

## Quiz Questions

### (1) Authentication vs Authorization

← Classroom

Introduction to Spring Security  
Authentication vs Authorization

?

V

Problem Submissions Doubts

✓ Authica vs athoriz

Easy • Score 20/20

Send feedback

Which of the following best describes the relationship between authentication and authorization?

Options: Pick one correct answer from below

☐ Authentication and authorization are the same concepts used interchangeably.

☐ Authentication is a subset of authorization.

☐ Authentication and authorization are not related to application security.

☒ Authorization determines what a user can do, while authentication verifies the user's identity. ✓

Solution description

The correct answer is (d) Authorization determines what a user can do, while authentication verifies the user's identity. Authentication establishes who the user is by checking their credentials, while authorization determines what the authenticated user can do, i.e. the resources(APIs) he can access in the system.

### (2) Example of Authentication

← Classroom

Introduction to Spring Security  
Example of Authentication

?

V

Problem Submissions Doubts

✓ E.G of authentication

Easy • Score 20/20

Send feedback

Which of the following is an example of authentication?

Options: Pick one correct answer from below

☐ Checking if a user has admin privileges to access an admin panel

☒ Verifying a user's email address during registration ✓

☐ Allowing a user to view their profile page

☐ Setting up access control lists (ACLs) for a file system

Solution description

The correct answer is (b) Verifying a user's email address during registration. This is an example of authentication because it involves confirming the user's provided information (email address) to create an account.

### (3) Remember Me

← Classroom

Introduction to Spring Security  
Remember Me

?

V

Problem Submissions Doubts

✓ Remember me

Easy • Score 20/20

Send feedback

What is the "Remember-Me" feature in web application security?

Options: Pick one correct answer from below

☐ A feature that remembers the user's favourite web pages

☒ A feature that automatically logs in the user on subsequent visits without requiring credentials ✓

☐ A feature that secures payment information during online shopping

☐ A feature that prevents session timeouts

Solution description

- Remember-Me session management in Spring Security enables users to stay logged in even after they close their browser or log out, avoiding re-entering their credentials on return visits.
- It generates a secure token during the initial login, stores it in a database, and distributes a cookie to the user's browser. When the user returns, the token is checked and, if valid, grants automatic access. While enhancing user convenience, it's crucial to implement security measures to protect against misuse, making it suitable for less sensitive applications.

#### (4) Third Party Authentication

← Classroom

Introduction to Spring Security  
Third Party Authentication

?

V

Problem Submissions Doubts

✓ Third party auth

Easy • Score 20/20

Problem statement

Which of the following mechanisms provide third-party authentication?

Send feedback

Options: Pick one correct answer from below

☒ openID Connect

☐ JWT (JSON Web Tokens)

☐ HTTP Basic Authentication

☐ Form-based Authentication

Solution description

OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 protocol designed to facilitate secure user authentication and authorization in modern web and mobile applications. It allows users to log in to a service using their preferred identity provider (like Google, Facebook, or an organization's identity server) and grants applications access to user information in a standardised and secure manner.

#### (5) Base-64 encoding

← Classroom

Introduction to Spring Security  
Mechanism that involves Encoding

?

V

Problem Submissions Doubts

✓ base 64 encoding

Easy • Score 20/20

Problem statement

Which authentication mechanism use base64 encoding to encode username and passwords?

Send feedback

Options: Pick one correct answer from below

☐ O.Auth

☐ JWT Authentication

☒ HTTP basic authentication

☐ Form-based authentication

Solution description

HTTP Basic Authentication uses base64 encoding to encode usernames and passwords before sending them over the network. When a client requests a server that requires HTTP Basic Authentication, the client includes an "Authorization" header in the request with the encoded username and password. The server then decodes this information to authenticate the user.

#### (6) LDAP

← Classroom

Introduction to Spring Security  
LDAP

?

V

Problem Submissions Doubts

✓ LDAP

Easy • Score 20/20

Problem statement

In which of the following scenarios is LDAP often employed?

Send feedback

Options: Pick one correct answer from below

☐ Secure email communication

☐ Social media account creation

☒ Managing employee access to corporate resources

☐ Streaming video content

Solution description

LDAP (Lightweight directory access protocol) is commonly used in corporate environments to manage employee access to servers, applications and network resources.

