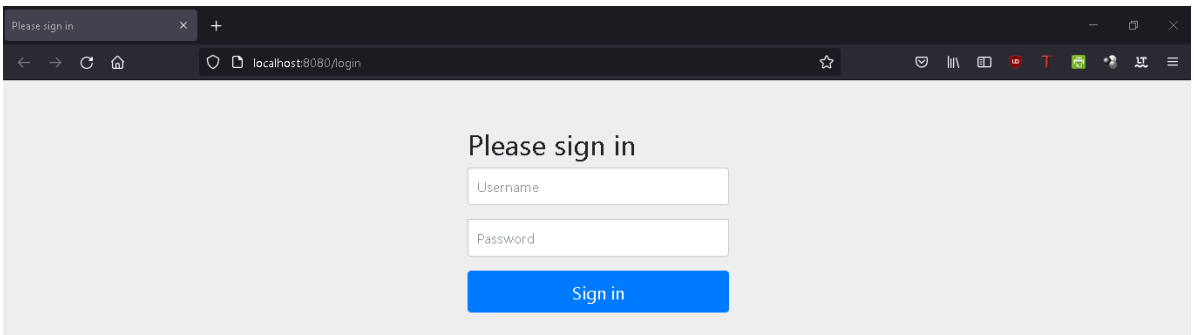
Spring Security

No mundo da tecnologia, em especial da Internet, nenhuma aplicação em execução na Nuvem pode ficar sem algum tipo de Segurança habilitada, devido aos inúmeros perigos existentes no mundo virtual como ataques Hackers, invasões de Servidores, roubos de dados, entre outros. No ecossistema Spring, isso é feito com a ajuda da **Dependência Spring Security**.

Vamos adicionar a **Dependência Spring Security** no Projeto Blog Pessoal, adicionando as linhas abaixo no arquivo **pom.xml**, Salvar e Executar a aplicação:

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

Abra a sua aplicação no Navegador da Internet através do endereço: **http://localhost:8080** e veja o que acontece.



Como num passe de mágica...

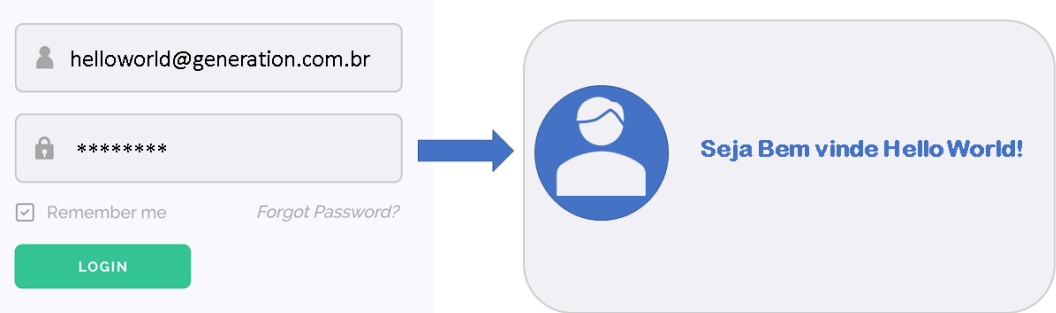
- ... Você tem uma página de login gerada automaticamente.
- ... Você não pode mais executar solicitações GET, POST, PUT e DELETE.
- ... Todo o seu aplicativo está bloqueado e solicita que você insira um nome de usuário e uma senha, que não existe.

Tendo sobrevivido ao susto inicial, você deve ter pensado: Como tudo isso aconteceu?

