PROGRAM STUDI INFORMATIKA
UNIVERSITAS SANATA DHARMA

PRAKTIKUM PBO LANJUT
MODUL SPRING SECURITY

I. TUJUAN PRAKTIKUM

1. Menerapkan **Spring Security** untuk autentikasi dan otorisasi di aplikasi Spring Boot.
2. Menggunakan **Spring Data JPA** untuk operasi **CRUD** pada **Product**, **Customer**, dan **Order**.
3. Menggunakan **Spring MVC** dan **Thymeleaf** untuk menampilkan data ke pengguna berdasarkan peran.
4. Mengelola perbedaan akses antara pengguna biasa dan admin.

II. DASAR TEORI

Spring Security adalah sebuah framework yang dirancang untuk menyediakan mekanisme autentikasi dan otorisasi yang kuat dalam aplikasi berbasis Java, termasuk Spring Boot. Ini adalah modul keamanan di ekosistem Spring yang menyediakan cara untuk mengamankan aplikasi web dan REST API dengan menambahkan lapisan perlindungan terhadap ancaman seperti akses tidak sah, serangan CSRF, dan serangan brute-force.

1. Fitur Utama Spring Security:

Autentikasi (Authentication):

- Proses memverifikasi identitas pengguna. Autentikasi dapat dilakukan menggunakan beberapa cara, seperti login berbasis form, login berbasis token (JWT), atau metode lainnya seperti OAuth2 dan SAML.

Otorisasi (Authorization):

- Proses menentukan apakah pengguna yang telah diautentikasi memiliki izin untuk mengakses sumber daya atau fitur tertentu dalam aplikasi, sering kali diterapkan menggunakan **role-based access control (RBAC)**.

Proteksi CSRF (Cross-Site Request Forgery):

- Spring Security menyediakan mekanisme untuk melindungi aplikasi dari serangan CSRF dengan menggunakan token yang disertakan di setiap permintaan POST.

Keamanan Berbasis URL dan Method:

- Spring Security memungkinkan pengaturan izin berdasarkan **URL** yang diakses atau bahkan pada level metode menggunakan anotasi.

Pengelolaan Session dan Remember-Me:

- Spring Security dapat mengelola session pengguna dan mendukung fitur **"Remember-Me"** yang memungkinkan pengguna tetap terautentikasi meskipun session asli telah berakhir.

Konsep Dasar Spring Security:

1. **Security Context:**
    - Setiap kali pengguna melakukan autentikasi, informasi autentikasi disimpan di dalam **Security Context**, yang digunakan untuk memeriksa izin dan akses pada setiap request yang dilakukan oleh pengguna.
2. **AuthenticationManager:**
    - Komponen inti yang bertanggung jawab untuk memproses autentikasi. **AuthenticationManager** memverifikasi kredensial pengguna dan mengembalikan objek autentikasi yang valid jika kredensial benar.
3. **Filter Chain:**
    - Spring Security menggunakan serangkaian filter yang dikenal sebagai **Security Filter Chain** untuk memproses setiap request dan menerapkan aturan keamanan sebelum request diteruskan ke controller.
4. **UserDetails dan UserDetailsService:**
    - **UserDetails** adalah antarmuka yang menyediakan informasi dasar pengguna yang diautentikasi, sedangkan **UserDetailsService** digunakan untuk mengambil informasi pengguna dari database atau sumber data lainnya.
5. **GrantedAuthority dan Role:**
    - **GrantedAuthority** adalah representasi izin yang diberikan kepada pengguna. Biasanya, otorisasi didasarkan pada **roles** (peran) seperti **ADMIN**, **USER**, dll.

