

# **Spring Security (Part 1)**

## **Gold - Chapter 6 - Topic 1**

Selamat datang di **Chapter 6 Topic 1**  
online course **Back End Java** dari  
**Binar Academy!**



### Halloo, we meet again ☺

Binarian, gimana belajar kamu? masih challenging? atau perlahan udah makin jago?

Setelah belajar cara menerapkan Spring Web untuk membuat Restful API, sekarang kita lanjut **mengimplementasikan security dalam membuat authorization dan authentication.**

Untuk memudahkan pemahaman kita, gimana kalau dimulai dari konsep dasarnya dulu? Iya, kita bahas tentang **Spring Security (Part 1)**, dulu. Let's go!



**Dari sesi ini, kita bakal bahas hal-hal berikut:**

- Konsep Autoauthentication dan Authorization
- Penerapan best practice dari Identity Database Scheme
- Cara melakukan Spring Security configuration
- Spring Security configuration - Create Login Page
- Spring Security configuration - Create Login API
- Spring Security configuration - Session
- Spring Security configuration - Bearer Token
- 



Binarian, kamu sadar nggak sih kalau sebenarnya kita udah sering banget nemuin konsep **authentication** ini?

Clue-nya adalah satpam. Nggak percaya? kita coba buktiin bareng-bareng ya abis slide ini~



**Sebelumnya, kita udah belajar implementasi RESTful API dan MVP. Tapi, kita belum banyak perhatian sama konsep keamanannya, nih~**

Padahal, “konsep keamanan” ini penting banget supaya resource atau database di sistem kita hanya bisa diakses oleh pihak tertentu.

Dengan tujuan, pihak tersebut bisa melakukan hal-hal yang udah diizinkan oleh sistem kita aja.



Konsep keamanan ini diimplementasikan dalam bentuk **authentication** dan **authorization**.

Walaupun sekilas nama kayaknya mirip, tapi keduanya berbeda. Kamu penasaran nggak bedanya dimana?



### Kita bisa bayangkan authentication itu ibarat seorang satpam di sekolah~

Misalnya, siswa SMA Binar hanya boleh masuk ke dalam sekolah kalau mereka pakai seragam dan mengenakan ID Card di saku bajunya.

Maksud aturan ini yaitu supaya satpam bisa membedakan mana siswa asli dan siswa yang menyamar masuk ke dalam sekolah.

Nah, peran satpam ini sama kayak gambaran dari proses authentication.

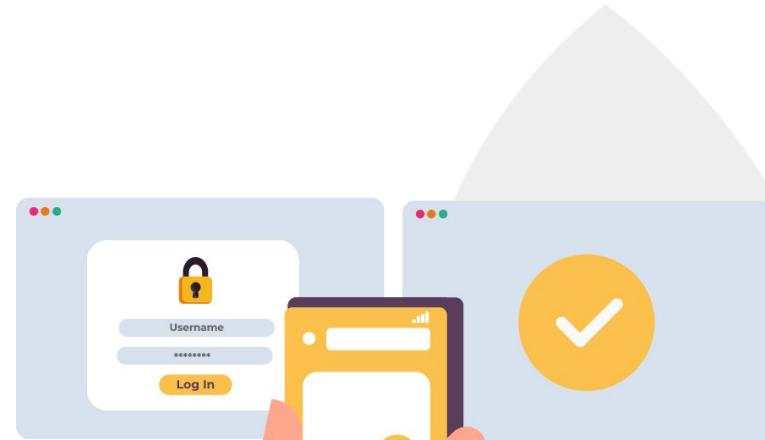


**Ternyata authentication sering kita temui lho~**

Authentication biasanya diterapkan misalnya pada penggunaan form login.

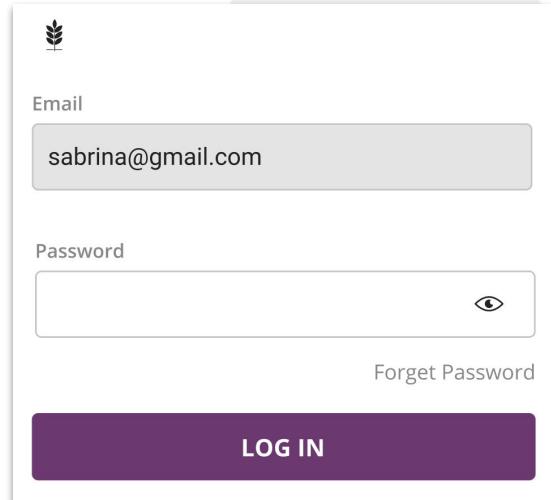
“Terus, sebenarnya apa sih yang terjadi ketika proses login itu? Kenapa masuk ke sistem authentication?”

Buat tahu jawabannya, kita next slide, yuk!



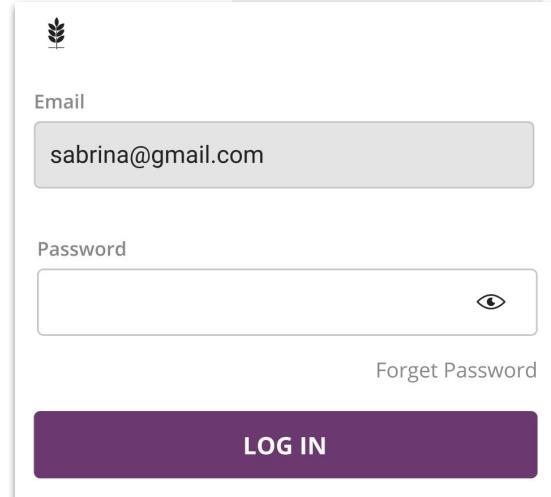
Lebih detail tentang proses login kayak gini, nih~

1. Pengguna memasukan data username dan password.
2. Data dari form diterima oleh sistem, dan sistem yang baik melakukan enkripsi data terhadap password yang dimasukkan oleh pengguna.



A screenshot of a login form. At the top is a small purple icon of three leaves. Below it is an "Email" field containing "sabrina@gmail.com". Below that is a "Password" field with an "eye" icon to its right. To the right of the password field is a "Forget Password" link. At the bottom is a large purple "LOG IN" button.

