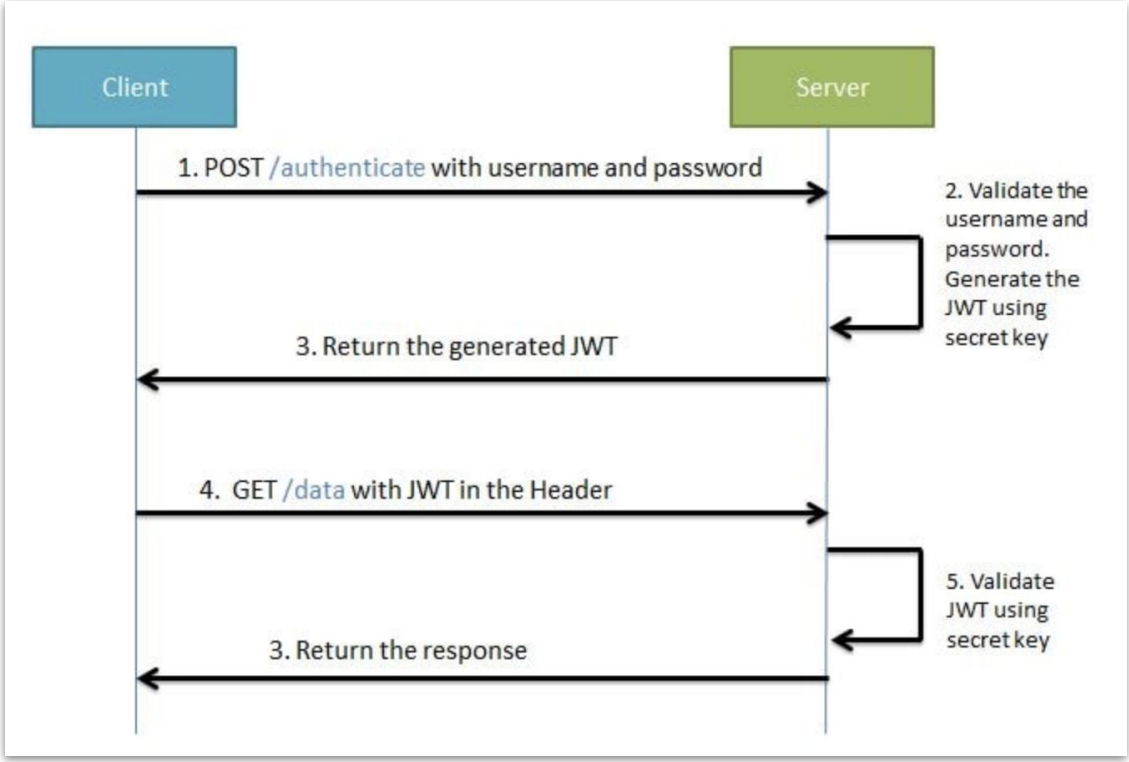
JWT Token

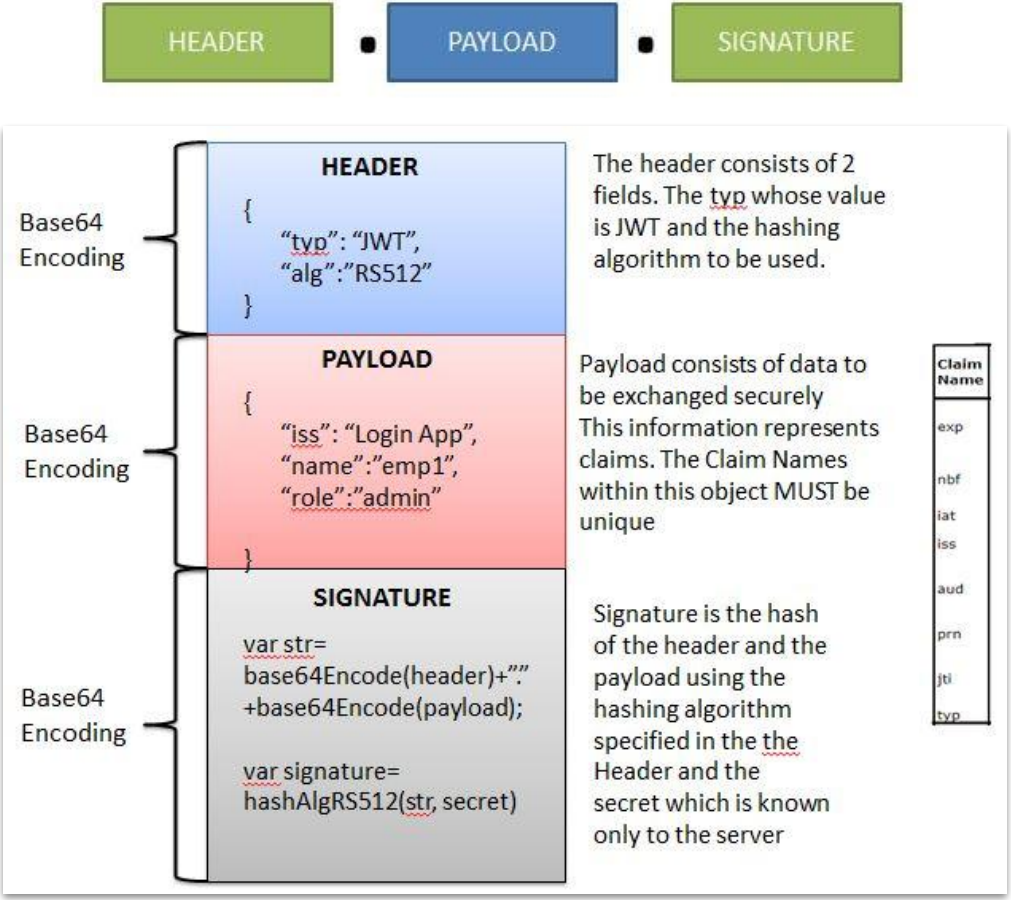
Authentication

What is JWT Token?

