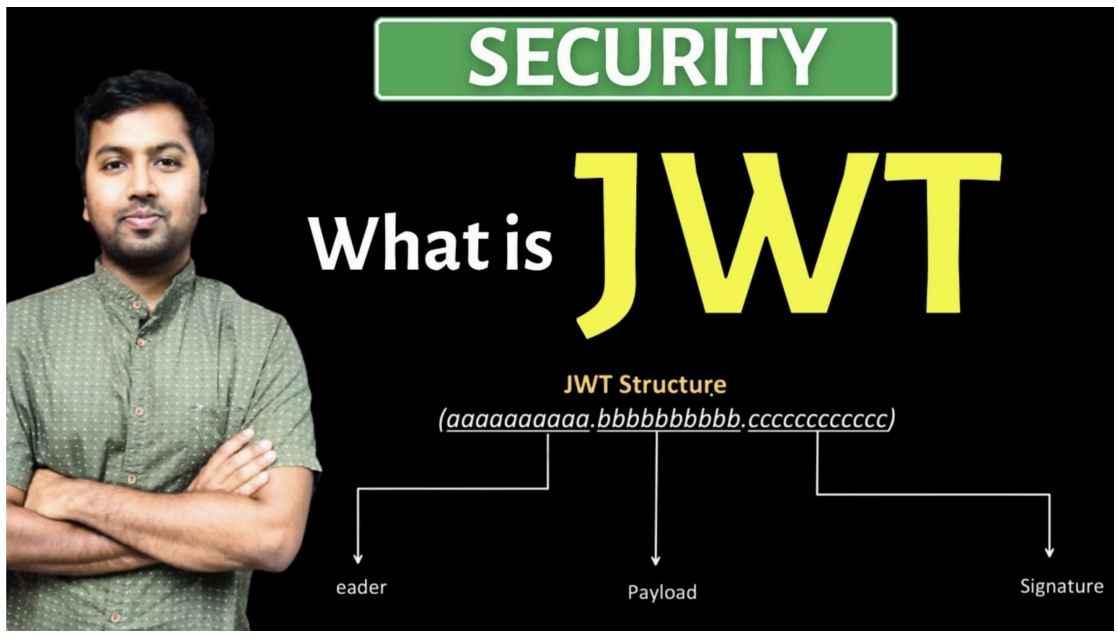
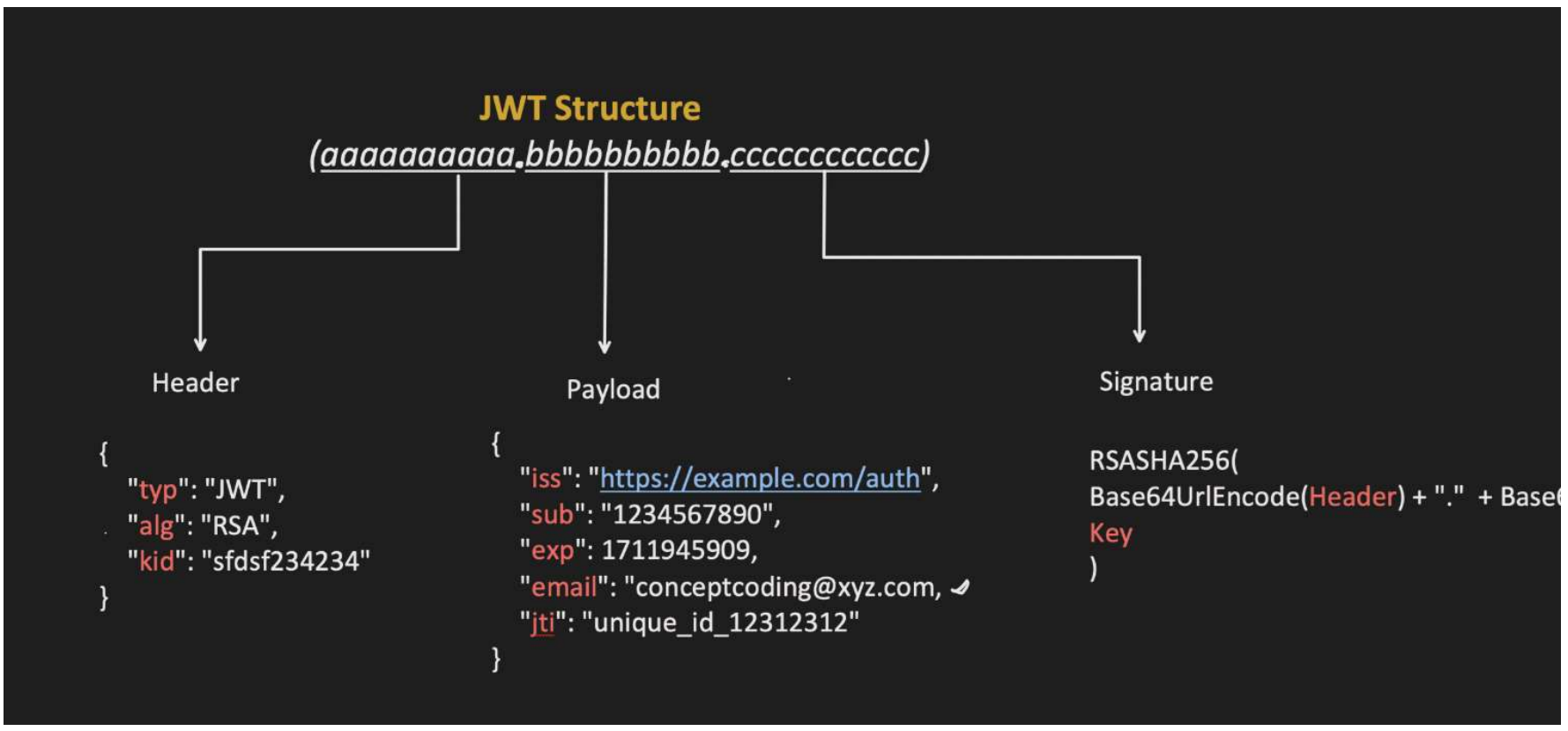
SpringBoot Security - Part5 (JWT Authentication)

Tuesday, 1 April 2025 12:41 PM

JWT(Json Web Token) Authentication

- It's a Stateless Authentication method.
  - Stateless authentication means, server do not maintain the user authentication state (aka
- As mentioned in previous video, JWT has 3 parts
  - HEADER.PAYLOAD.SIGNATURE





- **Header**: Metadata about the token, including the algorithm used (HMAC, RSA etc.)
- **Payload**: Contains **claims** (user details like userId, role, expiry time)
- **Signature**: Ensures token integrity (prevents tampering).
  Any change in payload like role from "user" to "admin" recalculated signature w the original.

