BINAR

Spring Security (Part 2)

Gold - Chapter 6 - Topic 2

BINAR

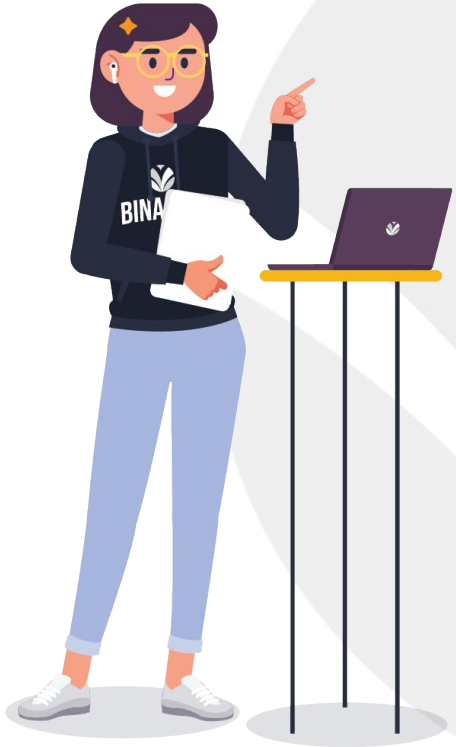
Selamat datang di Chapter 6 Topic 2 online course Back End Java dari Binar Academy!

Haii haii, Binarian

Ketemu lagi sama Sabrina, pada topik sebelumnya, kita udah belajar tentang Spring Security. Iya, yang ada konsep satpamnya itu.

Masih di pembahasan Spring Security yang sama, versi kedua ini bakal mengelaborasi tentang **implementasi JWT, perbedaan antara encryption dan decryption sampai dengan konsep SHA1, OAuth2 dan API Key.**

Yuk langsung aja kita kepoin~



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

- Implementasi JWT
- Konsep serta perbedaan Encryption dan Decryption
- Pengantar SHA1
- Jenis Grant type dalam Oauth2
- Konsep API Key

Binarian, diawal tadi kita sempet mention tentang JWT yaitu Json Web Token.

Dilihat dari kepanjangannya, udah ketebak dikit ya konsepnya bakal seperti apa.

Yuk, kita coba PDKT sama JWT!

“JWT? Apaan tuh?”

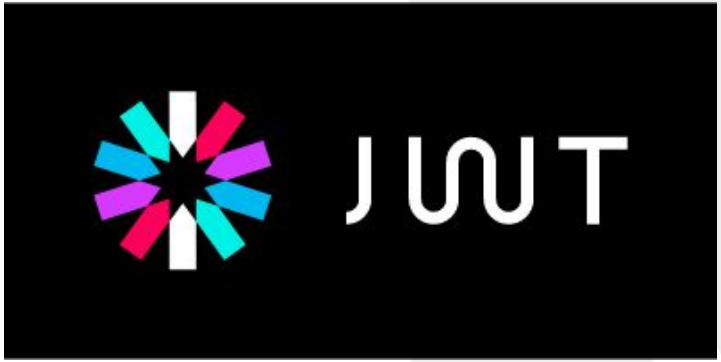
JWT atau Json Web Token merupakan sebuah token yang memanfaatkan JSON untuk mengirimkan data secara compact dan secured, serta bisa diverifikasi oleh dua pihak.

Keuntungan dari sebuah JWT adalah data dimuat secara compact, gengs.

Bisa dibilang kalau JWT merupakan **salah satu jenis token yang sering dipakai dalam melakukan bearer authentication**.

Cara kerjanya simple banget. Token ini dipakai sebagai tanda pengenal yang disimpan di dalam local storage.

Dokumentasi dari JWT bisa kamu pelajari di sini, yaaa~



BINAR

Berikut adalah contoh perbandingan JWT dengan SAML (Security Assertion Markup Language) token~

Kalau diperhatikan, JWT punya data yang jauh lebih compact, lho.

JWT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E
2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) □secret base64 encoded
```

VS

SAML

SAML ENCODED PASTE A TOKEN HERE

```
PHNhbWxwOlJlc3BvbnNlIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6b
mFtZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIElEPSJfNjIxYzRjNGFlN
WQ2MGM3NjhjYzliICBWZXJzaW9uPSIyLjAiIEIzc3VlSW5zdGFudD0iMj
AxNC0xMC0xNFQxNDozMjoxNToiICBEZXN0aW5hdGlvbj0iaHR0cHM
6Ly9hcHAuYXV0aDAuY29tL3Rlc3Rlci9zYW1scCI+PHNhbWw6SXNzd
WVyIHhtbG5zOnNhbWw9InVybjpvYXNpcpzuYW1lczp0YzpTQU1MOjI
MDphc3NlcnRpb24iIHZlcnNpb249IjIuMCIgSUQ9Ii81Vks3TFQ3Rmxp
VWtrYVF1VzZyNGJyRjBERzVFMTg3NiIgSXNzdWVJbnN0YW50PSIyMD
E0LTEwLTE0VDE0OjMyOjE3LjI1MVoiPjxzYW1sOklzc3Vlcjo1cm46bWF
0dWdpdC5hdXRoMC5jb208L3NhbWw6SXNzdWVyPjxTaWduYXR1cm
UgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHN
pZyMiPjxTaWduZWRJbmZvPjxDYW5vbmljYWxpemF0aW9uTWV0aG9
klEFsZ29yaXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMC94b
WwtZXhjLW MxNG4jli8+PFNpZ25hdHVyZUldGhvZCBBbGdvcml0aG0
9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNpZyNyc2Etc
2hhMSIvPjxSZWZlcmVuY2UgVVJJPSIjXzVWSzdMVDdGbGlVa2thUXV
XNnl0YnJGMERHNUUzWDc2Ij48VHJhbnNmb3Jtcz48VHJhbnNmb3Jtl
```

SAML DECODED Prettify (not editable) Expand

```
<samlp:Response
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="_621c4
  <saml:Issuer
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:matu
  </saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status
  </samlp:Status>
  <saml:Assertion
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="
    <saml:Issuer>urn:matugit.auth0.com</saml:Issuer>
    <Signature
      xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/2
        <SignatureMethod Algorithm="http://www.w3.org/2000/0
        <Reference URI="#_5VK7LT7FIiUkkaQuW6r4brF0DG5E3
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/x
            <Transform Algorithm="http://www.w3.org/2001/10/xm
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09
          <DigestValue>ZDkfGO3HtTu50hawzQVjsACzJwc=</Di
        </Reference>
      </SignedInfo>
      <SignatureValue>1Fgpt7AaHcME2gTA158achvGQVqDwHSH
```

JWT punya struktur kayak gini nih, gengs~

Struktur JWT terdiri dari tiga bagian, yaitu:

- Header
- Payload
- Verify Signature

Ketiganya membentuk suatu format xxxxx.yyyyy.zzzzz

Yang pertama adalah Header~

Header dari JWT ini terdiri dari dua bagian, yaitu:

- **Jenis token**, yaitu JWT.
- **Algoritma penandatanganan**, kayak HMAC SHA256 atau RSA.

Di samping merupakan contoh dari header, sob!

Yang kedua adalah Payload~

Payload dari JWT berisi informasi mengenai user.

Informasi tersebut biasanya dipakai untuk melakukan authorization dan authentication.

Kamu masih ingat konsep dari authorization dan authentication, kan?

Yang terakhir yaitu Verify Signature~

Verify signature adalah **gabungan dari isi encode Header dan Payloadnya yang ditambahkan kode secretnya**.

Signature ini dipakai untuk mem-verifikasi bahwa header maupun payload yang ada dalam token nggak berubah dari nilai aslinya.

Berikut adalah formatnya~

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Selanjutnya kita bakal menerapkan JWT di authentication pakai Spring security!

Dalam menambahkan JWT di project, kita bakal menambahkan dependency disamping ini, gengs.

Coba lirik dependency di samping, yaaaa~

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Sejauh ini, kita udah menambahkan 2 model entity di project kita sebelumnya yaitu User dan Role~

Yeayyy!

Selanjutnya, untuk melakukan query pakai Spring Data JPA, kita harus menambahkan Repository-nya dulu.

