JWT + Spring Security Interview Handbook

Project-Based Documentation (ADVANN - Spring Boot)

This handbook is designed for interview preparation. It contains a complete end-to-end explanation of JWT Authentication implementation in the ADVANN project, along with commonly asked interview questions and strong answers. The content is written in a way you can directly speak in interviews.

Project	ADVANN (Spring Boot + PostgreSQL + Spring Security)
Topic	JWT Authentication + Authorization
Prepared For	Interview Preparation (Backend / Full Stack)
Includes	Architecture Flow, File Responsibilities, Q&A, Common Mistakes, Advanced Topics

Table of Contents

1. Project Overview (ADVANN)

ADVANN is a Spring Boot based backend project using PostgreSQL. The project includes authentication and authorization using JWT and Spring Security. The main goal is to secure APIs so that only authenticated users can access protected resources such as user profile endpoints.

Main Features Implemented:

- User Registration API
- User Login API
- JWT Token Generation & Validation
- JWT Filter to secure APIs
- Role Support (ADMIN / CUSTOMER)
- Protected endpoint: /api/users/profile

2. Authentication Flow (End-to-End)

Interviewers often ask you to explain the entire authentication flow. Below is the exact flow implemented in ADVANN.

Flow Steps:

- User registers using POST /api/auth/register.
- Password is encrypted using BCryptPasswordEncoder.
- User logs in using POST /api/auth/login.
- AuthenticationManager validates credentials using CustomUserDetailsService.
- If login is successful, JwtService generates an Access Token (JWT).
- Frontend stores the token and sends it in every request header.
- JwtAuthenticationFilter validates token for every request.
- If token is valid, SecurityContextHolder stores Authentication.
- Request reaches controller and returns response.

Example Request Header

```
Authorization: Bearer
```

3. File-Wise Responsibilities (Interview Ready)

File Name	Responsibility (What it does)
application.yml	Stores DB config, JWT secret key, token expiration time.
entity/Role.java	Enum roles like ADMIN, CUSTOMER.
entity/User.java	User table mapping: id, name, email, password, role.
repository/UserRepository.java	DB queries: findByEmail(), existsByEmail().
dto/RegisterRequestDto.java	Register request data structure.
dto/LoginRequestDto.java	Login request structure.
dto/AuthResponseDto.java	Login response: token, email.
payload/ApiResponse.java	Standard API response wrapper.
security/jwt/JwtService.java	Generates token, extracts username, validates token.
security/jwt/CustomUserDetailsService.java	Loads user from DB for Spring Security.
security/jwt/JwtAuthenticationFilter.java	Extracts token from request, validates, sets SecurityContext.
security/SecurityConfig.java	Configures Spring Security, filter chain, stateless policy.
service/AuthService.java	Service interface for register and login.
service/AuthServiceImpl.java	Business logic for register/login, password encryption.
controller/AuthController.java	Exposes /register and /login endpoints.
controller/UserController.java	Protected API endpoint like /api/users/profile.

4. JWT Token Structure

JWT contains 3 parts separated by dots (.).

- **Header:** Contains algorithm info (HS256).
- **Payload:** Contains claims like username/email, issued time, expiry time.
- **Signature:** Created using secret key to ensure token integrity.

Example JWT Format

```
xxxxx.yyyyy.zzzzz
```

5. Core Interview Questions (Most Asked)

Q. Explain your authentication flow end-to-end in ADVANN project.

Answer: In ADVANN, user registers through /api/auth/register. Password is encrypted using BCrypt and stored in PostgreSQL. During login, user hits /api/auth/login and AuthenticationManager validates credentials using CustomUserDetailsService. If authentication succeeds, JwtService generates JWT token with expiry. Frontend stores token and sends it in Authorization header. JwtAuthenticationFilter validates token for each request and sets authentication in SecurityContextHolder. Protected APIs only work if token is valid.

Q. Why JWT is stateless?

Answer: JWT is stateless because server does not store any session. The token itself contains user identity and expiry. Backend only validates token signature and expiry, so no server-side session storage is required.

Q. What is stored inside JWT token?

Answer: JWT contains header, payload and signature. Payload stores claims like username/email and expiry. Signature ensures token cannot be modified.

