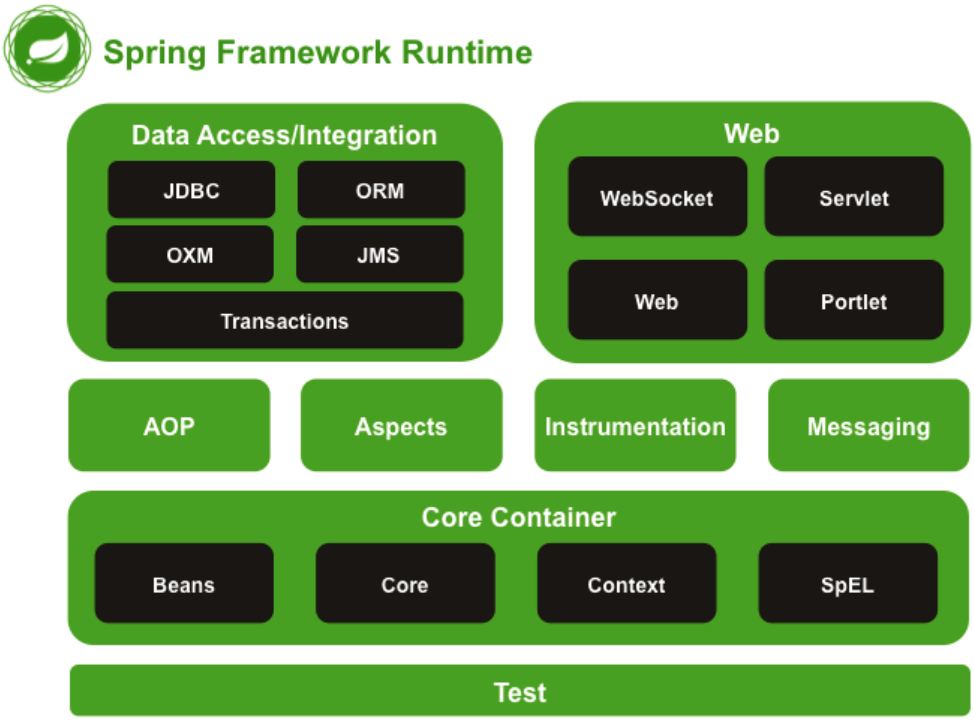
DESARROLLO WEB
CON SPRING BOOT



UNIDAD 11
SPRING BOOT SECURITY

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
gcoronelc@gmail.com
I N S T R U C T O R

CONTENIDO

CONTEXTO

Figura 1 Un usuario piensa principalmente en los requisitos funcionales. A veces, es posible que los vea conscientes también del rendimiento, que no es funcional, pero desafortunadamente es inusual que se preocupen por la seguridad. Los requisitos no funcionales tienden a ser más transparentes que los funcionales.

Hoy en día, cada vez más desarrolladores son conscientes de la seguridad. Desafortunadamente, no es una práctica común asumir la responsabilidad de la seguridad desde el comienzo del desarrollo de una aplicación de software. Esta actitud debe cambiar y todos los involucrados en el desarrollo de un sistema de software deben aprender a considerarlo desde el principio.

