Create a Simple Page (Also, An Intro to Groovy)

Create a file called "helloWorld.gsp" in yourmodule/omod/src/main/webapp/pages/.

```
Hello, world. Welcome to <b>OpenMRS</b>.
```

Now browse to the following link:

http://localhost:8080/openmrs/yourmodule/helloWorld.page

Congratulations, you've written your first page! A page that doesn't need to hit the database itself doesn't need a controller--it can be as simple as just a single GSP file.
Now let's make this page a bit more interesting. Change its code to be the following:

```
Hello, world.

<% if (context.authenticated) { %>
    And a special hello to you, $context.authenticatedUser.personName.fullName.
    Your roles are:
    <% context.authenticatedUser.roles.findAll { !it.retired }.each { %>
        $it.role ($it.description)
    <% } %>
<% } else { %>
    You are not logged in.
<% } %>
```

Now refresh the page. It should show you additional information. Also, notice that you did not have to reload anything for that change to take effect. Now let's step through what we did.

First, the GSP file extension tells the UI framework that this page is written as a Groovy template. Groovy templates are very similar to JSPs--they allow us to write regular HTML, with interpolated groovy code. You may insert groovy code using any of these forms:

```
$aVariableToPrint
${ anExpressionToPrint }
<%= "something to print which may include ${ somethingElse }" %>
<% log.debug("Calls a method but prints nothing") %>
```

In order to keep our code consistent and readable, please use the dollar-curly-brace construct as much as possible. For very short expressions, you may use just a dollar sign, but if you are going more than one property deep, or calling a method, use curly braces for clarity. Only use the less-than-percent-equals construct if you need to include an inner dollar-curly-brace expression within your outer one.

Also, please put a single space after the opening curly brace and before the closing one.

Note that the dollar sign is a special character in groovy templates. So if you are writing javascript, you need to escape it like \$. Since this makes it annoying to use jQuery, as a convenience you probably want to add something like this to the standard includes for all your pages:

```
<script>
    var jq = jQuery;
</script>
```

You can transform your groovy objects to be used in your javascript;

```
<script>
        // pass a list of names of diagnoses from the patientSummary received from the controller into a
javascriptFunction, i use '|s|' as my splitter and then remove empty items after the split
                javascriptFunction("<% patientSummary.diagnoses.each { d -> %>|s|${d.name}<% } %>".split("|s|").
filter(v=>v!=''));
</script>
```

You may be wondering what Groovy code is like. 99% of legal Java code is also legal Groovy code, but Groovy gives you a bunch of extra language constructs which allow the code to be a lot cleaner and shorter.

- object.property notation will automatically find Java bean getters and setters. So "context.authenticated" is equivalent to "context. isAuthenticated()".
- You don't need semi-colons. A line break is sufficient to end a line of code. For style, please avoid using unnecessary semi-colons.

- You don't need return statements. The value of the last line of any code block is returned. For legibility, please use an explicit "return" in long methods, but not for short code blocks.
- Groovy supports Java's loop constructs, but it also adds "closures", and adds several methods to Java's Collection and Map classes which work with closures. There's a lot of complexity behind closures, but for now you may think of them like anonymous functions whose argument is called "it". (See complete details.)
    - In this example, on the collection of roles, we call the "findAll" method and pass it a simple closure. The closure is run on every element of the roles collection, and all the elements for which the closure returns true are collected into a new collection. (Remember that an explicit "return" statement is unnecessary, so "!it.retired" is equivalent to "return !it.isRetired()" in Java.)
    - The resulting collection (i.e. the user's non-retired roles) then has the "each" method called on it. "each" also takes a closure, which it executes on each element in the collection. In this example (because the percent-greater-than takes us from Groovy back to HTML) the closure is effectively doing out.println("$it.role ($it.description)").
    - Don't worry if closures still seem a bit confusing, they will become clearer over time.

Oh, and where does "context" come from? The UI framework automatically binds several variables to the template, including an org.openmrs.api.context.Context as "context" so that you have a convenient way to access the OpenMRS API. This is incredibly powerful, but remember, "with great power comes great responsibility". To keep our code clean and legible, do *not* write significant API-related logic in GSP pages. It's fine to do simple things like "context.locationService.allLocations", but if code belongs in a controller, then put it there.