Komponen Utama Spring Security di Spring Boot:

- **Security Configuration (`WebSecurityConfigurerAdapter` atau `SecurityFilterChain`):**
    - Mengatur konfigurasi keamanan aplikasi, termasuk aturan autentikasi, otorisasi, dan pengaturan filter.
    - Di Spring Boot, ini biasanya diimplementasikan melalui kelas konfigurasi yang menurunkan dari **WebSecurityConfigurerAdapter** (Spring Security < 5.7) atau menggunakan bean **SecurityFilterChain** (Spring Security 5.7+).
- **UserDetailsService dan UserDetails:**
    - Mengimplementasikan **UserDetailsService** untuk menyediakan detail pengguna yang dikelola oleh aplikasi, termasuk nama pengguna, password, dan role yang terkait.
    - **UserDetails** mengembalikan objek pengguna yang menyimpan informasi tentang pengguna yang berhasil diautentikasi.
- **Password Encoding:**
    - Spring Security menyarankan penggunaan algoritma hashing (seperti **BCrypt**) untuk menyimpan password secara aman. Dengan **PasswordEncoder**, password dapat di-hash sebelum disimpan ke database dan diverifikasi saat autentikasi.
- **JWT (JSON Web Token):**
    - Dalam aplikasi berbasis REST API, Spring Security sering digunakan bersama **JWT** untuk memberikan autentikasi berbasis token.

- JWT digunakan untuk menghindari state management di server dan mendukung komunikasi tanpa status (stateless) dengan API.

- **Otorisasi Berbasis Anotasi:**
  - Spring Security mendukung anotasi seperti `@PreAuthorize`, `@Secured`, dan `@RolesAllowed` untuk menerapkan otorisasi pada level metode.

III. LATIHAN

Langkah 1: Konfigurasi Awal Proyek Spring Boot

1.1. Buat Proyek dengan Spring Initializr:

1. Buka Spring Initializr dan buat proyek Spring Boot.
   - **Project:** Maven
   - **Language:** Java
   - **Spring Boot Version:** 2.7.x atau lebih baru
   - **Dependencies:**
     - Spring Web
     - Spring Data JPA
     - H2 Database
     - Thymeleaf
     - Spring Security
2. **Generate Project** dan buka di IDE

1.2. Konfigurasi `application.properties`:

Konfigurasikan aplikasi untuk menggunakan H2 Database dan Spring Security.

```
# Konfigurasi H2 Database
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update

# Mengaktifkan H2 console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Langkah 2: Implementasi CRUD dengan Spring Data JPA

2.1. Buat Model `Product`, `Customer`, dan `Order`:

- `Product.java`:

```
import jakarta.persistence.*;
import java.util.List;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
    private String name;
    private double price;

    // Getters and Setters
}
```

Customer.java:

```
import jakarta.persistence.*;

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
}
```

Order.java:

```
import jakarta.persistence.*;
@Entity
public class Orders {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Customer customer;

    @ManyToOne
    private Product product;

    // Getters and Setters
}
```

2.2. Buat Repository untuk Product, Customer, dan Order:

ProductRepository.java:

```
import
org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface ProductRepository extends
JpaRepository<Product, Long> {
}
```

CustomerRepository.java:

```
import
org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends
JpaRepository<Customer, Long> {
}
```

OrderRepository.java:

```
import
org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepository extends
JpaRepository<Order, Long> {
}
```

2.3. Buat Layanan CRUD (Service) untuk Product, Customer, dan Order:

ProductService.java:

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

import java.util.List;

@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;

    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    public Product saveProduct(Product product) {
        return productRepository.save(product);
    }
     public Product getProductById(Long id) {
             Optional<Product> optionalProduct =
     productRepository.findById(id);
             if (optionalProduct.isPresent()) {
                 return optionalProduct.get();
             } else {
```

```
                throw new RuntimeException("Product not
     found for id :: " + id);
            }
        }

    // Menghapus produk berdasarkan ID
    public void deleteProduct(Long id) {
        productRepository.deleteById(id);
    }
}
```

CustomerService.java:

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

import java.util.List;
import java.util.Optional;

@Service
public class CustomerService {

    @Autowired
    private CustomerRepository customerRepository;

    // Mengambil semua data customer
    public List<Customer> getAllCustomers() {
        return customerRepository.findAll();
    }

    // Menyimpan atau memperbarui customer
    public Customer saveCustomer(Customer customer) {
        return customerRepository.save(customer);
    }

    // Mengambil customer berdasarkan ID
    public Customer getCustomerById(Long id) {
        Optional<Customer>        optionalCustomer        =
customerRepository.findById(id);
        return optionalCustomer.orElse(null);
    }