```java
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsByEmail(String email);
}
```

Ini lanjutannya ya, gengs~

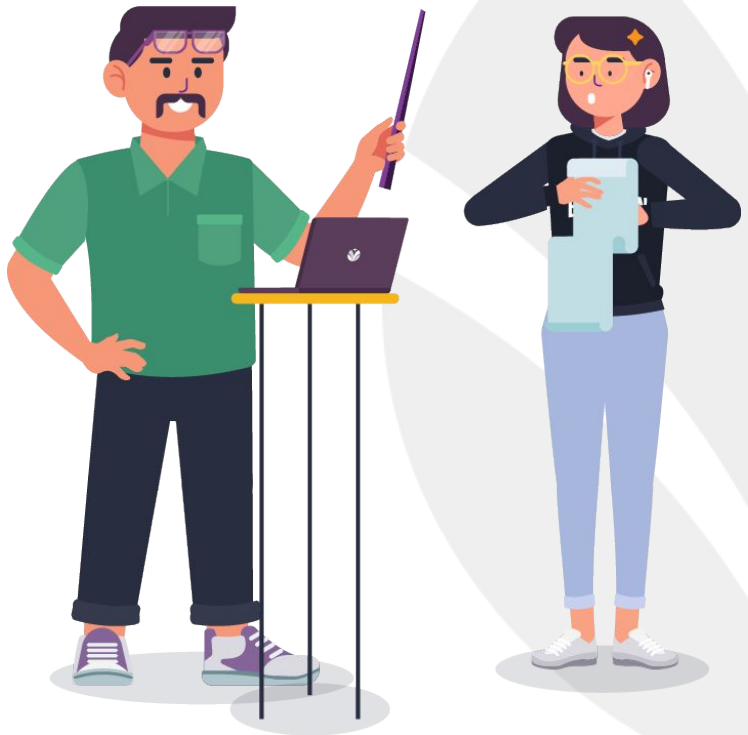
```java
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(ERole name);
}
```

Terus kita juga bakal menambahkan konfigurasi buat mengimplementasi JWT~

Berikut adalah konfigurasinya:

- Konfigurasi 1
- Konfigurasi 2
- Konfigurasi 3



Setelah itu, kita bakal bikin konfigurasi untuk Spring security!

Coba pelajari konfigurasi di bawah ini, ya.

Konfigurasi Spring Security

Next, kita bikin service untuk UserDetails!

Boleh kamu lirik tautan di bawah ini, ya!

- Konfigurasi UserDetails 1
- Konfigurasi UserDetails 2

Selanjutnya kita bikin controller untuk endpoint sign in dan sign up~

Coba kamu simak di sini, yaaaa~

Terakhir adalah bikin DTO request dan response-nya~

Silakan disimak secara berurutan ya, gengs~

- code pertama
- code kedua
- code ketiga
- code keempat

JWT udah beress~

Selanjutnya kita bakal ngobrolin tentang dua hal yang berbeda tapi berhubungan, yaitu Encryption dan Decryption.

Yuhuuu, selamat kenalan~

Kamu tahu password, kan? Iya, yang sifatnya rahasia gitu!

Password merupakan credential yang bersifat rahasia.

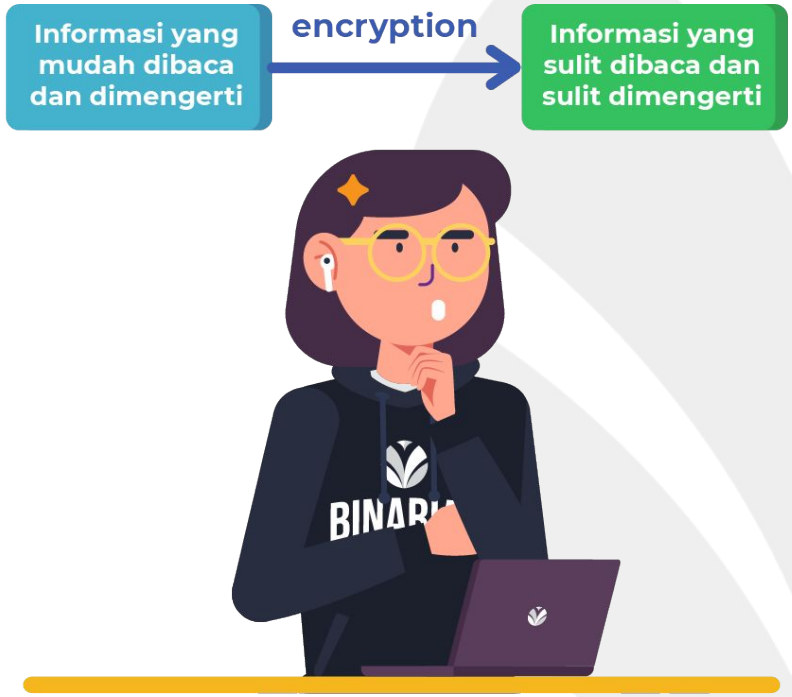
Berhubung sifatnya rahasia, jadi **nggak mungkin kan password disimpan dalam bentuk String yang bisa membahayakan user**.

Iya, soalnya kalau pakai string, seorang application owner dan DB Admin bisa aja tahu password pengguna. Tentu aja hal itu bahaya!

Setelah kita tahu bahwa string bukan solusi yang tepat untuk mengiringi password, maka **hal credential yang bersifat rahasia harus disamarkan dengan cara encryption atau hashing**.

“Encryption? Hashing? Maksudnya?”

Biar ada bayangan, kita langsung bahas aja, yuk!



Jadi, Encryption itu adalah…

Proses mengubah informasi yang mudah dibaca dan dimengerti menjadi informasi yang sulit dibaca dan dimengerti.

Tujuan dari encryption adalah untuk melindungi data dari orang yang nggak berwenang. Yaitu, orang yang mau baca atau mendapatkan informasi dari pesan yang nggak dimaksudkan di tujuan pengiriman.

Nah, selanjutnya ada **proses untuk menerjemahkan informasi yang udah disamarkan, disebut dengan decryption.**

“Cara terjemahinnya gimana? kan udah bersifat rahasia?”

Untuk menerjemahkan informasi yang telah disamarkan ini, pihak penerima udah punya key khusus untuk menerjemahkan informasi.

Kita bahas pakai analogi, yuk!

Misalnya nih, Sabrina mau minta uang sama Mas Gun, tapi karena Mas Gun nggak bawa uang, jadinya Mas Gun mau ngasih lewat transfer rekening aja.

Sebelum mengirim uang, pastinya mesin ATM bakal minta Mas Gun buat masukin password yang tujuannya untuk masuk rekening ATM-nya.

Password ini merupakan bentuk dari encryption.

Jadi, orang lain yang lagi antre di belakang Mas Gun nggak tahu tuh angka berapa aja yang dipilih sebagai password-nya.

Terus kalau passwordnya benar, maka Mas Gun bisa melanjutkan transaksi. Inilah yang disebut sebagai decryption.

Good Job!

Satu persatu materi udah terlewati~

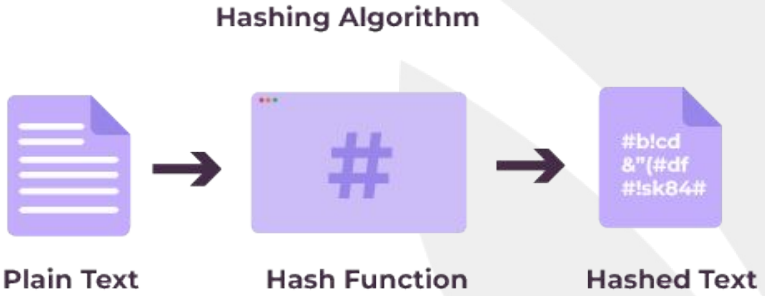
Tapi nih, ternyata eh ternyata, cara untuk menyamarkan informasi tuh bukan lewat encryption dan decryption aja, lho. Tapi ada SHA1 juga!

Ternyata ada cara lain untuk menyamarkan informasi, lho!

Cara lain untuk menyamarkan informasi adalah dengan menggunakan hashing.

Bedanya dengan encryption, **hashing mengamankan hanya secara satu arah**. Informasi yang udah disamarkan sama hashing, nggak bakal bisa dikembalikan lagi.

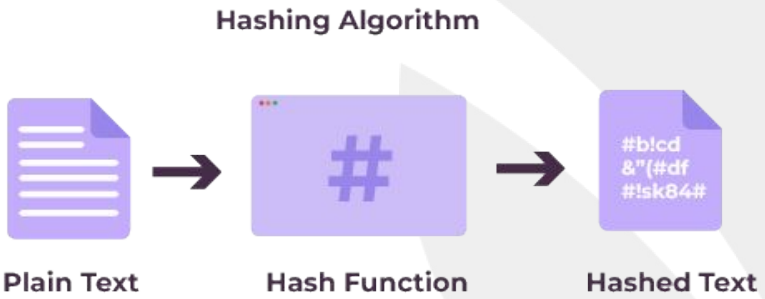
Jadi, buat mencocokkan password yang ada di hashing, dicocokkan dalam keadaan yang udah di-hash.



Ada beberapa tipe hashing yang bisa dilakukan, gengs~

Tipe hashing ini, yaitu:

1. SHA (Secured Hashing Algorithm)
2. MD5
3. Bcrypt



SHA atau Secure Hashing Algorithm

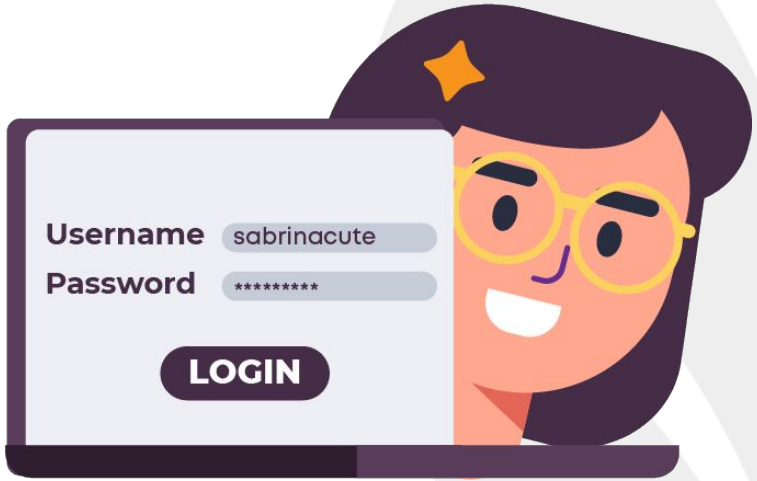