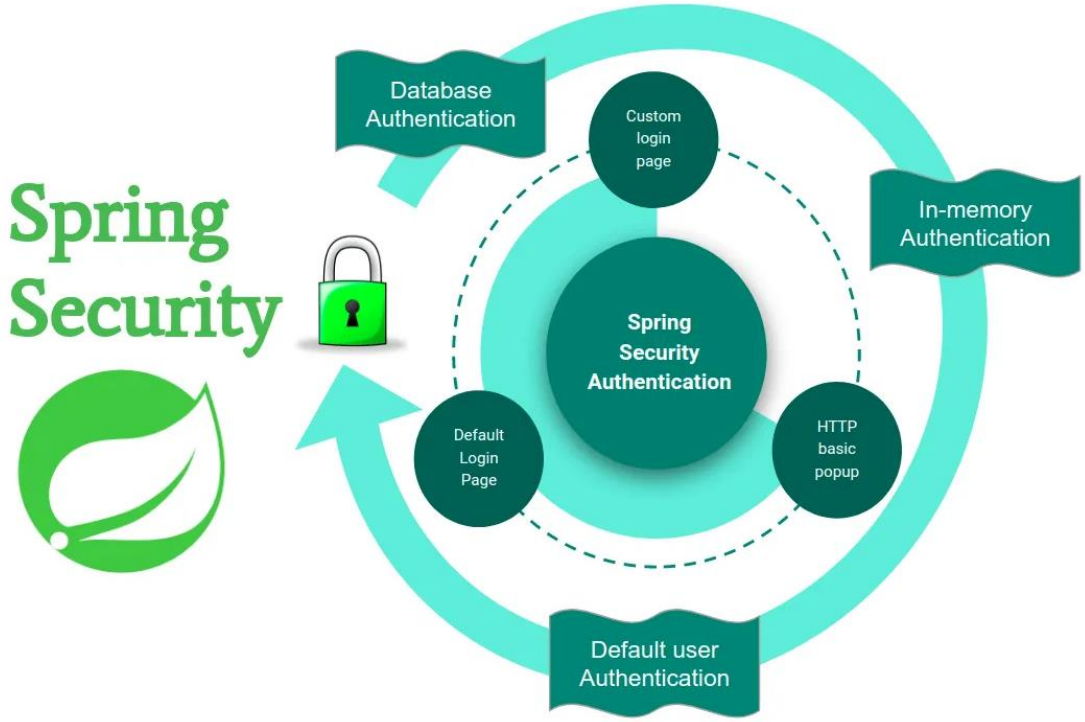
Generalmente, como desarrolladores, comenzamos aprendiendo que el propósito de una aplicación es resolver casos de negocios. Este propósito se refiere a algo donde los datos podrían procesarse de alguna manera, persistir y, finalmente, mostrarse al usuario de una manera específica según lo especificado por algunos requisitos. Esta visión general del desarrollo de software, que de alguna manera se impone desde las primeras edades del desarrollo, tiene la lamentable desventaja de ocultar prácticas que también forman parte del

proceso. Si bien la aplicación funciona correctamente desde la perspectiva del usuario, y al final, hace lo que el usuario espera en cuanto a funcionalidades, hay muchos aspectos ocultos en el resultado final.

Las características no funcionales del software, como el rendimiento, la escalabilidad, la disponibilidad y por supuesto, la seguridad, entre otras, pueden tener un impacto a lo largo del tiempo, puede ser a corto o largo plazo. Si no se tienen en cuenta desde el principio, estas características pueden afectar drásticamente en términos de rentabilidad a los propietarios de la aplicación. Además, podrían desencadenar fallas en otros sistemas (por ejemplo, por la participación involuntaria en un ataque distribuido de denegación de servicio (DDoS)). El aspecto oculto de los requisitos no funcionales (el hecho de que es mucho más difícil ver si faltan o están incompletos) los hace, sin embargo, más peligrosos. (Ver Figura 1)

Tiene que considerar varios aspectos no funcionales cuando trabajas en un software. En la práctica, todos estos son importantes y deben tratarse de manera responsable en el proceso de desarrollo de software. En esta separata aprenderás a proteger tu aplicación aplicando Spring Security.

