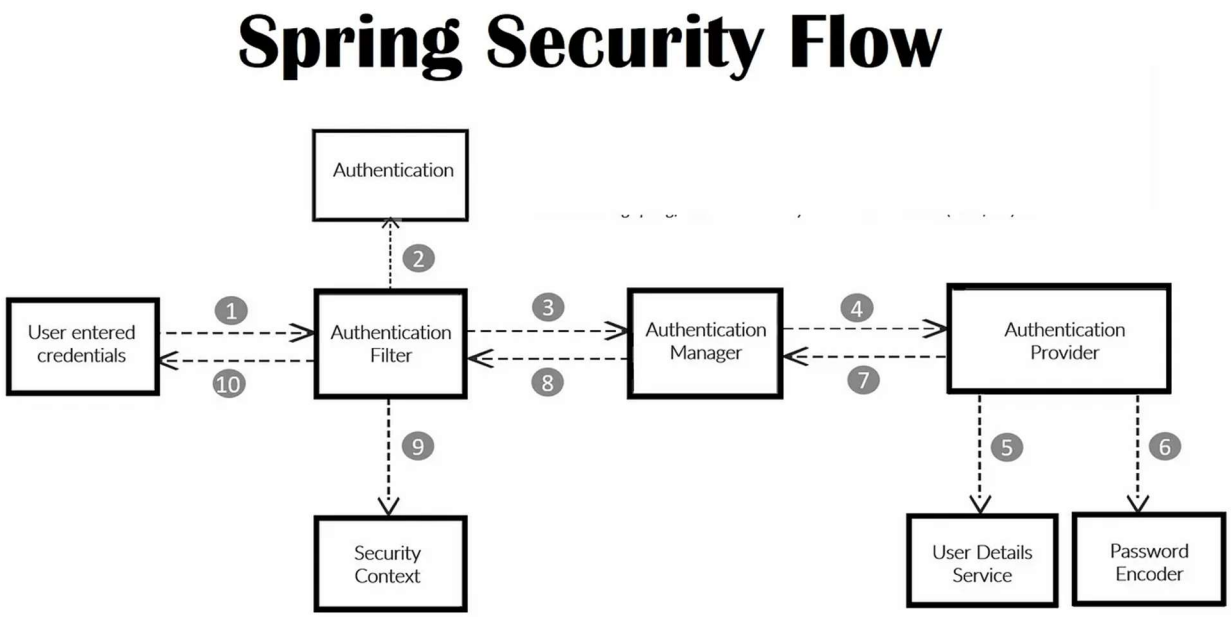
Spring Security:



Step 1: User Gives credentials to the login form and sends a request.

Step 2:

```
UserInformation userDetails = this.service.getUserByUserName(userInfo.getUserName());

  validating username and password by spring security and generation token for valid user
UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
        userInfo.getUserName(), userInfo.getPassword());

this.authenticationManager.authenticate(authToken);
String token = jwtTokenUtil.generateToken(userDetails);
```

Login forms details(username) gets used in the getUserByUserName(userInfo.getUserName()).

Now the authToken gets created like shown in the screenshot and it gets compared in the back with the password and username we got in the userDetails variable. If successful, we create a jwtToken for the user, and login is successful. If not, login is not successful and we send the message to the user.

And this all authentication process gets controlled by spring security in the security_config class.

```java
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

    authProvider.setUserDetailsService((UserDetailsService) this.userInformationService);
    authProvider.setPasswordEncoder(this.passwordEncoder());

    return authProvider;
}

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
    return authConfig.getAuthenticationManager();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
    // return NoOpPasswordEncoder.getInstance();
}
```

Here we create @Bean types for authentication manager which got used in the login method in userController. This contacts the authenticationProvider which compares the users credentials and allows user to login.

This marks the end for authentication when user is trying to login.

Step 3: Authorization part.

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    // "/login", "/logOut", "/checkTokenExpiry", "/checkTokenValidity", "/isLogin"

    http.cors(Customizer.withDefaults()).csrf(AbstractHttpConfigurer::disable)
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests((requests) -> requests.requestMatchers(
            "/login", "/logOut", "/checkTokenExpiry", "/checkTokenValidity", "/isLogin").permitAll()
            .anyRequest().authenticated())
            .exceptionHandling((exceptionHandling) -> exceptionHandling.authenticationEntryPoint(jwtAuthEntryPoint));

