

# Full Stack Project Overview



# Project

- Build Real-time e-commerce App



## Full Stack

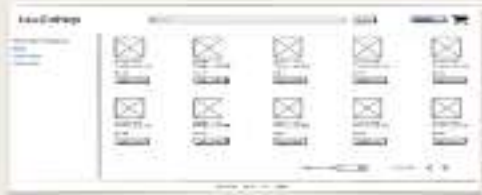


# Requirements

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



# Wireframes - Home Page



# Wireframes - Product Details



# Wireframes - Shopping Cart Details



# Wireframes - Check Out



# Release Plan

## Release 1.0

- Check out list of products

## Release 2.0

- add products to shopping cart (CHUD)
- Shopping cart check out

## Release 3.0

- User login/logout security
- Track previous orders for logged in users



# 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 video, but any Java IDE will work)
  - Maven
  - MySQL Database and MySQL Workbench

# About IntelliJ

Super Amazing TOC!!!

- 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 (if your trial version is available)

## Additional Java IDEs

- You can even 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**

# Spring Boot Back End

Leverage Spring Data REST for REST API

Minimizes the coding for Spring Boot back end



## Create Repository

Spring Data REST will scan your project for JpaRepositories

Expose REST APIs for each entity type for your JpaRepository

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

# 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 add an "s" to the end

```
public interface PersonRepository extends JpaRepository<Person, Long> {  
    ...  
}
```

person/s

# REST API

Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
POST	/products	Create a new product
GET	/products	Read a list of products
GET	/products/{id}	Read a single product
PUT	/products/{id}	Update an existing product
DELETE	/products/{id}	Delete an existing product



# Database Schema - Release 1.0



**Schema Design**  
Primary key: id, foreign key: user\_id



**Schema Design**  
Primary key: id, foreign key: product\_id

# Two Database Scripts

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

## About: 01-create-user.sql

1. Create a new MySQL user for our application:

- user id: ecommerceapp
- password: ecommerceapp

# Alum: 02-create-products.sql

1. Create new database tables: product, product\_category
2. Load tables with sample data

```
-- 02: initialize products
-- 02: tables
-- 02: product
-- 02: product_category
-- 02: data
-- 02: Product Properties
-- 02: functions
```

# Spring Boot Back End



# Development Process

Developing

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-solr  
spring-cloud-solr-jpa  
lombok
```

3. Develop the Product, Product and ProductCategory
4. Create REST APIs with Spring Data JPA Repositories and Spring Data REST

# Project Lombok

- Minimum Java project
- Lombok automatically 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>

## Project Lambek

1000

[illegible]

Webb

Leimicok  
www.leimicok.com

**Appendix 1** | **Continued**

1. **Introduction**  
 2. **Background**  
 3. **Methodology**  
 4. **Results**  
 5. **Conclusion**  
 6. **References**  
 7. **Appendix**  
 8. **Index**  
 9. **Table of Contents**  
 10. **Figure**  
 11. **Table**  
 12. **Figure**  
 13. **Table**  
 14. **Figure**  
 15. **Table**  
 16. **Figure**  
 17. **Table**  
 18. **Figure**  
 19. **Table**  
 20. **Figure**  
 21. **Table**  
 22. **Figure**  
 23. **Table**  
 24. **Figure**  
 25. **Table**  
 26. **Figure**  
 27. **Table**  
 28. **Figure**  
 29. **Table**  
 30. **Figure**  
 31. **Table**  
 32. **Figure**  
 33. **Table**  
 34. **Figure**  
 35. **Table**  
 36. **Figure**  
 37. **Table**  
 38. **Figure**  
 39. **Table**  
 40. **Figure**  
 41. **Table**  
 42. **Figure**  
 43. **Table**  
 44. **Figure**  
 45. **Table**  
 46. **Figure**  
 47. **Table**  
 48. **Figure**  
 49. **Table**  
 50. **Figure**  
 51. **Table**  
 52. **Figure**  
 53. **Table**  
 54. **Figure**  
 55. **Table**  
 56. **Figure**  
 57. **Table**  
 58. **Figure**  
 59. **Table**  
 60. **Figure**  
 61. **Table**  
 62. **Figure**  
 63. **Table**  
 64. **Figure**  
 65. **Table**  
 66. **Figure**  
 67. **Table**  
 68. **Figure**  
 69. **Table**  
 70. **Figure**  
 71. **Table**  
 72. **Figure**  
 73. **Table**  
 74. **Figure**  
 75. **Table**  
 76. **Figure**  
 77. **Table**  
 78. **Figure**  
 79. **Table**  
 80. **Figure**  
 81. **Table**  
 82. **Figure**  
 83. **Table**  
 84. **Figure**  
 85. **Table**  
 86. **Figure**  
 87. **Table**  
 88. **Figure**  
 89. **Table**  
 90. **Figure**  
 91. **Table**  
 92. **Figure**  
 93. **Table**  
 94. **Figure**  
 95. **Table**  
 96. **Figure**  
 97. **Table**  
 98. **Figure**  
 99. **Table**  
 100. **Figure**  
 101. **Table**  
 102. **Figure**  
 103. **Table**  
 104. **Figure**  
 105. **Table**  
 106. **Figure**  
 107. **Table**  
 108. **Figure**  
 109. **Table**  
 110. **Figure**  
 111. **Table**  
 112. **Figure**  
 113. **Table**  
 114. **Figure**  
 115. **Table**  
 116. **Figure**  
 117. **Table**  
 118. **Figure**  
 119. **Table**  
 120. **Figure**  
 121. **Table**  
 122. **Figure**  
 123. **Table**  
 124. **Figure**  
 125. **Table**  
 126. **Figure**  
 127. **Table**  
 128. **Figure**  
 129. **Table**  
 130. **Figure**  
 131. **Table**  
 132. **Figure**  
 133. **Table**  
 134. **Figure**  
 135. **Table**  
 136. **Figure**  
 137. **Table**  
 138. **Figure**  
 139. **Table**  
 140. **Figure**  
 141. **Table**  
 142. **Figure**  
 143. **Table**  
 144. **Figure**  
 145. **Table**  
 146. **Figure**  
 147. **Table**  
 148. **Figure**  
 149. **Table**  
 150. **Figure**  
 151. **Table**  
 152. **Figure**  
 153. **Table**  
 154. **Figure**  
 155. **Table**  
 156. **Figure**  
 157. **Table**  
 158. **Figure**  
 159. **Table**  
 160. **Figure**  
 161. **Table**  
 162. **Figure**  
 163. **Table**  
 164. **Figure**  
 165. **Table**  
 166. **Figure**  
 167. **Table**  
 168. **Figure**  
 169. **Table**  
 170. **Figure**  
 171. **Table**  
 172. **Figure**  
 173. **Table**  
 174. **Figure**  
 175. **Table**  
 176. **Figure**  
 177. **Table**  
 178. **Figure**  
 179. **Table**  
 180. **Figure**  
 181. **Table**  
 182. **Figure**  
 183. **Table**  
 184. **Figure**  
 185. **Table**  
 186. **Figure**  
 187. **Table**  
 188. **Figure**  
 189. **Table**  
 190. **Figure**  
 191. **Table**  
 192. **Figure**  
 193. **Table**  
 194. **Figure**  
 195. **Table**  
 196. **Figure**  
 197. **Table**  
 198. **Figure**  
 199. **Table**  
 200. **Figure**  
 201. **Table**  
 202. **Figure**  
 203. **Table**  
 204. **Figure**  
 205. **Table**  
 206. **Figure**  
 207. **Table**  
 208. **Figure**  
 209. **Table**  
 210. **Figure**  
 211. **Table**  
 212. **Figure**  
 213. **Table**  
 214. **Figure**  
 215. **Table**  
 216. **Figure**  
 217. **Table**  
 218. **Figure**  
 219. **Table**  
 220. **Figure**  
 221. **Table**  
 222. **Figure**  
 223. **Table**  
 224. **Figure**  
 225. **Table**  
 226. **Figure**  
 227. **Table**  
 228. **Figure**  
 229. **Table**  
 230. **Figure**  
 231. **Table**  
 232. **Figure**  
 233. **Table**  
 234. **Figure**  
 235. **Table**  
 236. **Figure**  
 237. **Table**  
 238. **Figure**  
 239. **Table**  
 240. **Figure**  
 241. **Table**  
 242. **Figure**  
 243. **Table**  
 244. **Figure**  
 245. **Table**  
 246. **Figure**  
 247. **Table**  
 248. **Figure**  
 249. **Table**  
 250. **Figure**  
 251. **Table**  
 252. **Figure**  
 253. **Table**  
 254. **Figure**  
 255. **Table**  
 256. **Figure**  
 257. **Table**  
 258. **Figure**  
 259. **Table**  
 260. **Figure**  
 261. **Table**  
 262. **Figure**  
 263

100

Absolutely no need to generate camera and pattern

**Edwards will do it on what he goes and is equally behind the scenes**

**Abstract**

1999-2000

© 2004 Blackwell Publishing Ltd *Journal of Internal Medicine* 255: 105–112







# REST API - Read Only



# REST API

I don't want REST.  
I want the REST API as FULL-ONLY

- Spring Data REST will require ALL the annotations that 

HTTP Method			CRUD Action
	POST	/products	Create a new product
	GET	/products	Get a list of products
	GET	/products/{id}	Get a single product
	PUT	/products/{id}	Update an existing product
	DELETE	/products/{id}	Delete an existing product

# REST API: PATCH

Spring Data 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

Read the detailed PATCH API docs

# HTTP PATCH Example

HTTP PATCH

JSON Patch

JSON Patch

URL:

Send

```
1 {
2   "name": "VB Fundamentals",
3   "description": "Learn Visual Basic"
4 }
```

JSON Patch

Send the following PATCH value

# Possible Solutions

## 1. Option 1: Don't use Spring Data JPA

1. Manually create and run @Entity annotation

2. Manually define methods for basic CRUD operations

3. But we lose the Spring Data JPA support for paging, sorting etc :-|

## 2. Option 2: Use Spring Data JPA

Online link

1. Configure in application HTTP methods: POST, PUT, GET etc

THE WEB CODE  
POST / PUT Security  
Index for Web resources



# Spring Data REST Configuration

- Disable HTTP methods POST, PUT, DELETE

HTTP methods to disable

```
@Configuration  
@EnableWebSecurity  
@EnableSpringDataWebSupport
```

```
@Configuration
```

```
public class SpringDataWebSecurity extends WebSecurityConfigurerAdapter {  
    // ...  
}
```

```
    @Override protected void configure(HttpSecurity http) throws Exception {  
        http.csrf().disable().authorizeRequests().antMatchers("/api/**").permitAll().and().authorizeRequests().antMatchers("/api/**").denyAll();  
    }
```

```
    // ...  
}
```

```
@EnableSpringDataWebSupport
```

```
    @Override protected void configure(HttpSecurity http) throws Exception {  
        http.csrf().disable().authorizeRequests().antMatchers("/api/**").permitAll().and().authorizeRequests().antMatchers("/api/**").denyAll();  
    }
```

```
    // ...  
}
```

# Spring Data REST Configuration

- Disable HTTP methods: POST, PUT, DELETE

```

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedMethods = {})
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/products/**").permitAll()
                .anyRequest().authenticated()
            .and()
                .httpBasic()
            .and()
                .sessionManagement()
                    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }
}

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedMethods = {})
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/products/**").permitAll()
                .anyRequest().authenticated()
            .and()
                .httpBasic()
            .and()
                .sessionManagement()
                    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }
}

```

Apply to: Hapi for: Product

Spring Data

Controller

# Spring Data REST Configuration

- Disable HTTP methods POST, PUT, DELETE

**Configuration**

**Enable the SpringDataRestResourceHandler**

**Annotation**

```
@Configuration  
public class SpringDataRestConfiguration {  
    @Bean  
    public SpringDataRestResourceHandler springDataRestResourceHandler() {  
        SpringDataRestResourceHandler handler = new SpringDataRestResourceHandler();  
        handler.setHttpMethods(HttpMethod.GET, HttpMethod.DELETE, HttpMethod.HEAD);  
        return handler;  
    }  
}
```

**Enable the SpringDataRestResourceHandler**

**Annotation**

```
@Configuration  
@EnableSpringDataWebSupport  
@SpringBootApplication  
public class Application {  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```

Spring Data

Configuration

Enable the



# Angular Front End



# Angular Front End

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



# Development Prerequisites

Prerequisites

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 Cross-browser support to Angular Node app

## Step 1: Create Angular project

- Create new project using Angular CLI

```
❯ ng new angular-exercises
```

## Step 2: Create Angular component for product-list

- 1. Create new component using Angular CLI

```
ng generate component components/product-list
```

```
ng generate component components/product-list
```

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

## Step 3: Develop TypeScript class for Product

Placed in sub-directory: common

Create new class

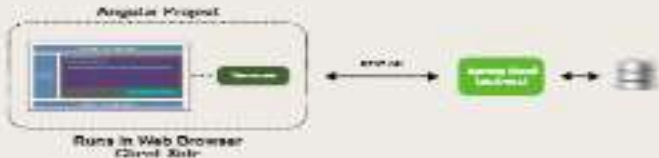
Ctrl + generate class common/product

```
import class Product {  
  id: string;  
  name: string;  
  description: string;  
  price: number;  
  category: string;  
  status: boolean;  
  createdAt: Date;  
  updatedAt: Date;  
}
```