## (7) Job Board Platform I

← Classroom

Introduction to Spring Security  
Job Board Platform I

75% - + Reset

Problem Submissions Doubts

Job Board Platform

Easy • Score 20/20

Send feedback

**Problem statement**

In a job board platform, recruiters can create job listings. Users can search for jobs and apply to them. The platform uses DTOs (Data Transfer Objects) and Lombok for efficient data management. The Entity and the Repository are provided below.

Entity Class:

```
#####
public class JobListing {
    private Integer id;
    private String title;
    private String description;
} // Getters and Setters method
#####
```

Repository class:

```
#####
public interface JobListingRepository extends JpaRepository<JobListing, Integer> {}
#####
```

Which code snippet correctly defines a DTO class with Lombok annotations for generating common methods?

#####

public class JobListingDTO {  
 private Integer id;  
 private String title;  
 private String description;  
}

#####

@Data  
public class JobListingDTO {  
 private String title;  
 private String description;  
}

#####

@Entity  
public class JobListingDTO {  
 private String title;  
 private String description;  
}

#####

@Getter  
@Setter  
@Data  
public class JobListingDTO {  
 private String title;  
 private String description;  
}

**Solution description**

Option B is correct because the JobListingDTO class is annotated with @Data annotation, automatically generating getters, setters, and other relevant methods for the relevant fields, i.e. title and description. Thus, it is an appropriate DTO class.

Option A is incorrect because the 'id' field is also added for the JobListingDTO class, which we don't need to use. Only the title and description are relevant fields required to map with the JobListing entity class object where the

## (8) Job Platform II

← Classroom

Introduction to Spring Security  
Job Board Platform II

75% ? V

Problem Submissions Doubts

Job Board Platform II

Easy • Score 20/20

Send feedback

**Problem statement**

What is the primary purpose of using Lombok in this scenario?

Options: One or more answers may be correct

☐ To transfer data through methods and layers.

☒ To use Lombok annotations for generating getters, setters and other methods. ✓

☐ To handle HTTP requests.

☒ To reduce the boilerplate code. ✓

**Solution description**

The correct options are Option B & D both because Lombok provides annotations such as @Data, @AllArgsConstructor, and @NoArgsConstructor, which autogenerates getters, setters, respectively, for all the fields, parameterised constructor, and default constructor of the JobListingDTO class. And thus, it also helps in reducing the boilerplate code. ##### Option A is incorrect because DTO (Data Transfer Object) enables data transfer through methods and layers using DTO objects, whereas Lombok is a Java library. ##### Option C is incorrect, as Lombok is not responsible for handling HTTP requests.

## (9) Job Board Platform III

Classroom

Introduction to Spring Security  
Job Board Platform III

Problem

Submissions

Doubts

Job Board Platform III

Easy • Score: 20/20

Send feedback

Problem statement

Below is the code for the "JobListingDTO" and "JobListingService" classes. What will be the correct implementation of the "createJobListing" in the service layer if "JobListingDTO jobListingDto" is passed as an argument in the method?

DTO Class Code

```
// DTO Class Code
@Data
public class JobListingDTO {
    private String title;
    private String description;
}
```

Service Class Code

```
// Service Class Code
@Service
public class JobListingService {
    private final JobListingRepository repository;

    public JobListing createJobListing(JobListingDTO jobListingDto) {
        // Implementation logic
    }
}
```

Options: Pick one correct answer from below

☐ Option A

```
public JobListing createJobListing(JobListingDTO jobListingDto) {
    return repository.save(jobListingDto);
}
```

☐ Option B

```
public JobListing createJobListing(JobListing jobListing) {
    JobListingDTO jobListingDto = new JobListingDTO();
    jobListingDto.setTitle(jobListing.getTitle());
    jobListingDto.setDescription(jobListing.getDescription());
    return repository.save(jobListingDto);
}
```

☒ Option C

```
public JobListing createJobListing(JobListingDTO jobListingDto) {
    JobListing jobListing = new JobListing();
    jobListing.setTitle(jobListingDto.getTitle());
    jobListing.setDescription(jobListingDto.getDescription());
    return repository.save(jobListing);
}
```

☐ Option D

```
public JobListing createJobListing(JobListingDTO jobListingDto) {
    // Create a new JobListing object
    JobListing jobListing = new JobListing();
    return repository.save(jobListing);
}
```