    http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
    http.authenticationProvider(this.authenticationProvider());
    return http.build();
}
```

After login, when a user trying to access or send any request to any api of the web application. It goes through some layers of security. Here any requests other than (/login, /logout, /checkTokenExpiry, /checkTokenValidity, /isLogin) requires authentication. And for that, we added http.addFilterBefore() method. Which calls an object jwtAuthfilter of the class JwtAuthFilter.

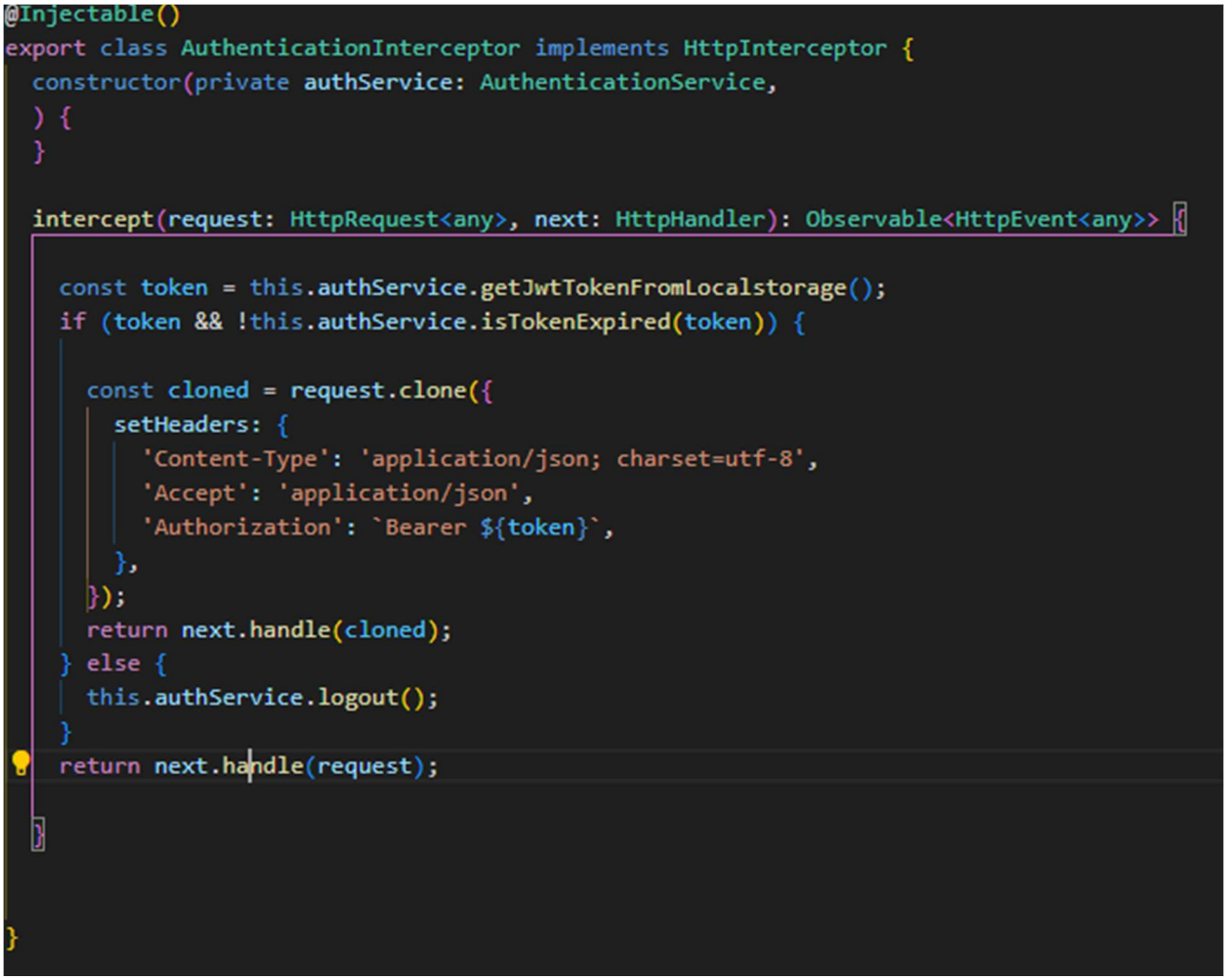
Step 4: Authentication Filter

```
30      @Override
        protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
32              throws ServletException, IOException {
33          String requestToken = request.getHeader("Authorization");
34          String userName = null;
            String jwtToken = null;
36          if (requestToken != null && requestToken.startsWith("Bearer ")) {
37              jwtToken = requestToken.substring(7);
38
39              try {
40                  userName = this.jwtTokenUtil.extractUsername(jwtToken);
41              } catch (Exception e) {
42  //                  e.printStackTrace();
43              }
44              // security
45              UserInformation userDetails = this.userDetailsService.loadUserByUsername(userName);
46
47              if (userName != null && SecurityContextHolder.getContext().getAuthentication() == null) {
48
49                  UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
50                          userDetails, null, null);  //grant authorities is null here
51
52                  authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
53                  SecurityContextHolder.getContext().setAuthentication(authToken);
54
55              }
56
57          }
```

When we send a request from the frontend, we send the jwttoken with it, so we can say that we're authenticated. Here that token kept in a variable requestToken in like 33. Than we cut the actual token from the authorization header and then extract the username from the token (line 40).

Then, we call the loadUserByUserName(userName) method to get user details to create the authToken, and that sets the authentication true for the user. And hence the user can access other api in the webapplication.

Step 5:



```
@Injectable()
export class AuthenticationInterceptor implements HttpInterceptor {
  constructor(private authService: AuthenticationService,
  ) {
  }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    const token = this.authService.getJwtTokenFromLocalstorage();
    if (token && !this.authService.isTokenExpired(token)) {

      const cloned = request.clone({
        setHeaders: {
          'Content-Type': 'application/json; charset=utf-8',
          'Accept': 'application/json',
          'Authorization': `Bearer ${token}`,
        },
      });
      return next.handle(cloned);
    } else {
      this.authService.logout();
    }
    return next.handle(request);

  }


}
```

Here in angular we made a class AuthenticationInterceptor which intercepts any requests made by the user, and sets the token with every request to be checked by the authFilter in the backend.

And finally this interceptor in imported in the app.module.ts so that it integrated with the frontend app, and intercepts every requests.