Spring Boot REST Security with JPA/Hibernate Tutorial

Introduction

In this tutorial, you will learn how to secure a Spring Boot REST API with Spring Security using JPA/Hibernate.

Prerequisites

This tutorial assumes that you have already completed the Spring Security videos in the Spring Boot 3, Spring 6 course. This includes the Spring Security videos for JDBC authentication for plain-text passwords and encrypted passwords.

Overview of Steps

1. Download and Import the code
2. Run database scripts
3. Review the source code
4. Test the App

Download and Import the Code

The code is provided as a full Maven project and you can easily import it into your IDE.

Download the Code

1. Download the code from: https://www.luv2code.com/bonus-lecture-spring-boot-rest-security-jpa-hibernate-bcrypt-code
2. Unzip the file.

Import the Code

1. In IntelliJ, select **File > Open**
2. Browse to the directory where you unzipped the code.
3. Click OK buttons etc., to import the code

Review the project structure

Make note of the following directories

- /src/main/java: contains the main java code
- /src/main/resources: contains the database configuration file
- /sql-scripts: the database script for the app (security accounts)

2. Run database scripts

In order to make sure we are all on the same page in terms of database schema names and user accounts/passwords, let's run the same database scripts.

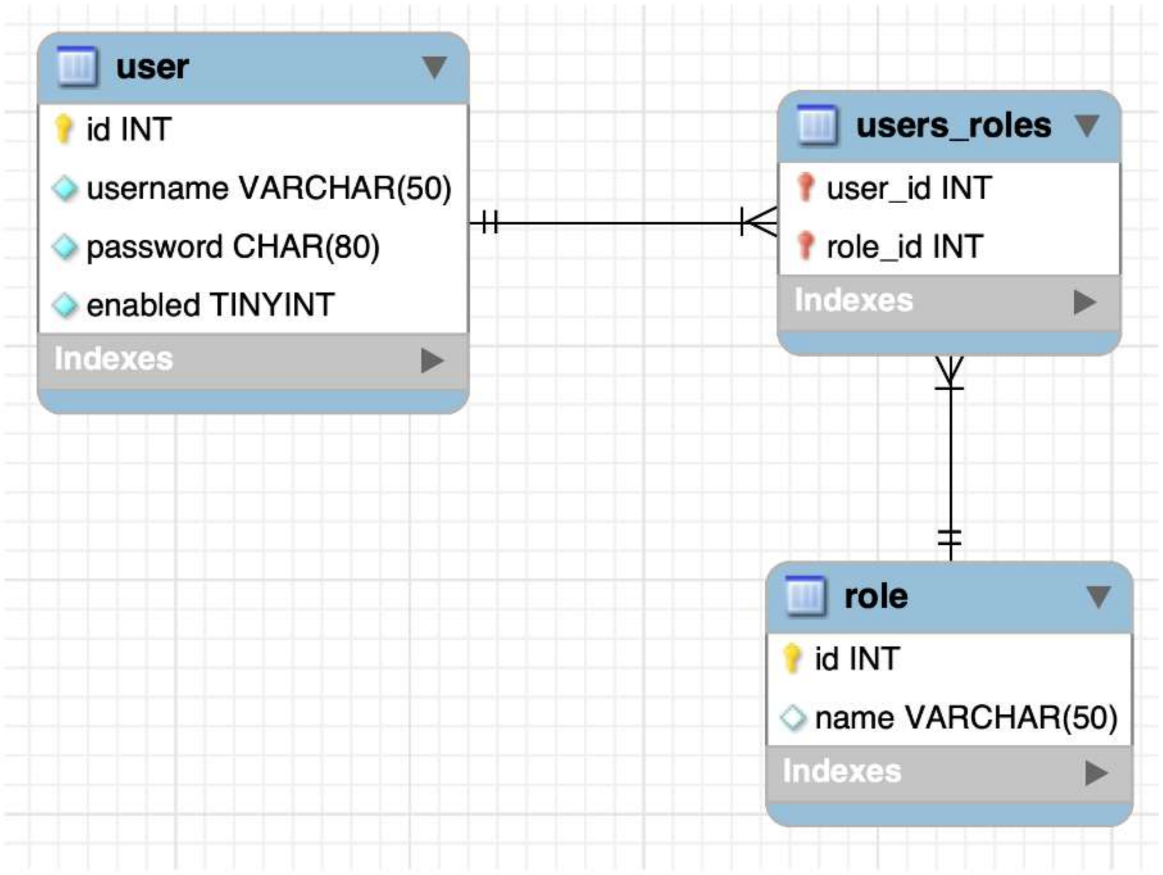
MySQL workbench

In MySQL workbench, run the following database scripts:

- `/sql-scripts/01-employee-directory.sql`
- `/sql-scripts/02-setup-spring-security-demo-database-hibernate-bcrypt.sql`

The first script adds sample employees to the database which is our sample data for Employee CRUD.