3. Data username dan password (yang udah ter-enkripsi) kemudian dicocokkan dengan data yang ada di database.
4. Kalau datanya cocok, berarti proses authentication udah oke!
5. Kalau nggak cocok sama data yang ada di database, maka sistem bakal menolak.



**“Emangnya gimana sih cara sistem melakukan konfirmasi data yang ada di form login?”**

Dalam konteks data user, hal tersebut bisa kita temukan di atribut-atribut pembangun data user itu sendiri, sob.

Kita bisa lihat contohnya dari tabel user yang sering digunakan di suatu sistem kayak yang ada di samping ini~

Users
id
username
password
auth_key
status

### Oke! sekarang bisa kita simpulkan nih~

Kesimpulannya, authentication adalah tindakan mengonfirmasi kebenaran suatu bagian dari data (datum atau data tunggal) atau suatu entity yang biasanya di dalamnya ada user authentication.

**User authentication ini berarti melakukan konfirmasi data user yang sebelumnya udah tersimpan di sistem.**



Kalau tadi kita udah bahas tentang Authentication, sekarang kita geser ke saudaranya nih, yaitu **Authorization**.

Kenapa dibilang saudara? Karena mereka sebenarnya berhubungan dan mirip-mirip gituu~

Nggak percaya? Ayo kita cek!



### Kali ini kita bakal kenalan sama yang namanya Authorization~

Authorization adalah proses menentukan hal yang bisa dilakukan pengguna yang udah jelas identitasnya (authenticated user).

Jadi, **sebelum adanya authorization, user harus melalui proses authentication terlebih dahulu.**

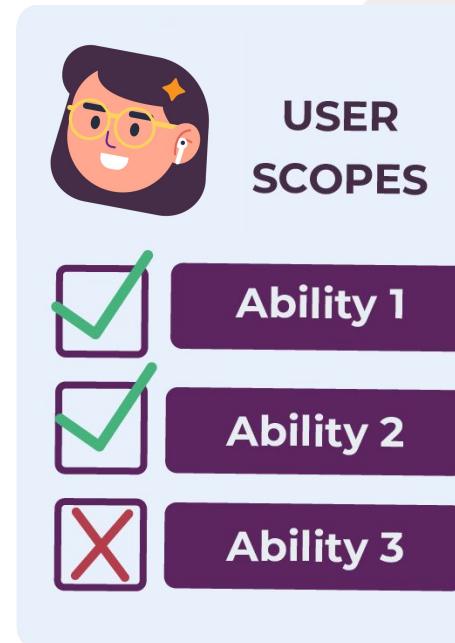
Yep, keduanya merupakan sebuah step berkelanjutan.



Identitas yang udah dibuktikan pada proses authentication tadi lalu menjadi dasar untuk menentukan privilege user berupa **hak akses segala resource yang ada**.

Biasanya nih, cara menentukan privilege dalam mengakses resource yaitu melalui pengelompokan user dengan role berbeda-beda.

Masing-masing role pastinya punya jumlah privilege yang berbeda.



### Kita coba pakai analogi satpam lagi, ya!

Kalau ada orang yang mau masuk ke ruangan tertentu, pasti ada seorang satpam yang bakal mengecek gini:

apakah orang itu diperbolehkan masuk ke suatu ruangan? apakah berkepentingan melakukan sesuatu di dalamnya? atau enggak?



Misalnya aturannya gini, cuma staff IT aja yang boleh ada di dalam ruang server sekolah.

Kalau ada guru mau masuk ruangan IT, masih boleh. Dengan catatan nggak diizinkan untuk memegang perangkat server.

Beda lagi nih, sama siswa. Tentu aja siswa dilarang masuk ruang IT.



Analogi yang tadi itu adalah gambaran proses yang dilakukan untuk authorization.

Sekarang kamu udah ada gambaran tentang authorization dan perbedaan antara authentication dan authorization, kan?



Selain Authentication dan Authorization yang saling berkelanjutan, ada lagi yang perlu kita tahu.

Yaitu, **Identify Database Scheme**.



**Sebelum jauh ke Identify Database Scheme, kita harus kenalan dulu sama beberapa pattern yang biasa dipakai dalam schema database~**

Pada scheme ini ternyata ada beberapa pattern yang biasa dipakai. Dengan pattern, kita bisa pakai solusi yang udah ada. Pattern yang paling sering dipakai antara lain:

1. **Role Based Access Control**
2. **Access Control List**
3. **Attribute Based Access Control**



Penjelasan lebih lengkap tentang pattern schema database dijelaskan [di sini](#), ya.

Khusus di course ini, kita bakal pakai **Role Based Access Control (RBAC)** karena pattern ini merupakan pattern yang paling sederhana.

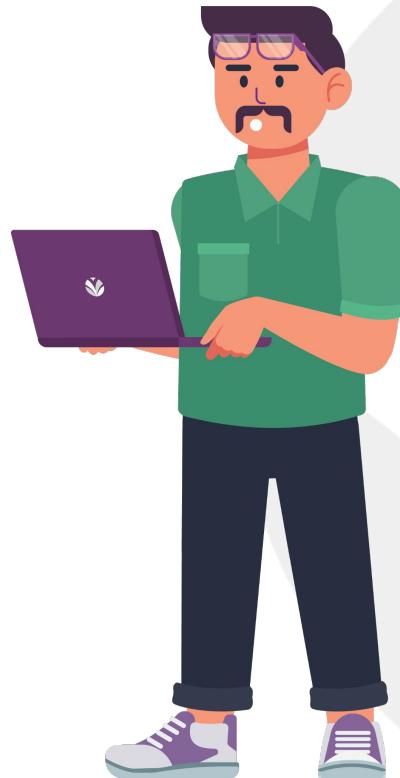
Penerapan schema dari database RBAC bisa kamu simak [di sini](#)~



### Dalam menerapkan RBAC, kita harus menerapkan modelnya juga nih, sob!

Yap. Untuk menerapkan RBAC pada Spring Security yang terintegrasi dengan database, kita wajib menerapkan model berikut pada database:

- **Tabel User**, dipakai untuk menyimpan informasi username dan password, serta role yang dimiliki.
- **Table Role**, dipakai untuk menyimpan informasi role.