## Step 4: Create Angular service to call REST API

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

# Application Interaction





## Step 4: Create Angular service to call REST API

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

Support for  
`HttpClientModule`

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

@NgModule({
  imports: [
    HttpClientModule
  ],
  providers: [
    // ...
  ],
})
export class AppModule {}
```

# Step 4: Create Angular service to call REST API.

Service is a class that provides data to the application.

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

```
@Injectable()  
{  
  constructor(private http: HttpClient) {}  
  
  getUsers(): Observable<User[]> {  
    return this.http.get<User[]>('http://localhost:3000/users')  
      .pipe(map(users => users.map(user => ({  
        id: user.id, name: user.name, email: user.email, password: user.password  
      }))))
```

```
}  
  
export { UsersService }  
export { User }  
export { Observable }  
export { map }
```

Import HttpClientModule in app.module.ts

Import UsersService in app.module.ts

Call getUsers() method

Assign the data to the user array in the component

## Step 5: Develop Angular to subscribe to data

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Observable } from 'rxjs';
```

```
export class DataService {
  constructor(private http: HttpClient) {}
  getPosts(): Observable<Post[]> {
    return this.http.get<Post[]>('http://localhost:3000/posts');
```

```
  }
  createPost(post: Post): Observable<Post> {
    return this.http.post<Post>('http://localhost:3000/posts', post);
```

```
  }
  updatePost(post: Post): Observable<Post> {
    return this.http.put<Post>('http://localhost:3000/posts', post);
```

```
  }
  deletePost(post: Post): Observable<Post> {
    return this.http.delete<Post>('http://localhost:3000/posts', {
      headers: new HttpHeaders({ 'Content-Type': 'application/json' })
    });
  }
}
```

Observable<Post[]> getPosts(): Observable<Post[]>

Observable<Post> createPost(post: Post): Observable<Post>

## Step 6: Display the Data in an HTML page

```
as_json = @products.to_json(
  :include => [:name],
  :only => [:name, :price, :currency])
```

### Products

- Product 1: \$100.00
- Product 2: \$200.00
- Product 3: \$300.00
- Product 4: \$400.00
- Product 5: \$500.00

## Step 7: Add CrossOrigin support to Spring Boot

WARNING: This warning is  
displayed because of a Java Security  
Update Policy

- 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, host name, port number
- Can solve this by adding "Cross-Origin Resource Sharing (CORS)" into our test side application

# Add CrossOrigin support to Spring Boot App

by @michaelroberts

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

ERROR

```
@CrossOrigin(["http://localhost:4200", "http://www.example.com"])
```

WARN: deprecated

```
@CrossOrigin
```

# Online Store Template Integration



# Release 2.0 - Plan

- 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 (AJAX)
- Shopping cart check out

Current Plan

Current look

More functionality than  
originally planned with 2.0

Feature creep  
Kinda overkill



# Current Status

- At the moment, we have a basic front end

- Proof of concept for listing the products: Angular + Spring Boot

Products			
ID	NAME	PRICE	STATUS
1	Product 1	1000	OK
2	Product 2	2000	OK
3	Product 3	3000	OK
4	Product 4	4000	OK
5	Product 5	5000	OK
6	Product 6	6000	OK

# Online Shop Template Integration

- We need a website that looks like an online shop



# Wireframes to Web Template

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

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



my2code

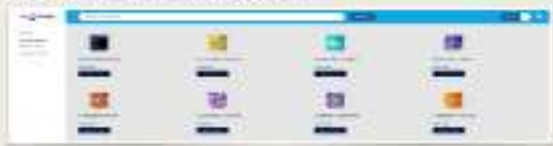


my2code.com

© my2code LLC

# Online Shop Template Integration

- Integrate the HTML template with our Angular + Environment + Bootstrap files
- We'll add the CSS styles to our project



# 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](http://www.w3schools.com/css)

[www.w3schools.com/bootstrap4](http://www.w3schools.com/bootstrap4)

# Development Prerequisites

Prerequisites

1. Download the HTML template starter solution files
2. Install Bootstrap CSS styles locally using npm
3. Add local version CSS styles for Angular core/ctrls/panels/rows files
4. Integrate template files into Angular app
5. Add support for mouse wheel legend
6. Enhance our application with product images



## Search By Category



# Release 2.0 - Plan

Completed Tasks

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 (Wishlist)

Shopping cart check out

Current task



# Search for Products by Category



# DEMO

# 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.



Order provides a toolbar off your page

Doesn't reload entire page

# Key Components

Member	Description
<code>router</code>	Used to find which <code>Route</code> is being requested. It contains a <code>RouterState</code> object that contains the current <code>Route</code> and the <code>Params</code> of the <code>Route</code> .
<code>Route</code>	Used to define a <code>Route</code> in the application. It contains a <code>path</code> property that defines the URL path for the <code>Route</code> .
<code>RouterState</code>	Used to store the current <code>Route</code> and the <code>Params</code> of the <code>Route</code> .
<code>RouterParams</code>	Used to store the <code>Params</code> of the current <code>Route</code> .
<code>RouterHistory</code>	The <code>RouterHistory</code> object that is used to store the history of the application. It contains a <code>history</code> property that is an instance of the <code>History</code> interface.

For more information, visit the [React Router](https://reactrouter.com/docs/en-6.0) documentation.

<https://reactrouter.com/docs/en-6.0>

# Development Process

Developing

1. Define routes
2. Configure Router based on our routes
3. Create the Route Child
4. Set up Router Link 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

## Step 1: Define Routes

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

When the user selects the link for the route path

Angular will create a new instance of component

```
const routes: Routes = [
```

```
  {path: 'products', component: ProductListComponent},
```

```
];
```

Path to match

Angular will interpret,  
create new instance of component

Note: The path has not been selected

## Step 1: Define Routes

- Add route to show products for a given category id

```
router.routes() routes = []  
[path: "/category/{id}", response: ProductListResponse?]  
[path: "/products", response: ProductListResponse?]  
1;
```

Category id provided

the component can now fetch data and  
show products for this category

# Step 1: Define Routes

- Add some routes for handling the below cases

```
router.routes() routes = []  
[path: "/category/:id", controller: ProductListController]  
[path: "/detail", controller: ProductDetailsController]  
[path: "/products", controller: productsController, method: "get"]  
[path: "*", redirectTo: "/products", statusCode: "not found"]  
[path: "/", redirectTo: "/products", statusCode: "not found"]  
}
```

ProductListController  
ProductDetailsController  
productsController

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



# Step 1: Define Routes

- Can add a custom PageNotFoundComponent for 404

```
routeNotFound: routes = 1  
...  
RouterModule.forRoot(routes, {  
  scrollPositionRestoration: 'preserve' // default: 'top'`  
})
```

Can create the custom 404 page.  
Routing will redirect to your 404  
page instead of  
showing a 404 error message

Custom Component that you create  
can give any routes and  
provide your own 404 error page

<https://angular.io/guide/router-faq#faq-404> is website of router

## Step 2: Configure Router based on our routes

- Configure the router in the application module:

```
@NgModule({
```

```
  imports: [RouterModule.forRoot(routes)]
```

```
})
```

```
export class AppModule {}
```

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

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { AppRoutingModule } from './app-routing.module';
```

```
import { AppComponent } from './app.component';
```

```
import { AppRoutingModule } from './app-routing.module';
```

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

```
import { BrowserModule } from '@angular/platform-browser';
```

## Step 3: Define the Router Outlet

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



Using RouterOutlet in templates to render the pages.

Example: Home, About, Contact, Login page.

## Step 3: Define the Router Outlet

- Update app component.html to use Router Outlet



```
<!-- ROUTE CONTENT -->  
<router-outlet></router-outlet>
```

Router can render multiple views  
(using components component.html)

```
<!-- ROUTE CONTENT -->  
<router-outlet></router-outlet>
```

Component will be rendered in this location for  
every pushed state -> every pushed state

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

- In our HTML page, set up links for each router

- Pass category id as a parameter



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

- In our HTML page, set up links for each product
- Pass category id as a parameter



Based on the data, we can create a ProductLinkComponent

```
1 // src/app/product-link/product-link.component.ts
2 import { Component } from '@angular/core';
3 import { Router } from '@angular/router';
4
5 @Component({
6   selector: 'app-product-link',
7   templateUrl: './product-link.component.html',
8   styleUrls: ['./product-link.component.css']
9 })
10 export class ProductLinkComponent {
11   constructor(private router: Router) {}
12
13   ngOnInit() {
14     // Get the category id from the URL
15     const url = window.location.href;
16     const categoryId = url.split('/').pop();
17
18     // Pass the category id to the router
19     this.router.navigate(['/product-link', categoryId]);
20   }
21 }
```

## Recall: Router Outlet

- When user clicks the link

**ProductListComponent** will appear in the location of **router-outlet**



This is where a number of your other components will be loaded into

## Step 3: Enhance ProductListComponent to read category id param

- Need to read the category id parameter

```
currentCategoryId = null;
```

```
this.currentCategoryId = (this.route.params.categoryId ? parseInt(categoryId, 10) :
```

Use the current id from  
currentCategoryId

Read the id parameter

Use the null checked code

Map of all the  
product categories



## Step 6: Modify Spring Boot app - KREST 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

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

- Spring Data R2DB and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions

Methods starting with: `find`, `read`, `query`, etc ...

Allowed

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
    Page<Product> findAllByCategory(String category);  
}
```

A query method

## Step 6: Modify Spring Boot app - KES1 Repository needs new method

So basically:

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
    Page<Product> findByCategory(Long categoryId, Pageable pageable);  
}
```

A query method

Method by  
category id

Query filter and pagination  
values

Behind the scenes, Spring will translate a query method like this:

```
SELECT * FROM product where category_id=1;
```

Mapped

## More on Query Methods

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

Support is available for conditionals and, or, like, not etc

For details on this topic

See the Query Method section in the Spring Data Reference Manual

[www.jav2code.com/spring-data-query-methods](http://www.jav2code.com/spring-data-query-methods)

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

- Dao and DaoImpl provides support for pagination

Public interface

```
public interface ProjectRepository extends JpaRepository<Project, Long> {  
    Page<Project> findByCategoryId(Long categoryId, Long id, Pageable pageable);  
}
```

Long id is added at the end of id page  
the pageable can be  
page number, size, sort, sort order etc.

pageable is replaced by pageable full method  
the full method can be  
page number, page size, sort order, sort etc.

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

- Spring Data JPA automatically exposes endpoints for many methods
- `findByCategory` `findByCategoryAndPrice`

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
    Page<Product> findByCategory(String categoryId) Page<Product> findByCategoryAndPrice  
>
```

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

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

My REST Repository

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
    Page<Product> findAll(Pageable pageable);  
}
```

<https://stackoverflow.com/questions/44334494/spring-data-jpa-repository-interface-method>

To pass data to REST API

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

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private url = 'http://localhost:8080/api/users';

  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.url);
  }
}
```

Steps: 1. Create a new Angular service to call the new URL on the Spring Boot app



# Development Process

Developing

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

## Search By Category



# Search for Products by Category



# Development Process

Developing Spring MVC

1. Modify Spring Boot app: `@SpringBootTest`
2. Create a class: `ProductCategory`
3. Create menu component for menu
4. Enhance menu component to read data from product service
5. Update product service to call IITD via Spring Boot app
6. In IITD-C, replace hard-coded links with menu component

## Step 1: Modify Spring Boot app - Expose entity ids

- By default, Spring Data REST shows and requires 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

# Step 1: Modify Spring Boot app - Expose entity ids

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

The screenshot shows the Spring Data REST API documentation for the `books` endpoint. The response is a JSON array of book objects. Each object contains fields like `id`, `title`, `author`, `year`, `pages`, `isbn`, and `price`. However, the `id` field in the response is not the primary key ID, but a generated ID. A red box highlights the text: "There is no entity id in the generated response". A yellow box highlights the text: "Including entity id is not possible in the response".

There is no entity id in the generated response

Including entity id is not possible in the response

That's not very helpful.  
We prefer providing IDs in the response.  
That's especially for  
complex resources used for other ...