1. Segurança da Aplicação

Antes de estudarmos a Spring Security, vamos compreender 3 conceitos importantes da Segurança da Informação:

1.1. Autenticação

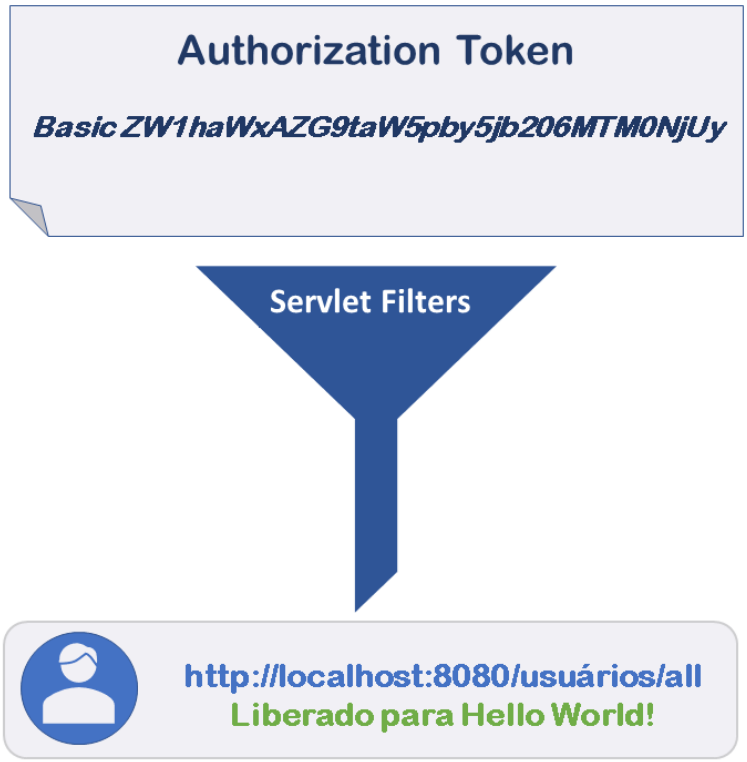


É o primeiro processo da Segurança da Informação, popularmente conhecido como Login no sistema. É o momento em que o usuário informa o seu usuário de acesso (e-mail) e a sua senha (criptografada), e o sistema fará a checagem se estas informações estão corretas.

1.2. Autorização

É o segundo processo da Segurança da Informação, popularmente conhecido como Direitos de acesso (Roles) no sistema. É o momento em que o sistema checará o que o usuário pode e não pode fazer no sistema, ou seja, as suas permissões dentro do sistema (Quais Recursos e Endpoints podem ser acessados?).