Di bawah ini merupakan contoh model dari role~

```
public enum ERole {
    ROLE_CUSTOMER, ROLE_ADMIN
}

import javax.persistence.*;
@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;
    public Role() {
    }
    //getter setter
}
```

Kalau ini adalah contoh model dari user~

```
● ● ●

@Entity
@Table( name = "users",
        uniqueConstraints = {
            @UniqueConstraint(columnNames = "username"),
            @UniqueConstraint(columnNames = "email")
        })
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank
    @Size(max = 20)
    private String username;
    @NotBlank
    @Size(max = 50)
    @Email
    private String email;
    @NotBlank
    @Size(max = 120)
    private String password;
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable( name = "user_roles",
                joinColumns = @JoinColumn(name = "user_id"),
                inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();
    public User() {
    }
}
```

Untuk menerapkan Spring Security yang terintegrasi dengan database, kita perlu schema kayak di bawah ini. Coba perhatikan baik-baik, ya!

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null
);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fkAuthorities_users foreign key(username) references users(username)
);
```

Kalau untuk authentication yang menggunakan token, database harus punya table, sob.

Jangan sampai lupa, ya!

```
● ● ●  
create table persistent_logins (  
    username varchar(64) not null,  
    series varchar(64) primary key,  
    token varchar(64) not null,  
    last_used timestamp not null  
);
```

# LANJUTTT

Lanjuttt!

Udah bahas konsep-konsep dasarnya, sekarang kita masuk ke hal yang lainnya yaitu Spring Security Configuration.

Ready?



### Spring Security Configuration itu apa?

Jadi, untuk mendukung authentication dan authorization, Spring telah menyediakan **module Spring Security**.

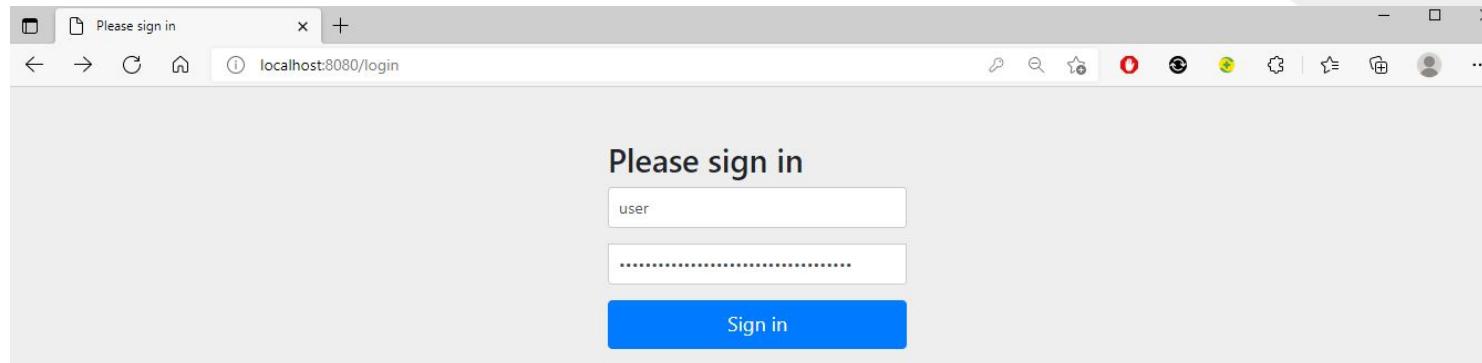
Dokumentasi Spring Security secara lengkap bisa kamu pelajari [di sini](#).

Untuk menambahkan modul ini, kita bisa menambahkan dependency di samping pada file pom.xml.

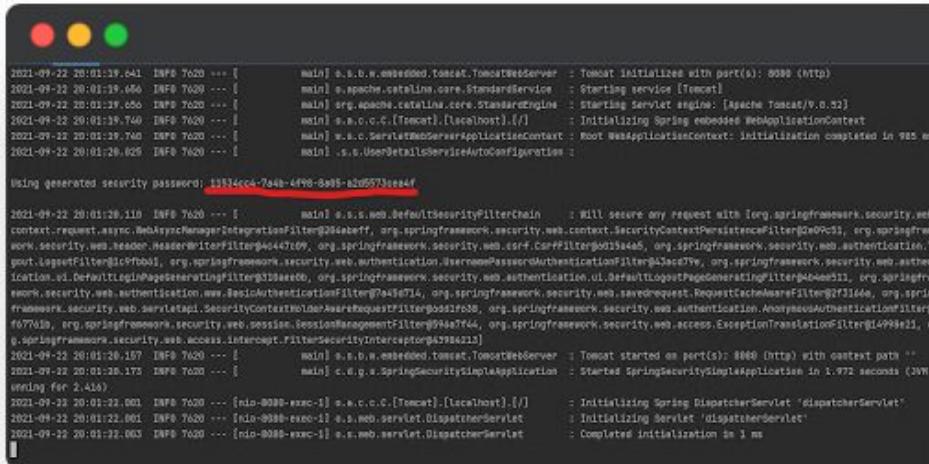


```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Setelah menambahkan dependency, maka kita nggak bakal bisa melakukan request secara langsung. Melainkan kita bakal diminta buat login dulu.



