Source Code for java capstone Project ---

Developer name – Sachin Kale

1) Code for frontend Stack -- (Html,CSS,typescript,Angular)

app.component.html –

```
<app-navbar></app-navbar>
<router-outlet></router-outlet>
<footer>
    <div class="fixed-bottom">
        <small class="bg-light">&#169; 2012-2023- Medicare. Designed &
Developed by Sachin Kale.</small>
    </div>
</footer>
```

app.component.spec.ts –

```
import { TestBed } from '@angular/core/testing'; import {
RouterTestingModule } from '@angular/router/testing'; import {
AppComponent } from './app.component';
 describe('AppComponent', () => {
beforeEach(async () => {     await
TestBed.configureTestingModule({
imports: [
        RouterTestingModule
      ],
declarations: [

        AppComponent
      ],
    }).compileComponents();
  });    it('should create the app', () => {     const fixture =
TestBed.createComponent(AppComponent);     const app =
fixture.componentInstance;     expect(app).toBeTruthy();
```

```
  });    it(`should have as title 'Medicare'`, () => {     const fixture
= TestBed.createComponent(AppComponent);     const app =
fixture.componentInstance;     expect(app.title).toEqual('Medicare');
  });    it('should render title', () => {     const fixture =
TestBed.createComponent(AppComponent);     fixture.detectChanges();
const compiled = fixture.nativeElement as HTMLElement;
expect(compiled.querySelector('.content
span')?.textContent).toContain('Medicare app is running!');
  });
});
```

app.component.ts ---

```
import { Component } from '@angular/core';

@Component({   selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
}) export class
AppComponent {   title =
'Medicare';
}
```

app.module.ts ---

```
import { NgModule } from '@angular/core'; import {
FormsModule } from '@angular/forms'; import {
BrowserModule } from '@angular/platform-browser';
 import { AppRoutingModule } from './app-routing.module'; import {
AppComponent } from './app.component'; import { HttpClientModule } from
'@angular/common/http'; import { NgxPaginationModule } from 'ngx-
pagination'; import { authInterceptorProviders } from
'./Services/auth.interceptor'; import { UserLoginComponent } from
'./Components/user-login/userlogin.component'; import { AdminLoginComponent
} from './Components/admin-login/adminlogin.component'; import {
AdminDashboardComponent } from './Components/Admin/admindashboard/admin-
dashboard.component'; import { UserHomeComponent } from
'./Components/User/user-home/userhome.component'; import { NavbarComponent }
from './Components/navbar/navbar.component'; import { UserSignupComponent }
from './Components/user-signup/usersignup.component'; import {
AddProductComponent } from './Components/Admin/add-
product/addproduct.component'; import { ShowAllProductsComponent } from
'./Components/Admin/show-allproducts/show-all-products.component'; import {
UpdateProductComponent } from './Components/Admin/updateproduct/update-
product.component'; import { HomeComponent } from
'./Components/home/home.component'; import { SearchProductComponent } from
'./Components/search-product/searchproduct.component'; import {
GetProductComponent } from './Components/get-product/getproduct.component';
import { CartDetailsComponent } from './Components/cart-
details/cartdetails.component'; import { OrderDetailsComponent } from
'./Components/User/order-details/orderdetails.component'; import {
OrderConfirmationComponent } from
'./Components/User/orderconfirmation/order-confirmation.component';
import { AllOrdersComponent } from './Components/Admin/all-
orders/allorders.component'; import { AllUserOrdersComponent } from
'./Components/User/all-user-orders/alluser-orders.component';
```

```
@NgModule({
declarations: [
AppComponent,
    UserLoginComponent,
    AdminLoginComponent,
    AdminDashboardComponent,
    UserHomeComponent,
    NavbarComponent,
    UserSignupComponent,
    AddProductComponent,
    ShowAllProductsComponent,
    UpdateProductComponent,
HomeComponent,
    SearchProductComponent,
    GetProductComponent,
    CartDetailsComponent,
    OrderDetailsComponent,
    OrderConfirmationComponent,
    AllOrdersComponent,
    AllUserOrdersComponent,
  ],
  imports: [
BrowserModule,
    AppRoutingModule,
FormsModule,
    HttpClientModule,
    NgxPaginationModule
  ],   providers:
[authInterceptorProviders],   bootstrap:
[AppComponent]
}) export class AppModule
{ }
```

