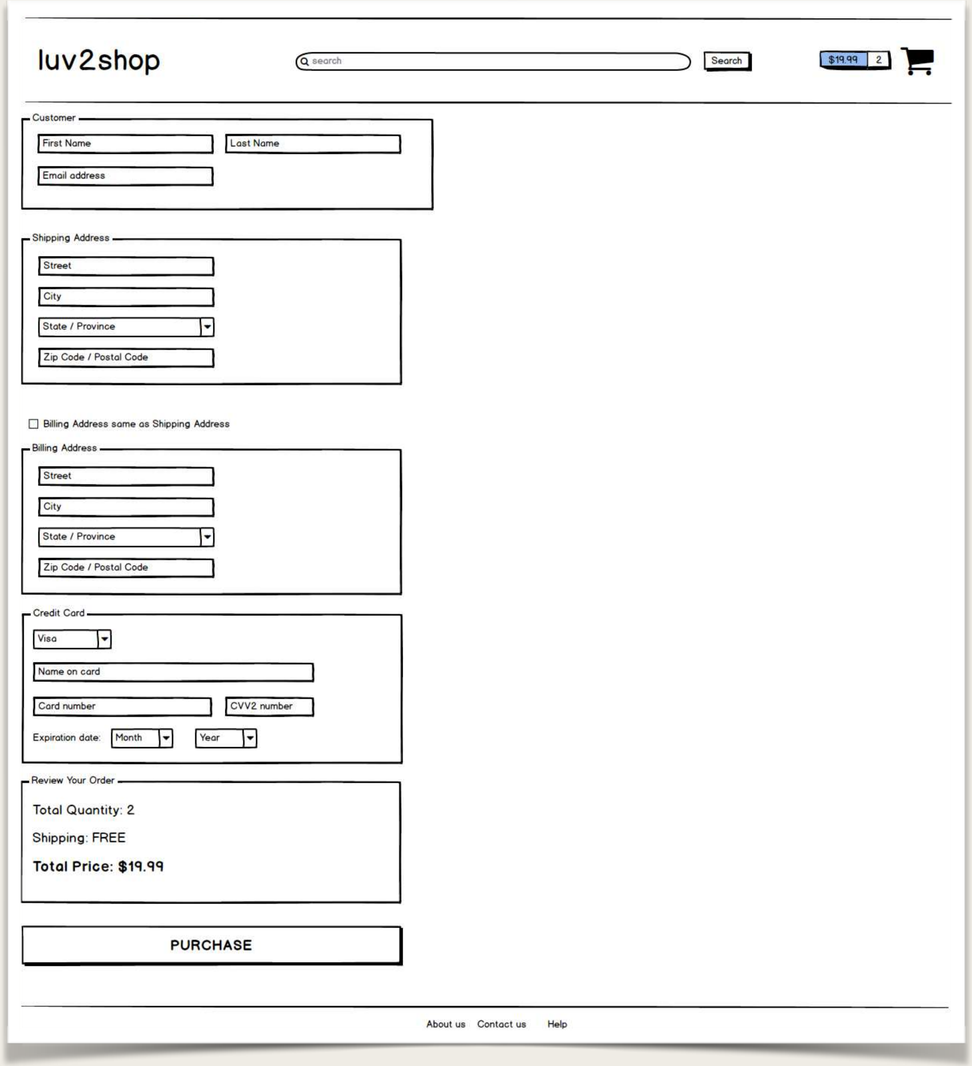
© luv2code LLC

Wireframes - Shopping Cart Details



© luv2code LLC

Wireframes - Check Out



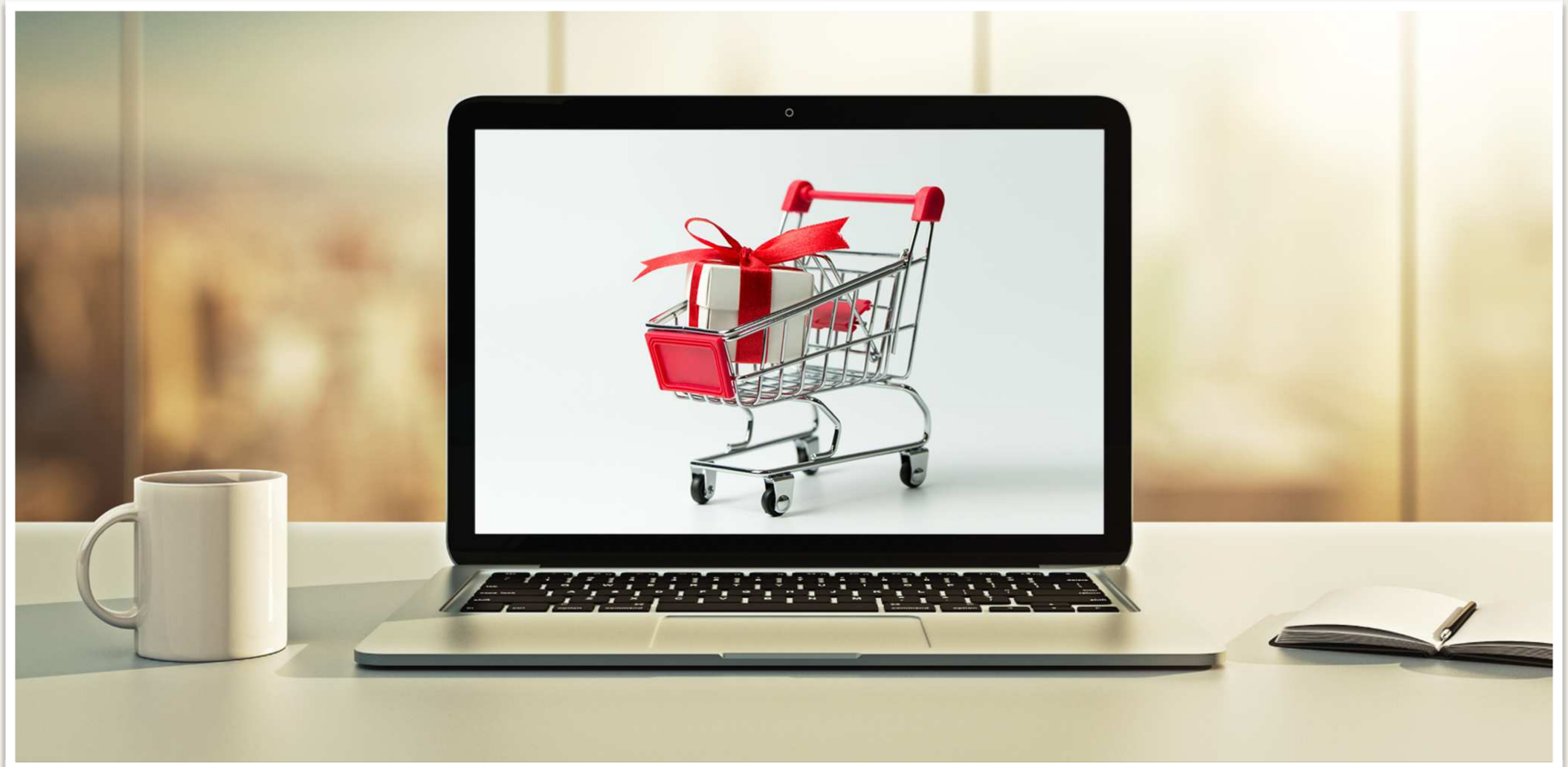
© luv2code LLC

Release Plan

- Release 1.0
  - Show a list of products
- Release 2.0
  - Add products to shopping cart (CRUD)
  - Shopping cart check out
- Release 3.0
  - User login/logout security
  - Track previous orders for logged in users

© luv2code LLC

Spring Boot Back End



Java Development Environment

- We assume that you are already an experienced Spring Boot Developer

- You should have the following items already installed
  - Java Development Kit (JDK)
  - Java IDE (we'll use IntelliJ in the videos, but any Java IDE will work)
  - Maven
  - MySQL Database and MySQL Workbench

© luv2code LLC

About IntelliJ

Super Amazing IDE!!!

- In this course, we will use the free version of IntelliJ
  - Known as **IntelliJ Community Edition**
  - Download from: **https://www.jetbrains.com/idea/download**
  - Select **Community Edition**

- You can also use the Ultimate Edition ($) ... free trial version is available

© luv2code LLC

Additional Java IDEs

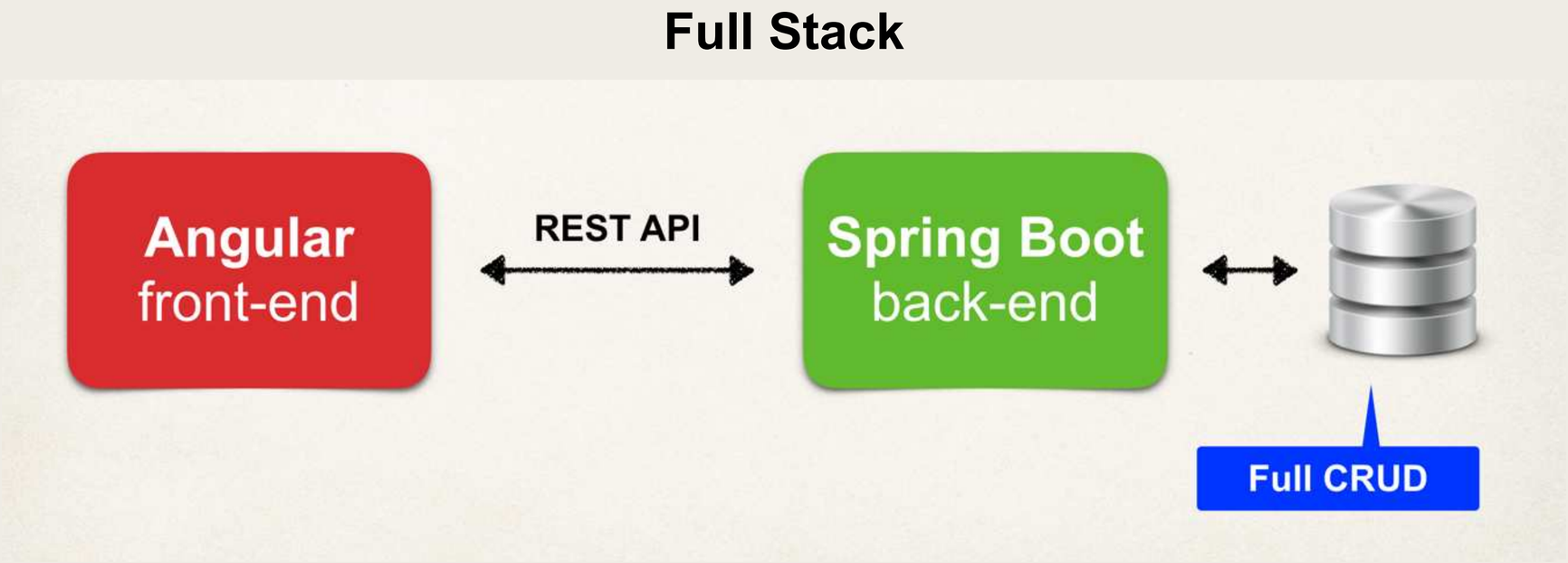
- You are free to use other Java IDEs such as Eclipse, VS Code, NetBeans
- All you need is a Java IDE that supports Maven ... that's it!
- You can easily follow along with any Java IDE

- We will provide tech support for IntelliJ, Eclipse, VS Code, NetBeans

© luv2code LLC

Spring Boot Back End

- Leverage Spring Data REST for REST API
- Minimizes the coding for Spring Boot back end



© luv2code LLC

Create Repository

- Spring Data REST will scan your project for **JpaRepository**
- Expose REST APIs for each entity type for your **JpaRepository**

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

}
```


© luv2code LLC

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity

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

}
```

/products

© luv2code LLC

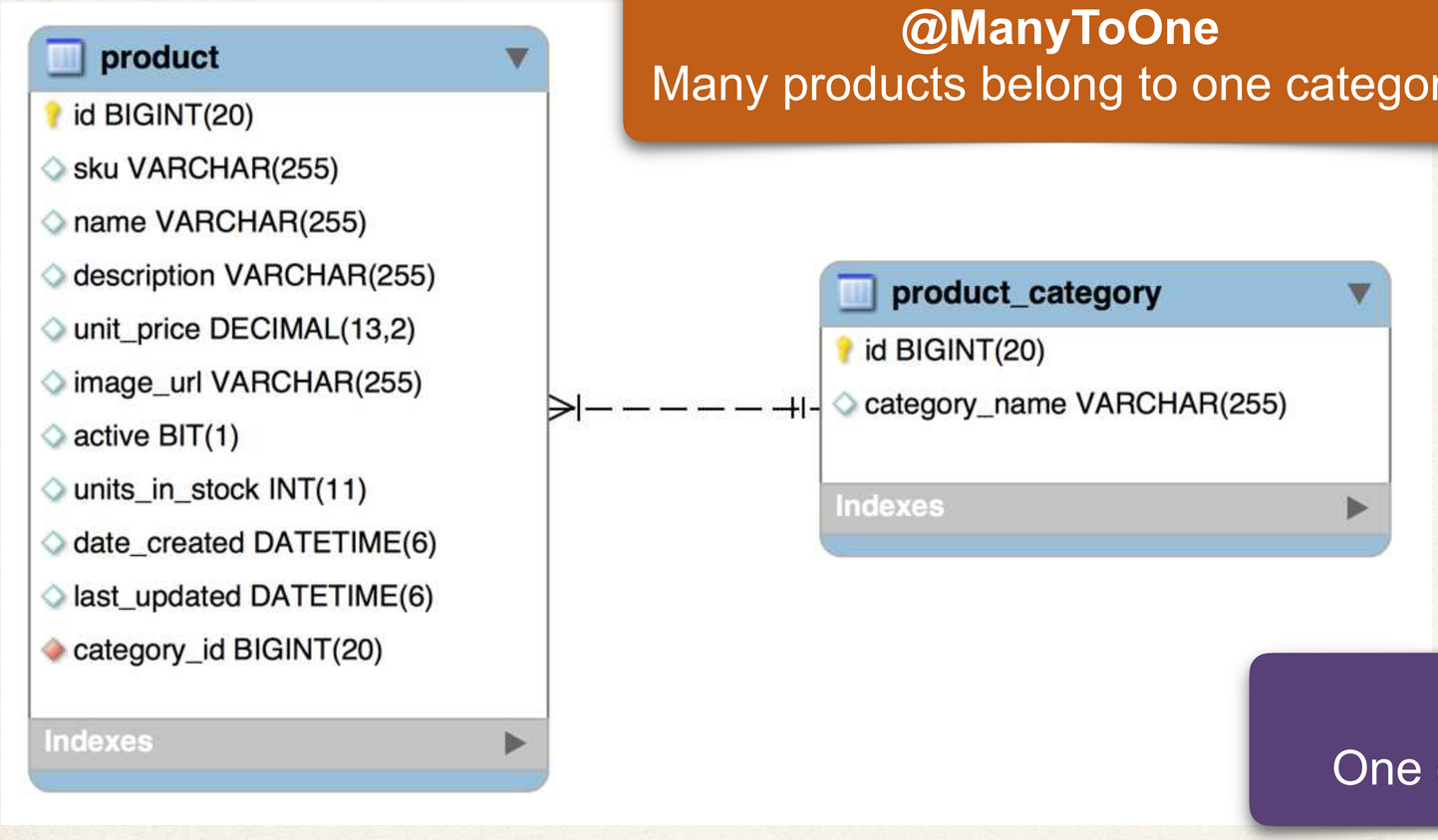
REST API

- Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
POST	/products	**C**reate a new product
GET	/products	**R**ead a list of products
GET	/products/{id}	**R**ead a single product
PUT	/products/{id}	**U**pdate an existing product
DELETE	/products/{id}	**D**elete an existing product

© luv2code LLC

Database Schema - Release 1.0



@ManyToOne
Many products belong to one category

@OneToMany
One category has many products

© luv2code LLC

Two Database Scripts

- **01-create-user.sql**
- **02-create-products.sql**

© luv2code LLC

About: 01-create-user.sql

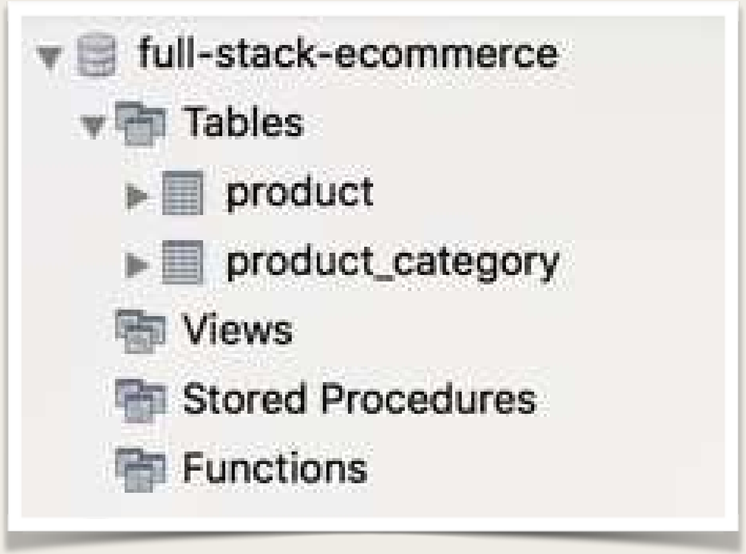
1. Create a new MySQL user for our application
   - user id: **ecommerceapp**
   - password: **ecommerceapp**

© luv2code LLC

About: 02-create-products.sql

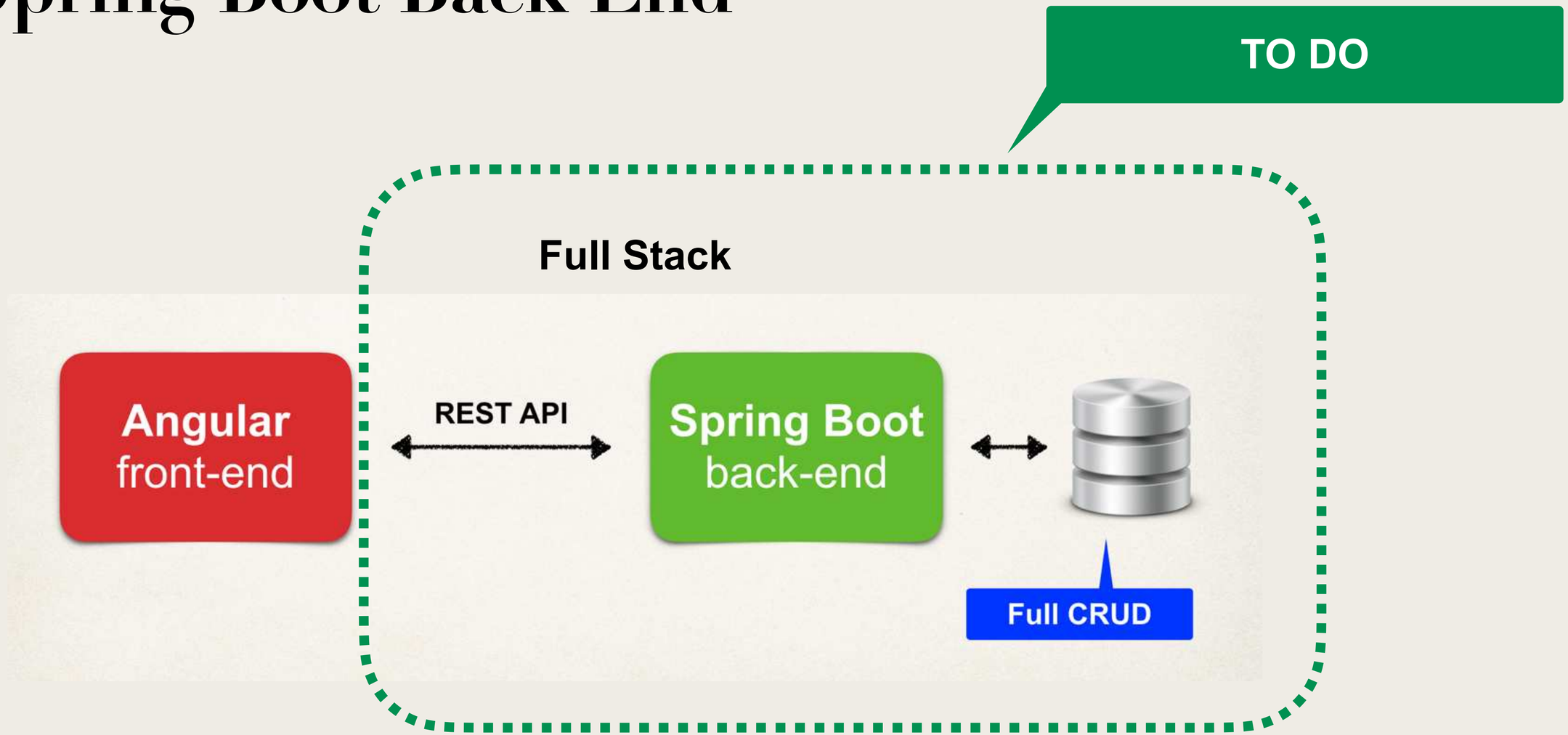
1. Create new database tables: **product, product_category**

2. Load tables with sample data



© luv2code LLC

Spring Boot Back End



© luv2code LLC

Development Process

Step-By-Step

1. Set up the database tables

2. Create a Spring Boot starter project (start.spring.io)

```
spring-boot-starter-data-jpa
spring-boot-starter-data-rest
mysql-connector-java
lombok
```

3. Develop the Entities: **Product** and **ProductCategory**

4. Create REST APIs with Spring Data JPA Repositories and Spring Data REST

© luv2code LLC

Project Lombok

- Modern Java project
- Lombok automagically generates the getters/setters (behind the scenes)
- No need for the developer to manually define getters/setters, etc ...
- Easy-to-use Annotations to eliminate boilerplate code

http://www.projectlombok.org

© luv2code LLC

Project Lombok

Before Lombok

```java
@Entity
@Table(name = "product")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name = "name")
    @NotBlank
    private String name;

    public Product() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
          return false;
        Product product = (Product) o;
        return Objects.equals(id, product.id) &&
                Objects.equals(name, product.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name);
    }
}
```

After Lombok

Lombok annotation

```java
@Entity
@Table(name = "product")
@Data
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name = "name")
    @NotBlank
    private String name;
}
```

That's It!!!
Absolutely no need to generate getters and setters

Lombok will do this work for you automagically behind the scenes

© luv2code LLC

REST API - Read Only

REST API

I don't want that!
I want the REST API as READ-ONLY

- Spring Data REST will expose ALL these endpoints for free!

HTTP Method		CRUD Action
POST	/products	**C**reate a new product
GET	/products	**R**ead a list of products
GET	/products/{id}	**R**ead a single product
PUT	/products/{id}	**U**pdate an existing product
DELETE	/products/{id}	**D**elete an existing product

© luv2code LLC

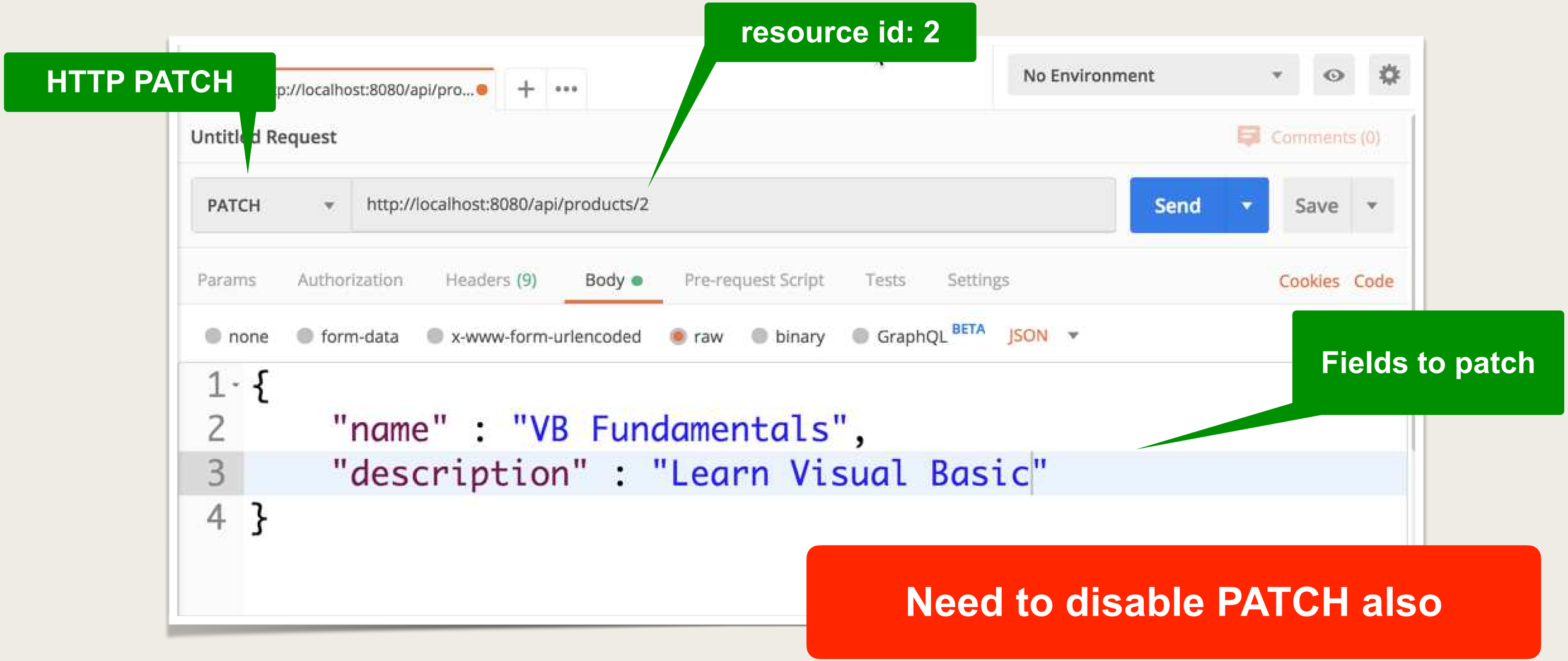
REST API: PATCH

- Spring Dat REST also exposes an endpoint to accept **PATCH** requests
- HTTP method: **PATCH** is used to perform partial updates on a resource
  - Provide the resource id in the URL path
  - List the fields to "patch"
  - Will update the "patch" fields, all other fields remain the same

Need to disable PATCH also

© luv2code LLC

HTTP PATCH Example



© luv2code LLC

Possible Solutions

1. Option 1: Don't use Spring Data REST
    1. Manually create our own @RestController
    2. Manually define methods for access: @GetMapping
    3. But we lose the Spring Data REST support for paging, sorting etc :-(

2. Option 2: Use Spring Data REST

Choose this one!

    1. Configure to disable certain HTTP methods: POST, DELETE etc...

We will cover REST API Security later in the course

© luv2code LLC

Spring Data REST Configuration

- Disable HTTP methods: POST, PUT, DELETE

HTTP methods to disable

```
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {

        HttpMethod[] theUnsupportedActions = {HttpMethod.PUT, HttpMethod.POST, HttpMethod.DELETE};

        // disable HTTP methods for Product:  PUT, POST and DELETE
        config.getExposureConfiguration()
                .forDomainType(Product.class)
                .withItemExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions))
                .withCollectionExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions));


    }
}
```


© luv2code LLC

Spring Data REST Configuration

- Disable HTTP methods: POST, PUT, DELETE

```
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

    @Override
    public void configureRep                                    estConfiguration config) {

        HttpMethod[] theUnsupporte Actions = {HttpMethod.PUT, HttpMethod.POST, HttpMethod.DELETE};

        // disable HTTP methods for Product:  PUT, POST and DELETE
        config.getExposureConfiguration()
                .forDomainType(Product.class)
                .withItemExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions))
                .withCollectionExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions));


    }
```

Apply to Repo for: Product

Single item

Collection

© luv2code LLC

Spring Data REST Configuration

- Disable HTTP methods: POST, PUT, DELETE

```
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {

        HttpMethod[] theUnsupportedActions = {HttpMethod.PUT, HttpMethod.POST, HttpMethod.DELETE};

        // disable HTTP methods for Product:  PUT, POST and DELETE
        config.getExposureConfiguration()
                .forDomainType(Product.class)
                .withItemExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions))
                .withCollectionExposure((metdata, httpMethods) -> httpMethods.disable(theUnsupportedActions));


    }
```

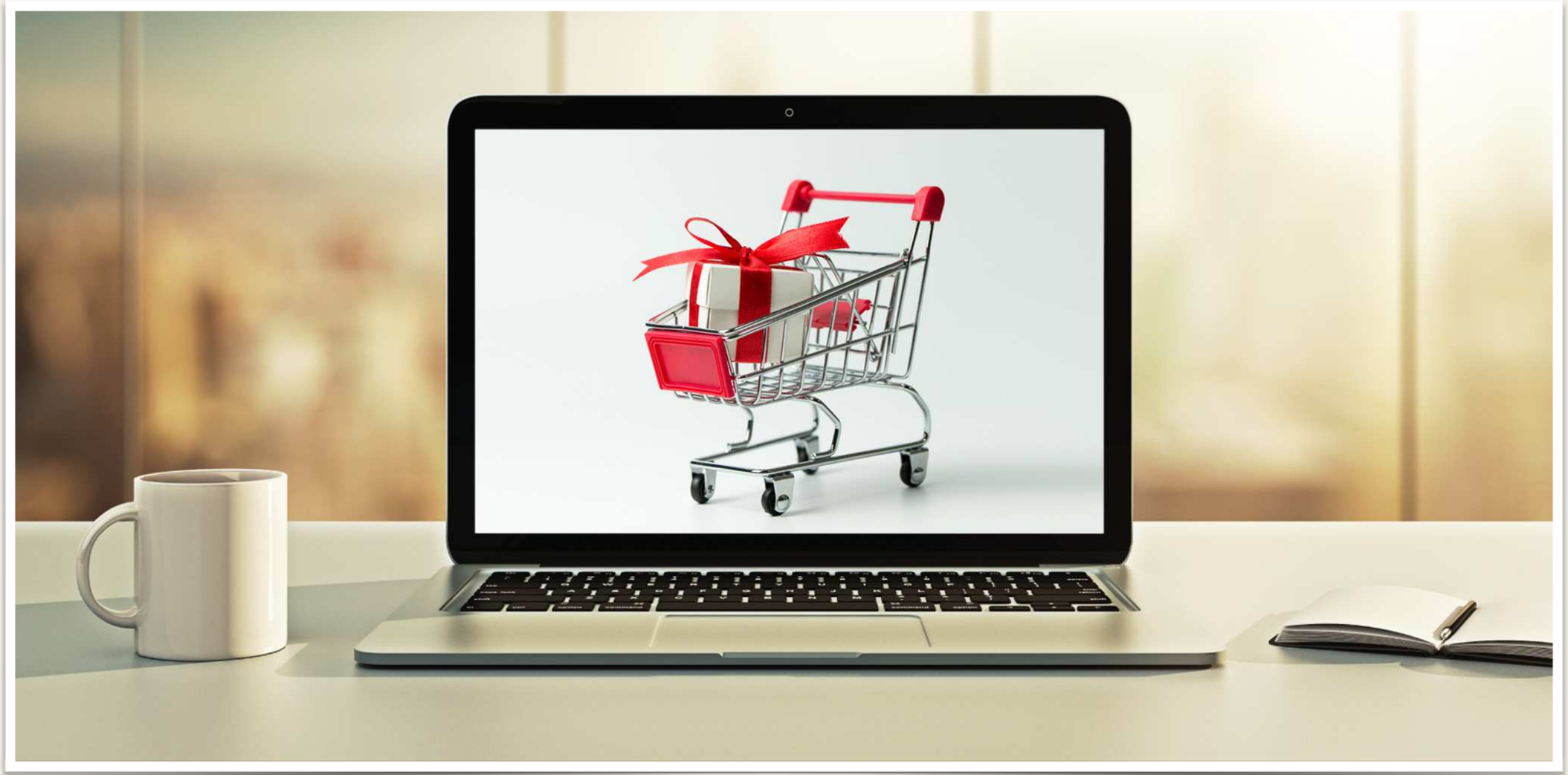
Single item

Collection

Disable

© luv2code LLC

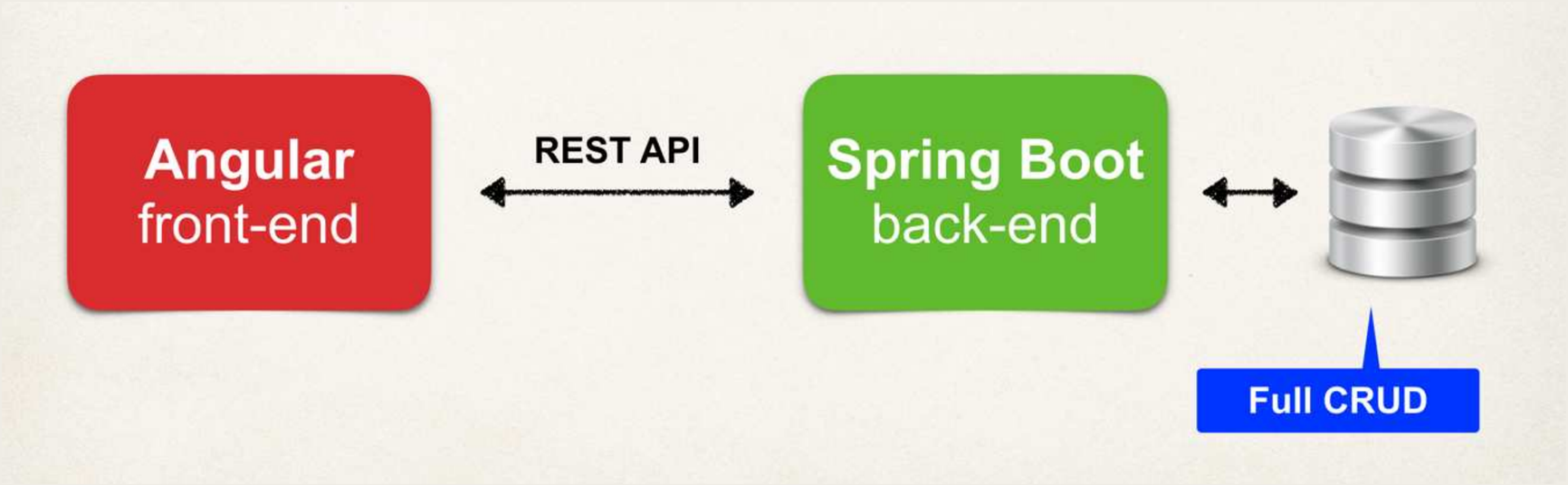
Angular Front End



Angular Front End

- Create Angular Front End components
- Retrieve data from Spring Boot REST APIs

Full Stack



© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project
2. Create Angular component for product-list
3. Develop TypeScript class for Product
4. Create Angular service to call REST APIs
5. Update Angular component to subscribe to data from Angular service
6. Display the data in an HTML page
7. Add CrossOrigin support to Spring Boot app

© luv2code LLC

Step 1: Create Angular project

- Create new project using Angular CLI

```
C:\> ng new angular-ecommerce
```


© luv2code LLC

Step 2: Create Angular component for product-list

- Create new component using Angular CLI

```
C:\> ng generate component components/product-list
```

Generated files placed in sub-directory: components/product-list

© luv2code LLC

Step 3: Develop TypeScript class for Product

- Create new class

Placed in sub-directory: common

```
C:\> ng generate class common/product
```

File: src/app/common/product.ts

```
export class Product {
    sku: string;
    name: string;
    description: string;
    unitPrice: number;
    imageUrl: string;
    active: boolean;
    unitsInStock: number;
    dateCreated: Date;
    lastUpdate: Date;
}
```

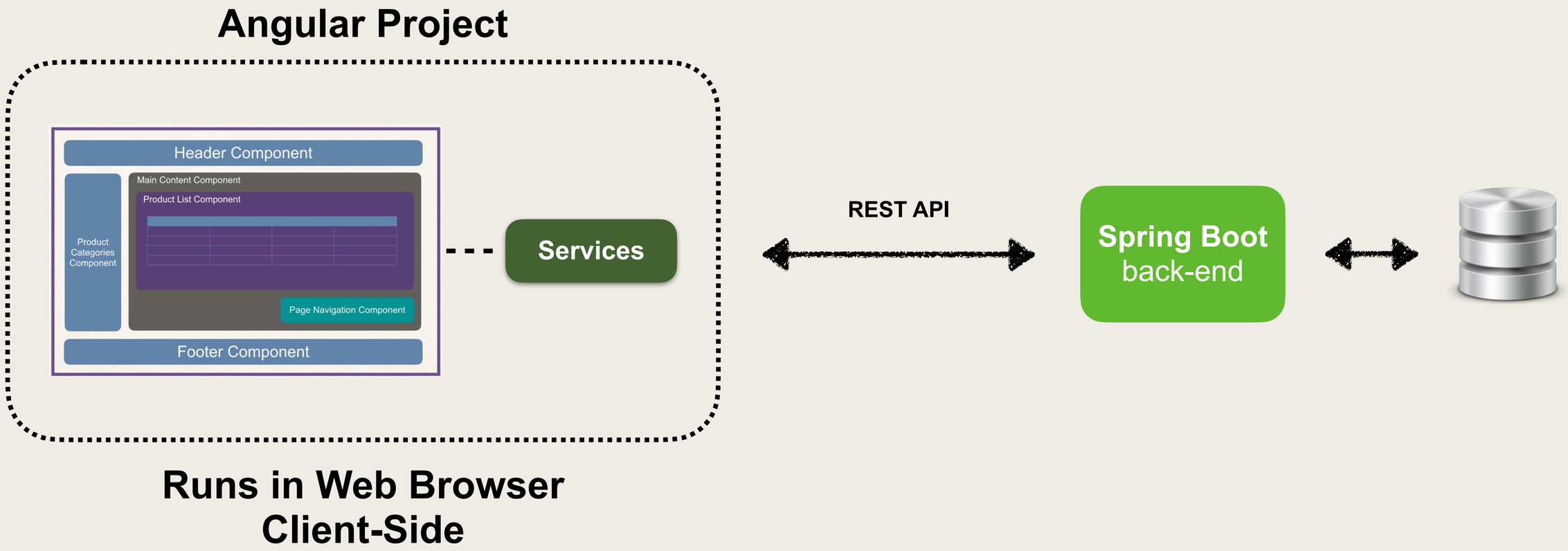

© luv2code LLC

Step 4: Create Angular service to call REST APIs

- Angular "Service" is code developed in TypeScript
- Service is a helper class that provides desired functionality
- Part of your Angular application and runs in the web browser client-side

© luv2code LLC

Application Interaction



© luv2code LLC

Step 4: Create Angular service to call REST APIs

- REST client provided by Angular:
  - **`HttpClient`** ... part of **`HttpClientModule`**
- Add support in the application module

File: src/app/app.module.ts

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

@NgModule({
  declarations: [
    AppComponent
 ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Support for HttpClientModule

© luv2code LLC

Step 4: Create Angular service to call REST APIs

File: src/app/services/product.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Product } from '../common/product';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

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

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

  constructor(private httpClient: HttpClient) { }

  getProductList(): Observable<Product[]> {
    return this.httpClient.get<GetResponse>(this.baseUrl).pipe(
      map(response => response._embedded.products)
    );
  }
}

interface GetResponse {
  _embedded: {
    products: Product[];
  }
}
```

Our service can be injected into other classes / components

Inject httpClient

Unwraps the JSON from Spring Data REST _embedded entry

Returns an observable

Map the JSON data from Spring Data REST to Product array

© luv2code LLC

Step 5: Develop Angular to subscribe to data

File: src/app/components/product-list/product-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ProductService } from 'src/app/services/product.service';
import { Product } from 'src/app/common/product';

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

  products: Product[];

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.listProducts();
  }

  listProducts() {
    this.productService.getProductList().subscribe(
      data => {
        this.products = data;
      }
    )
  }
}
```

Method is invoked once you "subscribe"

Assign results to the Product array

© luv2code LLC

Step 6: Display the Data in an HTML page

File: src/app/components/product-list/product-list.component.html

```
<p *ngFor="let tempProduct of products">
    {{ tempProduct.name}}: {{ tempProduct.unitPrice | currency:'USD'}}
</p>
```

Products

JavaScript - The Fun Parts: $19.99

Spring Framework Tutorial: $29.99

Kubernetes - Deploying Containers: $24.99

Internet of Things (IoT) - Getting Started: $29.99

The Go Programming Language: A to Z: $24.99

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

Restrictions are specific to scripts running in a web browser (JavaScript)

- By default, this coding will fail
- Web browsers will not allow script code to call APIs not on same origin
- Known as Same-origin policy
- Same-origin is composed of: scheme/protocol, hostname, port number
- Can relax this by adding "Cross-Origin Resource Sharing (CORS)" on server side application

© luv2code LLC

Add CrossOrigin support to Spring Boot App

File: ProductRepository.java

```
@CrossOrigin("http://localhost:4200")
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

Multiple

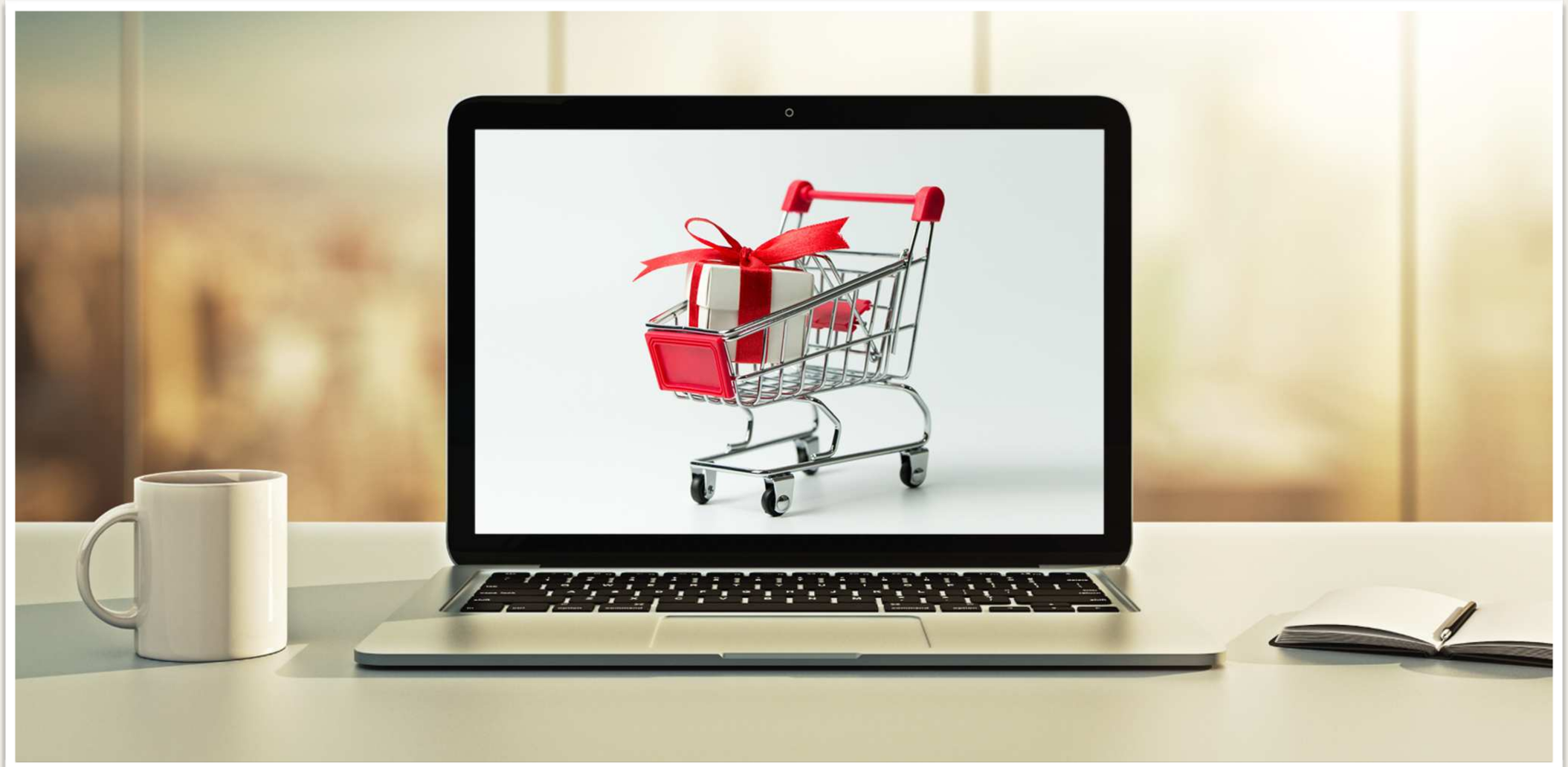
```
@CrossOrigin({"http://localhost:4200", "http://www.mycoolapp.com"})
```

Wildcard (any website)

