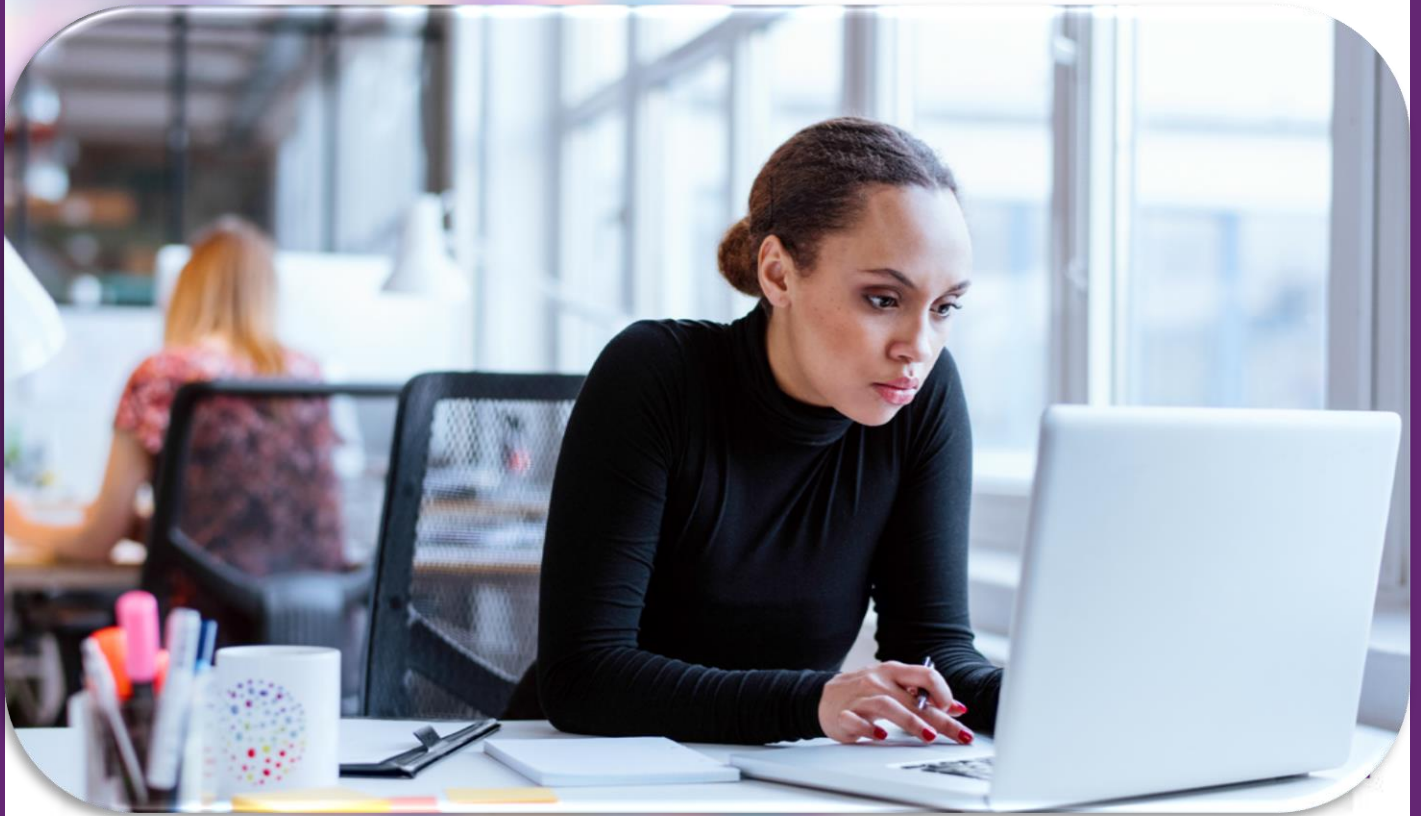


A Developers Overview: Getting Started in Banner 9

Self-Service Apps



Updated	Description
03/16/2015	Original version of training workbook.
04/16/2015	Revised 'Finding Your Files' exercises in Lab 5, typo corrections, updated White Paper in Appendix
05/01/2015	Refined course exercises
05/15/2015	Rearranged contents to accommodate length of course
07/10/2015	Deleted integration tests, which are no longer supported., expanded exercises to include creation of a new table/domain/view structure.
09/03/2015	Corrections and updates.
09/15/2015	Updated information, added additional exercise on Lists.
09/15/2015	Added pre-class environment setup information, added Greeting App preliminary work. Corrected CSS work to reflect best practices.
11/19/2015	Overall improvements for clarity. Changes to reflect changed database setup.
12/16/2015	Removed environment setup information for labs, since the necessary information will be demo'ed instead.
2/18/2016	Updates and improvements.
2/17/2017	Updates and improvements for version GenEvents 9.4 using AngularJS.
12/07/2017	Content edited for clarity.
1/24/2018	Add Information on Extensibility Tools
3/19/2018	Updated Course Title
4/13/2018	Added activities to sections
6/22/2018	Updated typos and activities
02/28/2019	Updated eCommunities to Communitas. HUB to Ellucian Customer Center pgs. 143.26.145

IP Disclaimer

Without limitation: Ellucian®, Banner®, Colleague®, and Luminis® are trademarks of the Ellucian group of companies that are registered in the U.S. and certain other countries; and Ellucian Advance™, Ellucian Course Signals™, Ellucian Degree Works™, Ellucian PowerCampus™, Ellucian Recruiter™, Ellucian SmartCall™, are also trademarks of the Ellucian group of companies. Other names may be trademarks of their respective owners.

© 2017 Ellucian. Contains confidential and proprietary information of Ellucian and its subsidiaries. Use of these materials is limited to Ellucian licensees and is subject to the terms and conditions of one or more written license agreements between Ellucian and the licensee in question.

Table of Content

Introduction	9
Lab 1 - Grails and IntelliJ Basics – The Building Exercise	14
Lab 1a	15
Create the App	16
Create MVC Components	17
Add an Entry to Message.properties	18
Run the App	19
Lab 1b	21
Lab 1 – Self Quiz	24
Lab 1 – Self Quiz Answers	26
Lab 2 - Use Git to Download a Banner 9 Self-Service App	29
Clone the Application Repository	30
Access the Submodules for the Application	31
Set Up the Configuration Files for the Banner 9 Application	33
Run Baseline Code from the Command Line	35
Lab 3 - Import Code and Configure Parameters	38
Importing Source Code into your IDE	39
Setup the Environment	40
Clean Configuration	41
Run the Application	42
Summary	44
Lab 4 - Extend Apps with New Data – Add a Comment Field to Account Registration Page	45
Scenario	46
Reference	46
Add a Comment Field to Account Registration	47
Implementation Steps	47

Extend UI Apps	53
Launch General Events Self-Service Application and Test Application	55
Summary and Notes	56
Lab 5 - Add the Gender field with Validation to the Account Registration Page ...	57
Scenario	57
Add the Gender Field	59
Code Location	60
Run and Test App	61
Add Validation	62
Alternate Validation Using Directives	63
Summary and Notes.....	65
Lab 6 - Add an Email Type Field with Validation to the Account Registration Page	66
Working with Decorator	71
Extend the Controller	72
Controller Imports	72
EmailType Decorator	73
Trace through the Controller 1	75
Add the fetchListofEmailTypes method to the Service.....	76
Trace through the Controller 2	76
Copying Logic	77
Working with the Web Page (View)	78
Test the Extension	80
Summary and Notes.....	81

Lab 7 - Add a Guest Policy Page	82
Scenario	82
Conditional Rendering of Page Content	82
Files Modified for Condition Rendering of Page Content	83
Managing Source Code	83
Gather information	84
Examine the Flow – How the Page Pieces are Displayed	84
Notes	94
Examine the Flow – How Database Data is Retrieved	95
Perform Modifications	96
Extend the Events List Page	97
Discussion	101
Add a Link and Stub Page	105
Notes	109
Lab 8 - Add a Title to the Guest Policy Page	110
Notes	116
Before you start the next lab	116
Lab 9 - Create a New Table, Domain, and HTML – the Guest Policy Page	117
Scenario	118
The New Oracle Table	120
Create a New Domain Model	121
Add the New Domain to the Hibernate File	123
Extend the Grails Service	124
Extend the Grails Controller	125
Discussion	126
Add the AngularJS Service for the Guest Policy Page	127
Update the AngularJS Controller	130
Extend the Guest Policy HTML	130
Testing	131

Summary	132
Lab 10 - Domain Integration Testing (Optional).....	133
Appendix 1: File Modification Checklist Reference	140
Appendix 2: Self-Service Banner 9 Application Downloading and Environment Setup	143
Appendix 3: Create Grails_User	151

Important Notices

In this guide important information is emphasized in the following types of notices.



Activity: Indicates class participation or interaction.



Alert: Indicates cautionary information.



Assessment: Indicates that there is a required assessment to complete.



Best Practices: Indicates best practices or recommended ideas for applying content.



Reference: Indicates additional reference material is available that is not part of the course materials.



Technical Tip: Indicates a technical tip or trick.



Note: Indicates helpful information to perform task.

Introduction

Description

This course introduces the basic concepts to work with Self-Service Banner 9 and introduces participants to Banner customizations (known as extensions) using Banner 9 technology. It provides hands-on knowledge, including an introduction to modern tools and best practices, to extend and test source code modifications to the Self-Service Banner user interface.

Please note that screenshots provided throughout this Participant's Guide may not match those seen on your system, due to varying versions of the tools used. So, please keep this in mind, noting that they are there for guidance.

What You'll Learn

In this module, you learn:

- ☐ Grails Framework
- ☐ Coding by Convention and the use of the IntelliJ IDE
- ☐ The MVC design model
- ☐ Domain models
- ☐ Composers, Controllers, Services, and Views

Prerequisites

- The Introduction to Java/OO Programming course or equivalent experience with OO (Object-Oriented) programming concepts.
- Completion of the Intro to Groovy, Grails and XE Web Technology for Banner course.

Audience

Higher Education Information Technology developers with PL/SQL experience who have taken the Introduction to Java/OO Programming or who have equivalent experience, and the Introduction to Groovy, Grails and Banner XE Web Technology for Banner course.

After this class: Setting up your own Environment

Please see the appendix entitled "*Appendix 2: Self-Service Banner 9 Application Downloading and Environment Setup*".

Extensibility Tools

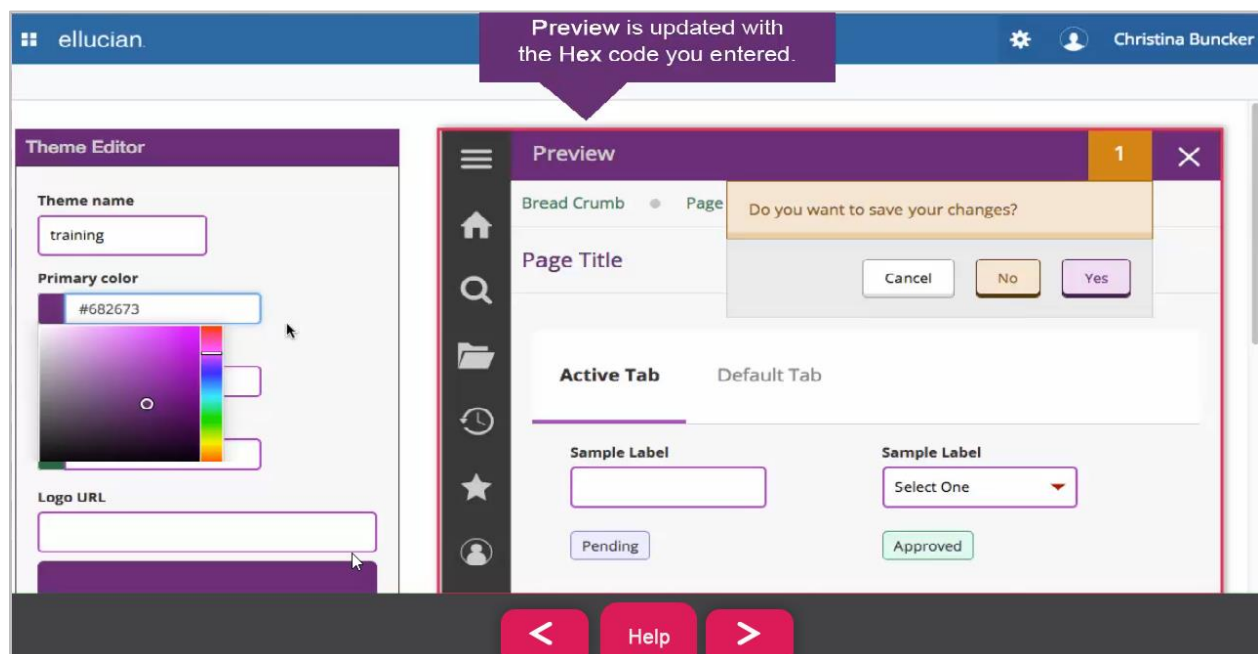
Before modifying Banner self-service pages, first consider the Ellucian tools available today that may allow you to perform actions you were considering performing in another way.

Note:

It is beyond the scope of this course to cover these topics in depth. We are providing this information to increase your awareness of their existence, purpose, and to provide links to more detailed information/training. More tools and functionality are continuing to be developed by Ellucian.

Theme Editor

Purpose	Create and apply your institution's custom colors and logos to the Banner interface.
Technical Information	JSON is used for themes while SCSS is used for theme templates. Color scheme utilizing hex codes PNG or SVG logo 18 pixels high or scalable
More Information	Self-Paced Training Banner 9 Upgrade Courses https://training.ellucian.com/auth/course/installing-and-using-banner-theme-editor-banner-theme-editor



Page Builder

Purpose	Provides a Browser front end to develop interactive and responsive Single Page applications productively. Create, deploy and maintain Banner custom pages and custom RESTful services using the Self-Service Page Builder.												
Technical Information	<p>Do not need a Grails development environment. Supported by the following application server and operating system combinations:</p> <table border="1"> <thead> <tr> <th>Tomcat (64 bit)</th><th>WebLogic (64 bit)</th></tr> </thead> <tbody> <tr> <td>Red Hat Linux 5.x, 6.x</td><td>Red Hat Linux 5.x, 6.x</td></tr> <tr> <td>Windows Server 2008</td><td>Windows Server 2008</td></tr> <tr> <td>Solaris 10</td><td>Solaris 10</td></tr> <tr> <td>AIX 6.1 (JDK 1.7 SR10 or later)</td><td>AIX 6.1 (JDK 1.7 SR10 or later)</td></tr> <tr> <td>HP-UX</td><td>HP-UX 11iV3 (11.31)</td></tr> </tbody> </table>	Tomcat (64 bit)	WebLogic (64 bit)	Red Hat Linux 5.x, 6.x	Red Hat Linux 5.x, 6.x	Windows Server 2008	Windows Server 2008	Solaris 10	Solaris 10	AIX 6.1 (JDK 1.7 SR10 or later)	AIX 6.1 (JDK 1.7 SR10 or later)	HP-UX	HP-UX 11iV3 (11.31)
Tomcat (64 bit)	WebLogic (64 bit)												
Red Hat Linux 5.x, 6.x	Red Hat Linux 5.x, 6.x												
Windows Server 2008	Windows Server 2008												
Solaris 10	Solaris 10												
AIX 6.1 (JDK 1.7 SR10 or later)	AIX 6.1 (JDK 1.7 SR10 or later)												
HP-UX	HP-UX 11iV3 (11.31)												
More Information	<p>Self-Paced Training Banner 9 Upgrade Courses</p> <p>https://training.ellucian.com/auth/course/page-builder-installation-and-implementation-banner-technical</p>												

Custom University

Virtual Domain Composer

- Export Domains
- Virtual Domain Roles

Visual Page Composer

- Export Pages
- Page Roles

CSS Manager

- Export CSS Stylesheets

Application Pages

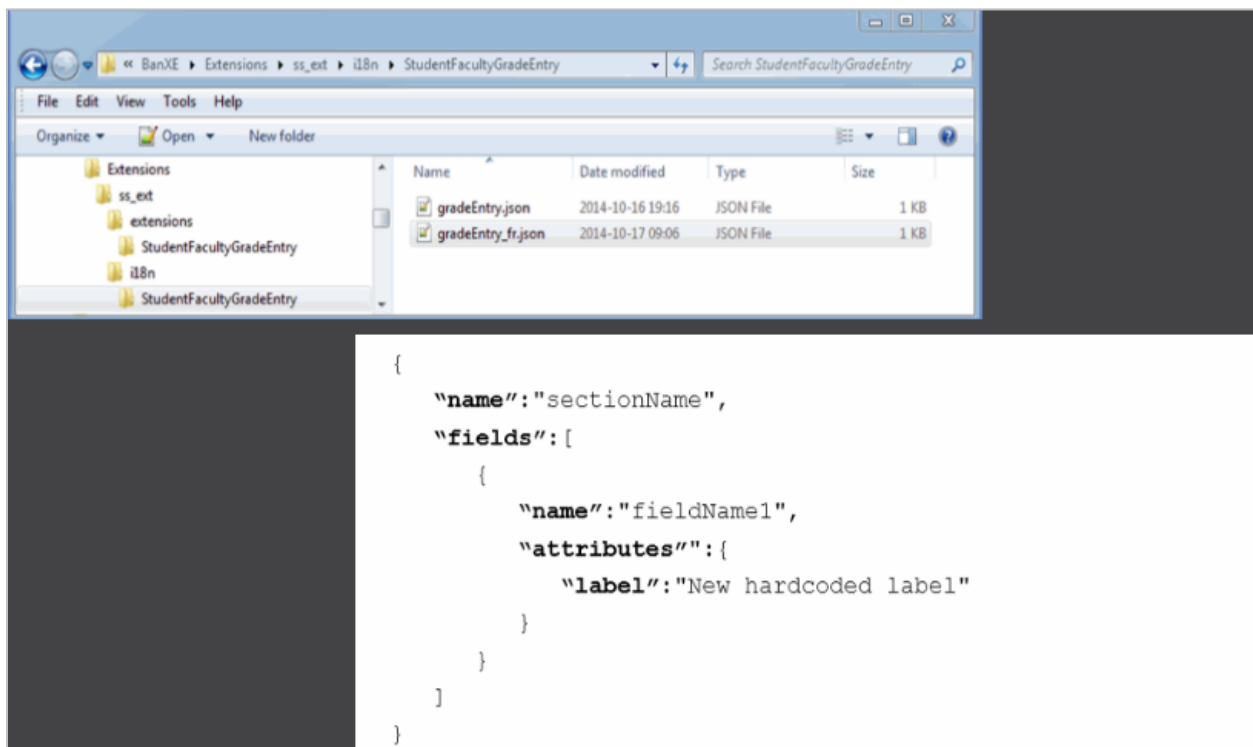
Page Name	Last Updated	Actions
NewPage	Jul 07th, 2016	[Edit] [Delete]
NewPage1	Jul 07th, 2016	[Edit] [Delete]
Duplicatepbadm.ExportCss	Jul 07th, 2016	[Edit] [Delete]
spriden_edit	Jul 07th, 2016	[Edit] [Delete]
xdfdfdfdf	Jul 05th, 2016	[Edit] [Delete]

Search Pages

Page Name	Last Updated	Actions
abcdpage	Jul 07th, 2016	[Edit] [Delete]
cst_testing	Jul 07th, 2016	[Edit] [Delete]
DuolicatepbadmPageRolesTest	Jul 07th, 2016	[Edit] [Delete]
Duplicatepbadm.ExportCss	Jul 07th, 2016	[Edit] [Delete]
NewPage	Jul 07th, 2016	[Edit] [Delete]

Configure Page Components

Purpose	The Extension Editor is available in the Tools menu and provides a simple interface to allow the user to add code to configure the UI of a page, such as, hiding fields that are not required, changing the order of components, and changing the labels of items on the UI. Any common component item on a page can be amended provided the component has the attributes required for extensibility.
Technical Information	Target audience JSON Developers
More Information	Self-Paced Training Banner 9 Upgrade Courses https://training.ellucian.com/auth/course/configure-page-components-implementation-banner-technical



Text Manager

Purpose	Provides institution customization of Banner user interface text strings in a base language and other languages. Customizations are retained from release to release and may be added to at any time.
Technical Information	Rights to access the Text Manager Administrator pages including: GMAPROJ page GMATRAN page etc.
More Information	Available March 2018

Text Manager Interactive Translator GMATRAN 9.3.4 (BUILD)

Project: Cindy_test2 Source Language: root Target Language: enUS

Populate Targets Pre-translate Fuzzy Match Delete Listed Records

SEARCH AND REPLACE

Find What: Description Replace With:

Search In: ☒ Source Text ☐ Target Text

Find Next Replace Replace All

EXTENDED SEARCH

INTERACTIVE TRANSLATOR

Project: Cindy_test2 Source Language: root Target Language: enUS ☐ Select All

Source Text	Target Text	Status	Sel...	Object Name	Source
Base URL	>	Untranslated	<input type="checkbox"/>	.title	NET/HEDTECH/BANNER/GENERAL/GTVLANG/VIEWS/LOCAL
Base Uri	>	Untranslated	<input type="checkbox"/>	.prompt	NET/HEDTECH/BANNER/GENERAL/GTVLANG/VIEWS/LOCAL
Description	> Description Changed	Translated	<input type="checkbox"/>	.prompt	NET/HEDTECH/BANNER/GENERAL/GTVLANG/VIEWS/LOCAL

What terminology does your institution use?

course

class

module

programme

program

freshman

first year

student

estudiante

Customize it with Text Manager



Lab 1 - Grails and IntelliJ Basics – The Building Exercise

What You'll Learn

In this module, you:

- ☐ Experience the makeup and workings of the MVC architecture, in action.
- ☐ Get practical experience using an IDE (Integrated Development Environment).
- ☐ Learn how to use information in CSS (Cascading Style Sheet) files to customize the look of Self-Service Banner pages.

Lab Overview

In this section, you learn how to create a gsp-based Groovy/Grails project and about its makeup. You name the project and create the following:

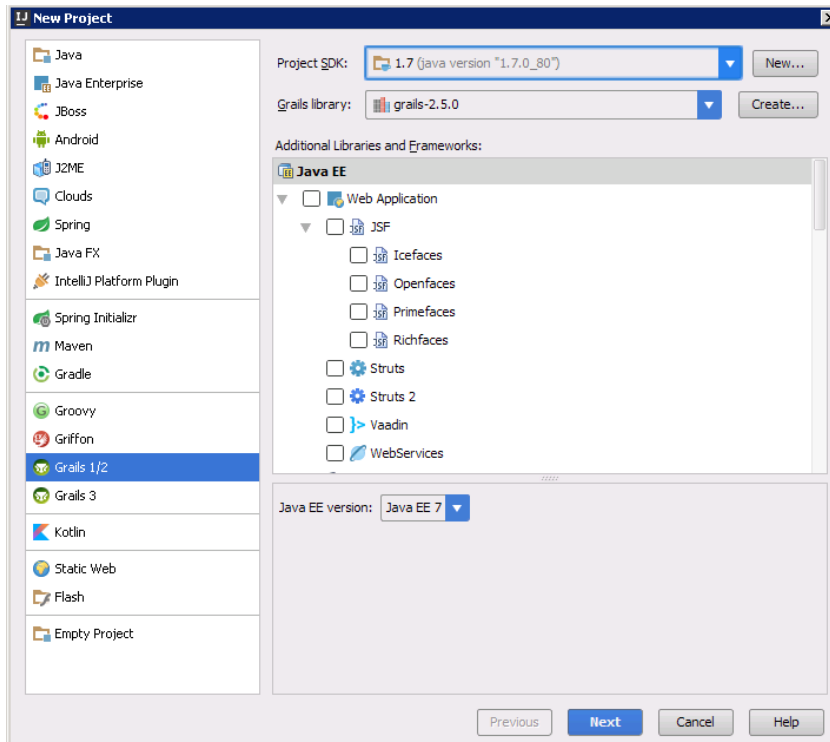
1. Domain: **Building.groovy**.
We are not persisting to a RDBMS (Relational Database Management System). Instead, we will be using the IDE's in-line memory database to simulate the database experience for us.
2. Controller: A controller that instantiates instance of domain object with initial values – **BuildingController.groovy**.
3. View: A set of gsp subviews that display the data. The gsps reside under **views/building**.
4. Label: A needed label is added to **messages.properties**.



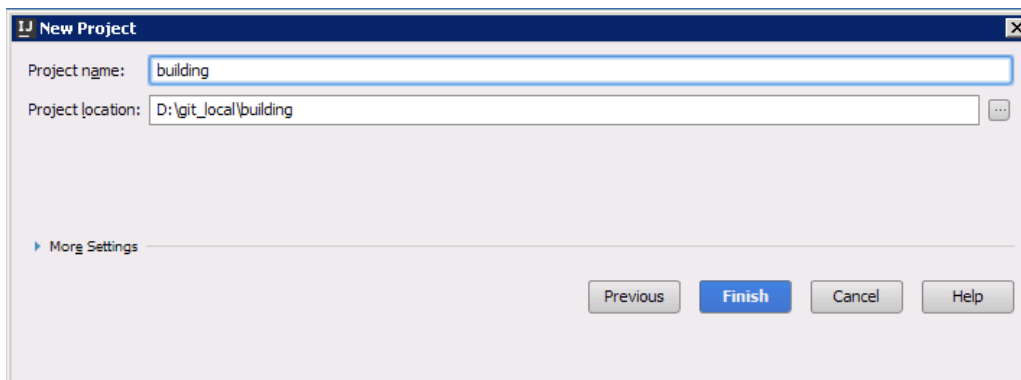
NOTE: The order of these steps is very important to IntelliJ to ensure proper code generation.

Lab 1a

1. Using IDE, choose the option to create a new Grails project.
2. In the dialog box, choose the **1.7 JDK version** and the **Grails version 2.5.0** and click **Next**.



3. Name your project 'building' and put it inside the new git_local directory.

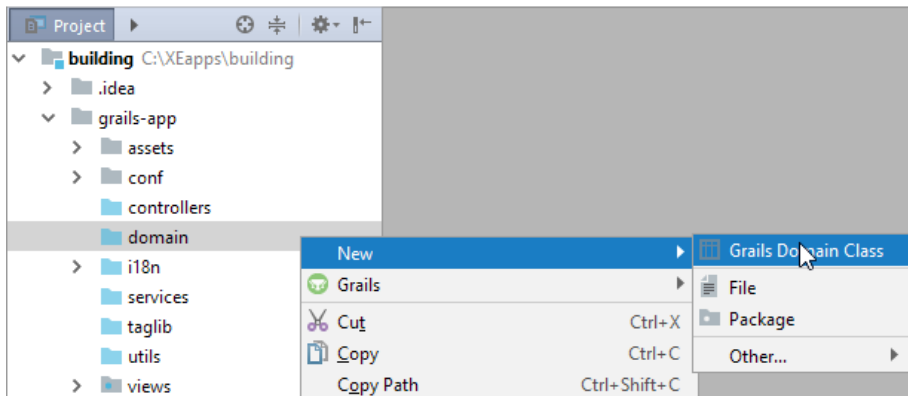


4. Click **Finish**.
5. Click the "Run create-app" button on the next screen. Wait for IntelliJ to finish creating the directory structure before continuing.

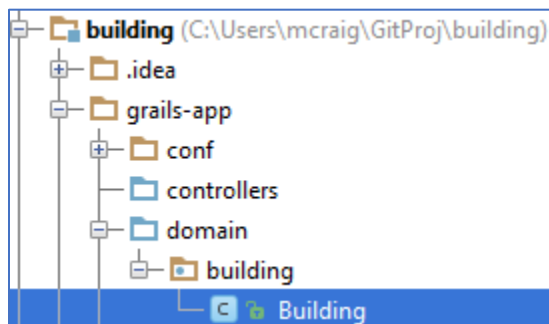
Create the App

Define the data. This involves creating a domain.

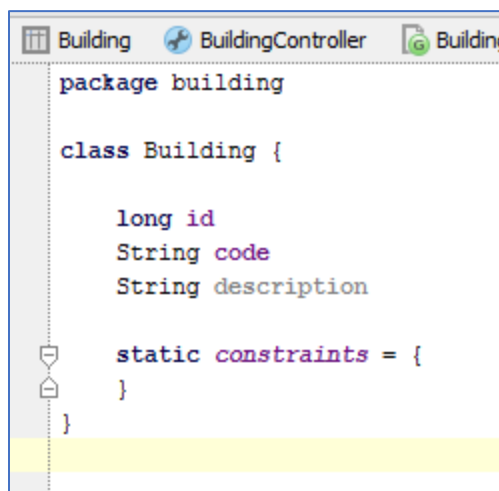
6. Right-click **domain**.
7. Choose **New** to create a New Domain and name it **Building**.



8. The new domain appears.



9. Create the following three fields.



Note:

We are not completing constraints.

Constraints is a static property for the domain that specifies constraints corresponding to the database constraints.

Field properties field are nullable (the default is not nullable) and whether a field needs to be of a certain value or of a certain length, etc.

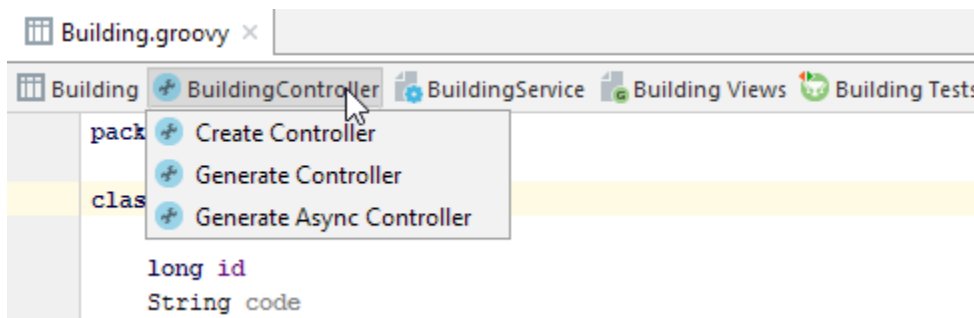
For this exercise, we are not mapping back to a back-end database.

We are working with an 'in-line' virtual database. The data is not saved after each run-app. In later exercises, we will persist to Oracle. Persistence involves more work in the domain. For now, the work is with the Domain.

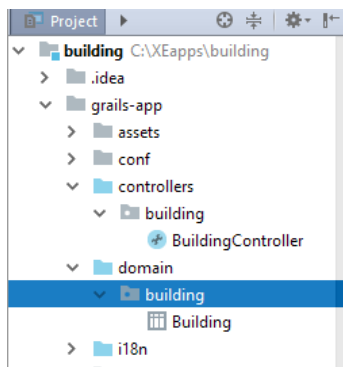
Create MVC Components

MVC components can be created by using Grails to generate default versions of these components. It is this method used in the following example.

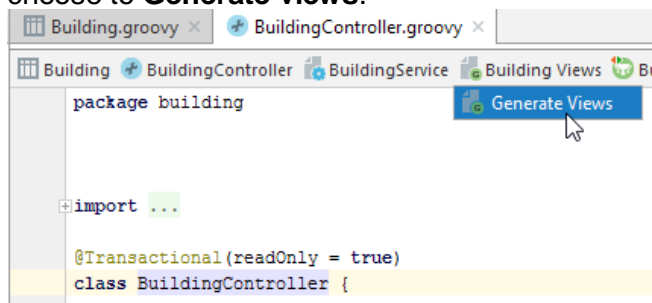
10. With the Domain in the edit window in IntelliJ, click the **Generate controller**. It is automatically named **BuildingController**.



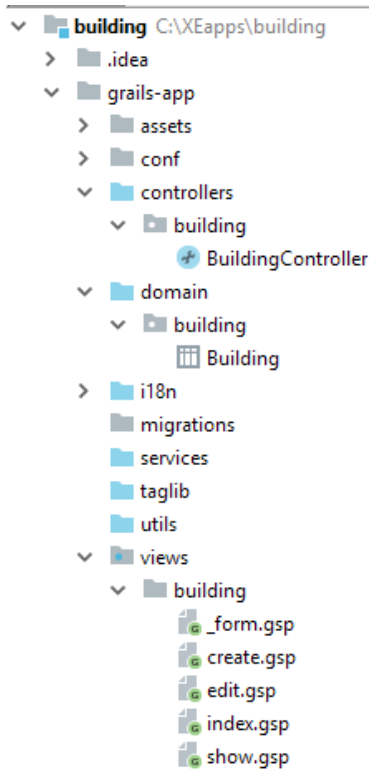
The BuildingController.groovy file is created. This code contains several Closure methods. These methods correspond to individual gsp's and are explained in the next steps.



11. Creation of views is similar to the creation of controllers. In the BuildingController window, choose to **Generate views**.



Note that gsp's are generated as a set, under the subdirectory **building**.



Examine the action methods created in the controller.



Which ones have corresponding gsp pages?
What does the index do?
The show method?

Add an Entry to Message.properties

During the creation of the Controller, the default label **building.label** was created.

