Exam Example for PG5100

This assignment represents a training example for the exam you will do on the 17-19 of October. It is an adaptation of last year PG6100 exam (no REST/WireMock, but added Arquillian and JaCoCo).

The exam assignment (not this training example) will have to be zipped in a zip file with name pg5100_<id>.zip, where you should replace <id> with your own student id, eg pg5100_123456.zip. No “rar”, no “tar.gz”, etc. You need to submit all source codes (eg., .java, .xhtml and .xml), and no compiled code (.class) or IDE files (.idea). Once I got a 175MB rar file as a submission, containing everything, compiled code included… you can guess what grade the student got… an F.

The delivered project should be compilable with Maven 3.x with commands like “mvn package -DskipTests” directly from your unzipped file. Compilation failures will *heavily* reduce your grade.

The assignment is divided in several parts. The parts are incremental, ie building on each other.

Note: during the exam period, I will NOT answer to any email. However, I will answer questions on the Its Learning forum of the course. I will answer questions regarding possible misunderstanding in the exam's instructions, or more general questions like “can we use library X?”. Questions like “How can we do Y?” will be of course left unanswered…

You can (and should when appropriate) reuse code from https://github.com/arcuri82/pg5100_autumn_2016, for example the pom files.

Note: the exam is deliberately **long** and **complex**. This is to distinguish the As from the Bs. I expect only very few of you can complete all of it, if any at all. A good grade (eg C or B) can still be achieved even if few parts are missing.

The source code of the solution (the one used to create the screenshots) will be discussed in class in the last lesson, ie Thursday 6th October.

Easy ways to get a straight F with no appeal:

- have production code (not the test one) that is exploitable by SQL injection
- submit a working solution with no test at all
- submit your delivery as a rar file instead of a zip

The Event List Application

In this exam, you will need to create a basic web application called “Event List” (the actual name does not really matter…). In short, people can post info about events (eg, a concert or a party), and users can specify whether they are going to attend to it or not.

The application should be implemented on the JEE stack, eg JPA, JTA, EJB, and JSF. For this exam, you do NOT need to write any CSS or JavaScript, although you might have to edit some HTML.

Note: when I show screenshots, it is only to clarify the text. You do not have to perfectly 100% match the layout/size of the widgets. Also note that those screenshots do still refer to “pg6100”… I was just too lazy to make new screenshots :)

You should use Maven to compile a WAR file that should be deployable on WildFly 10. You need to configure Maven to be able to start WildFly and automatically deploy the WAR when running “mvn wildfly:run”. You have to use an embedded database (eg H2).

The project should be structured in 3 submodules, in the same way as in the jsf/jacoco module shown in class: “backend” (containing Entity, EJB, and all other needed classes), “frontend” (JSF beans and XHTML) and “report” (for aggregated JaCoCo report).

Testing should be based on Arquillian (for EJB) and Selenium (for end-to-end) using Chrome. All tests should be automatically started (and connecting to WildFly if necessary) when running “mvn clean verify”. Tests should be independent, ie they should not fail based on the order in which they are run.

You need to provide a “readme.txt” file (in same folder as the pom.xml) where you *briefly* discuss your solution: eg, how you structured the code (eg, “@Entity classes are in package X”), and any other important info you want to provide.

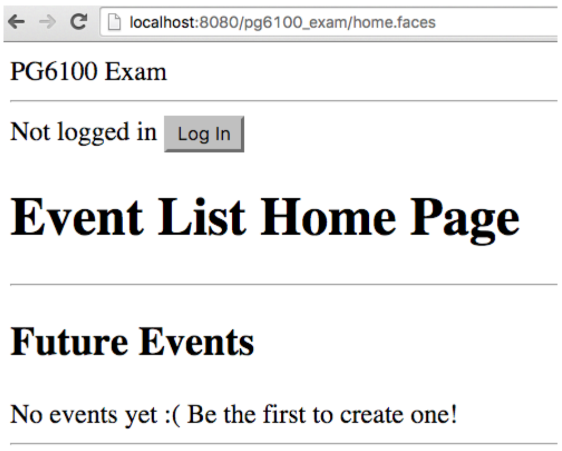
JSF tags you might find useful:

- c:if
- c:otherwise
- c:choose
- c:when
- f:facet
- f:selectItems
- fn:length
- h:form
- h:column
- h:commandButton

- h:dataTable
- h:selectBooleanCheckbox
- h:outputLabel
- h:outputText
- h:inputSecret
- h:inputText
- h:inputTextarea
- h:panelGrid
- h:button
- h:selectOneMenu
- ui:composition
- ui:define
- ui:insert

The Home Page

You need to create a home page in the file home.xhtml. Your application should have root name “exam_example”. When you start with “mvn wildfly:run”, from you browser writing “localhost:8080/exam_example” should lead to the following:



PG6100 Exam

Not logged in

Event List Home Page

Future Events

No events yet :(Be the first to create one!