# Step 1: Modify Spring Boot app - Expose entity ids

- This is what you need



## Step 1: Modify Spring Boot app - Expose entity ids

- Update Spring Data REST entity to expose entity ids

```
// expose entity ids
//
// - get 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
```



# Step 1: Modify Spring Boot app - Expose entity ids

- Update Spring Data REST config to expose entity ids

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.rest.config.annotation.ExposeIdsConfigurer;

@SpringBootApplication
public class SpringBootApplication implements ExposeIdsConfigurer {

    @Override
    public void configureEntityManager(EntityManager em) {
        em.setExposeIds(true);
    }
}
```

Update JPA entity manager



## Step 1: Modify Spring Boot app - Expose entity ids

- The configuration gives us the desired output



## Step 2: Create class: ProductCategory

```
it to generate class: common/ProductCategory
```

ProductCategory

```
export class ProductCategory {  
  add number:  
  categoryPhone: string;  
}
```

## Step 3: Create New component for menu



```
ng generate component components/primary-button menu
```

```
ng generate component components/primary-button menu
```



## Step 3: Update product service to call URL on Spring Boot app

URL for product categories

Call REST API

```
import java.util.List;
import org.springframework.web.client.RestTemplate;

public class ProductService {
    private RestTemplate restTemplate;

    public List<Product> getProducts() {
        // Call REST API to get product categories
        List<Product> products = restTemplate.getForObject(
            "http://localhost:8080/api/products", List.class);
        return products;
    }
}

// In Spring Boot app, call this method to get product categories
```

Product categories are returned as a list of Product objects. The Product class has attributes like name, price, and description.

Product class has attributes like name, price, and description.

## Step 6: Replace hard-coded links with menu component

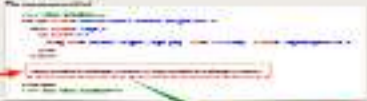
### Old Version

Before refactoring



### New Version

After refactoring



Old menu manager  
management



## Step 6: Replace hard-coded links with menu component



## Search By Keyword



# Search for Products by Keyword



# Development Process

Step 02/03

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

## 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

Application.java

MyApp

```
public interface Repository {  
    Page<Book> findAllByAuthor(String author);  
}
```

# Step 1: Modify Spring Boot app - Search Method

- Find products based on name

by @michaelson



# Step 1: Modify Spring Boot app - Search Method

@RestController

```
@RequestMapping("/api")  
public class ApiController {  
    @GetMapping("/search")  
    public List<Product> search(@RequestParam("q") String query) {  
        return productService.search(query);  
    }  
}
```

Search method is implemented in the controller

```
public List<Product> search(String query) {  
    return productService.search(query);  
}
```

# Step 1: Modify Spring Boot app - Search Method

by @mohamedreza



After modification, our application will be able to search for products by name.

To pass data to REST API



## Step 2: Create New component for Search



## Step 3: Add new Angular route for searching

app.routes.ts

```
export const routes: Routes = [
  { path: '/', component: HomeComponent },
  { path: '/search/:keyword', component: ProductListComponent },
  { path: '/product/:productId', component: ProductDetailsComponent },
  { path: '/about', component: AboutComponent },
  { path: '/contact', component: ContactComponent },
  { path: '/faq', component: FaqComponent },
  { path: '/privacy-policy', component: PrivacyPolicyComponent },
  { path: '/terms-and-conditions', component: TermsAndConditionsComponent },
  { path: '/error', component: ErrorComponent },
  { path: '**', component: ErrorComponent }
];
```

Parameters

# Quick Discussion

## Event Binding

# Quick Discussion - Event Binding



# Event Binding

- In Angular, you can listen for events with "event binding"

- In other languages / frameworks, also know

Event handler

```
<button (click) =doSomething() >Click me</button>
```

Listen for "click" event

Call a method in our Angular component code  
Can be any method name we define

# Event Binding - Example

Call a method in our AngularJS component code

`<button (click)="myFunction()">see the button</button>`

Invoke the `myFunction()` method

```
export class AppComponent {  
  myFunction() {  
    console.log('Hi! You clicked on button!');  
  }  
}
```

## Reading User Input

**Therapeutic index** =  $\frac{\text{Toxic dose}}{\text{Therapeutic dose}}$

[illegible]

[Home](#)
[About Us](#)
[Products](#)
[Services](#)
[Contact Us](#)

```

    public void search() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the name of the book to search:");
        String name = sc.nextLine();
        for (int i = 0; i < books.length; i++) {
            if (books[i].name.equals(name)) {
                System.out.println("Book found: " + books[i]);
            }
        }
    }
}

```

# Reading User Input

It's essential  
to get the value from the field  
to know what to do with it

```
var input = document.getElementById("input")  
input.addEventListener("keyup", function(event) {  
    // Do something with the value of the input  
    console.log(input.value);  
});
```

the text the user types in

Listen for "click" event

```
document.getElementById("button").addEventListener("click", function(event) {  
    // Do something when the button is clicked  
    console.log("Button clicked");  
});
```





## Step 4: Update SearchComponent to send data to search route

1. User enters search text

2. Clicks search button

3. SearchComponent has a clickHandler method

4. Passes search text

5. Route has data to the search route

6. Handled by the HomeComponent

We receive the data and show our listing products



## Step 4: Update SearchComponent to send data to search route

The diagram illustrates the data flow from the `SearchComponent` to the `SearchRoute` via a Redux state update. It consists of three main parts: a code editor for `SearchComponent`, a Redux DevTools state update log, and a code editor for `SearchRoute`.

**SearchComponent (Left):** The code shows the `handleSubmit` function. A callout box points to the `dispatch(searchAction(searchText))` line, indicating that this action triggers the state update. Below the code, a yellow box contains the text: "The state is updated and then the routing is triggered".

**Redux State Update (Middle):** A log entry shows the state being updated from `{}` to `{searchText: 'John Doe'}`. A red arrow points from the `dispatch` call in the `SearchComponent` to this log entry.

**SearchRoute (Right):** The code shows the `useEffect` hook. A callout box points to the `searchText` prop, indicating that the route receives the data from the state update. A red arrow points from the Redux state update log to this prop.

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';
import { searchAction } from './actions';

const SearchComponent = () => {
  const [searchText, setSearchText] = useState('');
  const dispatch = useDispatch();

  const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(searchAction(searchText));
  };

  return (
    <div>
      <input type="text" value={searchText} onChange={setSearchText} />
      <button onClick={handleSubmit}>Search</button>
    </div>
  );
};

export default SearchComponent;
```

```
import React from 'react';
import { useLocation } from 'react-router-dom';

const SearchRoute = () => {
  const { search } = useLocation();
  const searchText = search.substring(1);

  return (
    <div>
      <p>Search results for: {searchText}</p>
      <div>
        <div>John Doe</div>
        <div>John Doe</div>
        <div>John Doe</div>
      </div>
    </div>
  );
};

export default SearchRoute;
```

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

Enhanced

```
queryParams: {search: keyword, category: ProductServiceCategory}
```

src/products.component.ts

```
1 import { OnInit } from '@angular/core';
2 import { ActivatedRoute } from '@angular/router';
3 import { ProductService } from '../services/product.service';
4 import { Product } from '../models/product';
5
6 export class ProductsComponent implements OnInit {
7   products: Product[];
8   keyword: string;
9   category: string;
10
11   ngOnInit(): void {
12     this.route = this.router.routerState.snapshot.root.firstChild.firstChild.firstChild;
13     this.route.params.subscribe(params => {
14       this.keyword = params['keyword'];
15       this.category = params['category'];
16     });
17   }
18
19   searchProducts(): void {
20     this.products = this.productService.getProducts(this.keyword, this.category);
21   }
22 }
```

```
1 import { Component } from '@angular/core';
2 import { ActivatedRoute } from '@angular/router';
3 import { ProductService } from '../services/product.service';
4 import { Product } from '../models/product';
5
6 @Component({
7   selector: 'app-products',
8   templateUrl: './products.component.html',
9   styleUrls: ['./products.component.css']
10 })
11 export class ProductsComponent {
12   products: Product[];
13   keyword: string;
14   category: string;
15
16   ngOnInit(): void {
17     this.route = this.router.routerState.snapshot.root.firstChild.firstChild.firstChild;
18     this.route.params.subscribe(params => {
19       this.keyword = params['keyword'];
20       this.category = params['category'];
21     });
22   }
23
24   searchProducts(): void {
25     this.products = this.productService.getProducts(this.keyword, this.category);
26   }
27 }
```

Enhanced by Service  
Enhanced Component



## Product Master-Detail View



# Master-Detail View



**Master View**  
User selects item provided  
by user interface

**Detail View**  
Application shows the details  
for selected provided





# Development Process

Step 07/10

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 REST api Spring Boot app
6. Update HTML page for ProductDetailsComponent to display product details

## Step 1: Create new component for product details

```
➤ ng generate component components/ProductDetails
```

## Step 2: Add new Angular route for product details

app.routes.ts

```
export const routes: Routes = [
  { path: 'products/:id', component: ProductDetailsComponent },
```

ProductDetails

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



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

ProductListGrid.html

```
<div class="product-list">
```

```
<div class="product-item">
```

```
<a href="/product/{{product.id}}/edit">Edit</a> <a href="/product/{{product.id}}/delete">Delete</a>
```

```
<a href="/product/{{product.id}}/edit">Edit</a> <a href="/product/{{product.id}}/delete">Delete</a>
```

```
</div>
```

```
</div>
```

```
</div>
```

Added an Edit and Delete  
product button

Added an Edit and Delete  
product button



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

```
1<div class="product-list">
2  <div class="product-item">
3    <div class="product-image">
4      <img alt="Product image" />
5    </div>
6    <div class="product-info">
7      <div class="product-name">
8        <h3>Product Name</h3>
9      </div>
10     <div class="product-price">
11       <p>Product Price</p>
12     </div>
13     <div class="product-link">
14       <a href="#">Product Link</a>
15     </div>
16   </div>
17 </div>
```

Product Link

```
1<div class="product-item">
2  <div class="product-image">
3    <img alt="Product image" />
4  </div>
5  <div class="product-info">
6    <div class="product-name">
7      <h3>Product Name</h3>
8    </div>
9    <div class="product-price">
10     <p>Product Price</p>
11   </div>
12   <div class="product-link">
13     <a href="#">Product Link</a>
14   </div>
15 </div>
```





## Step 3: Update product service to call URL on Spring Boot app

```
getProductById(Integer id, Context ctx) throws ServiceException {  
    // Look up which URL needs to product id  
    String productId = String.format("%d/%d", id, productId);  
    return httpUtil.get(getProductUrl(productId));  
}
```

URL for retrieving a product

Should also implement createProduct()

Use HttpClient

Handled exceptions

return null when not found  
throw exception when error

getProductById should use **URL**  
from **external data base**  
(Using an **API** - **Database** **API**)

# Mapping JSON to Product class

Link for retrieving product list

Product class for representing product information  
attributes like name, price, category, etc.  
product id, name, price, category, etc.

main class to retrieve product list

```
import java.io.*;
import java.util.*;
import java.net.*;
import org.json.*;

public class ProductList {
    public static void main(String[] args) {
        try {
            // URL of the API
            URL url = new URL("http://localhost:8080/api/products");
            // Create a connection
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            // Set request method to GET
            conn.setRequestMethod("GET");
            // Get the response code
            int responseCode = conn.getResponseCode();
            // Check if the response is successful
            if (responseCode == 200) {
                // Get the response body
                BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                // Read the response body line by line
                String line;
                while ((line = br.readLine()) != null) {
                    // Parse the JSON response
                    JSONObject jsonObject = new JSONObject(line);
                    // Extract product details
                    String name = jsonObject.getString("name");
                    double price = jsonObject.getDouble("price");
                    String category = jsonObject.getString("category");
                    // Print the product details
                    System.out.println("Product Name: " + name + ", Price: " + price + ", Category: " + category);
                }
            } else {
                // Handle error
                System.out.println("Error: " + responseCode);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.util.*;
import org.json.*;

public class Product {
    private String name;
    private double price;
    private String category;
    private String id;
    private String description;
    private String image;
    private String status;
    private String type;
}
```

## Step 4: 1 page HTML page for Product/Item/Assignment to display product details





# Angular Language Service



# Angular / JavaScript - Lenient

Angular / JavaScript is VERY lenient

Even though we are using TypeScript ... some bugs can still slip through!







## 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

- Completion items, code completions, etc.

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

# 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



## 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



# Pagination Concepts

Page of data

Query database for  
a list of products

Only provide a "subset"  
of records at a time

for example: 100/200

Page size is 10

Only show the user a  
page of data at a time

# Pagination

We will need pagination support on the back-end: Spring Data

Also need pagination support on the front-end: Angular

# Pagination

Our Spring Boot backend uses Spring Data REST

Spring Data REST provides pagination support out of the box ... yeah!!

[www.kvzcode.com/spring-data-rest-pagination](http://www.kvzcode.com/spring-data-rest-pagination)

# Spring Data REST - Parameters

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

Parameter	Purpose
<code>page</code>	The page number to access. 0-based ... default is 0.
<code>size</code>	The size of the page to return (items per page). Defaults to 20.



# Spring Data REST - Pagination Examples

Get the first page, with page size of 10

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

Remember:  
Page size is limited.  
First page for all conditions is 1

Get the second page, with page size of 10

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

Second page for all conditions is 1

# Spring Data REST - Response Meta Data

The response meta data has valuable information

Example: `http://localhost:8080/api/products`

```
an array of products as
```

```
{  
  "page": 1,  
  "size": 10,  
  "totalElements": 100,  
  "totalPages": 20,  
  "number": 0  
}
```

size of this page

Current total of all elements in the database.  
You can use this returning all of the elements.  
And then "total" for incremental pagination only

Current page number

Total pages available

# Pagination with Angular

There are many pagination solutions available for Angular

We will make use of a popular component framework ng-bootstrap



# Components in ng-bootstrap

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



Table with 5 columns and 10 rows, likely showing a list of components or features.

