[Java Magazine 108 - Índice] [Versão para impressão]

Desenvolvendo com GAE e JSF 2

Praticidade e eficiência ao desenvolver e hospedar aplicações Web

De que se trata o artigo

O artigo apresenta a integração da tecnologia JavaServer Faces (JSF 2.0) com o Google APP Engine (GAE). Por meio do desenvolvimento de uma pequena aplicação serão demonstradas as configurações necessárias para o funcionamento dos componentes JSF no ambiente GAE, além de hospedar gratuitamente a aplicação nos servidores do Google. Por fim, é apresentado o mecanismo de armazenamento de dados utilizado pelo GAE.

Em que situação o tema é útil

Este tema é útil no processo de hospedagem e gerenciamento de aplicações web desenvolvidas a partir de componentes JSF, tirando do desenvolvedor a preocupação com hardware, proporcionando fácil desenvolvimento, implantação e escalabilidade.

Resumo Devman

Este artigo apresenta, por meio de uma visão geral, as vantagens e limitações do Google APP Engine (GAE), que é uma plataforma disponibilizada pelo Google para hospedar e gerenciar aplicações web de maneira gratuita. A integração da tecnologia JSF 2 com o GAE é demonstrada a partir da construção de uma aplicação web. Em relação à persistência, é apresentado o funcionamento do mecanismo de armazenamento de dados utilizado pelo Google APP Engine.

A Internet surgiu em plena Guerra Fria, mas foi somente em 1990, com a criação da World Wide Web (WWW), que ela alcançou uma maior popularização. Com o surgimento da World Wide Web tornou-se possível a utilização de uma interface gráfica e a criação de sites mais dinâmicos e visualmente mais interessantes. A partir deste momento, a Internet começou a

crescer em um ritmo acelerado.

A criação de sites antigamente se resumia basicamente à construção de páginas HTML, deixando o site com poucas funcionalidades interessantes e, em alguns casos, sem funcionalidades. Os serviços de hospedagem de sites possuíam muitas limitações de velocidade e ferramentas técnicas, além de um custo financeiro bastante elevado.

Atualmente, existem muitos recursos que auxiliam no desenvolvimento web. Entre eles está o JavaServer Faces (JSF), que é um framework para desenvolvimento web baseado em Java e se destina a facilitar o desenvolvimento de interfaces de usuário. Com o surgimento do conceito de computação em nuvem, a hospedagem desses aplicativos tornou-se menos complexa, pois utiliza servidores com grande capacidade de processamento e armazenamento de dados. Estes servidores, por sua vez, são interligados à Internet, tornando possível o acesso remoto a aplicativos, arquivos e serviços.

O Google APP Engine (GAE) é uma plataforma gratuita para hospedagem de aplicações web que utiliza o conceito de computação em nuvem. Essa é uma tecnologia recente, que proporciona um processo fácil de desenvolvimento e implantação, além de não ser necessário se preocupar com o hardware, já que os aplicativos serão hospedados em servidores oferecidos pelo Google.

Neste contexto, esse artigo possui o intuito de apresentar os conceitos relacionados ao Google APP Engine, bem como sua integração com o JavaServer Faces 2. A realização dessa integração será demonstrada por meio do desenvolvimento de uma aplicação web com as configurações necessárias para a sua hospedagem no GAE, além de utilizar o mecanismo de persistência do próprio Google APP Engine para o armazenamento de dados.

Visão geral do Google APP Engine

Google APP Engine é uma plataforma oferecida gratuitamente pelo Google que permite desenvolver e hospedar aplicações web nas linguagens Java e Python. Essa plataforma proporciona uma maneira prática e eficiente para criar e manter aplicações, sem se preocupar com o hardware.

A hospedagem das aplicações é gratuita até 500 MB de armazenamento, permitindo cinco milhões de visualizações de páginas por mês.

À medida que a aplicação for conquistando um maior número de visitas (público), é possível a contratação de um limite maior para armazenamento e visualização de páginas.

Esta plataforma oferece várias vantagens ao programador, tais como:

• Fácil escalabilidade das aplicações;
• Armazenamento de dados;
• Envio de e-mail;
• Memória cache de alto desempenho, que pode ser útil para dados temporários, isto é, que não serão armazenados em uma base de dados;
• Suporte a duas linguagens de programação: Java e Python;
• Suporte a Servlet, JSP, JSF, JPA, JDO e JavaMail;
• Suporte a manipulação de imagens, ou seja, é possível redimensionar, cortar, girar e inverter imagens nos formatos JPEG e PNG;
• Configurações do tempo de duração dos cookies;
• Logs de usuários.

Em contrapartida, algumas limitações precisam ser destacadas:

• Não suporta a escrita de arquivos;
• Não suporta EJB, RMI, Threads, JDBC, JMS e JNDI;

• Não suporta consultas SQL, pois não é oferecido um banco de dados relacional.

Para hospedar uma aplicação utilizando o Google APP Engine é necessário possuir uma conta no Google.

A criação de um espaço no GAE é efetuada no site http://code.google.com/intl/pt-BR/appengine/ clicando sobre o link Inscreva-se, lembrando que é possível hospedar até 10 aplicativos por conta no Google.

Na realização desse cadastro será necessário informar um título e um identificador para a aplicação que será hospedada. O identificador deverá conter somente letras minúsculas e sem caracteres especiais. Para verificar a validade do identificador basta clicar sobre o botão Check Availability. Este identificador deverá ser bem guardado, pois será utilizado para efetuar o deploy da aplicação. Feito isso, basta clicar no botão Create Application. Veja a Figura 1.

