Additional Techniques

1. Angular services and DI
2. Calling REST services

```
Demo app:    AngularDev/Demos/06-AdditionalTechniques/DemoApp
To install:  npm install
To run:      ng serve
```

Section 1: Angular Services and DI

- Overview
- Dependency injection
- Services and DI example
- Generating a service
- Implementing the service
- Injecting the service
- Singleton service instance

Overview

- Don't put this kind of code in your components:
  - Calling REST services
  - Interacting with Web Sockets
  - Accessing state in local storage
  - Etc.

- Instead:
  - Put this functionality in reusable *service classes*
  - Inject services into components, as needed

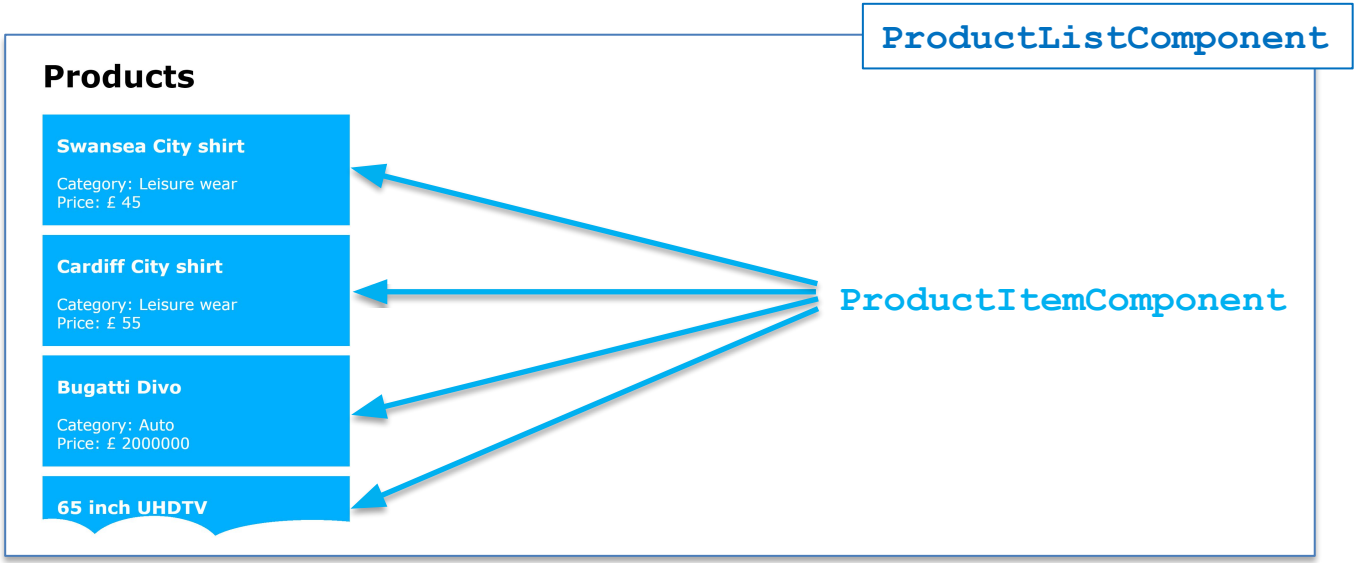
Dependency Injection

- Angular supports constructor dependency injection
  - In your component constructor, define the services you want to be injected as parameters
  - Angular will inject suitable objects

- How does dependency injection work in Angular?
  - By default, the root module *provides* (creates) services
  - Angular injects services into component constructors

Services and DI Example

- Let's see an example of services and DI
  - In the demo app, click the **Services and DI** link



Generating a Service

- We've put the responsibility for "getting products" into the `ProductService` class

- We used Angular CLI to generate the `ProductService` class as follows:

```
ng g service product
```

```
CREATE src/app/product.service.spec.ts (362 bytes)
CREATE src/app/product.service.ts (136 bytes)
```

Implementing the Service

- Here's our implementation of `ProductService`
  - Note the `providedIn` property in `@Injectable()`
  - Specifies the service will be provided by `'root'`, i.e. the root module in the application

```
import { Injectable } from '@angular/core';
import { Product } from './product'

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  getProducts(): Array<Product> {
    var products: Array<Product> = […];
    return products;
  }
}
```

product.service.ts

Injecting the Service

- You can inject service(s) into a component constructor as follows:

```
import { ProductService } from '../product.service'
…

@Component(…)
export class ProductListComponent {

  products: Array<Product>;

  constructor(productService: ProductService) {
      this.products = productService.getProducts();
  }
}
```

product-list.component.ts

Singleton Service Instances

- Recall the service class is decorated as follows:

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

- This means the service object is *provided* (created) by the root module
  - The service object is created as a singleton
  - The same instance will be injected into all components
  - Singleton is common for "infrastructure" services

Section 2: Calling REST Services

- Overview
- REST service example
- Pinging the REST service
- Example Angular REST client
- Adding support for REST clients
- Calling a REST service
- Consuming the REST result
- Displaying the REST result

Overview

- REST services are resource-centric services
  - Endpoints (URIs) represent resources
  - Endpoints are accessible via standard HTTP(S)
  - Endpoints can be represented in a variety of formats, e.g. JSON, XML, plain text, etc.