```
@CrossOrigin
```


© luv2code LLC

Online Store Template Integration



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

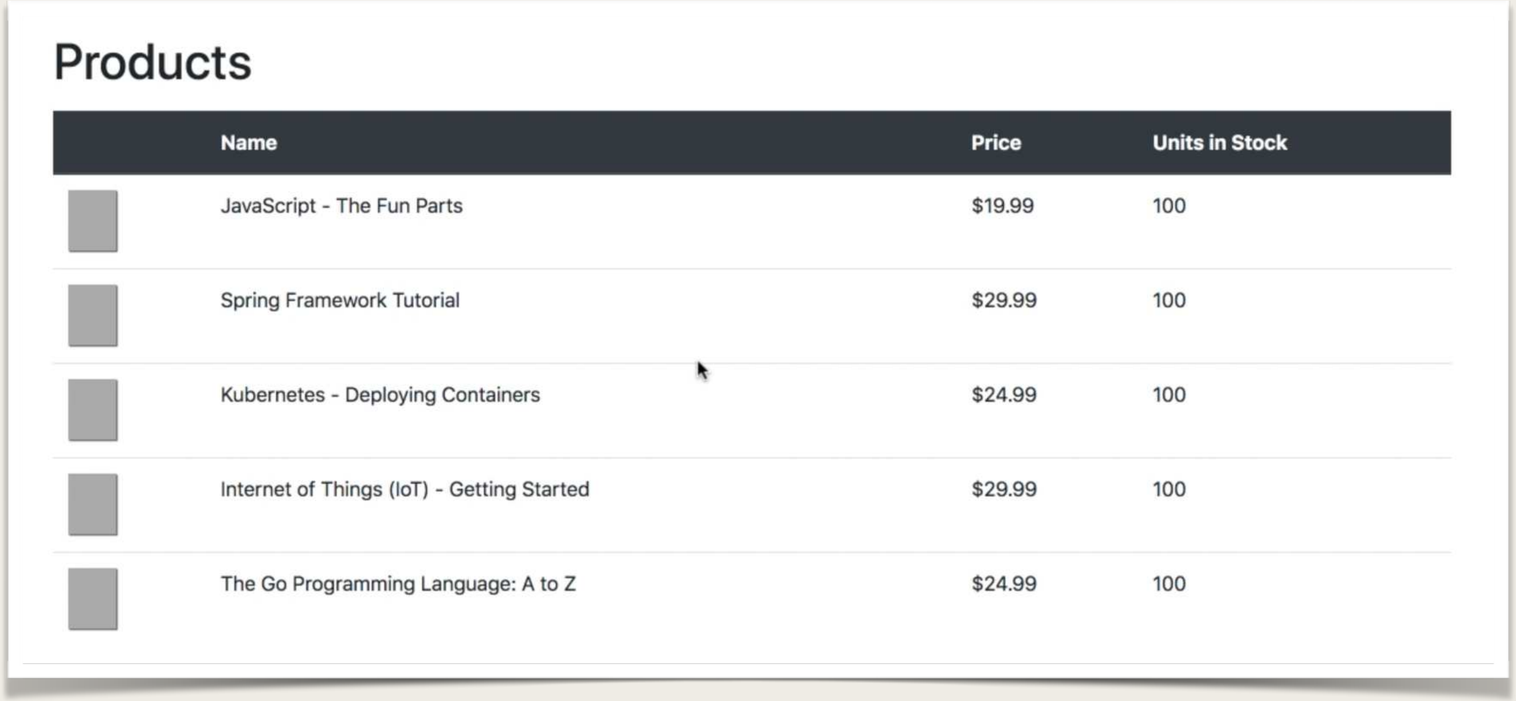
More functionality than originally planned with 2.0

"Feature creep"
Real-time LOL

© luv2code LLC

Current Status

- At the moment, we have a basic front end
- Proof of concept for listing the products: Angular + Spring Boot



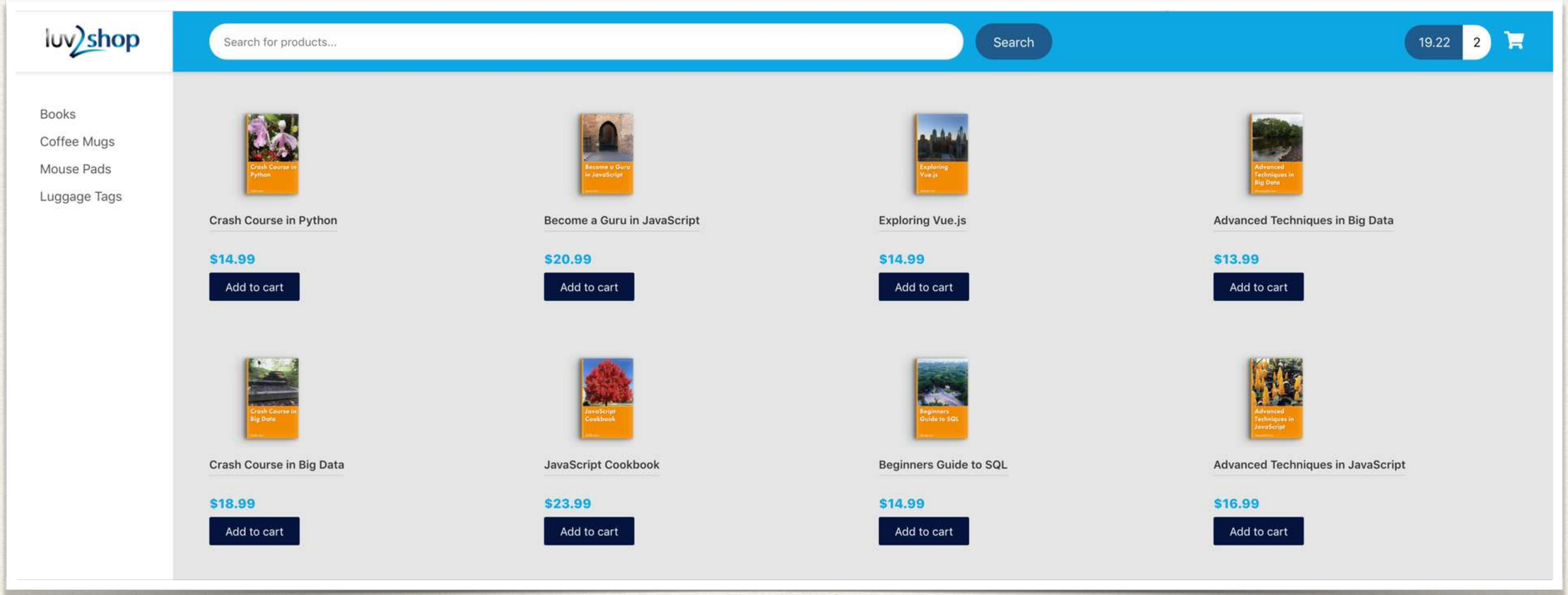
Products

	Name	Price	Units in Stock
	JavaScript - The Fun Parts	$19.99	100
	Spring Framework Tutorial	$29.99	100
	Kubernetes - Deploying Containers	$24.99	100
	Internet of Things (IoT) - Getting Started	$29.99	100
	The Go Programming Language: A to Z	$24.99	100

© luv2code LLC

Online Shop Template Integration

- We need a website that looks like an online shop

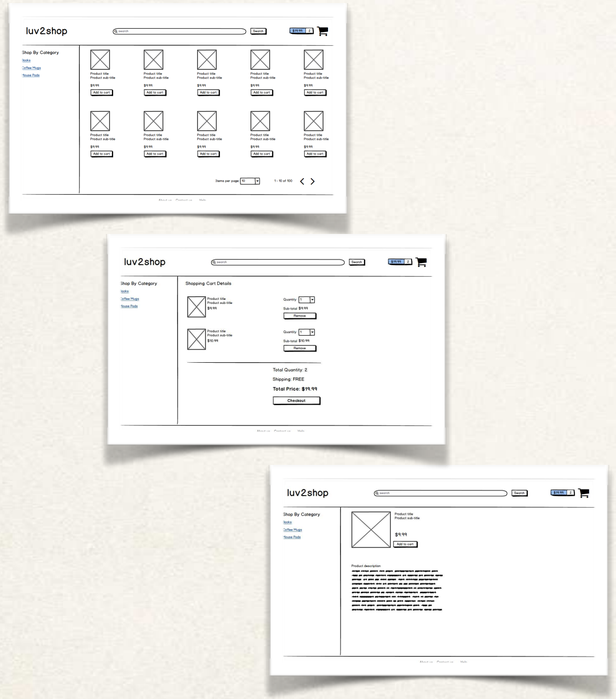


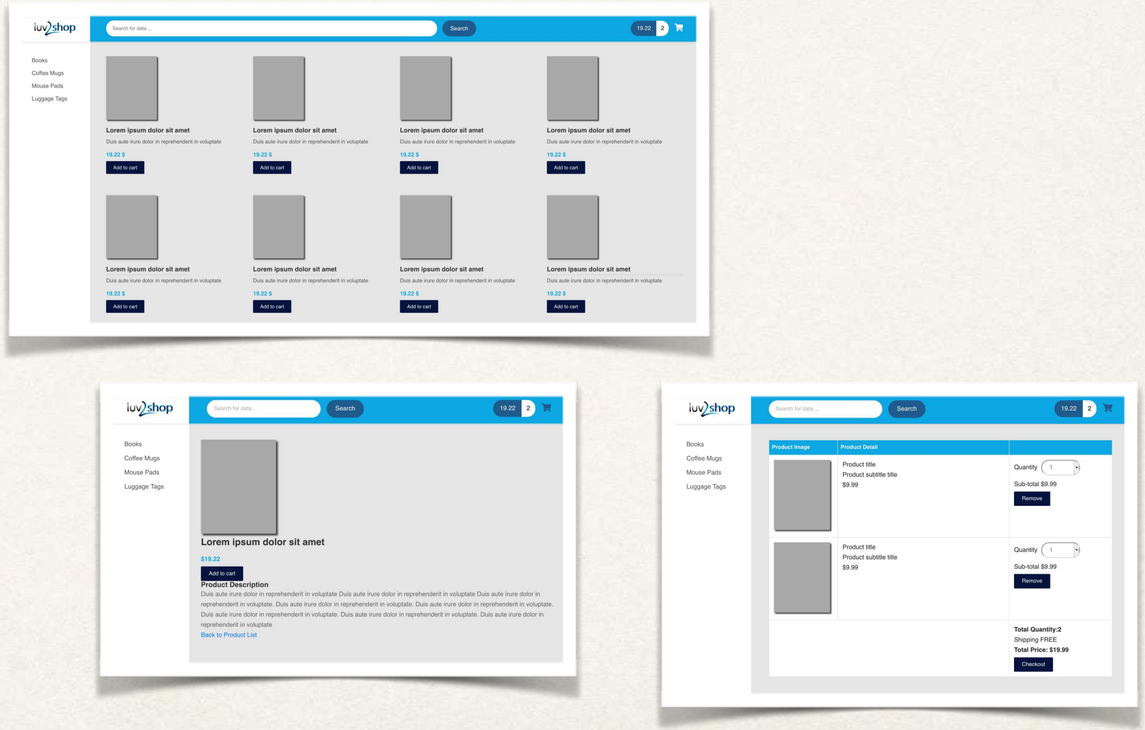
© luv2code LLC

Wireframes to Web Template

- I sent the wireframes to a friend
- He created a web template using HTML and CSS

He delivered a zip file *.html and *.css

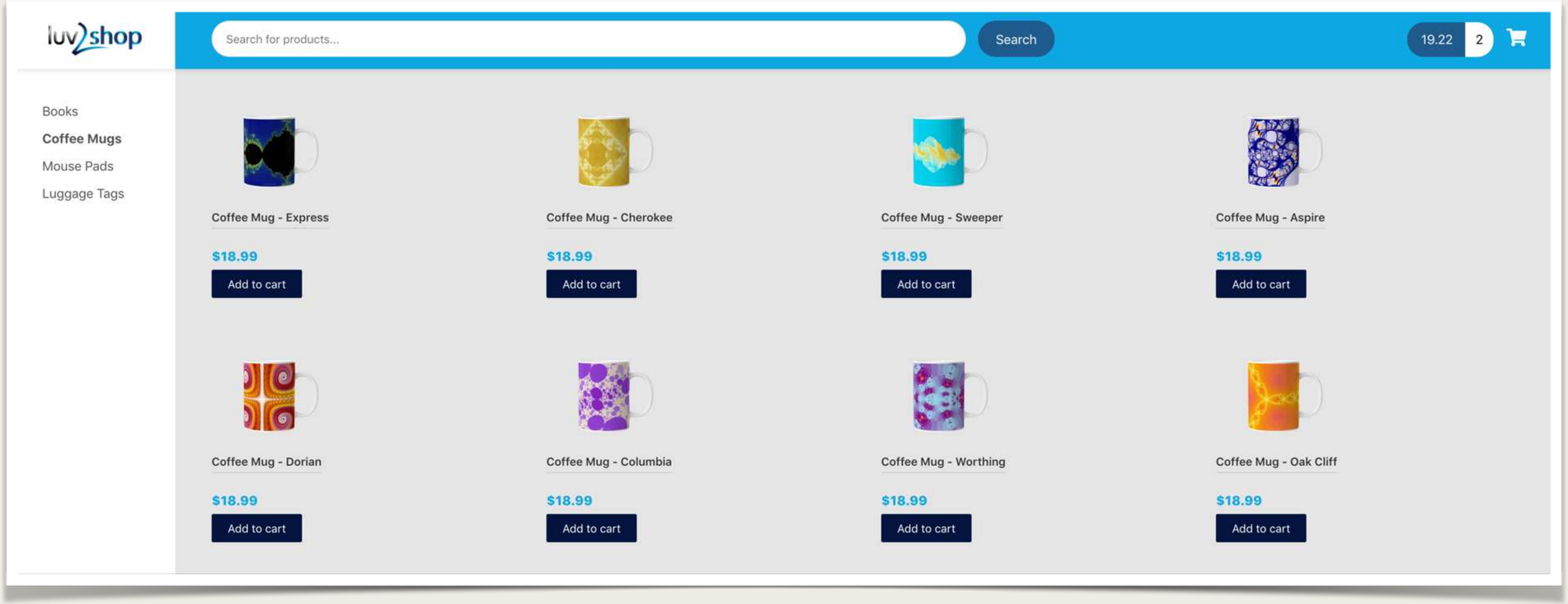




© luv2code LLC

Online Shop Template Integration

- Integrate the HTML template with our Angular `*.component.html` files
- We'll add the CSS styles to our project



© luv2code LLC

A Note about Cascading Style Sheets (CSS)

- The template uses CSS for styles, layout and colors
- Teaching CSS is beyond the scope of this course
- If you would like a tutorial on CSS, free online tutorials are available

www.w3schools.com/css

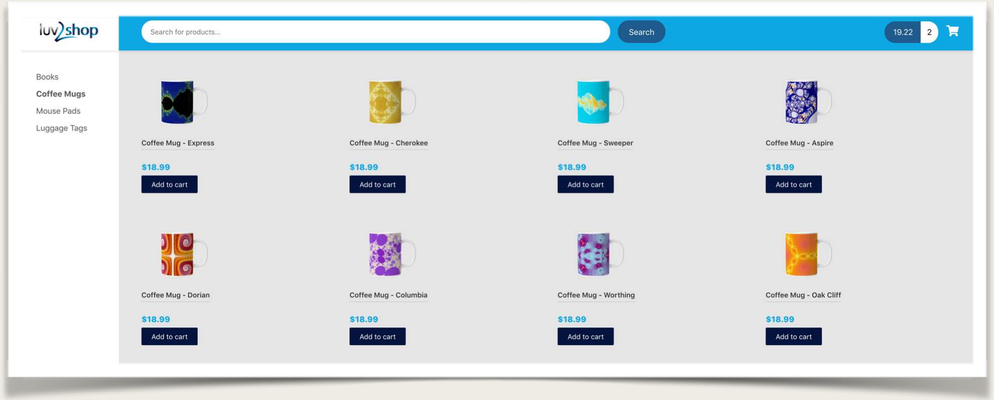
www.w3schools.com/bootstrap4

© luv2code LLC

Development Process

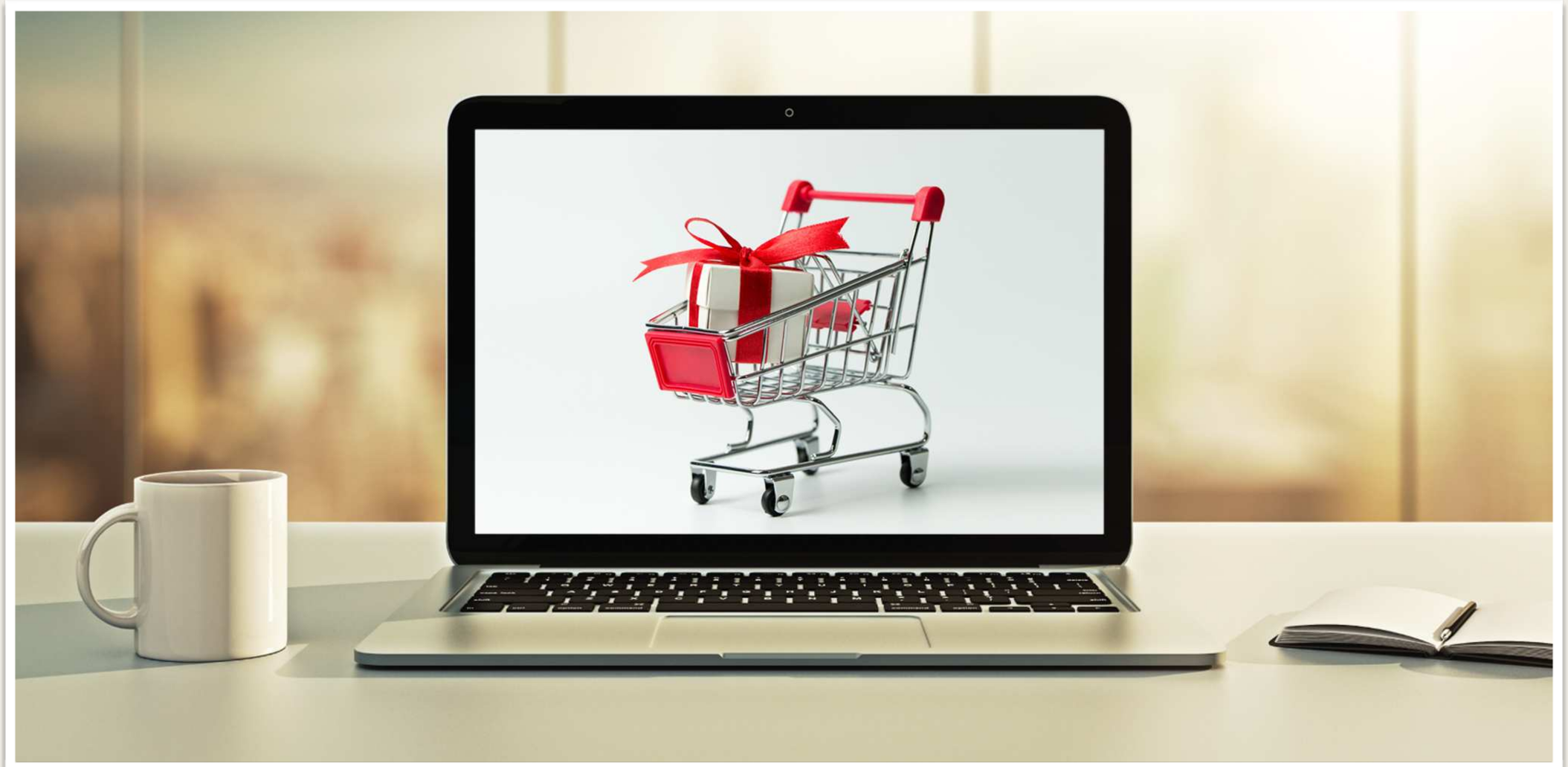
Step-By-Step

1. Download the HTML template starter files
2. Install Bootstrap CSS styles locally using `npm`
3. Add local custom CSS styles to Angular `src/styles.css` file
4. Integrate template files into Angular app
5. Add support for icons and logos
6. Enhance our application with product images



© luv2code LLC

Search By Category



Release 2.0 - Plan

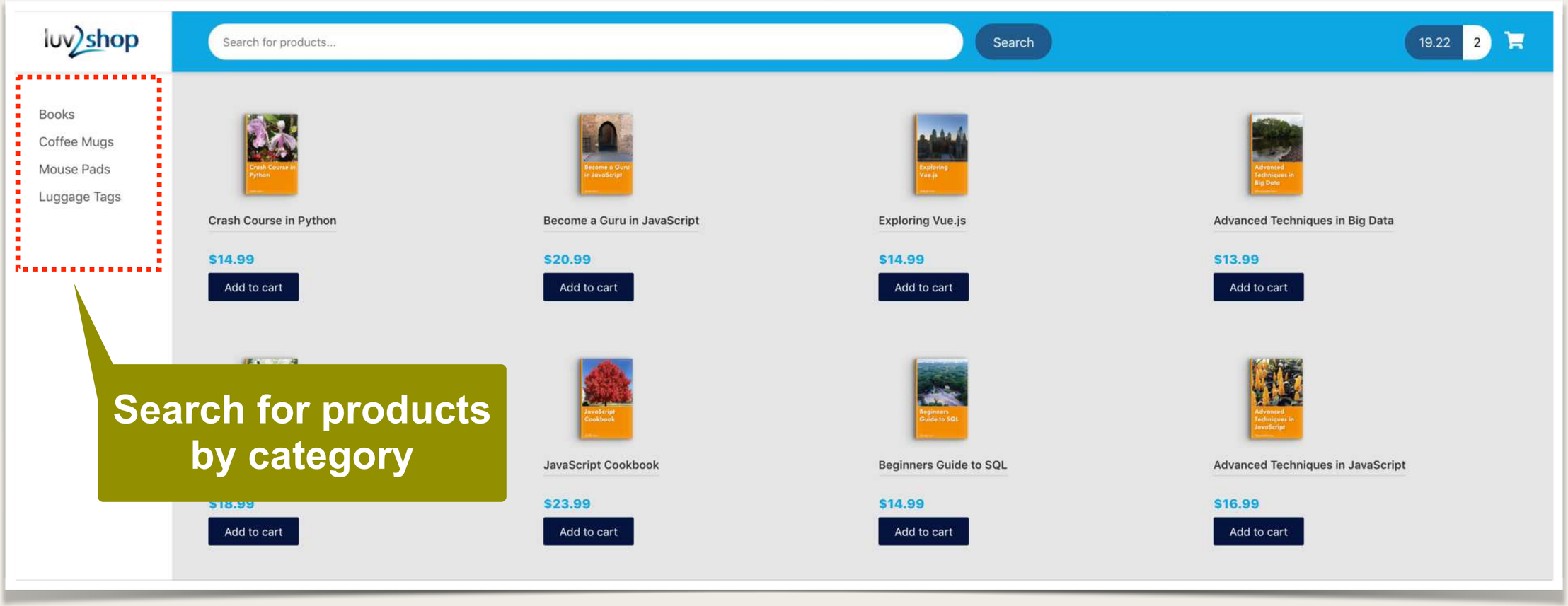
From: The Boss

- 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



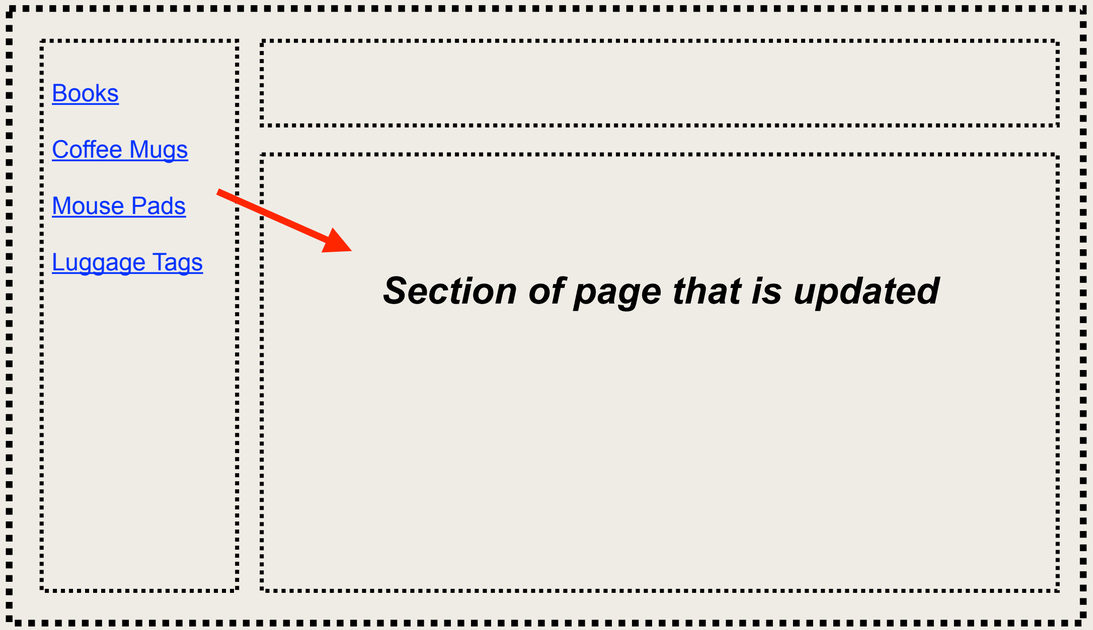
© luv2code LLC

DEMO

© 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

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

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

- 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

- 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

- 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

- 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

- Configure the router in the application module

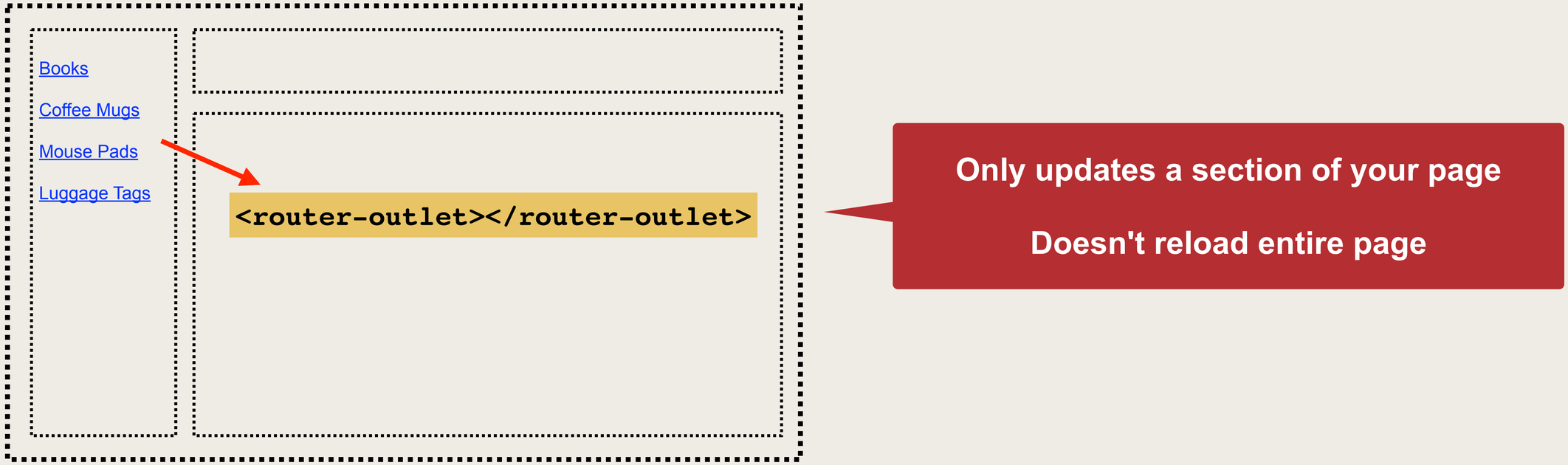
File: app.module.ts

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


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


© luv2code LLC

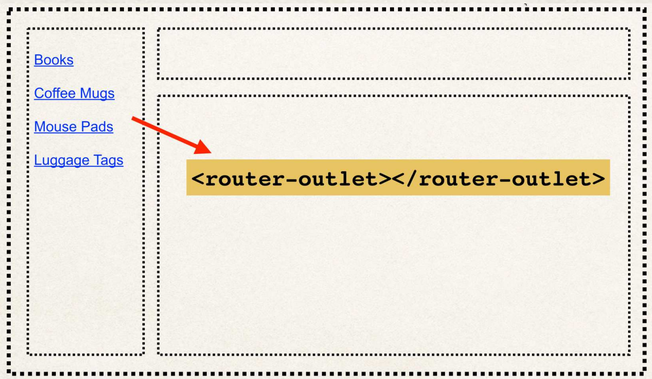
Step 3: Define the Router Outlet

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



© 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> -->

...
```

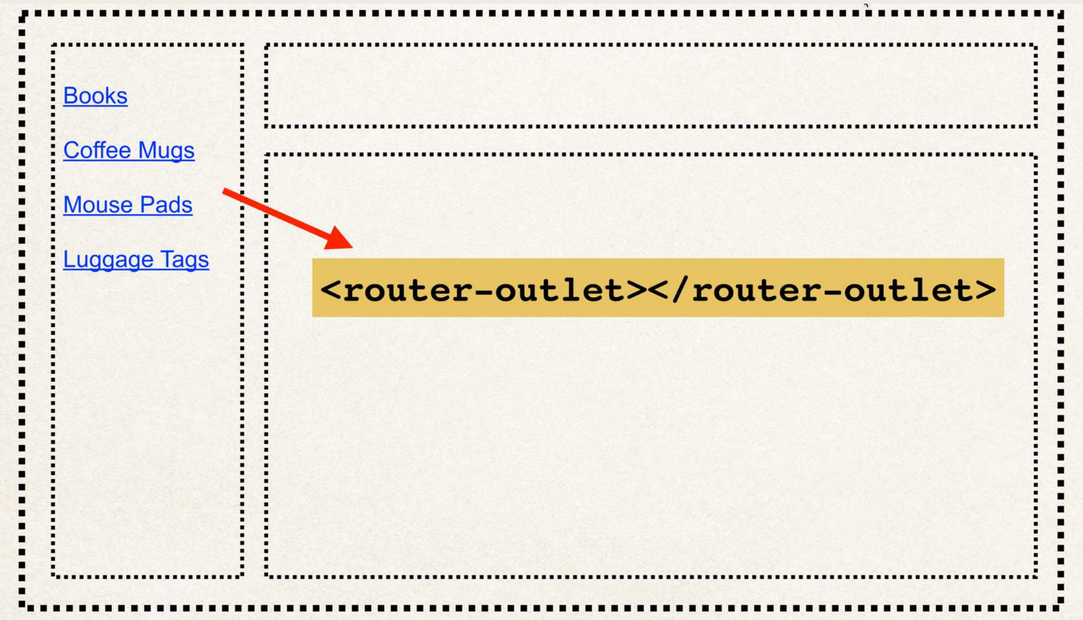
Based on Router configuration
Display appropriate component here

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

- 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

Based on Routes configuration, use ProductListComponent

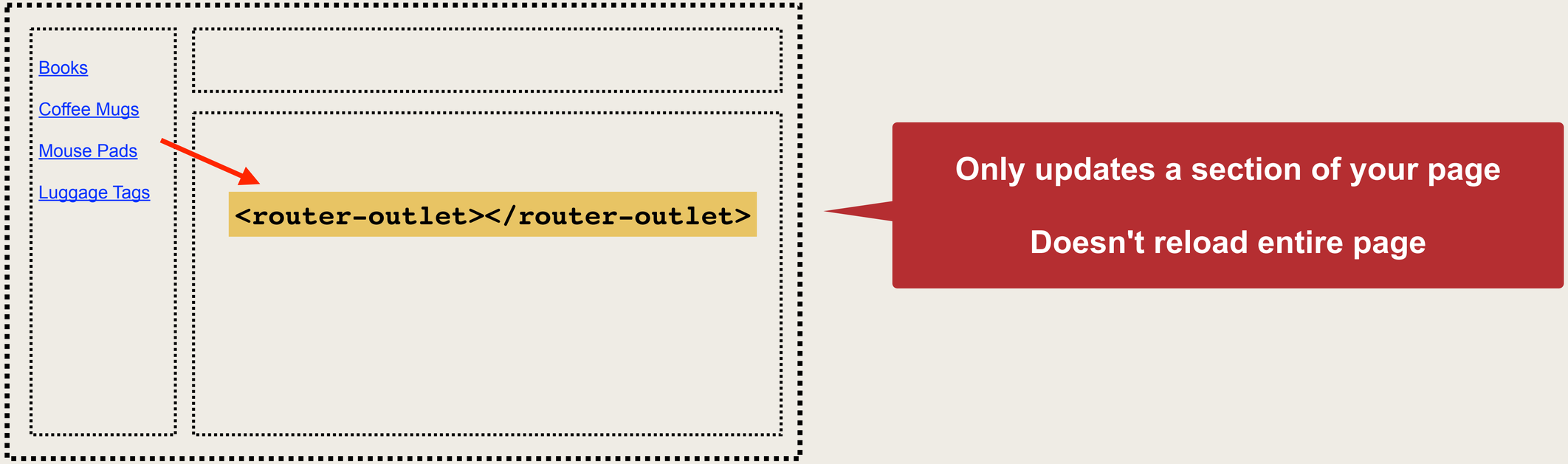
File: app.component.html

```
...
   <!-- 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

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

- 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 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

- 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);

}
```

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

- 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

- `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

- 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

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

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

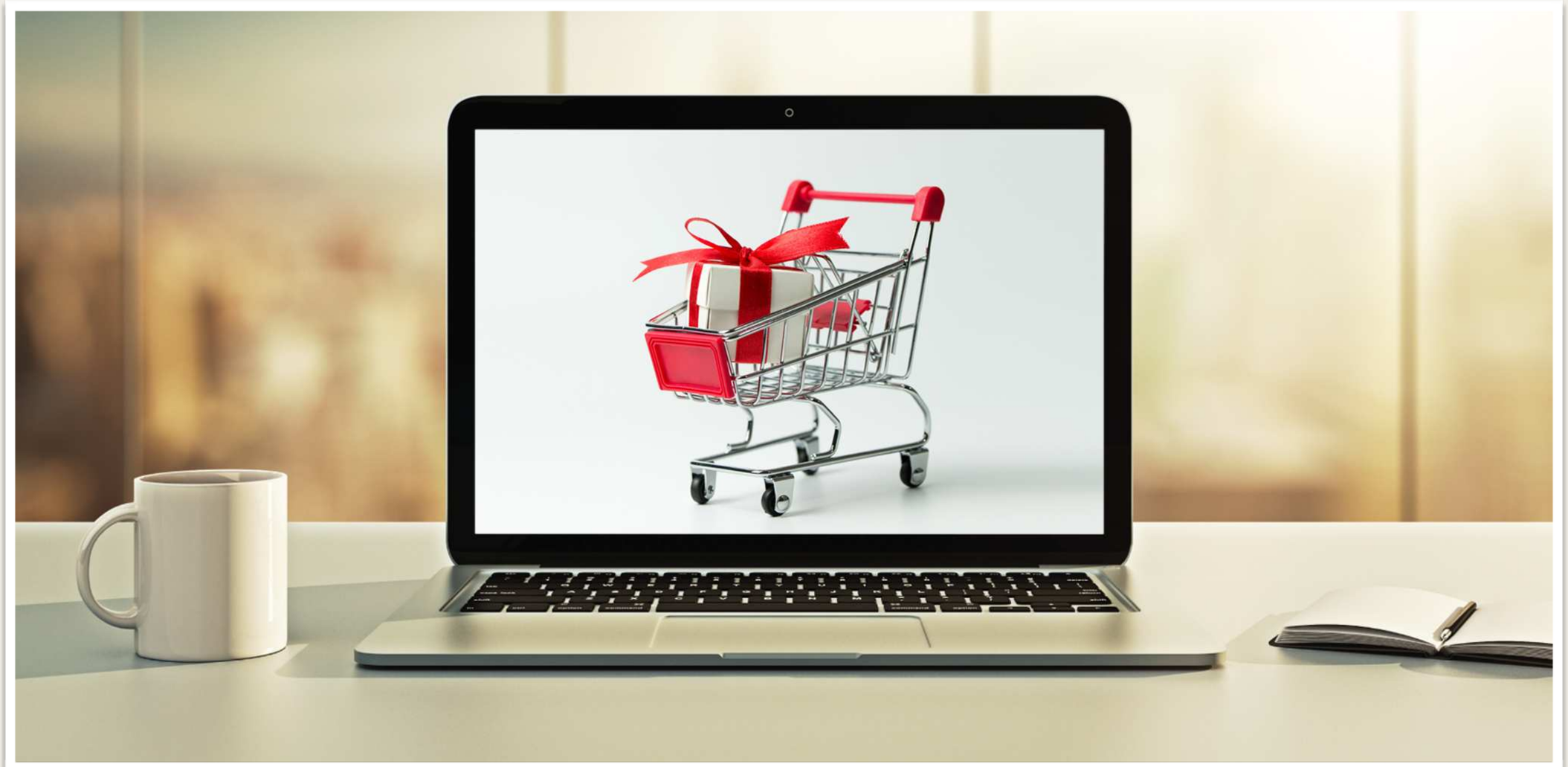
Development Process

Step-By-Step

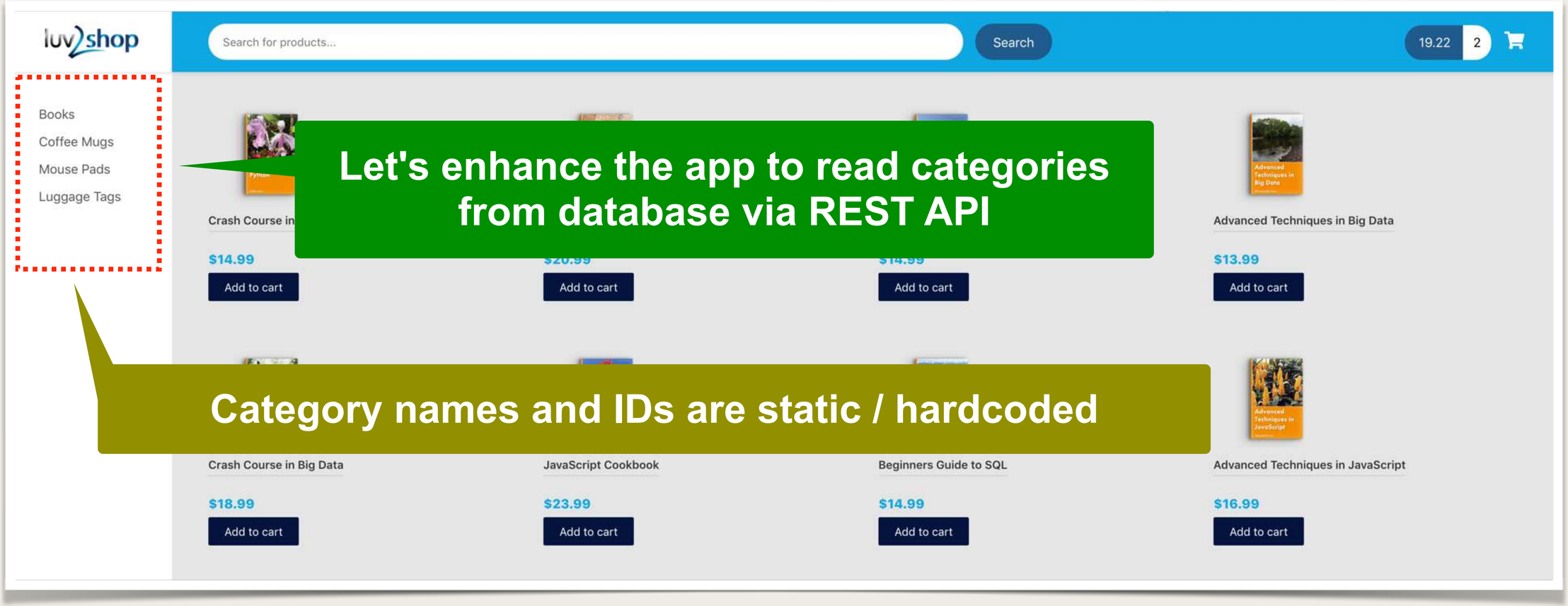
1. Define routes
2. Configure Router based on our routes
3. Define the Router Outlet
4. Set up Router Links to pass category id param
5. Enhance ProductListComponent to read category id param
6. Modify Spring Boot app - REST Repository needs new method
7. Update Angular Service to call new URL on Spring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

© luv2code LLC

Search By Category



Search for Products by Category



© luv2code LLC

Development Process

Step-By-Step

1. Modify Spring Boot app - Expose entity ids
2. Create a class: ProductCategory
3. Create new component for menu
4. Enhance menu component to read data from product service
5. Update product service to call URL on Spring Boot app
6. In HTML, replace hard-coded links with menu component

© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- By default, Spring Data REST does not expose entity ids
- We need entity IDs for a number of use cases
  - Get a list of product categories by id
  - Master/detail view ... get a product by id

© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- By default, Spring Data REST does not expose entity ids

localhost:8080/api/product-category

Apps luv2shop - Full St...


```
{
  "_embedded" : {
    "productCategory" : [ {
      "categoryName" : "Books",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "products" : {
          "href" : "http://localhost:8080/api/product-category/1/products"
        }
      }
    }, {
      "categoryName" : "Coffee Mugs",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/2"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/2"
        },
        "products" : {
          "href" : "http://localhost:8080/api/product-category/2/products"
        }
      }
```


There is no entity id at the productCategory level

Actually, entity id is embedded in the HATEOS links.

But no easy access.
Requires parsing URL string

This approach is cumbersome and brittle :-(

© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- This is what we need

localhost:8080/api/product-category

Apps luv2shop - Full St...


```
{
  "_embedded" : {
    "productCategory" : [ {
      "id" : 1,
      "categoryName" : "Books",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "products" : {
          "href" : "http://localhost:8080/api/product-category/1/products"
        }
      }
    }, {
      "id" : 2,
      "categoryName" : "Coffee Mugs",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/2"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/2"
        }
```


Entity id at the productCategory level

Easy access! :-)

© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- Update Spring Data REST config to expose entity ids

```
// expose entity ids
//
// - gets a list of all entity classes from the entity manager
// - create an array of the entity types
// - get the entity types for the entities
// - expose the entity ids for the array of entity/domain types
```


© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- Update Spring Data REST config to expose entity ids

File: MyDataRestConfig.java

```
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

 private EntityManager entityManager;

 @Autowired
 public MyDataRestConfig(EntityManager theEntityManager) {
     entityManager =  theEntityManager;
 }

 ...
}
```

Autowire JPA entity manager

© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

File: MyDataRestConfig.java

```java
@Override
public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {

    // ...

    // call an internal helper method
    exposeIds(config);
}

private void exposeIds(RepositoryRestConfiguration config) {

    // expose entity ids
    //

    // - gets a list of all entity classes from the entity manager
    Set<EntityType<?>> entities = entityManager.getMetamodel().getEntities();

    // - create an array of the entity types
    List<Class> entityClasses = new ArrayList<>();

    // - get the entity types for the entities
    for (EntityType tempEntityType : entities) {
        entityClasses.add(tempEntityType.getJavaType());
    }

    // - expose the entity ids for the array of entity/domain types
    Class[] domainTypes = entityClasses.toArray(new Class[0]);
    config.exposeIdsFor(domainTypes);
}
```


© luv2code LLC

Step 1: Modify Spring Boot app - Expose entity ids

- The configuration gives us the desired output

localhost:8080/api/product-category

Apps luv2shop - Full St...


```
{
  "_embedded" : {
    "productCategory" : [ {
      "id" : 1,
      "categoryName" : "Books",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/1"
        },
        "products" : {
          "href" : "http://localhost:8080/api/product-category/1/products"
        }
      }
    }, {
      "id" : 2,
      "categoryName" : "Coffee Mugs",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/product-category/2"
        },
        "productCategory" : {
          "href" : "http://localhost:8080/api/product-category/2"
        }
```


Entity id at the productCategory level

Easy access! :-)

© luv2code LLC

Step 2: Create class: ProductCategory

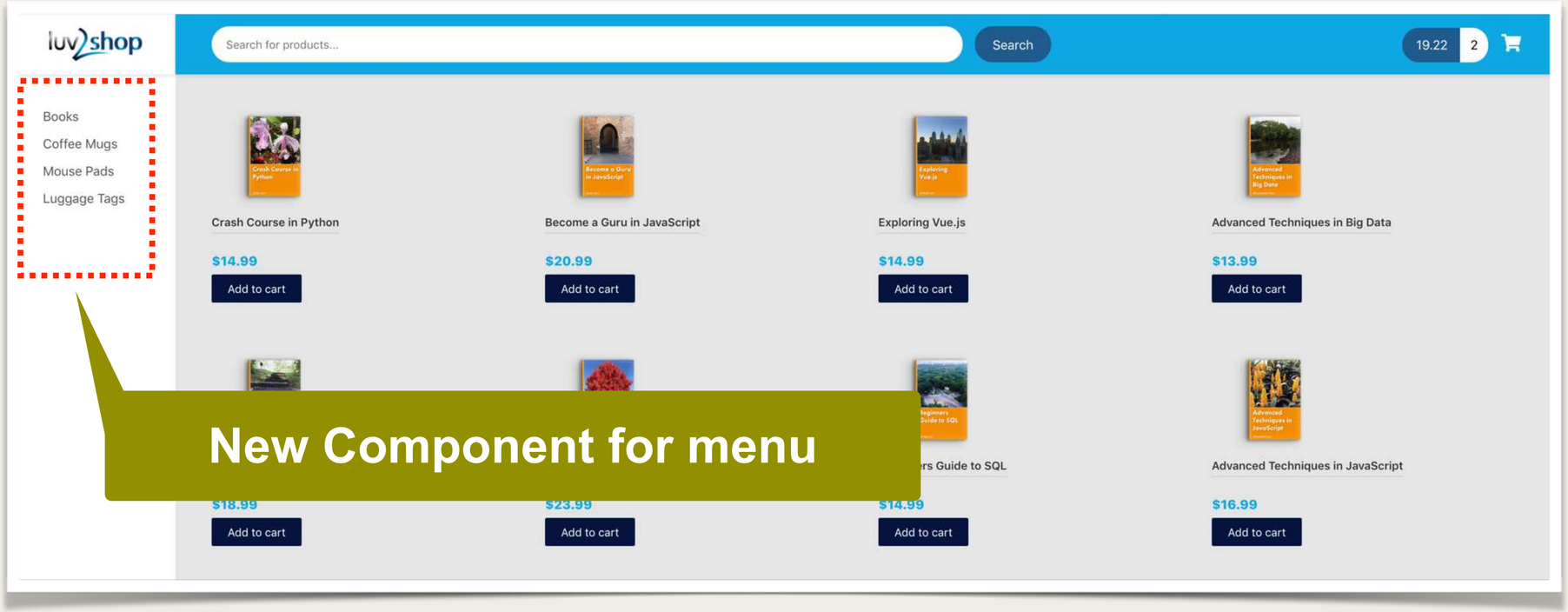
```
> ng generate class common/product-category
```

File: product-category.ts

```
export class ProductCategory {
    id: number;
    categoryName: string;
}
```


© luv2code LLC

Step 3: Create new component for menu




```
> ng generate component components/product-category-menu
```


© luv2code LLC

Step 4: Enhance component to read categories from service

File: product-category-menu.component.ts

```
export class ProductCategoryMenuComponent implements OnInit {

  productCategories: ProductCategory[];

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.listProductCategories();
  }

  listProductCategories() {

    this.productService.getProductCategories().subscribe(
      data => {
        console.log('Product Categories=' + JSON.stringify(data));
        this.productCategories = data;
      }
    )
  }

}
```

Define our property

Inject the service

Invoke the service

Log data returned from service

Assign data to our property

© luv2code LLC

Step 5: Update product service to call URL on Spring Boot app

File: product.service.ts

```
export class ProductService {

  private categoryUrl = 'http://localhost:8080/api/product-category';
  ...

  getProductCategories(): Observable<ProductCategory[]> {

    return this.httpClient.get<GetResponseProductCategory>(this.categoryUrl).pipe(
      map(response => response._embedded.productCategory)
    );
  }

}

interface GetResponseProductCategory {
  _embedded: {
    productCategory: ProductCategory[];
  }
}
```

URL for product categories

Call REST API

Returns an observable

Map the JSON data from Spring Data REST to ProductCategory array

Unwraps the JSON from Spring Data REST _embedded entry

© luv2code LLC

Step 6: Replace hard-coded links with menu component

Old Version

File: app.component.html

```
<!-- MENU SIDEBAR-->
<aside class="menu-sidebar d-none d-lg-block">
  <div class="logo">
    <a href="#">
      <img src="assets/images/logo.png" alt="luv2shop" class="img-responsive">
    </a>
  </div>
  <div class="menu-sidebar-content js-scrollbar1">
    <nav class="navbar-sidebar">
      <ul class="list-unstyled navbar-list">
        <li>
          <a routerLink="/category/1" routerLinkActive="active-link">Books</a>
        </li>
        <li>
          <a routerLink="/category/2" routerLinkActive="active-link">Coffee Mugs</a>
        </li>
      </ul>
    </nav>
  </div>
</aside>
<!-- END MENU SIDEBAR-->
```

New Version

File: app.component.html

```
<!-- MENU SIDEBAR-->
<aside class="menu-sidebar d-none d-lg-block">
  <div class="logo">
    <a href="#">
      <img src="assets/images/logo.png" alt="luv2shop" class="img-responsive">
    </a>
  </div>

  <app-product-category-menu></app-product-category-menu>

</aside>
<!-- END MENU SIDEBAR-->
```

Our new menu component

© luv2code LLC

Step 6: Replace hard-coded links with menu component

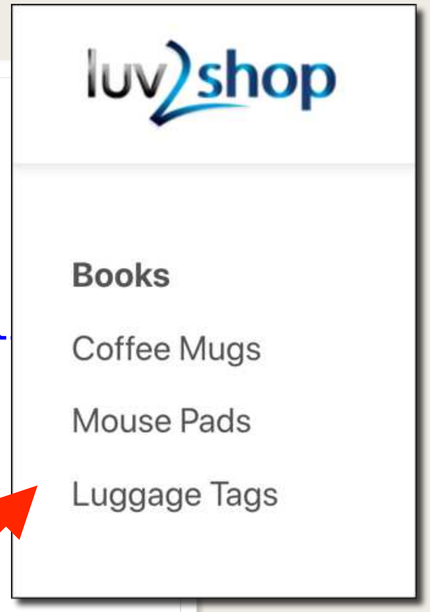
File: product-category-menu.component.html

```
<div class="menu-sidebar-content js-scrollbar1">
    <nav class="navbar-sidebar">
        <ul class="list-unstyled navbar-list">

            <li *ngFor="let tempProductCategory of productCategories">

                <a routerLink="/category/{{ tempProductCategory.id }}" routerLinkActive="active-l
                    {{ tempProductCategory.categoryName }}
                </a>

            </li>

        </ul>
    </nav>
</div>
```

Loop over categories and build links dynamically

Add category id to link

Display category name

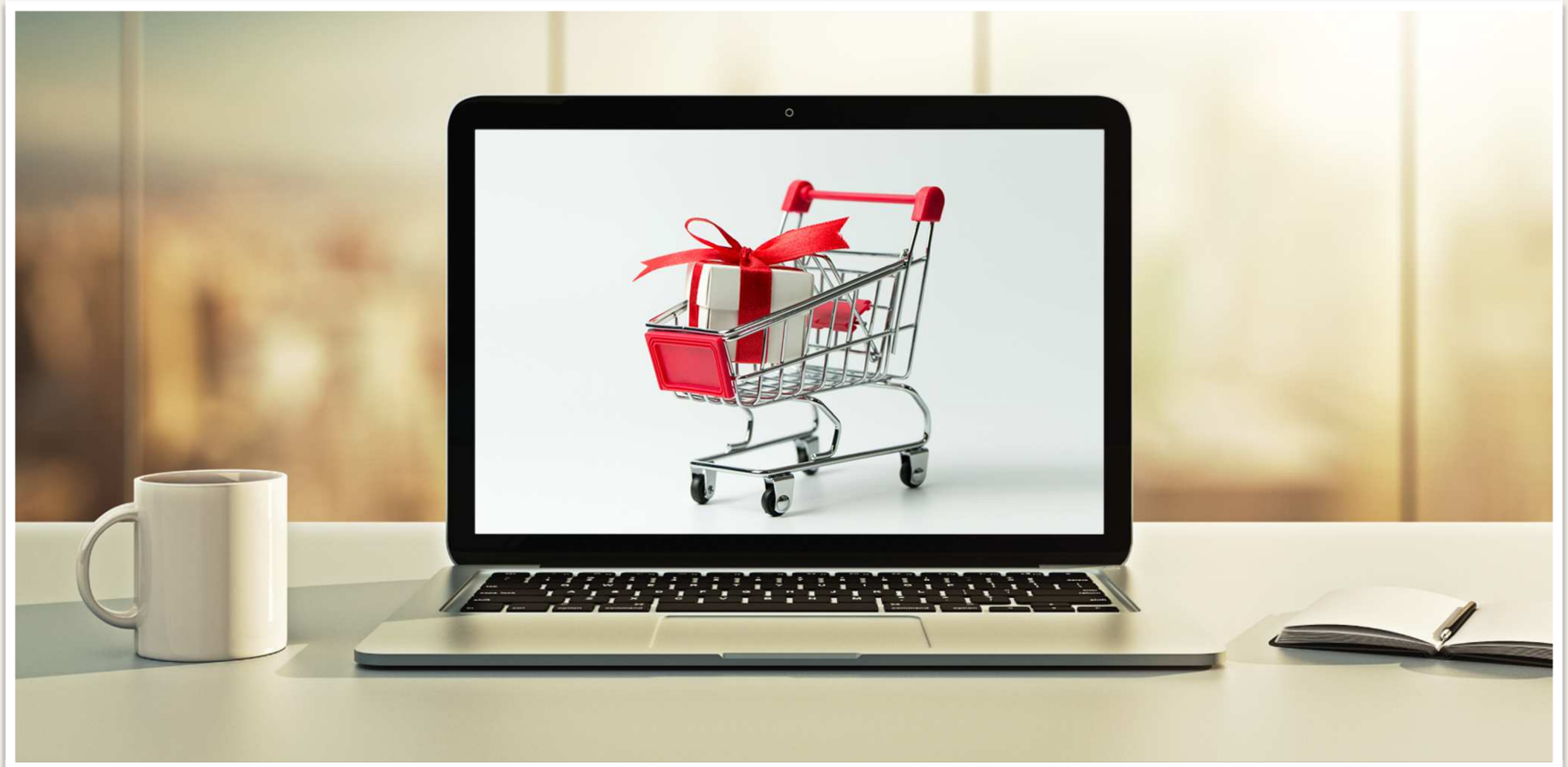


From REST API

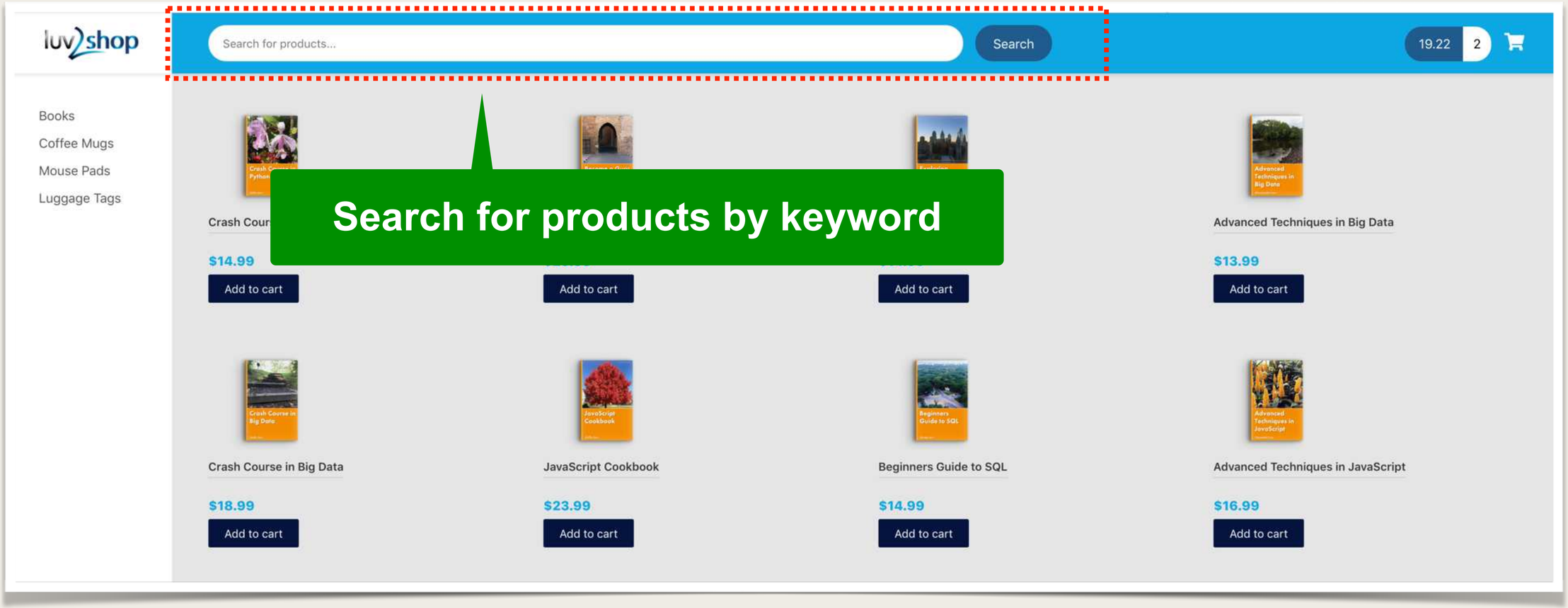
id	category_name
1	Books
2	Coffee Mugs
3	Mouse Pads
4	Luggage Tags

© luv2code LLC

Search By Keyword



Search for Products by Keyword



© luv2code LLC

Development Process

Step-By-Step

1. Modify Spring Boot app - Add a new search method
2. Create new component for search
3. Add new Angular route for searching
4. Update SearchComponent to send data to search route
5. Enhance ProductListComponent to search for products with ProductService
6. Update ProductService to call URL on Spring Boot app

© luv2code LLC

Step 1: Modify Spring Boot app - Search Method

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

Magic!

File: ProductRepository.java

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

    Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);

    ...

}
```


© luv2code LLC

Step 1: Modify Spring Boot app - Search Method

- Find products based on name

File: ProductRepository.java

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

    Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);

    ...

}
```

"Containing" ... similar to SQL: "LIKE"

© luv2code LLC

Step 1: Modify Spring Boot app - Search Method

File: ProductRepository.java

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

    Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);
```

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

```
SELECT * FROM Product p
WHERE
p.name LIKE CONCAT('%', :name ,'%')
```


© luv2code LLC

Step 1: Modify Spring Boot app - Search Method

File: ProductRepository.java

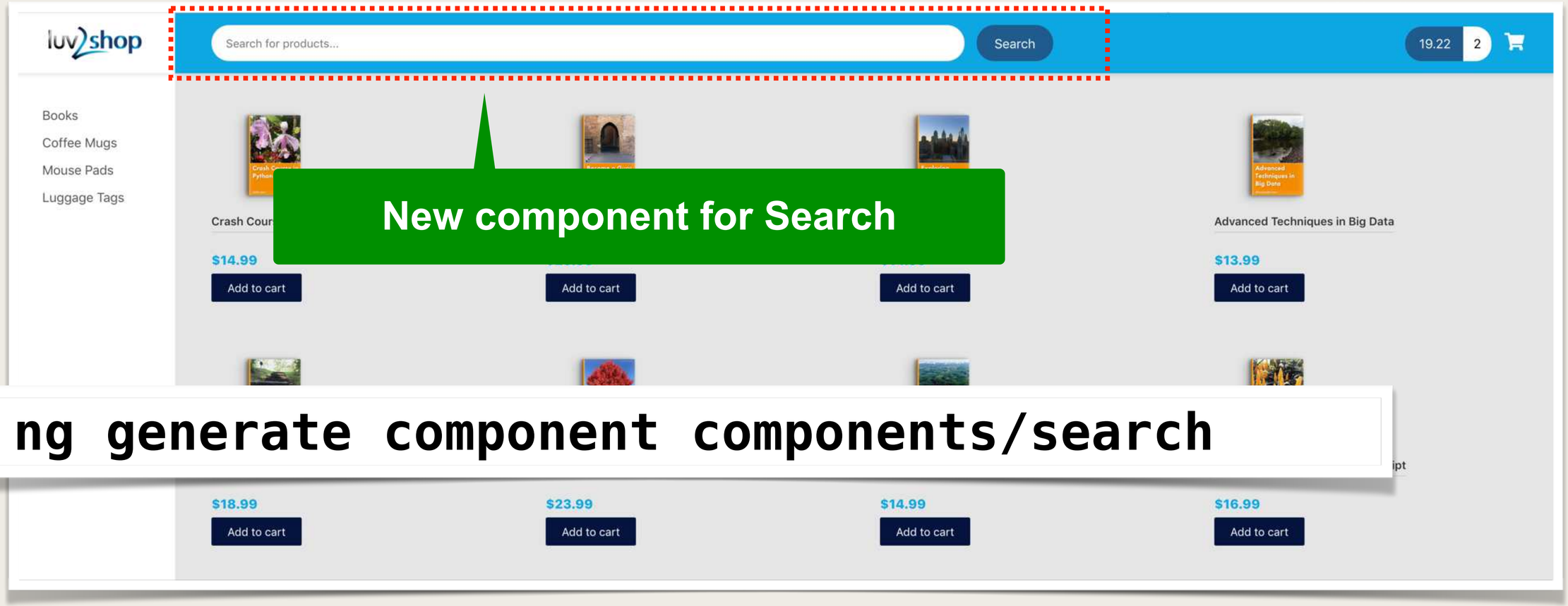
```
public interface ProductRepository extends JpaRepository<Product, Long> {

    Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);

    ...