Para concluir o cadastro, deve ser informado um número de celular, o qual receberá um código para ativar a conta. A informação deste número será necessária uma única vez, ou seja, somente na realização do cadastro no Google APP Engine.

[abrir imagem em janela]

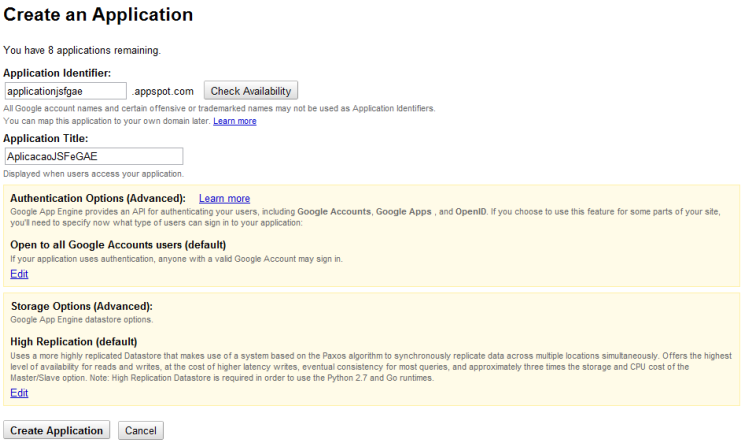


Figura 1. Criação da conta no Google APP Engine.

Criando uma aplicação no Google APP Engine

Para criar uma aplicação Google APP Engine será utilizada a IDE Eclipse Helios 3.6, que possui um plugin para o uso do GAE. No site http://code.google.com/intl/pt-BR/eclipse/docs/download.html está disponível um tutorial com os procedimentos para instalação do plugin para cada versão da IDE. Após a instalação do plugin, é disponibilizado um ícone do Google na barra de ferramentas do Eclipse. Veja a Figura 2.

[abrir imagem em janela]

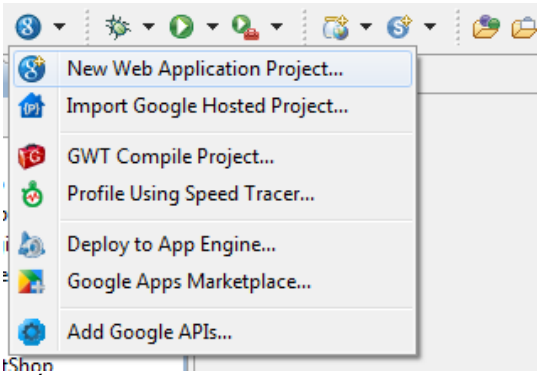


Figura 2. Ícone do Google na barra de ferramentas do Eclipse.

Para criar um novo projeto, clique sobre a opção New Web Application Project... e informe um nome e um pacote padrão para a aplicação. Ainda na mesma tela de criação do projeto, pode ser selecionada a opção Use Google Web Toolkit, caso deseje usar o Google Web Toolkit em seus aplicativos. Já a opção Generate project sample code pode ser desativada para não gerar o código de exemplo no projeto (Figura 3). Feito isso, clique em Finish. Ao finalizar a criação do projeto, é gerada uma estrutura semelhante à Figura 4.

[abrir imagem em janela]