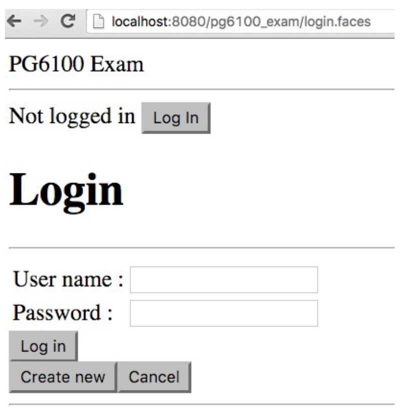
The header “PG5100 Exam Example”, “Not logged it” and the “Log in” should be in a layout.xml template, as it will be reused on all the pages. As the application is just started, no events should be displayed yet. You need a text message to specify it, eg “No events yet”.

Test Home Page

You need to create a WebPageIT file with Selenium tests. For now, write one test called "testHomePage" which simply opens the localhost:8080/exam_example URL and verifies it is the right one (eg, by checking the title of the page).

Login

Need to create a login.xhtml page which is opened when the user clicks on the "Log In" button. Page should look like:



Once username/password are typed, clicking "Log In" should try log in. If login fail, should stay on this page (no need to provide error message). Clicking on "Cancel" should return to home page. "Create new" will be discussed later.

Note: for the actual login, you can re-use the same kind of process (and code) used in the "unsecure" JSF example shown in class. However, recall this is very un-secure. For the sake of this exam, you do not need to worry about setting/updating cookies (ie login/logout of different users will be on the same session).

Test Login: Selenium

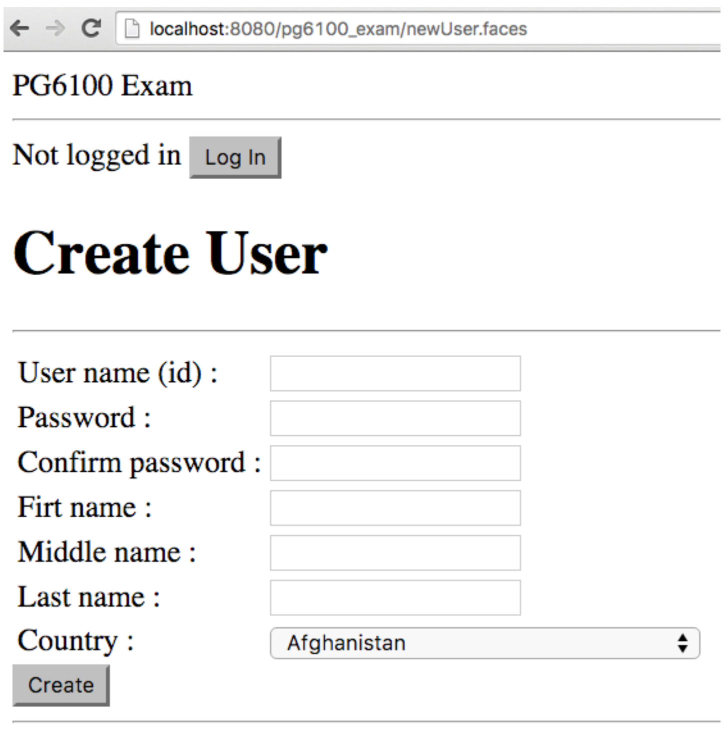
Create page objects called HomePageObject and LoginPageObject, which will be used when interacting with the web pages with Selenium. To WebPageIT, add the following two tests:

- testLoginLink: from home page, click on login, and verify it is on the login page (eg check page title)
- testLoginWrongUser: as above, but here also try to fill a wrong login/password combination, and verify that after pressing "Log in" the browser still stays on the same page

Note: at this point you cannot test a successful login, as you haven't the code yet to register a new user.