SPRING BOOT SECURITY

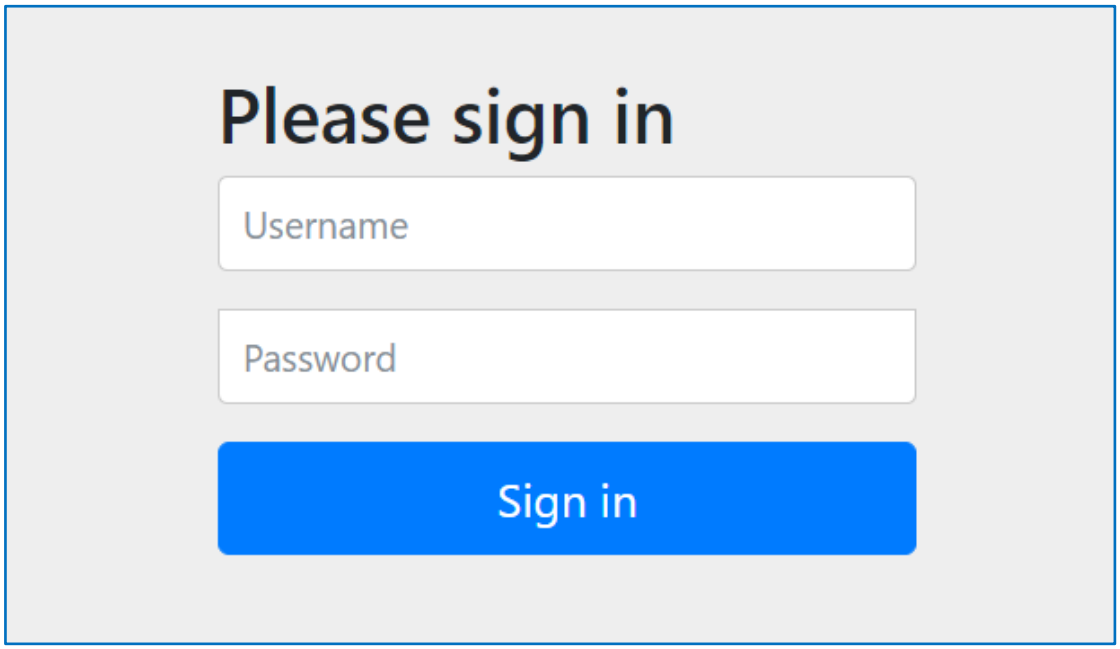


Aquí encontraras ejemplos de autenticación de seguridad de Spring Boot. Para la autenticación de la página de inicio de sesión predeterminada, la ventana emergente http básica o la página de inicio de sesión personalizada se pueden configurar fácilmente en Spring Security usando Spring Boot. Los detalles del usuario se pueden proporcionar desde la base de datos, en la memoria o incluso desde el archivo de propiedades.

La autenticación de seguridad de Spring Boot se habilita mediante la anotación `@EnableWebSecurity`. La clase abstracta `WebSecurityConfigurerAdapter` se extiende y el método de configuración se anula, lo que aplica la seguridad a los puntos finales de la aplicación. En el método de configuración, se especifica el tipo de autenticación de seguridad de Spring Boot para evitar el acceso no autenticado. Se implementa un filtro y se conecta en `httpsecurity` para inyectar el servicio de usuario para proporcionar detalles de usuario para autenticar y autorizar.

Spring Security es un marco de seguridad que permite a un desarrollador agregar restricciones de seguridad a las aplicaciones basadas en Web, así como a las aplicaciones basadas en Rest. La principal responsabilidad de Spring Security es autenticar y autorizar a las solicitudes entrantes, acceder a cualquier recurso que puede ser un punto final de Rest API, URL MVC, recurso estático, etc. Spring Security es un marco muy poderoso y proporciona un alto nivel de personalización.

CONFIGURACIÓN



La dependencia spring-boot-starter-security es el proyecto de Spring Boot que incluye todas las dependencias necesarias para la configuración de Spring Security. Ayuda mucho a un desarrollador a eliminar el boilerplate code y proporcionar todos los valores de configuración predeterminados. Un desarrollador puede personalizar la seguridad de spring fácilmente y puede concentrarse mejor en la lógica principal de la aplicación, sin preocuparse por configurar todas y cada una de las partes de la Spring Security.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

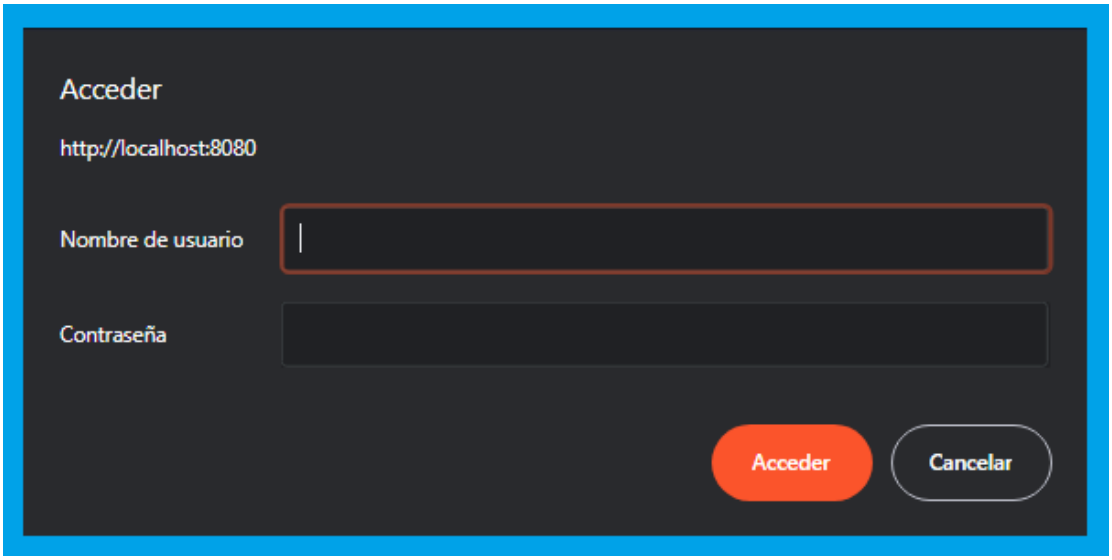
En este momento ya puedes probar la aplicación, tendrás un formulario por defecto de Spring Security, el usuario por defecto es `user` y la clave la tienes en la consola.