}
```

http://localhost:8080/api/products/search/findByNameContaining?name=Python

To pass data to REST API

© luv2code LLC

Step 2: Create new component for Search




```
> ng generate component components/search
```


© luv2code LLC

Step 3: Add new Angular route for searching

File: app.module.ts

```
...
  {path: 'search/:keyword', component: ProductListComponent},
...
```

Parameter

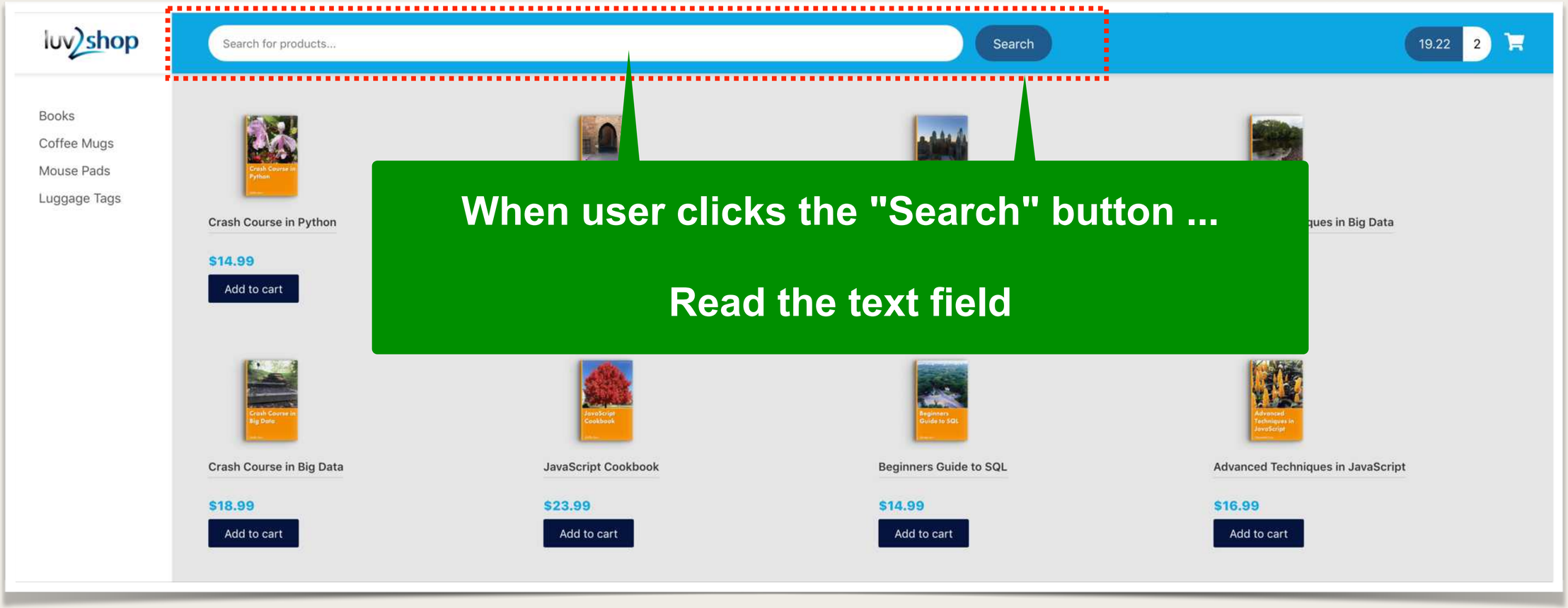
© luv2code LLC

Quick Discussion

Event Binding

© luv2code LLC

Quick Discussion - Event Binding



© luv2code LLC

Event Binding

- In Angular, you can listen for events with "event binding"
- In other languages/frameworks, also know "

Event handler

```
<button (click)="doMyCustomWork()">Search</button>
```

Listen for "click" event

Call a method in our Angular component code

Can be any method name we define

© luv2code LLC

Event Binding - Example

Call a method in our Angular component code

```
<button (click)="doMyCustomWork()">Search</button>
```

Listen for "click" event

```
export class SearchComponent implements OnInit {

  doMyCustomWork() {
    console.log(`Hey! You pushed my button!`);
 }

  ...
}
```


© luv2code LLC

Reading User Input

symbol
Template reference variable

Provides access to the element

```
<input #myInput type="text"
       (keyup.enter)="doMyCustomWork(myInput.value)"  />
```

The text the user typed in

Listen for the "enter" key

```
export class SearchComponent implements OnInit {

  doMyCustomWork(info: string) {
    console.log(`Hey! This is your data: ${info}`);
 }
 ...
}
```


© luv2code LLC

Reading User Input

symbol
Template reference variable

Provides access to the element

```
<input #myInput type="text"
       (keyup.enter)="doMyCustomWork(myInput.value)"  />

<button (click)="doMyCustomWork(myInput.value)">Search</button>
```

The text the user typed in

Listen for "click" event

```
export class SearchComponent implements OnInit {

  doMyCustomWork(info: string) {
    console.log(`Hey! This is your data: ${info}`);
 }
 ...
}
```


© luv2code LLC

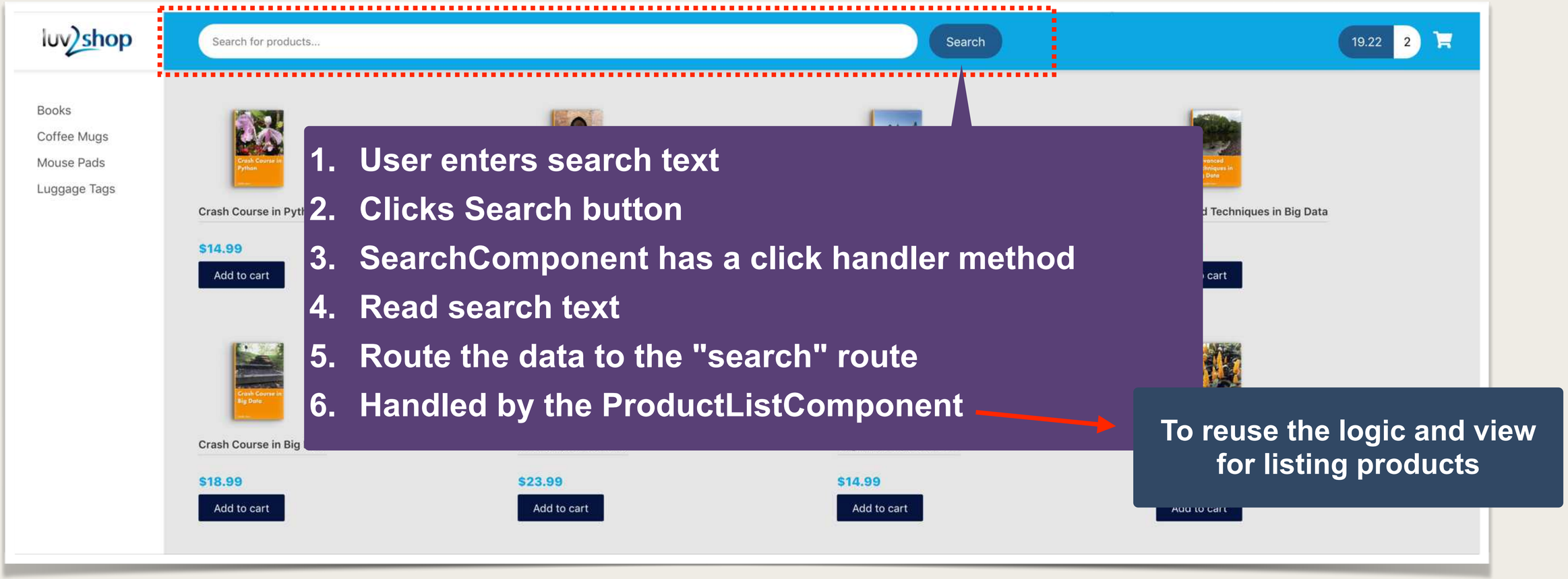
Other Events

Name	Description
focus	An element has received focus
blur	An element has lost focus
keyup	Any key is released. For a specific key, the ***enter*** key, use: **keyup.*enter***
keydown	Any key is pressed
dblclick	The mouse is clicked twice on an element
more ….	

https://developer.mozilla.org/en-US/docs/Web/Events

© luv2code LLC

Step 4: Update SearchComponent to send data to search route



© luv2code LLC

Step 4: Update SearchComponent to send data to search route

File: search.component.html

```
<div class="form-header">

    <input #myInput type="text"
           placeholder="Search for products ..."
           class="au-input au-input-xl"
           (keyup.enter)="doSearch(myInput.value)"  />

    <button (click)="doSearch(myInput.value)" class="au-btn-submit">
        Search
    </button>

</div>
```


© luv2code LLC

Step 4: Update SearchComponent to send data to search route

Method defined in our component

We can give it any name

File: search.component.ts

```
export class SearchComponent implements OnInit {

  constructor(private router: Router) { }

  doSearch(value: string) {
    console.log(`value=${value}`);
    this.router.navigateByUrl(`/search/${value}`);
  }

  ...
}
```

Route the data to our "search" route

It will be handled by the ProductListComponent

File: search.component.html

```
<div class="form-header">

    <input #myInput type="text"
           placeholder="Search for products ..."
           class="au-input au-input-xl"
           (keyup.enter)="doSearch(myInput.value)"  />

    <button (click)="doSearch(myInput.value)" class="au-btn-submit">
        Search
    </button>

</div>
```

To reuse the logic and view for listing products

© luv2code LLC

Step 5: Enhance component to search for products with product service

File: app.module.ts

```
...
  {path: 'search/:keyword', component: ProductListComponent},
...
```

File: product-list.component.ts

```
...
    const theKeyword: string = this.route.snapshot.paramMap.get('keyword');

    // now search for the products using keyword
    this.productService.searchProducts(theKeyword).subscribe(
      data => {
        this.products = data;
      }
    )
...
```

File: search.component.ts

```
export class SearchComponent implements OnInit {

 constructor(private router: Router) { }

  doSearch(value: string) {
    console.log(`value=${value}`);
    this.router.navigateByUrl(`/search/${value}`);
  }

  ...
}
```

Passed in from SearchComponent

© luv2code LLC

Step 6: Update product service to call URL on Spring Boot app

File: product.service.ts

```
...

  searchProducts(theKeyword: string): Observable<Product[]> {

    // need to build URL based on the keyword
    const searchUrl = `${this.baseUrl}/search/findByNameContaining?name=${theKeyword}`;

    return this.httpClient.get<GetResponseProducts>(searchUrl)
                                .pipe(
                                   map(response => response._embedded.products));

  }

...

interface GetResponseProducts {
  _embedded: {
    products: Product[];
  }
}
```

URL for searching products

Call REST API

Returns an observable

Map the JSON data from Spring Data REST to Product array

Unwraps the JSON from Spring Data REST _embedded entry

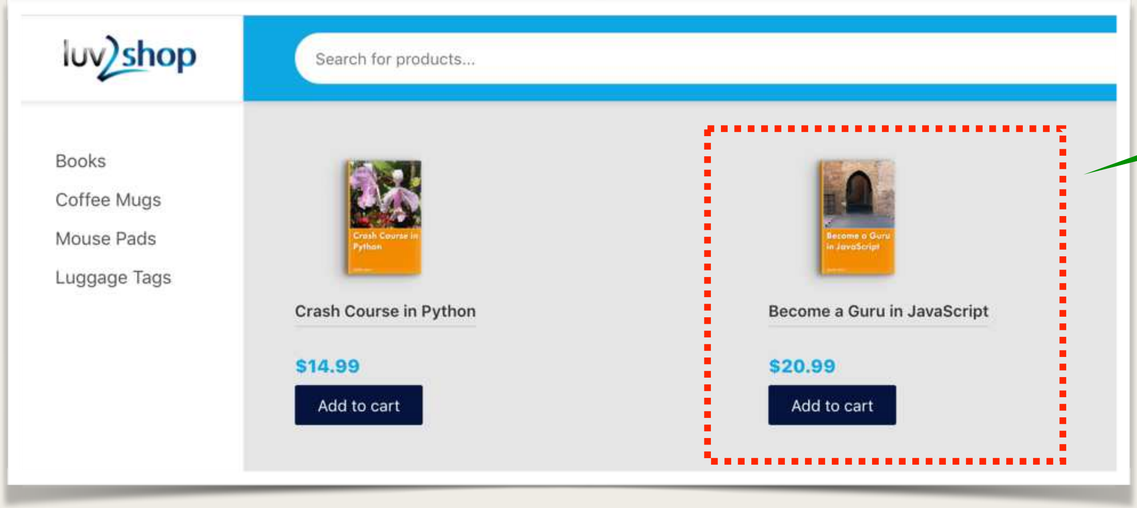
© luv2code LLC

Product Master-Detail View

Master-Detail View

Master View

User clicks the product to see details



Detail View

Application shows the details for selected product

© luv2code LLC

Development Process

Step-By-Step

1. Create new component for product details
2. Add new Angular route for product details
3. Add router links to the product-list-grid HTML page
4. Enhance ProductDetailsComponent to retrieve product from ProductService
5. Update ProductService to call URL on Spring Boot app
6. Update HTML page for ProductDetailsComponent to display product details

© luv2code LLC

Step 1: Create new component for product details

```
> ng generate component components/ProductDetails
```


© luv2code LLC

Step 2: Add new Angular route for product details

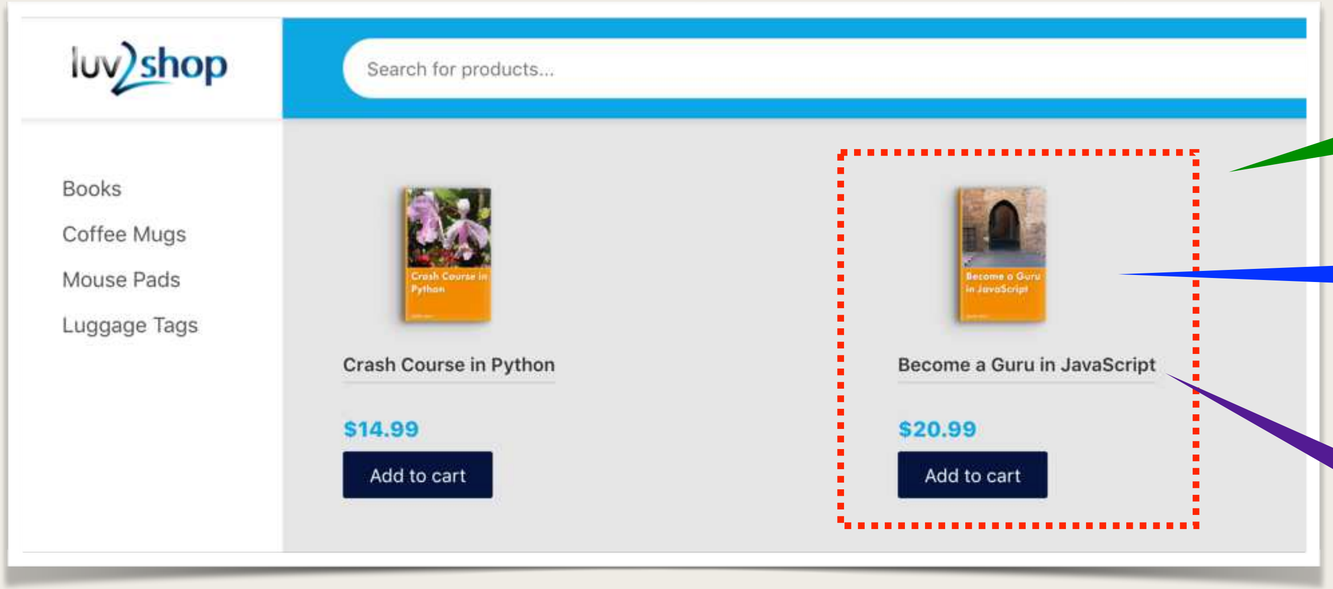
File: app.module.ts

```
...
  {path: 'products/:id', component: ProductDetailsComponent},
...
```

Parameter

© luv2code LLC

Step 3: Add router links to product-list-grid HTML



Master View

User clicks the product to see details

Add a link on the product image

Add a link on the product name

© luv2code LLC

Step 3: Add router links to product-list-grid HTML

File: product-list-grid.component.html

```
<!-- loop over the collection of products -->
<div *ngFor="let tempProduct of products" class="col-md-3">

   <div class="product-box">

      <a routerLink="/products/{{ tempProduct.id }}">
         <img src="{{ tempProduct.imageUrl }}" class="img-responsive">
      </a>

      <a routerLink="/products/{{ tempProduct.id }}">
         <h1>{{ tempProduct.name }}</h1>
      </a>
      ...
   </div>

</div>
```

Add a link on the product image

Add a link on the product name

© luv2code LLC

Step 3: Add router links to product-list-grid HTML

File: product-list-grid.component.html

```
<!-- loop over the collection of products -->
<div *ngFor="let tempProduct of products" class="col-md-3">

   <div class="product-box">

     <a routerLink="/products/{{ tempProduct.id }}">
        <img src="{{ tempProduct.imageUrl }}" class="img-responsive">
     </a>

     <a routerLink="/products/{{ tempProduct.id }}">
        <h1>{{ tempProduct.name }}</h1>
     </a>
      ...
   </div>

</div>
```

Parameter

File: app.module.ts

```
...
  {path: 'products/:id', component: ProductDetailsComponent},
...
```


© luv2code LLC

Step 3: Add router links to product-list-grid HTML

File: product-list-grid.component.html

```
<!-- loop over the collection of products -->
<div *ngFor="let tempProduct of products" class="col-md-3">

   <div class="product-box">

     <a routerLink="/products/{{ tempProduct.id }}">
        <img src="{{ tempProduct.imageUrl }}" class="img-responsive">
     </a>

     <a routerLink="/products/{{ tempProduct.id }}">
        <h1>{{ tempProduct.name }}</h1>
     </a>
      ...
   </div>

</div>
```

Parameter

File: app.module.ts

```
...
  {path: 'products/:id', component: ProductDetailsComponent},
...
```


© luv2code LLC

Step 4: Enhance ProductDetailsComponent to retrieve product from ProductService

File: product-details.component.ts

```
...
    // get the "id" param string. convert string to a number using the "+" symbol
    const theProductId: number = +this.route.snapshot.paramMap.get('id');

    this.productService.getProduct(theProductId).subscribe(
      data => {
        this.product = data;
      }
    )
...
```

Parameter

File: app.module.ts

```
...
  {path: 'products/:id', component: ProductDetailsComponent},
...
```


© luv2code LLC

Step 5: Update product service to call URL on Spring Boot app

File: product.service.ts

```
getProduct(theProductId: number): Observable<Product> {

  // need to build URL based on product id
  const productUrl = `${this.baseUrl}/${theProductId}`;

  return this.httpClient.get<Product>(productUrl);
}
```

URL for retrieving a product

http://localhost:8080/api/products/${theProductId}

Call REST API

Returns an observable

JSON Data return can be converted directly to Product object

NO NEED to unwrap the JSON from Spring Data REST

There is no _embedded entry

© luv2code LLC

Mapping JSON to Product class

URL for retrieving product id: 2

JSON Data can be converted directly to Product object

JSON properties map directly to properties in Product class

http://localhost:8080/api/products/2

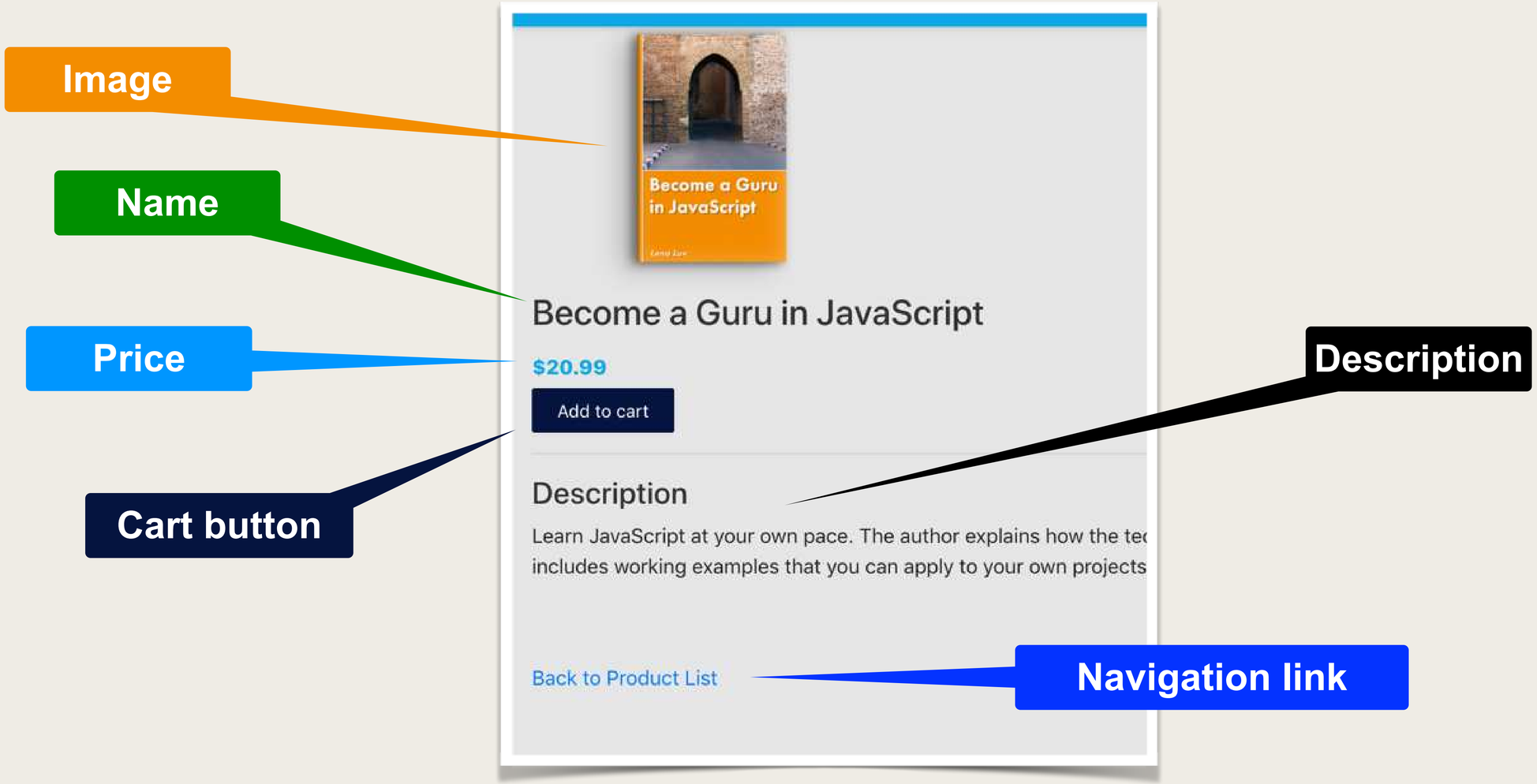

```
{
  "id" : 2,
  "sku" : "BOOK-TECH-1001",
  "name" : "Become a Guru in JavaScript",
  "description" : "Learn JavaScript at your own pace. The author explains how the
technology works in easy-to-understand language. This book includes working examples
that you can apply to your own projects. Purchase the book and get started today!",
  "unitPrice" : 20.99,
  "imageUrl" : "assets/images/products/books/book-luv2code-1001.png",
  "active" : true,
  "unitsInStock" : 100,
  "dateCreated" : "2020-01-11T16:58:24.000+0000",
  "lastUpdated" : null,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/products/2"
    },
    "product" : {
      "href" : "http://localhost:8080/api/products/2"
    },
    "category" : {
      "href" : "http://localhost:8080/api/products/2/category"
    }
  }
}
```


File: product.ts

```
export class Product {
    sku: string;
    name: string;
    description: string;
    unitPrice: number;
    imageUrl: string;
    active: boolean;
    unitsInStock: number;
    dateCreated: Date;
    lastUpdated: Date;
}
```


© luv2code LLC

Step 6: Update HTML page for ProductDetailsComponent to display product details

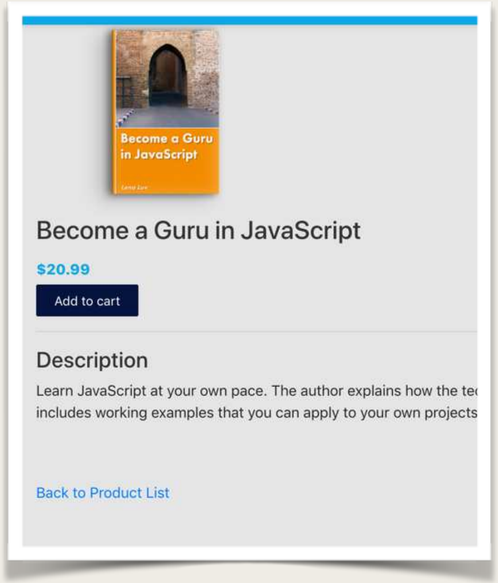


© luv2code LLC

Step 6: Update HTML page for ProductDetailsComponent to display product details

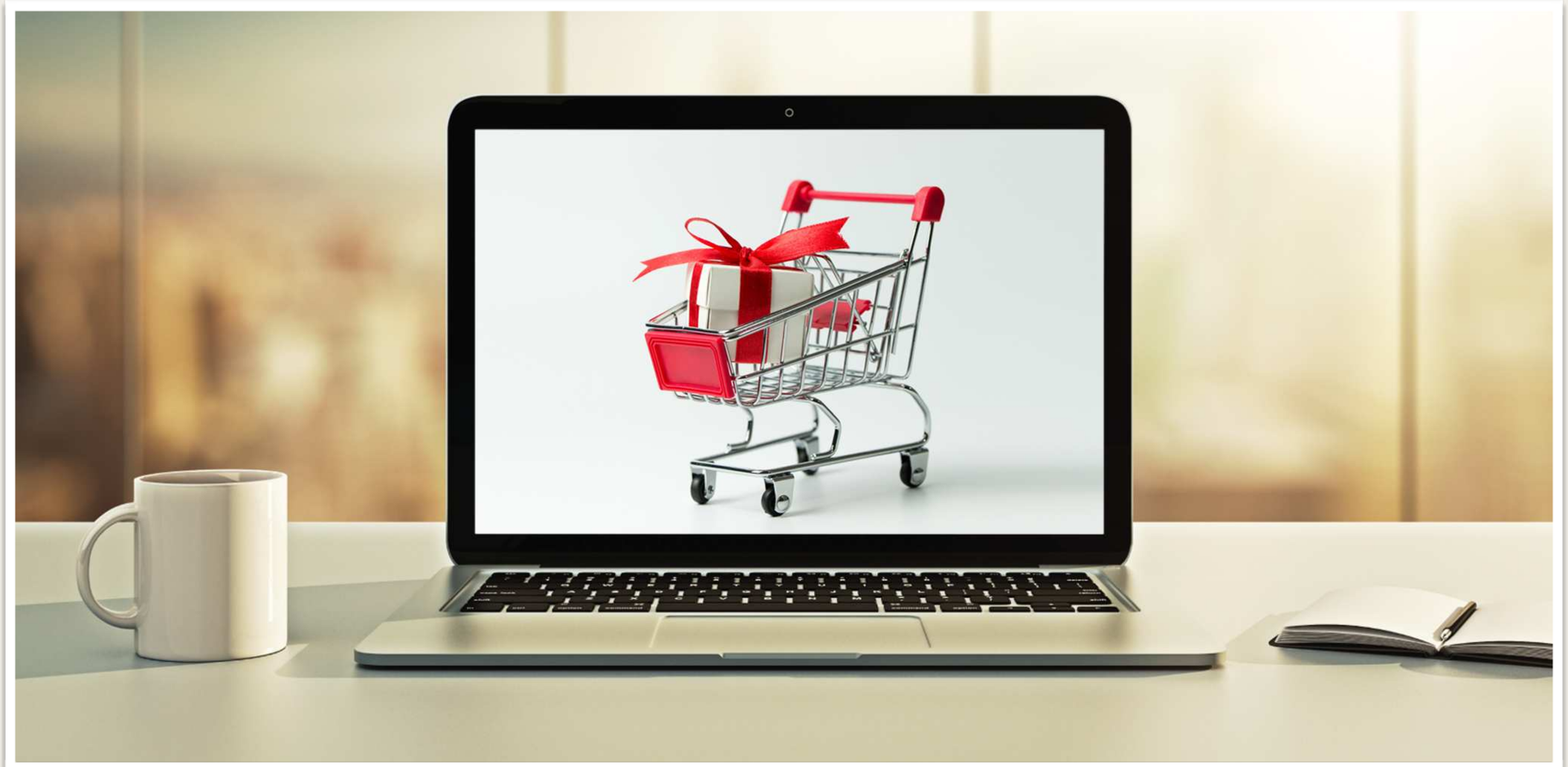
File: product-details.component.html

```
<div class="detail-section">
    <div class="container-fluid">

        <img src="{{ product.imageUrl }}" class="detail-img">

        <h3>{{ product.name }}</h3>
        <div class="price">{{ product.unitPrice | currency:'USD' }}</div>
        <a href="#" class="primary-btn">Add to cart</a>

        <hr>
        <h4>Description</h4>
        <p>{{ product.description }}</p>

        <a routerLink="/products" class="mt-5">Back to Product List</a>

    </div>
</div>
```



© luv2code LLC

Angular Language Service



Angular / JavaScript - Lenient

- Angular / JavaScript is VERY lenient
- Even though we are using TypeScript ... some bugs can still slip through!

© luv2code LLC

Angular / JavaScript - Lenient

- For example, our `Product` class does not have an `id` property defined
- But we can STILL assign data to it ... what????

http://localhost:8080/api/products/2


```
{
  "id" : 2,
  "sku" : "BOOK-TECH-1001",
  "name" : "Become a Guru in JavaScript",
  "description" : "Learn JavaScript at your own pace. The author explains how the
technology works in easy-to-understand language. This book includes working examples
that you can apply to your own projects. Purchase the book and get started today!",
  "unitPrice" : 20.99,
  "imageUrl" : "assets/images/products/books/book-luv2code-1001.png",
  "active" : true,
  "unitsInStock" : 100,
  "dateCreated" : "2020-01-11T16:58:24.000+0000",
  "lastUpdated" : null,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/products/2"
    },
    "product" : {
      "href" : "http://localhost:8080/api/products/2"
    },
    "category" : {
      "href" : "http://localhost:8080/api/products/2/category"
    }
  }
}
```


File: product.ts

```
export class Product {
    sku: string;
    name: string;
    description: string;
    unitPrice: number;
    imageUrl: string;
    active: boolean;
    unitsInStock: number;
    dateCreated: Date;
    lastUpdated: Date;
}
```

No `id` property

Our app still worked.
No errors in browser console.

© luv2code LLC

Angular / JavaScript - Lenient

- We also referenced the `Product id` property for our master / detail view

```
<!-- loop over the collection of products -->
<div *ngFor="let tempProduct of products" class="col-md-3">

    <div class="product-box">

        <a routerLink="/products/{{ tempProduct.id }}">
            <img src="{{ tempProduct.imageUrl }}" class="img-responsive">
        </a>

        <a routerLink="/products/{{ tempProduct.id }}">
            <h1>{{ tempProduct.name }}</h1>
        </a>
```

id property

id property

Our app still worked.
No errors in browser console.

© luv2code LLC

Angular Language Service

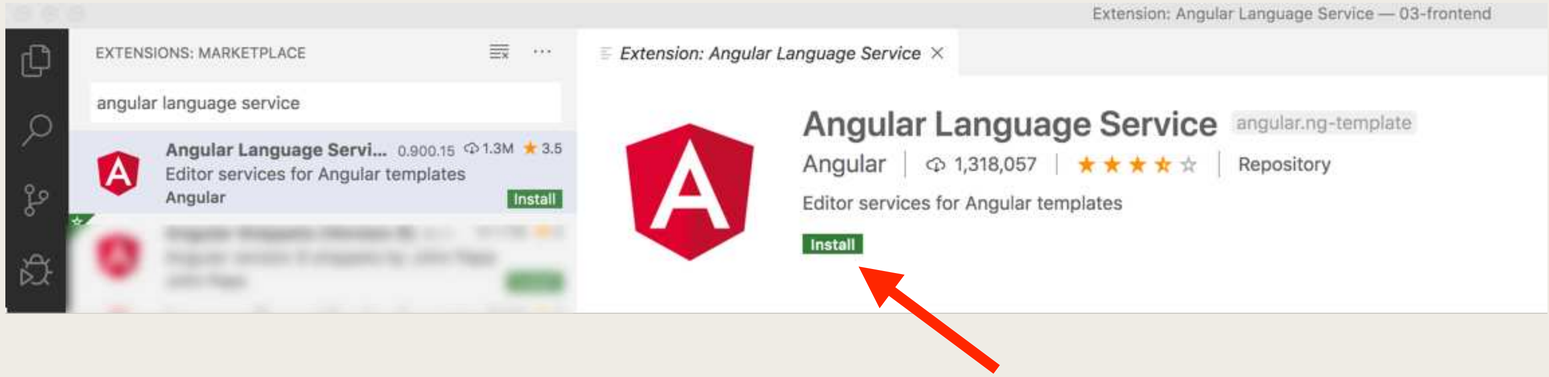
- Ideally, we would like to add more compile time checks for our application
- The Angular team provides the **Angular Language Service**
- Provides additional support for
  - Compilation errors, code completion, etc

https://angular.io/guide/language-service

© luv2code LLC

Angular Language Service

- Available as extensions for code editors: Visual Studio Code, WebStorm etc ...
- Install the extension in Visual Studio Code:
  - Menu option: **View > Extensions**
  - Search for: **Angular Language Service**
  - Click: **Install**

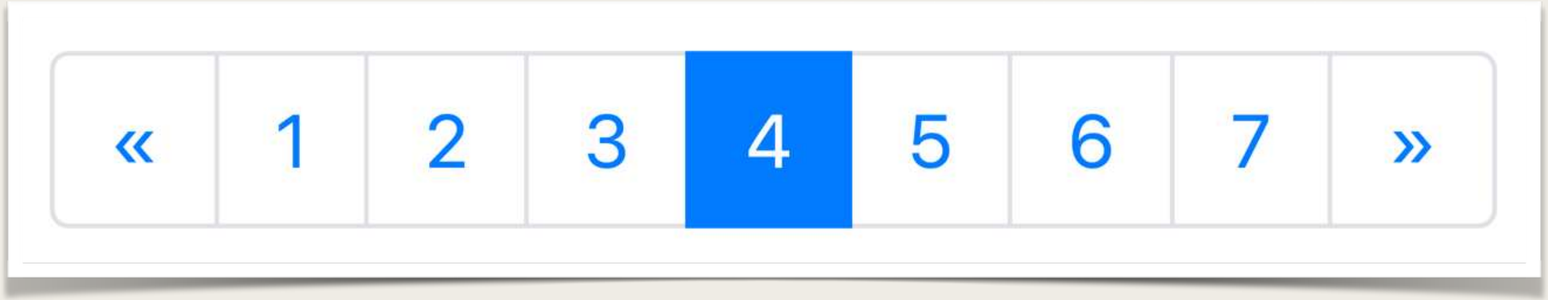


© luv2code LLC

Pagination

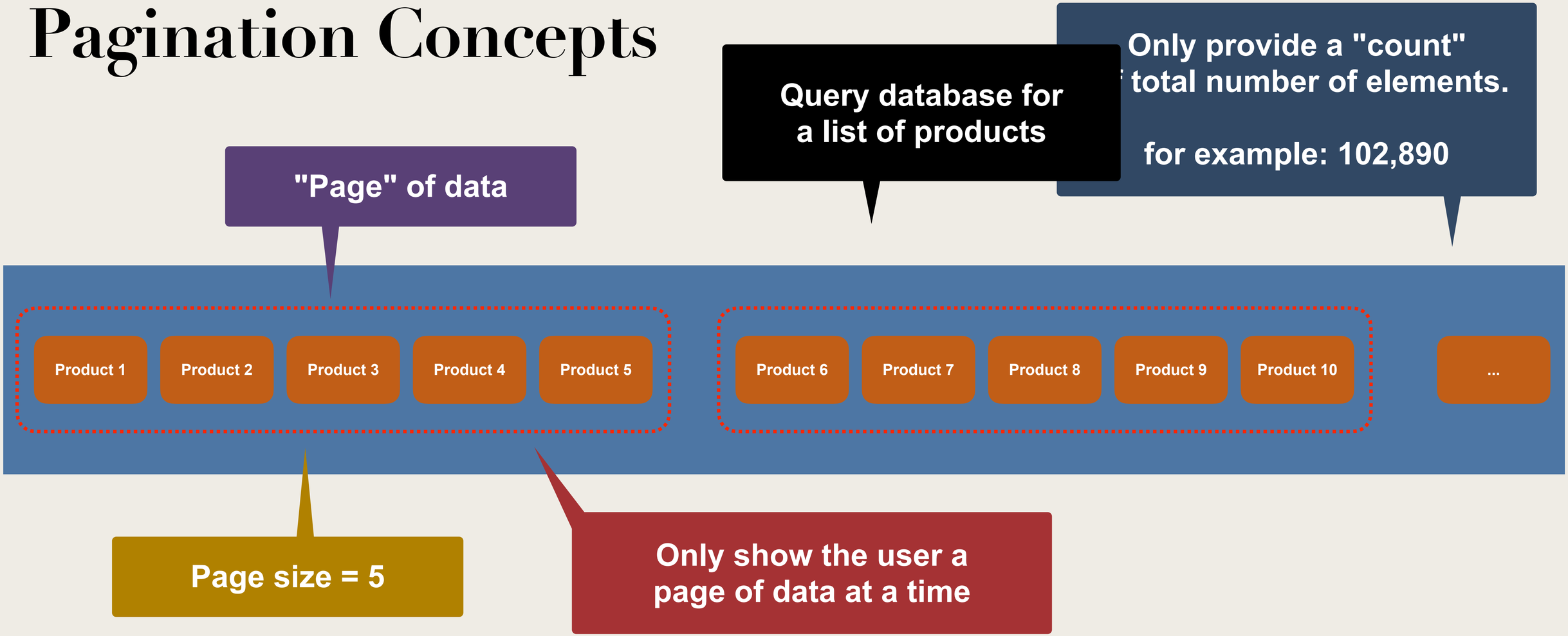
Pagination

- Pagination is useful for handling large amounts of data
- Show the users a small subset of data: "page" of data
- The user can click links to view other pages



© luv2code LLC

Pagination Concepts



© luv2code LLC

Pagination

- We will need pagination support on the back-end: Spring Boot
- Also need pagination support on the front-end: Angular

© luv2code LLC

Pagination

- Our Spring Boot backend uses Spring Data REST
- **Spring Data REST** provides pagination support out of the box ... yes!!!

www.luv2code.com/spring-data-rest-pagination

© luv2code LLC

Spring Data REST - Parameters

- By default, Spring Data REST returns: 20 elements
- We can customize this by passing in parameters

Parameter	Purpose
`page`	The page number to access. **0-based** ... defaults to 0.
`size`	The size of the page to return (items per page). Defaults to 20

© luv2code LLC

Spring Data REST - Pagination Examples

- Get the first page, with page size of 10

```
http://localhost:8080/api/products?page=0&size=10
```

Remember:
Pages are 0-based
First page is at position 0

- Get the second page, with page size of 10

```
http://localhost:8080/api/products?page=1&size=10
```

Second page is at position 1

© luv2code LLC

Spring Data REST - Response Meta Data

- The response meta data has valuable information

```
<< array of products >>
...
"page" : {
  "size" : 10,
  "totalElements" : 200,
  "totalPages" : 20,
  "number" : 0
}
```

Size of this page

Grand total of ALL elements in the database.
But we are NOT returning all of the elements.
Just the "count" for informational purposes only.

Total pages available

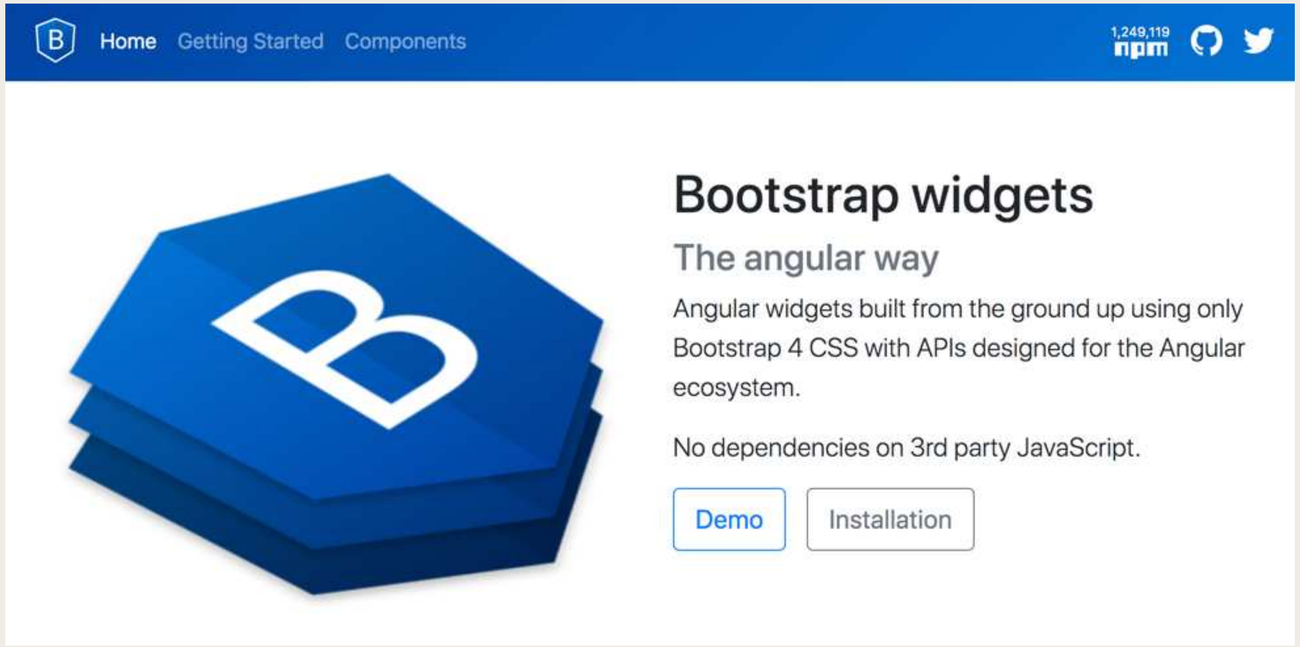
Current page number

© luv2code LLC

Pagination with Angular

- There are many pagination solutions available for Angular
- We will make use of a popular component framework: **ng-bootstrap**

https://ng-bootstrap.github.io/



© luv2code LLC

Components in ng-bootstrap

https://ng-bootstrap.github.io/

Components

- Accordion
- Alert
- Buttons
- Carousel
- Collapse
- **Datepicker**
- Dropdown
- Modal
- Nav
- Pagination
- Popover
- Progressbar
- Rating
- Table
- Tabset
- Timepicker
- Toast
- Tooltip
- Typeahead

Simple datepicker

Nov 2022

Mo	Tu	We	Th	Fr	Sa	Su
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

Progress bars with custom labels

</> Code StackBlitz

25

2 of 4

50%

Completed!

Basic pagination

Default pagination:

« 1 2 3 4 5 6 7 »

No direction links:

1 2 3 4 5 6 7

With boundary links:

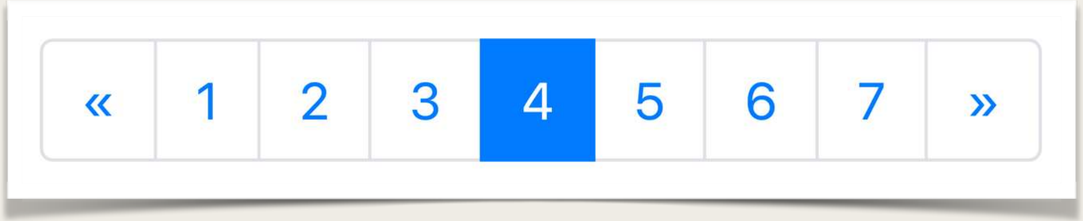
«« « 1 2 3 4 5 6 7 » »»

© luv2code LLC

Pagination component

https://ng-bootstrap.github.io/

Parameter	Purpose
`page`	The page number to access. 1-based ... defaults to 1
`pageSize`	The size of the page (items per page). Defaults to 10
`collectionSize`	The total number of items
`pageChange`	Event handler for page change events



© luv2code LLC

Pagination component

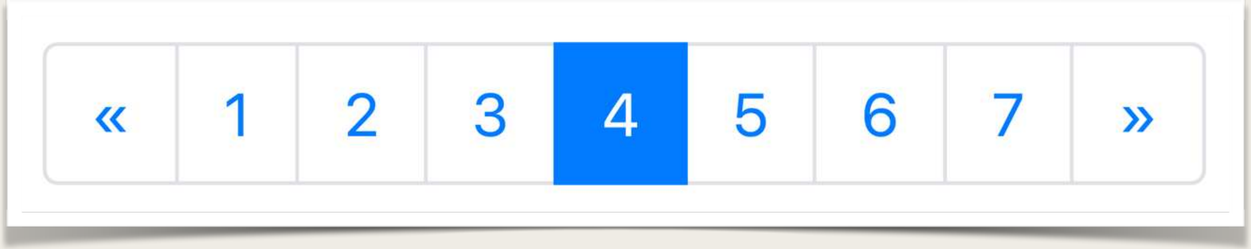
https://ng-bootstrap.github.io/

- Basic example of pagination component

```
<ngb-pagination [(page)]="thePageNumber"
                [pageSize]="thePageSize"
                [collectionSize]="theTotalElements"
                (pageChange)="listProducts()">
</ngb-pagination>
```

We will create code to combine this for our project

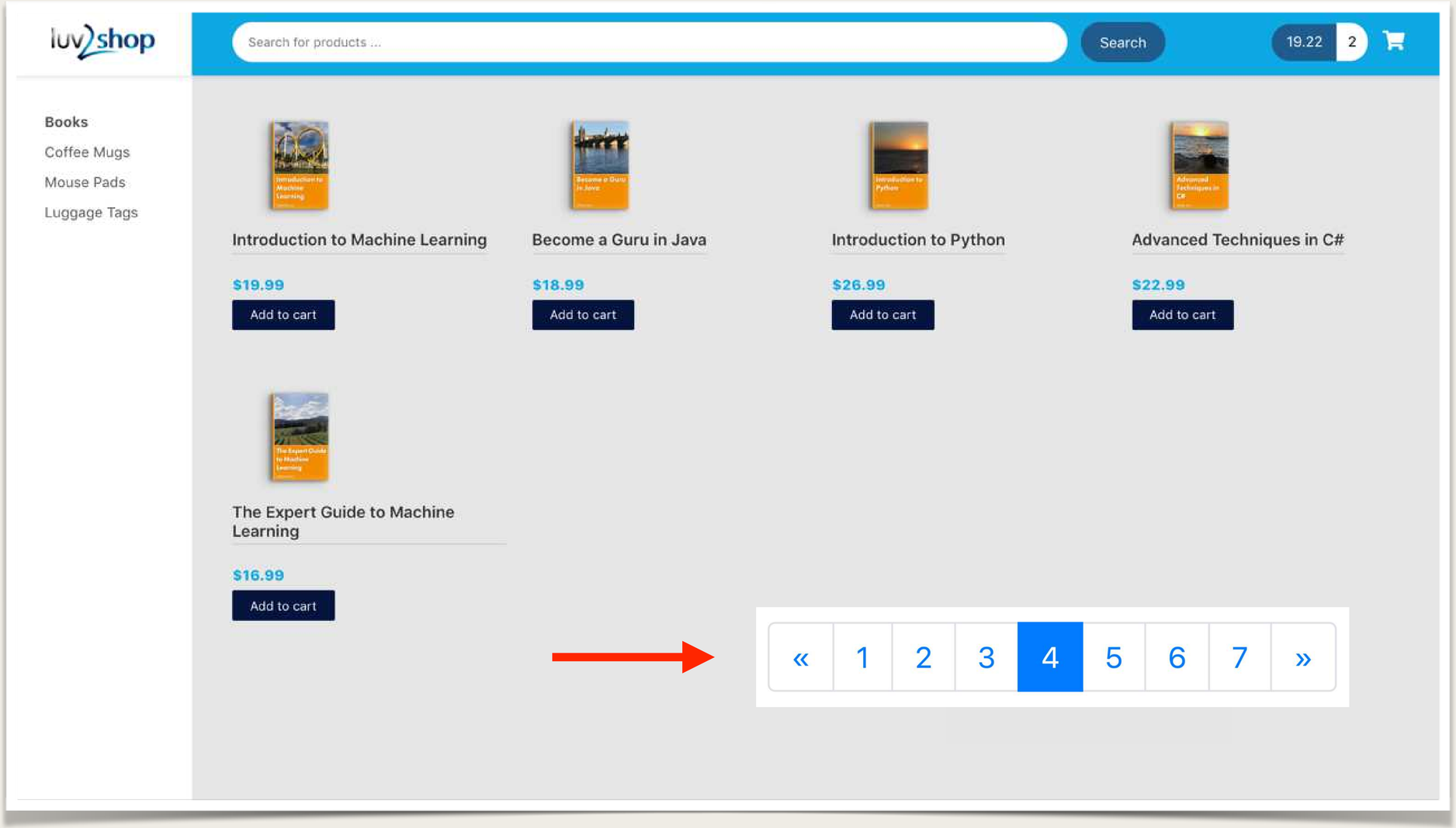
Component will generate links for pagination



© luv2code LLC

Pagination - Development Process

Pagination



© luv2code LLC

Development Process

Step-By-Step

1. Install ng-bootstrap
2. Refactor the interface for: GetResponseProducts
3. Add pagination support to ProductService
4. Update ProductListComponent to handle pagination
5. Enhance HTML template to use ng-bootstrap pagination component

© luv2code LLC

Step 1: Install ng-bootstrap

- Run the following commands in your Angular project directory

```
> ng add @angular/localize

> npm install @ng-bootstrap/ng-bootstrap
```

Dependency for Angular 9+

Installs ng-bootstrap

https://ng-bootstrap.github.io/#/getting-started

© luv2code LLC

Step 1: Install ng-bootstrap

- Import the module for ng-bootstrap

File: app.module.ts

```
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  …
 imports: [
    RouterModule.forRoot(routes),
    BrowserModule,
    HttpClientModule,
    NgbModule
  ],
  …
})
export class AppModule { }
```

ng-bootstrap module

© luv2code LLC

Step 2: Refactor the interface for GetResponseProducts

- We currently use the interface `GetResponseProducts`
- Maps JSON data from REST API to our TypeScript objects

JSON

```
{
  "_embedded" : {
    "products" : [ {
      "id" : 1,
      "sku" : "BOOK-TECH-1000",
      "name" : "Crash Course in Python",
    ...
    }, {
      "id" : 2,
      "sku" : "BOOK-TECH-1001",
      "name" : "Become a Guru in JavaScript",
    ...
   } ]
  }
}
```

```
interface GetResponseProducts {
  _embedded: {
    products: Product[];
  }
}
```


© luv2code LLC

Spring Data REST - Response Meta Data

- The response meta data also has valuable information for pagination

```
<< array of products >>
...
"page" : {
  "size" : 20,
  "totalElements" : 100,
  "totalPages" : 5,
  "number" : 0
}
```

Size of this page

Grand total of ALL elements in the database.
But we are NOT returning all of the elements.
Just the "count" for informational purposes only.

Total pages available

Current page number

© luv2code LLC

Step 2: Refactor the interface for GetResponseProducts

- Refactor the interface to support the pagination meta-data

JSON

```
{
  "_embedded" : {
  ...
 },
  "page" : {
    "size" : 20,
    "totalElements" : 100,
    "totalPages" : 5,
    "number" : 0
  }

}
```

```
interface GetResponseProducts {
  _embedded: {
    products: Product[];
  },
  page: {
    size: number,
    totalElements: number,
    totalPages: number,
    number: number
  }
}
```


© luv2code LLC

Step 3: Add pagination support to ProductService

File: product.service.ts

```
getProductListPaginate(thePage: number,
                       thePageSize: number,
                       theCategoryId: number): Observable<GetResponseProducts> {

  const url = `${this.baseUrl}/search/findByCategoryId`
    + `?id=${theCategoryId}&page=${thePage}&size=${thePageSize}`;

  return this.httpClient.get<GetResponseProducts>(url);
}
```

Pass in parameters for pagination

Spring Data REST supports pagination out of the box. Just send the parameters for page and size

© luv2code LLC

Step 4: Update ProductListComponent to handle pagination

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number = 1;
  thePageSize: number = 10;
  theTotalElements: number = 0;
  ...

  handleListProducts() {
    ...
    // now get the products for the given category id
    this.productService.getProductListPaginate(this.thePageNumber - 1,
                                               this.thePageSize,
                                               this.currentCategoryId).subscribe(this.processResult());
  }

  private processResult() {
    return data => {
      this.products = data._embedded.products;
      this.thePageNumber = data.page.number + 1;
      this.thePageSize = data.page.size;
      this.theTotalElements = data.page.totalElements;
    };
  }

}
```

When data arrives from product service ... then set properties based on the data

Everything on right-hand side of assignment is data from Spring Data REST JSON

© luv2code LLC

Step 4: Update ProductListComponent to handle pagination

File: product-list.component.ts

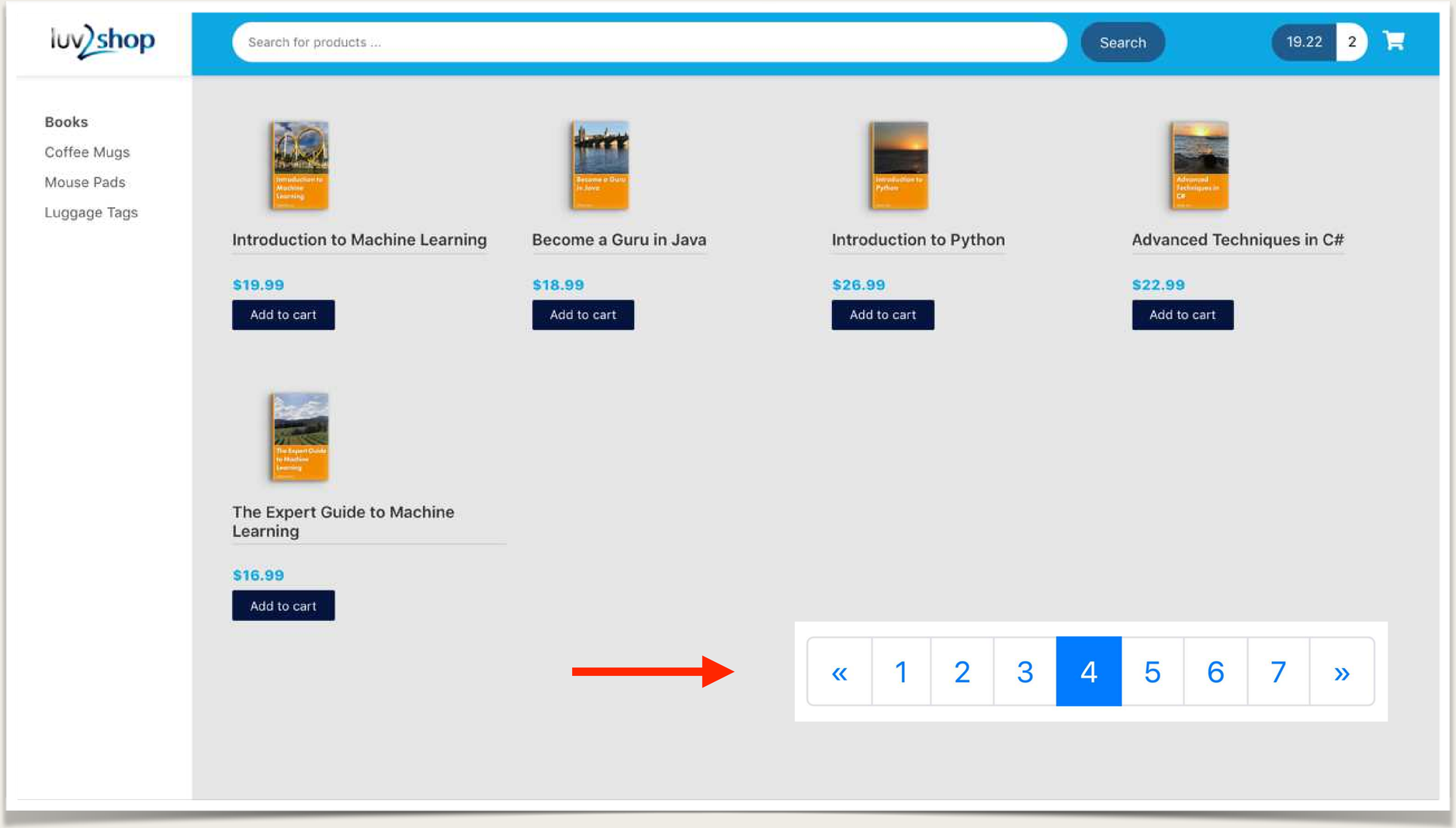
```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number = 1;
  thePageSize: number = 10;
  theTotalElements: number = 0;
  ...

  handleListProducts() {
    ...
    // now get the products for the given category id
    this.productService.getProductListPaginate(this.thePageNumber - 1,
                                               this.thePageSize,
                                               this.currentCategoryId).subscribe(this.processResult());
  }

  private processResult() {
    return data => {
      this.products = data._embedded.products;
      this.thePageNumber = data.page.number + 1;
      this.thePageSize = data.page.size;
      this.theTotalElements = data.page.totalElements;
    };
  }

}
```

Pagination component: pages are 1 based

Spring Data REST: pages are 0 based

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

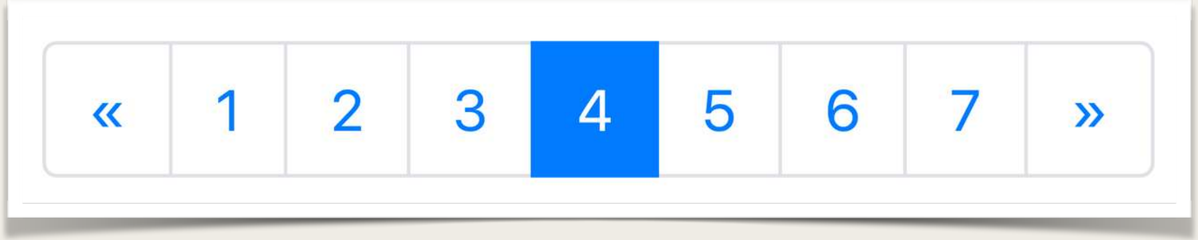


© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

Parameter	Purpose
`page`	The page number to access. 1-based ... defaults to 1
`pageSize`	The size of the page (items per page). Defaults to 10
`collectionSize`	The total number of items
`pageChange`	Event handler for page change events

https://ng-bootstrap.github.io/



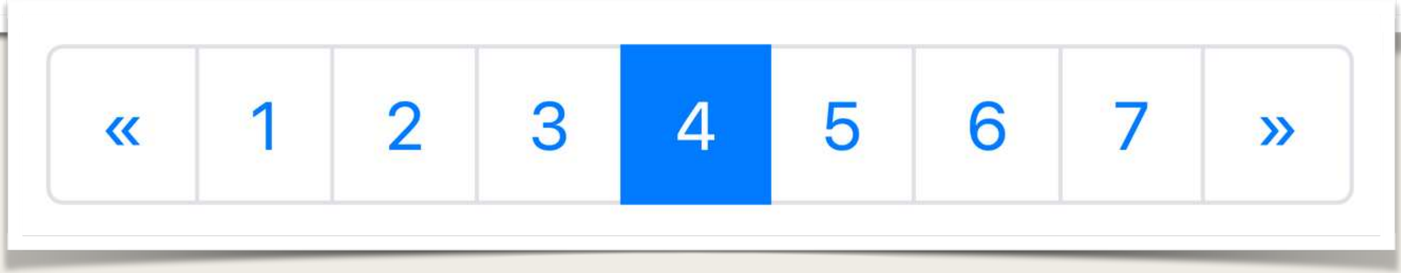
© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"
                   [pageSize]="thePageSize"
                   [collectionSize]="theTotalElements"
                   (pageChange)="listProducts()">
   </ngb-pagination>
...
</div>
```

Component will generate links for pagination



© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

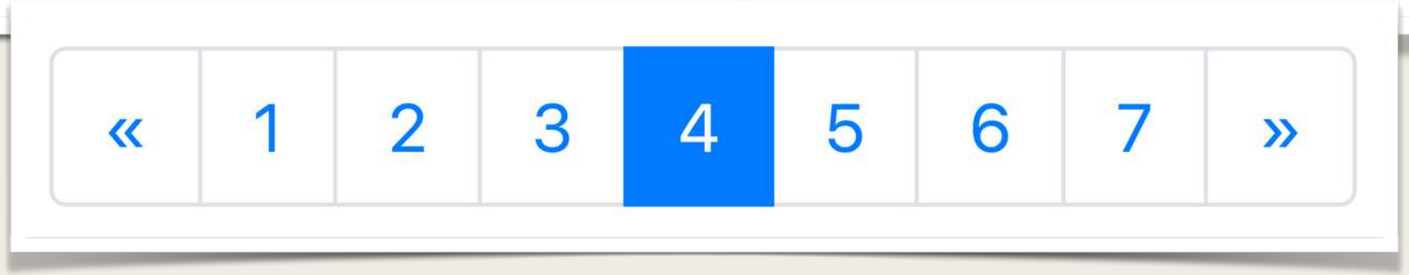
File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"



   </ngb-pagination>
...
</div>
```

Special Angular syntax for

Two-Way Data Binding

known as "banana in a box"



© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

Two-Way Data Binding

File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"



   </ngb-pagination>
...
</div>
```

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number;
  thePageSize: number;
  theTotalElements: number;
  ...
}
```

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

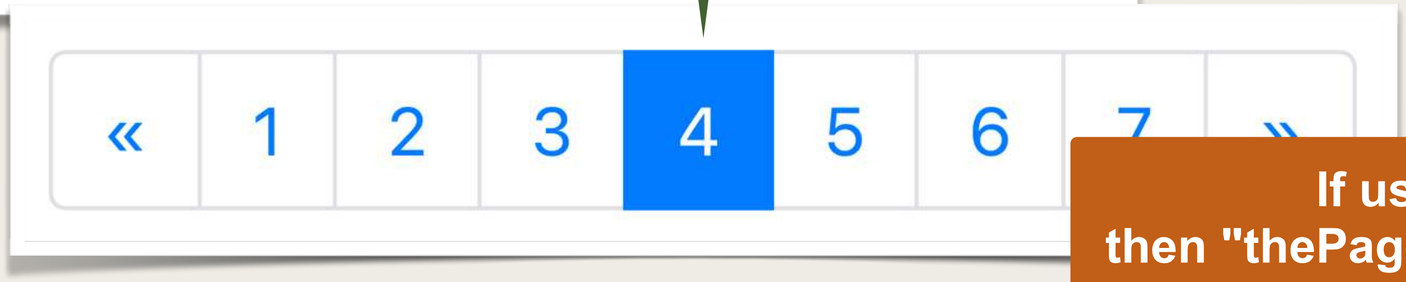
File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"



   </ngb-pagination>
...
</div>
```

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number;
  thePageSize: number;
  theTotalElements: number;
  ...
}
```

When user clicks a "page" navigation link



In our TypeScript component "thePageNumber" property is updated based on user action

If user clicks *4*, then "thePageNumber" is set to *4*

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"



   </ngb-pagination>
...
</div>
```

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number;
  thePageSize: number;
  theTotalElements: number;
  ...
}
```

Current page number

If our TypeScript code updates "thePageNumber" then UI component is updated automatically

Current page number is highlighted

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

Special Angular syntax for One-Way Data Binding

One-Way Data Binding

```
...
   <ngb-pagination [(page)]="thePageNumber"
                   [pageSize]="thePageSize"


   </ngb-pagination>
...
</div>
```

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number;
  thePageSize: number;
  theTotalElements: number;
  ...
}
```

If our TypeScript code updates "thePageSize" then UI component is updated automatically

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

One-Way Data Binding

File: product-list-grid.component.html

```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"
                   [pageSize]="thePageSize"
                   [collectionSize]="theTotalElements"

   </ngb-pagination>
...
</div>
```

File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number;
  thePageSize: number;
  theTotalElements: number;
  ...
}
```

If our TypeScript code updates "theTotalElements" then UI component is updated automatically

© luv2code LLC

Step 5: Enhance HTML template to use ng-bootstrap pagination component

File: product-list-grid.component.html

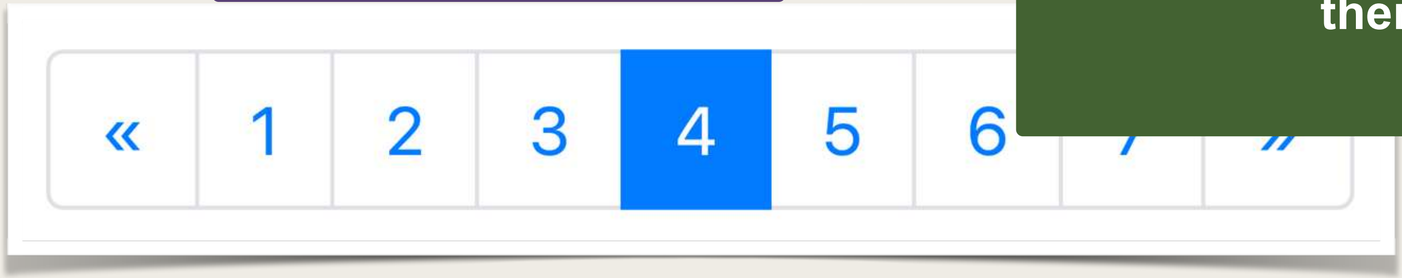
```
<div class="footer-pagination">
...
   <ngb-pagination [(page)]="thePageNumber"
                   [pageSize]="thePageSize"
                   [collectionSize]="theTotalElements"
                   (pageChange)="listProducts()">
  </ngb-pagination>
...
</div>
```

