Spring Security:

Spring Security is a powerful and customizable authentication and access control framework for Java applications. It provides comprehensive security services for Java EE-based enterprise software applications. The primary goal of Spring Security is to provide a robust and flexible authentication and authorization mechanism.

Spring Security with Spring Boot:

When using Spring Boot, incorporating Spring Security into your application is straightforward. By adding the **spring-boot-starter-security** dependency, you can easily secure your application with sensible defaults.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Basic Authentication:

With Spring Security, basic authentication can be set up quickly. When it's enabled, the application will prompt users for a username and password. Here's a simple example of configuring basic authentication in Spring Security:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .anyRequest().authenticated()
                .and()
            .httpBasic();
    }
}
```

Authentication with User Credentials from Database and Authorization:

When dealing with user credentials stored in a database, can customize the authentication process. Typically, extend **UserDetailsService** and override the **loadUserByUsername** method to fetch user details from your database.

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not found with username: " + username));

        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                getAuthorities(user.getRoles())
        );
    }

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

JWT Authorization:

JWT (JSON Web Token) is a compact, URL-safe means of representing claims to be transferred between two parties. In the context of Spring Security, JWTs can be used for authorization.

1. Generate JWT:

use libraries like **jjwt** to generate JWTs.

```
String token = Jwts.builder()
    .setSubject(username)
    .setExpiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME))
    .signWith(SignatureAlgorithm.HS512, SECRET)
    .compact();
```

2. Configure Spring Security to use JWT:

Configure Spring Security to accept JWTs and perform authentication and authorization based on the token.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsServiceImpl userDetailsService;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STAT
ELESS)
            .and()
            .authorizeRequests()
                .antMatchers("/api/public").permitAll()
                .antMatchers("/api/private").authenticated()
            .and()
            .addFilter(new JwtAuthenticationFilter(authenticationManager()))
            .addFilter(new JwtAuthorizationFilter(authenticationManager(),
userDetailsService));
    }


    @Override
    public void configure(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordE
ncoder());
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```