GRAILS TUTORIAL

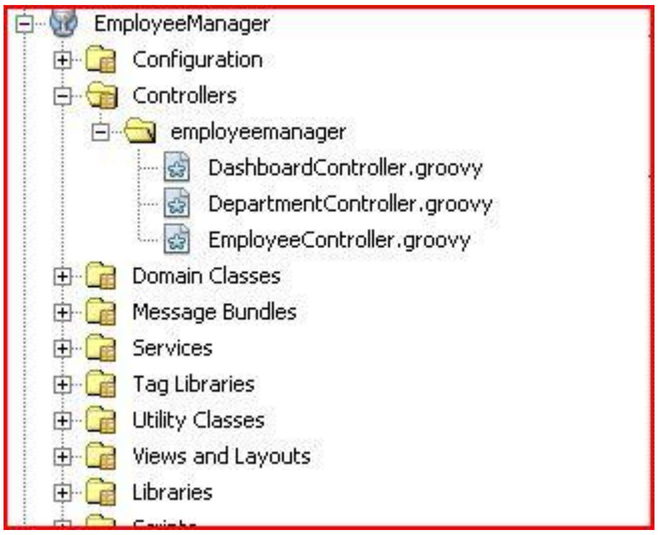
Assoc Prof Kaushik Dutta
Department of Information Systems
School Computing
National University of Singapore

Creating a controller and view:

So far, we have created a domain class and then used the generate-all command to create the controller classes. Today we will look at how to create a controller and view from scratch. For our EmployeeManager application, we will create a simple dashboard for the administrator to be able to perform all the actions from a single page.

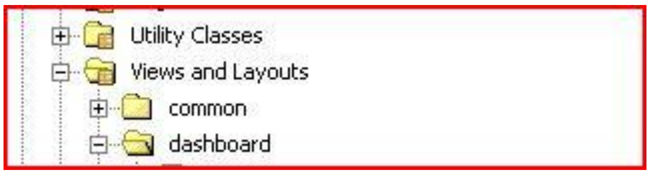
STEPS:

Create a controller first – right click on the controllers folder in the EmployeeManager Application and create a new controller called **Dashboard**



```
EmployeeManager
  Configuration
  Controllers
    employeemanager
      DashboardController.groovy
      DepartmentController.groovy
      EmployeeController.groovy
  Domain Classes
  Message Bundles
  Services
  Tag Libraries
  Utility Classes
  Views and Layouts
  Libraries
```

When this is created, we will see that a folder called dashboard has been created in the Views and layouts folder.



```
  Utility Classes
  Views and Layouts
    common
    dashboard
```

Open the file DashboardController.groovy and copy the following code into it

In this case, you can see that this controller was added to give administrators a easier way to manage all the departments from a single page. One can add/edit/delete and view all the departments from here. (When you are building a more complex application, you might want to restrict the access of this dashboard to just the administrators or other users with a role higher up in the hierarchy using the various security mechanisms available).

DashboardController.groovy

```
package employeemanager

class DashboardController {

    def index() { }

//to list the departments
   def deptlist = {
           params.max = Math.min(params.max ? params.int('max') : 10, 100)
        [deptInstanceList: Department.list(params), deptInstanceTotal: Department.count()]
    }
//to edit department
    def deptEdit = {
        def deptInstance = Department.get(params.id)
        if (!deptInstance) {
          flash.message = "Department not found"
            redirect(action: "deptlist")
        }
        else {
            return [deptInstance: deptInstance,deptInstanceTotal: Department.count(),deptInstanceList:
Department.list(params)]
        }
    }

//to save department
        def saveDept = {
        def deptInstance = new Department(params)
        String result= Department.findByName(params.name)
        if (params.name == ''){
            flash.message = "Please enter name of the department."
            redirect (action: "deptlist")
        }
        else if(result==null && deptInstance.save(flush: true)) {
            flash.message = "Department created"
            redirect(action: "deptlist")
        }
        else {
            flash.message = "Department ${params.name} already exists!"
            redirect(action: "deptlist")
        }

    }

//to update department
    def updateDept = {
        def deptInstance = Department.get(params.id)
        if (params.name == ''){
            flash.message= "No input detected."
            redirect (action:"deptlist")
```

```
      }

      else if (deptInstance) {
         String result= Department.findByName(params.name)
         if (result==null) {
            deptInstance.properties = params
            if (!deptInstance.hasErrors() && deptInstance.save(flush: true)) {
               flash.message = "Department'${params.name}' is updated."
               redirect(action: "deptlist")
            }
         }
         else {
            flash.message = "Department ${params.name} already exists!"
            redirect(action: "deptlist")
         }
      }
      else {
         flash.message = "No such department found"
         redirect(action: "deptlist")
      }
   }

//to delete department
   def deleteDepartment = {
      def deptInstance = Department.get(params.id)
      def name = deptInstance.name
      try{
         deptInstance.delete()
         flash.message = "Department '${name}' deleted."
         redirect(controller:"dashboard", action:"deptlist")
      }
      catch (Exception e) {
         flash.message = "Department'${name}' could not be deleted."
         redirect(controller:"dashboard", action:"deptlist")
      }
   }
}
```