Event-binding:
pageChange event

File: product-list.component.ts

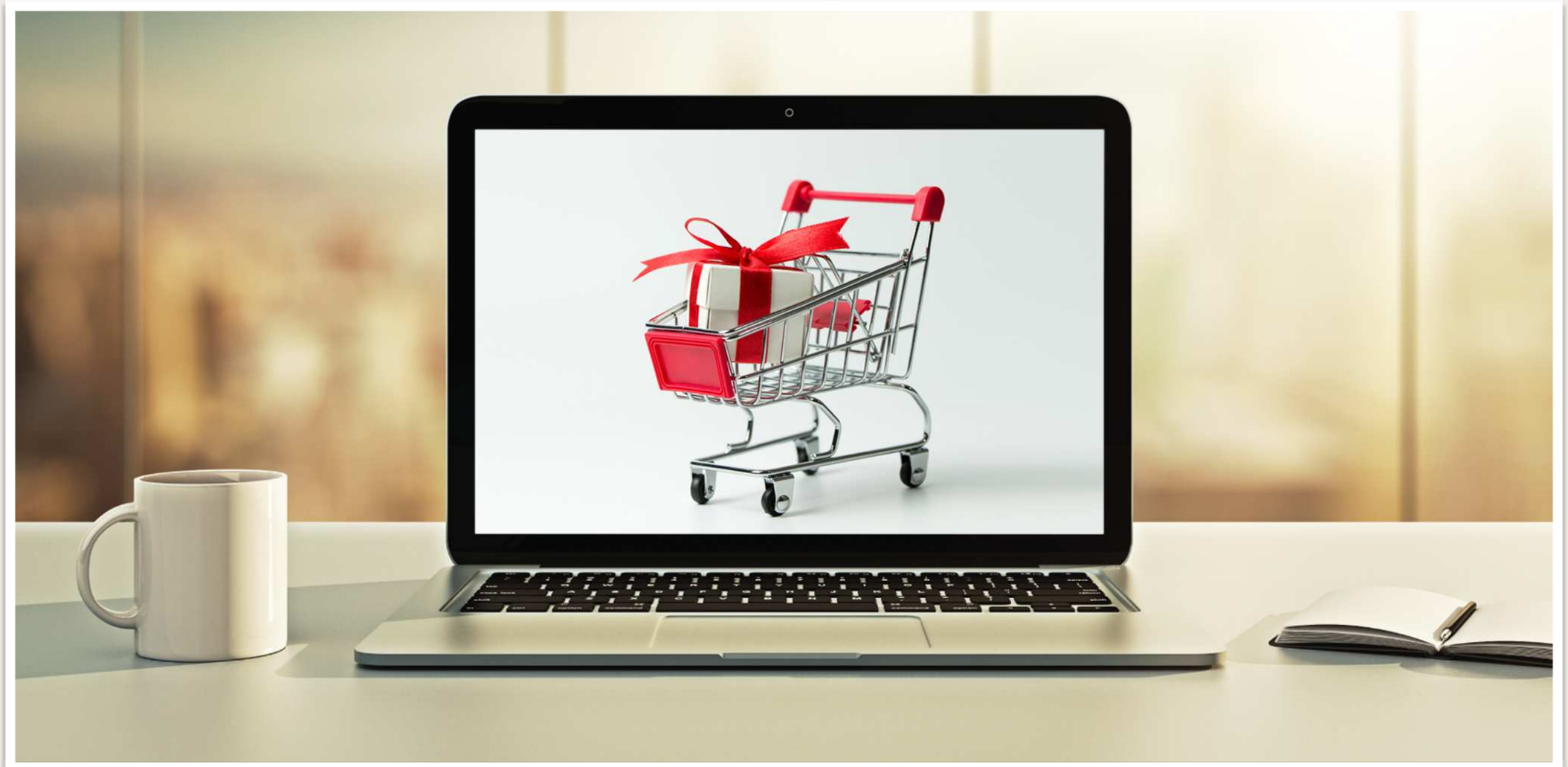
```
export class ProductListComponent implements OnInit {

  listProducts() {

    this.searchMode = ...;

    if (this.searchMode) {
      this.handleSearchProducts();
    }
    else {
      this.handleListProducts();
    }
  }
}
```

When user clicks a "page" navigation link
then call the method:
listProducts()

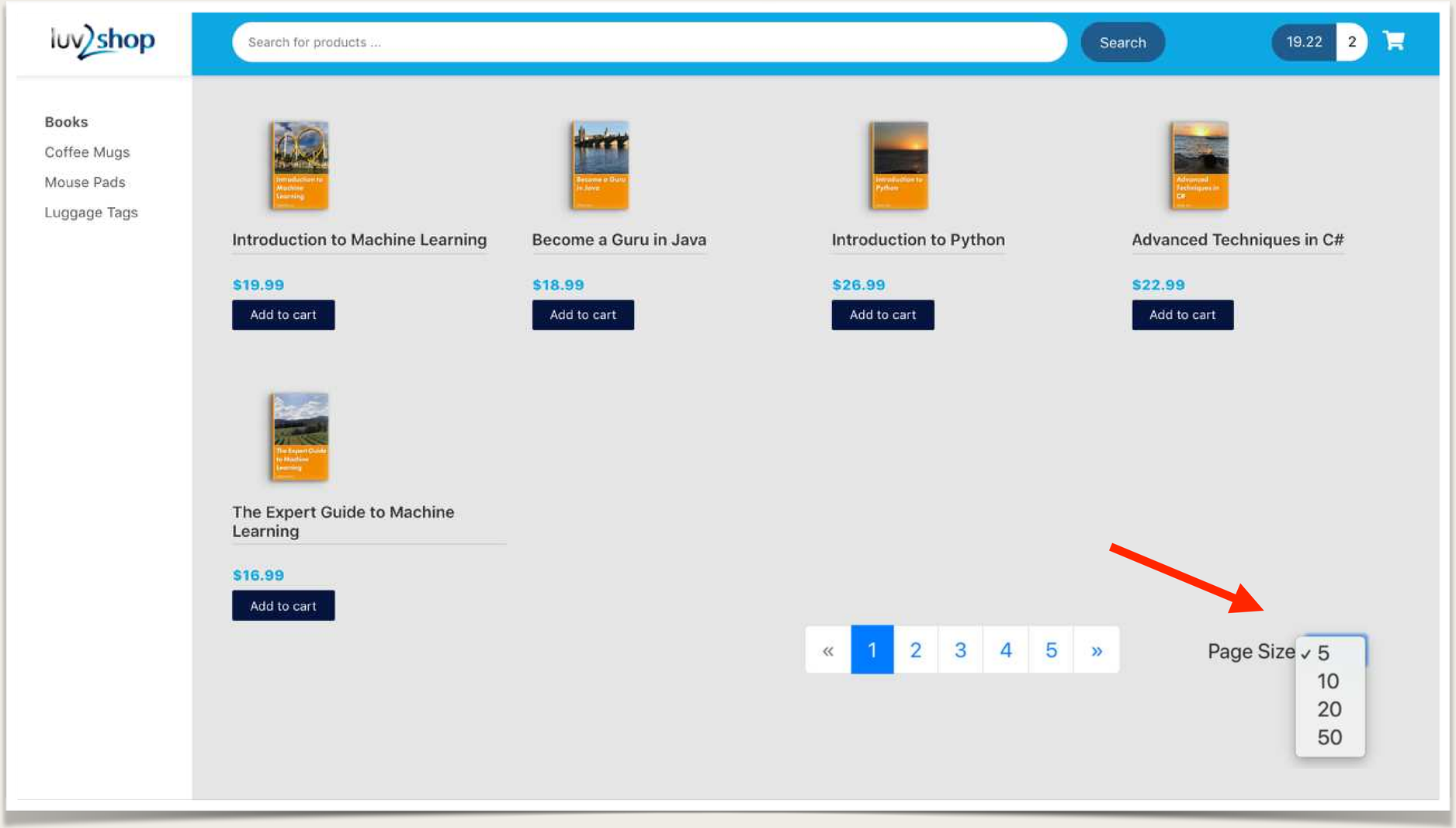


© luv2code LLC

Pagination - Select Page Size



Pagination



© luv2code LLC

Development Process

Step-By-Step

1. Add drop-down list for page size to HTML template
2. Update ProductListComponent for setting page size

© luv2code LLC

Step 1: Add drop-down list for page size to HTML template

When user selects a page, then call method: updatePageSize(...)

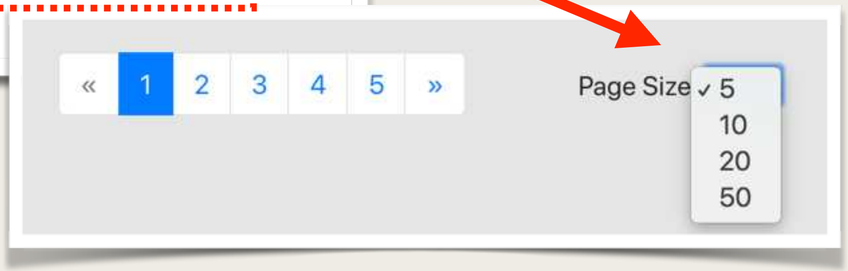
Event binding: on change

The value that is selected by the user

5, 10, 20 or 50

ponent.html

```
    <span class="mr-2">Page Size</span>

    <select (change)="updatePageSize($event.target.value)">
        <option selected="true">5</option>
        <option>10</option>
        <option>20</option>
        <option>50</option>
    </select>
...
```



© luv2code LLC

Step 2: Update ProductListComponent for setting page size

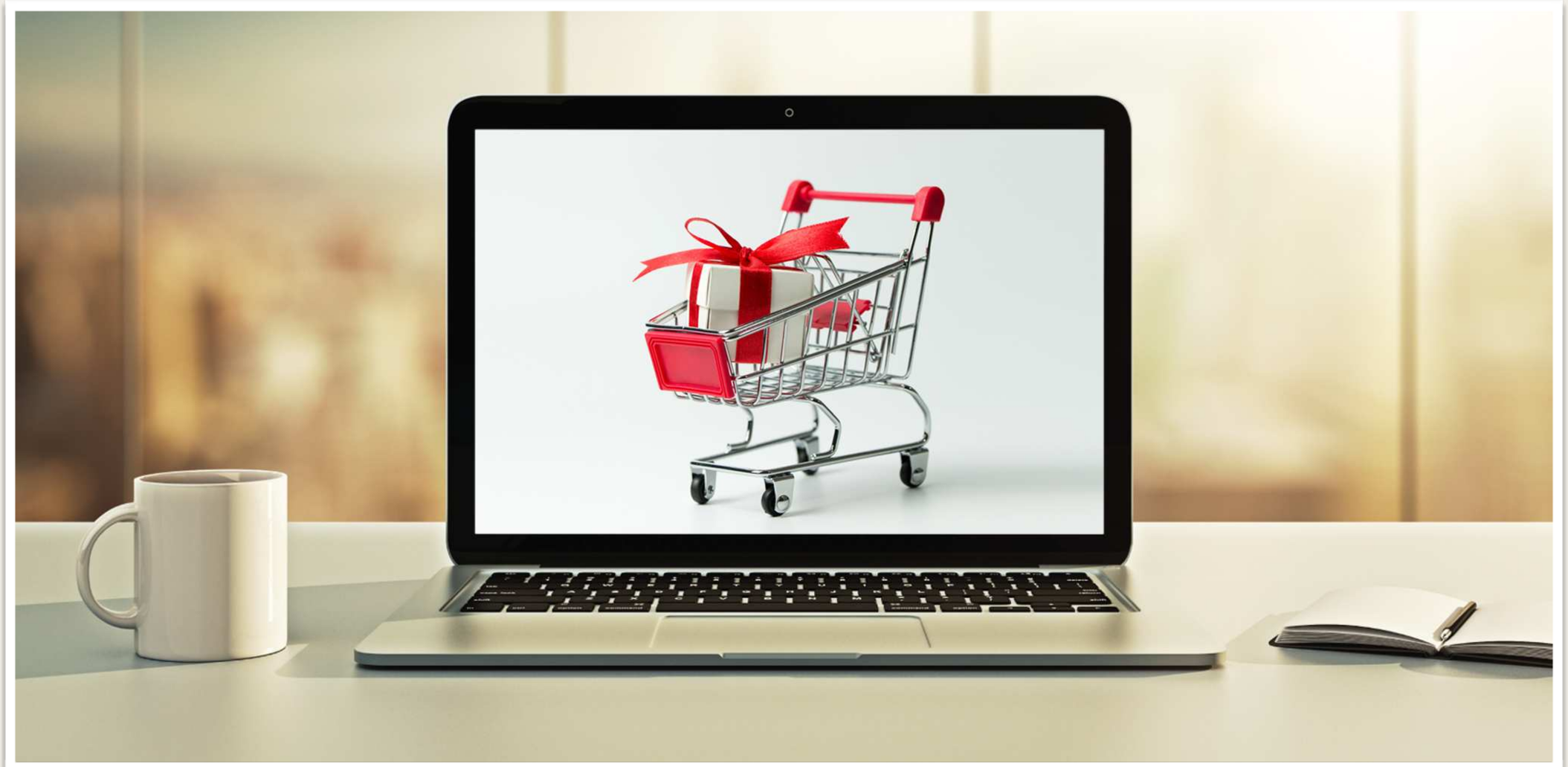
File: product-list.component.ts

```
export class ProductListComponent implements OnInit {

  // new properties for pagination
  thePageNumber: number = 1;
  thePageSize: number = 5;
  theTotalElements: number = 0;

 ...

  updatePageSize(pageSize: number) {
    this.thePageSize = pageSize;
    this.thePageNumber = 1;
    this.listProducts();
  }

}
```

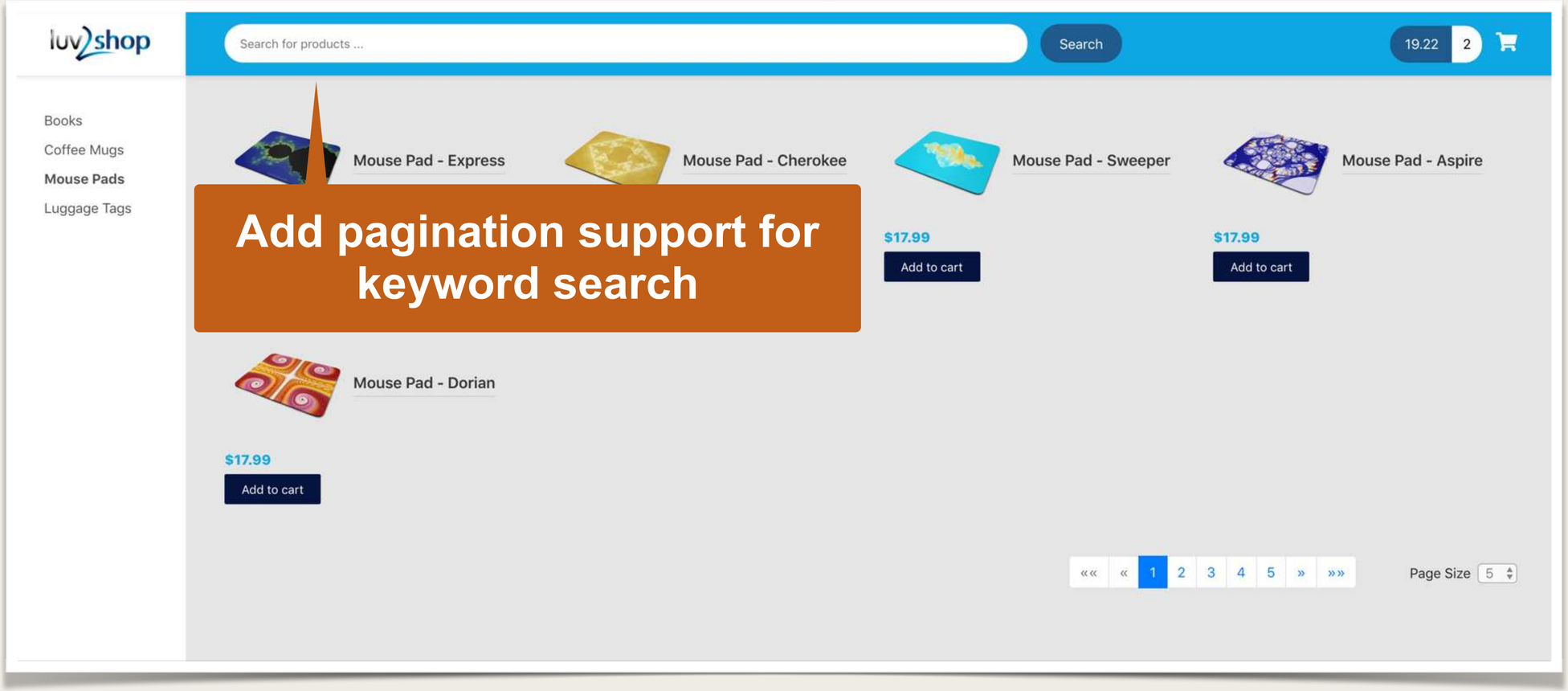
Set page size based on parameter value

© luv2code LLC

Pagination - Keyword Search



Pagination



© luv2code LLC

Development Process

Step-By-Step

1. Add pagination support to ProductService
2. Update ProductListComponent to handle pagination

© luv2code LLC

Step1: Add pagination support to ProductService

File: product.service.ts

Pass in parameters for pagination

```
searchProductsPaginate(thePage: number,
                       thePageSize: number,
                       theKeyword: string): Observable<GetResponseProducts> {

  const searchUrl = `${this.baseUrl}/search/findByNameContaining`
    + `?name=${theKeyword}&page=${thePage}&size=${thePageSize}`;

  return this.httpClient.get<GetResponseProducts>(searchUrl);
}
```

Spring Data REST supports pagination out of the box.
Just send the parameters for page and size

© luv2code LLC

Step 2: Update ProductListComponent to handle pagination

File: product-list.component.ts

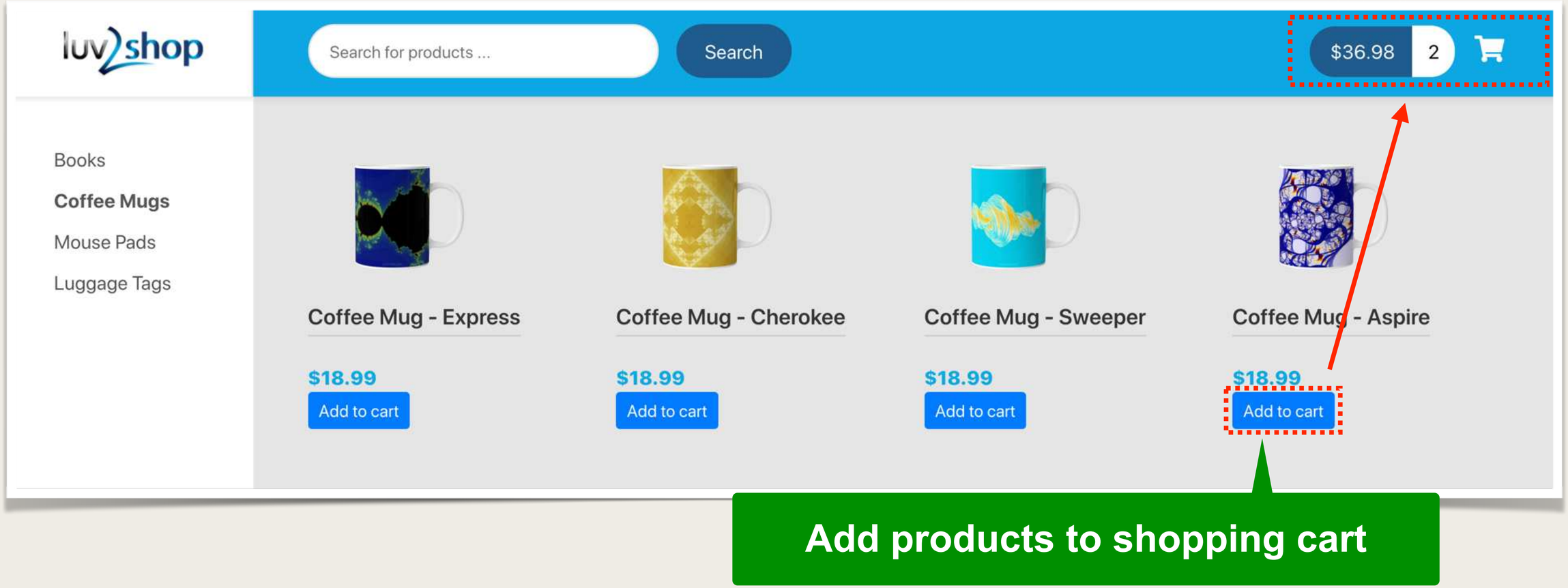
```
export class ProductListComponent implements OnInit {

  ...

  handleSearchProducts() {

    const theKeyword: string = this.route.snapshot.paramMap.get('keyword');
    ...
    // now get the products for the given category id
    this.productService.searchProductsPaginate(this.thePageNumber - 1,
                                               this.thePageSize,
                                               theKeyword).subscribe(this.processResult());
  }

  private processResult() {
    return data => {
      this.products = data._embedded.products;
      this.thePageNumber = data.page.number + 1;
      this.thePageSize = data.page.size;
      this.theTotalElements = data.page.totalElements;
    };
  }

}
```

Same method
from previous videos

© luv2code LLC

Add Products to Shopping Cart - Part 1

Add Products to Shopping Cart



Add products to shopping cart

© luv2code LLC

Overview of Entire Shopping Cart Process

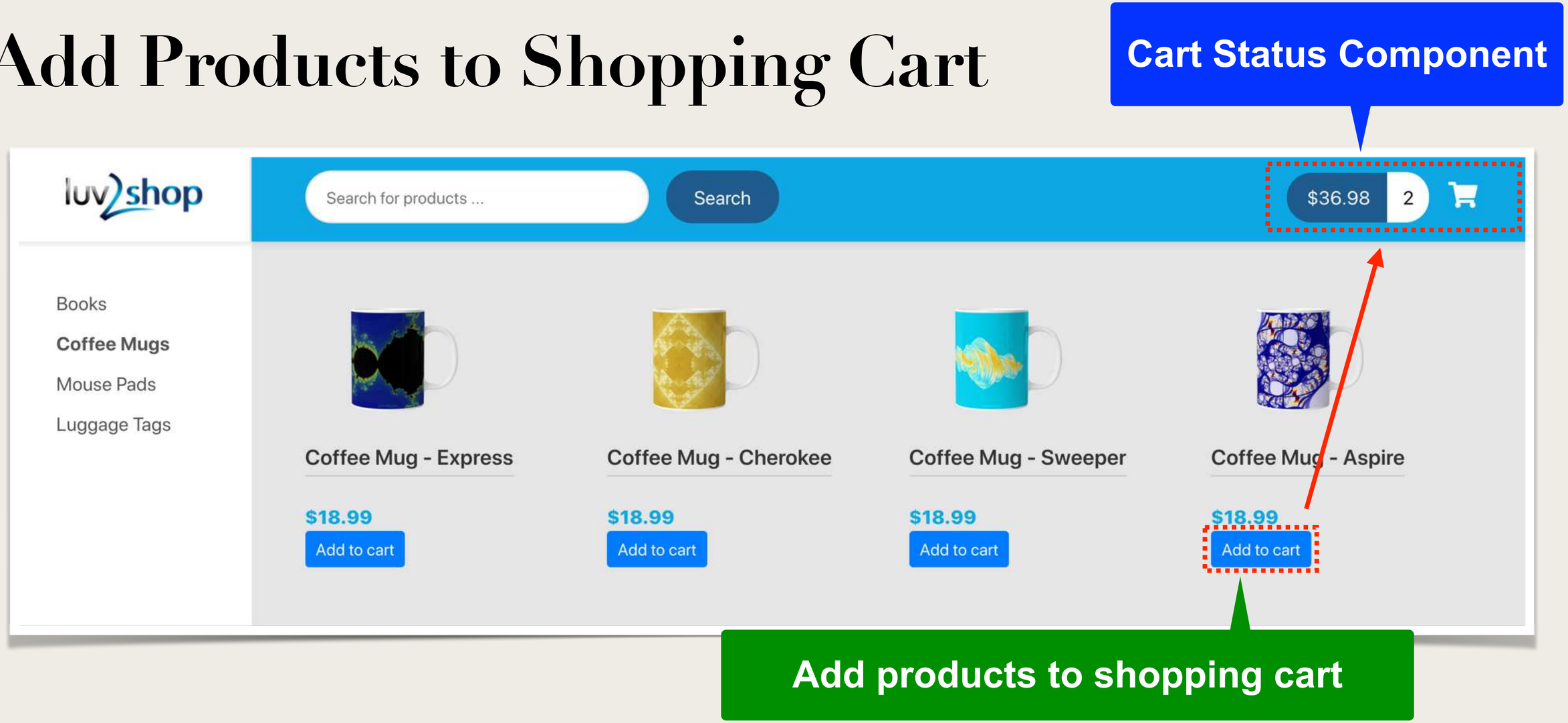
Step-By-Step

1. Cart Status Component: on main page, display total price and quantity
2. Cart Details Page: list the items in the cart
3. Cart Details Page: add / remove items
4. Checkout Button
5. Checkout Form

We will break this up into multiple videos

© luv2code LLC

Add Products to Shopping Cart



© luv2code LLC

Development Process - Part 1

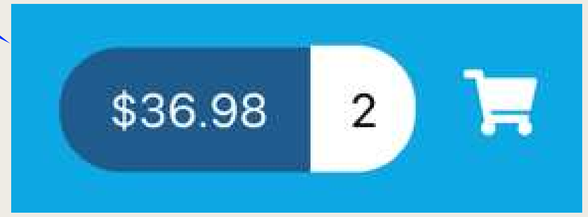
Step-By-Step

1. Create new component: CartStatusComponent
2. Add HTML template for CartStatusComponent
3. Add click handler for "Add to cart" button
4. Update ProductListComponent with click handler method

© luv2code LLC

Step 1: Create new component for Cart Status

Cart Status Component



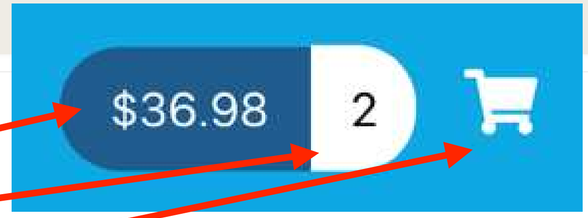

```
> ng generate component components/cart-status
```


© luv2code LLC

Step 2: Add HTML template for CartStatusComponent

File: cart-status.component.html

```
<div class="cart-area d-n">
    <a href="shopping-detail.html">
        <div class="total">$36.98
            <span>2</span>
        </div>
        <i class="fa fa-shopping-cart"></i>
    </a>
</div>
```



Font Awesome Shopping Cart Icon

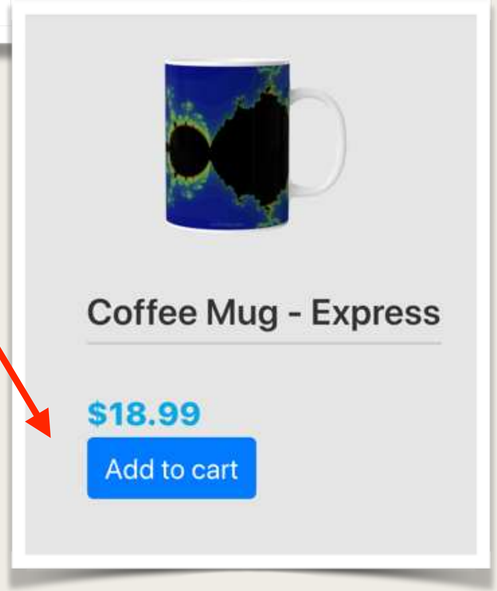
© luv2code LLC

Step 3: Add click handler for "Add to cart" button

File: product-list-grid.component.html

```
<!-- loop over the collection of products -->
<div *ngFor="let tempProduct of products" class="col-md-3">

    ...
    <button (click)="addToCart(tempProduct)" class="btn btn-primary btn-sm">Add to cart</button>

</div>
```

On click event
call the method: addToCart(...)



© luv2code LLC

Step 4: Update ProductListComponent with click handler method

File: product-list.component.ts

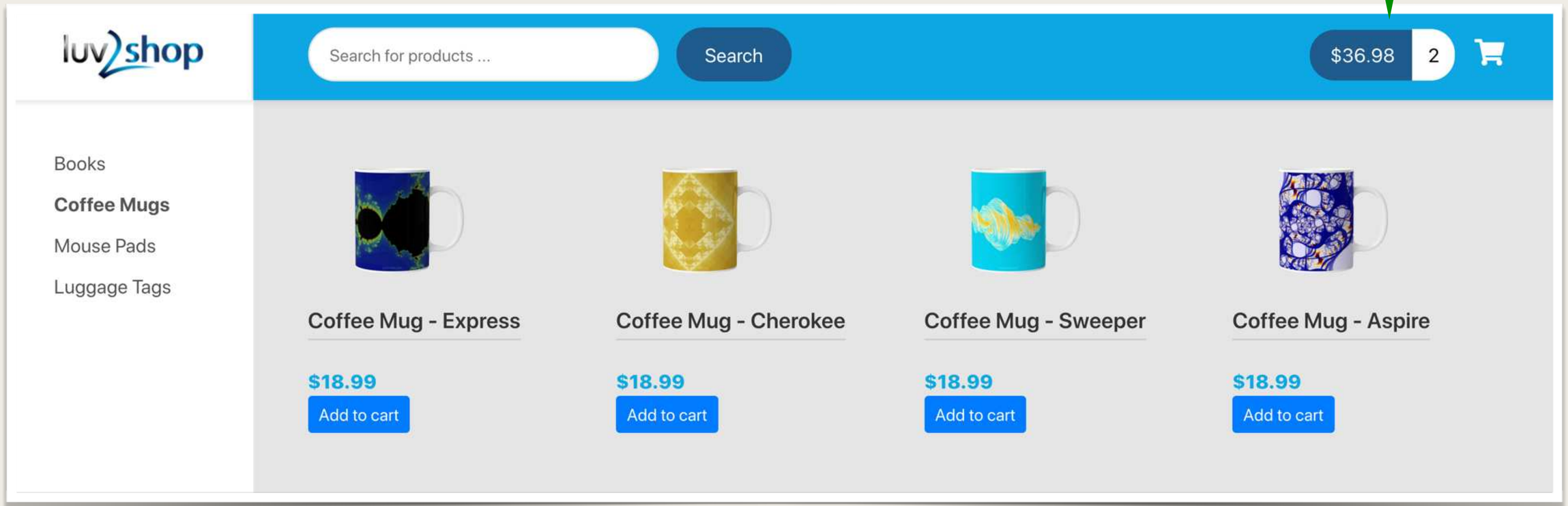
```
addToCart(theProduct: Product) {

  console.log(`Adding to cart: ${theProduct.name}, ${theProduct.unitPrice}`);

  // TODO ... do the real work

}
```


© luv2code LLC

Add Products to Shopping Cart - Part 2

Add Products to Shopping Cart

Update status



© luv2code LLC

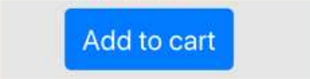
Development Process - Part 2

Step-By-Step

1. Create model class: CartItem
2. Develop CartService
3. Modify ProductListComponent to call CartService
4. Enhance CartStatusComponent to subscribe to CartService
5. Update CartStatusComponent HTML to display cart total price and quantity

© luv2code LLC

Application Interaction



CartStatusComponent

ProductListComponent

1. subscribe for events

4. update UI for total price and quantity

2. addToCart(...)

3. publish events to all subscribers

CartService

© luv2code LLC

Step 1: Create model class: CartItem

File: cart-item.ts

```
export class CartItem {

    id: string;
    name: string;
    imageUrl: string;
    unitPrice: number;
    quantity: number;

    constructor(product: Product) {
        this.id = product.id;
        this.name = product.name;
        this.imageUrl = product.imageUrl;
        this.unitPrice = product.unitPrice;
        this.quantity = 1;
    }
}
```

Contains essential fields of Product for use in the cart

Also has a field for quantity

We have "x" number of "widgets" in our cart

© luv2code LLC

Step 2: Develop CartService

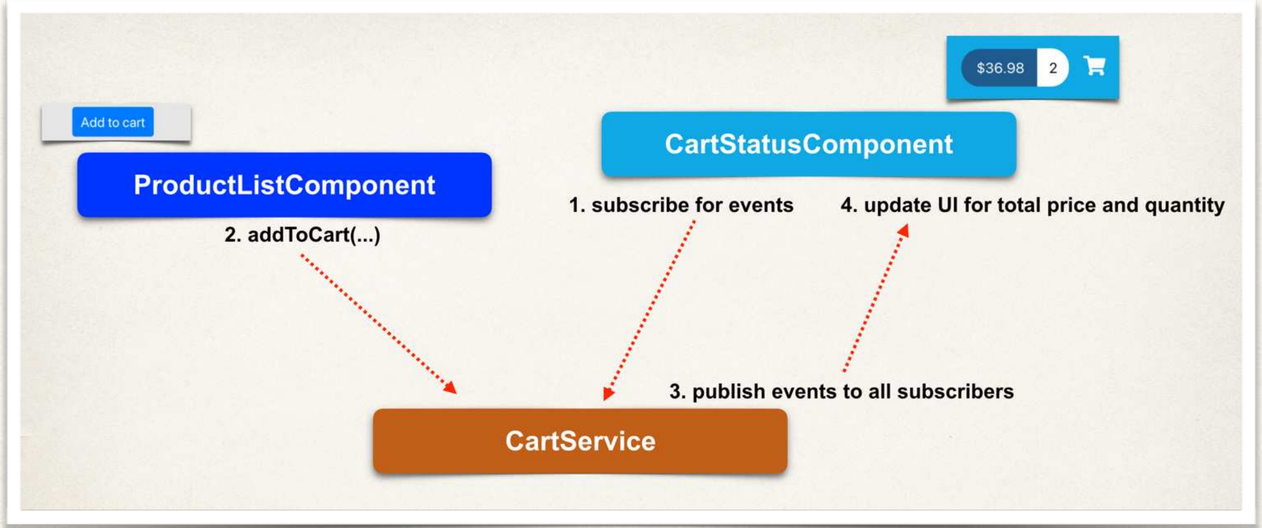
File: cart-service.ts

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new Subject<number>();
  totalQuantity: Subject<number> = new Subject<number>();

  ...
}
```

`Subject` is a subclass of `Observable`

We can use `Subject` to publish events in our code.

The event will be sent to all of the subscribers



© luv2code LLC

Step 2: Develop CartService

File: cart-service.ts

```
export class CartService {

  cartItems: CartItem[] = [];
  ...

  addToCart(theCartItem: CartItem) {

    // check if we already have the item in our cart
    let alreadyExistsInCart: boolean = false;
    let existingCartItem: CartItem = undefined;

    if (this.cartItems.length > 0) {
      // find the item in the cart based on item id

      for (let tempCartItem of this.cartItems) {
        if (tempCartItem.id === theCartItem.id) {
          existingCartItem = tempCartItem;
          break;
        }
      }

      // check if we found it
      alreadyExistsInCart = (existingCartItem != undefined)
    }
```

```
    if (alreadyExistsInCart) {
      // increment the quantity
      existingCartItem.quantity++;
    }
    else {
      // just add the item to the array
      this.cartItems.push(theCartItem);
    }

    // compute cart quantity and cart total
    this.computeCartTotals();
  }

  ...
}
```


© luv2code LLC

Step 2: Develop CartService

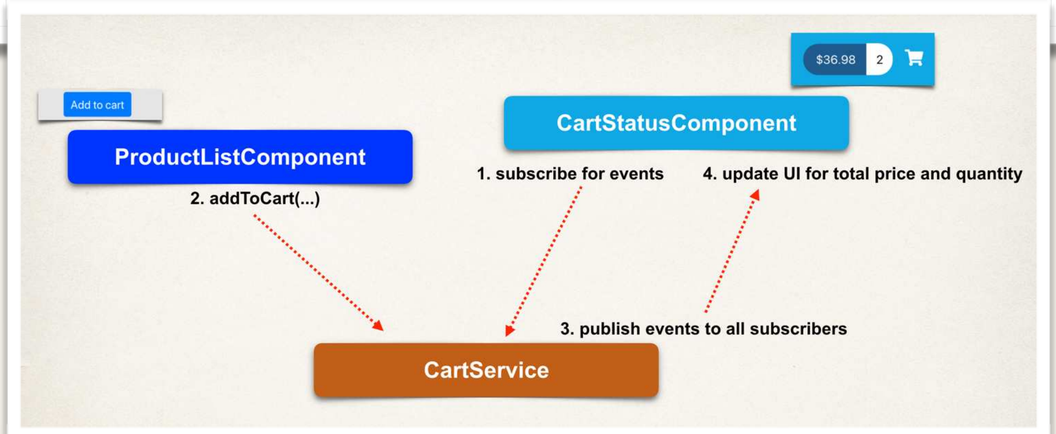
File: cart-service.ts

```
computeCartTotals() {

  let totalPriceValue: number = 0;
  let totalQuantityValue: number = 0;

  for (let currentCartItem of this.cartItems) {
    totalPriceValue += currentCartItem.quantity * currentCartItem.unitPrice;
    totalQuantityValue += currentCartItem.quantity;
  }

  // publish the new values ... all subscribers will receive the new data
  this.totalPrice.next(totalPriceValue);
  this.totalQuantity.next(totalQuantityValue);
}
```

Compute totals

This will publish events to all subscribers

one event for totalPrice
one event for totalQuantity



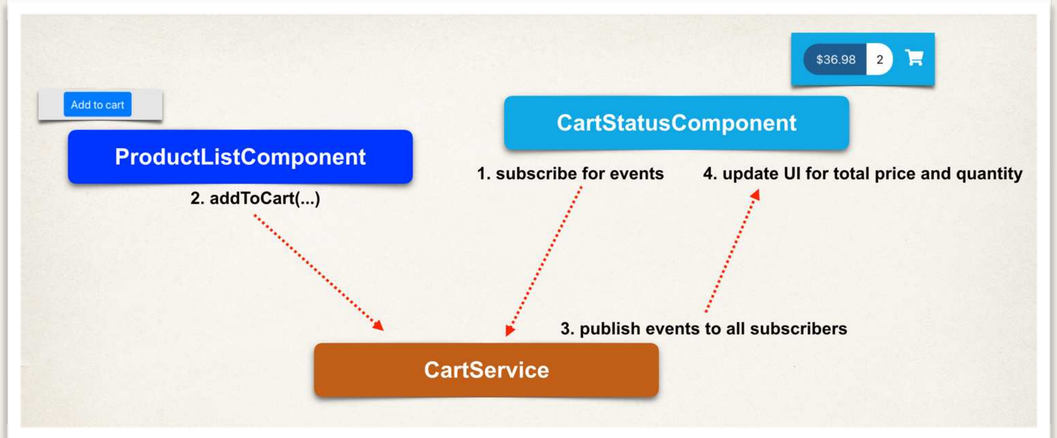
© luv2code LLC

Step 3: Modify ProductListComponent to call CartService

File: product-list.component.ts

```
addToCart(theProduct: Product) {
  console.log(`Adding to cart: ${theProduct.name}, ${theProduct.unitPrice}`);

  const theCartItem = new CartItem(theProduct);

  // TODO ... do the real work
  this.cartService.addToCart(theCartItem);
}
```

Call CartService

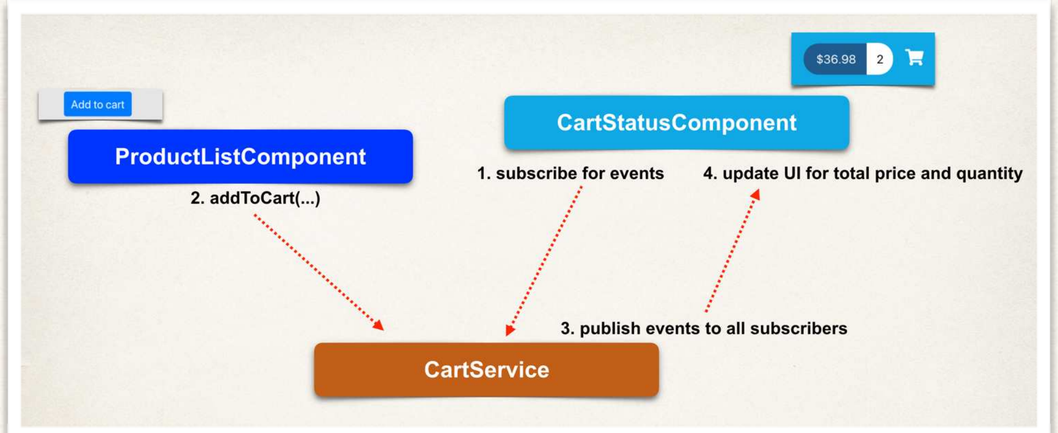


© luv2code LLC

Step 4: Enhance CartStatusComponent to subscribe to CartService

File: cart-status.component.ts

```
export class CartStatusComponent implements OnInit {

  totalPrice: number = 0.00;
  totalQuantity: number = 0;

  constructor(private cartService: CartService) { }

  ngOnInit(): void {
    this.updateCartStatus();
  }

  updateCartStatus() {

    // subscribe to the cart status totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart status totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );
  }
}
```



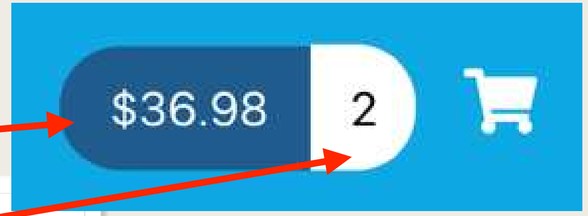
When new events are received, make the assignments to update UI

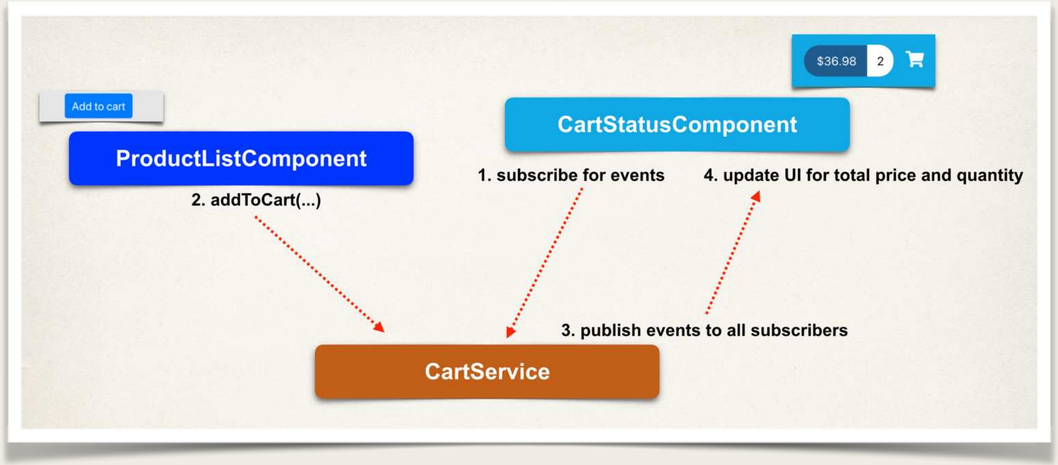
© luv2code LLC

Step 5: Update CartStatusComponent HTML to display cart total price and quantity

File: cart-status.component.html

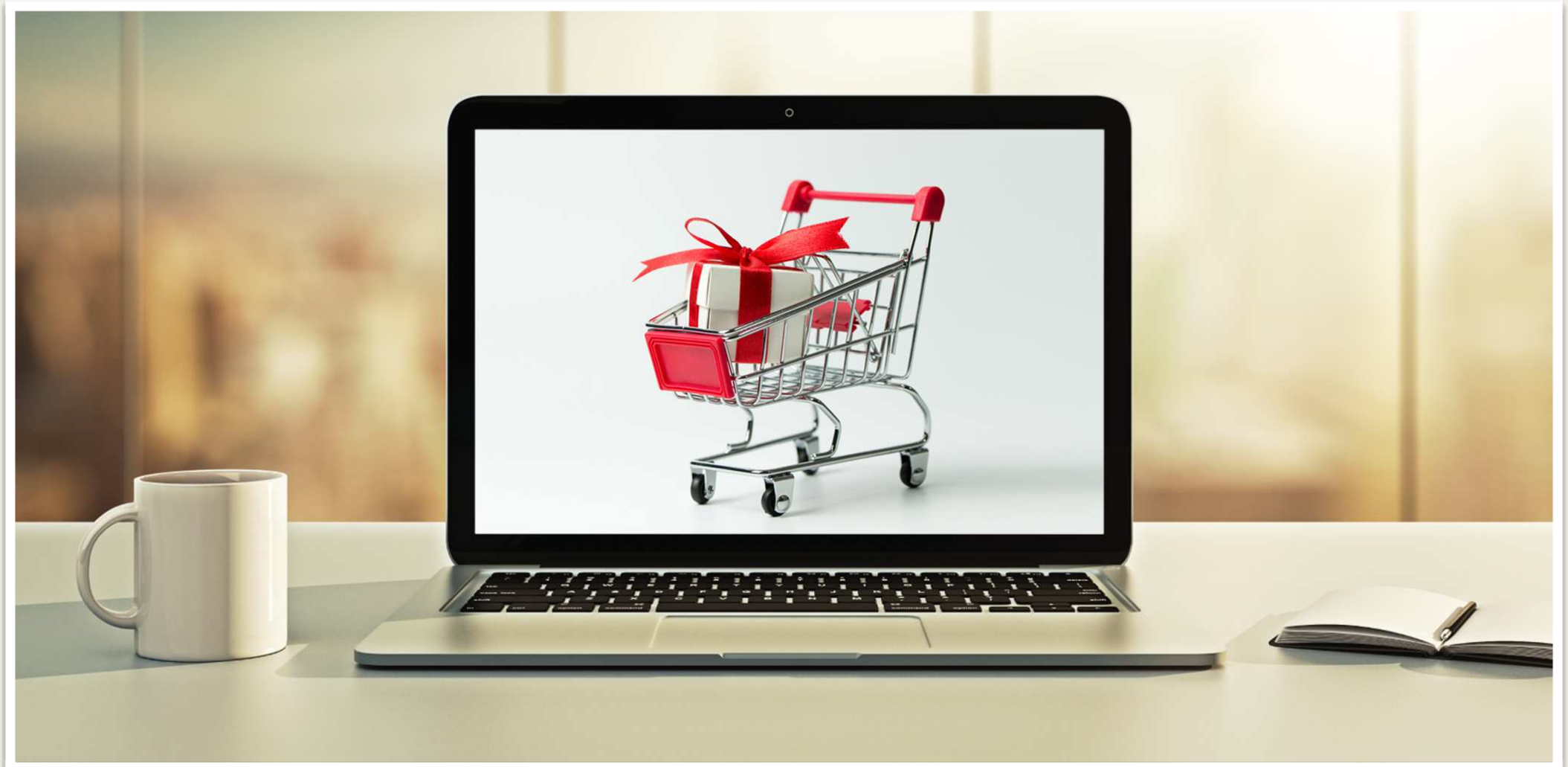
```
<div class="total">{{ totalPrice | currency: 'USD' }}
        <span>{{ totalQuantity }}</span>
</div>
```





© luv2code LLC

Refactor Cart Service



Currently: Basic code for finding item in cart

File: cart.service.ts

```
// check if we already have the item in our cart
let alreadyExistsInCart: boolean = false;
let existingCartItem: CartItem = undefined;

if (this.cartItems.length > 0) {
  // find the item in the cart based on item id

  for (let tempCartItem of this.cartItems) {
    if (tempCartItem.id === theCartItem.id) {
      existingCartItem = tempCartItem;
      break;
    }
  }

  // check if we found it
  alreadyExistsInCart = (existingCartItem != undefined)
}
```

Let's refactor this code

© luv2code LLC

Array.find(...)

- Method returns the first element in an array that passes a given test

```
existingCartItem = this.cartItems.find( tempCartItem => tempCartItem.id === theCartItem.id );
```

- Executes the test for each element in the array until the test passes
- If test passes, then returns the first element in the array that passed
- If test fails for ALL elements in the array, then returns `undefined`

© luv2code LLC

Array.find(...)

Current array element

Test Conditional

```
for (let tempCartItem of this.cartItems) {

  if (tempCartItem.id === theCartItem.id) {
     existingCartItem = tempCartItem;
     break;
  }

}
```

Returns first element that passes else returns `undefined`

Executes test for each element in the array until test passes

```
existingCartItem = this.cartItems.find( tempCartItem => tempCartItem.id === theCartItem.id );
```

Current array element

Test Conditional

© luv2code LLC

Refactored: Before and After

BEFORE

```
// check if we already have the item in our cart
let alreadyExistsInCart: boolean = false;
let existingCartItem: CartItem = undefined;

if (this.cartItems.length > 0) {
  // find the item in the cart based on item id

  for (let tempCartItem of this.cartItems) {
    if (tempCartItem.id === theCartItem.id) {
      existingCartItem = tempCartItem;
      break;
    }
  }

  // check if we found it
  alreadyExistsInCart = (existingCartItem != undefined)
}
```

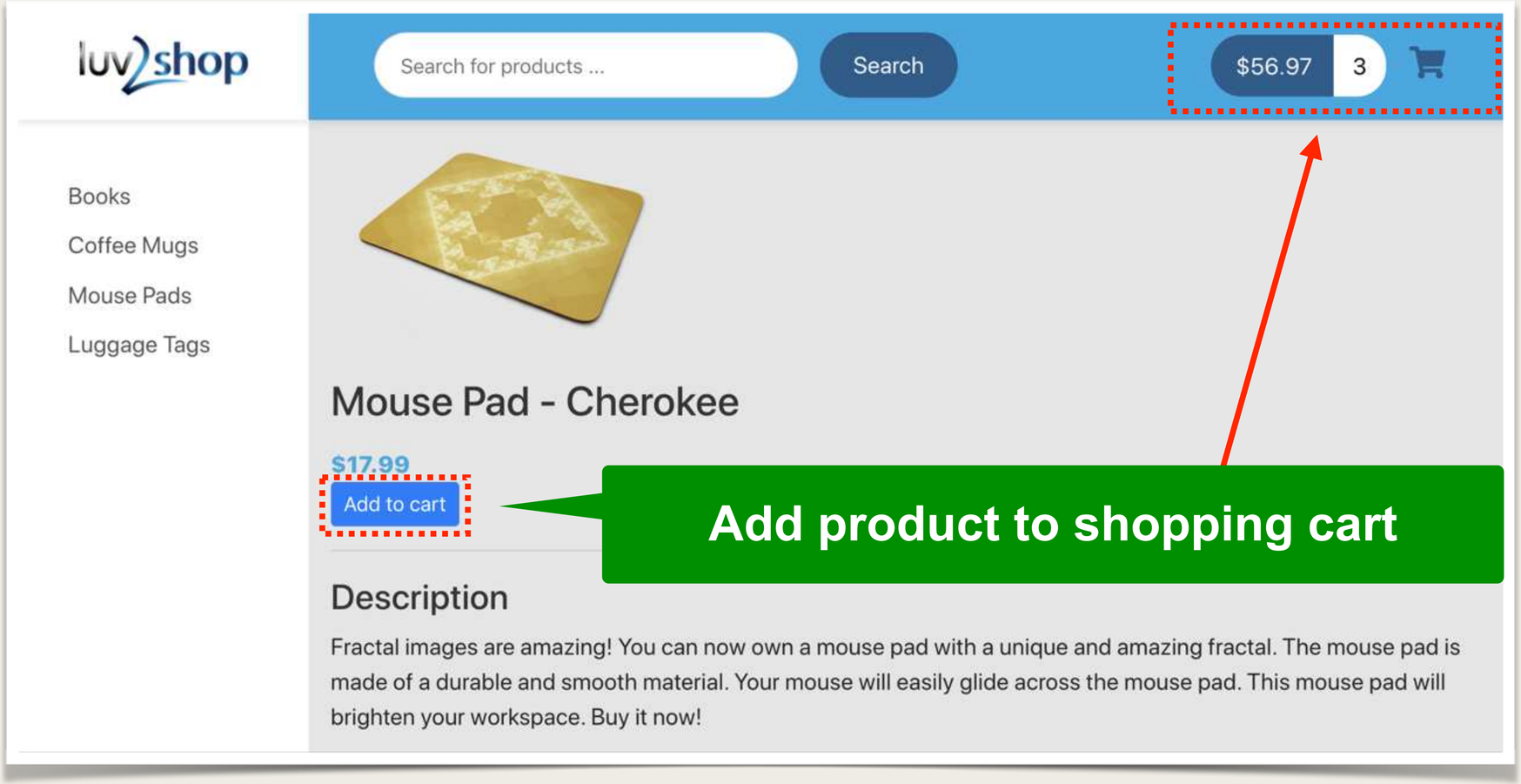
AFTER

```
// check if we already have the item in our cart
let alreadyExistsInCart: boolean = false;
let existingCartItem: CartItem = undefined;

if (this.cartItems.length > 0) {
  // find the item in the cart based on item id

  existingCartItem = this.cartItems.find( tempCartItem => tempCartItem.id === theCartItem.id );

  // check if we found it
  alreadyExistsInCart = (existingCartItem != undefined)
}
```


© luv2code LLC

Add Products to Cart - Details View

Add Products to Cart - Details View



© luv2code LLC

Development Process

Step-By-Step

1. Add click handler for "Add to cart" button on `product-details.component.html`

2. Update ProductDetailsComponent with click handler method

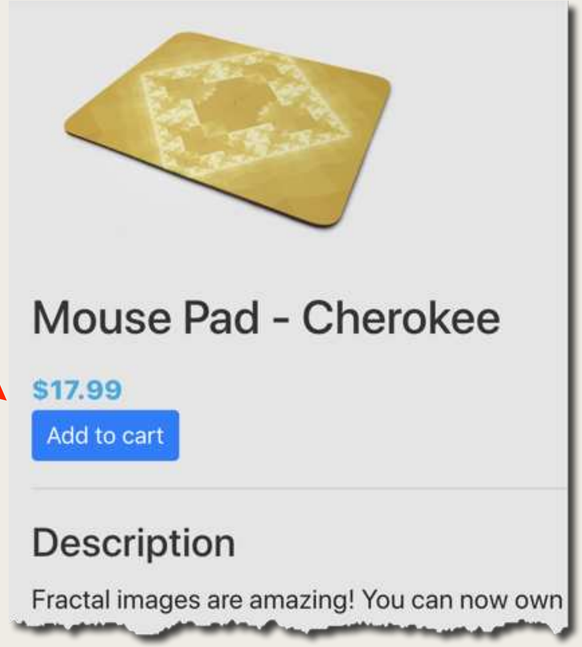
© luv2code LLC

Step 1: Add click handler for "Add to cart" button

File: product-details.component.html

```
...
<button (click)="addToCart()" class="btn btn-primary btn-sm">Add to cart</button>
...
```

On click event
call the method: addToCart()



© luv2code LLC

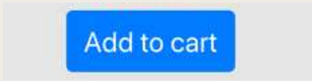
Step 2: Update ProductDetailsComponent with click handler method

File: product-details.component.ts

```
export class ProductDetailsComponent implements OnInit {

  product: Product = new Product();


  // retrieve product from REST API (Spring Boot Backend)
  // ...


  addToCart() {

    console.log(`Adding to cart: ${this.product.name}, ${this.product.unitPrice}`);

    const theCartItem = new CartItem(this.product);

    this.cartService.addToCart(theCartItem);

