1. What is Spring Security?

Answer:

Spring Security is a powerful, customizable authentication and access-control framework for Java applications, particularly Spring-based apps. It provides:

- Authentication (verifying user identity)
- Authorization (granting/denying access to resources)
- Protection against attacks (CSRF, session fixation, etc.)
- Integration with LDAP, OAuth2, JWT, and databases.

---

2. How does Spring Security work internally?

Answer:

Spring Security works through a filter chain (DelegatingFilterProxy → FilterChainProxy → SecurityFilterChain). Key steps:

1. Request enters the filter chain.
2. Authentication filters (e.g., `UsernamePasswordAuthenticationFilter`) check credentials.
3. AuthenticationManager validates credentials.
4. SecurityContextHolder stores the authenticated user.
5. Authorization filters (e.g., `FilterSecurityInterceptor`) check permissions.

---

3. What is the default authentication mechanism in Spring Security?

Answer:

- Form-based authentication (via `UsernamePasswordAuthenticationFilter`).
- Default login page (`/login`), logout (`/logout`).
- Uses HTTP sessions for security context.

---

4. What are the core components of Spring Security?

Answer:

Component	Purpose
`SecurityContextHolder`	Stores authentication details.
`AuthenticationManager`	Handles authentication logic.
`UserDetailsService`	Loads user-specific data.
`PasswordEncoder`	Encodes & verifies passwords.
`FilterChainProxy`	Manages security filters.
`AccessDecisionManager`	Decides if a user has access.

---

5. Explain the Spring Security filter chain.

Answer:

The filter chain processes requests in order:

1. SecurityContextPersistenceFilter – Restores security context from HTTP session.
2. UsernamePasswordAuthenticationFilter – Handles form login.
3. BasicAuthenticationFilter – Handles HTTP Basic Auth.

4. RememberMeAuthenticationFilter – Manages "remember-me" tokens.
5. AnonymousAuthenticationFilter – Assigns an anonymous user if not logged in.
6. FilterSecurityInterceptor – Checks authorization rules.

---

6. What is `UsernamePasswordAuthenticationFilter`?

Answer:

- A filter that processes form-based login (`/login`).
- Extracts `username` and `password` from the request.
- Creates an `Authentication` object and passes it to `AuthenticationManager`.

---

7. What is `SecurityContextHolder`?

Answer:

- Stores authentication details of the current user.
- Uses `ThreadLocal` by default (keeps data thread-safe).
- Accessed via:

```
java
Copy

Download
Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
```

---

8. Difference between `Principal` and `Authentication`?

Answer:

`Principal`	`Authentication`

Represents the logged-in user (just a name).	Contains full security details (user, credentials, roles).
`request.getUserPrincipal()`	`SecurityContextHolder.getContext().getAuthentication()`

---

9. What is `GrantedAuthority`?

Answer:

- Represents a permission/role (e.g., `ROLE_ADMIN`, `READ_PRIVILEGE`).
- Stored in `Authentication.getAuthorities()`.

---

10. How to configure Spring Security using `@EnableWebSecurity`?

Answer:

java

Copy

Download

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            .and()
            .formLogin();
    }
}
```

🔐 Authentication & Authorization

11. Difference between authentication and authorization?

Answer:

- Authentication (Who are you?) – Verifies identity (e.g., login).
- Authorization (What can you do?) – Checks permissions (e.g., `/admin` access).

---

12. How to implement custom authentication?

Answer:

1. Extend `AbstractAuthenticationToken`:
2. `public class CustomAuthToken extends AbstractAuthenticationToken { ... }`
3. Implement `AuthenticationProvider`:
4. `public class CustomAuthProvider implements AuthenticationProvider { ... }`
5. Register it in `SecurityConfig`:

```
@Bean
public AuthenticationProvider customProvider() {
    return new CustomAuthProvider();
```

6. `}`

Authentication & Authorization (Continued)

13. How to add role-based access control?

Answer:

Use `.hasRole()` or `.hasAuthority()` in `HttpSecurity`:

java

Copy

Download

