User Repository

➢ Creating the User Repository

When working with JPA in Spring, we create a repository interface to interact with our database:

- Create an interface called UserRepo in the DAO (Data Access Object) layer
- It extends JpaRepository which provides built-in database operations
- The first type parameter (User) is the entity class that maps to our database table
- The second type parameter (Integer) is the type of our primary key
- Add the @Repository annotation to mark it as a Spring repository component

Example:

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

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepo extends JpaRepository<User, Integer> {

}
```

➢ Creating the User Entity Class

For our repository to work, we need a User entity class that maps to our database table:

- Create this class in the model layer
- It has the same properties as our database table (id, username, password)
- @Entity marks this class as a JPA entity
- @Table(name = "users") specifies which database table this class maps to
- @Id marks the primary key field
- @Data is a Lombok annotation that automatically generates getters, setters, equals, hashCode, and toString methods

Example:

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

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Table(name = "users")
@Entity
public class User {

    @Id
    private int id;
    private String username;
    private String password;

}
```

➢ **Adding a Custom Query Method**

To find a user by username, we add a custom query method to our repository:

- findByUsername is a custom method we define
- Spring Data JPA automatically implements this method based on the naming convention
- It will generate a query like SELECT * FROM users WHERE username = ?
- The method returns a single User object (or null if not found)

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

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.telusko.springsecdemo.model.User;

@Repository
public interface UserRepo extends JpaRepository<User, Integer> {
    User findByUsername(String username);
}
```

➢ **Using the Repository in UserDetailsService**

Now we can use our repository in the loadUserByUsername method:

- Use the repository to find a user by their username
- Check if the user exists in the database
- If the user doesn't exist, we throw a UsernameNotFoundException

Example:

```java
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 null
    }
}
```