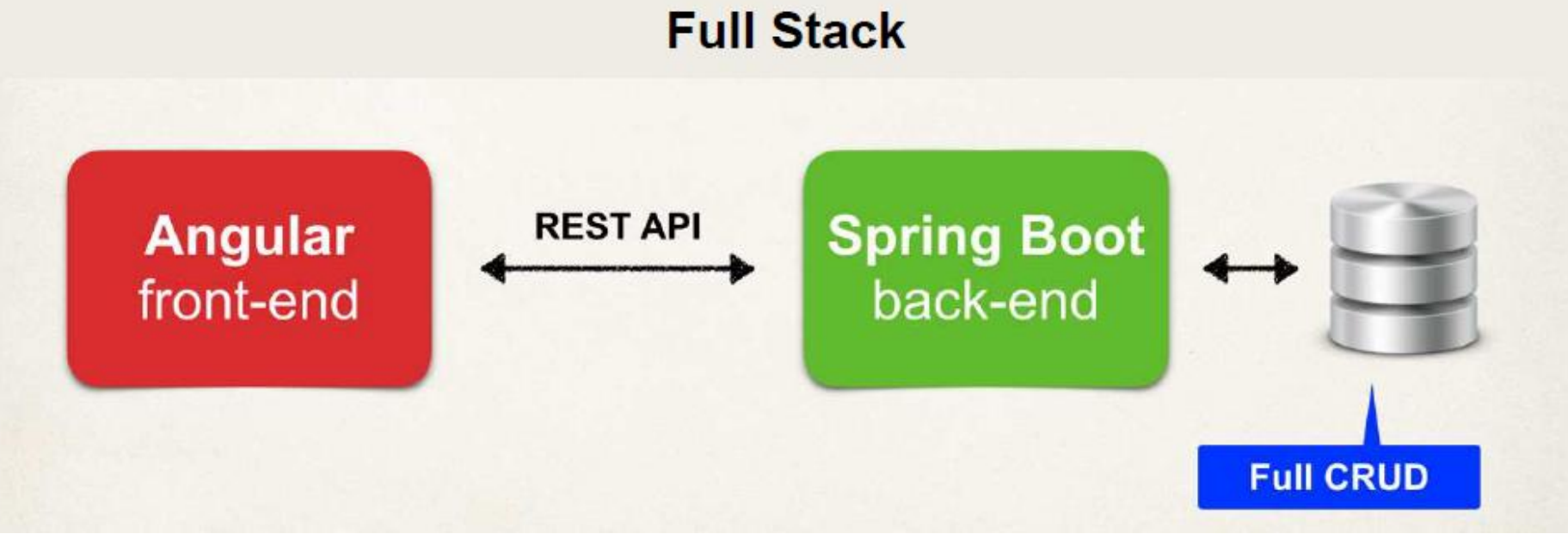
FullStack Project Overview

Full Stack Project Overview

Project

- Build Real-time eCommerce App

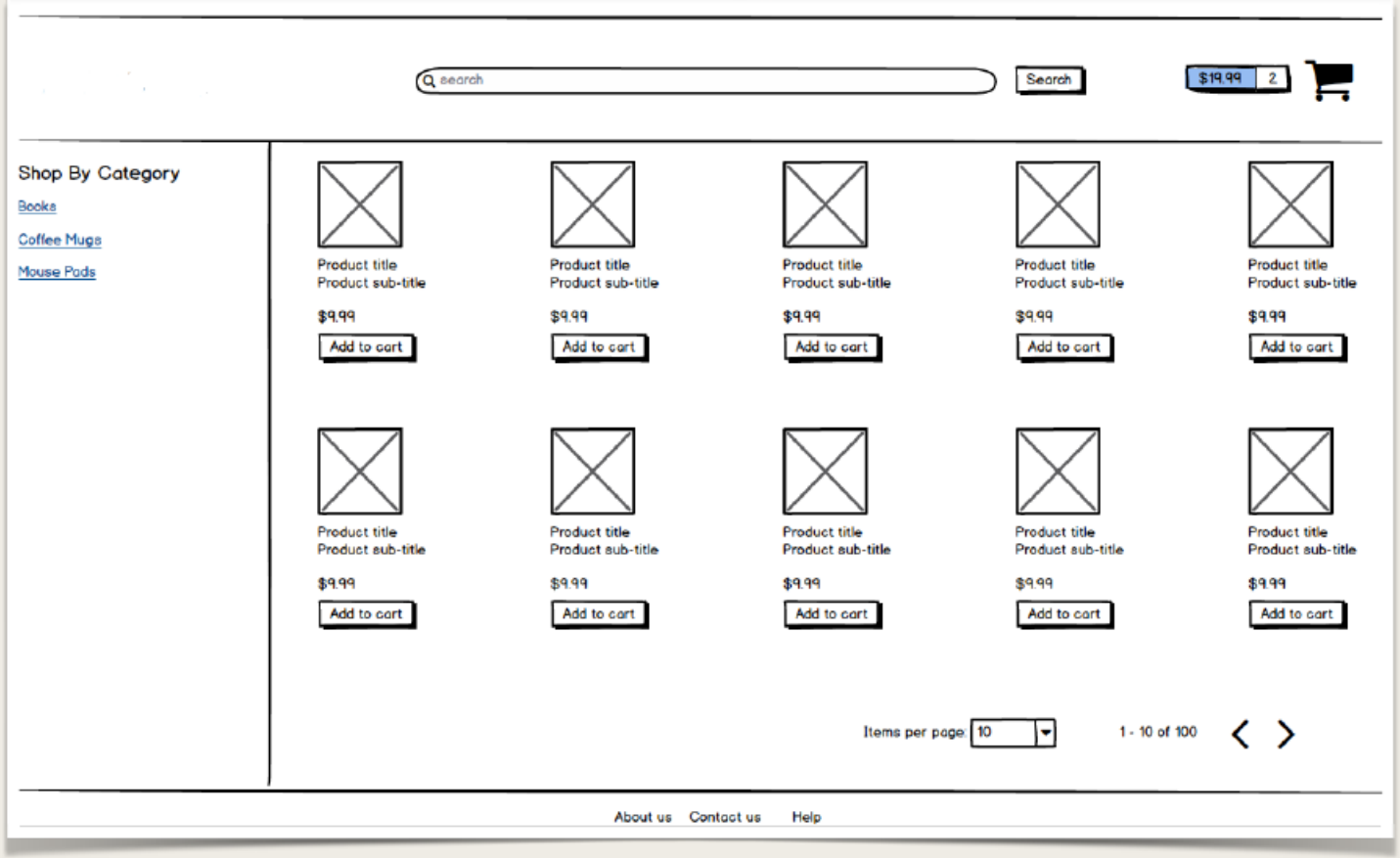


Requirements

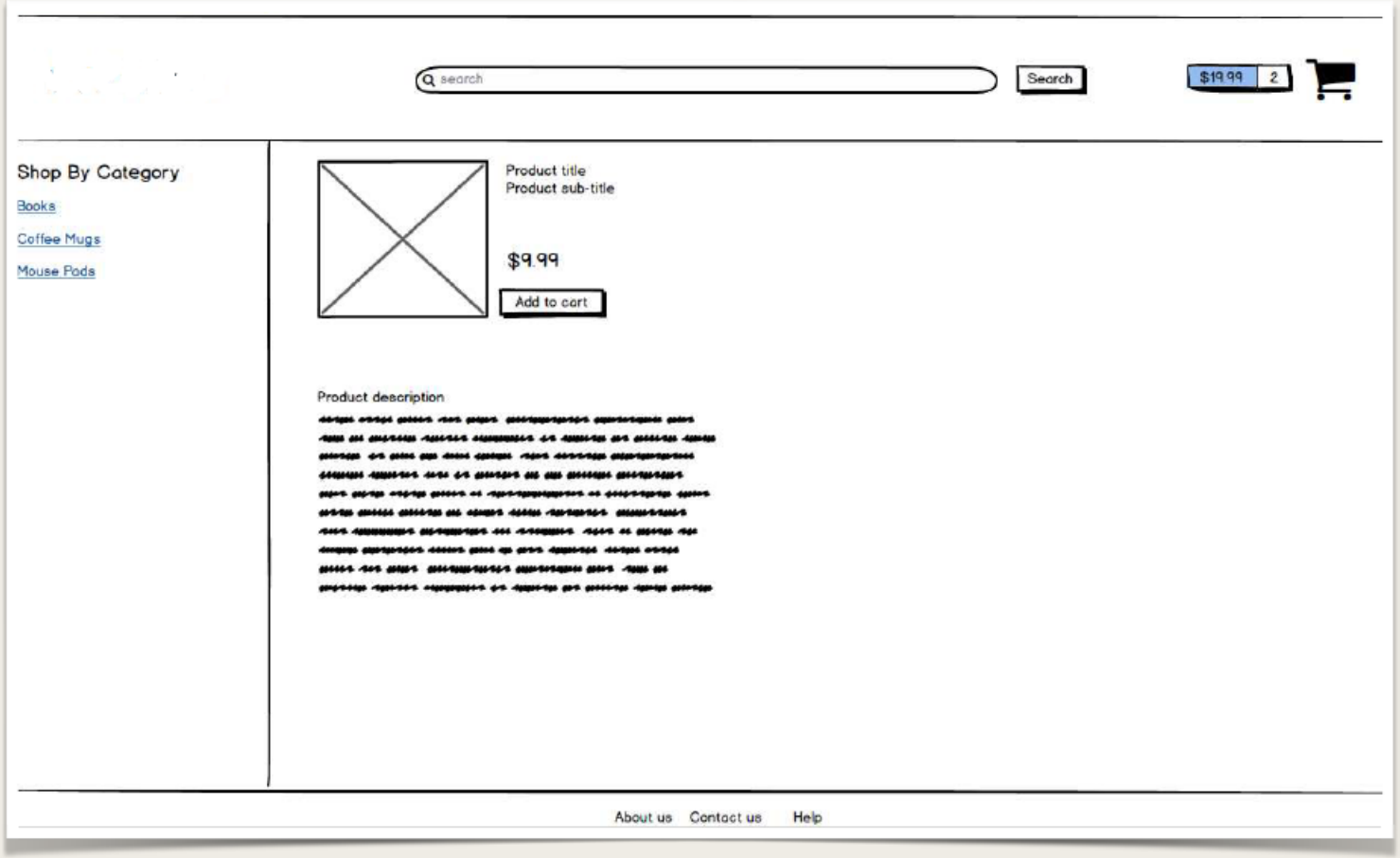
From: The Boss

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

Wireframes - Home Page



Wireframes - Product Details



Development Process

Step-By-Step

1. Set up the database tables

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