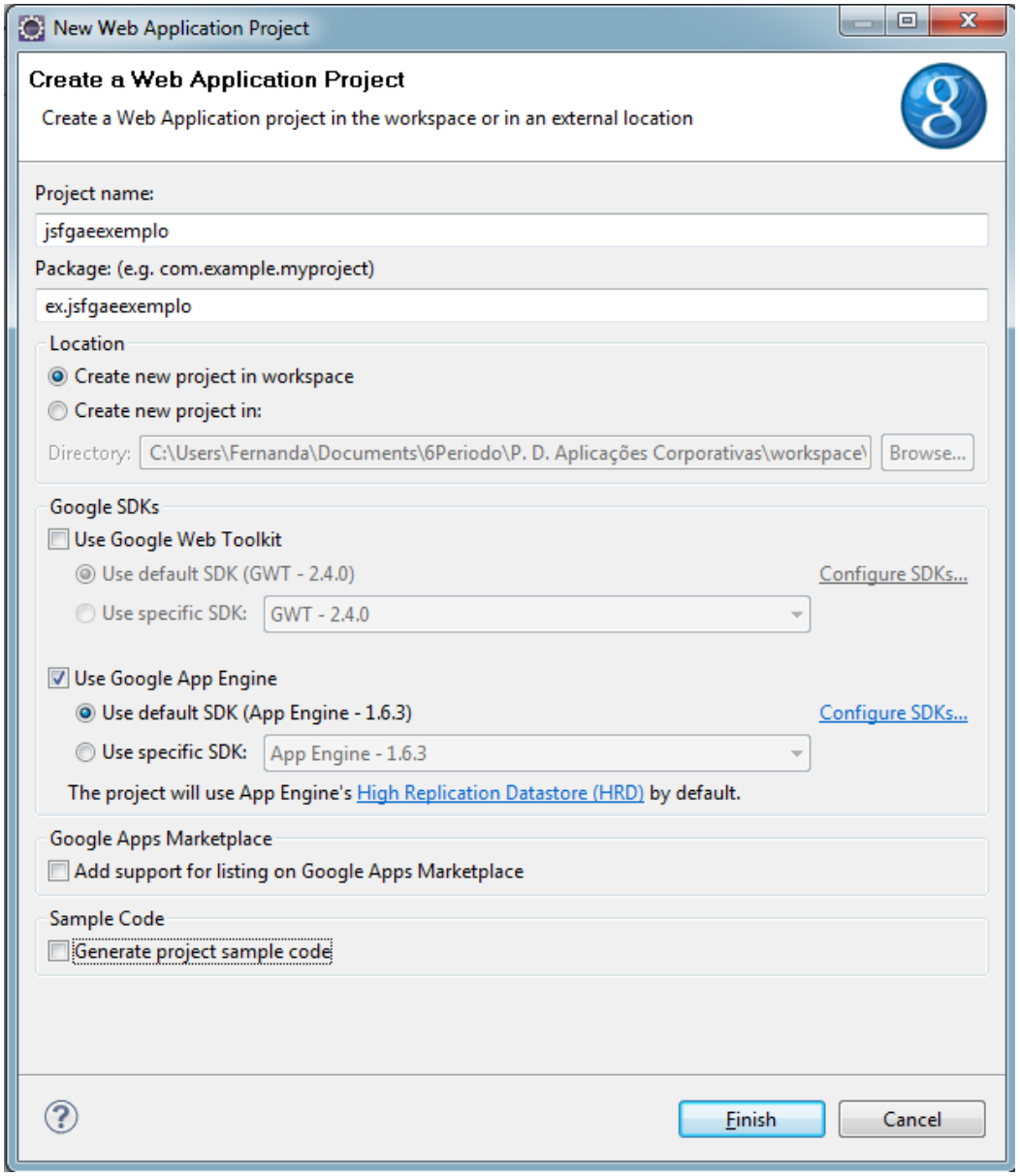


Figura 3. Criação do projeto.

[abrir imagem em janela]

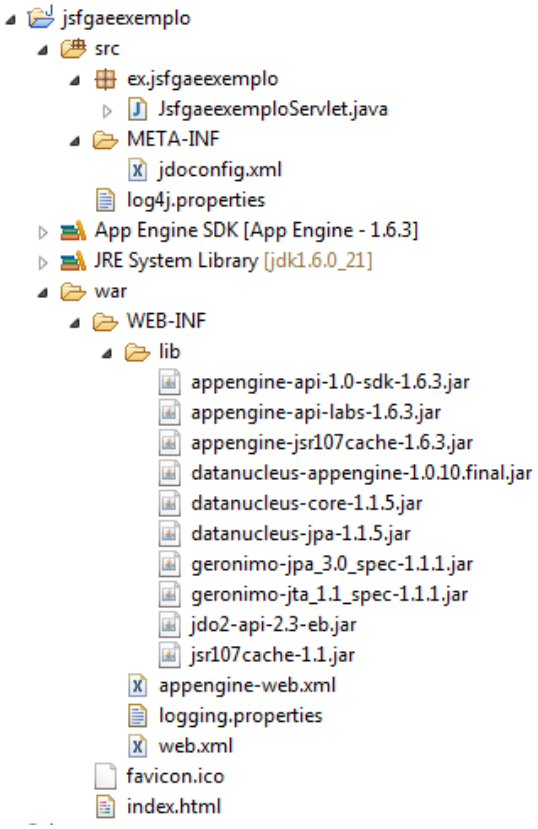


Figura 4. Estrutura do projeto criado.

Observe que as bibliotecas necessárias para a utilização do GAE já foram importadas. Também foi criada a página index.html, que possui um link para o servlet JsfgaeexemploServlet.java. Ao executar a aplicação e clicar sobre o link Jsfgaeexemplo, o servlet exibirá “Hello, World” como saída.

Para testar o funcionamento da aplicação, clique com o botão direito do mouse no nome do projeto e escolha a opção Run as > Web Application. Em seguida inicie o navegador de sua preferência e digite o endereço: http://localhost:8888/. O resultado deverá ser semelhante à Figura 5.

[abrir imagem em janela]

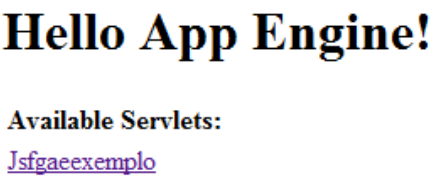


Figura 5. Resultado da aplicação ao executar a página index.xhtml.

Efetuando o deploy da aplicação no GAE

Para realizar o deploy da aplicação nos servidores do Google é preciso modificar o arquivo appengine-web.xml, que está localizado no diretório war/WEB-INF, atribuindo o nome informado ao identificador da aplicação no momento em que foi realizado o cadastro no Google App Engine. Esse arquivo de configuração é usado pelo GAE para implementar e executar a aplicação.

Na tag <application> deve ser informado o nome do identificador da aplicação, e na tag

<version> deve ser informado o identificador referente à versão mais recente do código do aplicativo.

Na tag <systemproperties> é definida uma propriedade para o armazenamento de logs do aplicativo usando a classe java.util.logging.Logger. No parâmetro value dessa propriedade é definido o arquivo logging.properties, que fará o controle do comportamento da classe Logger. No Eclipse, esse arquivo já vem configurado por padrão no diretório WEB-INF do projeto. As duas últimas tags também já vêm configuradas por padrão. Veja na Listagem 1 a configuração desse arquivo.

Antes de realizar o deploy ainda é necessário efetuar o login na conta Google de dentro do Eclipse.

Para isso, clique no botão Sign in to Google no canto inferior esquerdo da IDE. Veja a Figura 6.