app-routing.module.ts ---

```
import { NgModule } from '@angular/core'; import { RouterModule,
Routes } from '@angular/router'; import { AdminLoginComponent } from
'./Components/admin-login/adminlogin.component';
```

```
import { AddProductComponent } from './Components/Admin/add-
product/addproduct.component'; import { AdminDashboardComponent } from
'./Components/Admin/admindashboard/admin-dashboard.component'; import {
AllOrdersComponent } from './Components/Admin/all-
orders/allorders.component'; import { ShowAllProductsComponent } from
'./Components/Admin/show-allproducts/show-all-products.component'; import {
UpdateProductComponent } from './Components/Admin/updateproduct/update-
product.component'; import { CartDetailsComponent } from './Components/cart-
details/cartdetails.component'; import { GetProductComponent } from
'./Components/get-product/getproduct.component'; import { HomeComponent }
from './Components/home/home.component'; import { SearchProductComponent }
from './Components/search-product/searchproduct.component'; import {
UserLoginComponent } from './Components/user-login/userlogin.component';
import { UserSignupComponent } from './Components/user-
signup/usersignup.component'; import { AllUserOrdersComponent } from
'./Components/User/all-user-orders/alluser-orders.component'; import {
OrderConfirmationComponent } from './Components/User/orderconfirmation/order-
confirmation.component'; import { OrderDetailsComponent } from
'./Components/User/order-details/orderdetails.component'; import {
UserHomeComponent } from './Components/User/user-home/userhome.component';
import { AdminGuard } from './Services/admin.guard'; import { UserGuard }
from './Services/user.guard';
 const routes: Routes =
[
  { path: 'user/login', component: UserLoginComponent, pathMatch: 'full',
title: 'User Login' },
  { path: 'admin/login', component: AdminLoginComponent, pathMatch: 'full',
title: 'Admin Login' },
  { path: '', component: HomeComponent, pathMatch: 'full', title: 'Medicare' },
  { path: 'user-home', component: UserHomeComponent, canActivate:
[UserGuard], pathMatch: 'full', title: 'Home' },
  { path: 'admin-dashboard', component: AdminDashboardComponent, canActivate:
[AdminGuard], pathMatch: 'full', title: 'Admin Dashboard' },
  { path: 'user/signup', component: UserSignupComponent, pathMatch: 'full',
title: 'User Registration' },
  { path: 'admin/add-medicine', component: AddProductComponent, canActivate:
[AdminGuard], pathMatch: 'full', title: 'Add Medicine' },
```

```
  { path: 'admin/get/all/medicines', component: ShowAllProductsComponent,
canActivate: [AdminGuard], pathMatch: 'full', title: 'All Medicines' },
{ path: 'admin/update/medicine/:pid', component: UpdateProductComponent,
canActivate: [AdminGuard], pathMatch: 'full', title: 'Update Medicine' },
{ path: 'user/search/product/:name', component: SearchProductComponent,
pathMatch: 'full', title: 'Search results' },
  { path: 'show/product/class/:category', component: GetProductComponent,
pathMatch: 'full', title: 'All Products' },
  { path: 'get/all/class/:category', component: GetProductComponent,
pathMatch: 'full', title: 'All Products' },
  { path: 'get/cart/details', component: CartDetailsComponent, pathMatch:
'full', title: 'Cart Details' },
  { path: 'user/create/order', component: OrderDetailsComponent, canActivate:
[UserGuard], pathMatch: 'full', title: 'Order Details' },
  { path: 'order-confirmation/invoice/:oid', component:
OrderConfirmationComponent, canActivate: [UserGuard], pathMatch: 'full',
title: 'Order Confirmation' },
  { path: 'admin/all/user-orders', component: AllOrdersComponent, canActivate:
[AdminGuard], pathMatch: 'full', title: 'All Orders' },
  { path: 'order/details/:oid', component: OrderConfirmationComponent,
canActivate: [AdminGuard], pathMatch: 'full', title: 'Order Details' },   {
path: 'user/get/all-orders/:username', component: AllUserOrdersComponent,
canActivate: [UserGuard], pathMatch: 'full', title: 'Orders Placed' },
];

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

cart-item.ts ---

