

Setting user authorities



In springboot, when we create a new user how can we set authorities/authorization for a user?

Good question. I'll explain this **step-by-step, simple and practical**, the way it's actually done in **Spring Boot + Spring Security**.

Big Picture (very important)

When you **create a new user**, you don't directly give *authorization*.

You give the user **Authorities / Roles**, and **Spring Security uses them later** to authorize requests.

👉 **Authentication = who you are**

👉 **Authorization = what you can access**

1 Where are authorities stored?

Usually in **DB** when user is created.

Typical options:

- ROLE_USER
- ROLE_ADMIN
- READ_PRIVILEGE
- WRITE_PRIVILEGE

Spring Security internally works with:

```
java
GrantedAuthority
```

2 Common DB Design (Recommended)

User Table

```
java
User
- id
- username
- password
- enabled
```

Role / Authority Table

```
java

Role
- id
- name // ROLE_USER, ROLE_ADMIN
```

Mapping (Many-to-Many)

```
java

user_roles
- user_id
- role_id
```

3 Entity Example

User Entity

```
java

@Entity
public class User {

    @Id
    @GeneratedValue
    private Long id;

    private String username;
    private String password;
    private boolean enabled;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();
}
```

Role Entity

```
java

@Entity
public class Role {

    @Id
    @GeneratedValue
    private Long id;

    private String name; // ROLE_USER, ROLE_ADMIN
}
```

4 Setting Authorities When Creating User ✓

Step 1: Fetch Role

```
java

Role userRole = roleRepository.findByName("ROLE_USER")
    .orElseThrow(() -> new RuntimeException("Role not found"));
```

Step 2: Assign Role to User

```
java

User user = new User();
user.setUsername("saurabh");
user.setPassword(passwordEncoder.encode("password"));
user.setEnabled(true);

user.getRoles().add(userRole);

userRepository.save(user);
```

 This is the moment when you set authorization.

5 How Spring Security Uses These Roles

You convert DB roles → `GrantedAuthority`

Custom UserDetails

```
java

public class CustomUserDetails implements UserDetails {

    private User user;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return user.getRoles()
            .stream()
            .map(role -> new SimpleGrantedAuthority(role.getName()))
            .toList();
    }

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

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

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

6 Authorization in Controller / Config

Using `hasRole`

```
java

@PreAuthorize("hasRole('ADMIN')")
@GetMapping("/admin")
public String adminApi() {
    return "Admin access";
}
```

Spring internally checks:

```
nginx

ROLE_ADMIN
```

Using `hasAuthority`

```
java

@PreAuthorize("hasAuthority('ROLE_USER')")
@GetMapping("/user")
public String userApi() {
    return "User access";
}
```

7 Role vs Authority (INTERVIEW 🔥)

Role	Authority
High level	Low level
<code>ROLE_ADMIN</code>	<code>READ, WRITE</code>
Uses <code>hasRole()</code>	Uses <code>hasAuthority()</code>
Automatically prefixed with <code>ROLE_</code>	No prefix

8 JWT Case (Very Common)

If you use **JWT**, roles are stored in token claims.

```
json

{
    "sub": "saurabh",
    "roles": ["ROLE_USER"]
}
```

On every request:

- JWT → extract roles
- Convert to `GrantedAuthority`
- Spring authorizes request

9 Summary (One-Line Flow)

```
pgsql
User Creation
→ Assign Role(s)
→ Save in DB
→ Convert to GrantedAuthority
→ Spring Security authorizes endpoints
```

If you want, I can next explain:

- **Role vs Permission best design**
- **JWT + role based auth flow**
- **Dynamic role assignment**
- **Multi-tenant authorization**
- **How to change user role after creation**

Just tell 



jab hum user create toh uske liye role kaise set karte hai, kya phle hum `GrantedAuthority` object banate hai?

Short answer:  **Nahi**

User create karte time **hum `GrantedAuthority` object nahi banate.**

Ab clearly, step-by-step samjho 

Simple Rule (Yaad rakhna)

- 👉 User Creation time → DB level pe ROLE / AUTHORITY set hoti hai
- 👉 Login / Request time → `GrantedAuthority` object banta hai

User Create karte waqt kya hota hai?

1 User signup / creation flow

text

```
API call aayi
→ User object bana
→ Role assign kiya (DB entity)
→ User save kar diya
```

X Yaha GrantedAuthority ka koi role nahi hota

Example: User Create with Role

Role already DB me hai

text

```
ROLE_USER
ROLE_ADMIN
```

User creation code

```
java

Role roleUser = roleRepository.findByName("ROLE_USER")
    .orElseThrow(() -> new RuntimeException("Role not found"));