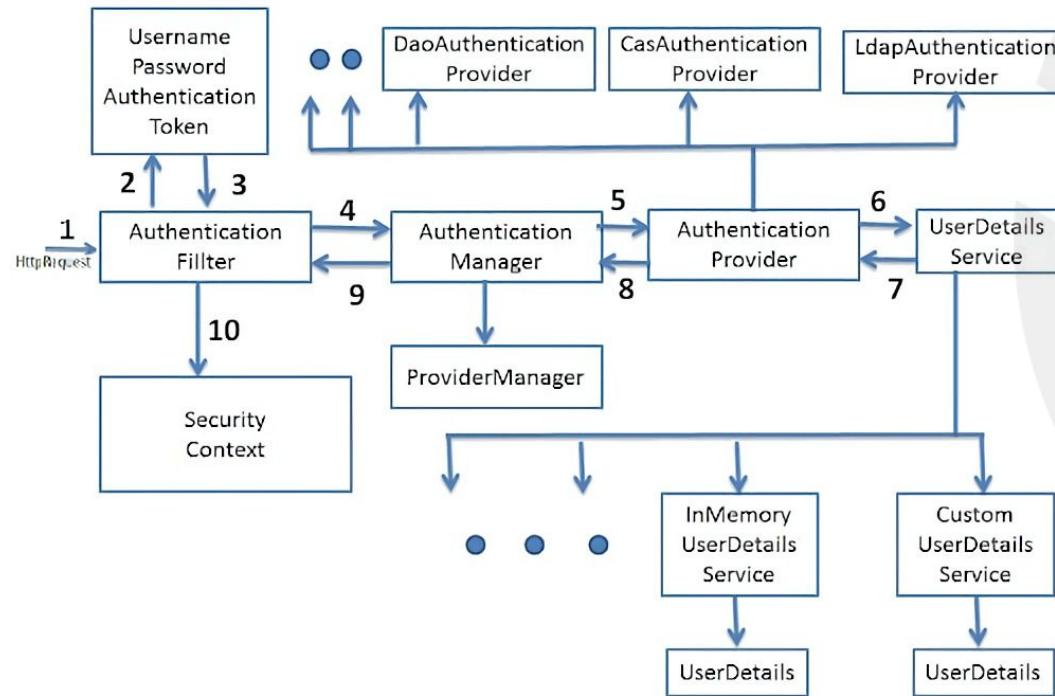
Untuk melewati authentication ini, kita pakai user default dan juga password yang ada di console dari Spring Security. Eits, jangan lupa kalau **password default ini berubah-ubah setiap aplikasi di-start**, ya.



```
2021-09-22 20:01:19.641 INFO 7620 --- [main] o.s.o.w.embedded.tomcat.TomcatWebServer : Tomcat initializes with port(s): 8080 (http)
2021-09-22 20:01:19.656 INFO 7620 --- [main] o.apache.catalina.core.StandardService : Starting service [tomcat]
2021-09-22 20:01:19.656 INFO 7620 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-09-22 20:01:19.740 INFO 7620 --- [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2021-09-22 20:01:19.740 INFO 7620 --- [main] w.s.c.WebApplicationContext : Root WebApplicationContext: initialization completed in 925 ms
2021-09-22 20:01:26.025 INFO 7620 --- [main] s.s.UserDetailsServiceAutoConfiguration : Using generated security password: 13334cc4-7a4b-4f98-8985-e205573eaa4f

2021-09-22 20:01:26.110 INFO 7620 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@284eabff, org.springframework.security.web.context.SecurityContextPersistenceFilter@269c931, org.springframework.security.web.header.HeaderWriterFilter@a4471c09, org.springframework.security.web.csrf.CsrfFilter@b0194a45, org.springframework.security.web.authentication.logout.LogoutFilter@1cfbbd1, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@3a3cc7fe, org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@30aeed0b, org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@4b4e6d11, org.springframework.security.web.authentication.www.BasicAuthenticationFilter@3a47d714, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@73144e, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestWrapperFilter@ad93208, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@f077030, org.springframework.security.web.session.SessionManagementFilter@9447f44, org.springframework.security.web.access.ExceptionTranslationFilter@14998e21, org.springframework.security.web.access.intercept.FilterSecurityInterceptor@4798a212]
2021-09-22 20:01:26.157 INFO 7620 --- [main] o.s.o.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-22 20:01:26.173 INFO 7620 --- [main] c.d.g.s.SpringSecuritySimpleApplication : Started SpringSecuritySimpleApplication in 1.972 seconds (JVM running for 2.416)
2021-09-22 20:01:22.001 INFO 7620 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-22 20:01:22.001 INFO 7620 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-09-22 20:01:22.003 INFO 7620 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

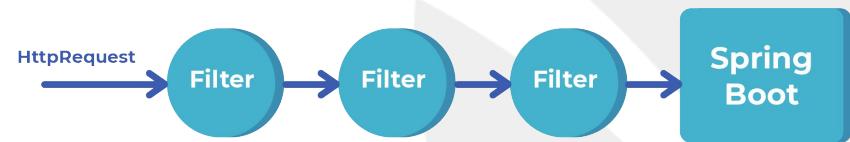
## Berikut gambaran besar dari Spring Security~



### Secara umum, prosesnya kayak gini bestie~

Sebelum request masuk ke dalam servlet, request akan di-intercept oleh filter-filter.

Pada gambaran besar tadi, proses ini merupakan **AuthenticationFilter**.



Ketika request di-intercept, maka credential pada request bakal di-extract untuk mendapatkan user dan password. Baru deh setelah itu, object Authentication bakalan dibuat.

Object UsernamePasswordAuthenticationToken bakal dibikin kalau credential yang di-extract merupakan username dan password.

Oh iya, buat yang tanya tentang **Credential**, ini artinya informasi yang dibutuhkan dalam melakukan authentication, ya. Misalnya kayak password.

**Authentication**  
**<<Interface>>**

**implements**

**UsernamePasswordAuthentication  
Token**

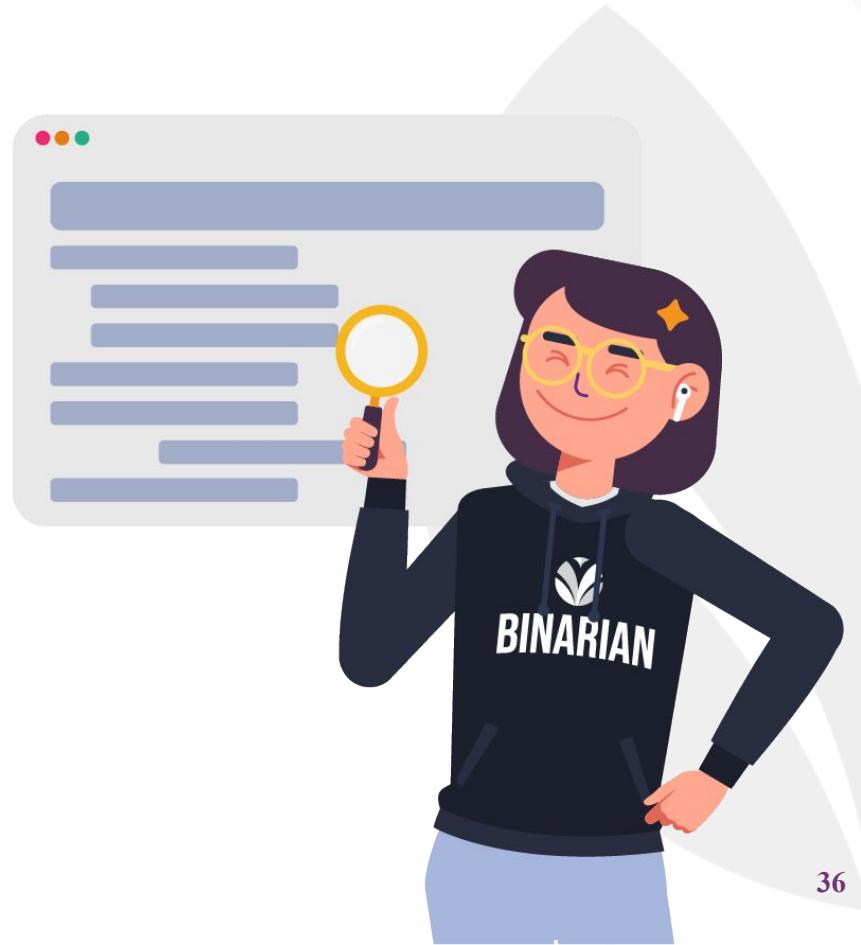
### Ternyata Security filter juga punya beberapa tugas, lho!

Sebuah Security filter punya minimal empat tugas nih, yaitu:

1. Filter bakal meng-extract username dan password dari request.
2. Filter harus mem-validate username dan password apakah sesuai atau nggak.



3. Filter harus mengecek apakah user tersebut authorized buat mengakses URI tersebut atau nggak.
4. Meneruskan request kalau udah berhasil.



Berikut adalah isi dari  
SecurityServletFilter!