Listagem 1. Configuração inicial do arquivo appengine-web.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>appjsfgae</application>
  <version>1</version>

  <system-properties>
    <property name="java.util.logging.config.file" value="WEB-
INF/logging.properties"/>
  </system-properties>

</appengine-web-app>
```

[abrir imagem em janela]

Figura 6. Efetuando login na conta Google por meio do Eclipse.

Depois de realizado o login é possível efetuar o deploy clicando com o botão direito do mouse no nome do projeto e, em seguida, Google > Deploy to App Engine. Em seguida aparecerá uma janela onde deverá ser selecionado o projeto que será publicado. Para finalizar, clique em Deploy e aguarde a finalização do processo.

Para verificar se o deploy foi realizado com sucesso, pode-se acessar o endereço: http://[identificador_aplicacao].appspot.com/.

Configurando o JavaServer Faces 2 no GAE

Uma vez que o foco do exemplo é demonstrar a integração entre JSF 2 e GAE, fazendo uso do mecanismo de persistência deste último, a aplicação desenvolvida e apresentada é simples, sendo composta unicamente por uma página com funcionalidades para manipulação de um cadastro de clientes. O acesso a este cadastro se dará por meio de um link oferecido na página principal da aplicação.

Para que seja possível utilizar os componentes do JSF 2 no GAE é necessário importar as bibliotecas do JSF 2 (jsf-api.jar e jsf-impl-gae.jar) e da Unified Expression Language (el-api-2.2.1-b04.jar e el-impl-2.2.1-b05.jar) para o classpath da aplicação.

Além de importar as bibliotecas descritas acima, o arquivo appengine-web.xml também deve ser modificado para que possa ser possível a implementação de sessões na aplicação. Este recurso está desativado por padrão. Para ativá-lo deve-se atribuir o valor true à tag <sessions-enabled>. Quando esse recurso for ativado, a implementação poderá armazenar os dados de sessão do usuário (login, e-mail e IP, por exemplo) no mecanismo de armazenamento de dados do GAE e na memória cache, proporcionando consistência na aplicação e maior velocidade.

Para realizar esse armazenamento, o mecanismo implementado pelo Google cria registros compostos pelos dados de sessão dos usuários e os armazena em uma tabela default chamada "_ah_SESSION" no banco de dados.

Por padrão, o engine da aplicação envia requisições de maneira serial para o servidor, porém, é possível alterar esta configuração para que envios simultâneos de várias requisições possam ocorrer. Para isso, no arquivo appengine-web.xml, a tag <threadsafe> deve ser configurada com o valor true. Com esta alteração, para requisições concorrentes, é preciso que sua aplicação esteja preparada para o uso correto de sincronização de threads.

A Listagem 2 apresenta o arquivo appengine-web.xml com as configurações necessárias.

Listagem 2. Arquivo appengine-web.xml configurado.

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>appjsfgae</application>
  <version>1</version>
  <sessions-enabled>true</sessions-enabled>

  <system-properties>
    <property name="java.util.logging.config.file" value="WEB-
INF/logging.properties"/>
  </system-properties>

  <threadsafe>true</threadsafe>

</appengine-web-app>
```

Os sistemas desenvolvidos em Java e projetados para executar por meio de um navegador, na Internet ou Intranet, utilizam um arquivo chamado web.xml que contém todas as informações do projeto. Esse arquivo é lido diretamente pelo servidor e determina como as URLs farão o mapeamento para os servlets e em quais URLs será necessária a autenticação, entre outras informações. Esse arquivo segue o mesmo formato no GAE.

O web.xml está localizado no diretório war/WEB-INF e também deve ser alterado para definir onde será realizado o controle de sessão da aplicação (cliente ou servidor) e para desativar a inicialização multithreading. Estas alterações são realizadas nos parâmetros:

• javax.faces.STATE_SAVING_METHOD: neste parâmetro é definido onde será realizado o controle de sessão. O valor a ser atribuído pode ser client ou server;

• com.sun.faces.enableThreading: este parâmetro desativa a inicialização multithreading do Mojarra, pois o Google APP Engine não suporta múltiplas threads.

Na Listagem 3 é apresentado o arquivo web.xml com as configurações supracitadas. Neste arquivo também devem ser informadas as configurações necessárias para o funcionamento do JSF 2. Entre tais configurações estão o mapeamento do Faces Servlet e a definição de sufixos, por meio do parâmetro javax.faces.DEFAULT_SUFFIX, que serão utilizados para ler as

páginas que possuem componentes JSF.

Listagem 3. Arquivo web.xml configurado.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      version="2.5">
<context-param>
 <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
 <param-value>client</param-value>
</context-param>
<context-param>
 <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
 <param-value>.xhtml</param-value>
</context-param>
<context-param>
 <param-name>com.sun.faces.enableThreading</param-name>
 <param-value>false</param-value>
</context-param>
<servlet>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
<welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

Desenvolvendo uma aplicação utilizando JSF 2 e GAE

Como ponto inicial para o desenvolvimento da aplicação exemplo, se faz necessária uma página de início, a qual é representada pelo arquivo index.jsp, substituindo o index.xhtml, padrão para o JSF 2. Essa página possuirá dois links: um para o cadastro de clientes e outro para a listagem dos clientes cadastrados. A página implementada por este arquivo é apresentada na Listagem 4.

