# Controller

```
package com.simplilearn.controller;

import java.time.LocalDate;
import java.util.stream.Collectors;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import com.simplilearn.exception.OrderException;
import com.simplilearn.exception.ProductCategoryException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.exception.UserException;
import com.simplilearn.exception.UserRoleException;
import com.simplilearn.entity.ErrorInfo;

@RestControllerAdvice
public class GlobalExceptionHandler {

	// This method will handle all custom Exceptions UserException, ProductException, OrderException, UserRoleException
and ProductCategoryException
	@ExceptionHandler({ OrderException.class, ProductException.class, UserException.class, UserRoleException.class,
ProductCategoryException.class })
	public ResponseEntity<ErrorInfo> customExceptionHandler(Exception exception) {
		ErrorInfo errorResponse = new ErrorInfo(HttpStatus.BAD_REQUEST.value() + " : BAD_REQUEST",
exception.getMessage(),
				LocalDate.now());
		return new ResponseEntity<ErrorInfo>(errorResponse, HttpStatus.BAD_REQUEST);
	}

	// This method will handle all general exceptions
	@ExceptionHandler(Exception.class)
	public ResponseEntity<ErrorInfo> generalExceptionHandler(Exception exception) {
		String message = "Some error occured. Please contact administrator. ";
		ErrorInfo errorResponse = new ErrorInfo(HttpStatus.INTERNAL_SERVER_ERROR.value() + " :
INTERNAL_SERVER_ERROR",
				message + exception.getMessage(), LocalDate.now());
		return new ResponseEntity<ErrorInfo>(errorResponse, HttpStatus.INTERNAL_SERVER_ERROR);
	}

	// This method will handle Argument Validation Exceptions
	@ExceptionHandler(MethodArgumentNotValidException.class)
	public ResponseEntity<ErrorInfo> exceptionHandler(MethodArgumentNotValidException exception) {
		String errorMessage = exception.getBindingResult().getAllErrors().stream().map(x -> x.getDefaultMessage())
				.collect(Collectors.joining(", "));

		ErrorInfo errorInfo = new ErrorInfo();
		errorInfo.setErrorCode(HttpStatus.BAD_REQUEST.value() + " : BAD_REQUEST");
		errorInfo.setErrorMessage(errorMessage);
		errorInfo.setTimeStamp(LocalDate.now());

		return new ResponseEntity<ErrorInfo>(errorInfo, HttpStatus.BAD_REQUEST);
	}
}
```

```
package com.simplilearn.controller;

import com.simplilearn.model.JwtRequest;
import com.simplilearn.model.JwtResponse;
import com.simplilearn.service.JwtService;

import org.springframework.beans.factory.annotation.Autowired;
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;

@RestController
@CrossOrigin
public class JwtController {

    @Autowired
    private JwtService jwtService;

    // Accessible for All | End Point URL -> http://localhost:9090/authenticate
    @PostMapping({"/authenticate"})
    public JwtResponse createJwtToken(@RequestBody JwtRequest jwtRequest) throws Exception {
        return jwtService.createJwtToken(jwtRequest);
    }
}
```

```
package com.simplilearn.controller;

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.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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.simplilearn.entity.Order;
import com.simplilearn.exception.OrderException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.exception.UserException;
import com.simplilearn.model.OrderRequest;
import com.simplilearn.service.OrderService;

@RestController
@RequestMapping("/api/orders")
public class OrderController {
	
	@Autowired
	private OrderService orderService;
	
	// Accessible for User | End Point URL -> http://localhost:9090/api/orders/order
	@PostMapping("/order")
    @PreAuthorize("hasRole('User')")
	public ResponseEntity<String > placeOrder(@Valid @RequestBody OrderRequest orderDTO) throws OrderException,
ProductException, UserException{
		String preMessage = "Order Successfully placed. Order Tracking Number is ";
		return new  ResponseEntity<String>(preMessage+orderService.insertOrder(orderDTO), HttpStatus.OK);
	}
	
	// Accessible for User | End Point URL -> http://localhost:9090/api/orders/modifyOrder
	@PutMapping("/modifyOrder")
    @PreAuthorize("hasRole('User')")
	public ResponseEntity<Order> modifyOrder(@Valid @RequestBody OrderRequest orderDTO) throws OrderException,
ProductException{
		return new ResponseEntity<Order>(orderService.updateOrder(orderDTO), HttpStatus.OK);
	}
	
	// Accessible for User | End Point URL -> http://localhost:9090/api/orders/deleteOrder/1
	@DeleteMapping("/deleteOrder/{orderId}")
    @PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<String> deleteOrder(@PathVariable("orderId") Integer orderId) throws ProductException,
OrderException{
		String message = "Order with order id " + orderService.deleteOrder(orderId) + " deleted successfully.";
		return new ResponseEntity<String>(message, HttpStatus.OK);
	}
	
	// Accessible for Admin | End Point URL -> http://localhost:9090/api/orders/orderByDateCreated
	@GetMapping("/orderByDateCreated")
    @PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<List<Order>> getOrdersSortedByDateCreated(){
		return new ResponseEntity<List<Order>>(orderService.getOrdersSortedByDateCreated(), HttpStatus.OK);
	}
	
	// Accessible for Admin | End Point URL -> http://localhost:9090/api/orders/orderByProductCategory
	@GetMapping("/orderByProductCategory")
  @PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<List<Order>> getOrdersSortedByProductCategory(){
		return new ResponseEntity<List<Order>>(orderService.getOrdersSortedByProductCategory(), HttpStatus.OK);