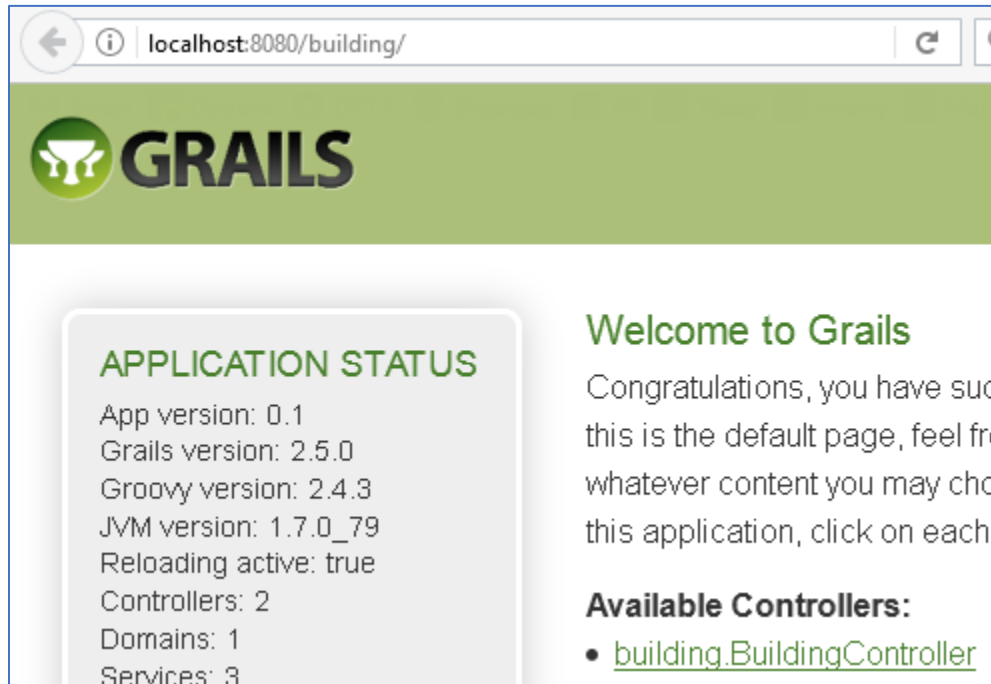
12. Add this label to the **grails-app/i18n/messages.properties** file.

```
default.button.delete.label=Delete
default.button.delete.confirm.message=Are you sur
building.label=Building Table Maintenance
```

13. **CTRL-S** to **Save All**.

Run the App

14. From the menu at the top of the IDE screen choose **Run**. This may appear differently with different versions of the IDE.
15. From **Available Controllers**, click the link for the **building.BuildingController**.



Your URL is `http://localhost:8080/building/building/index`



Note: The end of your URL is **index**.

This enables the index method in your BuildingController file to execute, which also automatically renders the index gsp subview.

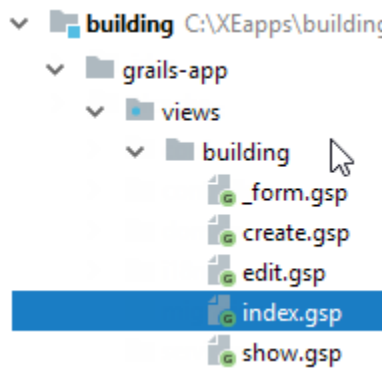
16. From IntelliJ, double-click the BuildingController file.
17. From the editor window, find the **index closure** method.

```
@Transactional(readOnly = true)
class BuildingController {

    static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]

    def index(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        respond Building.list(params), model:[buildingInstanceCount: Building.count()]
    }
}
```

18. From the **grails-app/views/building** directory, **views > index.gsp**.



19. Return to examine the methods in the Controller. Note as files open, they remain open in the IntelliJ editor, each with a different tab.



It is a best practice keep these files open for quick reference.

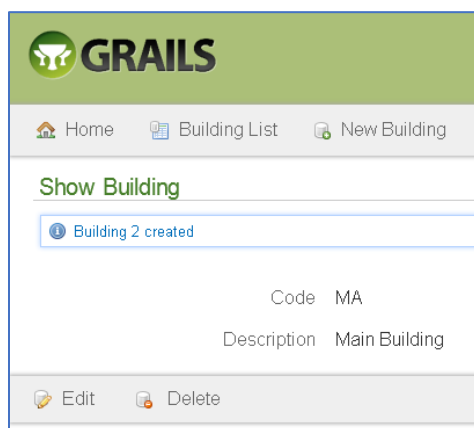
20. In the Controller, locate the closure methods for index, create, save, show.
In your browser review the URLs: http://localhost:8080/building/building/<insert method_name here> for index, create, save, list, show.



Review the code and understand what the code is doing for each closure.
Do the behaviors of the URLs you have entered make sense?

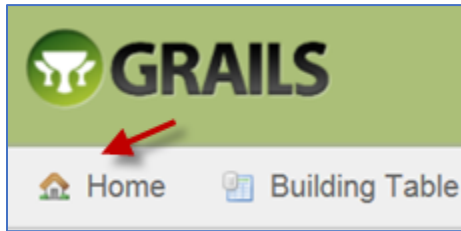


- Explore the Building App.
- Create some new buildings.
- When finished your display will look as shown below.

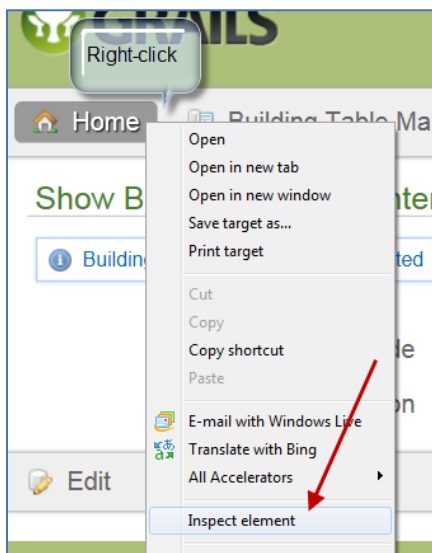


Lab 1b

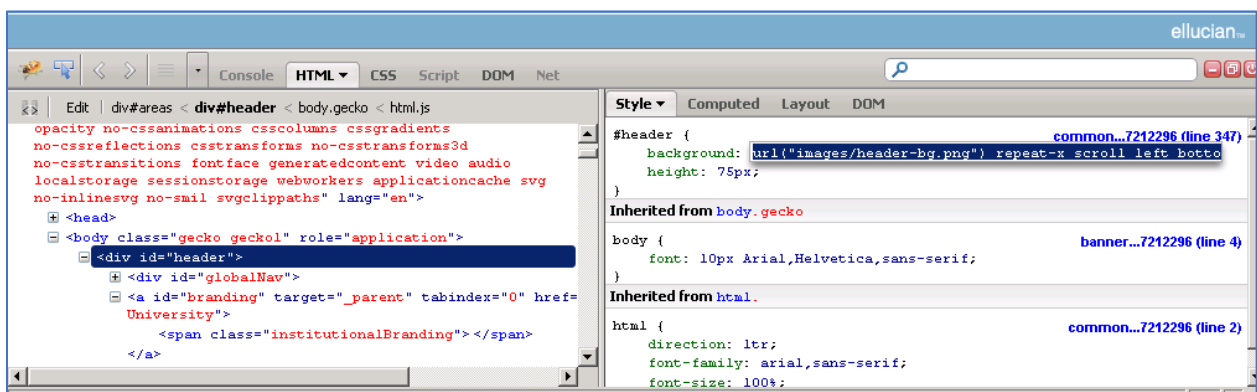
In this lab learn more about how CSS files work with Banner apps.
The **Home** icon is shown below.



1. Right-click the home icon and choose **Inspect element**. The resulting screen will look different with different browsers and browser versions.



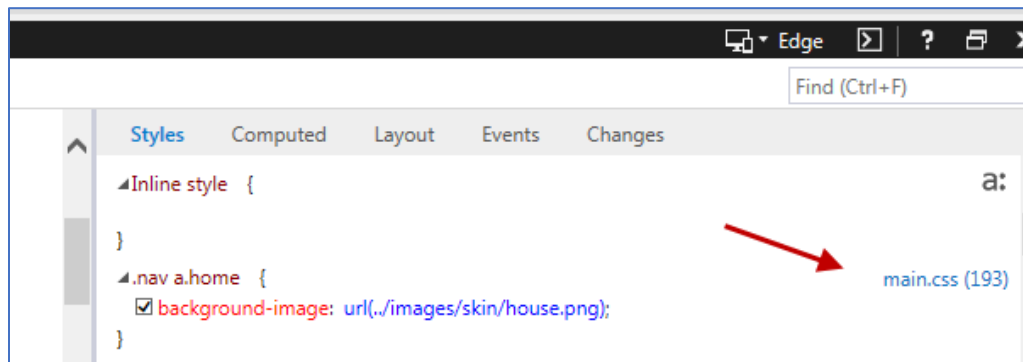
2. At the bottom of the screen review the code. The href code producing the image is highlighted.



It has a property called "class="home". This refers to a label pointing to a set of formatting in a CSS (Cascading Style Sheet) file. Review the code, the home anchor is nested in a div class, .nav.

On the right-side panel, the first arrow to the right points to an entry '.nav a.home'. This is the literal label that will be in a .CSS file.

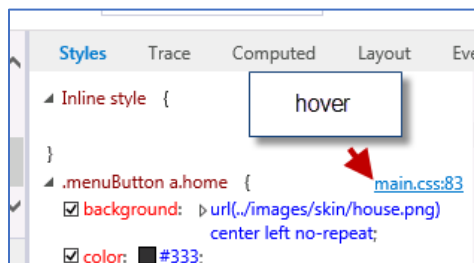
The image file along with its directory path displays. The name of the .CSS file is **main.css** and the number beside it refers to the code line within this file where this information is held.



3. Hover your cursor over **main.css** file to display the directory structure within the project. Note this name.

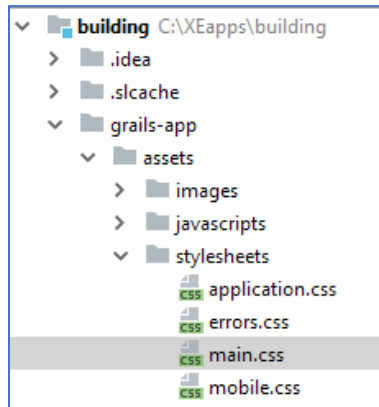


Note: Sometimes, in virtual environments the hover does not display.




Working with the .CSS File

4. In IntelliJ, open **grails-app > assets > stylesheets > main.css**.



5. Perform a search for the string **nav a.home**.
6. Locate the entry for the background image as shown below:

```
.nav a.home, .nav a.list, .nav a.create {  
    background-position: 0.7em center;  
    background-repeat: no-repeat;  
    text-indent: 25px;  
}  
  
.nav a.home {  
    background-image: url(../images/skin/house.png);  
}
```



7. Confirm that it is the correct image. From the project directory, open **web-app > images > skin** and double-click the **house.png** file. The following image previews:



The image is updated in a following exercise.

Notes



Lab 1 – Self Quiz

Review the previous topics and answer the following questions.

The answers follow this quiz.

This quiz is challenging. It will make you think about the structure and the behavior of the web App.

Domain:

- a) Review the Building domain you created.
When creating the attributes in the domain, did you need to specify lengths? ____
- b) Are we performing data mapping here to the back-end Oracle table? ____
- c) What would you use to represent an attribute that to map to an Oracle field of type Varchar2? _____
- d) What type of information is entered in to the static constraints? _____

Controller:

- a) Locate the name of the controller you used in this lab.
What naming convention is used when creating the controller? It is _____ + _____.
- b) The controller contains _____ methods.

Views:

- a) In IntelliJ, review the directory structure.
Under the Views directory, standard subviews under a grouping directory (building) and actions are generated when views are generated. List the gsp views that have corresponding action methods in the controller.

- b) Are there any subviews (gsps) that do not have corresponding methods in the controller?
List one: _____
- c) Are there any action methods in the Controller that do not have corresponding gsps in the view directory?
List one: _____

Instantiation of Data Objects

- a) From the Groovy/Grails prerequisite class, a row of data is the instantiation of a domain object, and a list of rows of data is a list collection object containing multiple instantiations, or rows.
Review the action methods in the controller and locate where a new instance of the Building object is created. Identify one of the action methods where this happens.
-

Page Traversal

- a) As your App is running, go to: `http://localhost:8080/building/building/index`.
Google 'port 8080' and be prepared to discuss what this port is used for.
- b) What is 'index'?
- c) Look at your BuildingController file and find the method for `index()`. What does this method do?
- d) Under Views / building. Is there a list view? Why not?
- e) Go to your running App and make sure the following URL is in the URL box at the top:
`http://localhost:8080/building/building/index`
Change the last part of it, 'index' to say 'list'. What happens?
- f) Now, take away the last word in the URL, which is 'index' and refresh your page. What happens?



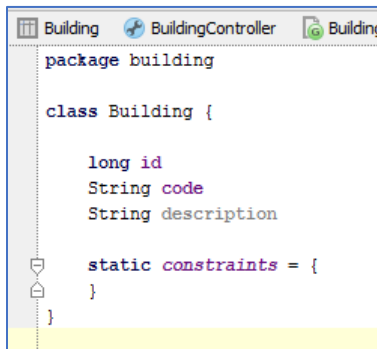
Lab 1 – Self Quiz Answers

Domain:

- a) Review the Building domain you created.

When creating the attributes in the domain, did you need to specify lengths?

ANSWER: No.



```
package building

class Building {

    long id
    String code
    String description

    static constraints = {
    }
}
```

- b) Are we performing data mapping here to the back-end Oracle table?

ANSWER: No, we have not covered this yet. As you can see in the above screenshot, no information on an Oracle table is provided. Stay tuned for more information on mapping to the back end

- c) What would you use to represent an attribute that to map to an Oracle field of type Varchar2?

ANSWER: String

- d) What type of information is entered in to the static constraints?

ANSWER: We would put not null and check type of constraints here, to reflect those in the back-end Oracle table if there was one associated with this domain.

Controller:

- a) Find the name of the controller you used in this lab.

Looking at this name, what naming convention is used when creating the controller?

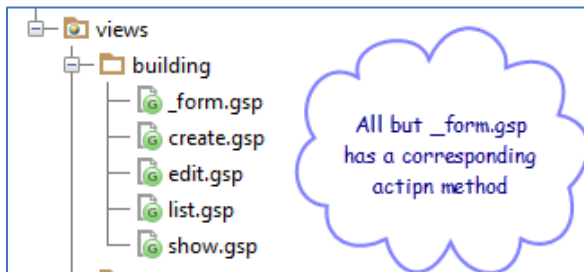
ANSWER: It is Domain name + Controller.

- b) The controller contains _____ methods.

ANSWER: Action.

Views:

- a) In IntelliJ, review the directory structure. Under the Views directory, standard subviews under a grouping directory (building) and actions are generated when views are generated. List the gsp views that have corresponding action methods in the controller.
- ANSWER:** All of them except `_form.gsp` have a corresponding action method. `_form.gsp` does not because it is a template and is used by the other gsps.



- b) Are there any subviews (gsps) that do not have corresponding methods in the controller?
Answer: `_form.gsp`
- c) Are there any action methods in the Controller that do not have corresponding gsps in the view directory?
Answer: "save" is one. It is not a gsp page in itself.

Instantiation of Data Objects

- a) From the Groovy/Grails prerequisite class, a row of data is the instantiation of a domain object, and a list of rows of data is a list collection object containing multiple instantiations, or rows.
- Review the action methods in the controller and locate where a new instance of the Building object is created. Identify one of the action methods where this happens.
- ANSWER:** The clue below gives the answer. The 'new' keyword causes a new Building to be instantiated. It makes sense that this is performed in the create controller method.

```
def create() {  
    [buildingInstance: new Building(params)]  
}
```

Page Traversal

- a) As your App is running, go to: <http://localhost:8080/building/building/index>.

Google 'port 8080' and be prepared to discuss what this port is used for.

ANSWER: Port 8080 is used for internet communication using the HTTP protocol.

- b) What is 'index'?

ANSWER: It is part of your URL and it corresponds to a method in the controller.

- c) Look at your BuildingController file and find the method for index(). What does this method do?

ANSWER: It returns the list of buildings. It also passes whatever parameters the user has been entering. This is stored in a list variable called params.

```
def index(Integer max) {  
  params.max = Math.min(max ?: 10, 100)  
  respond Building.list(params), model: [buildingInstanceCount: Building.count()]  
}
```

- d) Under Views / building. Is there a list view? Why not?

ANSWER: No, since the index view will display the list of buildings, as just discussed.

- e) Go to your running App and make sure the following URL is in the URL box at the top:
<http://localhost:8080/building/building/index>

Change the last part of it, 'index' to say 'list'. What happens?

ANSWER: The list method in the Controller does not exist, so you will just get a page not found.

- f) Now, take away the last word in the URL, which is 'list' and refresh your page. What happens?

ANSWER: When an action is not specified at the end of your URL, like "list" or "index", the page will automatically execute "index".

Lab 2 - Use Git to Download a Banner 9 Self-Service App

What You'll Learn

In this lab, download Banner 9 source code and verify there is a working application before making changes. In this module, you learn:

- ☐ That the origin we are using for the labs is a different directory on the same machine. In real life scenarios, the origin server will be a server with a host name and directory structure. The git commands are the same regardless of the location of the remote git files
- ☐ The source code has been verified so that no errors should occur when cloning and running the baseline app. Sometimes cloning baseline source code and trying to run the app will result in errors even if you have not changed the code. Check the client support site and Communities for any known issues when trying to run baseline source code.

Lab Overview

The goal of this lab is to provide an understanding of how to access the Banner 9 Self Service Application source code and verify it works before making changes.

Notes

Clone the Application Repository

1. Create a directory for your local repository on the D: drive, e.g. D:\git_local.
2. Open a Git Bash prompt by right clicking on the directory you just created.
3. Type in the following command to clone the git repository or access it from the **d:\labfiles\lab2 directory** and paste it into the window.

```
git clone D:/git_origin/banner/apps/banner_general_events_ssb_app.git
```

Run 'git help git' to display the help index.

Run 'git help <command>' to display help for specific commands.

```
blee@USD1400400 /D/git_local
```

```
$ git clone D:/git_origin/banner/apps/banner_general_events_ssb_app.git
```

```
Cloning into 'banner_general_events_ssb_app'...
```

```
done.
```

```
blee@USD1400400 /D/git_local
```

Access the Submodules for the Application

Banner rails plugin are managed by using Git submodules.

NOTE:

Steps 1-2 are already completed in the training environment. It is provided here for explanation.

Typically, the person managing your Git repositories would do this once for all your Banner Grails Self Service application source code to reflect the location of where the Banner rails plugin repositories exist in your environment.

For the app to work, use Git to set up the Banner rails plugins needed for the main Self-Service Banner application.

1. Open the app directory created by the git clone.
2. Open the .gitmodules file in the directory and verify that the URL for all the submodules contain the correct location. Change any URLs that do not match the source of your origin. Be careful performing a global search and replace as they do not always work as expected. For example, in the banner_general_events_ssb_app.git repository, most URL lines in the .gitmodules file display as follows:

```
url = https://git.ellucian.com/scm/banxeplug/banner\_spring\_security\_cas.git
```

and start with:

```
https://git.ellucian.com/scm/banxeplug.
```

But if you just do a global search and replace on

```
https://git.ellucian.com/scm/banxeplug
```

you will miss at least one url which is this

```
url = https://git.ellucian.com/scm/ext/web-app-extensibility.git
```



Best Practice: Examine all the URLs in the .gitmodules file and don't rely on global search and replace.

Ideally the person managing your Git origin server would update the .gitmodules in the app reflect the plugin locations based on your origin server so that any developers who clone the app have the URLs for the .gitmodules file pointing to the correct location already.



HINT: After you have changed all the URL lines in .gitmodules, search for <https://git.ellucian.com>.

If you find it, you have missed a URL that needs to be updated.

3. Return to the Git Bash prompt and get the submodules/plugins for the grails application.

```
blee@USD1400400 /D/git_local
$ cd banner_general_events_ssb_app/

blee@USD1400400 /D/git_local/banner_general_events_ssb_app (master)
$ git submodule update --init
Submodule 'plugins/banner_codenarc.git' (C:/source_to_bitbucket/origin/banner/pl
ugins/banner_codenarc.git) registered for path 'plugins/banner_codenarc.git'
Submodule 'plugins/banner_core.git' (C:/source_to_bitbucket/origin/banner/plugin
s/banner_core.git) registered for path 'plugins/banner_core.git'
Submodule 'plugins/banner_general_common.git' (C:/source_to_bitbucket/origin/ban
ner/plugins/banner_general_common.git) registered for path 'plugins/banner_gener
al_common.git'

Cloning into 'plugins/il8n_core.git'...
done.
Submodule path 'plugins/il8n_core.git': checked out 'f677c41981cbaad96676f5838c8
a295ea5f6eedc'
Cloning into 'plugins/sghe_aurora.git'...
done.
Submodule path 'plugins/sghe_aurora.git': checked out '7f95f17bb1eaf536476dbbd11
aba481d269f89f3'

Cloning into 'plugins/web-app-extensibility.git'...
done.
Submodule path 'plugins/web-app-extensibility.git': checked out '2177abfe96e7a8f
235de9578eff555039b19db3a'

blee@USD1400400 /D/git_local/banner_general_events_ssb_app (master)
$
```


Set Up the Configuration Files for the Banner 9 Application

1. Return to the directory with the cloned Banner 9 Self-Service Banner application.
2. Copy the two files that end in **.example** to the **C:\Users\<your username>\.grails** folder.
3. Rename them to end in **.groovy**.
4. Make the following changes to the banner_configuration.groovy.
 - a. In the bannerDataSource section:
 - Change the URL to reflect the Oracle Banner database you will be connecting to the database information provided by the instructor.
 - Change the username to **banproxy**.
 - Change the password to the banproxy password.

```
bannerDataSource {

    // JNDI configuration for use in 'production' environment
    jndiName = "jdbc/bannerDataSource"

    // Local configuration for use in 'development' and 'test' environments
    url      = "jdbc:oracle:thin:@HOST:PORT:SID"

    username = "USERNAME"
    password = "PASSWORD"
    driver    = "oracle.jdbc.OracleDriver"

    // Local configuration for using elvyx to view SQL statements that are bound. To
    // enable this driver, simply uncomment the
    // elvyx driver and url below. Do NOT comment out the 'myDataSource.url' above --
    // it is still needed for the authentication data source.
    // To use elvyx, download from "http://www.elvyx.com", unzip, and run from it's
    // top-level directory: java -jar lib/elvyx-1.0.24.jar
    //
    //elvyx.driver = "com.elvyx.Driver"
    //elvyx.url     = jdbc:elvyx:
    //localhost:4448/?elvyx.real_driver=${bannerDataSource.driver}&elvyx.real_jdbc=${ba
    //nnerDataSource.url}&user=${bannerDataSource.username}&password=${bannerDataSource.p
    //assword}"
}
```

NOTE:

The ampersand (&) remains in the connection string.

```
url      = "jdbc:oracle:thin:@HOST:PORT:SID"
```

b. In the bannerSsbDataSource section:

- Change the URL to reflect the Oracle Banner database you will be connecting to the database information provided by the instructor.
- Change the username to **ban_ss_user**.
- Change the password to the ban_ss_user password.

Again, the ampersand (&) remains in the connection string.

```
bannerSsbDataSource {
    // JNDI configuration for use in 'production' environment
    //
    jndiName = "jdbc/bannerSsbDataSource"

    // Local configuration for use in 'development' and 'test' environments
    //
    url      = "jdbc:oracle:thin:@HOST:PORT:SID"

    username = "USERNAME"
    password = "PASSWORD"
    driver    = "oracle.jdbc.OracleDriver"

    // Local configuration for using elvyx to view SQL statements that are bound. To
    // enable this driver, simply uncomment the
    // elvyx driver and url below. Do NOT comment out the 'myDataSource.url' above --
    // it is still needed for the authentication data source.
    // To use elvyx, download from "http://www.elvyx.com", unzip, and run from it's
    // top-level directory: java -jar lib/elvyx-1.0.24.jar
    //
    //elvyx.driver = "com.elvyx.Driver"
    //elvyx.url    = "jdbc:elvyx:
    //localhost:4448/?elvyx.real_driver=${bannerSsbDataSource.driver}&elvyx.real_jdbc=${
    //bannerSsbDataSource.url}&user=${bannerSsbDataSource.username}&password=${bannerSsb
    DataSource.password}"
}
```

NOTE:

The ampersand (&) remains in the connection string.

```
url      = "jdbc:oracle:thin:@HOST:PORT:SID"
```

5. No changes are needed to the **SelfServiceBannerGeneralEventManagement_configuration.groovy** file at this time.
6. Verify it is in **C:\Users\<your username>\.grails** folder. If you want to have logging of messages change from its delivered settings that is performed in this file.

Run Baseline Code from the Command Line

1. **Shift - Right-click** the folder where your App repository source code is located.
2. Select **Open Command Prompt Here**.
3. Verify the grails VM memory options are set by copying the **set_grails_opt.bat** file from **D:\labfiles\lab2** to your **Banner 9 General Events SSB** app directory.
4. Run **set_grails_opt.bat** from the command line. The contents of the grails_opt batch file are as follows:

```
SET GRAILS_OPTS=-server -Xms2048m -Xmx4096m -XX:MaxPermSize=256m
```

5. Run the grails commands to clean, compile and run the app.



Note: Your display may differ if the necessary jar files need to be downloaded. Also, using the `--plain-output` parameter prints all the output rather than condensing it.

```
grails clean --plain-output
grails compile --plain-output
grails run-app --plain-output
```

Example of what you might see displays below:

```
D:\git_local\banner_general_events_ssb_app>SET GRAILS_OPTS=-Xmx1G -Xms256m -XX:MaxPermSize=256m

D:\git_local\banner_general_events_ssb_app>grails clean --plain-output
|Loading Grails 2.5.0
|Configuring classpath
.
|Environment set to development
.....
|Application cleaned.

D:\git_local\banner_general_events_ssb_app>grails compile --plain-output
|Loading Grails 2.5.0
|Configuring classpath
.
|Environment set to development
.....
|Compiling 10 source files
..
```

```

|Compiling 1041 source files
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
.....
|Compiling 2 source files
..
|Compiling 33 source files
.
D:\git_local\banner_general_events_ssb_app>grails run-app --plain-output
|Loading Grails 2.5.0
|Configuring classpath
.
|Environment set to development
.....
|Packaging Grails application
....
|Compiling 2 source files
.....Configuration:
file:C:\Users\Administrator\.grails\banner_configuration.groovy
Configuration:
file:C:\Users\Administrator\.grails\SelfServiceBannerGeneralEventManager_configuration.groovy
Configuration: classpath:WebAppExtensibilityConfig.class
.....
2017-02-18 03:48:54,310 [main] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
'$HOME/.grails/banner_configuration.groovy'
2017-02-18 03:48:54,677 [main] INFO configuration.ApplicationConfigurationUtils
- Using configuration file
'$HOME/.grails/SelfServiceBannerGeneralEventManager_configuration.groovy'
2017-02-18 03:48:54,682 [main] INFO configuration.ApplicationConfigurationUtils
- Using configuration file WebAppExtensibilityConfig.class from the classpath
(e.g., from within the war file)
Configuration: file:C:\Users\Administrator\.grails\banner_configuration.groovy
Configuration:
file:C:\Users\Administrator\.grails\SelfServiceBannerGeneralEventManager_configuration.groovy
Configuration: classpath:WebAppExtensibilityConfig.class
.
|Running Grails application
2017-02-18 03:49:13,895 [localhost-startStop-1] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
'$HOME/.grails/banner_configuration.groovy'
2017-02-18 03:49:13,900 [localhost-startStop-1] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
'$HOME/.grails/SelfServiceBannerGeneralEventManager_configuration.groovy'
2017-02-18 03:49:13,908 [localhost-startStop-1] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
WebAppExtensibilityConfig.class from the classpath (e.g., from within the war file)
Configuration: file:C:\Users\Administrator\.grails\banner_configuration.groovy
Configuration:
file:C:\Users\Administrator\.grails\SelfServiceBannerGeneralEventManager_configuration.groovy
Configuration: classpath:WebAppExtensibilityConfig.class
2017-02-18 03:49:22,471 [localhost-startStop-1] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
'$HOME/.grails/banner_configuration.groovy'
2017-02-18 03:49:22,526 [localhost-startStop-1] INFO
configuration.ApplicationConfigurationUtils - Using configuration file
'$HOME/.grails/SelfServiceBannerGeneralEventManager_configuration.groovy'
2017-02-18 03:49:22,664 [localhost-startStop-1] INFO

```

```
configuration.ApplicationConfigurationUtils - Using configuration file
WebAppExtensibilityConfig.class from the classpath (e.g., from within the war file)
Configuration: file:C:\Users\Administrator\.grails/banner_configuration.groovy
Configuration:
file:C:\Users\Administrator\.grails/SelfServiceBannerGeneralEventManager_configuration.groovy
Configuration: classpath:WebAppExtensibilityConfig.class

Configuring Spring Security Core ...
... finished configuring Spring Security Core

|Server running. Browse to
http://localhost:8080/SelfServiceBannerGeneralEventManager
```

6. From the browser, verify the application is running.
7. **Ctrl-C** to stop the app and close the command line window.

Lab 3 - Import Code and Configure Parameters

What You'll Learn

In this module, you learn to:

- ☐ Validate the environment setup with Grails
- ☐ Validate the Banner 9 General Events Self Service Banner application

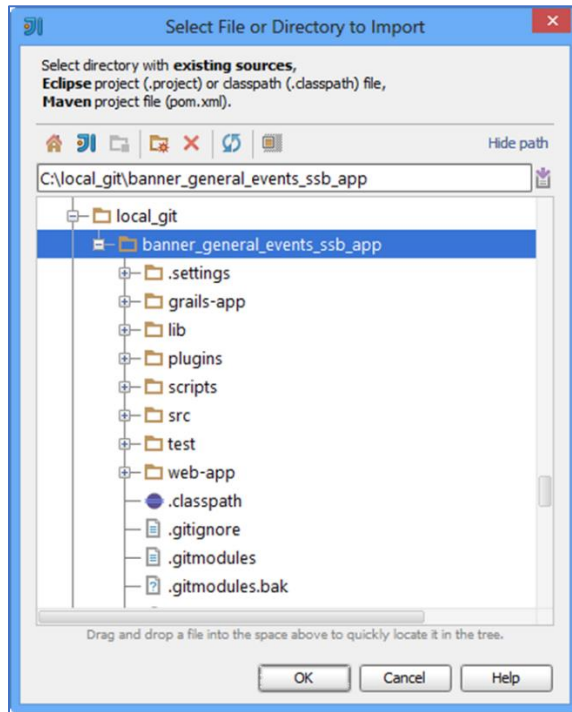
Lab Overview

The goal of this lab is to provide you with the skills needed to import the Banner 9 Self Service in IntelliJ for development and testing. In this exercise, you will configure the settings for memory, import the source code from git, and add new runtime configurations for the project.

Notes

Importing Source Code into your IDE

1. Open IntelliJ.
2. Go to **File > New > Project from Existing Sources**.
Older versions of IntelliJ may have **File > Import Project**.
3. Click the directory of the local version of the **banner_general_events_ssb_app**.

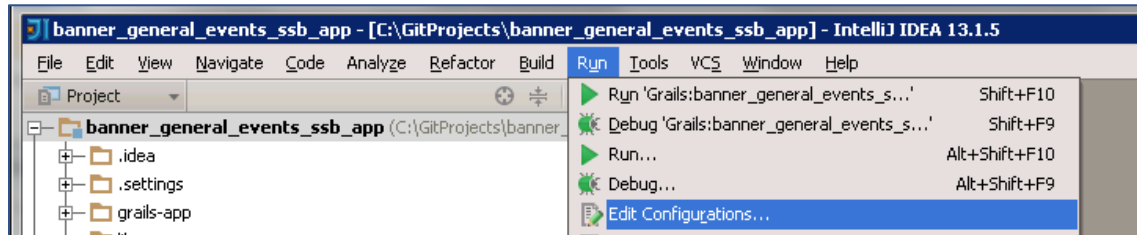


4. Select **Create Project from existing sources**.
5. Use the defaults for project name and location.
6. Leave the sources the same and it should show up as a grails project.
7. Select the **1.7 SDK**.
8. Select the **Grails 2.5.0 library**.
9. Leave the Spring Framework dialog box as is and click **Finish**.
10. Depending on whether other projects are open in IntelliJ, you might get prompted if the project should go in the current window or new window. Select **Current Window**.

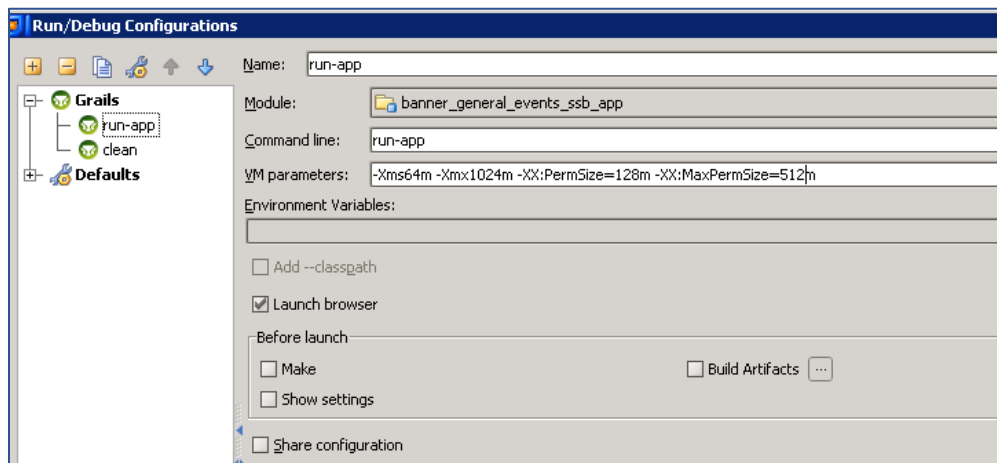
Setup the Environment

In your Virtual Machine, set up Run and Clean commands to use special Virtual Memory (VM) commands needed for all Banner 9 projects.

