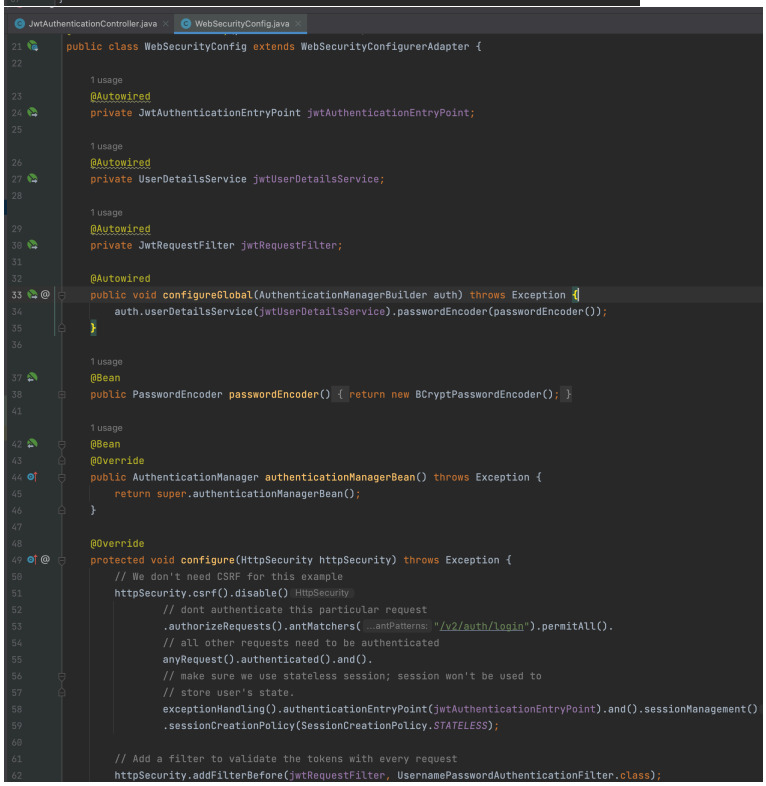
Implement JWT Security In SpringBoot

1. JWT Generation
    a. Code

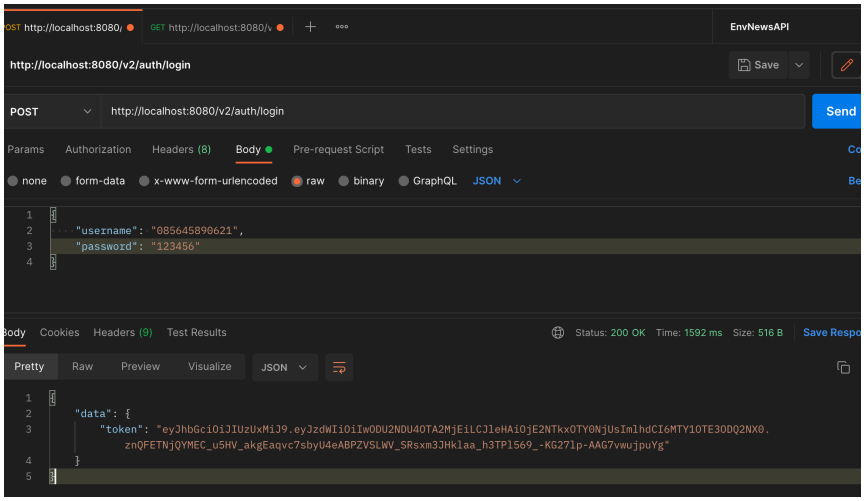
```java
public class JwtAuthenticationController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserDetailsService jwtInMemoryUserDetailsService;

    @RequestMapping(value = "/v2/auth/login", method = RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest)
            throws Exception {

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

        final UserDetails userDetails = jwtInMemoryUserDetailsService
                .loadUserByUsername(authenticationRequest.getUsername());

        final String token = jwtTokenUtil.generateToken(userDetails);

        Map<String, Object> res = new HashMap<>();
        res.put("data", new JwtResponse(token));

        return ResponseEntity.ok(res);
    }

    private void authenticate(String username, String password) throws Exception {
        Objects.requireNonNull(username);
        Objects.requireNonNull(password);

        try {
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```



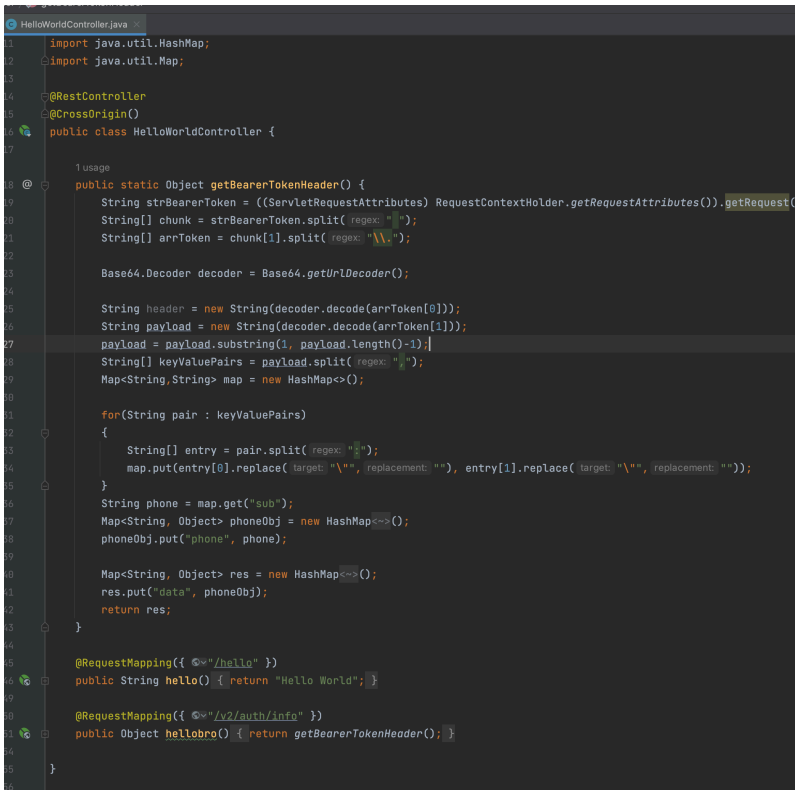
```java
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    @Autowired
    private UserDetailsService jwtUserDetailsService;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());
    }