Create New User

Create a page newUser.xhtml, which should be reached when clicking "Create new" on the login page. It should look like:


localhost:8080/pg6100_exam/newUser.faces

PG6100 Exam

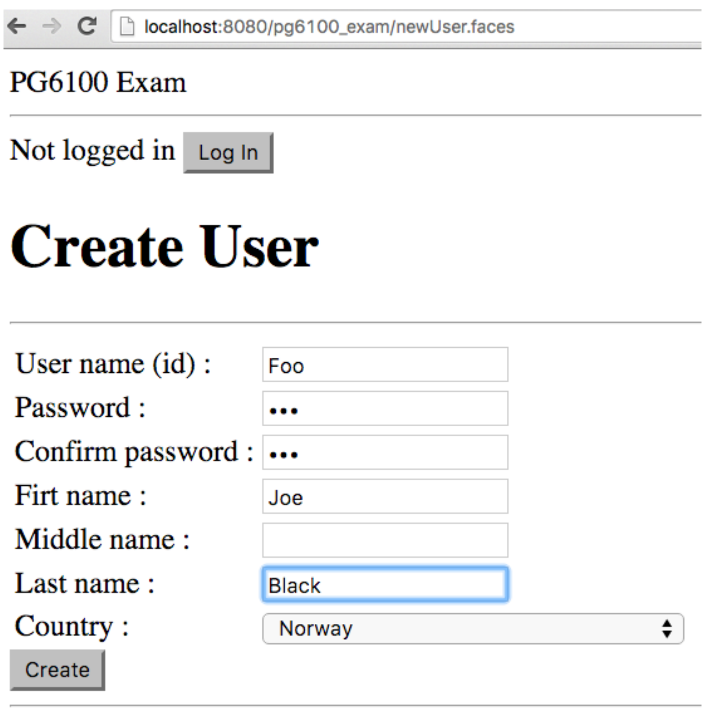
Not logged in

Create User

User name (id) :
Password :
Confirm password :
Firt name :
Middle name :
Last name :
Country : Afghanistan

The values in this form are self-explanatory. The values in the "Country" drop-list should come from the a predefined set of countries (just choose 5-6). On valid inputs, clicking "Create" should create a new user, and its data should be mapped to an @Entity that will be stored on the embedded database. With wrong data (eg., user already exist or password is not properly confirmed), clicking on “Create” should just let the browser stay on same page (no need for error message).

When a new user is created (eg., “Foo”), it will be automatically log in, and the browser should return to the home page. The header should change now, eg displaying a welcome message including the username of the logged-in user, plus the “Log in” should turn into “Log out”:



localhost:8080/pg6100_exam/newUser.faces

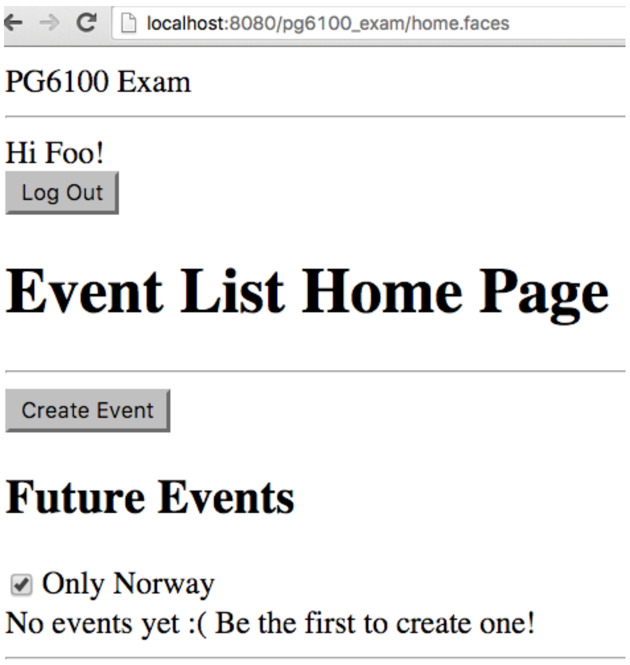
PG6100 Exam

Not logged in [Log In]

Create User

User name (id) :	Foo
Password :	•••
Confirm password :	•••
Firt name :	Joe
Middle name :	
Last name :	Black
Country :	Norway

[Create]



localhost:8080/pg6100_exam/home.faces

PG6100 Exam

Hi Foo!
[Log Out]

Event List Home Page

[Create Event]

Future Events

[✓] Only Norway
No events yet :(Be the first to create one!