Puedes configurar tus propias credenciales de autenticación en el archivo `application.properties`:

```
spring.security.user.name=gustavo
spring.security.user.password=123456
spring.security.user.roles=ADMIN
```

Para este caso, ya no se genera una contraseña en la consola.

VENTANA EMERGENTE DE AUTENTICACIÓN



La ventana emergente de autenticación es una forma tradicional y fácil de autenticarse. Si solo tiene un usuario de inicio de sesión, puede usar archivos de propiedades para guardar las credenciales del usuario directamente. No es necesario implementar una base de datos o un proveedor de autenticación en memoria.

Los siguientes son los pasos para implementar la seguridad http en la aplicación Spring Boot:

1. Crea una clase (llamemos a esta clase `WebSecurityConfig`, pero puedes darle otro nombre) que extienda de la clase abstracta `WebSecurityConfigurerAdapter`.
2. A la clase `WebSecurityConfig`, agregue dos anotaciones: `@Configuration` y `@EnableWebSecurity`.
3. En `WebSecurityConfig`, debes sobre escribir el método de configuración para configurar `HttpSecurity` para la aplicación.
4. Ahora debes configurar la seguridad `HttpSecurity` para autenticar todas las solicitudes y aplique la autenticación `HttpBasic` usando el método `httpBasic()`.

5. Tu clase se quedar así:

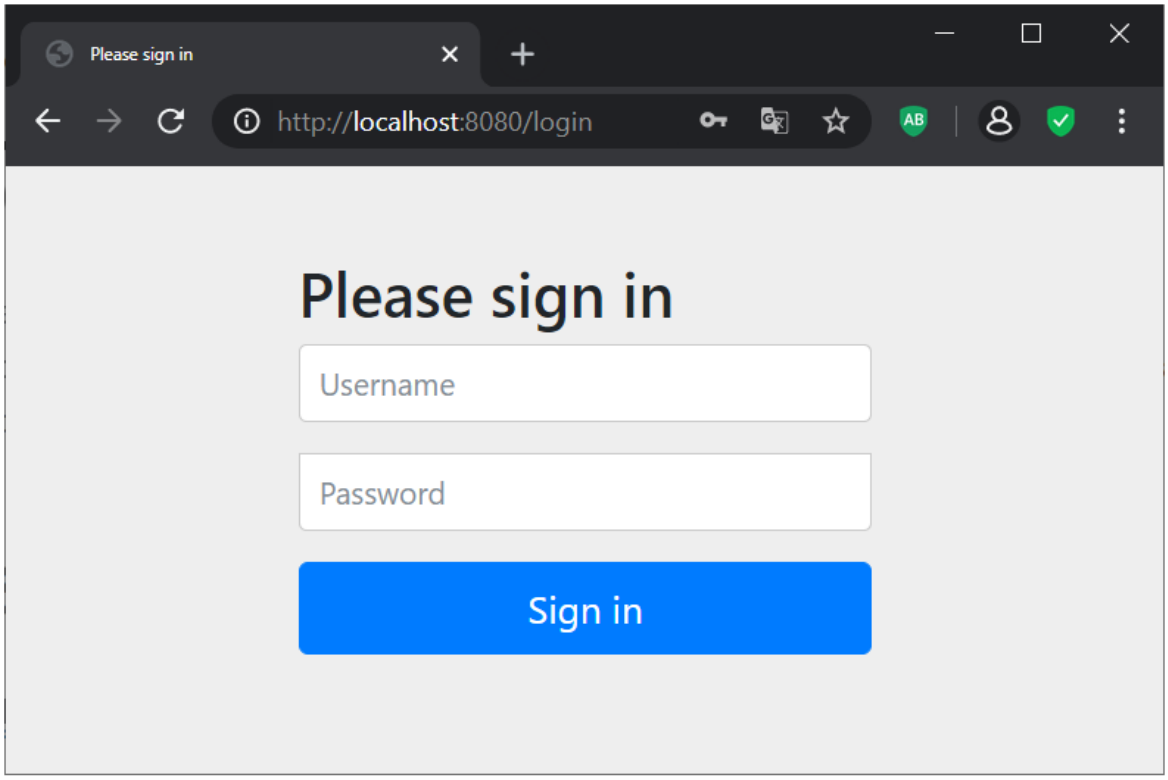
```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests()
         .anyRequest()
         .authenticated()
         .and()
         .httpBasic();
   }

}
```