Solution description

Option C is correct because it creates a new JobListing entity, sets its properties based on the values from the provided "JobListingDto", and then saves the entity in the repository (since we can only save entity in the repository). This is the proper way to create a new JobListing record using the data from a DTO. ##### Whereas in other options like Option A, it tries to save the JobListingDTO directly into the repository, but the repository expects a JobListing entity. It doesn't perform the necessary mapping from the DTO to the entity. ##### Option B attempts to save a JobListingDTO object directly into the repository, which is incorrect. You should save a JobListing entity in the repository, not a DTO. Moreover, it creates a new JobListingDTO object unnecessarily. ##### And Lastly, Option D is incorrect because it attempts to call a save() method on the JobListing object, but such a method is not in the JobListing class. Additionally, it doesn't show how the mapping between the DTO and the entity is performed. Correct mapping logic is missing, making this option incorrect.

## (10) Default Auth Mechanism

Classroom

Introduction to Spring Security  
Default Authentication Mechanism

Problem

Submissions

Doubts

Default auth mechanism

Easy • Score: 20/20

Send feedback

Problem statement

Which of the following is the default authentication mechanism of the spring security framework without configuration?

Options: Pick one correct answer from below

☐ HTTP basic authentication

☒ Form-based authentication

☐ JWT authentication

☐ None of the above

Solution description

The default authentication mechanism for Spring Security is form-based Authentication. Form-based authentication involves presenting users with a login form, where they enter their credentials, and then the credentials are sent to the server for validation.

## (11) Auth Flow

Classroom

Introduction to Spring Security  
Authentication Flow

Problem

Submissions

Doubts

auth flow

Easy • Score: 20/20

Send feedback

Problem statement

What is the flow of the authentication mechanism that processes the incoming API request?

Options: Pick one correct answer from below

☐ Authentication manager -> Authentication filter-> Authentication provider

☐ Authentication provider -> Authentication manager -> Authentication filter

☒ Authentication filter -> Authentication manager -> Authentication provider

☐ None of the above

Solution description

The correct option is (C) Authentication filter -> Authentication manager -> Authentication provider

- Authentication filter: The incoming API request first passes through one or more authentication filters. These filters are responsible for extracting authentication credentials from the request (e.g., username and password from HTTP headers or tokens) and creating an authentication object.
- Authentication manager: Once the authentication filter has created the authentication object, it forwards it to the authentication manager. The authentication manager's role is to verify the authenticity of the credentials provided in the authentication object. It does this by delegating the authentication to one or more authentication providers.
- Authentication provider: Authentication providers are responsible for performing the actual authentication. They validate the credentials against the configured user store (e.g., a database, LDAP, or an in-memory store) and return an authenticated user object if the credentials are valid.

## (12) Job Board Platform IV

Classroom

Introduction to Spring Security  
Job Board Platform IV

Problem Submissions Doubts

Job Board Platform IV

Easy • Score 20/20

Problem statement

Send feedback

The JobListing application is now ready with Lombok and DTO, and there is now a `JobListingSecurityConfig` class for custom users. Based on the previous and current code, identify the class-level annotation that should be used to make Spring Security Accessible in the `JobListingSecurityConfig` class.

```
***
@Configuration
public class JobListingSecurityConfig {
    // The code to configure custom user with form based authentication
}
```

Options: Pick one correct answer from below

☐ @EnableGlobalMethodSecurity

☐ @SpringBootApplication

☒ @EnableWebSecurity

☐ @ComponentScan

Solution description

Option C is the correct answer as the `@EnableWebSecurity` annotation enables Spring Security configuration in a Java class. This annotation typically indicates that the class contains Spring Security configuration settings. ### Option a is incorrect as `@EnableGlobalMethodSecurity` is not required to enable Spring Security configuration in a Java class. It is used to enable method-level security, which we will study later. ### Option b is also an incorrect option as here `@SpringBootApplication` is used, which marks the main application class with Spring Boot configuration. It is not explicitly related to Spring Security configuration. ### Option d is also incorrect as here `@ComponentScan` is used, which specifies the base packages to scan for Spring components. While it's a common annotation in Spring applications, it is not specifically related to Spring Security configuration.