  }

}
```


© luv2code LLC

Application Interaction



ProductDetailsComponent

CartStatusComponent

1. subscribe for events

4. update UI for total price and quantity

2. addToCart()

New code

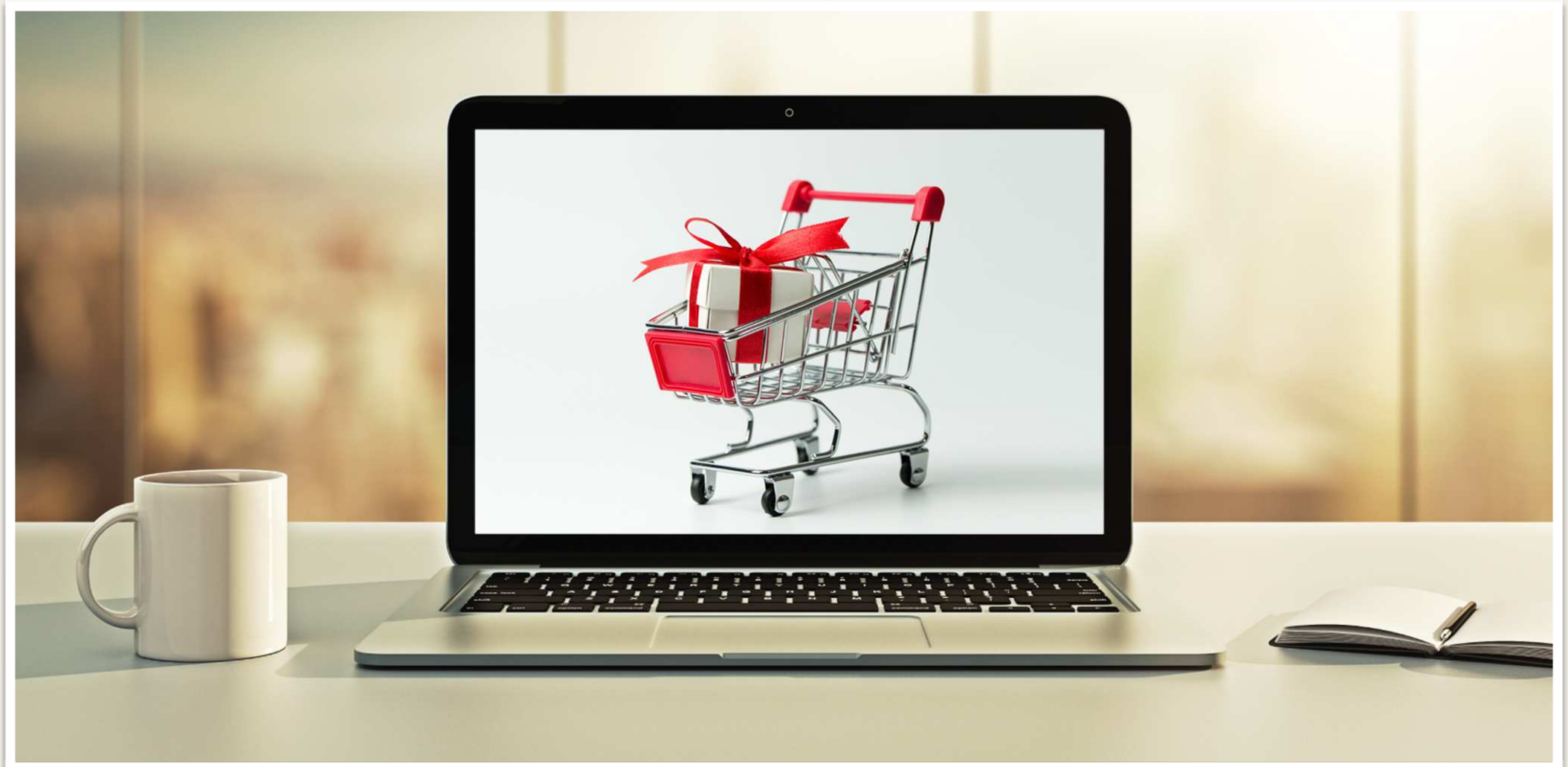
3. publish events to all subscribers

CartService

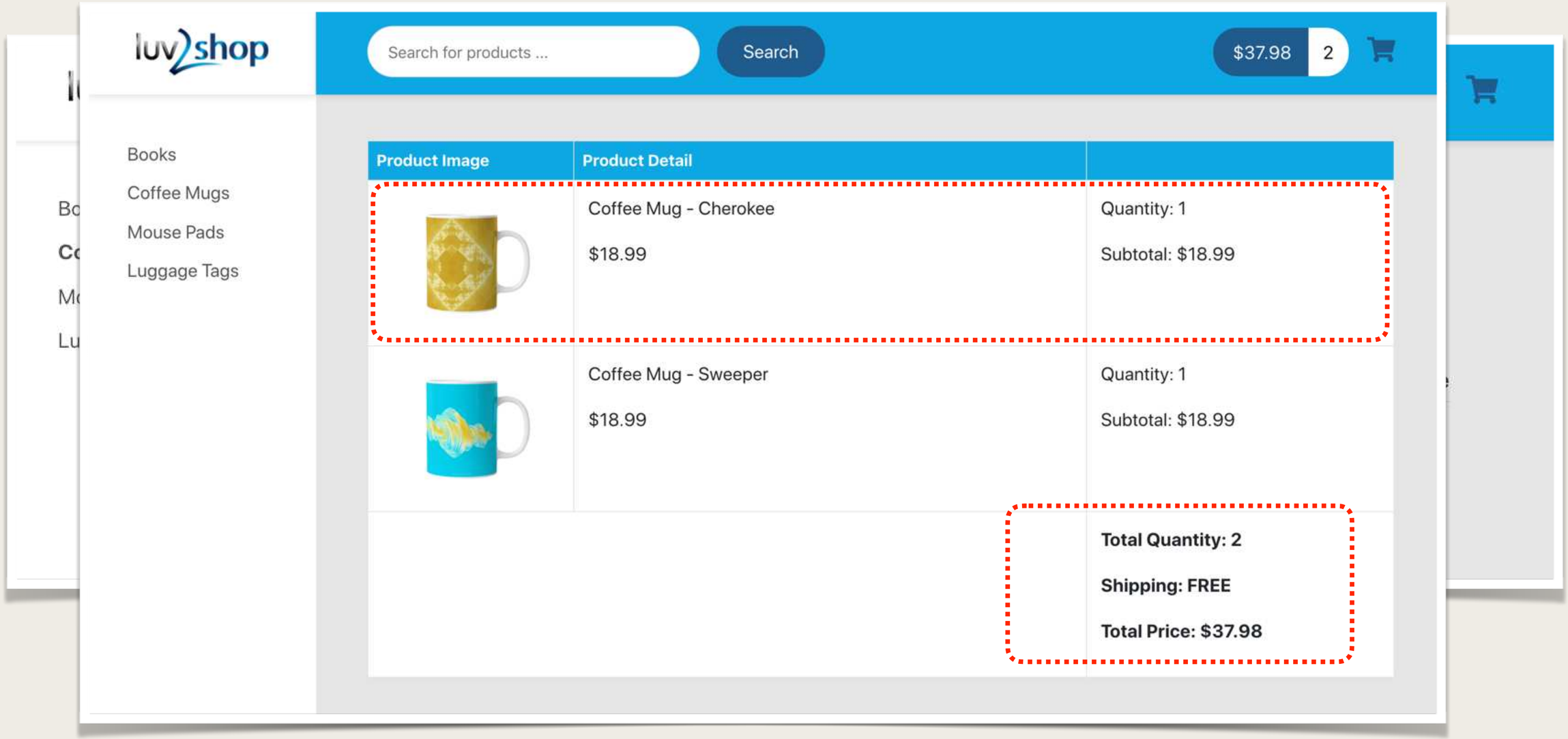
No changes
For steps 1, 3 and 4

© luv2code LLC

Shopping Cart - List Items

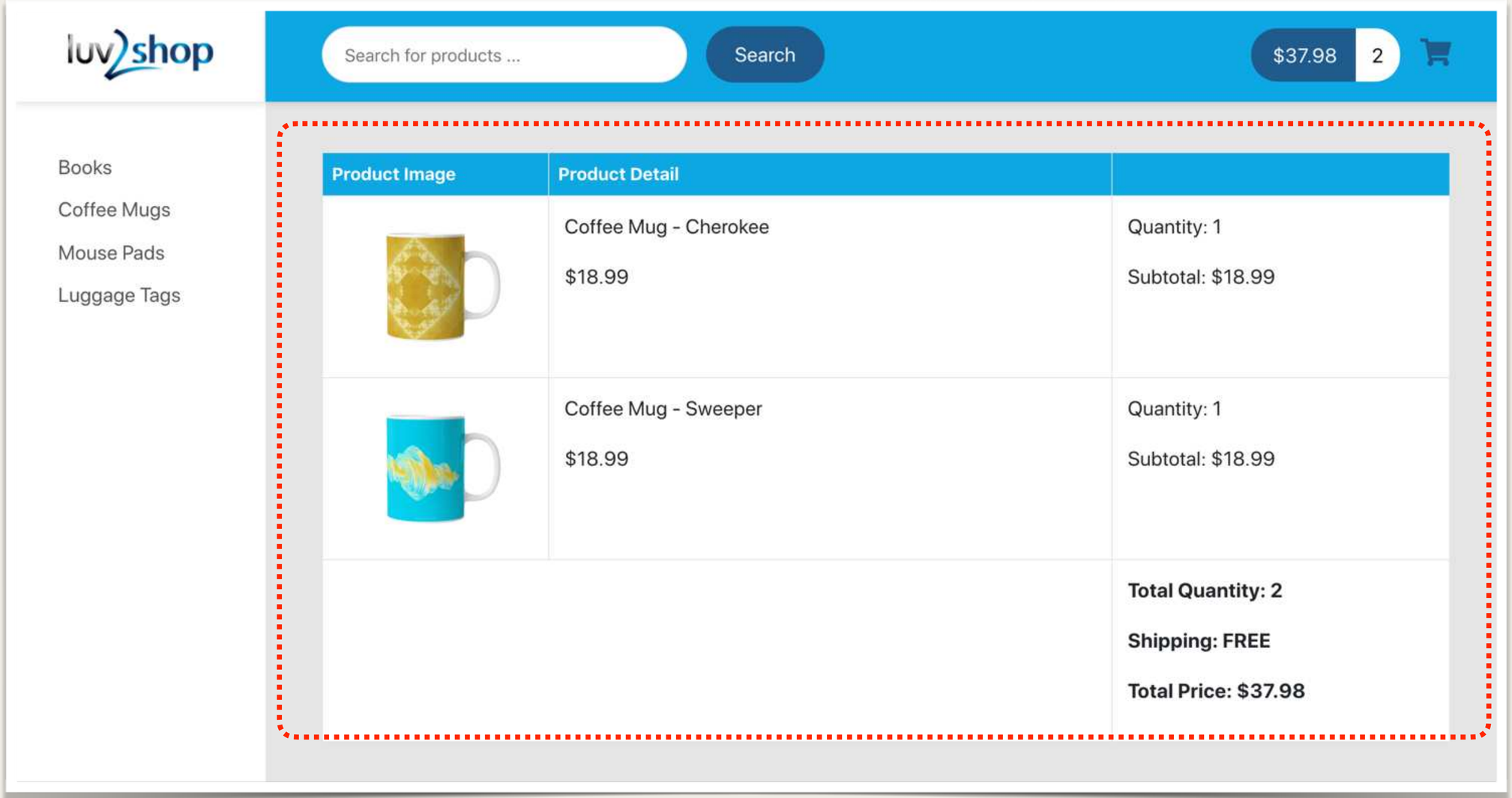


List Items in the Shopping Cart



© luv2code LLC

New Component: CartDetailsComponent



© luv2code LLC

Development Process

Step-By-Step

1. Create new component: CartDetailsComponent
2. Add new route for CartDetailsComponent
3. Update link for Shopping Cart icon
4. Modify CartDetailsComponent to retrieve cart items
5. Add HTML template for CartDetailsComponent

© luv2code LLC

Step 1: Create new component for Cart Details

```
> ng generate component components/cart-details
```


© luv2code LLC

Step 2: Add new route for CartDetailsComponent

File: app.module.ts

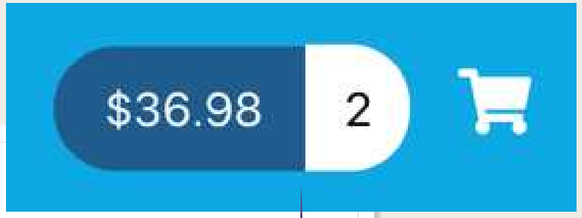
```
const routes: Routes = [
  {path: 'cart-details', component: CartDetailsComponent},
  ...

];
```


© luv2code LLC

Step 3: Update link for Shopping Cart icon

File: cart-status.component.html

```
<div class="cart-area d-n">

    <a routerLink="/cart-details">
        <div class="total">{{ totalPrice | currency: 'USD' }}
            <span>{{ totalQuantity }}</span>
        </div>
        <i class="fa fa-shopping-cart" aria-hidden="true"></i>
    </a>

</div>
```



When user clicks `CartStatusComponent` then use new route: `/cart-details` to view `CartDetailsComponent`

© luv2code LLC

Step 4: Modify CartDetailsComponent to retrieve cart items

File: cart-details.component.ts

```
export class CartDetailsComponent implements OnInit {

  cartItems: CartItem[] = [];
  totalPrice: number = 0;
  totalQuantity: number = 0;

  constructor(private cartService: CartService) { }

  ngOnInit() {
    this.listCartDetails();
  }
```

```
  listCartDetails() {

    // get a handle to the cart items
    this.cartItems = this.cartService.cartItems;

    // subscribe to the cart totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );

    // compute cart total price and total quantity
    this.cartService.computeCartTotals();
  }

}
```

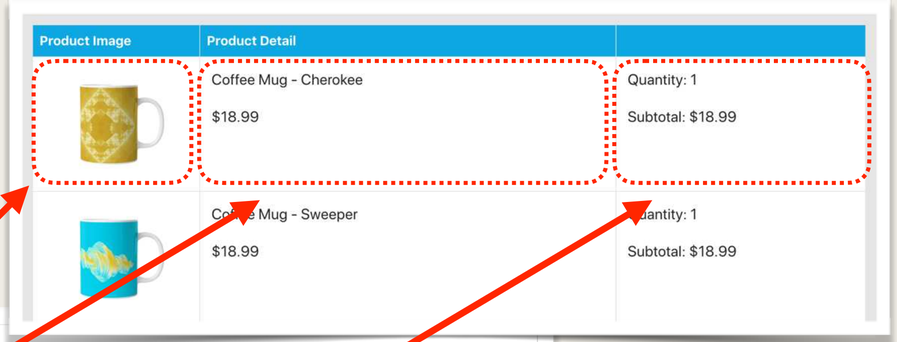

© luv2code LLC

Step 5: Add HTML template for CartDetailsComponent

File: cart-details.component.ts

```
<table class="table table-bordered">
   <tr>
      <th width="20%">Product Image</th>
      <th width="50%">Product Detail</th>
      <th width="30%"></th>
   </tr>
```

```
  <tr *ngFor="let tempCartItem of cartItems">
    <td>
      <img src="{{ tempCartItem.imageUrl }}" class="img-responsive" width="150px" />
    </td>

    <td>
      <p>{{ tempCartItem.name }}</p>
      <p>{{ tempCartItem.unitPrice | currency:'USD' }} </p>
    </td>

    <td>
      <div class="items">
        <label>Quantity:</label> {{ tempCartItem.quantity }}
      </div>

      <p class="mt-2">Subtotal: {{ tempCartItem.quantity * tempCartItem.unitPrice | currency: 'USD' }}</p>
    </td>
  </tr>

</table>
```



© luv2code LLC

Step 5: Add HTML template for CartDetailsComponent

File: cart-details.component.ts

```
...
 <tr>
  <td colspan="2"></td>
  <td style="font-weight: bold">

   <p>Total Quantity: {{ totalQuantity }}</p>

   <p>Shipping: FREE</p>

   <p>Total Price: {{ totalPrice | currency: 'USD' }}</p>

  </td>
 </tr>
```

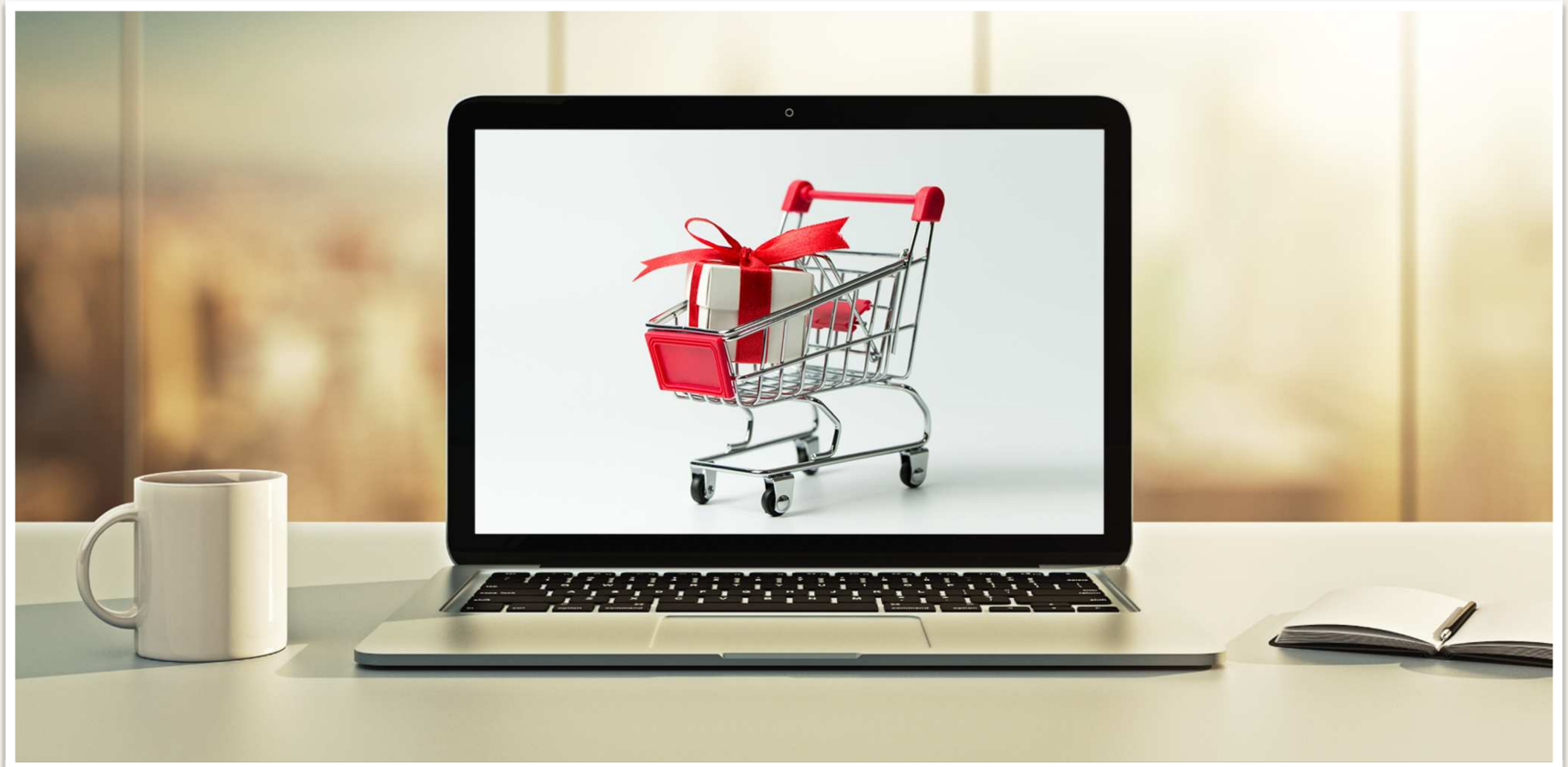
Total Quantity: 2

Shipping: FREE

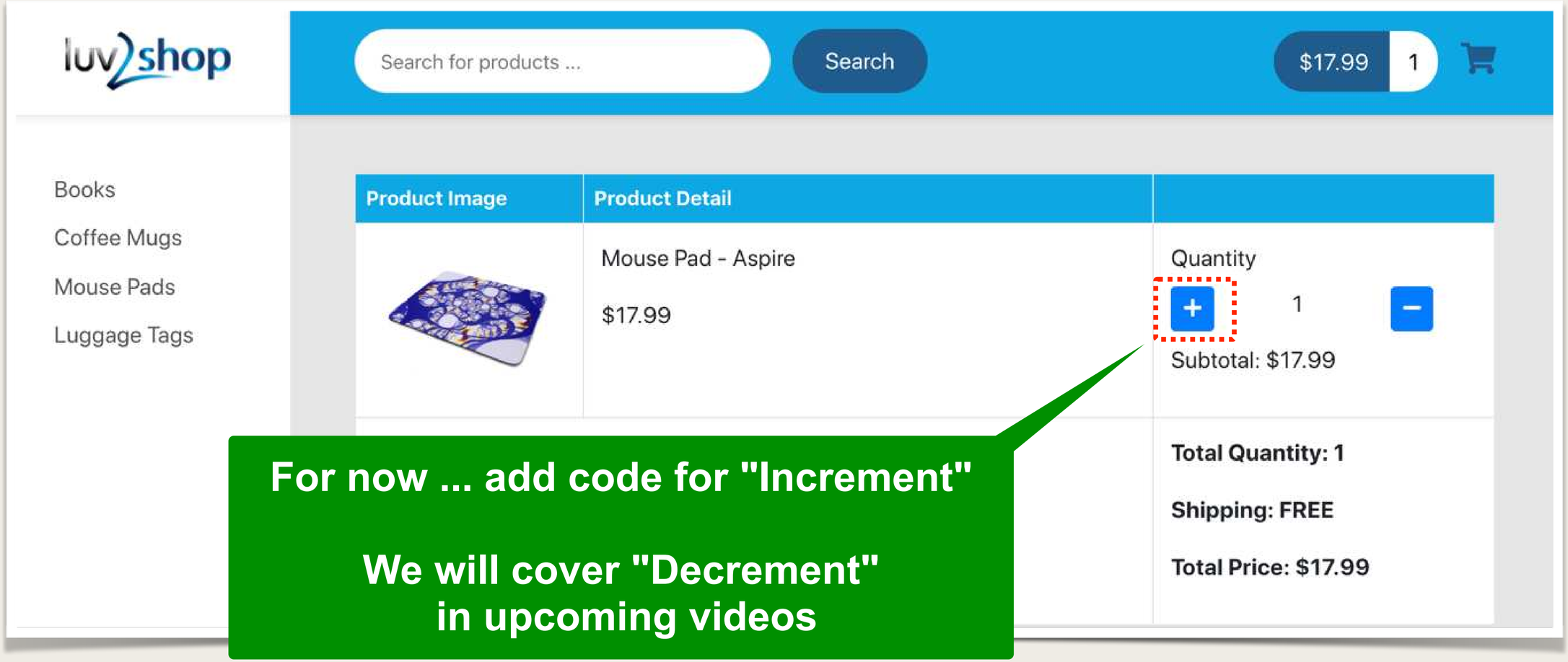
Total Price: $37.98

© luv2code LLC

Shopping Cart - Increment Item Quantity



Increment Item Quantity in the Shopping Cart



For now ... add code for "Increment"

We will cover "Decrement" in upcoming videos

© luv2code LLC

Development Process

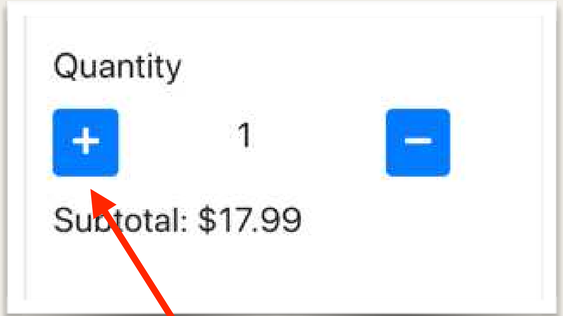
Step-By-Step

1. Modify CartDetailsComponent HTML template
    1. Add the "increment" button
    2. Add click handler for the "increment" button on HTML template

2. Update CartDetailsComponent with click handler method

© luv2code LLC

Step 1: Modify CartDetailsComponent HTML template



File: cart-details.component.html

```
...
<button (click)="incrementQuantity(tempCartItem)" class="btn btn-primary btn-sm"><i class="fas fa-plus"></i></button>
...
```

On click event
call the method: `incrementQuantity(...)`

Font Awesome
"Plus" icon

© luv2code LLC

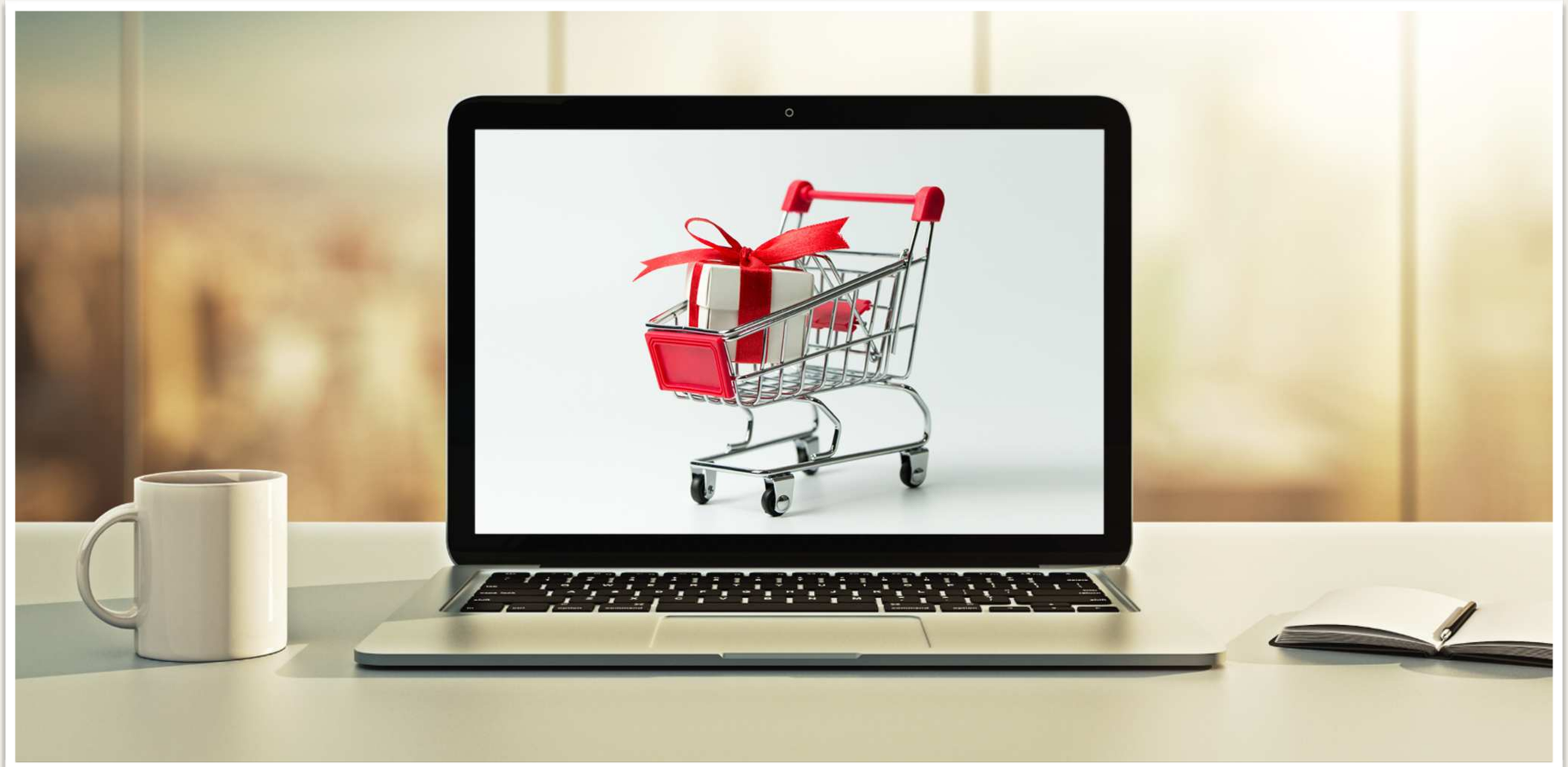
Step 2: Update CartDetailsComponent with click handler method

File: cart-details.component.ts

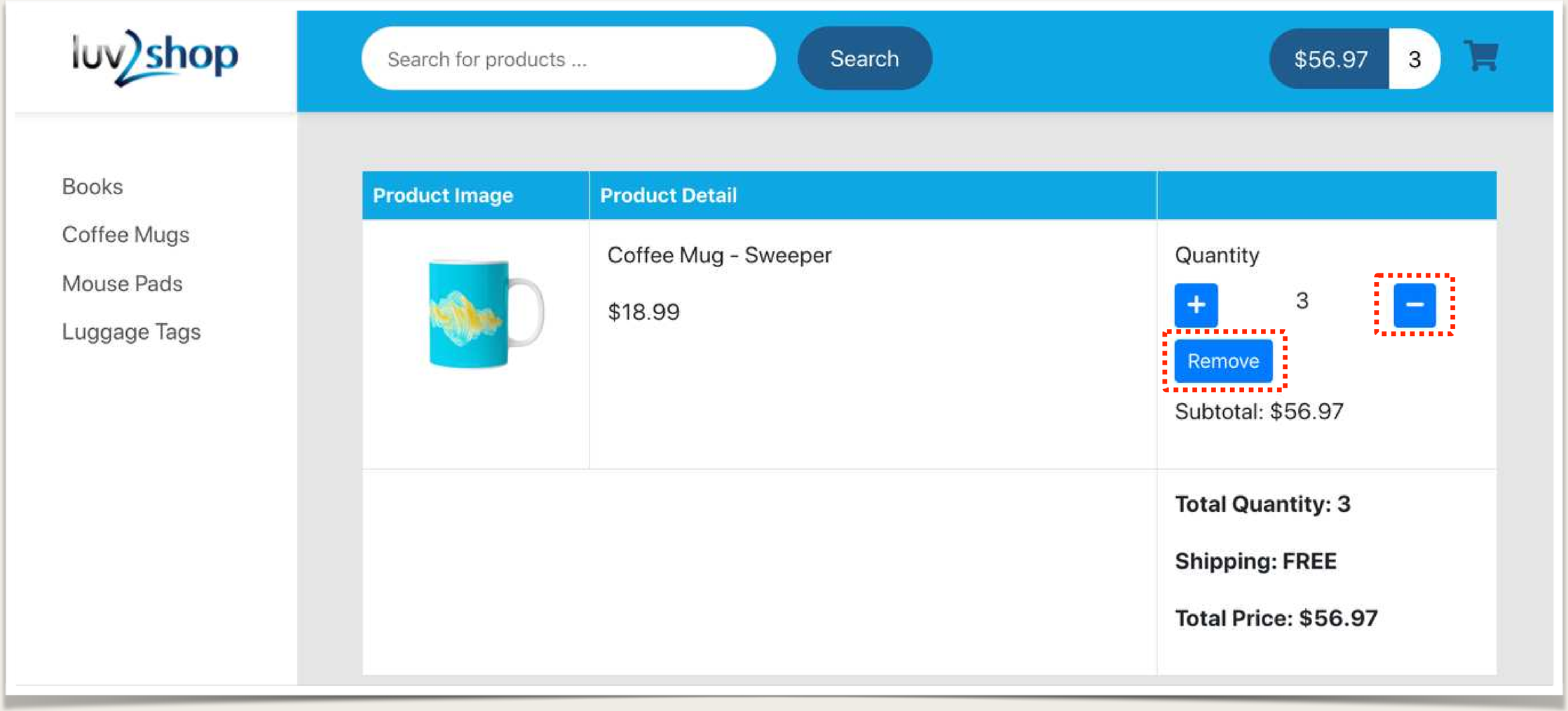
```
incrementQuantity(theCartItem: CartItem) {
  this.cartService.addToCart(theCartItem);
}
```


© luv2code LLC

Shopping Cart - Decrement / Remove Item



Decrement / Remove Item from the Shopping Cart



© luv2code LLC

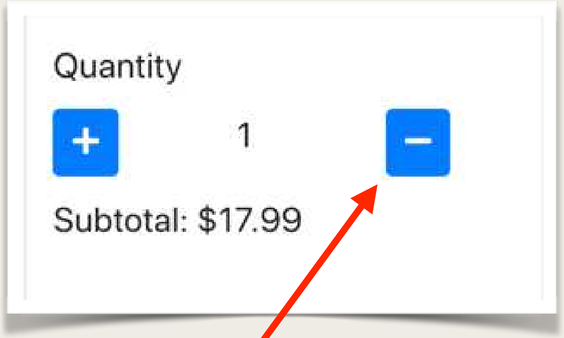
Development Process

Step-By-Step

1. Add click handler for the "decrement" button on HTML template
2. Update CartDetailsComponent with click handler method
3. Modify CartService with supporting method
4. Repeat process for "Remove" button

© luv2code LLC

Step 1: Modify CartDetailsComponent HTML template



File: cart-details.component.html

```
...
<button (click)="decrementQuantity(tempCartItem)" class="btn btn-primary btn-sm"><i class="fas fa-minus"></i></button>
...
```

On click event
call the method: `decrementQuantity(...)`

Font Awesome
"Minus" icon

© luv2code LLC

Step 2: Update CartDetailsComponent with click handler method

File: cart-details.component.ts

```
decrementQuantity(theCartItem: CartItem) {
  this.cartService.decrementQuantity(theCartItem);
}
```


© luv2code LLC

Step 3: Modify CartService with supporting method

File: cart.service.ts

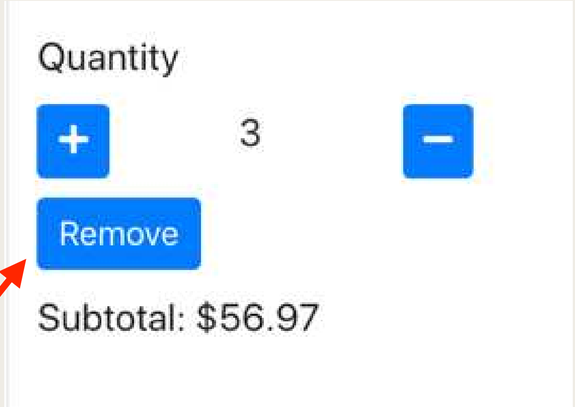
```
decrementQuantity(theCartItem: CartItem) {

  theCartItem.quantity--;

  if (theCartItem.quantity === 0) {
    this.remove(theCartItem);
  }
  else {
    this.computeCartTotals();
  }
}
```

www.luv2code.com/array-splice

```
remove(theCartItem: CartItem) {

  // get index of item in the array
  const itemIndex = this.cartItems.findIndex(
                              tempCartItem => tempCartItem.id == theCartItem.id);

  // if found, remove the item from the array at the given index
  if (itemIndex > -1) {
    this.cartItems.splice(itemIndex, 1);

    this.computeCartTotals();
  }
}
```


© luv2code LLC

Step 4: "Remove" button



File: cart-details.component.html

```
...
<button (click)="remove(tempCartItem)" class="btn btn-primary btn-sm">Remove</button>
...
```

On click event
call the method: `remove(...)`

© luv2code LLC

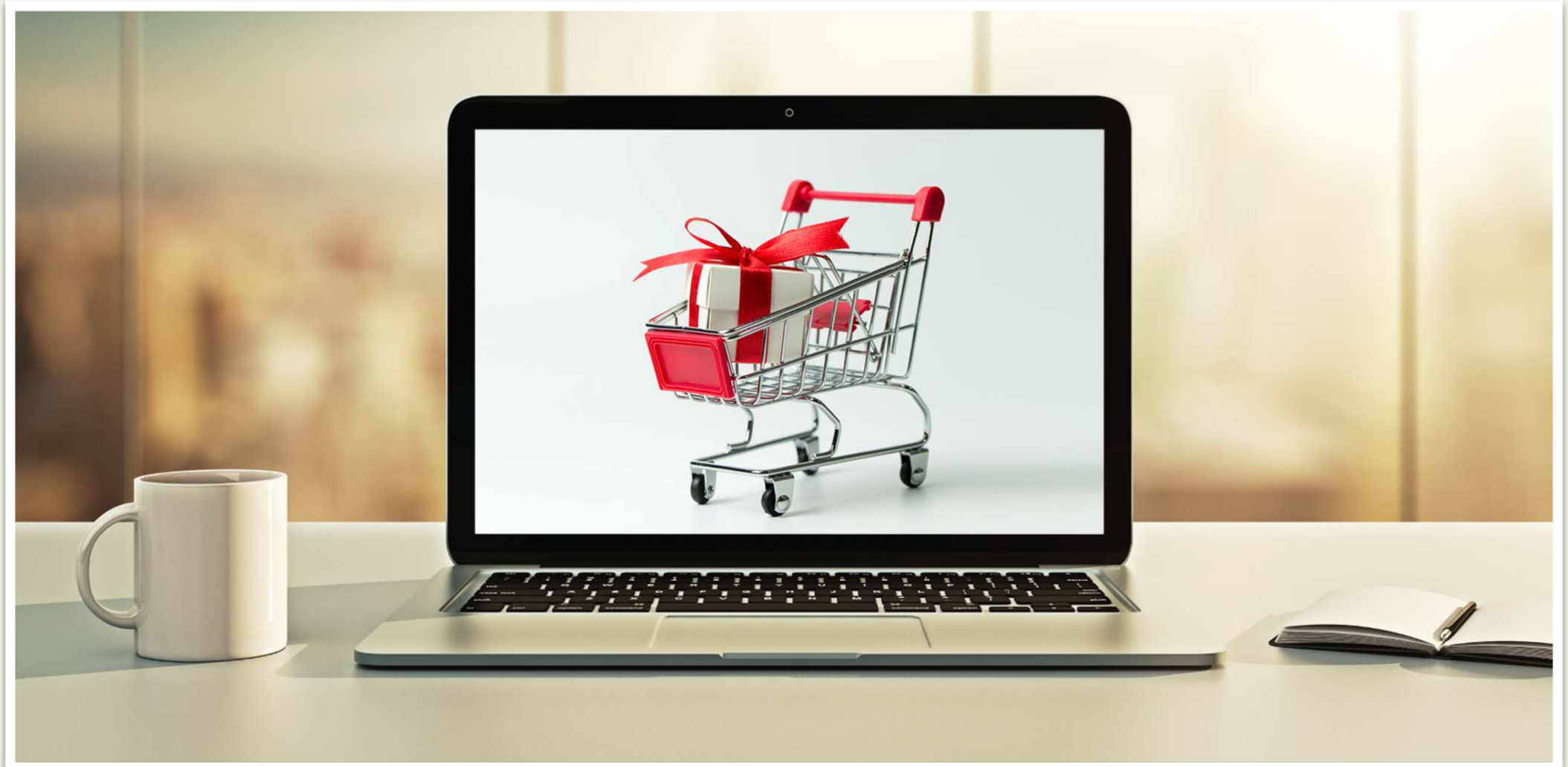
Step 4: "Remove" button

File: cart-details.component.ts

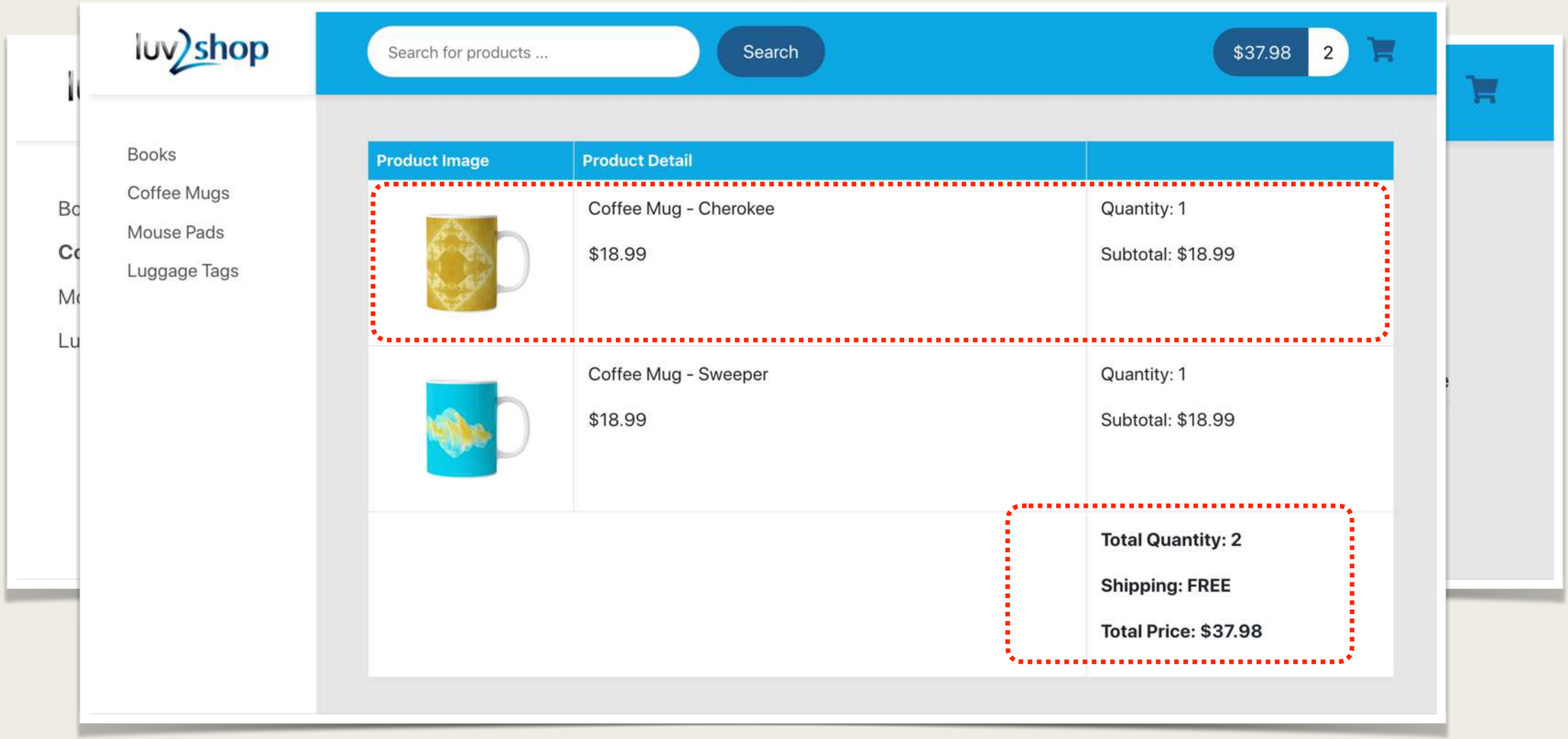
```
remove(theCartItem: CartItem) {
  this.cartService.remove(theCartItem);
}
```


© luv2code LLC

Shopping Cart - List Items

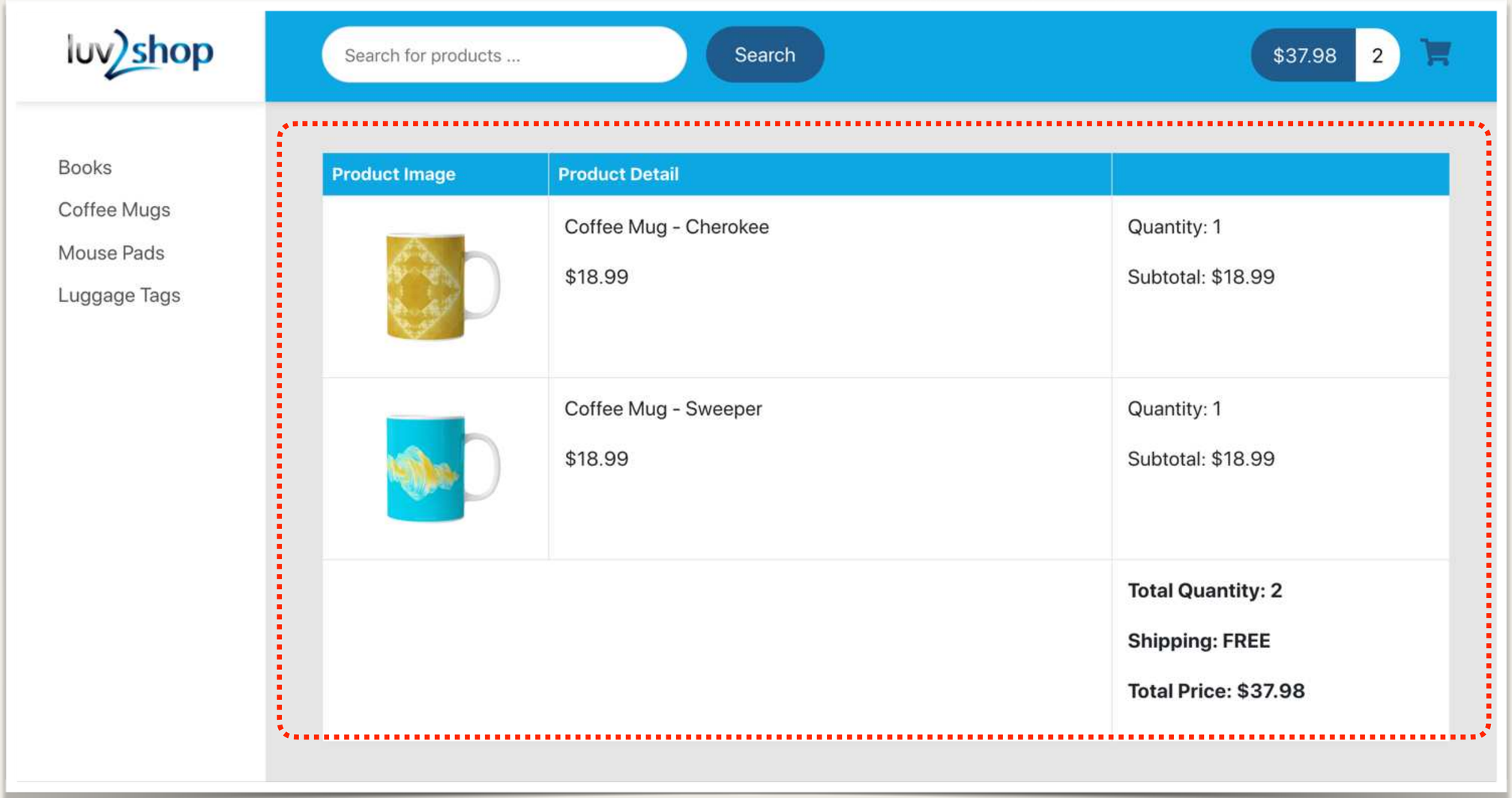


List Items in the Shopping Cart



© luv2code LLC

New Component: CartDetailsComponent



© luv2code LLC

Development Process

Step-By-Step

1. Create new component: CartDetailsComponent
2. Add new route for CartDetailsComponent
3. Update link for Shopping Cart icon
4. Modify CartDetailsComponent to retrieve cart items
5. Add HTML template for CartDetailsComponent

© luv2code LLC

Step 1: Create new component for Cart Details

```
> ng generate component components/cart-details
```


© luv2code LLC

Step 2: Add new route for CartDetailsComponent

File: app.module.ts

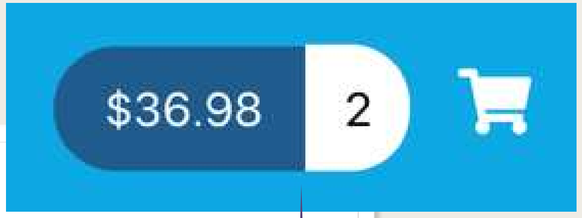
```
const routes: Routes = [
  {path: 'cart-details', component: CartDetailsComponent},
  ...

];
```


© luv2code LLC

Step 3: Update link for Shopping Cart icon

File: cart-status.component.html

```
<div class="cart-area d-n">

    <a routerLink="/cart-details">
        <div class="total">{{ totalPrice | currency: 'USD' }}
            <span>{{ totalQuantity }}</span>
        </div>
        <i class="fa fa-shopping-cart" aria-hidden="true"></i>
    </a>

</div>
```



When user clicks `CartStatusComponent` then use new route: `/cart-details` to view `CartDetailsComponent`

© luv2code LLC

Step 4: Modify CartDetailsComponent to retrieve cart items

File: cart-details.component.ts

```
export class CartDetailsComponent implements OnInit {

  cartItems: CartItem[] = [];
  totalPrice: number = 0;
  totalQuantity: number = 0;

  constructor(private cartService: CartService) { }

  ngOnInit() {
    this.listCartDetails();
  }
```

```
  listCartDetails() {

    // get a handle to the cart items
    this.cartItems = this.cartService.cartItems;

    // subscribe to the cart totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );

    // compute cart total price and total quantity
    this.cartService.computeCartTotals();
  }

}
```

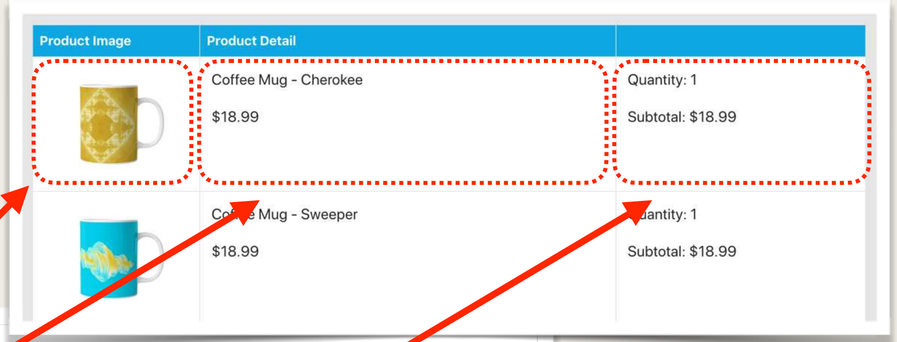

© luv2code LLC

Step 5: Add HTML template for CartDetailsComponent

File: cart-details.component.ts

```
<table class="table table-bordered">
   <tr>
      <th width="20%">Product Image</th>
      <th width="50%">Product Detail</th>
      <th width="30%"></th>
   </tr>
```

```
  <tr *ngFor="let tempCartItem of cartItems">
    <td>
      <img src="{{ tempCartItem.imageUrl }}" class="img-responsive" width="150px" />
    </td>

    <td>
      <p>{{ tempCartItem.name }}</p>
      <p>{{ tempCartItem.unitPrice | currency:'USD' }} </p>
    </td>

    <td>
      <div class="items">
        <label>Quantity:</label> {{ tempCartItem.quantity }}
      </div>

      <p class="mt-2">Subtotal: {{ tempCartItem.quantity * tempCartItem.unitPrice | currency: 'USD' }}</p>
    </td>
  </tr>

</table>
```



© luv2code LLC

Step 5: Add HTML template for CartDetailsComponent

File: cart-details.component.ts

```
...
 <tr>
  <td colspan="2"></td>
  <td style="font-weight: bold">

   <p>Total Quantity: {{ totalQuantity }}</p>

   <p>Shipping: FREE</p>

   <p>Total Price: {{ totalPrice | currency: 'USD' }}</p>

  </td>
 </tr>
```

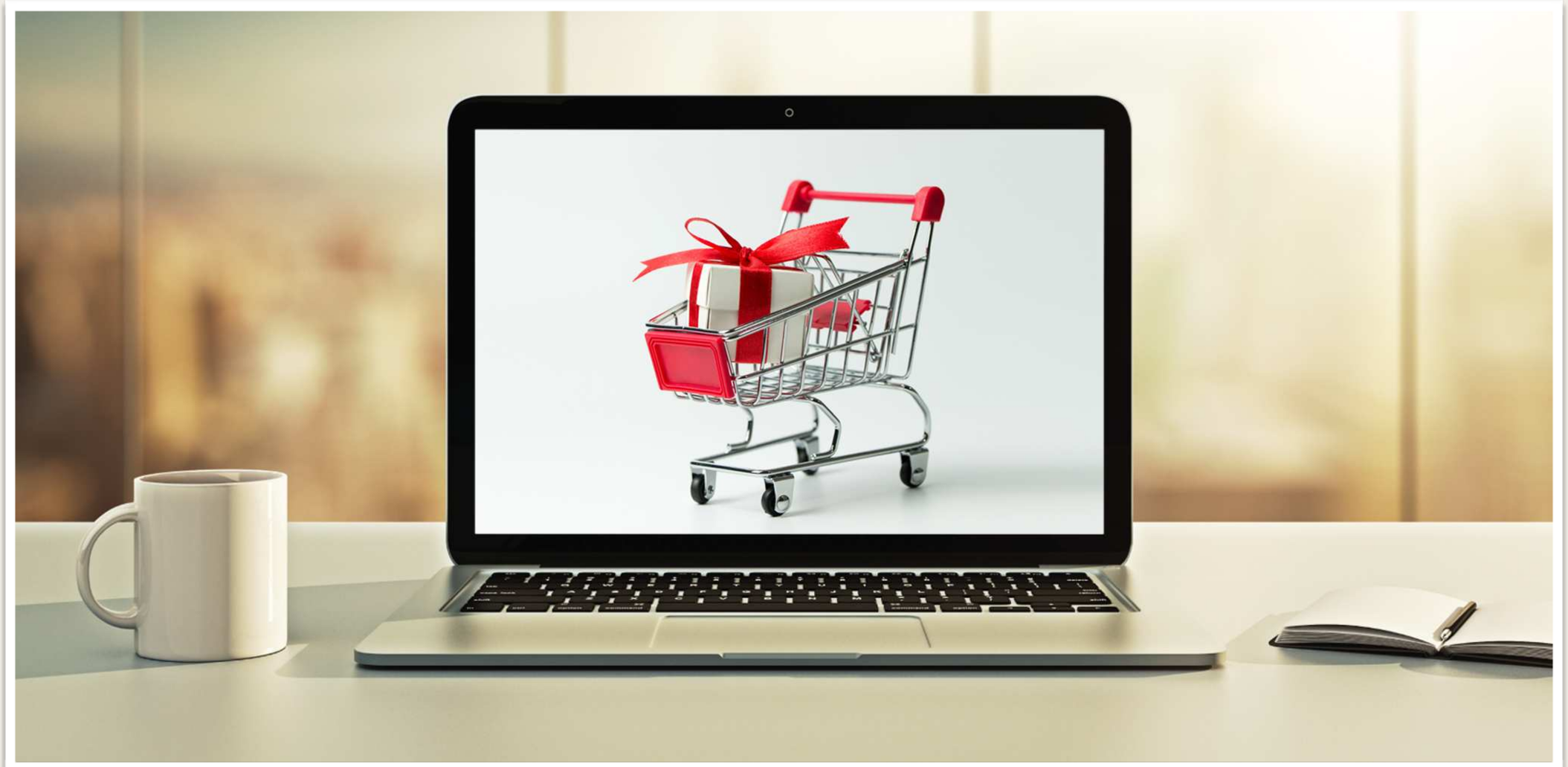
Total Quantity: 2

Shipping: FREE

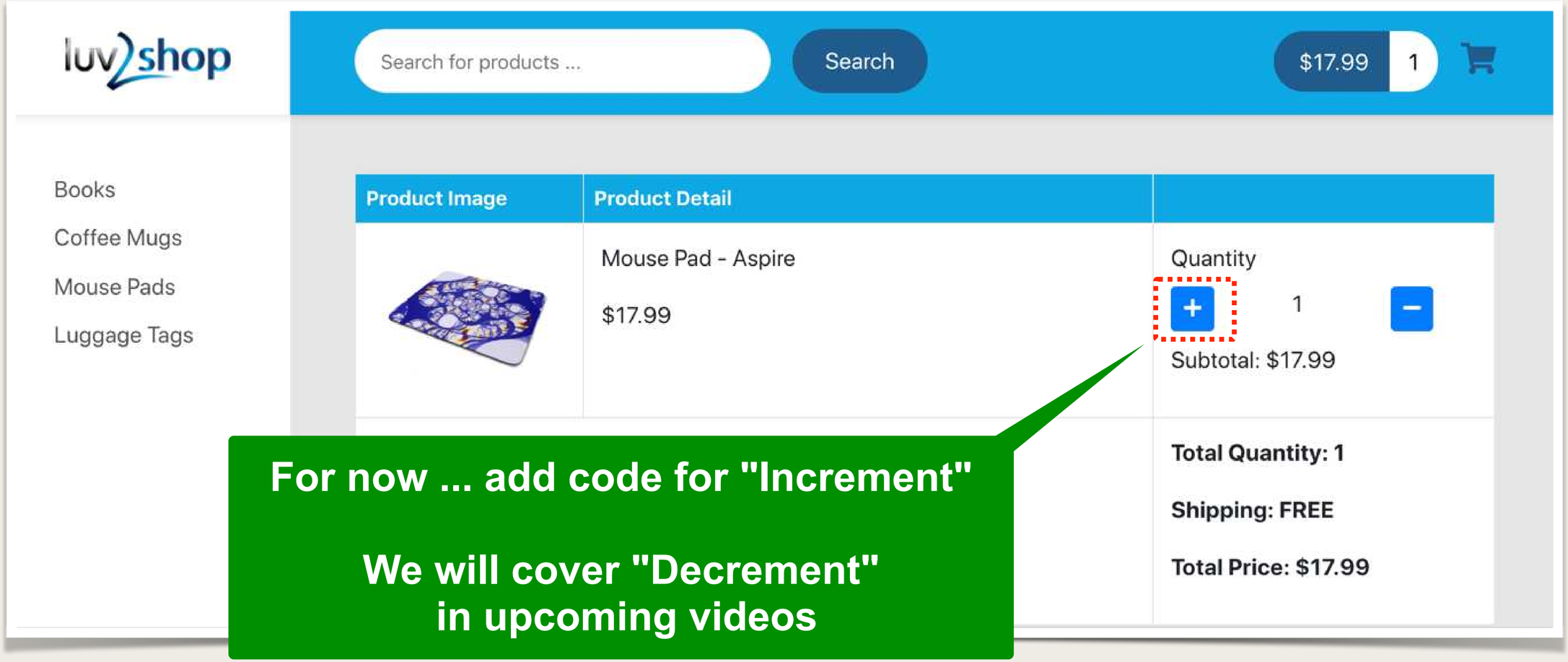
Total Price: $37.98

© luv2code LLC

Shopping Cart - Increment Item Quantity



Increment Item Quantity in the Shopping Cart



For now ... add code for "Increment"

We will cover "Decrement" in upcoming videos

© luv2code LLC

Development Process

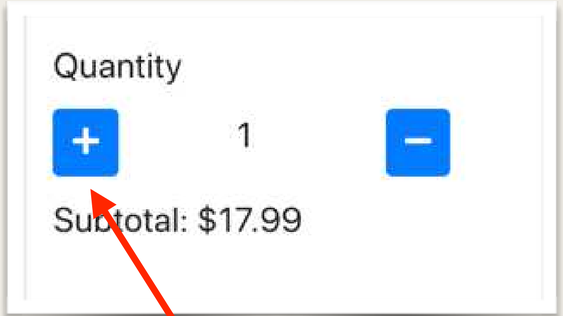
Step-By-Step

1. Modify CartDetailsComponent HTML template
    1. Add the "increment" button
    2. Add click handler for the "increment" button on HTML template

2. Update CartDetailsComponent with click handler method

© luv2code LLC

Step 1: Modify CartDetailsComponent HTML template



File: cart-details.component.html