In the above code, we are adding various actions to perform the different tasks.

Note: observe how ‘redirect’ is used to navigate to different actions in the controller. By default, if a controller is not specified, the current controller is assumed to contain the action. Also, when an action completes, in the absence of the ‘redirect’ statement, it will look to return the data to a view that has the same name as the action. Before we move on, let us work on some views for this controller class.

Step 2: Right click on the dashboard folder under the views and layouts section, and create a new gsp file called deptlist. As mentioned earlier, the actions in the controller return the values to a view of the same name unless explicitly specified otherwise. Open deptlist.gsp and copy the following code. (same goes for deptEdit)

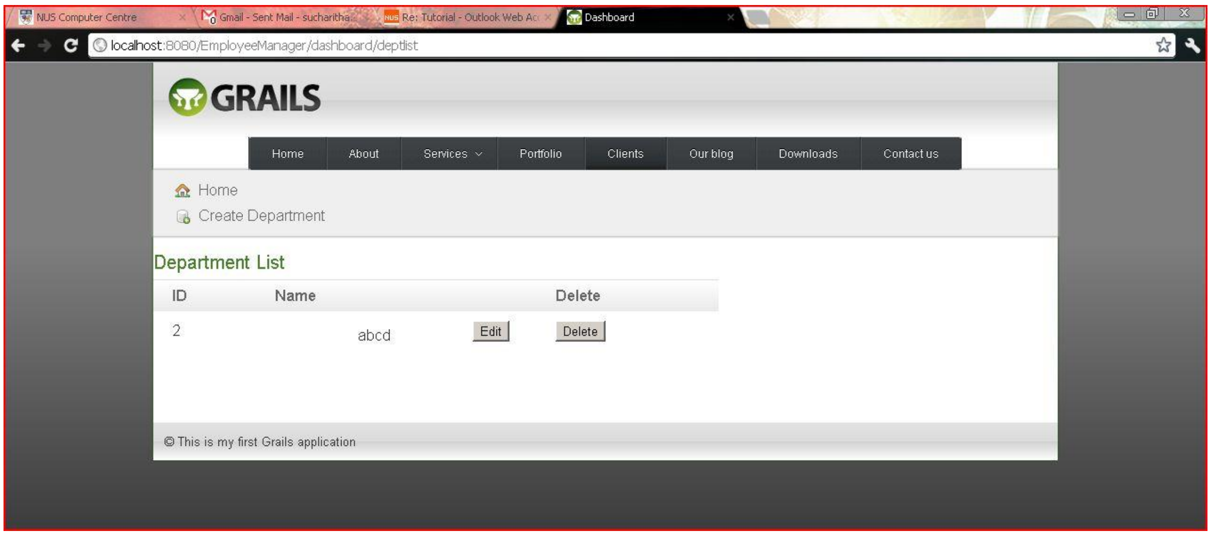
deptlist.gsp

```
<html>
 <head>
  <title>Dashboard</title>
  <meta name="layout" content="main" />
 </head>
 <body>
  <div class="nav">
  <span class="menuButton">
  <a class="home" href="${resource(dir:'')}">Home</a>
  </span>
  <span class="menuButton">
  <g:link class="create" controller="department" action="create">
  Create Department
  </g:link>
  </div>
  </span>
  <h1>Department List</h1>
  <g:if test="${flash.message}">
         <div class="message">${flash.message}</div>
         </g:if>
         <g:hasErrors bean="${deptInstance}">
          <div class="errors">
           <g:renderErrors bean="${deptInstance}" as="deptlist" />
          </div>
         </g:hasErrors>
  <div class="list">
  <table style =" width: 600px;">
  <thead>
   <tr>
    <g:sortableColumn property="id" title="ID" />
    <g:sortableColumn property="name" title="Name" />
    <g:sortableColumn property="Delete" title="Delete"/>
   </tr>
   </thead>
    <tbody>
     <g:each in="${deptInstanceList}" status="i" var="deptInstance">
     <tr>
      <td>${fieldValue(bean: deptInstance, field: "id")}</td>
      <td style =" width:280px;">
       <div style ="text-align: center; width: 210px; float:left; margin-top:5px;">
        ${fieldValue(bean: deptInstance, field: "name")}
       </div>
       <g:link controller="dashboard" action="deptEdit" id="${deptInstance.id}">
        <button type="button" >Edit</button>
       </g:link>
       <g:form method="post">
        <g:hiddenField name="id" value="${deptInstance?.id}" />
       </g:form>
      </td>
```

```
          <td>
            <g:link onclick= "return confirm('Are you sure?');" controller="dashboard"
action="deleteDepartment" id="${deptInstance.id}">
              <button type="button" >Delete</button>
            </g:link>
          </td>
         </tr>
        </g:each>
       </tbody>
      </table>
    </div>
   <br>
   <br>
  </body>
</html>
```