Q. How do you generate JWT token in your project?

Answer: After successful login, AuthServiceImpl calls JwtService.generateToken(email). JwtService uses secret key from application.yml and adds expiry claim.

Q. How do you validate token?

Answer: JwtAuthenticationFilter extracts token from Authorization header. JwtService validates signature and expiry, extracts username, and then loads user using CustomUserDetailsService.

Q. What happens if token is expired?

Answer: JwtService validation fails. JwtAuthenticationFilter does not authenticate user. Spring Security returns 401 Unauthorized.

Q. What happens if user sends request without token?

Answer: JwtAuthenticationFilter will not set authentication in SecurityContext. Since endpoint is protected, Spring Security blocks it and returns 401.

Q. Where do you store token in frontend?

Answer: Token is stored in localStorage/sessionStorage and is sent with each request in Authorization header. For better security, HttpOnly cookies can be used.

Q. How do you implement logout in JWT?

Answer: Normally logout is client-side by deleting token. But for secure logout we implement token blacklisting in DB/Redis so backend rejects logged-out tokens.

Q. How do you test JWT in Postman?

Answer: Register user, then login to get token. Copy token and call protected API like /api/users/profile with Authorization: Bearer . Without token it fails with 401.

6. Spring Security Interview Questions

Q. Why did you create CustomUserDetailsService?

Answer: Spring Security requires UserDetails object to authenticate. CustomUserDetailsService loads user from DB using email and returns UserDetails to Spring Security.

Q. Why JwtAuthenticationFilter is required?

Answer: JwtAuthenticationFilter checks token in every request before controller execution. It validates token and sets SecurityContext so Spring Security knows user is authenticated.

Q. What is SecurityContextHolder?

Answer: SecurityContextHolder stores the authentication object for the current request. After setting authentication, controllers can access user info using Principal.

Q. Why did you disable CSRF?

Answer: CSRF is mainly for cookie/session authentication. Since JWT is stateless and passed in Authorization header, CSRF is not needed in REST APIs.

Q. Why session policy is STATELESS?

Answer: Because JWT is stateless and we do not want Spring Security to create HTTP sessions.

Q. Why did you add JWT filter before UsernamePasswordAuthenticationFilter?

Answer: JWT filter must validate token before Spring Security processes default authentication. That is why it is placed before UsernamePasswordAuthenticationFilter.

Q. How AuthenticationManager works in your project?

Answer: AuthenticationManager takes UsernamePasswordAuthenticationToken(email,password). It internally uses CustomUserDetailsService and BCryptPasswordEncoder to validate credentials.

Q. How PasswordEncoder works?

Answer: BCryptPasswordEncoder hashes password. During login, Spring compares raw password with stored hash using BCrypt algorithm.

Q. How do you permit register and login endpoints?

Answer: In SecurityConfig I used requestMatchers('/api/auth/**').permitAll() and all other endpoints are authenticated().

7. DTO / API Design Interview Questions

Q. Why did you use DTO instead of directly using Entity in controller?

Answer: DTO prevents exposing internal database entity structure. Entity may contain sensitive fields like password. DTO helps validation and keeps API clean.

Q. What is the use of ApiResponse wrapper?

Answer: ApiResponse provides standard response format: success, message, data. This improves frontend handling and debugging.

Q. Why AuthResponseDto is needed?

Answer: AuthResponseDto sends token and user details like email to frontend after successful login.

Q. Why repository has findByEmail() method?

Answer: Email is unique identifier for login. findByEmail() is used in login and in CustomUserDetailsService to fetch user details.

8. Advanced JWT Questions (Product Company Level)

Q. What is the biggest drawback of JWT?

Answer: JWT cannot be invalidated easily because it is stateless. If token is leaked, attacker can use it until token expires.

Q. How do you solve JWT invalidation problem?

Answer: By implementing refresh tokens and token blacklisting. Blacklisting ensures even a valid token is rejected if it is logged out.

Q. How refresh token works?

Answer: Access token is short-lived, refresh token is long-lived. When access token expires, refresh token is used to generate new access token.