6. Ahora es el momento que proporcionar las credenciales a la capa de seguridad de Spring, para que pueda comparar y autenticar las credenciales del usuario. En `application.properties` debes agregar las credenciales, aquí tienes un ejemplo:

```
spring.security.user.name=gustavo
spring.security.user.password=123456
spring.security.user.roles=ADMIN
```

7. Debes crear un controlador y una vista para que puedas probar la configuración.
8. La instalación está completa. Ahora ejecuta la aplicación y ábrala en el navegador, la dirección predeterminada es: `http://localhost:8080`
9. Al abrir la aplicación en el navegador, verás una ventana emergente en la que debes ingresar las credenciales correctas para tener acceso a la vista.

AUTENTICACIÓN EN MEMORIA DEL FORMULARIO DE INICIO DE SESIÓN POR DEFECTO



La página de inicio de sesión predeterminada de Spring Security es una página incorporada que es proporcionada por el proyecto Spring Boot Security. Esta página mapea automáticamente el nombre de usuario y la contraseña, y proporciona una autenticación sencilla. Todos los mensajes de error y las opciones de cierre de sesión son manejados por esta página.

La autenticación en memoria es el mecanismo para mantener las credenciales de usuario en la propia memoria JVM, pero guardando en un archivo o en una base de datos. Si está tratando de probar algo en Spring Boot o crear algún tipo de prueba de concepto, generalmente se usa la autenticación en memoria.

Los siguientes son los pasos para implementar la página de inicio de sesión predeterminada de seguridad utilizando la autenticación de memoria:

1. Debe extender la clase abstracta `WebSecurityConfigurerAdapter`. Por ejemplo, crea la clase `WebSecurityConfig` para este propósito.

2. La clase `WebSecurityConfig` debe tener anotaciones `@Configuration` y `@EnableWebSecurity`.

3. En `WebSecurityConfig`, debes configurar HttpSecurity para autenticar todas las solicitudes entrantes y habilitar la página de inicio de sesión predeterminada de Spring Security.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests()
   .anyRequest()
   .authenticated()
   .and()
   .formLogin();
}
```

4. Ya que estas utilizando la autenticación en memoria. Se recomienda utilizar un codificador de contraseñas para guardar las contraseñas en la memoria o en la base de datos. Spring Security tiene soporte incorporado para `BCryptPasswordEncoder`. Entonces puedes crear un bean y usarlo directamente.

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

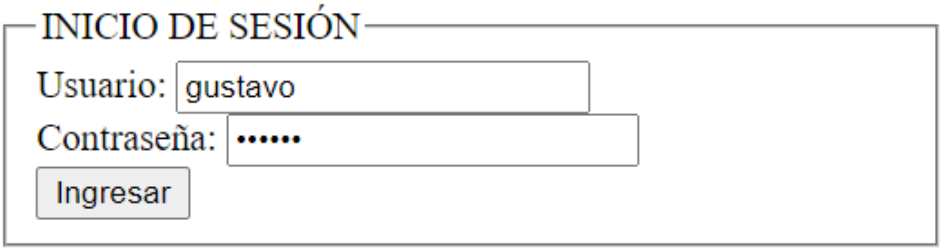
5. En `WebSecurityConfig`, debes configurar `AuthenticationManagerBuilder` para configurar credenciales en memoria.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
   auth.inMemoryAuthentication()
   .passwordEncoder(new BCryptPasswordEncoder())
      .withUser("gustavo")
      .password(passwordEncoder().encode("123456"))
      .roles("ADMIN")
   .and()
      .withUser("reader")
      .password(passwordEncoder().encode("123456"))
      .roles("USER");
}
```

10. Ahora ejecuta la aplicación y ábrala en el navegador, la dirección predeterminada es:
    `http://localhost:8080`

11. Ya puedes hacer las pruebas de las credenciales.

INICIO DE SESIÓN PERSONALIZADO Y AUTENTICACIÓN EN MEMORIA



La página de inicio de sesión personalizada de Spring Security es una página creada en su totalidad por el desarrollador y personalizada según sus propias necesidades. Es muy común que necesite tener una página de inicio de sesión personalizada para mostrar su propia marca, opciones personalizadas para iniciar sesión o alguna otra información para mostrar en la página de inicio de sesión.

