

Seção 08 - Resumo

Serviço de e-mail STMP

Um recurso muito utilizado por sites que trabalham com cadastro de usuários é a confirmação do cadastro via e-mail. Isso passou a ser utilizado para evitar que usuários criassem cadastros falsos com e-mails inexistentes ou mesmo de outros usuários.

O sistema de confirmação de cadastro deve enviar um e-mail para o usuário com um link de confirmação e somente após o usuário acessar esse link ele estará apto a logar no sistema.

Para se trabalhar com o recurso de confirmação é necessário configurar um serviço de e-mail em nossa aplicação e o primeiro passo é incluir no *pom.xml* a dependência referente ao serviço de e-mail. O Spring Boot tem um starter definido para isso, o qual está descrito a seguir:

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

Esse starter trás as bibliotecas necessárias como a javamail e uma pré implementação na qual vamos apenas informar alguns dados de conexão com o servidor de e-mail.

Como servidor de e-mail podemos usar o GMail, Bol, UOL, Terra, Outlook, HotMail e até mesmo um serviço de e-mail próprio de sua empresa. O necessário apenas é que esse servidor de e-mail forneça um recurso baseado no protocolo SMTP o qual é utilizado para o envio de e-mail.

Para que a conexão entre a aplicação e o servidor de e-mail aconteça devemos informar alguns dados sobre o acesso a conta de e-mail do servidor escolhido. Como exemplo, vamos trabalhar com o GMail e no arquivo *application.properties* é o local onde elas são adicionadas:

```
#JAVAMAIL
spring.mail.host= smtp.gmail.com
spring.mail.port= 587
spring.mail.username= $eu_email@gmail.com
spring.mail.password= $ua_$enha
spring.mail.properties.mail.smtp.auth= true
```

```
spring.mail.properties.mail.smtp.starttls.enable= true  
spring.mail.test-connection= true
```

As propriedades acima são necessárias para a conexão entre a aplicação e o servidor de e-mail porque quem realiza o envio do e-mail não é a aplicação mas sim o servidor de e-mail. Por conta disso, é preciso que a aplicação realize o login na conta e essa etapa é realizada via protocolo SMTP.

Alguns servidores de e-mail precisam de informações adicionais além do `host`, `port`, `username` e `password`, que são sempre obrigatórias. No caso do GMail é preciso informar com o valor `true` as propriedades `smtp.auth` e `smtp.starttls.enable` que indicam que é necessário uma autenticação segura via `tls` (**Transport Layer Security**). Caso utilize outro servidor de e-mail é necessário consultar o serviço SMTP do servidor escolhido para ter certeza do que será necessário configurar.

Por fim, a propriedade `spring.mail.test-connection` tem como objetivo informar a aplicação que ela quando inicializada deve testar se o servidor de e-mail está on-line.

Saiba também que alguns servidores de e-mail, como o Gmail, podem exigir que você libere na sua conta de e-mail a permissão de acesso por parte da aplicação. Caso essa permissão não seja liberada a aplicação não conseguirá se conectar ao seu e-mail no servidor.

Alguns servidores de e-mail estão bloqueados para o acesso de aplicativos SMTP e é preciso liberar nas configurações de sua conta de e-mail. No caso do Gmail, para liberar o acesso a aplicativos menos seguros acesse o seguinte link:

<https://myaccount.google.com/lesssecureapps>

Caso esteja com acesso desativado, ative-o para que consiga usar o Gmail com servidor SMTP de sua aplicação.

Alguns usuários estão tendo problemas com o Gmail e exceções estão sendo lançadas como estas:

```
Caused by: javax.net.ssl.SSLHandshakeException:  
sun.security.validator.ValidatorException:
```

```
PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target

org.springframework.beans.factory.BeanCreationException: Error creating bean
with name
'org.springframework.boot.autoconfigure.mail.MailSenderValidatorAutoConfigur
ation': Invocation of init method failed; nested exception is
java.lang.IllegalStateException: Mail server is not available

Caused by: javax.net.ssl.SSLException: java.lang.RuntimeException:
Unexpected error: java.security.InvalidAlgorithmParameterException: the
trustAnchors parameter must be non-empty
```

Ao que parece é uma mudança no protocolo de segurança, nesse caso, para resolver inclua a seguinte propriedade junto as demais:

```
spring.mail.properties.mail.smtp.ssl.trust= smtp.gmail.com
```

E-mail baseado em template

Quando realizamos o envio de e-mail via aplicação podemos trabalhar com mensagens baseadas em texto plano ou em HTML. Com a escolha pelo HTML será possível criar uma página com um layout pré definido e ter variáveis nessa página que recebam valores gerados pela aplicação.