Component	Version	Status	Dependencies	Notes
Alert	1.0.0	Stable	ng-bootstrap	
Carousel	1.0.0	Stable	ng-bootstrap	
Dropdown	1.0.0	Stable	ng-bootstrap	
Form	1.0.0	Stable	ng-bootstrap	
Modal	1.0.0	Stable	ng-bootstrap	
Nav	1.0.0	Stable	ng-bootstrap	
Navbar	1.0.0	Stable	ng-bootstrap	
Offcanvas	1.0.0	Stable	ng-bootstrap	
Popover	1.0.0	Stable	ng-bootstrap	
Tooltip	1.0.0	Stable	ng-bootstrap	



# Pagination component

[Pagination component](#)

Prop/Event	Purpose
<code>page</code>	The page number (1-based). Defaults to 1
<code>perPage</code>	The size of the page (items per page). Defaults to 10
<code>totalItems</code>	The total number of items
<code>onChange</code>	Event handler for page change events



# Pagination component

Library: [React pagination component](#)

## Basic example of pagination component

```
import { useState } from 'react';
import { usePagination } from 'react-pagination';
import { useTable } from 'react-table';

const Example = () => {
  const [page, setPage] = useState(1);
  const { pagination } = useTable({
    // ...
  });
  return (
    <div>
      <Table pagination={pagination} />
    </div>
  );
};
```

You will create code to combine  
this for your project.

Component will generate  
links for pagination





## Pagination - Development Process



# Pagination





# Development Process

Chapter 10

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

## Step 1: Install ng-bootstrap

- Run the following commands in your Angular project directory

```
➤ ng add @angular/localize
```

Dependency for Angular 5:

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

Install ng-bootstrap

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

## Step 1: Install ng-bootstrap

- Import the module for ng bootstrap

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

@NgModule({
  imports: [
    NgbModule
  ],
  exports: [
    NgbModule
  ],
  declarations: []
})
export class AppModule {}
```

ng-bootstrap module

## Step 2: Refactor the interface for GetResponseProducts

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

```
1 interface GetResponseProducts {  
2   products: Product[];  
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  
31 // ...  
32  
33 // ...  
34  
35 // ...  
36  
37 // ...  
38  
39 // ...  
40  
41 // ...  
42  
43 // ...  
44  
45 // ...  
46  
47 // ...  
48  
49 // ...  
50  
51 // ...  
52  
53 // ...  
54  
55 // ...  
56  
57 // ...  
58  
59 // ...  
60  
61 // ...  
62  
63 // ...  
64  
65 // ...  
66  
67 // ...  
68  
69 // ...  
70  
71 // ...  
72  
73 // ...  
74  
75 // ...  
76  
77 // ...  
78  
79 // ...  
80  
81 // ...  
82  
83 // ...  
84  
85 // ...  
86  
87 // ...  
88  
89 // ...  
90  
91 // ...  
92  
93 // ...  
94  
95 // ...  
96  
97 // ...  
98  
99 // ...  
100
```

```
1 interface GetResponseProducts {  
2   products: Product[];  
3 }
```

# Spring Data REST - Response Meta Data

The response meta data also has valuable information for pagination

Let's look at the response for a GET request

```
{  
  "array of products": [  
    ...  
  ],  
  "page": 1,  
  "size": 20,  
  "totalElements": 100,  
  "totalPages": 5,  
  "number": 0  
}
```

size of this page

Current total of all elements in the database.  
You can use this returning all of the elements.  
And then "total" for incremental pagination only

Current page number

Total pages available

## Step 2: Refactor the interface for GetResponseProducts

- Refactor the interface to request the pagination meta data



## Step 3: Add pagination support to ProductService

Pass in parameters for pagination

```
getProductsListPage(Integer pageNumber,
                    Integer pageSize,
                    Integer pageNumber) ObservableList<Product> products {
    String url = "http://localhost:8080/api/products?page=" + pageNumber + "&size=" +
        pageSize;
    return this.httpClient.get(url).flatMap(response -> {
        // ...
    });
}
```

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

### Step 4: Update ProductListComponent to handle pagination

100

**Keywords:** *transformational leadership, ethical leadership, employee voice, organizational citizenship behaviors, turnover intentions*

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

\_\_\_\_\_

100

[illegible]

100

Figure 1. The effect of the concentration of the polymer on the gelation time.

\_\_\_\_\_

© 2000 Blackwell Science Ltd *Journal of Internal Medicine* 247: 395–402

[illegible]

100

1000

1. [Download the software](#)

[illegible][illegible]

1000

1

100

100

[Contact Us](#)
[About Us](#)
[Privacy Policy](#)
[Terms of Service](#)

Reprints are available from the publisher at a special price.  
 10 copies for \$10.00; 25 copies for \$25.00; 50 copies for \$45.00; 100 copies for \$80.00.  
 All orders must be prepaid.





## Step 3: Enhance HTML template by using bootstrap pagination management



## Step 4: Enhance HTML template for using bootstrap pagination management

Parameter	Purpose
<code>page</code>	1 for page number 1st number, 2 for 2nd ... defaults to 1
<code>pageSize</code>	The size of the page (items per page). Defaults to 10
<code>totalRecords</code>	The total number of items...
<code>pageChange</code>	Event handler for page change events

`pageChange: function() {`









## Step 2: Enhance HTML template for using bootstrap pagination management

The code is as follows:

```
val class="bootstrap-pagination">  
  <div class="page-item"><span>1</span></div>  
  <div class="page-item"><span>2</span></div>  
  <div class="page-item"><span>3</span></div>  
  <div class="page-item"><span>4</span></div>  
  <div class="page-item"><span>5</span></div>  
  <div class="page-item"><span>6</span></div>  
</div>
```

Update page number to "page" using pagination link



The code is as follows:

```
<div class="page-item"><span>1</span></div>  
<div class="page-item"><span>2</span></div>  
<div class="page-item"><span>3</span></div>  
<div class="page-item"><span>4</span></div>  
<div class="page-item"><span>5</span></div>  
<div class="page-item"><span>6</span></div>
```

In the bootstrap component, the "page-item" property is required to show the page number.

Update the code to show the pagination link.

## Step 2: Enhance HTML template for using bootstrap pagination management

The code is as follows:

```
<div class="pagination">  
  <span>Page 1 of 10</span>  
  <span>1</span> <span>2</span> <span>3</span> <span>4</span> <span>5</span> <span>6</span> <span>7</span>  
</div>
```

Current page number



The code is as follows:

```
<div class="pagination">  
  <span>Page 1 of 10</span>  
  <span>1</span> <span>2</span> <span>3</span> <span>4</span> <span>5</span> <span>6</span> <span>7</span>  
</div>
```

If you specified under  
attribute "current" in  
the 10 management for specified  
and current

Current page number in the specified



### Step 3: Enhance HTML template to use ng-bootstrap pagination component

Specialty: *Angiosperms: angiosperms: Tax*  
 Order: *Malvales: Dipsacales: Malvaceae*

Flower Valley, Ontario, 1986-1987

[illegible]

► **Pharmaceutical Industry**

Figure 1

1. Identify the main idea  
2. Identify the supporting details  
3. Identify the conclusion  
4. Identify the evidence  
5. Identify the counter-evidence  
6. Identify the author's purpose  
7. Identify the author's bias  
8. Identify the author's tone  
9. Identify the author's style  
10. Identify the author's audience

[www.pearsoned.com](http://www.pearsoned.com)  
 Pearson Education, Inc.  
 501 Boylston Street, Suite 900  
 Boston, MA 02116-5046

## Step 4: Enhance HTML template for using bootstrap pagination management

The output file is demonstrated below

```

<div class="pagination">
  <div class="page-item">
    <span>1</span>
  </div>
  <div class="page-item">
    <span>2</span>
  </div>
  <div class="page-item">
    <span>3</span>
  </div>
  <div class="page-item">
    <span>4</span>
  </div>
  <div class="page-item">
    <span>5</span>
  </div>
  <div class="page-item">
    <span>6</span>
  </div>
  <div class="page-item">
    <span>7</span>
  </div>
</div>

```

Link type data binding

```

<div class="page-item">
  <span>1</span>
</div>
<div class="page-item">
  <span>2</span>
</div>
<div class="page-item">
  <span>3</span>
</div>
<div class="page-item">
  <span>4</span>
</div>
<div class="page-item">
  <span>5</span>
</div>
<div class="page-item">
  <span>6</span>
</div>
<div class="page-item">
  <span>7</span>
</div>

```

A new type of data binding is used. The HTML template is updated. Then all components are updated automatically.

## Step 2: Enhance HTML template for using bootstrap pagination management

The code is as follows:

```
html class="pagination"
<div class="page-item">
  <span>1</span>
</div>
<div class="page-item">
  <span>2</span>
</div>
<div class="page-item">
  <span>3</span>
</div>
<div class="page-item">
  <span>4</span>
</div>
<div class="page-item">
  <span>5</span>
</div>
```

When user click a "page" navigation link  
(then call the method: `getPage()`)

When user click a "page" navigation link  
(then call the method: `getPage()`)

## Pagination - Select Page Size



# Pagination



## Development Process

1. Add image slider to the page using the HTML template
2. Update ProductListComponent for setting page size



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

When user selects a page size and clicks, update page size.

Current page size is 10 items

10 items 20 items 30 items 40 items 50 items

```
<select id="page" onchange="updatePageSize()" parent.target.value="1">
  <option value="10">10 items</option>
  <option value="20">20 items</option>
  <option value="30">30 items</option>
  <option value="40">40 items</option>
  <option value="50">50 items</option>
</select>
```

The value that is selected by the user

10 20 30 40 50

10 items 20 items 30 items 40 items 50 items

## Step 2: Update ProductListComponent for setting page size

```
export class ProductListComponent implements OnInit {  
  // this constructor for pagination  
  @Input() products: Product[] = [];  
  @Output() update: EventEmitter<Product> = new EventEmitter<Product>();  
  // ...  
  
  updatePageSize(pageSize: number): void {  
    this.currentPage = 1;  
    this.pageSize = pageSize;  
    this.refreshList();  
  }  
}
```

Set page size based on  
provided value



## Pagination - Keyword Search



# Pagination



# Development Process

Development

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

# Step1: Add pagination support to ProductService

Pass in parameters for pagination

```
searchProductsPaginate(@page: number,  
                        @pageSize: number,  
                        @keyword: string): Observable<IProduct> {  
    const searchUrl = `${baseUrl}/search/${keyword}/${page}/${pageSize}`;  
    return this.httpClient.get<IProduct>(searchUrl);  
}
```

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

## Step 2: Update ProductListComponent to handle pagination

ProductListComponent

Angular 10.1.0 | TypeScript 3.9.7 | Angular CLI 10.1.0

```
import { Component, OnInit } from '@angular/core';
import { Product } from '../models/product';
import { ProductService } from '../services/product-service';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {
  products: Product[] = [];
  page: number = 1;
  pageSize: number = 10;
  totalItems: number = 0;
  totalPages: number = 0;

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

  loadProducts(): void {
    const start = (this.page - 1) * this.pageSize;
    const end = start + this.pageSize;
    this.products = this.productService.getProducts(start, end);
    this.totalItems = this.productService.getTotalItems();
    this.totalPages = Math.ceil(this.totalItems / this.pageSize);
  }

  onPageChange(event: PageEvent): void {
    this.page = event.pageIndex + 1;
    this.loadProducts();
  }
}
```

Angular 10.1.0 | TypeScript 3.9.7 | Angular CLI 10.1.0

## Add Products to Shopping Cart - Part 1



# Add Products to Shopping Cart



# Overview of Entire Shopping Cart Process



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



# Add Products to Shopping Cart

Cart Status: Empty



Add products to shopping cart

# Development Process - Part 1



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

## Step 1: Create New component for Cart Status

Cart Status Component



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

## Step 2: Add HTML template for CartStatusComponent

```
<div class="cart-status">  
  <div class="total">Total: $20.00</div>  
  <div class="shipping">Shipping: $5.00</div>  
</div>
```

Fixed Amount  
Shipping Cost Item



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

Remember the JavaScript:

```
const cart = {  
  items: [],  
  add: function (item) {  
    // Add item to the cart  
    // ...  
  },  
  remove: function (item) {  
    // Remove item from the cart  
    // ...  
  },  
  clear: function () {  
    // Clear the cart  
    // ...  
  },  
  total: function () {  
    // Calculate the total price of items in the cart  
    // ...  
  }  
};
```

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



## Step 4: Update ProductListComponent with click handler method

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

```
onAddClick(product: Product) {  
  console.log('Add to cart', product.name, product.price);  
  // TODO ... do the cart work  
}
```

## Add Products to Shopping Cart - Part 2



# Add Products to Shopping Cart

Update status





## Development Process - Part 2

Developing

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

# Application Interaction



Product

ProductService

1. getProducts()

CartService

1. subscribeForEvents()

2. update() to set total price and quantity

CartService

2. submit events to all subscribers

# Step 1: Create model class: CartItem

Requirements:

```
class CartItem:
    def __init__(self, product, quantity):
        self.product = product
        self.quantity = quantity

    def __str__(self):
        return f'CartItem({self.product}, {self.quantity})'

    def __repr__(self):
        return f'CartItem({self.product}, {self.quantity})'

    def __eq__(self, other):
        return self.product == other.product and self.quantity == other.quantity
```

CartItem is a simple class that represents a single item in the cart.

It has two attributes: product and quantity.

It also has a \_\_str\_\_ method that returns a string representation of the object.

## Step 2: Develop CartService

Project > Add New Project

Project Name: CartService

Location: C:\src\CartService

Project type: Web project, ASP.NET MVC 5.2.3, .NET Framework 4.5.2, MVC 5.2.3, Web API 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2

Project type: Web project, ASP.NET MVC 5.2.3, .NET Framework 4.5.2, MVC 5.2.3, Web API 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2

Project type: Web project, ASP.NET MVC 5.2.3, .NET Framework 4.5.2, MVC 5.2.3, Web API 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2

Project type: Web project, ASP.NET MVC 5.2.3, .NET Framework 4.5.2, MVC 5.2.3, Web API 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2, WebResourceProvider 2.2.2



## Step 2: Develop CartService

Step 2: Develop CartService

CartService Interface