Sample token:

eyJhbGciOiJSUzI1NiIsInR5cCI6Ik.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9l.SflKxwRJSMeKKF2C

Steps we will follow:

1. User Creation

"/api/user-register" API

We have to create User (with username, password and Role)

2. Token Generation

"/generate-token" API

In this, User will pass its Username and password.
If details matched, we will generate JWT token and return the same

3. Token Validation

When User try to access any resource, it will pass bacl the
JWT, we will validate the toke
If JWT token is valid, we will Authenticate the user and let the
user access the resources.

JWT Authentication implementation in Spring boot:

1st: User Creation (dynamically)

We have already seen its implementation in


```java
@RestController
@RequestMapping("/api")
public class UserController {

    @Autowired
    UserRegisterEntityService userRegisterEntityService;

    @Autowired
    PasswordEncoder passwordEncoder;

    /*
    using this API to register the user into the system. username, password, role.
     */
    @PostMapping("/user-register")
```

```java
@Service
public class UserRegister

    @Autowired
    private UserRegisterE

    @Override
    public UserDetails lo
        return userAuthEr
    }

    public UserDetails sa
```

```
    public ResponseEntity<String> register(@RequestBody UserRegisterEntity userRegisterDetails) {
        // Hash the password before saving
        userRegisterDetails.setPassword(passwordEncoder.encode(userRegisterDetails.getPassword()));

        userRegisterEntityService.save(userRegisterDetails);

        return ResponseEntity.ok( body: "User registered successfully!");
    }

    @GetMapping("/users")
    public String getUsersDetails(){
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        return "fetched user details successfully";
    }
}
```

```
    }

}
```

```
@Repository
public interface

    Optional<User
}
```

```
@Entity
@Table(name = "user_register")
public class UserRegisterEntity implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    private String role;

    public void setPassword(String password) { this.password = password; }

    public String getRole() { return role; }

    public void setRole(String role) { this.role = role; }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role));
    }

    @Override
    public String getPassword() { return password; }

    @Override
    public String getUsername() { return username; }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passw
        return new BCryptPassword
    }

    @Bean
    public SecurityFilterChain s

        http.authorizeHttpReques
                        .request
                        .anyReque
                .sessionManageme
                        .session
                .csrf(csrf -> cs
        return http.build();
    }
}
```

POST localhost:8080/api/user-register

Params • Authorization Headers (9) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "username" : "sj",
  "password" : "123",
  "role" : "ROLE_USER"
}
```

Body Cookies (1) Headers (10) Test Results

Raw Preview Visualize

```
User registered successfully!
```

Run | Run Selected | Auto complete | Clear