## (13) Job Board Platform V

Classroom

Introduction to Spring Security  
Job Board Platform V

Problem Submissions Doubts

Job Board Platform V

Easy • Score 20/20

Problem statement

Send feedback

Given Below is the code for the `JobListingSecurityConfig` class for custom users. However, a key component is missing in the class code. Select the option that best describes what is wrong (or missing) in this code.

```
***
@Configuration
@EnableWebSecurity
public class JobListingSecurityConfig {
    // ...

    public SecurityFilterChain (filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeHttpRequests()
            .addRequestMatchers()
            .authenticated()
            .and()
            .formLogin()
            .return http.build();
    }

    // ...

    @Bean
    public UserDetailsServiceImpl users(){
        UserDetails user1 = User.builder()
            .username("john")
            .password("password1")
            .roles("guest")
            .build();

        UserDetails user2 = User.builder()
            .username("steve")
            .password("password2")
            .roles("guest")
            .build();

        return new InMemoryUserDetailsManager(user1,user2);
    }
}
```

Options: Pick one correct answer from below

☐ A)

☐ B)

☐ C)

☒ D)

Solution description

Option d is correct as the bean definition for `PasswordEncoder` is necessary because Spring Security requires a `PasswordEncoder` to securely encode passwords before storing them and to compare passwords during authentication. Also, the code will throw a compilation error as the `passwordEncoder()` method cannot be resolved without the bean definition. The other options either suggest unnecessary changes or incorrect practices. ### Option A is incorrect because this option suggests that the roles assigned to user1 and user2 should be different. While having the same role for both users is not necessarily a problem. Although having different roles based on user permissions is a good practice it is dependent on application requirements. ### Option B suggests that there is no need to use the `passwordEncoder()` method for encoding passwords. However, it is crucial to use a password encoder to encode passwords before storing or comparing them during authentication security. Storing plaintext passwords is a security risk, so using `passwordEncoder()` is necessary. Also, the application will throw an `IllegalArgumentException` after trying to log in from the `localhost:8082/login` API using the username and password, as the application won't be able to encode the password for the user coming from the client side. Thus, Option B is also incorrect. ### Option C will throw a compilation error as here `PasswordEncoder` cannot be instantiated as it is an interface. This option suggests defining a `PasswordEncoder` bean without specifying a specific `PasswordEncoder` implementation, and therefore, it is also an incorrect option.

- A) The roles assigned to user1 and user2 should be different.  
b) No need to use passwordEncoder().

```
***
@Bean
public UserDetailsServiceImpl users(){
    UserDetails user1 = User.builder()
        .username("john")
        .password("password1")
        .roles("guest")
        .build();

    UserDetails user2 = User.builder()
        .username("steve")
        .password("password2")
        .roles("guest")
        .build();

    return new InMemoryUserDetailsManager(user1,user2);
}
```

- c) A bean of PasswordEncoder should be defined as below:

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

- d) A bean of PasswordEncoder should be defined as below:

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

## (14) Ant Matchers

Classroom

Introduction to Spring Security

AntMatchers

Problem

Submissions

Doubts

antmatchers

Easy • Score 20/20

Send feedback

Problem statement

AntMatchers support both positive and negative expressions, allowing you to permit or deny access to specific URL patterns.

True

False

Solution description

AntMatchers provide a flexible way to define security rules based on URL patterns, and you can use them to either permit or deny access to specific URL paths based on roles or other criteria. This flexibility allows you to finely control access to different parts of your application based on your security requirements. For example:

```
// Permit access to the home page for all users
.antMatchers("/").permitAll()
// Permit access to /public/** for all users
.antMatchers("/public/**").permitAll()
// Deny access to /admin/** for users without the ADMIN role
.antMatchers("/admin/**").hasRole("ADMIN")
// Deny access to /private/** for users without the USER role
.antMatchers("/private/**").hasRole("USER")
// Deny access to /private/secret/** for users without the USER role
.antMatchers("/private/secret/**").hasAnyRole("USER", "ADMIN")
// All other requests must be authenticated
```

## (15) Job Platform VI

Classroom

Introduction to Spring Security

Job Board Platform VI

Problem

Submissions

Doubts

Job Board Platform VI

Easy • Score 20/20

Send feedback

Problem statement

The application has been updated with role-based authorisation with basic HTTP authentication. The application uses the URL "localhost:8082/createJobListing" to create jobListing entities in the database and should be used by Admins only. Based on this condition, which option best replaces the "filterChain" method?

