This Project is based on:

spring boot,

spring security,

spring JPA,

Android and H2 Database

What is Authentication?

It is the process of verifying one's identity. This can be utilized in a system where we want to associate an identity with some resources and make sure that no other identities have access to those resources. A password is a common way of implementing this method in practice which is also used in this project.

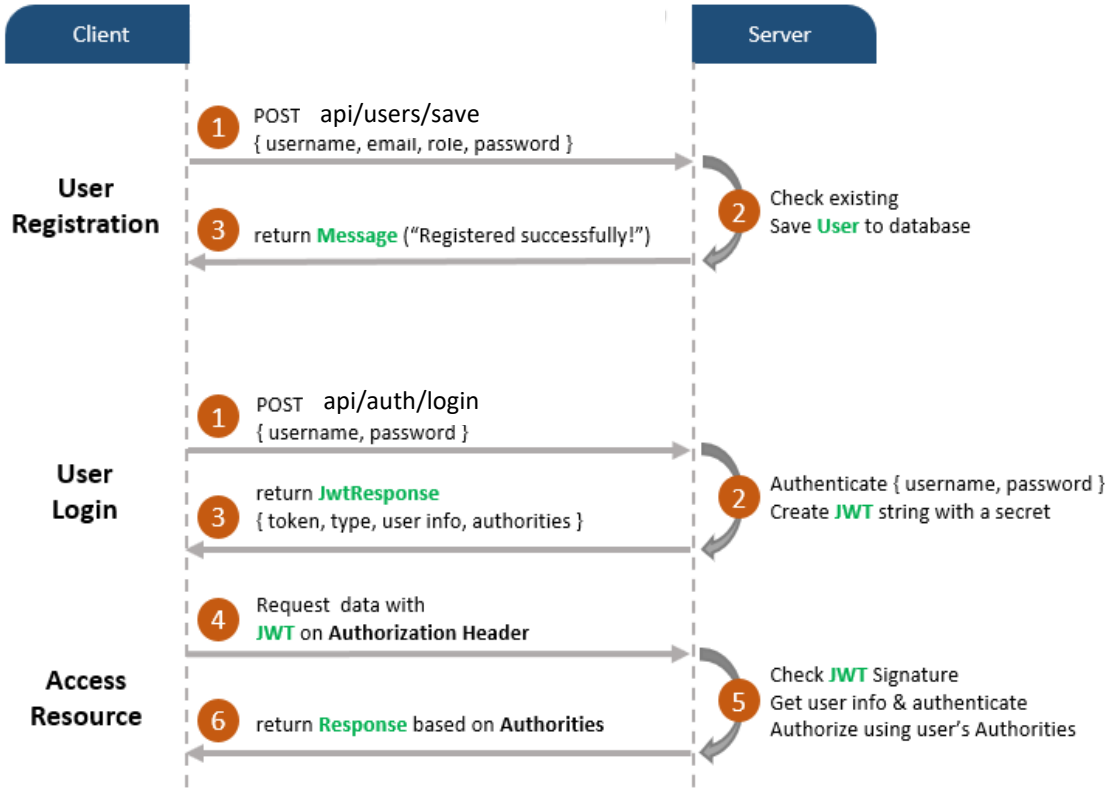
What is Authorization?

Authorization generally comes after successful authentication. Authorization procedures verify whether you have the authority to access the content or resources you have requested access to. Some of these procedures occur via access tokens.

These tokens contain security credential information concerning a user's level of privilege and the extent of their access rights. For example, when a user provides credentials to log into a system and that login information is authenticated, an access token, which indicates what access is permitted, is generated. When a user tries to access a specific resource, the contents of that token are then checked to determine if the action is authorized. In this project Json web tokens are used to authorize users.