En este ejemplo, implementaras una página de inicio de sesión personalizada utilizando Thymeleaf.

Los siguientes son los pasos para implementar la seguridad de arranque de Spring con una página de inicio de sesión personalizada con autenticación en memoria y Thymeleaf.

1. En primer lugar, agregue las dependencias necesarias.
2. Los pasos del codificador de contraseña y de autenticación en memoria seguirán siendo los mismos que en el ejemplo anterior.
3. Crea una página de inicio (home.html) en el directorio resources/templates:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>APP DEMO</title>
</head>
<body style="text-align: center;">
  <h1>Bienvenido al ejemplo de Spring Boot Security.</h1>
  <a href="/logout">Salir</a>
</body>
</html>
```

4. Crea una página de inicio de sesión (login.html) en el directorio resources/templates.

```
<html xmlns:th="https://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>INICIO DE SESIÓN</title>
</head>
<body>
  <div style="width:350px; margin: 10 auto;">
     <form name="f" th:action="@{/login}" method="post">
        <fieldset>
           <legend>INICIO DE SESIÓN</legend>
           <div th:if="${param.error}" class="alert alert-error">Usuario
              y contraseña incorrectos.</div>
           <div th:if="${param.logout}" class="alert alert-success">Se ha
              cerrado la sesión, vuelva a iniciar sesión.</div>
           <div>
              <label for="username">Usuario: </label> <input type="text"
                 id="username" name="username" value="gustavo" />
           </div>
           <div>
              <label for="password">Contraseña: </label> <input type="password"
                 id="password" name="password" value="123456" />
           </div>
           <div class="form-actions">
              <button type="submit" class="btn">Ingresar</button>
           </div>
        </fieldset>
     </form>
  </div>
</body>
</html>
```

5. Crea un controlador para mapear la página de inicio y la página de inicio de sesión.

```
@Controller
public class AppController {

  @GetMapping("/")
  public ModelAndView home(Principal principal) {
     return (new ModelAndView("home")).addObject("principal", principal);
  }

  @GetMapping("/login")
  public ModelAndView login() {
     return new ModelAndView("login");
  }

}
```

6. Ahora es el momento que configures `HttpSecurity` en la clase `WebSecurityConfig`. Después del método `formLogin ()`, debes agregar un método `loginPage` y proporcione la `uri` de la página de inicio de sesión. Dado que todos pueden ver la página de inicio de sesión para ingresar credenciales, en `loginPage` debes agregar el método `permitAll()`.

```
  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.authorizeRequests()
        .antMatchers("/resources/**").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
          .loginPage("/login").permitAll()
        .and()
          .logout().permitAll()
          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
  }
```

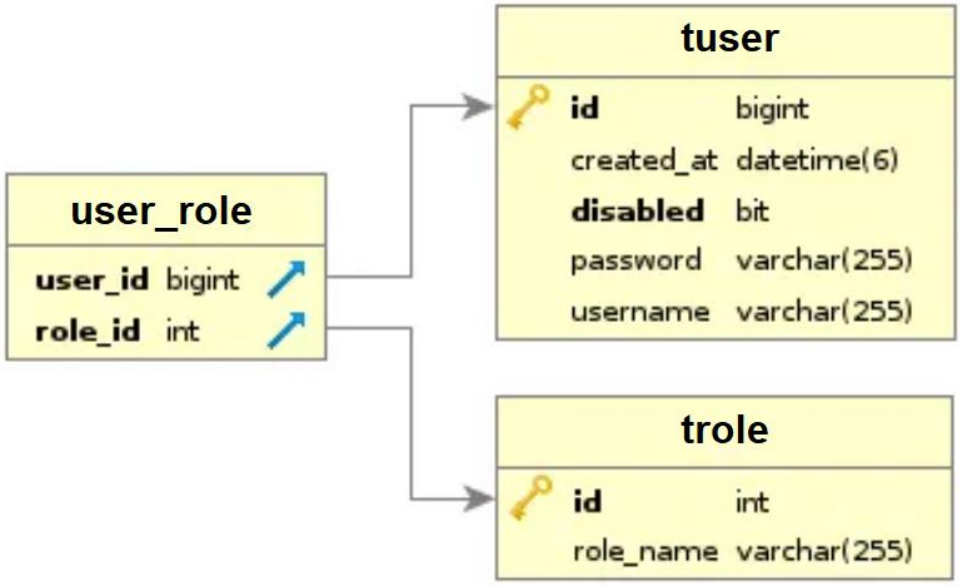
7. En la configuración anterior, se manejado el cierre de sesión para que sea redirigido a la página de inicio de sesión con un mensaje de cierre de sesión. La última línea del código anterior representa esta configuración.

