PRÁCTICA UT-5
PROGRAMACIÓN SEGURA

Antonio León Almodóvar DAM 2ºB

Índice

Desarrollo

Decidí comenzar el desarrollo por el servicio REST, implementando las clases y los métodos que había en el temario, adaptando el User para que implemente UserDetails, y también los métodos de encriptado, des encriptado y hasheado, una vez toda las etiquetas y clases con las anotaciones pertinentes me lance al método “filterChain”, escribiendo todas las restricciones.

```
x -> x.requestMatchers( ...patterns: "/user").anonymous()
      .requestMatchers(HttpMethod.DELETE, ...patterns: "/user/{id}").hasAnyRole( ...roles: "admin", "usuario")
      .requestMatchers(HttpMethod.PUT, ...patterns: "/user/{id}").hasAnyRole( ...roles: "admin", "usuario")
      .requestMatchers(HttpMethod.GET, ...patterns: "/user/{id}").hasAnyRole( ...roles: "admin", "personal", "usuario")
      .requestMatchers( ...patterns: "/flights/add", "/flights/{cod}/delete", "/flights/{cod}/update").hasRole("admin")
      .requestMatchers(HttpMethod.GET, ...patterns: "/flights", "/flights/{cod}").authenticated()
      .requestMatchers( ...patterns: "/flights/{cod}/passenger", "/flights/{cod}/passenger/{nif}" , "/flights/{cod}/passengers").hasAnyRole( ...roles: "admin", "personal")
      .requestMatchers( ...patterns: "/flights/{cod}/passengers/{nif}/luggage/{id}").hasRole("personal")
      .requestMatchers( ...patterns: "/flights/{cod}/passengers/{nif}/luggage").hasRole("personal")
      .requestMatchers( ...patterns: "/flights/{cod}/passengers/luggages").hasRole("personal")
      //.anyRequest().authenticated()
```

Posteriormente me lancé a crear los DTOs tanto de User como de Role, dándome cuenta de que, ¿para qué necesito un RoleDTO cuando los roles van a ser predeterminados? De ese debate interno concluí que el UserDTO no tendría una lista de RoleDTO, tendría una lista de Strings con el campo ‘rol’ de los Roles.

```
private List<Role> fromStringToRole(List<String> strRoles) {
    List<Role> roles = new ArrayList<>();
    for (String rol : strRoles) {
        switch (rol) {
            case "ROLE_admin":
                roles.add(new Role(rol, name: "Administrador", description: "El que administra"));
                break;
            case "ROLE_personal":
                roles.add(new Role(rol, name: "Personal", description: "El que personalea"));
                break;
            case "ROLE_usuario":
                roles.add(new Role(rol, name: "Usuario", description: "Es un usuario"));
                break;
        }
    }
    return roles;
}

1 usage  ntonioLeon
private List<String> fromRoleToString(List<Role> roles) {
    List<String> strRoles = new ArrayList<>();
    for (Role role : roles) {
        strRoles.add(role.getAuthority());
    }
    return strRoles;
}
```

Tras eso decidí ocuparme del UserMapper, encriptando email, nif y hasheando la password hacia entity y des encriptando hacia DTO, para ello le añadí el @Component “SecurityUtil” que contiene los métodos necesarios para esas acciones.

```
2 usages  ntonioLeon
public User toEntity(UserDTO userDTO) throws MiValidacionException, NoSuchAlgorithmException, IllegalBlockSizeException, NoSuchPaddingException, BadPaddingException, Invali
    return new User(userDTO.getUsername(), securityUtil.createHash(userDTO.getPassword()), securityUtil.crypt(validadorDeCampos.checkDni(userDTO.getNif())), userDTO.getName
}

1 usage  ntonioLeon
public UserDTO toDTO(User user) throws IllegalBlockSizeException, NoSuchPaddingException, BadPaddingException, NoSuchAlgorithmException, InvalidKeyException, UserNotFoundEx
    if (user != null) {
        return new UserDTO(user.getUsername(), password: null, securityUtil.decrypt(user.getNif()), user.getName(), user.getSurname(), securityUtil.decrypt(user.getEmail()),
    } else {
        throw new UserNotFoundException();
    }
```

