3. **Problem Statement:** : **E-commerce Product Catalog**

Create a RESTful API to manage an e-commerce product catalog. The API should support product listing, filtering by category or price, and searching by product name.

Answer :

```
package com.example.catalog;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.stereotype.Repository;
import org.springframework.web.bind.annotation.*;

import javax.persistence.*;
import javax.validation.constraints.*;
import java.util.List;

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

@Entity
class Product {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long productId;

  @NotBlank
```

```
    private String name;

    private String description;

    @DecimalMin("0.0")
    private double price;

    @NotBlank
    private String category;

    @Min(0)
    private int stockQuantity;

    // Getters and Setters
}

@Repository
interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByCategory(String category);

    @Query("SELECT p FROM Product p WHERE p.price BETWEEN :min AND :max")
    List<Product> findByPriceRange(@Param("min") double min, @Param("max") double max);

    List<Product> findByNameContainingIgnoreCase(String name);
}

@RestController
@RequestMapping("/api/products")
class ProductController {

    private final ProductRepository repository;

    public ProductController(ProductRepository repository) {
        this.repository = repository;
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return repository.findAll();
    }

    @PostMapping
    public ResponseEntity<Product> createProduct(@RequestBody @Valid Product product) {
        return new ResponseEntity<>(repository.save(product), HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product> updateProduct(@PathVariable Long id, @RequestBody
@Valid Product updatedProduct) {
        return repository.findById(id).map(product -> {
            updatedProduct.setProductId(id);
            return new ResponseEntity<>(repository.save(updatedProduct), HttpStatus.OK);
        }).orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        return repository.findById(id).map(p -> {
            repository.deleteById(id);
            return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
        }).orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/category/{category}")
    public List<Product> getByCategory(@PathVariable String category) {
        return repository.findByCategory(category);
    }

    @GetMapping("/search")
    public List<Product> searchByName(@RequestParam String name) {
        return repository.findByNameContainingIgnoreCase(name);
    }

    @GetMapping("/filter")
    public List<Product> filterByPrice(@RequestParam double min, @RequestParam double max)
{
        return repository.findByPriceRange(min, max);
    }
}

@EnableWebSecurity
class SecurityConfig {
    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        UserDetails admin = User.withDefaultPasswordEncoder()
                .username("admin")
                .password("adminpass")
                .roles("ADMIN")
                .build();
        UserDetails user = User.withDefaultPasswordEncoder()
                .username("user")
                .password("userpass")
                .roles("USER")
                .build();
        return new InMemoryUserDetailsManager(admin, user);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/api/products/**").hasAnyRole("ADMIN", "USER")
            .antMatchers(HttpMethod.POST, "/api/products/**").hasRole("ADMIN")
            .antMatchers(HttpMethod.PUT, "/api/products/**").hasRole("ADMIN")
            .antMatchers(HttpMethod.DELETE, "/api/products/**").hasRole("ADMIN")
            .and().httpBasic();
        return http.build();
    }
}
```