Spring boot react full stack

1. Basic Auth

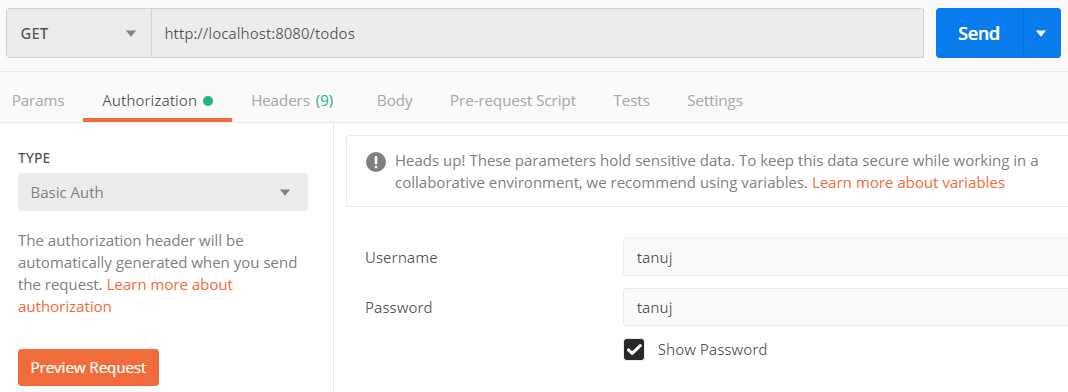
Suppose you have react app and it wants to talk to the backend, in our case it is sprinb boot. You need to have spring security dependency which will secure all the rest end point automatically.

You need to have user name and password.

spring.security.user.name=tanuj  
spring.security.user.password=tanuj

Now, every time when you make request you need to give these credentials to make all the rest end point available.

If you want to make async call to these rest endpoints, you need to paas these creds as a part of headers while making the request such as below.



If you want to make request through rest app also you need to pass these credentials as a part of headers such as below.

  getAllTodos() {

        const GET\_URL = 'http://localhost:8080/todos';

        let username = 'tanuj';

        let password = 'tanuj';

        let basicAuthHeader = 'Basic ' + window.btoa(username + ':' + password)

        return axios.get(GET\_URL, {

            headers: {

                authorization: basicAuthHeader

            }

        })

    }

Signature of the basic authorization is “Basic <encripted form of username and password>” .

In the above you will need to send header in every request. So to prevent writing code in every request we can use axios interceptor like this

  // Each request will get intercept after login successful

    setUpAxiosInterceptors() {

        let username = 'tanuj';

        let password = 'tanuj';

        let basicAuthHeader = 'Basic ' + window.btoa(username + ':' + password)

        axios.interceptors.request.use(

config => {

                    config.headers.authorization = basicAuthHeader;

                return config;

            })

    }

Flow

1. Verify from the valid user

**In handleLogin()**

* Calls a method in AuthenticateService class

 AuthenticateService.*authenticateUser*(this.state.username, this.state.password)

* authenticateUser(username, password) will call backend using axios and sends headers along with the request.

 // Hits Backend to authenticate user

*authenticateUser*(username, password) {

        let basicAuthHeader = 'Basic ' + window.btoa(username + ':' + password)

        const GET\_URL = 'http://localhost:8080/authenticate';

        return axios.get(GET\_URL, {

            headers: {

                authorization: basicAuthHeader

            }

        })

    }

1. In first(1.) step, If its 200 or success, store the details in sessionStorage

AuthenticateService.authenticateUser(this.state.username, this.state.password)

            .then(() => {

                AuthenticateUser.*storeLoginDetails*(this.state.username, this.state.password)

                this.getAllTodos();

                this.setState({

                    loginSuccess: true

                })

            })

            .catch(() => {

                this.setState({

                    invalidCredentials: true

                })

            })

    }

* After storing in session storage set up axios interceptor that will intercept all the request sent by react app.

*storeLoginDetails(username, password) {*

        let basicAuthHeader = 'Basic ' + window.btoa(username + ':' + password);

        sessionStorage.setItem('authenticatedUser', username);

        GetRequestService.*setUpAxiosInterceptors*(basicAuthHeader);

    }

* This is how we set up axios interceptor

// Each request will get intercept after login successful

*setUpAxiosInterceptors*(basicAuthHeader) {

        axios.interceptors.request.use(

            config => {

                if (AuthenticateUser.*isUserLoggedIn*()) {

                    config.headers.authorization = basicAuthHeader;

                }

                return config;

            })

    }

*isUserLoggedIn*() {

        let user = sessionStorage.getItem('authenticatedUser');

        if (user === null) return false;

        return true;

    }

JWT

In basic authenticated, there is no expiry time. So, once you get the username and password credentials, you can call the rest api’s unlimited times.