Esse tipo de recurso é possível devido a uma implementação baseada em template. Tempos atrás, um template bastante usado em aplicações Java era o [Apache Velocity](#) que fornece uma fácil integração com o Spring MVC. Entretanto, como estamos trabalhando com o template de páginas Thymeleaf vamos usá-lo como template de e-mail também.

Veja no exemplo HTML que temos algumas variáveis baseadas em expressões Java e Thymeleaf para adicionar valores ao template a partir da aplicação:

```
<body style="margin: 0; padding: 0; ">
  <table align="center" border="0" cellpadding="0" cellspacing="0"
width="600"
    style="border-collapse: collapse; padding: 40px 30px 40px 30px;
font-family: Arial, Helvetica, sans-serif;">
    <tr>
      <td align="center" style="padding: 20px 0 10px 0;">
        
        <h1 style="font-weight: 200;">Clínica Spring Security</h1>
        <p style="margin-bottom: 0;">Especialidades
Médicas</p>
      </td>
```

```

        </tr>
        <tr>
            <td align="center" bgcolor="MediumSeaGreen">
                <h4 class="text-center" th:text="${titulo}"><strong>Bem
vindo...</strong></h4>
                <p th:text="${texto}">Precisamos que confirme seu
cadastro...</p>
                <p th:text="${verificador}"><strong>8050jk</strong></p>
                <p class="text-center mb-0" th:if="${linkConfirmacao !=
null}">
                    <a th:href="${linkConfirmacao}">Confirmação</a>
                </p>
            </td>
        </tr>
    </table>
</body>

```

Outro ponto a destacar é o código CSS da página que deve ser adicionado nas tags html para que a página tenha 100% de compatibilidade com o servidor de e-mail. É possível também ter algumas instruções entre as tags `<style>`, mas em alguns casos elas podem não ser interpretadas pelo servidor de e-mail. Links apontando para CSS externo também não são interpretados pelo servidor de e-mail.

Método para envio de e-mail

Para realizar o envio de e-mail é preciso criar uma classe como um *bean* do Spring e injetar nesta classes dois novos *beans* que são:

- `JavaMailSender`: responsável por encapsular e enviar o e-mail;
- `SpringTemplateEngine`: responsável por lidar com o template de e-mail junto ao Thymeleaf.

Nesta classe teremos o método que fará o envio do e-mail com a página de template como corpo do e-mail. Variáveis do tipo `MimeMessage` e `MimeMessageHelper` são usadas para criar e encapsular o objeto que será enviado como e-mail junto ao `JavaMailSender`.

```

@Service
public class EmailService {
    @Autowired
    private JavaMailSender mailSender;
    @Autowired
    private SpringTemplateEngine template;

    public void enviarPedidoDeConfirmacaoDeCadastro(String destino, String
codigo)

    throws MessagingException {

```

```

        MimeMessage message =
            mailSender.createMimeMessage();
        MimeMessageHelper helper =
            new MimeMessageHelper(message,

MimeMessageHelper.MULTIPART_MODE_MIXED_RELATED, "UTF-8");

        Context context = new Context();
        context.setVariable("titulo", "Bem vindo.");
        context.setVariable("texto", "Confirme seu cadastro no link
abaixo");
        context.setVariable("linkConfirmacao",

"http://localhost:8080/u/confirmacao/cadastro?codigo=" + codigo);

        String html = template.process("email/confirmacao", context);
        helper.setTo(destino);
        helper.setText(html, true);
        helper.setSubject("Confirmacao de Cadastro");
        helper.setFrom("nao-responder@clinica.com.br");

        helper.addInline("logo", new
ClassPathResource("/static/image/spring-security.png"));

        mailSender.send(message);
    }
}

```

Em uma variável do tipo `Context`, pertencente ao pacote `org.thymeleaf.context`, inserimos as mensagens que vão para a página de template via variáveis nomeadas com os nomes daquelas que foram declaradas no template.

Esse objeto `context` deve ser incluído como parâmetro do método `template.precess()`, onde também informações qual e onde está a página de template. O retorno do método `process()` é um objeto `String` que será incluído como corpo da mensagem via método `setText()` da variável `helper`.

