02

Integrando nossa nova lib com a aplicação principal

Transcrição

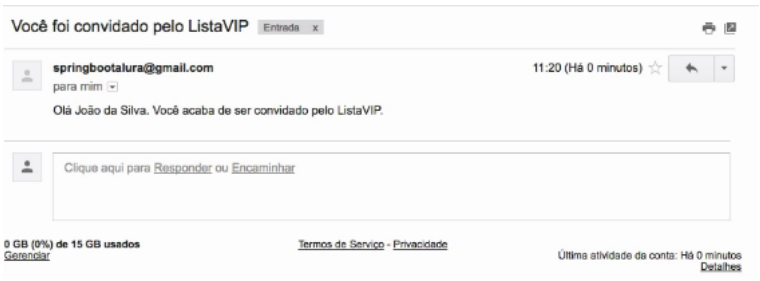
Agora que temos nossa biblioteca de envio de e-mail pronta, precisamos integrar a mesma com a aplicação Lista VIP. Fazemos isso adicionando-a como uma dependência do projeto no `pom.xml` .

```
<dependency>
    <groupId>br.com.alura.enviadorEmail</groupId>
    <artifactId>enviadorEmail</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Com isto teremos a biblioteca disponível e podemos enviar e-mails instanciando um objeto da classe `EmailService` e utilizando o método `enviar` . No método `salvar` da classe `ConvidadoController` após salvar o convidado no banco de dados, enviaremos o e-mail.

```
@RequestMapping(value= "salvar", method = RequestMethod.POST)
public String salvar(@RequestParam("nome") String nome, @RequestParam("email") String email,
                  @RequestParam("telefone") String telefone, Model model ){

    Convidado novoConvidado = new Convidado(nome, email, telefone);
    repository.save(novoConvidado);

    new EmailService().enviar(nome, email);

    Iterable<Convidado> convidados = repository.findAll();
    model.addAttribute("convidados", convidados);

    return "listaconvidados";
}
```

Desta forma, caso tenha configurado um servidor de email corretamente. Verifique, o e-mail deve ter sido enviado com sucesso.



Uma camada de serviço: Convidado

Por questões de boas práticas faremos uma pequena refatoração em nosso projeto. Observe que o *controller* de convidados está acessando diretamente o *repositório* de convidados, o que de certa forma pode não parecer problemático, mas não está dentro dos padrões de projetos adequados.

A solução adequada é que para obter e salvar convidados em nossa aplicação, não é acessar a base de dados diretamente do *controller*, mas sim por meio de um serviço. Criaremos então uma nova classe chamada `ConvidadoService`, no pacote `br.com.alura.service` e anotaremos esta classe com `@Service`.

Após isso, moveremos o código que recupera todos os convidados que está na classe `ConvidadoController` para um método que chamaremos de `obterTodos` nesta nova classe. Faremos o mesmo com a lógica de salvar o convidado. A classe `ConvidadoService` deverá ficar assim:

```
@Service
public class ConvidadoService {

    @Autowired
    private ConvidadoRepository repository;


    public Iterable<Convidado> obterTodos(){
        Iterable<Convidado> convidados = repository.findAll();
        return convidados;
    }

    public void salvar(Convidado convidado){
        repository.save(convidado);
    }
}
```

Lembre-se de que agora é a classe `ConvidadoService` que acessa o repositório. Por isso precisamos do objeto `ConvidadoRepository`. E lembre-se também de atualizar a classe `ConvidadoController` para refletir estas mudanças.

```
@Controller
public class ConvidadoController {

    @Autowired
    private ConvidadoService service;

    @RequestMapping("/")
    public String index(){
        return "index";
    }

    @RequestMapping("listaconvidados")
    public String listaConvidados(Model model){

        Iterable<Convidado> convidados = service.obterTodos();
        model.addAttribute("convidados", convidados);

        return "listaconvidados";
    }

    @RequestMapping(value= "salvar", method = RequestMethod.POST)
    public String salvar(@RequestParam("nome") String nome, @RequestParam("email") String email,
                         @RequestParam("telefone") String telefone, Model model ){

        Convidado novoConvidado = new Convidado(nome, email, telefone);
        service.salvar(novoConvidado);
```

```
        new EmailService().enviar(nome, email);

        Iterable<Convidado> convidados = service.obterTodos();
        model.addAttribute("convidados", convidados);

        return "listaconvidados";
    }
}
```