```
interface CartService {  
    // Add item to cart  
    void addItem(String productId, int quantity)  
    // Remove item from cart  
    void removeItem(String productId)  
    // Get items in cart  
    List<CartItem> getItems()  
    // Get total price of items in cart  
    double getTotalPrice()  
    // Clear the cart  
    void clearCart()  
}
```

```
class CartService implements CartService {  
    // Private field to store items  
    private List<CartItem> items = new ArrayList<>();  
    // Private field to store total price  
    private double totalPrice = 0.0;  
    // Add item to cart  
    public void addItem(String productId, int quantity) {  
        // Create a new CartItem  
        CartItem item = new CartItem(productId, quantity);  
        // Add item to the list  
        items.add(item);  
        // Calculate the total price  
        calculateTotalPrice();  
    }  
    // Remove item from cart  
    public void removeItem(String productId) {  
        // Find the item  
        CartItem item = findItem(productId);  
        // Remove item from the list  
        items.remove(item);  
        // Calculate the total price  
        calculateTotalPrice();  
    }  
    // Get items in cart  
    public List<CartItem> getItems() {  
        return items;  
    }  
    // Get total price of items in cart  
    public double getTotalPrice() {  
        return totalPrice;  
    }  
    // Clear the cart  
    public void clearCart() {  
        items.clear();  
        totalPrice = 0.0;  
    }  
    // Private method to calculate total price  
    private void calculateTotalPrice() {  
        totalPrice = 0.0;  
        for (CartItem item : items) {  
            totalPrice += item.getPrice() * item.getQuantity();  
        }  
    }  
    // Private method to find item  
    private CartItem findItem(String productId) {  
        for (CartItem item : items) {  
            if (item.getProductId().equals(productId)) {  
                return item;  
            }  
        }  
        return null;  
    }  
}
```

## Step 2: Develop CartService

Project setup:

dependencies: 1

```
let dependencies() {  
  use kotlinx.coroutines.core() as  
  use kotlinx.coroutines() as coroutines
```

```
let use kotlinCoroutineContext() as kotlinCoroutineContext  
let kotlinCoroutineContext() as kotlinCoroutineContext  
let kotlinCoroutineContext() as kotlinCoroutineContext
```

```
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  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

CartService (kotlin)

1. The user wants to be notified

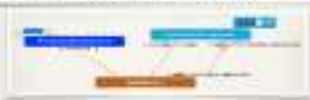
2. The user wants to be notified  
and wants to be notified

## Step 3: Modify ProductListComponent to call CartService

Step 3: Modify ProductListComponent

```
addProductToCart(productId: Product): boolean {  
  // Add logic for adding to cart  
  // ...  
  // Call CartService  
  this.cartService.addToCart(productId);  
}
```

Call Cart Service



## Step 1: Enhance CartStatusComponent to subscribe to CartService

cart-status.component.ts

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

import { CartService } from '../cart.service';

import { CartStatus } from '../models/cart-status';

import { CartItem } from '../models/cart-item';

import { CartItemComponent } from './cart-item.component';

import { CartSummaryComponent } from './cart-summary.component';

@Component({

selector: 'cart-status',

templateUrl: './cart-status.component.html',

providers: [CartService],

imports: [CartItemComponent, CartSummaryComponent],

})

export class CartStatusComponent implements OnInit {

cart: CartStatus;

constructor(private cartService: CartService) {}

ngOnInit(): void {

this.cartService.getCart().subscribe(cart => {

this.cart = cart;

});



Enhance the CartStatusComponent to subscribe to the CartService and update the cart status.



## Step 8: Update Carthage dependencies with the latest version and display the version and quantity

1. Update dependencies

2. Display version and quantity

```
carthage update --use-lockfile --no-build
carthage --verbose --use-lockfile --no-build
carthage --verbose --use-lockfile --no-build
```



# Refactor Cart Service



Currently: Basic code for finding item in cart

```

// Check if we already have the data in our cache
if (localStorage.getItem('urlList') != null) {
    // If we do, then we can just return it
    return urlList;
}

// If we don't have the data, we need to fetch it
// First, we need to get the URL from the user
let url = prompt('Enter a URL: ');

// If the user didn't enter a URL, we can't do anything
if (url == null || url == '') {
    return null;
}

// Now we can fetch the data
fetch(url)
    .then(response => {
        // If the response is not OK, we can't do anything
        if (!response.ok) {
            return null;
        }

        // If the response is OK, we can parse the JSON
        return response.json();
    })
    .then(data => {
        // If the data is null, we can't do anything
        if (data == null) {
            return null;
        }

        // If the data is not null, we can add it to our cache
        urlList.push(url);
        localStorage.setItem('urlList', JSON.stringify(urlList));

        // Now we can return the data
        return data;
    })
    .catch(error => {
        // If there is an error, we can't do anything
        return null;
    });
}

// If we don't have the data, we need to fetch it
// First, we need to get the URL from the user
let url = prompt('Enter a URL: ');

// If the user didn't enter a URL, we can't do anything
if (url == null || url == '') {
    return null;
}

// Now we can fetch the data
fetch(url)
    .then(response => {
        // If the response is not OK, we can't do anything
        if (!response.ok) {
            return null;
        }

        // If the response is OK, we can parse the JSON
        return response.json();
    })
    .then(data => {
        // If the data is null, we can't do anything
        if (data == null) {
            return null;
        }

        // If the data is not null, we can add it to our cache
        urlList.push(url);
        localStorage.setItem('urlList', JSON.stringify(urlList));

        // Now we can return the data
        return data;
    })
    .catch(error => {
        // If there is an error, we can't do anything
        return null;
    });
}

```

## Array.find(...)

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

```
arr.find(function(element, index, array) {  
  // only pass the test if the element is even  
  return element % 2 === 0;  
})
```

- 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

# Array.find(...)

Current array element

Test Conditional

```
for (let [element] of this.array()) {  
  // element is a double element array  
  if (element[0] == this.element[0]) {  
    existingPosition = thisPosition;  
    break;  
  }  
}
```

Remember, find element that passes  
what value, index found

Remember, find the index element  
in the array with test position

```
existingPosition = this.getPosition() if (thisPosition == thisPosition) {  
  // element is a double element array  
}
```

Current array element

Test Conditional

# Refactored: Before and After

## BEFORE

```
1 // create an array, increment each element  
2 let arr = [1,2,3,4,5,6,7,8,9,10];  
3 let arrLength = arr.length;  
4  
5 for (let i = 0; i < arrLength; i++) {  
6   // increment each element  
7   arr[i] = arr[i] + 1;  
8 }  
9  
10 // check if we have an  
11 arrLength = arr.length;
```

## AFTER

```
1 // create an array, increment each element  
2 let arr = [1,2,3,4,5,6,7,8,9,10];  
3 let arrLength = arr.length;  
4  
5 for (let i = 0; i < arrLength; i++) {  
6   // increment each element  
7   arr[i] = arr[i] + 1;  
8 }  
9  
10 // check if we have an  
11 arrLength = arr.length;
```

## Add Products to Cart - Details View



# Add Products to Cart - Details View





# Development Process



1. Add click handler for "Add to cart" button on `product-detail.component.html`
2. Update `ProductDetailComponent` with click handler method

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

Required information:

Product objects (products) of type `Product` are provided to the browser

On click event,  
call the method `addToCart()`



## Step 2: Update ProductDetailsComponent with click handler method

ProductDetailsComponent

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

import { ProductService } from '../services/product.service';

@Component({
  selector: 'app-product-details',
  templateUrl: './product-details.component.html',
  styleUrls: ['./product-details.component.css']
})
export class ProductDetailsComponent implements OnInit {
  productId: string;
  product: any;

  constructor(private router: Router, private productService: ProductService) {}

  ngOnInit(): void {
    this.productId = this.router.snapshot.paramMap.get('id');
    this.getProductDetails();
  }

  getProductDetails(): void {
    this.productService.getProductById(this.productId).subscribe(
      (product) => {
        this.product = product;
      },
      (error) => {
        console.error('Error fetching product details:', error);
      }
    );
  }
}
```

# Application Interaction



## Shopping Cart - List Items



# List Items in the Shopping Cart



# New Component: CartDetailsComponent



# Development Process

Step 02/03

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



## Step 1: Create New component for Cart Details

```
> ng generate component component1 --name=cart1-details
```

## Step 2: Add new route for CartDetailsComponent

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



## Step 4: Modify CartDetailsComponent to retrieve cart items

src/app/cart-details

```
import { Observable } from 'rxjs';
import { CartItem } from '../models';
import { CartService } from '../services';

@Component({
  selector: 'app-cart-details',
  templateUrl: './cart-details.component.html',
  styleUrls: ['./cart-details.component.css']
})
export class CartDetailsComponent {
  items: Observable<CartItem> = this.cartService.getItems();
}
```

```
import { Component } from '@angular/core';
import { CartService } from '../services';
import { CartItem } from '../models';

@Component({
  selector: 'app-cart-details',
  templateUrl: './cart-details.component.html',
  styleUrls: ['./cart-details.component.css']
})
export class CartDetailsComponent {
  items: Observable<CartItem> = this.cartService.getItems();
}
```

## Step 5: Add HTML template for CartDetailsComponent

```
<div>  
  <h3>Cart Details</h3>  
  <table>  
    <tr>  
      <th>Item</th>  
      <th>Quantity</th>  
      <th>Price</th>  
    </tr>  
    <tr>  
      <td>{{ item.name }}</td>  
      <td>{{ item.quantity }}</td>  
      <td>{{ item.price }}</td>  
    </tr>  
  </table>  
  <p>Total Price: {{ totalPrice }}</p>  
</div>
```

The screenshot shows a code editor with the HTML template for the CartDetailsComponent. The code is as follows:

```
<div>  
  <h3>Cart Details</h3>  
  <table>  
    <tr>  
      <th>Item</th>  
      <th>Quantity</th>  
      <th>Price</th>  
    </tr>  
    <tr>  
      <td>{{ item.name }}</td>  
      <td>{{ item.quantity }}</td>  
      <td>{{ item.price }}</td>  
    </tr>  
  </table>  
  <p>Total Price: {{ totalPrice }}</p>  
</div>
```

Red arrows indicate the mapping from the code to the UI elements in the top right corner:

- The `<h3>Cart Details</h3>` tag maps to the title of the component.
- The `<table>` tag maps to the table structure.
- The `<tr>` tag maps to the table header.
- The `<td>{{ item.name }}</td>` tag maps to the item name column.
- The `<td>{{ item.quantity }}</td>` tag maps to the quantity column.
- The `<td>{{ item.price }}</td>` tag maps to the price column.
- The `<p>Total Price: {{ totalPrice }}</p>` tag maps to the total price display.



# Shopping Cart - Increment Item Quantity



# Increment Item Quantity in the Shopping Cart



The screenshot shows a web browser displaying an e-commerce site. The top navigation bar is blue with a search bar, a login button, and a shopping cart icon. The main content area shows a product listing for a blue t-shirt. A green callout box with a pointer to the shopping cart icon contains the text: "For now ... add code for 'Increment' We will cover 'Decrement' in upcoming videos". The shopping cart icon in the top right corner shows a blue t-shirt and the text "Add to cart: \$1.00". Below the cart icon, there is a summary section with the text: "Subtotal: \$1.00", "Shipping: \$0.00", and "Total: \$1.00".

For now ... add code for "Increment"  
We will cover "Decrement"  
in upcoming videos



# Development Process



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



## Step 2: Update CartDetailsComponent with click handler method

Here is the code:

```
(cartItem.quantity) ? this.addToCart(cartItem) :  
this.removeFromCart(cartItem);
```

## Shopping Cart - Decrement / Remove Item



# Decrement / Remove Item from the Shopping Cart



# Development Process



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

## Step 1: Modify CartDetailsComponent HTML template



File: src/app/cart-details.component.html

```
<div class="modal" *ngIf="showModal" >
  <div class="modal-content" >
    <div class="modal-header" >
      <h3>Details</h3>
    </div>
    <div class="modal-body" >
      <div class="text-center" >
        <img alt="Product image" data-bbox="100 450 150 550" />
        <p>Product Name</p>
        <p>Price: $100.00</p>
        <p>Quantity: 1</p>
        <p>Total: $100.00</p>
      </div>
    </div>
    <div class="modal-footer" >
      <button class="btn btn-danger" (click)="closeModal()">Close</button>
    </div>
  </div>
</div>
```

The click event  
calls the method closeModal() by (----)

Product Name: name  
"Name" is on

## Step 2: Update CartDetailsComponent with click handler method

Here is the code:

```
onItemClick={() => { this.updateCartDetails(  
  this.cartDetails.id, this.selectedItem.id, this.selectedItem.qty  
)}  
}
```







## Step 4: "Remove" button

Remove button:

```
removeTheCartItems: CartItem <  
  this.cartService.removeTheCartItem()  
>
```

## Shopping Cart - List Items



# List Items in the Shopping Cart



# New Component: CartDetailsComponent



# Development Process

Step 02/03

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

## Step 1: Create New component for Cart Details