```
SELECT * FROM USER_REGISTER
```

SELECT * FROM USER_REGISTER;

ID	PASSWORD
1	$2a$10$uyRFax7bjDk6OwCrLUb4O.i.I

(1 row, 2 ms)

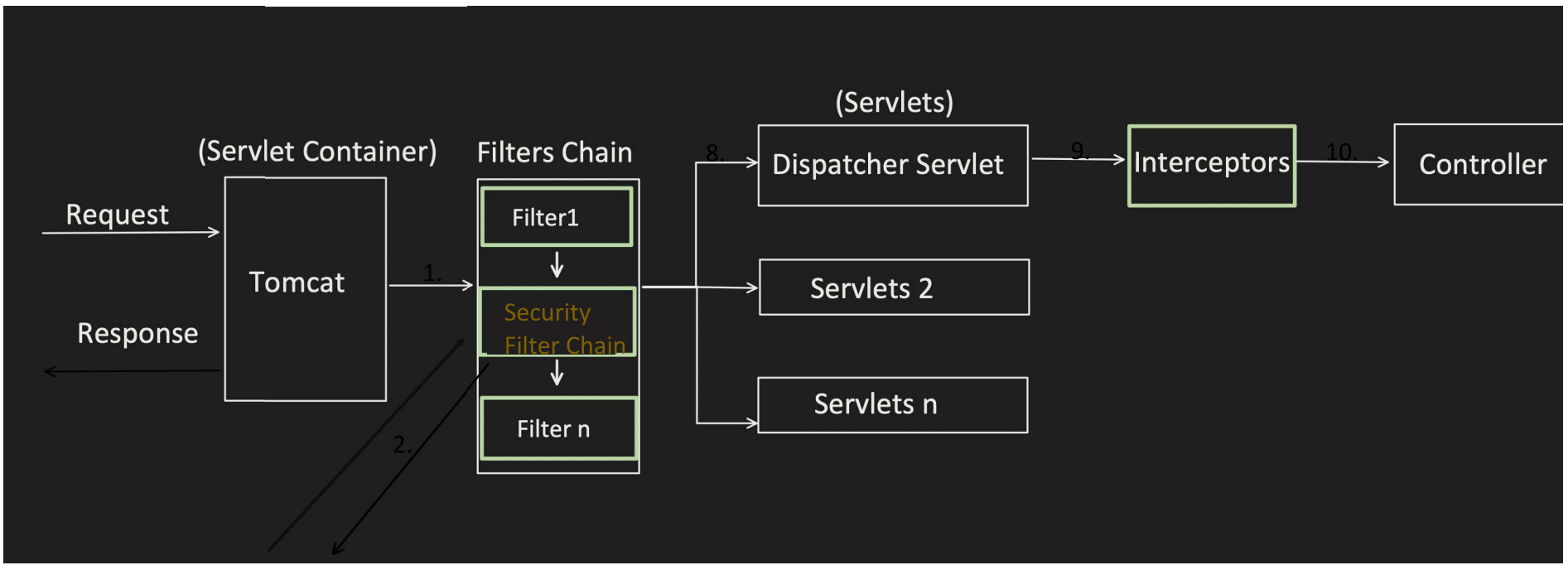
2nd: Token Generation

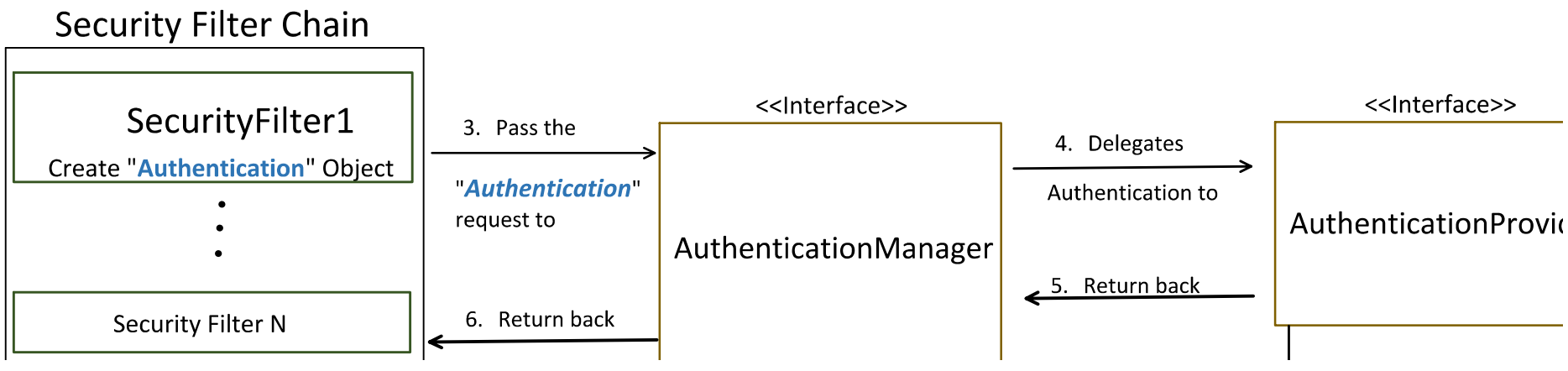
- Spring boot do not provide any default implementation for JWT Authentication.
- Because different application can have different requirement regarding:
  - Payload (some need to put only username, some need to put Id etc.)
  - Signing Algorithm (some want RSA, some want HMAC etc.)
  - Token Refreshing Strategy (some might need it, some don't)

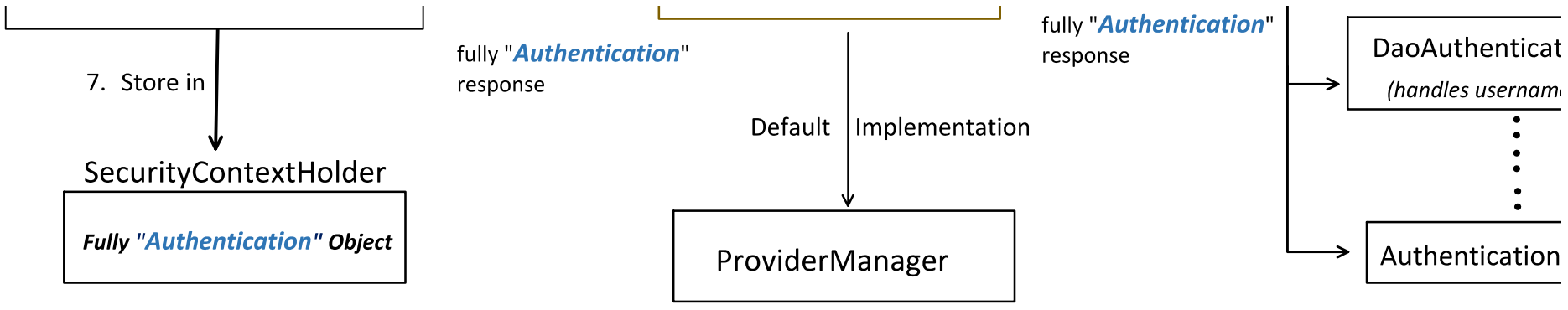
Now, the onus comes to engineer to implement the JWT functionality, that's why there is n
Different engineers, different ways to implement.

But we will try to stick to Security Framework only to implement the JWT functionality.

Quick recap of the Architecture:







Notice one behavior in above flow:

- Security filter creates "*Authentication*" Object, with data coming from request like usern Session-ID or Token etc. But it does not know, which Authentication Provider can handle

- Authentication Manager has a list of Authentication Provider. It calls Support Method of AuthenticationProvider and pass the "*Authentication*" object and checks, if they can han

ProviderManager.java (framework code)

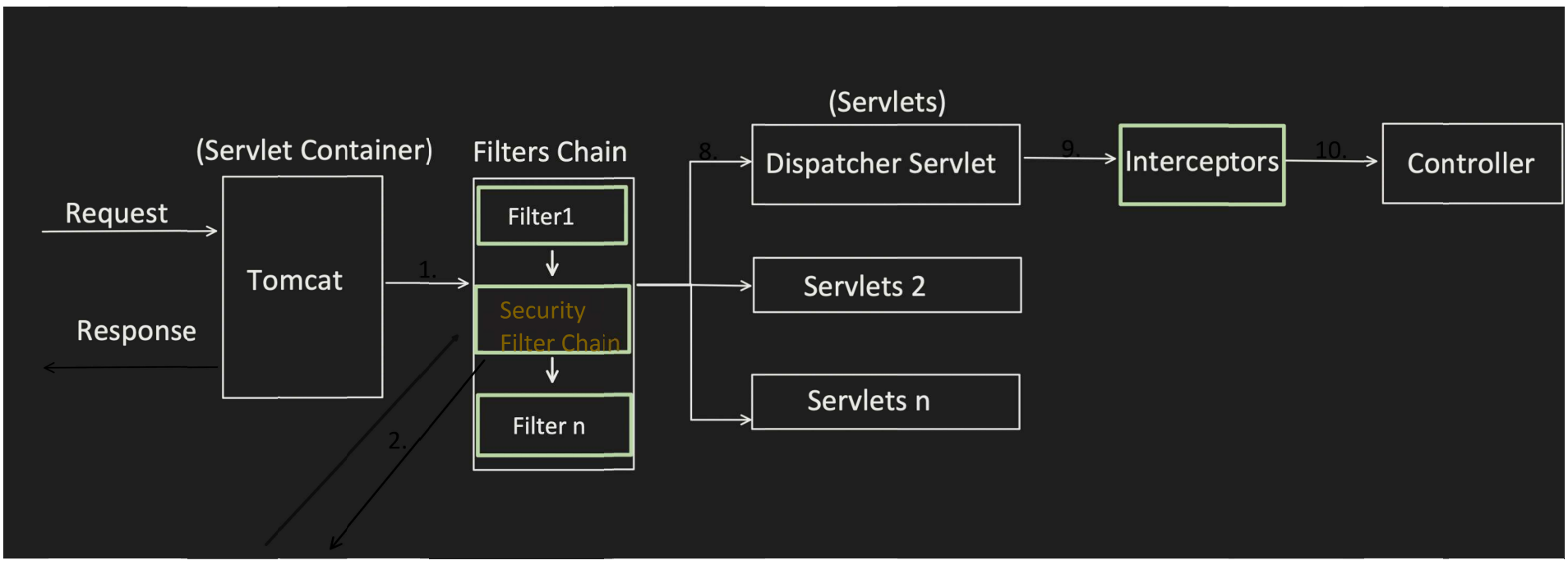
```
@Override
public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    Class<? extends Authentication> toTest = authentication.getClass();
    AuthenticationException lastException = null;
    AuthenticationException parentException = null;
    Authentication result = null;
    Authentication parentResult = null;
