

## FRONT-END SCREENSHOT

### Home.component.html

The screenshot shows the Visual Studio Code interface with the file `home.component.html` open. The code displays a template for a home page component. It includes a welcome message, a search bar with a placeholder image, and a carousel section with three items. The code uses Angular syntax (HTML, CSS, and TypeScript) and imports components like `InputGroup`, `FormControl`, and `CarouselModule`.

```
<br>
<div class="container">
  <div class="row">
    <div class="col-6 mx-auto">
      <h4>Welcome to Medicare</h4><br>
      <h4>What are you looking for?</h4><br>
      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text"></span>
        </div>
        <input type="text" class="form-control" placeholder="Search medicine, health products and more.." [(ngModel)]="name" (keydown.enter)="onSearch(name)">
      </div>
    </div>
  </div>
<div id="img" class="carousel slide" data-ride="carousel">
  <ul class="carousel-indicators">
    <li data-target="#img" data-slide-to="0" class="active"></li>
    <li data-target="#img" data-slide-to="1"></li>
    <li data-target="#img" data-slide-to="2"></li>
    <li data-target="#img" data-slide-to="3"></li>
  </ul>
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      <img alt="Image 2 of the carousel" data-slide-to="1" data-target="#img" data-slide-to="2"/>
    </div>
    <div class="carousel-item">
      <img alt="Image 3 of the carousel" data-slide-to="3" data-target="#img" data-slide-to="4"/>
    </div>
  </div>
</div>
```

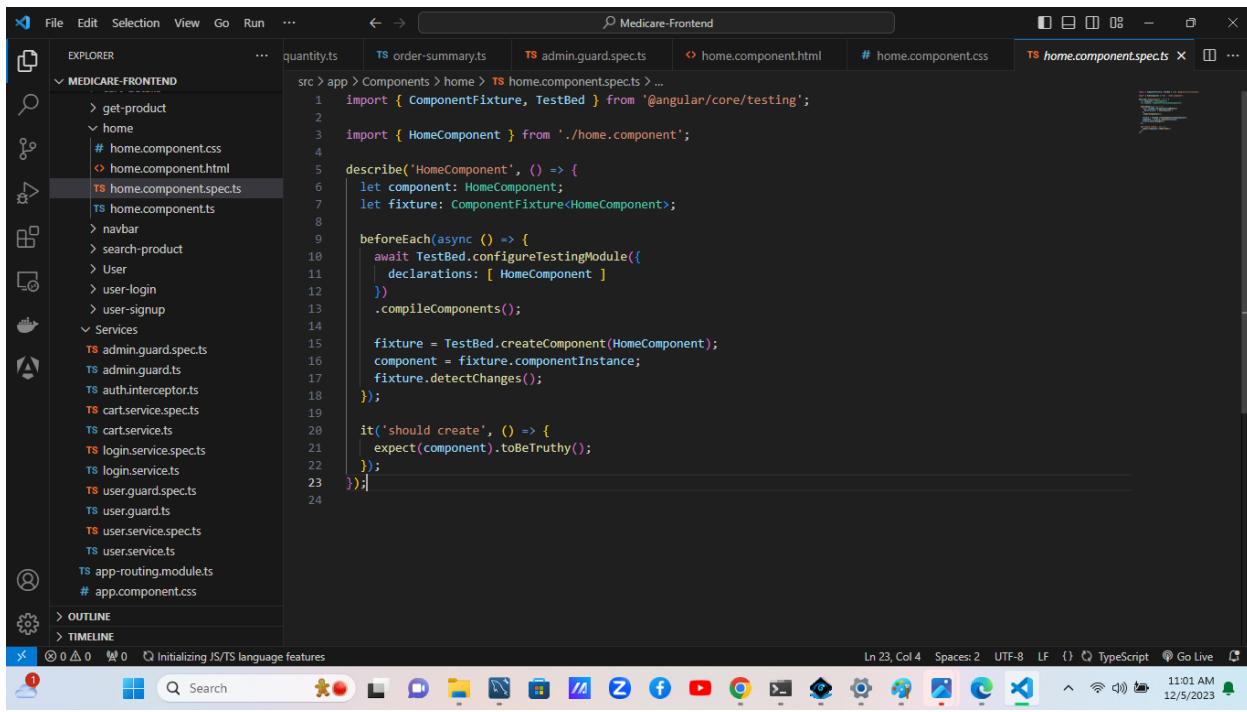
### Home.component.css

The screenshot shows the Visual Studio Code interface with the file `home.component.css` open. The code contains CSS styles for the carousel images, specifically setting the width and height of the `img` elements within the `carousel-inner` and `input-group-text` classes.

```
.carousel-inner img {
  width: 100%;
  height: 250px;
}

.input-group-text img {
  width: 20px;
  height: 20px;
}
```