```
http.authorizeRequests()
    .antMatchers("/admin/**").hasRole("ADMIN")
    .antMatchers("/user/**").hasAnyRole("ADMIN", "USER")
    .anyRequest().authenticated();
```

---

14. What is `@PreAuthorize`, `@Secured`, and `@RolesAllowed`?

Answer:

Annotation	Usage	Example
`@Secured`	Simple role check	`@Secured("ROLE_ADMIN")`
`@RolesAllowed`	JSR-250 alternative	`@RolesAllowed("ADMIN")`
`@PreAuthorize`	SpEL-based checks	`@PreAuthorize("hasRole('ADMIN')")`

Enable them with:

java

Copy

Download

```
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true,
jsr250Enabled = true)
```

---

15. How to secure URLs using `antMatchers`?

Answer:

java

Copy

Download

```
http.authorizeRequests()
    .antMatchers("/public/**").permitAll()
    .antMatchers("/private/**").authenticated()
    .antMatchers("/admin/**").hasRole("ADMIN");
```

---

16. What is method-level security? How to enable it?

Answer:

- Method-level security applies checks on individual methods (e.g., `@PreAuthorize`).
- Enable with:

- java
- Copy
- Download

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

- `public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration { ... }`

---

17. How to implement form-based login & logout?

Answer:

java

Copy

Download

```
http.formLogin()
    .loginPage("/custom-login")
    .defaultSuccessUrl("/home")
    .and()
    .logout()
    .logoutUrl("/custom-logout")
    .logoutSuccessUrl("/login");
```

---

18. How to restrict REST endpoints by roles?

Answer:

java

Copy

Download

```
@RestController
@RequestMapping("/api")
public class ApiController {
    @GetMapping("/admin")
    @PreAuthorize("hasRole('ADMIN')")
    public String adminOnly() { ... }
}
```

---

19. `hasAuthority()` vs `hasRole()`?

Answer:

`hasAuthority("ROLE_ADMIN")`	`hasRole("ADMIN")`
Checks exact authority string.	Automatically adds `ROLE_` prefix.

---

20. Can you disable CSRF? When is it safe?

Answer:

```
java
Copy
Download
http.csrf().disable();
```

When to disable:

- Stateless APIs (JWT/OAuth2).
- Non-browser clients (mobile apps).

🔁 JWT & OAuth2

21. How does JWT work in Spring Security?

Answer:

1. User logs in → Server generates JWT.
2. Client stores JWT (localStorage/cookie).
3. Each request includes JWT in `Authorization: Bearer <token>`.
4. Server validates JWT using a secret key.

---

22. Key components of JWT?

Answer:

- Header (`alg: HS256`, `typ: JWT`)
- Payload (`sub: user123`, `roles: ["ADMIN"]`)
- Signature (HMAC + secret key)

---

23. How to implement JWT in Spring Boot?

Answer:

1. Add dependency:

```
xml
Copy

Download
Run
```

```xml
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
```

```
</dependency>
```

7. Create `JwtTokenFilter` (extends `OncePerRequestFilter`).
8. Configure `JwtAuthenticationProvider`.

---

24. How does OAuth2 work in Spring Security?

Answer:

- Roles:
    1. Resource Owner (User)
    2. Client (App)
    3. Authorization Server (Issues tokens)
    4. Resource Server (Validates tokens)
- Flow:
    1. User grants permission → Client gets authorization code.
    2. Client exchanges code for access token.
    3. Client accesses APIs with the token.

---

25. Access Token vs Refresh Token?

Answer:

Access Token	Refresh Token
Short-lived (1 hour).	Long-lived (7 days).
Used for API calls.	Used to get a new access token.

---

26. How to store JWT securely?

Answer:

- Frontend: Store in `HttpOnly` cookies (prevents XSS).
- Backend: Validate signature + expiration.

---

27. Stateless vs Stateful Authentication?

Answer:

Stateless (JWT)	Stateful (Sessions)
No server-side storage.	Uses server-side sessions.
Scalable (microservices).	Less scalable (session replication needed).

---

28. How to implement stateless auth?

Answer:

java

Copy

Download

```
http.sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
```

🔧 Advanced Customization

29. How to customize `UserDetailsService`?

Answer:

java

Copy

Download

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) {
        // Fetch user from DB and return UserDetails
    }
}
```

---

30. `UserDetails` vs `User`?

