Spring Professional Exam Tutorial v5.0

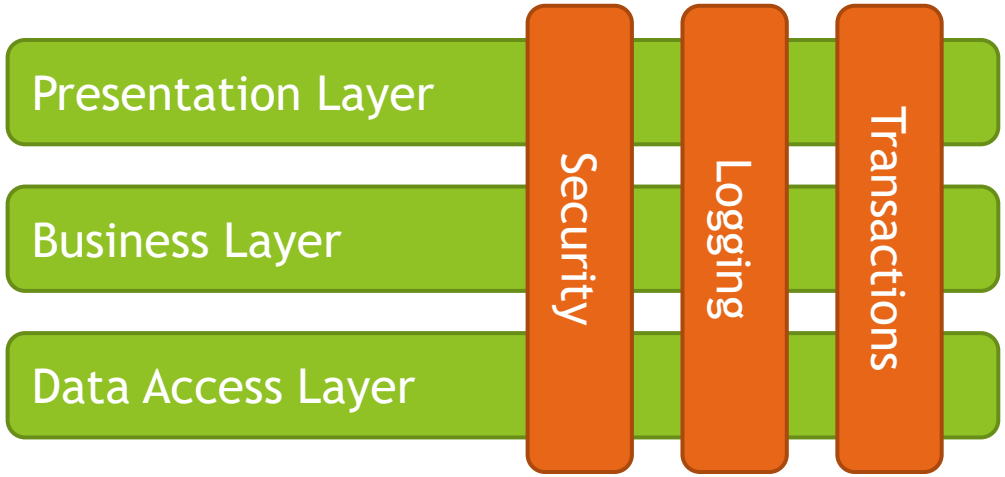
Question 02

Question 02 - Is security a cross cutting concern? How is it implemented internally?

Yes, Security is a cross cutting concern.

Cross cutting concern – functionality of a program not immediately associated with the business logic, applicable throughout the application and affecting multiple parts of the system.

Security fits well into above definition, other examples of cross cutting concerns include functionalities like logging or transactions.



Question 02 - Is security a cross cutting concern? How is it implemented internally?

Security in Spring is implemented on two levels:

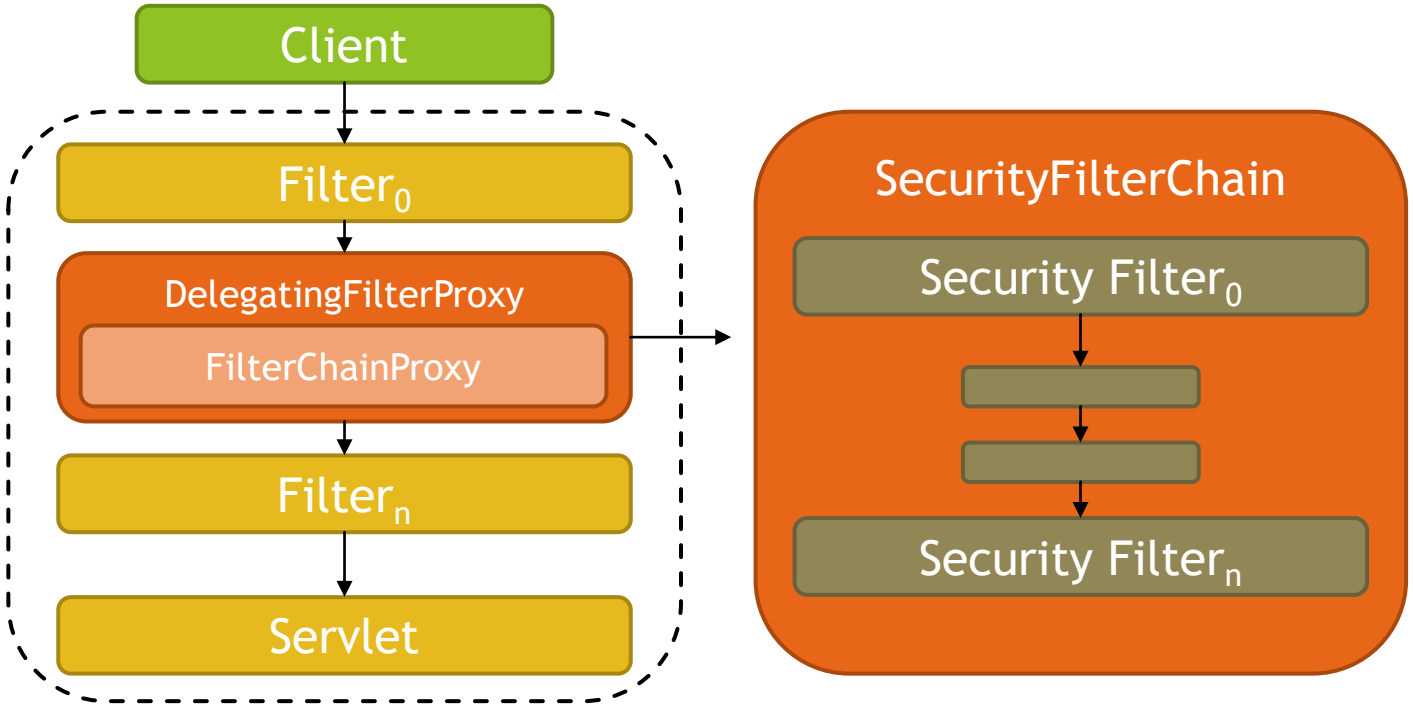
- Web Level – based on Servlet Filters
- Method Security Level – based on Spring AOP

