UserDetails And UserPrincipal

In Spring Security, UserDetails is an interface that provides core user information. Spring Security needs this information to authenticate users and manage their access to resources. When we implement database authentication, we need to create a class that implements UserDetails to bridge our database User model with Spring Security's authentication system.

- **Creating UserPrincipal Class**
  Create a class called UserPrincipal in the model package that implements the UserDetails interface:

Example:

```java
package com.telusko.springsecdemo.model;

import java.util.Collection;
import java.util.Collections;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

public class UserPrincipal implements UserDetails {

    private static final long serialVersionUID = 1L;

    private User user;

    public UserPrincipal(User user) {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singleton(new SimpleGrantedAuthority("USER"));
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

1. **Constructor**

```
public UserPrincipal(User user) {
    this.user = user;
}
```

- Takes a User object (our database entity) and stores it
- This allows us to access the user's information from our database

2. **getAuthorities()**

```
@Override public Collection<? extends GrantedAuthority> getAuthorities() {
    return Collections.singleton(new SimpleGrantedAuthority("USER"));
}
```

- Returns the roles/permissions assigned to the user
- In this example, we're assigning a simple "USER" role to all users
- Collections.singleton() creates a collection with just one element
- For multiple roles, we would return a list of authorities

3. **getPassword()**

```
@Override public String getPassword() {
  return user.getPassword();
}
```

- Returns the user's password from our database entity
- Spring Security uses this to verify the password during authentication

4. **getUsername()**

```
@Override public String getUsername() {
    return user.getUsername();
}
```

- Returns the user's username from our database entity
- Used as the identifier for the user in Spring Security

5. **Account Status Methods**

```
@Override public boolean isAccountNonExpired() {
  return true;
}
@Override public boolean isAccountNonLocked() {
  return true;
}
```

```
@Override public boolean isCredentialsNonExpired() {
  return true;
}
@Override public boolean isEnabled() {
 return true;
}
```

- These methods control various aspects of account status
- Returning true means the account is valid in all aspects
- In a more complex application, these might check database flags for account status

➢ **Using UserPrincipal in UserDetailsService**

In MyUserDetailsService, use the UserPrincipal class to wrap our database user:

Example:

```
package com.telusko.springsecdemo.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import com.telusko.springsecdemo.dao.UserRepo;
import com.telusko.springsecdemo.model.User;
import com.telusko.springsecdemo.model.UserPrincipal;

@Service
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepo repo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = repo.findByUsername(username);

        if (user == null) {
            System.out.println("User 404");
            throw new UsernameNotFoundException("User 404");
        }

        return new UserPrincipal(user);
    }
}
```

- We fetch the User from the database using the repository
- If the user doesn't exist, we throw a UsernameNotFoundException
- If the user exists, we create a new UserPrincipal with the user data
- We return this UserPrincipal object, which Spring Security will use for authentication

- The loadUserByUsername() method is called during authentication
- It retrieves our database User entity
- We wrap this entity in a UserPrincipal object
- Spring Security uses the information from UserPrincipal to:
  - Verify the password
  - Check account status
  - Determine user authorities/roles

Output: