Spring Security

- Utilizando o Spring Security, ao realizar um pedido de API, você é redirecionado para o /login.

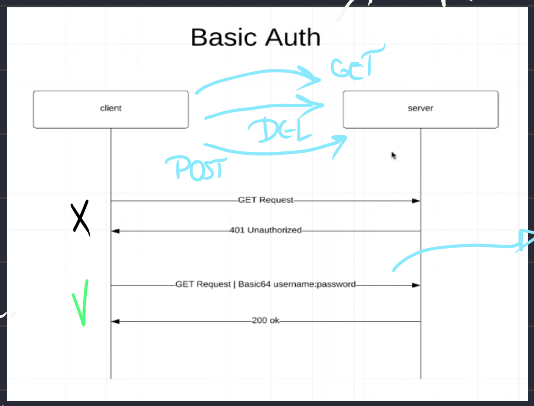
↳ Portanto, o Spring faz todo o serviço da segurança.

↳ Para efetuar login: → Forma padrão
name: user
password: password gerada pelo log do Spring

- Esse sistema é baseado em login e password (form based authentication).

URL's basicos: .../login
.../logout
↳ Gerados por padrão no Spring Security.

⇨ Basic Auth Authentication → mais simples!



- É muito bom quando esta sendo acessado externamente

→ É preciso enviar nome e senha em (todas) as requisições.

→ Pela web.

- Um dos grandes problemas do basic auth é que não é possível deslogar.

↳ O login é enviado todas as vezes.

◦ Configuração básica

Spring Config ⇒ @Configuration @EnableWebSecurity

Whitelist de endpoints

Autorizar os requests

Permitir os endpoints da whitelist

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    // Any request must be authenticated, with http basic.
    http.authorizeRequests()
        .antMatchers( ...antPatterns: "/", "index", "/css/*", "/js/*")
        .permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .httpBasic();
}
```

Todos os requests

Devem ser autenticadas.

Utilizando basic auth.

⊛ OBS: Durante a execução, um usuário é criado para acessar o Spring Security e então é deletado automaticamente.

↳ Para resolver isso — colocar users em uma db:

- Username.
- Password (encoded). ↷ Usar o BCryptPasswordEncoder
- Roles. ↶ ROLE_NAME
- Authorities.

⇒ Password Hashing

- Hashing é o processo de gerar um hash (string) a partir de uma mensagem. Para gerar esse hash, utiliza-se uma função matemática cryptographic hash function.

↳ Existem ④ propriedades para uma hash function ser segura:

1. It should be *deterministic*: the same message processed by the same hash function should *always* produce the same *hash*
2. It's not *reversible*: it's impractical to generate a *message* from its *hash*
3. It has high *entropy*: a small change to a *message* should produce a vastly different *hash*
4. And it resists *collisions*: two different *messages* should not produce the same *hash*

↳ Também é importante ser lenta, para não ser possível adivinhar as senhas.

- As funções hash ideais são PBKDF2, BCrypt and SCrypt.

⇒ Role e Permissions

↳ São as autorizações em si.

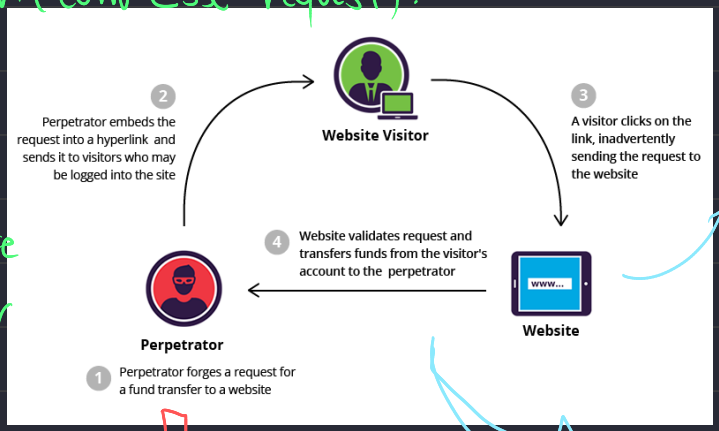
↳ É o cargo em si, que contém algumas permissões.

* No antMatchers(), a ordem importa!

⇒ CSRF (Class Site Request Forgery)

Gera um link (com esse request).

Manda um link para algum visitante que pode estar no site.

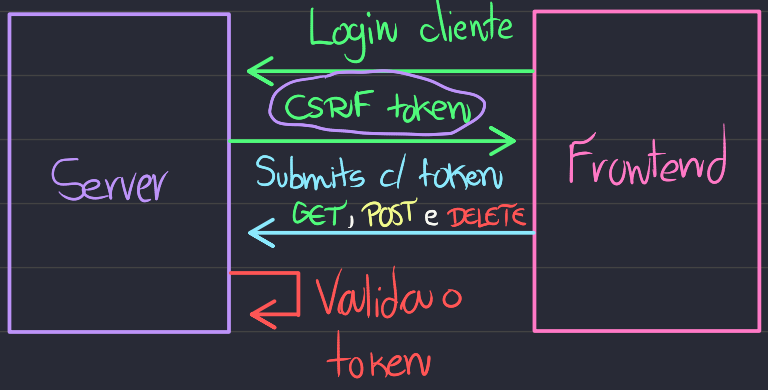


Esse visitante entra no link mandando o request pr o site.

O site valida essa requisição e o banco envia o dinheiro.

Forja um link para gerar uma requisição para outro programa (banco, por exemplo).

↳ Para resolver esse problema:

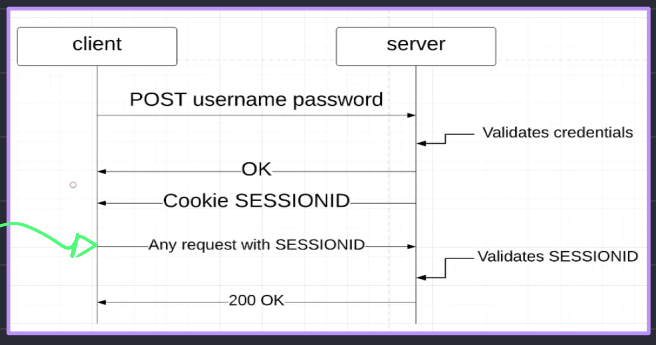


* Esse token é recomendado para aplicações com clientes que utilizam/possam utilizar, um web browser.

* Base64 não é uma criptografia, é somente um modo em que a mensagem está disponível.

=> Form Based Authentication

Manda o SESSIONID em todos os requests



↳ algumas vantagens:

- Usuario e Senha.
- Forms
- HTTPS recomendado.
- Padrão na maioria dos sites.
- Logout.

* SessionId expira em 30 minutos.
  ↳ Form based