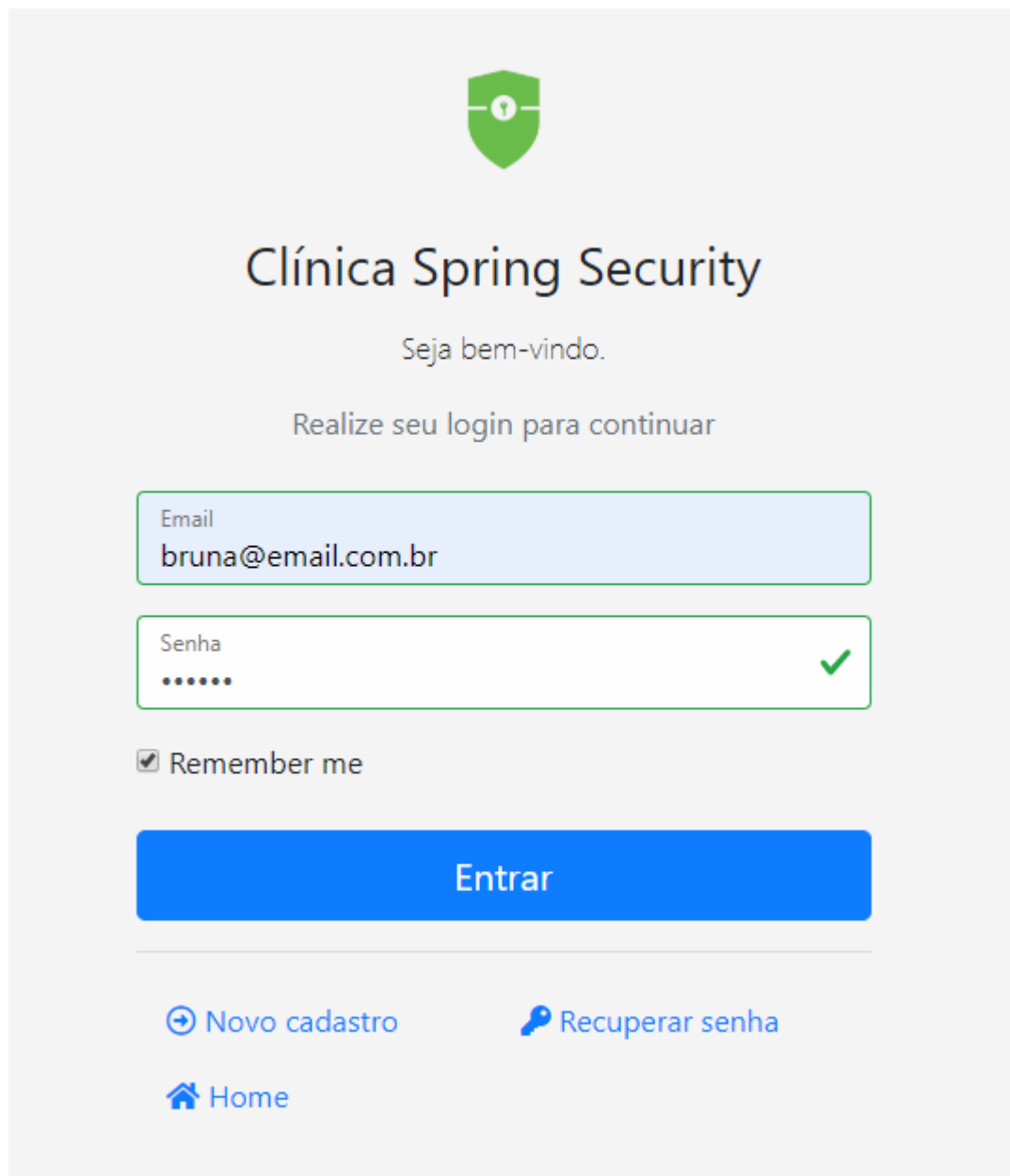
Em `helper` informamos também o e-mail do destinatário via método `setTo()` e o assunto do e-mail via `setSubject()`. E o envio de e-mail é realizado pelo método `send()`, de `JavaMailSender`, parametrizado com a variável `message`, de `MimeMessage`.

Remember-me

O sistema de *remember-me* é útil para manter o usuário logado no sistema mesmo que ele feche seu navegador ou desligue sua máquina. Porém, temos que levar em consideração que, se o usuário clicar no botão de

logout ele irá cancelar o *remember-me*, assim como, se o servidor de aplicação for reiniciado o *remember-me* também perderá a validade.

Mas como o *remember-me* funciona? Quando o usuário realiza o seu login com o checkbox selecionado o Spring Security vai gerar um *cookie* e salva-lo no cache do navegador. Esse *cookie* terá uma validade, que por padrão, são 14 dias e, o nome do cookie será *remember-me*.



