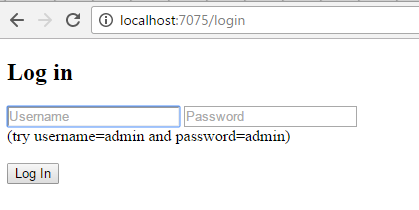
One service is running on 7075 port .



On clicking Login **LoginController.java** executed to generate JWT token and set the token to cookies.

String token = JwtUtil.*generateToken*(***signingKey***, username);

**JwUtil.java** is responsible to genereate jwt token :

***public******static*** *String generateToken(String signingKey, String subject) {*

***long*** *nowMillis = System.currentTimeMillis();*

*Date now =* ***new*** *Date(nowMillis);*

*JwtBuilder builder = Jwts.builder()*

*.setSubject(subject)*

*.setIssuedAt(now)*

*.signWith(SignatureAlgorithm.****HS256****, signingKey);*

*System.****out****.println("generated token inside JwtUtil ::::"+builder.compact());*

***return*** *builder.compact();*

*}*

generated token inside JwtUtil ::::

CookieUtil.*create*(httpServletResponse, ***jwtTokenCookieName***, token, **false**, -1, "localhost");

On the Othe Side one more service running on another port to fetch the encoded user data in the decoded format.



JWtFilter.java files does this decoding

String username = JwtUtil.*getSubject*(httpServletRequest, ***jwtTokenCookieName***, ***signingKey***);.

Originally JWT token combining in to 3 parts (header,payload,signature)

Payload has the original data.

#### 1. Header

The first part of a JWT is an encoded string representation of a simple JavaScript object which describes the token along with the hashing algorithm used.

#### 2. Payload

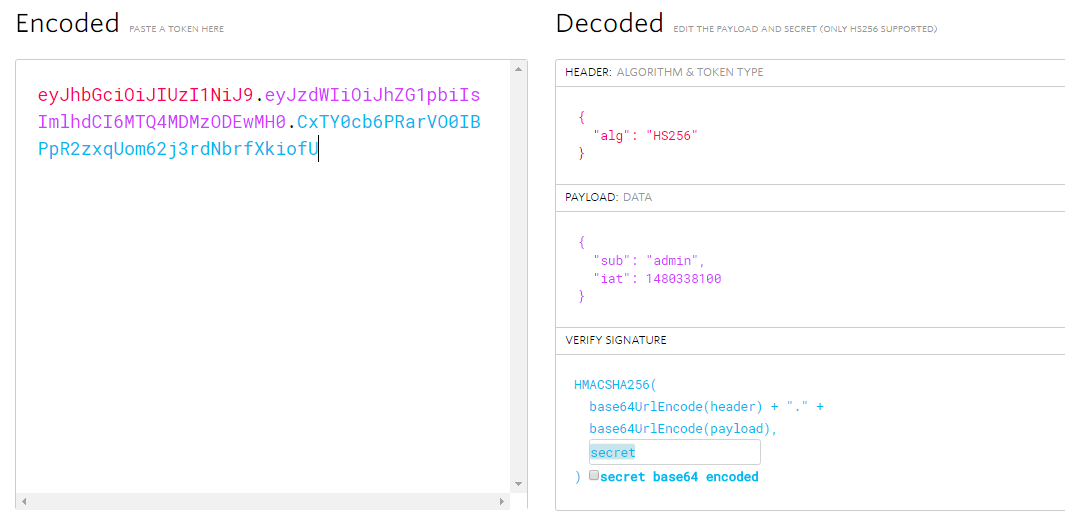
The second part of the JWT forms the core of the token. Payload length is proportional to the amount of data you store in the JWT. General rule of thumb is: store the bare minimum in the JWT.

#### 3. Signature

The third, and final, part of the JWT is a signature generated based on the header (part one) and the body (part two) and will be used to verify that the JWT is valid.

We have predefined toll to decode the encoded String .

<https://jwt.io/#debugger>



Output string (“**admin**”) placed under payload section. This is how JWT works.