Explanation of `ActivatedRoute` in Angular:

In Angular, the `ActivatedRoute` is a service that provides information about the route associated with a component that is loaded in an outlet. It allows you to access information about the route, including route parameters, query parameters, and the URL segments.

The `ActivatedRoute` service is part of the `@angular/router` module and is typically injected into a component's constructor. Once injected, you can use it to access various properties and methods that provide information about the current route.

Key Properties and Methods of `ActivatedRoute`:

1. `snapshot` **Property:**
    - Provides a snapshot of the current route information when the component is first loaded.
    - Usage: `this.route.snapshot`
2. `params` **Observable:**
    - An observable that emits the route parameters as an object.
    - Usage: `this.route.params.subscribe(params => { /* handle params */ });`
3. `queryParams` **Observable:**
    - An observable that emits the query parameters as an object.
    - Usage: `this.route.queryParams.subscribe(queryParams => { /* handle queryParams */ });`
4. `fragment` **Property:**
    - Provides the URL fragment (the part of the URL after the `#` symbol).
    - Usage: `this.route.fragment`
5. `url` **Property:**
    - Provides the URL segments as an array.
    - Usage: `this.route.url.subscribe(urlSegments => { /* handle urlSegments */ });`

Example Using `ActivatedRoute`:

Let's extend the previous example to demonstrate how to use `ActivatedRoute` to access route parameters.

Step 1: Update Fashion Component

Update the `FashionComponent` to use `ActivatedRoute` to access the route parameters:

```
// fashion.component.ts
import { Component, OnInit } from '@angular/core';
```

```
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-fashion',
  template: `
    <h3>Fashion Component</h3>
    <p>Category: {{ fashionCategory }}</p>
    <p>Subcategory: {{ fashionSubcategory }}</p>
  `,
})
export class FashionComponent implements OnInit {
  fashionCategory: string;
  fashionSubcategory: string;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    // Access route parameters using snapshot
    this.fashionCategory = this.route.snapshot.params['category'];

    // Access route parameters using observable
    this.route.params.subscribe(params => {
      this.fashionSubcategory = params['subcategory'];
    });
  }
}
```

Step 2: Update Route Configuration

Update the route configuration in `app-routing.module.ts` to include a subcategory route parameter:

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { OnlineApplicationComponent } from './online-application/online-
application.component';
import { AppliancesComponent } from './appliances/appliances.component';
import { FurnitureComponent } from './furniture/furniture.component';
import { LightingComponent } from './lighting/lighting.component';
import { ElectronicsComponent } from './electronics/electronics.component';
import { MobilesComponent } from './mobiles/mobiles.component';
import { LaptopComponent } from './laptop/laptop.component';
import { FashionComponent } from './fashion/fashion.component';
import { MenComponent } from './men/men.component';
import { WomenComponent } from './women/women.component';
import { LoginComponent } from './login/login.component';

const routes: Routes = [
  // ... (previous routes)
```

```
  { path: 'online-application/fashion/:category', component: FashionComponent,
children: [
    { path: ':subcategory', component: FashionComponent }
  ]},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Step 3: Run Your App

Run your Angular application:

```
ng serve
```

Navigate to `http://localhost:4200/online-application/fashion/men` or `http://localhost:4200/online-application/fashion/women/shoes`, and observe how the `FashionComponent` now handles both the category and subcategory route parameters.

This example demonstrates how to use `ActivatedRoute` to access route parameters in Angular. The `ActivatedRoute` service provides a flexible and powerful way to interact with route-related information in your components.

Example

Let's create a more extensive example that includes multiple levels of nested routes, route parameters, and query parameters. In this example, we'll build a product catalog with categories, subcategories, and individual product details.

```
src/
|-- app/
|   |-- app.component.html
|   |-- app.component.ts
|   |-- app.module.ts
|   |-- app-routing.module.ts
|-- categories/
|   |-- categories.component.ts
|   |-- categories.component.html
|-- subcategories/
```

```
|   |-- subcategories.component.ts
|   |-- subcategories.component.html
|-- products/
|   |-- products.component.ts
|   |-- products.component.html
|-- online-application/
|   |-- online-application.component.ts
|   |-- online-application.component.html
|-- appliances/
|   |-- appliances.component.ts
|   |-- appliances.component.html
|-- furniture/
|   |-- furniture.component.ts
|   |-- furniture.component.html
|-- lighting/
|   |-- lighting.component.ts
|   |-- lighting.component.html
|-- electronics/
|   |-- electronics.component.ts
|   |-- electronics.component.html
|-- mobiles/
|   |-- mobiles.component.ts
|   |-- mobiles.component.html
|-- laptop/
|   |-- laptop.component.ts
|   |-- laptop.component.html
|-- fashion/
|   |-- fashion.component.ts
|   |-- fashion.component.html
|-- men/
|   |-- men.component.ts
|   |-- men.component.html
|-- women/
|   |-- women.component.ts
|   |-- women.component.html
|-- login/
|   |-- login.component.ts
|   |-- login.component.html
|-- product-catalog/
|   |-- product.model.ts
|   |-- subcategory.model.ts
|   |-- category.model.ts
|   |-- product.service.ts
|-- assets/
|   |-- (any static assets like images)
|-- environments/
|   |-- environment.prod.ts
|   |-- environment.ts
|-- index.html
|-- main.ts
|-- styles.css
|-- tsconfig.app.json
|-- tsconfig.json
```

```
|-- tsconfig.spec.json
|-- angular.json
|-- package.json
|-- README.md
```