```
import { Product } from "./product";
 export class CartItem
{     pid!: number;
name!: string;
brand!: string;
price!: number;
img!: any;

    quantity!: number;
     constructor(product: Product) {         this.pid = product.pid;
this.name = product.name;         this.brand = product.brand;
```

```
this.price = product.price;          this.img = product.img;
this.quantity = 1;
    }
}
```

Credential.ts –

```
export class Credentials {     username!: string;
password!: string;
}
```

order-details.ts –

```
import { OrderItem } from "./order-item";
 export class OrderDetails {
oid!: number;     username!:
string;     firstName!: string;
lastName!: string;
address!: string;
district!: string;
pinCode!: number;     state!:
string;     contact!: string;
paidAmount!: number;
paymentMode!: string;
cartItem: OrderItem[] = [];

}
```

Order-item.ts –

```
export class OrderItem {     pid!: number;
quantity!: number;
}
```

Order-summary.ts –

```
import { ProductQuantity } from "./product-quantity";
 export class OrderSummary {
oid!: number;     username!: string;
firstName!: string;     lastName!:
string;     address!: string;
district!: string;     pinCode!:
number;     state!: string;
contact!: string;     paidAmount!:
number;     paymentMode!: string;
status!: string;     date!: string;
products: ProductQuantity[] = [];
}
```

product-quantity.ts –

```
import { Product } from "./product";
 export class ProductQuantity
{     pqid!: number;

    product!: Product;     quantity!: number;
}
```

product.ts –

```
export class Product {     pid!: number;
available!: boolean;     name!: string;     brand!:
string;     category!: string;     description!:
string;     salt!: string;     totalAvailable!:
number;     price!: number;     productImage: any;
img!: any;
}
```

user.ts –

```
export class User {     userId!: number;
username!: string;     password!: string;
firstName!: string;     lastName!: string;
contactNumber!: string;
}
```

index.html –

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
```

```
  <title>Medicare</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
</head>

<body>

  <app-root></app-root>


  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/a076d05399.js"
crossorigin="anonymous"></script>

</body>
</html>
```

main.ts –

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
 import { AppModule } from
'./app/app.module';