Basic structure of JWT

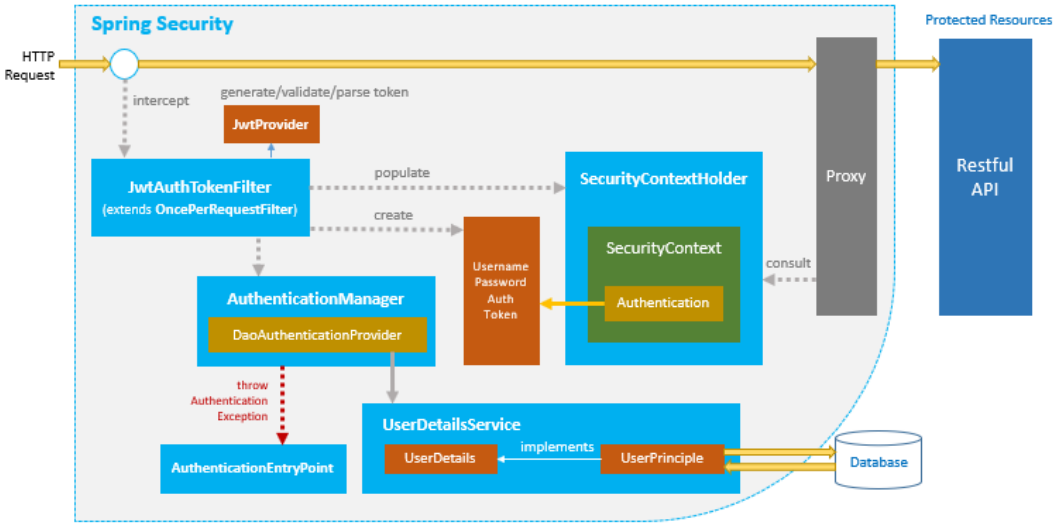
- header: It contains token type and algorithm used to make signature. Gets encoded to base64.
- payload: Any custom user data like username and email.
- signature: Hash of encoded header, payload and a secret key.



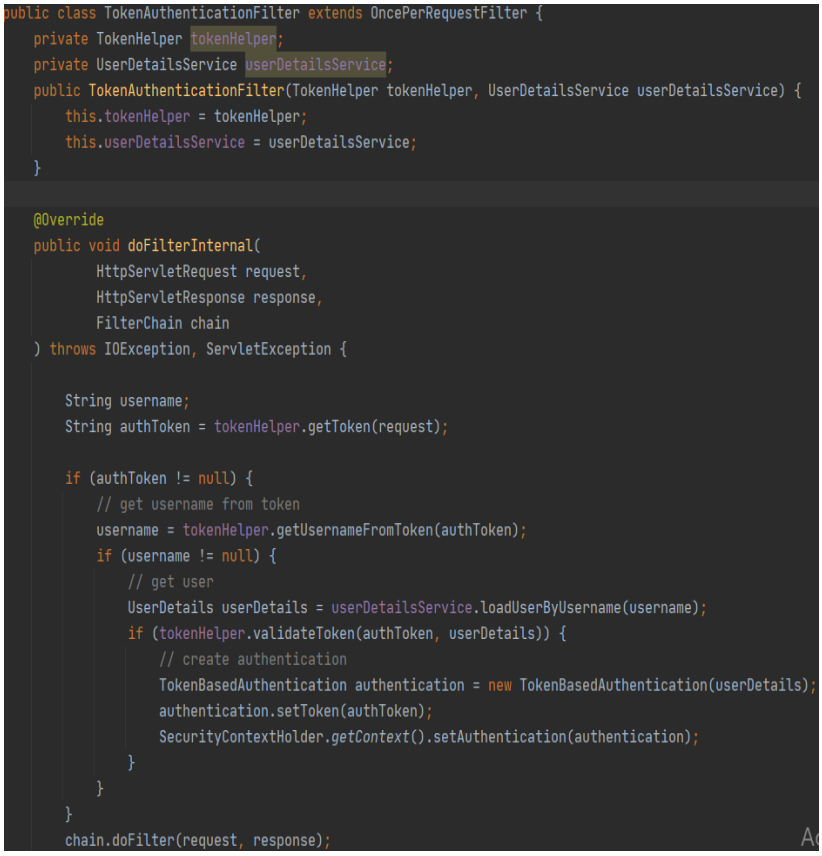
How to implement Authentication and Authorization

Using Spring Security which has an architecture that is designed to handle authentication and authorization.