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

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        // We don't need CSRF for this example
        httpSecurity.csrf().disable()
                // dont authenticate this particular request
                .authorizeRequests().antMatchers( ...antPatterns: "/v2/auth/login").permitAll().
                // all other requests need to be authenticated
                anyRequest().authenticated().and().
                // make sure we use stateless session; session won't be used to
                // store user's state.
                exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        // Add a filter to validate the tokens with every request
        httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
```

```java
@Configuration
public class JwtRequestFilter extends OncePerRequestFilter {

    1 usage
    @Autowired
    private JwtUserDetailsService jwtUserDetailsService;

    2 usages
    @Autowired
    private JwtTokenUtil jwtTokenUtil;

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

        final String requestTokenHeader = request.getHeader( s: "Authorization");

        String username = null;
        String jwtToken = null;
        // JWT Token is in the form "Bearer token". Remove Bearer word and get only the Token
        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
            jwtToken = requestTokenHeader.substring( beginIndex: 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( o: "JWT Token does not begin with Bearer String");
        }

        //Once we get the token validate it.
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);

            // if token is valid configure Spring Security to manually set authentication
            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(
                        userDetails, credentials: null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                // After setting the Authentication in the context, we specify
                // that the current user is authenticated. So it passes the Spring Security Configurations successfully.
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
```

b. Postman



2. JWT Extraction
    a. Code



```java
import java.util.HashMap;
import java.util.Map;

@RestController
@CrossOrigin()
public class HelloWorldController {

    1 usage
    public static Object getBearerTokenHeader() {
        String strBearerToken = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest(
        String[] chunk = strBearerToken.split( regex: " ");
        String[] arrToken = chunk[1].split( regex: "\\.");

        Base64.Decoder decoder = Base64.getUrlDecoder();

        String header = new String(decoder.decode(arrToken[0]));
        String payload = new String(decoder.decode(arrToken[1]));
        payload = payload.substring(1, payload.length()-1);
        String[] keyValuePairs = payload.split( regex: ",");
        Map<String,String> map = new HashMap<>();

        for(String pair : keyValuePairs)
        {
            String[] entry = pair.split( regex: ":");
            map.put(entry[0].replace( target: "\"", replacement: ""), entry[1].replace( target: "\"", replacement: ""));
        }
        String phone = map.get("sub");
        Map<String, Object> phoneObj = new HashMap<~>();
        phoneObj.put("phone", phone);

        Map<String, Object> res = new HashMap<~>();
        res.put("data", phoneObj);
        return res;
    }

    @RequestMapping({ ~"/hello" })
    public String hello() { return "Hello World"; }

    @RequestMapping({ ~"/v2/auth/info" })
    public Object hellobro() { return getBearerTokenHeader(); }

}
```

    b. Postman