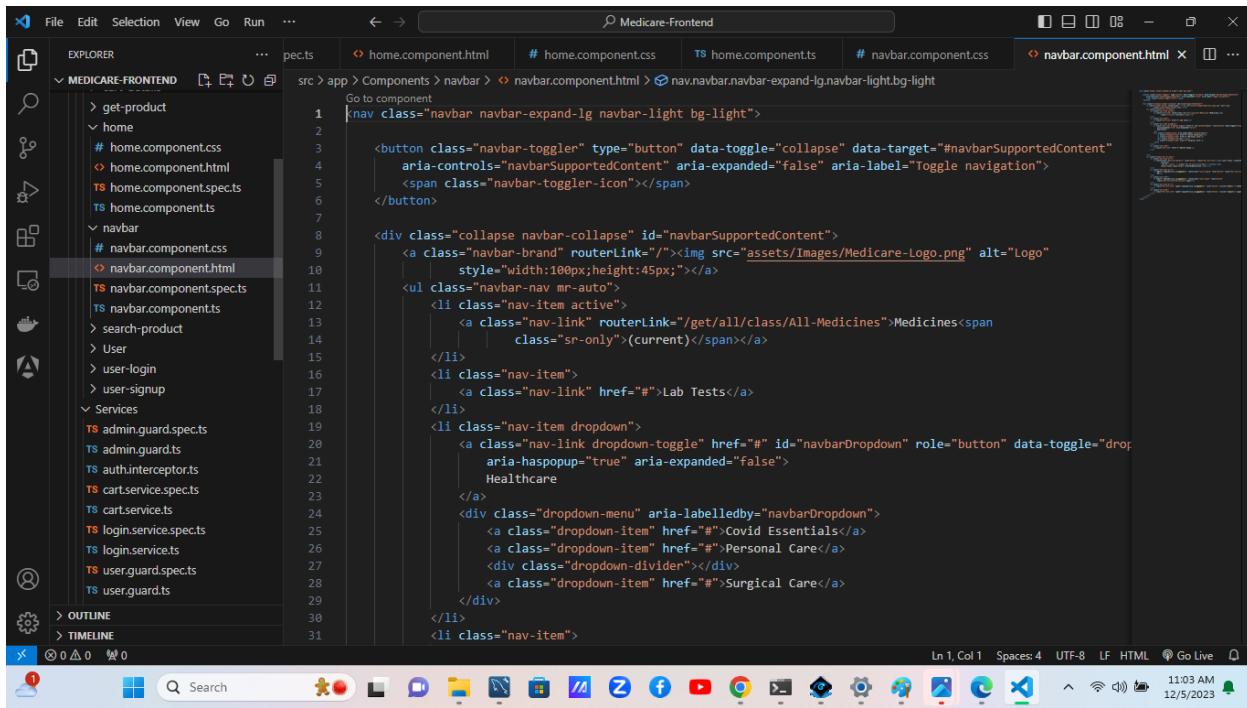
## Home.component.spec.ts



The screenshot shows the VS Code interface with the file `home.component.spec.ts` open in the editor. The code is a unit test for the `HomeComponent`. It imports `ComponentFixture`, `TestBed`, and `HomeComponent`. It uses `beforeEach` to set up a testing module with declarations of `HomeComponent`. It then creates a component instance, detects changes, and asserts that the component is truthy.

```
src > app > Components > home > TS home.component.spec.ts > ...
1 import { ComponentFixture, TestBed } from '@angular/core/testing';
2
3 import { HomeComponent } from './home.component';
4
5 describe('HomeComponent', () => {
6   let component: HomeComponent;
7   let fixture: ComponentFixture<HomeComponent>;
8
9   beforeEach(async () => {
10     await TestBed.configureTestingModule({
11       declarations: [ HomeComponent ]
12     })
13     .compileComponents();
14
15     fixture = TestBed.createComponent(HomeComponent);
16     component = fixture.componentInstance;
17     fixture.detectChanges();
18   });
19
20   it('should create', () => {
21     expect(component).toBeTruthy();
22   });
23 });
24
```