The image shows a login page for 'Clínica Spring Security'. At the top is a green shield icon with a white keyhole. Below it, the title 'Clínica Spring Security' is centered in a large, dark font. Under the title, the text 'Seja bem-vindo.' is centered. Below that, the text 'Realize seu login para continuar' is centered. There are two input fields: 'Email' with the value 'bruna@email.com.br' and 'Senha' with masked characters '.....'. A green checkmark is visible to the right of the password field. Below the fields is a checkbox labeled 'Remember me' which is checked. A large blue button labeled 'Entrar' is centered below the checkbox. At the bottom, there are three links: 'Novo cadastro' with a plus icon, 'Recuperar senha' with a key icon, and 'Home' with a house icon.

Remember-me selecionado

A configuração de sistema junto ao Spring Security é muito simples e tem apenas dois passos. Um deles é adicionar na página de login o componente de checkbox para a opção de remember-me. Esse

componente deverá ter o atributo name setado com o valor remember-me o qual será aquele que o Spring Security irá procurar na requisição para saber se a opção foi ou não selecionada.

```
<div class="checkbox mb-3">
  <label>
    <input type="checkbox" name="remember-me"/>
    Remember me
  </label>
</div>
```

O próximo e último passo é ir até a classe de configuração das regras do Spring Security e adicionar a instrução `.rememberMe()`:

```
.and()
    .rememberMe()
```

Essa configuração vai então seguir aqueles padrões citados, 14 dias de validade do *cookie* e o nome deste *cookie* será `remember-me`. O *cookie* será criado com as seguintes especificações:

```
base64(
    username + ":" + expirationTime + ":" +
    md5Hex(username + ":" + expirationTime + ":" + password + ":" + key)
)
```

- **username**: identificador do nome do usuário na aplicação;
- **password**: senha do usuário para a operação de login;
- **expirationTime**: a data e hora que marca a validade do *cookie*. Esse tempo estará em milissegundos;
- **key**: uma chave privada mantida pelo Spring para validar a autenticidade deste *cookie*.

Os valores padrões como data de expiração do *cookie* e também seu nome podem ser alterados da seguinte forma:

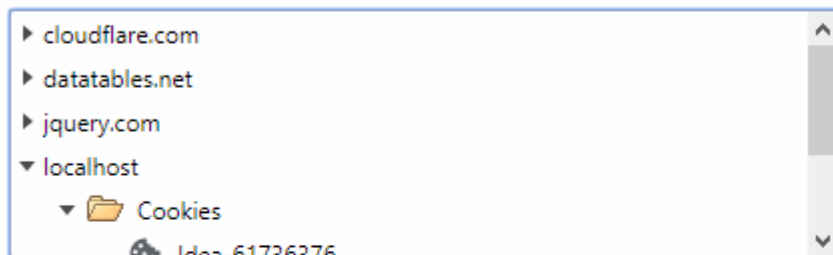
```
.and()
    .rememberMe()
    .rememberMeCookieName("clinica-spring-me")
    .tokenValiditySeconds(604800);
```

Cookies em uso

Permitido

Bloqueado

Os cookies a seguir foram definidos quando você visualizou esta página



Nome	clinica-spring-me
Conteúdo	k5ZDc4MzU0OTVjM2NhZDRINmYyM2FiODYxNQ
Domínio	localhost
Caminho	/
Enviar para	Qualquer tipo de conexão
Criado em	segunda-feira, 10 de junho de 2019 15:45:22
Expira em	segunda-feira, 17 de junho de 2019 15:45:22

Bloquear

Remover

Concluído

Cookie com nome

alterado e validade de 7 dias

No código apresentado o método `rememberMeCookieName()` modifica o nome do *cookie* armazenado no navegador. Já o método `tokenValiditySeconds()` recebe um valor em segundos que represente a quantidade de tempo desejado para a validade do *cookie*. No exemplo, `604800` equivale a 7 dias:

```
DIAS = TOTAL
1 = 86400
7 = 604800
10 = 864000
```

Também há uma configuração que pode forçar a criação do *cookie* mesmo que o checkbox não seja selecionado. Para isso, inclua na lista de métodos o `alwaysRemember()` setado com o valor `true`:


```
.and()  
    .rememberMe()  
    .alwaysRemember(true);
```

Referencias

- Spring Mail -
<https://docs.spring.io/spring/docs/5.1.7.RELEASE/spring-framework-reference/integration.html#mail>
- Spring Boot Properties -
<https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE/reference/htmlsingle/#appendix>
- Transport Layer Security (TLS) -
https://pt.wikipedia.org/wiki/Transport_Layer_Security
- Apache Velocity - <https://velocity.apache.org/>
- Thymeleaf e-mail template -
<https://www.thymeleaf.org/doc/articles/springmail.html>
- Remember-Me Authentication -
<https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#remember-me>