```
> ng generate component component1 --name=cart1-details
```



## Step 2: Add new route for CartDetailsComponent

```
// src/app/app.routes.ts
import { Routes } from '@angular/router';

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



## Step 4: Modify CartDetailsComponent to retrieve cart items

src/app/cart/cart.component.ts

```
import { Injectable } from '@angular/core';
import { Cart } from '../models/cart';
import { CartService } from '../services/cart.service';

@Injectable({
  providedIn: 'root'
})
export class CartComponent {
  cart: Cart;

  constructor(private cartService: CartService) {}

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

  loadCartItems(): void {
    this.cartService.getCartItems().subscribe(
      (items) => {
        this.cart = new Cart(items);
      }
    );
  }
}
```

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

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

  constructor(private cartService: CartService) {}

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

  loadCartItems(): void {
    this.cartService.getCartItems().subscribe(
      (items) => {
        this.cart = new Cart(items);
      }
    );
  }
}
```

## Step 5: Add HTML template for CartDetailsComponent

```
<div>  
  <h3>Cart Details</h3>  
  <table>  
    <tr>  
      <th>Item</th>  
      <th>Quantity</th>  
      <th>Price</th>  
    </tr>  
    <tr>  
      <td>{{ item.name }}</td>  
      <td>{{ item.quantity }}</td>  
      <td>{{ item.price }}</td>  
    </tr>  
  </table>  
  <p>Total: {{ totalPrice }}</p>  
</div>
```

The screenshot shows a code editor with the HTML template for the CartDetailsComponent. The code defines a table with columns for Item, Quantity, and Price, and a total price calculation. Red arrows point from the code to a visual representation of the component's UI in the top right corner, which shows a table with the same structure and a total price calculation.



# Shopping Cart - Increment Item Quantity



# Increment Item Quantity in the Shopping Cart



The screenshot shows a Magento 2 shopping cart interface. On the left is a sidebar with navigation links: Home, Catalog, Account, and Cart. The main content area displays a shopping cart with one item: a blue book titled "The Hobbit" by J.R.R. Tolkien, priced at \$19.99. A green callout box with a pointer to the "Qty" column contains the text: "For now ... add code for 'Increment' We will cover 'Decrement' in upcoming videos." The right side of the cart shows a summary: "Subtotal: \$19.99", "Grand Total: \$19.99", and "Total Due: \$19.99".

For now ... add code for "Increment"  
We will cover "Decrement"  
in upcoming videos

# Development Process



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





## Step 2: Update CartDetailsComponent with click handler method

Here is the code:

```
(cartItem.quantity) ? this.addToCart(cartItem) :  
this.removeFromCart(cartItem);
```

## Shopping Cart - Decrement / Remove Item



# Decrement / Remove Item from the Shopping Cart



# Development Process



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

### Step 1: Modify CartDetailsComponent HTML template



1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 26

**QUESTION**

Can the method be demonstrated by

Received 17 June 2007; accepted 10 October 2007  
© 2008 The Authors  
Journal compilation © 2008 Blackwell Publishing Ltd

## Step 2: Update CartDetailsComponent with click handler method

Here is the code:

```
onItemClick={() => { this.updateCartDetails(  
  this.cartDetails.id, this.selectedItem.id, this.selectedItem.qty  
)}  
}
```





### Step 4: "Remove" button



## Step 4: "Remove" button

Remove button:

```
removeTheCartItems: CartItem <  
  this.cartService.removeTheCartItem()  
>
```

## 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**

# Forms

We will use  
Flask-WTF forms.

## Relative forms

- ↳ Leverages programmable API for form building
- ↳ Scalable 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 suitable solution for large, complex forms

## Comparison

- For comparison of Deviator Sums and Regular Interval Sums

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

# Key Components

Class	Description
FormControl	Individual control that tracks the value and validation status
FormGroup	A collection of controls. Can contain nested groups
Validators	

# Our Checkout Form

Front-end  
form validation  
and layout

The image displays three overlapping browser window mockups, each showing a checkout form. The first window (left) shows the initial form with empty fields for name, email, password, and address. The second window (middle) shows the form with some fields filled out, including a name, email, and address. The third window (right) shows the form with more fields filled out, including a shipping address and a 'Place Order' button. The form is titled 'Checkout' and includes a sidebar with navigation links.



# Development Process



1. Generate new structural component
2. Add a new route for structural component
3. Create a new checkout button and link to checkout component
4. Add appropriate API endpoint routes
5. Define frontend component to file
6. Tagout form controls in HTML template
7. Add event listener for form validation

## Step 1: Generate our checkout component

```
> nx generate component --component=src/checkout
```

## Step 2: Add a new route for checkout component

```
router.routes = routes & 1  
  {path: "/checkout", component: CheckoutComponent},  
];
```

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



## Step 4: Add support for negative forms

Formatters:

```
requires {  
  RunLiveFormatters() {  
  }  
}
```

## Step 5: Define form in component .ts file



## Step 5: Define form in component .ts file

```
import { FormGroupDirective, NgFormError, NgMax } from '@angular/forms';
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm: FormGroup;
  submitted = false;
  returnUrl: string;

  constructor(
    private formBuilder: FormBuilder,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.loginForm = this.formBuilder.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  onSubmit(): void {
    this.submitted = true;
    if (this.loginForm.invalid) {
      return;
    }
    this.router.navigate(['/home']);
  }
}
```

Import the form module

Initialize the form group

Define the form

Method for user login  
redirects user to /home

## Step 6: Layout form controls in HTML template

Insert all property form components into

Insert class="form-control" (or class="form-control")

class="form-control" type="text"

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
<div class="form-control" style="width: 100%;">
```

```
</div>
```

```
</div>
```

```
</div>
```

First Name	<input type="text"/>
Last Name	<input type="text"/>
Email	<input type="text"/>
Phone	<input type="text"/>
Address	<input type="text"/>
City	<input type="text"/>
State	<input type="text"/>
Zip	<input type="text"/>

Property form controls are inserted into the form



## Step 7: Add event handler for form submission

FormSubmissionHandler

```
class FormSubmissionHandler implements ActionListener {
```

```
    public void actionPerformed(ActionEvent e) {  
        // Handle form submission logic  
    }  
}
```

FormSubmissionHandler

FormSubmissionHandler

FormSubmissionHandler

```
    // Handle form submission logic  
    // ...  
    // ...  
    // ...  
}
```

FormSubmissionHandler

FormSubmissionHandler

Method for handling  
form submission logic

Method for handling  
form submission logic

Method for handling  
form submission logic

## Step 7: Add event handler for form submission

Prevent Default Form Behavior

```
submitHandler() {  
  //validate form (functionally this is same as validate)  
  if (this.validate() === false) {  
    return; //stop form submission  
  }  
  //process and display data  
  this.processFormAndDisplayOutput();  
}
```

Output

```
//process the submitted values  
this.setState({  
  firstName: 'John',  
  lastName: 'Doe',  
  email: 'john.doe@example.com',  
  password: 'password',  
  confirmPassword: 'password',  
  isValid: true  
});  
this.setState({  
  isValid: false  
});
```

# Our Checkout Form

Repeat the process for other  
conditions of the form.



A screenshot of a web application showing a checkout form. On the left is a sidebar menu with a 'Home' link and a 'Checkout' link. The main content area contains a form with several input fields and a 'Checkout' button. The form is titled 'Checkout' and has a 'Back' button at the top left.



A screenshot of a web application showing a checkout form. On the left is a sidebar menu with a 'Home' link and a 'Checkout' link. The main content area contains a form with several input fields and a 'Checkout' button. The form is titled 'Checkout' and has a 'Back' button at the top left.



A screenshot of a web application showing a checkout form. On the left is a sidebar menu with a 'Home' link and a 'Checkout' link. The main content area contains a form with several input fields and a 'Checkout' button. The form is titled 'Checkout' and has a 'Back' button at the top left.

# Checkout Form Populate Credit Card Expiration Dates



# Credit Card Expiration Dates

Credit Card

Cardholder Name

Card Number

Expiration Date

CVV

Cardholder Email

Cardholder Phone

Regular Monthly

Regular Yearly

# Populating Drop-Down Lists

We could just hard code the values ... but not ideal :-)

Years

Years will become outdated ... if we are in year 2021, why show 2020???

Dynamically populate current year ... up to next 50 years.

# Development Process

Step 02/03

1. Generate new Java method `getSleepDataByYears`
2. Add methods to the `TermService` for months and years.
3. Update checkbox component to retrieve the months and years from service
4. Update HTML template to populate sleep data for months and years

## Step 1: Generate our form service

```
➤ ng generate service services/Lux2ShopForm
```



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

```
import { Observable } from 'rxjs';  
import { User } from './types';  
  
class AuthService {  
  constructor() {}  
  
  getProfile(userId: string): Observable<User> {  
    let user: User[] = [];  
  
    // TODO: add logic for "user" database that  
    // will return array of users and user with id  
    // that is passed as a parameter, method is (id: string) {}  
    return Observable.of({});  
  }  
}
```

Return an **Observable array**

Use **Observable** from **rxjs** library  
`Observable.of(arrayOfItems)`

The `of` operation from `rxjs` will wrap an object as an **Observable**



## Step 3: Update checkout component to retrieve months and years

Prerequisites:

```
import java.time.temporal.ChronoUnit.MONTHS
```

```
monthOffset = monthOffset + 1;  
yearOffset = yearOffset + 1;
```

```
month = LocalDate.of(year, monthOffset, dayOfMonth);  
if (month.isBefore(LocalDate.now())) month = month.plusMonths(12);
```

```
return month;
```

```
// adding years and months
```

```
month = month.plus(monthOffset, ChronoUnit.MONTHS);  
month = month.plus(yearOffset, ChronoUnit.YEARS);
```

```
return month.plus(monthOffset, ChronoUnit.MONTHS);
```

[www.kryztofc.com/java-script/jsp-ds-1c](http://www.kryztofc.com/java-script/jsp-ds-1c)

Inject our form service

Get the current month

Japan Festival never changed  
the month is Jan-2020  
(0 - 11)

So we add 1 to get  
(1 - 12)

## Step 3: Update checkout component to retrieve months and years

```
export class CheckoutFormComponent implements OnInit {  
  
  creditCardForm = FormGroup({  
    cardNumber: FormControl(''),  
    cardholderName: FormControl(''),  
    expiryMonth: FormControl(''),  
    expiryYear: FormControl(''),  
    cvv: FormControl('')  
  });  
  
  ngOnInit(): void {  
    // Retrieve credit card details from the database  
    this.http.get('http://localhost:3000/api/credit-cards').subscribe((res: any) => {  
      data = res;  
      console.log('Credit Card Details:', data);  
      this.creditCardForm.setValue({  
        cardNumber: data.cardNumber,  
        cardholderName: data.cardholderName,  
        expiryMonth: data.expiryMonth,  
        expiryYear: data.expiryYear,  
        cvv: data.cvv  
      });  
    });  
  }  
}
```

Set the  
credit card years

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

ProductTemplate

```
div class="row">

### Product



Product Name



Product Description



Product Price



Product Category



### Product



Product Name



Product Description



Product Price



Product Category


```

Loop over array

```
for (var i = 0; i < products.length; i++) {  
    <div class="card">  
        <div class="card-body">  
            <div class="text-center">  
                <h3>Product</h3>  
                <div>  
                    <div>  
                        <div>Product Name</div>  
                        <div>Product Description</div>  
                        <div>Product Price</div>  
                        <div>Product Category</div>  
                    </div>  
                </div>  
            </div>  
        </div>  
    </div>  
}
```

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

ProductTemplate

```
active: true, name: 'My Product', state: 'CA', price: 100, year: 2015, status: 'Active'
```

```
name: 'class', value: 'Product'
```

```
children: [{ type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

```
  { type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

```
  { type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

```
  { type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

```
  { type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

```
  { type: 'Product', value: 'Product', label: 'Product', year: 2015, status: 'Active' }
```

Product name: five years

```
name: 'class', value: 'Product', label: 'Product', year: 2015, status: 'Active'
```

```
name: 'class', value: 'Product', label: 'Product', year: 2015, status: 'Active'
```

```
name: 'class', value: 'Product', label: 'Product', year: 2015, status: 'Active'
```

# Checkout Form Dependent Fields



# Dependent Fields

Order Code

Product

Category

Product Name

Model Name

Manufacturer

Price

Populate Month

Populate Year

Period: Demand for department

The values of month selected department in the year



# 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

# Development Process



## 1. Update HTML template

1. For the Expiration Years drop-down list
2. Add event handling for change event

## 2. Update JavaScript program, add event listener

1. Read the selected year
2. Update the list of movies based on selected year





# Checkout Form

## Populate Country-State Drop-Down Lists



# Address - Country and State

Shipping Address

Country	<input type="text"/>
Street	<input type="text"/>
City	<input type="text"/>
State	<input type="text"/>
Zip	<input type="text"/>

Populate Country

Populate State

# 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

Since "state" is different in each country

Similar for "Province", "District", "Region", "Municipality" etc

# Development Process - Backend



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



# Step 1: Create Database Tables

Create inventory item table

Create product category table

inventory
id SERIAL(1)
name VARCHAR(255)
description VARCHAR(255)
price DECIMAL(10,2)

category
id SERIAL(1)
name VARCHAR(255)
description VARCHAR(255)
price DECIMAL(10,2)

1:1 relationship

# Step 1: Create Database Tables