    int currentPosition = 0;
    int size = this.providers.size();
    for (AuthenticationProvider provider : getProviders()) {
        if (!provider.supports(toTest)) {
            continue;
        }
        if (logger.isTraceEnabled()) {
            logger.trace(LogMessage.format( format: "Authenticating request with %s (%d/%d)",
                    provider.getClass().getSimpleName(), ++currentPosition, size));
        }
        try {
            result = provider.authenticate(authentication);
            if (result != null) {
                copyDetails(authentication, result);
                break;
            }
        }
```

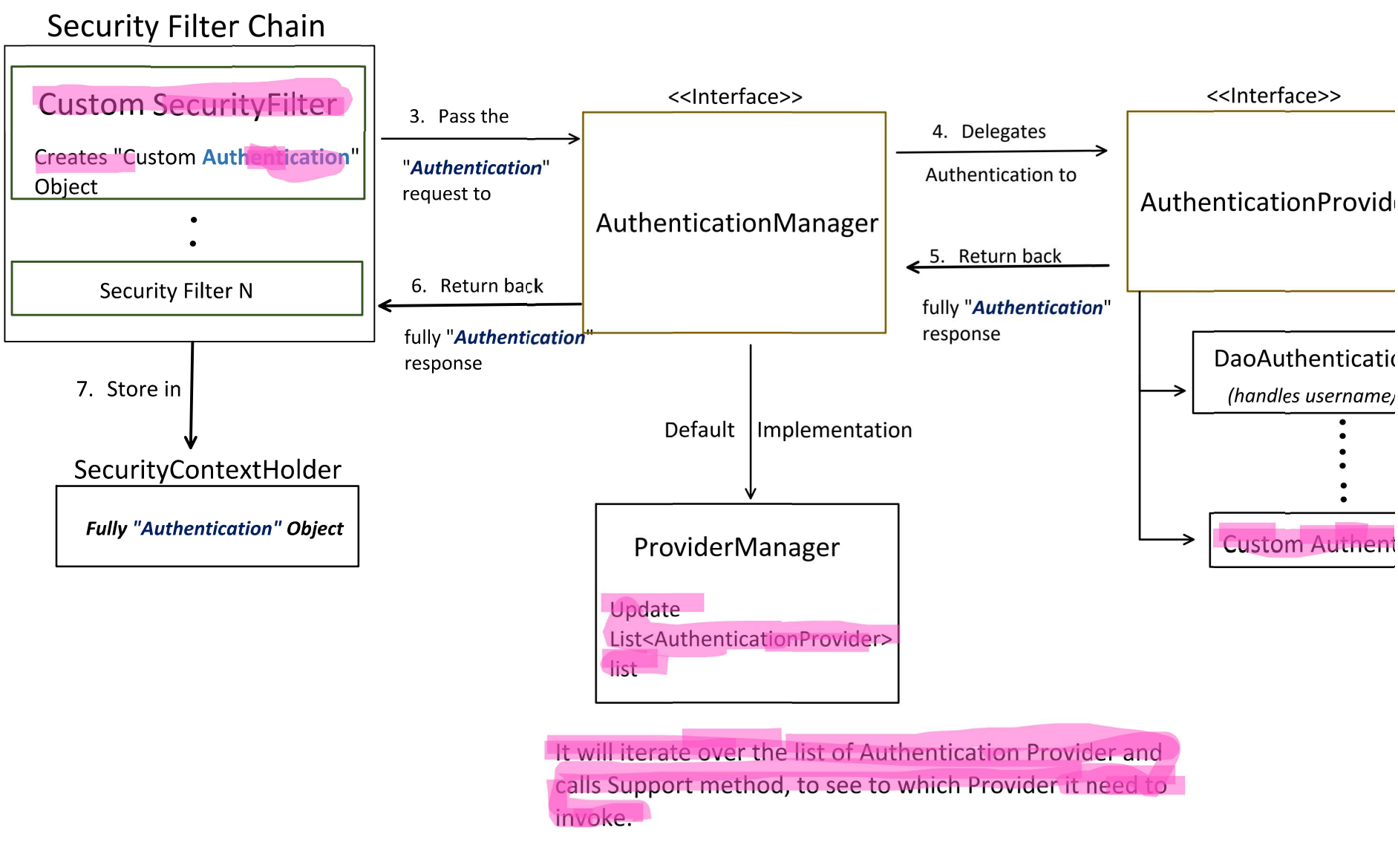
Iterating over the list of Aut

Calls "support" method and checks, if given Aut can handles the incoming Authentication reques

If yes, then only it calls its "au

We just need to enhance this functionality for JWT implementatic





1. Add our Custom Filter.
2. This Custom Filter creates an object of Custom Authentication Object.
3. Create new Custom Authentication Provider and also update Authentication Manager's "Au Custom Authentication Provider in it.

4. Now Authentication Manager get the custom Authentication Object, it will check all the pro
   our CustomAuthencationProvider will return true and thus Authentication Manager will pas

Lets, code for Token generation part.

- First, we need to add JWT dependencies.
- In Token Generation part, user passes username/password in request and we have to matcl
  password.
- This functionality is similar to DaoAuthenticationProvider, so we will try to use that.
- And if username/password is matched, we have to generate the token.

Pom.xml