Merupakan fungsi kriptografi yang dirancang khusus oleh penyedia otoritas keamanan internet untuk menjaga keamanan data.

SHA ini bekerja dengan cara melakukan transformasi data pakai fungsi HASH.

SHA 1 dikembangkan pada tahun 1993 oleh lembaga standar pemerintah Amerika Serikat yakni National Institute of Standard and Technology (NIST). SHA 1 ini banyak dipakai pada protokol keamanan TLS.

Contoh penggunaan dari SHA-1 adalah ketika kamu memasukkan kata sandi kamu ke halaman login situs web.

Meskipun itu terjadi di latar belakang tanpa sepengetahuan kamu, itu mungkin aja jadi metode yang dipakai situs web untuk mem-verifikasi bahwa kata sandi yang kamu tuliskan itu asli.

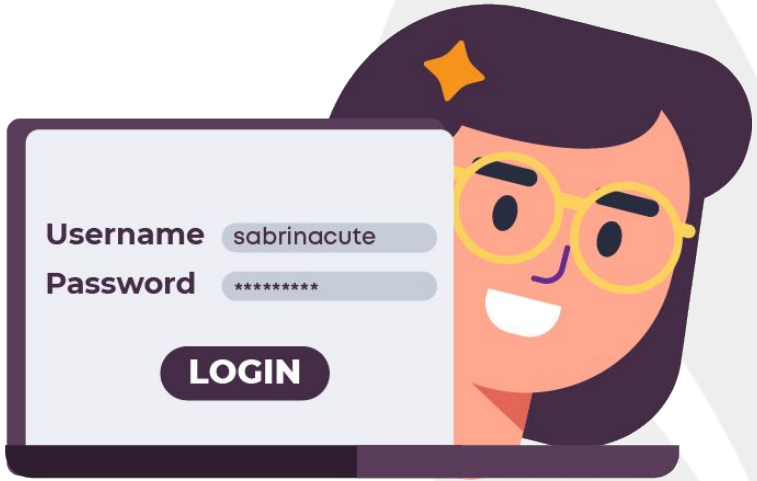


Bayangkan kamu mau masuk ke situs web yang sering kamu kunjungi~

Setiap kali kamu mau masuk, kamu selalu diminta memasukkan nama pengguna dan kata sandi.

Kalau situs web pakai fungsi hash kriptografi SHA-1, itu berarti kata sandi kamu diubah jadi checksum pas kamu memasukkannya.

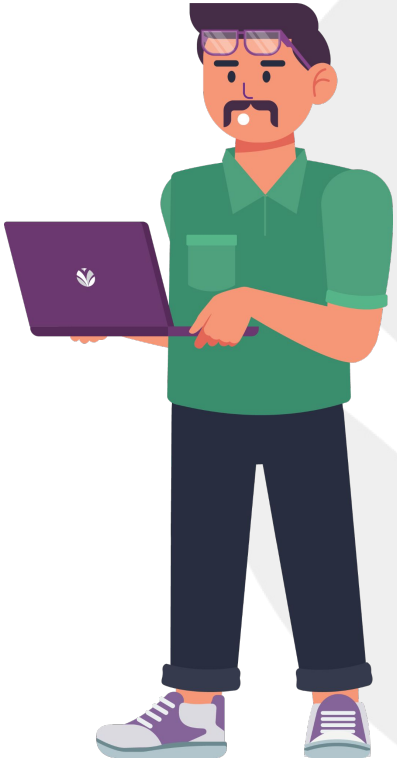
Checksum tersebut kemudian dibandingkan dengan checksum yang disimpan di situs web terkait, yaitu melalui kata sandi kamu saat ini.



Kenapa sih perlu cross-check password yang kita masukin dengan yang udah tersimpan di web?

Tujuannya untuk mengetahui apakah kamu nggak mengubah kata sandi sejak kamu mendaftar. Atau ketika kamu baru aja mengubahnya beberapa saat yang lalu.

Kalau keduanya cocok, kamu bakal diberi akses. Tapi kalau nggak, kamu bakal diberi tahu bahwa kata sandi yang dimasukkan adalah salah.



Hal ini yang jadi dasar untuk teknologi selanjutnya dalam menggunakan Hashing, gengs.

Saat ini, SHA dikembangkan jadi SHA 256 atau sama dengan SHA 2.

Begitu juga dengan tipe hashing yang lain, kayak BCrypt dan MD5. Perbedaannya ada di hashing functionnya.

Sejauh ini, pasti bayang-bayang kamu tentang Spring Security udah buanyak banget, kan?

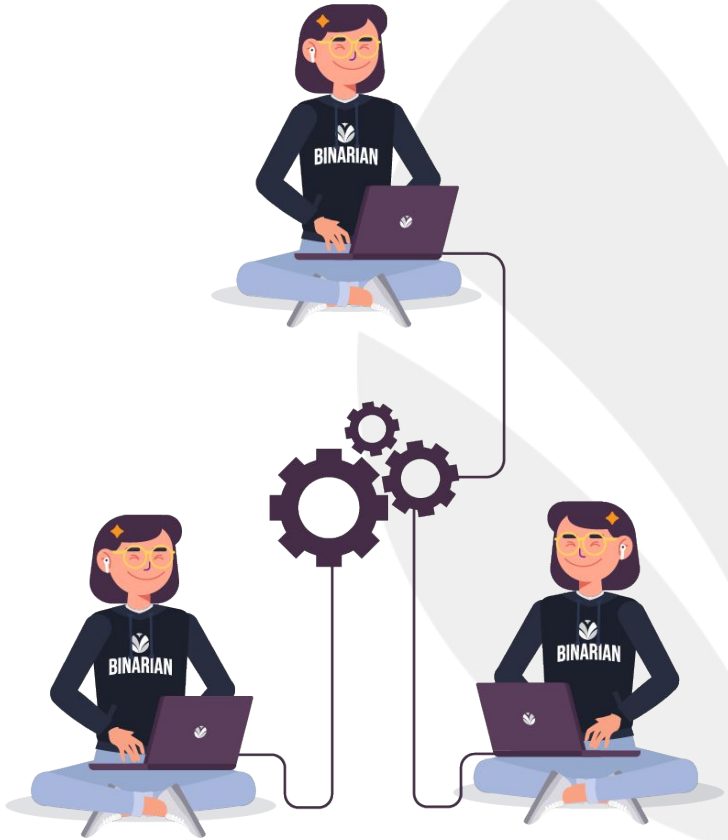
