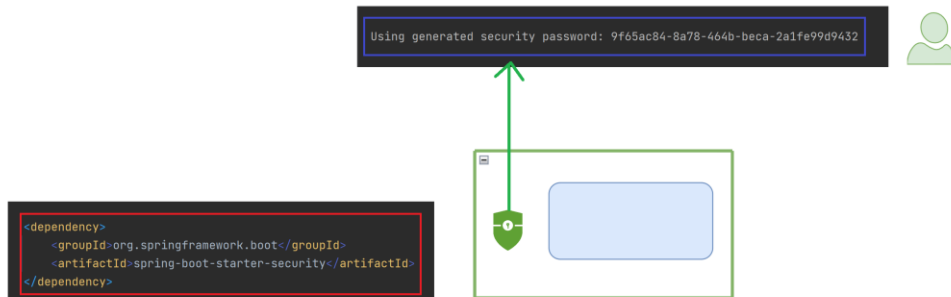




## Autenticação Padrão - Spring Security

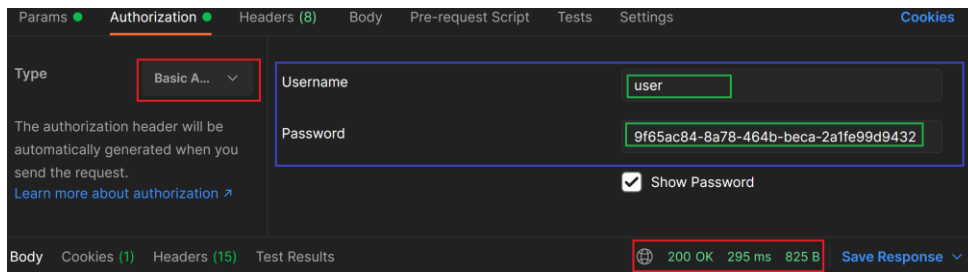
O **Spring Security** é um projeto do **Spring** focado em **autenticação e autorização** dos usuários, provendo recursos de segurança para aplicação. Para usar o projeto **basta adicionar a dependência** e a **API** se torna **protegida automaticamente com uma credencial padrão gerado pelo próprio Spring Security** (até alguma configuração de fato ser feita):



Os requests que forem feitos a uma API com o módulo do Spring Security serão **não autenticados** se o client não passar as credencias:



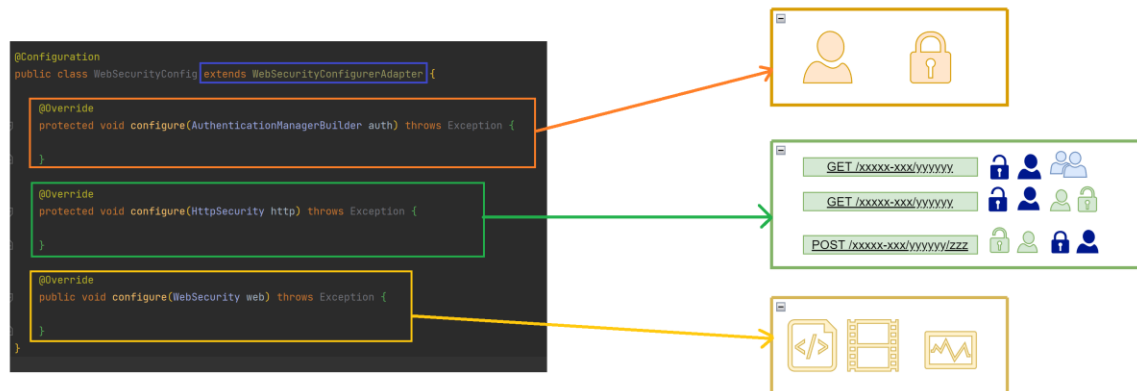
Para **autenticar o client no sistema** e assim permitir com que os requests sejam feitos é necessário passar as **credencias de autenticação**, nesse caso aqui, os **próprios valores default do Spring Security** já suprem essa necessidade (**a autenticação padrão é basic auth**):



## Configurando autenticação Deprecated - Spring Security

Um **modo de configuração que atualmente está depreciado**, mas ainda pode ser utilizado em projetos com o **Spring** numa versão mais antiga é através da classe **WebSecurityConfigurerAdapter**. Essa **classe** traz 3 métodos principais de configuração, sendo cada um deles para algo específico.

- **AuthenticationManagerBuilder**: É uma configuração responsável por autenticação do usuário no sistema.
- **HttpSecurity**: É uma configuração responsável por autorização de request em endpoints, quem pode acessar qual url e etc..
- **WebSecurity**: É a configuração para recursos estáticos como JS, imagens e etc... caso o frontend fosse integrado.



A

anotação **@EnableWebSecurity** "desliga" todas as configurações padrões de segurança do Spring, fazendo com que ele leve em consideração SOMENTE o que você tiver definido na classe.

Um exemplo do que podemos fazer com o configure de **HttpSecurity** é concatenar as configurações de segurança que os requests precisam seguir, eu também poderia controlar o acesso a endpoints específicos:

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .httpBasic().httpBasicConfigurer<HttpSecurity>()
        .and() HttpSecurity
        .authorizeHttpRequests().authorizeHttpRequests<AuthorizationManagerRequestMatcherRegistry>
        .anyRequest().authenticated()
        .and() HttpSecurity
        .csrf().disable();
}

```

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers(HttpMethod.GET, "/topics/**").permitAll()
        .antMatchers(HttpMethod.GET, "/topics/**").permitAll()
        .anyRequest().authenticated()
}

```

Com o configure de **AuthenticationManagerBuilder** poderíamos definir uma autenticação em memória com os dados pré definidos, ou usar uma implementação específica do Spring para autenticar via banco de dados (a melhor opção em web):

```

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().inMemoryUserDetailsManager<AuthenticationManagerBuilder>
        .withUser( username: "Lucas" ) UserDetailsService<UserDetailsBuilder>
        .password(passwordEncoder().encode( newPassword: "1234567" ))
        .roles("ADMIN");
}

```

```

@Autowired
private AutenticacaoService autenticacaoService;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(autenticacaoService).passwordEncoder(new BCryptPasswordEncoder());
}

```