1. Choose **Run > Edit Configurations**.



2. Set up a run-app configuration as shown below.



- a. Confirm the use of the VM option:

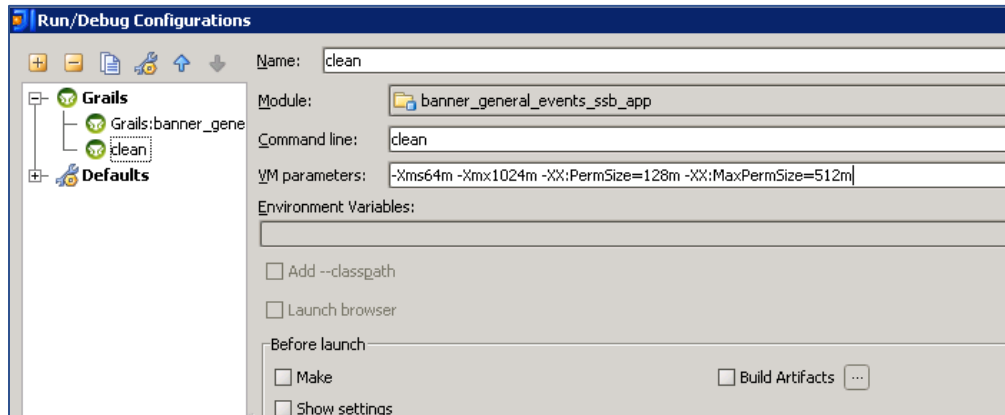
```
-Xms64m -Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=512m
```

You don't have to type this in – it is provided for you in your hard drive in the file
D:\labfiles\lab3\vm_options.txt.

- b. Check the **launch browser** option. Confirm the **Make** option is unchecked.
- c. Click **Apply**.

Clean Configuration

3. Click the + sign to create a new configuration.
4. Choose to create a Grails configuration. Confirm the **Make** option and the **Launch Browser** buttons are unchecked.

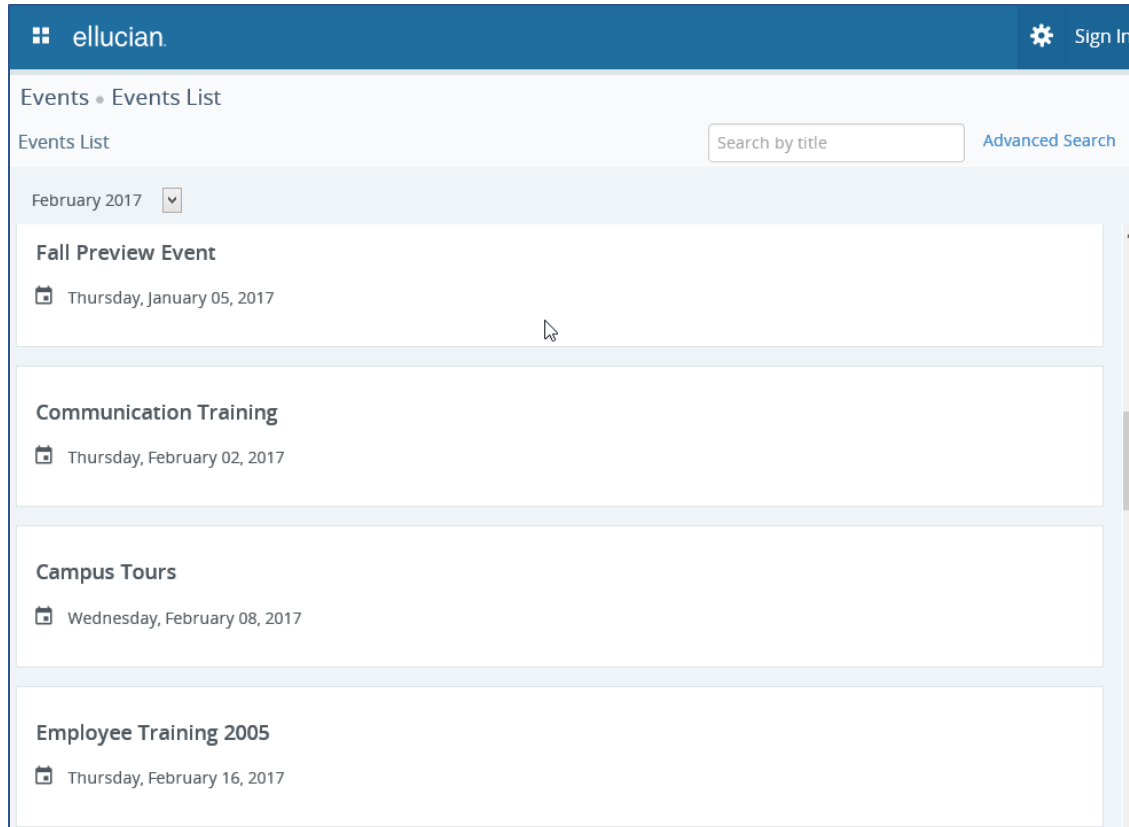


5. Click **Apply** and **Close**:
6. Start the General Events Self-Service Banner application using the run-app configuration. This make take a few minutes.
7. Verify that a list of events appear on the screen.

Run the Application

Before continuing with the exercise, verify that the Banner General Events application is working as designed. Follow the steps outlined below.

- From the startup page, select one of the non-restricted events. This results in a new page with details about that event to be displayed.



Note:

Your display may not look exactly like the one shown above. Different data may be loaded at the time of the exercises, showing different events. Please note that some events are restricted, and those are listed in orange. We will be working only with non-restricted events.

9. From the detail page, click the blue **Register** buttons to register for that particular event.
10. Click **Create Account**.

Note: Some events are restricted, and those are listed in orange, like the first one, below. In this scenario, you will only be working with non-restricted events.

The screenshot shows the ellucian application interface. At the top, there's a blue header with the ellucian logo and a 'Sign In' button. Below the header, a breadcrumb trail reads 'Events • Events List • Event Details'. A yellow warning banner states 'Please sign in or create an account to continue registration.' with 'Create Account' and 'Cancel' buttons. The main content area is titled 'Fall Preview Event' and includes a 'Like 0' button. Under 'Information Sessions', the event details are listed: 'Thursday, January 05, 2017 - Sunday, March 05, 2017', '9:00 AM-12:00 PM', and 'Main, Freed Center Building, 100'. A blue 'Register' button is at the bottom.

The following screen appears. The successful operation of the App is verified.

The screenshot shows the 'Account Registration' page in the ellucian application. The breadcrumb trail is 'Events • Events List • Event Details • Account Registration'. The page is divided into two columns of input fields. The left column includes: 'Last Name*' (text input), 'Surname Prefix' (text input), 'Preferred First Name' (text input), 'Date of Birth' (calendar input), 'E-mail Address*' (text input), 'Phone' (split into Country, Area Code, Number, and Extension), 'Username and Password' section with 'Username' (text input), 'Password*' (text input), and 'Re-Type Password*' (text input). The right column includes: 'Street Address Line 3' (text input), 'Street Address Line 4' (text input), 'Zip or Postal Code' (text input), 'City' (text input), 'State or Province' (dropdown menu), 'County' (dropdown menu), and 'Country' (dropdown menu).

Summary

This ends part one of the course agenda.

Part 1 Common Fundamentals

1. Tools

2. Basic Groovy
Grails Code

3. MVC and
IntelliJ

Lab 1 –
Grails and
IntelliJ Basics

4. RESTful Web
Application

5. Git

Lab 2 –
Use Git to
Download a
Banner 9 Self-
Service App

Lab 3 –
Import Code to
Configure
Parameters

Take a few moments to go over your notes. What questions do you have?



Lab 4 - Extend Apps with New Data – Add a Comment Field to Account Registration Page

What You'll Learn

In this module, you learn to:

- ☐ Practice finding files in need of modification
- ☐ Add a comment field to account registration
- ☐ Extend UI Pages
- ☐ Launch the General Events SSB Application and Test Application

Lab Overview

The goal of this lab is to provide you with the hands-on skills needed to extend applications with new data using Banner 9 Self-Service Banner.

Notes

Scenario

Add a field called comment to the AccountRegistration page for display. The desired placement is shown in the following screenshot. This involves modifying the backend database table as well as the domain model.

The screenshot shows the 'Account Registration' page in the ellucian application. The page has a blue header with the ellucian logo and a 'Sign In' link. Below the header is a breadcrumb trail: 'Events > Events List > Event Details > Account Registration'. The main content area contains two columns of form fields. The left column includes: 'First Name' (with a red asterisk), 'Middle Name', 'Last Name' (with a red asterisk), 'Surname Prefix', 'Preferred First Name', 'Date of Birth' (with a calendar icon), 'E-mail Address' (with a red asterisk), and 'Phone' (with sub-fields for Country, Area Co, Numbe, and Extensio). The right column includes: 'Street Address', 'Street Address Line 2', 'Street Address Line 3', 'Street Address Line 4', 'Zip or Postal Code', 'City' (with a red asterisk), 'State or Province' (a dropdown menu), 'County' (a dropdown menu), and 'Country' (a dropdown menu). At the bottom left, a red circle highlights the text 'We want to add a comment field here'.

Reference

Determine the tasks for an extension by reviewing a checklist of files that may need to be modified and locating those files prior to starting modification work. A basic checklist of files to consider includes:

- Backend Oracle table and Domain Model
- DB Table View, Instead of Triggers, APIs
- HTML files
- JavaScript Files
- Decorator File
- messages.properties
- Controller
- Service File

You will not always need to extend all these items, but it is important that they be considered when you are planning your work.

Add a Comment Field to Account Registration

Implementation Steps

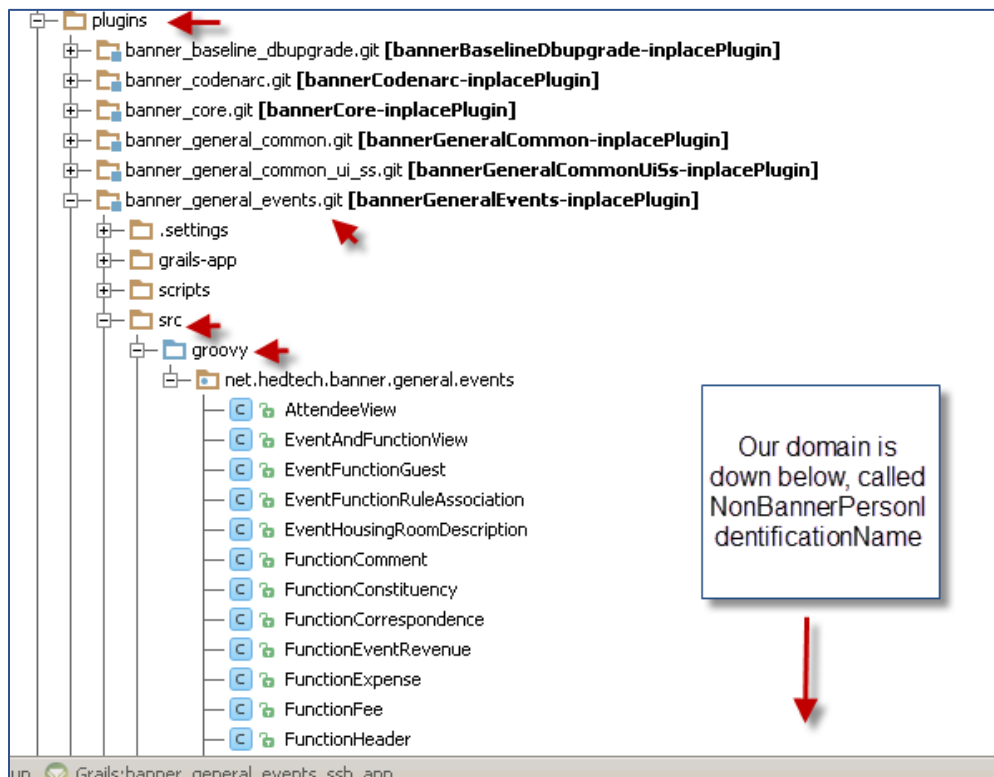
This lab uses the addition of a field to your database for the comment field. This has already been completed for you.

Review the files located at D:\labfiles\lab4, if you have questions about what was changed, and what they do.

The GV_GENIDEN view must be changed as that is the one referenced in the NonBannerPersonIdentificationName file. We also modify the following views, triggers and APIs:

- dml_geniden
- gp_geniden
- geniden_view_create_trg
- geniden_view_update_trg
- gv_geniden

1. Open the **NonBannerPersonIdentificationName.groovy** domain to add a column corresponding to the newly added database column.
2. In the IDE, locate the NonBannerPersonIdentificationName.groovy file at **banner_general_events_ssb_app > plugins > banner_general_events.git > src > groovy > net.hedtech.banner.general.events**.



3. Open and extend the **NonBannerPersonIdentificationName** file, by adding the code below after the last item (County). Copy and paste the code from the field you are mimicking (ZIP Code) This adds the new column to the domain.

```
/**
 * COMMENT: This field maintains the comment associated with the person.
 */
@Column(name = "GENIDEN_COMMENT", length = 255)
String comment
```

4. Edit the toString() method to add the comment.

```
        surnamePrefix=$surnamePrefix,
        systemCode=$systemCode,
        telephoneType=$telephoneType,
        comment=$comment,
        zip=$zip]"""
    }
```

5. Edit the equals() method to add the comment.

```
        if (loadDate != that.loadDate) return false
        if (phoneArea != that.phoneArea) return false
        if (pidm != that.pidm) return false
        if (comment != that.comment) return false
        return true
    }
```

6. Edit the hashCode() method to add the comment.

```
        result = 31 * result + (loadDate != null ? loadDate.hashCode() : 0)
        result = 31 * result + (phoneArea != null ? phoneArea.hashCode() : 0)
        result = 31 * result + (pidm != null ? pidm.hashCode() : 0)
        result = 31 * result + (comment != null ? comment.hashCode() : 0)
        return result
    }
```

7. Edit the constraints closure, adding the following to specify that the comment is nullable, along with the existing constraints. Comment will not be a required field, so it will be nullable. Also, we will give it a maxsize of 255:

```
        comment(nullable: true, maxSize: 200)
```

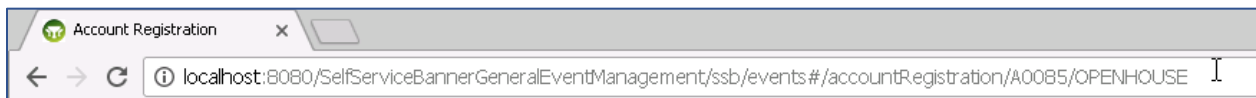

- Many pages use Decorators, which are intermediate groovy files that overlay the domain object. Add any new columns added to the page to the appropriate decorator file. To find the correct decorator, trace through the **AccountRegistrationController's accountRegistration** method.

You would see that the functionality we are tracing through has to do with creating a NewUser. The object being created is newUser. This has a decorator class associated called NewUserDecorator.

Add the comment property to the NewUserDecorator.groovy file found at banner_general_events_ssb_app/src/groovy/net.hedtech.banner/general.events.

```
.  
.   
.   
    NationDecorator nation  
  
    String securityQuestion  
  
    String answer  
    String comment  
}
```

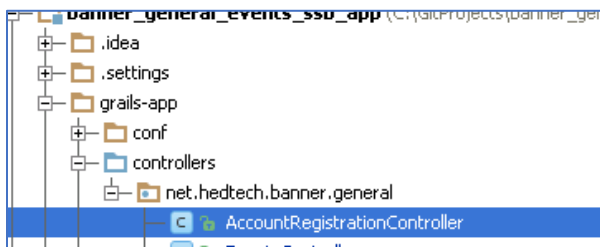
- Recall the flow of the rendering logic. Review the URL after the ssb and #, so accountRegistration.



10. Find the state for `accountCreation` in `/web-app/ssb/eventsApp/config.route.js` to see which controller is used.

```
.state('accountCreation', {
  url: '/accountRegistration/:eventId/:functionId',
  templateUrl: '../ssb/eventsApp/accountRegistration/accountRegistration.html',
  controller: 'AccountRegistrationController as accountRegistration',
  resolve: {
    accountCreationResolve: function (AccountRegistrationService) {
      return AccountRegistrationService.getDefaultDataMap();
    },
    eventId: function ($stateParams) {
      if (angular.isDefined($stateParams.eventId)) {
        return $stateParams.eventId;
      } else {
        return null;
      }
    }
  },
  onEnter: function (eventId) {
    this.data = {
      'pageTitle': 'events.breadcrumb.createAccount',
      'breadcrumb': [
        {label: 'events.breadcrumb.events'},
        {label: 'events.breadcrumb.eventList', url: '/eventList'},
        {label: 'events.breadcrumb.eventDetail', url: '/eventDetail/' + eventId + '/'},
        {label: 'events.breadcrumb.createAccount'}
      ]
    };
  }
});
```

11. Locate the controller.



12. From the controller, find the action method with the name `accountRegistration`.

Several tasks are to be completed. For this exercise, we are only going to concentrate on one of those tasks – the creation of the desired new record in the back-end database.

Some of the other tasks are reviewed in the following exercise.

13. Analyze the controller, the task of creating our new user account record is completed by calling a Service in the below code:

```
70 }  
71 }  
72  
73 bindData(newUser, params)  
74 proxyAccessAccountService.createNewUser(newUser)  
75 //Authenticate the newly created user, so explicit log in is  
76 //Get the list of Roles  
77 def config = SpringSecurityUtils.securityConfig
```

The Controller relies on this Service to create a new user, and so the Controller is not edited in this exercise. Instead, review this Service, and to the method shown, called **createNewUser**.

The service name is **proxyAccessAccountService**. To find the location of this file, use the key combination **CTRL +clicking the method name** to access the definition of that method.

14. In the IDE, the ProxyAccessAccountService.groovy file is located at:
banner_general_events_ssb_app > grails-app > services > net.hedtech.banner.general.events.

As with the Controller, locate the method called **createNewUser**. Review the code and see that it calls a method called **createGenIden**:

```
}
NonBannerPersonIdentificationName genidenInfo = createGenIden(newUser,
```

15. From the **createGenIden** code, find where the new record for display, **NonBannerPersonIdentificationName** is instantiated.
 16. Add code for the comment as shown below:

```
private NonBannerPersonIdentificationName createGenIden(newUser, state, nation) {
    def nonBannerPersonIdentificationName = new NonBannerPersonIdentificationName(systemCode: "EVENT_REGD_USER", entityCode: "P", lastName: newUser.lastName,
        firstName: newUser.firstName, streetLine1: newUser.streetAddressLine1, streetLine2: newUser.streetAddressLine2,
        city: newUser.city, state: state, zip: newUser.zip,
        nation: nation, phoneArea: newUser.phoneArea, phoneNumber: newUser.phoneNumber, phoneExtension: newUser.phoneExtension,
        countryPhone: newUser.countryCodePhone, birthDate: newUser.dateOfBirth, emailId: newUser.emailAddress, surnamePrefix: newUser.surNamePrefix, comment: newUser.comment)
    //Need to set the mail,phone,ssn and email types for which query GENOTYPE
    def session = sessionFactory.getCurrentSession()

    "P", lastName: newUser.lastName,
    sion,
    refix: newUser.surNamePrefix, comment: newUser.comment)
```

Extend UI Apps

1. Add the Comment field to the accountRegistration.html file found at: **banner_general_events_ssb_app/web-app/ssb/eventsApp/accountRegistration/**.
2. Open the file and add a comment to one of the lines in this file. Any comment will do (comments start with `<!-- -->`).
3. **CTRL-S** to save. Entering a change before you save forces a history snapshot to be saved when you have the situation of fresh file.
4. Find the lines of code that display the 'Address Line 3', information.
5. Copy and paste that code and extend it to contain information for our field, **comment**.

Note: Where you place the code is very important. If you want it to be displayed after the 'phone' field on your page, and we do, then you need to put this code after the code after phone group DIV but still inside the userInfo DIV, so it appears on the left side of the SSB page.



6. Remove the ng-if that checks the configMap.display and reference the comment field instead of the addr line3.
7. Remove the ng-required.

```
<div class="form-group" show-errors xe-field="comment">
<label for="comment" class="control-label">
{{'events.registration.profile.comment' | i18n}}
</label>
<input id="comment" name="comment" data-ngmodel="
accountRegistration.newUser.comment" type="text" class="formcontrol
input-sm reqInput"
placeholder="{{'events.registration.profile.comment' |
i18n}}"
aria-required="false" maxlength="200">
```

8. Note the labels used in the new code to be displayed:
- events.registration.profile.comment
 - events.message.comment.invalid

The corresponding entry should be added to the messages.properties file found at **banner_general_events_ssb_app /grails-app/i18n**.

```
events.registration.profile.state=State or Province  
events.registration.profile.zip=Zip or Postal Code  
events.registration.profile.country=Country  
events.registration.profile.comment=Comments  
events.registration.login.username=Username  
events.registration.login.password=Password  
events.registration.login.passwordconfirm=Re-Type Password
```

```
events.message.city.required=You must enter the City.  
events.message.state.required=You must select the State or Province.  
events.message.zip.required=You must enter the Zip or Postal Code.  
events.message.country.required=You must select the Country.  
events.message.password.required=You must enter the Password.  
events.message.comment.invalid=Your comment cannot be larger than 200  
events.message.confirm.password.required=You must re-type the password.  
event.message.requiredFields=Please enter values for required fields.  
events.message.confirm.password.passwordMismatch=Please check that your password
```

Launch General Events Self-Service Application and Test Application

Launch the application and navigate to the Account Creation page to see the updated UI and perform the operations below to confirm that the Platform-offered functionalities are working as expected.

1. Select one of the Events.
2. On the detail page, click the **Register** button.
3. Click the **Create Account** button.
4. Complete the form to create a new user and be sure to populate the new **Comment** field. Remember the name you entered, exactly, the last name and first name.
5. **Save** the changes.

Summary and Notes

- In these exercises, you followed client-side JavaScript to the server-side controller responsible for displaying a form.
- You updated the appropriate groovy domain object, to allow an additional field to pass to the persistence layer.
- You then moved back to the client-side and updated JavaScript to allow the new field to be bound to the AngularJS model, and updated the appropriate HTML fragment to make AngularJS aware of the new data field.
- Finally, you added the new field's prompts to the messages.properties file.



Write what you have you learned in this section and how will you apply it as your school.

Lab 5 - Add the Gender field with Validation to the Account Registration Page

What You'll Learn

In this module, you learn to:

- ☐ Launch the General Events SSB Application and Test Application Add a New field to your MVC components
- ☐ Add validation that are executed on the client side.

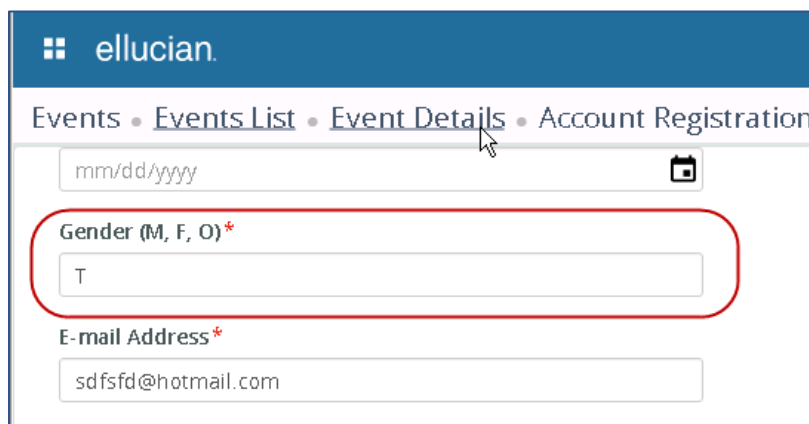
Lab Overview

The goal of this lab is to provide practice of adding another field to the application, complete with client-side validation

Scenario

Add a field called 'gender' to the Account Registration page. The following screenshot shows the desired placement.

Add a validation code to the Account Registration form to ensure that if the user does not enter a 'M', 'F' or 'O', then an alert will appear to prompt the user for that input.



ellucian.

Events • Events List • Event Details • Account Registration

mm/dd/yyyy

Gender (M, F, O) *

T

E-mail Address *

sdfsfd@hotmail.com

1. The data is added to the back-end table for you. Perform a describe, and the screen is similar to the one shown below:

```
SQL> desc geniden;
```

Name	Null?	Type
GENIDEN_GIDM	NOT NULL	NUMBER(8)
GENIDEN_SYST_CODE	NOT NULL	VARCHAR2(30 CHAR)
GENIDEN_ENTITY_CODE	NOT NULL	VARCHAR2(1 CHAR)
GENIDEN_ID	NOT NULL	VARCHAR2(9 CHAR)
GENIDEN_LAST_NAME		VARCHAR2(60 CHAR)
GENIDEN_FIRST_NAME		VARCHAR2(60 CHAR)
GENIDEN_MI		VARCHAR2(60 CHAR)
GENIDEN_HOUSE_NUMBER		VARCHAR2(10 CHAR)
GENIDEN_STREET_LINE1		VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE2		VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE3		VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE4		VARCHAR2(75 CHAR)
GENIDEN_CITY		VARCHAR2(50 CHAR)
GENIDEN_STAT_CODE		VARCHAR2(3 CHAR)
GENIDEN_ZIP		VARCHAR2(30 CHAR)
GENIDEN_MAIN_CODE		VARCHAR2(5 CHAR)
GENIDEN_PHONE_AREA		VARCHAR2(6 CHAR)
GENIDEN_PHONE_NUMBER		VARCHAR2(12 CHAR)
GENIDEN_PHONE_EXT		VARCHAR2(10 CHAR)
GENIDEN_CTRY_CODE_PHONE		VARCHAR2(4 CHAR)
GENIDEN_SSN		VARCHAR2(15 CHAR)
GENIDEN_BIRTH_DATE		DATE
GENIDEN_SEX		VARCHAR2(1 CHAR)
GENIDEN_EMAIL_ADDRESS		VARCHAR2(120 CHAR)
GENIDEN_OTHER_CODE		VARCHAR2(20 CHAR)

2. Verify this field is correctly included in the Domain Model. The domain model was identified in the previous lab as:
[plugins/banner_general_events.git/src/groovy/net.hedtech.banner.general.events/NonBannerPersonIdentificationName.groovy](#).



Check off each step as they are completed.
 Proceed through each of steps carefully.

3. Search for field name, **GENIDEN_SEX**. Verify it is included in the column mappings within the domain. Note the variable / property name being used.
4. Ensure this field is included in the toString, equals, and hashCode methods.
 Examine the constraints closure. Is there an entry for this field? As it is a nullable field leave the constraint as-is.
5. Review the named queries within this file.

Do the named queries list out the selected fields one by one?

If they do not, then all fields will be selected by default.

If this was the case, you need to add your desired field to this list. Read the fetch functions to include it where needed.



In this example, this work is not necessary.

The domain is ready.

Add the Gender Field

Any displayed field on the form needs to be included in the same Decorator used in the previous exercise, **NewUserDecorator.groovy**, found at:

banner_general_events_ssb_app/src/groovy/net.hedtech.banner/general.events.

6. Add the following:

```
String sex
```

7. Open the Controller/Service, AccountRegistrationController.

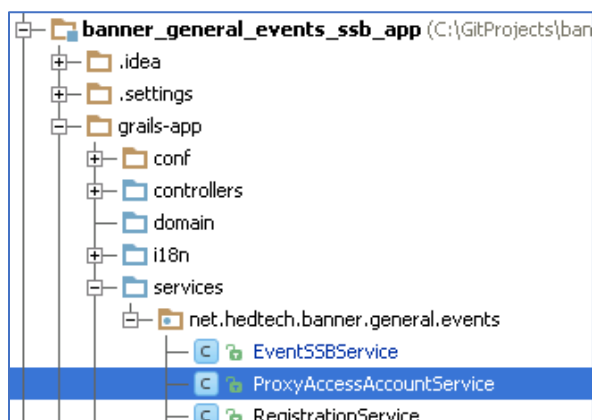
What does the Controller do?

- a. Fetches the data, typically using a Service
- b. Passes the data to web page.

This typically takes time and effort – read the code and understand what it does. In this example, we have already researched how the data is fetched, which is by using the service code we worked with previously. In the Controller, this is:

```
70 }
71 }
72
73 bindData(newUser, params)
74 proxyAccessAccountService.createNewUser(newUser)
75 //Authenticate the newly created user, so explicit log in is
76 //Get the list of Roles
77 def config = SpringSecurityUtils.securityConfig
```

8. Open this file, **proxyAccessAccountService.groovy**:



9. Find the method **createNewUser** and examine the code. Locate the domain and the call to the method **createGenIden**.
The new comment field code created in the previous lab is seen here. The fields displayed in the web page are also listed.
10. Add an entry for the sex / gender field after the entry for **comment**.

```

n,
newUser.surNamePrefix, comment:newUser.comment, sex:newUser.sex)

```

Code Location

11. Add the sex / gender field to banner_general_events_ssb_app/web-app/ssb/eventsApp/accountRegistration/ **accountRegistration.html**.
12. Place this field where it is expected to appear in the finished product of the form.
We want this to display under the date of birth information. Therefore, paste the new code after the code for the date of birth code.



Copy and paste the code created for the comment field to reduce any typing errors.

13. Modify the code to represent the **sex** field.
14. Make the maxlength = 1.

```

<div class="form-group" show-errors xe-field="sex">
  <label for="sex" class="control-label">
    {{'events.registration.profile.sex' | i18n}}<span class="asterisk-
input"></span>
  </label>
  <input id="sex" name="sex" data-ng-
model="accountRegistration.newUser.sex" type="text" class="form-control
input-sm reqInput"
    placeholder="{{'events.registration.profile.sex' | i18n}}"
    aria-required="false" maxlength="1"
    aria-describedby="sexRequiredMessage"/>
  <span id="sexRequiredMessage" class="help-block" ng-
if="accountRegistration.accountRegistrationForm.sex.$touched &&
accountRegistration.accountRegistrationForm.sex.$invalid">{{'events.message.s
ex.invalid' | i18n}}</span>
</div>

```

15. Open the **messages.properties** file and add the label, and the value for that label.
16. Apply the label Gender (M, F, or O).



Place the label in the same area as the other labels that start with **events.registration.profile**.

```
events.registration.profile.sex=Gender (M, F or O)
```

17. **Save All.**

Run and Test App

1. Run the app and test.
2. Verify that a gender value can be saved.
3. Enter a valid entry, **M, F, or O**.
4. From Oracle PL/SQL developer, query GENIDEN to verify that the record appears in the database.

Example query:

```
select * from GENIDEN where geniden_last_name = 'xxx';)
```

5. Navigate to the **AccountRegistration** form display in your App. Decide if the Gender field can be resized smaller.
6. Locate the you code added to the **AccountRegistration** html file for the Gender field.
7. Reduce the size of the field.
8. **Save All.**
9. To test, refresh your screen.

Add Validation

1. Review **message.properties** and locate any error messages. The labels reference the ProxyAccessAccountService groovy file.

```
net.hedtech.banner.general.events.ProxyAccessAccountService.phoneNumber.extension.required=Phone extension is required.
net.hedtech.banner.general.events.ProxyAccessAccountService.dateOfBirth.invalid=Please enter a valid date of birth.
net.hedtech.banner.general.events.ProxyAccessAccountService.dateOfBirth.futureDate=Future birth date not permitted.
events.message.date.of.birth.format=Please enter a valid date of birth.
events.message.date.of.birth.future=Future birth date not permitted.
events.message.date.of.birth.checking=Validating Date of Birth...
```

2. Review the logic of the Date of Birth field.

```
if (newUser.dateOfBirth) {
    try {
        newUser.birthDate = LocalizeUtil.parseDate(newUser.dateOfBirth)
    } catch (e) {
        if (e.wrapperException instanceof ParseException) {
            throw new ApplicationException(ProxyAccessAccountService, "@@r1:dateOfBirth.invalid@@")
        }
    }
    Date today = new Date()
    if (newUser.birthDate > today) {
        throw new ApplicationException(ProxyAccessAccountService, "@@r1:dateOfBirth.futureDate@@")
    }
}
```

3. Add validation to the ProxyAccessAccountService.groovy by copying the code for another error and changing it to reference the Date of Birth field.
4. Note the way they reference the label when throwing an application error.