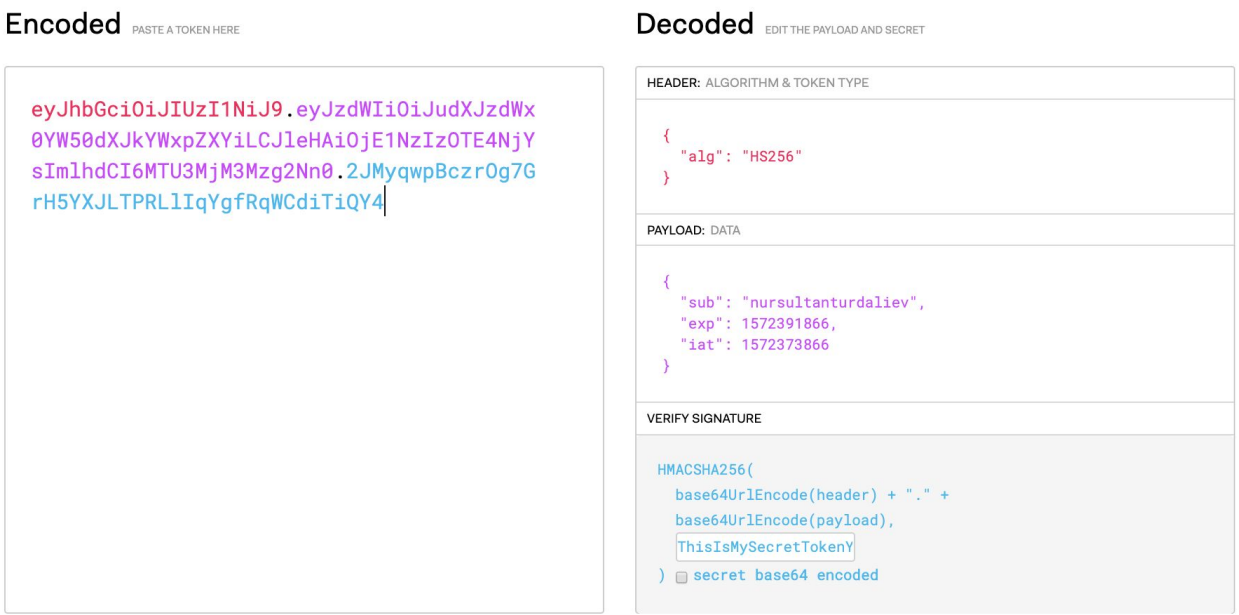
- JWT stands for JSON Web Token
- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object