platformBrowserDynamic().bootstrapModule(AppModule)
```

```
  .catch(err => console.error(err));
```

package.json –

```
{
  "name": "medicare",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^15.0.0",
    "@angular/common": "^15.0.0",
    "@angular/compiler": "^15.0.0",
    "@angular/core": "^15.0.0",
    "@angular/forms": "^15.0.0",
    "@angular/platform-browser": "^15.0.0",
    "@angular/platform-browser-dynamic": "^15.0.0",
    "@angular/router": "^15.0.0",
    "ngx-pagination": "^6.0.3",
    "rxjs": "~7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.12.0"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^15.0.4",
    "@angular/cli": "~15.0.4",
    "@angular/compiler-cli": "^15.0.0",
    "@types/jasmine": "~4.3.0",
    "jasmine-core": "~4.5.0",
    "karma": "~6.4.0",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage": "~2.2.0",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.0.0",
    "typescript": "~4.8.2"
  }
}
```

angular.json –

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "Medicare": {
      "projectType": "application",
      "schematics": {},
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/medicare",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": [
              "zone.js"
            ],
            "tsConfig": "tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.css"
            ],
            "scripts": []
          },
          "configurations": {
            "production": {
```

```
            "budgets": [
              {
                "type": "initial",
                "maximumWarning": "500kb",
                "maximumError": "1mb"
              },
              {
                "type": "anyComponentStyle",
                "maximumWarning": "2kb",
                "maximumError": "4kb"
              }
            ],
            "outputHashing": "all"
          },
          "development": {
            "buildOptimizer": false,
            "optimization": false,
            "vendorChunk": true,
            "extractLicenses": false,
            "sourceMap": true,
            "namedChunks": true
          }
        },
        "defaultConfiguration": "production"
      },
      "serve": {
        "builder": "@angular-devkit/build-angular:dev-server",
        "configurations": {
          "production": {
            "browserTarget": "Medicare:build:production"
          },
          "development": {
            "browserTarget": "Medicare:build:development"
          }
        },
        "defaultConfiguration": "development"
      },
      "extract-i18n": {
        "builder": "@angular-devkit/build-angular:extract-i18n",
        "options": {
          "browserTarget": "Medicare:build"
        }
      },
      "test": {
        "builder": "@angular-devkit/build-angular:karma",
        "options": {
          "polyfills": [
            "zone.js",
```

```
              "zone.js/testing"
            ],
            "tsConfig": "tsconfig.spec.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/styles.css"
            ],
            "scripts": []
          }
        }
      }
    }
  },
  "cli": {
    "analytics": "8066b760-d79d-4439-97d7-a034bc01e88e"
  }
}
```

tsconfig.app.json –

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "outDir": "./out-tsc/app",
    "types": []
  },
  "files": [
    "src/main.ts"
  ],
  "include": [
    "src/**/*.d.ts"
  ]
}
```

tsconfig.spec.json ---

```
{
  "extends": "./tsconfig.json",
```

```
  "compilerOptions": {
    "outDir": "./out-tsc/spec",
    "types": [
      "jasmine"
    ]
  },
  "include": [
    "src/**/*.spec.ts",
    "src/**/*.d.ts"
  ]
}
```

2) Code for backend stack ---

MedicareBackendApplication.java –

```
package com.medicare;
 import org.springframework.boot.SpringApplication; import
org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication public class
MedicareBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(MedicareBackendApplication.class, args);
    }



}
```

Controller –

jwtcontroller.java –

```
package com.medicare.controller;
 import
java.security.Principal;
 import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.ResponseEntity; import
org.springframework.security.authentication.AuthenticationManager; import
org.springframework.security.authentication.BadCredentialsException; import
org.springframework.security.authentication.DisabledException; import
org.springframework.security.authentication.UsernamePasswordAuthenticationToke
n; import org.springframework.security.core.userdetails.UserDetails; import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.CrossOrigin; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestBody; import
org.springframework.web.bind.annotation.RestController;
 import com.medicare.config.JwtUtil; import
com.medicare.entities.JwtRequest; import
com.medicare.entities.JwtResponse; import
com.medicare.entities.User; import
com.medicare.services.UserDetailService;

@RestController
@CrossOrigin(origins = "*") public
class JwtController {

    @Autowired     private AuthenticationManager
authenticationManager;

    @Autowired
```

```
    private UserDetailService userDetailService;

    @Autowired     private
JwtUtil jwtUtil;

    //generate token
    @PostMapping("/generate-token")     public ResponseEntity<?>
generateToken(@RequestBody JwtRequest jwtRequest) throws Exception{
try {             authenticate(jwtRequest.getUsername(),
jwtRequest.getPassword());         }catch(UsernameNotFoundException e) {
            e.printStackTrace();             throw new
Exception("User does not exist or invalid credentials!");
        }
        // validated
        UserDetails userDetails =
this.userDetailService.loadUserByUsername(jwtRequest.getUsername());
String token = this.jwtUtil.generateToken(userDetails);         return
ResponseEntity.ok(new JwtResponse(token));
    }
    private void authenticate(String username, String password)
throws Exception {         try {
            this.authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(username, password));
        } catch (BadCredentialsException e) {             throw new
Exception("Invalid Credentials! "+e.getMessage());
        }catch(DisabledException e) {             throw new
Exception("User Disabled! "+e.getMessage());
        }
    }
    @GetMapping("/current-user")     public User
getCurrentUser(Principal principal) {         return
(User)this.userDetailService.loadUserByUsername(principal.getName());
    }
}
```

productcontroller.java –