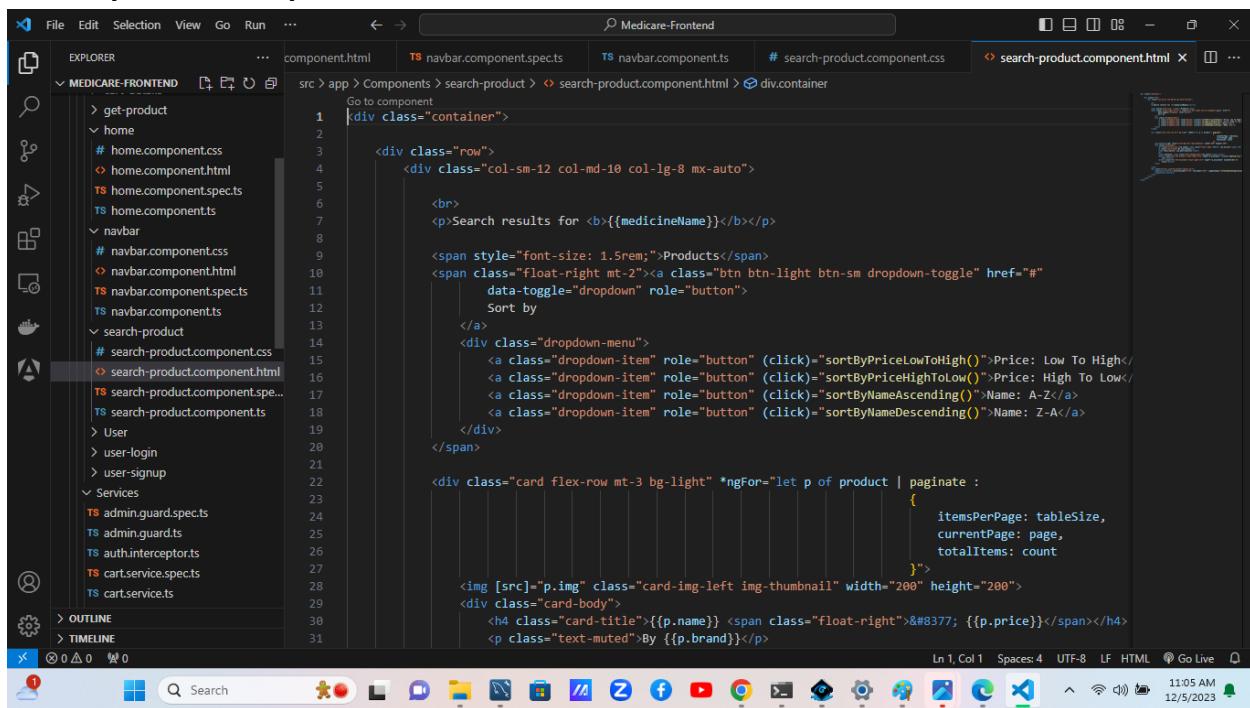
## Navbar.component.html



The screenshot shows the VS Code interface with the file `navbar.component.html` open in the editor. The code defines an `nav` bar with a brand logo, a toggle button, and a collapse menu. The collapse menu includes links for Medicines, Lab Tests, and a dropdown for Healthcare with sub-links for Covid Essentials, Personal Care, and Surgical Care.

```
src > app > Components > navbar > TS navbar.component.html > nav.navbar.navbar-expand-lg.navbar-light.bg-light
1 <nav class="navbar navbar-expand-lg navbar-light bg-light">
2
3   <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
4     aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
5     <span class="navbar-toggler-icon"></span>
6   </button>
7
8   <div class="collapse navbar-collapse" id="navbarSupportedContent">
9     <a class="navbar-brand" routerLink="/"></a>
10    <ul class="navbar-nav mr-auto">
11      <li class="nav-item active">
12        <a class="nav-link" routerLink="/get/all/class/All-Medicines">Medicines<span
13          |           class="sr-only">(current)</span></a>
14      </li>
15      <li class="nav-item">
16        <a class="nav-link" href="#">Lab Tests</a>
17      </li>
18      <li class="nav-item dropdown">
19        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown"
20          aria-haspopup="true" aria-expanded="false">
21          Healthcare
22        </a>
23        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
24          <a class="dropdown-item" href="#">Covid Essentials</a>
25          <a class="dropdown-item" href="#">Personal Care</a>
26          <div class="dropdown-divider"></div>
27          <a class="dropdown-item" href="#">Surgical Care</a>
28        </div>
29      </li>
30      <li class="nav-item">
31        <a class="nav-link" href="#">About Us</a>
32      </li>
33    </ul>
34  </div>
35</nav>
```