```
  .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
```

12. Ahora ejecuta la aplicación y ábrala en el navegador, la dirección predeterminada es: `http://localhost:8080`

13. Ya puedes hacer las pruebas de las credenciales.

INICIO DE SESIÓN PERSONALIZADO CON BASE DE DATOS



La seguridad de Spring Boot con autenticación de base de datos es la forma preferida en las aplicaciones estándar. La base de datos puede ser cualquiera, pero el enfoque de implementación en la seguridad de Spring Boot sigue siendo el mismo.

En este ejemplo, implementaras un inicio de sesión personalizado con autenticación de base de datos Oracle.

Los siguientes son los pasos para el ejemplo de autenticación de Spring Boot con base de datos:

1. El inicio de sesión personalizado lo implementaste en el ejemplo anterior. Todos los pasos siguen siendo los mismos, por lo que puede consultarlos nuevamente.
2. Para la autenticación de la base de datos, debe agregar las dependencias necesarias.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
   <groupId>com.oracle.database.jdbc</groupId>
   <artifactId>ojdbc8</artifactId>
   <scope>runtime</scope>
</dependency>
```

3. En el archivo `application.propeties` debes agregar las credenciales de la base de datos. Deja que hibernar cree automáticamente las tablas de su base de datos.

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/XE
spring.datasource.username=demo
spring.datasource.password=demo
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

4. Todo usuario que inicia sesión debe tener un rol. Bebes crear una tabla de roles donde se agregarán todos los roles, que se pueden asignar al usuario. Crea la clase de `Role` de la siguiente manera:

```
@Entity
@Table(name = "TRole")
public class Role {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE,
     generator = "SEQ_ROLE")
  @SequenceGenerator(name = "SEQ_ROLE",
     sequenceName = "SEQ_ROLE", allocationSize = 1)
  private Long id;
  private String roleName;

  public Role() {
  }

  public Role(String roleName) {
     this.roleName = roleName;
  }

  public Long getId() {
     return id;
  }

  public void setId(Long id) {
     this.id = id;
```

```
  }

  public String getRoleName() {
     return roleName;
  }

  public void setRoleName(String roleName) {
     this.roleName = roleName;
  }

}
```

5. Necesitas una clase de dominio de usuario, que se asignará a una tabla en la base de datos. Crea la clase Use5 de la siguiente manera:

```
@Entity
@Table(name = "TUser")
public class User {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE,
     generator = "SEQ_USER")
  @SequenceGenerator(name = "SEQ_USER",
     sequenceName = "SEQ_USER", allocationSize = 1)
  private Long id;
  private String username;
  private String password;
  private boolean disabled = false;
  private LocalDateTime createdAt = LocalDateTime.now();
  @ManyToMany(fetch = FetchType.EAGER)
  @JoinTable(name = "user_role",
     joinColumns = @JoinColumn(name = "user_id"),
     inverseJoinColumns = @JoinColumn(name = "role_id"))
  private List<Role> roles;

  public User() {
  }

  public User(String username, String password, Role role) {
     this.username = username;
     this.password = password;
     roles = new ArrayList<>();
```

```
        roles.add(role);
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public boolean isDisabled() {
        return disabled;
    }

    public void setDisabled(boolean disabled) {
        this.disabled = disabled;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
```

```
  public List<Role> getRoles() {
     return roles;
  }

  public void setRoles(List<Role> roles) {
     this.roles = roles;
  }
}
```

En la clase `User`, hemos agregado el mapeo Many to Many entre usuario y rol. Por lo tanto, esta asignación creará una tercera tabla `user_role` que contendrá todos los identificadores de usuario con sus identificadores de rol.