```
package com.medicare.controller;
 import
java.io.IOException;
import java.util.List;
 import
javax.validation.Valid;
 import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
 import
com.fasterxml.jackson.core.JsonProcessingException; import
com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper; import
com.medicare.config.ImageUtil; import
com.medicare.entities.Product; import
com.medicare.entities.ProductImage; import
com.medicare.services.ProductService;

@RestController
@CrossOrigin(origins = "*")
public class ProductController {

    @Autowired     private ProductService
productService;

    @Autowired     private
ObjectMapper objectMapper;



    //add new product
    @PreAuthorize("hasAuthority('ADMIN')")
    @PostMapping("/add/product")     public ResponseEntity<?>
addNewProduct(@RequestParam("product") String product,


        @RequestParam("image")
```

```
MultipartFile file) throws IOException{

        ProductImage img = new ProductImage();
img.setName(file.getOriginalFilename());
img.setType(file.getContentType());

img.setImageData(ImageUtil.compressImage(file.getBytes()));
Product p = null;         try {             p =
objectMapper.readValue(product,Product.class);
            p.setProductImage(img);
} catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Invalid
Request");
        }
        Product saveProduct = this.productService.addProduct(p);
return ResponseEntity.ok(saveProduct);
    }

    //update existing product
    @PreAuthorize("hasAuthority('ADMIN')")
@PutMapping("/update/product/{id}")     public ResponseEntity<?>
updateProduct(@PathVariable("id") Long id,@Valid
@RequestBody Product product){
        Product updateProduct = this.productService.findProduct(id);
updateProduct.setName(product.getName());
updateProduct.setBrand(product.getBrand());
updateProduct.setCategory(product.getCategory());
updateProduct.setDescription(product.getDescription());
updateProduct.setSalt(product.getSalt());
updateProduct.setTotalAvailable(product.getTotalAvailable());
updateProduct.setPrice(product.getPrice());
this.productService.addProduct(updateProduct);         return
ResponseEntity.status(HttpStatus.CREATED).build();
    }

    //find product by id
    @GetMapping("get-product/{id}")
    public ResponseEntity<?> getProductById(@PathVariable("id") Long id){
        Product product = this.productService.findProduct(id);
ProductImage img =  new ProductImage();
img.setImageData(ImageUtil.decompressImage(product.getProductImage().g
etImageData()));         img.setImgId(product.getProductImage().getImgId());
img.setName(product.getProductImage().getName());
```

```
img.setType(product.getProductImage().getType());
product.setProductImage(img);         return
ResponseEntity.ok(product);
    }

    //find all products
    @GetMapping("/get/all-products")
    public ResponseEntity<?> getAllProducts(){
        List<Product> allProducts = this.productService.findAllProducts();
allProducts.forEach(product -> {
            ProductImage img =  new ProductImage();
img.setImageData(ImageUtil.decompressImage(product.getProductImage
().getImageData()));
img.setImgId(product.getProductImage().getImgId());
img.setName(product.getProductImage().getName());
img.setType(product.getProductImage().getType());
product.setProductImage(img);
        });         if(allProducts.isEmpty()) {             return
ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }else {             return
ResponseEntity.ok(allProducts);
        }
    }

    @GetMapping(value = {"/get/products/{name}"})     public
ResponseEntity<?> getProductByName(@PathVariable("name") String
name,@PathVariable("name") String salt){
        List<Product> products = this.productService.findByNameOrSalt(name,
salt);
        products.forEach(product -> {
            ProductImage img =  new ProductImage();
img.setImageData(ImageUtil.decompressImage(product.getProductImage
().getImageData()));
img.setImgId(product.getProductImage().getImgId());
img.setName(product.getProductImage().getName());
img.setType(product.getProductImage().getType());
product.setProductImage(img);
        });         if(products.isEmpty()) {             return
ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }else {             return
ResponseEntity.ok(products);
        }
    }
    @GetMapping("/get/products-by-category/{category}")
    public ResponseEntity<?> getProductsByCategory(@PathVariable("category")
String category){
```