User user = new User();
user.setUsername("saurabh");
user.setPassword(passwordEncoder.encode("1234"));

user.getRoles().add(roleUser);

userRepository.save(user);
```

💡 Bas itna hi

Yahi authorization setup hai.

Fir GrantedAuthority kab banta hai? 🔥

Login ke time / request ke time

Spring Security internally ye flow follow karta hai:

text

```
Request aayi
→ UserDetailsService.loadUserByUsername()
→ User DB se load hua
→ Roles ko GrantedAuthority me convert kiya
→ SecurityContext me set kiya
```

Conversion Example

DB Role

```
java
ROLE_USER
```

Converted to GrantedAuthority

```
java
new SimpleGrantedAuthority("ROLE_USER");
```

Code

```
java
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return user.getRoles()
        .stream()
        .map(role -> new SimpleGrantedAuthority(role.getName()))
        .toList();
}
```

👉 Yahi jagah hai jaha GrantedAuthority banta hai

Important Interview Line 🔥

"GrantedAuthority is a **runtime security object**, not a persistence object."

Agar JWT use kar rahe ho toh?

Still same logic:

- Signup → role DB me save
- Login → role JWT claim me daalo
- Request → JWT → GrantedAuthority

✗ Signup ke time GrantedAuthority nahi

Visual Flow (One look)

```
text
USER CREATE
DB: USER + ROLE
```

 LOGIN / REQUEST
 Role → GrantedAuthority
 → Authorization

Final Answer (One Line)

Nahi, user create karte waqt GrantedAuthority object nahi banta.

Hum sirf role/permission DB me set karte hain, aur **Spring Security login/request ke time unko GrantedAuthority me convert karta hai**.

Agar chaho toh next:

- JWT ke saath exact flow
- Role vs Permission real project design
- Why **ROLE_** prefix is important
- Custom Permission based auth

Bas bol 



Why we convert role into GrantedAuthority when doing login

Very important conceptual question 

Isko **core Spring Security design** se samjho — tab life-time yaad rahega.

Short Answer (One-liner)

Because Spring Security internally understands only GrantedAuthority, not Role or DB entities.

Deep but Simple Explanation

1 Role kya hota hai?

text

ROLE_USER, ROLE_ADMIN

- 👉 Ye **business / DB level concept** hai
- 👉 Spring Security ko isse directly matlab nahi

2 Spring Security ko kya chahiye?

Spring Security ka **authorization engine** sirf ye check karta hai:

```
java
Collection<? extends GrantedAuthority>
```

Examples:

```
java
new SimpleGrantedAuthority("ROLE_USER");
new SimpleGrantedAuthority("ROLE_ADMIN");
```

👉 Authority = security engine ka language

Why Role directly use nahi kar sakte? 🤔

Reason 1: Decoupling (Design Principle)

- DB entities change ho sakti hain
- Security engine stable rehna chahiye

👉 Isliye Spring bolta hai:

"Tum jo bhi ho — role, permission, scope —
mujhe bas **GrantedAuthority** do."

Reason 2: Uniform Authorization Model

Spring Security ko farak nahi padta:

- Role hai
- Permission hai
- OAuth scope hai
- LDAP group hai

Sabko ek hi format me convert karta hai:

```
text
GrantedAuthority
```

Reason 3: Fast Runtime Checks ⚡

Authorization hota hai **har request pe**:

```
java
hasRole("ADMIN")
hasAuthority("READ_PRIVILEGE")
```

Agar DB entity use kare:

- Lazy loading
- Proxy
- DB hit
- Performance issue

👉 **GrantedAuthority = in-memory, lightweight object**

Login Flow (Real Flow)

```
text
Login request
→ UserDetailsService.loadUserByUsername()
→ User + Roles DB se aaye
→ Roles → GrantedAuthority me convert
→ SecurityContextHolder me store
→ Authorization checks start
```

Code Point jaha conversion hota hai

```
java
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return user.getRoles()
        .stream()
        .map(role -> new SimpleGrantedAuthority(role.getName()))
        .toList();
}
```

👉 Yahi **bridge** hai DB aur Security Engine ke beech

Important Interview Question 🔥

Q: Why not use Role entity directly for authorization?

Answer:

- Spring Security authorization works only with **GrantedAuthority**
- It avoids tight coupling with persistence layer
- Provides a unified, runtime-friendly authorization model

Real-Life Analogy 🧠

Socho:

- **Role** = Government ID (Aadhaar)
- **GrantedAuthority** = Entry Pass for office

Security guard:

- Aadhaar verify nahi karta
- Entry pass check karta hai

Final Summary (Crystal Clear)

- Role → **stored in DB**
- GrantedAuthority → **runtime security token**
- Login ke time conversion isliye hota hai kyunki
Spring Security sirf GrantedAuthority ko authorize karta hai

Agar chaho next main explain kar sakta hoon:

- **hasRole vs hasAuthority** internal working
- JWT me authorities kaise inject hoti hain
- Permission-based design vs role-based
- Custom AccessDecisionVoter (advanced)

Bas bolo 👍