- Rest services play a vital role in Angular applications
  - SPAs invoke REST services to get/update state
  - Use JSON data (or maybe XML)

REST Service Example

- We've implemented a simple REST service
  - Returns product data

- The REST service is a Node.js app, run as follows:
  - Open a Command Prompt window in the `Server` folder
  - Run the following commands:

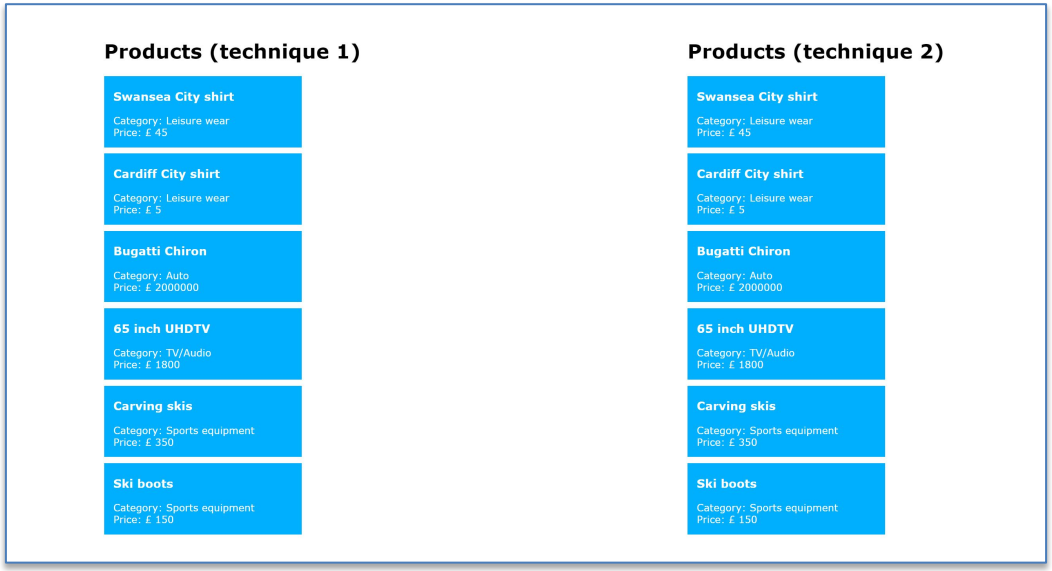
```
npm install

npm start
```

Pinging the REST Service

- To ping the REST service, open a browser window and enter the following URL:
  - http://localhost:8080/api/products

Example Angular REST Client

- Let's see an example of an Angular REST client
  - In the demo app, click the **Angular REST client** link



Adding Support for REST Clients

- If you want your Angular app to call a REST service, your module must import `HttpClientModule`

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

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

app.module.ts

Calling a REST Service (1 of 2)

- To call a REST service from Angular:
  - Inject the standard `HttpClient` service
  - Call `get(),put(),post(),delete()`

- For full details, see:
  - https://angular.io/api/common/http/HttpClient

Calling a REST Service (2 of 2)

- We've defined an Angular service class to encapsulate calls to the "get products" REST endpoint
  - The call is asynchronous, returns an `Observable<T>`

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
…
@Injectable({providedIn: 'root'})
export class ProductViaRestService {

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

  constructor(private http: HttpClient) {}

  getProducts(): Observable<Array<Product>> {
    return this.http.get(this.baseUrl) as Observable<Array<Product>>;
  }
}
```

product-via-rest.service.ts

Consuming the REST Result

- We consume the `Observable<T>` in 2 ways…

```
@Component(…)
export class ProductListViaRestComponent implements OnInit {

  productsTechnique1!: Observable<Array<Product>>;
  productsTechnique2!: Array<Product>;

  constructor(private productViaRestService: ProductViaRestService) {}

  ngOnInit(): void {
    this.productsTechnique1 = this.productViaRestService.getProducts();

    this.productViaRestService.getProducts().subscribe({
        next: (data:any) => this.productsTechnique2 = data,
        error: (_:any)  => console.log("Error")
    });
  }
}
```

product-list-via-rest.component.ts

Displaying the REST Result

- We display the REST result in 2 ways…

```
<div>
    <div class="halfColumn">
        <h1>Products (technique 1)</h1>
        <div *ngFor="let p of productsTechnique1 | async">
            <app-product-item [product]="p"></app-product-item>
        </div>
    </div>

    <div class="halfColumn">
        <h1>Products (technique 2)</h1>
        <div *ngFor="let p of productsTechnique2">
            <app-product-item [product]="p"></app-product-item>
        </div>
    </div>
</div>
```

product-list-via-rest.component.html

Summary

- Angular services and DI
- Calling REST services