```
import javax.servlet.*;
import javax.servlet.http.HttpFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class SecurityServletFilter extends HttpFilter {

    @Override
    protected void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {
        UsernamePasswordToken token = extractUsernameAndPasswordFrom(request);
        if (notAuthenticated(token)) { // either no or wrong username/password
            // unfortunately the HTTP status code is called "unauthorized", instead of
            "unauthenticated"
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED); // HTTP 401.
            return;
        }
        if (notAuthorized(token, request)) {
            // you are logged in but don't have the proper rights
            response.setStatus(HttpServletResponse.SC_FORBIDDEN); // HTTP 403
            return;
        }
        // allow the HttpServletRequest to go to Spring's DispatcherServlet
        // and @RestControllerControllers@Controllers.
        chain.doFilter(request, response); // (4)
    }

    private UsernamePasswordToken extractUsernameAndPasswordFrom(HttpServletRequest request) {
        // Either try and read in a Basic Auth HTTP Header, which comes in the form of user:password
        // Or try and find form login request parameters or POST bodies, i.e. "username=me" &
        "password=myPass"
        return checkVariousLoginOptions(request);
    }

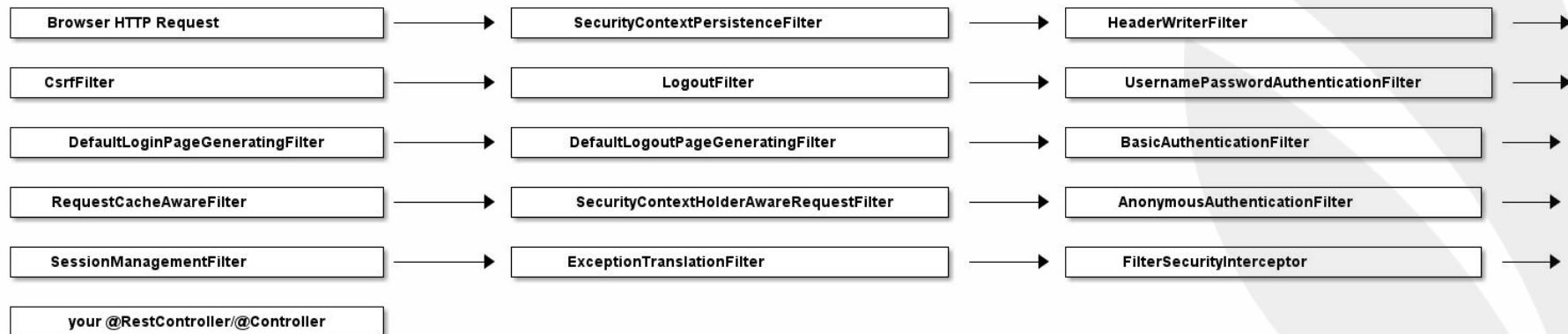
    private boolean notAuthenticated(UsernamePasswordToken token) {
        // compare the token with what you have in your database...or in-memory...or in LDAP...
        return false;
    }