You might notice that now on the home page there is also a “Create Event” button and a “Only Norway” checkbox. These should only be visible when a user is logged in, and will be discussed in the next parts.

Constraints on Users

All the fields in the Entity representing a User do need proper constraints (eg, avoid unbound strings). You also need to create two custom constraints: @NotEmpty for strings, and @Country for the field representing a country (which should re-use the pre-defined list of 5-6 countries you made before).

Test Login/User creation: Arquillian

For each public method in the EJB dealing with login and creation of new users, write one Arquillian test. Furthermore, for each field explicitly test if the constraint handling is working.

Test Creation of New Users: Selenium

Add a CreateUserPageObject page object. To WebPageIT, add the following tests:

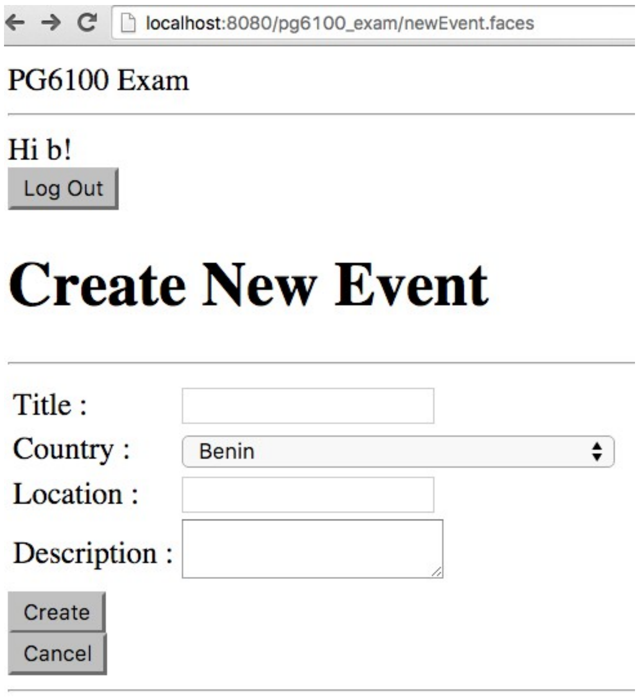
- testCreateUserFailDueToPasswordMismatch: from home, navigate to the create new user page. Try to create a new user, but that should fail due to password not confirmed, ie what in "Password" should not be equal to "ConfirmPassword"
- testCreateValidUser: from home, navigate to the create new user page, and fill the form with valid data. Do successfully create a new user. Once back on home page, do confirm the user is actually logged in (eg, check if id appears in the welcome message and that the login button is gone). Note, to alter the selection in the drop-list, you need to use a command like:

```
new Select(driver.findElement(By.id("createUserForm:country"))).selectByVisibleText(country);
```

- testLogin: create a new user (see previous tests). From the home page, do logout, and confirm you are logged out. Click on "Log In". On the log in page, re-login with the user you just created. Confirm that, once logged in, the browser goes back to home page, and user is indeed logged in.

Creation of New Event

Create a newEvent.xhtml page, accessible from the home page when one clicks on the "Create Event" button. It should look like:


localhost:8080/pg6100_exam/newEvent.faces

PG6100 Exam

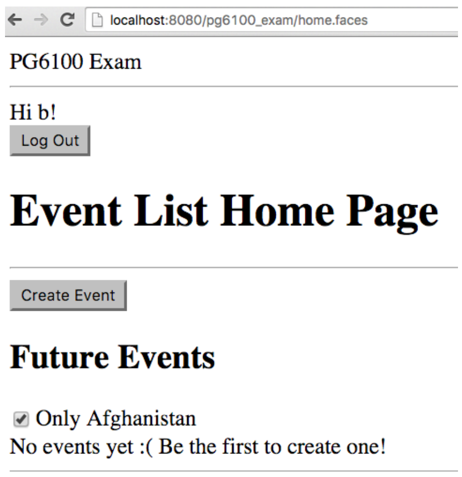
Hi b!

Create New Event

Title :	
Country :	Benin
Location :	
Description :	

Once “Create” is clicked, the data should be stored with an @Entity in the embedded database, and browser should go back to the home page. In case of error, should stay on current page (no need to display error messages).

Note: once at least one event is created, those should be displayed in the home page. However, when creating the very first event from that user, none should be displayed:


localhost:8080/pg6100_exam/home.faces

PG6100 Exam

Hi b!