	}
	
	// Accessible for Admin | End Point URL -> http://localhost:9090/api/orders/orderByDateUpdated
	@GetMapping("/orderByDateUpdated")
  @PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<List<Order>> getOrdersSortedByDateUpdated(){
		return new ResponseEntity<List<Order>>(orderService.getOrdersSortedByDateUpdated(), HttpStatus.OK);
	}
}
```

```
package com.simplilearn.controller;

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.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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.simplilearn.exception.ProductCategoryException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.model.ProductDTO;
import com.simplilearn.entity.Product;
import com.simplilearn.service.ProductService;

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

	@Autowired
	private ProductService productService;

	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/product
	@PostMapping("/product")
	@PreAuthorize("hasRole('Admin')")
	public ResponseEntity<Product> addProduct(@Valid @RequestBody ProductDTO productDTO) throws
ProductException, ProductCategoryException{
		return new ResponseEntity<Product>(productService.addProduct(productDTO), HttpStatus.CREATED);
	}

	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/updateProduct
	@PutMapping("/updateProduct")
	@PreAuthorize("hasRole('Admin')")
	public ResponseEntity<Product> updateProduct(@Valid @RequestBody ProductDTO productDTO) throws
ProductException, ProductCategoryException{
		return new ResponseEntity<Product>(productService.updateProduct(productDTO), HttpStatus.OK);
	}

	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/product/5
	@DeleteMapping("/product/{productId}")
	@PreAuthorize("hasRole('Admin')")
	public ResponseEntity<String> deleteProduct(@PathVariable("productId") Integer productId) throws ProductException
{
		productService.deleteProduct(productId);
		return ResponseEntity.ok("Product with "+ productId +" Deleted Successfully.");
	}

	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/getProduct/4
	@GetMapping("/getProduct/{productId}")
	@PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<Product> getProductDetails(@PathVariable("productId") Integer productId) throws
ProductException{
		return new ResponseEntity<Product>(productService.getProduct(productId), HttpStatus.FOUND);
	}
	
	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/getProducts
	@GetMapping("/getProducts")
	@PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<List<Product>> getAllProducts() {
		return new ResponseEntity<List<Product>>(productService.getAllProducts(), HttpStatus.FOUND);
	}
	
	// Accessible for Admin | End Point URL -> http://localhost:9090/api/products/sortedProducts
	@GetMapping("/sortedProducts")
	@PreAuthorize("hasAnyRole('Admin','User')")
	public ResponseEntity<List<Product>> sortProductByCategory() {
		return new ResponseEntity<List<Product>>(productService.sortProductByCategory(), HttpStatus.FOUND);
	}
	
	
	

}
```

```
package com.simplilearn.controller;

import com.simplilearn.entity.Role;
import com.simplilearn.service.RoleService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/userRoles")
public class RoleController {

    @Autowired
    private RoleService roleService;

    // Accessible for Admin | End Point URL -> http://localhost:9090/api/userRoles/createNewRole
    @PostMapping({"/createNewRole"})
    @PreAuthorize("hasRole('Admin')")
    public Role createNewRole(@RequestBody Role role) {
        return roleService.createNewRole(role);
    }
}
```

```
package com.simplilearn.controller;

import com.simplilearn.entity.User;
import com.simplilearn.exception.UserException;
import com.simplilearn.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.PostConstruct;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostConstruct
    public void initRoleAndUser() {
        userService.initRoleAndUser();
    }

    // Accessible for All | End Point URL -> http://localhost:9090/api/users/registerNewUser
    @PostMapping({"/registerNewUser"})
    public User registerNewUser(@RequestBody User user) {
        return userService.registerNewUser(user);
    }

    // Accessible for All | End Point URL -> http://localhost:9090/api/users/getUserDetails/admin123
    @GetMapping({"/getUserDetails/{userName}"})
    @PreAuthorize("hasAnyRole('Admin','User')")
    public User getUserDetails(@PathVariable("userName") String userName) throws UserException{
        return userService.getUserDetails(userName);
    }

    // Accessible for Admin | End Point URL -> http://localhost:9090/api/users/updatePassowrd/admin123/232063
    @GetMapping({"/updatePassowrd/{userName}/{password}"})
    @PreAuthorize("hasRole('Admin')")
    public User updatePassowrd(@PathVariable("userName") String userName, @PathVariable("password") String password)
throws UserException{
        return userService.updatePassword(userName, password);
    }




    // Authorization Testing Functionality

    // Accessible for Admin | End Point URL -> http://localhost:9090/api/users/forAdmin
    @GetMapping({"/forAdmin"})
    @PreAuthorize("hasRole('Admin')")
    public String forAdmin(){
        return "This URL is only accessible to the admin";
    }

    // Accessible for Admin | End Point URL -> http://localhost:9090/api/users/forUser
    @GetMapping({"/forUser"})
    @PreAuthorize("hasRole('User')")
    public String forUser(){
        return "This URL is only accessible to the user";
    }
}
```

