**Giving Your Blog to the World**

**Goals:** In this lab, we will be adding code to allow us to better visualize our blog.

**Instructions:**

In the lab 3 (Keeping Things In Check), you created the controllers to perform actions on your application on the web. Now you will write the code to actually get your blog displayed in your browser how you want it! Feel free at anytime to navigate to localhost:8080/ZynxBlog/post to see how your blog is looking.

Create a new file called list.gsp in the grails-app/views/post folder. Add the following code to the file:

<%@ page contentType="text/html;charset=UTF-8" %>

<html>

<head>

<title>My Posts</title>

</head>

<body>

<h1>My Posts</h1>

<g:each in="${posts}" var="post">

<div>

<h2>${post.title}</h2>

<p>${post.teaser}</p>

<p>Last Updated: ${post.lastUpdated}</p>

</div>

</g:each>

</body>

</html>

|  |
| --- |
|  |

Groovy Sever Pages is a presentation language for web applications. Therefore, this code tells the website how to display the list of posts. More technically, the code goes through all the posts and displays their title and their teaser text. Also, note the tag “g:each.” These are specifically Grails tags, as noted by the g: in front of the keyword. We will use more of these later in the lab.

You are almost ready to display your posts now. The next step is to load some test data when you start the application to verify that you can see the posts. Grails provides a BootStrap Groovy class in the grails-app/conf directory. This class has an init closure that is executed when the application starts up. If you put some initialization data in this closure it will be available to your application. The following code shows how to use the Grails BootStrap class to load data into the application before start up:

import groovypublish.Post

class BootStrap {

def init = { servletContext ->

new Post(title:"First Grails Project",

teaser:"Clearing out the clutter",

content:"The full content of the article",

published:true).save()

new Post(title:"Getting Ready for Go Live",

teaser:"The follow up on some real world issues.",

content:"The full content of the article",

published:false).save()

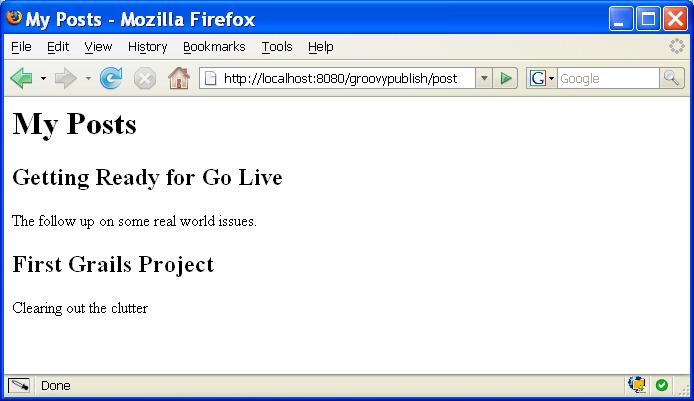
}

def destroy = {

}

}

Restart the running Grails application, go to the post index page at http://localhost:8080/groovypublish/post, and you can see the two posts you have created, listed in the order you created them in the bootstrap.

[](javascript:showSupportItem('figure2');)

|  |
| --- |
|  |
|  |

**Create a New Post**  
To allow users to create a post, you need to provide a link from the page with the lists of posts, create a GSP input form, and create a new action to save the data to your database. So far, you have used your first Grails tag (in the list.gsp) to create a link to the edit Post action (by default, Grails tags are defined in the g namespace). Now define the controller and the action that the link should send the user to as follows:

<g:link controller="post" action="edit">

Create a new post

</g:link>

Remember that you have already defined the edit action. What you don't have is an edit page to render the form that allows posts to be created or edited. Grails provides a number of tag libraries for rendering forms, input fields, and validation errors that occur from a form submission:

* The <g:renderErrors> tag displays the validation errors from your submission as an HTML list.
* The <g:form> tag renders an HTML form that submits—using the post method by default—the data to the specified action on the controller.
* The <g:textArea> and <g:textField> tags render the HTML for a text area input field and a text input field, respectively.

Below is the code for the Groovy server page that allows users to edit or create a Post. The edit.gsp code needs to go in the same location as your list.gsp: grails-app/views/post.

<%@ page contentType="text/html;charset=UTF-8" %>

<html>

<head>

<title>Manage Post</title>

</head>

<body>

<h1>Edit your post</h1>

<div id="validationerrors">

<g:renderErrors bean="${post}"/>

</div>

<g:form controller="post" action="save">

<g:hiddenField name="id" value="${post.id}"/>

<dl>

<dt>Post Title:</dt>

<dd>

<g:textField name="title" value="${post.title}" size="50"/>

</dd>

<dt>Post Teaser:</dt>

<dd>

<g:textArea name="teaser" value="${post.teaser}" cols="50"/>

</dd>

<dt>Post:</dt>

<dd>

<g:textArea name="content" value="${post.content}" rows="20" cols="50"/>

</dd>

</dl>

<g:submitButton name="submit" value="Save"/>

</g:form>

</body>

</html>

You can see from the <g:form> tag in above that you need to create an action called save on the PostController to handle the form submission:

def save = {

def post = loadPost(params.id)

post.properties = params

if(post.save()) {

redirect(action:'list')

} else {

render(view:'edit', model:[post:post])

}

}

private loadPost(id) {

def post = new Post();

if(id) {

post = Post.get(id)

}

return post

}

Let’s think logically about this for a moment. To enable the save action for use during both creating a new post and updating an existing post, the first thing you do is load the post. Once you have the post, you need to update the properties so it can be saved. Grails provides all the values you submitted from the form as a Map called params and also provides a properties property on each domain object to expose all the properties of the object as named values in a Map. This allows you to set all the values you have sent from the form directly onto the domain object by assigning the request params to the domain objects properties field.

The validation is performed when you try to save the post. If the validation succeeds, you send the user back to the list page to view the post. If it fails, you render the edit page again and put the updated post object in the model.

*Edit a Post*

To allow a user to edit a post, add a link to the list.gsp file for Post, within the g:each and div tags:

<g:link controller="post" action="edit" id="${post.id}">

Edit this post

</g:link>

This is almost the same as the link to create a new post, but in this case you specify the identifier of the post. This will allow the edit action to load the post to be edited from the database and display it.

*View a Post*  
You now can allow users to view the full text of a post. Create a view action on the PostController:

def view = {

render(view:'view', model:[post:Post.get(params.id)])

}

Next, create a view.gsp in the grails-app/views/post directory:

<%@ page contentType="text/html;charset=UTF-8" %>

<html>

<head>

<title>${post.title}</title>

</head>

<body>

<h1>${post.title}</h1>

<p>${post.teaser}</p>

<div>${post.content}</div>

</body>

</html>

This listing shows the Groovy server page used to render the details of a post.

Finally, add the link to the post list.gsp file to allow a post to be viewed:

<g:link controller="post" action="view" id="${post.id}">

View this post

</g:link>