This structure is based on the assumption that each component has its own folder containing the component's TypeScript and HTML files. The `product-catalog` folder contains data models and services related to the product catalog. The `assets` folder can be used to store static assets like images, and the `environments` folder contains environment configuration files. The main configuration files like `angular.json`, `tsconfig.json`, and `package.json` are also included. Adjust the structure based on your specific needs and preferences.

Step 1: Define the Data Models

Create data models to represent categories, subcategories, and products. For simplicity, we'll use simple TypeScript classes:

```
// product.model.ts
export class Product {
  constructor(public id: number, public name: string, public description:
string) {}
}

// subcategory.model.ts
import { Product } from './product.model';

export class Subcategory {
  constructor(public id: number, public name: string, public products:
Product[] = []) {}
}

// category.model.ts
import { Subcategory } from './subcategory.model';

export class Category {
  constructor(public id: number, public name: string, public subcategories:
Subcategory[] = []) {}
}
```

Step 2: Create a Mock Data Service

Create a service to provide mock data for our product catalog:

```
// product.service.ts
import { Injectable } from '@angular/core';
import { Category } from './category.model';
```

```
@Injectable({
  providedIn: 'root',
})
export class ProductService {
  private categories: Category[] = [
    new Category(1, 'Electronics', [
      new Subcategory(11, 'Smartphones', [
        new Product(111, 'iPhone 13', 'The latest iPhone with powerful
features.'),
        new Product(112, 'Samsung Galaxy S21', 'A flagship Android
smartphone.'),
      ]),
      new Subcategory(12, 'Laptops', [
        new Product(121, 'MacBook Pro', 'High-performance laptop for
professionals.'),
        new Product(122, 'Dell XPS 13', 'Sleek and powerful ultrabook.'),
      ]),
    ]),
    new Category(2, 'Furniture', [
      new Subcategory(21, 'Living Room', [
        new Product(211, 'Sofa Set', 'Comfortable and stylish sofa set.'),
        new Product(212, 'Coffee Table', 'Modern coffee table for your living
room.'),
      ]),
      // Add more furniture subcategories and products as needed
    ]),
    // Add more categories as needed
  ];

  getCategories(): Category[] {
    return this.categories;
  }
}
```

Step 3: Create Components for Product Catalog

Create components to display the product catalog:

```
ng generate component categories
ng generate component subcategories
ng generate component products
```

Step 4: Update Routing Configuration

Update the routing configuration to handle the product catalog hierarchy:

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
```

```
import { RouterModule, Routes } from '@angular/router';
import { CategoriesComponent } from './categories/categories.component';
import { SubcategoriesComponent } from
'./subcategories/subcategories.component';
import { ProductsComponent } from './products/products.component';

const routes: Routes = [
  { path: 'product-catalog', component: CategoriesComponent, children: [
    { path: ':categoryId', component: SubcategoriesComponent, children: [
      { path: ':subcategoryId', component: ProductsComponent },
    ]},
  ]},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Step 5: Implement Components

Implement the components to display categories, subcategories, and products:

categories.component.ts

```
// categories.component.ts
import { Component } from '@angular/core';
import { ProductService } from '../product.service';

@Component({
  selector: 'app-categories',
  template: `
    <h2>Categories</h2>
    <ul>
      <li *ngFor="let category of categories">
        <a [routerLink]="['/', 'product-catalog', category.id]">{{
category.name }}</a>
      </li>
    </ul>
    <router-outlet></router-outlet>
  `,
})
export class CategoriesComponent {
  categories = this.productService.getCategories();

  constructor(private productService: ProductService) {}
}
```