Q. How do you prevent replay attack?

Answer: Use HTTPS, short expiry tokens, refresh token rotation, store tokens securely, and implement token blacklist.

Q. What is difference between authentication and authorization?

Answer: Authentication verifies identity (who you are). Authorization checks permissions (what you can access). JWT handles authentication, roles handle authorization.

9. Role Based Authorization (RBAC)

Q. How did you implement Role Based Access Control in ADVANN?

Answer: User entity contains Role enum (ADMIN, CUSTOMER). Spring Security can restrict endpoints using hasRole('ADMIN') or @PreAuthorize annotations.

Q. Where do you store role?

Answer: Role is stored in PostgreSQL User table and loaded into UserDetails during authentication.

Q. How will you restrict admin-only APIs?

Answer: In SecurityConfig or controller, we can use requestMatchers('/api/admin/**').hasRole('ADMIN') or @PreAuthorize('hasRole("ADMIN")').

10. Logout, Refresh Token, Token Blacklisting

Q. How do you implement logout properly in JWT?

Answer: Frontend removes token. For strong logout, backend maintains a blacklist store (DB/Redis). Token is stored in blacklist during logout and every request checks blacklist before allowing access.

Q. What is Token Blacklisting?

Answer: Token blacklisting means storing invalidated tokens and rejecting them even if they are not expired. It is used to implement secure logout.

Q. How would you implement token blacklist in Spring Boot?

Answer: Create a table like blacklisted_tokens with token, expiry, userId. On logout insert token. In JwtAuthenticationFilter check if token exists in blacklist before authenticating.

Q. Why Redis is better for blacklist?

Answer: Redis is faster and supports automatic expiry. Token can be stored with TTL equal to remaining expiry time.

11. Common Errors & Debugging Questions

Q. Why you may get 403 Forbidden even after sending token?

Answer: Possible reasons: wrong role mapping, missing 'ROLE_' prefix, SecurityConfig restricting endpoint, or SecurityContext not being set correctly.

Q. Why you may get 401 Unauthorized even with token?

Answer: Token expired, secret key mismatch, token corrupted, filter not running, or Authorization header format incorrect.

Q. What if Authorization header is missing 'Bearer '?

Answer: JwtAuthenticationFilter will not extract token and request will be unauthorized.

Q. How do you debug JWT issues quickly?

Answer: Check Postman headers, log token extraction in filter, verify expiry, verify secret key, and verify SecurityConfig filter order.

12. Postman Testing Questions

Q. How do you test register API?

Answer: Call POST /api/auth/register with JSON body containing name, email, password. Verify user saved in DB.

Q. How do you test login API?

Answer: Call POST /api/auth/login with email/password. Verify response contains JWT token.

Q. How do you test protected API?

Answer: Call GET /api/users/profile with header Authorization: Bearer . Without token it returns 401.

13. Microservices & API Gateway JWT Questions

Q. How JWT works in microservices architecture?

Answer: Client authenticates with auth service and gets token. API Gateway validates token before routing requests to microservices. This ensures centralized security.

Q. Should every microservice validate token?

Answer: Best practice is gateway validates token first. But microservices may also validate token for extra security (defense in depth).

Q. How load balancing is done in your system?

Answer: Using Eureka service discovery and Spring Cloud Gateway. Gateway routes requests to multiple instances of same service using service ID.

14. Final 2-Minute Interview Explanation Script

Below is the best final answer you can speak directly in interview when asked: "Explain JWT implementation in your project".

In my ADVANN project, I implemented JWT based authentication using Spring Boot and Spring Security.
I created register and login APIs. During registration, password is encrypted using BCrypt and stored in PostgreSQL.
During login, I use AuthenticationManager which internally uses CustomUserDetailsService and PasswordEncoder to validate credentials.
After successful authentication, JwtService generates a JWT token signed with a secret key and includes an expiry time.
Frontend stores the token and sends it in Authorization header for every request.
I implemented JwtAuthenticationFilter which extracts the token, validates it, and sets authentication inside SecurityContextHolder.
In SecurityConfig, I configured stateless session policy and allowed /api/auth/** endpoints without token while protecting all other endpoints.
This ensures only authenticated users can access protected APIs like /api/users/profile.

End of Handbook