Una vez hecho eso comencé con los Endpoints de User, tratando de controlar las salidas de estos, supervisando que el User autenticado tenga los roles que necesita para realizar sus acciones o sea quien dice ser, para ello se usé las líneas en el enunciado para obtener los datos del usuario autenticado.

```
3 usages  ntonioLeon
private UserDetails getAutenticated() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    return (UserDetails) auth.getPrincipal();
}

3 usages  ntonioLeon
private boolean esAdmin(UserDetails userDetail) {
    return userDetail.getAuthorities().contains(new Role( rol: "ROLE_admin",  name: "Administrador",  description: "El que administra"))
}

1 usage  ntonioLeon
private boolean esPersonal(UserDetails userDetail) {
    return userDetail.getAuthorities().contains(new Role( rol: "ROLE_personal",  name: "Personal",  description: "El que personalea"));
}
```

Puesto que como dije antes decidí hacer que los Roles sean tres e inmutables solo tenía que comparar que el usuario autenticado tenía dichos roles para comprobar si podía interactuar con otros usuarios distintos a él según los criterios del enunciado.

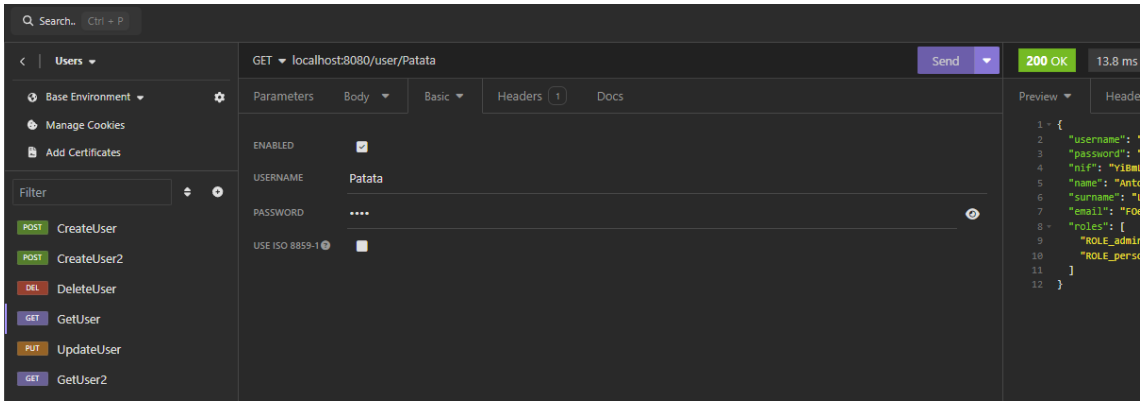
```
UserDetails userDetail = getAutenticated();
if (userDetail.getUsername().equals(id) || esAdmin(userDetail) || esPersonal(userDetail)) {
    if (inMemoryUserRepository.existUser(id)) {
        return inMemoryUserRepository.getUser(id);
    } else {
        return null;
    }
} else {
    throw new NoTenesPoderAquiException();
}
```

Puesto que el POST para crear user es anónimo tuve que adaptar el método “doPost” para que pueda aceptar peticiones sin el authorization en el caso de crear el primer User.