subcategories.component.ts

```
// subcategories.component.ts
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { ProductService } from '../product.service';

@Component({
  selector: 'app-subcategories',
  template: `
    <h3>Subcategories for Category {{ categoryId }}</h3>
    <ul>
      <li *ngFor="let subcategory of subcategories">
        <a [routerLink]="['.', subcategory.id]">{{ subcategory.name }}</a>
      </li>
    </ul>
    <router-outlet></router-outlet>
  `,
})
export class SubcategoriesComponent {
  categoryId: number;
  subcategories = [];

  constructor(private route: ActivatedRoute, private productService:
ProductService) {
    this.route.params.subscribe(params => {
      this.categoryId = +params['categoryId'];
      const category = this.productService.getCategories().find(c => c.id ===
this.categoryId);
      if (category) {
        this.subcategories = category.subcategories;
      }
    });
  }
}
```

products.component.ts

```
// products.component.ts
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { ProductService } from '../product.service';

@Component({
  selector: 'app-products',
  template: `
    <h4>Products for Subcategory {{ subcategoryId }}</h4>
    <ul>
      <li *ngFor="let product of products">
```

```
        {{ product.name }} - {{ product.description }}
      </li>
    </ul>
  `,
})
export class ProductsComponent {
  subcategoryId: number;
  products = [];

  constructor(private route: ActivatedRoute, private productService:
ProductService) {
    this.route.params.subscribe(params => {
      this.subcategoryId = +params['subcategoryId'];
      const category = this.productService.getCategories().find(c => c.id ===
+params['categoryId']);
      const subcategory = category?.subcategories.find(s => s.id ===
this.subcategoryId);
      if (subcategory) {
        this.products = subcategory.products;
      }
    });
  }
}
```

Step 6: Update App Component

Update the `app.component.html` file to include the router outlet:

```
<!-- app.component.html -->
<router-outlet></router-outlet>
```

Step 7: Run Your App

Run your Angular application:

```
ng serve
```

Navigate to `http://localhost:4200/product-catalog` and explore the product catalog with categories, subcategories, and product details.

This example demonstrates a more extensive use of Angular routing with nested routes, route parameters, and dynamic data loading based on the selected routes. Customize and expand this example based on your application's requirements.

Let's break down the routes and how they work in the provided Angular project.

Routes and Navigation:

1. **Default Route:**
   - **Path:** None (empty path)
   - **Component:** `CategoriesComponent`
   - **Usage:** The default route loads the `CategoriesComponent` when the application is accessed without a specific route.
2. **Product Catalog Route:**
   - **Path:** `/product-catalog`
   - **Component:** `CategoriesComponent`
   - **Usage:** Accessing this route displays the list of product categories in the `CategoriesComponent`.
3. **Category Route:**
   - **Path:** `/product-catalog/:categoryId`
   - **Component:** `SubcategoriesComponent`
   - **Usage:** Clicking on a category in `CategoriesComponent` navigates to this route, displaying the subcategories for the selected category in the `SubcategoriesComponent`.
4. **Subcategory Route:**
   - **Path:** `/product-catalog/:categoryId/:subcategoryId`
   - **Component:** `ProductsComponent`
   - **Usage:** Clicking on a subcategory in `SubcategoriesComponent` navigates to this route, displaying the products for the selected subcategory in the `ProductsComponent`.

How It Works:

1. **Categories Component (`CategoriesComponent`):**
   - Displays a list of categories.
   - Each category is a link that, when clicked, navigates to the corresponding category route.
2. **Subcategories Component (`SubcategoriesComponent`):**
   - Displays the subcategories for the selected category.
   - Each subcategory is a link that, when clicked, navigates to the corresponding subcategory route.
3. **Products Component (`ProductsComponent`):**
   - Displays the products for the selected subcategory.
   - Products are dynamically loaded based on the selected category and subcategory.

Navigation Flow:

1. Access the application at `http://localhost:4200/`.
    - The default route loads `CategoriesComponent`.
2. Click on a category in `CategoriesComponent`.
    - Navigates to `/product-catalog/:categoryId`.
    - `SubcategoriesComponent` loads with the subcategories for the selected category.
3. Click on a subcategory in `SubcategoriesComponent`.
    - Navigates to `/product-catalog/:categoryId/:subcategoryId`.
    - `ProductsComponent` loads with the products for the selected subcategory.

Example URLs:

- Category "Electronics": `http://localhost:4200/product-catalog/1`
- Subcategory "Smartphones" under "Electronics": `http://localhost:4200/product-catalog/1/11`
- Subcategory "Laptops" under "Electronics": `http://localhost:4200/product-catalog/1/12`

This navigation structure allows users to explore the product catalog, moving from categories to subcategories to individual product details. Adjustments and enhancements can be made based on your specific requirements.