Here, as it can be seen, a simple menu has been created which will allow the admin to navigate back to the home page and also create a new department apart from being able to view the list of departments present with the options to edit and delete each. Go to this link to view the changes you have made

http://localhost:8080/EmployeeManager/dashboard/deptlist


NUS Computer Centre
Gmail - Sent Mail - sucharitha
Re: Tutorial - Outlook Web Acc
Dashboard
localhost:8080/EmployeeManager/dashboard/deptlist
GRAILS
Home
About
Services
Portfolio
Clients
Our blog
Downloads
Contact us
Home
Create Department
Edit
Delete
© This is my first Grails application

Department List

ID	Name		Delete
2	abcd	Edit	Delete

Similarly, we create another file called deptEdit.gsp to allow the edit functionality. Create deptEdit.gsp in the dashboard folder under views and layouts. Copy paste the following code into it

deptEdit.gsp

```
<html>
  <head>
    <title> DASHBOARD </title>
        <meta name="layout" content="main" />
    </head>
<h3>EDIT DEPARTMENT</h3>
 <g:if test="${flash.message}">
```

```
              <div class="message">${flash.message}</div>
              </g:if>
              <g:hasErrors bean="${deptInstance}">
                <div class="errors">
                  <g:renderErrors bean="${deptInstance}" as="deptlist" />
                </div>
              </g:hasErrors>
<body>
<g:form method="post" >
      <g:hiddenField name="id" value="${deptInstance?.id}" />
      <div class="dialog">
        <table>
          <tbody>
            <tr class="prop">
              <td valign="top" class="name">
                <label for="name"><g:message default="Name" /></label>
              </td>
              <td valign="top" class="value ${hasErrors(bean: deptInstance, field: 'name', 'errors')}">
          <g:textField name="name" value="${deptInstance?.name}" />
          <span class="button"><g:actionSubmit class="saveDept" action="updateDept"
value="${message(code:'default.button.update.label' ,default: 'Update')}" /></span>

          </td>
          </tr>
          </tbody>
        </table>
      </div>
    </g:form>
<div class="list">
    <table style =" width: 600px;">
    <thead>
      <tr>
        <g:sortableColumn property="id" title="ID" />
        <g:sortableColumn property="name" title="Name" />
        <g:sortableColumn property="Delete" title="Delete"/>
      </tr>
      </thead>
        <tbody>
          <g:each in="${deptInstanceList}" status="i" var="deptInstance">
          <tr>
            <td>${fieldValue(bean: deptInstance, field: "id")}</td>
            <td style =" width:280px;">
              <div style ="text-align: center; width: 210px; float:left; margin-top:5px;">
                ${fieldValue(bean: deptInstance, field: "name")}
              </div>
            </td>
            <td>
              <g:link onclick= "return confirm('Are you sure?');" controller="dashboard"
action="deleteDepartment" id="${deptInstance.id}">
                <button type="button" >Delete</button>
              </g:link>
            </td>
            <td> <g:link controller="dashboard" action="deptedit" id="${deptInstance.id}">
                       <button type="button" >Edit</button>
                     </g:link></td>
           </tr>
          </g:each>
         </tbody>
        </table>
         </body>
</html>
```