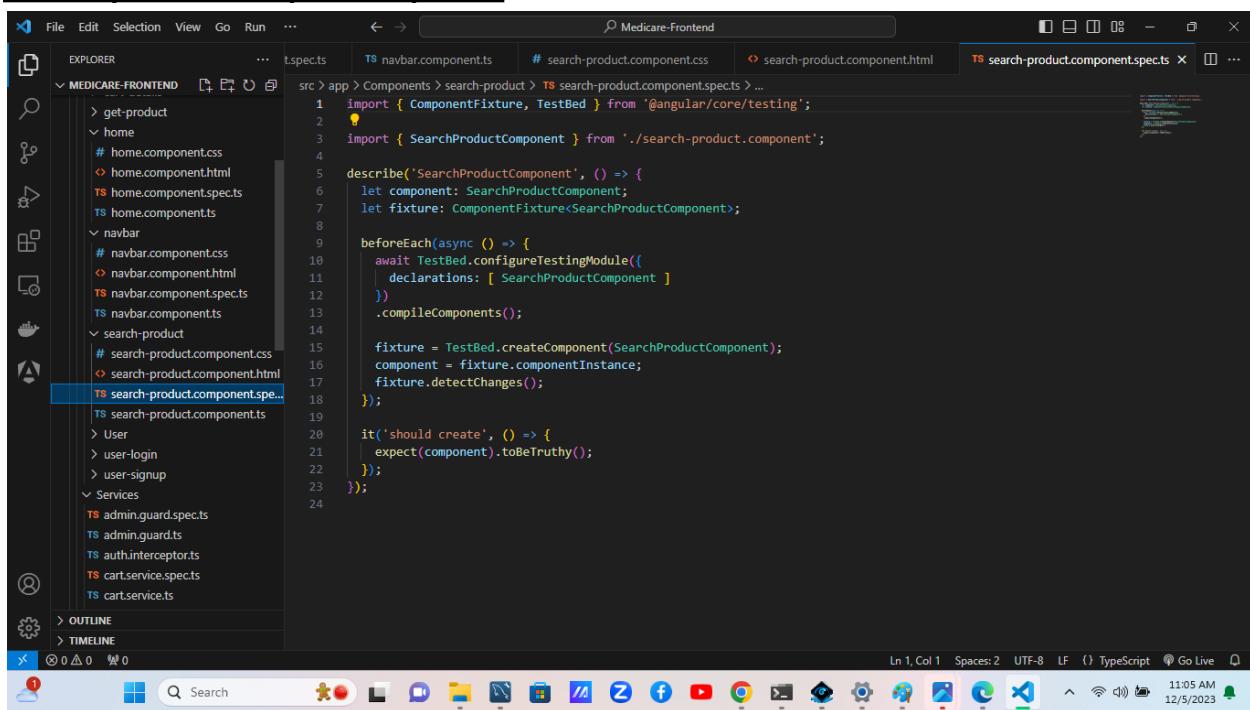
## Search-product.component.html



A screenshot of Visual Studio Code showing the `search-product.component.html` file. The code is an Angular component template. It includes a container div, a row div, a search results p tag, a dropdown menu for sorting, and a card component displaying product details. The code uses CSS classes like `container`, `row`, and `col-sm-12`, and Angular directives like `ngFor` and `ngIf`. The status bar at the bottom shows the file is 1105 AM on 12/5/2023.

```
<div class="container">
  <div class="row">
    <div class="col-sm-12 col-md-10 col-lg-8 mx-auto">
      <br>
      <p>Search results for <b>{{medicineName}}</b></p>
      <span style="font-size: 1.5rem;">Products</span>
      <span class="float-right mt-2"><a class="btn btn-light btn-sm dropdown-toggle" href="#" data-toggle="dropdown" role="button">Sort by</a>
        <div class="dropdown-menu">
          <a class="dropdown-item" role="button" (click)="sortByPriceLowToHigh()">Price: Low To High</a>
          <a class="dropdown-item" role="button" (click)="sortByPriceHighToLow()">Price: High To Low</a>
          <a class="dropdown-item" role="button" (click)="sortByNameAscending()">Name: A-Z</a>
          <a class="dropdown-item" role="button" (click)="sortByNameDescending()">Name: Z-A</a>
        </div>
      </span>
      <div class="card flex-row mt-3 bg-light" *ngFor="let p of product | paginate : { itemsPerPage: pageSize, currentPage: page, totalItems: count }">
        <img [src]="p.img" class="card-img-left img-thumbnail" width="200" height="200">
        <div class="card-body">
          <h4 class="card-title">{{p.name}} <span class="float-right" style="color: #800000;">{{p.price}}
```

