Master 2 MIASHS
2019-2020
Jérôme DAVID

TP2 Java EE

JSF Basis

Before starting, you can follow the tutorial:
https://javaee.github.io/tutorial/jsf-facelets003.html#GIPOB

The goal of this work is to develop a module allowing to register users and list them.
It will composed of a class representing, a user (an entity), a service class that will interact with the database (an EJB), a web form (JSF page/Faclelet), and a backing bean (CDI Bean) that will acts as a bridge between the view (the facelet) and the service (the EJB).

Preliminaries

Create a new maven → Web Application project called "Sempic". Choose Group Id "`fr.uga.miashs`".
Add the JSF framework to the project: Right click on the project → Properties → Frameworks → Add… → JavaServer Faces

1- SempicUser entity

Create a package `fr.uga.miashs.sempic.entities`
Inside this package, create a class `SempicUser` with attributes: `firstname`, `lastname`, `email`, `password`. Make this class a Bean i.e. add an empty constructor and getter and setter methods for each attributes. Generate the `equals(...)` and `hashcode()` methods (source→insert code) in order that two instances with the same email are equals. Generate also the `toString()` method.

2- CreateUser backing bean

Create a package `fr.uga.miashs.sempic.backingbeans`
Inside this package, create a class `CreateUser`.
Annotate the class with
`@Named`
`@RequestScoped`

`@Named` means that an instance of this class will be accessible within the application (in facelets or in other beans). By default the name of the instance within facelets is the name of the class with a lowercase letter at the beginning i.e. registerForm. You can assign another name by using `@Named("aPrettyName")`
`@RequestScoped` means that the instances of this class will be stored in the request context, i.e. it is valid only for one request.
Add the attribute `current` of type `SempicUser`, and its associated getter and setter methods.

Add the method ;

```
@PostConstruct
public void init() {
    current=new SempicUser();
}
```

This method will be called to initialize the object just after the call to the constructor.

Add a method `public String create()` that will display the instance `current` on the console, i.e. `System.out.println(current);` This method will be called when the user submit the form that will be created during the next step.

3- The facelet create-user.xhtml

Create a new facelet entitled `create-user.xhtml` (File → New File → Web → JSF Page) in the directory WebPages.

Complete the page in order to have a form asking for the firstname, lastname, email and password of the new user. This form has to be linked to the `CreateUser` backing bean.
To write the page, you can use the HTML 5 markups to handle password, email, placeholders (see https://javaee.github.io/tutorial/jsf-facelets009.html#BABGECCJ).
The commandButton of this form will call the action `create()` of the backing bean.
Run the `create-user.xhtml` page (right click on it → run)
Register a new user and observe the result in the GlassFish console.

4- The model

After having created the view (the facelet) and the controller (the managed beans), we will implement the application logic, i.e. the model. To start, we will create only a basic store that save the registered users in main memory (in a map).

Create a package `fr.uga.miashs.sempic.dao`..
Inside this package, create a class `SempicUserMap`. This class has be a singleton EJB: Annotate the class with `@Singleton` (from class `javax.ejb.Singleton` and not `javax.inject.Singleton`). This means that only one instance of this class will be created by the application and it will stay in memory until the server is shutdown.
This class will have an attribute called `users` of type (`Map<String,SempicUser>`) that maps the email addresses to the instances of `SempicUser`.
This attribute will be initialized in he following method:

```
@PostConstruct
public void init() {
    users = new HashMap<>();
}
```

Add the methods `void create(SempicUser u) throws SempicException` and `List<SempicUser> findAll()`. The method `create` adds a user in the map. It will throw an Exception if a user with the same email already exists in the map (You have to create the class `SempicException`, subclass of `Exception`). The method `findAll` returns all the users contained in the map (`new ArrayList<>(users.values())`).

Now, the link between the service and the backing bean has to be done.

In the class `CreateUser`, add the attribute `userDao` of type `SempicUserMap` annotated with `@EJB`.

Modify the method `create()` of `CreateUser` in order to add the current instance to the store, i.e. call the `userDao.create(...)` method of `SempicUserMap`. Add an error message using the following line of code (where `e` is the instance of the Exception):

```
FacesContext.getCurrentInstance().addMessage(null,
                                    new FacesMessage(e.getMessage()));)
```

5- Navigation between views

The Facelet list-users.xhtml and its backing bean

Create a backing bean `ListUsers` with attributes :

```
private DataModel<T> dataModel;
@EJB
private SempicUserMap userDao;
```

add the following method :

```
public DataModel<T> getDataModel() {
    if (dataModel == null) {
        dataModel = new ListDataModel<>(userDao.findAll());
    }
    return dataModel;
}
```

Create a new facelet `list-users.xtml`
Using the component `h:dataTable`, create a page that display the list of registered users (given by the ListUsers backing bean). See Section “Using Data-Bound Table Components” of https://javaee.github.io/tutorial/jsf-page002.html#BNARZ

Test your facelet and see the list of users.

Navigation rules

Now we want to add navigation between theses pages.
To navigate from list-users to create-user, we just have to add the following link to end of list-user:

```
<h:link  value="Add a new user link" outcome="create-user"/>
```

This is called implicit navigation in JSF: the outcome attribute is the view id (i.e. the name) of the targeted view.

From the page create-user, it is a bit more complex. The redirection to the next view depends on the result of the action create(). If it succeed, we want to see the list of users, if not we want to stay on the create-user view. This is an example of rule-based navigation.
To implements this, modify the method create() of CreateUser in order it returns the string SUCCESS (if the user has been added) or FAILURE if the user already exists.
Now open the file WEB-INF/faces-config.xml (if it does not exists, add it: File → New File → JavaServer Faces → JSF Faces Configuration).
Now add the following lines:

```
<navigation-rule>
    <from-view-id>/create-user.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{createUser.create()}</from-action>
        <from-outcome>SUCCESS</from-outcome>
        <to-view-id>/list-users.xhtml</to-view-id>
    </navigation-case>
```

```
    <navigation-case>
        <from-action>#{createUser.create()}</from-action>
        <from-outcome>FAILURE</from-outcome>
        <to-view-id>/create-user.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

Test the navigation between pages.
For more details see https://javaee.github.io/tutorial/jsf-intro006.html#BNAQL

6- Constraints

Until now, you can enter what ever you want in the form and the user will be registered, or there will be an error.
JavaEE platform allows to add constraints in a declarative way at the entity level. These constraints are verified at the server side. Contrary to client side validation, it will required a request to be send but it is not possible to pass-by it by hacking javascript.

For instance add the annotation `@Email` (import `javax.validation.constraints.Email`) to the field email of `SempicUser`. Then test again your registration form and verify that now you are only allowed to put an email (if you have use an inputText element with `p:type="email"`, then replace it with `p:type="text"` in order to deactivate client side validation).

You can also add the `@NotBlank` annotation to all the fields of `SempicUser`. This annotation forces the field to be not null and have at least a non-white space character.
To get more information about bean validation constraints, see:
https://javaee.github.io/tutorial/bean-validation002.html#GIRCZ