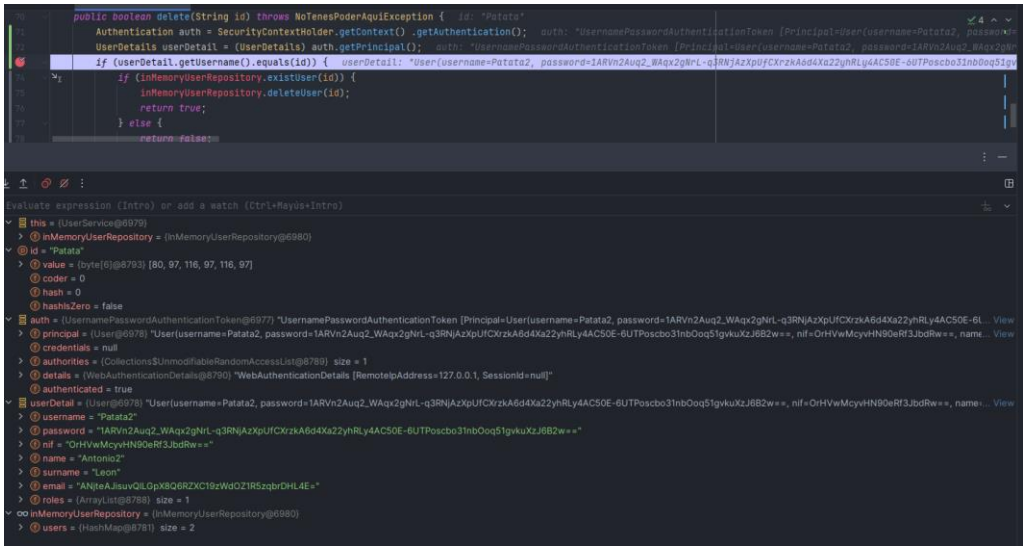
```
HttpRequest request;
if ("".equals(Cliente.userName)  || "".equals(Cliente.password)) {
    request = HttpRequest.newBuilder()
            .uri(URI.create(url))
            .POST(HttpRequest.BodyPublishers.ofString(body))
            .header( name: "Content-Type",  value: "application/json")
            .build();
} else {
    request = HttpRequest.newBuilder()
            .uri(URI.create(url))
            .POST(HttpRequest.BodyPublishers.ofString(body))
            .header( name: "Content-Type",  value: "application/json")
            .header( name: "Authorization", getBasicAuthenticationHeader(Cliente.userName, Cliente.password))
            .build();
}
```

Depuración

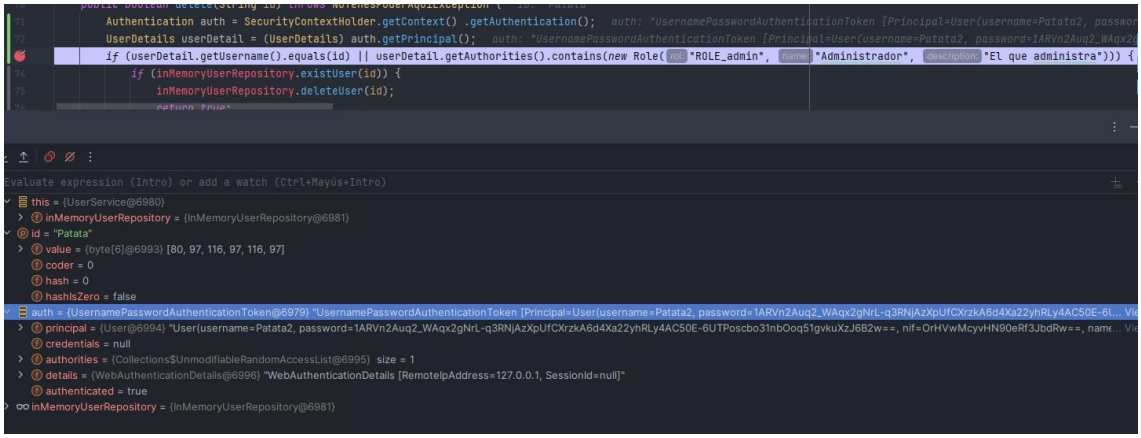
La mayoría de la depuración ha sido realizada en por medio de insomnia ya que ha resultado ser extremadamente ágil a la hora de ejecutar una secuencia de pruebas básicas.



Aunque uno de los problemas que me obligó a depurar fue el hecho de que en mi intento de comprobar que el usuario era el mismo a la hora de querer buscarse, borrarse o modificarse olvidé añadir en la cláusula del 'if' que si era administrador también podía hacer dichas cosas. Tanto me enfoqué ciegamente que hasta que no vi las variables en el debugger no vi que me faltaba comprobar el rol. En definitiva, debuggear me ayudó a ver el problema con más calma y ver lo que estaba pasando.