```
        List<Product> products =
this.productService.findProductByCategory(category);
products.forEach(product -> {
            ProductImage img =  new ProductImage();
img.setImageData(ImageUtil.decompressImage(product.getProductImage
().getImageData()));
img.setImgId(product.getProductImage().getImgId());
img.setName(product.getProductImage().getName());
img.setType(product.getProductImage().getType());
product.setProductImage(img);
        });         if(products.isEmpty()) {             return
ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }else {             return
ResponseEntity.ok(products);
        }
    }

    @PreAuthorize("hasAuthority('ADMIN')")
@DeleteMapping("/delete/product/{id}")     public ResponseEntity<?>
deleteProduct(@PathVariable("id") Long id){
this.productService.deleteProductById(id);
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @PreAuthorize("hasAuthority('ADMIN')")     @PutMapping("/set-
availability/product/{id}")     public ResponseEntity<?>
setAvailability(@PathVariable("id") Long id,
@RequestBody Product product){
        Product updateProduct =
this.productService.findProduct(id);
updateProduct.setAvailable(product.isAvailable());
this.productService.addProduct(updateProduct);         return
ResponseEntity.status(HttpStatus.CREATED).build();
    }

    @GetMapping("/get/{name}")
    public ResponseEntity<?> getAvailable(@PathVariable("name") String
name){         List<Product> products =
this.productService.findTrueProduct(name);         if(products.isEmpty()) {
return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }else {             return
ResponseEntity.ok(products);
        }
    }



}
```

usercontroller.java –

```
package com.medicare.controller;
 import java.net.URI;
import
java.util.HashSet;
import java.util.Set;
 import
javax.annotation.PostConstruct; import
javax.validation.Valid;
 import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.CrossOrigin; import
org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestBody; import
org.springframework.web.bind.annotation.RestController; import
org.springframework.web.servlet.support.ServletUriComponentsBuilder;
 import com.medicare.entities.Role;
import com.medicare.entities.User; import
com.medicare.entities.UserRole; import
com.medicare.services.UserService;

@RestController
@CrossOrigin(origins = "*")
public class UserController {

    @Autowired     private
UserService userService;

    //init admin user
@PostConstruct
    public void createAdmin(){         User
admin = new User();
admin.setUsername("admin@medicare.com");
admin.setPassword("admin12345");
admin.setFirstName("Sachin");
admin.setLastName("Kale");
        admin.setContactNumber("88301413766");
        Role role = new Role();


        role.setRoleId(101L) ;
```

```
        role.setRoleName("ADMIN");
UserRole ur = new UserRole();
ur.setUser(admin);
ur.setRole(role);
        Set<UserRole> userRole = new HashSet<>();
userRole.add(ur);
        User adminCreated = this.userService.createUser(admin, userRole);
        System.out.println("Admin username: "+adminCreated.getUsername());
    }

    //create new user
    @PostMapping("/user/signup")
    public ResponseEntity<?> createNewUser(@Valid @RequestBody User user){
Role role = new Role();         role.setRoleId(102L);
role.setRoleName("USER");          UserRole ur = new UserRole();
ur.setUser(user);         ur.setRole(role);
        Set<UserRole> userRole = new HashSet<>();
userRole.add(ur);
        if(this.userService.getByUsername(user.getUsername())!=null)
{             System.out.println("Username already exists!");
return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }else {
            User newUser = this.userService.createUser(user, userRole);
            URI location =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(
newUser.getUserId()).toUri();             return
ResponseEntity.created(location).build();
        }
    }