Listagem 4. Código da página index.jsp.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <title>Google App Engine</title>
 </head>
 <body>
  <h1>Bem Vindo!</h1><br /><br />
  <a style="font-size: 18pt" href="cadastrar.jsf">Cadastrar Cliente</a><br />
```

```
  <a style="font-size: 18pt" href="listar.jsf">Listar Clientes</a>
 </body>
</html>
```

O cadastro de clientes será realizado pelas funcionalidades oferecidas pela página cadastrar.xhtml, que pode ser acessada por meio da página principal (index.jsp). Nessa página, existem controles para a informação do nome, endereço, telefone e dependentes e, para finalizar o processo, um botão, Enviar, que deve ser clicado para o envio dos dados informados. Para cada cliente será possível adicionar vários dependentes, o que pode ser realizado clicando sobre o botão Adicionar. O botão Enviar possui um método para que os dados informados pelo usuário sejam armazenados no banco de dados utilizando o mecanismo de persistência do GAE.
Essa página deve ser criada no diretório WAR do projeto.

Observe que essa tela foi desenvolvida utilizando componentes JSF. O JavaServer Faces é adicionado na página por meio da tag <html> nos parâmetros xmlns:f, xmlns:h e xmlns:ui. Os componentes principais da página são:

• <h:outputText>: representa um texto estático na tela;

• <h:inputText>: é utilizado para representar uma caixa de texto que receberá os dados (nome, endereço, telefone e dependentes) informados pelo usuário. Por meio do parâmetro value é indicado o objeto declarado na classe ClienteBean que receberá esses dados. Neste caso, é um objeto da classe Cliente que receberá o nome, o endereço e o telefone e um objeto da classe Dependente que receberá o nome dos dependentes do cliente a ser cadastrado. Como um cliente pode possuir mais de um dependente, na classe Cliente será declarada uma lista (List) de dependentes;

• <h:commandButton>: representa os botões na tela.

O primeiro botão é o Adicionar, que será responsável por adicionar vários dependentes de um cliente em uma lista (List) na classe ClienteBean, e o segundo botão é o Enviar, que será responsável por armazenar as informações (nome, endereço, telefone e os dependentes) no armazenamento de dados do GAE.

Por meio do parâmetro action do componente, o método que realizará tais funções é invocado. Nesse caso, temos os métodos cadastrarCliente() e cadastrarDependente() da classe ClienteBean. A implementação desses métodos será apresentada mais adiante.

Na Listagem 5 é exibido o código da página cadastrar.xhtml.

Listagem 5. Código da página cadastrar.xhtml.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xml:lang="en" lang="en"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head>
  <meta http-equiv="content-type" content="text/html;
   charset=UTF-8"/>
  <title>Cadastro de Cliente</title>
 </h:head>
 <h:body>
  <h1>Cadastro de Cliente</h1><br /><br />
  <h:form>
```

```
    <table>
     <tr>
      <td><h:outputText value="Nome: "/></td>
      <td><h:inputText value="#{clienteMB.cliente.nome}">
      </h:inputText></td>
     </tr>
     <tr>
      <td><h:outputText value="Endereco: "/></td>
      <td><h:inputText value="#{clienteMB.cliente.endereco}">
      </h:inputText></td>
     </tr>
     <tr>
      <td><h:outputText value="Telefone: "/></td>
      <td><h:inputText value="#{clienteMB.cliente.telefone}">
      </h:inputText></td>
     </tr>
     <tr>
      <td><h:outputText value="Depedente: "/></td>
      <td><h:inputText value="#{clienteMB.depen.nome}">
       </h:inputText></td>
      <td><h:commandButton action="#{clienteMB.cadastrarDependente}"
       value="Adicionar"></h:commandButton></td>
     </tr>
     <tr>
      <td><h:commandButton action="#{clienteMB.cadastrarCliente}"
       value="Enviar"/><td>
     </tr>
    </table>
   </h:form>
  </h:body>
 </html>
```

A página listar.xhtml também é acessada por meio da página principal (index.jsp), e nela serão listados todos os clientes cadastrados. De forma semelhante, o JSF também foi adicionado a essa página.

Note na Listagem 6 que foi utilizado o componente <h:dataTable> do JSF. Este componente representa uma tabela para listar os dados. A tag <h:dataTable> possui uma chamada ao método getClientes() da classe ClienteBean que é responsável por recuperar e apresentar na página todos os cadastros realizados em cadastrar.xhml.

Também foi usado o componente <h:column> para representar as colunas Nome, Endereço e Telefone da tabela. Como o método getClientes() retornará uma lista de clientes, será criada uma linha na tabela para cada registro da lista retornada. A implementação desse método será apresentada mais adiante. O componente <h:datatable> também foi utilizado para exibir os dependentes de cada cliente por meio do método getDependentes() da classe Cliente.

Listagem 6. Código da página listar.xhtml.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
```

```
  <title>Listar Clientes</title>
 </h:head>
 <h:body>
  <h1>Clientes Cadastrados</h1><br /><br />
  <h:form>
   <h:dataTable value="#{clienteMB.clientes}" var="c">
    <h:column>
     <f:facet name="header">Nome</f:facet>
      #{c.nome}
    </h:column>
    <h:column>
     <f:facet name="header">Endereço</f:facet>
      #{c.endereco}
    </h:column>
    <h:column>
     <f:facet name="header">Telefone</f:facet>
      #{c.telefone}
     </h:column>
     <h:column>
       <h:dataTable value="#{c.dependentes}" var="d">
        <h:column>
         #{d.nome}
        </h:column>
       </h:dataTable>
      </h:column>
    </h:dataTable>
   </h:form>
  </h:body>
</html>
```

Para fornecer objetos que serão manipulados pela classe ClienteBean, a aplicação também deve conter uma classe chamada Dependente com o atributo nome e outra classe chamada Cliente que possuirá os atributos nome, endereco, telefone e uma lista (List) de dependentes. Não há necessidade de qualquer tipo de anotação nessas classes, somente métodos construtores e getters/setters.

Geralmente, aplicações JSF utilizam classes que sejam Managed Bean (bean gerenciado) para receber as requisições vindas das páginas JSF e realizar as operações requisitadas, como por exemplo, salvar ou consultar os dados de um cliente na base de dados.
No nosso projeto, a classe ClienteBean será registrada como um bean gerenciado. Dessa maneira, todos os métodos públicos dessa classe poderão ser acessados por qualquer página JSF.

Para registrar a classe ClienteBean como um Managed Bean, devem ser adicionadas as anotações @ManagedBean, para ser possível gerenciar a comunicação entre os componentes JSF e as classes Java da aplicação, e @ApplicationScoped, para definir o tipo de escopo do Managed Bean. Deste modo, a classe ClienteBean estará disponível na memória desde o momento que o servidor iniciar a aplicação até o final da sua execução. Nessa classe será implementado o método cadastrarCliente(), que terá por função realizar o cadastro do cliente e retornar para o usuário a página principal da aplicação (index.jsp). A princípio, não será implementando o código para armazenamento no banco de dados. Isto será feito mais adiante.

Na Listagem 7 é apresentado o código de ClienteBean.

Listagem 7. Implementação da classe ClienteBean.