To follow good coding standards, place this code after the Date of Birth code as Date of Birth is displayed before gender / sex on the web page.

```
if (newUser.sex) {
    if (newUser.sex.toUpperCase() != 'M' && newUser.sex.toUpperCase() != 'F' &&
        newUser.sex.toUpperCase() != 'O') {
        throw new ApplicationException(ProxyAccessAccountService,
            "@@r1:sex.invalid@@")
    }
}
```

5. Add a new label in **message.properties** for your error.

```
net.hedtech.banner.general.events.ProxyAccessAccountService.sex.invalid=Gender must be M, F, or O.
```

6. To test, restart your app and test your data entry as you have done before. Test, ensuring that an alert box appears if you enter a value that is not M, F, or O.

Events • Events List • Event Details • Account Registration

mm/dd/yyyy sdf sdfsd

Gender (M, F, O)* State or Province

Gender must be M, F, or O.

Alternate Validation Using Directives

An alternative method to provide validation is achieved by using Directives.

1. AngularJS method is used within the Banner logic and it provides visual feedback to the user. Review how this is performed for the email field.

The following screen is in two parts to aid readability.

```
<div class="form-group" show-errors xe-field="emailAddress">
  <label for="emailAddress" class="control-label">
    {{'events.registration.profile.email' | i18n}}<span class="asterisk-input"></span>
  </label>
  <input id="emailAddress" name="emailAddress" data-ng-model="accountRegistration.newUser
    placeholder="{{'events.registration.profile.email' | i18n}}" required aria-require
    aria-describedby="emailAddressRequiredMessage emailAddressInvalidMessage emailAdd
  <span class="help-block" ng-if="accountRegistration.accountRegistrationForm.emailAddress
    <span id="emailAddressRequiredMessage" ng-if="accountRegistration.accountRegistratio
    <span id="emailAddressInvalidMessage" ng-if="accountRegistration.accountRegistration
    <span id="emailAddressDuplicateMessage" ng-if="accountRegistration.accountRegistrati
    <span id="emailAddressCheckingMessage" ng-if="accountRegistration.accountRegistratio
  </span>
</div>
```

```
er.emailAddress" class="form-control input-sm"
ired="true" maxlength="128" xe-email duplicate-email
ddressDuplicateMessage emailAddressCheckingMessage" />
ss.$touched && accountRegistration.accountRegistrationForm.emailAddress.$invalid">
ionForm.emailAddress.$error.required"> {{'events.message.email.address.required' | i18n}}</span>
onForm.emailAddress.$error.email">{{'events.message.email.address.invalid' | i18n}}</span>
tionForm.emailAddress.$error.duplicate">{{'events.message.email.address.duplicate' | i18n}}</span>
ionForm.emailAddress.CHECKING_EMAIL">{{'events.message.email.address.checking' | i18n}}</span>
```

2. SPAN tags are visible if the error condition is true, notice the ng-if. If visible, the label with the error message shows in red below the field. Review the fields below.

E-mail Address*

You must enter the E-mail Address.

E-mail Address*

Please enter a valid e-mail address.

E-mail Address*

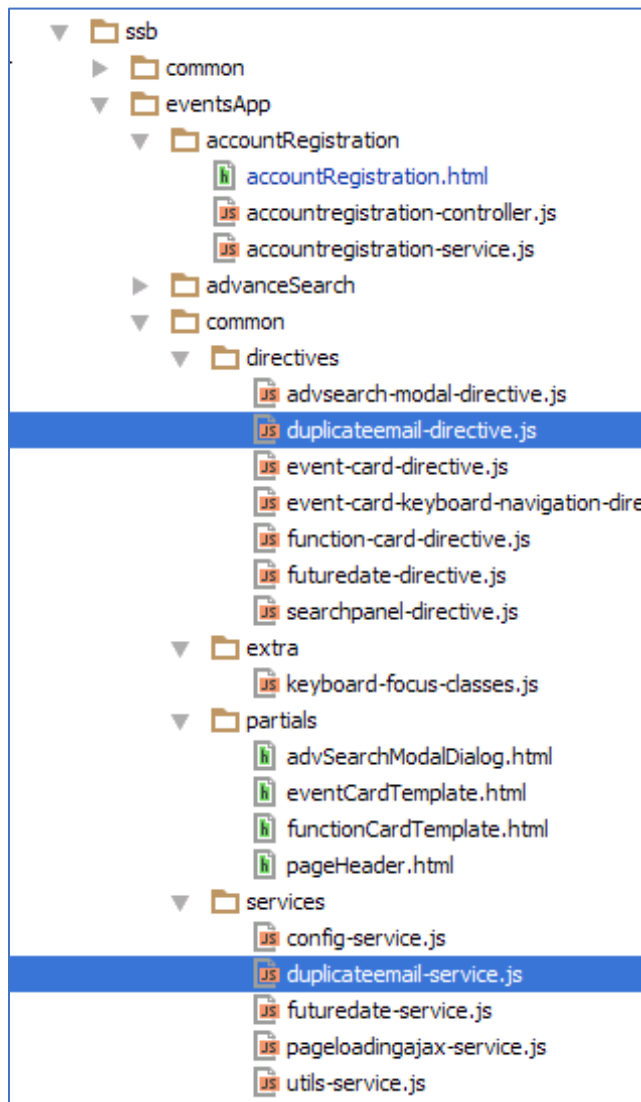
E-mail address already in use. Please sign in using this email address or provide an alternative email.

The duplicate-email within the input tag tells the input to follow the directive which is named duplicateEmail.



In HTML, the directives are lowercase with dashes, and in the JS, they are camelCase. This helps when searching in files to see where it is used.

3. In the directives folder, view the highlighted JavaScript files below to see how it validates the data.



The duplicate check queries the database.

In this lab, we do not go into further depth as it requires a greater level of AngularJS expertise.

Summary and Notes

- You added another field, this time with validation.
- First, you added the new field to the server-side domain object.
- Next, you updated the client-side JavaScript to include the new field in the client's model and updated the HTML template so the new field would display.
- You then updated the `messages.properties` file to provide a user prompt for the field.
- Finally, back on the server-side, you updated the service responsible for processing our form's data with logic to reject invalid input.



Take a few minutes and add your own notes and observations from the previous section.

[illegible]

Lab 6 - Add an Email Type Field with Validation to the Account Registration Page

What You'll Learn

In this module, you learn to:

- ☐ Add a drop-down field to a form.
- ☐ Add an email type field.

Lab Overview

In this lab, practice is provided to add a field with a drop-down box to a form. Additionally, it is shown how to add an email type field to the AccountRegistration form the text box for email address. The final product will appear as shown below:

The screenshot displays the 'Account Registration' page within a web application. The breadcrumb trail at the top reads: 'Events • [Events List](#) • [Event Details](#) • Account Registration'. The form contains several fields: 'Date of Birth' with a date input and a calendar icon; 'Gender (M, F, O)*' with a text input; and 'E-mail Address*' with a text input containing 'suzysmith@hotmail.com'. Below these is the 'E-mail Type' section, which features a dropdown menu. The dropdown is open, showing a list of options: 'Choose Email Type' (selected), 'Boulder E-mail', 'Business E-mail', 'Campus E-mail', 'Colorado Springs E-mail' (highlighted in blue), 'Corporate Headquarters', 'Corporate Subsidiary', 'Counselor's E-mail', 'Denver E-mail', and 'Facebook E-mail'. A mouse cursor is visible over the 'Colorado Springs E-mail' option.

1. Return to the lab in which the comment field was added to the AccountRegistration form. Review the steps performed to create it as the steps to add a drop-down to a field are similar.
2. The drop-down maps to a field with a foreign key relationship in the data source back-end Oracle table.
3. As before, copy the code of another field on your form with similar characteristics. Copy the drop-down box code from **state**.
4. Return to the Domain Model.
5. The back-end table is an Oracle View called GV_GENIDEN. The view and field appear as shown below:

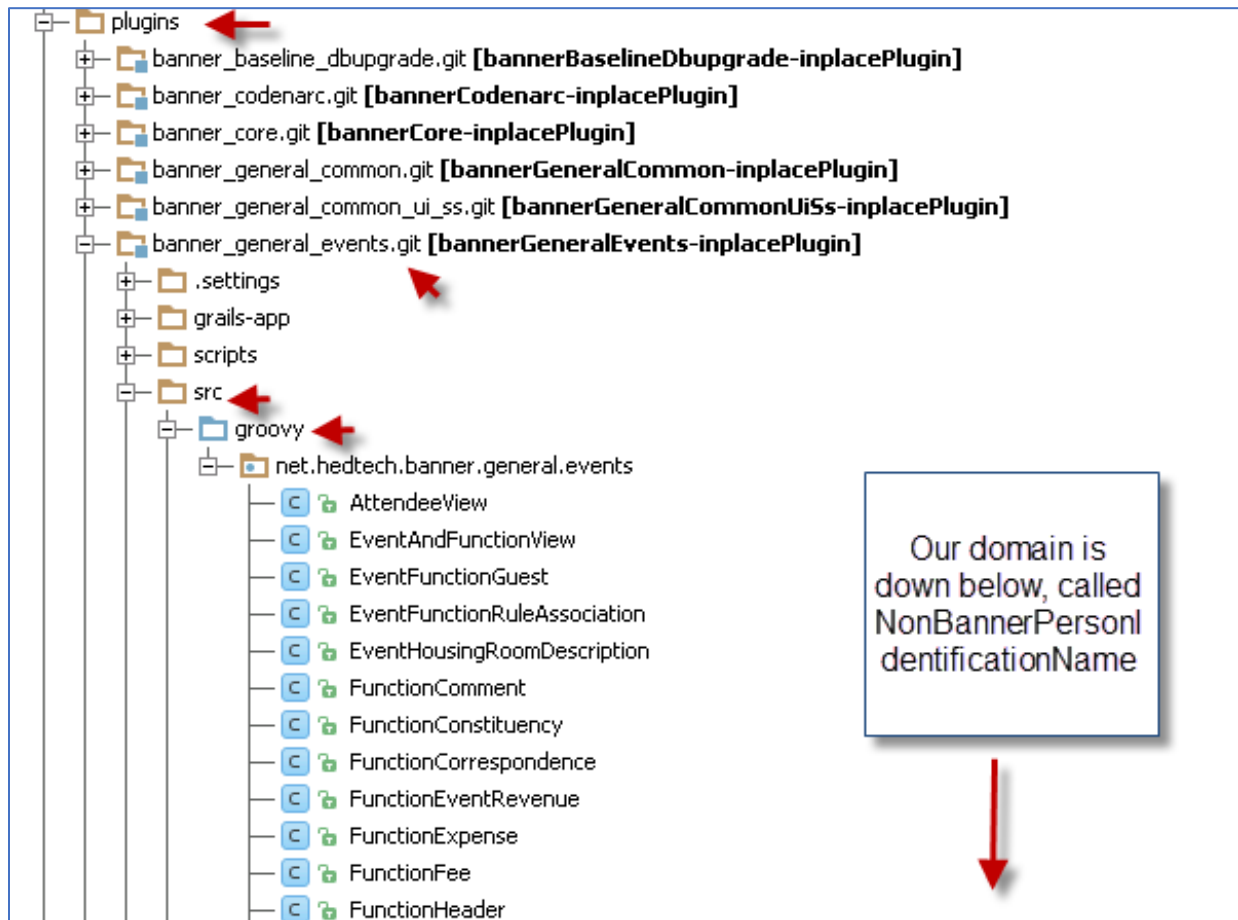
```

C:\Windows\system32\cmd.exe
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.7.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> desc gv_geniden
Name                               Null?    Type
-----
GENIDEN_GIDM                       NOT NULL NUMBER(8)
GENIDEN_SYST_CODE                   NOT NULL VARCHAR2(30 CHAR)
GENIDEN_ENTITY_CODE                 NOT NULL VARCHAR2(1 CHAR)
GENIDEN_ID                          NOT NULL VARCHAR2(9 CHAR)
GENIDEN_LAST_NAME                   VARCHAR2(60 CHAR)
GENIDEN_FIRST_NAME                  VARCHAR2(60 CHAR)
GENIDEN_MI                          VARCHAR2(60 CHAR)
GENIDEN_HOUSE_NUMBER                VARCHAR2(10 CHAR)
GENIDEN_STREET_LINE1                VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE2                VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE3                VARCHAR2(75 CHAR)
GENIDEN_STREET_LINE4                VARCHAR2(75 CHAR)
GENIDEN_CITY                        VARCHAR2(50 CHAR)
GENIDEN_STAT_CODE                   VARCHAR2(3 CHAR)
GENIDEN_ZIP                         VARCHAR2(30 CHAR)
GENIDEN_NATN_CODE                   VARCHAR2(5 CHAR)
GENIDEN_PHONE_AREA                  VARCHAR2(6 CHAR)
GENIDEN_PHONE_NUMBER                VARCHAR2(12 CHAR)
GENIDEN_PHONE_EXT                   VARCHAR2(10 CHAR)
GENIDEN_CTRY_CODE_PHONE              VARCHAR2(4 CHAR)
GENIDEN_SSN                         VARCHAR2(15 CHAR)
GENIDEN_BIRTH_DATE                  DATE
GENIDEN_SEX                         VARCHAR2(1 CHAR)
GENIDEN_EMAIL_ADDRESS               VARCHAR2(128 CHAR)
GENIDEN_ATYP_CODE                   VARCHAR2(2 CHAR)
GENIDEN_TELE_CODE                   VARCHAR2(4 CHAR)
GENIDEN_EMAL_CODE                   VARCHAR2(4 CHAR)
GENIDEN_ASRC_CODE                   VARCHAR2(4 CHAR)
GENIDEN_SURNAME_PREFIX              VARCHAR2(60 CHAR)
GENIDEN_PIDM                        NUMBER(8)
GENIDEN_MATCH STATUS                VARCHAR2(1 CHAR)

```

6. The Domain Model file name and location was found in the previous lab. Open the file in the editor as shown below:



7. Perform a search for the field name, **GENIDEN_EMAL_CODE**. Locate the one with a GORM (Groovy Object Relational Mapping) entry as shown below:

```
/**
 * EMAIL TYPE: This field maintains the default email type code for the person for matching purposes
 */
@ManyToOne
@JoinColumn([
@JoinColumn(name = "GENIDEN_EMAL_CODE", referencedColumnName = "GTVEMAL_CODE")
])
EmailType emailType
```

```
EmailType emailType
```

If your entry has red text, as shown above, it may or may not indicate a problem. The red text indicates a missing import or other missing dependency, which IntelliJ attempts to solve.

8. Hover the cursor over the red text and accept the solution if one is offered. The solution may appear in the form of a red-light bulb or as a hover bubble. Choose the automatic solution and the red text disappears.
9. If no solution was offered, or accepting the solution doesn't work, confirm the following import:

```
import net.hedtech.banner.general.system.EmailType
```

10. If there are other classes that are red in the file, and if, when the code is run, and unrecognized imports occur, import the whole package by:

```
import net.hedtech.banner.general.system.*
```



Note: The name of the OO variable is **emailType** and its type is **EmailType**. Copy these names to use later.



Note: The field is mapped differently than the comment field as this field has a foreign key relationship to the GTVEMAL_CODE field in the GTVEMAL table.

11. Query the GTVEMAL table and the values below appear.

```
C:\Windows\system32\cmd.exe
SQL> select distinct gtvemal_code from gtvemal order by gtvemal_code;
GTVEMAL_CODE
-----
AOL
BUS
BUSI
CAMP
CCS
CELL
DEN
DELT
EMAL
HOME
HPAG
GTVEMAL_CODE
-----
HQ
HSC
LONG
MA
PEML
PERS
POA
PWEB
SB
SCHL
UCB
GTVEMAL_CODE
-----
XL
23 rows selected.
```



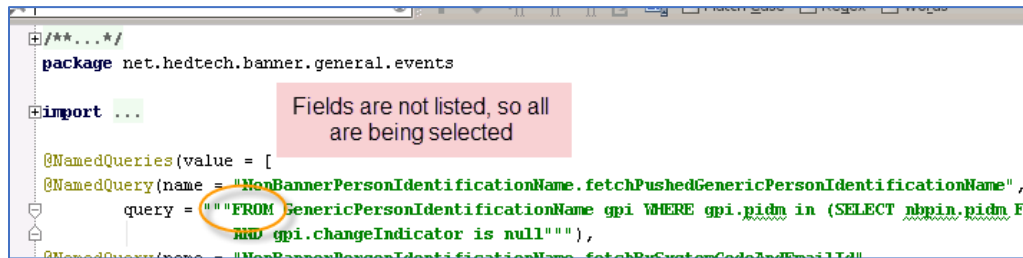
Note: Record these values to confirm in the final test.

12. From the domain file, review the HQL.

If the HQL lists out the fields being selected, then, all fields are selected.

If the select statement listed out the fields, verify that the field of interest was explicitly selected.

In this case, the fields are not selected and so all the fields are selected.



```
/**...*/
package net.hedtech.banner.general.events

import ...

@NamedQueries(value = [
    @NamedQuery(name = "NonBannerPersonIdentificationName.fetchPushedGenericPersonIdentificationName",
        query = "FROM GenericPersonIdentificationName gpi WHERE gpi.pidm in (SELECT nbpin.pidm F
            AND gpi.changeIndicator is null)"
    ],
    @NamedQuery(name = "NonBannerPersonIdentificationName.fetchRefCustomerCodeAndEmailId"
```

Therefore, there is no need to extend the Domain Model.

Working with Decorator

If the web page involves a Decorator, it must be included in the field.

As before, this is located by reading and tracing through the code.

The decorator we need is the `NewUserDecorator.groovy` file found at

`banner_general_events_ssb_app/src/groovy/net.hedtech.banner/general.events.`

1. Open this file in the editor.
2. Search for the variable name previously found, **`emailType`**. It is not found in the file and will need to be added.
3. Add the file and declare the type.



Again, to follow good coding standards, as we specified that this will appear after the email field in our finished product, it is then added after this field's entry in the Decorator file.

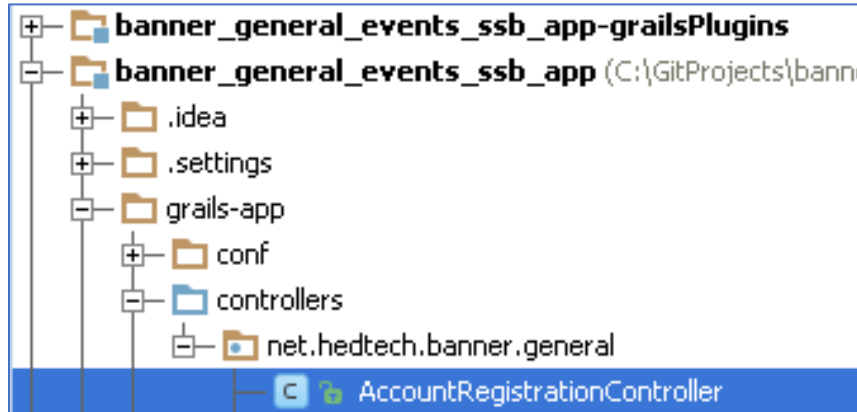
4. The **`state`** and **`nation`** are of the Decorator type, so the code can be copied for this field. Additionally, it is good to add a comment explaining the change, your name, date, and other supporting documentation, like the change request information your school may use.

```
/**
 * Added for Lab
 */
EmailTypeDecorator emailType
```

5. The new entry has a problem with the **`EmailTypeDecorator`** displaying as red text. The problem is that the `EmailTypeDecorator` needs to be created and imported into this file.
6. We will create the `EmailTypeDecorator` file on the next page.
7. Confirm what code exists for the copied field. Perform a search for any code that handles **`state`**. There should be no other coding, aside from the entry we already added.

Extend the Controller

The name of the controller, according to convention, is **AccountRegistrationController**.



1. Open the **AccountRegistrationController** file.
2. Search for code performed for the copied State field.
Numerous results are returned for the State field. These will be duplicated for the new field, **emailType**,
3. Start at the top of the file and copy and paste the **State** code.
4. Modify the pasted entries to handle the **EmailType** field.

Controller Imports

Two imports relate to state.

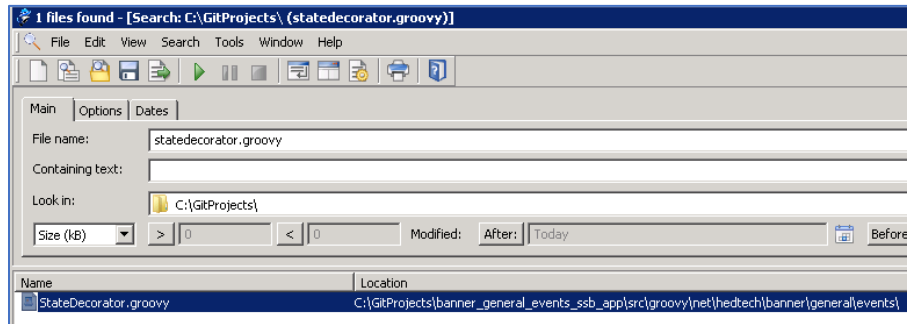
When these imports are copied and pasted and modified to refer to EmailType, EmailTypeDecorator appears as red text.

This indicates EmailTypeDecorator has a problem, where the entry for EmailTypeDecorator does not seem to exist:

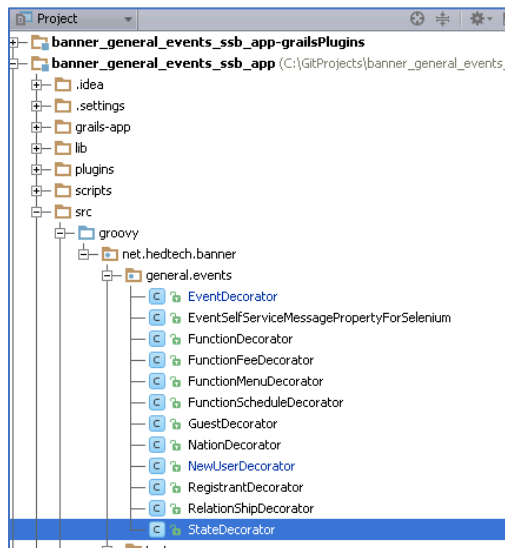
```
import grails.converters.JSON
import java.text.ParseException
import net.hedtech.banner.controllers.BaseRestfulControllerMixin
import net.hedtech.banner.exceptions.ApplicationException
import net.hedtech.banner.general.events.NationDecorator
import net.hedtech.banner.general.events.NewUserDecorator
import net.hedtech.banner.general.events.StateDecorator
import net.hedtech.banner.general.events.EmailTypeDecorator
import net.hedtech.banner.general.system.Nation
import net.hedtech.banner.general.system.State
import net.hedtech.banner.general.system.EmailType
```


EmailType Decorator

1. Locate the StateDecorator file. Use Agent Ransack, and search for the file **StateDecorator.groovy** in the project directory.

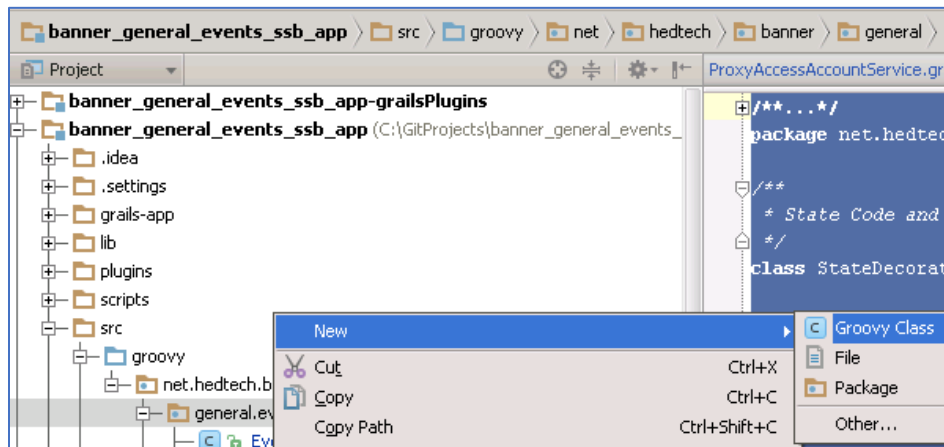


2. The resulting file is located in the project directory as shown below.



Note: The NationDecorator which provides the nation drop-down box for the form is also located here.

3. Open and examine the `StateDecorator` file.
It contains two fields, code and description.
These are standard fields that map to the Rule and Validation table's corresponding fields in a standardized way.
4. Copy and paste the contents of the **StateDecorator** body.
5. Right-click the directory that holds the `StateDecorator` file and choose **New > Groovy Class**.
6. Name the new class, **EmailTypeDecorator**. Capitalization is important.



7. Add the new file to Git.
8. Paste the contents of your clipboard into the new file.
9. Edit the entries so they match the screenshot below.

```
package net.hedtech.banner.general.events

class EmailTypeDecorator {
    /**
     * Code of the email Type object
     */
    String code

    /**
     * Description of the email Type object
     */
    String description
}
```

10. **CTRL-S** to save.

Trace through the Controller 1

In the Controller, from imports, the EmailTypeDecorator code has changed to grey indicating that the entry is valid, but not yet used in the code.

1. From imports, resume the search for “**state**”.
2. At **AccountRegistration getDefaultData** method, note the line similar to the following, which invokes a method that declares and populates a variable, called “**state**” with a list of States.
3. Copy and paste these lines, as shown below. From the Service, add a method to match this call.

```
def getDefaultData = {  
  def model = [:]  
  try {  
    def emailType = proxyAccessAccountService.fetchListOfEmailTypes() // Added  
    def state = proxyAccessAccountService.fetchListOfStates()  
    def nation = proxyAccessAccountService.fetchListOfNations()  
    def county = proxyAccessAccountService.fetchListOfCounties()  
    Map configMap = userAccountCreationConfigService.getConfigMap()  
    configMap.put("displayI18NFields", EventsUtility.isI18NFieldDisplayEnabled())  
    model.put("nation", nation)  
    model.put("state", state)  
    model.put("county", county)  
    model.put("emailType", emailType) // Added  
    model.put("configMap", configMap)  
  } catch (ApplicationException ae) {  
    model = EventsUtility.returnFailureMessage(ae)  
  }  
  render model as JSON  
}
```

Add the fetchListOfEmailTypes method to the Service

1. Open the **proxyAccessAccountService** file, located in the project directory, under `grails-app/services`.
2. Search for the method called "fetchListOfStates".
3. Copy and paste this code.
4. Modify the new method to reflect the emailType field.

```
/**
 * List of values for emailType field
 */
def fetchListOfEmailTypes() {
    def emailTypeList = EmailType.findAll([sort: "description", order: "asc"])
    def decoratorList = []
    emailTypeList.each { emailType ->
        decoratorList.add(new EmailTypeDecorator(code: emailType.code, description: emailType.description))
    }
    return decoratorList
}
```

5. **CTRL-S** to save.

Trace through the Controller 2

1. Locate the **if** block for both state and nation.

```
def accountRegistration = {
    def formData = request?.JSON?.form
    def model = [:]
    if (formData != null) {
        try {
            def newUser = new NewUserDecorator()
            if (formData.emailType) {
                EmailType submitEmailType = EmailType.findByCode(formData.emailType)
                formData.emailType = new EmailTypeDecorator(code: submitEmailType.code, description: submitEmailType.description)
            }
            if (formData.state) {
                State submitState = State.findByCode(formData.state)
                formData.state = new StateDecorator(code: submitState.code, description: submitState.description)
            }
        }
    }
}
```

2. Copy and paste the block for 'state'.
3. Modify the pasted code to reflect emailType.
4. **CTRL-S** to save.

This completes extending the Controller.

Copying Logic

1. Review the **ProxyAccessAccountService**.
2. Locate where **'state'** is used in pertinent code and copy any additional logic for the new emailType field.

```
//copy the geniden properties and persist it
EmailType emailType
if (newUser?.emailType?.code) {
    emailType = EmailType.findByCode(newUser.emailType.code)
}
State state
if (newUser?.state?.code) {
    state = State.findByCode(newUser.state.code)
}
Nation nation
if (newUser?.nation?.code) {
    nation = Nation.findByCode(newUser.nation.code)
}
```



Note: If EmailType is not recognized, verify the service has the following import

```
import net.hedtech.banner.general.system.EmailType
```

3. Resume the search for **'state'** in the Service file, and copy them as shown below:

```
private NonBannerPersonIdentificationName createGeniden(newUser, state, nation, county, emailType){
    def nonBannerPersonIdentificationName = new NonBannerPersonIdentificationName(systemCode: "EVENT_F
        firstName: newUser.firstName, streetLine1: newUser.streetAddressLine1, streetLine2: newUse
        city: newUser.city, state: state, zip: newUser.zip, emailType: emailType,
        nation: nation, phoneArea: newUser.phoneArea, phoneNumber: newUser.phoneNumber, phoneExter
        countryCodePhone: newUser.countryCodePhone, birthDate: newUser.birthDate, emailId: newUser.ema
        middleName: newUser.middleName, namePrefix: newUser.namePrefix, preferredFirstName: newUse
        streetLine3: newUser.streetAddressLine3, streetLine4: newUser.streetAddressLine4, county:
    //Need to set the mail, phone, asrc and email types for which query GIVOTYP
    def session = sessionFactory.getCurrentSession()
    def resultSet
    def sysCode = 'PROXY'

    List otypCodes = []
    otypCodes.add('LOAD_EMAIL_TYPE')
    otypCodes.add('LOAD_PHONE_TYPE')
    otypCodes.add('LOAD_ADDRESS_TYPE')
    otypCodes.add('LOAD_ADDRESS_SOURCE')

    resultSet = session.createSQLQuery("SELECT GIVOTYP_OPTION_DEFAULT, GIVOTYP_CODE FROM GIVOTYP WHERE
    if (resultSet) {
        resultSet.each {
            //if (it[1] == 'LOAD_EMAIL_TYPE')
            //    nonBannerPersonIdentificationName.emailType = EmailType.findByCode(it[0])
            if (it[1] == 'LOAD_PHONE_TYPE')
                nonBannerPersonIdentificationName.telephoneType = TelephoneType.findByCode(it[0])
            if (it[1] == 'LOAD_ADDRESS_TYPE')
                nonBannerPersonIdentificationName.addressType = AddressType.findByCode(it[0])
        }
    }
```

This the work for this service file. Any additional uses of **'state'** in the code are not necessary to complete this task.

Working with the Web Page (View)

The next step is to add the emailType field to the accountRegistration.html file from banner_general_events_ssb_app/web-app/ssb/eventsApp/accountRegistration/.

The easiest approach to extending this file is to repeat the same process completed previously:

- Locate another drop-down box in this code.
 - Copy and paste the code for that field.
 - Modify it.
1. Search for the code for the “**state**” drop-down box.
 2. Copy the “**state**” code.
 3. Paste the code into the appropriate place.
The new field is to appear on the web page after the email address text box.
Paste the new code after the email code.
 4. Modify the code to match the screenshot below. You can also find this code in the lab6 folder, or by clicking the text file after this screenshot.

```
<div class="form-group" show-errors xe-field="emailType">
  <label for="emailType" class="control-label">
    {{ 'events.registration.profile.emailType' | i18n }}<span class="asterisk-input"
  </label>
  <select id="emailType" name="emailType" data-ng-model="accountRegistration.newUse
    ng-options="emailType.code as emailType.description for emailType in ::ac
    placeholder="{{ 'events.registration.profile.emailType' | i18n }}" ng-requi
    aria-required="{{ accountRegistration.configMap.emailType }}"
    aria-describedby="emailTypeRequiredMessage" aria-label="emailTypeRequired
    <option value="" label="{{ 'events.registration.profile.emailType.select' | i18
  </select>
  <span id="emailTypeRequiredMessage" class="help-block" ng-if="accountRegistration.
    {{ 'events.message.emailType.required' | i18n }}</span>
</div>
```



emailTypeDiv.txt

5. Add the declared labels to messages.properties. Any red text related to this action will change to grey.


6. Open the `i18n/messages.properties` file.
Add the entry as shown below:

```
events.registration.profile.subtitle=to continue you w  
events.registration.login.title=Username and Password  
events.registration.pickOne=Please Select...  
events.registration.dateOfBirth.text.format=MM/dd/yyyy  
events.registration.profile.comment=Comment  
events.registration.profile.emailType=Email Type|
```

It's important to add it along with the other entries for `events.registration.profile`.