Json web token contains

1. Header: contains the algo u want to use

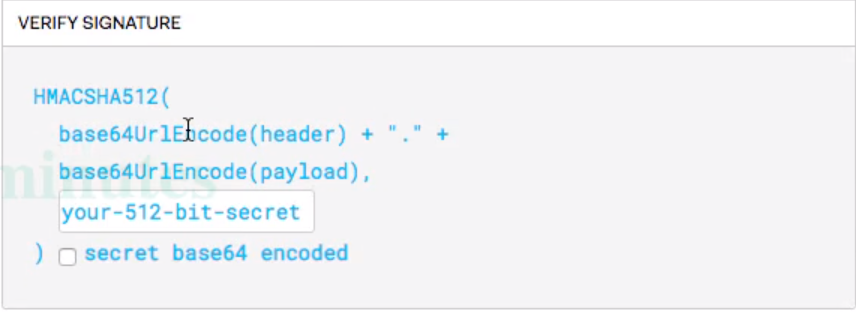
* {

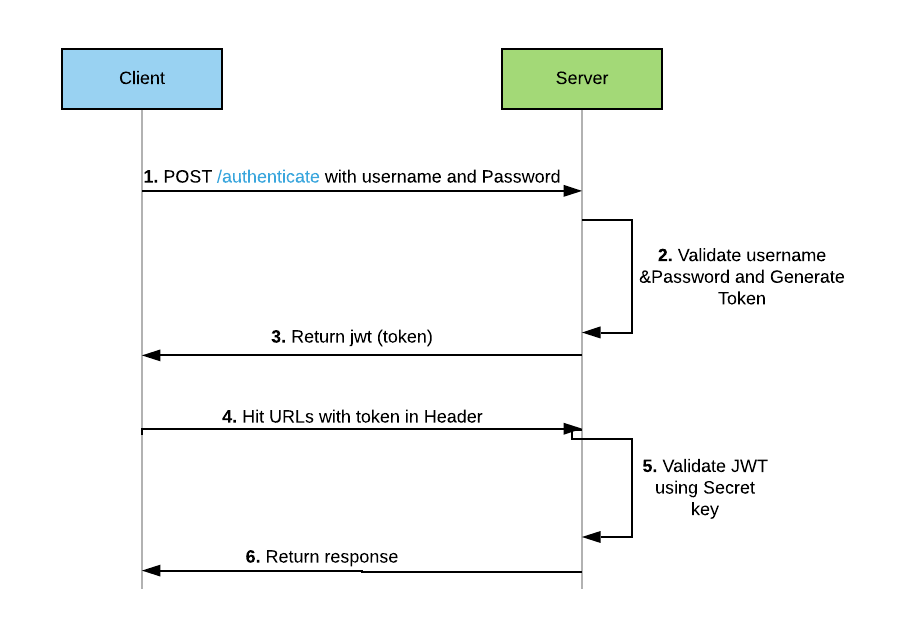
“alg”: ”HS512”,

“typ”: ”JWT”

}

1. Payload
2. Verify signature





**Generating JWT:** Expose a POST API with mapping **/authenticate**. On passing the correct username and password, it will generate a JSON Web Token (JWT).

**Validating JWT:** If a user tries to access the GET API with mapping **/hello**, it will allow access only if a request has a valid JSON Web Token (JWT).

**Generating JWT**



**application.properties**

logging.level.org.springframework=info  
jwt.signing.key.secret=mySecret  
jwt.get.token.uri=/authenticate  
jwt.refresh.token.uri=/refresh  
jwt.http.request.header=Authorization  
jwt.token.expiration.in.seconds=604800

**JwtTokenRequest.java**

* This is the username and password that POST request will contain as a body

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class JwtTokenRequest implements Serializable {  
  
 private String username;  
 private String password;  
}

**JwtTokenResponse.java**

* This will contain token in it e.g,

{

  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0YW51aiIsImV4cCI6MTU3ODIzODUwNCwiaWF0IjoxNTc3NjMzNzA0fQ.-Xi9jvew78d\_cZxXix91wcMSfNhg5UQZVLN8alKSHod4uE\_UeKVd3K9jxeGbvtCaEHhWD7lyZVkkex2QEs6VxA"

}

@Data  
@AllArgsConstructor  
public class JwtTokenResponse implements Serializable {  
  
 private static final long *serialVersionUID* = 8317676219297719109L;  
  
 private final String token;  
}

* ***Body:*** http://localhost:8080/authenticate

{

"username":"tanuj",

"password":"dummy"

}

**JwtController**

1. *Authenticate username and password\*
2. *Get username and password from the DB and create UserDetails From it  
    OR  
    In our case we have used inMemory*
3. *Generate token from the UserDetails*
4. *Returning token as JSON*

This method returns ResponseEntity that contains JwtTokenResponse as JSON

@PostMapping(value = "/authenticate")

public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtTokenRequest authenticationRequest)  
 throws AuthenticationException {

*/\*\*  
 \* 1. Authenticate username and password  
 \*/*

*authenticate* (authenticationRequest.getUsername(), authenticationRequest.getPassword());

*/\*\*  
 \* 2. Get username and password from the DB and create UserDetails From it  
 \* OR  
 \* In our case we have used inMemory  
 \*/*

final UserDetails userDetails = jwtInMemoryUserDetailsService

.*loadUserByUsername*(authenticationRequest.getUsername());

*/\*\*  
 \* 3. Generate token from the UserDetails  
 \*/*

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

*/\*\*  
 \* 4. Returning token as JSON  
 \*/*  
 return ResponseEntity.*ok*(new JwtTokenResponse(token));  
}

***/\*\*  
 \* 1. Authenticate username and password  
 \*/***

import org.springframework.security.authentication.AuthenticationManager;

@Autowired  
private AuthenticationManager *authenticationManager***;**

private void *authenticate*(String username, String password) {  
 Objects.*requireNonNull*(username);  
 Objects.*requireNonNull*(password);  
  
 try {  
 authenticationManager.