Backend **Architecture of Spring Security**



Spring Security Authentication process: receive HTTP request, filter, authenticate, store Authentication data, generate token, get User details, authorize and handle exception.



```
public class TokenAuthenticationFilter extends OncePerRequestFilter {
    private TokenHelper tokenHelper;
    private UserDetailsService userDetailsService;
    public TokenAuthenticationFilter(TokenHelper tokenHelper, UserDetailsService userDetailsService) {
        this.tokenHelper = tokenHelper;
        this.userDetailsService = userDetailsService;
    }

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

        String username;
        String authToken = tokenHelper.getToken(request);

        if (authToken != null) {
            // get username from token
            username = tokenHelper.getUsernameFromToken(authToken);
            if (username != null) {
                // get user
                UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                if (tokenHelper.validateToken(authToken, userDetails)) {
                    // create authentication
                    TokenBasedAuthentication authentication = new TokenBasedAuthentication(userDetails);
                    authentication.setToken(authToken);
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            }
        }
        chain.doFilter(request, response);
```

TokenAuthenticationFilter (extends OncePerRequestFilter) pre-processes HTTP request, from Token, validate token ,create Authentication and populate it to SecurityContext.

Protect Resources with HTTPSecurity

To help Spring Security know when we want to require all users to be authenticated, which Exception Handler to be chosen, which filter and when we want it to work. We implement WebSecurityConfigurerAdapter and provide a configuration in the configure (HttpSecurity http) method.

```
protected void configure(HttpSecurity http) throws Exception {
    http
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .exceptionHandling().authenticationEntryPoint(restAuthenticationEntryPoint).and()
            .authorizeRequests()
            .antMatchers(
                    HttpMethod.GET,
                    "/",
                    "/auth/**",
                    "/webjars/**",
                    "/*.html",
                    "/favicon.ico",
                    "/**/*.html",
                    "/**/*.css",
                    "/**/*.js"
            ).permitAll()
            .antMatchers("/auth/**").permitAll()
            .antMatchers(HttpMethod.PUT , "/users/**").permitAll()
            .anyRequest().authenticated().and()
            .addFilterBefore(new TokenAuthenticationFilter(tokenHelper, jwtUserDetailsService), BasicAuthe

    http.csrf().disable();
}
```

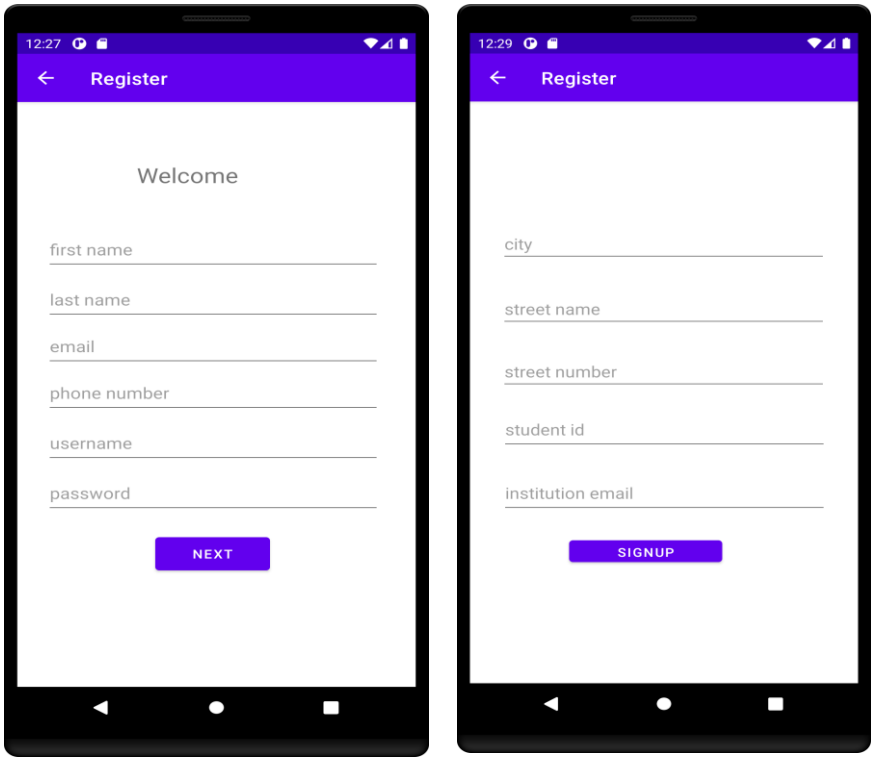
GET requests for Auth/login and PUT requests for users/update: accessed without any authentication (permit all).