```
CREATE TABLE users (
  id SERIAL UNIQUE NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  name VARCHAR(255) NOT NULL,
  created_at TIMESTAMPTZ NOT NULL,
  PRIMARY KEY (id)
);
```



```
CREATE TABLE posts (
  id SERIAL UNIQUE NOT NULL,
  user_id INT REFERENCES users(id),
  text VARCHAR(255) NOT NULL,
  created_at TIMESTAMPTZ NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE comments (
  id SERIAL UNIQUE NOT NULL,
  post_id INT REFERENCES posts(id),
  user_id INT REFERENCES users(id),
  text VARCHAR(255) NOT NULL,
  created_at TIMESTAMPTZ NOT NULL,
  PRIMARY KEY (id)
);
```

# Step 1: Create Database Tables



```
USE Adventureworks
GO
CREATE TABLE dbo.Employee (
    EmployeeID INT NOT NULL,
    LastName NVARCHAR(100) NOT NULL,
    FirstName NVARCHAR(100) NOT NULL,
    MiddleInitial NVARCHAR(10) NULL,
    Suffix NVARCHAR(10) NULL,
    Title NVARCHAR(100) NOT NULL,
    Department NVARCHAR(50) NOT NULL,
    BusinessEntityID INT NOT NULL
)
```

sql server, sql server

```
USE Adventureworks
GO
CREATE TABLE dbo.Employee (
    EmployeeID INT NOT NULL,
    LastName NVARCHAR(100) NOT NULL,
    FirstName NVARCHAR(100) NOT NULL,
    MiddleInitial NVARCHAR(10) NULL,
    Suffix NVARCHAR(10) NULL,
    Title NVARCHAR(100) NOT NULL,
    Department NVARCHAR(50) NOT NULL,
    BusinessEntityID INT NOT NULL
)
```

Thread 1

Thread 2

sql server, sql server

# Step 1: Create Database Tables



TABLE: COUNTRIES

ID	code	name
1	US	United States
2	CA	Canada
3	UK	United Kingdom
4	FR	France
5	DE	Germany
6	IT	Italy
7	JP	Japan
8	IN	India
9	BR	Brazil
10	RU	Russia

TABLE: COUNTRIES

ID	code	name
11	US	United States
12	CA	Canada
13	UK	United Kingdom
14	FR	France
15	DE	Germany
16	IT	Italy
17	JP	Japan
18	IN	India
19	BR	Brazil
20	RU	Russia



## Step 3: Create Spring Data REST Repositories

```
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface UserRepository extends CrudRepository<User, Long> {}
```

3

Repository `findAll()` is implemented

To retrieve all repositories

```
curl -X GET http://localhost:8080/api/users
```

To retrieve repository `id=1`

```
curl -X GET http://localhost:8080/api/users/1
```

## Step 3: Create Spring Data REST Repositories

To retrieve all associations

ex.g. `POST /associations` `POST /api/associations`



To retrieve association id 4

ex.g. `POST /associations/4` `POST /api/associations/4`



## Step 3: Create Spring Data REST Repositories

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface UserRepository extends JpaRepository<User, String> {}
```

Exposes JPA-like endpoint

To retrieve all users:

`http://localhost:8080/api/users`





# Step 3: Create Spring Data REST Repositories

The following defines the REST API for creating a new book:

```
POST /books HTTP/1.1 Host: localhost:8080 Content-Type: application/json
```

```
{ "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "year": 1925 }
```

200 OK

201 Created

## Step 4: Update Spring Data REST Configs

- Update our config for `MyDataRestConfig.java`
- Make the APIs for `/country` and `/state` read-only
  - This is reference data, not meant for changing it via REST API
  - Disable HTTP: PUT, POST, etc ...
- Config details covered in next slides

# 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



1. User selects a country

2. Then populate states  
for selected country

# Development Process - Frontend



1. Create TypeScript interfaces, classes and enums
2. Add methods to the http service for countries and states
3. Update checkbox component to retrieve the countries from service
4. Update HTML template to populate drop-down lists for countries
5. Add error handler for checkbox component
6. Load the selected country entries that are states based on selected country
7. Update HTML template to populate day-dropdown lists for states

# Backend REST API

https://github.com/42code/42code-backend

https://github.com/42code/42code-backend/tree/master/src/controllers



# Step 1: Create TypeScript classes for Country and State

File Explorer

```
src> create class Country {  
  name: string;  
  code: string;  
}
```

File Explorer

```
src> create class State {  
  name: string;  
  code: string;  
  country: string;  
}
```

File Explorer

```
src> create class State {  
  name: string;  
  code: string;  
}
```

File Explorer

```
src> create class State {  
  name: string;  
  code: string;  
  country: string;  
}
```



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

File: `src/Form/Service.php`

```
1 namespace Form;
2
3 use Symfony\Component\HttpFoundation\Request;
4 use Symfony\Component\HttpFoundation\Response;
5
6 class Service
7 {
8     public function __construct($url)
9     {
10         $this->url = $url;
11     }
12
13     public function getCountryList()
14     {
15         $url = $this->url . 'countries';
16         $response = $this->get($url);
17         $data = json_decode($response->getContent(), true);
18         return $data['countries'];
19     }
20
21     public function getStates($country)
22     {
23         $url = $this->url . 'states/' . $country;
24         $response = $this->get($url);
25         $data = json_decode($response->getContent(), true);
26         return $data['states'];
27     }
28
29     private function get($url)
30     {
31         $client = new \GuzzleHttp\Client();
32         $response = $client->get($url);
33         return $response;
34     }
35 }
```

URL

Response: get country list  
get states by country

URL: get country list  
get states by country

## Step 3: Update checkout component to retrieve countries from service

```
export class CheckoutComponent implements OnInit {  
  countries: Country[] = [];  
  constructor(private router: Router,  
              private httpClient: HttpClient) {}  
  ngOnInit() {  
    // retrieve countries  
    this.httpClient.get('http://localhost:3000/countries').subscribe(  
      res => {  
        console.log('Retrieved countries: ', res);  
        this.countries = res;  
      }  
    );  
  }  
}
```

When I run the application, page does not render





## Step 5: Update checkout component, add event handler

src/components/Checkout.js

```
import React, {useState} from 'react';
```

```
const Checkout = () => {
```

```
  const [country, setCountry] = ''
```

```
  const [state, setState] = ''
```

```
  const handleChange = (e) => {
```

```
    const {name, value} = e.target.value
```

```
    if (name === 'country') {
```

```
      setCountry(value)
```

```
    } else if (name === 'state') {
```

```
      setState(value)
```

```
    }
```

```
    setCountry(value)
```

```
  } // add event listener to form
```

```
  return (
```

```
    <div>
```

```
      <input type="text" value={country} onChange={handleChange} name="country" />
```



# Checkout Form Validation



# Form Validation

- Before we submit the form to the backend we perform validation
- Check for required fields, min length etc ...



# Angular Validation

- Angular has a set of built-in validation rules.

Class	Description
<code>required</code>	Value must be non-empty value
<code>email</code>	Must be a valid email address
<code>max</code>	Must be a number no greater
<code>minlength</code>	Length must be at least
<code>max-length</code>	Length must be at most
<code>pattern</code>	Value must be regular expression pattern
<code>unique</code>	Value must be the unique value in the collection
<code>url</code>	

<https://angular.io/reference/forms/validators>

# Additional Validation Features

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

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

# Checkout Form Validation

From: The Road

## 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)

# Development Process

Step 03/04

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

## 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 validation

Example

```
new FormControl('initial value', validators)
```

array of validators

NOTE

```
new FormControl('', [Validators.required, Validators.maxLength(20)])
```

# Step 1: Specify validation rules for form controls

40. @Validations

```
1000. @Validations(1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

```
1000. formValidation (1000) { 1000. formValidation (1000)
```

Must be a valid email address

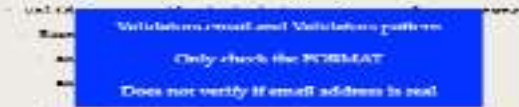
Must be a valid email address

# Validators.email vs Validators.pattern

- This may sometimes be used to validate AngularJS 1.4 domains, even 1.3-1.3.9
- `Validators.email`, even 1.3, only checks for `username@domain.tld`
- Based on this — the following is valid email format
  - `mail@low2code` ————— **This is valid!**
  - `mail@pycomail`
- It does not check for domain name length, `mail@low2code`, versus `mail@pycomail`, etc.
- As a result, we use a regular expression with `Validators.pattern`

# Validators.email vs Validators.pattern

When many wonder why we are not using Angular's `Validators.email` we get



It does not check for domain name validity, email server, etc. as email validation

As a result, we use a regular expression with `Validators.pattern`



## Step 2: Define Getter methods to access form controls

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

```
get LastName() {  
    return this.checkoutFormGroup.get('customer-lastName');  
}
```

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

...

```
this.checkoutFormGroup = this.formBuilder.group({  
    customer: this.formBuilder.group({  
        firstName: this.formBuilder.control(''),  
        lastName: this.formBuilder.control(''),  
        email: this.formBuilder.control(''),  
        password: this.formBuilder.control(''),  
        confirmPassword: this.formBuilder.control('')  
    })  
});  
  
// Add form controls to the form group  
this.checkoutFormGroup.addControl('customer-firstName', this.formBuilder.control(''));  
this.checkoutFormGroup.addControl('customer-lastName', this.formBuilder.control(''));  
this.checkoutFormGroup.addControl('customer-email', this.formBuilder.control(''));  
this.checkoutFormGroup.addControl('customer-password', this.formBuilder.control(''));  
this.checkoutFormGroup.addControl('customer-confirmPassword', this.formBuilder.control(''));
```





## Step 3: Update HTML template to display error messages

isDirty: did error change field value?

```
<label>{{name}} Name: {{value}}  
<input type="text" value="{{value}}" isDirty="{{ isDirty }}" />
```

```
<div ng-if=" $scope.isDirty || $scope.isDirty || $scope.isDirty" class="alert alert-danger">
```

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

## Step 3: Update HTML template to display error messages

touched: did field lose focus?

```
<label>{{name.name}} </label>  
<input type="text" value="{{name.name}}" class="{{name.touched ? 'error' : ''}}"/>
```

```
<div {{input.isInvalid as isValid}} {{input.touched}} class="{{isValid ? 'error' : ''}}">
```

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



## Step 3: Update HTML template to display error messages

```
<label class="name" for="name">
  Name: <input type="text" value="" id="name" />
</label>
```

Display special error message

```
<div class="form-group" id="form-group" <div class="form-group" id="form-group">
  <input type="text" value="" id="name" />
  <div class="error" id="error">
```

```
<div class="error" id="error">
  <div class="error" id="error">
    <div class="error" id="error">
      <div class="error" id="error">
        <div class="error" id="error">
```

```
<div class="error" id="error">
  <div class="error" id="error">
    <div class="error" id="error">
      <div class="error" id="error">
        <div class="error" id="error">
```

## Step 3: Update HTML template to display error messages

```
<label class="name" for="name">Name</label>  
<input type="text" id="name" value="" class="form-control" />
```

```
<div class="form-group">{{ form.name.label }} {{ form.name }} {{ form.name.errors }} </div>
```

```
<div class="form-group">{{ form.name.label }} {{ form.name }} {{ form.name.errors }}</div>  
<div class="form-group">{{ form.name.label }} {{ form.name }} {{ form.name.errors }}</div>
```

```
<div class="form-group">{{ form.name.label }} {{ form.name }} {{ form.name.errors }}</div>  
<div class="form-group">{{ form.name.label }} {{ form.name }} {{ form.name.errors }}</div>
```

```
</div>
```

Displaying validation error messages



A screenshot of a web form with a single text input field. The input field is empty and has a light blue border. Below the input field, there is a red error message that reads: "Name is required". The form is styled with a light blue background and a white border.



## Step 3: Update HTML template to display error messages

Form validation failed

```
<div id="emailForm">
```

```
<input type="text" value="Email Address" /> <input type="button" value="Send" />
```

```
<div id="errorMsg" style="display: none; color: red; text-align: center; font-weight: bold; margin-top: 10px;"></div>
```

```
<div id="errorMsg" style="display: none; color: red; text-align: center; font-weight: bold; margin-top: 10px;"></div>
```

```
<div id="errorMsg" style="display: none; color: red; text-align: center; font-weight: bold; margin-top: 10px;"></div>
```

```
</div>
```

```
<div id="errorMsg" style="display: none; color: red; text-align: center; font-weight: bold; margin-top: 10px;"></div>
```

```
<div id="errorMsg" style="display: none; color: red; text-align: center; font-weight: bold; margin-top: 10px;"></div>
```

```
</div>
```

```
</div>
```

```
</div>
```

Display error messages

## Step 4: Add event handler to check validation on submit

Triggering all fields triggers the display of the error messages

```
onSubmit: function(e) {
    // Prevent the default form submission
    e.preventDefault();

    // Trigger the validation
    $('#form').validate();