# Configuration

```
package com.simplilearn.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpHeaders;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class  WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

    @Autowired
    private UserDetailsService jwtService;

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.cors();
        httpSecurity.csrf().disable()
                .authorizeRequests().antMatchers("/authenticate", "/api/users/registerNewUser").permitAll()
                .antMatchers(HttpHeaders.ALLOW).permitAll()
                .anyRequest().authenticated()
                .and()
                .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint)
                .and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        ;
        httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder.userDetailsService(jwtService).passwordEncoder(passwordEncoder());
    }
}
```

```java
package com.simplilearn.configuration;


import com.simplilearn.service.JwtService;
import com.simplilearn.util.JwtUtil;


import io.jsonwebtoken.ExpiredJwtException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;


import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;


@Component
public class JwtRequestFilter extends OncePerRequestFilter {


    @Autowired
    private JwtUtil jwtUtil;


    @Autowired
    private JwtService jwtService;


    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {


        final String requestTokenHeader = request.getHeader("Authorization");


        String username = null;
        String jwtToken = null;
        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
            jwtToken = requestTokenHeader.substring(7);
            try {
                username = jwtUtil.getUsernameFromToken(jwtToken);
            } catch (IllegalArgumentException e) {
                System.out.println("Unable to get JWT Token");
            } catch (ExpiredJwtException e) {
                System.out.println("JWT Token has expired");
            }
        } else {
            System.out.println("JWT token does not start with Bearer");
        }

        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            UserDetails userDetails = jwtService.loadUserByUsername(username);

            if (jwtUtil.validateToken(jwtToken, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        filterChain.doFilter(request, response);

    }

}
```

```
package com.simplilearn.configuration;


import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;


import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;


@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {


    @Override

    public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException
authException) throws IOException, ServletException {

        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "This endpoint is not accessible for you.");

    }


}
```

```
package com.simplilearn.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfiguration {

    private static final String GET = "GET";
    private static final String POST = "POST";
    private static final String PUT = "PUT";
    private static final String DELETE = "DELETE";

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedMethods(GET, POST, PUT, DELETE)
                        .allowedHeaders("*")
                        .allowedOriginPatterns("*")
                        .allowCredentials(true);
            }
        };
    }
}
```

