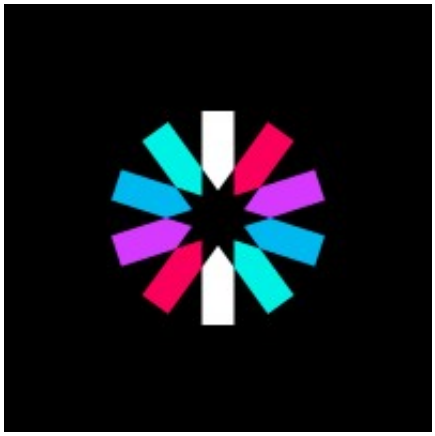


JWT

(JSON Web Token)

Daftar Isi

1. Apa itu JWT

JWT merupakan standar method (berbasis JSON) yang digunakan untuk mengamankan data (claims) yang dikirimkan atau diterima antara 2 aplikasi. Ada 3 bagian penting dari JWT yaitu header, claims, dan signature.

- Header : Berisi Algoritma dan Tipe Token

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

  Ada beberapa algoritma yang bisa digunakan yaitu :

  - HS256
  - HS384
  - HS512
  - RS256
  - RS384
  - RS512
  - ES256
  - ES384
  - ES512

- Claims : Berisi payload atau data yang dikirim

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true,
    “iss”: “John”
}
```

Ada beberapa field Claims (Payload) yang sudah disediakan oleh JWT yaitu :

- iss : Sistem atau aplikasi yg menerbitkan token Token
- sub : Subjek Token
- aud : Subscriber atau yang menggunakan token
- exp : Batas waktu expired dari token
- nbf : Digunakan untuk memvalidasi agar token tidak diproses sebelum waktu yg didefinisikan
- iat : Waktu token digenerate
- jti : Unique ID dari JWT

- Signature : Untuk memverifikasi apakah data header dan claims yang dikirim itu valid atau tidak

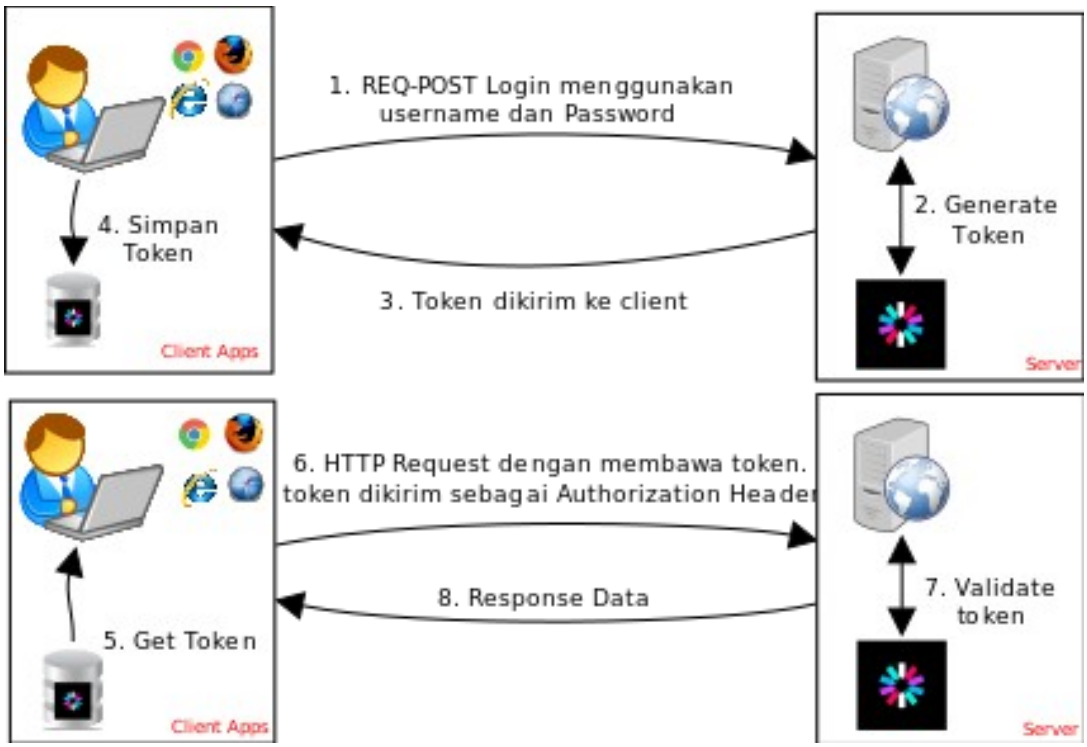
```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
     “secretkey”
)
```

Dari ketiga bagian diatas, token yang dihasilkan adalah

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJqb2huIiwic3ViIjoiMTIzNDU2Nzg5MCIs
Im5hbWUiOiJKb2huIERvZSIsImFkbWluIjoidHJ1ZSJ9.Fn8qqM2IOK0juXnDDw2hdLBGJMle9QGmMjTB
VP8DU-o
```

2. Konsep JWT

A. JWT untuk Authentication



Penjelasan dari gambar diatas yaitu mengenai penggunaan JWT untuk proses Otentikasi.

- Pertama client app mengirimkan username dan password ke server pada saat login menggunakan HTTP POST

- Server melakukan validasi username dan password, jika valid maka server akan menggenerate token, dan token tersebut akan dikirimkan ke client melalui cookies atau response body. Jika username dan password tidak valid maka server mengirimkan pesan login error

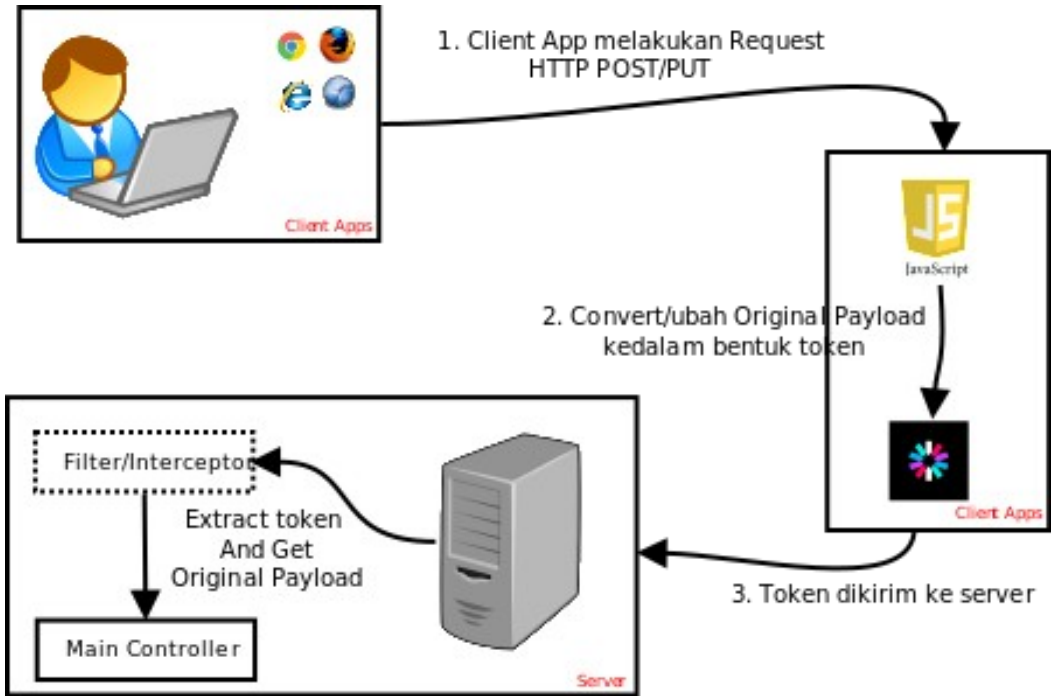
- Client App menerima token dari server, dan disimpan didalam cookies atau local storage browser

- Ketika Client App melakukan Request Data ke server, Client app diharuskan mengirim token yang diterima sebelumnya ke dalam header (misal: `Authorization: Bearer <token>`) sebagai pengganti username/password atau security session

- Server akan melakukan validasi terhadap token yang dikirim, sebelum mengirimkan response data ke client. Item-item yang divalidasi oleh server antara lain adalah, username, expiry token, granted authority, dll.

- Jika semua data dalam token itu valid, maka server akan mengirimkan response data ke client.

B. JWT untuk Encode Request Payload



- Pada saat client apps mengirim Payload ketika melakukan HTTP Request POST/PUT , Javascript melakukan Intercept atau perubahan original payload ke dalam bentuk token.

- Kemudian data yang sudah berbentuk token dikirim ke server
- Ketika server menerima request, HttpFilter atau Interceptor akan menjalankan method untuk meng-extract token untuk mendapatkan original payload.
- Setelah original payload didapat maka request tersebut diteruskan ke servlet controller

3. Sample Code

A. Authentication

1. Server Side (Java)

Dalam tulisan ini untuk membuat server REST dengan java, build tools yang digunakan adalah maven dengan membawa framework **spring-boot** dan dengan tambahan dependency sebagai berikut:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.6.0</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.5</version>
</dependency>
```

Membuat Main Class

Karena menggunakan framework **spring-boot** maka configurasi Main.class seperti dibawah ini. Dan juga ada tambahan anotasi @**EnableMongoRepositories** karena akan menggunakan **mongodb** sebagai ***Data Storage.*** Buat file dengan nama **MainApplication.java** di dalam package **belajar.app**

```
@SpringBootApplication
@EnableMongoRepositories
public class BelajarApplication {
    public static void main(String[] args) {
        SpringApplication.run(BelajarApplication.class, args);
    }
}
```

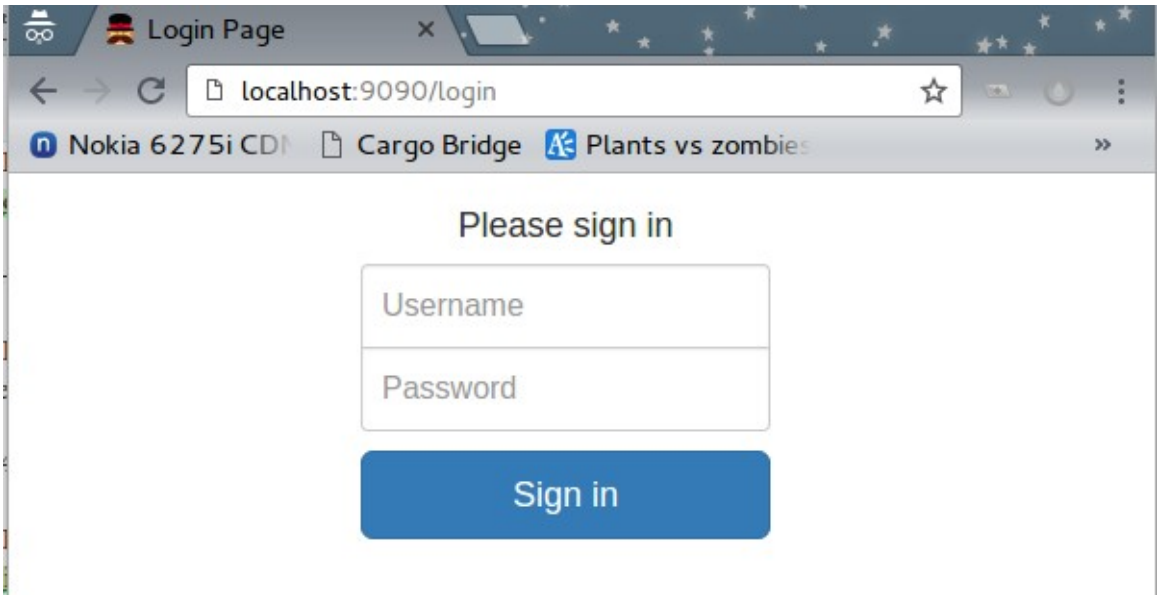
Property Configuration

Karena menggunakan framework **spring-boot** maka perlu membuat file **application.properties** sebagai konfigurasi aplikasi.

```
debug=true
server.port=9090
server.tomcat.basedir=/tmp/belajar-jwt

spring.data.mongodb.uri=mongodb://localhost/belajar_mongodb

spring.jackson.serialization.INDENT_OUTPUT= true

jwt.header= Authorization
jwt.secret= mySecret
jwt.expiration= 604800

logging.level.org.springframework.security= DEBUG
```

Login Page

Karena menggunakan library thymeleaf maka halaman login dengan nama **login.html** dan disimpan di **src/main/resources/templates** dengan tampilan sebagai berikut :



Source code dari tampilan diatas adalah :

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Login Page</title>
        <meta charset="utf-8"/>
        <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
        <meta name="viewport" content="width=device-width, initial-scale=1"/>

        <link rel="stylesheet" href="styles/vendor.2ac5f564.css"/>
        <link rel="stylesheet" href="styles/main.1f9c5951.css"/>
    </head>
    <body>
        <div class="container">
            <div th:if="${param.error} and $
{session.SPRING_SECURITY_LAST_EXCEPTION} != null" class="alert alert-danger">
                <span th:text="${session.SPRING_SECURITY_LAST_EXCEPTION}"></span>
            </div>
            <div th:if="${param.logout}" class="alert alert-success">
                You have been logged out.
            </div>

            <form class="form-signin">
                <!--<input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}" />-->
                <h4 class="form-signin-heading" style="text-align:center;">Please
sign in</h4>
                <label for="inputUsername" class="sr-only">Username</label>
                <input type="text" name="username" id="inputUsername"
class="form-control" placeholder="Username"/>
                <label for="inputPassword" class="sr-only">Password</label>
                <input type="password" name="password" id="inputPassword"
class="form-control" placeholder="Password"/>
                <button class="btn btn-lg btn-primary btn-block" type="button"
```

```
id="btn-login">Sign in</button>
            </form>
        </div>

        <script src="scripts/vendor.a0c1af36.js"></script>
        <script type="text/javascript">
            $('.form-signin').on('keyup', function (e) {
                e.preventDefault();
                if (e.which == 13 || e.keyCode == 13) {
                    processLogin();
                }
            });

            $('#btn-login').click(function () {
                processLogin();
            });

            function processLogin() {
                $.ajax({
                    url: window.location.href,
                    method: 'POST',
                    type: 'text/html',
                    contentType: 'application/x-www-form-urlencoded',
                    data: $('.form-signin').serialize(),
                    success: function (data, textStatus, request) {
                        localStorage.setItem('token',
request.getResponseHeader('Authorization'));

window.location.replace(request.getResponseHeader('Location'));
                    },
                    error: function (xhr, ajaxOptions, thrownError) {
                        window.location.href = 'login?error';
                    }
                });
            }
        </script>
    </body>
</html>
```

Kemudian buat file **WebConfig.java** didalam package **belajar.app.config**. File WebConfig ini digunakan untuk konfigurasi interceptor, cors handler, dll. Dan juga bisa digunakan untuk mendaftarkan URL, misal halaman login yang sudah dibuat sebelumnya juga harus didaftarkan didalam view controller. Karena jika tidak nanti halaman login tidak akan dapat diakses (Muncul error 404). berikut isi konfigurasi nya

```
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
    }
```

```
}
```

Domain Class User, Role dan Permission

Domain Class digunakan untuk menampung objek User dan Permission.

Domain class ini disimpan didalam package **belajar.app.domain.** Berikut ini adalah class **User, Role** dan **Permission**

```
@Document(collection = "users")
public class User implements Serializable {
    @Id
    private String id;

    private String username;

    private String password;

    @DBRef
    private Role role;

    //Getter Setter
}
```

```
@Document(collection = "roles")
public class Role implements Serializable {
    @Id
    private String id;

    private String name;

    @DBRef
    private Set<Permission> permissions = new HashSet<>();

    //Getter Setter
}
```

```
@Document(collection = "permissions")
public class Permission implements Serializable {
    @Id
    private String id;

    private String label;
    private String value;

    //Getter Setter
}
```

UserDao

UserDao ini digunakan sebagai query data ke data storage. File **UserDao** disimpan didalam package **belajar.app.dao**

```
public interface UserDao extends MongoRepository<User, String>{
    public User findByUsername(String username);
}
```

Spring Security Config

Spring security config merupakan class file yang digunakan untuk menyimpan konfigurasi proses login, mulai dari UserDetailsService, PasswordEncoder, Intercept URL, sampai dengan LoginHandler dan LogoutHandler. Untuk membuat konfigurasi ini harus membuat class dengan nama **SecurityConfig** dan disimpan di package **belajar.app.config.** Berikut isi dari class SecurityConfig

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired private SecUserDetailService userDetailService;

    //Authentication Builder check username and password
    //implement interface UserDetailService
    @Autowired
    public void configureGlobal(
            AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailService);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/login").permitAll()
                .antMatchers(HttpMethod.GET, "/api/user/**")
                        .hasRole("USER_VIEW")
                .antMatchers("/api/user/**").hasRole("USER_EDIT")
            .and()
                .formLogin().loginPage("/login").permitAll()
            .and()
                .logout()
                .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                .invalidateHttpSession(true);
    }
}
```

Kemudian agar Spring Security bisa mengetahui query atau method mana yang digunakan untuk proses login (mengecek available user/password) maka perlu implementasi interface **UserDetailsService.** Buat class dengan nama **SecUserDetailsService** didalam package **belajar.app.security.** Berikut ini adalah implementasi **UserDetailsService**

```
@Component
public class SecUserDetailService implements UserDetailsService {

    private final Logger LOGGER =
LoggerFactory.getLogger(SecUserDetailService.class);

    @Autowired
    private UserDao userDao;

    @Override
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        LOGGER.debug("Login with username [{}]", username);
        User user = userDao.findByUsername(username);
        if(user == null){
            throw new UsernameNotFoundException(username);
        } else{
            SecManUserDetails details = new SecManUserDetails(
                    user.getId(), user.getUsername(),
                    user.getPassword(),
                mapToGrantedAuthorities(new ArrayList<>(
                    user.getRole().getPermissions())), true);
            return details;
        }
    }

    private List<GrantedAuthority> mapToGrantedAuthorities(
            List<Permission> permissions) {
        List<GrantedAuthority> authorities = new ArrayList<>();
        for(Permission p : permissions){
            authorities.add(new SimpleGrantedAuthority(
                    p.getValue()));
        }
        return authorities;
    }
}
```

Session Management

Karena model authentication menggunakan token ini setiap request http harus melakukan pengecekan token. Maka tidak perlu adanya session (Stateless session). Berikut ini adalah konfigurasi yang harus ditambahkan di **HttpSecurity**

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()
            .sessionCreationPolicy(
                  SessionCreationPolicy.STATELESS);
}
```

Login Handler

Dengan Spring Security kita bisa membuat custom handler untuk proses login, karena mungkin login handler yang disediakan oleh spring security belum memenuhi kebutuhan aplikasi kita. Dalam studi kasus kali ini, Handler yang perlu dibuat adalah **LoginFailureHandler** untuk merubah behavior ketika login gagal. Dan juga **LoginSuccessHandler** digunakan untuk mengirim token ketika login sukses.

Untuk mengimplementasi **LoginSuccessHandler,** buat class LoginSuccessHandler didalam package **belajar.app.handler.** Berikut ini adalah sample code untuk LoginSuccessHandler

```
@Component
public class LoginSuccessHandler implements AuthenticationSuccessHandler{
    protected Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserDetailsService userDetailsService;

    @Value("${jwt.header}")
    private String tokenHeader;

    @Override
    public void onAuthenticationSuccess(
            HttpServletRequest request,
            HttpServletResponse response, Authentication a)
            throws IOException, ServletException {
        handle(request, response, a);
        clearAuthenticationAttributes(request);
    }

    protected void handle(
            HttpServletRequest request,
            HttpServletResponse response,
            Authentication authentication) throws IOException {

        String username = authentication.getName();
        logger.debug("Handler Login Success for username [{}]",
```

```
                username);

        final UserDetails userDetails =
                userDetailsService.loadUserByUsername(username);
        final String token = jwtTokenUtil
                .generateToken(userDetails);

        response.setHeader(tokenHeader, token);
        response.setHeader(HttpHeaders.LOCATION,
                request.getServletContext()
                        .getContextPath() + "/#/");
    }

    protected void clearAuthenticationAttributes(
            HttpServletRequest request) {
        HttpSession session = request.getSession(false);
        if (session == null) {
            return;
        }
        session.removeAttribute(
                WebAttributes.AUTHENTICATION_EXCEPTION);
    }

}
```

Perhatikan statement dibawah ini, statement ini digunakan untuk menggenerate token pada saat login berhasil. Sehingga perlu dibuatkan class **JwtTokenUtil,** dan JwtTokenUtil di inisialisasi sebagai object di spring.

```
...
final String token = jwtTokenUtil
                .generateToken(userDetails);
...

dan
...
@Autowired
    private JwtTokenUtil jwtTokenUtil;
...
```

Berikut ini adalah isi dari class **JwtTokenUtil**

```
@Component
public class JwtTokenUtil implements Serializable {

    private static final long serialVersionUID = -3301605591108950415L;

    private static final String CLAIM_KEY_USERNAME = "sub";
    private static final String CLAIM_KEY_AUDIENCE = "audience";
    private static final String CLAIM_KEY_CREATED = "created";

    @Value("${jwt.secret}")
```

```
    private String secret;

    @Value("${jwt.expiration}")
    private Long expiration;

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        claims.put(CLAIM_KEY_USERNAME, userDetails.getUsername());
        claims.put(CLAIM_KEY_AUDIENCE, "web");
        claims.put(CLAIM_KEY_CREATED, new Date());
        return generateToken(claims);
    }

    private String generateToken(Map<String, Object> claims) {
        return Jwts.builder()
                .setClaims(claims)
                .setExpiration(generateExpirationDate())
                .signWith(SignatureAlgorithm.HS512, secret)
                .compact();
    }
}
```

Kemudian untuk mengimplementasi **LoginFailureHandler,** buat class LoginFailureHandler didalam package **belajar.app.handler.** Flow-nya yaitu ketika Login Gagal maka server akan mengirimkan pesan error dalam bentuk JSON ke client. Berikut ini adalah sample code untuk LoginFailureHandler

```
@Component
public class LoginFailureHandler implements AuthenticationFailureHandler {
    @Override
    public void onAuthenticationFailure(
            HttpServletRequest req,
            HttpServletResponse res,
            AuthenticationException ae)
            throws IOException, ServletException {
        res.setStatus(HttpStatus.UNAUTHORIZED.value());

        HttpSession session = req.getSession();
        session.setAttribute(
                WebAttributes.AUTHENTICATION_EXCEPTION,
                ae.getMessage());

        try (PrintWriter writer = res.getWriter()) {
            writer.write("{\"code\":\""+res.getStatus()
                    + "\", \"status\":\"ERR\", "
                    + "\"message\":\""+ae.getMessage()+"\"}");
            writer.flush();
            writer.close();
        }
    }
}
```

Setelah membuat class LoginSuccessHandler dan LoginFailureHandler maka

langkah selanjutnya adalah mengimplement handler tersebut kedalam konfigurasi **HttpSecurity**. Langkah pertama yaitu buat variable loginFailureHandler dan loginSuccessHandler didalam class **SecurityConfig**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    ...
    @Autowired private LoginFailureHandler loginFailureHandler;
    @Autowired private LoginSuccessHandler loginSuccessHandler;

    ...
}
```

Kemudian tambahkan baris berikut ini di konfigurasi **HttpSecurity**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        ...
        http
            .authorizeRequests()
                ...
                .anyRequest().authenticated()
            .and()
                .formLogin().loginPage("/login").permitAll()
                .successHandler(loginSuccessHandler)
                .failureHandler(loginFailureHandler)
        ...
    }
}
```

HttpRequest with Token

Agar setiap HttpRequest server melakukan pengecekan token maka perlu menambahkan filter di HttpSecurity. Caranya adalah buat bean **authenticationTokenFilter** di class **SecurityConfig**

```
@Bean
public JwtAuthenticationTokenFilter
         authenticationTokenFilterBean() throws Exception {
    JwtAuthenticationTokenFilter authenticationTokenFilter =
             new JwtAuthenticationTokenFilter();
    authenticationTokenFilter.setAuthenticationManager(
             authenticationManagerBean());
    return authenticationTokenFilter;
}
```

Kemudian daftarkan filter tersebut didalam konfigurasi **HttpSecurity**

```
http
    .addFilterBefore(authenticationTokenFilterBean(),
         UsernamePasswordAuthenticationFilter.class);
```

Setelah itu buat Filter untuk mengecek token setiap ada HttpRequest. Buat class dengan nama **JwtAuthenticationTokenFilter** didalam package **belajar.app.security**

```
public class JwtAuthenticationTokenFilter
        extends UsernamePasswordAuthenticationFilter {

    private final Logger logger =
            LoggerFactory.getLogger(this.getClass());

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Value("${jwt.header}")
    private String tokenHeader;

    @Override
    public void doFilter(
            ServletRequest request,
            ServletResponse response,
            FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String authToken = httpRequest.getHeader(this.tokenHeader);

        if(StringUtils.hasText(authToken)
                && authToken.startsWith("Bearer "))
            authToken = authToken.substring(7);
        String username = jwtTokenUtil.getUsernameFromToken(authToken);
        logger.info("Username login is {}", username);

        if (username != null
                && SecurityContextHolder.getContext()
                        .getAuthentication() == null) {

            logger.info("Security Context authentication is null");
            UserDetails userDetails =
                    this.userDetailsService.loadUserByUsername(username);
            if (jwtTokenUtil.validateToken(authToken, userDetails)) {
```

```
                    UsernamePasswordAuthenticationToken authentication =
                            new UsernamePasswordAuthenticationToken(
                                    userDetails,
                                    null,
                                    userDetails.getAuthorities());
                    authentication.setDetails(
                            new WebAuthenticationDetailsSource()
                                    .buildDetails(httpRequest));
                    SecurityContextHolder.getContext()
                            .setAuthentication(authentication);
                }
            }

            chain.doFilter(request, response);
        }
    }
```

Kemudian buat method di class **JwtTokenUtil,** method nya yaitu **getUsernameFromToken()** untuk mengambil nama user yang berada didalam token

```
    ...
    private Claims getClaimsFromToken(String token) {
        Claims claims;
        try {
            claims = Jwts.parser()
                    .setSigningKey(secret)
                    .parseClaimsJws(token)
                    .getBody();
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }

    public String getUsernameFromToken(String token) {
        String username;
        try {
            final Claims claims = getClaimsFromToken(token);
            username = claims.getSubject();
        } catch (Exception e) {
            username = null;
        }
        return username;
    }

    ...
```

Setelah itu buat method **validateToken()** didalam class JwtTokenUtil. Yang berfungsi untuk memvalidasi apakah token tersebut masih berlaku atau tidak.

```
...
public Date getExpirationDateFromToken(String token) {
    Date expiration;
    try {
        final Claims claims = getClaimsFromToken(token);
        expiration = claims.getExpiration();
    } catch (Exception e) {
        expiration = null;
    }
    return expiration;
}

private Boolean isTokenExpired(String token) {
    final Date expiration = getExpirationDateFromToken(token);
    return expiration.before(new Date());
}

public Boolean validateToken(String token, UserDetails userDetails) {
    SecManUserDetails user = (SecManUserDetails) userDetails;
    final String username = getUsernameFromToken(token);
    return (
            username.equals(user.getUsername())
                    && !isTokenExpired(token));
}

...
```

2. Client Side (JS)

Function Login (Jquery)

Pada saat login sukses client app akan mendapatkan response berupa JSON dari server. Oleh karena itu di client app harus dibuatkan function untuk menyimpan token tersebut ke dalam sebuah variable atau browser storage. Berikut ini adalah contoh untuk menyimpan token.

```
function processLogin() {
    $.ajax({
        url: window.location.href,
        method: 'POST',
        type: 'text/html',
        contentType: 'application/x-www-form-urlencoded',
        data: $('.form-signin').serialize(), '
        success: function (data, textStatus, request) {
            localStorage.setItem('token',
                    request.getResponseHeader('Authorization'));
            window.location.replace(
                    request.getResponseHeader('Location'));
        },
        error: function (xhr, ajaxOptions, thrownError) {
            window.location.href = 'login?error';
        }
```

```
    });
}
```

Ajax Request (JQuery)

Karena setiap request harus mengirim token sebagai authentication maka pada saat melakukan Ajax Request method nya harus seperti ini

```
function listUser() {
    var token = localStorage.getItem('token');
    $.ajax({
        url: '/api/user',
        method: 'GET',
        type: 'text/html',
        contentType: 'application/json',
        headers: {
            'Authorization': 'Bearer ' + token
        }
    });
}
```

Ajax Request (AngularJS)

Jika menggunakan AngularJS proses penambahan token pada headers bisa dilakukan di interceptors, sehingga secara otomatis setiap melakukan HttpRequest selalu ditambahkan token. Berikut ini adalah sample code penggunaan interceptor pada AngularJS

```
...
.service('APIInterceptor', function ($q, $location, $window) {
    var service = this;
    service.request = function (config) {
        var token = localStorage.getItem('token');
        if (token) {
            config.headers = config.headers || {};
            config.headers.Authorization = 'Bearer ' + token;
        }

        return config;
    };
})
.config(function ($httpProvider) {
  $httpProvider.interceptors.push('APIInterceptor');
})
...
```

3. Client Side (Android)

B. Request Payload

Agar payload data yang kita kirim ke server lebih aman, maka dari client app perlu melakukan konversi payload data (JSON) ke dalam bentuk token. Kemudian disisi server harus melakukan decode untuk merubah token ke bentuk original payload. Berikut ini adalah urutan untuk mengirim Request Payload Token ke server

1. Server Side (Java)

Agar request data token yang dikirim dari client bisa diambil original payload-nya maka di server harus membuat **CustomRequestHandler** untuk konversi dari token ke original payload.

```
public class CustomRequestHandler
        extends HttpServletRequestWrapper {

    private final Logger log =
            LoggerFactory.getLogger(CustomRequestHandler.class);

    public CustomRequestHandler(
            HttpServletRequest request) throws IOException {
        super(request);
    }

    @Override
    public BufferedReader getReader() throws IOException {
        return new BufferedReader(
                new InputStreamReader(this.getInputStream()));
    }

    public byte[] getRequestBody() throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        String body = IOUtils.toString(super.getInputStream(), "UTF-8");
        Object originalPayload = parseToken(body);

        String sOrigPayload = mapper.writeValueAsString(originalPayload);
        log.info("ORIGINAL PAYLOAD FROM TOKEN [{}]", sOrigPayload);

        return sOrigPayload.getBytes("UTF-8");
    }

    public Object parseToken(String token) {
        log.info("TOKEN TO PARSE [{}]", token);
        try {
            ObjectMapper mapper = new ObjectMapper();
            Claims body = Jwts.parser()
```

```
                    .setSigningKey("bismillah".getBytes("UTF-8"))
                    .parseClaimsJws(token)
                    .getBody();

            log.info("JSON PAYLOAD [{}]",
                    mapper.writerWithDefaultPrettyPrinter()
                            .writeValueAsString(body));

            return body;
        } catch (JwtException | ClassCastException
                | JsonProcessingException
                | UnsupportedEncodingException e) {
            log.error(e.getMessage(),e);
            return null;
        }
    }

    @Override
    public ServletInputStream getInputStream() throws IOException {
        final ByteArrayInputStream byteArrayInputStream =
                new ByteArrayInputStream(getRequestBody());
        ServletInputStream servletInputStream =
                new ServletInputStream() {
            @Override
            public int read() throws IOException {
                return byteArrayInputStream.read();
            }

            @Override
            public boolean isFinished() {
                return byteArrayInputStream.available() == 0;
            }

            @Override
            public boolean isReady() {
                return true;
            }

            @Override
            public void setReadListener(ReadListener rl) {
                throw new RuntimeException("Not implemented");
            }
        };
        return servletInputStream;
    }
}
```

