Role-Based Access Control (RBAC) in Spring Boot - Detailed Concepts

1. What is RBAC?
Role-Based Access Control (RBAC) is a security paradigm that restricts system access based on the roles assigned to users. Instead of assigning permissions to each user individually, permissions are assigned to roles, and users are then granted roles. This approach makes managing user permissions easier and more secure.

Example:
- Role: ADMIN
- Permissions: Create User, Delete Product, View Reports
- User John is assigned the ADMIN role, so he inherits all the associated permissions.

2. Why Use RBAC?
RBAC is widely used because it simplifies permission management and enhances security:
- Security: Prevents unauthorized access by restricting access to critical functionalities.
- Maintainability: Easier to manage as the number of users grows.
- Scalability: Adaptable to growing teams and expanding features.
- Compliance: Helps ensure access policies comply with regulations (GDPR, HIPAA, etc.).

3. Key Concepts in RBAC
- Users: Entities who access the system (e.g., employees, customers).
- Roles: Defined job functions (e.g., ADMIN, USER, MANAGER).
- Permissions: Actions allowed in the system (e.g., VIEW_DASHBOARD, DELETE_USER).
- Sessions: User login sessions where roles are activated.

4. How RBAC Works in Spring Boot
Spring Boot integrates RBAC through Spring Security.

Main Components:
- Entities: User, Role, and optionally Permission.
- Authentication: Verifies user identity (e.g., username/password).
- Authorization: Determines access rights based on assigned roles.
- Security Configuration: Enforces role-based access to URLs or methods.

5. Typical Role Setup
Roles are typically designed according to business needs:

Role	Permissions
ADMIN	Manage users, products, categories, orders
USER	View products, manage own orders
MANAGER	View analytics, manage reports

6. Implementation Steps in Spring Boot

Step 1: Define Entities

```
@Entity
public class Role {
    @Id
    private Long id;
    private String name;
}
```

```
@Entity
public class User {
    @Id
    private Long id;
    private String username;
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;
}
```

Step 2: Create UserDetailsService

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        return new org.springframework.security.core.userdetails.User(
            user.getUsername(),
            user.getPassword(),
            user.getRoles().stream()
                .map(role -> new SimpleGrantedAuthority("ROLE_" + role.getName()))
                .collect(Collectors.toList())
        );
    }
}
```

Step 3: Configure Security

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests()
            .requestMatchers("/admin/**").hasRole("ADMIN")
            .requestMatchers("/user/**").hasRole("USER")
            .anyRequest().authenticated()
            .and()
            .formLogin();
        return http.build();
    }
}
```

7. Method-Level Security

```
@EnableMethodSecurity
public class MethodSecurityConfig {}
```

```
@PreAuthorize("hasRole('ADMIN')")
public void createProduct(Product product) {
    // Only admins can access
}
```

8. Role-based UI Access (Frontend)

```
<sec:authorize access="hasRole('ADMIN')">
  <a href="/admin/dashboard">Admin Panel</a>
</sec:authorize>
```

9. Advantages of RBAC
- Better security with minimal access.
- Centralized control of permissions.
- Easier auditing and tracking.
- Simplified user management.

10. Best Practices
- Define roles clearly based on job functions.
- Regularly review and update roles.
- Apply the least privilege principle.
- Avoid role explosion; use permission groups if needed.
- Use annotations or configuration to enforce access.