```xml
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId> <!-- Required for JSON processing
    <version>0.12.6</version>
</dependency>
```

```java
public class JWTAuthenticationFilter extends OncePerRequestFilter {

    private final AuthenticationManager authenticationManager;
    private final JWTUtil jwtUtil;

    public JWTAuthenticationFilter(AuthenticationManager authenticationManager, JWTUtil jwtUtil) {
        this.authenticationManager = authenticationManager;
        this.jwtUtil = jwtUtil;
    }

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

        if (!request.getServletPath().equals("/generate-token")) {
            filterChain.doFilter(request, response);
            return;
        }

        ObjectMapper objectMapper = new ObjectMapper();
        LoginRequest loginRequest = objectMapper.readValue(request.getInputStream(), LoginRequest.class);
```

I am specifically using this Authentication object, so t DaoAuthenticationProvide can handle it.

```
        UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword());

        Authentication authResult = authenticationManager.authenticate(authToken);

        if (authResult.isAuthenticated()) {
            String token = jwtUtil.generateToken(authResult.getName(), expiryMinutes: 15); //15min
            response.setHeader( s: "Authorization", s1: "Bearer " + token);
        }
    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private JWTUtil jwtUtil;
    private UserDetailsService userDetailsService;

    @Autowired
    public SecurityConfig(JWTUtil jwtUtil, UserDetailsService userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public DaoAuthenticationProvider daoAuthenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

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

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http, AuthenticationManager aut
                                                   JWTUtil jwtUtil) throws Exception {

        // Authentication filter responsible for login
        JWTAuthenticationFilter jwtAuthFilter = new JWTAuthenticationFilter(authenticationManag

        http.authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/api/user-register").permitAll()
                        .anyRequest().authenticated())
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class) ; /
        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager() {
        return new ProviderManager(Arrays.asList(
                daoAuthenticationProvider()));
    }
}
```

```
}
```

POST localhost:8080/api/user-register

Params • Authorization Headers (9) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary

```
{
    "username" : "sj",
    "password" : "123",
    "role" : "ROLE_USER"
}
```

Body Cookies (1) Headers (10) Test Results

Raw Preview Visualize