# Entity

```
package com.simplilearn.entity;

import javax.persistence.*;

import com.simplilearn.entity.Order;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.HashSet;
import java.util.Set;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "USER_TABLE")
public class User {
        @Id
        private String userName;
        private String userFirstName;
        private String userLastName;
        private String userPassword;
        @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
        @JoinTable(name = "USER_ROLE", joinColumns = { @JoinColumn(name = "USER_ID") }, inverseJoinColumns = {
                        @JoinColumn(name = "ROLE_ID") })
        private Set<Role> role;

        @OneToMany(targetEntity = Order.class,cascade = CascadeType.ALL)
  @JoinColumn(name ="username_fk",referencedColumnName = "userName")
        private Set<Order> orders = new HashSet<Order>();

        public Set<Order> addOrder(Order order){
                this.orders.add(order);
                return this.orders;
```

```
	}

	public Set<Order> removeOrder(Order order){
		this.orders.remove(order);
		return this.orders;
	}

}
```

```
package com.simplilearn.entity;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "ROLE_TABLE")
public class Role {

    @Id
    private String roleName;
    private String roleDescription;

}
```

```
package com.simplilearn.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity(name = "PRODUCT_CATEGORY")
@Table(name = "PRODUCT_CATEGORY")
public class ProductCategory {
	@Id
//	@GeneratedValue(strategy = GenerationType.AUTO)
//	private Integer id;
	private String categoryName;
	private String categoryDescription;
	
	@OneToMany(targetEntity = Product.class,cascade = CascadeType.ALL)
    @JoinColumn(name ="prod_category_fk",referencedColumnName = "categoryName")
	private Set<Product> products = new HashSet<>();
	
	public Set<Product> addProduct(Product product){
		this.products.add(product);
		return this.products;
	}
	public Set<Product> removeProduct(Product product){
		this.products.remove(product);
		return this.products;
	}
}
```

```
package com.simplilearn.entity;


import java.util.HashSet;
import java.util.Set;


import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity(name = "PRODUCTS")
@Table(name = "products")
public class Product {
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Integer id;
        private String productName;
        private String productDescription;
        private Double unitPrice;
        private String manufacturer;
        private Integer unitsInStock;

        @OneToMany(targetEntity = Order.class,cascade = CascadeType.ALL)
   @JoinColumn(name ="product_fk",referencedColumnName = "id")
        private Set<Order> orders = new HashSet<>();
```

```

	public Set<Order> addOrder(Order order){
		this.orders.add(order);
		return this.orders;
	}

	public Set<Order> removeOrder(Order order){
		this.orders.remove(order);
		return this.orders;
	}

}
```

```
package com.simplilearn.entity;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "orders")
public class Order {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer orderId;
	private String orderTrackingNumber;
	private int totalQuantity;
	private Double totalPrice;
	private String address;
	private Integer productId;
	@CreationTimestamp
	private Date dateCreated;
	@UpdateTimestamp
	private Date lastUpdated;
}
```

```
package com.simplilearn.entity;


import java.time.LocalDate;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class ErrorInfo{

        private String errorCode;
        private String errorMessage;
        private LocalDate timeStamp;


}
```