    private boolean notAuthorized(UsernamePasswordToken token, HttpServletRequest request) {
        // check if currently authenticated user has the permission/role to access this request's /URI
        // e.g. /admin needs a ROLE_ADMIN , /callcenter needs ROLE_CALLCENTER, etc.
        return false;
    }
}
```

### Tapi...

Pas implementasinya, ada banyak filter yang bakal dilalui.

Berikut adalah Filter yang harus dilalui dari sebuah request~



Kalau kita pakai object Authentication nih, filter bakal manggil si method authenticate punya interface AuthenticationManager. Implementasi dari interface tersebut udah disediakan oleh ProviderManager.

**Method authenticate ini memakai object Authentication dan bakal mengembalikan object Authentication lagi kalau proses authentication berhasil.**



Berikut isi dari object Authentication sebelum dan sesudah melewati method authenticate!

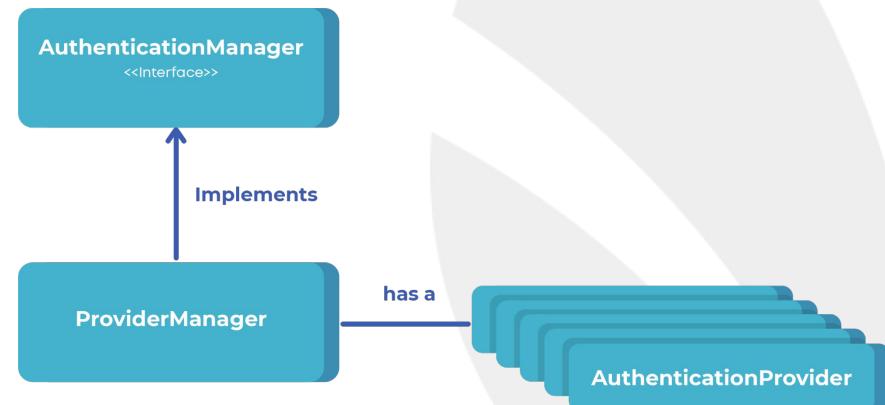
- Principal artinya informasi detail user ketika user udah authenticated.
- Granted Authorities artinya role yang dimiliki sama user.

Field	Authentication User Request before Authentication	Authentication User Request After Authentication
Principal	Username	User Object
Granted Authorities	Not granted any authorities	ROLE_ADMIN
Authenticated	False	True

### ProviderManager punya list AuthenticationProvider, lho~

Provider manager inilah yang bakal bantu dalam memilih Authentication provider yang tepat.

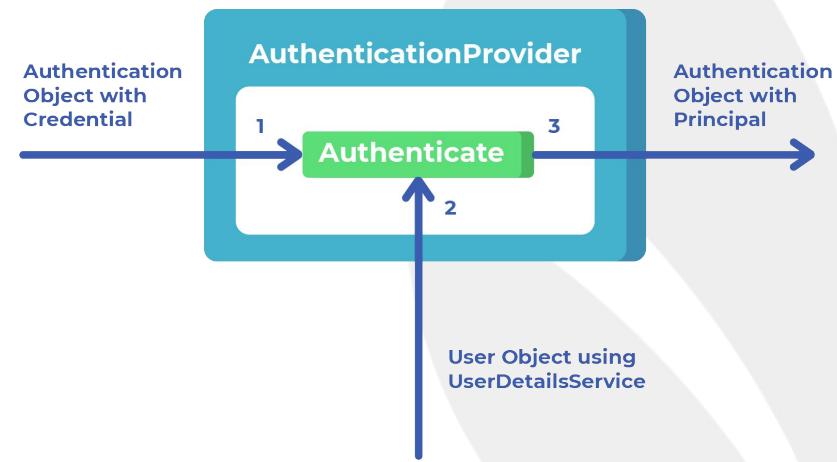
Keren banget, kan?!



**Kalau pakai UserDetailsService,  
AuthenticationProvider bakal  
mengembalikan user object secara lengkap!**

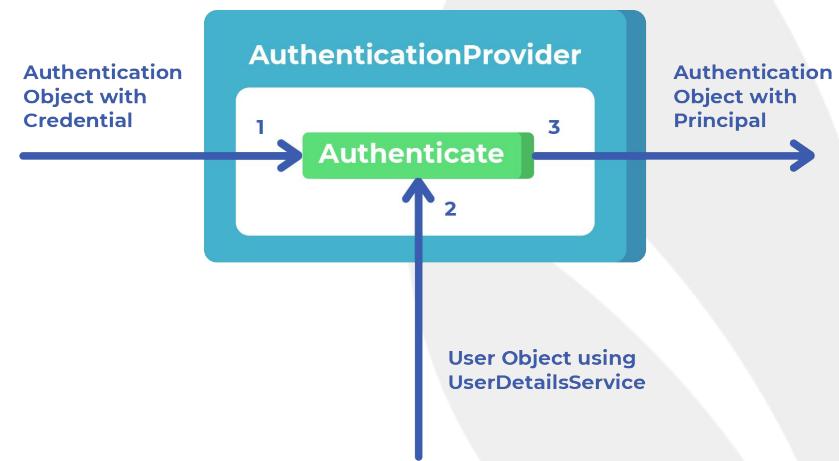
“UserDetailsService? Apaan tuh?”

Jadi, UserDetailsService merupakan sebuah interface yang punya satu method yaitu loadUserByUsername.



User Object yang berasal dari db atau cache bakal dibandingkan dengan yang ada pada authentication object.

Kalau object Authentication sesuai sama principal, maka bakal dikembalikan sebagai response.



**“Kalau gitu, gimana caranya supaya bisa menerapkan Spring Security?”**

Pertama, kita harus bikin class yang berisi konfigurasi dulu. Class tersebut pakai annotation **@Configuration** dan **@EnableWebSecurity**.

Eitss, nggak cuma itu aja. Disini si Class juga harus meng-extend WebSecurityConfigurerAdapter.

Untuk melakukan konfigurasi dasar, class tersebut harus meng-override **method configure (HttpSecurity)**.



### HttpSecurity merupakan object yang memiliki peranan sangat penting~

Banyak banget method yang dipakai untuk melakukan konfigurasi dengan object ini. Berikut beberapa method utamanya:

- **authorizeRequests()**,

dibutuhkan supaya bisa menentukan URL yang memerlukan authorization. Contohnya bisa kamu cek di samping, ya!



```
http.authorizeRequests()
    .antMatchers(PUBLIC_URL).permitAll()
    .antMatchers("/admin/**").hasRole("ADMIN")
    .anyRequest().authenticated();
```

- **antMatchers**,

memuat atau mencocokkan URL ataupun pattern URI buat diproses lebih lanjut.

- **permitAll()**,

dipakai untuk memperbolehkan URL yang udah didefinisikan sama antMatchers.



```
http.authorizeRequests()
    .antMatchers(PUBLIC_URL).permitAll()
    .antMatchers("/admin/**").hasRole("ADMIN")
    .anyRequest().authenticated();
```

- **hasRole(String),**

dipakai untuk meng-authorisasi role yang dapat mengakses URL dan udah didefinisikan sama antMatchers.

- **anyRequest(),**

dipakai untuk semua URI selain yang disebutkan.

- **Authenticated(),**

artinya URI harus diakses sama authentication terlebih dahulu.



```
http.authorizeRequests()
    .antMatchers(PUBLIC_URL).permitAll()
    .antMatchers("/admin/**").hasRole("ADMIN")
    .anyRequest().authenticated();
```

**formLogin()** dipakai buat menentukan login page. Berikut contohnya:

```
formLogin()  
    .loginPage("/login").permitAll();
```

Kalau **logout()** dipakai buat menentukan URI dari logout. Berikut contohnya:

```
.logout()  
    .logoutUrl("/api/logout")  
    .logoutSuccessHandler.ajaxLogoutSuccessHandler()  
    .deleteCookies("JSESSIONID")  
    .permitAll()
```

Buat melakukan chaining antara method yang dimiliki HttpSecurity, kita bisa pakai method **and()**

### Berikut contoh dari class yang berisi security config!