```
User registered successfully!
```

POST http://localhost:8080/generate-token

Params Authorization Headers (9) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

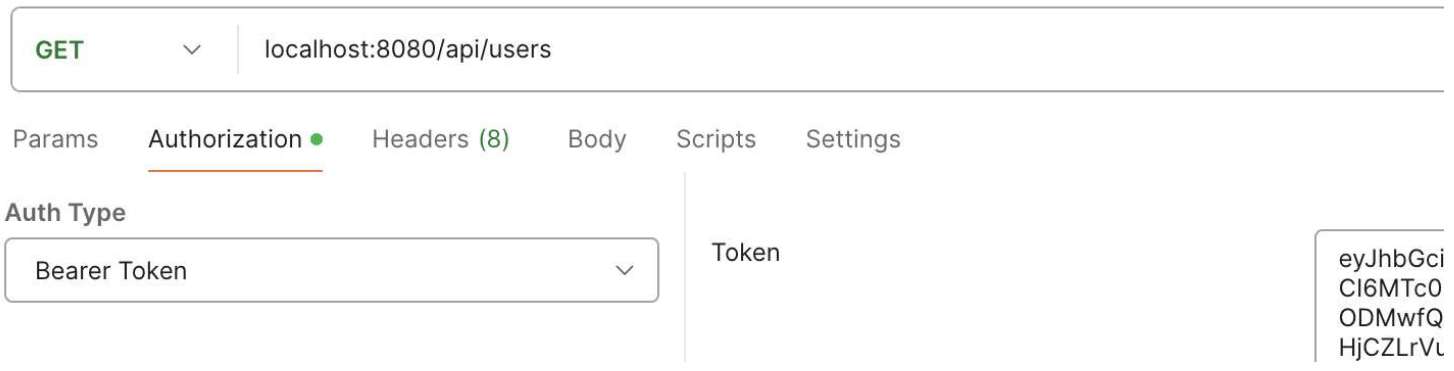
```
{
    "username" : "sj",
    "password" : "123"
}
```

Body Cookies (1) Headers (10) Test Results

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdV AYgpAW6UNTL4nJ6GhFrWPwI1cjK3
X-Content-Type-Options	nosniff

3rd: Token Validation

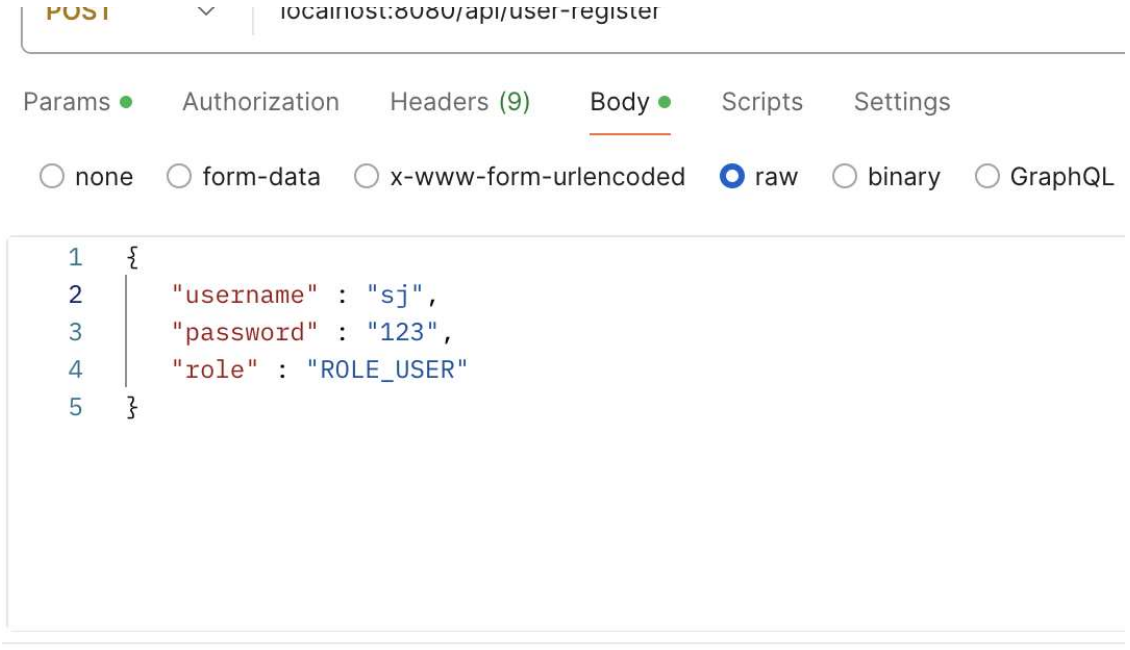
- Now, if user try to access any restricted resource. They need to pass the Token in the A

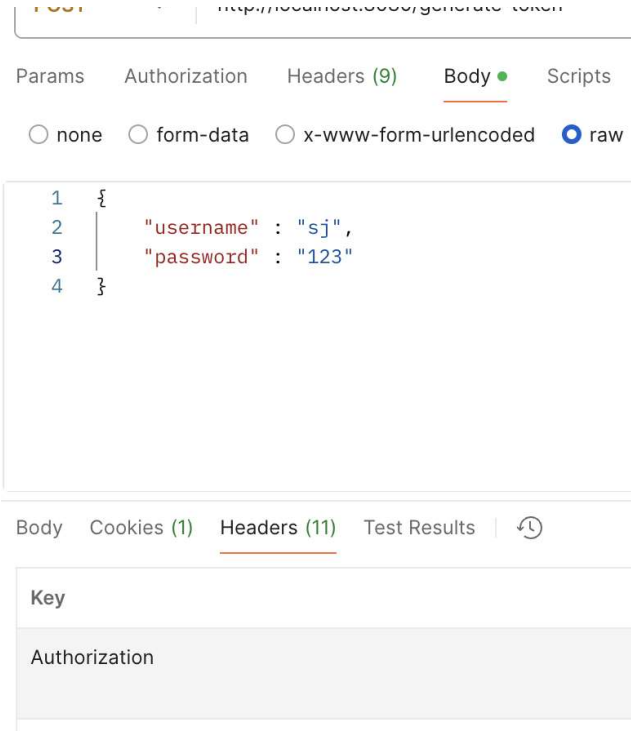


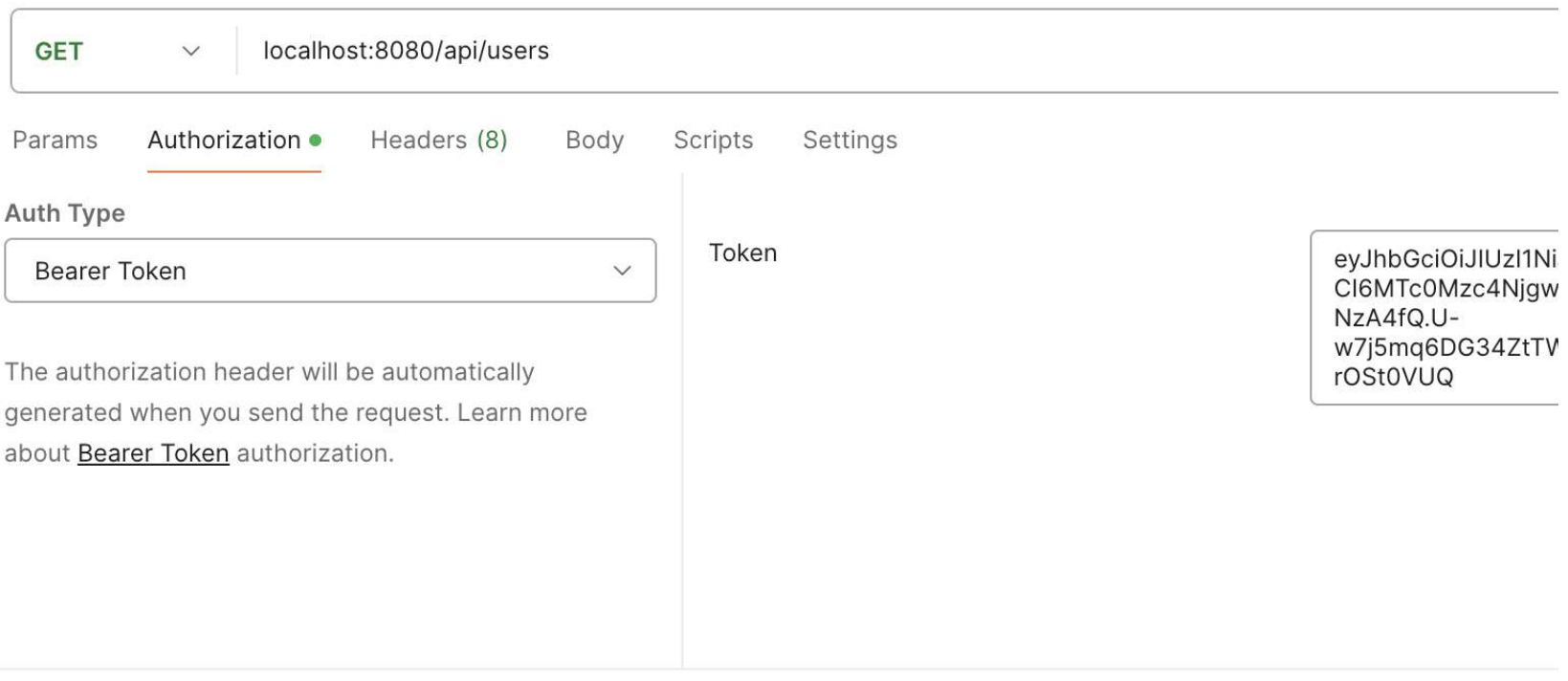

```java
public class JwtValidationFilter extends OncePerRequestFilter {

    private final AuthenticationManager authenticationManager;

    public JwtValidationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        String token = extractJwtFromRequest(request);
        if (token != null) {

            JwtAuthenticationToken authenticationToken = new JwtAuthenticationToken(token);
            Authentication authResult = authenticationManager.authenticate(authenticationToken);
            if(authResult.isAuthenticated()) {
                SecurityContextHolder.getContext().setAuthentication(authResult);
            }
        }

        filterChain.doFilter(request, response);
    }

    private String extractJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader( s: "Authorization");
        if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring( beginIndex: 7);
        }
        return null;
    }
}
```

```java
@Component
public class JWTUtil {

    private static final String SECRET_KEY = "your-secure-secret-key-min-32bytes";
    private static final Key key = Keys.hmacShaKeyFor(SECRET_KEY.getBytes(StandardCharsets.UTF_8));


    // Generate JWT Token
    public String generateToken(String username, long expiryMinutes) {
```

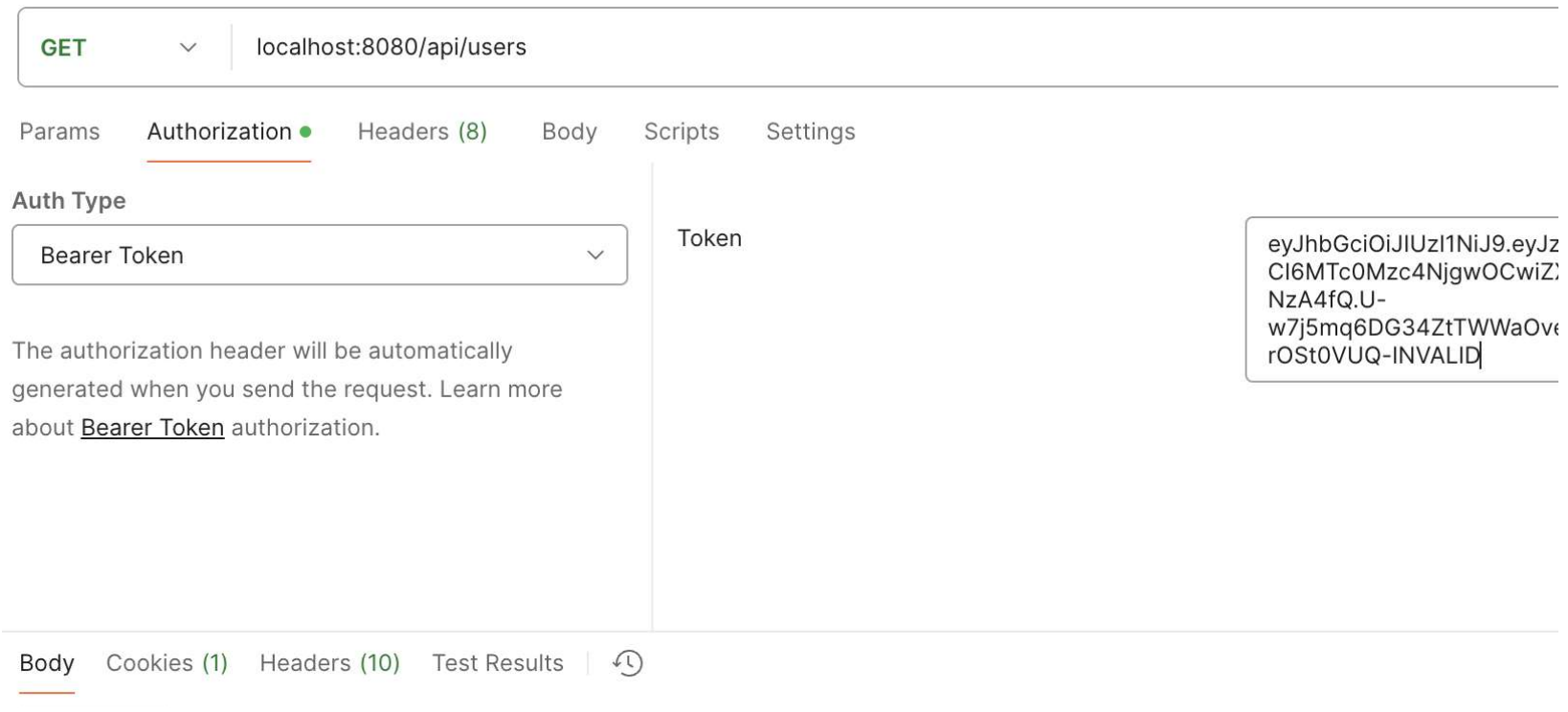
```
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + expiryMinutes * 60 * 1000)) //in milliseconds
                .signWith(key, SignatureAlgorithm.HS256)
                .compact();
    }

    public String validateAndExtractUsername(String token) {
        try {
            return Jwts.parser() JwtParserBuilder
                    .setSigningKey(key)
                    .build() JwtParser
                    .parseClaimsJws(token) Jws<Claims>
                    .getBody() Claims
                    .getSubject();
        } catch (JwtException e) {
            return null; // Invalid or expired JWT
        }
    }
}
```







When tried to add Invalid Token:



Raw Preview Visualize

1

4th: Refresh Token

- Generally access tokens are short lived.
- Once access token get expired, Refresh token (generally long lived) is used to obtain token, without requiring user to log in again.

```
public class JWTAuthenticationFilter extends OncePerRequestFilter {

    private final AuthenticationManager authenticationManager;
    private final JWTUtil jwtUtil;

    public JWTAuthenticationFilter(AuthenticationManager authenticationManager, JWTUtil jwtUtil) {
        this.authenticationManager = authenticationManager;
        this.jwtUtil = jwtUtil;
    }

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

        if (!request.getServletPath().equals("/generate-token")) {
            filterChain.doFilter(request, response);
            return;
        }

        ObjectMapper objectMapper = new ObjectMapper();
        LoginRequest loginRequest = objectMapper.readValue(request.getInputStream(), LoginRequest.class);

        UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword());
        Authentication authResult = authenticationManager.authenticate(authToken);

        if (authResult.isAuthenticated()) {
            String token = jwtUtil.generateToken(authResult.getName(), expiryMinutes: 15); //15min
            response.setHeader( s: "Authorization", s1: "Bearer " + token);

            String refreshToken = jwtUtil.generateToken(authResult.getName(), expiryMinutes: 7 * 24 * 60); //7day
            // Set Refresh Token in HttpOnly Cookie
            //we can also send it in response body but then client has to store it in local storage or in-memory
            Cookie refreshCookie = new Cookie( name: "refreshToken", refreshToken);
            refreshCookie.setHttpOnly(true); //prevent javascript from accessing it
            refreshCookie.setSecure(true); // sent only over HTTPS
            refreshCookie.setPath("/refresh-token"); // Cookie available only for refresh endpoint
            refreshCookie.setMaxAge(7 * 24 * 60 * 60); // 7 days expiry
            response.addCookie(refreshCookie);
```

```
        }
    }
}
```

```
public class JwtAuthenticationToken extends AbstractAuthenticationToken {

    private final String token;

    public JwtAuthenticationToken(String token) {
        super( authorities: null);
        this.token = token;
        setAuthenticated(false);
    }

    public String getToken() { return token; }

    @Override
    public Object getCredentials() { return token; }

    @Override
    public Object getPrincipal() { return null; }
}
```

```
@Component
public class JWTUtil {

    private static final String SECRET_KEY = "your-secure-secret-key-min-32bytes";
    private static final Key key = Keys.hmacShaKeyFor(SECRET_KEY.getBytes(StandardCharsets.UTF_8));