Kemudian **CustomRequestHandler** tersebut harus di register sebagai **Filter**

```
public class RestFilter implements Filter {

    @Override
    public void init(FilterConfig fc) throws ServletException {
    }
```

```
    @Override
    public void doFilter(ServletRequest sr,
            ServletResponse sr1, FilterChain fc)
            throws IOException, ServletException {
        CustomRequestHandler requestHandler =
                new CustomRequestHandler((HttpServletRequest) sr);
        fc.doFilter(requestHandler, sr1);
    }

    @Override
    public void destroy() {
    }

}
```

Setelah didaftarkan sebagai **Filter** maka langkah berikutnya yaitu mendaftarkan **RestFilter** tersebut sebagai bean di spring. Konfigurasi ini di tambahkan di class **WebConfig**

```
@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {
    @Bean
    public FilterRegistrationBean httpRequestFilter() {
        FilterRegistrationBean bean =
                new FilterRegistrationBean(new RestFilter());
        bean.setOrder(0);
        return bean;
    }
}
```

2. Client Side (JS)

Ajax Request (JQuery)

Pada saat melakukan ajax request, sebelum proses mengirim request ke server dilakukan maka original payload (JSON) harus dikonversi dulu ke bentuk token

```
function save() {
    var data_json = {
       subject : cookie_data.data.username,
       msg_id : Cookies.getJSON('chat_cookie').msg_id,
       msg  : myhtml,
       msgType  : 'text/plain'
    }

    var token = localStorage.getItem('token');
    var tNow = KJUR.jws.IntDate.getNow();
    var tEnd = tNow + 1;
```

```
    var oHeader = {'alg': 'HS512', 'typ':'JWT'};
    var oPayload = {};
    oPayload.data = data_json;
    oPayload.iss = 'windrunner-customer-web';
    oPayload.iat = tNow;
    oPayload.exp = tEnd;
    oPayload.aud = url;

    var sHeader = JSON.stringify(oHeader);
    var sPayload = JSON.stringify(oPayload);
    var sJWT = KJUR.jws.JWS.sign(
                    "HS512", sHeader, sPayload, "bismillah");
    $.ajax({
        url: '/api/send',
        method: 'POST',
        dataType: 'json',
        contentType: 'application/json',
        data : sJWT,
        headers: {
            'Authorization': 'Bearer ' + token
        }
    });
}
```

Ajax Request (AngularJS)

Jika menggunakan AngularJS konversi original payload ke bentuk token bisa dilakukan di interceptors

```
...
.service('APIInterceptor', function ($q, $location, $window) {
    var service = this;
    service.request = function (config) {
        var token = localStorage.getItem('token');
        if (token) {
            config.headers = config.headers || {};
            config.headers.Authorization = 'Bearer ' + token;
        }

        if(config.method === 'POST' || config.method === 'PUT') {
          var oHeader = {'alg': 'HS512', 'typ':'JWT'};
          var oPayload = {};

          if(config.data) oPayload=config.data;

          var sHeader = JSON.stringify(oHeader);
          var sPayload = JSON.stringify(oPayload);
          var sJWT = KJUR.jws.JWS.sign(
                      "HS512", sHeader, sPayload, "bismillah");
          if(config.data) config.data=sJWT;
        }
        return config;
    };
})
.config(function ($httpProvider) {
  $httpProvider.interceptors.push('APIInterceptor');
```

```
})
```

3. Client Side (Android)

4. Penutup

Syarat yang harus diketahui oleh pembaca sebelum mencoba mempraktekan apa yang ada dibuku ini, yaitu pembaca harus memahami dulu :

- Pemrograman Java
- MongoDB
- Konsep Spring (XML / Java Config)
- Konsep Spring Security

Untuk lebih lengkapnya contoh program dari sample code yang ada dibuku ini dapat didownload di https://github.com/sulistionoadi/springboot-mongodb