    // Menghapus customer berdasarkan ID
    public void deleteCustomer(Long id) {
        customerRepository.deleteById(id);
    }
}
```

OrderService.java:

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

import java.util.List;
import java.util.Optional;

@Service
public class OrderService {

    @Autowired
    private OrderRepository orderRepository;

    // Mengambil semua data order
    public List<Orders> getAllOrders() {
        return orderRepository.findAll();
    }

    // Menyimpan atau memperbarui order
    public Orders saveOrder(Orders order) {
        return orderRepository.save(order);
    }

    // Mengambil order berdasarkan ID
    public Order getOrderById(Long id) {
        Optional<Orders>           optionalOrder           =
orderRepository.findById(id);
        return optionalOrder.orElse(null);
    }

    // Menghapus order berdasarkan ID
    public void deleteOrder(Long id) {
        orderRepository.deleteById(id);
    }
}
```

Langkah 3: Implementasi Spring Security

3.1. Konfigurasi Spring Security:

Buat konfigurasi Spring Security untuk mengatur autentikasi dan otorisasi pengguna dan admin. Letakkan dalam package com.example.demo.config (sesuaikan)

SecurityConfig.java:

```
package com.example.demo.config;

import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authenticati
on.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.authenticati
on.configuration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.web.builders
.HttpSecurity;
import
org.springframework.security.config.annotation.web.configur
ation.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsSe
rvice;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEn
coder;
import
org.springframework.security.crypto.password.PasswordEncode
r;
import
org.springframework.security.provisioning.InMemoryUserDetai
lsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public                                SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorize -> authorize

.requestMatchers("/admin/**").hasRole("ADMIN")            //
Mengamankan URL /admin/**

.requestMatchers("/user/**").hasRole("USER")              //
Mengamankan URL /user/**
                .requestMatchers("/",  "/login").permitAll()
// Mengizinkan akses tanpa login
```

```
                .anyRequest().authenticated()                //
Memerlukan autentikasi untuk URL lain
            )
            .formLogin(form -> form
                .loginPage("/login")                         //
Mengatur halaman login khusus
                .permitAll()                                 //
Mengizinkan akses tanpa login
            )
            .logout(logout -> logout
                .permitAll()                                 //
Mengizinkan semua pengguna logout
            );
        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails admin = User.builder()
                .username("admin")

.password(passwordEncoder().encode("admin123"))
                .roles("ADMIN")
                .build();

        UserDetails user = User.builder()
                .username("user")

.password(passwordEncoder().encode("user123"))
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(admin, user);
    }

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

3.2. Tambahkan Otentikasi Berbasis In-Memory:
Pada konfigurasi di atas, pengguna **admin** dan **user** didefinisikan dengan password yang di-hash menggunakan **BCrypt**.
Role-based Authorization: Mengatur akses berbasis peran untuk endpoint **admin** dan **user**.
`formLogin()`: Mengaktifkan login berbasis form untuk autentikasi pengguna.

Langkah 4: Membuat Halaman Web untuk User dan Admin

4.1. Buat Halaman Web untuk Login:

Buat file **`login.html di folder templates`**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Login Page</title>
</head>
<body>
    <h1>Login</h1>
    <form th:action="@{/login}" method="post">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username"/>
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password"/>
        </div>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

4.2. Buat Controller untuk memastikan bahwa URL `/login` terhubung dengan halaman `login.html` yang ada di folder templates

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping("/login")
    public String login() {
        return "login";  // Mengembalikan login.html di
templates
    }
}
```

4.3. Buat Halaman Web untuk menampilkan Produk:

Buat file **`product-list.html`** di folder **`src/main/resources/templates`**:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Product List</title>
</head>
```

```
<body>
    <h1>Product List</h1>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Price</th>
            <th>Actions</th>
        </tr>
        <tr th:each="product : ${products}">
            <td th:text="${product.id}">ID</td>
            <td th:text="${product.name}">Name</td>
            <td th:text="${product.price}">Price</td>
            <td>
                <a
th:if="${#authorization.expression('hasRole(''ADMIN'')')}"

th:href="@{/product/edit/{id}(id=${product.id})}">Edit</a> |
                <a
th:if="${#authorization.expression('hasRole(''ADMIN'')')}"

th:href="@{/product/delete/{id}(id=${product.id})}"
                   onclick="return confirm('Are you sure you
want to delete this product?')">Delete</a>
            </td>
        </tr>
    </table>
    <a
th:if="${#authorization.expression('hasRole(''ADMIN'')')}"
href="/product/add">Add New Product</a>
</body>
</html>
```

4.2. Buat Controller untuk Mengelola Produk:

ProductController.java:

```
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
```

```
    private ProductService productService;

    @GetMapping("/list")
    @PreAuthorize("hasAnyRole('USER', 'ADMIN')")
    public String getAllProducts(Model model) {
        model.addAttribute("products",
productService.getAllProducts());
        return "product-list";
    }

    @GetMapping("/edit/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public String editProductForm(@PathVariable Long id,
Model model) {
        try {
            Product product =
productService.getProductById(id);
            model.addAttribute("product", product);
            return "product-edit";
        } catch (RuntimeException e) {
            model.addAttribute("error", "Product not found
for id :: " + id);
            return "product-list";   // Arahkan kembali ke
daftar produk jika produk tidak ditemukan
        }
    }

    @PostMapping("/edit/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public String editProduct(@PathVariable Long id,
@ModelAttribute("product") Product product) {
        product.setId(id);
        productService.saveProduct(product);
        return "redirect:/product/list";
    }
}
```

4.3. File Tambah Produk (`product-add.html`)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Product</title>
</head>
<body>
    <h1>Add New Product</h1>
    <form th:action="@{/product/add}" method="post">
        <div>
            <label for="name">Product Name:</label>
            <input type="text" id="name" name="name" />
        </div>
```

```
        <div>
            <label for="price">Price:</label>
            <input type="text" id="price" name="price" />
        </div>
        <button type="submit">Add Product</button>
    </form>
</body>
</html>
```

4.4. File Edit Produk (`product-edit.html`)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Product</title>
</head>
<body>
    <h1>Edit Product</h1>
    <form th:action="@{/product/edit/{id}(id=${product.id})}"
method="post">
        <div>
            <label for="name">Product Name:</label>
            <input type="text" id="name" name="name"
th:value="${product.name}" />
        </div>
        <div>
            <label for="price">Price:</label>
            <input type="text" id="price" name="price"
th:value="${product.price}" />
        </div>
        <button type="submit">Save Changes</button>
    </form>
</body>
</html>
```

Langkah 5: Uji Coba Aplikasi dan Penyempurnaan (10 Menit)

5.1. Uji Login dan Akses Halaman:

1. **Buka browser dan akses halaman login**:
    - URL: **`http://localhost:8080/login`**
    - Coba login dengan pengguna berikut:
        - **Admin**:
            - Username: `admin`
            - Password: `admin123`
        - **User**:
            - Username: `user`
            - Password: `user123`
2. **Login Sebagai ADMIN:**

- Masukkan **username**: `admin` dan **password**: `admin123`.
- Setelah login, Anda dapat mengakses semua halaman berikut:
    - **Daftar Produk:** `http://localhost:8080/product/list`
    - **Tambah Produk:** `http://localhost:8080/product/add`
    - **Edit Produk:** `http://localhost:8080/product/edit/{id}`

- **Hapus Produk:** `http://localhost:8080/product/delete/{id}`

3. **Login Sebagai USER:**

- Masukkan **username**: `user` dan **password**: `user123`.
- Setelah login, Anda hanya dapat mengakses:
  - **Daftar Produk:** `http://localhost:8080/product/list`
- Anda tidak dapat mengakses halaman tambah, edit, atau hapus produk karena otorisasi tidak sesuai.

TUGAS

1. Membuat Halaman Order dan Customer untuk Pengguna:

Tambahkan halaman web untuk mengelola **order** dan **customer** dari sisi pengguna.

2.Membuat Controller untuk Mengelola Order dan Customer:

3. Uji Kembali Aplikasi untuk akses Order dan Customer:

1. **Login sebagai admin**:
2. **Login sebagai user**:
3. **Akses H2 Console untuk Memeriksa Data:**

IV. LAPORAN

1. Penjelasan kode program, capture GUI untuk setiap soal (9 x (@10%)

Referensi Utama:

1. Craig Walls, "Spring in Action," 6th Edition, Manning Publications, 2022.
2. Juergen Hoeller, "Spring Framework Reference Documentation," Spring.io.