1. What is Spring Security and why do we use it?

Answer:
Spring Security is a framework to secure Spring applications. It handles authentication (who are you?) and authorization (what can you access?).

Real-time example:
In your food delivery app:

- Only **registered users** can place orders.
- Only **admins** can view all orders.
  Spring Security ensures this access control is enforced.

2. How does Spring Security handle authentication?

Answer:
Spring Security provides filters that intercept requests and authenticate using methods like form login, JWT, OAuth2, etc.

Real-time example:
A user logs in to your app. Spring Security validates their username/password via the UserDetailsService, generates a session or JWT, and then lets them access /place-order.

3. What is the difference between Authentication and Authorization?

Answer:

- **Authentication**: Verifying the user's identity.
- **Authorization**: Granting access to specific resources based on roles.

Real-time example:

- **Authentication**: "Are you Sathya? Prove it with your password."
- **Authorization**: "Now that you're Sathya (authenticated), can you access /admin/view-all-orders? Only if you're an ADMIN."

4. What are roles and authorities in Spring Security?

Answer:
Roles are high-level groupings (like ADMIN, USER). Authorities are specific permissions (like READ_PRIVILEGES, WRITE_PRIVILEGES).

Real-time example:

- Role: ADMIN
- Authorities: CAN_ADD_MENU, CAN_DELETE_ORDER
  This setup lets you fine-tune access within the same role.

5. What is JWT and how does Spring Security support it?

Answer:
JWT (JSON Web Token) is a compact token used to authenticate users in a **stateless** way—no session needed.

Real-time example:
Your mobile app logs in once, receives a JWT, and sends it in each request header like:

makefile

CopyEdit

Authorization: Bearer <JWT-TOKEN>

Spring Security validates this token on every API call, ensuring secure and stateless communication.

6. How can you secure specific URLs in Spring Security?

Answer:
Use HttpSecurity configuration to restrict access by URL and roles.

Real-time example:

java

CopyEdit

http

.authorizeRequests()

.antMatchers("/admin/**").hasRole("ADMIN")

.antMatchers("/user/**").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated();

This ensures:

- /admin/orders → only for ADMINs
- /user/view-orders → for USER and ADMIN

7. How do you implement OAuth2 login with Spring Security?

Answer:
Spring Security provides easy OAuth2 integration (Google, GitHub, etc.). It handles token generation, login redirect, and user info fetching.

Real-time example:
Your food delivery app allows users to "Login with Google". Behind the scenes:

- Spring Security redirects to Google.
- Google returns user profile and token.
- Spring Security logs them in using that token.

8. How do you define custom user roles and secure endpoints accordingly?

Answer:
Define roles in your user model and secure endpoints using @PreAuthorize or antMatchers().

Real-time example:
In a **Learning Management System**:

- @PreAuthorize("hasRole('STUDENT')") → for accessing lessons
- @PreAuthorize("hasRole('ADMIN')") → for managing users and content

9. What is CSRF and how is it handled in Spring Security?

Answer:
CSRF (Cross Site Request Forgery) is an attack where an unauthorized command is sent from a user that the website trusts. Spring Security enables CSRF protection by default.

Real-time example:
In a banking app, CSRF prevents users from being tricked into transferring money by clicking a hidden link on a malicious site.

10. How do you create a custom login page in Spring Security?

Answer:
Override the default login page by configuring .loginPage("/custom-login").

Real-time example:
In your **E-commerce app**, instead of the Spring default form, you create a stylish login UI. Spring Security uses this to authenticate users.

11. What is method-level security and how do you implement it?

Answer:
Method-level security protects specific methods using annotations like @Secured, @PreAuthorize, @PostAuthorize.

Real-time example:
In a **Healthcare app**, a doctor can view their patients:

java

CopyEdit

@PreAuthorize("hasRole('DOCTOR')")

public List<Patient> getMyPatients() {}

12. How do you store and retrieve user details in Spring Security?

Answer:
Implement UserDetailsService and load users from a DB.

Real-time example:
In a **job portal**, users and recruiters register. You store their credentials and roles in MySQL and load them like:

java

CopyEdit

UserDetails user = userRepository.findByUsername(username);

13. How do you implement remember-me functionality?

Answer:
Use .rememberMe() in Spring Security config and store tokens in DB or cookies.

Real-time example:
In an e-learning site, students who click “Remember me” stay logged in for 7 days—even if they close the browser.

14. What is the difference between form login and JWT-based login?

Answer:

- **Form Login**: Creates a session after login.
- **JWT**: Stateless login using tokens sent in headers.

Real-time example:

- **Form login**: Good for web apps with stateful sessions.
- **JWT**: Ideal for mobile apps or REST APIs. The client stores the token and uses it in every request.

15. How do you log out a user in Spring Security?

Answer:
Use .logout() in configuration. Spring Security clears the session or JWT.

Real-time example:
In a banking application, clicking "Logout" invalidates the session and redirects to the login page to ensure no sensitive info is cached.