Each aspect of Security – Authentication and Authorization is handled on both of those levels with different set of components:

- Authenthicaiton
  - AuthenticationManager
  - ProviderManager
  - AuthenticationProvider
  - UserDetailsService
- Authorization
  - AccessDecisionManager
    - AccessDecisionVoter
  - AfterInvocationManager
  - Authorities

Question 02 - Is security a cross cutting concern? How is it implemented internally?

Web Level Spring Security uses Servlet Filters to analyze each request made to the system, and based on rules specified through `WebSecurityConfigurerAdapter` and `HttpSecurity` object, performs certain decision against authentication or authorization. Such decision may include redirecting request to login page, or rejecting request because of roles not being assigned to the user.



Question 02 - Is security a cross cutting concern? How is it implemented internally?

Method Security Level uses Spring AOP to proxy invocations to objects, applied advices ensures that during invocation, security rules are met to allow invocation, for example user needs to contain set of `roles/authorities` to execute method.

To enable method level security you need to use `@EnableGlobalMethodSecurity` annotation and enable support to one of annotation types:

- `prePostEnabled` - Security's pre post annotations - `@PreAuthorize`
- `securedEnabled` - `@Secured` annotation - Spring Security's `@Secured` annotations
- `jsr250Enabled` - JSR 250 annotations `@RolesAllowed, @PermitAll, @DenyAll, ...`

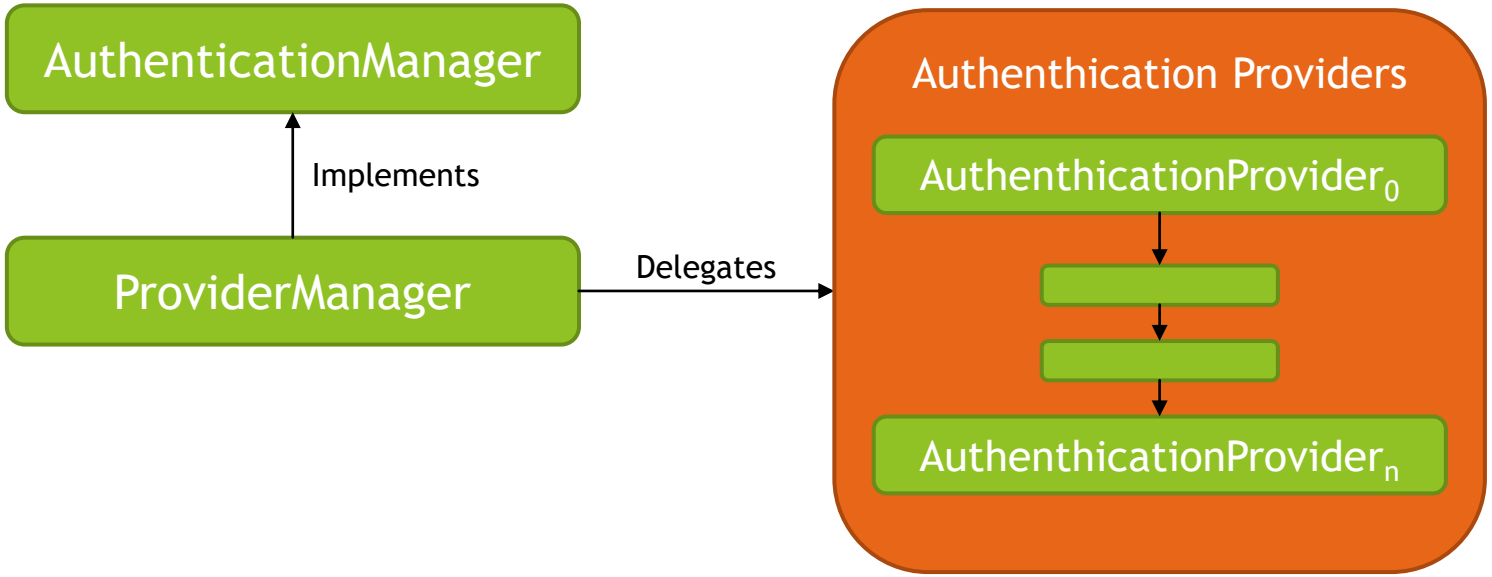
Question 02 - Is security a cross cutting concern? How is it implemented internally?