But the other requests needs to authenticated(anyrequests.authenticated)

Front End Implementation of the project

Registration of users: clients provide identity information which contains access credentials like password, username… and other KYC data like address, phone number to know our customers.

Registration Page



API: users/save : this API doesn’t need authentication and authorization

Client: to communicate with spring boot from android client using retrofit.

```
public class client {
    private TokenManager tokenManager;

    private static Retrofit getRetrofit() {
        HttpLoggingInterceptor httpLoggingInterceptor = new HttpLoggingInterceptor();
        httpLoggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient okHttpClient = new OkHttpClient.Builder().addInterceptor(httpLoggingInterceptor)
           .connectTimeout( timeout: 5, TimeUnit.MINUTES).readTimeout( timeout: 5, TimeUnit.MINUTES).
                    writeTimeout( timeout: 5, TimeUnit.MINUTES).build();
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://160.97.244.16:8080/")
                .addConverterFactory(GsonConverterFactory.create())
                .client(okHttpClient)
                .build();
        return retrofit;
```

Request:

```
@POST("users/save")
Call<UserResponse> userregister(@Body BodyRegister register);
```

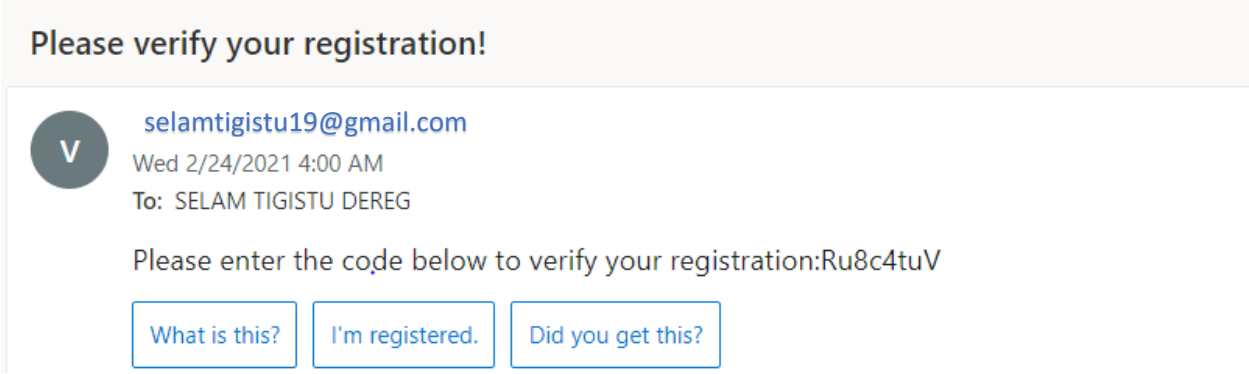
Service:

To save users information originating from the client request, in addition generate confirmation code to check whether email is student email or not.

```
@Override
public User addUser(User user) {
    String pass = passwordEncoder.encode(user.getPassword());
    String CHAR_LOWER = "abcdefghijklmnopqrstuvwxyz";
    String CHAR_UPPER = CHAR_LOWER.toUpperCase();
    String NUMBER = "0123456789";
    String DATA_FOR_RANDOM_STRING = CHAR_LOWER + CHAR_UPPER + NUMBER;

    int targetStringLength = 8;
    Random random = new Random();


    StringBuilder sb = new StringBuilder(targetStringLength);
    for (int i = 0; i < targetStringLength; i++) {
        int rndCharAt = random.nextInt(DATA_FOR_RANDOM_STRING.length());
        char rndChar = DATA_FOR_RANDOM_STRING.charAt(rndCharAt);
         sb.append(rndChar).toString();

    }

    user.setVerificationCode(sb.toString());
    user.setPassword(pass);

      return userRepository.save(user);
}
```

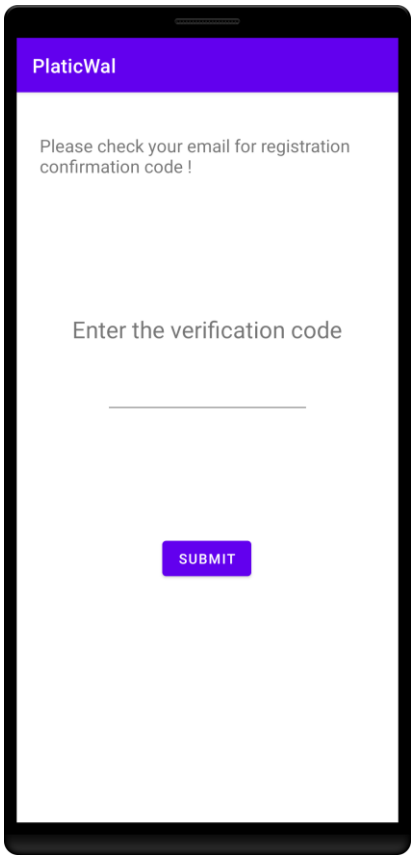
Response: returns full details of registered user.

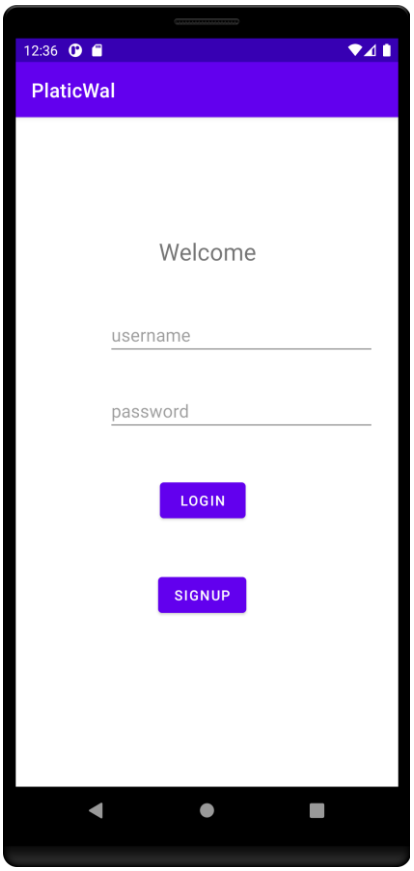
After Registration confirmation email send to institution email address with code.



Authenticate Users by username and password.

Login Page: user redirected to this page after successful confirmation of email.





API: /auth/login

Authenticate users by username and password then generate JWT token for subsequent requests.

Request

```
@POST("auth/login")
Call<Example> createPost(@Body UsernamePassReq user);
```

Spring security provides Authentication interface to authenticate, which accepts users credential username and password.