# **Exception**

```java
package com.simplilearn.exception;

public class UserRoleException extends Exception{

	private static final long serialVersionUID = 1L;

	public UserRoleException(String message) {
		super(message);
	}

}
```

```java
package com.simplilearn.exception;

public class UserException extends Exception{

	private static final long serialVersionUID = 1L;

	public UserException(String message) {
		super(message);
	}
}
```

```java
package com.simplilearn.exception;

public class ProductException extends Exception{

	private static final long serialVersionUID = 1L;

	public ProductException(String message) {
		super(message);
	}

}
```

```java
package com.simplilearn.exception;

public class ProductCategoryException extends Exception{

	private static final long serialVersionUID = 1L;

	public ProductCategoryException(String message) {
		super(message);
	}

}
```

```java
package com.simplilearn.exception;

public class OrderException extends Exception {

	private static final long serialVersionUID = 1L;

	public OrderException(String message) {
		super(message);
	}



}
```

## **Model**

```
package com.simplilearn.model;


import java.util.ArrayList;
import java.util.List;


import javax.validation.constraints.NotNull;
import javax.validation.constraints.Positive;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
@AllArgsConstructor
public class ProductDTO {
	private Integer productId;
	@NotNull(message = "Product name cannot be null")
	private String productName;
	@NotNull(message = "Product description cannot be null")
	private String productDescription;
	@NotNull(message = "Product category name cannot be null")
	private String productCategoryName;
	@NotNull(message = "Product category description cannot be null")
	private String productCategoryDescription;
	@NotNull(message = "Product manufacturer cannot be null")
	private String manufacturer;
	@NotNull(message = "Unit price for product cannot be null")
	@Positive(message = "Unit Price should be positive number.")
	private Double unitPrice;
	@NotNull(message = "Units in stock for product cannot be null")
	@Positive(message = "Units In Stock should be positive number.")
	private Integer unitsInStock;
	private List<OrderRequest> orderDTOList = new ArrayList<>();
}
```

```
package com.simplilearn.model;


import javax.validation.constraints.NotNull;
import javax.validation.constraints.Positive;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
@AllArgsConstructor
public class OrderRequest {


	private Integer orderId;
	private String orderTrackingNumber;
	@NotNull(message = "Total quantity of products cannot be null")
	@Positive(message = "Total quantity should be positive number.")
	private int totalQuantity;
	@NotNull(message = "Billing address cannot be null")
	private String address;
	@NotNull(message = "Product id cannot be null")
	@Positive(message = "Product id should be positive number.")
	private Integer productId;
	@NotNull(message = "Username cannot be null")
	private String userName;
}
```

```
package com.simplilearn.model;


import com.simplilearn.entity.User;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class JwtResponse {


    private User user;
    private String jwtToken;
}
```

```
package com.simplilearn.model;


import javax.validation.constraints.NotNull;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class JwtRequest {


		@NotNull(message = "User name cannot be null.")
    private String userName;
		@NotNull(message = "User Password cannot be null.")
    private String userPassword;
}
```

# **Repository**

```
   package com.simplilearn.repository;


import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;


import com.simplilearn.entity.User;


@Repository
public interface UserRepository extends CrudRepository<User, String> {
}
```

```
package com.simplilearn.repository;


import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;


import com.simplilearn.entity.Role;


@Repository
public interface RoleRepository extends CrudRepository<Role, String> {


}
```

```
package com.simplilearn.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import com.simplilearn.entity.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer>{

//	This will fetch the product category for product whose product id is passed as input parameter
	@Query("SELECT C.categoryName FROM PRODUCT_CATEGORY C JOIN C.products P WHERE P.id = :productId")
	String getProductCategory(@Param("productId") Integer productId);

}
```

```
package com.simplilearn.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.simplilearn.entity.ProductCategory;

@Repository
public interface ProductCategoryRepository extends JpaRepository<ProductCategory, Integer>{

//	This method will fetch product category based on product category name
	public Optional<ProductCategory> findByCategoryName(String categoryName);
}
```

```
package com.simplilearn.repository;


import java.util.Optional;


import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;


import com.simplilearn.entity.Order;
import com.simplilearn.entity.Product;


@Repository
public interface OrderRepository extends JpaRepository<Order, Integer>{
	// This will fetch product associated with order whose order id is passed as input parameter.
	@Query("SELECT P FROM PRODUCTS P JOIN P.orders O WHERE O.orderId = :orderId")
	Optional<Product> getProductFromOrderId(@Param("orderId") Integer orderId);
}
```