}
```

UserOrderController.java ---

```
package com.medicare.controller;
```

```
 import
java.text.DateFormat; import
java.util.Calendar; import
java.util.HashSet; import
java.util.List; import
java.util.Set;
 import
javax.validation.Valid;
 import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
 import com.medicare.config.ImageUtil; import
com.medicare.entities.CartItem; import
com.medicare.entities.CartOrder; import
com.medicare.entities.Product; import
com.medicare.entities.ProductImage; import
com.medicare.entities.ProductQuantity; import
com.medicare.entities.UserOrder; import
com.medicare.services.ProductService; import
com.medicare.services.UserOrderService;

@RestController
@CrossOrigin(origins = "*") public
class UserOrderController {

    @Autowired     private UserOrderService
userOrderService;

    @Autowired     private ProductService
productService;

    @PreAuthorize("hasAuthority('USER')")
@PostMapping("/user/create/order")     public ResponseEntity<?>
createOrder(@Valid @RequestBody CartOrder cartOrder){

        UserOrder userOrder = new UserOrder();
userOrder.setUsername(cartOrder.getUsername());
```

```
userOrder.setFirstName(cartOrder.getFirstName());
userOrder.setLastName(cartOrder.getLastName());
userOrder.setAddress(cartOrder.getAddress());
userOrder.setDistrict(cartOrder.getDistrict());
userOrder.setState(cartOrder.getState());
userOrder.setContact(cartOrder.getContact());
userOrder.setPinCode(cartOrder.getPinCode());
        DateFormat df = DateFormat.getDateInstance();
        Calendar cl = Calendar.getInstance();         String
orderDate = df.format(cl.getTime());
userOrder.setDate(orderDate);
userOrder.setStatus("PLACED");
userOrder.setPaidAmount(cartOrder.getPaidAmount());
userOrder.setPaymentMode(cartOrder.getPaymentMode());
        Set<CartItem> cartItems  = cartOrder.getCartItem();
Set<ProductQuantity> pq = new HashSet<>();         for(CartItem
item : cartItems) {
            Product product = this.productService.findProduct(item.getPid());
int quantity = item.getQuantity();
            ProductQuantity productQuantity = new ProductQuantity();
productQuantity.setProduct(product);
productQuantity.setQuantity(quantity);
this.userOrderService.saveProductQuantity(productQuantity);
pq.add(productQuantity);
        }
        userOrder.setProducts(pq);
        UserOrder orderCreated = this.userOrderService.saveOrder(userOrder);
return ResponseEntity.ok(orderCreated);
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("/get/all/orders")
    public ResponseEntity<?> getAllOrders(){
        List<UserOrder> orders = this.userOrderService.getAll();
return ResponseEntity.ok(orders);
    }

    @PreAuthorize("hasAuthority('USER')")
@GetMapping("/get/orders/{username}")     public ResponseEntity<?>
userOrders(@PathVariable("username") String username){
        List<UserOrder> orders =
this.userOrderService.getUserOrders(username);
if(orders.isEmpty()) {             return
ResponseEntity.status(HttpStatus.NOT_FOUND).build();         }else {
```

```
            return ResponseEntity.ok(orders);
        }
    }

    @PreAuthorize("hasAuthority('USER') or hasAuthority('ADMIN')")
    @GetMapping("/get/order-invoice/{oid}")
    public ResponseEntity<?> getUserOrderById(@PathVariable("oid") Long oid){
        UserOrder order = this.userOrderService.getOrderById(oid);
Set<ProductQuantity> products = order.getProducts();
products.forEach(p -> {
            ProductImage img = new ProductImage();
img.setImageData(ImageUtil.decompressImage(p.getProduct().getProdu
ctImage().getImageData()));
img.setName(p.getProduct().getProductImage().getName());
img.setImgId(p.getProduct().getProductImage().getImgId());
img.setType(p.getProduct().getProductImage().getType());

p.getProduct().setProductImage(img);
});          order.setProducts(products);
return ResponseEntity.ok(order);
    }

    @PreAuthorize("hasAuthority('ADMIN')")
@DeleteMapping("/delete/order/{oid}")     public ResponseEntity<?>
deleteOrderById(@PathVariable("oid") Long oid){
this.userOrderService.deleteOrder(oid);          return
ResponseEntity.status(HttpStatus.OK).build();
    }

}
```