```
@Configuration  
@EnableWebSecurity  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http  
            .authorizeRequests()  
                .antMatchers("/", "/home").permitAll()  
                .anyRequest().authenticated()  
                .and()  
            .formLogin() // (5)  
                .loginPage("/login")  
                .permitAll()  
                .and()  
            .logout() // (6)  
                .permitAll()  
                .and()  
            .httpBasic(); // (7)  
    }  
}
```

Well..

Kamu masih semangat buat materi selanjutnya,  
kan?

Tenang ajaaa~ materinya sederhana kok, kita  
cuma bakal melakukan **Create Login Page**.



### Sekarang kita coba bikin Login Page, yuk!

Karena menggunakan servlet, Spring security nggak hanya dipakai untuk memberikan security pada REST API aja, tapi juga di web application.

Berikut referensi buat bikin Login Page secara custom, pakai Spring security, Thymeleaf dan Bootstrap.

[Spring Security Custom](#)



Kalau tadi bahas yang create login page. Yang ini, kita bakal coba **Create Login API**.

Pasti kamu penasaran, kan?

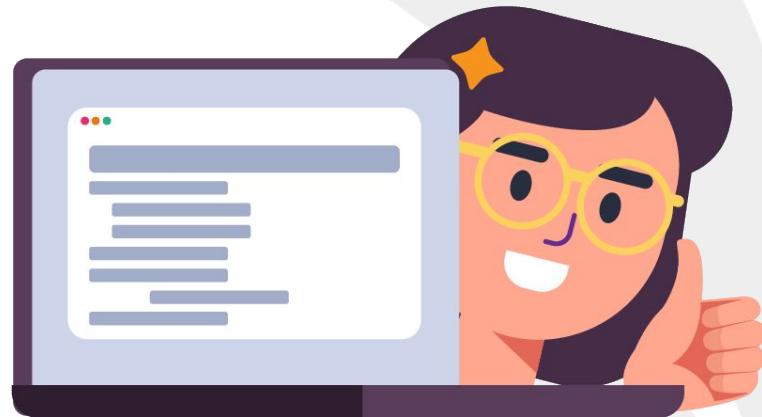
Yuk yuk, kita langsung ajaaa~



**Buat bikin login API yang dipakai Spring security, ada beberapa hal yang harus diperhatikan nih, sob~**

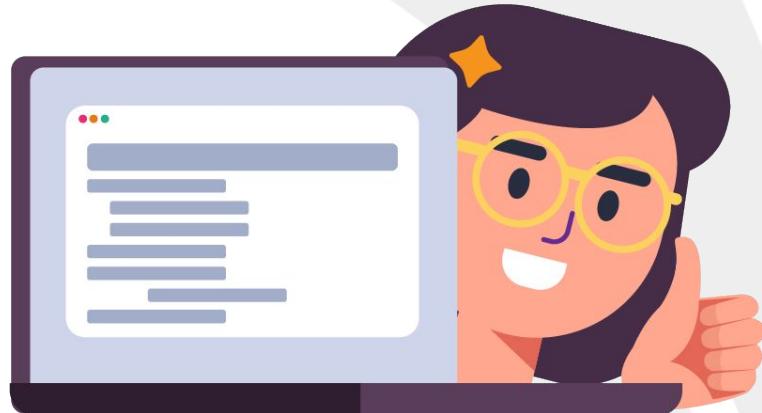
Coba simak di bawah ini, ya!

1. Request body terdiri dari username dan password.
2. Pakai AuthenticationManager untuk melakukan authentication.
3. Pakai SecurityContextHolder untuk mengubah status jadi authenticated.



4. Pakai token generator kalau kamu menggunakan token.
5. Memberikan response setelah login sukses ataupun gagal.

Nah, untuk yang implementasi code dari Login API bakal kita bahas di topic 2, ya!



**Kamu tahu nggak? Ternyata ada metode lain untuk melakukan authentication, lho, namanya Session!**

**Kira-kira session itu apa, ya?**

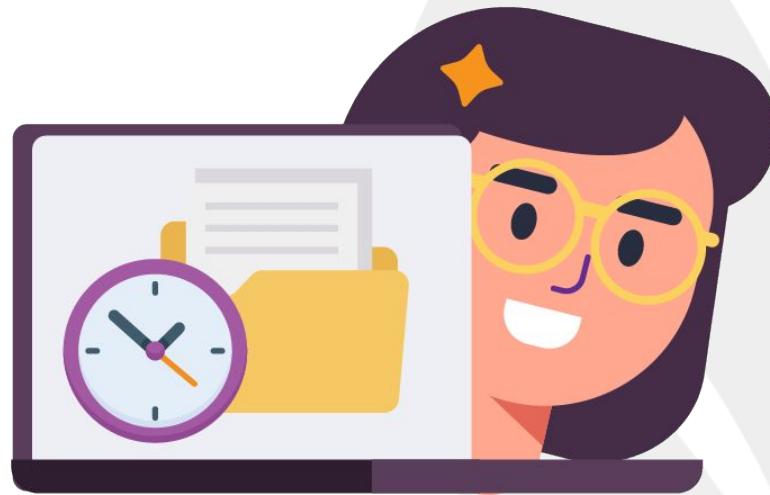
**Beli es naik angkot. Let's cekidottt~**



### Salah satu metode authentication adalah dengan menggunakan session~

Session merupakan sebuah file kecil yang isinya informasi tentang user, seperti id, waktu login dan waktu session akan expired.

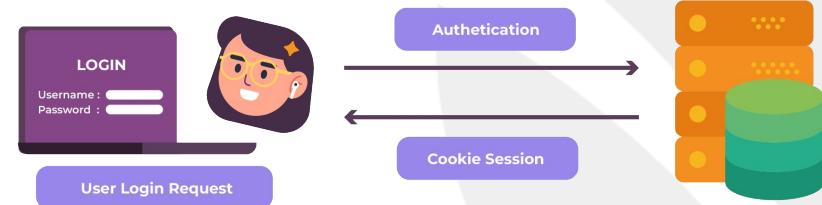
User bakal menerima informasi ini terutama id. Dan id bakal di-update setiap kali user melakukan request baru.



### “Terus, cara kerjanya gimana?”

Gini, gini~

1. User melakukan login request.
2. Server bakal melakukan authentication. kalau cocok, maka session bakal dikirim ke dalam database, kemudian mengembalikan cookie yang isinya session ID dari user.