## **Service**

```
package com.simplilearn.service;

import com.simplilearn.entity.Role;
import com.simplilearn.entity.User;
import com.simplilearn.exception.UserException;
import com.simplilearn.repository.RoleRepository;
import com.simplilearn.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.HashSet;
import java.util.Set;

@Service
public class UserService {

  @Autowired
  private UserRepository userRepository;

  @Autowired
  private RoleRepository roleRepository;

  @Autowired
  private PasswordEncoder passwordEncoder;

  // This method will create Admin user
  public void initRoleAndUser() {
  	// Create Role object for Admin Role and persisting it to DB.
    Role adminRole = new Role();
    adminRole.setRoleName("Admin");
    adminRole.setRoleDescription("Admin role");
    roleRepository.save(adminRole);

  	// Create Role object for User Role and persisting it to DB.
```

```
    Role userRole = new Role();
    userRole.setRoleName("User");
    userRole.setRoleDescription("Default role for newly created record");
    roleRepository.save(userRole);

        // Create User object for Admin and assigning it with Admin role and persisting it to DB.
    User adminUser = new User();
    adminUser.setUserName("admin123");
    adminUser.setUserPassword(getEncodedPassword("admin@pass"));
    adminUser.setUserFirstName("admin");
    adminUser.setUserLastName("admin");
    Set<Role> adminRoles = new HashSet<>();
    adminRoles.add(adminRole);
    adminUser.setRole(adminRoles);
    userRepository.save(adminUser);
}

// This method will create new user and assign it with User Role.
public User registerNewUser(User user) {
    Role role = roleRepository.findById("User").get();
    Set<Role> userRoles = new HashSet<>();
    userRoles.add(role);
    user.setRole(userRoles);
    user.setUserPassword(getEncodedPassword(user.getUserPassword()));

    return userRepository.save(user);
}

// This method will fetch User details based on user name.
public User getUserDetails(String userName) throws UserException {
        return userRepository.findById(userName).orElseThrow(() -> new UserException("User not found."));
}

// This method will fetch update password for user
public User updatePassword(String userName, String password) throws UserException {
        User user = userRepository.findById(userName).orElseThrow(() -> new UserException("User not found."));
```

```
            user.setUserPassword(getEncodedPassword(password));
            return userRepository.save(user);
    }

    // This method will encode the raw string password provided.
    public String getEncodedPassword(String password) {
        return passwordEncoder.encode(password);
    }
}
```

```
package com.simplilearn.service;


import com.simplilearn.entity.Role;
import com.simplilearn.repository.RoleRepository;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


@Service
public class RoleService {


    @Autowired
    private RoleRepository roleRepository;


    // This method will create New Role and persist it to DB
    public Role createNewRole(Role role) {
        return roleRepository.save(role);
    }
}
```

```
package com.simplilearn.service;

import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.simplilearn.entity.Product;
import com.simplilearn.entity.ProductCategory;
import com.simplilearn.exception.ProductCategoryException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.model.ProductDTO;
import com.simplilearn.repository.ProductCategoryRepository;
import com.simplilearn.repository.ProductRepository;

@Service
@Transactional
public class ProductServiceImpl implements ProductService {

	private ProductRepository productRepository;
	private ProductCategoryRepository productCategoryRepository;

	// List of valid product categories -> SPORTS, TREKKING, FORMAL, CASUAL, LOAFER.
	String[] productCategories = { "SPORTS", "TREKKING", "FORMAL", "CASUAL", "LOAFER" };

	@Autowired
	public ProductServiceImpl(ProductRepository productRepository,
			ProductCategoryRepository productCategoryRepository) {
		this.productRepository = productRepository;
		this.productCategoryRepository = productCategoryRepository;
	}
	// This method will insert product into DB as per input productDTO object
	// provided.
	@Override
	public Product addProduct(ProductDTO productDTO) throws ProductException, ProductCategoryException {
		// Initialization
		Product product = null;
		Product savedProduct = null;
		ProductCategory productCategory = null;
		Boolean validCategoryFlag = false;

		// If input productDTO is null then throwing Product Exception.
		if (productDTO == null) {
			throw new ProductException("Product Input is NULL");
		} else {

			// Validating product category. if product category provided is not valid then
			// throwing Product Category Exception.
			for (String category : productCategories) {
				if (category.equals(productDTO.getProductCategoryName()))
					validCategoryFlag = true;
			}
			if (!validCategoryFlag)
				throw new ProductCategoryException(
						"Provided input product category is not valid. Valid product categories are
['SPORTS','TREKKING','FORMAL','CASUAL','LOAFER']");

			// If product category provided in input productDTO is exists in DB then
			// fetching product category or else creating new product category to persist it
			// to DB along with product.
			if
(productCategoryRepository.findByCategoryName(productDTO.getProductCategoryName()).isPresent()) {
				productCategory =
productCategoryRepository.findByCategoryName(productDTO.getProductCategoryName())
						.orElseThrow(() -> new ProductCategoryException(
								"Please provide valid product category. Valid product
categories are ['SPORTS','TREKKING','FORMAL','CASUAL','LOAFER']"));
			} else {
				productCategory = new ProductCategory();
```

```
                productCategory.setCategoryName(productDTO.getProductCategoryName());
                productCategory.setCategoryDescription(productDTO.getProductCategoryDescription());
            }

            // Creating new Product object and populating data and persisting it to DB.
            product = new Product();
            product.setProductName(productDTO.getProductName());
            product.setProductDescription(productDTO.getProductDescription());
            product.setManufacturer(productDTO.getManufacturer());
            product.setUnitPrice(productDTO.getUnitPrice());
            product.setUnitsInStock(productDTO.getUnitsInStock());
            savedProduct = productRepository.save(product);

            // Adding product to product category and persisting modified product category
            // to DB.
            productCategory.addProduct(savedProduct);
            productCategoryRepository.save(productCategory);

            return savedProduct;
        }
    }

    // This method will update product from DB as per input productDTO object
    // provided.
    @Override
    public Product updateProduct(ProductDTO productDTO) throws ProductException, ProductCategoryException {
        // Initialization
        Product product = null;

        if(productDTO.getProductId() == null) {
            throw new ProductException("Please provide product id to update the product.");
        }
        // If product id provided in input productDTO is exists in DB then fetching
        // corresponding product or else throwing Product Exception.
        product = productRepository.findById(productDTO.getProductId())
                .orElseThrow(() -> new ProductException("Product not found. Please try again with valid
product id."));
```

```
		// Creating new Product object and populating data and persisting it to DB.