## Search-product.component.spec.ts



A screenshot of Visual Studio Code showing the `search-product.component.spec.ts` file. This is a unit test for the `SearchProductComponent`. It imports `ComponentFixture` and `TestBed` from Angular testing utilities. The test describes the component and sets up a fixture. It then performs an asynchronous beforeEach, creates a component instance, and checks if it was created successfully. The status bar at the bottom shows the file is 1105 AM on 12/5/2023.

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { SearchProductComponent } from './search-product.component';

describe('SearchProductComponent', () => {
  let component: SearchProductComponent;
  let fixture: ComponentFixture<SearchProductComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ SearchProductComponent ]
    })
    .compileComponents();
  });

  fixture = TestBed.createComponent(SearchProductComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

## User-Login.html.component

The screenshot shows the VS Code interface with the file `user-login.component.html` open. The code is an Angular template for a user login form. It includes a container div, a row div, and a col-md-8 div. Inside, there's a card with a title "Welcome User" and a sign-in message. A form group contains fields for username and password, both with required attributes and ngModel. Error messages are shown for invalid inputs. A submit button is at the bottom.

```
<div class="container">
  <div class="row">
    <div class="col-lg-4 col-md-8 mx-auto">
      <div class="card mt-5 bg-info">
        <div class="card-body">
          <div class="text-center">
            <h4>Welcome User</h4>
            <p>Sign in to continue.</p>
          </div>
          <br>
          <form #login="ngForm" (ngSubmit)="onSubmit()">
            <div class="form-group">
              <label for="email">Username:</label> <input type="text" name="username" #username="ngModel" class="form-control" placeholder="Enter Username" id="email" [(ngModel)]="credentials.username" required ngModel>