```
@ManagedBean(name="clienteMB")
@ApplicationScoped
public class ClienteBean{

   private Cliente cliente = new Cliente();
   private List<Cliente> clientes = new ArrayList<Cliente>();
   private List<Dependente> dependentes = new ArrayList<Dependente>();

   //métodos getters e setters omitidos

   public String cadastrarCliente(){
     System.out.println("Testando JSF 2 e GAE");
     return "index.jsp";
   }

   public void cadastrarDependente(){
      dependentes.add(depen);
      depen = new Dependentes();

   }

}
```

Após concluir todas as configurações, a aplicação estará pronta para ser publicada no Google APP Engine; lembrando que até o momento não foram implementados os mecanismos de persistência. Para efetuar o deploy, basta seguir os passos descritos na seção “Efetuando o deploy da aplicação no GAE” ou, também, executar a aplicação localmente acessando o endereço http://localhost:8888.

Mecanismo de armazenamento de dados do GAE

O Google APP Engine oferece um mecanismo de persistência de dados por meio do pacote com.google.appengine.api.datastore. Esse tipo de armazenamento é baseado no sistema BigTable, que não suporta o modelo relacional. No sistema BigTable uma tabela é representada por um mapa esparso, distribuído, persistente e multidimensional, organizado em três dimensões: linha, coluna e timestamp. Estas três dimensões formam uma célula. Baseado nesta estrutura, para um sistema de armazenamento BigTable, é possível a existência de várias versões de uma mesma célula. Estas versões podem ser armazenadas e indexadas pelo timestamp, permitindo que os registros sejam associados a uma data/hora.

Dessa maneira, é possível recuperar, por exemplo, a informação de quando um registro foi modificado pela última vez.

Os dados armazenados no Google APP Engine podem ser acessados por meio da linguagem Google Query Language (GQL), que é uma linguagem inspirada e bastante semelhante a SQL. Essa linguagem possui suporte a consultas e ordenação, porém não é possível realizar consultas utilizando joins, pois o modelo não é relacional.

Cada objeto armazenado no GAE é conhecido como entidade e cada entidade possui um identificador exclusivo (ID) que não pode ser alterado, semelhante à chave primária no banco de dados relacional.

No pacote com.google.appengine.api.datastore também é disponibilizada a classe Entity, que é responsável pela criação das entidades, sendo possível especificar em sua criação o tipo (numérico ou String) do identificador (ID). Para definir que o identificador será numérico, é

necessário fornecer como argumento do construtor apenas o nome que a entidade possuirá. Caso seja necessário definir o identificador como String, deve ser fornecido um segundo argumento no construtor da entidade, informando o seu nome. Nas Listagens 8 e 9 são demonstradas as criações das entidades com o identificador do tipo String e numérico, respectivamente.

Listagem 8. Criação da entidade cliente com identificador do tipo String.

```
Entity cliente = new Entity("cliente","codigo");
```

Listagem 9. Criação da entidade cliente com identificador numérico.

```
Entity cliente = new Entity("cliente");
```

O mecanismo de persistência do Google APP Engine ocorre por meio da interface DatastoreService. Esta interface fornece acesso síncrono a um sistema de armazenamento de dados sem esquema e utiliza como unidade fundamental a entidade. O objeto DatastoreService é criado a partir do método getDatastoreService() da classe DatastoreServiceFactory e, a partir desse objeto, é possível realizar operações como salvar, editar, excluir e consultar entidades na base de dados. Veja o exemplo de criação de um objeto DatastoreService na Listagem 10.

Listagem 10. Criação do objeto datastore.

```
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
```

Uma entidade (registro) é armazenada no GAE por meio do método put(). Este método recebe como parâmetro uma entidade já populada com os dados a serem armazenados no banco.

Na Listagem 11 é apresentado o código para salvar uma entidade.

Listagem 11. Salvando uma entidade cliente.

```
Entity cliente = new Entity(“cliente”);
cliente.setProperty(“nome”, ”Maria”);
cliente.setProperty(“endereco”, ”Av. Brasil”);
datastore.put(cliente);
```

O método put() também é utilizado para o caso de edição de uma entidade. Neste caso, primeiro é necessário obter a entidade e depois alterar seus dados. Ao executar o método put(), se o identificador da entidade já existir, seus dados serão atualizados.

O método delete(), por sua vez, é executado para excluir uma entidade. Ele apenas recebe como parâmetro o identificador da entidade a ser excluída. Veja um exemplo na Listagem 12.

Listagem 12. Excluindo uma entidade.

```
datastore.delete(123);
```

Para efetuar consultas, é utilizada a classe Query para construir as consultas e a classe PreparedQuery para buscar e retornar as informações da base de dados.

Na Listagem 13 é apresentada a consulta que retorna todos os registros da tabela cliente que possuem o nome “Maria”.

Listagem 13. Efetuando uma consulta na base de dados.

