**Complete Case Study: E-Commerce System with Spring Security**

In this case study, we will design a **Spring Boot application** with **Spring Data JPA**, **MySQL**, and **Spring Security**, featuring a basic **e-commerce system**. We will go through the problem statement, objectives, and the various layers of the application, explaining how the entities interact, and how security is implemented.

**Problem Statement**

You are tasked with creating an **e-commerce system** with the following entities:

1. **Customer**: Users who can place orders.
2. **Product**: Items available for purchase.
3. **Order**: Represents a customer order, including a list of products.
4. **Payment**: Represents the payment made for an order.

The application should allow customers to:

* View products.
* Place orders.
* Make payments.

Additionally, the system should restrict access based on user roles using **Spring Security**.

**Objectives**

1. **Design Database Models**: Create entities and their relationships.
2. **Implement REST API Endpoints**: Create controllers to handle CRUD operations.
3. **Add Spring Security**: Implement basic security to protect the application and allow only authenticated users to perform certain actions.
4. **Implement MySQL Integration**: Use MySQL as the database to store data.
5. **Test the Application**: Ensure security and functionality by running integration tests.

**Entities and Relationships**

1. **Customer**: A user in the system who can place orders.
   * **One-to-Many** relationship with **Orders**.
2. **Product**: An item available for purchase.
   * **Many-to-Many** relationship with **Orders**.
3. **Order**: A record of products ordered by a customer.
   * **Many-to-One** relationship with **Customer**.
   * **Many-to-Many** relationship with **Product**.
4. **Payment**: A payment made by a customer for an order.
   * **One-to-One** relationship with **Order**.

**Entities Definitions**

**1. Customer Entity**

java

CopyEdit

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password;

@OneToMany(mappedBy = "customer")

private List<Order> orders;

// Getters and Setters

}

**2. Product Entity**

java

CopyEdit

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private Double price;

@ManyToMany

@JoinTable(

name = "order\_product",

joinColumns = @JoinColumn(name = "product\_id"),

inverseJoinColumns = @JoinColumn(name = "order\_id"))

private List<Order> orders;

// Getters and Setters

}

**3. Order Entity**

java

CopyEdit

@Entity

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "customer\_id")

private Customer customer;

@ManyToMany(mappedBy = "orders")

private List<Product> products;

@OneToOne(mappedBy = "order")

private Payment payment;

private LocalDateTime orderDate;

// Getters and Setters

}

**4. Payment Entity**

java

CopyEdit

@Entity

public class Payment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@OneToOne

@JoinColumn(name = "order\_id")

private Order order;

private Double amount;

private String status;

private LocalDateTime paymentDate;

// Getters and Setters

}

**Repositories**

java

CopyEdit

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

Optional<Customer> findByEmail(String email);

}

@Repository

public interface ProductRepository extends JpaRepository<Product, Long> {

Optional<Product> findByName(String name);

}

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {

List<Order> findByCustomer(Customer customer);

}

@Repository

public interface PaymentRepository extends JpaRepository<Payment, Long> {

Optional<Payment> findByOrder(Order order);

}

**Services**

**Customer Service**

java

CopyEdit

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

public Optional<Customer> getCustomerByEmail(String email) {

return customerRepository.findByEmail(email);

}

}

**Product Service**

java

CopyEdit

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public Product createProduct(Product product) {

return productRepository.save(product);

}

public Optional<Product> getProductByName(String name) {

return productRepository.findByName(name);

}

}

**Order Service**

java

CopyEdit

@Service

public class OrderService {

@Autowired

private OrderRepository orderRepository;

@Autowired

private CustomerService customerService;

@Autowired

private ProductService productService;

public Order createOrder(Long customerId, List<Long> productIds) {

Optional<Customer> customer = customerService.getCustomerByEmail("customer@example.com");

if (!customer.isPresent()) {

throw new RuntimeException("Customer not found");

}

List<Product> products = productIds.stream()

.map(productId -> productService.getProductByName("product" + productId).get())

.collect(Collectors.toList());

Order order = new Order();

order.setCustomer(customer.get());

order.setProducts(products);

order.setOrderDate(LocalDateTime.now());

return orderRepository.save(order);

}

}

**Payment Service**

java

CopyEdit

@Service

public class PaymentService {

@Autowired

private PaymentRepository paymentRepository;

@Autowired

private OrderRepository orderRepository;

public Payment createPayment(Long orderId, Double amount) {

Optional<Order> order = orderRepository.findById(orderId);

if (!order.isPresent()) {

throw new RuntimeException("Order not found");

}

Payment payment = new Payment();

payment.setOrder(order.get());

payment.setAmount(amount);

payment.setStatus("Completed");

payment.setPaymentDate(LocalDateTime.now());

return paymentRepository.save(payment);

}

}

**Controllers**

**Customer Controller**

java

CopyEdit

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

return ResponseEntity.status(HttpStatus.CREATED).body(customerService.createCustomer(customer));

}

@GetMapping("/{email}")

public ResponseEntity<Customer> getCustomerByEmail(@PathVariable String email) {

return customerService.getCustomerByEmail(email)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

}

**Product Controller**

java

CopyEdit

@RestController

@RequestMapping("/products")

public class ProductController {

@Autowired

private ProductService productService;

@PostMapping

public ResponseEntity<Product> createProduct(@RequestBody Product product) {

return ResponseEntity.status(HttpStatus.CREATED).body(productService.createProduct(product));

}

}

**Order Controller**

java

CopyEdit

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired

private OrderService orderService;

@PostMapping

public ResponseEntity<Order> createOrder(@RequestParam Long customerId, @RequestBody List<Long> productIds) {

return ResponseEntity.status(HttpStatus.CREATED).body(orderService.createOrder(customerId, productIds));

}

}

**Payment Controller**

java

CopyEdit

@RestController

@RequestMapping("/payments")

public class PaymentController {

@Autowired

private PaymentService paymentService;

@PostMapping

public ResponseEntity<Payment> createPayment(@RequestParam Long orderId, @RequestBody Double amount) {

return ResponseEntity.status(HttpStatus.CREATED).body(paymentService.createPayment(orderId, amount));

}

}

**Spring Security Configuration**

java

CopyEdit

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/customers/\*\*").permitAll() // Allow public access to customer endpoints

.antMatchers("/orders/\*\*").hasRole("USER")

.antMatchers("/payments/\*\*").hasRole("ADMIN")

.anyRequest().authenticated()

.and().httpBasic();

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("user").password(passwordEncoder().encode("password")).roles("USER")

.and()

.withUser("admin").password(passwordEncoder().encode("admin")).roles("ADMIN");

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

**Database Configuration (application.properties)**

properties

CopyEdit

spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.show-sql=true

**Testing the Application**

1. **Create Customer**:
   * POST /customers
2. **Create Product**:
   * POST /products
3. **Create Order**:
   * POST /orders?customerId=1 with product list.
4. **Create Payment**:
   * POST /payments?orderId=1 with amount.

**Conclusion**

This e-commerce system allows **customers** to place **orders** and make **payments**. **Spring Security** restricts access to certain endpoints based on the user's role, and **Spring Data JPA** manages the database. The **MySQL** database stores all the necessary data, and the system is accessible via RESTful API endpoints.

This case study can be extended to include more advanced features, such as:

* **User authentication** using JWT or OAuth.
* **Order history** and status updates.
* **Email notifications** on order status.