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

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

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

Dependencies to be added

- Spring Data JPA
- Rest Repositories
- MySQL Driver
- Lombok

Project Lombok

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

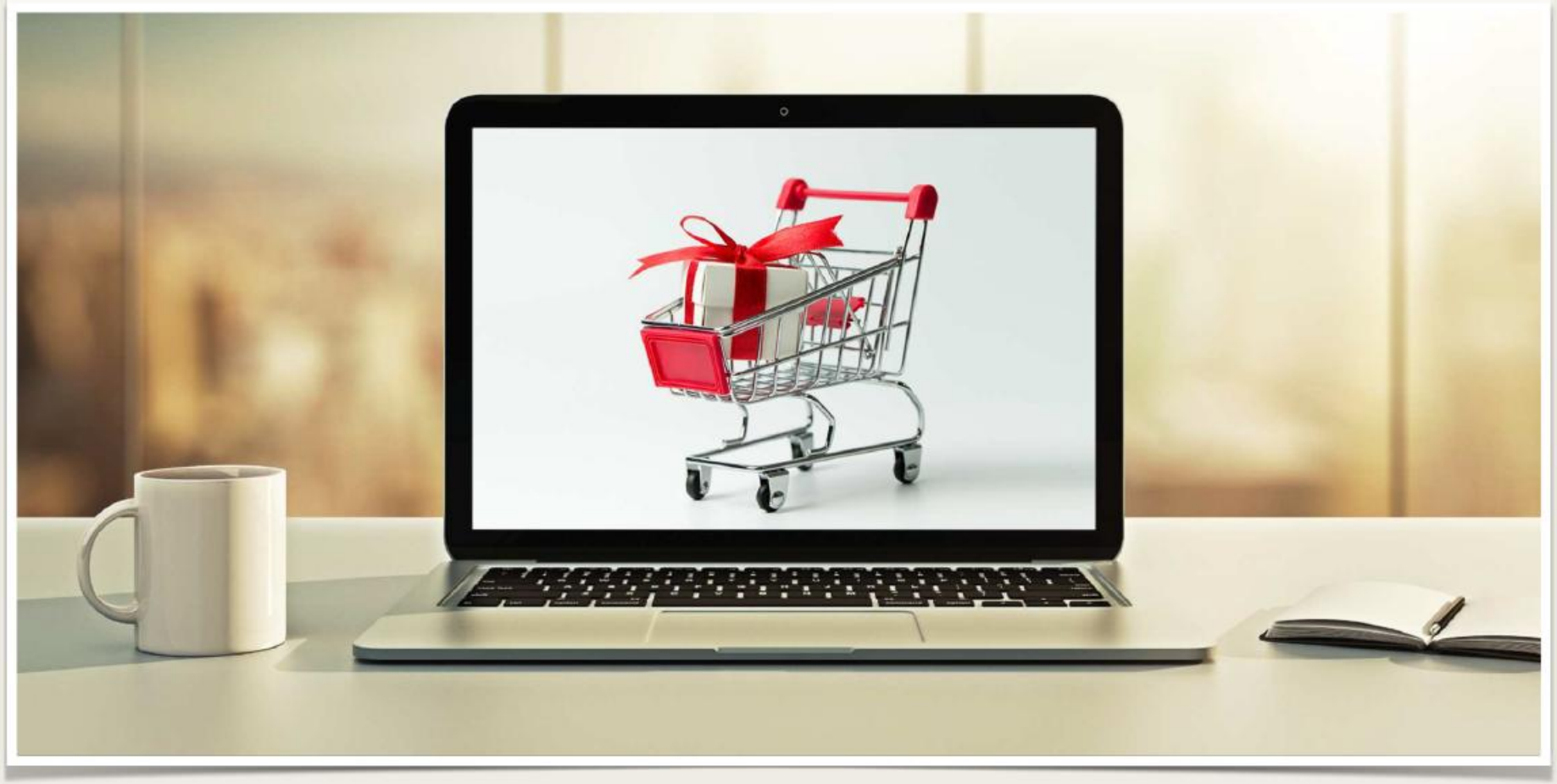
http://www.projectlombok.org

Develop Product and ProductCategory

Application.properties

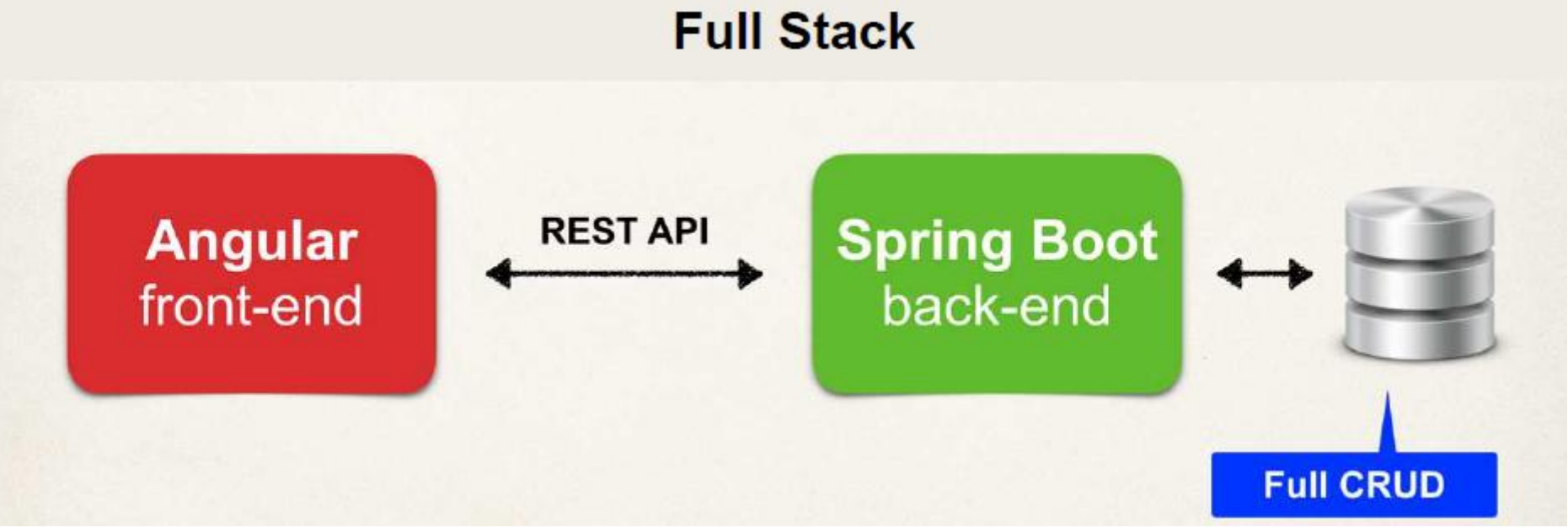
```
spring.application.name=spring-boot-ecommerce
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/full-stack-ecommerce
spring.datasource.username=root
spring.datasource.password=admin

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.data.rest.base-path=/api
```

Angular Front End



Angular Front End

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



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

Step 1: Create Angular project

- Create new project using Angular CLI

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

ng new - -no-standalone angular-ecommerce

For angular 17 onwards

Go to getbootstrap.com paste in index.html

- <meta charset="utf-8">
- <meta name="viewport" content="width=device-width, initial-scale=1">
- <title>Bootstrap demo</title>
- <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.7/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-LN+7fdVzj6u52u30Kp6M/trliBMCMKTyK833zpbD+pXdCLuTusPj697FH4R/5mcr" crossorigin="anonymous">

