Angular Front End

Angular Front End

© luv2code LLC

Angular Front End

- Create Angular Front End components

© luv2code LLC

Angular Front End

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

© luv2code LLC

Angular Front End

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

© luv2code LLC

Development Process

Step-By-Step

© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project

© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project
2. Create Angular component for product-list

© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project
2. Create Angular component for product-list
3. Develop TypeScript class for Product

© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project
2. Create Angular component for product-list
3. Develop TypeScript class for Product
4. Create Angular service to call REST APIs

© luv2code LLC

Development Process

Step-By-Step

1. Create Angular project
2. Create Angular component for product-list
3. Develop TypeScript class for Product
4. Create Angular service to call REST APIs
5. Update Angular component to subscribe to data from Angular service

© luv2code LLC

Development Process

Step-By-Step

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

© luv2code LLC

Development Process

Step-By-Step

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

© luv2code LLC

Step 1: Create Angular project

© luv2code LLC

Step 1: Create Angular project

- Create new project using Angular CLI

© luv2code LLC

Step 1: Create Angular project

- Create new project using Angular CLI

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


© luv2code LLC

Step 2: Create Angular component for product-list

© luv2code LLC

Step 2: Create Angular component for product-list

- Create new component using Angular CLI

© luv2code LLC

Step 2: Create Angular component for product-list

- Create new component using Angular CLI

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


© luv2code LLC

Step 2: Create Angular component for product-list

- Create new component using Angular CLI

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

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

© luv2code LLC

Step 3: Develop TypeScript class for Product

© luv2code LLC

Step 3: Develop TypeScript class for Product

- Create new class

© luv2code LLC

Step 3: Develop TypeScript class for Product

- Create new class

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


© luv2code LLC

Step 3: Develop TypeScript class for Product

- Create new class

Placed in sub-directory: common

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


© luv2code LLC

Step 3: Develop TypeScript class for Product

- Create new class

Placed in sub-directory: common

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

File: src/app/common/product.ts

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


© luv2code LLC

Step 4: Create Angular service to call REST APIs

© luv2code LLC

Step 4: Create Angular service to call REST APIs

- Angular "Service" is code developed in TypeScript

© luv2code LLC

Step 4: Create Angular service to call REST APIs

- Angular "Service" is code developed in TypeScript
- Service is a helper class that provides desired functionality

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

© luv2code LLC

Application Interaction

© luv2code LLC

Application Interaction

Angular Project

© luv2code LLC

Application Interaction

© luv2code LLC

Application Interaction

© luv2code LLC

Application Interaction

© luv2code LLC

Step 4: Create Angular service to call REST APIs

© luv2code LLC

Step 4: Create Angular service to call REST APIs

- REST client provided by Angular:

© luv2code LLC

Step 4: Create Angular service to call REST APIs

- REST client provided by Angular:
  - **`HttpClient`** ... part of **`HttpClientModule`**

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

File: src/app/app.module.ts

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

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


© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

File: src/app/app.module.ts

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

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

Support for HttpClientModule

© luv2code LLC

Step 4: Create Angular service to call REST APIs

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

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

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

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

  constructor(private httpClient: HttpClient) { }

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

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


© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

Our service can be injected into other classes / components

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

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

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

  constructor(private httpClient: HttpClient) { }

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

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


© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

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

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

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

  constructor(private httpClient: HttpClient) { }

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

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

Our service can be injected into other classes / components

Inject httpClient

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

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

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

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

  constructor(private httpClient: HttpClient) { }

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

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

Our service can be injected into other classes / components

Inject httpClient

Returns an observable

Map the JSON data from Spring Data REST to Product array

© luv2code LLC

Step 4: Create Angular service to call REST APIs

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

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

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

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

  constructor(private httpClient: HttpClient) { }

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

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

Our service can be injected into other classes / components

Inject httpClient

Unwraps the JSON from Spring Data REST _embedded entry

Returns an observable

Map the JSON data from Spring Data REST to Product array

© luv2code LLC

Step 5: Develop Angular to subscribe to data

© luv2code LLC

Step 5: Develop Angular to subscribe to data

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

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

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

  products: Product[];

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.listProducts();
  }

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


© luv2code LLC

Step 5: Develop Angular to subscribe to data

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

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

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

  products: Product[];

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.listProducts();
  }

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

Method is invoked once you "subscribe"

© luv2code LLC

Step 5: Develop Angular to subscribe to data

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

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

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

  products: Product[];

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.listProducts();
  }

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

Method is invoked once you "subscribe"

Assign results to the Product array

© luv2code LLC

Step 6: Display the Data in an HTML page

© luv2code LLC

Step 6: Display the Data in an HTML page

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

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


© luv2code LLC

Step 6: Display the Data in an HTML page

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

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

Products

JavaScript - The Fun Parts: $19.99

Spring Framework Tutorial: $29.99

Kubernetes - Deploying Containers: $24.99

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

The Go Programming Language: A to Z: $24.99

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

- By default, this coding will fail

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

- By default, this coding will fail
- Web browsers will not allow script code to call APIs not on same origin

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

- 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

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

- By default, this coding will fail
- Web browsers will not allow script code to call APIs not on same origin
- Known as Same-origin policy
- Same-origin is composed of: scheme/protocol, hostname, port number

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

- By default, this coding will fail
- Web browsers will not allow script code to call APIs not on same origin
- Known as Same-origin policy
- Same-origin is composed of: scheme/protocol, hostname, port number

- Can relax this by adding "Cross-Origin Resource Sharing (CORS)" on server side application

© luv2code LLC

Step 7: Add CrossOrigin support to Spring Boot

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

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

© luv2code LLC

Add CrossOrigin support to Spring Boot App

© luv2code LLC

Add CrossOrigin support to Spring Boot App

File: ProductRepository.java

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

}
```


© luv2code LLC

Add CrossOrigin support to Spring Boot App

File: ProductRepository.java

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

}
```

Multiple

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


© luv2code LLC

Add CrossOrigin support to Spring Boot App

File: ProductRepository.java

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

}
```

Multiple

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

Wildcard (any website)

```
@CrossOrigin
```


© luv2code LLC