Authentication manager delegate username and password for specific authentication provider. This returns Authenticate object which contain the current user information, isauthenticate: True and so on**.**

The authenticate object hold by security context holder for avoiding reauthenticate same user.

```
public class JwtAuthenticationRequest {
    private String username;
    private String password;

    public JwtAuthenticationRequest() { super(); }

    public JwtAuthenticationRequest(String username, S
        this.setUsername(username);
        this.setPassword(password);
    }
```

```
@Override
public ResponseEntity<?> createAuthenticationToken(JwtAuthenticationRequest
    log.info("Create Jwt Token");
    final Authentication authentication = authenticationManager.authenticat
            new UsernamePasswordAuthenticationToken(authenticationRequest.g
                    authenticationRequest.getPassword()
            )
    );
    SecurityContextHolder.getContext().setAuthentication(authentication);
    // token creation
    User user = (User) authentication.getPrincipal();
    String jws = CommonUtil.BEARER + " " + tokenHelper.generateToken(user.g
    int expiresIn = tokenHelper.getExpiredIn();
    Map<String, Object> response = new HashMap<>(); //new
    response.put( "token",new UserTokenState(jws, expiresIn));
    response.put("user details",user);
    return  ResponseEntity.ok(response);//new UserTokenState(jws, expiresIn
}
```

It authenticates the user by comparing the password submitted in a UsernamePasswordAuthenticationToken against the one loaded by the UserDetailsService.

After successful authentication, it is possible to get `UserDetails` from `Authentication` object.

`UserDetailsService` for getting `UserDetails` object.

After successful Authentication JWT will be generated and returned to user

```
public String generateToken(String username) {
    return Jwts.builder()
            .setIssuer(APP_NAME)
            .setSubject(username)
              .setAudience("vdfbdvdfsf")
            .setIssuedAt(new Date())
            .setExpiration(generateExpirationDate())
            .signWith(SIGNATURE_ALGORITHM, SECRET)
            .compact();
}
```

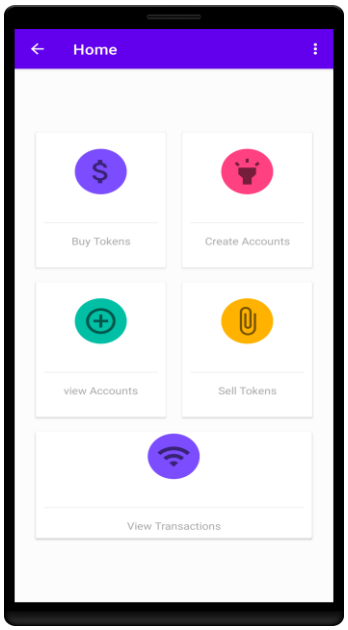
```
D/OkHttp: <-- 200 http://160.97.244.16:8080/auth/login (3183ms)
D/OkHttp: Content-Type: application/json
    Content-Length: 726
    Date: Wed, 24 Feb 2021 09:13:44 GMT
D/OkHttp: {"headers":{},"body":{"user details":{"id":85,"username":"sel","email":"drgsmt92m51z315e@studenti.unical.it","enabled":true,"studentid":
D/OkHttp: <-- END HTTP (726-byte body)
I/System.out: Bearer eyJhbGciOiJIUzUxMiJ9.eyJpc3MiOiJTcHJpbmcgSnd0LUJvaWxlcnBsYXRlIiwic3ViIjoic2VsIiwiaWF0IjoxNjE0MTU4MDI0LCJleHAiOjE2MTQxNTg2MjR9
I/System.out: ROLE_USER
```

The client save token on local storage to use it for authorized resource access using shared preference.

```
private Context context;
private SharedPreferences prefs;
private SharedPreferences.Editor editor;
private static final String REFNAME = "Bearer";
private static final String KEY_USER_NAME = "username";
private static final String KEY_JWT_TOKEN = "jwttoken";
private static final String AUTHORITY = "role";

private static TokenManager INSTANCE = null;
int Mode = 0;

public TokenManager(Context context) {
    this.context = context;
    prefs = context.getSharedPreferences(REFNAME, Mode);
    editor = prefs.edit();
}
```

