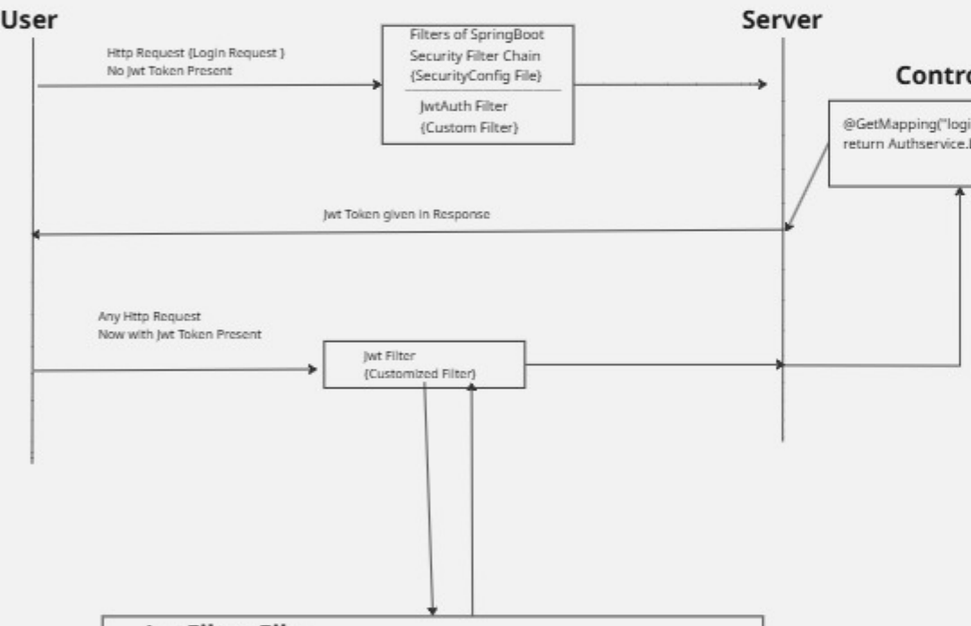
JwtUtil File

📌 **Role:**
- Utility class that handles **JWT (JSON Web Token) operations** like token creation, extracting info, and validation.
- It is a **helper component** used by authentication filters and services.

In simple words:
- **Generate** JWT when user logs in.
- **Extract** data (like username, role) from JWT.
- **Validate** JWT (check signature + expiration + username).
- **Support methods** for expiration date & signing key.

Basic Structure

```
public class JwtUtil {

   private SecretKey getSigningKey();

   public String generateToken(CustomUserDetails user);

   public String getUsernameFromToken(String token);

   public boolean validateToken(String token, UserDetails userDetails);

   public boolean isTokenExpired(String token);

   public Date getExpirationDateFromToken(String token);
}
```



JwtFilter File

📌 **Role:**
- **Role**
- JwtFilter is a **custom filter** that intercepts every HTTP request **before it reaches your controllers**.
- It checks whether the request has a valid **JWT token** and, if yes, sets the authentication inside the SecurityContext.
- Essentially, it replaces the session mechanism with **stateless token-based authentication**.

Basic Code Structure

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain)
        throws ServletException, IOException;
```

AuthService

📌 **Role:**
- Acts as the **business logic layer** for authentication & signup.
- Uses Spring Security's AuthenticationManager, PasswordEncoder, JwtUtil, and JPA repository (JpaRepo).
- Handles **login** and **signup** logic separately.

Basic Structure

```
@Component
public class AuthService {

    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JpaRepo table;
    @Autowired
    private JwtUtil jwtutil;
    @Autowired
    private PasswordEncoder passwordEncoder;

    // 1. Login
    public LoginResponseDto LoginAuth(LoginRequestDto loginRequestDto) {
        // authenticate user
        // generate JWT token if valid
        // return LoginResponseDto(token)
        // throw exception if invalid
    }

    // 2. Signup
    public SignupResponseDto SignupAuth(SignupRequestDto signupRequestDto) {
        // check if username already exists
        // if exists → return "User Already Exists"
        // if not → save new user with encoded password
        // return success response
    }
}
```

Login Structure

LoginRequestDto

📌 **Role:**
- A **DTO (Data Transfer Object)** for carrying login request data (username, password) from client → server.
- Used in the @RequestBody of the login API.

Basic Structure

```
public class LoginRequestDto {
    private String username;
    private String password;

    // getters and setters
}
```

LoginResponseDto

📌 **Role:**
- A **DTO** for sending login response back to client, usually containing the **JWT token** after successful authentication.

Basic Structure

```
public class LoginResponseDto {
    private String token;

    // constructor, getter
}
```

Signup Structure

⚡ Summary:
- SignupRequestDto → input container (client → backend).
- SignupResponseDto → output container (backend → client).

SignupRequestDto

📌 **Role:**
- Acts as a **request DTO** for signup API.
- Holds the data that a client (frontend) sends when creating a new user.

Basic Structure

```
public class SignupRequestDto {
private String username; // new user's name
private String password; // new user's password
private String email;    // new user's email
private String role;     // role of the user (e.g., USER, ADMIN)
}
```

SignupResponseDto

📌 **Role:**
- Acts as a **response DTO** for signup API.
- Used to return feedback after signup (success or failure).

Basic Structure

```
public class SignupResponseDto {
    private String username; // username involved in signup
    private String message;  // success or error message
}
```