Solución:

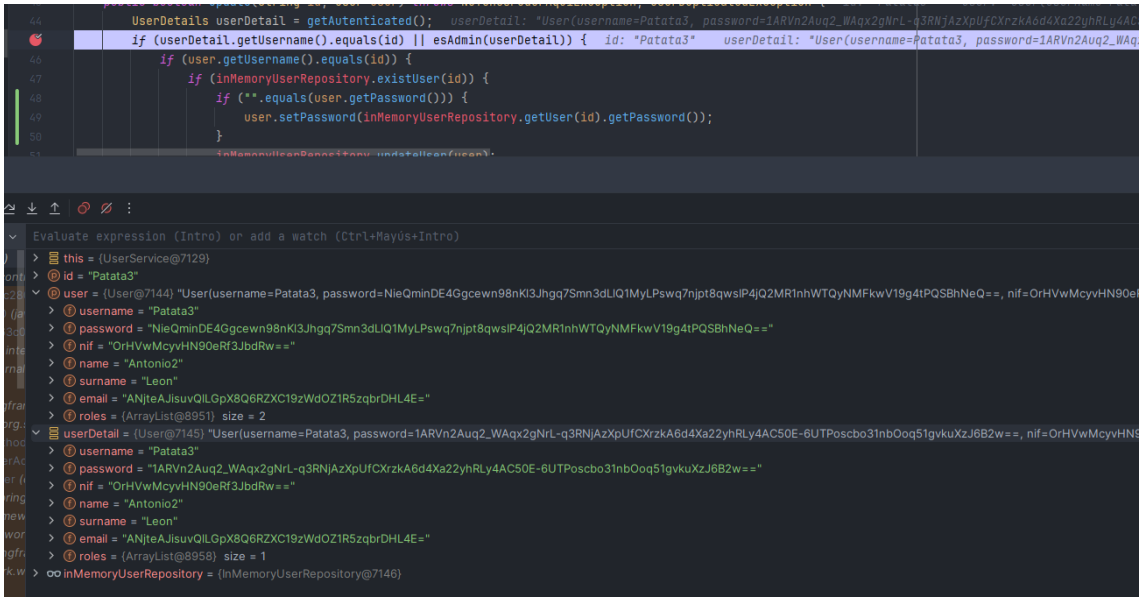


Tampoco negaré que también he tirado del viejo y confiable “sout” ya que no me fiaba de que la información de usuario autenticado fuera real.

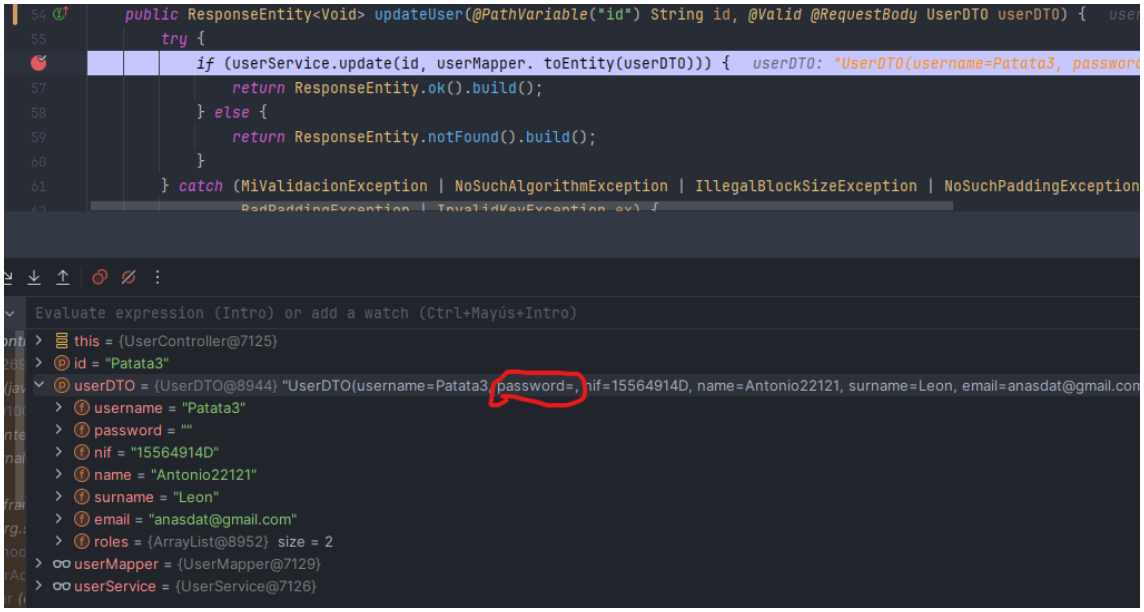
En lo referente al update de users encontré un problema, ya que un UserDTO valido no puede tener null como password. Cosa que en el cliente se solventó fácil.

