Spring Boot Security

Securitizar Api REST

- Agregar dependencia spring security y **probar REST por browser**.
- Agregar Service UserService.java. {noop} indica que la clave no está - aún - encriptada.

```
@Service
public class UserService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return new User("[username]", "{noop}[password]", new ArrayList<>());
    }
}
```

- Crear SecurityConfig.java: Hacer **@Override** del método configure(AuthenticationManagerBuilder auth)

```
@Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(service);
    }
```

- Probar

JWT

- Agregar dependencia

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

- Crear JWTUtil.java

```
@Component
public class JWTUtil {
    private static final String KEY="br31n_1w1k2l1b.";

    public String generateToken(UserDetails userDetails){
        return Jwts.builder()
                .setSubject(userDetails.getUsername())
```

```
                .setIssuedAt(new Date())
               .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
               .signWith(SignatureAlgorithm.HS256, KEY)
                .compact();
    }

    public boolean validateToken(String token, UserDetails userDetails){
        return userDetails.getUsername().equals(extractUsername(token)) && !isTokenExpired(token);
    }
    public String extractUsername(String token){
        return getClaims(token).getSubject();
    }
    public boolean isTokenExpired(String token){
        return getClaims(token).getExpiration().before(new Date());
    }
    private Claims getClaims(String token){
        return Jwts.parser()
                .setSigningKey(KEY)
                .parseClaimsJws(token)
                .getBody();
    }
}
```

- Crear DTO’s.

```
public class AuthenticationRequest {
    private String username;
    private String password;

    /** Getters y Setters **/
}
```

```
public class AuthenticationResponse {
    private String jwt;

    public AuthenticationResponse(String jwt) {
        this.jwt = jwt;
    }

    /** Getters y Setters **/
}
```

- Crear @RestController

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    private AuthenticationManager authenticationManager;
    private UserService service;
    private JWTUtil jwtUtil;

    public AuthController(AuthenticationManager authenticationManager, UserDetailsService service, JWTUtil jwtUtil) {
        this.authenticationManager = authenticationManager;
        this.service = service;
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> createToken(@RequestBody AuthenticationRequest request){
        try {
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));
            UserDetails userDetails = service.loadUserByUsername(request.getUsername());
            String jwt = jwtUtil.generateToken(userDetails);
            return new ResponseEntity<>(new AuthenticationResponse(jwt), HttpStatus.OK);
        } catch (BadCredentialsException e){
            return new ResponseEntity<>(HttpStatus.FORBIDDEN);
        }

    }
}
```

- Modificar SecurityConfig.java: Sobre escribe métodos configure(HttpSecurity http), y el método authenticationManagerBean()

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable().authorizeRequests().antMatchers("/**/authenticate").permitAll()
```

```
            .anyRequest().authenticated().and().sessionManagement();

}

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

- Crear filter

```
@Component
public class JwtFilterRequest extends OncePerRequestFilter {

    private JWTUtil jwtUtil;
    private UserDetailsService service;

    public JwtFilterRequest(JWTUtil jwtUtil, UserDetailsService service) {
        this.jwtUtil = jwtUtil;
        this.service = service;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletExcept
        String authorizationHeader = request.getHeader("Authorization");
        if (authorizationHeader!=null && authorizationHeader.startsWith("Bearer")){
            String jwt = authorizationHeader.substring(7);
            System.out.println(jwt);
            String username = jwtUtil.extractUsername(jwt);
            if (username != null && SecurityContextHolder.getContext().getAuthentication() == null){
                UserDetails userDetails = service.loadUserByUsername(username);
                if (jwtUtil.validateToken(jwt, userDetails)){
                    UsernamePasswordAuthenticationToken authToken =
                            new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                    authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                    SecurityContextHolder.getContext().setAuthentication(authToken);

                }
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

- Run!

Conectar Spring Security a una BD

- Crea los DTO de User y Rol

```
public class Usuario {

    private String username;
    private String password;
    private Boolean enabled;
    private List<Rol> roles;

    /** Getters y Setters **/
}
```

```
public class Rol {
    private Integer rolId;
    private String rolName;

    /** Getters y Setters **/
}
```

- Crea las Interfaces de los Repository (en Domain) de los DTO’s
- Crea los Entity de Rol y Usuario

```
@Entity
@Table(name = "rol")
```

```
@Entity
@Table(name = "usuario")
```

```
public class Rol {
    @Id
    @Column(name = "rol_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer rolId;
    @Column(length = 50)
    private String rolName;

    /** Getters y Setters **/
}
```

Toma nota de la relación @ManyToMany, y el @JoinTable.

```
public class Usuario {

    @Id
    @Column(length = 50)
    private String username;
    @Column(length = 100)
    private String password;

    private Boolean enabled;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL
    @JoinTable(name = "roles_usuario",
            joinColumns = @JoinColumn(name = "username", reference
            inverseJoinColumns = @JoinColumn(name="rol_id", refere
    private List<Rol> roles;

    /** Getters y Setters **/
}
```