              <span *ngIf="username.invalid && username.touched" style="color: red">Please enter username.</span>
            </div>
            <div class="form-group">
              <label for="pwd">Password:</label> <input type="password" name="password" #password="ngModel" minlength="6" class="form-control" placeholder="Enter password" id="pwd" [(ngModel)]="credentials.password" required ngModel>
              <span *ngIf="password.invalid && password.touched" style="color: red">Please enter password.</span>
            </div>
            <button type="submit" class="btn btn-danger btn-block" [disabled]="login.invalid">Login</button>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

## User.component.spec.ts

The screenshot shows the VS Code interface with the file `user-login.component.spec.ts` open. This is a unit test for the `UserLoginComponent`. It imports `ComponentFixture`, `TestBed`, and the component under test. It sets up a testing module, compiles the component, and creates a fixture. The test then checks if the component was created successfully.

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { UserLoginComponent } from './user-login.component';

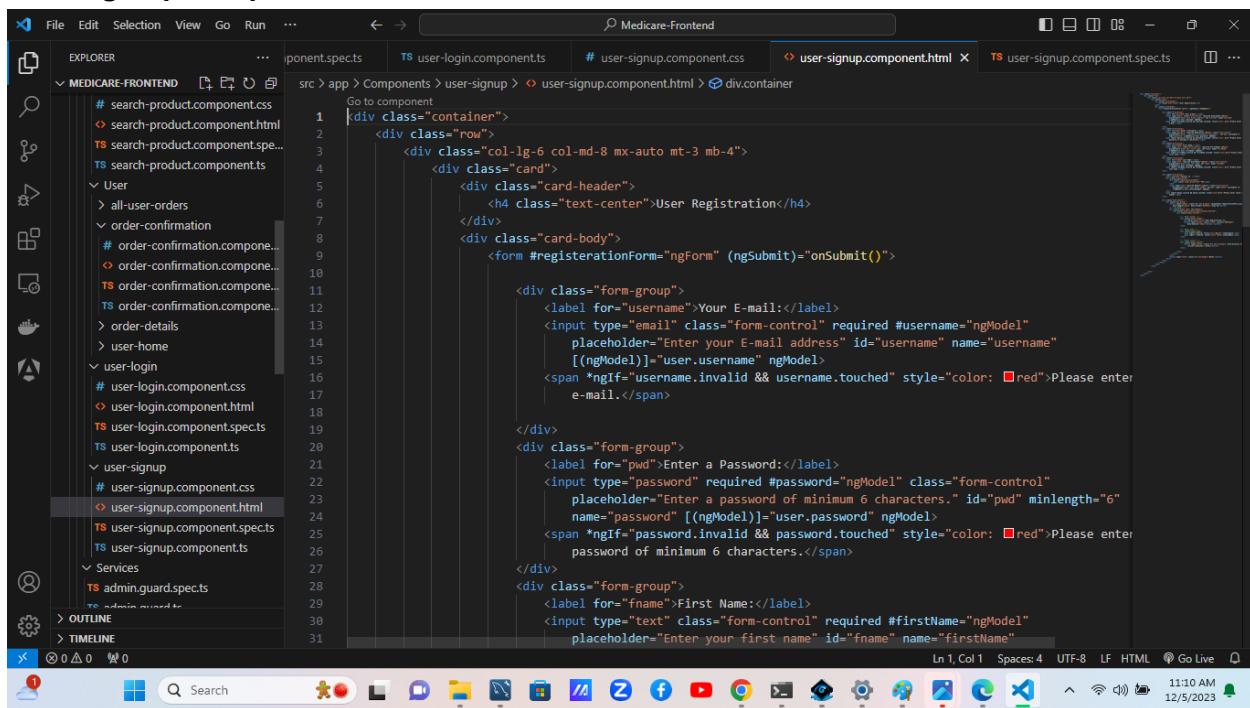
describe('UserLoginComponent', () => {
  let component: UserLoginComponent;
  let fixture: ComponentFixture<UserLoginComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ UserLoginComponent ]
    })
    .compileComponents();
  });

  fixture = TestBed.createComponent(UserLoginComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

## User sign up.component.html

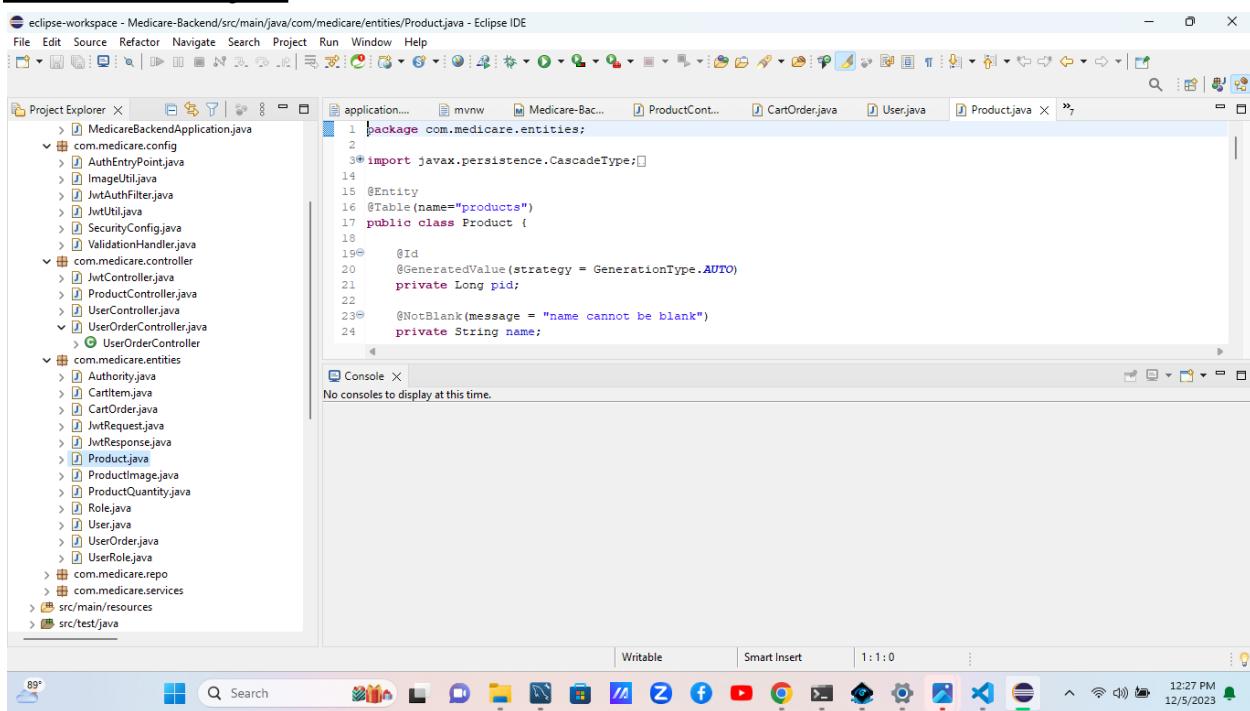


The screenshot shows a code editor window for a component named "user-signup.component.html". The code is written in HTML and includes Angular-specific directives like "ngForm" and "ngSubmit". It features three input fields: "username", "password", and "firstName". Each field has a corresponding label and a placeholder. Validation messages are displayed using the "ngIf" directive. The code is part of a larger project structure, with other components and services visible in the sidebar.