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests()
        .antMatchers("/createJobListing").hasRole("ADMIN")
        .and()
        .httpBasic();
    return http.build();
}
```

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests()
        .antMatchers("/createJobListing").hasRole("USER")
        .and()
        .httpBasic();
    return http.build();
}
```

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests()
        .antMatchers("/createJobListing").hasAuthority("ADMIN")
        .and()
        .httpBasic();
    return http.build();
}
```

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests()
        .antMatchers("/createJobListing").hasAuthority("user")
        .and()
        .httpBasic();
    return http.build();
}
```

Solution description

Option B is the correct answer because the code correctly configures Spring Security to achieve role-based authorization for the "createJobListing" URL, allowing only users with the "ADMIN" role to access it. This matches the requirement of our problem statement, thus making it the correct answer. ##### Option A is incorrect because this code snippet specifies that the "createJobListing" URL should have the role "USER" to access it. However, the requirement is to allow

## (16) Job Board Platform VII

← Classroom

Introduction to Spring Security  
Job Board Platform VII

?

V

Problem Submissions Doubts

✓ Job Board platform VII

Easy • Score 20/20

Problem statement

Send feedback

What will be the response status if an authenticated user with the role "USER" is trying to access the following API: "/deleteJobPosting/id/{id}".

Options: Pick one correct answer from below

☐ 401 Unauthorized

☒ 403 Forbidden

☐ 200 OK

☐ 404 Not Found

Solution description

Option B is the correct choice because a user with a role other than "ADMIN" is not allowed to access the "/deleteJobPosting/id/{id}" API, resulting in a 403 Forbidden status code. ### Option A is incorrect since the user is authenticated but lacks the authorized role for accessing the "/deleteJobPosting/id/{id}" API, leading to a 401 error code typically received when an unauthenticated user (with incorrect credentials) tries to access the API. ### Option C is also incorrect because the application doesn't grant access to users without an admin role for the "/deleteJobPosting/id/{id}" API, thus resulting in a status code other than 200. ### Option D is also incorrect because the API path is found in the JobListingController class, so a 404 response is not expected.

## (17) Ant Matchers II

← Classroom

Introduction to Spring Security  
AntMatchers II

?

V

Problem Submissions Doubts

✓ antmatchers 2

Easy • Score 20/20

Problem statement

Send feedback

Below are four APIs that the users can use. Based on the details given, which of the options do you think will grant access to users with the "ADMIN" role to all the API and "NORMAL" roles for users to access only the "order" and "profile" API paths?  
  
"/user/dashboard": API to fetch dashboard details for a user  
"/user/settings": API to see fetch available settings for a user  
"/user/profile/edit": API to edit Profile of a user  
"/user/orders": API to fetch all the order posted by a user

Options: Pick one correct answer from below

☐ antMatchers("/user").permitAll("ADMIN").antMatchers("/users/\*\*").permitAll("NORMAL")

☒ antMatchers("/user/\*\*").hasRole("ADMIN").antMatchers("/user/profile/edit", "/user/orders").hasRole("NORMAL")

☐ antMatchers("/users/\*\*").admin().authenticated().antMatchers("/user/profile/edit", "/user/orders").normal().authenticated()

☐ antMatchers("/user/\*\*").denyAll("NORMAL").antMatchers("/user/\*\*").allowAll("ADMIN")

Solution description

- Option B, correctly uses hasRole to specify that users with the "ADMIN" role should have access to all paths under "/user/\*\*" and users with the "NORMAL" role should have access only to the "/user/profile/edit" and "/user/orders" paths.
- Option A is incorrect because it misuses the permitAll method. permitAll is typically used to allow unrestricted access to specific URLs or patterns, not to specify roles like "ADMIN" or "NORMAL." The correct usage would be to define which URLs should be accessible to everyone (regardless of role).
- Option C is incorrect because it uses non-standard methods like admin() and normal() which are not recognized by Spring Security. There is no way to specify roles using these methods.
- Option D is incorrect because it uses non-standard methods like denyAll() and allowAll(), which are not recognized by Spring Security. There is no way to specify roles using these methods.