The second script creates the database tables for security.



The script also creates the user accounts with encrypted passwords. It also includes the user roles.

User ID	Password	Roles
john	fun123	EMPLOYEE
mary	fun123	EMPLOYEE, MANAGER
susan	fun123	EMPLOYEE, MANAGER, ADMIN

Review: BCryptPasswordEncoder and DaoAuthenticationProvider beans

In our security configuration file, `DemoSecurityConfig.java`, we define a BcryptPasswordEncoder and DaoAuthenticationProvider beans. We assign the UserService and PasswordEncoder to the DaoAuthenticationProvider. These classes are used by Spring Security for custom authentication and authorization.

File: /src/main/java/com/luv2code/springboot/cruddemo/config/DemoSecurityConfig.java

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

//authenticationProvider bean definition
@Bean
public DaoAuthenticationProvider authenticationProvider(UserService userService) {
    DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
    auth.setUserDetailsService(userService); //set the custom user details service
    auth.setPasswordEncoder(passwordEncoder()); //set the password encoder - bcrypt
    return auth;
}
```

We are assigning the custom user details and password encoder to the DaoAuthenticationProvider.

Review: Custom User Details entity classes (User, Role)

Creating the User and Role entity classes (We can use any name for these entities)

File: /src/main/java/com/ luv2code/springboot/cruddemo/entity/User.java

```
package com.luv2code.springboot.cruddemo.entity;

import jakarta.persistence.*;
import java.util.Collection;

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "username")
    private String userName;

    @Column(name = "password")
    private String password;


  // other fields methods omitted for brevity

  …

}
```

File: /src/main/java/com/luv2code/ luv2code/springboot/cruddemo/entity/Role.java

```
package com.luv2code.springboot.cruddemo.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    private String name;

    public Role() {
    }

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

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
  // other fields methods omitted for brevity

  …

}
```

Review: Service and Dao classes

The service class has a method to find the user by username. This is used by Spring Security to find a user during the login process.

The UserService extends UserDetailsService.

File: /src/main/java/com/luv2code/springboot/cruddemo/service/UserService.java

```java
package com.luv2code.springboot.cruddemo.service;

import com.luv2code.springboot.cruddemo.entity.User;
import org.springframework.security.core.userdetails.UserDetailsService;

public interface UserService extends UserDetailsService {

   public User findByUserName(String userName);

}
```

In the UserServiceImpl we implement the methods to lookup a user by username

File: /src/main/java/com/luv2code/springboot/cruddemo/service/UserServiceImpl.java

```java
package com.luv2code.springboot.cruddemo.service;

import com.luv2code.springboot.cruddemo.dao.RoleDao;
import com.luv2code.springboot.cruddemo.dao.UserDao;
import com.luv2code.springboot.cruddemo.entity.User;
import com.luv2code.springboot.cruddemo.entity.Role;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.stream.Collectors;

@Service
public class UserServiceImpl implements UserService {

   private UserDao userDao;

   private RoleDao roleDao;

   @Autowired
   public UserServiceImpl(UserDao userDao, RoleDao roleDao) {
      this.userDao = userDao;
      this.roleDao = roleDao;
   }

   @Override
   public User findByUserName(String userName) {
      // check the database if the user already exists
      return userDao.findByUserName(userName);
   }

   @Override
   public UserDetails loadUserByUsername(String userName) throws
UsernameNotFoundException {
      User user = userDao.findByUserName(userName);
      if (user == null) {
         throw new UsernameNotFoundException("Invalid username or password.");
      }
      return new org.springframework.security.core.userdetails.User(user.getUserName(),
user.getPassword(),
            mapRolesToAuthorities(user.getRoles()));
   }

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

The corresponding method calls in the Dao layer.