Client
Server
1. POST /authenticate with username and password
2. Validate the username and password. Generate the JWT using secret key
3. Return the generated JWT
4. GET /data with JWT in the Header
5. Validate JWT using secret key
3. Return the response

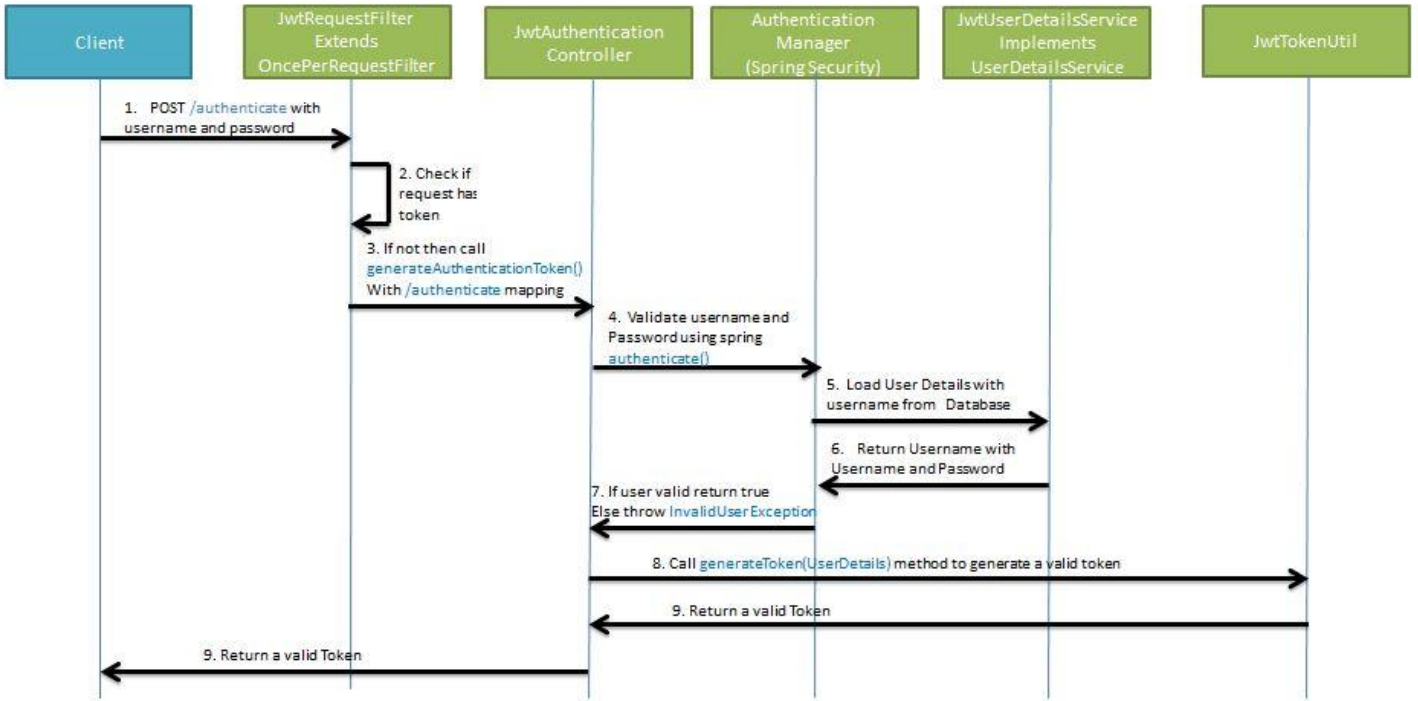
Structure Of JWT



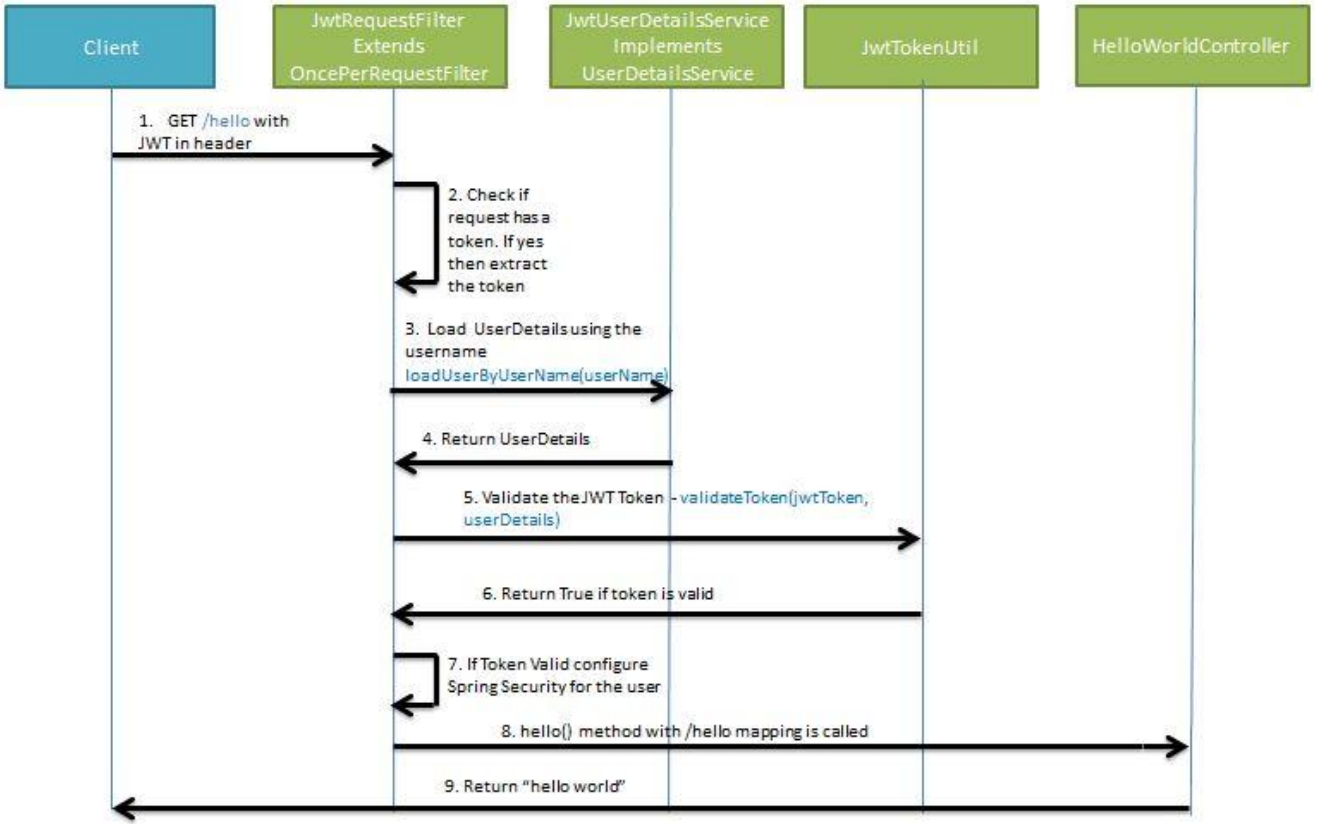
Encoded and Decoded JWT Token



Generating JWT



Validating JWT



Dependency

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
```

Jwt Token Util

```
@Component
public class JwtTokenUtil implements Serializable {
   private static final long serialVersionUID = -2550185165626007488L;
   public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;
   @Value("${jwt.secret}")
   private String secret;