```
Query q = new Query(“cliente”);
```

```
q.addFilter(“nome”, Query.FilterOperator.EQUAL, “Maria”);
PreparedQuery pq = datastore.prepare(q);
for(Entity clientes : pq.asIterable()){
cliente.setNome((String) clientes.getProperty(“nome”));
cliente.setEndereco((String) clientes.getProperty(“endereco”));
cliente.setTelefone((String) clientes.getProperty(“telefone”));
}
```

Na Listagem 13, note que foi adicionado um filtro por meio do método addFilter() da classe Query, visando recuperar do banco de dados somente as entidades que possuem o nome “Maria” no atributo nome. São vários os tipos de filtros disponíveis para realizar consultas, facilitando a busca por registros armazenados no GAE, a saber:

• Query.FilterOperator.LESS_THAN;
• Query.FilterOperator.LESS_THAN_OR_EQUAL;
• Query.FilterOperator.EQUAL;
• Query.FilterOperator.GREATER_THAN;
• Query.FilterOperator.GREATER_THAN_OR_EQUAL;
• Query.FilterOperator.NOT_EQUAL;
• Query.FilterOperator.IN.

A consulta é executada a partir do método prepare() da interface DatastoreService, passando o objeto q (instância da classe Query).

O mecanismo de persistência do GAE não suporta associações.

Dessa maneira os relacionamentos devem ser implementados diretamente na aplicação. Para implementar um exemplo de associação, uma nova classe será necessária, a classe Dependente. Esta classe possui apenas a declaração do atributo nome, o método construtor e os métodos getters/setters. Cada objeto desta classe poderá se associar (neste caso) a apenas um objeto da classe Cliente. Mapeando para o mecanismo de persistência do GAE, tem-se as tabelas cliente e dependente e considera-se que um cliente pode possuir vários dependentes. Para realizar a associação entre as tabelas cliente e dependente, primeiramente deve ser persistido o objeto cliente com nome, endereço e telefone no banco de dados e, depois, os objetos dependente que representam os dependentes de cada cliente. Cada objeto dependente deve possuir, além do nome, o id do objeto cliente cadastrado. Note como é realizada a associação entre as tabelas cliente e dependente na Listagem 14.

Listagem 14. Associação entre as tabelas cliente e dependente.

```
public String cadastrarCliente(Cliente cliente){

 //criando o objeto datastore
 DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
 //criando a entidade cli
 Entity cli = new Entity("cliente");
 //populando a entidade cli com os dados informados pelo usuário
 cli.setProperty("nome", cliente.getNome());
 cli.setProperty("endereco", cliente.getEndereco());
 cli.setProperty("telefone", cliente.getTelefone());
 //salvando a entidade cli
 datastore.put(cli);

 //percorrendo todos os dependentes adicionados a esse cliente pelo usuário
 for (int i = 0; i < dependentes.size(); i++){
  //criando a entidade dependente
```

```
  Entity dependente = new Entity("dependente");
  //populando a entidade dependente, ou seja, para cada objeto dependente
  //é adicionado um nome e o id do cliente cadastrado anteriormente
  dependente.setProperty("nome", dependentes.get(i).getNome());
  dependente.setProperty("idCliente", cli.getKey());
  //salvando cada entidade dependente
  datastore.put(dependente);
 }
 //limpando a lista de dependentes para que possa ser utilizada novamente
 dependentes.clear();
 //retornando para a página principal
 return "index.jsp;
}
```

Adicionando o mecanismo de persistência do GAE na aplicação

Até o momento os métodos cadastrarCliente() e getClientes() da classe ClienteBean não foram implementados para suportar o armazenamento de dados oferecido pelo GAE. Para que isso seja possível, o método cadastrarCliente() precisa ser alterado. Neste método deve ser criado e instanciado um objeto datastore, que por meio do método put(), armazenará os dados (nome, endereço, telefone e dependentes) do cliente. Na Listagem 15 é apresentado o código desse método.

Listagem 15. Alterando o método cadastrarCliente().

```
 public String cadastrarCliente(Cliente cliente){

//criando o objeto datastore
      DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
//criando a entidade cli
Entity cli = new Entity("cliente");
//populando a entidade cli com os dados informados pelo usuário
cli.setProperty("nome", cliente.getNome());
cli.setProperty("endereco", cliente.getEndereco());
cli.setProperty("telefone", cliente.getTelefone());
//salvando a entidade cli
datastore.put(cli);

//percorrendo todos os dependentes adicionados a esse cliente pelo usuário
for (int i = 0; i < dependentes.size(); i++){
//criando a entidade dependente
Entity dependente = new Entity("dependente");
//populando a entidade dependente, ou seja, para cada objeto dependente é adicionado
//um nome e o id do cliente cadastrado anteriormente
dependente.setProperty("nome", dependentes.get(i).getNome());
dependente.setProperty("idCliente", cli.getKey());
//salvando cada entidade dependente
datastore.put(dependente);
}
//limpando a lista de dependentes para que possa ser utilizada novamente
dependentes.clear();
//retornando para a página principal
      return "index.jsp;
}
```

O método getClientes(), também da classe ClienteBean, pode ser alterado de maneira que exiba na página listar.xhtml todos os clientes cadastrados. Na Listagem 16 é exibido o método

com as alterações necessárias.

Listagem 16. Alterando o método getClientes().