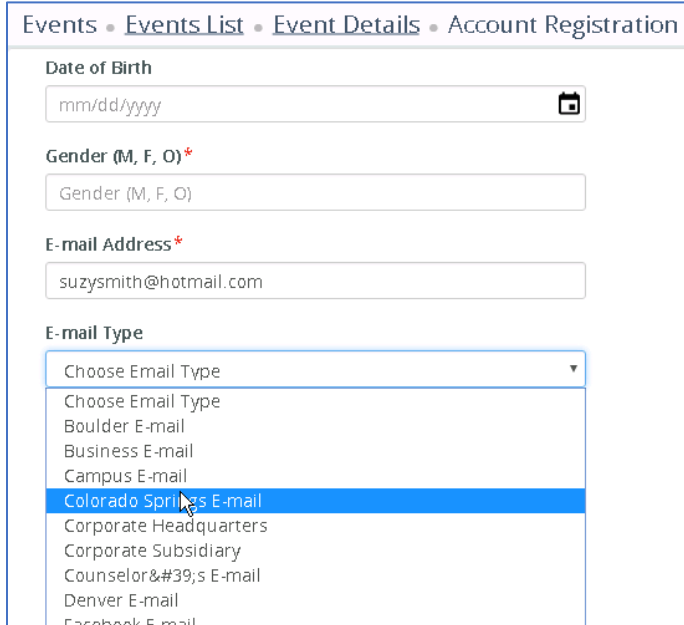
7. **CTRL-S** to save.
8. Add the line for the `emailTypeList` to modify the web-app/ssb/eventsApp/accountRegistration/accountregistration-controller.js file.

```
function AccountRegistrationController($scope, $state, $window,  
    var vm = this;  
    vm.eventId = $state.params.eventId;  
    vm.functionId = $state.params.functionId;  
    vm.newUser = {};  
  
    vm.configMap = accountCreationResolve['configMap'];  
    vm.nationList = accountCreationResolve['nation'];  
    vm.stateList = accountCreationResolve['state'];  
    vm.countyList = accountCreationResolve['county'];  
    vm.emailTypeList = accountCreationResolve['emailType'];  
  
    vm.createAccount = accountCreation;  
    vm.cancel = cancelAccountCreation;  
    vm.preventValidationBlurEvent = preventValidationBlurEvent;
```



Test the Extension

The AccountRegistration page, will look similar to the following:



The screenshot shows a web form titled "Account Registration" with a breadcrumb trail: "Events • Events List • Event Details • Account Registration". The form contains the following fields:

- Date of Birth**: A text input field with a calendar icon, containing the placeholder "mm/dd/yyyy".
- Gender (M, F, O)***: A text input field containing the placeholder "Gender (M, F, O)".
- E-mail Address***: A text input field containing the placeholder "suzysmith@hotmail.com".
- E-mail Type**: A dropdown menu with the following options: "Choose Email Type", "Boulder E-mail", "Business E-mail", "Campus E-mail", "Colorado Springs E-mail" (highlighted), "Corporate Headquarters", "Corporate Subsidiary", "Counselor's E-mail", "Denver E-mail", and "Facebook E-mail".

1. Add a record and submit it.
2. Verify that there are no error messages.
3. Notice the html encoding in the description, use `htmlDecode` to handle any special characters. Find the `ng-options` in the html file, change `state.description` to `(state.description | htmlDecode)` to use the AngularJS `htmlDecode` filter.
4. From the SQL Developer tool, query the records to confirm your entry with a query like this:

```
Select geniden_email_address, geniden_email_code from geniden where  
geniden_last_name = 'Rashad'
```



Note: THE SQL Developer tool is accessed from the icon in the virtual environment' The instructor will provide you the password.

Summary and Notes

- In this lab you added a select box (dropdown) and a validated email field to our form.
- You located and modified the appropriate controller on the server-side, by using a Decorator return a list of email types our user will select from.
- You updated the appropriate Groovy service to provide the list of email types from the database.
- You then updated the client-side code to accommodate displaying a list of options.

This is the end of part two of the agenda.

Part 2 Server-Side Work

6. Debugging
Tips

7. Object Relational
Mapping

8. The Domain
Model

Extend Apps with New Data – The Event App

Lab 4 –
Add a Comment
Field to the Account
Registration

Lab 5 –
Add the Gender
Field with Validation
to the Account
Registration Page

Lab 6 –
Add an Email Type
field to the Account
Registration Page

Take a few moments to go over your notes. What questions do you have?



Lab 7 - Add a Guest Policy Page

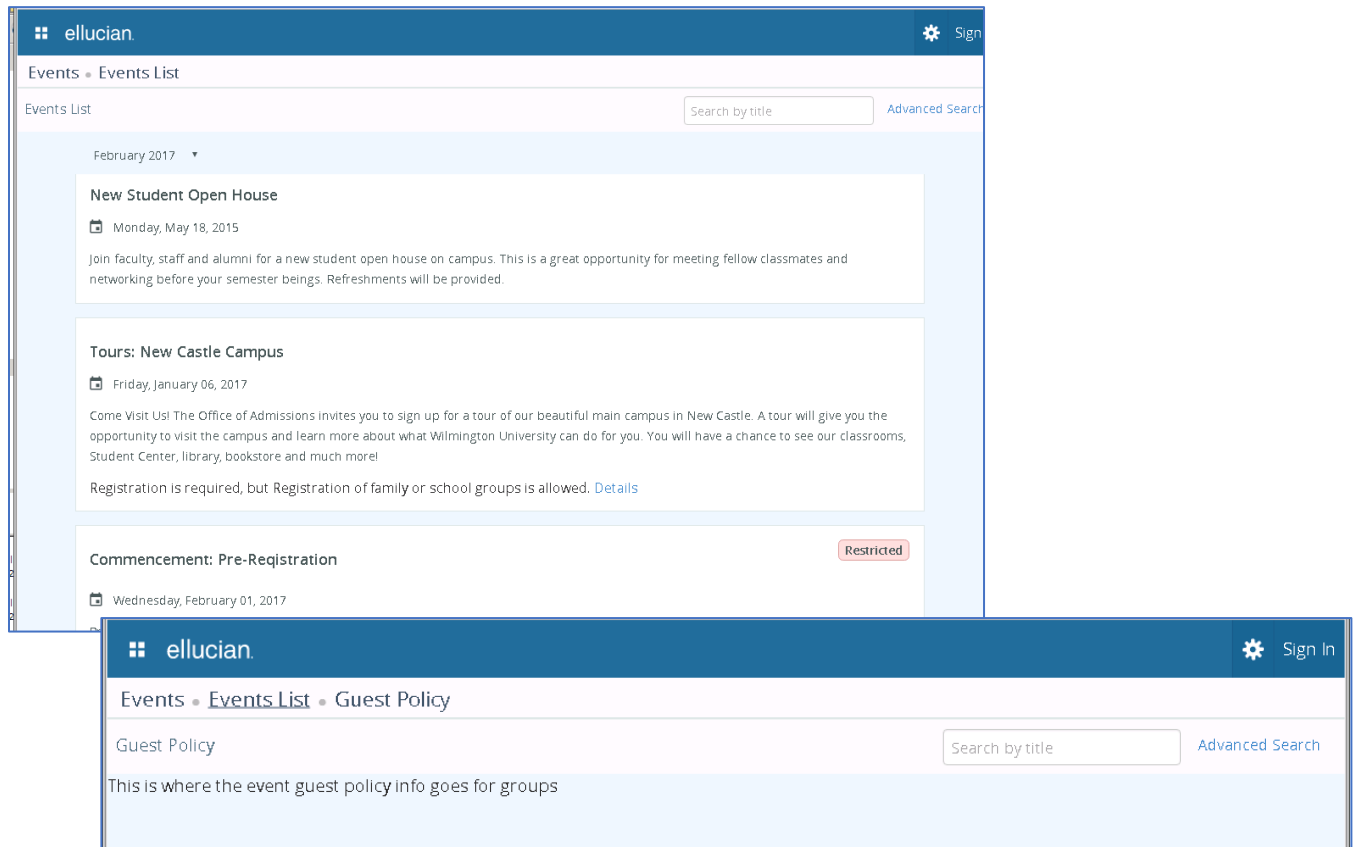
Scenario



You want to add a note message and a clickable link in the Events Management App to display a popup message if guests of registered attendees are welcome. This is determined by checking against data in an Oracle database.

Conditional Rendering of Page Content

This scenario provides you practice with the conditional rendering of page content. Showing an indicator for group registration is allowed and adding a clickable link for more information about the guest policy.



Files Modified for Condition Rendering of Page Content

Files worked with in previous exercises include:

- Domain Model
- View Files
- HTML files (AngularJS template)
- JavaScript files (AngularJS code)
- Controller
- Services
- Domain
- Messages.properties
- CSS (Cascading Style Sheets)

Two additional files we are introducing are the **service** file and the **Decorator** file.

For this exercise, there will be no changes to the fonts, images, etc. and so the CSS files will not be modified.

Managing Source Code



- Before starting any changes to source code, understand how your organization is managing changes within Git.
- Create an appropriate branch name or checking the correct existing branch.

For the following labs, create a branch for each lab.

For this lab, create a branch with Git Bash or TortoiseGit, a lab called **lab 7** in the general events repository.

```
git checkout -b lab_new_page
```

```
1. blee@USD1400400 /D/git_local/banner_general_events_ssb_app (master)
2. $ git checkout -b lab_new_page
3. Switched to a new branch 'lab_new_page'
4. blee@USD1400400 /D/git_local/banner_general_events_ssb_app (lab_new_page)
```

The -b option in the checkout creates the branch at the current commit point in the current branch. Note, that the branch name changed from (master) to (lab_new_page) in the Git Bash window.

Gather information

Determine which domain model presents this data and how it is coming into the App, so you can use it.

- This information is in the Events Management App
- The data to evaluate it is in the Oracle Database field SLBEVNT_LIMIT_ONE

This value is either **Y** or **N**. Verify that this data is coming in and verify which domain is containing it.

Examine the Flow – How the Page Pieces are Displayed

Examine the process by which the web page is displayed by the actions of the controller.

- The process starts with a URL request.
- The app user clicks a link or a button, etc.
- The events page is requested in the form of a URL in the code similar to:
`http://localhost:8080/SelfServiceBannerGeneralEventManagement/ssb/events`

The URL ends in **events**. The control goes to the **EventsController** which executes its action method called **events**.

1. Open the EventsController, **grails-app > Controllers**, locate the events action method. At the end of this method, the events gsp page is rendered.

```
def events = {  
    def model = [:]  
    if(multiEntityProcessingService.isMEP()) {  
        if(params.mepCode || session?.getAttribute("mep")) {  
            setMepCode();  
        } else {  
            throw new MepCodeNotFoundException(mepCode: "NO_MEP_CODE_PROVIDED")  
        }  
    }  
  
    render model: model, view: "events"  
}
```

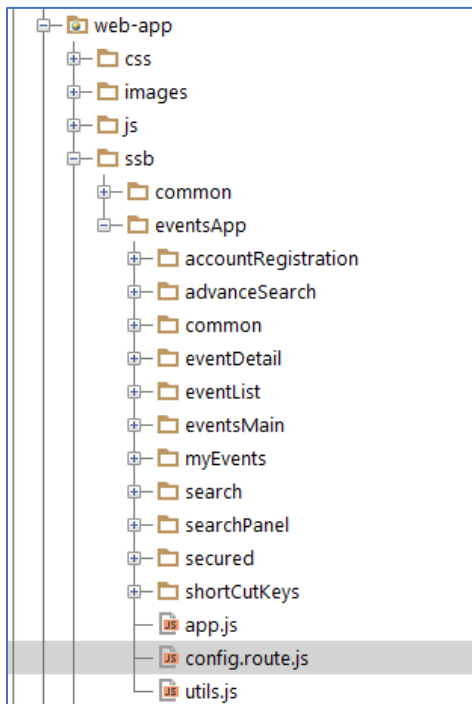
2. From the rails-app, go to **views > events**, locate the **events**.
3. Open the events.gsp file.

```
<body>
<div id="fb-root"></div>
<div id="content" class="container-fluid" role="main">
  <!--Place holder for title as mentioned in 9.14 .1 platform release -->
  <div class="">
    <div id="title-panel" class="aurora-theme"></div>
  </div>

  <span id="screenReaderMessageSupport" class="sr-only"></span>
  <div ui-view class="ui-view-height"></div>
</div>
</body>
```

Note, the <body> tag has little content. The <div> with the ui-view attribute is where the content of the page is displayed. This is part of the ui-router plugin for AngularJS.

- Review the AngularJS files in the web-app folder to see how the data is displayed. From the **web-app/ssb/eventsApp** directory find the AngularJS \$stateProvider and \$urlRouterProvider function. It is defined in the config.route.js file.



```
/*
Copyright 2016 Ellucian Company L.P. and its affiliates.
*/
(function() {
  'use strict';

  angular.module('eventsApp')
    .provider('routeConfig', eventsRouteConfigProvider)
    .config(['$stateProvider', '$urlRouterProvider', 'routeConfigProvider', routeAppConfig]);

  function eventsRouteConfigProvider () {
    var sidePanelRouteViews = {};

    return {
      getSidePanelRouteViews: function() {
        return sidePanelRouteViews;
      },
      $get: function () {
        return {
          sidePanelRouteViews: sidePanelRouteViews
        };
      }
    };
  };
})();
```


The function that handles routing all the browser-side traffic is the **routeAppConfig** function. Review the **routeAppConfig** function for its default state and the definition for that default state.

The **otherwise** statement at the beginning of the function has the eventList as the **default** state.

Reviewing the eventList state:

- The template html file displays the data.
- The AngularJS controller (as opposed to a Grails controller) handles any processing on the page.
- The AngularJS service (as opposed to a Grails service) interacts with the server to retrieve or update data.
- The resolve line provides the controller with content or data custom to the state.
- Resolve is an optional map of dependencies which is injected into the controller.
- If any of these dependencies are promises, they are resolved and converted to a value before the controller is instantiated and the \$stateChangeSuccess event is fired.

The data for the Event List page then is populated from the `getEventsList` function in the `EventListService`.



The screenshot shows the `routeAppConfig` function with several lines highlighted in yellow. Red arrows point from a text box on the right to these highlighted lines:

- Arrow 1 points to `templateUrl: '../ssb/eventsApp/eventList/eventList.html'` in the `content@eventsMain` state.
- Arrow 2 points to `controller: 'EventListController as eventList'` in the `content@eventsMain` state.
- Arrow 3 points to `return EventListService.getEventsList();` in the `eventListResolve` function.

The text box on the right contains the text: "AngularJS template, controller and service for the Event List page".

```
function routeAppConfig($stateProvider, $urlRouterProvider, routeConfigProvider) {
  $urlRouterProvider.otherwise('/eventList');
  $stateProvider
    .state('eventsMain', {
      templateUrl: '../ssb/eventsApp/eventsMain/eventsMain.html',
      controller: 'EventsMainPageController as mainPage'
    })
    .state('side-panel', {
      parent: 'eventsMain',
      views: routeConfigProvider.getSidePanelRouteViews()
    })
    .state('eventList', {
      url: '/eventList',
      parent: 'side-panel',
      views: {
        'content@eventsMain': {
          templateUrl: '../ssb/eventsApp/eventList/eventList.html',
          controller: 'EventListController as eventList',
          resolve: {
            eventListResolve: function (EventListService) {
              return EventListService.getEventsList();
            },
            dateDropDownDataResolve: function (EventListService) {
              return EventListService.getDateDropDownData();
            }
          }
        }
      }
    })
  },
  data: {
    'pageTitle': 'events.breadcrumb.eventList',
  }
}
```

5. **Ctrl + Click** the **EventListService** function to access its definition.

```
function getEventsList() {  
  var eventListModel = $resource('../ssb/:controller/:action',  
    {controller: 'events', action: 'eventsList'},  
    {query: {method: 'GET', isArray: true}}  
  );  
  
  return eventListModel.query().$promise;  
};
```

`$resource` is an AngularJS factory which helps interaction with RESTful server-side data sources. The generic format of `$resource` is:

```
$resource(url, [paramDefaults], [actions], options);
```

In the `getEventsList` function, a colon (:) character proceeding a word indicated a variable populated by the values in the `paramDefaults` section of the `$resource` call.

In generic AngularJS terminology, the resource plugin is going to access the URL:

`../ssb/events/eventList`.

It is a RESTful GET action and returns an array of data.

From a Grails perspective, `events` is a Grails controller (as opposed to an AngularJS controller) and `eventList` is a method in that Grails controller.

6. Open the controller and locate the eventsList action method. Review the code at the end and verify it causes the eventsList to be rendered.

When using AngularJS, grails becomes a provider of data. The eventList returns as JSON. No view is referenced.

The eventsList is a RESTful service for the Events List page.

```
def eventsList = {  
    def selectedDate = (params.selectedDate) ? params.selectedDate : new Date()  
    Integer monthSelected = (params.selectedMonth) ? params.selectedMonth as Integer  
    Integer yearSelected = (params.selectedYear) ? params.selectedYear as Integer : r  
    ...  
    def registrantID = getRegistrantId()  
    def registrantPidm = session?.getAttribute(registrantPIDMKey)  
    def registrantUserType = session?.getAttribute(registrantUserTypeKey)  
  
    def eventList = eventSSBService.fetchEventListForDefaultCalendar(registrantID, re  
    eventList?.each {EventDecorator event ->  
        event.description = markdownToHtml(atomMarkdownContent(event))}  
  
    render eventList as JSON  
}
```

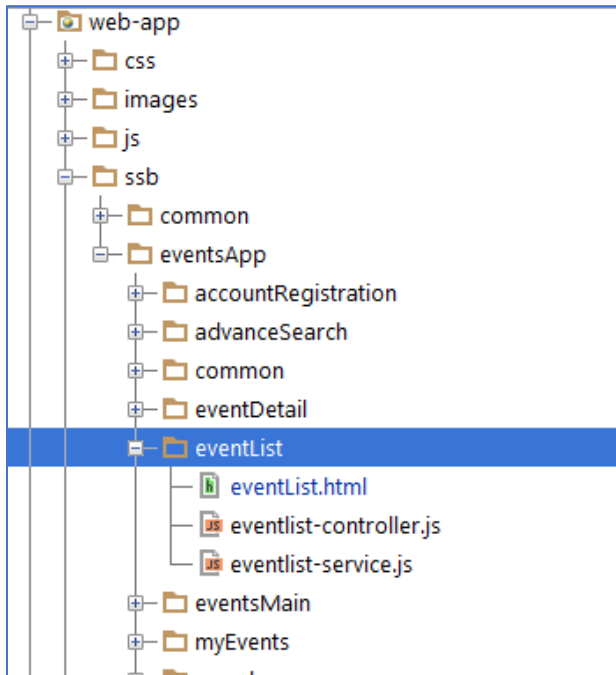
The config.route.js, the eventsList state has the following

```
function routeAppConfig($stateProvider, $urlRouterProvider, routeConfigProvider) {
  $urlRouterProvider.otherwise('/eventList');
  $stateProvider
    .state('eventsMain', {
      templateUrl: '../ssb/eventsApp/eventsMain/eventsMain.html',
      controller: 'EventsMainPageController as mainPage'
    })
    .state('side-panel', {
      parent: 'eventsMain',
      views: routeConfigProvider.getSidePanelRouteViews()
    })
    .state('eventList', {
      url: '/eventList',
      parent: 'side-panel',
      views: {
        'content@eventsMain': {
          templateUrl: '../ssb/eventsApp/eventList/eventList.html',
          controller: 'EventListController as eventList',
          resolve: {
            eventListResolve: function (EventListService) {
              return EventListService.getEventsList();
            },
            dateDropDownDataResolve: function (EventListService) {
              return EventListService.getDateDropDownData();
            }
          }
        }
      }
    })
  },
  data: {
    'pageTitle': 'events.breadcrumb.eventList',
  }
}
```

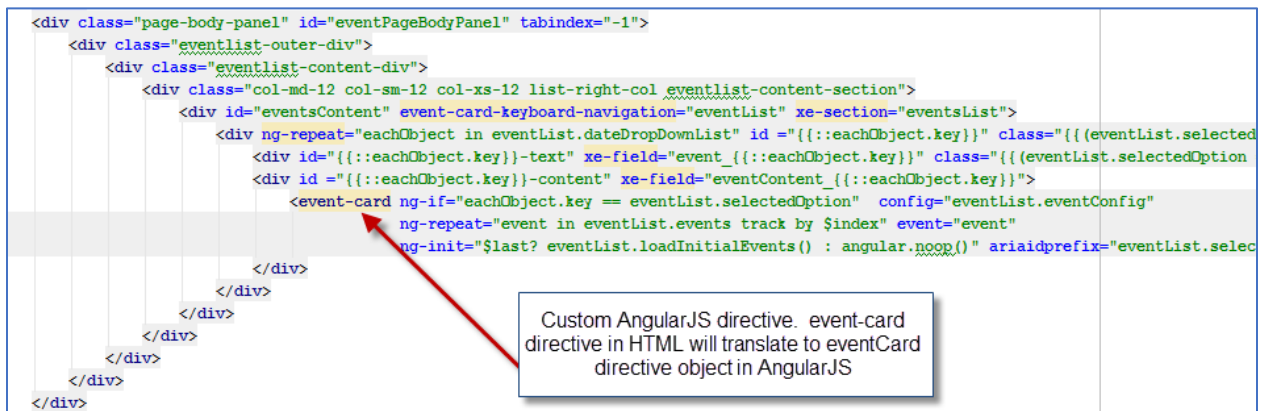


AngularJS template, controller and service for the Event List page

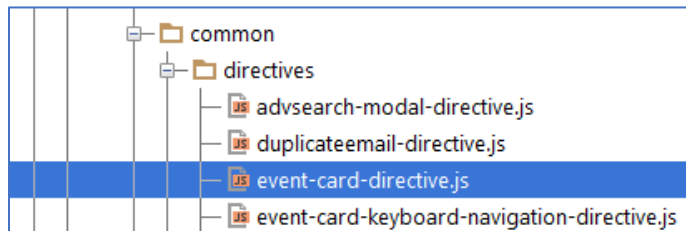
The template html, **eventList.html**, associated with the ui state determines how the data displays. Typically, each state is in a separate directory under the main application directory.



7. Locate and open the **eventList.html** template file. Note the HTML tag, **event-card**. This is a custom AngularJS directive created by Ellucian for this app.



8. Open **web-app > ssb > eventsApp > common > directives > event-card-directive.js**.



```

/*****
Copyright 2016 Ellucian Company L.P. and its affiliates.
*****/
(function () {
    'use strict';


    angular.module('eventsApp').
        directive('eventCard', [eventCard]);

    function eventCard() {
        return {
            restrict: 'E',
            replace: true,
            scope: {
                event: '=',
                config: '=',
                user: '=',
                ariaidprefix: '='
            },
            templateUrl: '../ssb/eventsApp/common/partials/eventCardTemplate.html',
            controller: function($scope, $timeout){
                $timeout(function(){
                    $scope.fbUrl = window.location.href;
                }, 0);
            },
            link: function($scope, $elem){
                $elem.bind('keydown', function (event) {
                    if (event.keyCode == 13) {
                        $elem.find('.cursor-icon').click();
                    }
                });
            }
        };
    }
})();
```

html file that displays information for each event "card"

From below, note the following:

- Within the HTML, the known AngularJS directives, **ng-if**.
- The information for each event displayed on the event list is in the **eventCardTemplate.html**.
- Another custom directive, **function-card**.



```

<div class="event-card-div" xe-dynamic>
  <div class="event-card-title-outer" xe-section="eventTitleSection">
    <h4 class="event-card-title pull-left text-semi-bold-medium" xe-field=
      <span ng-if="config.eventTitleLinkEnabled" class="event-title cu
        <label class="sr-only">{{'default.srOnly.event.title.link'
          {{::event.title | htmlDecode}}
        </span>
        <span ng-if="!config.eventTitleLinkEnabled" class="event-title"
      </h4>
      <div ng-if="event.registered && config.eventRegistrationStatus" xe-f
      <div ng-if="!event.registered && event.restrictedEvent && config.eve
      <div ng-if="event.facebookEnabled" xe-fb-like class="event-title-fac
    </div>
    <div id="{{ariaidprefix}}{{event.eventId}}">
      <label class="sr-only">
        <div ng-if="event.registered && config.eventRegistrationStatus"
        <div ng-if="!event.registered && event.restrictedEvent && config
      </label>
      <div class="event-card-heading" xe-section="eventDateSection" ng-if=
        <div class="event-card-calendar-icon" xe-field="eventCardCalenda
        <label class="sr-only">{{'default.srOnly.event.date' | i18n}}</l
        <div class="event-card-icon-padding text-regular-small" xe-field
      </div>
      <div class="event-card-heading" ng-if="config.restrictToSingleFuncti
        <span class="event-restrict-single-function-icon" xe-field="rest
        <div class="event-restrict-single-function-text-align" xe-field=
      </div>
      <label ng-if="config.description" class="sr-only" >{{'events.by.desc
      <div ng-if="config.description" class="event-card-desc text-regular-
      <function-card ng-if="config.function" ng-repeat="function in event
    </div>
  </div>
</div>

```

Another custom directive.

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Examine the Flow – How Database Data is Retrieved

1. Return to the **EventsController** file under `grails-app/controllers`.
2. Locate the **eventsList** closure method.
3. Within the `eventsList` method, search for a method starting with **fetch**. This is where the data is obtained. This usually involves the use of a Service:

```
def eventList =
eventSSBService.fetchEventListForDefaultCalendar(registrantID, registrantPidm
as String, registrantUserType, yearSelected, monthSelected)
```

4. In IntelliJ, Ctrl-Click, **fetchEventListForDefaultCalendar**. IntelliJ opens the **EventsSSBService.groovy** file to the **fetchEventListForDefaultCalendar** method.

In the **fetchEventListForDefaultCalendar** method, note it uses another fetch method, which belongs to the **EventAndFunctionView** class.

The domain is **EventAndFunctionView** and invokes a **fetch** method. Locate the **EventAndFunctionView** file and locate the fetch method.

```
/**
 * fetchEventList returns the list of events published on web for a given month and year depending on
 */
def fetchEventListForDefaultCalendar(String registrantID, String registrantPidm = null, String regist
Date firstDayOfMonthInGregorian = dateConverterService.getGregorianFromDefaultCalendar(yearSelect
Date lastDayOfMonthInGregorian = dateConverterService.getGregorianFromDefaultCalendar(yearSelecte
List events = EventAndFunctionView.fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents(la
return createEventDecoratorObjects(events, registrantID, registrantPidm, registrantUserType)
```

5. Ctrl + click the **fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents** method.
6. The **EventAnd FunctionView** opens **fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents**.

The method in the **EventAndFunctionView** domain uses a **NamedQuery**. The named query is located at the top of the domain file.

7. Locate the **@Table** Hibernate annotation near the top of the groovy file. Opening this domain class, verify it is mapped to the view **GVQ_GEEVNT**. This is the Oracle data source, which is also an Oracle View.

```
@Entity
@Table (name = "GVQ_GEEVNT")
```

Perform Modifications

1. Verify the **EventAndFunctionView** file is open and at the correct NamedQuery, **fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents**.
2. Locate the column mapping entry for the database field **SLBEVNT_LIMIT_ONE** in the domain file. Note the variable name **limitOne** used.

```
@Type(type = "yes_no")
@Column(name = "SLBEVNT_LIMIT_ONE")
Boolean limitOne
```

3. Return to **fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents**.
4. Add entries for **limitOne** to this query. Note that this NamedQuery selects each required field.



Add both the **limitOne** field to the **NamedQuery** and add it to a **Group By** clause.

5. From **fetchAllCurrentMonthlyPublishedEventsForDefaultCalendarEvents**, extend the fetch code by locating the code in the domain and modifying it as shown below.

```
@NamedQuery(name = "EventAndFunctionView.fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents",
    query = "" select efv.eventCourseReferenceNumber,efv.eventDescription,efv.eventLongDescription, efv.restricted, min(efv.startDate)
    , efv.limitOne
    from EventAndFunctionView efv where trunc(sysdate) between efv.publishFrom and efv.publishTo and efv.publish = true
    and ( efv.startDate <= :lastDayOfMonth and efv.endDate >= :firstDayOfMonth)
    group by efv.eventCourseReferenceNumber,efv.eventDescription,efv.eventLongDescription, efv.restricted
    , efv.limitOne
    order by min(efv.startDate), efv.restricted desc
    """),
```

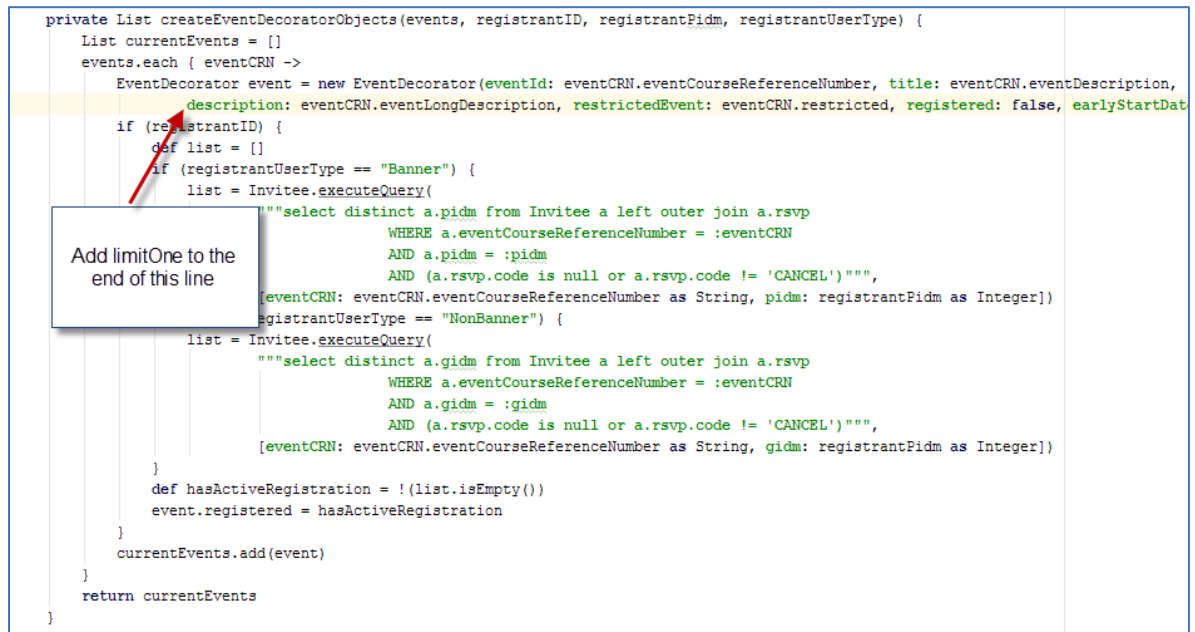
```
public static List <fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents>( Date lastDayOfMonth, Date firstDayOfMonth )
{
    def eventAndFunctionList = []
    def eventAndFunctionFetchList = []
    lastDayOfMonth.clearTime()
    firstDayOfMonth.clearTime()
    EventAndFunctionView.withSession {session ->
        eventAndFunctionFetchList = session.getNamedQuery( 'EventAndFunctionView.fetchAllCurrentMonthPublishedEventsForDefaultCalendarEvents',
            [lastDayOfMonth, firstDayOfMonth] )
    }

    eventAndFunctionFetchList?.each {meetingTime ->
        eventAndFunctionList.add( new EventAndFunctionView( eventCourseReferenceNumber: meetingTime[0],
            eventDescription: meetingTime[1],
            eventLongDescription: meetingTime[2],
            restricted: meetingTime[3],
            startDate: meetingTime[4],
            limitOne: meetingTime[5])
        )
    }
    return eventAndFunctionList
}
```


Extend the Events List Page

Extend the Decorator file and the code within Domain and Service code that uses it. Decorators are Sitemesh files.

1. Examine the **grails-app > services > net.hedtech.banner.general.events > EventSSBService** code.
Note that the EventDecorator Decorator file is instantiated and used.
2. Ctrl-click where the EventDecorator class is used in the EventSSBService class and extend it to include the new field:

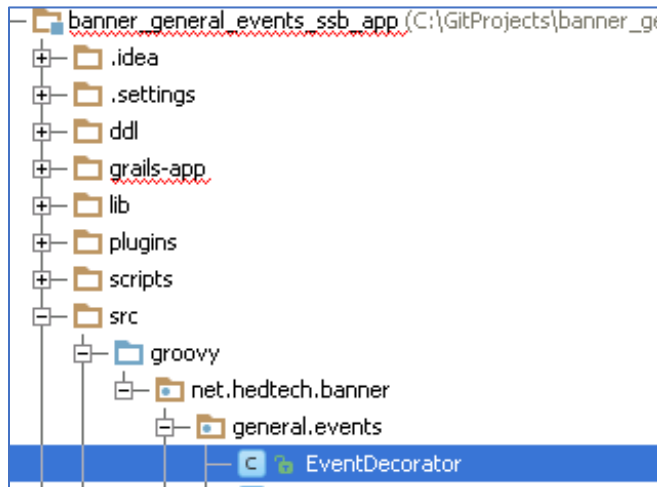


```
private List createEventDecoratorObjects(events, registrantID, registrantPidm, registrantUserType) {
    List currentEvents = []
    events.each { eventCRN ->
        EventDecorator event = new EventDecorator(eventId: eventCRN.eventCourseReferenceNumber, title: eventCRN.eventDescription,
            description: eventCRN.eventLongDescription, restrictedEvent: eventCRN.restricted, registered: false, earlyStartDate:
            if (registrantID) {
                def list = []
                if (registrantUserType == "Banner") {
                    list = Invitee.executeQuery(
                        """select distinct a.pidm from Invitee a left outer join a.rsvp
                            WHERE a.eventCourseReferenceNumber = :eventCRN
                            AND a.pidm = :pidm
                            AND (a.rsvp.code is null or a.rsvp.code != 'CANCEL')""",
                        [eventCRN: eventCRN.eventCourseReferenceNumber as String, pidm: registrantPidm as Integer])
                    registrantUserType == "NonBanner") {
                        list = Invitee.executeQuery(
                            """select distinct a.gidm from Invitee a left outer join a.rsvp
                                WHERE a.eventCourseReferenceNumber = :eventCRN
                                AND a.gidm = :gidm
                                AND (a.rsvp.code is null or a.rsvp.code != 'CANCEL')""",
                                [eventCRN: eventCRN.eventCourseReferenceNumber as String, gidm: registrantPidm as Integer])
                    }
                    def hasActiveRegistration = !(list.isEmpty())
                    event.registered = hasActiveRegistration
                }
                currentEvents.add(event)
            }
        }
    }
    return currentEvents
}
```

The EventDecorator line is too long in the source code, so the content of that line is shown here with line breaks to make it easier to see the change.

```
EventDecorator event = new EventDecorator(
    eventId: eventCRN.eventCourseReferenceNumber,
    title: eventCRN.eventDescription,
    description: eventCRN.eventLongDescription,
    restrictedEvent: eventCRN.restricted,
    registered: false,
    earlyStartDate: dateConverterService.convertGregorianToDefaultCalendar(
        eventCRN.startDate,
        MessageUtility.message("events.date.format.long")
    ),
    limitOne: eventCRN.limitOne)
```

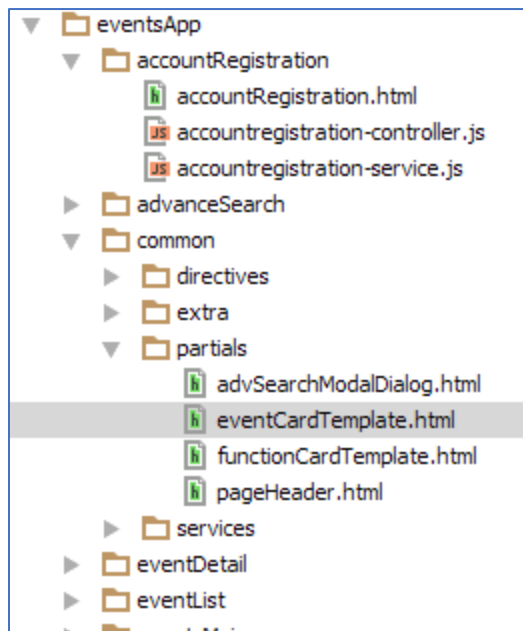
3. Modify the Decorator class as shown below.