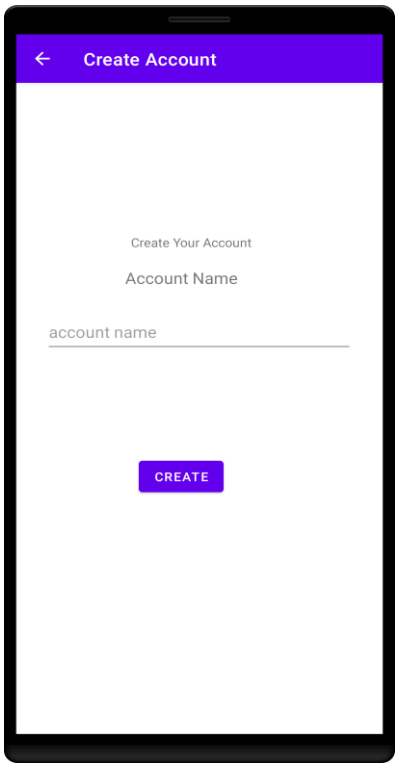
```
public void createSession(String username, String jwt,String role
    editor.putString(KEY_USER_NAME, username);
    editor.putString(KEY_JWT_TOKEN, jwt);
    editor.putString(AUTHORITY,role);
    editor.commit();
}
```

After successful Login Home page displayed.

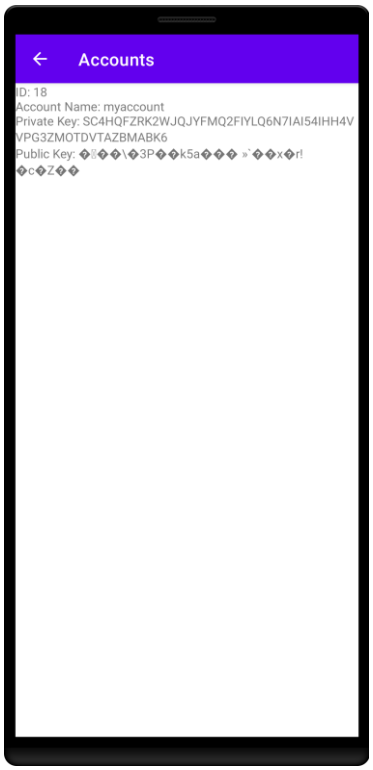


Now let us see sample request to access authorized resource: create Account and view accounts accessed by authorized users which provides valid tokens.

Create Account



View Account



API: accounts/save

```
@POST("accounts/save")
Call<AccountRegister> accountregister(@Header("Authorization") String authorization, @Body CreateAccount acc);
```

```
D/OkHttp: --> POST http://160.97.244.16:8080/accounts/save
D/OkHttp: Content-Type: application/json; charset=UTF-8
D/OkHttp: Content-Length: 44
D/OkHttp: Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJpc3MiOiJTcHJpbmcgSnd0LUJvaWxlcnBsYXRlIiwic3ViIjoic2VsIiwiaWF0IjoxNjE0MTU4MDI0LCJleHAiOjE2MTQ
```

API: accounts/username

```
@GET("accounts/{username}")
Call<Example_Acct> getAccountById(@Header("Authorization") String authorization,@Path("username") String username);
```

```
D/OkHttp: --> GET http://160.97.244.16:8080/accounts/sel
D/OkHttp: Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJpc3MiOiJTcHJpbmcgSnd0LUJvaWxlcnBsYXRlIiwic3ViIjoic2VsIiwiaWF0IjoxNjE0MTU4MDI0LCJleHAiOjE2MT
    --> END GET
```