		product.setProductName(productDTO.getProductName());
		product.setProductDescription(productDTO.getProductDescription());
		product.setManufacturer(productDTO.getManufacturer());
		product.setUnitPrice(productDTO.getUnitPrice());
		product.setUnitsInStock(productDTO.getUnitsInStock());

		// Saving modified product to DB and resturning it.
		return productRepository.save(product);
	}

	// This method will delete product from DB as per input product id provided.
	@Override
	public void deleteProduct(Integer productId) throws ProductException {
		// If product id provided exists in DB then fetching corresponding product or
		// else throwing Product Exception.
		Product product = productRepository.findById(productId)
				.orElseThrow(() -> new ProductException("Product not found. Please try again with valid
product id."));

		// Deleting product from DB.
		productRepository.delete(product);
	}

	// This method will fetch product details from DB as per input product id
	// provided.
	@Override
	public Product getProduct(Integer productId) throws ProductException {
		// If product id provided exists in DB then fetching corresponding product or
		// else throwing Product Exception.
		Product product = productRepository.findById(productId)
				.orElseThrow(() -> new ProductException("Product Not Found. Please try again with valid
product id."));
		return product;
	}
```

```
    // This method will fetch all products from DB and then returns them.
    @Override
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    // This method will fetch all products from DB and sort them by product category
    // and then returns them.
    @Override
    public List<Product> sortProductByCategory() {
        // Comparator to sort List based on product category
        Comparator<Product> sortByProductCategory = new Comparator<Product>() {
            @Override
            public int compare(Product product1, Product product2) {
                return productRepository.getProductCategory(product1.getId())
                        .compareTo(productRepository.getProductCategory(product2.getId()));
            }
        };

        // Fetching and sorting all orders from DB
        List<Product> productList = productRepository.findAll();
        Collections.sort(productList, sortByProductCategory);

        return productList;
    }
}
```

```
package com.simplilearn.service;

import java.util.List;

import com.simplilearn.entity.Product;
import com.simplilearn.exception.ProductCategoryException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.model.ProductDTO;

public interface ProductService {

	Product addProduct(ProductDTO productDTO) throws ProductException, ProductCategoryException;

	Product updateProduct(ProductDTO productDTO) throws ProductException, ProductCategoryException ;

	void deleteProduct(Integer productId) throws ProductException ;

	Product getProduct(Integer productId) throws ProductException ;

	List<Product> getAllProducts();

	List<Product> sortProductByCategory();

}
```

```
package com.simplilearn.service;

import java.util.Calendar;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.UUID;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.simplilearn.entity.Order;
import com.simplilearn.entity.Product;
import com.simplilearn.entity.User;
import com.simplilearn.exception.OrderException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.exception.UserException;
import com.simplilearn.model.OrderRequest;
import com.simplilearn.repository.OrderRepository;
import com.simplilearn.repository.ProductRepository;
import com.simplilearn.repository.UserRepository;

@Service
@Transactional
public class OrderServiceImpl implements OrderService {

	private OrderRepository orderRepository;
	private ProductRepository productRepository;
	private UserRepository userRepository;

	@Autowired
	public OrderServiceImpl(OrderRepository orderRepository, ProductRepository productRepository,
			UserRepository userRepository) {
		this.orderRepository = orderRepository;
		this.productRepository = productRepository;
		this.userRepository= userRepository;
	}

	// This method will insert order into DB as per input OrderRequest object
	// provided
	@Override
	public String insertOrder(OrderRequest orderRequest) throws OrderException, ProductException, UserException {
		// Variable Initialization
		Order order = null;
		Order savedOrder = null;

		if (orderRequest != null) {
			// Fetching user from DB based on user name provided in orderRequest object or
			// else throwing user exception.
			User user = userRepository.findById(orderRequest.getUserName()).orElseThrow(() -> new
UserException(
					"User not found, so cannot place this order. Please try again with valid User
name."));

			// Fetching product from DB based on product id provided in orderRequest object
			// or else throwing product exception.
			Product product = productRepository.findById(orderRequest.getProductId())
					.orElseThrow(() -> new ProductException(
							"Product not found. So cannot place your order. Please try again
with valid product."));
			product.setUnitsInStock(product.getUnitsInStock() - orderRequest.getTotalQuantity());

			// Creating and populating Order object to persist to DB
			order = new Order();
			order.setAddress(orderRequest.getAddress());
			order.setDateCreated(Calendar.getInstance().getTime());
			order.setLastUpdated(Calendar.getInstance().getTime());
			order.setTotalQuantity(orderRequest.getTotalQuantity());
			order.setTotalPrice(product.getUnitPrice() * orderRequest.getTotalQuantity());
			order.setProductId(product.getId());
			order.setOrderTrackingNumber(generateUniqueTrackingNumber());
			savedOrder = orderRepository.save(order);

			// Adding order object in product fetched previously and saving modified product
			// to DB.
			product.addOrder(savedOrder);
			productRepository.save(product);
			// Adding order object in user fetched previously and saving modified user to
			// DB.
			user.addOrder(savedOrder);
			userRepository.save(user);

		} else {
			throw new OrderException("Order input cannot be null");
		}
		return savedOrder.getOrderTrackingNumber();
	}

	// This method will update order as per input OrderRequest object provided
	@Override
	public Order updateOrder(OrderRequest orderRequest) throws OrderException, ProductException {
		// Fetching order from DB based on order id provided in orderRequest object or
		// else throwing order exception.
		Order order = orderRepository.findById(orderRequest.getOrderId())
				.orElseThrow(() -> new OrderException("Order Not found. Cannot update details."));

		// Fetching product from DB based on order id provided in orderRequest object or
		// else throwing product exception.
		Product product = orderRepository.getProductFromOrderId(orderRequest.getOrderId())
				.orElseThrow(() -> new ProductException(
						"Product associated with this order is removed from application. Please
contact administrator."));

		if (product.getId() != orderRequest.getProductId()) {
			throw new OrderException("This order wasn't placed for product id " + orderRequest.getProductId()
					+ ". This order was placed for product id " + product.getId() + ".");
		}
		if (orderRequest.getTotalQuantity() == 0) {
			throw new OrderException("Products quantity cannot be 0 for Order");
		}
		int orderQuantity = order.getTotalQuantity();
		order.setTotalQuantity(orderRequest.getTotalQuantity());
		order.setAddress(orderRequest.getAddress());
		order.setTotalPrice(product.getUnitPrice() * order.getTotalQuantity());
		Order savedOrder = orderRepository.save(order);