4. Add the field to the EventsDecorator after the entry for earlyStartDate.

```
boolean limitOne
```

5. Add an entry in the toString function to the EventDecorator class.
6. We will be adding the field entry to the event on the event list page. Find where the title, description, etc. fields are displayed so we know where to place our new code.
7. The code will be entered after the event description. Individual events on the Event List page are located at **eventCardTemplate.html**.



8. We want the following logic:
If **limitOne** is false, it displays the message, **on-site registration of guests is required.**

```

<div id="{{ariaidprefix}}{{event.eventId}}">
  <label class="sr-only">
    <div ng-if="event.registered && config.eventRegistrationStatus" class="card-regis
    <div ng-if="!event.registered && event.restrictedEvent && config.eventRegistratio
  </label>
  <div class="event-card-heading" xe-section="eventDateSection" ng-if="config.eventDate
  <div class="event-card-calendar-icon" xe-field="eventCardCalendarIcon"></div>
  <label class="sr-only">{{'default.srOnly.event.date' | i18n}}</label>
  <div class="event-card-icon-padding text-regular-small" xe-field="eventEarlyStart
  </div>
  <div class="event-card-heading" ng-if="config.restrictToSingleFunction && !event.canR
  <span class="event-restrict-single-function-icon" xe-field="restrictRegistrationI
  <div class="event-restrict-single-function-text-align" xe-field="restrictRegistra
  </div>
  <label ng-if="config.description" class="sr-only" >{{'events.by.description' | i18n}}
  <div ng-if="config.description" class="event-card-desc text-regular-small" xe-section
  <function-card ng-if="config.function" ng-repeat="function in event.functions" displ
  </div>
</div>
div>

```

Add new element here

9. To do this, Add the following **ng-if** directive to the text.

```
<div ng-if="event.limitOne">{{'events.registrationRequired' | i18n}}</div>
```

The result displays as shown below.

```

<div id="{{ariaidprefix}}{{event.eventId}}">
  <label class="sr-only">
    <div ng-if="event.registered && config.eventRegistrationStatus" class="card-regis
    <div ng-if="!event.registered && event.restrictedEvent && config.eventRegistratio
  </label>
  <div class="event-card-heading" xe-section="eventDateSection" ng-if="config.eventDate
  <div class="event-card-calendar-icon" xe-field="eventCardCalendarIcon"></div>
  <label class="sr-only">{{'default.srOnly.event.date' | i18n}}</label>
  <div class="event-card-icon-padding text-regular-small" xe-field="eventEarlyStart
  </div>
  <div class="event-card-heading" ng-if="config.restrictToSingleFunction && !event.canR
  <span class="event-restrict-single-function-icon" xe-field="restrictRegistrationI
  <div class="event-restrict-single-function-text-align" xe-field="restrictRegistra
  </div>
  <label ng-if="config.description" class="sr-only" >{{'events.by.description' | i18n}}
  <div ng-if="config.description" class="event-card-desc text-regular-small" xe-section
  <div ng-if="event.limitOne">{{'events.registrationRequired' | i18n}}</div>
  <function-card ng-if="config.function" ng-repeat="function in event.functions" displ
  </div>
</div>

```

10. This code uses the label **events.registrationRequired** and in AngularJS it is passed to the **i18n** directive. Add this to the **i18n/messages.properties** file.
11. Open the file and locate other labels starting with **event** or **events**. Check under the comment header **SSB Events**.
12. Add the code as shown below:

```
events.registrationRequired=Registration is required, but  
Registration of family or school groups is allowed.
```

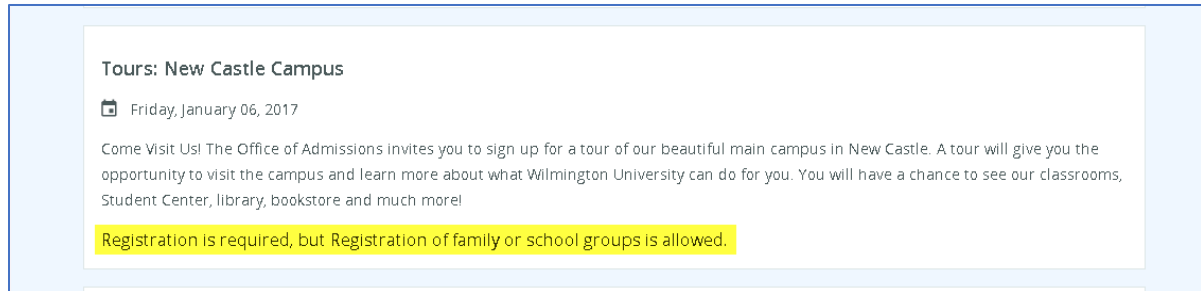
NOTE:

This code must be on one line.

13. **CTRL- S** to save all.
14. Refresh the page to test the following:
 - a. That the label appears.
 - b. No errors appear.
15. At this time, the page is unlinked. Close and stop the App when testing is complete.

Discussion

What has been your greatest challenge in this lab so far?



16. If the message on events successfully displays, where `SLBEVNT_LIMIT_ONE = 'Y'`, then open a Git Bash window on the general events ssb app git repository to commit the changes to Git.
17. Perform a git status to produce the following:

```
$ git status
On branch lab_new_page
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  (commit or discard the untracked or modified content in submodules)

        modified:   grails-app/i18n/messages.properties
        modified:   grails-
app/services/net/hedtech/banner/general/events/EventSSBService.groovy
        modified:   plugins/banner_general_events.git (new commits, modified
content)
        modified:
src/groovy/net/hedtech/banner/general/events/EventDecorator.groovy
        modified:   web-
app/ssb/eventsApp/common/partials/eventCardTemplate.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .slcache/

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app (lab_new_page)
$
```

The files were modified but the plugins are their own repositories. Access the banner_general_events plugin to commit those changes.

18. Open a second Git Bash window on the **plugins > general_events.git** directory and commit those changes. Create a branch here also. Steps in the plugin directory are as follows:
 - a. Create a new branch, **git checkout -b lab_new_page**.
 - b. Add the changes to the current commit, **git add -u**.
 - c. Commit the changes, **git commit -m "Added message when event is not limited to one person"**.

```
Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:
src/groovy/net/hedtech/banner/general/events/EventAndFunctionView.groovy

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git (master)
$ git checkout -b lab_new_page
M       src/groovy/net/hedtech/banner/general/events/EventAndFunctionView.groovy
Switched to a new branch 'lab_new_page'

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git
(lab_new_page)
$ git status
On branch lab_new_page
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:
src/groovy/net/hedtech/banner/general/events/EventAndFunctionView.groovy

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git
(lab_new_page)
$ git add -u

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git
(lab_new_page)
$ git status
On branch lab_new_page
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:
src/groovy/net/hedtech/banner/general/events/EventAndFunctionView.groovy

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git
(lab_new_page)
$ git commit -m "add limitOne to named query for events list"
[lab_new_page a4ee81d] add limitOne to named query for events list
1 file changed, 4 insertions(+), 3 deletions(-)

Administrator@WIN-VKP26P9V4DT MINGW64
/d/git_local/banner_general_events_ssb_app/plugins/banner_general_events.git
(lab_new_page)
```

19. Return to the main app Git repository and commit the changes.

```
$ git status
On branch lab_new_page
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   rails-app/i18n/messages.properties
        modified:   rails-
app/services/net/hedtech/banner/general/events/EventSSBService.groovy
        modified:   plugins/banner_general_events.git (new commits)
        modified:   src/groovy/net/hedtech/banner/general/events/EventDecorator.groovy
        modified:   web-app/ssb/eventsApp/common/partials/eventCardTemplate.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .slcache/

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@WIN-VKP26P9V4DT MINGW64 /d/git_local/banner_general_events_ssb_app
(lab_new_page)
$ git add -u

Administrator@WIN-VKP26P9V4DT MINGW64 /d/git_local/banner_general_events_ssb_app
(lab_new_page)
$ git commit -m "add message when event is not limited to one person"
[lab_new_page f4d77ea8] add message when event is not limited to one person
 5 files changed, 8 insertions(+), 2 deletions(-)

Administrator@WIN-VKP26P9V4DT MINGW64 /d/git_local/banner_general_events_ssb_app
(lab_new_page)
$ git status
On branch lab_new_page
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .slcache/

nothing added to commit but untracked files present (use "git add" to track)

Administrator@WIN-VKP26P9V4DT MINGW64 /d/git_local/banner_general_events_ssb_app
(lab_new_page)
$
```


Add a Link and Stub Page

1. Add a link to a guestPolicy page to the eventCardTemplate.html to display after the recently created new message. Use the ui-sref directive from the AngularJS ui-router plug to go to a new page. The generic format is shown below.

Usage:

ui-sref='stateName' - Navigate to state, no params. 'stateName' can be any valid absolute or relative state, following the same syntax rules as `$state.go()`

ui-sref='stateName({param: value, param: value})' - Navigate to state, with params.

2. Create a new state **eventGuestPolicy** using the eventDetail state as a model. The code in the eventCardTemplate.html for the link to the eventDetail state is as follows:

```
<h4 class="event-card-title pull-left text-semi-bold-medium" xe-field="eventTitle" ng-
class="{ 'event-title-facebook-padding': event.facebookEnabled}">
  <span ng-if="config.eventTitleLinkEnabled" class="event-title cursor-icon"
  ui-sref="eventDetail({eventId:event.eventId})" aria-
  describedby="{ {ariaidprefix} {event.eventId} }">
    <label class="sr-only">{{ 'default.srOnly.event.title.link' | i18n }}</label>
    {{::event.title | htmlDecode}}
  </span>
  <span ng-if="!config.eventTitleLinkEnabled" class="event-title" aria-
  describedby="{ {ariaidprefix} {event.eventId} }">{{::event.title | htmlDecode}}</span>
</h4>
```

The ui-sref for eventDetail is:

```
ui-sref="eventDetail({eventId:event.eventId})"
```

In a future lab exercise, the event id will be required.

3. Add a ui-sref directive, confirming the correct placement of parens () and curly braces { } in the code.

```
<a ui-sref="eventGuestPolicy({eventId:event.eventId})">Details</a>
</div>
<div class="event-restrict-single-function-text-align" xe-field="restrictRegi
</div>
<label ng-if="config.description" class="sr-only" >{{ 'events.by.description' | i1
<div ng-if="config.description" class="event-card-desc text-regular-small" xe-sec
<div ng-if="event.limitOne">{{ 'events.registrationRequired' | i18n }}
  <a ui-sref="eventGuestPolicy({eventId:event.eventId})">Details</a>
</div>
<function-card ng-if="config.function" ng-repeat="function in event.functions" d
</div>
```

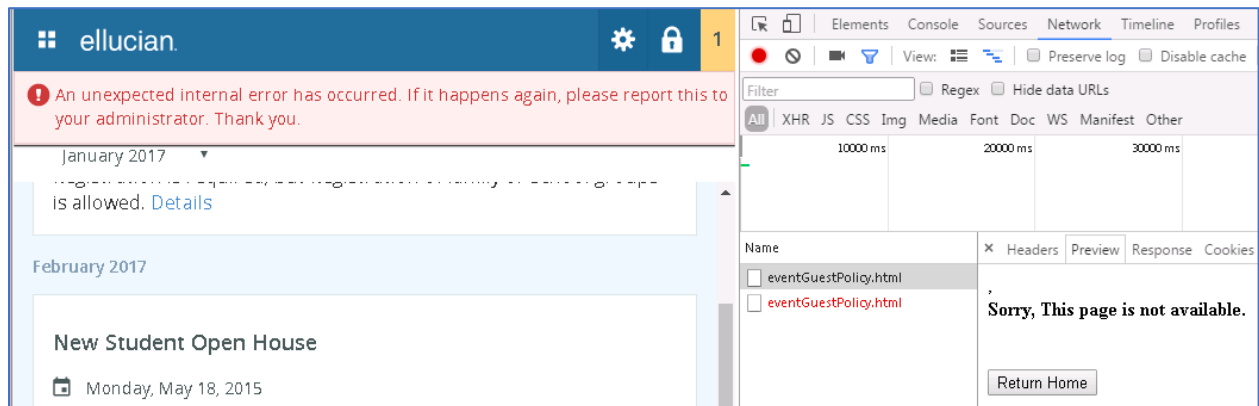
4. Run the app.
The new link appears. However, it is unclickable because AngularJS doesn't have a definition for the new state.
5. Copy the eventDetailState in the config.route.js and name it eventGuestPolicy. The code below is located at: D:\labfiles\lab7and8\guest_policy_state_snippet.js



Note: The resolve section is commented out for now as no data is retrieved in the lab.

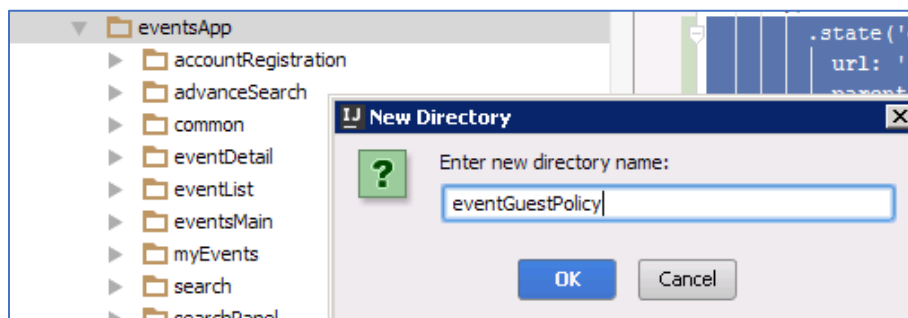
```
.state('eventGuestPolicy', {
  url: '/eventGuestPolicy/:eventId',
  parent: 'eventList',
  views: {
    'content@eventsMain': {
      templateUrl: '../ssb/eventsApp/eventGuestPolicy/eventGuestPolicy.html',
      controller: 'EventGuestPolicyController as eventGuestPolicy',
      /*
      resolve: {
        eventDetailResolve: function (EventDetailService, $stateParams) {
          var params = {
            functionId: $stateParams.functionId
          };
          return
          EventDetailService.getEventDetail($stateParams.eventId,$stateParams.functionId);
        }
      }
      */
    },
  },
  data: {
    'pageTitle': 'events.breadcrumb.eventGuestPolicy',
    'breadcrumb': [
      {label: 'events.breadcrumb.events'},
      {label: 'events.breadcrumb.eventList', url: '/eventList'},
      {label: 'events.breadcrumb.eventGuestPolicy'}
    ]
  }
})
```

- Run the app again. Click the detail link and review the returned error. Inspect the network tab for more information.

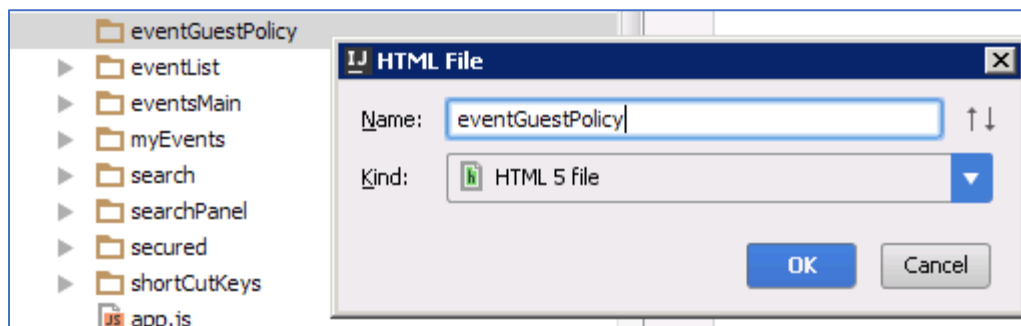


While the state is defined, the page is not defined.

- Add the guest info page.
This only contain stubs information as the nature of the ui-router requires us to only define the required HTML, and not an entire HTML page.
- Stop the App.
- Using the eventDetail state as a model, create the eventGuestPolicy objects.
- Create a new directory under the eventsApp, **eventGuestPolicy**.



- Create a new HTML file in the eventGuestPolicy directory, **eventGuestPolicy.html**.



12. Hard-coded text is required to show we accessed the page.

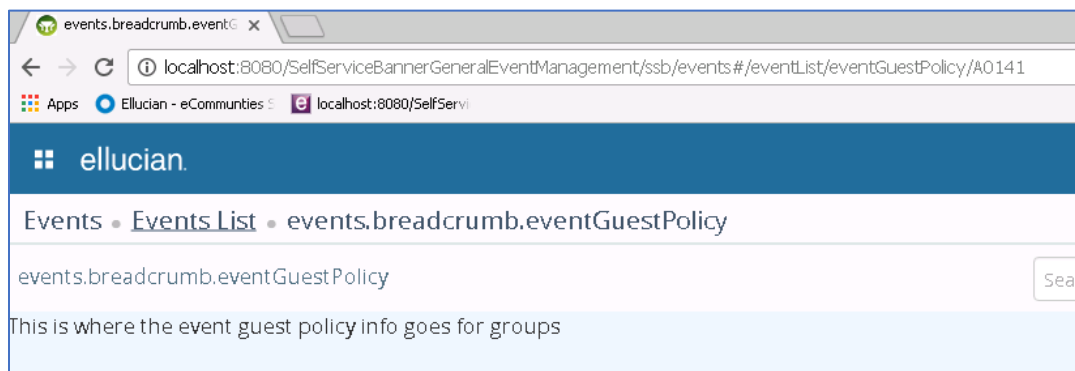
```
<div class="event-list-page" xe-dynamic>
  <div class="eventguestpolicy-page-body-panel" tabindex="-1">
    This is where the event guest policy info goes for groups
  </div>
</div>
```

No data is being received and only the state is changing in AngularJS, there is no need to change anything in grails.

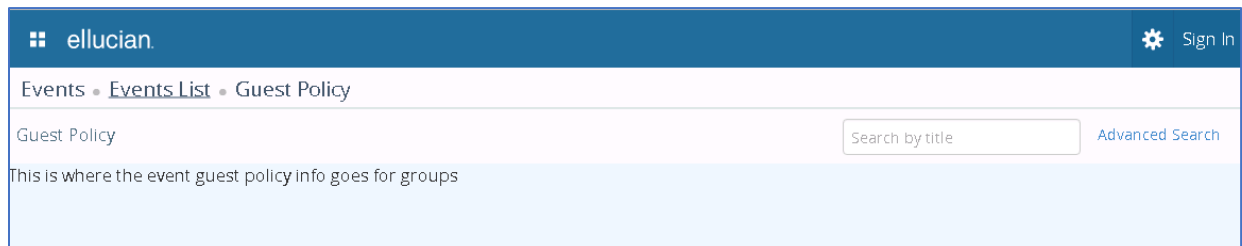
13. Run the App.



Note: The bread crumb path displays incorrectly and needs defining.



14. Refresh the page.



15. Save all your work and commit the changes to Git.

Notes

Lab 8 - Add a Title to the Guest Policy Page

Lab Overview

Extend the event's GuestPolicy page to display the event title.
The purpose of the lab is to understand more fully how data is passed between states in AngularJS using the ui-router'

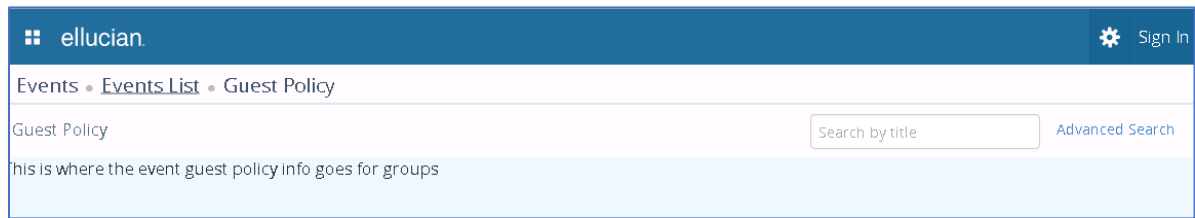
Presently, the code within the eventCardTemplate.html file contains code like that shown below:

```
<div ng-if="event.limitOne">{{ 'events.registrationRequired' | i18n }}  
  <a ui-sref="eventGuestPolicy({eventId:event.eventId})">Details</a>  
</div>
```

This results of the **Details** link appear as shown below.

The screenshot displays the Ellucian Banner 9 Self-Service interface. At the top, there's a blue header with the Ellucian logo and a 'Sign In' button. Below the header, the page title is 'Events • Events List'. A search bar with the placeholder 'Search by title' and a link to 'Advanced Search' is visible. The main content area shows a list of events for February 2017. The first event is 'New Student Open House' on Monday, May 18, 2015, with a description: 'Join faculty, staff and alumni for a new student open house on campus. This is a great opportunity for meeting fellow classmates and networking before your semester beings. Refreshments will be provided.' The second event is 'Tours: New Castle Campus' on Friday, January 06, 2017, with a description: 'Come Visit Us! The Office of Admissions invites you to sign up for a tour of our beautiful main campus in New Castle. A tour will give you the opportunity to visit the campus and learn more about what Wilmington University can do for you. You will have a chance to see our classrooms, Student Center, library, bookstore and much more!' and a 'Details' link. The third event is 'Commencement: Pre-Registration' on Wednesday, February 01, 2017, marked as 'Restricted'.

1. Click the link, the display page appears.



Change this page to display the title of the event. To do so, add the AngularJS controller.

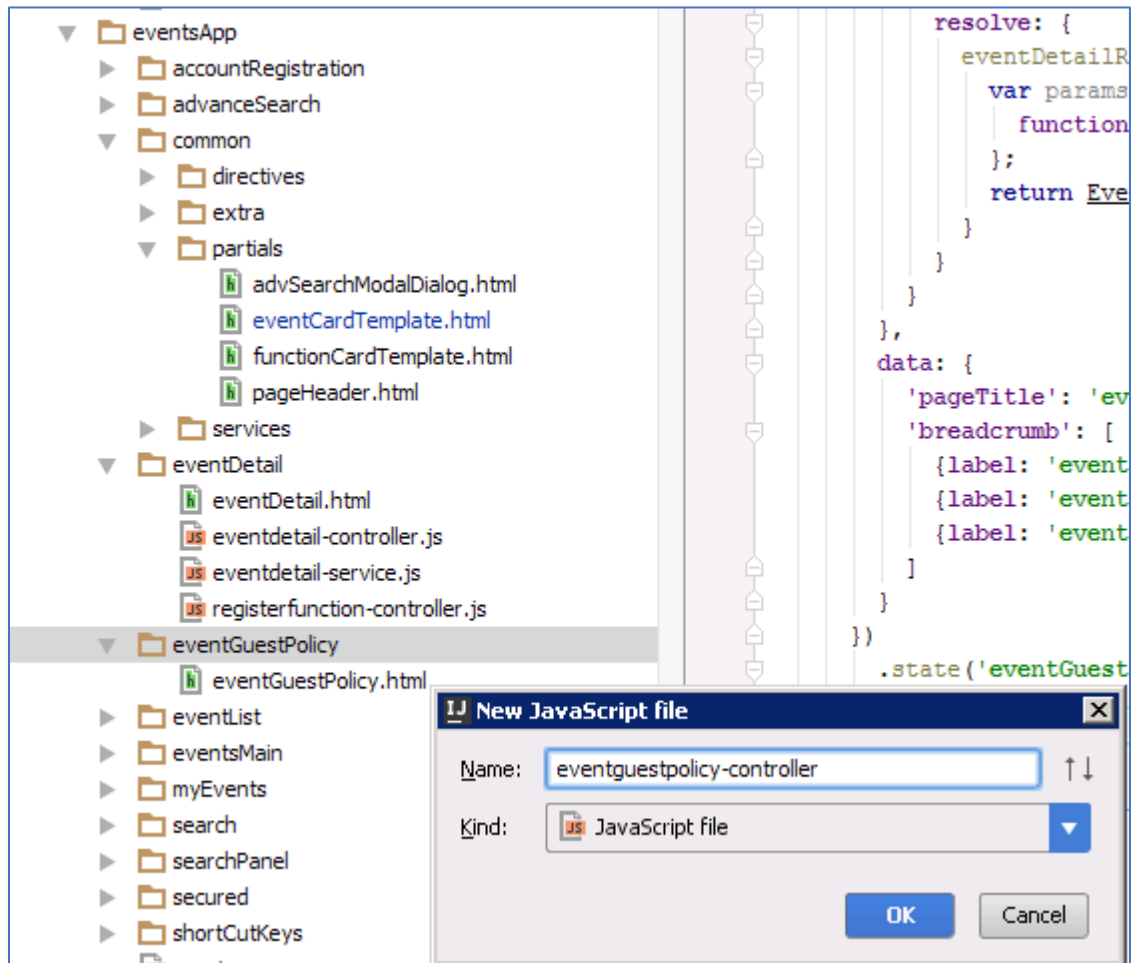
2. Return to the code added in the previous exercise that created the Details link. Modify the code as shown below.

```
<div ng-if="event.limitOne">{{'events.registrationRequired' | i18n}}
  <a ui-sref="eventGuestPolicy({eventId:event.eventId
,eventTitle:event.title})">Details</a>
</div>
```

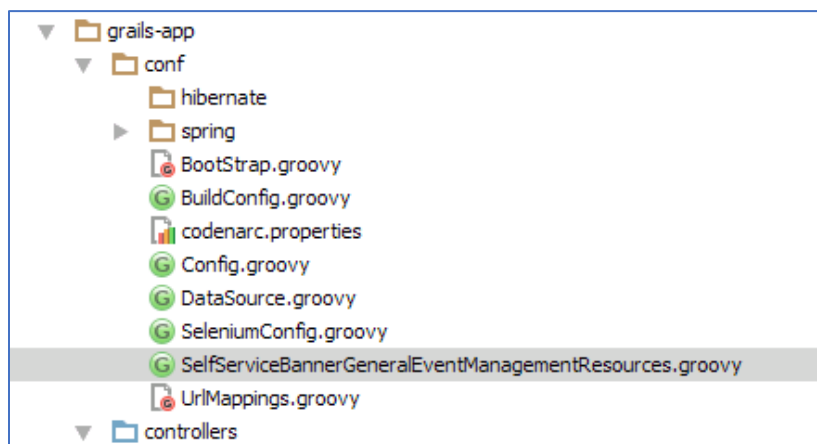
3. Change the config.router.js to resolve the state parameter and pass it in to the controller. In the line, **EventGuestPolicyController as eventGuestPolicy**, the **eventGuestPolicy** allows the reference of any variables global to the controller in the HTML page by using **eventGuestPolicy.xxxxx**. This is further explained later in the lab.

```
.state('eventGuestPolicy', {
  url: '/eventGuestPolicy/:eventId',
  parent: 'eventList',
  params: {
    eventTitle: null
  },
  views: {
    'content@eventsMain': {
      templateUrl: '../ssb/eventsApp/eventGuestPolicy/eventGuestPolicy.html',
      controller: 'EventGuestPolicyController as eventGuestPolicy',
      resolve: {
        eventGuestPolicyResolve: function ($stateParams) {
          if (angular.isDefined($stateParams.eventTitle)) {
            return $stateParams.eventTitle;
          } else {
            return null;
          }
        }
      }
    }
  },
  data: {
    'pageTitle': 'events.breadcrumb.eventGuestPolicy',
    'breadcrumb': [
      {label: 'events.breadcrumb.events'},
      {label: 'events.breadcrumb.eventList', url: '/eventList'},
      {label: 'events.breadcrumb.eventGuestPolicy'}
    ]
  }
})
```

4. Add the AngularJS controller file for the eventGuestPolicy state.



5. When a JavaScript file is added, grails needs to include it in the resources to the browser. Add it to the application resource groovy file in grails-app/conf.
6. For the General Events Self Service app, it is **SelfServiceBannerGeneralEventManagementResources.groovy**.



7. The file has several sections. Review the **events** section.

```
'events' {
  dependsOn "common"
  defaultBundle environment == "development" ? false : "events"
```

8. Add the **eventGuestPolicy** controller after the eventList controller.

```
resource url: [file: 'ssb/eventsApp/eventList/eventlist-controller.js']
resource url: [file: 'ssb/eventsApp/eventList/eventlist-service.js']

resource url: [file: 'ssb/eventsApp/eventGuestPolicy/eventguestpolicy-controller.js']
```

9. The controller code is here but also in D:\labfiles\lab7and8.



Note: The structure is similar to the event detail controller with the controller name changed and the code simplified.

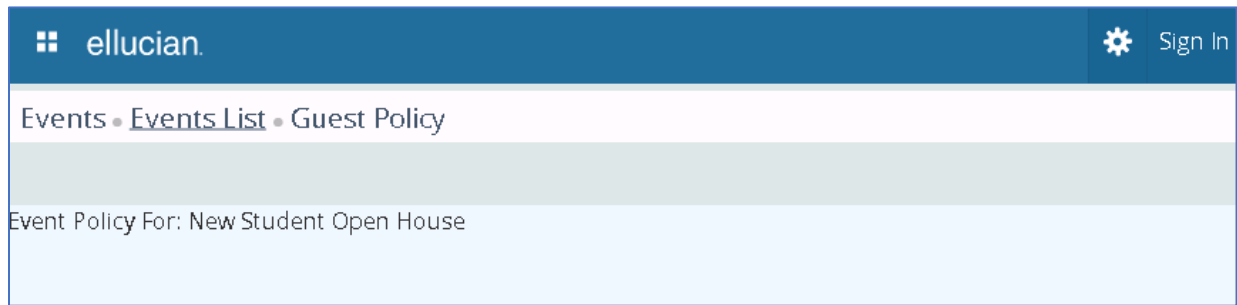
```
/**
 * Created by Administrator on 2/20/2017.
 */
/*****
Copyright 2016 Ellucian Company L.P. and its affiliates.
*****/
(function () {
  'use strict';
  angular
    .module('eventsApp')
    .controller('EventGuestPolicyController', ['eventGuestPolicyResolve',
'$state', 'Config', '$timeout', '$rootScope', 'UtilsService',
EventGuestPolicyController]);

  function EventGuestPolicyController(eventGuestPolicyResolve, $state, Config,
$timeout, $rootScope, UtilsService) {
    var vm = this;
    vm.eventTitle=$state.params.eventTitle;
  };
})();
```

10. Return to the eventGuestPolicy.html file and modify it to include the title. This is where the **EventGuestPolicyController** as **eventGuestPolicy** in the config.router.js for the state is used.

```
<div class="event-list-page" xe-dynamic>
  <div class="eventguestpolicy-page-body-panel" tabindex="-1">
    Event Policy For: {{eventGuestPolicy.eventTitle}}
  </div>
</div>
```

The page appears as below. The changes occur in the browser and not in the web server.