Spring Security uses many objects to implement security:

- `SecurityContextHolder` - heart of Spring Security authentication model, place were Spring stores the details of who is authenticated
- `SecurityContext` - held by `SecurityContextHolder`, gives access to `Authentication` object
- `Authentication` object - used as input to `AuthenticationManager` to provide the credentials that user has provided to authenticate, also represents the currently authenticated user, contains principal, credentials, authorities
- `GrantedAuthority` - high level permissions the user is granted, for example roles, `ROLE_ADMIN`, `ROLE_EDITOR`, `ROLE_VIEWER` etc.

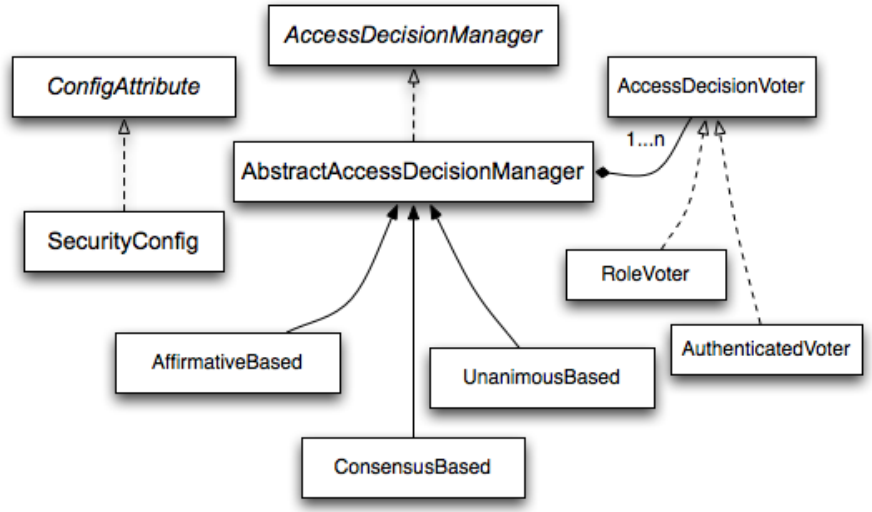
Question 02 - Is security a cross cutting concern? How is it implemented internally?

- `AuthenticationManager` - API that defines how Spring Security's Filters perform authentication, usually implemented by `ProviderManager`
- `ProviderManager` - is an `AuthenticationManager` that delegates to list of `AuthenticationProviders`, if at least `AuthenticationProvider` will successfully authenticate user, user is logged into the system



Question 02 - Is security a cross cutting concern? How is it implemented internally?