```
...
<button (click)="incrementQuantity(tempCartItem)" class="btn btn-primary btn-sm"><i class="fas fa-plus"></i></button>
...
```

On click event
call the method: `incrementQuantity(...)`

Font Awesome
"Plus" icon

© luv2code LLC

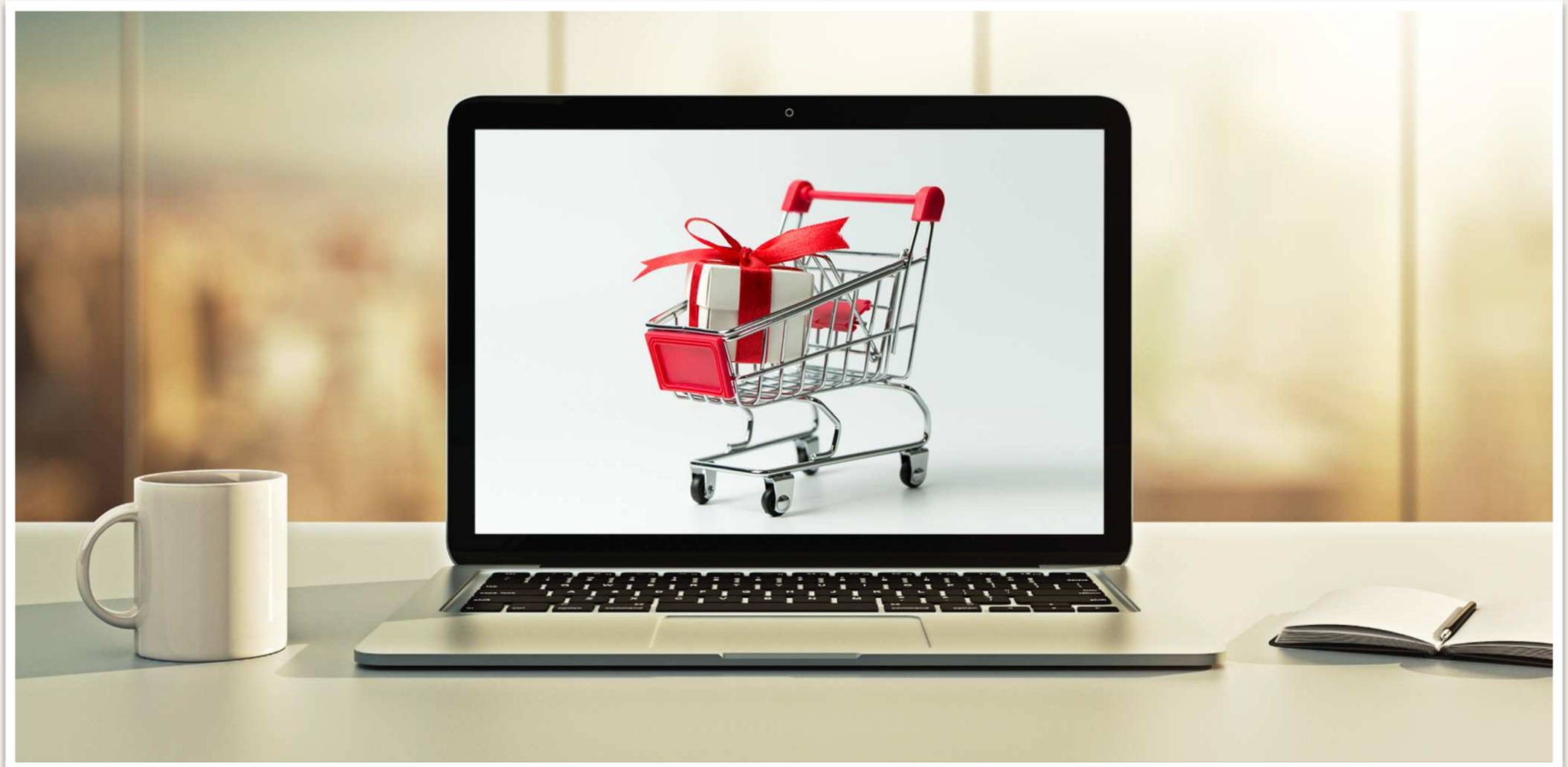
Step 2: Update CartDetailsComponent with click handler method

File: cart-details.component.ts

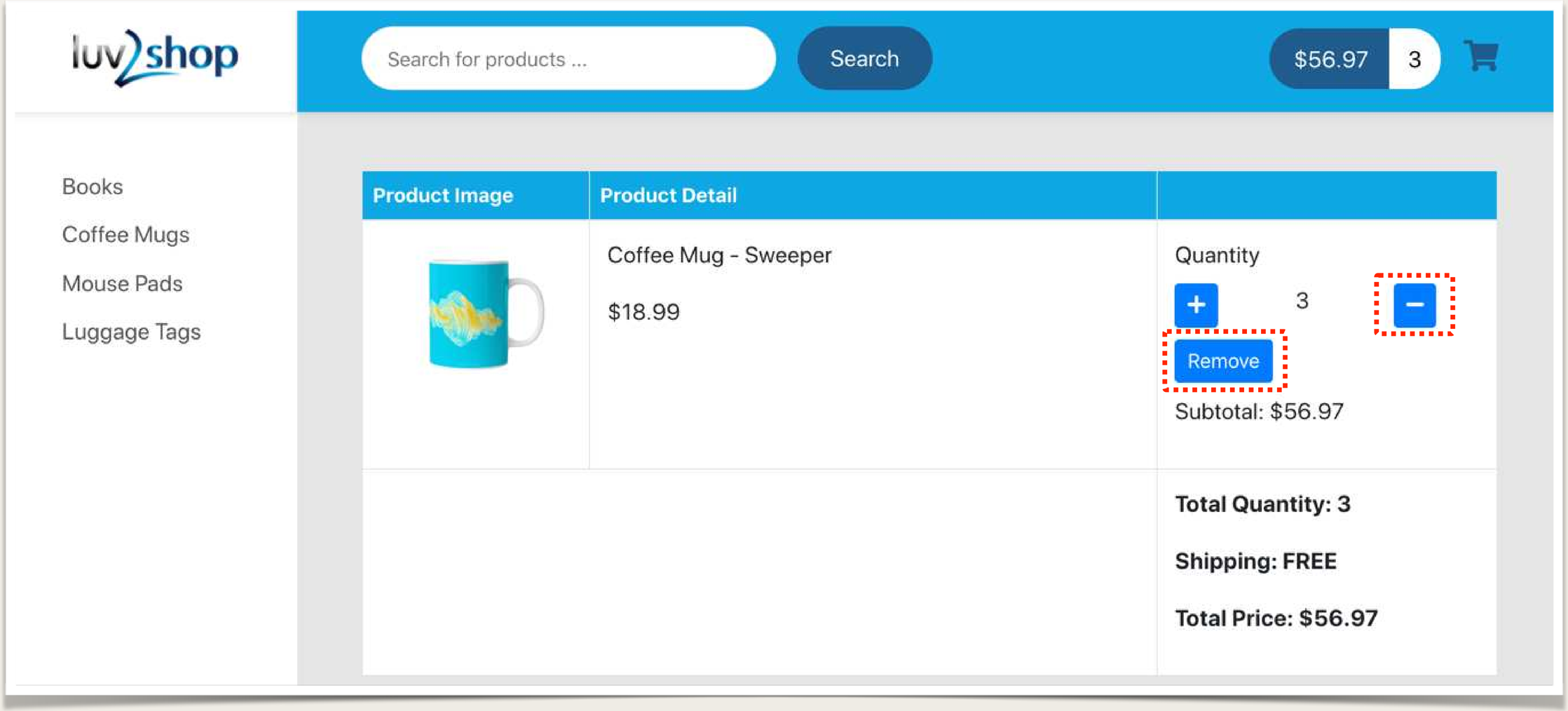
```
incrementQuantity(theCartItem: CartItem) {
  this.cartService.addToCart(theCartItem);
}
```


© luv2code LLC

Shopping Cart - Decrement / Remove Item



Decrement / Remove Item from the Shopping Cart



© luv2code LLC

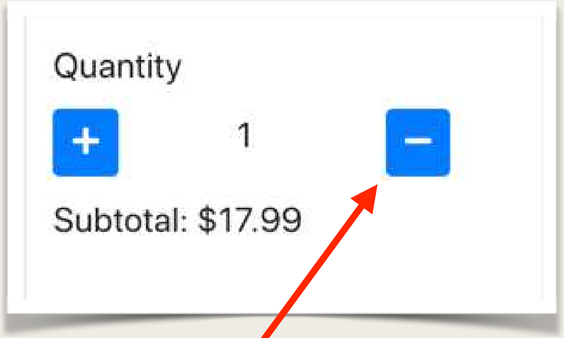
Development Process

Step-By-Step

1. Add click handler for the "decrement" button on HTML template
2. Update CartDetailsComponent with click handler method
3. Modify CartService with supporting method
4. Repeat process for "Remove" button

© luv2code LLC

Step 1: Modify CartDetailsComponent HTML template



File: cart-details.component.html

```
...
<button (click)="decrementQuantity(tempCartItem)" class="btn btn-primary btn-sm"><i class="fas fa-minus"></i></button>
...
```

On click event
call the method: `decrementQuantity(...)`

Font Awesome
"Minus" icon

© luv2code LLC

Step 2: Update CartDetailsComponent with click handler method

File: cart-details.component.ts

```
decrementQuantity(theCartItem: CartItem) {
  this.cartService.decrementQuantity(theCartItem);
}
```


© luv2code LLC

Step 3: Modify CartService with supporting method

File: cart.service.ts

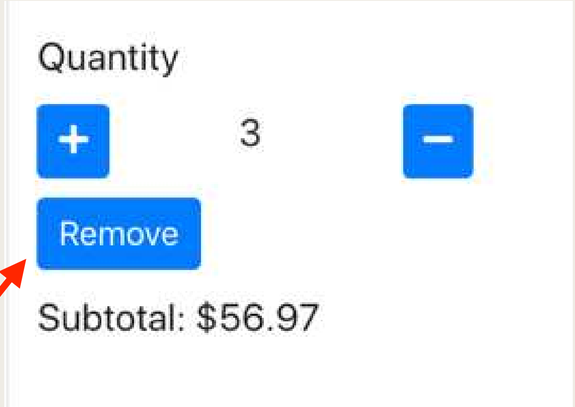
```
decrementQuantity(theCartItem: CartItem) {

  theCartItem.quantity--;

  if (theCartItem.quantity === 0) {
    this.remove(theCartItem);
  }
  else {
    this.computeCartTotals();
  }
}
```

www.luv2code.com/array-splice

```
remove(theCartItem: CartItem) {

  // get index of item in the array
  const itemIndex = this.cartItems.findIndex(
                              tempCartItem => tempCartItem.id == theCartItem.id);

  // if found, remove the item from the array at the given index
  if (itemIndex > -1) {
    this.cartItems.splice(itemIndex, 1);

    this.computeCartTotals();
  }
}
```


© luv2code LLC

Step 4: "Remove" button



File: cart-details.component.html

```
...
<button (click)="remove(tempCartItem)" class="btn btn-primary btn-sm">Remove</button>
...
```

On click event
call the method: `remove(...)`

© luv2code LLC

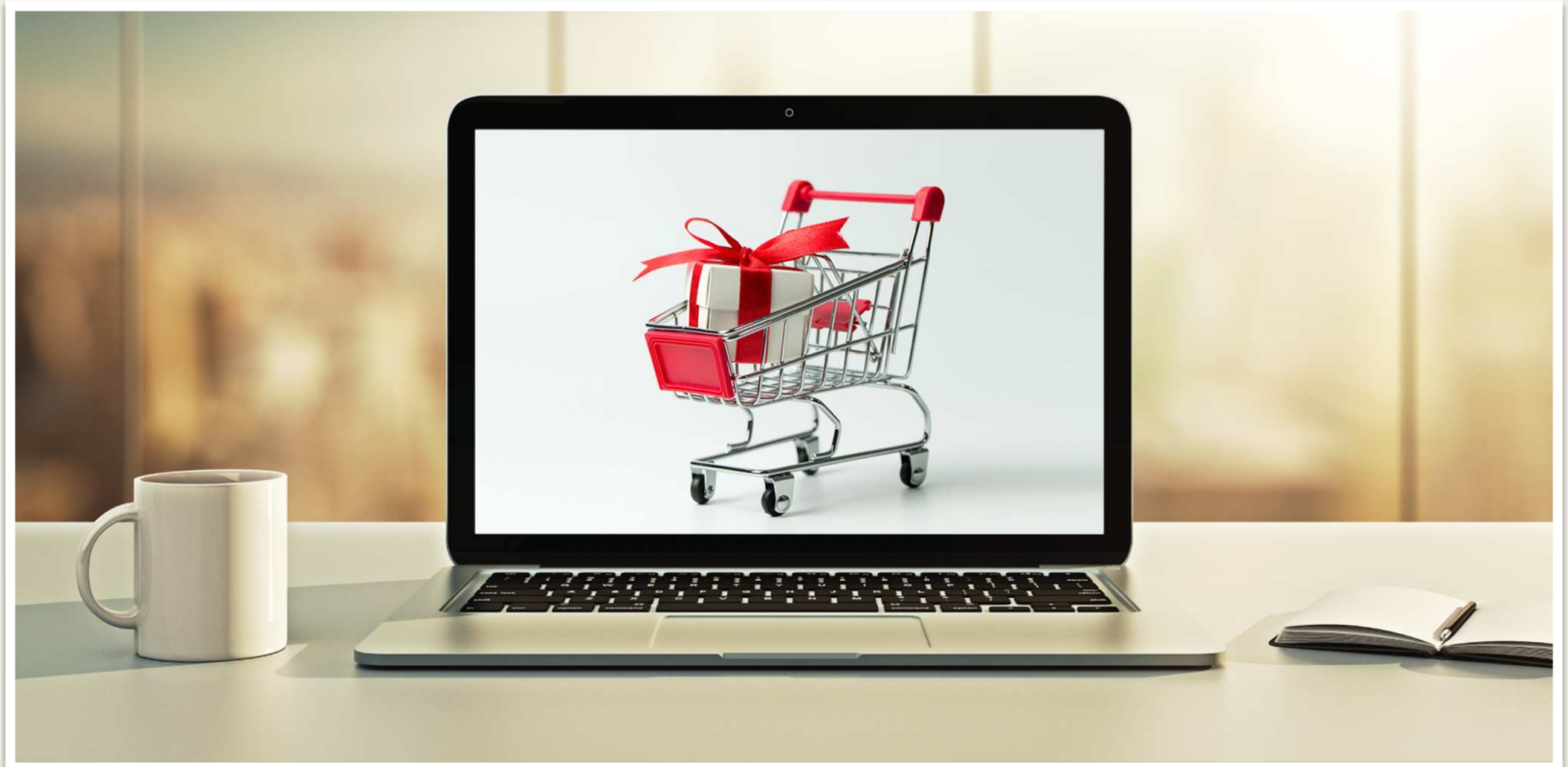
Step 4: "Remove" button

File: cart-details.component.ts

```
remove(theCartItem: CartItem) {
  this.cartService.remove(theCartItem);
}
```


© luv2code LLC

Checkout Form - Layout



Overview of Angular Forms

- Can easily build forms in Angular
- Supports form data-binding, validation and processing
- Angular provides two types of forms
  - Reactive forms
  - Template-driven forms

© luv2code LLC

Forms

We will use
Reactive forms

- **Reactive forms**
  - Leverages programmatic API for form building
  - Scaleable solution that is designed for large, complex forms
  - Forms can be easily reused and tested

- **Template-driven forms**
  - Targeted for small, simple forms
  - Not a scalable solution for large, complex forms

© luv2code LLC

Comparison

- For comparison of Reactive forms and Template-driven forms

https://angular.io/guide/forms-overview

© luv2code LLC

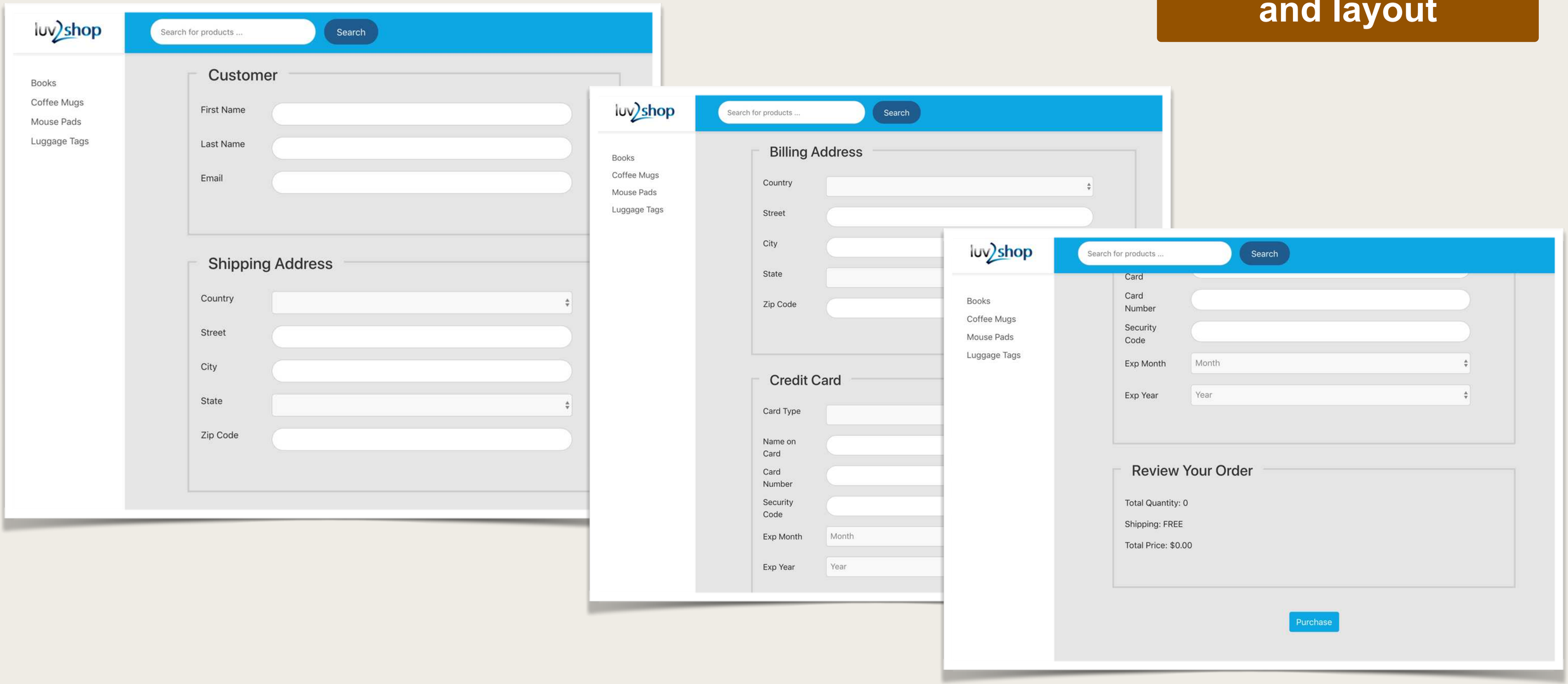
Key Components

Name	Description
FormControl	Individual control that tracks the value and validation status
FormGroup	A collection of controls. Can create nested groups.
others	...

© luv2code LLC

Our Checkout Form

Focus on form construction and layout



© luv2code LLC

Development Process

Step-By-Step

1. Generate our checkout component
2. Add a new route for checkout component
3. Create a new checkout button and link to checkout component
4. Add support for reactive forms
5. Define form in component .ts file
6. Layout form controls in HTML template
7. Add event handler for form submission

© luv2code LLC

Step 1: Generate our checkout component

```
> ng generate component components/checkout
```


© luv2code LLC

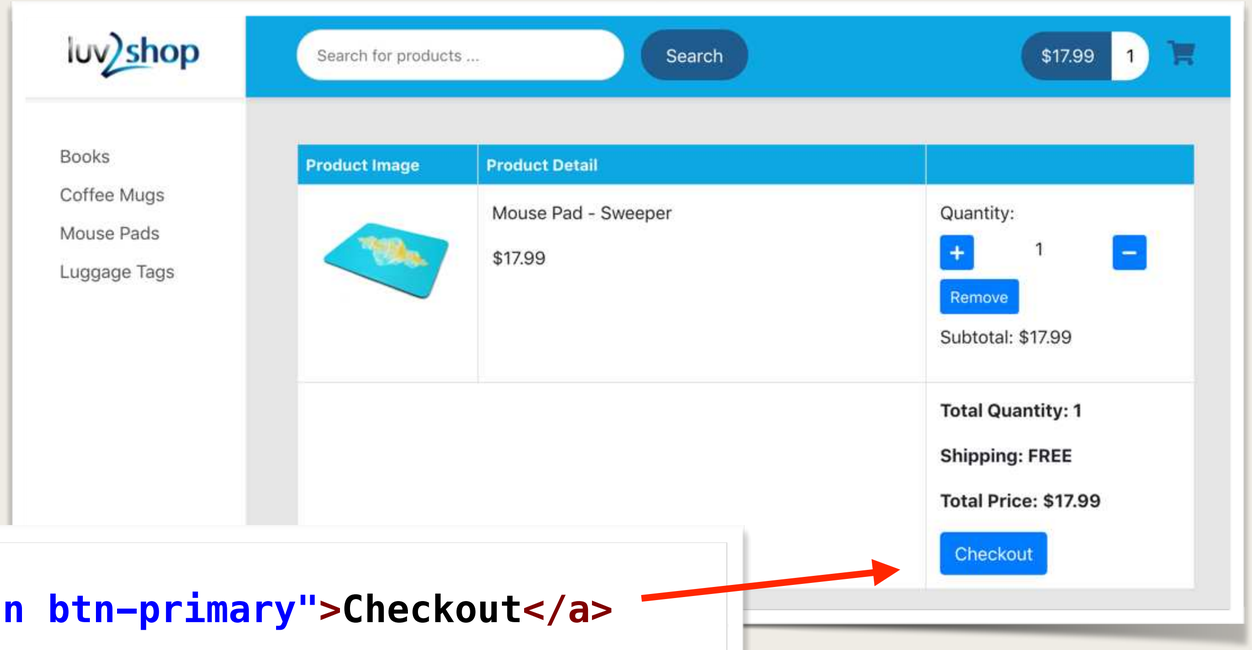
Step 2: Add a new route for checkout component

File: app.module.ts

```
const routes: Routes = [
  {path: 'checkout', component: CheckoutComponent},
  ...

];
```


© luv2code LLC

Step 3: Create new checkout button and link to checkout component



File: cart-details.component.html

```
...
<a routerLink='/checkout' class="btn btn-primary">Checkout</a>
...
```


© luv2code LLC

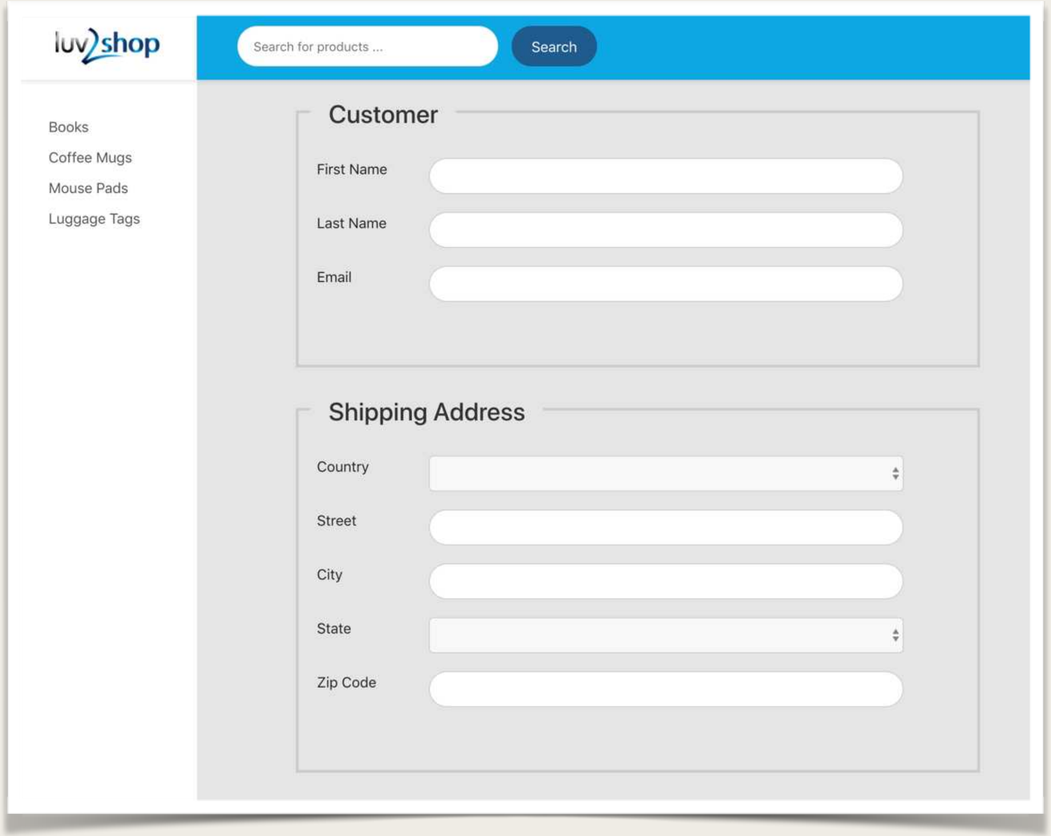
Step 4: Add support for reactive forms

File: app.module.ts

```
...
  imports: [
    ...
    ReactiveFormsModule
  ],
...
```


© luv2code LLC

Step 5: Define form in component .ts file



© luv2code LLC

Step 5: Define form in component .ts file

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  checkoutFormGroup: FormGroup;

  constructor(private formBuilder: FormBuilder) { }

  ngOnInit(): void {

    this.checkoutFormGroup = this.formBuilder.group({
      customer: this.formBuilder.group({
        firstName: [''],
        lastName: [''],
        email: ['']
      })
    });
  }

  onSubmit() {
    console.log("Handling the submit button");
    console.log(this.checkoutFormGroup.get('customer').value);
  }
}
```

Inject the form builder

Declare our form group

Build the form

Method to call when submit button is clicked

© luv2code LLC

Step 6: Layout form controls in HTML template

Name of property from component class

File: checkout.component.html

```
<form [formGroup]="checkoutFormGroup">

    <!-- customer form group -->
    <div formGroupName="customer" class="form-area">
        <h3>Customer</h3>

        <div class="row">
            <div class="col-md-2"> <label>First Name</label></div>
            <input formControlName="firstName" type="text">
        </div>
        ...
    </div>
...
</form>
```

```
export class CheckoutComponent implements OnInit {

 checkoutFormGroup: FormGroup;
```

Repeat process for rows: Last Name, Email

© luv2code LLC

Step 7: Add event handler for form submission

File: checkout.component.html

```
<form [formGroup]="checkoutFormGroup" (ngSubmit)="onSubmit()">

    <!-- customer form group -->
    <div formGroupName="customer" class="form-area">
        <h3>Customer</h3>

        <div class="row">
             <div class="col-md-2"> <label>First Name</label></div>
             <input formControlName="firstName" type="text">
        </div>
    </div>

    <div class="text-center">
           <button type="submit" class="btn btn-info">Purchase</button>
    </div>

...
</form>
```

Method to call when submit button is clicked

```
onSubmit() {
  console.log("Handling the submit button");
  console.log(this.checkoutFormGroup.get('customer').value);
}
```

Read the form data

Our submit button

© luv2code LLC

Step 7: Add event handler for form submission

Read the form data

```
onSubmit() {
  console.log("Handling the submit button");
  console.log(this.checkoutFormGroup.get('customer').value);
}
```

```
console.log("Email is " + this.checkoutFormGroup.get('customer').value.email);
```

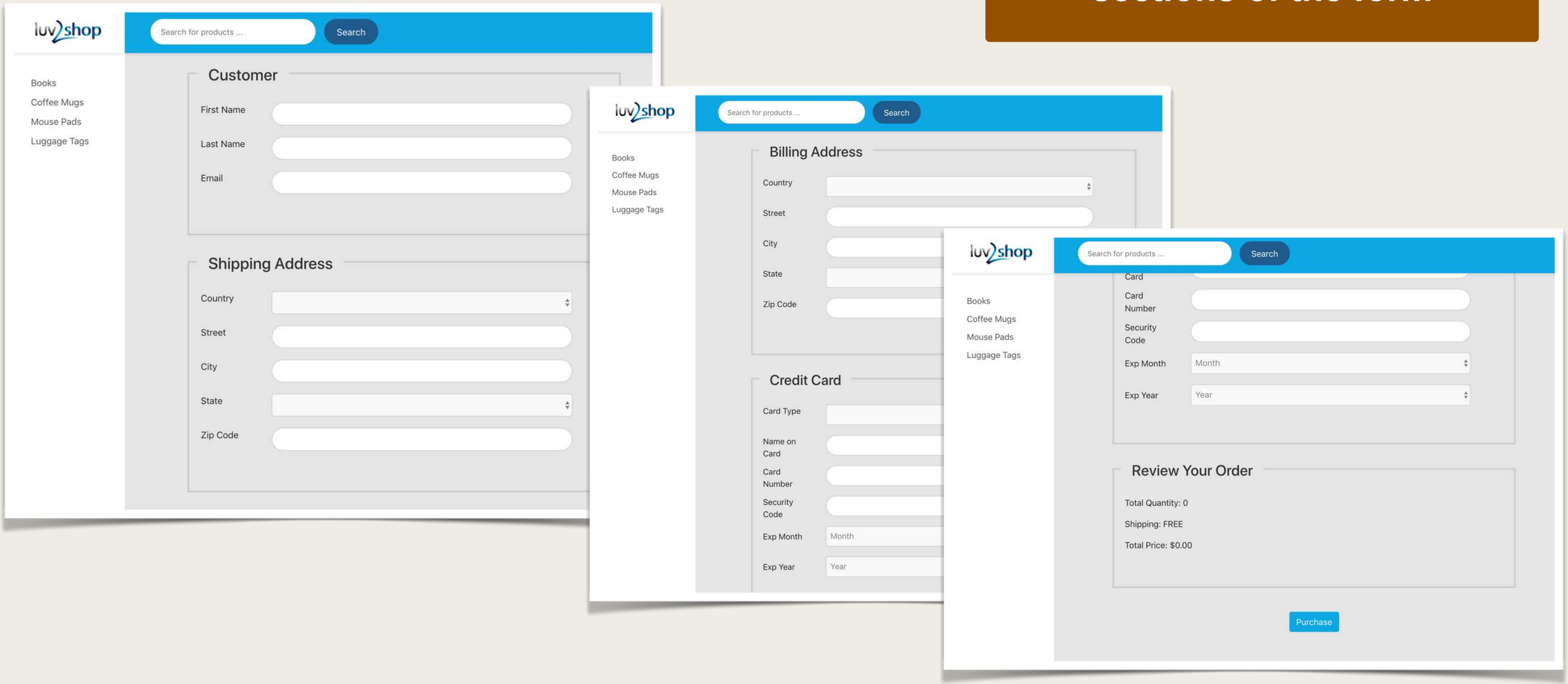
Output

```
Handling the submit button
▼{firstName: "John", lastName: "Doe", email: "john.doe@luv2code.com"}
    email: "john.doe@luv2code.com"
    firstName: "John"
    lastName: "Doe"
  ▶ __proto__: Object
```


© luv2code LLC

Our Checkout Form

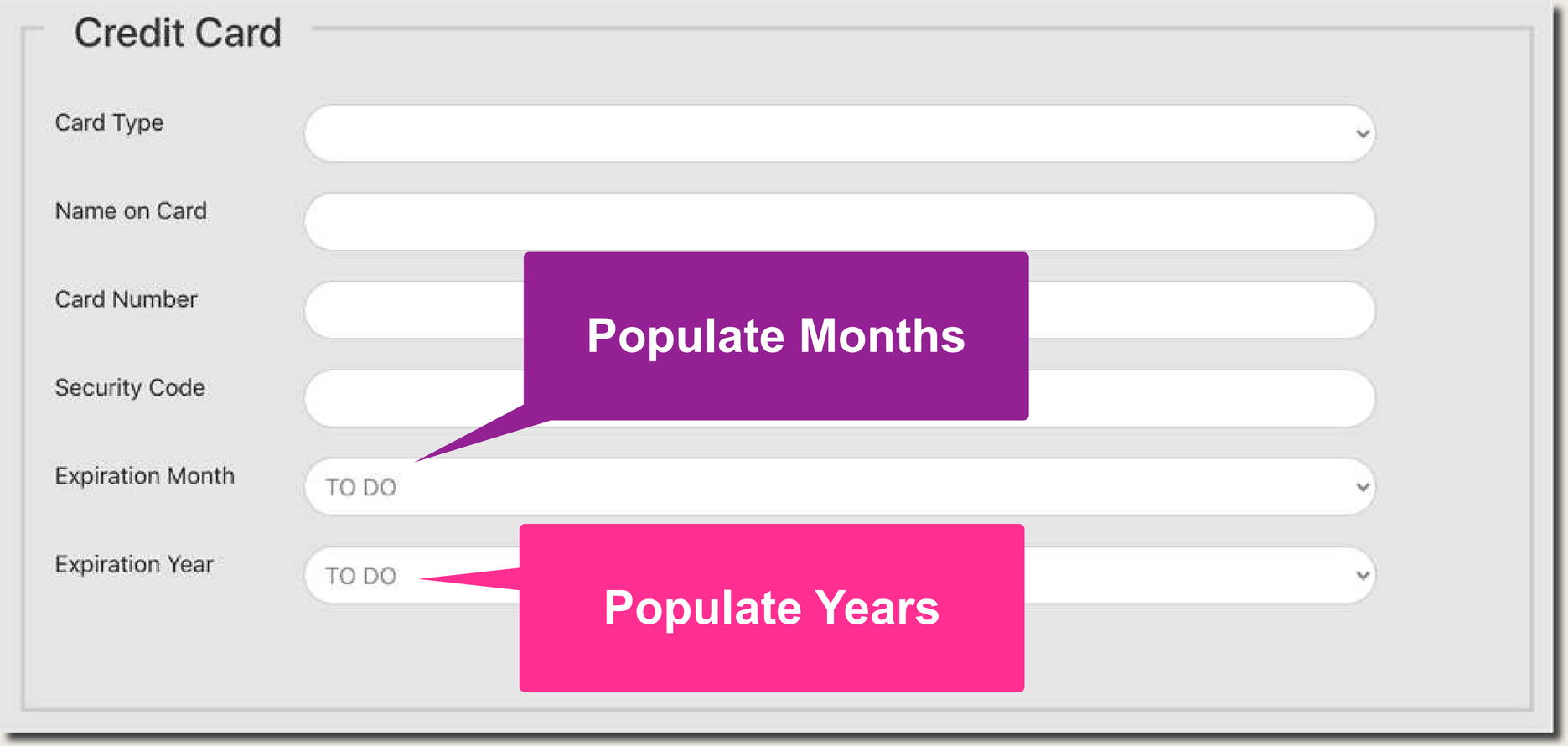
Repeat the process for other sections of the form



© luv2code LLC

Checkout Form
Populate Credit Card Expiration Dates

Credit Card Expiration Dates



© luv2code LLC

Populating Drop-Down Lists

- We could just hard code the values ... but not ideal :-(
- Years
  - Years will become outdated ... if we are in year 2022, why show 2020???
  - Dynamically populate current year .... up to next 10 years

© luv2code LLC

Development Process

Step-By-Step

1. Generate our form service: `Luv2ShopFormService`
2. Add methods to the form service for months and years
3. Update checkout component to retrieve the months and years from service
4. Update HTML template to populate drop-down lists for months and years

© luv2code LLC

Step 1: Generate our form service

```
> ng generate service services/Luv2ShopForm
```


© luv2code LLC

Step 2: Add methods to form service for months and years

File: luv2-shop-form.service.ts

```
import { Observable } from 'rxjs';
import { of } from 'rxjs';

...
export class Luv2ShopFormService {

  constructor() { }

  getCreditCardMonths(startMonth: number): Observable<number[]> {

    let data: number[] = [];

    // build an array for "Month" dropdown list
    // - start at desired startMonth and loop until 12
    //
    for (let theMonth = startMonth; theMonth <= 12; theMonth++) {
      data.push(theMonth);
    }

    return of(data);
  }


}
```

Return an Observable array

rxjs: Reactive JavaScript
https://angular.io/guide/rx-library

The "of" operator from rxjs, will wrap an object as an Observable

© luv2code LLC

Step 2: Add methods to form service for months and years

File: luv2-shop-form.service.ts

```
import { Observable } from 'rxjs';
import { of } from 'rxjs';

export class Luv2ShopFormService {

  ...

  getCreditCardYears(): Observable<number[]> {

    let data: number[] = [];

    // build an array for "Year" dropdown list
    // - start at current year and loop for next 10
    //
    const startYear: number = new Date().getFullYear();
    const endYear: number = startYear + 10;

    for (let theYear = startYear; theYear <= endYear; theYear++) {
      data.push(theYear);
    }

    return of(data);
  }

}
```

Add similar method for Years

Get the current year

The "of" operator from rxjs, will wrap an object as an Observable

© luv2code LLC

Step 3: Update checkout component to retrieve months and years

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ...
  creditCardYears: number[] = [];
  creditCardMonths: number[] = [];

  constructor(private formBuilder: FormBuilder,
              private luv2ShopFormService: Luv2ShopFormService) { }

  ngOnInit(): void {
    ...

    // populate credit card months
    const startMonth: number = new Date().getMonth() + 1;
    console.log("startMonth: " + startMonth);

    this.luv2ShopFormService.getCreditCardMonths(startMonth).subscribe(
                                                                    ata));

    );
  }

}
```

Inject our form service

Get the current month

JavaScript `Date` object
the months are zero-based
(0 ... 11)

So we add 1 to get
(1 ... 12)

www.luv2code.com/javascript-date

© luv2code LLC

Step 3: Update checkout component to retrieve months and years

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ...
  creditCardYears: number[] = [];
  creditCardMonths: number[] = [];

  constructor(private formBuilder: FormBuilder,
              private luv2ShopFormService: Luv2ShopFormService) { }

  ngOnInit(): void {
    ...

    // populate credit card years
    this.luv2ShopFormService.getCreditCardYears().subscribe(
      data => {
        console.log("Retrieved credit card years: " + JSON.stringify(data));
        this.creditCardYears = data;
      }
    );

 }

}
```

Get the credit card years

© luv2code LLC

Step 4: Update HTML template to populate drop-down lists

File: checkout.component.html

```
<div class="col-md-2"> <label>Expiration Month</label></div>

<div class="input-space">
    <select formControlName="expirationMonth">
      <option *ngFor="let month of creditCardMonths">
        {{ month }}
      </option>
    </select>
</div>
```

Loop over array

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ...
  creditCardMonths: number[] = [];
```


© luv2code LLC

Step 4: Update HTML template to populate drop-down lists

File: checkout.component.html

```
<div class="col-md-2"> <label>Expiration Year</label></div>

<div class="input-space">
    <select formControlName="expirationYear">
      <option *ngFor="let year of creditCardYears">
        {{ year }}
      </option>
    </select>
</div>
```

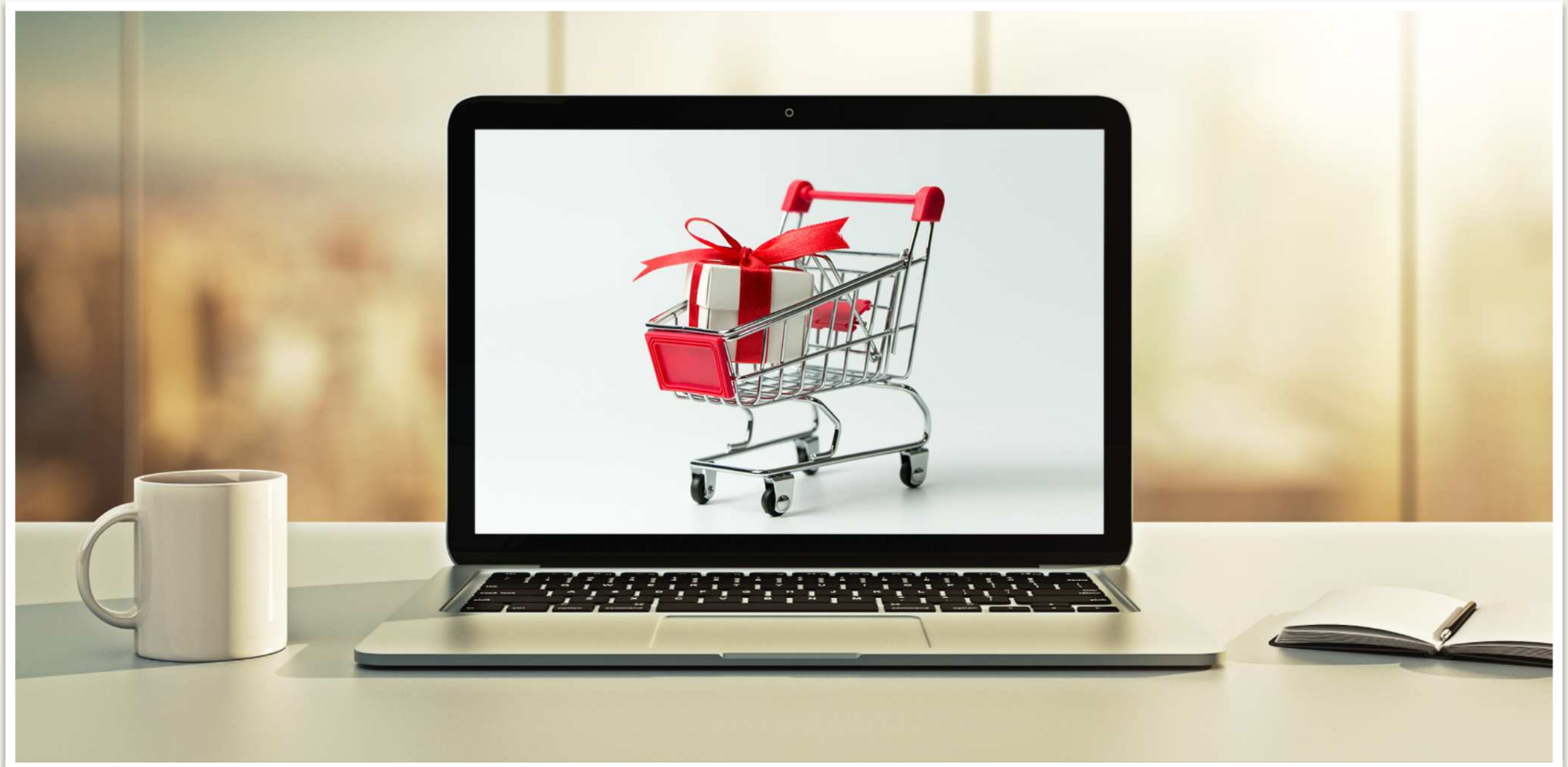
Similar code for years

File: checkout.component.ts

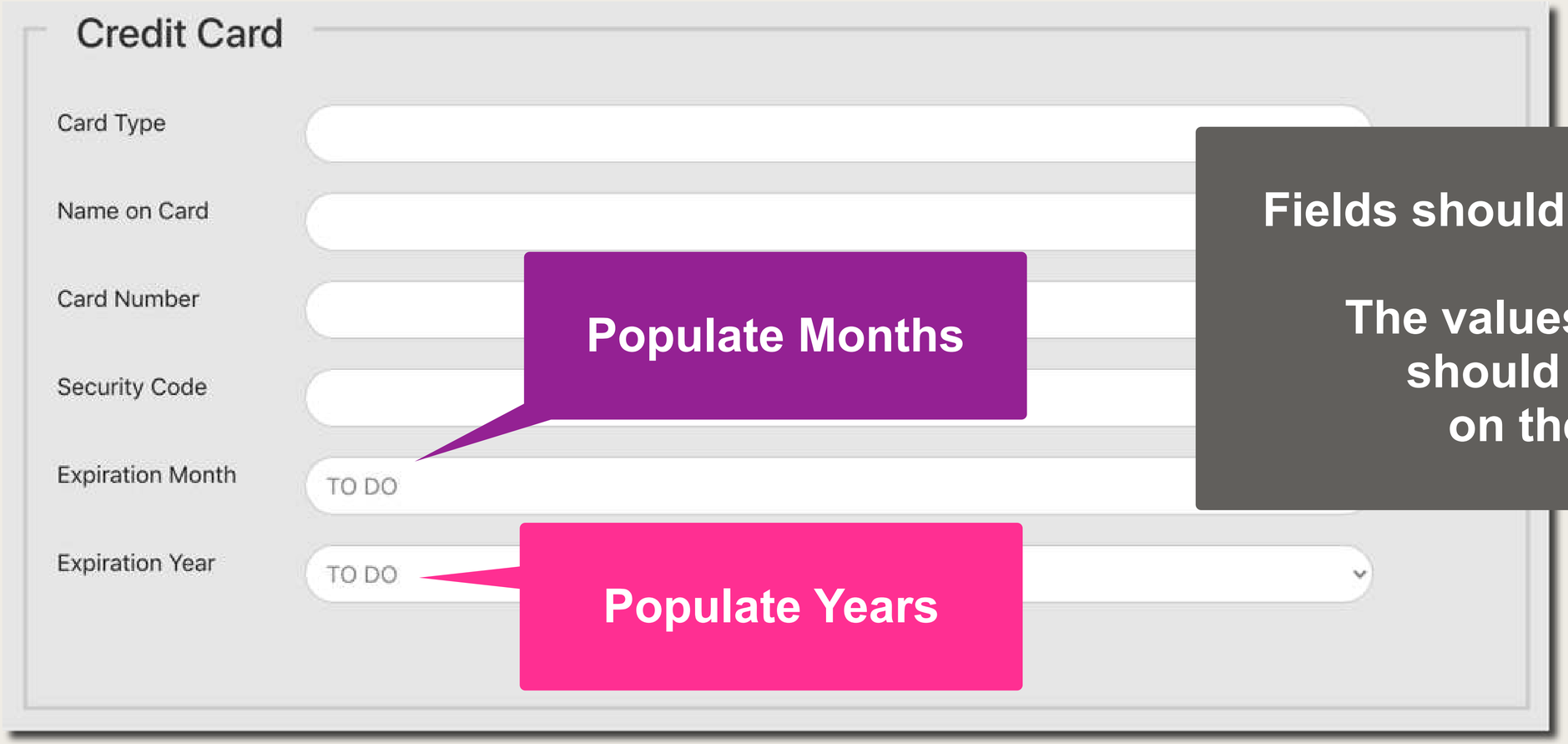
```
export class CheckoutComponent implements OnInit {

  ...
  creditCardYears: number[] = [];
```


© luv2code LLC

Checkout Form
Dependent Fields



Dependent Fields



Fields should be dependent

The values of month should depend on the year

© luv2code LLC

Dependent Fields

- Values for month should depend on the year

- If the current year is selected
  - Then only show the remaining months for the year
  - Start at current month to 12

- If a future year is selected
  - Then show months: 1 - 12

© luv2code LLC

Development Process

Step-By-Step

1. Update HTML template
    1. For the Expiration Years drop-down list
    2. Add event binding for change event

2. Update checkout component, add event handler
    1. Read the selected year
    2. Update the list of months based on selected year

© luv2code LLC

Step 1: Update HTML template for event binding

File: checkout.component.html

```
<div class="col-md-2"> <label>Expiration Year</label></div>

<div class="input-space">
    <select formControlName="expirationYear" (change)="handleMonthsAndYears()">
      <option *ngFor="let year of creditCardYears">
        {{ year }}
      </option>
    </select>
</div>
```

For `change` event
call method
`handleMonthsAndYears()`

© luv2code LLC

Step 2: Update checkout component, add event handler

File: checkout.component.ts

```
handleMonthsAndYears() {

  const creditCardFormGroup = this.checkoutFormGroup.get('creditCard');

  const currentYear: number = new Date().getFullYear();
  const selectedYear: number = Number(creditCardFormGroup.value.expirationYear);

  // if the current year equals the selected year, then start with current month

  let startMonth: number;

  if (currentYear === selectedYear) {
    startMonth = new Date().getMonth() + 1;
  }
  else {
    startMonth = 1;
  }

  this.luv2ShopFormService.getCreditCardMonths(startMonth).subscribe(
    data => {
      console.log("Retrieved credit card months: " + JSON.stringify(data));
      this.creditCardMonths = data;
    }
  );

}
```

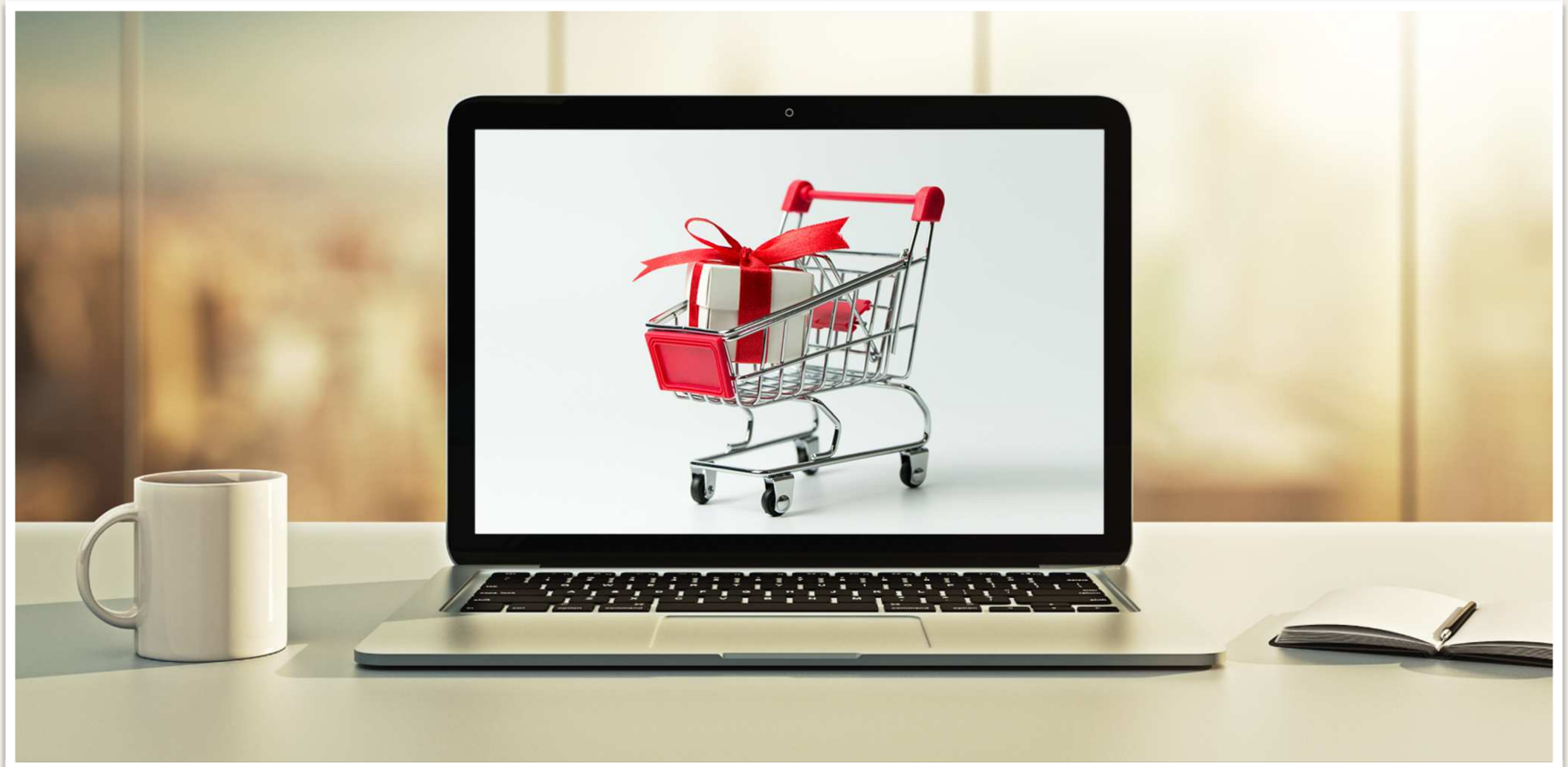
Read the selected year from the form

Get the current month

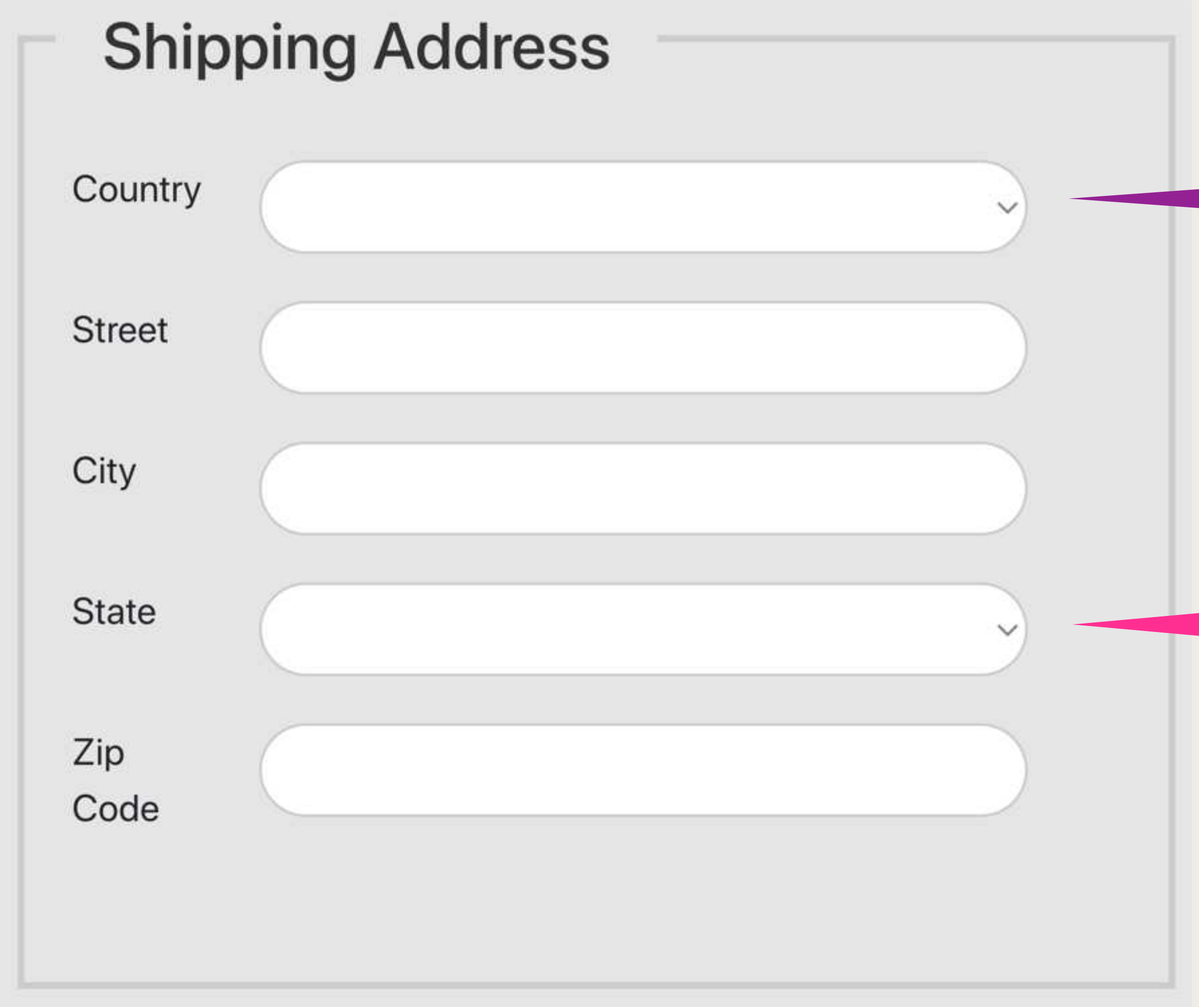
Get the credit card months

© luv2code LLC

Checkout Form
Populate Country-State Drop-Down Lists



Address - Country and State



Populate Country

Populate State

© luv2code LLC

Populating Drop-Down Lists

- Populate countries and states from the backend REST API (database)
- The user will select a country
- Depending on country selected, populate list of states
- Term "State" is different in each country
  - Similar to "Province", "District", "Region", "Préfecture" etc ...

© luv2code LLC

Development Process - Backend

Step-By-Step

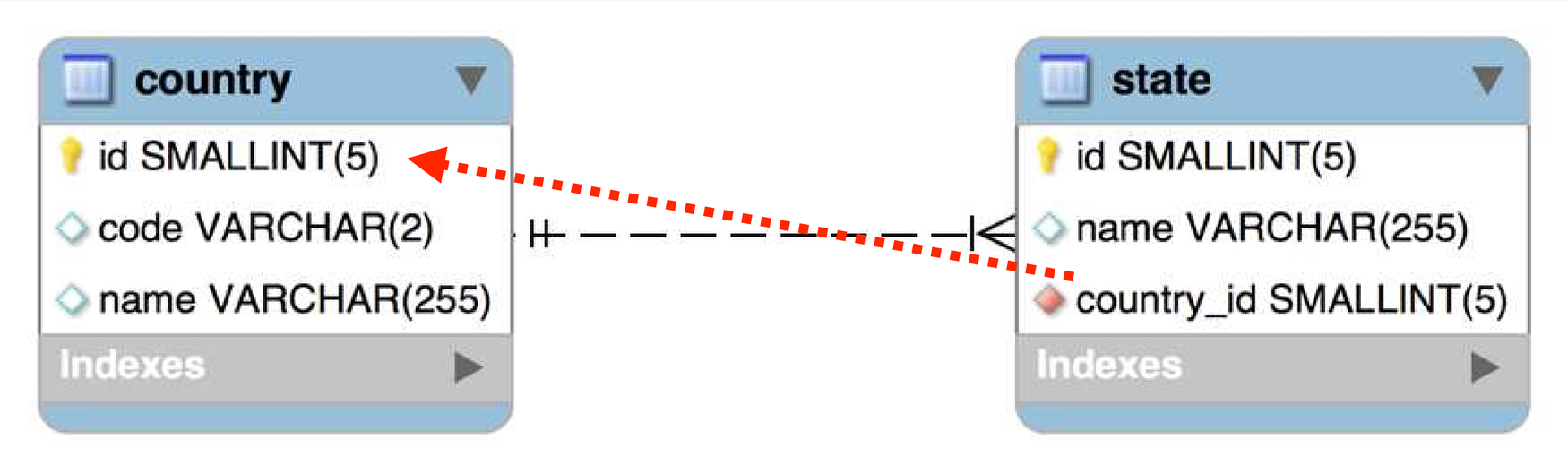
1. Create database tables
2. Develop JPA Entities (Country, State)
3. Create Spring Data Repositories
4. Update Spring Data REST Configs

© luv2code LLC

Step 1: Create Database Tables

One country has many states

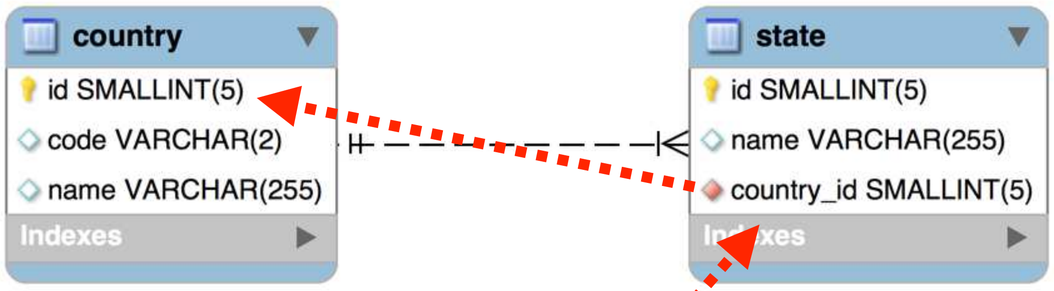
Many states belong to one country



© luv2code LLC

Step 1: Create Database Tables

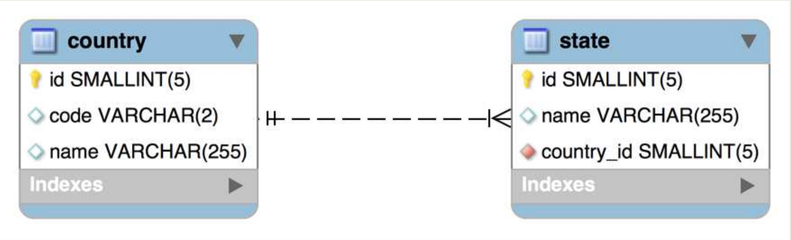
```
CREATE TABLE country (
  id smallint unsigned NOT NULL,
  code varchar(2) DEFAULT NULL,
  name varchar(255) DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB;
```