11. Format the page title to look like the other event pages.

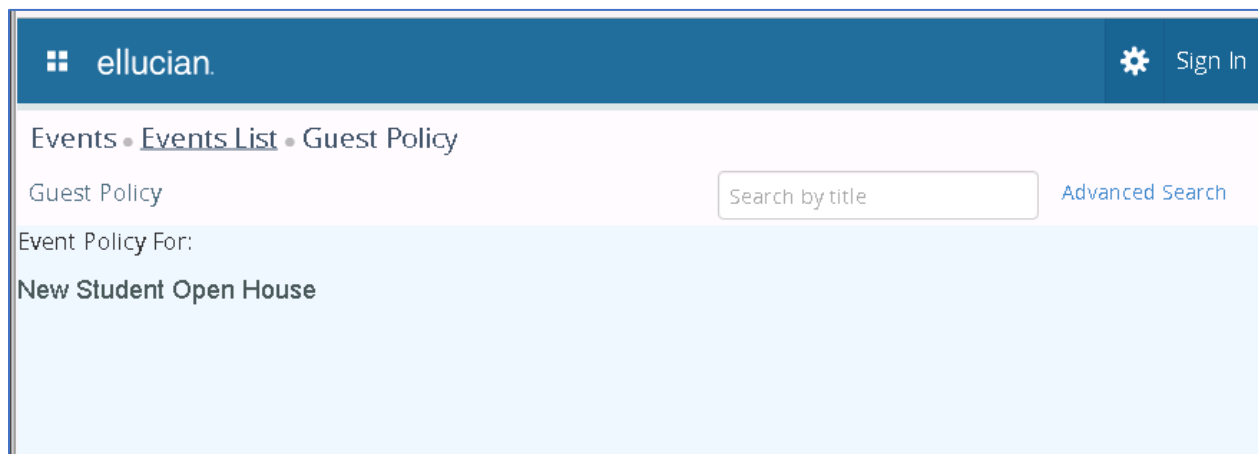
In the **event card template html**, the event title is defined as shown below where the CSS class is included on the page.

```
<div class="event-card-title-outer" xe-section="eventTitleSection">
  <h4 class="event-card-title pull-left text-semi-bold-medium" xe-field="eventTitle"
ng-class="{ 'event-title-facebook-padding': event.facebookEnabled }">
    <span ng-if="config.eventTitleLinkEnabled" class="event-title cursor-icon" ui-
sref="eventDetail({ eventId: event.eventId })" aria-
describedby="{{ ariaidprefix }}{{ event.eventId }}">
      <label class="sr-only">{{ 'default.srOnly.event.title.link' |
i18n }}</label>
      {{ ::event.title | htmlDecode }}
    </span>
    <span ng-if="!config.eventTitleLinkEnabled" class="event-title" aria-
describedby="{{ ariaidprefix }}{{ event.eventId }}">{{ ::event.title | htmlDecode }}</span>
  </h4>
  <div ng-if="event.registered && config.eventRegistrationStatus" xe-
field="registeredTag" class="card-registered text-semi-bold-small" aria-hidden=true
>{{ 'events.registered' | i18n }}</div>
  <div ng-if="!event.registered && event.restrictedEvent &&
config.eventRegistrationStatus" xe-field="restrictedTag" class="card-restricted text-
semi-bold-small">{{ 'events.restricted' | i18n }}</div>
  <div ng-if="event.facebookEnabled" xe-fb-like class="event-title-facebook" xe-
field="fbTag" data-href="{{ fbUrl }}" data-width="23" data-layout="button_count" data-
action="like" data-show-faces="false" data-share="false"></div>
</div>
```

12. Add CSS classes for the HTML to appear as shown below.

```
<div class="event-list-page" xe-dynamic>
  <div class="eventguestpolicy-page-body-panel" tabindex="-1">
    Event Policy For:
    <h4 class="event-card-title">
      <span class="event-title">{{eventGuestPolicy.eventTitle}}</span>
    </h4>
  </div>
</div>
```

The Details page appears as shown below.



13. Shut down the running app and commit the changes to Git.

14. Verify that the new files are included in the commit along with any modified files.

Notes

Before you start the next lab

Try the next exercise by yourself. Think what steps you will need to perform to complete the task. Without looking at the provided steps ask yourself the following questions:

- What will I need to do?
- Why will I need to do those tasks?
- How will I perform those tasks?

After your attempt at the exercise, the instructor will walk through each step and explain the steps. See how you did, and where you still have some questions.

Lab 9 - Create a New Table, Domain, and HTML – the Guest Policy Page

What You'll Learn

In this lab, download Banner 9 source code and verify there is a working application before making changes. In this module, you learn:

- ☐ Add a new table in the back end, and reflect it in a new Domain Model
- ☐ Extend the Controller and Services to fetch the user-chosen data from the Domain Model and to pass it to the web page
- ☐ Extend the Guest Policy web page to display the Guest Policy data for the user's chosen event.

Lab Overview

The goal of this lab is to provide you with the ability to display newly-created data on your new Guest Policy page, complete with the addition of a new Domain Model mapped to a new Oracle table.

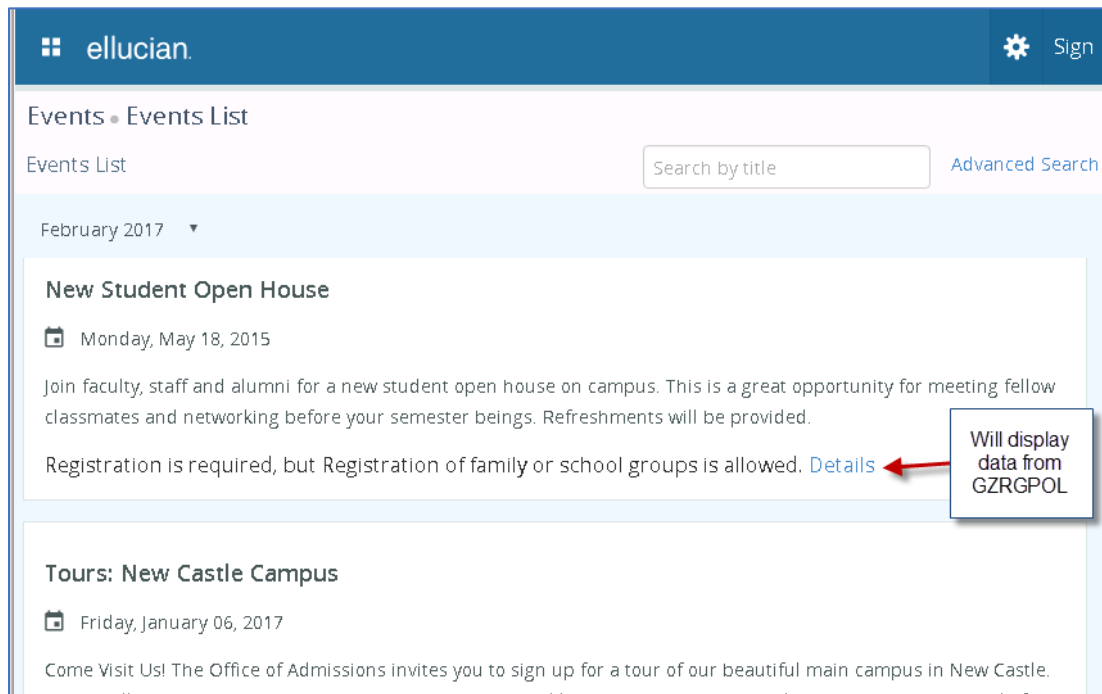
Notes

Scenario

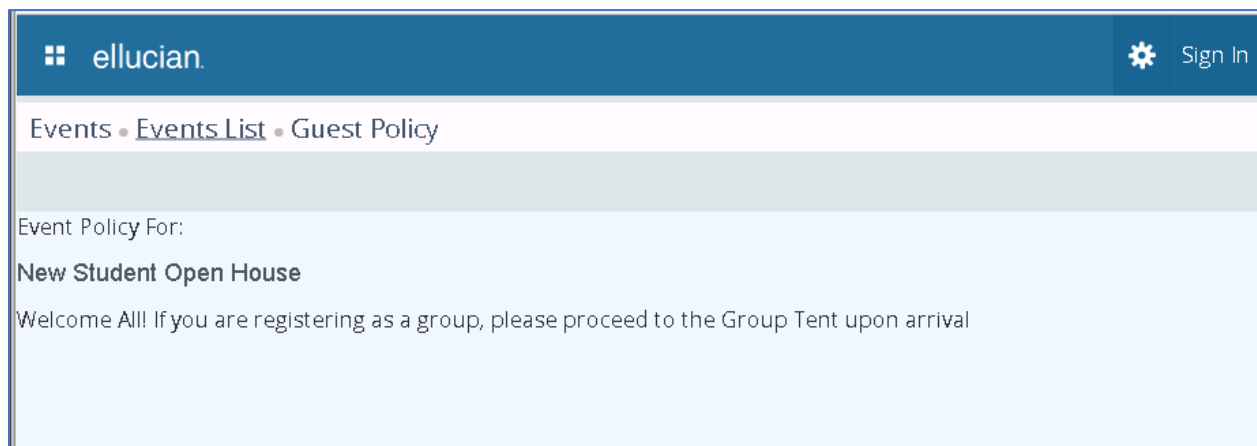
A stub page was added to show a placeholder label indicating 'this is where the guest policy info would go', but it was not completed with the Guest Policy information.

In this lab, the task is finished. We design a back-end Oracle table that will display Guest Policy information for a particular event.

The user clicks on the **Details** link as shown below.



The final product web page displays.



Tasks

Branches

1. Create a branch in Git for this new lab.
2. Create a branch for the banner_general_events.git plugin.

Table

3. Create the new table to hold the Guest Policy information.
This table is already created for this lab, as the GZRGPOL table.
Note the following:



Note: GZRGPOL is a General table, which is also a Repeating table.
The 'Z' denotes that it is not Banner baseline, but something we created ourselves.



Instructor Note: If there are problems with a participant's lab, perform the following:
Grant all on gzrgpol to public; commit;

Domain Model class

4. Add a Domain Model class to represent the table data as an Object Oriented (OO) object.
5. Add this to the same plugin that contains the other events domain models.

Controller/Service

6. Follow the previously used convention of data fetched by using a Controller call to a Service, which in turn calls a Domain Model method that used a Named Query.

HTML Page

7. Add the field to display to the HTML page. This requires adding an AngularJS service to retrieve the data from the Grails controller, Grails service and Grails domain.

Labels

8. Add labels used for the display fields to messages.properties.

The New Oracle Table

For this lab, the table is provided. As reference, the code for creating the table is included in an appendix at the end of this document.

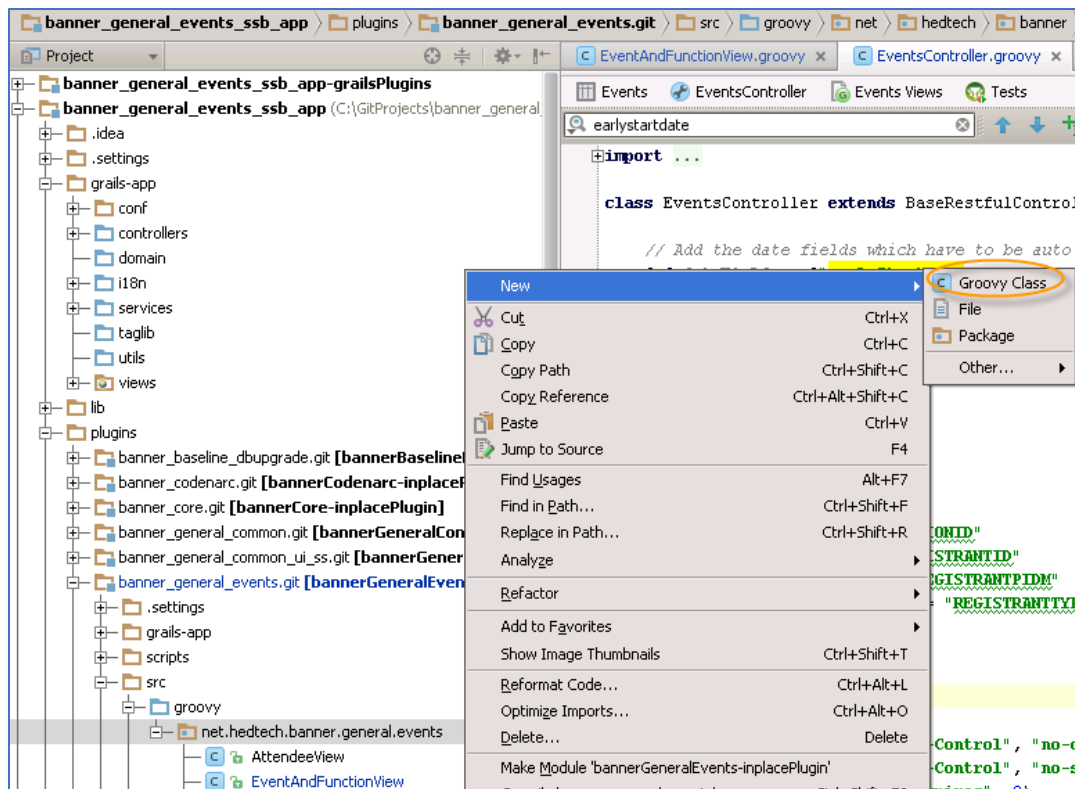
The table is populated with test data that will be linked to the record below: Additional events can have data added by using the following SQL and change the event id to the value from SLBEVNT_CRN that you want to have additional information for.

```
insert into GZRGPOL (
    GZRGPOL_EVENTID
, GZRGPOL_DESCRIPTION
, GZRGPOL_POLICY
)
VALUES (
    'XXXX'
, 'Welcome, All'
, 'Welcome All! If you are registering as a group, please proceed to the
Group Tent upon arrival'
);
```

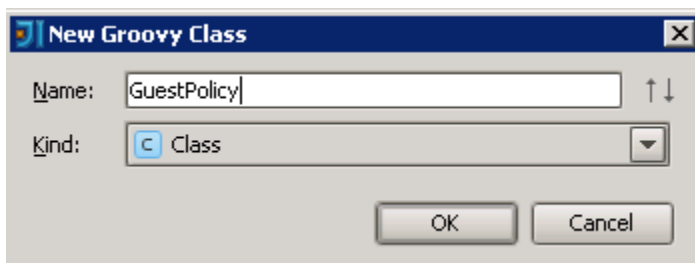

Create a New Domain Model

In IntelliJ, create the domain model. Create this within the plugin that already contains the other events domains, **banner_general_events**.

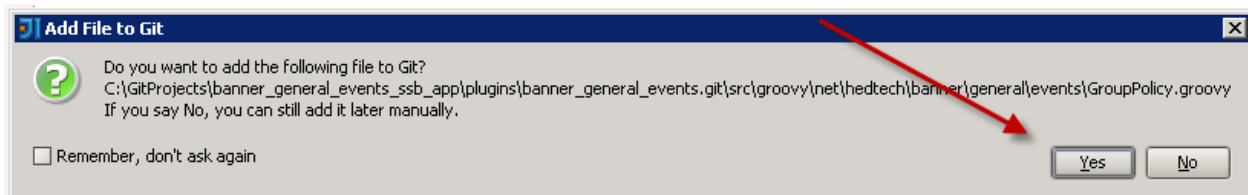
1. Right-click the directory shown in the screenshot below. Choose **New > Groovy Class**.



2. Name the new groove class, **GuestPolicy**. Capitalize the first letter and use camel case.



3. Add it to Git when prompted.



4. With the stub for this file created, complete the following contents, as provided in the **GuestPolicy.groovy** in the **d:\labfiles\lab9** directory.

Normally, copy and paste the entries for the dbeu-created fields from another domain, and modify them to reflect the new table name. Alternatively, copy-paste-and-modify other, similar fields from another domain.

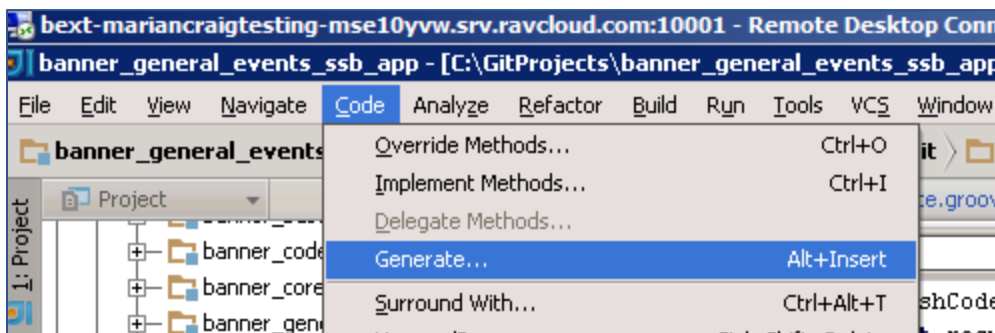


Note: Creating nullable constraints for any un-required field is essential. This code is in the Appendix pertaining to this lab for reference.



Please note that if import statements are showing error conditions, save, resynch, and clean your project.

5. Use IntelliJ to help generate the `toString`, `hashCode` and `equals` methods. With the domain open in the editor, place the mouse cursor before the “public static string `fetchByEventId`” method declaration.
6. Click **Code** > **Generate**.



7. This will open a window to select the methods you want to generate.
8. Click the **toString()** method to generate it.
9. Repeat the process for the **equals()** and **hashCode()**, placing the mouse cursor where the new methods are to be created.
10. Accept the default popup settings as you generate the methods.

Add the New Domain to the Hibernate File

Hibernate must be made aware of the new domain to process it.

1. Modify the **hibernate-banner-general-events.cfg.xml** file, located under your plugin name **banner_general_events.git**, and under **grails-app > conf > hibernate**.
2. Locate the entries for **mapping class**. Add an entry for the new GuestPolicy to this list, by copying and pasting an existing entry and modifying it:



NOTE: Alternatively, if you put the domain class inside your app, not a plugin, you would need to create your own version of the cfg.xml file and put it in your grails-app/conf/hibernate folder.

Extend the Grails Service

The logic must be created to fetch the data based on the Event Id and to pass it to the Event Guest Policy page for display.

Page Logic Flow

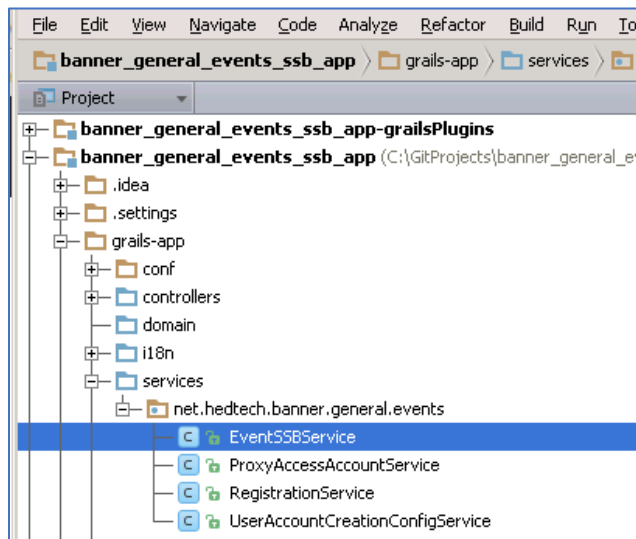
Add the logic to the EventGuestPolicy method within the EventsController.

User requests the **EventGuestPolicy** page

Control goes to the **EventsController** page.

Control goes to the method corresponding to that page.

1. Extend the Service to call the fetch method in the domain. Open the EventSSBService:



2. Remember the name of the fetch method you created in your domain? It is fetchByEventId. Therefore, the code we add to the service is:

- Add an import.

```
import net.hedtech.banner.general.events.GuestPolicy
```

- Add the following method. Be careful to add it between existing methods, taking care not to put it into the middle of another method.

```
def fetchGuestPolicyByEventId(String inEventId) {  
  
    def returnedGuestPolicy = GuestPolicy.fetchByEventId(inEventId)  
    return returnedGuestPolicy  
  
}
```

Extend the Grails Controller

1. Extend the grails-app/controllers/EventsController's eventGuestPolicy method to use the new service call.
2. Add the code as shown below.

```
def eventGuestPolicy = {  
    def model = [:]  
    try {  
        def String guestPolicy =  
eventSSBService.fetchGuestPolicyByEventId(params.eventId)  
        model = [guestPolicy: guestPolicy]  
    } catch (ApplicationException ae) {  
        model = EventsUtility.returnFailureMessage(ae)  
        log.error(ae)  
    }  
    render model as JSON  
}
```

We are returning the guestPolicy as a JSON object so it can be used by the HTML page and AngularJS service and controller.

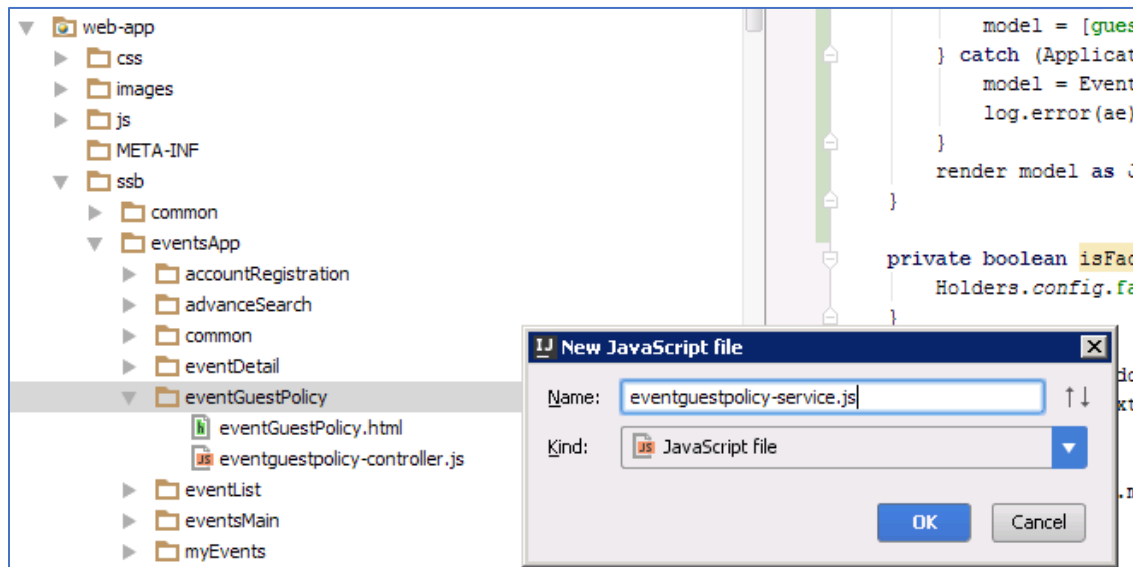
Discussion

What have you learned in this lab that you didn't know before starting the class?

Add the AngularJS Service for the Guest Policy Page

The previous two labs set up the new ui-router state, template html and AngularJS controller. The next step is to add an AngularJS service to retrieve the guest policy information for the specific event from the RESTful API in the Grails Controller.

1. Create the service in the same directory as the AngularJS guest policy controller.



2. The content of the AngularJS service file is shown below.



NOTE: The structure is similar to the AngularJS event detail service. The code is available in **D:\labfiles\lab9**.

```
/**
 * Created by Administrator on 2/20/2017.
 */
/*****
Copyright 2016 Ellucian Company L.P. and its affiliates.
*****/
(function () {
    'use strict';
    angular
        .module('eventsApp')
        .factory('EventGuestPolicyService', ['$resource', EventGuestPolicyService]);

    function EventGuestPolicyService($resource) {
        return {
            GetEventGuestPolicy: GetEventGuestPolicy
        };

        function GetEventGuestPolicy(eventIdParam) {
            var eventGuestPolicyModel = $resource('../ssb/:controller/:action',
                {controller: 'events', action: 'eventGuestPolicy'},
                {get: {method: 'GET'}}
            );

            return eventGuestPolicyModel.get({eventId: eventIdParam}).$promise;
        }
    }
})();
```

3. Add the new AngularJS service java script file to the application resources file so that it's included in files sent from the server to the browser.

If you don't remember what that is or how to do it, check back in lab 8 or check with the instructor.

Update the config.route.js

The guest policy state was created in a previous lab.

The next step is to update the resolve parameter so that it returns the value of the `EventGuestPolicyService.GetEventGuestPolicy` function based on the event ID that was passed in.

```
.state('eventGuestPolicy', {
  url: '/eventGuestPolicy/:eventId',
  parent: 'eventList',
  params: {
    eventId: null
  },
  views: {
    'content@eventsMain': {
      templateUrl: '../ssb/eventsApp/eventGuestPolicy/eventGuestPolicy.html',
      controller: 'EventGuestPolicyController as eventGuestPolicy',
      resolve: {
        eventGuestPolicyResolve: function (EventGuestPolicyService,
$stateParams) {
          return
EventGuestPolicyService.GetEventGuestPolicy($stateParams.eventId);
        }
      }
    }
  },
  data: {
    'pageTitle': 'events.breadcrumb.eventGuestPolicy',
    'breadcrumb': [
      {label: 'events.breadcrumb.events'},
      {label: 'events.breadcrumb.eventList', url: '/eventList'},
      {label: 'events.breadcrumb.eventGuestPolicy'}
    ]
  }
})
```

NOTE: The return must be on the same line as the Service method's name. It is wrapping here to fit, but this will cause errors during runtime.

Update the AngularJS Controller

The basic AngularJS controller was created in the previous lab to set the event title.

Get the result of the resolve function and make it available to the page

```
(function () {  
    'use strict';  
    angular  
        .module('eventsApp')  
        .controller('EventGuestPolicyController', ['eventGuestPolicyResolve',  
        '$state', 'Config', '$timeout', '$rootScope', 'UtilsService',  
        EventGuestPolicyController]);  
  
    function EventGuestPolicyController(eventGuestPolicyResolve,  
    $state, Config, $timeout, $rootScope, UtilsService) {  
        var vm = this;  
        vm.eventTitle=$state.params.eventTitle;  
        vm.policyText = eventGuestPolicyResolve;  
    };  
})();
```

Extend the Guest Policy HTML

The AngularJS service retrieving the policy data is complete, along with the AngularJS controller providing the data. It now must be accessed on the page.

Add the information to the eventGuestPolicy.html.

```
<div class="event-list-page" xe-dynamic>  
    <div class="eventguestpolicy-page-body-panel" tabindex="-1">  
        Event Policy For:  
        <h4 class="event-card-title">  
            <span class="event-title">{{eventGuestPolicy.eventTitle}}</span>  
        </h4>  
        <div class="event-guest-policy-text">  
            {{eventGuestPolicy.policyText.guestPolicy}}  
        </div>  
    </div>  
</div>
```

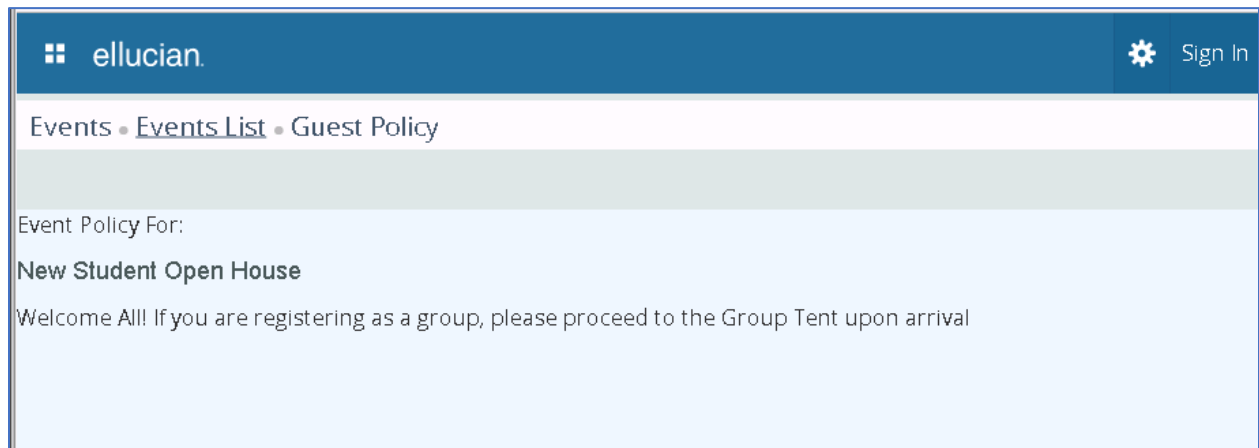
Testing

1. **CTRL-S** to save all.
2. Run a clean.
3. Synchronize your project.



Note: Whenever modifying anything related to fetching data from the domain, always perform a save all, clean, and synch. Then, test by running the app.

4. As the app runs perform the following actions:
 - a. Review the code for the new mods.
 - b. Review for any typos.
 - c. Understand the process and the steps performed.
5. The Guest Policy message displays.



Summary

This ends part three of the agenda.

Part 3 Client-Side Work

9. Plan Code
Mods

10. Review
Module Page
Flow

11. AngularJS

12. JSON

Lab 7 –
Add a Guest
Policy Page

Lab 8 –
Add a Title to the
Guest Policy Page

Lab 9 –
Create a New
Table, Domain
and HTML

Take a few moments to go over your notes and the labs you have completed. What questions do you have?



Lab 10 - Domain Integration Testing (Optional)

Lab Overview

NOTE:

You will execute the “add user script” described in Appendix 3 to complete this exercise.

Take advantage of the automated testing capabilities that come with Grails and IntelliJ to make the work more test-driven.

There is an events record we want to work with the eventId of SSB04.
This is the record shown below:

My University

Home Banner Events

EVENTS LIST

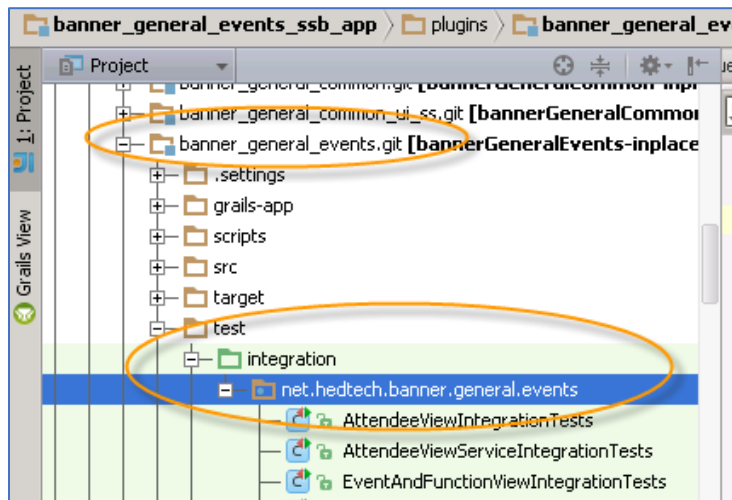
Global Warming (Restricted)
Starts On: Wednesday, February 01, 2012
February 2012 - Effects Of Global Warming Conference (Restricted)
Registration is required, but Registration of family or school groups is allowed. [Details](#)

November 2011 - Campus Visit
Starts On: Wednesday, February 01, 2012
November 2011 - Campus Visit. 12 grade passouts of 2011 can visit the campus. ID card required.
Registration is required, but Registration of family or school groups is allowed. [Details](#)

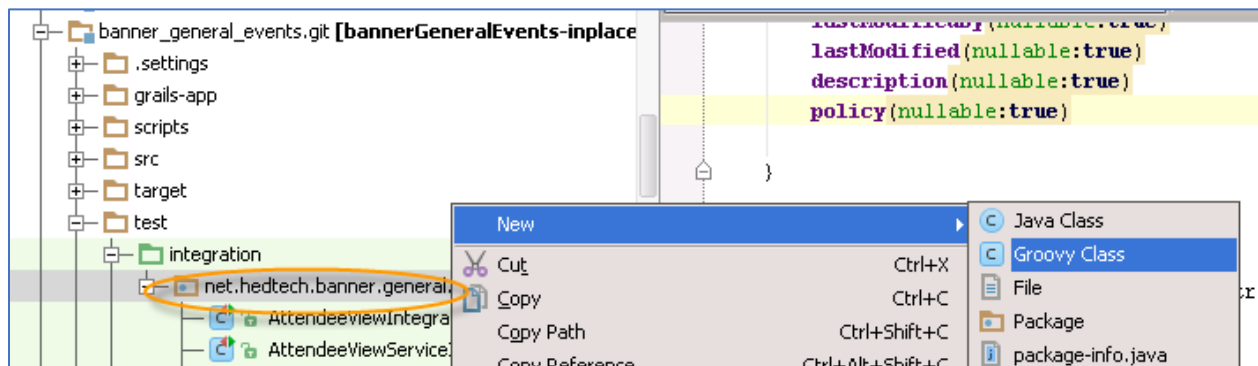
This will be used for the test to see if the fetch method in our domain can find this record.

Notes

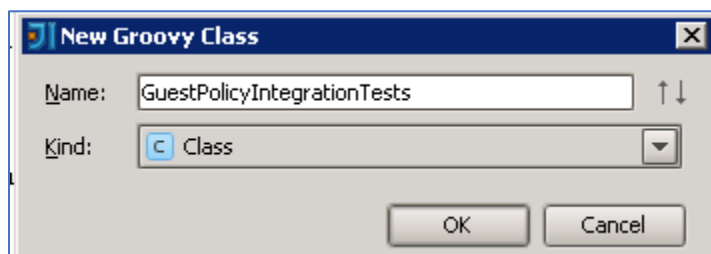
1. Create a test file in the same plugin where our domain resides, within the following directory structure:



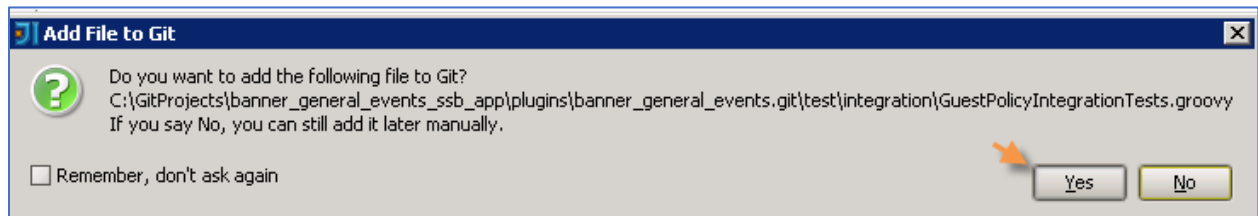
2. Right-click the **integration/net.hedtech.banner.general.events** folder and create a new Groovy Class for the test.