```
<div class="form-group">
  <label for="username">Your E-mail:</label>
  <input type="email" class="form-control" required #username="ngModel" 
    placeholder="Enter your E-mail address" id="username" name="username"
    [(ngModel)]="user.username" ngModel>
  <span *ngIf="username.invalid && username.touched" style="color: red">Please enter
    e-mail.</span>
</div>
<div class="form-group">
  <label for="pwd">Enter a Password:</label>
  <input type="password" required #password="ngModel" class="form-control" 
    placeholder="Enter a password of minimum 6 characters." id="pwd" minlength="6"
    name="password" [(ngModel)]="user.password" ngModel>
  <span *ngIf="password.invalid && password.touched" style="color: red">Please enter
    password of minimum 6 characters.</span>
</div>
<div class="form-group">
  <label for="fname">First Name:</label>
  <input type="text" class="form-control" required #firstName="ngModel" 
    placeholder="Enter your first name" id="fname" name="firstName">
</div>
```

## BACK-END SCREENSHOT

### Product.service.java



The screenshot shows the Eclipse IDE interface with a Java file named "Product.java" open in the editor. The code defines a class "Product" with an ID and a name. The "name" field is annotated with "@NotNull" and a validation message. The "Product" class is part of a package "com.medicare.entities". The Project Explorer on the left shows various Java files across different packages like "com.medicare.config", "com.medicare.controller", and "com.medicare.repo".

```
1 package com.medicare.entities;
2
3 import javax.persistence.CascadeType;
4
5 @Entity
6 @Table(name="products")
7 public class Product {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private Long pid;
12
13    @NotNull(message = "name cannot be blank")
14    private String name;
15}
```

## UserDetailsService.java

The screenshot shows the Eclipse IDE interface with the UserDetailsService.java file open in the editor. The code implements the UserDetailsService interface and uses autowiring to inject a UserRepository. It overrides the loadUserByUsername method to return a User object from the repository. The Project Explorer on the left shows the project structure, and the Outline view on the right shows the class hierarchy.

```
1 package com.medicare.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Service
6 public class UserDetailsService implements UserDetailsService{
7
8     @Autowired
9     private UserRepository userRepo;
10
11     @Override
12     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
13         User user = this.userRepo.findByUsername(username);
14         if(user == null) {
15             throw new UsernameNotFoundException("User not found");
16         }
17         return user;
18     }
19 }
```

## UserOrderservice.java

The screenshot shows the Eclipse IDE interface with the UserOrderservice.java file open in the editor. The code implements the UserOrderService interface and uses autowiring to inject OrderRepo and ProductQuantityRepo. It overrides the saveOrder method to save a UserOrder. The Project Explorer on the left shows the project structure, and the Outline view on the right shows the class hierarchy.