So we have now created a new controller without a corresponding Domain class and modified the view to suit our requirements. We need to be able to add the link to the dashboard somewhere. We will do it on the front page of our application. (If you have not edited your index file already, you will have a link for the newly added controller as well, and if you click on that link, you will get an error because it will look to access an index.gsp file corresponding to the controller you just created). So change this and point to the view that you want to. In our case, we will point it to deptlist. For this, first open the file index.gsp present in the employee folder under the views and layouts section.

Modify the below part

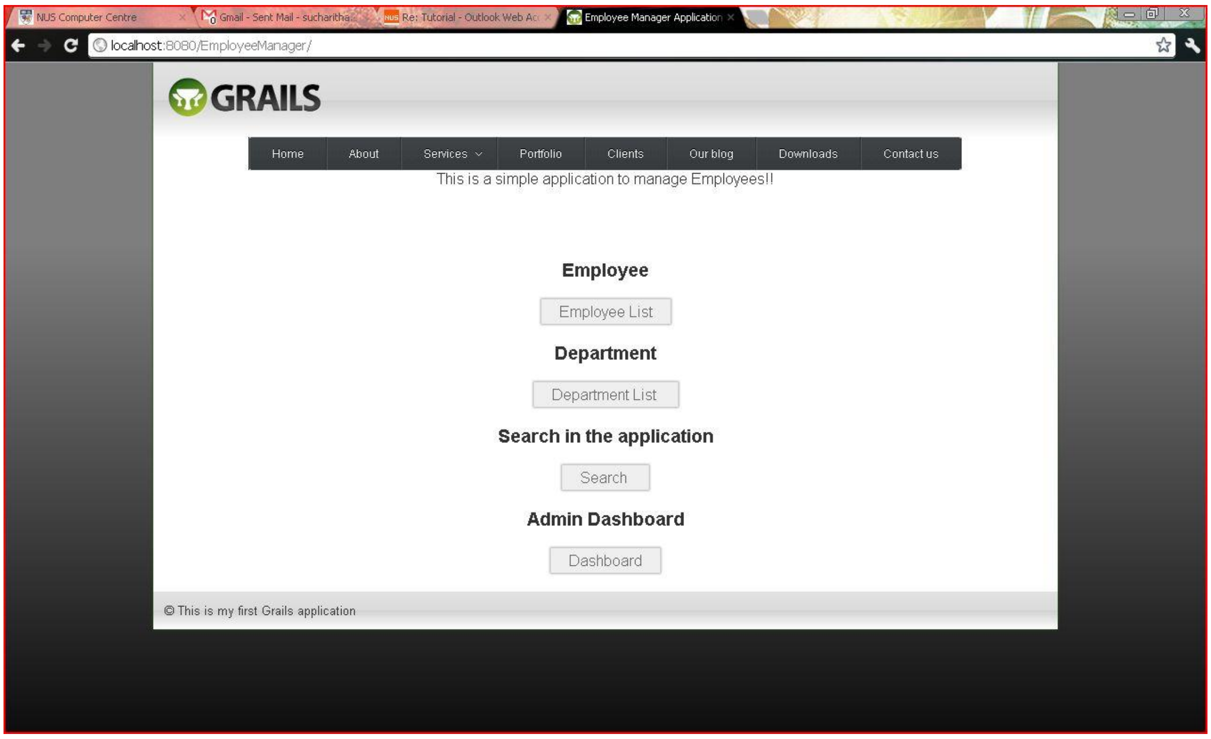
Index.gsp

```
<!doctype html>
<html>
<head>
<title>Employee Manager Application</title>
<meta name="layout" content="main" />
</head>
<body>
  <center>
    <p> This is a simple application to manage Employees!! </p>
    <br>
    <br>
    <br>
    <br>
<div class="homeCell" >
<h3>Employee</h3>
<br>
<span class="buttons" >
<g:link controller="employee" action="list" >Employee List</g:link>
</span>
</div>
    <br>
<div class="homeCell" >
<h3>Department</h3>
<br>
<span class="buttons" >
<g:link controller="department" action="list" >Department List</g:link>
</span>
<br>
```

```
<br>
<div class="homeCell" >
<h3>Search in the application</h3>
<br>
<span class="buttons" >
<g:link controller="searchable" action="index" >Search</g:link>
</span>
<br>
<br>
<div class="homeCell" >
<h3>Admin Dashboard</h3>
<br>
<span class="buttons" >
<g:link controller="dashboard" action="deptlist" >Dashboard</g:link>
</span>
</div>
  </center>
 </body>
</html>
```