Tapi masih ada yang kurang nih, sob. Kita belum bahas tentang OAuth2.

Penasaran kan?

Istilah baru nih, namanya OAuth. Kenalan, yuk!

OAuth (Open Authorization) merupakan framework authorization yang memungkinkan application third party untuk mendapatkan akses dengan access token yang dipakai bersama-sama.

Cara untuk mendapatkan access token disebut dengan **Grant type**.



3 jenis Grant type dalam Oauth2~

Berikut adalah jenis-jenisnya:

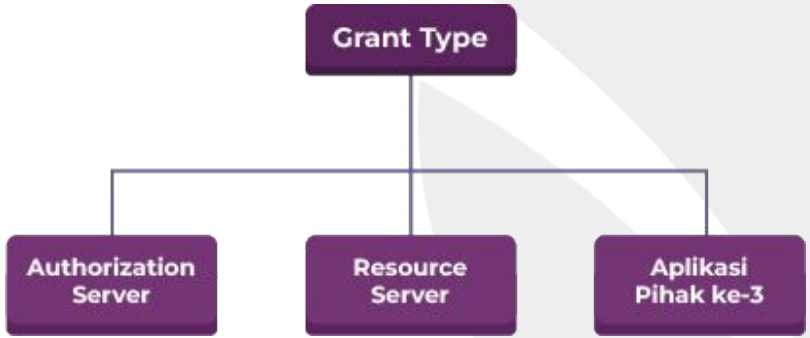
1. Grant Type Authorization Code
2. Grant Type Resource Owner Password Credential
3. Grant Type Client Credentials

Grant Type Authorization Code

Dengan grant type ini, sebuah code yang berasal dari authorization server bisa ditukarkan dengan access token.

Token ini bakal dipakai untuk mengakses resource.

Ada 3 aplikasi di grant type ini, yaitu **authorization server** (penyedia authorization), **resource server** (penyedia API), dan **aplikasi pihak ketiga**.



Pas ada aplikasi pihak ketiga yang mau mengakses API, API bakal bertanya dulu kepada authorization server, apakah token yang dikirim valid atau nggak.

Kalau valid, berarti resource server bakal memberikan resource yang dibutuhkan.

Tapi, kalau ternyata nggak valid, berarti bakal menampilkan bahwa aplikasi tersebut nggak berhak melakukan akses terhadap resource tersebut.

Dengan grant type ini, user bakal melakukan login pada authorization server. Dimana URl pada login ada client id dan authorization type dari aplikasi yang mau mengakses resource tersebut.

Lalu, authorization server bakal melakukan redirect ke aplikasi pihak ketiga dengan membawa sebuah code.

Nah, code ini bakal dipakai sama aplikasi pihak ketiga untuk menukarkan-nya dengan access token.

Grant Type Resource Owner Password Credentials

Grant type ini menggunakan username dan password langsung dari ownernya.

Biasanya dilakukan di aplikasi yang dimiliki oleh perusahaan yang sama.

Sayangnya, nggak disarankan untuk aplikasi pihak ketiga.