    // If the form is valid, submit it
    if ($('#form').valid()) {
        // Submit the form
        $('#form').submit();
    }
}
```

# Checkout Form Custom Validator Rule



# White Space

- This form internally has a problem with white space

In the required fields

If you enter ONLY whitespace ... it **passes** ... YIPES!

Should **fail**!

## Custom Validator

- We can resolve this with a custom validator, and don't get it triggered
  - Check the field value
  - If it **ONLY** has whitespace
    - then return an error (failed)
    - else return null (passed)

# Development Process

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



# Step 1: Define custom validator rule

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

```
export class CustomValidator {
```

```
  // Custom validation
```

```
  validate(FormControl): ValidationErrors | null {
```

```
    // Check if the value is not empty (assuming we have a required field)  
    if (!FormControl.value || !value) { return ValidationErrors.required; }
```

```
    // Optionally, return other errors  
    return { 'customError': true; }
```

```
  }  
  // Optionally, return null  
  return null;  
}
```

CustomValidator

If value is not empty, check other

If value is not empty, validation is required

If validation is not passed,

return validation error

Validation error key

The HTML template will check for this error key

## Step 2: Specify custom validator rule for form controls

Our custom validator  
method name

```
FORM_VALIDATOR_RULES = [TYPE, FORM_VALIDATOR_RULES]
```

```
def validate(form_data, form_data_group):
```

```
    """Validate the form data group"""
```

```
    if not isinstance(form_data_group, dict):
```

```
        raise ValueError("form_data_group must be a dict")
```

```
    for field in form_data_group:
```

```
        if not isinstance(field, str):
```

```
            raise ValueError("field must be a string")
```

```
        if not isinstance(form_data_group[field], dict):
```

```
            raise ValueError("form_data_group[field] must be a dict")
```

```
        if not isinstance(form_data_group[field]['value'], dict):
```

```
            raise ValueError("form_data_group[field]['value'] must be a dict")
```

```
        if not isinstance(form_data_group[field]['value']['value'], dict):
```

```
            raise ValueError("form_data_group[field]['value']['value'] must be a dict")
```

```
        if not isinstance(form_data_group[field]['value']['value']['value'], dict):
```

```
            raise ValueError("form_data_group[field]['value']['value']['value'] must be a dict")
```

```
        if not isinstance(form_data_group[field]['value']['value']['value']['value'], dict):
```

```
            raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
    if not isinstance(form_data_group[field]['value']['value']['value']['value'], dict):
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```

```
        raise ValueError("form_data_group[field]['value']['value']['value']['value'] must be a dict")
```



## Step 3: Update ITTML template to display error messages

If validation fails,  
then display error messages

```
<label>{{first_name}} {{last_name}}</label>  
<input type="text" value="{{first_name}}" id="first_name"/>
```

```
<div class="{{ first_name.isvalid and (first_name.dirty || first_name.touched) ? 'is-valid' : 'is-invalid' }}">
```

```
<div class="{{ first_name.errors.required || first_name.errors.maxLength }}">  
  {{first_name.errors.required || first_name.errors.maxLength}}</div>  
</div>
```

```
<div class="{{ first_name.errors.maxLength }}">  
  {{first_name.errors.maxLength}}</div>  
</div>
```

Use {{first\_name.errors.required}}  
validation error key

## Step 3: Update HTML template to display error messages

If validation fails,  
then display error messages

```
<label>{{form.name.label}}</label>  
<input type="text" value="{{form.name.value}}" id="{{form.name.id}}"/>
```

```
<div class="{{ form.name.errors | len }} {{ form.name.errors | first }}">
```

```
<div class="{{ form.name.errors | len }} {{ form.name.errors | first }}">  
  <div class="{{ form.name.errors | len }} {{ form.name.errors | first }}">  
    <div class="{{ form.name.errors | len }} {{ form.name.errors | first }}">  
      <div class="{{ form.name.errors | len }} {{ form.name.errors | first }}">
```

Use the error message in the  
validation error key

# Checkout Form Validation

## Shipping, Billing and Credit Card



# Finish up the remaining sections



## Development Process

More of the same

Test Library

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

## Checkout Form Review Cart Totals



# Update Checkout Form - Cart Totals

- This form currently has a button for cart totals.

We need to add the appropriate code to support this.



The screenshot shows a web form titled "Review Your Order". It contains three input fields: "Card number", "Expiry date", and "CVV". The "CVV" field is currently filled with "123 456 7890". Below these fields is a blue button labeled "Check out".

## Cart Service - Publishing messages/events

- Recall that we send messages/events to other components in our application
- For example, `CartLELisLineComponent` will subscribe to the `CartLine` event

```
CARTSERVICE will publish messages for:  
totalprice and totalquantity
```



## Application Interjection





# CartService - Publish events

CartService

dependencies:

```
use javax.persistence.EntityManager;
use javax.persistence.Query;
```

```
use com.vaadin.event.EventRouter;
use com.vaadin.event.EventUtil;
use com.vaadin.event.EventUtil;
use com.vaadin.event.EventUtil;
```

```
1 public void publishEvent(Event event) {
2     // ...
3     // ...
4     // ...
5 }
```

CartService

1. Use the publishEvent() method to publish events

2. Use the publishEvent() method to publish events

# CartStatusComponent subscribes to CartService

CartStatusComponent

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

import { CartService } from '../cart.service';

import { CartStatus } from '../cart-status';

import { CartItem } from '../cart-item';

import { CartItemComponent } from '../cart-item.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusService } from '../cart-status.service';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

import { CartStatusComponent } from './cart-status.component';

Import the CartService

Subscribe for events

When the user clicks the button, make the assignments to update UI

# CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService

CartStatusComponent subscribes to CartService



CartStatusComponent subscribes to CartService

## Publish / Subscribe

Similar approach for checkout component ... almost:

`CheckoutComponent` will subscribe to events that `Cart` fires upon

However, since `CheckoutComponent` is instantiated later in the application

will miss out on previous messages.

As a result, `CheckoutComponent` cart totals will incorrectly show as

- 0 for total quantity
- 0.00 for total price



Should be got a message of the  
shopping cart total

# 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 `replay/subject`

## ReplaySubject

- Recall, this just **is** used for **current events** for subscribers
- **ReplaySubject is a sub-class of Subject**
- Will also "replay events" for new subscribers who join later
- Keep a buffer of previous events and send to new subscribers

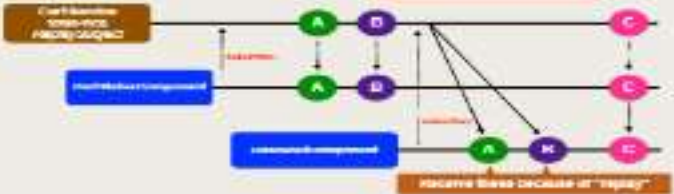
Copy, I was late to the meeting  
What was first said when I arrived?



# Subject (without replay)



# Replay Subject



# ReplySubject

```
export class CartDriver {  
  cartItems: CartItem[] = [];  
  emailSender: Subject<Email> = new ReplySubject<Email>();  
  emailReceiver: Subject<Email> = new ReplySubject<Email>();  
}
```

Keeps a buffer of previous answers.  
Sends previous answers. It never updates them.

## Replay Subject

- For more details on ReplaySubject, see online docs

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

## But wait ... there's another solution

- At first glance, ReplayCache just 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?

# BehaviorSubject

- Behaves like BehaviorSubject in a cache-like or Cache-like
- Has a notion of "current value"
- Stores the latest message / event ... and sends to new subscribers

Hi Mary, I have added the class according to what you have asked about today!

## From the Docs

Performance tests that prove to be useful for representing "subject over time"

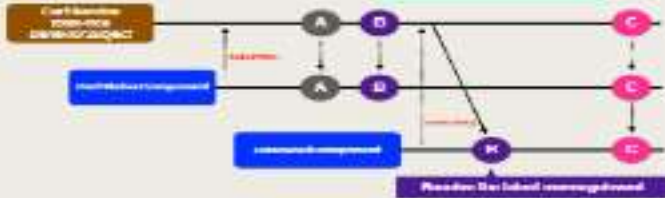
For instance, an event stream of birthdays is a `Grp` series, but the stream of a person's age would be a `BirthAndCumGrp` series.

<http://www.luv2code.com/rxjs/believing-subject>

# BehaviorSubject

Observable  
Observer

BehaviorSubject is a special kind of Observable that can also act as an Observer.





# 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

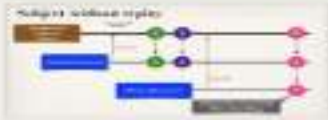
# BehaviorSubject

- For more details on BehaviorSubject Code just, use similar class

<http://www.fv2code.com/rxjs-behavior-subject>

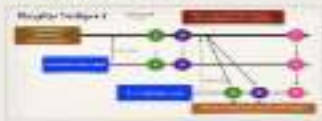
## Recap

Name	Description
<b>Subject</b>	<ul style="list-style-type: none"><li>• There, we'll keep up buffer of previous events, <i>if it's not empty</i></li><li>• <b>Subscriber only receives new events after they are subscribed</b></li></ul>



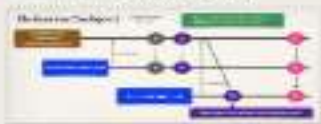
# Recap

TL;DR	Things to know
Event-driven architecture	<ul style="list-style-type: none"><li>• Event is buffer of all previous events</li><li>• Once subscribed, subscriber receives a replay of all previous events</li></ul>



# Recap

TL;sum	Things to know
Behavioral pattern	<ul style="list-style-type: none"><li>Has a buffer of the last event</li><li>Once subscribed, subscriber receives the latest event sent prior to subscribing</li></ul>



# Recap

Flavor	Description
Default	<ul style="list-style-type: none"><li>Does not keep a buffer of previous events.</li><li>Subscriber only receives new events after they are subscribed.</li></ul>
Timestamped	<ul style="list-style-type: none"><li>Has a buffer of all previous events.</li><li>Does not discard, subscribe to receive a copy of all previous events.</li></ul>
Instantaneous	<ul style="list-style-type: none"><li>Has a buffer of the last event.</li><li>Once subscribed, subscriber receives the latest event sent prior to subscribing.</li></ul>

# Development Process

Developing

## 1. Update for Checkout Component

1. Inject `CartService` into `CheckoutComponent`
2. In `ngOnInit` method, add new method: `reviewCartItems()`
3. Add code for new method: `reviewCartDetails()`

## 2. Update for Cart Service

1. Change `subject` to `Observable<Subject>`

## Step 1.1: Inject CartService into CheckoutForm

src/main/java

```
package com.mindia2code.springboot.initialdata.model;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class ShoppingCartService {
```

Inject CartService



## Step 1.2: ngOnit, call new method: reviewCartDetails

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {  
  ngOnInit(): void {  
    this.reviewCartDetails();  
  }  
}
```

Call new method



## Step 2.1: Change Subject to BehaviorSubject

Change the BehaviorSubject Subject

```
export class CartService {  
  cartItems: CartItem[] = [];  
  totalPrice: BehaviorSubject<number> = new BehaviorSubject<number>(0);  
  totalQuantity: BehaviorSubject<number> = new BehaviorSubject<number>(0);  
}
```

Set initial value

totalPrice = 0  
totalQuantity = 0

# Checkout Form

## Save The Order - Backend

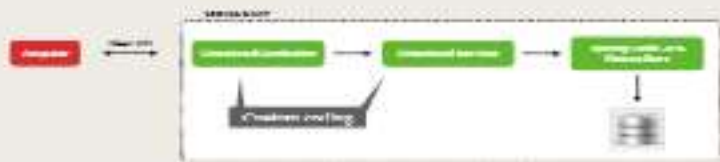


## Save The Order

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



# Application Architecture



## 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???

# Why Not Spring Data REST???

- Spring Data REST is great for head-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
  - Offer custom business logic ...

Spring Data REST  
is very limited in terms of customization  
for custom business logic and processing.  
There is custom controller API service



# Database Diagram



# Class Diagram - Entity Classes



# Data Transfer Object

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



# REST API

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

HTTP Method	URI	Action
POST	/api/checkout/purchase	new purchase order

Example:

```
{
  "id": 1,
  "productId": 1,
  "quantity": 1,
  "unitPrice": 1000,
  "totalPrice": 1000
}
```

### Sample JSON

## Page 10000

Copyright

**Energy** **Analysis**

### Training Activities

100

**Keywords:** child sexual abuse; disclosure; social support

[illegible]

# Development Process - Spring Boot



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

I will write some code!

# UUID

- We want a tracking identifier that is
  - Hard to guess: random / unique
- UUID: Universally Unique Identifier
  - Standardized methods for generating unique IDs
  - Available in four versions, we will use Version 4 (random)

Google: wikipedia uuid

# What about Uniqueness / Collisions??

The probability of collision is VERY low ... negligible

[Simple UUIDs](#)

And collisions are not a Wikipedia entry

Probability of a duplicate in 100 billion version-4 UUIDs is one in a trillion

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 yes, then repeat previous two steps until no duplicates found



# Checkout Form

## Save The Order - Front End



# Save The Order

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



# Data Transfer Object

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



# Development Process



## 1. Create database schema

1. `createUser`, `updateUser`, `deleteUser`, `Address` Functions

## 2. Create Client Application

1. Make REST API call to `System` from `WebApp`

## 3. Update CharacterService Component

1. Inject `CharacterService` and `Trainer`
2. Update `initAndStart()` method to collect from `data` via `CharacterCollection`

Let's Write Some Code!