    // Generate JWT Token
    public String generateToken(String username, long expiryMinutes) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + expiryMinutes * 60 * 1000)) //in milliseconds
                .signWith(key, SignatureAlgorithm.HS256)
                .compact();
    }

    public String validateAndExtractUsername(String token) {
        try {
            return Jwts.parser() JwtParserBuilder
                    .setSigningKey(key)
                    .build() JwtParser
                    .parseClaimsJws(token) Jws<Claims>
                    .getBody() Claims
                    .getSubject();
        } catch (JwtException e) {
            return null; // Invalid or expired JWT
        }
    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private JWTUtil jwtUtil;
    private UserDetailsService userDetailsService;

    @Autowired
    public SecurityConfig(JWTUtil jwtUtil, UserDetailsService use
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }
```

```java
@Bean
public JWTAuthenticationProvider jwtAuthenticationProvider() {
    return new JWTAuthenticationProvider(jwtUtil, userDetails
}

@Bean
public DaoAuthenticationProvider daoAuthenticationProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticatio
    provider.setUserDetailsService(userDetailsService);
    provider.setPasswordEncoder(passwordEncoder());
    return provider;
}

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

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity
                                               JWTUtil jwtU

    // Authentication filter responsible for login
    JWTAuthenticationFilter jwtAuthFilter = new JWTAuthentic

    // Validation filter for checking JWT in every request
    JwtValidationFilter jwtValidationFilter = new JwtValidat

    // refresh filter for checking JWT in every request
    JWTRefreshFilter jwtRefreshFilter = new JWTRefreshFilter

    http.authorizeHttpRequests(auth -> auth
                    .requestMatchers( ...patterns: "/api/user-re
                    .anyRequest().authenticated())
            .sessionManagement(session -> session
                    .sessionCreationPolicy(SessionCreationPo
            .csrf(csrf -> csrf.disable())
            .addFilterBefore(jwtAuthFilter, UsernamePasswordA
            .addFilterAfter(jwtValidationFilter, JWTAuthenti
            .addFilterAfter(jwtRefreshFilter, JwtValidationF
    return http.build();
}

@Bean
public AuthenticationManager authenticationManager() {
    return new ProviderManager(Arrays.asList(
            daoAuthenticationProvider(),
            jwtAuthenticationProvider()
    ));
}
}
```

POST localhost:8080/api/user-register

POST http://localhost:8080/generate-token

Params Authorization Headers (9) Body Scripts Settings

Params ● Authorization Headers (9) Body ● Scripts Settings

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL

```
1  {
2      "username" : "sj",
3      "password" : "123",
4      "role" : "ROLE_USER"
5  }
```

Body Cookies (1) Headers (11) Test Results

Raw ▷ Preview Visualize

```
1  User registered successfully!
```

Params Authorization Headers (9) Body ● Scripts Settings

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary

```
1  {
2      "username" : "sj",
3      "password" : "123"
4  }
```

Body Cookies (1) Headers (12) Test Results

Key
Authorization
Set-Cookie
X-Content-Type-Options

GET localhost:8080/refresh-token Send

Params Authorization Headers (7) Body Scripts Settings Coo

☑	Postman-Token	<calculated when request is sent>	
☑	Host	<calculated when request is sent>	
☑	User-Agent	PostmanRuntime/7.43.2	
☑	Accept	*/*	
☑	Accept-Encoding	gzip, deflate, br	
☑	Connection	keep-alive	
	Key	Value	Description

Body Cookies (2) Headers (11) Test Results 200 OK • 9 ms • 444 B

Name	Value	Domain	Path	Expires	HttpOnly	Secure
refreshToken	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzaiIsImlhdCI6MTc0Mzc5MDM2NSwiZXhwIjoxNzQ0Mzk1MTY1fQ.jmBkb-21PK8_lkoolY3AD7bz2bS9llv2w_IREYhLJQ	localhost	/refresh-token	Fri, 11 Apr 2025 1...	true	true

Authorization:

- Works exactly the same as form and basic Authentication.
- AuthorizationFilter gets invokes, which matches the ROLE required for the Al

```java
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http, AuthenticationManager auth
                                               JWTUtil jwtUtil) throws Exception {

    // Authentication filter responsible for login
    JWTAuthenticationFilter jwtAuthFilter = new JWTAuthenticationFilter(authenticationManage

    // Validation filter for checking JWT in every request
    JwtValidationFilter jwtValidationFilter = new JwtValidationFilter(authenticationManager)

    // refresh filter for checking JWT in every request
    JWTRefreshFilter jwtRefreshFilter = new JWTRefreshFilter(jwtUtil, authenticationManager)

    http.authorizeHttpRequests(auth -> auth
                    .requestMatchers( ...patterns: "/api/user-register").permitAll()
                    .requestMatchers( ...patterns: "/api/users").hasRole("USER")
                    .anyRequest().authenticated())
            .sessionManagement(session -> session
                    .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf(csrf -> csrf.disable())
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class) // g
            .addFilterAfter(jwtValidationFilter, JWTAuthenticationFilter.class) // validate
            .addFilterAfter(jwtRefreshFilter, JwtValidationFilter.class); // refresh token f
    return http.build();
}
```

POST localhost:8080/api/user-register

Params • Authorization Headers (9) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2      "username" : "sj",
3      "password" : "123",
4      "role" : "ROLE_ADMIN"
5  }
```

Body Cookies (1) Headers (11) Test Results

Raw Preview Visualize

```
1  User registered successfully!
```

POST http://localhost:8080/generate-token

Params Authorization Headers (9) Body • Script

none form-data x-www-form-urlencoded ra

```
1  {
2      "username" : "sj",
3      "password" : "123"
4  }
```

Body Cookies (1) Headers (12) Test Results

Key
Authorization
Set-Cookie

As "/api/users" API needs role *ROLE_USER* but user has *ROLE_ADMIN*.

GET localhost:8080/api/users

Params Authorization • Headers (8) Body Scripts Settings

Auth Type

Bearer Token

Token

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzail
CI6MTc0Mzc5MjQwNiwiZXhwIjoxNzQ

M2A2TQ.mk2BnseVGmpiD_2u16AXPtM
Bxx-Qfaudqfs6hll

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Body Cookies (1) Headers (10) Test Results

Raw Preview Visualize

```
1
```