		product.setUnitsInStock(product.getUnitsInStock() + (orderQuantity - orderRequest.getTotalQuantity()));
		productRepository.save(product);

		return savedOrder;
	}

	// This method will delete order from DB based on input order id
	@Override
	public Integer deleteOrder(Integer orderId) throws ProductException, OrderException {
		// Fetching order based on input order id from DB and deleting if order fetched
		// successfully or else throwing order exception.
		Order order = orderRepository.findById(orderId)
				.orElseThrow(() -> new OrderException("No orders found to delete."));
		orderRepository.deleteById(orderId);

		// Fetching product based on product id associated with previously fetched order
		// or else throwing product exception
		Product product = productRepository.findById(order.getProductId()).orElseThrow(() -> new ProductException(
				"No product linked with this order. Order cannot be deleted. Please contact adminitrator."));

		// Updating Units In Stock for product as order is deleted and then saving
		// product.
		product.setUnitsInStock(product.getUnitsInStock() + order.getTotalQuantity());
		productRepository.save(product);

		return orderId;
	}
```

```
// This method will fetch all orders from DB and sort them based on date of
// order creation
@Override
public List<Order> getOrdersSortedByDateCreated() {
	// Comparator to sort List based on date creation for order
	Comparator<Order> sortByDateCreated = new Comparator<Order>() {
		@Override
		public int compare(Order O1, Order O2) {
			if (O1.getDateCreated().compareTo(O2.getDateCreated()) > 0) {
				return 1;
			} else if (O1.getDateCreated().compareTo(O2.getDateCreated()) < 0) {
				return -1;
			} else {
				return 0;
			}
		}
	};

	// Fetching and sorting all orders from DB
	List<Order> orders = orderRepository.findAll();
	Collections.sort(orders, sortByDateCreated);

	return orders;
}

// This method will fetch all orders from DB and sort them based on product
// category
@Override
public List<Order> getOrdersSortedByProductCategory() {
	// Comparator to sort List based on product category
	Comparator<Order> sortByProductCategory = new Comparator<Order>() {
		@Override
		public int compare(Order O1, Order O2) {
			return productRepository.getProductCategory(O1.getProductId())
					.compareTo(productRepository.getProductCategory(O2.getProductId()));
```

```
                }
            };

            // Fetching and sorting all orders from DB
            List<Order> orders = orderRepository.findAll();
            Collections.sort(orders, sortByProductCategory);

            return orders;
    }

    // This method will fetch all orders from DB and sort them based on last date of
    // order modification
    @Override
    public List<Order> getOrdersSortedByDateUpdated() {
            // Comparator to sort List based on last date of order modification
            Comparator<Order> sortByDateUpdated = new Comparator<Order>() {
                @Override
                public int compare(Order O1, Order O2) {
                        return O1.getLastUpdated().compareTo(O2.getLastUpdated());
                }
            };

            // Fetching and sorting all orders from DB
            List<Order> orders = orderRepository.findAll();
            Collections.sort(orders, sortByDateUpdated);

            return orders;
    }

    public String generateUniqueTrackingNumber() {
            // Generating UUID(Unique Universal Identifier)
            return UUID.randomUUID().toString();
    }

}
```

```java
package com.simplilearn.service;

import java.util.List;

import com.simplilearn.entity.Order;
import com.simplilearn.exception.OrderException;
import com.simplilearn.exception.ProductException;
import com.simplilearn.exception.UserException;
import com.simplilearn.model.OrderRequest;

public interface OrderService {
	String insertOrder(OrderRequest orderDTO) throws OrderException,
ProductException, UserException;

	Order updateOrder(OrderRequest orderDTO) throws OrderException, ProductException;

	Integer deleteOrder(Integer orderId) throws ProductException, OrderException;

	List<Order> getOrdersSortedByDateCreated();

	List<Order> getOrdersSortedByProductCategory();

	List<Order> getOrdersSortedByDateUpdated();
}
```

```
package com.simplilearn.service;


import com.simplilearn.entity.User;
import com.simplilearn.model.JwtRequest;
import com.simplilearn.model.JwtResponse;
import com.simplilearn.repository.UserRepository;
import com.simplilearn.util.JwtUtil;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.DisabledException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;


import java.util.HashSet;
import java.util.Set;


@Service
public class JwtService implements UserDetailsService {


    @Autowired
    private JwtUtil jwtUtil;


    @Autowired
    private UserRepository userDao;


    @Autowired
    private AuthenticationManager authenticationManager;


    // This method will create JWT token while Authenticating
    public JwtResponse createJwtToken(JwtRequest jwtRequest) throws Exception {
```

```
    String userName = jwtRequest.getUserName();
    String userPassword = jwtRequest.getUserPassword();
    authenticate(userName, userPassword);

    UserDetails userDetails = loadUserByUsername(userName);
    String newGeneratedToken = jwtUtil.generateToken(userDetails);

    User user = userDao.findById(userName).get();
    return new JwtResponse(user, newGeneratedToken);
}

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userDao.findById(username).get();

    if (user != null) {
        return new org.springframework.security.core.userdetails.User(
                user.getUserName(),
                user.getUserPassword(),
                getAuthority(user)
        );
    } else {
        throw new UsernameNotFoundException("User not found with username: " + username);
    }
}

private Set<SimpleGrantedAuthority> getAuthority(User user) {
    Set<SimpleGrantedAuthority> authorities = new HashSet<>();
    user.getRole().forEach(role -> {
        authorities.add(new SimpleGrantedAuthority("ROLE_" + role.getRoleName()));
    });
    return authorities;
}

private void authenticate(String userName, String userPassword) throws Exception {
    try {
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(userName, userPassword));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

## <u>Util</u>

```
package com.simplilearn.util;


import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Component;


import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;


@Component

public class JwtUtil {


    private static final String SECRET_KEY = "learn_programming_yourself";


    private static final int TOKEN_VALIDITY = 3600 * 1;


    public String getUsernameFromToken(String token) {

        return getClaimFromToken(token, Claims::getSubject);

    }


    public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

        final Claims claims = getAllClaimsFromToken(token);

        return claimsResolver.apply(claims);

    }


    private Claims getAllClaimsFromToken(String token) {

        return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();

    }


    public Boolean validateToken(String token, UserDetails userDetails) {

        final String username = getUsernameFromToken(token);

        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public String generateToken(UserDetails userDetails) {

        Map<String, Object> claims = new HashMap<>();

        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + TOKEN_VALIDITY * 1000))
                .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
                .compact();
    }
}
```