```
CREATE TABLE state (
  id smallint unsigned NOT NULL AUTO_INCREMENT,
  name varchar(255) DEFAULT NULL,
  country_id smallint unsigned NOT NULL,
  PRIMARY KEY (id),
  KEY fk_country (country_id),
  CONSTRAINT fk_country FOREIGN KEY (country_id) REFERENCES country (id)
) ENGINE=InnoDB AUTO_INCREMENT=1;
```


© luv2code LLC

Step 1: Create Database Tables




```
INSERT INTO `country` VALUES
(1,'BR','Brazil'),
(2,'CA','Canada'),
(3,'DE','Germany'),
(4,'IN','India'),
(5,'TR','Turkey'),
(6,'US','United States');
```

id, code, name

```
INSERT INTO `state` VALUES
(1,'Acre',1),
(2,'Alagoas',1),
(3,'Amapá',1),
...
(57,'Andhra Pradesh',4),
(58,'Arunachal Pradesh',4),
(59,'Assam',4),
(60,'Bihar',4),
(61,'Chhattisgarh',4),
(62,'Goa',4),
...
```

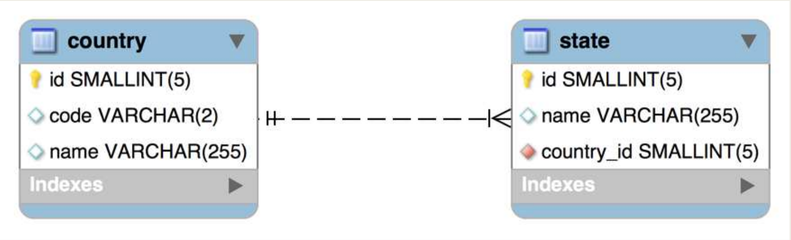
Brazil: 1

India: 4

id, name, country

© luv2code LLC

Step 1: Create Database Tables




```
SELECT * FROM country;
```

id	code	name
1	BR	Brazil
2	CA	Canada
3	DE	Germany
4	IN	India
5	TR	Turkey
6	US	United States

```
SELECT * FROM state where country_id=4;
```

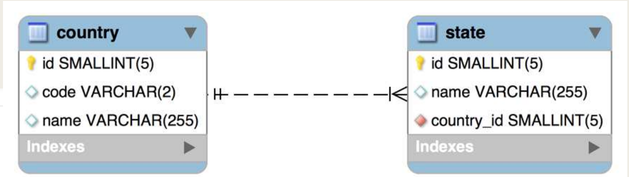
id	name	country_id
57	Andhra Pradesh	4
58	Arunachal Pradesh	4
59	Assam	4
60	Bihar	4
61	Chhattisgarh	4
62	Goa	4
63	Gujarat	4
64	Haryana	4

© luv2code LLC

Step 2: Develop JPA Entities: Country and State

```java
@Entity
@Table(name="country")
@Getter
@Setter
public class Country {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="code")
    private String code;

    @Column(name="name")
    private String name;

    @OneToMany(mappedBy="country")
    private List<State> states;

}
```

@OneToMany
One country has many states




```java
@Entity
@Table(name="state")
@Data
public class State {

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="name")
    private String name;

    @ManyToOne
    @JoinColumn(name="country_id")
    private Country country;

}
```

@ManyToOne
Many states belong to one country

© luv2code LLC

Step 3: Create Spring Data REST Repositories

```
@CrossOrigin("http://localhost:4200")
@RepositoryRestResource(collectionResourceRel = "countries", path = "countries")
public interface CountryRepository extends JpaRepository<Country, Integer> {

}
```

Expose /countries endpoint

To retrieve all countries

`http://localhost:8080/api/countries`

To retrieve country id=4

`http://localhost:8080/api/countries/4`

© luv2code LLC

Step 3: Create Spring Data REST Repositories

To retrieve all countries

`http://localhost:8080/api/countries`


```
{
  "_embedded" : {
    "countries" : [ {
      "id" : 1,
      "code" : "BR",
      "name" : "Brazil",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/countries/1"
        },
        "country" : {
          "href" : "http://localhost:8080/api/countries/1"
        }
      }
    }, {
      "id" : 2,
      "code" : "CA",
      "name" : "Canada",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/countries/2"
        },
        "country" : {
          "href" : "http://localhost:8080/api/countries/2"
        }
      }
    }, {
```


To retrieve country id=4

`http://localhost:8080/api/countries/4`


```
{
  "id" : 4,
  "code" : "IN",
  "name" : "India",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/countries/4"
    },
    "country" : {
      "href" : "http://localhost:8080/api/countries/4"
    }
  }
}
```


© luv2code LLC

Step 3: Create Spring Data REST Repositories

```
@CrossOrigin("http://localhost:4200")
@RepositoryRestResource
public interface StateRepository extends JpaRepository<State, Integer> {

}
```

Expose `/states` endpoint

To retrieve all states

`http://localhost:8080/api/states`

© luv2code LLC

Step 3: Create Spring Data REST Repositories

```
@CrossOrigin("http://localhost:4200")
@RepositoryRestResource
public interface StateRepository extends JpaRepository<State, Integer> {

   List<State> findByCountryCode(@Param("code") String code);

}
```

India (IN)

To retrieve states for a given country code

http://localhost:8080/api/states/search/findByCountryCode?code=IN

http://localhost:8080/api/states/search/findByCountryCode?code=US

United States (US)

© luv2code LLC

Step 3: Create Spring Data REST Repositories

To retrieve states for a given country code

`http://localhost:8080/api/states/search/findByCountryCode?code=IN`

`http://localhost:8080/api/states/search/findByCountryCode?code=US`

code=IN


```
{
  "_embedded" : {
    "states" : [ {
      "id" : 57,
      "name" : "Andhra Pradesh",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/57"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/57"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/57/country"
        }
      }
    }, {
      "id" : 58,
      "name" : "Arunachal Pradesh",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/58"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/58"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/58/country"
        }
      }
```


code=US


```
{
  "_embedded" : {
    "states" : [ {
      "id" : 93,
      "name" : "Alabama",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/93"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/93"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/93/country"
        }
      }
    }, {
      "id" : 94,
      "name" : "Alaska",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/94"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/94"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/94/country"
        }
      }
```


© luv2code LLC

Step 4: Update Spring Data REST Configs

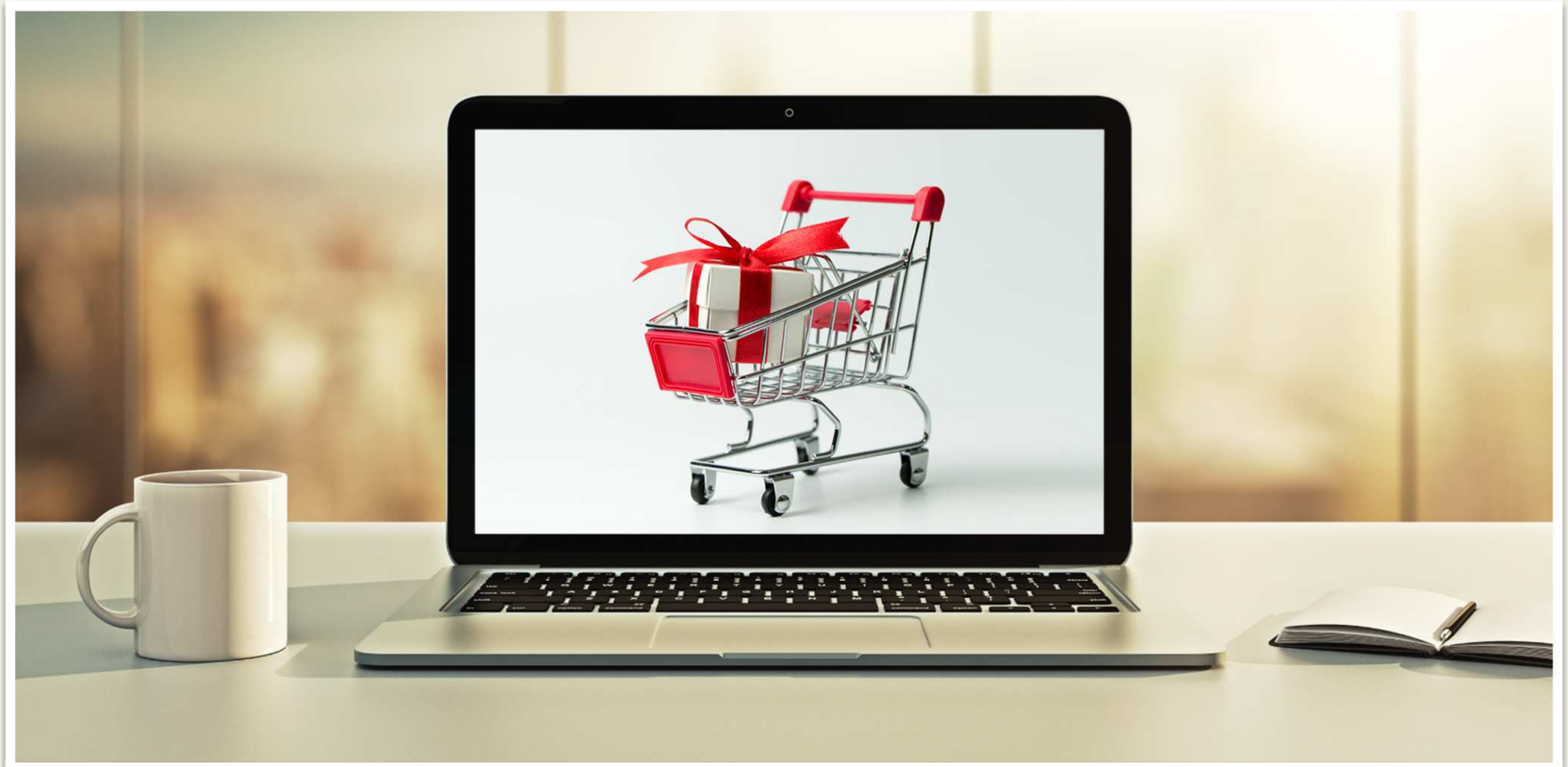
- Update our coding for MyDataRestConfig.java

- Make the APIs for `/country` and `/state` read-only
  - This is reference data, no need to change it via REST API
  - Disable HTTP: PUT, POST, etc ...

- Coding details covered in next video

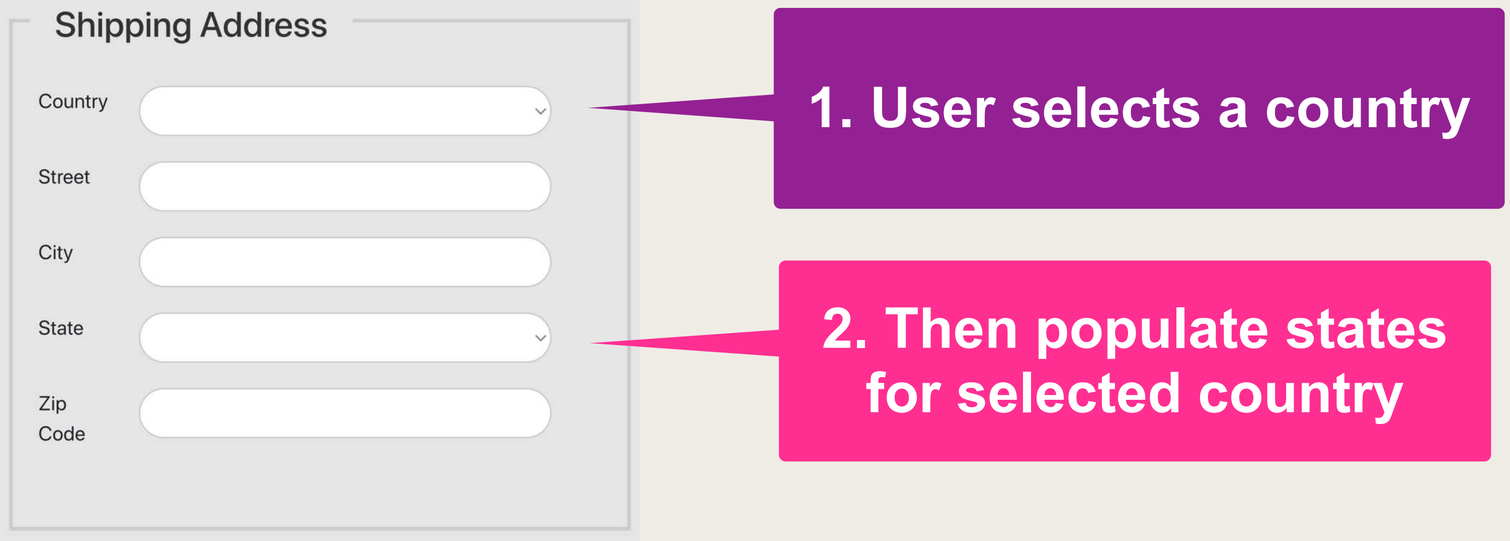
© luv2code LLC

Checkout Form
Populate Country-State Drop-Down Lists



Populating Drop-Down Lists

- Backend work for REST API is done ... now let's handle front-end
- The user will select a country
- Depending on country selected, populate list of states



© luv2code LLC

Development Process - Frontend

Step-By-Step

1. Create TypeScript classes for: `Country` and `State`
2. Add methods to the form service for countries and states
3. Update checkout component to retrieve the countries from service
4. Update HTML template to populate drop-down lists for countries
5. Add event handler for checkout component
    1. Read the selected country, retrieve list of states based on selected country
6. Update HTML template to populate drop-down lists for states

© luv2code LLC

Backend REST APIs

http://localhost:8080/api/countries


```
{
  "_embedded" : {
    "countries" : [ {
      "id" : 1,
      "code" : "BR",
      "name" : "Brazil",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/countries/1"
        },
        "country" : {
          "href" : "http://localhost:8080/api/countries/1"
        }
      }
    }, {
      "id" : 2,
      "code" : "CA",
      "name" : "Canada",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/countries/2"
        },
        "country" : {
          "href" : "http://localhost:8080/api/countries/2"
        }
      }
    }, {
```


http://localhost:8080/api/states/search/findByCountryCode?code=IN


```
{
  "_embedded" : {
    "states" : [ {
      "id" : 57,
      "name" : "Andhra Pradesh",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/57"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/57"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/57/country"
        }
      }
    }, {
      "id" : 58,
      "name" : "Arunachal Pradesh",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/states/58"
        },
        "state" : {
          "href" : "http://localhost:8080/api/states/58"
        },
        "country" : {
          "href" : "http://localhost:8080/api/states/58/country"
        }
      }
    }, {
```


© luv2code LLC

Step 1: Create TypeScript classes for: Country and State

File: country.ts

```
export class Country {
    id: number;
    code: string;
    name: string;
}
```

File: state.ts

```
export class State {
    id: number;
    name: string;
}
```

JSON from REST API


```
{
  "id" : 4,
  "code" : "IN",
  "name" : "India",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/countries/4"
    },
    "country" : {
      "href" : "http://localhost:8080/api/countries/4"
    },
    "states" : {
      "href" : "http://localhost:8080/api/countries/4/states"
    }
  }
}
```


JSON from REST API


```
{
  "id" : 57,
  "name" : "Andhra Pradesh",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/states/57"
    },
    "state" : {
      "href" : "http://localhost:8080/api/states/57"
    },
    "country" : {
      "href" : "http://localhost:8080/api/states/57/country"
    }
  }
}
```


© luv2code LLC

Step 2: Add methods to the form service for countries and states

File: luv2-shop-form.service.ts

```
export class Luv2ShopFormService {

  private countriesUrl = 'http://localhost:8080/api/countries';
  private statesUrl = 'http://localhost:8080/api/states/';

  constructor() { }

  getStates(theCountryCode: string): Observable<State[]> {

    // search url
    const searchStatesUrl = `${this.statesUrl}/search/findByCountryCode?code=${theCountryCode}`;

    return this.httpClient.get<GetResponseStates>(searchStatesUrl).pipe(
      map(response => response._embedded.states)
    );
  }
  ...
}

interface GetResponseStates {
  _embedded: {
    states: State[];
  }
}
```

Get States

Returns an observable

Map the JSON data from Spring Data REST to `State` array

Unwraps the JSON from Spring Data REST `_embedded` entry

© luv2code LLC

Step 3: Update checkout component to retrieve countries from service

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ...
  countries: Country[] = [];

  constructor(private formBuilder: FormBuilder,
              private luv2ShopFormService: Luv2ShopFormService) { }

  ngOnInit(): void {
    ...

    // populate countries
    this.luv2ShopFormService.getCountries().subscribe(
      data => {
        console.log("Retrieved countries: " + JSON.stringify(data));
        this.countries = data;
      }
    );

  }

}
```

When form is initially displayed, populate the countries

© luv2code LLC

Step 4: Update HTML template to populate drop-down lists for countries

File: checkout.component.html

```
<div class="col-md-2"> <label>Country</label></div>

<div class="input-space">
     <select formControlName="country">
        <option *ngFor="let country of countries" [ngValue]="country" >
          {{ country.name }}
        </option>
     </select>
</div>
```

Loop over array

© luv2code LLC

Step 5: Update checkout component, add event handler

For change event call method getStates(..)

File: checkout.component.html

```
<div class="col-md-2"> <label>Country</label></div>

<div class="input-space">
     <select formControlName="country" (change)="getStates('shippingAddress')">
        <option *ngFor="let country of countries" [ngValue]="country" >
          {{ country.name }}
        </option>
     </select>
</div>
```


© luv2code LLC

Step 5: Update checkout component, add event handler

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  checkoutFormGroup: FormGroup;
  shippingAddressStates: State[] = [];
  billingAddressStates: State[] = [];

  getStates(formGroupName: string) {

    const formGroup = this.checkoutFormGroup.get(formGroupName);
    const countryCode = formGroup.value.country.code;

    this.formDataService.getStates(countryCode).subscribe(
      data => {

        if (formGroupName === 'shippingAddress') {
          this.shippingAddressStates = data;
        }
        else {
          this.billingAddressStates = data;
        }

        // select first item by default
        formGroup.get('state').setValue(data[0]);
      }
    );
  }
```

Read the selected country code from the form

Retrieve the states for the selected country code

© luv2code LLC

Step 6: Update HTML template to populate drop-down lists for states

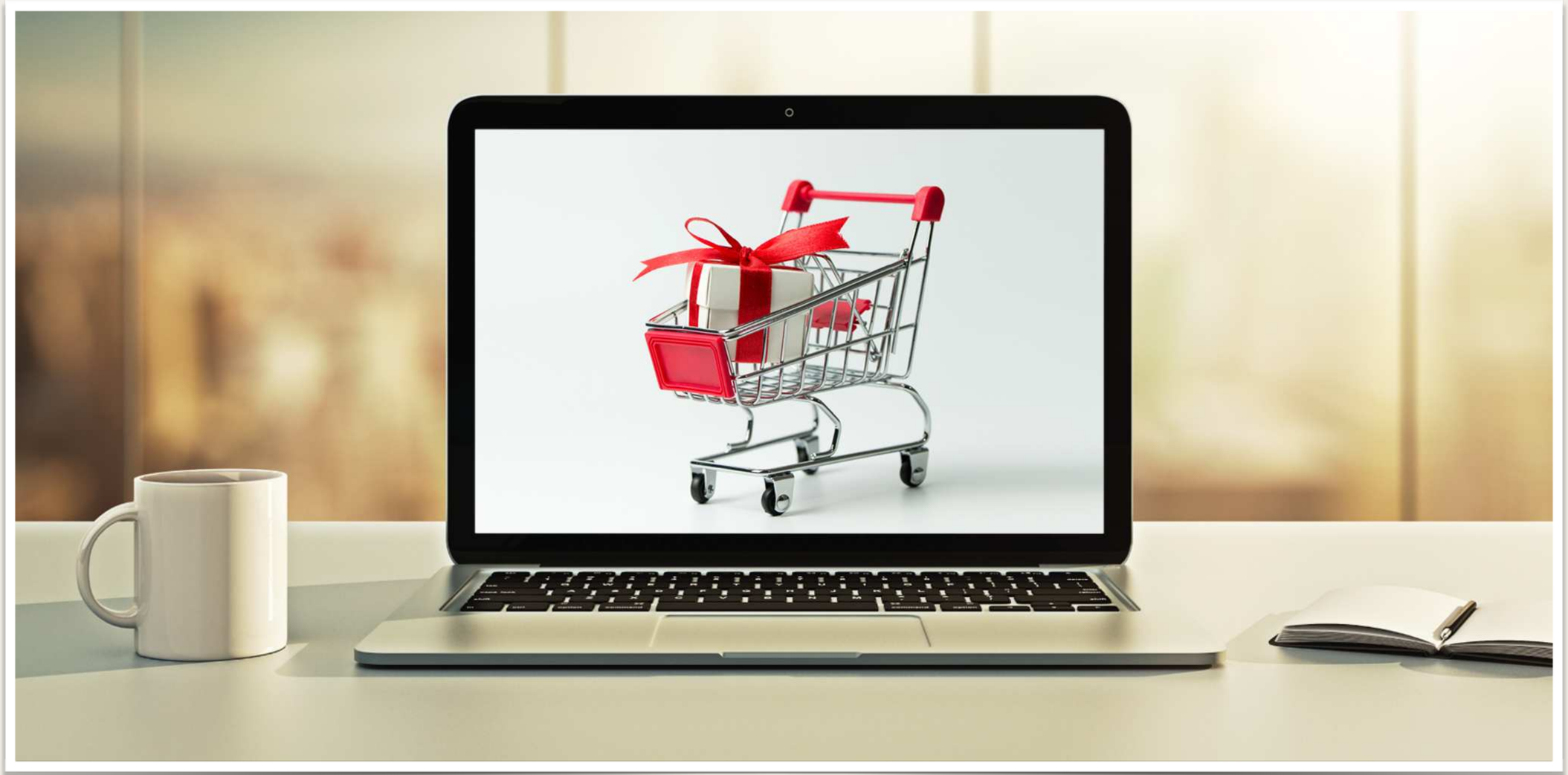
File: checkout.component.html

```
<div class="col-md-2"> <label>State</label></div>

<div class="input-space">
     <select formControlName="state">
        <option *ngFor="let state of shippingAddressStates" [ngValue]="state" >
          {{ state.name }}
        </option>
     </select>
</div>
```

Loop over array

© luv2code LLC

Checkout Form Validation



Form Validation

- Before we submit the form to the backend ... let's perform validation

- Check for required fields, min length etc ...

© luv2code LLC

Angular Validation

- Angular has a set of built-in validation rules

Name	Description
required	Must be a non-empty value
min	Must be a number >= value
max	Must be a number <= value
minLength	Length must be >= value
maxLength	Length must be <= value
pattern	Must match a regular expression pattern
email	Match email against common regular expression patterns
others …	

https://angular.io/api/forms/Validators

© luv2code LLC

Additional Validation Features

- Define custom validators
- Cross-field validation
- Asynchronous validators

https://angular.io/guide/form-validation

© luv2code LLC

Checkout Form Validation

From: The Boss

- Our requirements
  - All fields are required
  - Email address: has proper email format
  - Credit card field: only numbers allowed (16 digits)
  - CVC number: only numbers allowed (3 digits)

© luv2code LLC

Development Process

Step-By-Step

1. Specify validation rules for the form controls
2. Define Getter methods to access form controls
3. Update HTML template to display error messages
4. Add event handler to check validation status when submit button clicked

© luv2code LLC

Step 1: Specify validation rules for form controls

- Recall that a form field is represented by FormControl object
- Create the FormControl and pass in initial value and validators

Basic Syntax

```
new FormControl(initialValue, validators, ...)
```

array of validators

Example

```
new FormControl('', [Validators.required, Validators.minLength(2)])
```

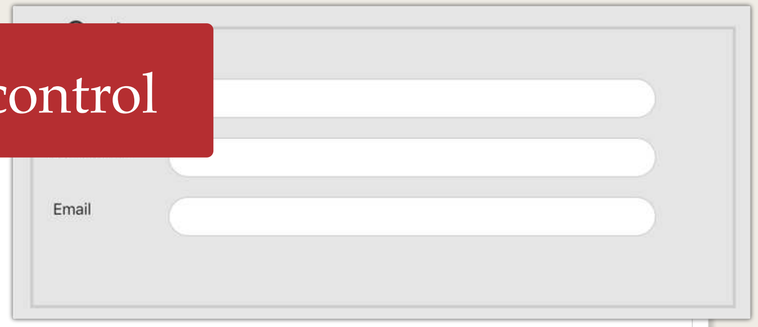

© luv2code LLC

Step 1: Specify validation rules for form controls

File: checkout.component.ts

```
    this.checkoutFormGroup = this.formBuilder.group({

      customer: this.formBuilder.group({
        'firstName': new FormControl('',
          [Validators.required, Validators.minLength(2)]
        ),

        'lastName': new FormControl('',
          [Validators.required, Validators.minLength(2)]
        ),

        'email': new FormControl('',
          [Validators.required,
          Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,4}$')] // works for anuva@gmail.com
        )
      }),
  ...
  });
```

firstName form control



Regular Expression

© luv2code LLC

Validators.email vs Validators.pattern

- You may wonder why we aren't using Angular: `Validators.email` ???
- `Validators.email` only checks for: `<someText>@<moreText>`
  - Based on this ... the following is valid email format:
    - `anil@luv2code`
    - `anil@gmail`

Too lenient :-(

- It does not check for domain name format: `anil@luv2code.com` or `anil@gmail.com`
- As a result, we use a regular expression with `Validators.pattern`

© luv2code LLC

Validators.email vs Validators.pattern

- You may wonder why we aren't using Angular: `Validators.email` ???
- `Valid` … `ext>`
  - Base
    - `an`
    - `an`

Validators.email and Validators.pattern

Only check the FORMAT

Does not verify if email address is real

- It does not check for domain name format: `anil@luv2code.com` or `anil@gmail.com`
- As a result, we use a regular expression with `Validators.pattern`

© luv2code LLC

Step 2: Define Getter methods to access form controls

File: checkout.component.ts

```
get firstName() {
  return this.checkoutFormGroup.get('customer.firstName');
}

get lastName() {
  return this.checkoutFormGr
}

get email() {
  return this.checkoutFormGr
}

...
```

```
this.checkoutFormGroup = this.formBuilder.group({

  customer: this.formBuilder.group({
    'firstName': new FormControl('', [
      Validators.required, Validators.minLength(2)
    ]),

    'lastName': new FormControl('', [
      Validators.required, Validators.minLength(2)
    ]),

    'email': new FormControl('', [
      Validators.required,
      Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a
    ])
  }),
...
});
```


© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

If validation fails
then display error messages

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
  First Name must be at least 2 characters long
 </div>

</div>
```


© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

invalid: did validations fail?

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
  First Name must be at least 2 characters long
 </div>

</div>
```


© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

dirty: did user change field value?

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

</div>
```

Only display validation errors if user has interacted with the form
- When the user changes field value, the control is marked as "dirty"
- When the field loses focus, the control is marked as "touched"

© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

touched: did field lose focus?

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

</div>
```

Only display validation errors if user has interacted with the form
- When the user changes field value, the control is marked as "dirty"
- When the field loses focus, the control is marked as "touched"

© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required">
  First Name is required
 </div>

 <div *ngIf="firstName.errors
  First Name must be at least
 </div>

</div>
```

errors object contains list of validations that failed for this form control

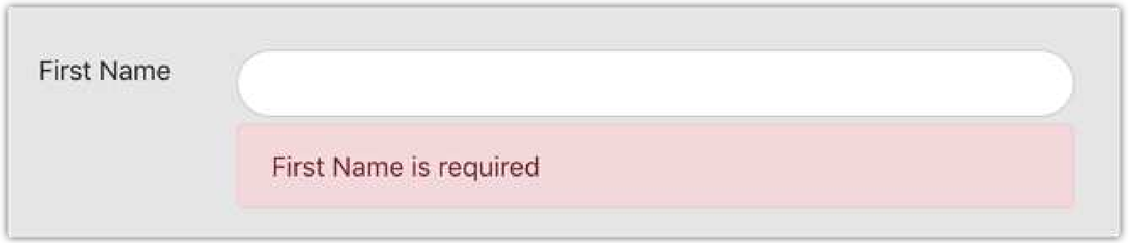
© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
  First Name must be at least 2 characters long
 </div>

</div>
```

Display specific error message



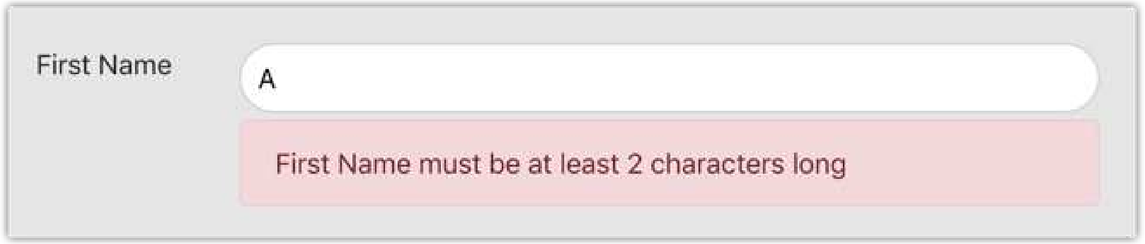
© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
  First Name must be at least 2 characters long
 </div>

</div>
```

Display specific error message



© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

Email address field

```
<label>Email</label>
<input formControlName="email" type="text">

<div *ngIf="email.invalid && (email.dirty || email.touched)" class="alert alert-danger">

 <div *ngIf="email.errors.required">
  Email is required
 </div>

 <div *ngIf="email.errors.pattern">
  Email must be a valid email address
 </div>

</div>
```

Display error messages

© luv2code LLC

Step 4: Add event handler to check validation on submit

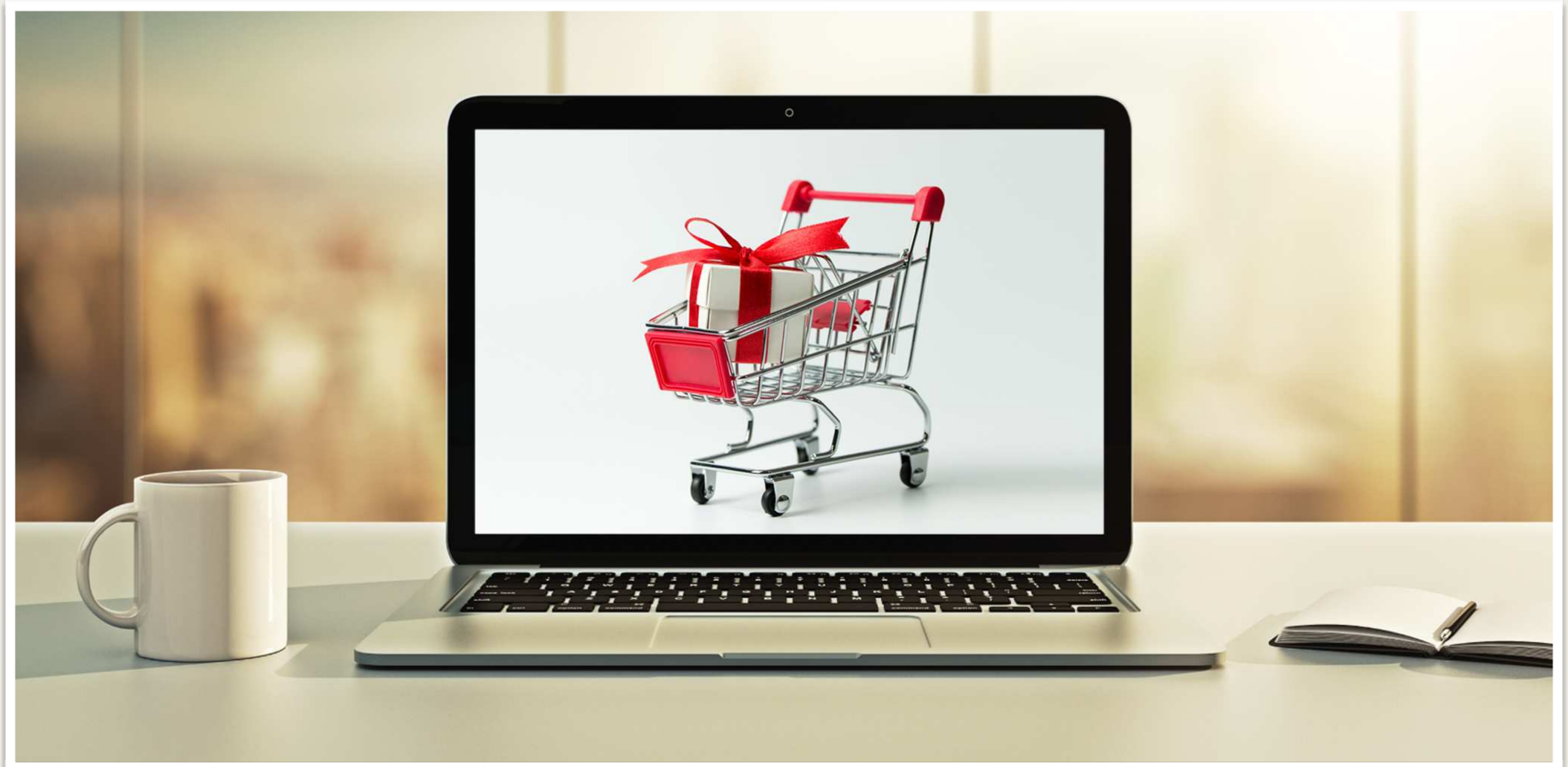
File: checkout.component.ts

```
onSubmit() {
  console.log("Handling the submit button");
  ...

  if (this.checkoutFormGroup.invalid) {
    this.checkoutFormGroup.markAllAsTouched();
  }

  console.log("CheckoutFormGroup is valid: " + this.checkoutFormGroup.valid);
}
```

Touching all fields triggers the display of the error messages

© luv2code LLC

Checkout Form
Custom Validator Rule



White Space

- Our form currently has a problem with white space
- In the required fields
  - If you enter ONLY whitespace ... it **passes** ... YIKES!
  - Should **fail**!

© luv2code LLC

Custom Validator

- We can resolve this with a custom validator: **`notOnlyWhitespace`**
  - Check the field value
  - If it ONLY has whitespace
    - then return an error (failed)
    - else return null (passed)

© luv2code LLC

Development Process

Step-By-Step

1. Define custom validator rule
2. Specify custom validator rule for the form controls
3. Update HTML template to display error messages

© luv2code LLC

Step 1: Define custom validator rule

Don't allow only whitespace

File: luv2-shop-validators.ts

```
import { FormControl, ValidationErrors } from '@angular/forms';

export class Luv2ShopValidators {

    // whitespace validation
    static notOnlyWhitespace(control: FormControl) : ValidationErrors {

        // check if string only contains whitespace
        if ((control.value != null) && (control.value.trim().length === 0)) {

          // invalid, return error object
          return { 'notOnlyWhitespace': true };
        }
        else {
          // valid, return null
          return null;
        }

    }
}
```

If validation check fails then return validation error(s)

If validation check passes then return null

Validation error key

The HTML template will check for this error key

© luv2code LLC

Step 2: Specify custom validator rule for form controls

File: checkout.component.ts

```
    this.checkoutFormGroup = this.formBuilder.group({

      customer: this.formBuilder.group({
        'firstName': new FormControl('',
          [Validators.required, Validators.minLength(2), Luv2ShopValidators.notOnlyWhitespace ]
        ),

        ...
    }),
  ...
  });
```

Our custom validator method name

```
export class Luv2ShopValidators {

    // whitespace validation
    static notOnlyWhitespace(control: FormControl) : ValidationErrors {

        // check if string only contains whitespace
        if ((control.value != null) && (control.value.trim().length === 0)) {
```


© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required || firstName.errors.notOnlyWhitespace">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
  First Name must be at least 2 characters long
 </div>

</div>
```

If validation fails
then display error messages

Reference to our custom
validation error key

© luv2code LLC

Step 3: Update HTML template to display error messages

File: checkout.component.html

```
<label>First Name</label>
<input formControlName="firstName" type="text">

<div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)" class="alert alert-danger">

 <div *ngIf="firstName.errors.required || firstName.errors.notOnlyWhitespace">
  First Name is required
 </div>

 <div *ngIf="firstName.errors.minlength">
```

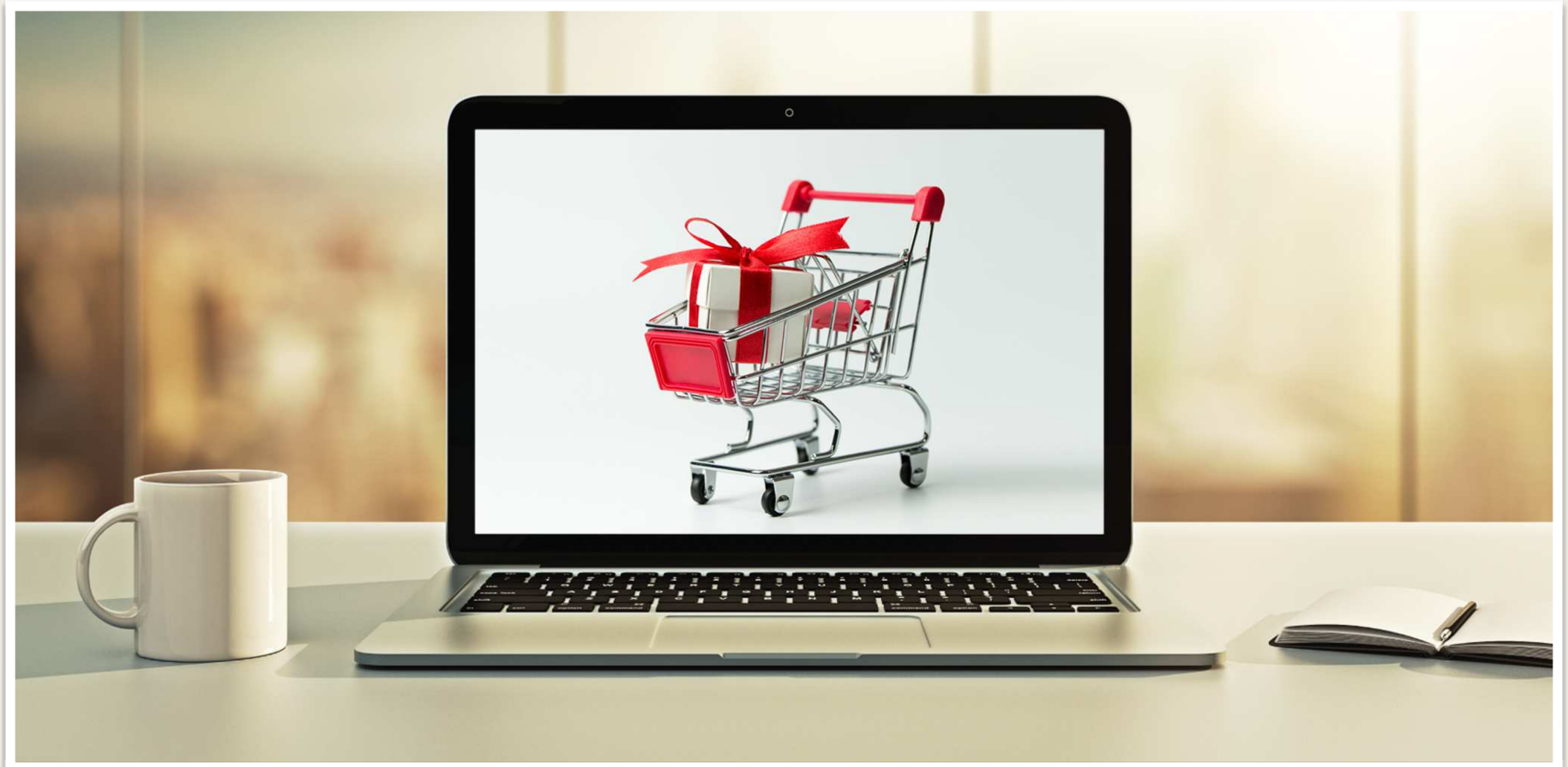
```
    // whitespace validation
    static notOnlyWhitespace(control: FormControl) : ValidationErrors {

        // check if string only contains whitespace
        if ((control.value != null) && (control.value.trim().length === 0)) {

          // invalid, return error object
          return { 'notOnlyWhitespace': true };
        }
        else {
```

If validation fails then display error messages

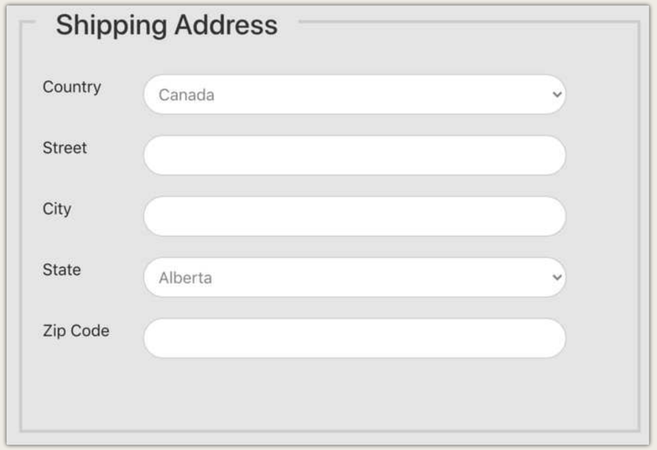
Reference to our custom validation error key

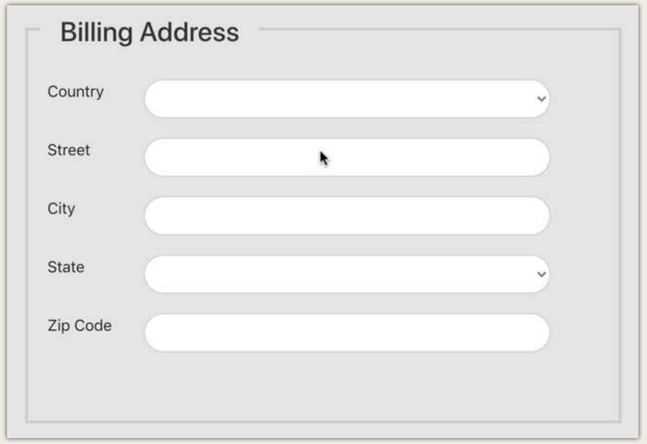
© luv2code LLC

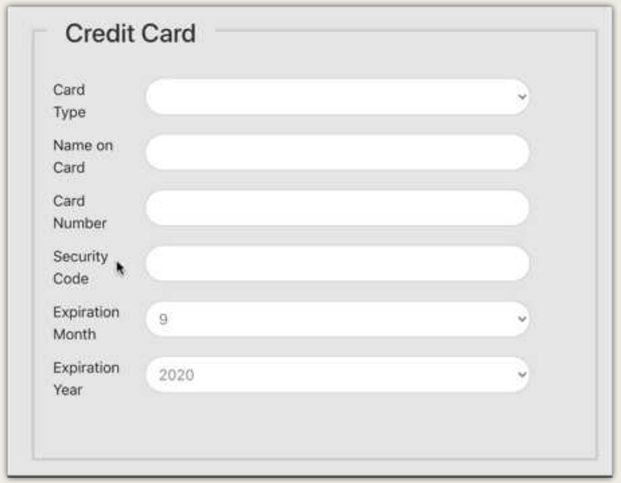
Checkout Form Validation
Shipping, Billing and Credit Card



Finish up the remaining sections







© luv2code LLC

Development Process

More of the same

Step-By-Step

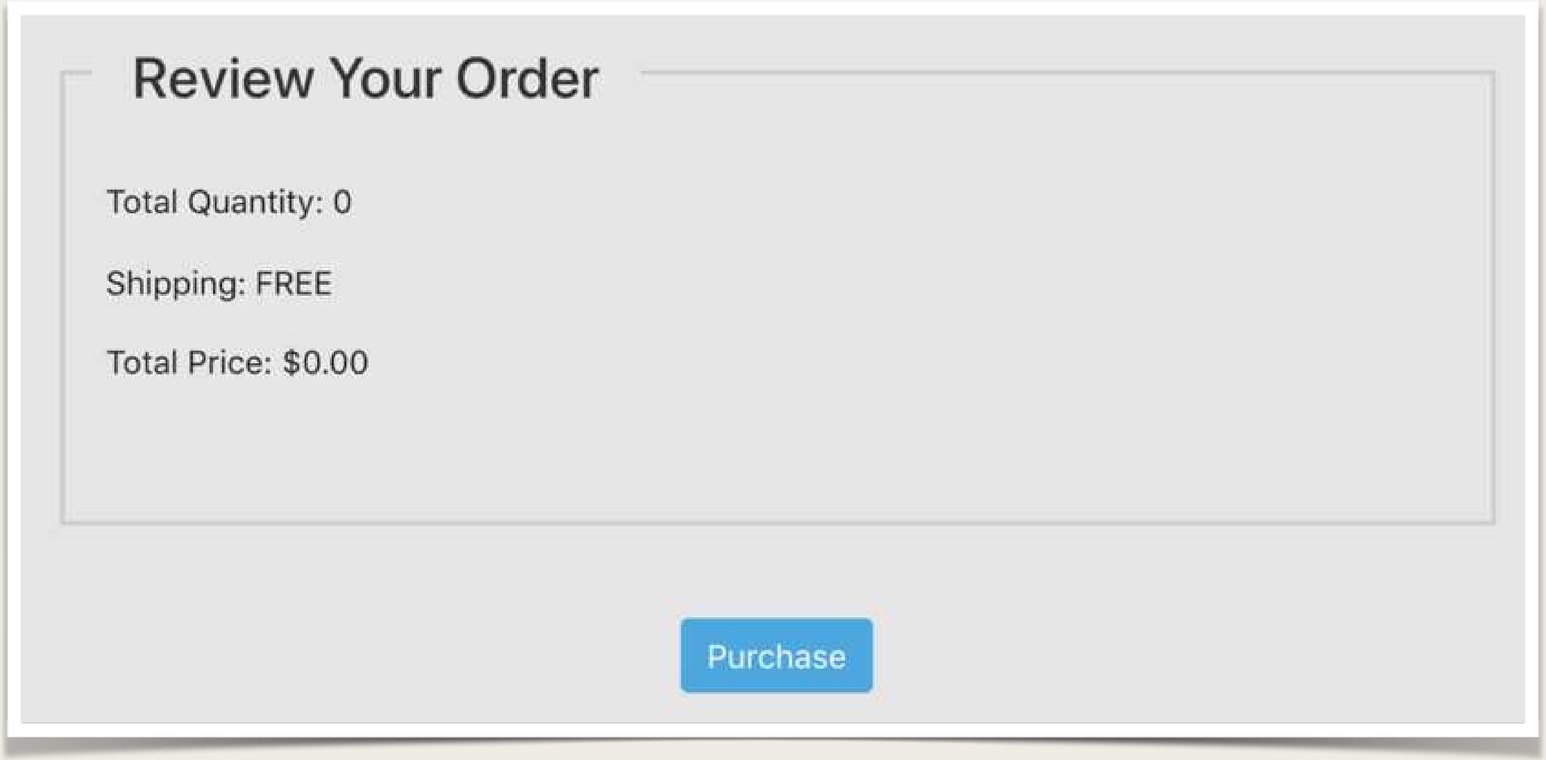
1. Specify validation rules for the form controls
2. Define Getter methods to access form controls
3. Update HTML template to display error messages

© luv2code LLC

Checkout Form
Review Cart Totals

Update Checkout Form - Cart Totals

- Our form currently has a section for cart totals
- We need to add the appropriate code to support this



© luv2code LLC

Cart Service - Publishing messages/events

- Recall that we send messages/events to other components in our application

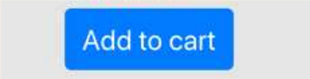
- For example, `CartStatusComponent` will **subscribe** to the `CartService`

- `CartService` will **publish** messages for:
  - `totalPrice` and `totalQuantity`

© luv2code LLC

Application Interaction

Remember



CartStatusComponent

ProductListComponent

1. subscribe for events

4. update UI for total price and quantity

2. addToCart(...)

3. publish events to all subscribers

CartService

© luv2code LLC

Remember

CartService

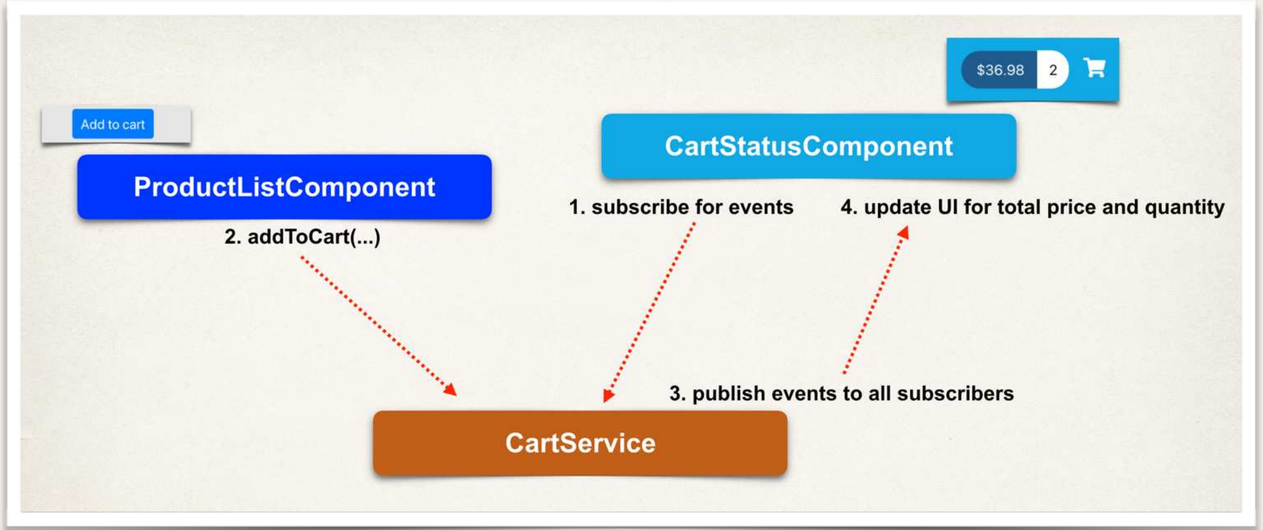
File: cart-service.ts

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new Subject<number>();
  totalQuantity: Subject<number> = new Subject<number>();

  ...
}
```

`Subject` is a subclass of `Observable`

We can use `Subject` to publish events in our code.