Clean up app.component.html

app.component.html M

angular-ecommerce > src > app > app.component.html > div.container

Go to component

```
1 <div class="container">
2   <h1 class="mt-3 mb-3">Products</h1>
3 </div>
```

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

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;
}
```

Step 3: Develop TypeScript class for Product

File: src/app/common/product.ts

```
export class Product {

    constructor(public sku: string,
                public name: string,
                public description: string,
                public unitPrice: number,
                public imageUrl: string,
                public active: boolean,
                public unitsInStock: number,
                public dateCreated: Date,
                public lastUpdated: Date
        ) {
    }
}
```

Remember, these are Parameter Properties

Declared by prefixing constructor argument with access modifier: `public`, `protected`, `private`, or `readonly`

Declares properties and assigns properties automagically.

Minimizes boilerplate coding!

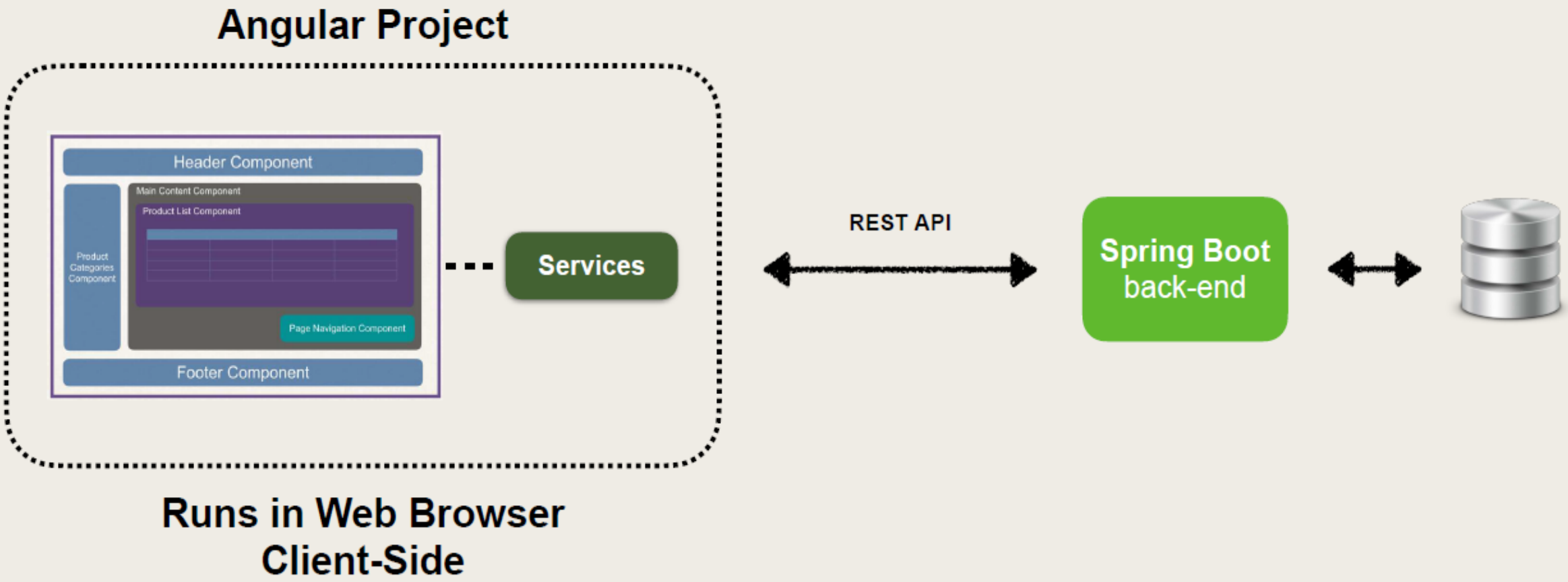
Step 4: Create Angular service to call REST APIs

- 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

Create angular service

- ng generate service services/product

Application Interaction



app.module.ts

angular-ecommerce > src > app > TS app.module.ts > ...

```
1   import { NgModule } from '@angular/core';
2   import { BrowserModule } from '@angular/platform-browser';
3
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6   import { ProductListComponent } from './components/product-list/product-list.component';
7   import { HttpClientModule } from '@angular/common/http';
8   import { ProductService } from './service/product.service';
9
10
11  @NgModule({
12    declarations: [
13      AppComponent,
14      ProductListComponent
15    ],
16    imports: [
17      BrowserModule,
18      AppRoutingModule,
19      HttpClientModule
20    ],
21    providers: [ProductService],
22    bootstrap: [AppComponent]
23  })
24  export class AppModule { }
25
```

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

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

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

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

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

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
```