Now run the project and observe the changes that you have made.

Front Page:


NUS Computer Centre
Gmail - Sent Mail - sucharitha
Re: Tutorial - Outlook Web Acc
Employee Manager Application
localhost:8080/EmployeeManager/
GRAILS
Home
About
Services
Portfolio
Clients
Our blog
Downloads
Contact us

This is a simple application to manage Employees!!

Employee

Employee List

Department

Department List

Search in the application

Search

Admin Dashboard

Dashboard

© This is my first Grails application

Deploying a Web Application to Tomcat

We will now deploy the web application to Tomcat on the VM.

MySQL Connector

Paste the **mysql-connector-java-5.1.16-bin** into the project's **Libraries** folder. This allows your application to connect to MySQL database.

Test Connection

In **DataSource.groovy**, change the **development** block of code to reflect the following, changing the **NUSNETID** and **PASSWORD** to your own.

```
development {
    dataSource {
        pooled = true
        driverClassName = "com.mysql.jdbc.Driver"
        dbCreate = "update"
        url = " jdbc:mysql://mysqldb.comp.nus.edu.sg/NUSNETID"
        username = "NUSNETID "
        password = "PASSWORD"
    }
}
```

Run the project. It should run, connecting to the **mysqldb** database.

Production Datasource

Again, in **DataSource.groovy**, change the **production** block of code to reflect the following.

```
production {
    dataSource {
        pooled = true
        driverClassName = "com.mysql.jdbc.Driver"
        dbCreate = "update"
        url = " jdbc:mysql://mysqldb.comp.nus.edu.sg/NUSNETID"
        username = "NUSNETID "
        password = "PASSWORD"
    }
}
```

Tomcat Configuration

We will now change the default port of Tomcat (installed on the VM) to point to port 80 (if you have not done so during installation).

In **server.xml** located in the **Tomcat/conf** folder, change the connector port to reflect the following:

```
<Connector port="80" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

Next we add a user role so that we can login to the Tomcat Manager later. In **tomcat-users.xml** located in the same **Tomcat/conf** folder, add the following line (make sure its outside comments but inside the **tomcat-users** element.

```
<tomcat-users>
  ...
  <user username="admin" password="admin" roles="manager-gui"/>
</tomcat-users>
```

Start the Tomcat service and point your browser to http://localhost.

You'll see the Tomcat welcome page. Look for the **Tomcat Manager** link on the left and click it. Enter the username and password that we have set earlier. In this case its **admin** and **admin** respectively.

The Manager page displays currently deployed web applications in the central table of the **List Applications** page.

Create the Production WAR File

We will now build the project to deploy in Tomcat.

Right click on the project in Netbeans and click **Build**. This creates the file: **/target/EmployeeManager-0.1.war**.

"0.1" designation comes from the application.properties file in the application top level, from the line: **app.version=0.1**

Rename the war file simply: **EmployeeManager.war**

Deploy WAR File in Tomcat

From the Tomcat Manager, go to the **Deploy>War file to deploy** section at the bottom. Click the **Choose File** button and locate the newly created **EmployeeManager.war** file (if you have created the WAR file on a different machine, say your laptop, transfer it to the VM first).

Click the **Deploy** button and wait, because WAR files are huge.

You should get confirmation by the appearance of the new **Application** entry:

Path	**Display Name**	**Running**
/EmployeeManager	/EmployeeManager-production-0.1	true

Browse to http://localhost/EmployeeManager and see your deployed web app!