*authenticate (*new *UsernamePasswordAuthenticationToken(username, password));* } catch (DisabledException e) {  
 throw new AuthenticationException("USER\_DISABLED", e);  
 } catch (BadCredentialsException e) {  
 throw new AuthenticationException("INVALID\_CREDENTIALS", e);  
 }  
}

AuthenticationManager class has authenticate method that authenticates the token created by username and password.

***/\*\*  
 \* 2. Get username and password from the DB and create UserDetails From it  
 \* OR  
 \* In our case we have used inMemory  
 \*/***

*JwtInMemoryUserDetailsService implements UserDetailsService*

public interface *UserDetailsService* {  
 UserDetails loadUserByUsername(String var1) throws UsernameNotFoundException;  
}

* ***JwtUserDetails implements Spring’s UserDetails***

public class JwtUserDetails implements UserDetails {  
  
 private final Long id;  
 private final String username;  
 private final String password;  
 private final Collection<? extends GrantedAuthority> authorities;  
  
 public JwtUserDetails(Long id, String username, String password, String role) {  
 this.id = id;  
 this.username = username;  
 this.password = password;  
  
 List<SimpleGrantedAuthority> authorities = new ArrayList<SimpleGrantedAuthority>();  
 authorities.add(new SimpleGrantedAuthority(role));  
 this.authorities = authorities;  
 }  
  
 @JsonIgnore  
 public Long getId() { return id; }  
  
 @Override  
 public String getUsername() {return username; }  
  
 @JsonIgnore  
 @Override  
 public boolean isAccountNonExpired() { return true; }  
  
 @JsonIgnore  
 @Override  
 public boolean isAccountNonLocked() { return true; }  
  
 @JsonIgnore  
 @Override  
 public boolean isCredentialsNonExpired() { return true; }  
  
 @JsonIgnore  
 @Override  
 public String getPassword() {return password;}  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() { return authorities; }  
  
 @Override  
 public boolean isEnabled() { return true; }  
}

@Service  
public class JwtInMemoryUserDetailsService implements UserDetailsService {  
  
 static List<JwtUserDetails> *users* = new ArrayList<>();  
  
 static {  
 *users*.add(

new JwtUserDetails

(

1L,

"tanuj",  
 "$2a$10$3zHzb.Npv1hfZbLEU5qsdOju/tk2je6W6PnNnY.c1ujWPcZh4PL6e", "ROLE\_USER\_2"

));  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException{

Optional<JwtUserDetails> findFirst = *users***.**stream()  
 .filter(user -> user.getUsername().equals(username))

.findFirst();  
  
 if (!findFirst.isPresent()) {  
 throw new UsernameNotFoundException(String.*format*("USER\_NOT\_FOUND '%s'.", username));   
 }  
  
 JwtUserDetails jwtUserDetails = findFirst.get();  
 return jwtUserDetails;

}  
}

***/\*\*  
 \* 3. Generate token from the UserDetails  
 \*/***

public String generateToken(UserDetails userDetails) {  
 Map<String, Object> claims = new HashMap<>();  
 return doGenerateToken (claims, userDetails.getUsername());  
}

private String doGenerateToken(Map<String, Object> claims, String subject) {  
 final Date createdDate = clock.now();  
 final Date expirationDate = calculateExpirationDate(createdDate);  
 return Jwts.*builder*().setClaims(claims).setSubject(subject).setIssuedAt(createdDate)  
 .setExpiration(expirationDate).signWith(SignatureAlgorithm.*HS512*, secret).compact();  
}

Refresh Token

@GetMapping(value = "/refresh")  
public ResponseEntity<?> refreshAndGetAuthenticationToken(HttpServletRequest request) {  
 String authToken = request.getHeader(tokenHeader);  
 final String token = authToken.substring(7);  
 String username = jwtTokenUtil.getUsernameFromToken(token);

JwtUserDetails user = (JwtUserDetails). jwtInMemoryUserDetailsService.loadUserByUsername(username);  
  
 if (jwtTokenUtil.canTokenBeRefreshed(token)) {  
 String refreshedToken = jwtTokenUtil.refreshToken(token);  
 return ResponseEntity.*ok*(new JwtTokenResponse(refreshedToken));  
 } else {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
}

public String refreshToken (String token) {  
 final Date createdDate = clock.now();  
 final Date expirationDate = calculateExpirationDate(createdDate);  
  
 final Claims claims = getAllClaimsFromToken(token);  
 claims.setIssuedAt(createdDate);  
 claims.setExpiration(expirationDate);  
  
 return Jwts.*builder*().setClaims(claims).signWith(SignatureAlgorithm.*HS512*, secret).compact();  
}