The event will be sent to all of the subscribers



© luv2code LLC

Remember

CartService - Publish events

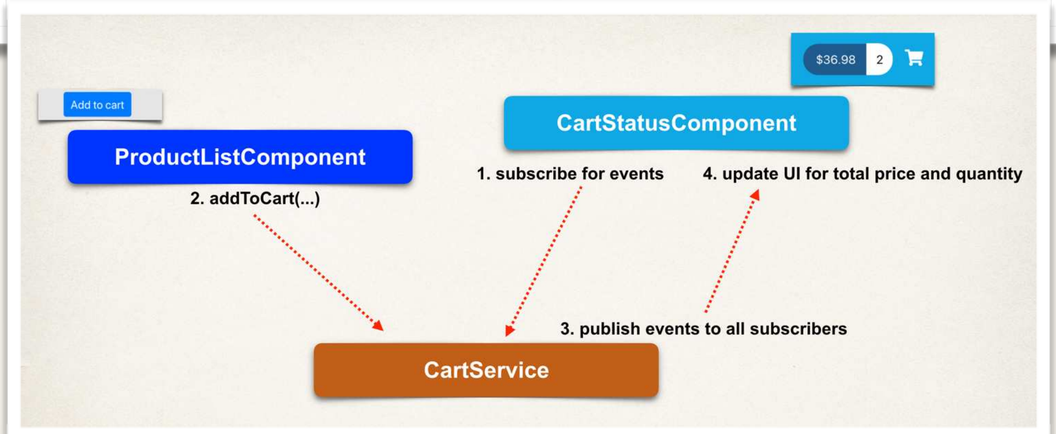
File: cart-service.ts

```
computeCartTotals() {

  let totalPriceValue: number = 0;
  let totalQuantityValue: number = 0;

  for (let currentCartItem of this.cartItems) {
    totalPriceValue += currentCartItem.quantity * currentCartItem.unitPrice;
    totalQuantityValue += currentCartItem.quantity;
  }

  // publish the new values ... all subscribers will receive the new data
  this.totalPrice.next(totalPriceValue);
  this.totalQuantity.next(totalQuantityValue);
}
```

Compute totals

This will publish events to all subscribers

one event for totalPrice
one event for totalQuantity



© luv2code LLC

Remember

CartStatusComponent subscribes to CartService

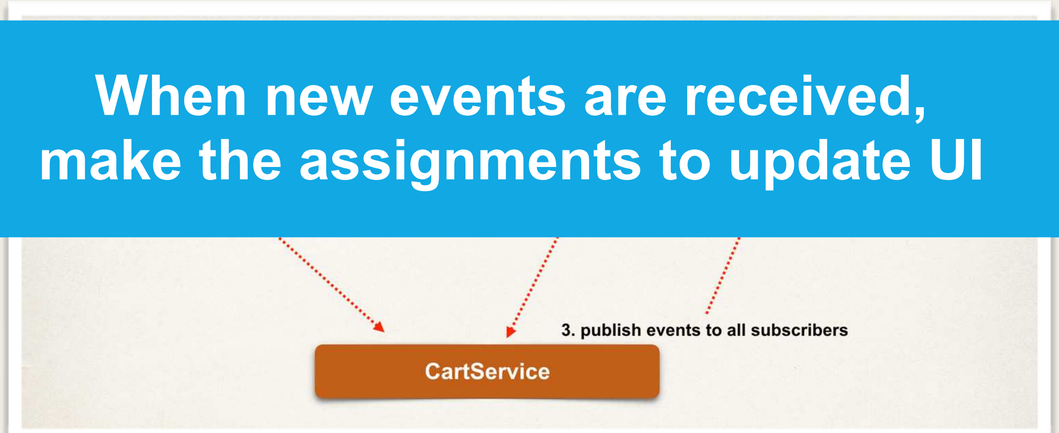
File: cart-status.component.ts

```
export class CartStatusComponent implements OnInit {

  totalPrice: number = 0.00;
  totalQuantity: number = 0;

  constructor(private cartService: CartService) { }

  ngOnInit(): void {
    this.updateCartStatus();
  }

  updateCartStatus() {

    // subscribe to the cart status totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart status totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );
  }
}
```

Inject the CartService

Subscribe for events

When new events are received, make the assignments to update UI



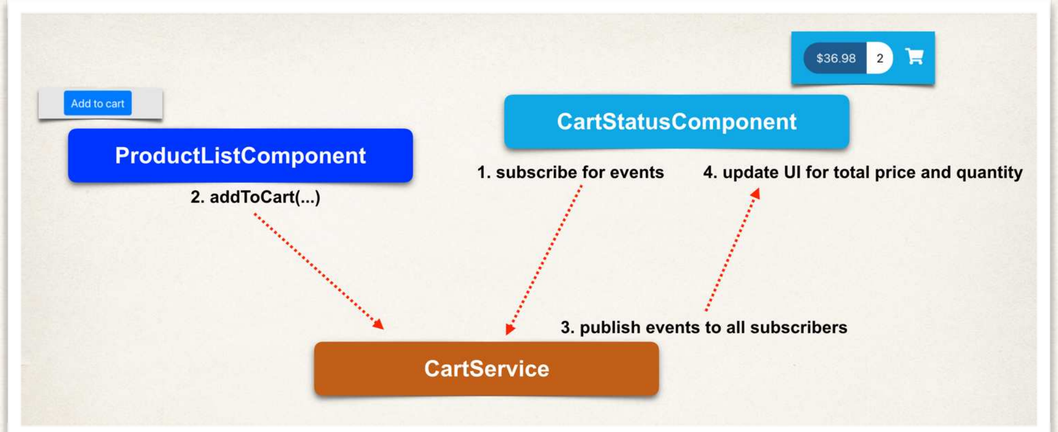
© luv2code LLC

Remember

CartStatusComponent subscribes to CartService

File: cart-status.component.ts

```
export class CartStatusComponent implements OnInit {

  totalPrice: number = 0.00;
  totalQuantity: number = 0;

  constructor(private cartService: CartService) { }

  ngOnInit(): void {
    this.updateCartStatus();
  }

  updateCartStatus() {

    // subscribe to the cart status totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart status totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );
  }
}
```

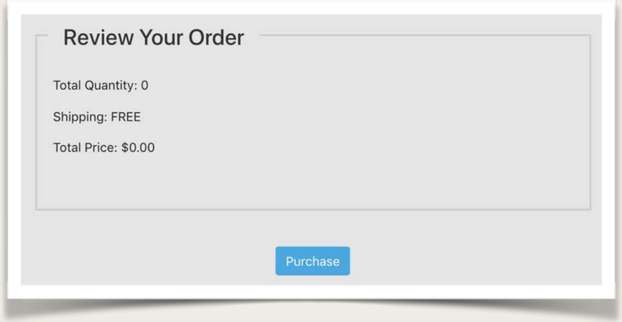


When new events are received, make the assignments to update UI

© luv2code LLC

Publish / Subscribe

- Similar approach for `CheckoutComponent` ... almost
- `CheckoutComponent` will subscribe to events from `CartService`
- However, since `CheckoutComponent` is instantiated later in the application
  - Will miss out on previous messages
- As a result, `CheckoutComponent` cart totals will erroneously show as
  - `0` for total quantity
  - `0.00` for total price



Need to get a replay of the messages missed

© luv2code LLC

Replay Messages

- Similar to the real world

Sorry, I am late to the meeting.
Can you tell me what I missed?

- We can get this functionality with `ReplaySubject`

© luv2code LLC

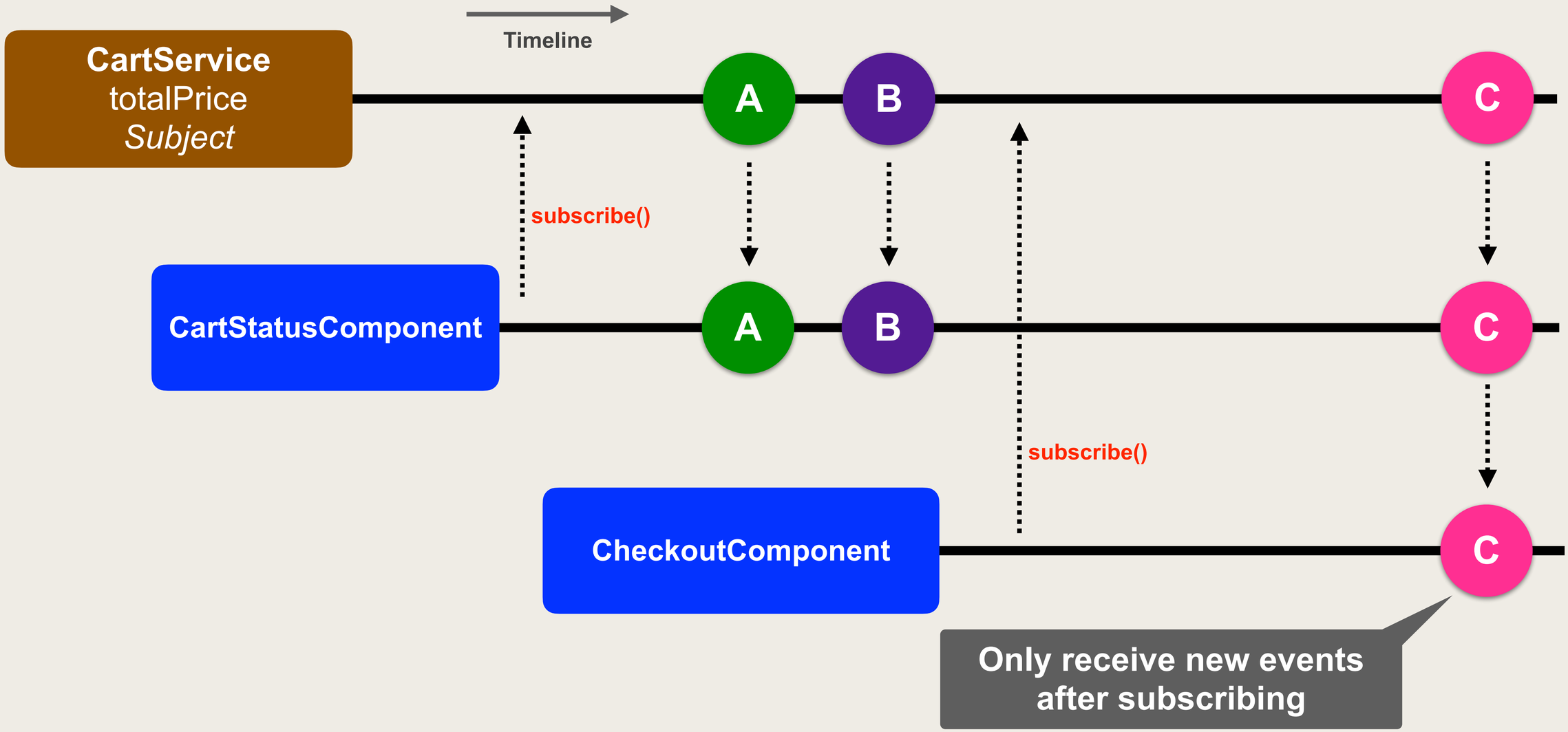
ReplaySubject

- Recall, `Subject` is used to send events to subscribers
- **`ReplaySubject`** is a subclass of `Subject`
  - Will also "replay events" for new subscribers who join later
  - Keep a buffer of previous events ... and send to **new subscribers**

Sorry, I am late to the meeting.
Can you tell me what I missed?

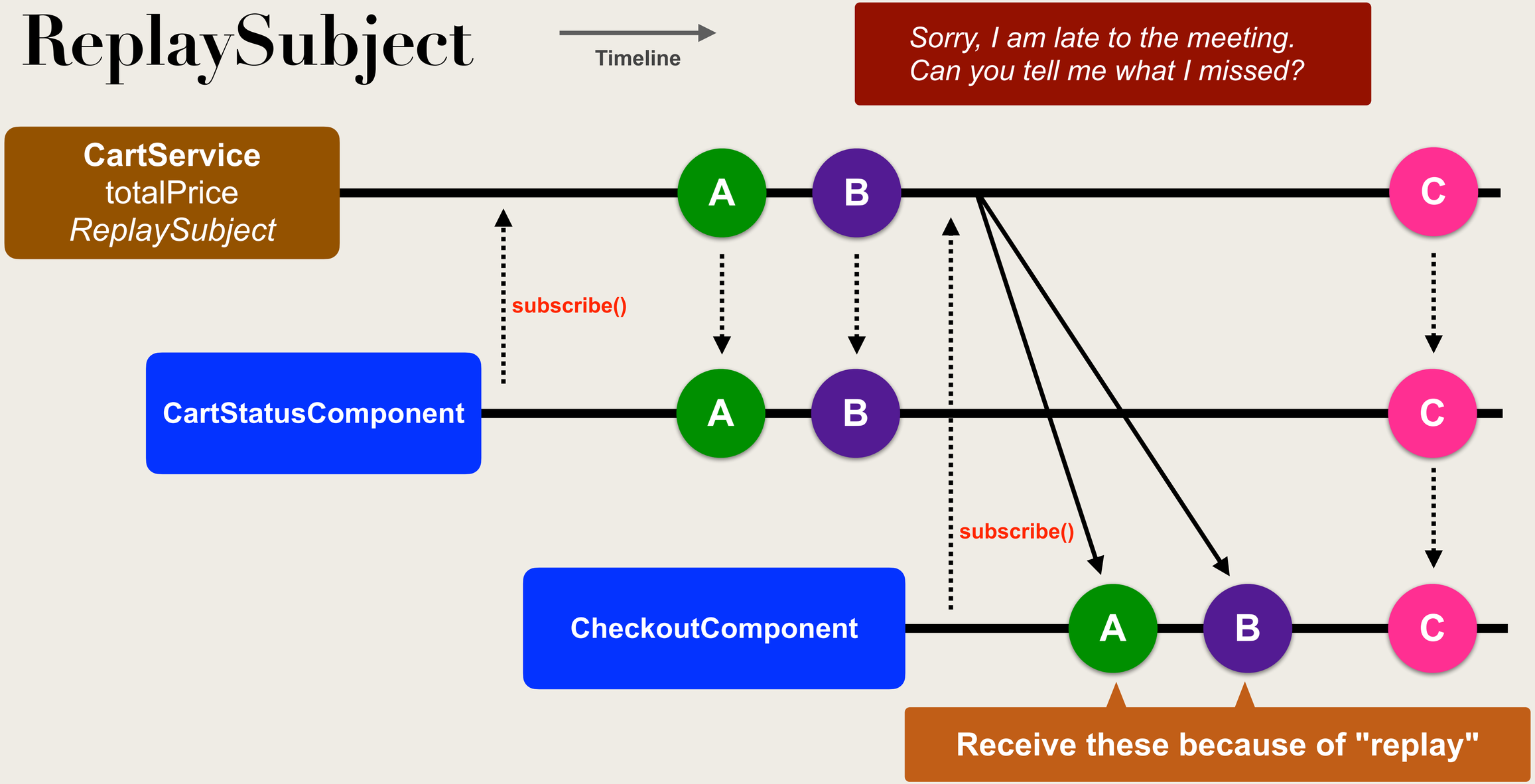
© luv2code LLC

Subject (without replay)



© luv2code LLC

ReplaySubject



© luv2code LLC

ReplaySubject

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new ReplaySubject<number>();
  totalQuantity: Subject<number> = new ReplaySubject<number>();

  ...
}
```

Keep a buffer of previous events
Send previous events to new subscribers

© luv2code LLC

ReplaySubject

- For more details on `ReplaySubject`, see online docs

http://www.luv2code.com/rxjs-replay-subject

© luv2code LLC

But wait ... there's another solution

- At first glance, `ReplaySubject` seems like the perfect solution
- However, for totals, we really don't need to replay the previous totals
- We are only interested in the latest total ... the last computed total.
- We are only interested in the last event/message.

Sorry, I am late to the meeting.
What is the latest cart total?

© luv2code LLC

BehaviorSubject

- **`BehaviorSubject`** is a subclass of `Subject`
  - Has a notion of "current value"
  - Stores the latest message/event ... and sends to **new subscribers**

Sorry, I am late to the meeting.
What is the latest cart total?

© luv2code LLC

From the Docs

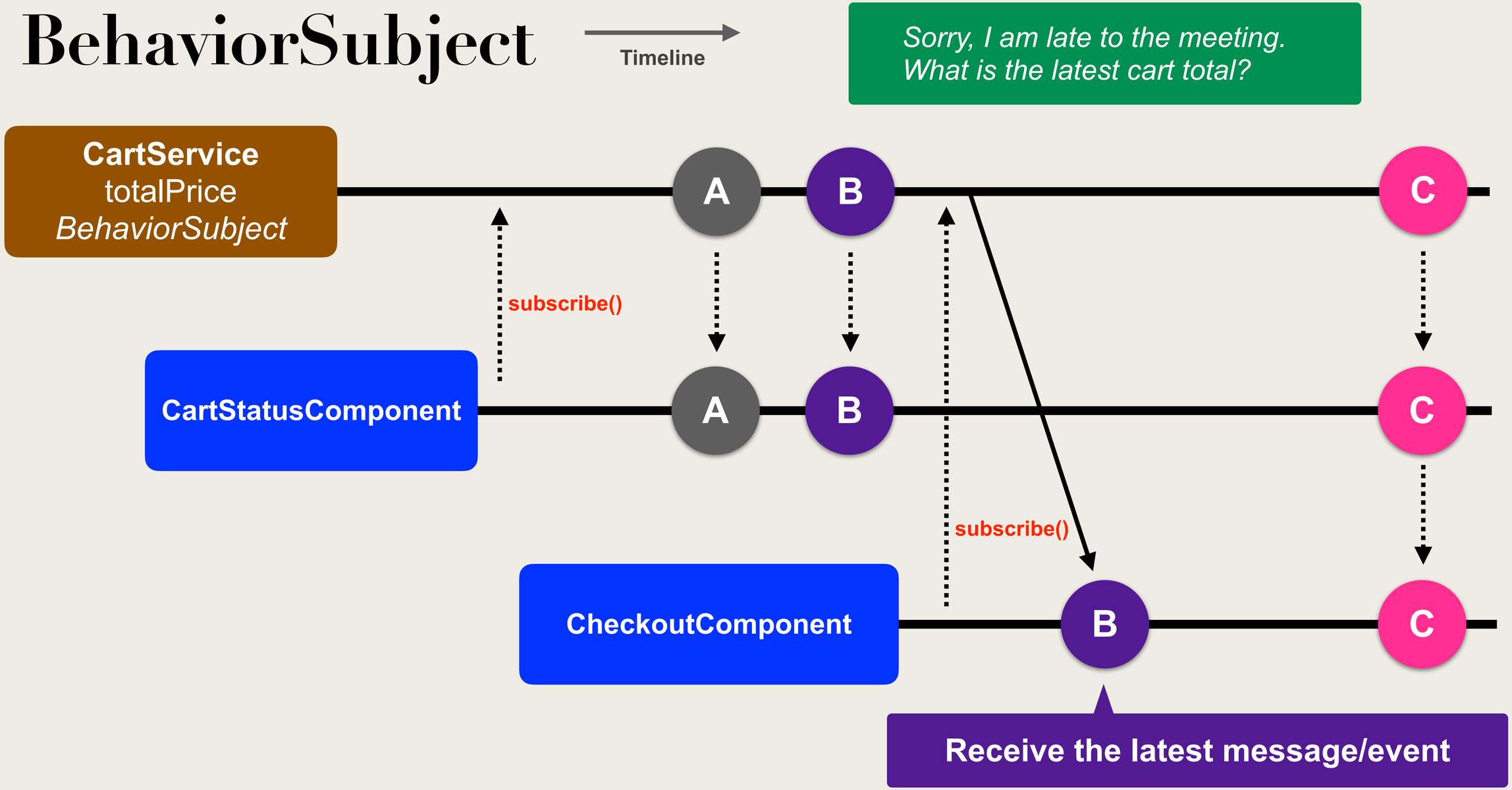
`BehaviorSubject`s are useful for representing "values over time".

For instance, an event stream of birthdays is a `Subject`, but the stream of a person's age would be a `BehaviorSubject`.

http://www.luv2code.com/rxjs-behavior-subject

© luv2code LLC

BehaviorSubject



© luv2code LLC

BehaviorSubject

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new BehaviorSubject<number>(0);
  totalQuantity: Subject<number> = new BehaviorSubject<number>(0);

  ...
}
```

Set initial value

totalPrice = 0
totalQuantity = 0

© luv2code LLC

BehaviorSubject

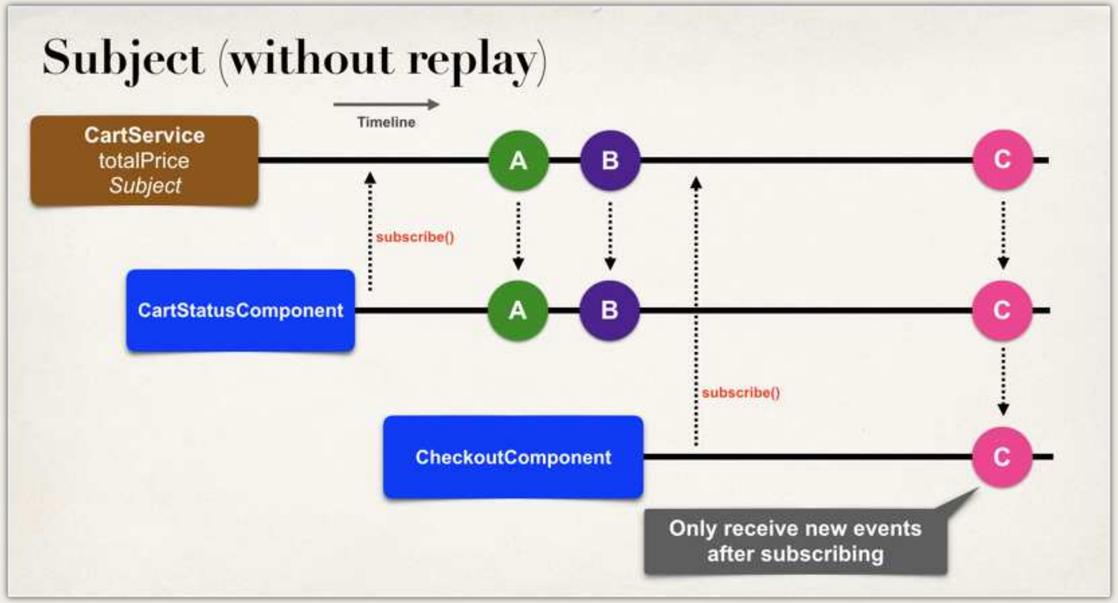
- For more details on `BehaviorSubject`, see online docs

http://www.luv2code.com/rxjs-behavior-subject

© luv2code LLC

Recap

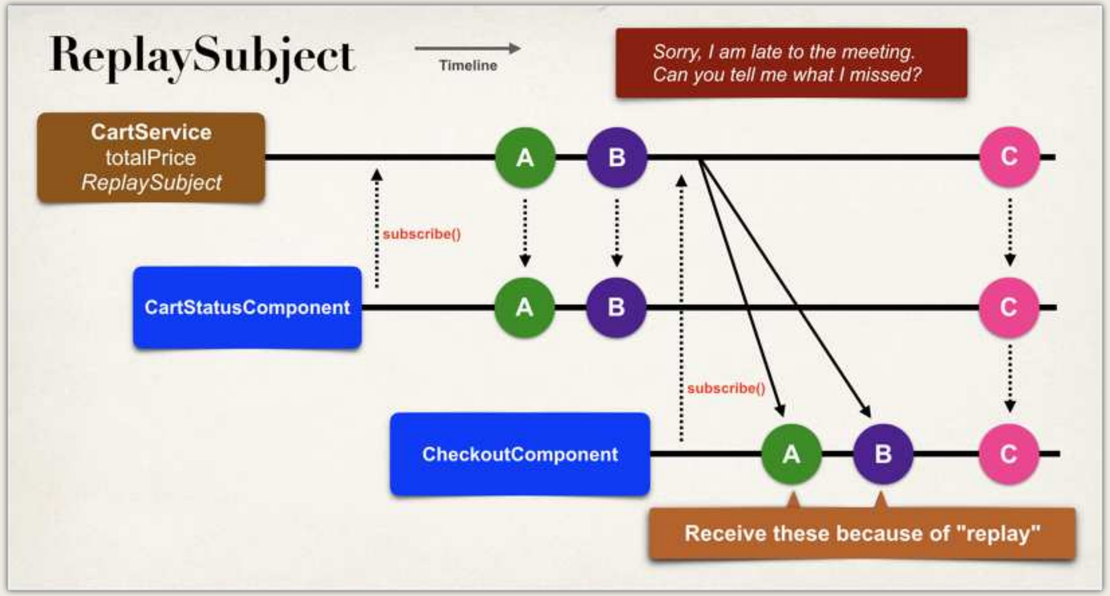
Name	Description
`Subject`	• Does not keep a buffer of previous events • Subscriber only receives new events after they are subscribed



© luv2code LLC

Recap

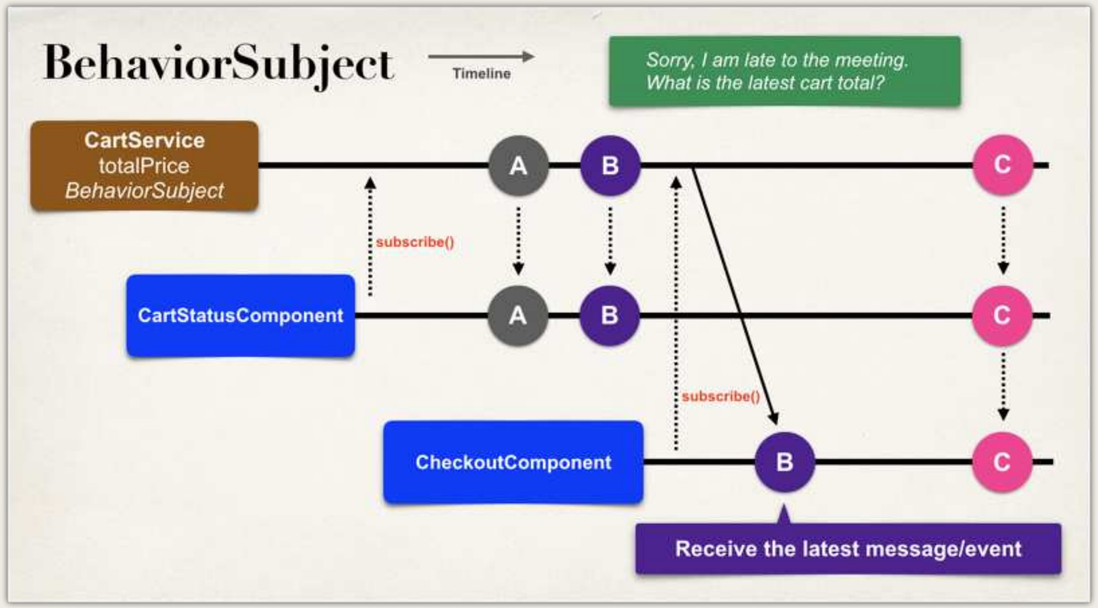
Name	Description
`ReplaySubject`	• Has a buffer of all previous events • Once subscribed, subscriber receives a replay of all previous events



© luv2code LLC

Recap

Name	Description
`BehaviorSubject`	• Has a buffer of the last event • Once subscribed, subscriber receives the latest event sent prior to subscribing



© luv2code LLC

Recap

Name	Description
`Subject`	• Does not keep a buffer of previous events • Subscriber only receives new events after they are subscribed
`ReplaySubject`	• Has a buffer of all previous events • Once subscribed, subscriber receives a replay of all previous events
`BehaviorSubject`	• Has a buffer of the last event • Once subscribed, subscriber receives the latest event sent prior to subscribing

© luv2code LLC

Development Process

Step-By-Step

1. Updates for `CheckoutComponent`
    1. Inject `CartService` into `CheckoutComponent`
    2. In `ngOnit` method, call new method: `reviewCartDetails()`
    3. Add code for new method: `reviewCartDetails()`

2. Update for `CartService`
    1. Change `Subject` to **`BehaviorSubject`**

© luv2code LLC

Step 1.1: Inject CartService into CheckoutForm

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  constructor(private formBuilder: FormBuilder,
              private luv2ShopFormService: Luv2ShopFormService,
              private cartService: CartService) { }

 ...
}
```

Inject CartService

© luv2code LLC

Step 1.2: ngOnit, call new method: reviewCartDetails

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ngOnInit(): void {

   this.reviewCartDetails();
   ...
  }

}
```

Call new method

© luv2code LLC

Step 1.3: New method: reviewCartDetails

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ngOnInit(): void {

   this.reviewCartDetails();
   ...
  }

  reviewCartDetails(){
    this.cartService.totalQuantity.subscribe(
      totalQuantity => this.totalQuantity = totalQuantity
    );

    this.cartService.totalPrice.subscribe(
       totalPrice => this.totalPrice = totalPrice
    );
  }

}
```

Subscribe to totalQuantity

Subscribe to totalPrice

© luv2code LLC

Step 2.1: Change Subject to BehaviorSubject

File: cart.service.ts

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new BehaviorSubject<number>(0);
  totalQuantity: Subject<number> = new BehaviorSubject<number>(0);

  ...
}
```

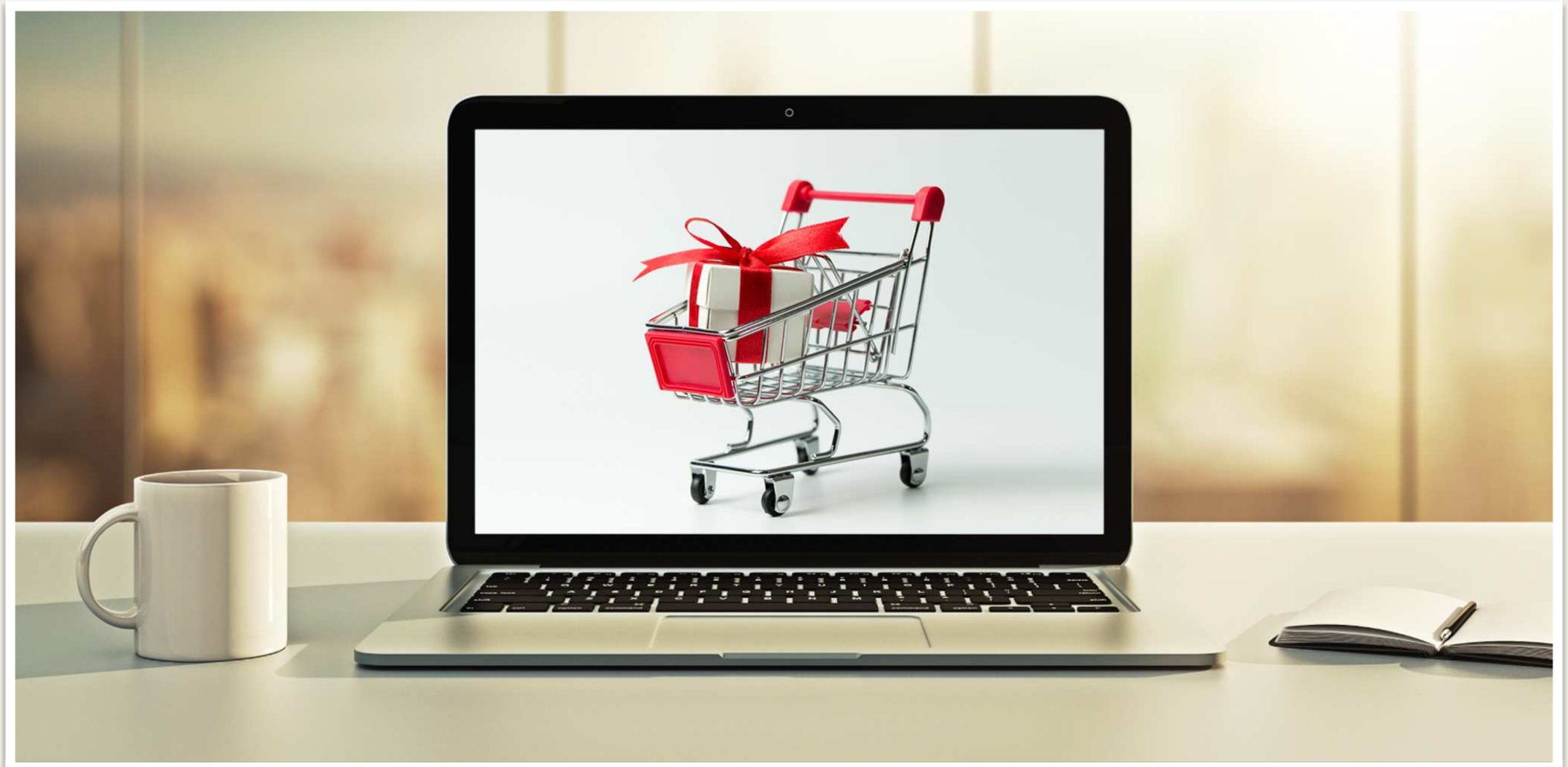
Change to BehaviorSubject

Set initial value

totalPrice = 0
totalQuantity = 0

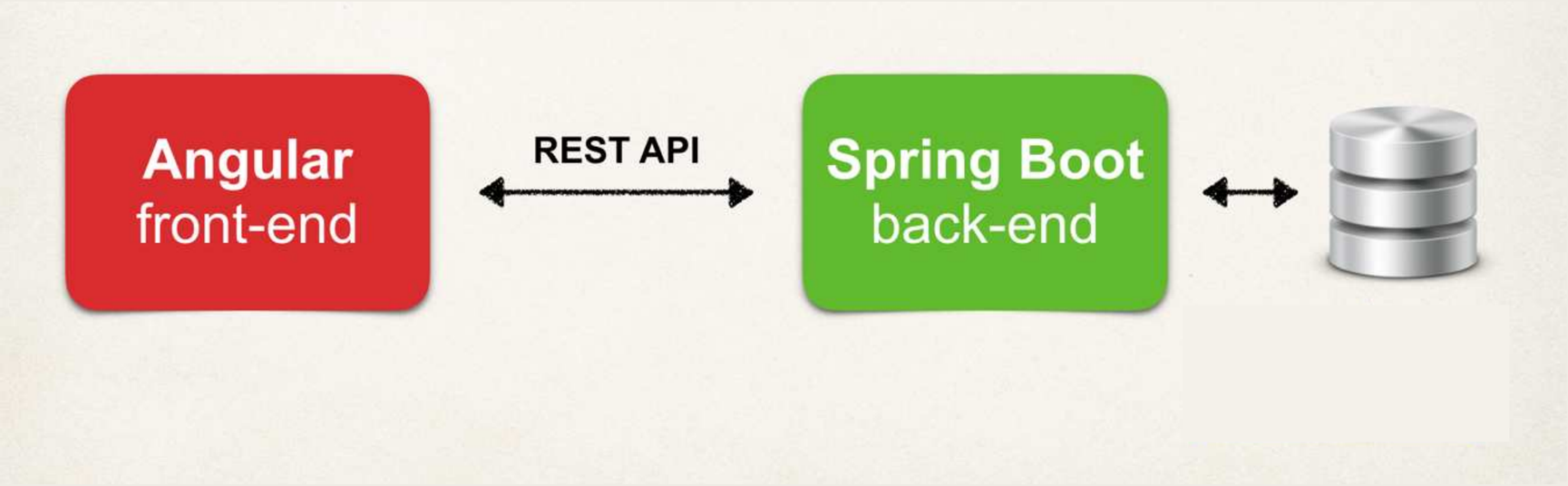
© luv2code LLC

Checkout Form
Save The Order - Backend



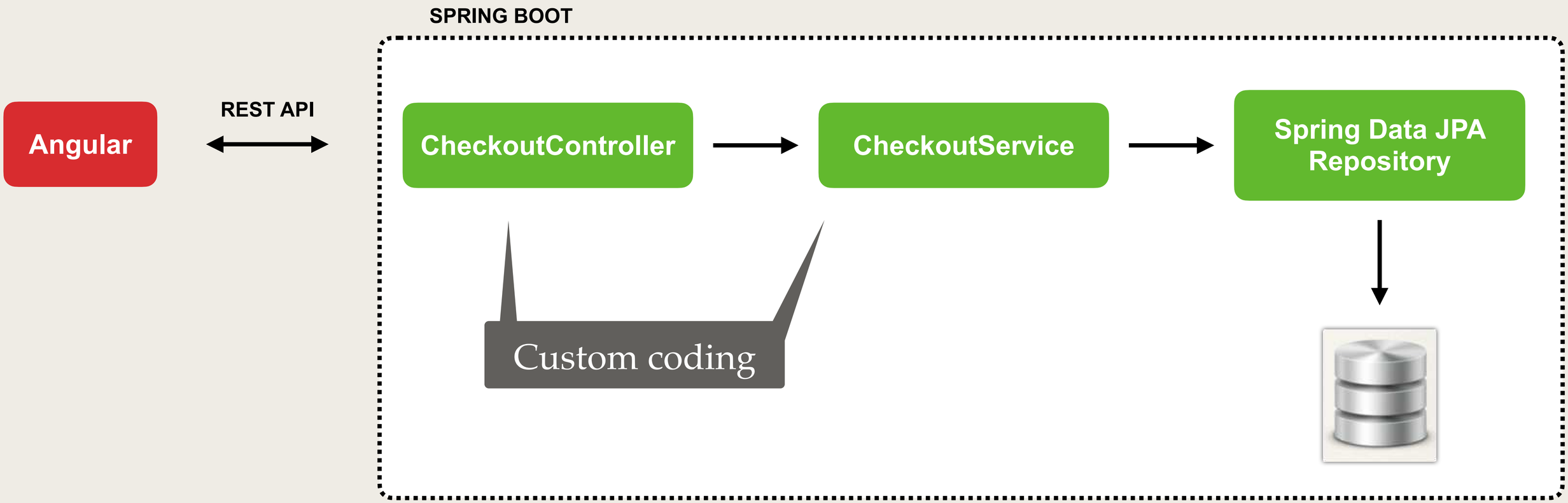
Save The Order

- We have our shopping cart full with products ... let's checkout!
- Send the order to the backend and store it in the database



© luv2code LLC

Application Architecture



© luv2code LLC

Custom Controller and Service

- For the architecture, we will create a custom controller and service
  - CheckoutController
  - CheckoutService

- You may wonder ... why not Spring Data REST???

© luv2code LLC

Why Not Spring Data REST???

- Spring Data REST is great for basic CRUD
  - We are currently using it for product catalog

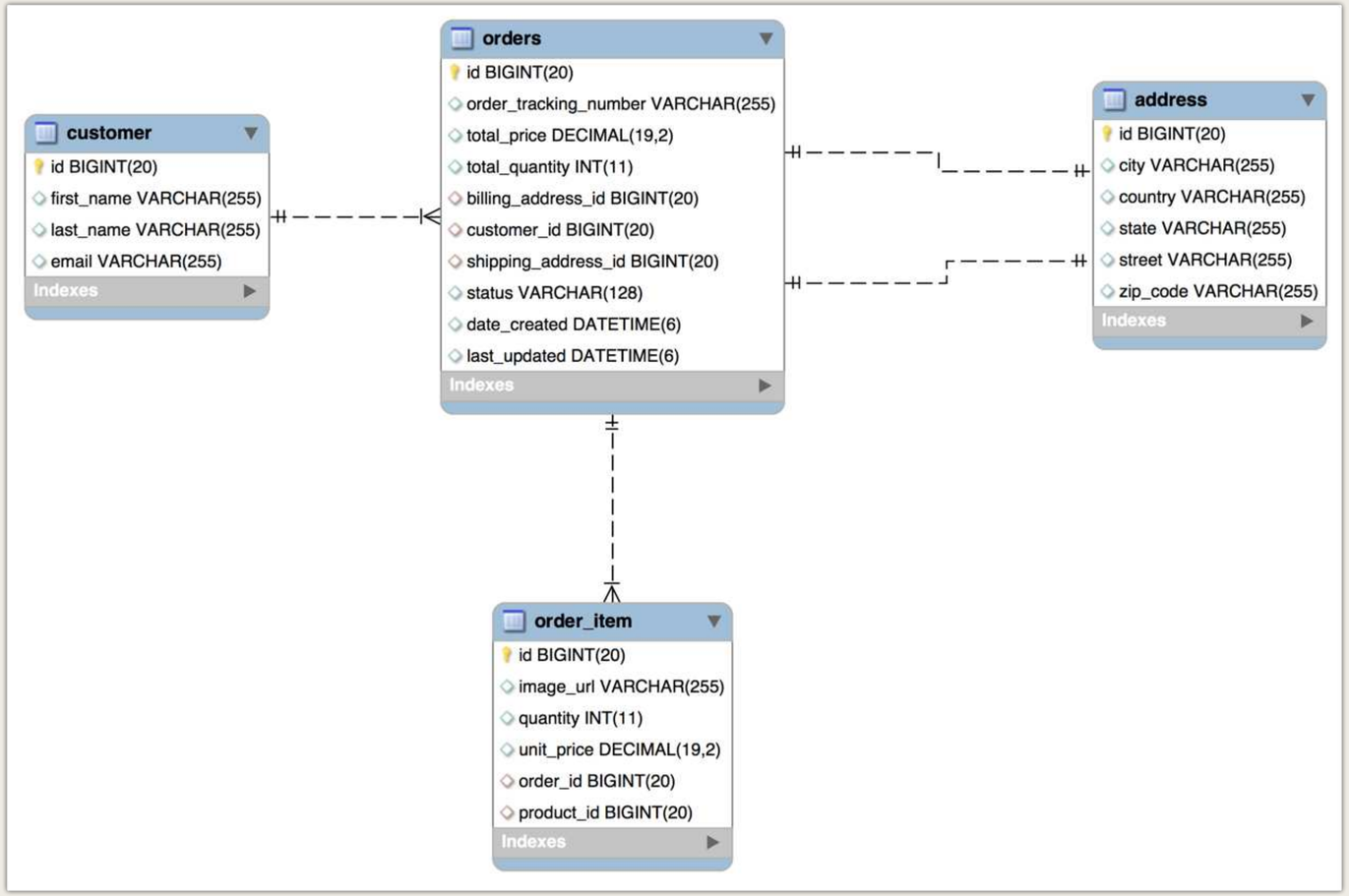
- Not the best for processing the order using custom business logic
  - Generate custom tracking number
  - Save order in database
  - *Other custom business logic ...*

Spring Data REST
is very limited in terms of customization

For custom business logic and processing,
create a custom controller and service

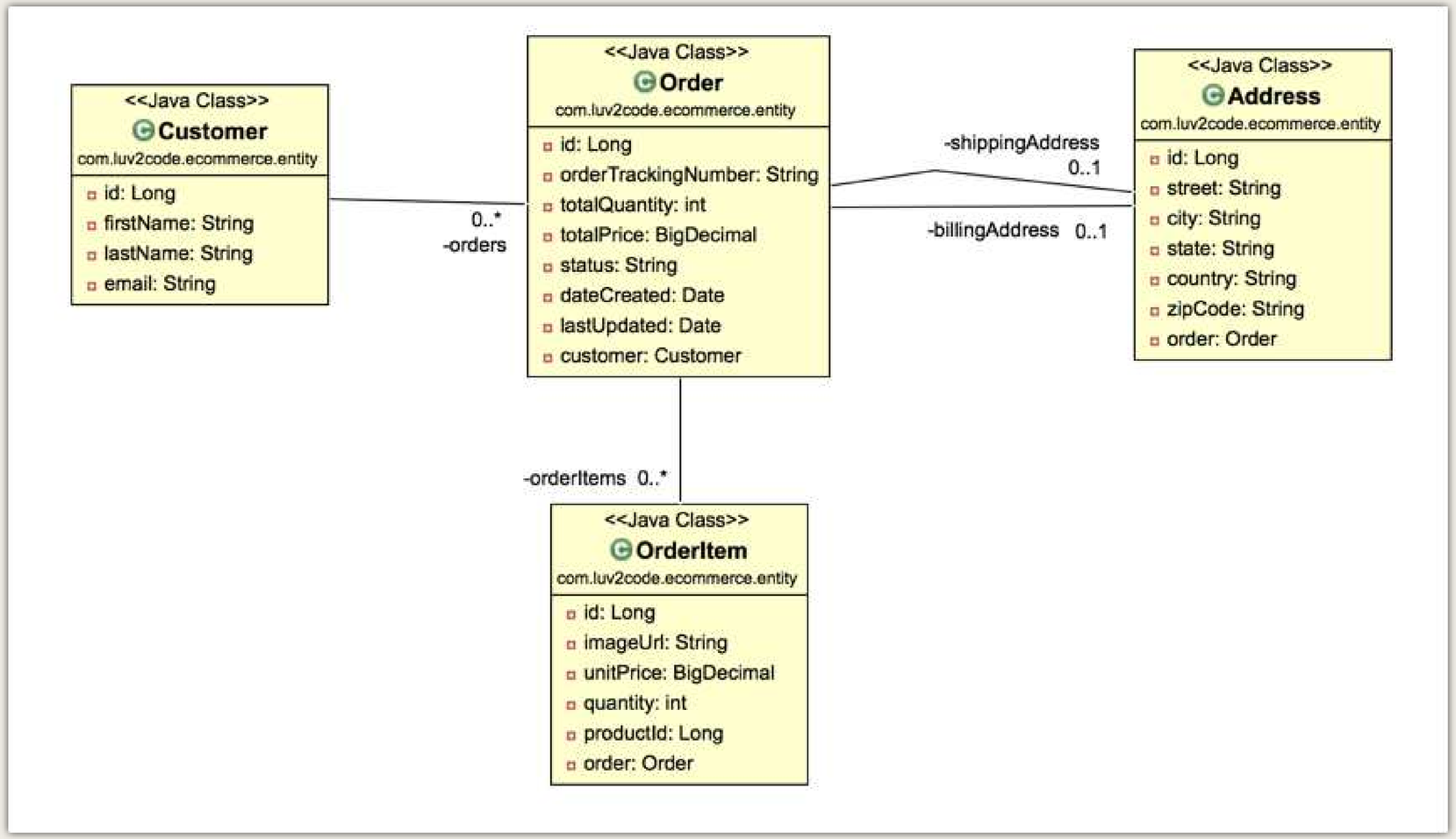
© luv2code LLC

Database Diagram



© luv2code LLC

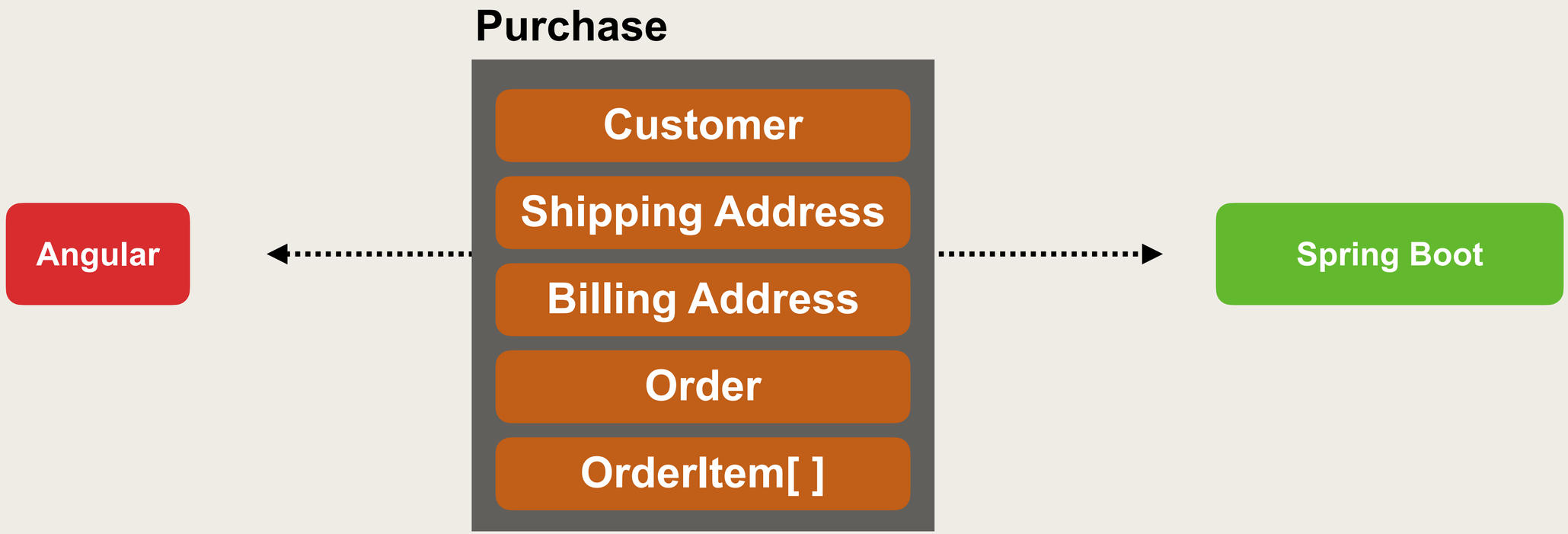
Class Diagram - Entity Classes



© luv2code LLC

Data Transfer Object

- Data transfer between Angular front-end and Spring Boot back-end

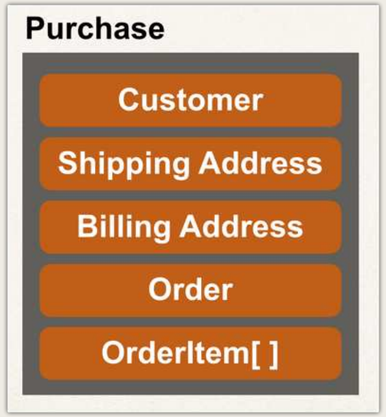


© luv2code LLC

REST API

- Support the POST method for checkout purchase
- Request body contains JSON for Purchase data transfer object

HTTP Method		Action
POST	/api/checkout/purchase	New purchase order



© luv2code LLC

Sample JSON

Purchase

- Customer
- Shipping Address
- Billing Address
- Order
- OrderItem[ ]

```
{
   "customer":{
      "firstName":"Nancy",
      "lastName":"Smith",
      "email":"nancy.smith@luv2code.com"
   },
   "shippingAddress":{
      ...
  },
   "billingAddress":{
      ...
   },
   "order":{
      "totalPrice":36.98,
      "totalQuantity":2
   },
   "orderItems":[
      {
         "imageUrl":"assets/images/products/coffeemugs/coffeemug-luv2code-1000.png",
         "quantity":1,
         "unitPrice":18.99,
         "productId":26
      },
      {
         "imageUrl":"assets/images/products/mousepads/mousepad-luv2code-1000.png",
         "quantity":1,
         "unitPrice":17.99,
         "productId":51
      }
   ]
}
```


© luv2code LLC

Development Process - Spring Boot

Step-By-Step

1. Run database script
2. Create entities
3. Create data transfer objects
4. Create repository
5. Create service
6. Create controller

Let's Write Some Code!

© luv2code LLC

UUID

- We want a tracking number that is
  - Hard to guess: random / unique

Google: wikipedia uuid

- UUID: **U**niversally **U**nique **ID**entifier
  - Standardized methods for generating unique IDs
  - Available in four versions, we will use Version 4 (random)

© luv2code LLC

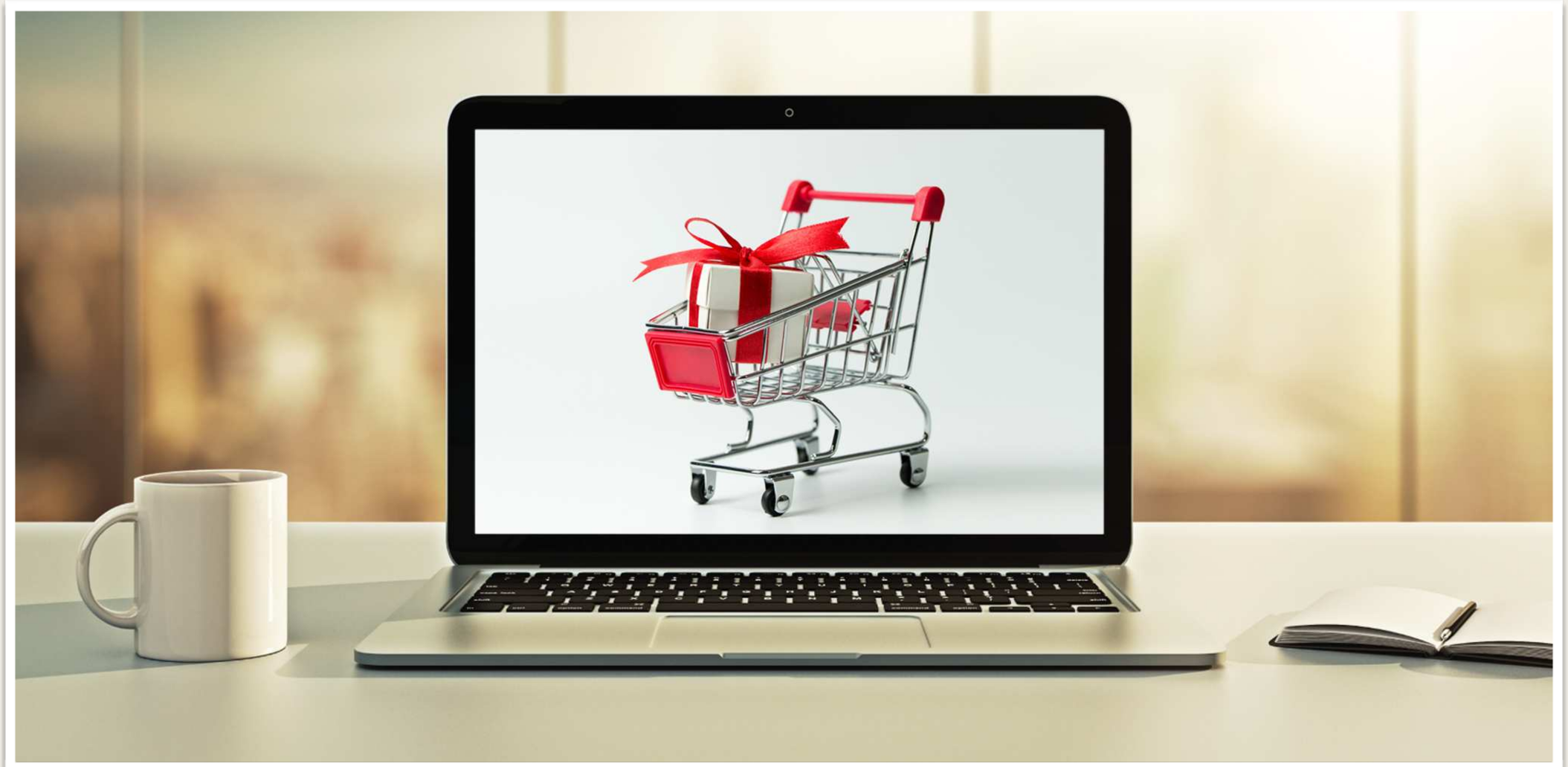
What about Uniqueness / Collisions??

Google: wikipedia uuid

- The probability of collision is VERY low ... neglible
- See *Collisions* section of Wikipedia entry
  - Probability of a duplicate in 103 trillion version-4 UUIDs is one in a billion
- However, if you need absolute 0% of collision, you could do this instead
  - generate random UUID
  - query your db and see if UUID has been used by your app
  - if so, then repeat previous two steps until no duplicates found

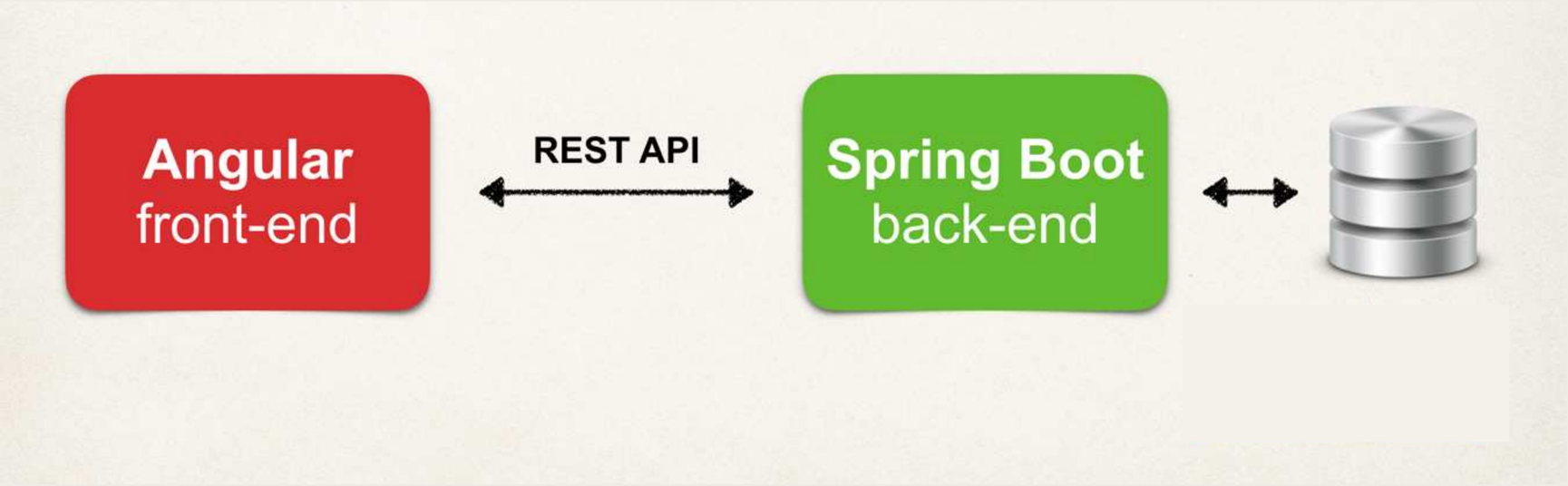
© luv2code LLC

Checkout Form
Save The Order - Front End



Save The Order

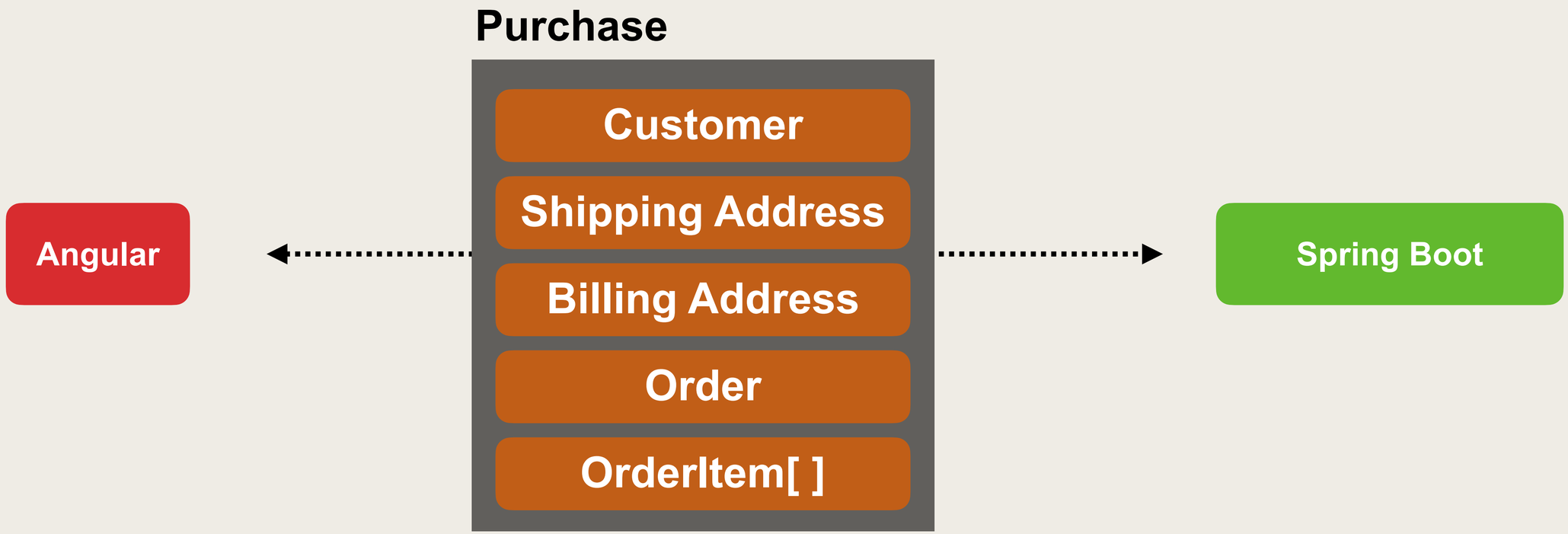
- The Spring Boot back-end code is up and running
- Let's focus on developing Angular front-end code



© luv2code LLC

Data Transfer Object

- Data transfer between Angular front-end and Spring Boot back-end



© luv2code LLC

Development Process

Step-By-Step

1. Create common classes
    1. Customer, Order, OrderItem, Address, Purchase

2. Create CheckoutService
    1. Make REST API call to Spring Boot backend

3. Update CheckoutComponent
    1. Inject CheckoutService and Router
    2. Update onSubmit() method to collect form data, call CheckoutService

Let's Write Some Code!

© luv2code LLC