- Implementa los Mappers para Usuario y Rol DTO

```
@Mapper(componentModel = "spring")
public interface RolMapper {
    @Mappings({
            @Mapping(source = "rolId", target = "rolId"),
            @Mapping(source = "rolName", target = "rolName"),
    })
    Rol toRol(RolEntity rol);
    List<Rol> toRoles(Iterable<RolEntity> roles);
    @InheritInverseConfiguration
    RolEntity toRolEntity(Rol rol);
}
```

```
@Mapper(componentModel = "spring", uses = {RolMapper.class})
public interface UserMapper {

    @Mappings({
            @Mapping(source = "username", target = "username"),
            @Mapping(source = "password", target = "password"),
            @Mapping(source = "enabled", target = "enabled"),
            @Mapping(source = "roles", target = "roles"),
    })
    User toUser(UserEntity user);
    List<User> toUsers(List<UserEntity> users);
    @InheritInverseConfiguration
    UserEntity toUserEntity(User user);
}
```

- Implementa los Crud y @Repository de *persistence*, para Rol y Usuario.
- Actualiza tu UserService:

```
@Service
public class UserService implements UserDetailsService {

    private final UserRepository repository;

    public UserDetailsService(UserRepository repository) {
        this.repository = repository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //return new User("[username]", "{noop}[password]", new ArrayList<>());
        User user = repository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));
        return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), mapperRols(user.getRoles()));
    }

    private Collection<? extends GrantedAuthority> mapperRols(List<Rol> roles){
            return roles.stream()
                .map(rol -> new SimpleGrantedAuthority(rol.getRolName()))
                .collect(Collectors.toList());
    }
}
```

- Si indicaste en tu application.properties que actualice o genere el ddl, ejecuta tu app para que cree las tablas nuevas (usuario, rol y roles_usuario).

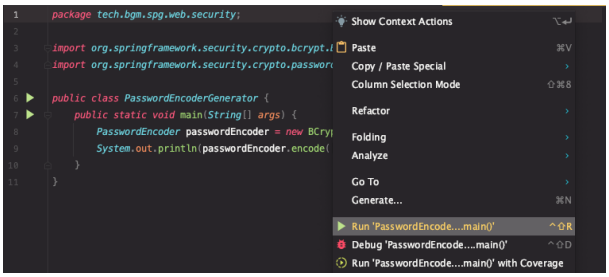
```
spring.jpa.hibernate.ddl-auto=update
```

- Inserta en la BD al menos un rol, un usuario y asígnale el rol a usuario.

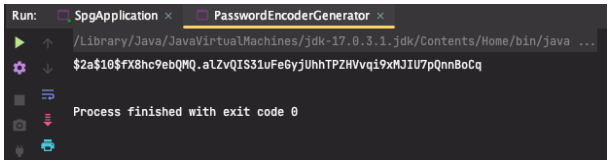
- Para crear la password en BCrypt, puedes crear un archivo como el siguiente, y ejecutarlo directamente

```
public class PasswordEncoderGenerator {
    public static void main(String[] args) {
        PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        System.out.println(passwordEncoder.encode("[contraseña]"));
    }
}
```

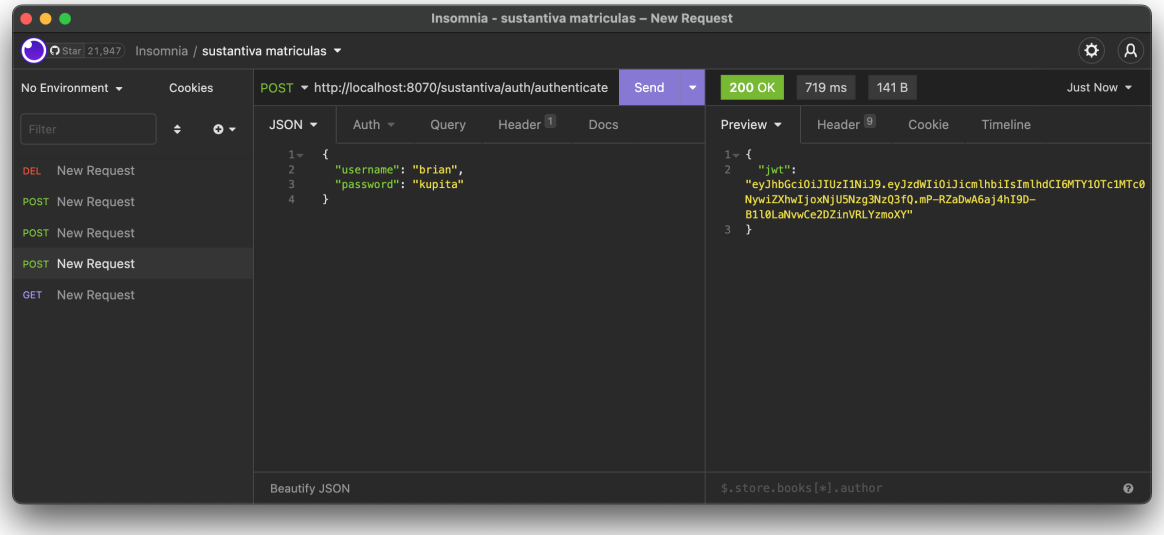
Ejecútalo así:



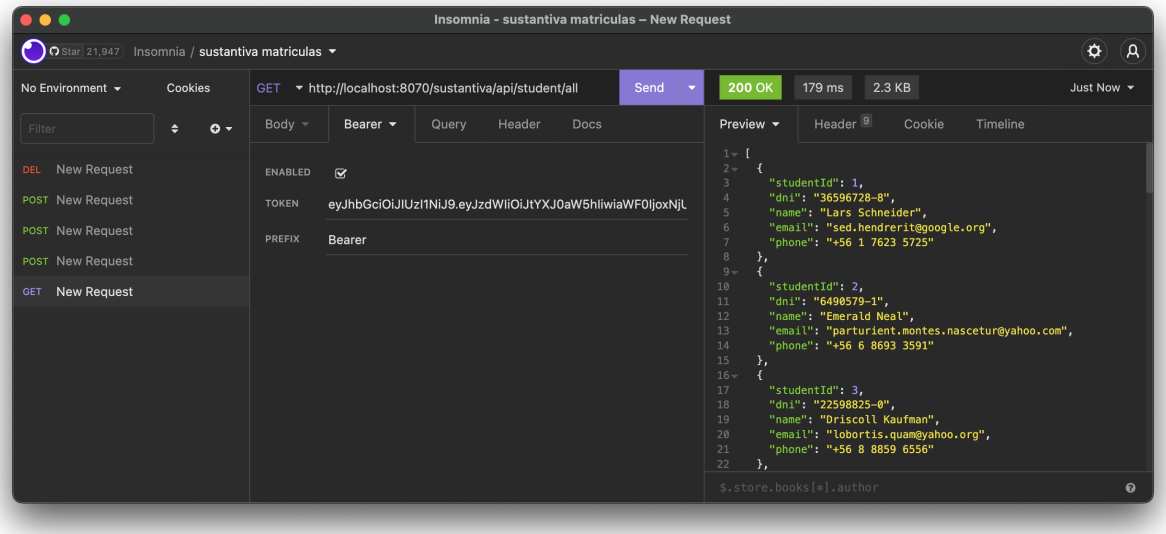
Esto, generará una contraseña, la que puedes copiar para insertarla en el campo password de Usuario.



Ejecuta tu app, enviando por Insomnia o por Postman, los datos de autenticación (usuario y password) en formato Json, como se ejemplifica en la siguiente imagen.



Luego, el token lo puedes adjuntar a tu request por medio del Bearer Token, como se muestra en la siguiente imagen.



Securitizar Web App

Hasta donde este humilde obrero de la formación ha experimentado, El Filter no conversa bien con una forma de autenticación carente del Token. Y el Token no lo necesitamos para autenticar nuestra app web.

Entonces, para efectos de este ejemplo, vamos a duplicar el proyecto (le cambiamos el nombre a la carpeta nueva), y eliminamos los siguientes archivos:

- model>domain>dto>AuthenticationRequest.java
- model>domain>dto>AuthenticationResponse.java
- web>restcontroller>AuthRestController.java
- web>restcontroller>StudentRestController.java
- web>restcontroller
- web>security>filter>JwtFilterRequest.java
- web>security>filter
- web>security>JWTUtil.java
- web>security>PasswordEncoderGenerator.java

Actualiza el SecurityConfig a lo siguiente (toma nota de los nombres de los roles, en la BD debe anteponerse el prefijo “ROLE_”:

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserService service;

    public SecurityConfig(UserService service) {
        this.service = service;
    }

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

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(service).passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf()
                .disable()
                .authorizeRequests()
                .antMatchers("/js/**", "/css/**").permitAll()
                .antMatchers("/login*").permitAll()
                .antMatchers("/").authenticated()
                .antMatchers("/home").hasAnyRole("ADMIN", "USER")
                .antMatchers("/admin").hasRole("ADMIN")
                .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/home", true)
                .permitAll()
                .and()
                .logout().permitAll();
    }
    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

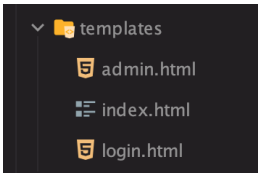
- Crea los siguientes controladores:

```
@Controller
@RequestMapping("/home")
public class HomeController {

    @GetMapping
    public String index(){
        return "index";
    }
}
```

```
@Controller
@RequestMapping("/login")
public class LoginController {

    @GetMapping
    public String login(){
        return "login";
    }
}
```

```
@Controller
@RequestMapping("/admin")
public class AdminController {
    @GetMapping
    public String admin(){
        return "admin";
    }
}
```

Crea los templates correspondientes a cada controlador.



El formulario de Login, debe contener un atributo username, y otro password, para que sean comprendidos por Spring Security, como se muestra más abajo.

```
<form th:action="@{/login}" method="post">
<div>
  <div>
    <label for="username">Usuario:</label>
    <input type="text" id="username" name="username" placeholder="ElDemoledor">
    </div>
  <div>
    <label for="password">Contraseña:</label>
    <input type="password" id="password" name="password">
  </div>
  <div>
    <div><button type="submit">Ingresar</button> </div>
  </div>
</div>
</form>
```