```
public class CustomUserDetails implements UserDetails {

    private User user;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return user.getRoles()
            .stream()
            .map(role -> new SimpleGrantedAuthority(role.getName()))
            .toList();
    }

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

```

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

@Override
public boolean isEnabled() {
    return user.isEnabled();
}
}

```

what is the use of this class and userDetailsService?

Excellent question.

Ab yahan **Spring Security ka heart** samajh aayega.

Main **bilkul flow-based, simple language** me samjha raha hoon.

Short Answer (one line)

- 👉 **CustomUserDetails** = Spring Security ko batata hai “ye user kaun hai & kya rights hain”
- 👉 **UserDetailsService** = Spring Security ko batata hai “user ko kahan se aur kaise laana hai”

1 Spring Security ko by default kya chahiye?

Spring Security internally sirf **ek cheez pe kaam karta hai**:

```

java
UserDetails

```

✖️ Tumhara **User** entity usse directly samajh nahi aata.

Isliye **adapter** banana padta hai.

2 CustomUserDetails ka exact role

Problem

```

text
Tumhara User entity (DB)
↓
Spring Security ko chahiye UserDetails

```

Solution

👉 CustomUserDetails = Adapter class

Is class ka kaam kya hai?

```
java
public class CustomUserDetails implements UserDetails {
```

Ye Spring Security ko batata hai:

Method	Meaning
getUsername()	User ka username kya hai
getPassword()	Encrypted password kya hai
getAuthorities()	User ke roles / permissions
isEnabled()	User active hai ya nahi

🔥 Sabse important method

```
java
getAuthorities()
```

Yahan:

- DB Role
- → GrantedAuthority
- → Authorization possible hota hai

Agar ye empty return kare:

✗ @PreAuthorize
 ✗ hasRole()
 ✗ sab fail

3 UserDetailsService ka role (Very Important)

Spring Security ka question:

"Login ke time mujhe user ka data kahan se milega?"

Answer:

👉 UserDetailsService

Interface

```
java

public interface UserDetailsService {
    UserDetails loadUserByUsername(String username);
}
```

Custom Implementation

```
java

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));

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

 **Yahin tum apna DB, JPA, Mongo, LDAP jo chaaho use karte ho**

4 Login ke time full flow (Step-by-step)

```
text

Login request aayi
↓
AuthenticationFilter
↓
AuthenticationManager
↓
UserDetailsService.loadUserByUsername()
↓
User DB se load hua
↓
CustomUserDetails bana
↓
Roles → GrantedAuthority
↓
Password verify
↓
SecurityContextHolder me store
```

 **Yahin authentication + authorization dono prepare ho jaate hain**

5 Agar ye dono na ho toh kya hogा?

✗ CustomUserDetails nahi banaya

- Spring ko pata hi nahi:
 - Username kya hai
 - Roles kya hain

- Authorization fail
-

✗ **UserDetailsService nahi banaya**

- Spring ko user kahan se laana hai pata hi nahi
 - Login fail
-

6 Interview One-Liner Answers 🔥

Q1: Why CustomUserDetails?

To adapt application-specific User entity into Spring Security's **UserDetailsService** model.

Q2: Why UserDetailsService?

To provide Spring Security with a way to load user information during authentication.

Q3: Can we skip CustomUserDetails?

Only if:

- Tum Spring ka default **User** use kar rahe ho
 - ✗ Real projects me almost never
-

7 Real-life Analogy 🧠

Real Life	Spring Security
ID Card	UserDetailsService
HR Database	UserDetailsService
Security Guard	Spring Security
Entry Pass	GrantedAuthority

Guard:

- ID card check karega (**UserDetailsService**)
 - Entry pass dekhega (**Authorities**)
-

Final Summary (Yaad rakhna)

- **CustomUserDetails** = User info + roles ko Spring format me convert karta hai
- **UserDetailsService** = User ko DB se laata hai

- Dono milke **authentication + authorization** possible banate hain
-

Agar chaho next:

- **AuthenticationProvider** ka role
- JWT me UserDetails ka use
- **SecurityContextHolder** deep dive
- Password matching ka internal flow

Bas bol 