```
public List<Cliente> getClientes(){

 //criando o objeto datastore
 DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
 //construindo a consulta, que retornará todos
 //os registros da entidade cliente
 Query q = new Query("cliente");
 //executando a consulta
 PreparedQuery pq = datastore.prepare(q);
 //criando uma lista para receber todos os clientes retornados pela consulta
 clientes = new ArrayList<Cliente>();

 //para cada entidade será criado um objeto cliente
 for(Entity c : pq.asIterable()){
  //criando um objeto cliente
  Cliente cli = new Cliente();
  //populando o objeto cliente com os dados da entidade
  cli.setId(c.getKey());
  cli.setNome((String) c.getProperty("nome"));
  cli.setEndereco((String) c.getProperty("endereco"));
  cli.setTelefone((String) c.getProperty("telefone"));
  dependentes.clear();
  //criando uma consulta para retornar os dependentes de cada cliente
  Query qy = new Query("dependente");
  //filtro para buscar os dependentes pelo ID do cliente
  qy.addFilter("idCliente", Query.FilterOperator.EQUAL, c.getKey());
  //executando a consulta
  PreparedQuery pqy = datastore.prepare(qy);

  //percorrendo todos os dependentes do cliente
  for(Entity d : pqy.asIterable()){
    //para cada dependente é criado um objeto dep
    Dependentes dep = new Dependentes();
    //populando o objeto dep com os dados retornados da consulta
    dep.setNome((String) d.getProperty("nome"));
    //adicionando o objeto dep na lista de dependentes
    dependentes.add(dep);
  }
  //adicionando a lista de dependentes no objeto cliente
  cli.setDependentes(dependentes);
  //adicionando o objeto cliente em uma lista de clientes
  clientes.add(cli);
 }

 //retornando a lista de clientes com os
 //dependentes para a página listar.xhtml
 return clientes;
}
```

Depois de alterar o método getClientes(), a aplicação estará pronta para ser hospedada no Google APP Engine. O deploy do projeto é efetuado seguindo os mesmos procedimentos descritos no tópico “Efetuando o deploy da aplicação no GAE”. O resultado da execução será semelhante às Figuras 7, 8 e 9.

[abrir imagem em janela]

Figura 7. Execução da página index.jsp.

[abrir imagem em janela]

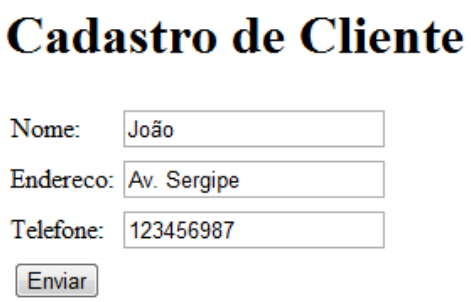


Figura 8. Execução da página cadastrar.xhtml.

[abrir imagem em janela]

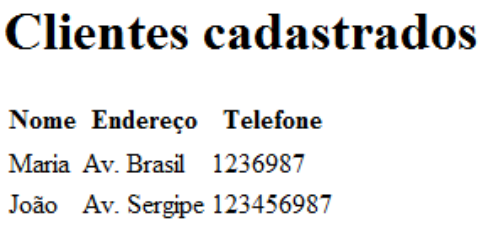


Figura 9. Execução da página listar.xhtml.

Conclusão

Neste artigo foi apresentada a maneira como é realizada a integração entre as tecnologias JSF 2 e Google APP Engine, e o mecanismo de persistência de dados utilizado pelo GAE.

A integração entre essas tecnologias proporciona aos desenvolvedores de webapps praticidade e agilidade na implementação, já que usa os componentes JSF 2 juntamente com a estrutura de projetos do Google. Além disso, é possível fazer uso da infraestrutura que o Google oferece para hospedar e manter os aplicativos gratuitamente. As aplicações hospedadas nos servidores do Google também contam com um mecanismo de armazenamento de dados escalável e robusto.

Apesar de não suportar o modelo relacional, como foi possível notar, a persistência de dados é bastante simples.

O Google APP Engine oferece suporte para pequenas e médias aplicações, ou seja, é ideal para os desenvolvedores que estão começando no mercado de trabalho. Ao hospedar um aplicativo no APP Engine é possível compartilhá-lo imediatamente com outros usuários,

possibilitando a divulgação e utilização de sistemas web logo após sua publicação.

EVERTON COIMBRA DE ARAÚJO

https://www.facebook.com/evertoncoimbradearaujo Desde 1987 atua na área de treinamento e desenvolvimento. É mestre em Ciência da Computação, e professor efetivo da UTFPR, Campus Medianeira. Tem Atuado também em cursos de EaD oferecidos pela UAB e ETEC. Autor de livros pela Visual Books: http://www.visualbooks.com.br/shop/MostraAutor.asp?proc=191

Fernanda Cristina Girelli

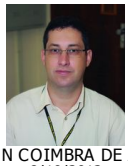
Graduada em Análise e Desenvolvimento de Sistemas pela Universidade Tecnológica Federal do Paraná (UTFPR). Trabalha como Analista de Suporte Técnico na SWA Sistemas para Gestão de Ensino, empresa especializada na produção de softwares para gerenciamento de instituições de ensino.

CLIQUE AQUI E INCLUA SEU COMENTÁRIO

Gustavo Lira e Silva
5/10/2012

Kd as classes Cliente e Dependente que não esta no artigo?

Responder



EVERTON COIMBRA DE ARAÚJO
8/10/2012

Olá Gustavo. As classes não estão listadas no artigo, mas fazem parte do projeto, que está disponível para download.

Responder

LUCAS LEMOS COSTA
30/11/2012

A Aplicação ta rolando, gostaria de melhorá-la, os senhores sabem se o GAE suporta JSF 2.0, facelets e prime faces?
Valeu !!

Responder

Diogo Souza
30/11/2012

Sim, perfeitamente..

Dá uma olhada nesse link:
http://consultingblogs.emc.com/jaddy/archive/2009/11/20/jsf2-in-google-app-engine.aspx

Responder