3. Terus, server juga bakal mengecek session ID dari setiap request yang dikirimkan. Kalau ada session ID yang sesuai, maka user bisa mengakses resource yang ada.



Oh iya. **Metode authentication yang pakai session biasanya digunakan di web application**, gengs.

Metode ini cukup kompleks untuk diterapkan pakai Spring security.

Untuk membantu belajar kamu dalam menerapkan Authentication pakai session ini, kamu bisa coba pelajari referensi [di sini](#), yaaaa.



Terakhir bin akhiron nih, gengs.

Sebagai penutup di topic pertama ini, kita bakal bahas tentang **Bearer Token**.

Eits, ini bukan token yang buat isi pulsa listrik lho, ya.



### Metode authentication lainnya adalah authentication pakai token~

Authentication ini disebut juga dengan bearer authentication atau token authentication.

Cara kerja bearer authentication adalah siapapun pemegang token dari hasil authentication, maka ia bakal mendapatkan privilege dari authenticated user.



### “Kenapa harus pakai Bearer Token?”

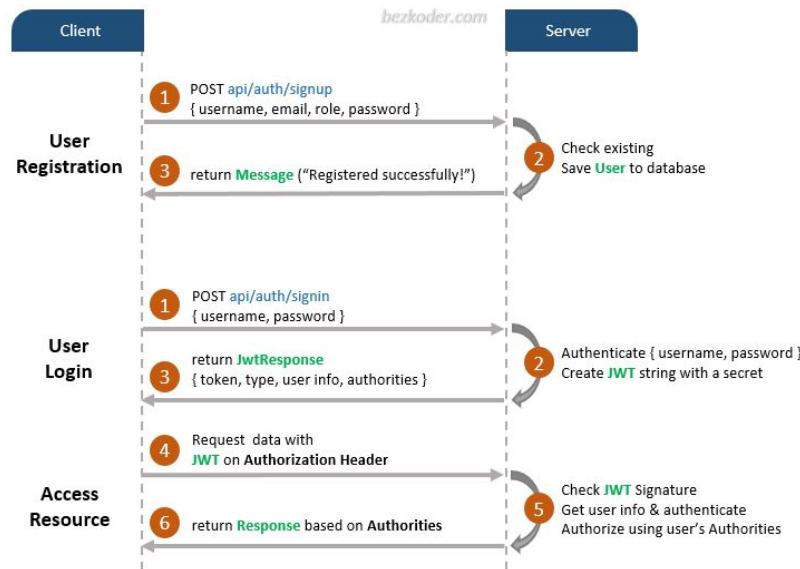
Karena Bearer Token ini punya kelebihan yaitu nggak perlu library yang kompleks buat didevelop.

Bearer token biasanya dipakai di HTTP Header dengan key Authorization.

Format value dari bearer token adalah sebagai berikut:

**Bearer <Your Token Value>**



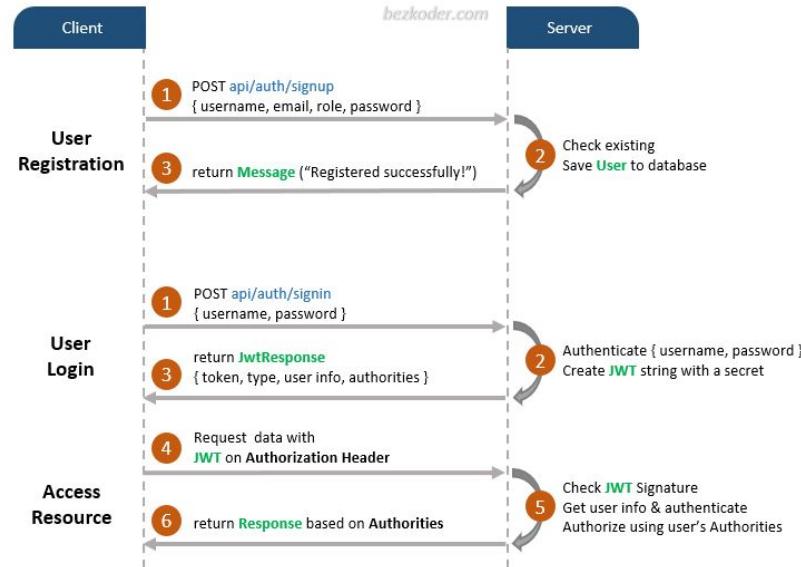


**Authentication ini biasanya dipakai untuk mengakses open API, gengs~**

Authentication ini pertama kali dipakai di OAuth2.

Kalau token nggak sesuai atau nggak ada token sedangkan API tersebut butuh token, maka response HTTP status yang dikembalikan adalah **401 Unauthorized**.

Coba cek gambar di samping, ya.



Salah satu token yang jadi standar dan sering dipakai di Bearer Authentication ini adalah **JWT atau Json Web Token**.

Kelebihan authentication yang pakai token dibandingkan session adalah penggunaan yang lebih luas, jadi bukan cuma di web application aja.

Canggih banget, kan?

**Biar semakin paham, yuk ikuti exercise hari ini~**

Sebelum lanjut ke topic selanjutnya, ada latihan yang perlu kamu kerjakan

**Silakan kamu buat login page, login API, Session dan Bearer Token.**

Setelah kamu kerjakan, nanti diskusikan jawabannya bersama teman sekelas dan facilitator ya!



Gimana gengs mengenai impresi kamu terhadap si **Spring Security** ini?

By the way, dari ketiga cara melakukan login Page secara custom (menggunakan Spring security, Thymeleaf atau Bootstrap), mana menurutmu yang paling nyaman kamu gunakan dan kenapa?



Nah, selesai sudah pembahasan kita di Chapter 6 Topic 1 ini.

Selanjutnya, kita masih di Spring Security yang khusus ngomongin cara menyamarkan informasi penting ☐

Penasaran kayak gimana? Yuk langsung ke topik selanjutnya~