1.3. Filtros de Servlet

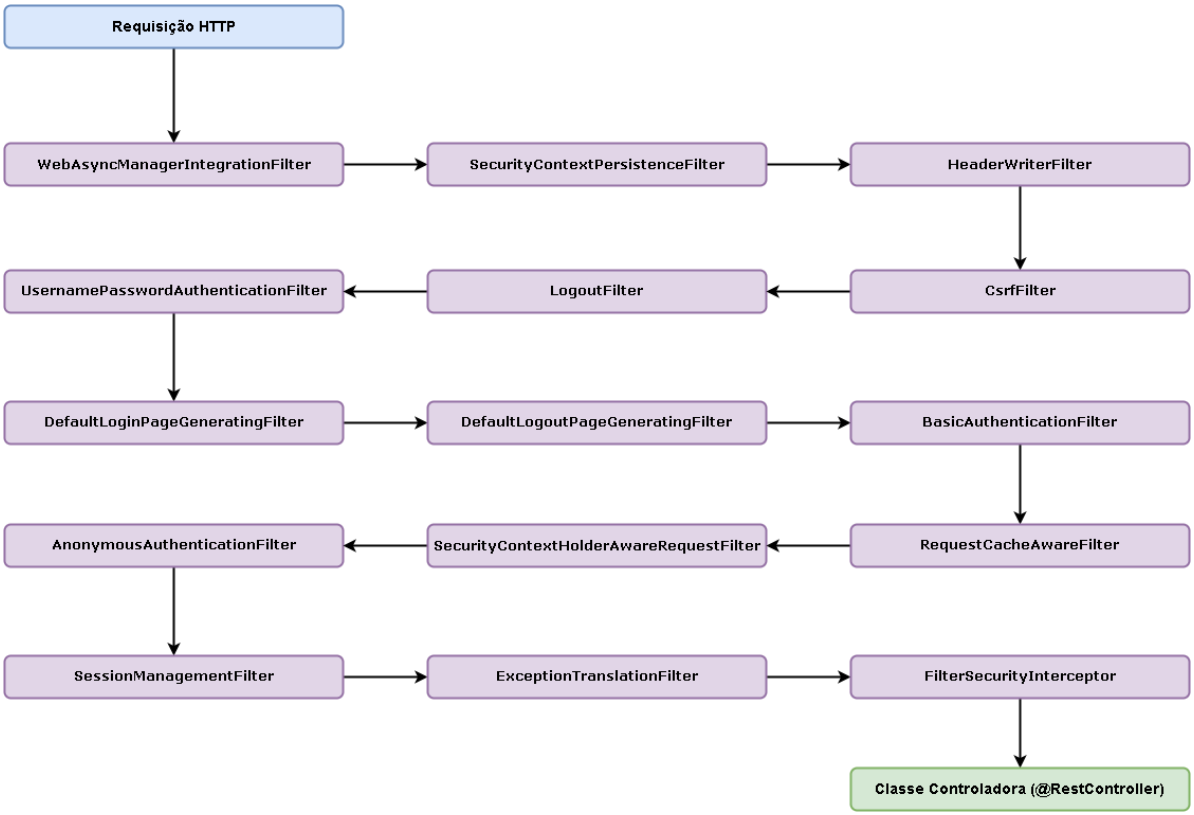


Qualquer aplicativo da web Spring é apenas um **Servlet** (é uma Classe Java usada para criar aplicações WEB), que redireciona todas as Requisições HTTP recebidas (por exemplo, do Front-end Angular ou React), para as suas respectivas Classes Controladoras (**@RestControllers**).

Como nestas requisições não existe uma segurança e a autenticação e a autorização devem ser efetuadas antes das Requisições HTTP, a Spring Security oferece como solução para este problema os **Filtros**, que na prática são Objetos que interceptam toda e qualquer Requisição HTTP recebida antes de chegarem ao controlador. Por isso que o seu Blog Pessoal está "trancado" após a inserção da Dependência Spring Security.

A figura acima, ilustra o filtro **BasicAuthenticationFilter** checando o Token de Autorização do Usuário enviado no cabeçalho da requisição. Como ele está dentro do padrão Basic, o filtro libera o envio da requisição para a Classe Controladora específica.

A Dependência Spring Security ao ser inserida, habilita 15 filtros, onde cada um tem uma função específica e precisam ser configurados. Não abordaremos este assunto em detalhes por ser muito extenso, além de demandar tempo e dedicação, de qualquer forma você já sabe como a sua aplicação ficou totalmente bloqueada. A Imagem abaixo mostra os 15 filtros:



2. Spring Security

Analisando nossos projetos podemos perceber que nossa API, até este momento, não possuía nenhuma segurança, ou seja, qualquer pessoa poderia acessar todos os nossos endpoints e ter acesso à todos os recursos livremente. Precisamos entender que algumas aplicações contém informações vitais como: dados pessoais, dados bancários, usuário e senha de acesso, e portanto precisamos garantir que a nossa API e estes dados estejam devidamente protegidos. E para isto podemos contar com a dependência do Spring chamada **Spring Security** (adicionada acima).