Log Out

Event List Home Page

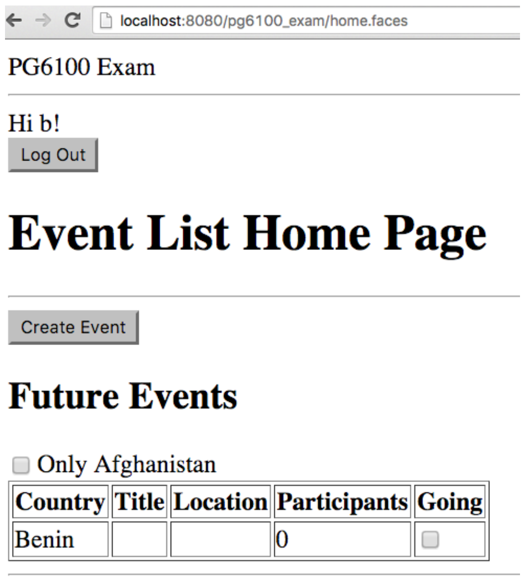
Create Event

Future Events

✓ Only Afghanistan

No events yet :(Be the first to create one!

The point is that only events from the same country as the user (in this case Afghanistan) should be displayed, ie the “Only” checkbox should be on by default. When unchecked, all events from all countries should be displayed:


localhost:8080/pg6100_exam/home.faces

PG6100 Exam

Hi b!

Log Out

Event List Home Page

Create Event

Future Events

☐ Only Afghanistan

Country	Title	Location	Participants	Going
Benin			0	☐

Note: who created an event has no bearing on the display of the events.

Note: for now, ignore the “Participants” and “Going” columns.

New Events: Constraints and Arquillian

Add proper constraints to the Entity representing an event. Add Arquillian tests for each public method in the EJB dealing with events.

Test Creation of New Events: Selenium

Create a page object called CreateEventPageObject. To WebPageIT, add the following tests:

- testCreateOneEvent
    - create and log in a new user (re-use existing page objects)
    - navigate to the create event page, and create a new event in the same country as the username
    - on the home page, do verify that the number of displayed events has been increased by one
- testCreateEventInDifferentCountries
    - create a new user
    - with this user, create two events in two different countries, but one has to be the same as the one of the user. For example, if the user is from Norway, create one event in Norway and one in Sweden (for example).
    - On the home page, when the “Only” checkbox is off, verify the number of displayed events has been increased by 2. When it is on, it should by increased by one. Note: pay particular attention on the fact that, when you run this test, you do not know the kind of events currently displayed.
- testCreateEventsFromDifferenUsers
    - create a new user
    - create an event
    - logout
    - in the same test, create a second new user
    - create an event
    - verify that both the events are displayed on the home page

Event Attendance

For each event, you should keep track of who expressed his/her intention to attend it. Likewise, should keep track for each user of which events s/he will attend. To represent it, you can use a many-to-many relation in the @Entity files.

When an event is displayed, you should also show the number of users that is going to attend it (ie, the "Participants" column in the previous screenshots).

When a user is logged in, for each event there should be a checkbox (ie, the "Going" column in the previous screenshots) to specify whether s/he will attend that particular event. The checkboxes should be off by default, even for the events created by the same user (ie, the author of an event has to explicitly click on its checkbox to express the intention s/he is going to the event s/he just created).

Test Event Attendance: Selenium

To WebPageIT, add the following tests:

- testCreateAndAttendEvent
    - create a new user
    - create a new event
    - verify that the "Participants" count for that event is 0
    - click the checkbox to mark this user is going to attend the event
    - verify that the "Participants" count is 1
    - unclick the checkbox
    - verify that the "Participants" count is 0
- testTwoUsersAttendingSameEvent
    - create a new user
    - create a new event
    - attend the event (ie click on check box)
    - verify that the "Participants" count for that event is 1
    - logout
    - create a second user
    - attend the event with this second user

  - verify that the “Participants” count is now 2
  - logout
  - login back with the first user
  - verify that the “Participants” count is still 2
  - uncheck the attendance checkbox
  - verify that the “Participants” count goes down to 1

JaCoCo

Configure JaCoCo in such a way that, when running “mvn clean verify”, a test report should be generated under the report/target folder. You need an average instruction coverage of AT LEAST 70%. If it is lower, add more tests.

THIS MARKS THE END OF THE EXAM TEXT