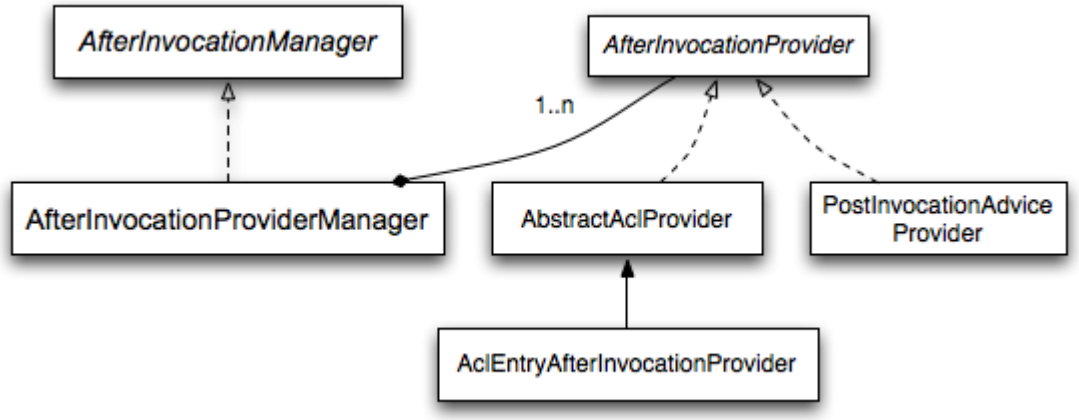
- `AccessDecisionManager` - called by `SecurityInterceptors` before executing method/action, used for authorization to check if user is allowed to perform certain action or access certain resource in the system based on `GrantedAuthority` objects



From Spring Security Documentation

Question 02 - Is security a cross cutting concern? How is it implemented internally?

- `AfterInvocationManager` - called after executing method/action, used for authorization to ensure the principal is permitted to access the domain object instance returned by a service layer bean

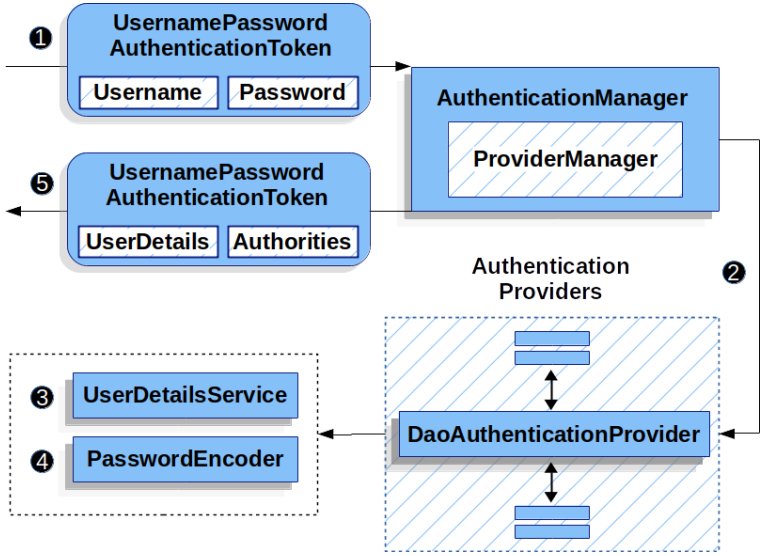


From Spring Security Documentation

Question 02 - Is security a cross cutting concern? How is it implemented internally?

Spring Security is able to access list of users and authorities based on `UserDetailsService` abstraction, following storage types are supported:

- Simple Storage with In Memory Authentication
- Relational Databases with JDBC Authentication
- Custom data stores with UserDetailsService
- LDAP storage with LDAP Authentication



From Spring Security Documentation

Question 02 - Is security a cross cutting concern? How is it implemented internally?

Here is an example scenario that Spring Security can handle with usage of `Authenthiation` and `Authorization` components:

1. `User` tries to access protected resource.
2. Application requires the user to provide `username` and `password` (`form login`). `Username` is identifier, `password` is credential.
3. Credentials are verified by the `AuthenticationManager`, implemented by `ProviderManager`, which delegates to `AuthenthicationProviders`, user is granted access to the application, `SecurityContext` will hold authorization rights for this user.
4. User tries to edit some resource, which is implemented by method on controller level, `SecurityInterceptor` intercepts the request.
5. `SecurityInterceptor` extracts the user authorization data from the `SecurityContext`.
6. `AccessDecisionManager` is invoked to check if user is allowed to perform requested operation.
7. `AccessDecisionManager` delegates call to a list of `AccessDecisionVoters` to check if user is allowed to perform requested operation.
8. Access is `granted` or `denied`.