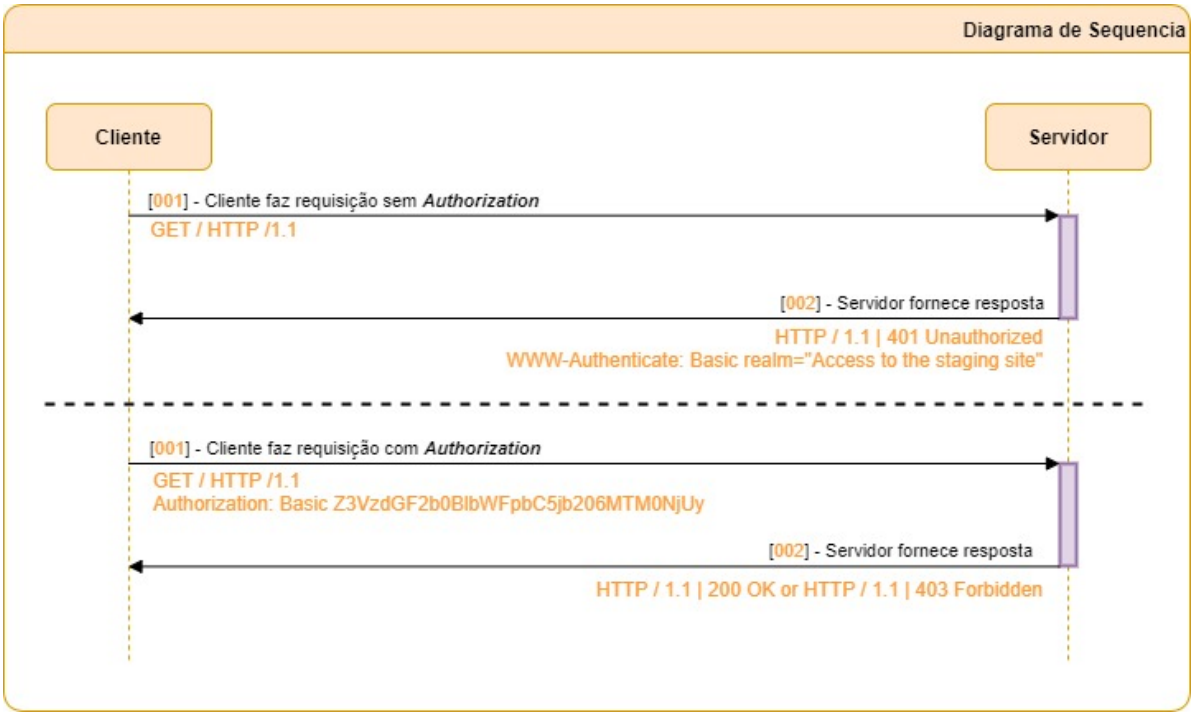
A **Spring Security** é um framework para Java, que provê autenticação, autorização, filtros e diversas outras funcionalidades para aplicações corporativas, com o objetivo de proteger a nossa aplicação contra acessos indevidos. Iniciado em 2003 por Ben Alex, a Spring Security é distribuída sob a licença Apache Licence. Ela oferece diversos recursos que permitem que muitas práticas comuns de segurança sejam implementadas ou configuradas de forma direta.

O esquema de autenticação que utilizaremos em nosso projeto é o **HTTP Basic**, onde entraremos com o **e-mail (usuário) e a senha** do usuário e através de um endpoint liberado, o Spring Security irá criptografar a senha e fazer uma consulta no nosso Banco de dados para saber se o usuário já existe. Esta checagem será feita através da Camada de Serviço (service) da aplicação. Se a consulta encontrar o usuário e a senha, a Spring Security devolverá como resposta um Authorization com o prefixo Basic + token. Este token ficará registrado na nossa aplicação na camada de Security, e apenas por meio dele que o usuário poderá consumir a API.

Para melhor compreensão deste sistema, é necessário dividi-lo em 2 ecossistemas: **Usuário** e **Segurança**, que veremos mais a frente.

3. Conhecendo HTTP authentication

O **IETF (*Internet Engineering Task Force*)** tem como missão identificar e propor soluções para as questões/problemas relacionados à utilização da Internet, além de propor a padronização das tecnologias e protocolos envolvidos. O mesmo define a estrutura de autenticação HTTP que pode ser usada por um servidor para definir uma solicitação do cliente. O servidor responde ao cliente com uma mensagem do tipo **HTTP Status 401(Não autorizado)** e fornece informações de como autorizar com um cabeçalho de resposta **WWW-Authenticate** contendo ao menos uma solicitação. Um cliente que deseja autenticar-se com um servidor pode fazer isso incluindo um campo de cabeçalho de solicitação **WWW-Authenticate** com as credenciais. No Diagrama de Sequência abaixo pode se observar este relacionamento:



No caso de uma **autorização "Basic"** (como a mostra a figura acima), a troca deve acontecer por meio de uma conexão HTTP (TLS) para ser segura. Se um servidor recebe credenciais válidas, mas que não são adequadas para ter acesso a um determinado recurso, o servidor responderá com o código de status **HTTP Status 403 (Proibido!)** . Ao contrário de **HTTP Status 401(Não autorizado)**, a autenticação é impossível para este usuário. O cabeçalho de requisição **Authorization** contém as credenciais para autenticar um agente de usuário com um servidor. Aqui o tipo é novamente necessário, seguido pelas credenciais, que podem ser codificadas ou criptografadas dependendo do esquema de autenticação usado. No caso acima foi utilizado o esquema de autenticação Basic que será explicado na sequencia.


[Documentação: HTTP Status Code 401 - Unauthorized]


[Documentação: HTTP Status Code 403 - Forbidden]


[Documentação: Cabeçalho HTTP WWW-Authenticate]


[Documentação: Cabeçalho de Requisição HTTP Authorization]

3.1. Esquema de autenticação Basic

A estrutura geral de autenticação HTTP é usado por vários esquemas de autenticação. Os esquemas podem divergir na força da segurança e na disponibilidade do software cliente ou servidor. O esquema mais comum de autenticação é o "Basic", mas existem outros esquemas oferecidos por serviços de hospedagem, como Amazon AWS, Google ou Microsoft. Os esquemas de autenticação mais comuns são:

Esquema de autenticação	Documentação
Basic	[RFC7617]
Bearer	[RFC6750]
Digest	[RFC7616]
HOBA	[RFC7486, Section 3]
Mutual	[RFC8120]
Negotiate	[RFC4559, Section 3]
OAuth	[RFC5849, Section 3.5.1]
SCRAM-SHA-1	[RFC7804]
SCRAM-SHA-256	[RFC7804]
vapid	[RFC 8292, Section 3]

ATENÇÃO: Para melhor compreensão no momento, vamos focar apenas no entendimento do formato Basic, que é considerado o principal esquema para compreender os demais meios de autorização. Vale mencionar que para aprender os demais é necessário tempo e muita dedicação.

O esquema **Basic**, segundo sua documentação, consiste em um conjunto de caracteres que posicionados após a palavra **"Basic "** no formato **“email:senha”** codificados utilizando o modelo **Base64**, formando um token ***Authorization*** para ser passado ao sistema. Abaixo veremos um exemplo de código para gerar a estrutura em Java:

Dependência:

```
<!-- Dependência para Codificação do Token -->
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
</dependency>
```

Exemplo de código em java:

```
import java.nio.charset.Charset;
import org.apache.commons.codec.binary.Base64;

private static String geradorBasicToken(String email, String senha) {
    String estrutura = email + ":" + senha;
    byte[] estruturaBase64 =
        Base64.encodeBase64(estrutura.getBytes(Charset.forName("US-ASCII")));
    return "Basic " + new String(estruturaBase64);
}
```

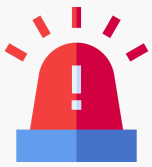
O resultado do código acima proporciona o retorno de um esquema de autenticação Basic respeitando as regras ja definidas pela IETF (*Internet Engineering Task Force*).

Exemplo:

Usuário: *admin@email.com.br*

Senha: *admin123*

Token gerado: *Basic YWRtaW5AZW1haWwuY29tLmJyOmFkbWluMTIz*



ALERTA DE BSM: Mantenha a Atenção aos Detalhes ao escrever o formato Basic, o mesmo é representado da palavra "*Basic* " com um espaço na frente + a criptografia de um conjunto de caracteres (*email:senha*) fornecidos ao se autenticar no sistema. No link abaixo é possível testar a codificação 64 bits.

Site: Codificador 64 bits

O Token gerado será enviado no Header (cabeçalho) de uma requisição ao servidor devidamente configurado para acessar os seus recursos.