UserDao

File: /src/main/java/com/luv2code/springboot/cruddemo/dao/UserDao.java

```
package com.luv2code.springboot.cruddemo.dao;

import com.luv2code.springboot.cruddemo.entity.User;

public interface UserDao {

    User findByUserName(String userName);

}
```

UserDaoImpl

File: /src/main/java/com/luv2code/springboot/cruddemo/dao/UserDaoImpl.java

```
package com.luv2code.springboot.cruddemo.dao;

import com.luv2code.springboot.cruddemo.entity.User;
import jakarta.persistence.EntityManager;
import jakarta.persistence.TypedQuery;
import org.springframework.stereotype.Repository;

@Repository
public class UserDaoImpl implements UserDao {

   private EntityManager entityManager;

   public UserDaoImpl(EntityManager theEntityManager) {
      this.entityManager = theEntityManager;
   }

   @Override
   public User findByUserName(String theUserName) {

      // retrieve/read from database using username
      TypedQuery<User> theQuery = entityManager.createQuery("from User where
userName=:uName", User.class);
      theQuery.setParameter("uName", theUserName);

      User theUser = null;
      try {
         theUser = theQuery.getSingleResult();
      } catch (Exception e) {
         theUser = null;
      }
```

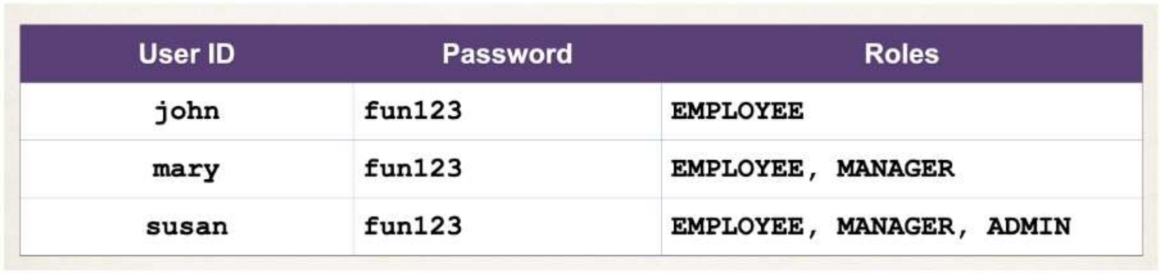
```
        return theUser;
    }

}
```

Test the App

At this point, you can test the application.

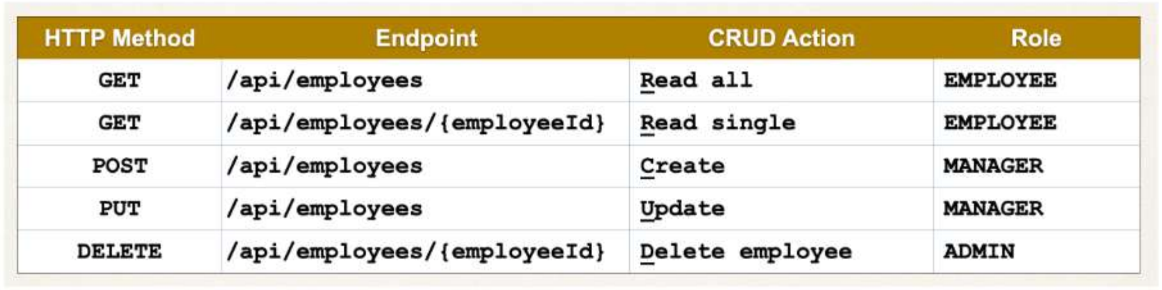
1. Run the app
2. Start Postman
3. Test the following scenarios

User Accounts



User ID	Password	Roles
john	fun123	EMPLOYEE
mary	fun123	EMPLOYEE, MANAGER
susan	fun123	EMPLOYEE, MANAGER, ADMIN

User Actions



HTTP Method	Endpoint	CRUD Action	Role
GET	/api/employees	Read all	EMPLOYEE
GET	/api/employees/{employeeId}	Read single	EMPLOYEE
POST	/api/employees	Create	MANAGER
PUT	/api/employees	Update	MANAGER
DELETE	/api/employees/{employeeId}	Delete employee	ADMIN

Confirm the REST API is secured based on the user roles.

Success! You implemented Spring Security using Hibernate!