3. Name the new groovy class, **GuestPolicyIntegrationTests**. This follows the appropriate naming convention for such tests, which is Domain Name + IntegrationTests.



- Click **Yes** to add the new file to git.



- In editor, copy and paste import code from another class. Open another file for integration tests in the same directory and copy and paste the imports from that code into the top of your code.

In addition, change the class line to implement the base class as shown below:

```
package net.hedtech.banner.general.events

import groovy.sql.Sql
import net.hedtech.banner.general.system.EventType
import net.hedtech.banner.general.system.SystemIndicator
import net.hedtech.banner.testing.BaseIntegrationTestCase
import org.junit.After
import org.junit.Before
import org.junit.Test
import net.hedtech.banner.general.events.GuestPolicy

class GuestPolicyIntegrationTests extends BaseIntegrationTestCase {
```

- All tests need to have an `@Before` and `@After` section that goes with the `BaseIntegration` setup. These are display below. Add these to your class:

```
class GuestPolicyIntegrationTests extends BaseIntegrationTestCase {

    @Before
    public void setUp() {
        formContext = ['GUAGMNU'] // Since we are not testing a controller,
we need to explicitly set this
        super.setUp()
    }

    @After
    public void tearDown() {
        super.tearDown()
    }
}
```

- Save all.

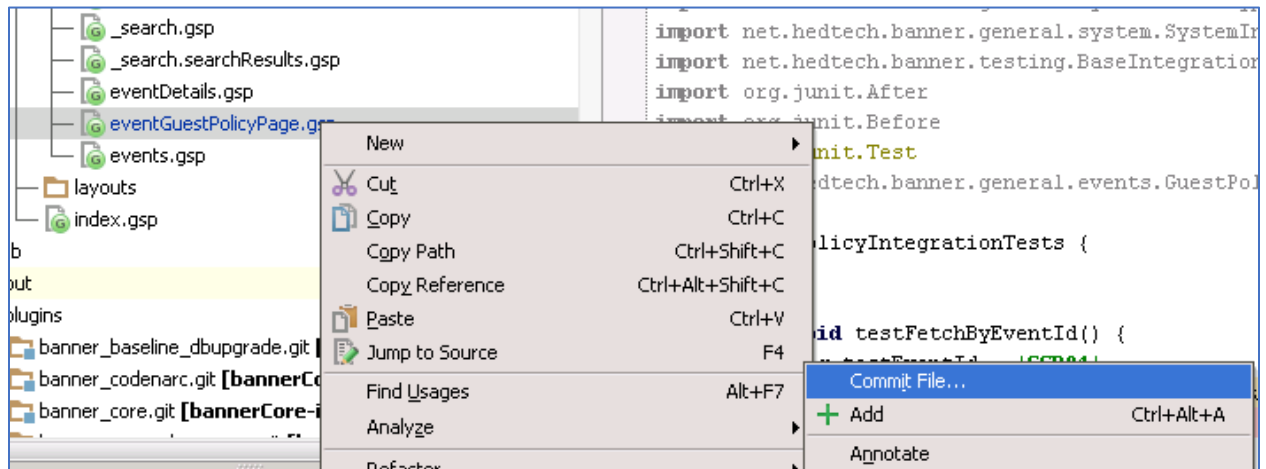
8. Run the test.

Note the **@Test** annotation before the test method and that the test has the name **test** + the name of the fetch method we are testing.

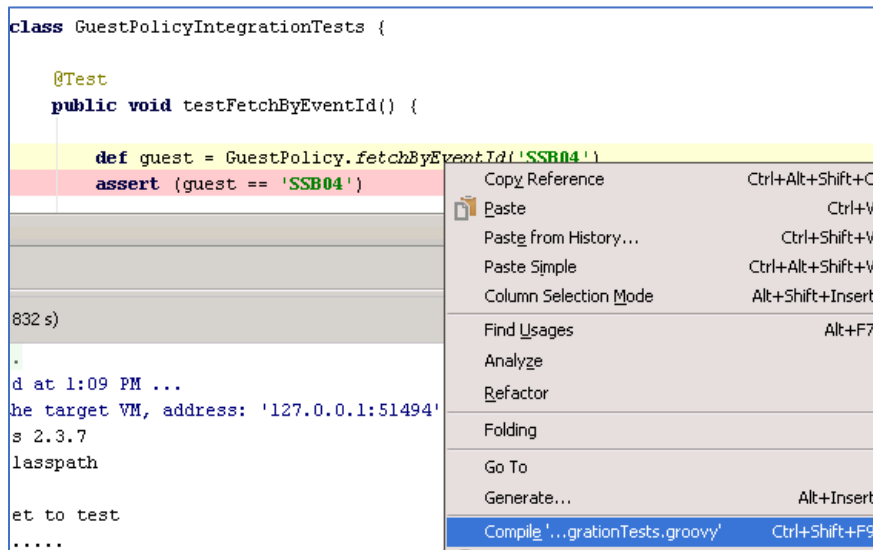
```
@Test
void testFetchByEventId() {
    def guest = GuestPolicy.fetchByEventId('SSB04')
    assert (guest == 'SSB04')
}
```

9. Save all, synchronize your test and clean your project.

10. Commit the files you have changed to date. Commit despite any warnings.



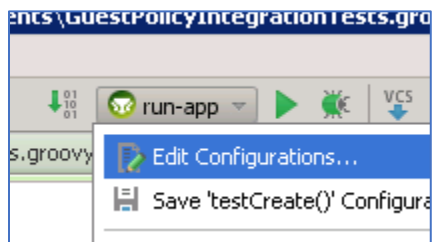
11. Right-click to compile the test.



12. Run the test in Debug mode.

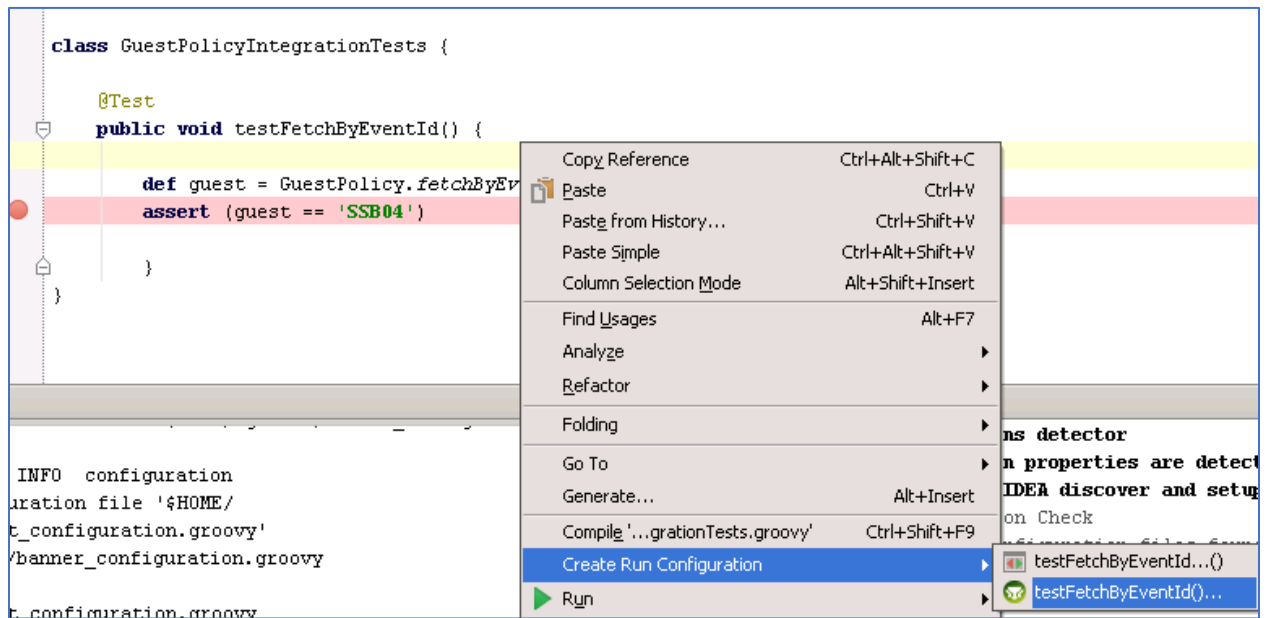
Set a breakpoint by clicking the gutter to the left of the assert line.

13. Open the **run-app** > **Edit Configurations**, and copy the VM options.

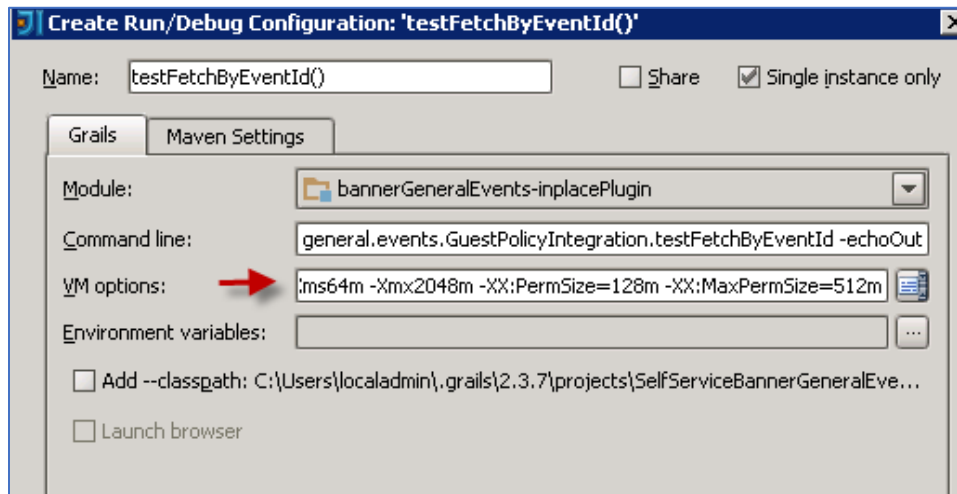


```
-Xms64m -Xmx2048m -XX:PermSize=128m -XX:MaxPermSize=512m
```

14. Set up the test run configuration.
Place your cursor within your test method and right-click.
Choose the following:

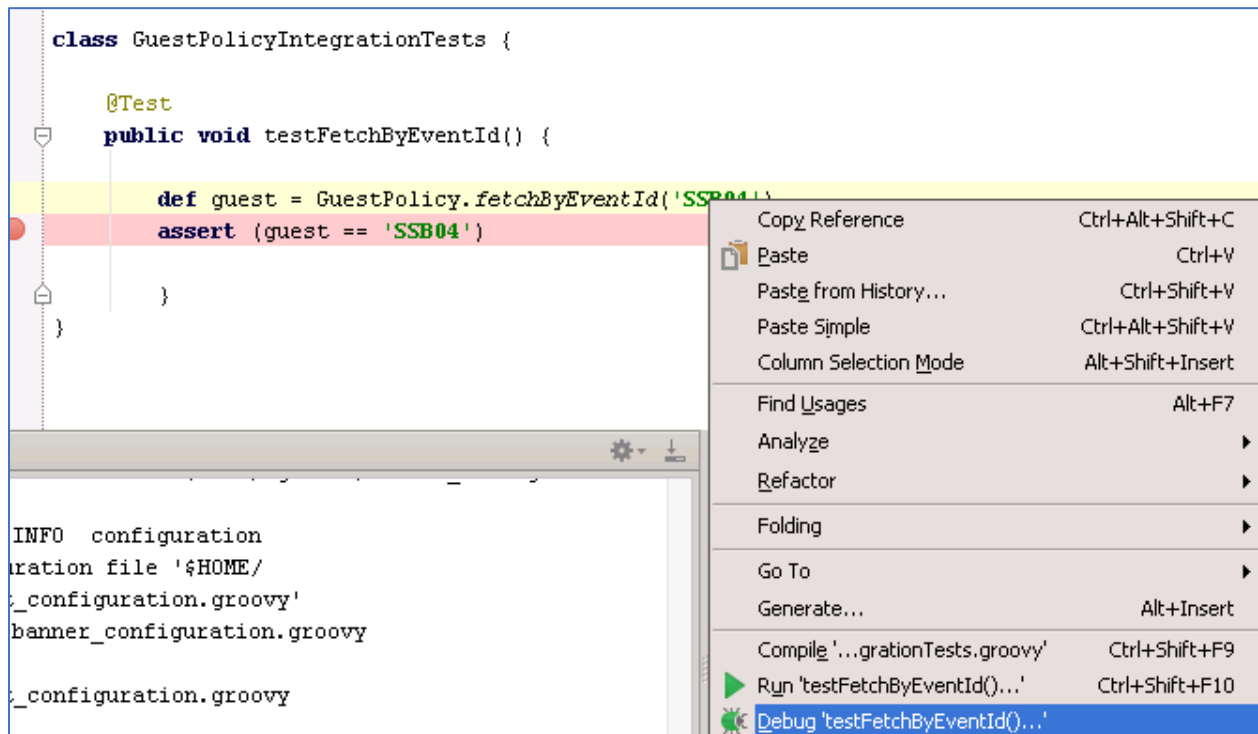


15. Paste your VM string into the test configuration creation dialog box as shown below:

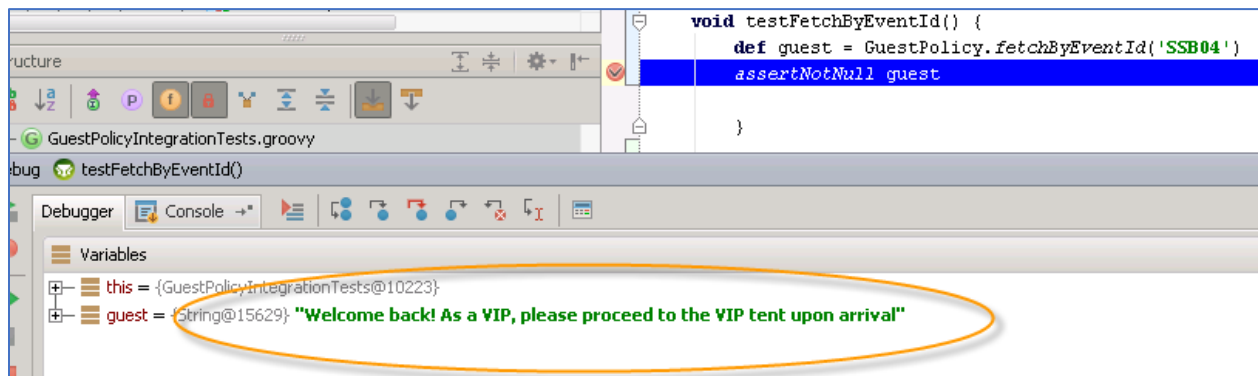


16. Click **Apply** and then **OK**.

17. Right-click to run the test with your cursor within the **testFetchByEventId** method and choosing to Debug:



18. The code will execute until the breakpoint.
The debugging output should show the VIP message as shown in the screenshot, below.
This is a successful output:



19. Stop the execution.

Appendix 1: File Modification Checklist Reference

This checklist lists the most commonly-modified files for Banner 9 Apps and how to identify which ones are involved in adding a field to a form.

For example: You want to add a field similar to **ZIP code** in the Account Registration form.

Use this checklist to identify all the required ZIP code files to copy for your new field.

Backend Oracle table and Domain Model file name	<ol style="list-style-type: none"> 1. Open your running web page that displays the data to be modified. 2. The last word of the of your URL should correspond to a state or route name, such as, events. 3. From the IDE controllers directory, open your controller with the name similar to EventsController. 4. Locate the actions corresponding action method. Actions begin with "def events =". 5. In the closure method, locate the line within the method with the string "Service". This is the name of the service file. If Located: Record this in your programmer's notes. If not Located: <ol style="list-style-type: none"> a) Search for render statements in the action method. b) Open the html file being rendered. c) Search for lines with the string include. There will be more than one string. Find the string that relates to the data you are displaying. For example, you can eliminate an include using the action item search and instead deduce that the other, one called eventsList is the one you want, because you are looking for a page that lists events. The include you choose will point to a controller and an action. d) Return to the referenced controller file and find this action method. e) Search for the string Service within this action method. Select the line that also has the string fetch in it. This line contains the name of your service file. f) Record this in your programmer's notes. <ol style="list-style-type: none"> 6. With the name of your service file, search within the directory where your project is downloaded for the file with the suffix .groovy. 7. In IDE, locate and open the file. 8. Within your service file, locate the fetch method used to retrieve the data. This fetch method is preceded by its calling class. This is the name of the domain. <div data-bbox="391 1577 1073 1759"> <pre> /** * fetchEventList returns a list of events published on the web for a given * registrant. */ def fetchEventListForRegistrant(String registrantID, String registrantID, Date firstDayOfMonthInGregorian = dateConverterService.getGregorian Date lastDayOfMonthInGregorian = dateConverterService.getGregorian List events = EventAndFunctionView.fetchAllCurrentMonthPublishedEv </pre> </div>
--	---

Backend Oracle table and Domain Model file name (cont.)	<p>9. Search for this file. It will end in groovy. In this example, it is the file EventAndFunctionView.groovy.</p> <p>10. Record this in your programmer's notes.</p> <p>11. Locate and open your domain file in your IDE.</p> <p>12. Search for the hibernate tag starting with @table. Your table or view name appears after that tag.</p>
HTML Pages	<p>1. In the running application, position your cursor on the page you wish to extend.</p> <p>2. In the application URL, the directory is located after ssb and the # symbol.</p> <div data-bbox="435 573 1446 646" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> om:7005/SelfServiceBannerGeneralEventManagement/ssb/events#/eventList </div> <p>Note: The directory is located in the web-app/ssb.</p> <p>3. The config.route.js shows the states/routes and the associated template URL (html file), controller, etc. are located.</p>
.CSS files	<p>By default, the formatting of field values, is performed in the /web-app/css/views director and then within the directory corresponding to either the form name or the form grouping name.</p> <p>Additionally, you could follow the same process explained earlier in this document wherein the subject running web page was examined and the right-click and 'inspect element' was used.</p> <p>1. With your web application running and web page open, right-click the image to modify.</p> <p>2. Choose inspect element. The dom/html tool console opens. Use this console to identify the name of the image file and in which .CSS file it is specified.</p>
Decorator file	<p>Decorator files are used with self-service pages to format search and scrollbar capabilities.</p> <p>Decorator files, if used for a page, list the fields that are displayed on your form.</p> <p>It is standard to have a decorator in your project's <app name>/src/groovy folder with the same name as the form or the grouping belong to the form.</p>
Message .properties	<p>For example: banner_general_events_ssb_app/src/groovy/</p> <p>Confirm the application is open as project in the IDE.</p> <p>Under your project app, from grails-app/i18n/ and find this file.</p>

Controller	<p>Confirm the application is open as project in the IDE.</p> <ul style="list-style-type: none">• For a group of subviews that are grouped under a subdirectory, the controller name will have your subdirectory name (like 'events') + Controller.groovy.• For a stand-alone html pages, such as AccountRegistration, it will be your html page name + Controller.groovy. <p>Under the project app, go to grails-app/Controllers and find your controller file there.</p>
Generated Integration and Service Integration Tests (currently not supported, but this information is for reference)	<p>Make sure the application is open as a project in your IDE.</p> <p>The name of the Integration test is the: Domain Name + IntegrationTest.groovy.</p> <p>The name of the Service Integration test is the: Service name + Integration Test.</p> <p>These are under the same plugins as the Domain/Service respectively, and then under Test/Integration.</p>

Appendix 2: Self-Service Banner 9 Application Downloading and Environment Setup

This White Paper provides guidance for the primary technical point of contact at a University or College to download Banner 9 Self Service Banner applications to provide them to their development team for initial examination.

Audience

The audience for this exercise is the administrator responsible for downloading Banner 9 applications for the institution.

Scope

The scope is limited to how to:

- ☐ Get the Apps first downloaded
- ☐ Set up your IntelliJ Environment.

Setting up the Git repository is not covered, and information for doing that can be found in the Ellucian Communities. The download process presented here is also on the Ellucian Communities and on the Ellucian Customer Center website in FAQ (Frequently Asked Questions) documents.

Introduction

The Administrator sets up the Git repository for the institution and downloads the Ellucian Apps into it. Developers at the institution download from that school's repository.

Again, setting up the institution's Git repository is not covered in this paper, and information for doing that can be found in the Communities, accessible at <https://ecomunities.ellucian.com/>. The download process presented here is also available on the Commons and on the Ellucian Customer Center websites in FAQ (Frequently Asked Questions) documents.

It is recommended that administrators and developers complete the courses offered by Education Services that teach the underlying technology. Learn more about these courses by accessing training information at www.ellucian.com. Alternatively, email us at edcenter@ellucian.com.

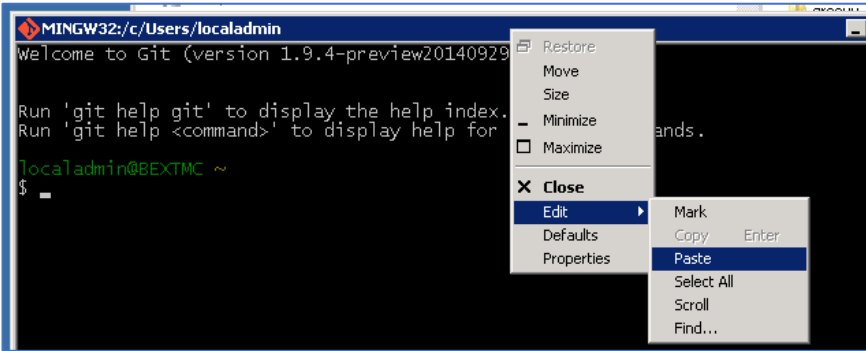
Process

The process to get this download has the following steps:

1. Download Git software which will be needed for the next steps and set up the Git repository.
2. Generate an RSA key pair on the machine that will be receiving the download and then submit that key as part of a Request. This request will be in the form of a Case (Service Request) using the Ellucian Customer Center.
3. Download and set up an IDE (Integrated Development Environment) like IntelliJ. You must use IntelliJ version 13 or higher, along with specific versions of Java Development Kit (JDK, also known as an SE. or Software Environment) and Grails needed for the specific Apps you will be downloading.
4. Download the desired Banner 9 Apps using Git Bash commands.
5. Configure your Database Connectivity
6. Open the Banner 9 App in the IDE and run the App as a project.

Download Git Software	
1	Download Git , http://git-scm.com/ .
2	<p>Choose to run git from the git bash only. This tool, "Git Bash" will run on your desktop.</p> <p>Note: The setup of the Git Repository is not included in this document. You can find information for this topic on the Communities site at https://ecomunities.ellucian.com/.</p>
3	Generate an RSA key pair on the machine receiving the download and submit that key as part of a Request to Ellucian.
4	Use the Git Bash window on your system to create an RSA key pair used to secure the connection for downloading from Ellucian's 9 software repository.
5	Place the generated public key into a text file, and submit it to Ellucian along with a Case / Service Request. Create this Case using the Ellucian Customer Center at https://ellucian.force.com/clients/s/ . Please make this a priority 4 Case.
6	In the request, specify which applications are planned for download and to ask the Service Desk to provide the correct Java version and correct Groovy/Grails (if applicable) version for the App.
7	<p>Generate the RSA key.</p> <ol style="list-style-type: none"> Open Git bash. At the prompt, enter: ssh-keygen -t rsa -C your.name@school.edu. When prompted to enter a file, click Enter. Enter and note the passphrase you choose. Messages that look like the following (with your user directory name): <p>Your identification has been saved in /c/Users/YourUserName/.ssh/id_rsa. Your public key has been saved in /c/Users/YourUserName/.ssh/id_rsa.pub.</p> <ol style="list-style-type: none"> Attach the id_rsa.pub file to your Case and submit it to Ellucian. Upon confirmation from Ellucian that access has been granted, test the access in your Git Bash window by entering the following: ssh git@source.ellucian.com info
8	<p>You should be shown a listing of files that can be downloaded. If you don't, then troubleshoot by performing the following steps:</p> <p>Go to your c:/Users/YourUserName/.ssh (where 'YourUserName' means to substitute your actual user name) directory and make sure you have at least two files within that directory: id_rsa and id_rsa.pub. If you don't have these two files in this specific directory, follow the directions again, re-generating the keys.</p>

Download and Setup an Integrated Development Environment (IDE)	
9	<p>Download and set up an IDE (Integrated Development Environment) such as IntelliJ along with specific versions of Java and Grails. The IntelliJ version must be the Ultimate version 13 or higher.</p> <p>Note: Different Banner 9 applications require different versions of Java/Grails.</p> <ul style="list-style-type: none"> • Some applications require specific versions • Some applications only specify minimum versions <p>It is necessary to install the correct or the application will not run successfully. The version requirements for each application are available in the installation guide on the Ellucian Customer Center.</p>
10	Verify that you have the correct version of Grails in its Application.properties file.
11	You can have multiple versions of Grails and Java downloaded and available, which is needed if you have more than one App installed.
12	<p>Download Java: Java is a free download available from Oracle. Locate and install the Java version for your Ellucian application. The default installation folder is c:/Program Files/Java.</p>
13	<p>Download Grails: Grails is a free download available from Grails.org. Locate and install the Grails version for your Ellucian application. The default installation folder is c:/Users/YourUserName/.</p>
14	<p>Set the ClassPath: Configure classpaths to enable the software to be available in the IDE.</p> <p>Windows7/8/10 Right-click My Computer/This PC > Properties. Click Advanced System Settings > Advanced > Environment Variables, and configure the path variable.</p> <p>The classpath should include entries similar to: c:\Program Files\grails-2.2.1\bin and C:\Program Files\Java\jdk1.6.0_45\bin. (Specify the appropriate versions).</p> <p>Verify the Grails classpath In the target environment open a windows command prompt and enter grails -version</p> <p>Verify the Java classpath In the target environment open a windows command prompt and enter java -version</p>

Download Banner 9 Applications Using Git Bash Command	
15	Review the available downloadable applications by using the Git Bash tool and enter: ssh git@source.ellucian.com info
16	Note the name of the application to download: ex: banner_general_events_app .
17	Create a directory to hold your downloaded apps. Example: c:/GitProjects .
18	In your application, open the Git Bash window. Note: Use the Git Bash window and not the command prompt window.
19	Click CTRL-Insert , or right-click the blue bar of the git bash window and choose Edit > Paste , to copy the code from the notepad into a Git Bash window. 

Git Bash Commands	
20	<p>Repeat these steps for each of the commands:</p> <ol style="list-style-type: none"> Enter the command. Wait for the command to successfully process. Verify there are no error codes before continuing. <div data-bbox="381 401 466 485" data-label="Image"> </div> <p>example git commands.txt</p> <p>Commands are located in the text folder and copied below.</p>
a	<pre>cd c:/GitProjects git clone ssh://git@source.ellucian.com/banner/apps/banner_general_events_app</pre>
b	<pre>cd banner_general_events_app sed -i "s/ssh:\/\/\/git@devgit1.ellucian.com/git:\/\/85.190.181.207/g" .gitmodules</pre> <p>Modifies the .gitmodules file to point to the downloaded server. This enables the plugins associated with the project to be downloaded on the first opening of your project or updated on subsequent openings.</p>
c	<pre>git submodule sync</pre> <p>Verifies the most recent, committed updates to the various code components.</p>
d	<pre>git submodule update --init</pre> <p>Be patient as this may take several minutes.</p>
e	<pre>git checkout -b myBranch</pre> <p>Name of the checkout and be anything you wish.</p>

Configure Database Connectivity

Connectivity information is provided by the DBA with DB configuration responsibilities at your institution.

This information will be used in the file:

c:/Users/YourUserName/.grails/banner_configuration.groovy.

An example copy of this file may be requested from:

- The Ellucian Service Desk via Service Request
- From the instructor of the Banner 9 App courses

An example copy of this file is attached.

Within this file, search for the strings:

- bannerDataSource.url
- bannerSsbDataSource.url

Modify these lines to contain the correct database connectivity, including password information, provided by your DBA.

Example:

```
bannerDataSource.url = jdbc:oracle:thin:@000.000.00.00:1521:MYDB
```

Run the Application as a Project

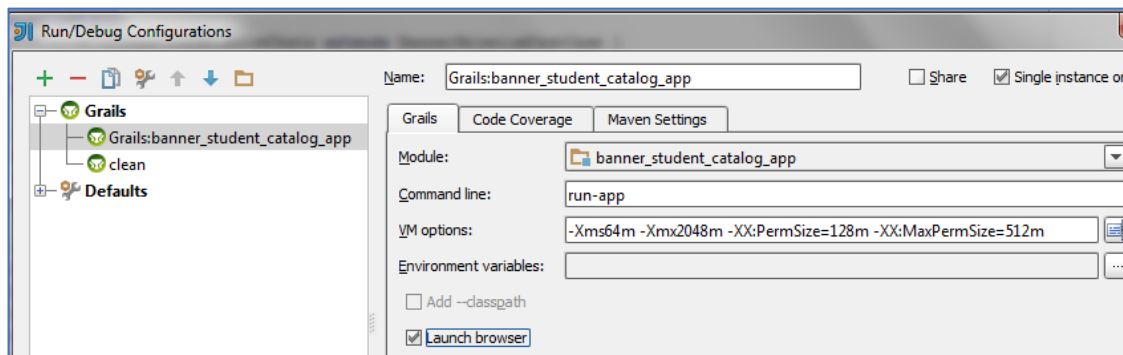
Open the Banner 9 application in the IDE and run the application as a project. This process varies with the version of IDE used. Follow this checklist to help with successfully opening the application.

- 1 Create a new project from existing sources. The source would be the App you downloaded in your **c:/GitProjects** file.
- 2 Confirm the project you create points to this source.
- 3 Choose the correct versions of Java (SDK) and Grails.

If the IDE does not allow the choice of version, then the downloads may not have been successful, or the class path entries for them may have been set up incorrectly. Return and review the directions as shown earlier in this document.

- 4 With the application project successfully created, configure a run-app command with the appropriate VM parameters.

- 5 The appearance of the interface varies depending on the version of the IDE. Referring to the screen below:
 - The green arrow at the top right-hand corner of the IDE screen is the run command icon.
 - Beside the green arrow is the name of a default run configuration used to edit the configuration.



- Modify the default configuration to those shown above:
 - Change the name to **run-app**.
 - Choose your module from the grey drop-down box.
 - Enter the VM options.

- 6 If you are working with a 9.3 app, which uses Grails 2.2.1, then you must use the following VM options:
-Xms64m -Xmx2048m -XX:PermSize=128m -XX:MaxPermSize=512m

 If you are working with a less recent App, one that uses Grails 2.3.7, you must use the following VM options:
-Xms64m -Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=256m

- 7 Verify **Launch Browser** is checked.
Verify **Make** is unchecked.

- 8 Click **Apply**.

Appendix 3: Create Grails_User

The integrations tests require a `grails_user` to exist in the database as an Oracle user. The following script may be used:

```
connect bansecr/%%bansecr_password
-- create the grails_user test ID
whenever sqlerror continue
create user GRAILS_USER identified by u_pick_it;
whenever sqlerror exit rollback
grant connect, resource, ban_default_m to GRAILS_USER;
alter user GRAILS_USER grant connect through banproxy;
insert into gurucsls ( gurucsls_userid, gurucsls_class_code,
gurucsls_activity_date, gurucsls_user_id)
select 'GRAILS_USER', 'BAN_GENERAL_C', sysdate, user
from dual
where not exists (select 1 from gurucsls
where gurucsls_userid = 'GRAILS_USER'
and gurucsls_class_code = 'BAN_GENERAL_C');
insert into gurucsls ( gurucsls_userid, gurucsls_class_code,
gurucsls_activity_date, gurucsls_user_id)
select 'GRAILS_USER', 'BAN_STUDENT_C', sysdate, user
from dual
where not exists (select 1 from gurucsls
where gurucsls_userid = 'GRAILS_USER'
and gurucsls_class_code = 'BAN_STUDENT_C');
insert into gurucsls ( gurucsls_userid, gurucsls_class_code,
gurucsls_activity_date, gurucsls_user_id)
select 'GRAILS_USER', 'BAN_ARSYS_C', sysdate, user
from dual
where not exists (select 1 from gurucsls
where gurucsls_userid = 'GRAILS_USER'
and gurucsls_class_code = 'BAN_ARSYS_C');
insert into gurucsls ( gurucsls_userid, gurucsls_class_code,
gurucsls_activity_date, gurucsls_user_id)
select 'GRAILS_USER', 'BAN_PAYROLL_C', sysdate, user
from dual
where not exists (select 1 from gurucsls
where gurucsls_userid = 'GRAILS_USER'
and gurucsls_class_code = 'BAN_PAYROLL_C');
Commit;
```