Answer:

- `UserDetails` is an interface (contract for user data).
- `User` is Spring's default implementation.

---

31. How to define a custom `AuthenticationProvider`?

Answer:

java

Copy

Download

```java
@Component
public class CustomAuthProvider implements AuthenticationProvider {
    @Override
    public Authentication authenticate(Authentication auth) {
        // Custom logic
    }
}
```

**32. Purpose of `AuthenticationManager`?

Answer:

- Central interface for authentication.
- Delegates to `AuthenticationProvider`s.

---

33. How to handle access denied exceptions?

Answer:

java

Copy

Download

```java
http.exceptionHandling()
    .accessDeniedHandler((request, response, ex) -> {
        response.sendRedirect("/access-denied");
    });
```

---

34. What are `AccessDecisionManager` & `AccessDecisionVoter`?

Answer:

- `AccessDecisionVoter`: Votes on access (e.g., `RoleVoter`).
- `AccessDecisionManager`: Tallies votes to grant/deny access.

---

35. How to implement "remember-me"?

Answer:

java

Copy

Download

```
http.rememberMe()
    .key("uniqueAndSecret")
    .tokenValiditySeconds(86400); // 1 day
```

---

36. How to integrate Spring Security with a database?

Answer:

java

Copy

Download

```
@Autowired
private DataSource dataSource;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery("SELECT username, password, enabled FROM users WHERE
username=?")
        .authoritiesByUsernameQuery("SELECT username, role FROM roles WHERE
username=?");
}
```

---

37. What is `BCryptPasswordEncoder`?

Answer:

- Hashes passwords using BCrypt (salted, slow hashing).
- Usage:

- java
- Copy
- Download

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

- }

---

38. How to handle session management?

Answer:

java

Copy

Download

```
http.sessionManagement()
    .maximumSessions(1)
    .maxSessionsPreventsLogin(true); // Prevent concurrent logins
```

⚙️ Real-World Scenarios

39. How to secure a REST API with JWT?

Answer:

1. Add JWT dependencies.
2. Create `JwtTokenUtil` (generates/validates tokens).
3. Implement `JwtRequestFilter` (checks tokens in requests).
4. Configure `HttpSecurity` for stateless auth.

---

40. Single Sign-On (SSO) for microservices?

Answer:

- Use OAuth2 (Keycloak, Okta).
- JWT for stateless communication.

---

41. Method-level vs URL-level security?

Answer:

- Method-level: Fine-grained (e.g., `@PreAuthorize`).
- URL-level: Coarse-grained (e.g., `.antMatchers()`).

---

42. Prevent brute force attacks?

Answer:

- Use `RateLimitingFilter`.
- Lock accounts after N failed attempts.

---

43. What if a JWT is stolen?

Answer:

- Short expiry (15-30 mins).
- Refresh tokens (rotatable).
- Blacklist tokens (if using a token store).

---

44. How to manually invalidate a JWT?

Answer:

- Option 1: Maintain a blacklist in Redis.
- Option 2: Change the JWT signing key.

---

45. How to test security configurations?

Answer:

java

Copy

Download

```java
@SpringBootTest
@AutoConfigureMockMvc
public class SecurityTest {
    @Test
    @WithMockUser(roles = "ADMIN")
    public void testAdminEndpoint() throws Exception {
        mockMvc.perform(get("/admin")).andExpect(status().isOk());
    }
}
```

🧪 Testing

46. How to test secure endpoints?

Answer:

- Use `@WithMockUser`:

- java
- Copy

- Download

```
@Test
@WithMockUser(username = "user", roles = {"USER"})
```

- `public void testUserEndpoint() { ... }`

---

47. What is `@WithMockUser`?

Answer:

- Simulates an authenticated user in tests.
- Example:

- java
- Copy
- Download

```
@WithMockUser(username = "admin", roles = {"ADMIN"})
```

---

48. How to test method-level security?

Answer:

java

Copy

Download

```
@SpringBootTest
public class MethodSecurityTest {
    @Autowired
    private SecuredService service;

    @Test
    @WithMockUser(roles = "ADMIN")
    public void testAdminMethod() {
        assertNotNull(service.adminOnlyMethod());
    }
}
```