Grant Type Implicit Client

Biasanya dipakai untuk aplikasi yang pakai Javascript gitu~

Grant type yang dipakai di aplikasi Spring Boot biasanya pakai Grant Type Authorization Code, sob!

Berikut contoh implementasi penggunaan OAuth2 buat men-secure REST API.

Spring Boot OAuth2

Karena butuh client id dan client secret supaya bisa melakukan Oauth, berikut referensi petunjuk untuk mendapatkannya:

- **Google**

  OAuth2 Authentication - Google

- **Facebook**

  OAuth2 Authentication - Facebook

Lastly, untuk menutup materi pada topik ini kita bakal PDKT sama API lagi. Cukup berbeda dengan sebelumnya, kali ini kita obrolin tentang API Key.

Langsung aja kita bahas, yuk!

“Apa itu API Key?”

Ada beberapa konsep terkait API key yang bisa kamu intip pada beberapa poin dibawah ini:

- API Key merupakan sebuah identifier yang dipakai untuk menggunakan API.
- API key juga salah satu cara untuk melakukan authorization seperti token.
- API key biasanya dipakai di header dengan key X-API-Key.

Biasanya API Key dikombinasikan dengan authentication lain.

Untuk penggunaan API key di Spring security sendiri, bisa menggunakan referensi berikut:

A Trip Through Spring Security

Bip bip~

Wah, ternyata ada misi lagi yang perlu kamu selesaikan nih!

Misinya adalah: **kamu bisa melakukan login menggunakan akun Google dan Email.**

Kalau tugasnya selesai, kamu boleh minta facilitator untuk mengecek penugasan yang kamu kerjakan di pertemuan selanjutnya. Selamat mengerjakan

Dengan mempelajari Spring Security, kita jadi nggak was-was lagi deh mengenai keamanan website yang kita rancang~

Misalkan kamu diminta mengembangkan sebuah website, keamanan apa saja yang bakalan kamu gunakan?

Nah, selesai sudah pembahasan kita di Chapter 6 Topic 2 ini.

Selanjutnya, kita bakal belajar tentang **Java Logging** yang berkaitan dengan tempat untuk menyimpan info penting!

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~