```
1 package com.medicare.services;
2
3 import java.util.List;
4
5 @Service
6 public class UserOrderService {
7
8     @Autowired
9     private OrderRepo orderRepo;
10
11     @Autowired
12     private ProductQuantityRepo productQuantityRepo;
13
14     public UserOrder saveOrder(UserOrder userOrder) {
15         return orderRepo.save(userOrder);
16     }
17 }
```

## UserService.java

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a tree view of files and folders.
- Code Editor:** Displays the `UserService.java` file content:1 package com.medicare.services;
2
3 import java.util.Set;
4
5 @Service
6 public class UserService {
7
8 @Autowired
9 private UserRepo userRepo;
10
11 @Autowired
12 private RoleRepo roleRepo;
13
14 @Autowired
15 private BCryptPasswordEncoder passwordEncoder;
16
17 public void createUser(User user) {
18 user.setPassword(passwordEncoder.encode(user.getPassword()));
19 userRepo.save(user);
20 }
21
22 public void updateUser(User user) {
23 userRepo.save(user);
24 }
25
26 public void deleteUser(User user) {
27 userRepo.delete(user);
28 }
29
30 public User getUser(String username) {
31 return userRepo.findByUsername(username);
32 }
33 }
- Outline View:** Shows the class structure and methods.
- Console:** Displays "No consoles to display at this time."
- Bottom Bar:** Includes tabs for Writable, Smart Insert, and 1:1:0, along with a toolbar and system status.

## PomXML

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a tree view of files and folders.
- Code Editor:** Displays the `pom.xml` file content:<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.7.7</version>
 <relativePath/> <!-- lookup parent from repository -->
 </parent>
 <groupId>com.medicare</groupId>
 <artifactId>Medicare-Backend</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>Medicare-Backend</name>
 <description>Capstone project Medicare-backend</description>
 <properties>
 <java.version>17</java.version>
 </properties>
 <dependencies>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-data-jpa</artifactId>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-security</artifactId>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-validation</artifactId>
 </dependency>
 </dependencies>
- Bottom Bar:** Includes tabs for Overview, Dependencies, Dependency Hierarchy, Effective POM, and pom.xml, along with a toolbar and system status.

eclipse-workspace - Medicare-Backend/pom.xml - Eclipse IDE

```
File Edit Source Navigate Search Project Run Design Window Help
51     <groupId>io.jsonwebtoken</groupId>
52     <artifactId>jjwt</artifactId>
53     <version>0.9.0</version>
54   </dependency>
55   <!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->
56   <dependency>
57     <groupId>javax.xml.bind</groupId>
58     <artifactId>jaxb-api</artifactId>
59   </dependency>
60
61   <dependency>
62     <groupId>org.springframework.boot</groupId>
63     <artifactId>spring-boot-starter-test</artifactId>
64     <scope>test</scope>
65   </dependency>
66   <dependency>
67     <groupId>org.springframework.security</groupId>
68     <artifactId>spring-security-test</artifactId>
69     <scope>test</scope>
70   </dependency>
71 </dependencies>
72
73 <build>
74   <plugins>
75     <plugin>
76       <groupId>org.springframework.boot</groupId>
77       <artifactId>spring-boot-maven-plugin</artifactId>
78     </plugin>
79   </plugins>
80 </build>
81
82 </project>
83
```

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

Writable Insert 1:1:0

89° Search

12:25 PM 12/5/2023

## CartOrder.java

eclipse-workspace - Medicare-Backend/src/main/java/com/medicare/entities/CartOrder.java - Eclipse IDE

```
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Application mvnw Medicare-Bac... ProductCont... CartOrderjava Userjava
MedicareBackendApplication.java
com.medicare.config
AuthEntryPoint.java
ImageUtil.java
JwtAuthFilter.java
JwtUtil.java
SecurityConfig.java
ValidationHandler.java
com.medicare.controller
JwtController.java
ProductController.java
UserController.java
UserOrderController.java
UserOrderController
com.medicare.entities
Authority.java
CartItem.java
CartOrder.java
JwtRequest.java
JwtResponse.java
Product.java
ProductImage.java
ProductQuantity.java
Role.java
User.java
UserOrder.java
UserRole.java
com.medicare.repo
com.medicare.services
src/main/resources
src/test/java
```

private CartOrder(String document, String address, String district, String pincode, String state, String contact, Double paidAmount, String paymentMode, Set<CartItem> cartItem) {  
super();  
this.username = username;  
this.firstName = firstName;  
this.lastName = lastName;  
this.address = address;  
this.district = district;  
this.pinCode = pinCode;  
this.state = state;  
this.contact = contact;  
this.paidAmount = paidAmount;  
this.paymentMode = paymentMode;  
this.cartItem = cartItem;  
}

Console

No consoles to display at this time.

Writable Smart Insert 1:1:0

89° Search

12:27 PM 12/5/2023