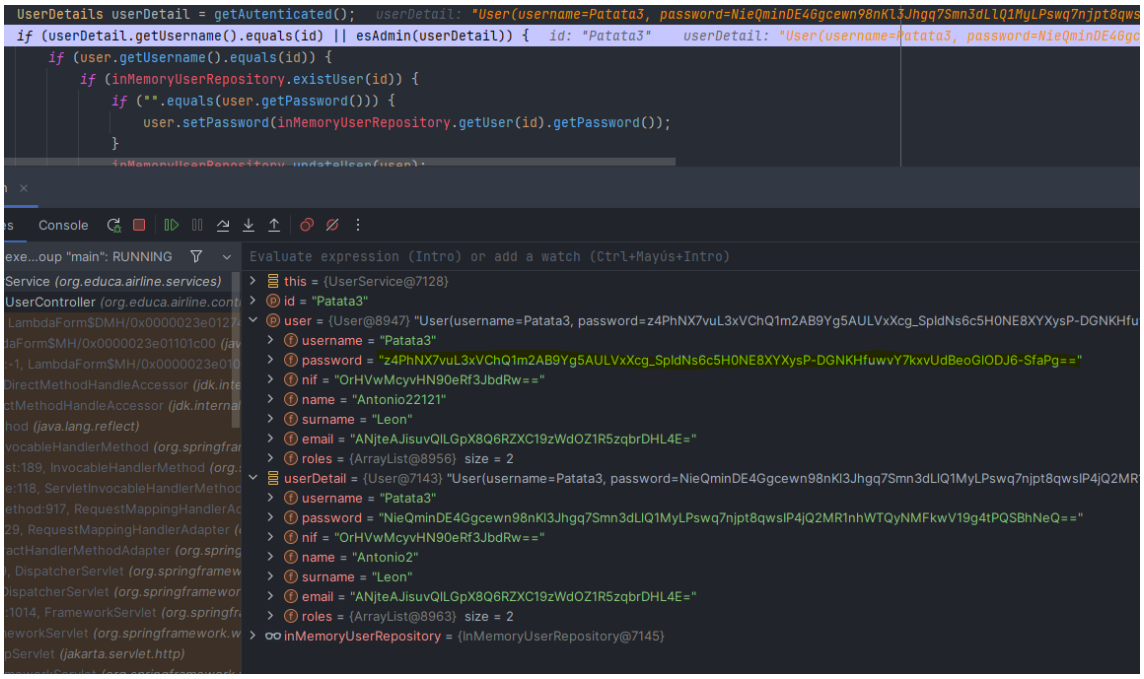
```
modificacion(scanner, userDTO);
if (userDTO.getPassword() == null)
    userDTO.setPassword("");
apiUserService.updateUser(id, userDTO);
```

En cambio en el server no fue tan fácil, ya que no podemos dejar que los usuarios actualizados tengan siempre la contraseña resultante de hashear un espacio vacio.. cosa que tardé en darme cuenta.



Cuando los Users actualizados vienen con una contraseña distinta a un campo vacío les dejo ese hash como contraseña, ya que asumo que es que la han cambiado en el modificar. En cambio, me costaba entender el motivo por el cual el if de la línea 48 no estaba resultando… Debuggeando vi por qué el if de la línea 48 estaba mal, no tenía que comparar con un espacio vacío, tenía que comparar con el hash de un espacio vacío.





Una vez actualizada la cláusula del if la comparación funcionaba correctamente y ya podía actualizar Users que decidieran no cambiar la Password.