  private String doGenerateToken(Map<String, Object> claims, String subject) {
      return Jwts.builder().setClaims(claims)
              .setSubject(subject)
              .setIssuedAt(new Date(System.currentTimeMillis()))
              .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY *
1000))
              .signWith(SignatureAlgorithm.HS512, secret).compact();
   }
   public Boolean validateToken(String token, UserDetails userDetails) {
      final String username = getUsernameFromToken(token);
      return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
   }
}
```

JWT Authentication Controller

```
@RestController
@CrossOrigin
public class JwtAuthenticationController {
  @Autowired
  private AuthenticationManager authenticationManager;
  @Autowired
  private JwtTokenUtil jwtTokenUtil;
  @Autowired
  private UserDetailsServiceImpl userDetailsService;
  @RequestMapping(value = "/authenticate", method = RequestMethod.POST)
  public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) {
      authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());
      final UserDetails userDetails = userDetailsService
              .loadUserByUsername(authenticationRequest.getUsername());
      final String token = jwtTokenUtil.generateToken(userDetails);
      return ResponseEntity.ok(new JwtResponse(token));
  }
  private void authenticate(String username, String password) throws Exception {
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));
  }
}
```

Jwt Request

```
import java.io.Serializable;
public class JwtRequest implements Serializable {
   private static final long serialVersionUID = 5926468583005150707L;
   private String username;
   private String password;
}
```

JwtResponse

```
import java.io.Serializable;

public class JwtResponse implements Serializable {
   private static final long serialVersionUID = -8091879091924046844L;
   private final String jwttoken;
   public JwtResponse(String jwttoken) {
      this.jwttoken = jwttoken;
   }
   public String getToken() {
      return this.jwttoken;
   }
}
```

Jwt Request Filter

```
@Component
public class JwtRequestFilter extends OncePerRequestFilter {
  @Autowired
  private UserDetailsServiceImpl jwtUserDetailsService;
  @Autowired
  private JwtTokenUtil jwtTokenUtil;
  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws ServletException, IOException {
    final String requestTokenHeader = request.getHeader("Authorization");
    String username = null,jwtToken = null;
    if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
      jwtToken = requestTokenHeader.substring(7);
      try {
        username = jwtTokenUtil.getUsernameFromToken(jwtToken);
      } catch (IllegalArgumentException e) {
        System.out.println("Unable to get JWT Token");
      } catch (ExpiredJwtException e) {
        System.out.println("JWT Token has expired");
      }
    } else {
      logger.warn("JWT Token does not begin with Bearer String");
    }
    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
      UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);
      if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {
        UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(
            userDetails, null, userDetails.getAuthorities());
        usernamePasswordAuthenticationToken
            .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
      }
    }
    chain.doFilter(request, response);
  }
}
```

Authentication Entry Point

```
@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint,
Serializable {
   private static final long serialVersionUID = -7858869558953243875L;

   @Override
   public void commence(HttpServletRequest request, HttpServletResponse
response,
                        AuthenticationException authException) throws IOException {
      response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
"Unauthorized");
   }
}
```

Web Security Configuration

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/authenticate").permitAll()
        .anyRequest().authenticated()
        .and()
        .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

  http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
}
```

Exercises

1. Enable both form login and jwt login
2. For all api requests require JWT, for others form login
3. Automatically attach transaction creation to the currently logged in user
4. Whenever there is a new transaction send transaction pending email

Recources

1. https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world