6. Ahora debes implementar el repositorio de roles, en este caso `RoleRepository` que hereda de `JpaRepository` y proporcionará todos los métodos estándar por defecto.

```
public interface RoleRepository extends JpaRepository<Role, Long>{

}
```

7. Debes crear un repositorio de usuarios, con un método para encontrar un usuario con un nombre de usuario determinado. Agrega un campo adicional para usuarios no activos. Solo se permite iniciar sesión a usuarios activos.

```
public interface UserRepository extends JpaRepository<User, Long> {

  User findByUsernameAndDisabled(String username, boolean disabled);

}
```

8. Ahora necesitamos crear una clase que implemente la clase `UserDetailsService` de Spring y sobre escriba el método `loadUserByUsername()` para proporcionar una implementación personalizada para encontrar al usuario.

```
public class UserDetailsServiceImpl implements UserDetailsService {

  private UserRepository userRepository;

  public UserDetailsServiceImpl(UserRepository userRepository) {
```

```
    this.userRepository = userRepository;
  }

  @Override
  public UserDetails loadUserByUsername(String userName)
  throws UsernameNotFoundException {

    User user = userRepository.findByUsernameAndDisabled(userName, false);
    if (user == null) {
      throw new UsernameNotFoundException("No existe el usuario.");
    }
    return new org.springframework.security.core.userdetails
      .User(user.getUsername(), user.getPassword(),
      mapRolesToAuthorities(user.getRoles()));
  }

  private Collection<? extends GrantedAuthority>
  mapRolesToAuthorities(List<Role> roles) {
    return roles.stream().map(role -> new
    SimpleGrantedAuthority(role.getRoleName())).collect(Collectors.toList());
  }

}
```

9. No se deben guardar las contraseñas en la base de datos como texto sin formato por razones de seguridad. Así que debes encriptar la contraseña, por lo que la contraseña en la base de datos se guarda en forma encriptada y Spring Security la usa directamente como forma encriptada. Entonces, en WebSecurityConfig, crearas un bean `BCryptPasswordEncoder` y se lo proporcionaremos al administrador de autenticación para usarlo en el cifrado de contraseñas entrantes desde la página de inicio de sesión.

10. Después de crear todos los componentes. Ahora es el momento que configures la capa de seguridad. En `WebSecurityConfig`, debes configurar `AuthenticationManagerBuilder` y proporcione la clase `UserDetailsServiceImp` personalizada.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  UserRepository userRepository;
```

```
  @Bean
  public UserDetailsService userDetailsServiceImpl() {
     return new UserDetailsServiceImpl(userRepository);
  }

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

  @Override
  public void configure(AuthenticationManagerBuilder auth) throws Exception {
     auth.userDetailsService(userDetailsServiceImpl())
     .passwordEncoder(passwordEncoder());
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.authorizeRequests()
        .antMatchers("/resources/**").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin().loginPage("/login").permitAll()
        .and()
        .logout().permitAll()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
  }

}
```

11. Todo está configurado ahora. Necesitamos algunos datos de prueba para poder iniciar sesión dentro de la aplicación. Para este propósito, debes crear la clase `DemoData`, que inserta automáticamente datos en la base de datos al iniciar el servidor. Este paso es puramente opcional, puede insertar datos manualmente o puede crear un script SQL para insertar datos.

```
@Component
public class DemoData {

  @Autowired
  private UserRepository userRepository;
```

```
  @Autowired
  private RoleRepository roleRepository;

  @Autowired
  private PasswordEncoder passwordEncoder;

  @EventListener
  public void appReady(ApplicationReadyEvent event) {
     if (roleRepository.count() == 0) {
        System.out.println("Registrando datos de demostración");
        Role roleAdmin = new Role("ADMIN");
        Role roleUser = new Role("USER");
        roleRepository.save(roleAdmin);
        roleRepository.save(roleUser);
        User userAdmin =
           new User("gustavo", passwordEncoder.encode("admin"), roleAdmin);
        User userReader =
           new User("claudia", passwordEncoder.encode("suerte"), roleUser);
        userRepository.save(userAdmin);
        userRepository.save(userReader);
        System.out.println("Datos de demostración registrados correctamente.");
     }
  }
}
```

12. Ahora ya puedes iniciar la aplicación. Verifique en la base de datos, sus 3 tablas (tuser, trole, user_role) deben ser creadas. Estas tablas también deben tener 2 credenciales de inicio de sesión de usuarios.

13. En el navegador utiliza la URL http: // localhost: 8080 o htp: //127.0.0.1: 8080 e ingrese las credenciales correctas y debería poder ver la pantalla de bienvenida.

EJERCICIO

Implementa Spring Boot Security en una aplicación CRUD.