Scala Play Framework Tutorial

We are going to develop a simple application using Scala and Play Framework 2.4.

Type in the terminal:

```
activator new
```

Write 6 (play-scala)

and write a name for the application: playScalaTutorial.

First, we are going to build the model. A user will have two attributes: username and password. Both are Strings. Create a new package inside app called models, and inside this package create a case class called User.

```
package models
case class User(username: String, password: String)
```

Now, we are going to write a companion object which will act as a database.

```
object User {
  var users = Set(
    User("Josh", "Josh"),
    User("Dave", "Dave")
  )
}
```

The variable users will act as a database, everytime we want to look for a user, we will use it.

Now, we can start using this model.

List of users

We are going to create a page in which a list of all users will be displayed.

We don't have a method to retrieve all users, so it will be the first thing to do.

In the companion object User, write the following method:

```
def findAll = users.toList
```

Now, in the controller, we need to create a function that takes all the users in the variable users and gives it to the view. Write in controllers.Application the following:

```
def list = Action { implicit request =>
```

```
  val users = User.findAll
  Ok(views.html.users.list(users)) //users is the argument which will be passed to the 
view
}
```

Don't forget to import:

```
import models.User
```

Now, we only have to write the view!

Create the package users inside views and create list.scala.html inside it.

.scala.html is a html file with Scala code embedded. This HTML file is the following:

```
@(users : List[User]) //Argument. This view has one argument, a list of users.
@main("List of users") { //main is a template created by play.
     <ol>
          @for(user <- users){ //Iterate through the list
               <li>@user.username</li> //display the name of the user in a <li>
          }
     </ol>
}
```

The main template has two arguments: a string (title of the page) and a block of code. You can see its code in views.main.scala.html

All Scala code inside a html file starts with “@”.

The last thing to do is to write this feature in the routes file. Go to conf/routes and type:

```
GET     /users                    controllers.Application.list
```

Go to the terminal and then go to the directory of the app.

Then type:

```
activator run
```

http://localhost:9000/users

You should be able to read the contents of the variable users in the companion object User.

Sign up

Implementing signing up in the model

Signing up is just adding a new User in the variable users defined in the companion User object. We can do that with this method:

```scala
def signUp(user: User): Unit = {
  users = users + user
}
```

View

I am going to write the sign up page in the index page (host:9000/)

```
@(userForm: Form[User])(implicit messages: Messages)
@import helper._
@main("Sign up") {
    @form(action=routes.Application.signUp()) {
        @inputText(userForm("username"))
        @inputText(userForm("password"))
        <input type="submit" value="Submit">
    }
}
```

userForm is the form to sign up. Yes, that's right! We send the form as a parameter. Why? It's play's way of communicating the view with the controller. Messages is something we need because we are using @form which is defined in the package helper._ Just to make things easy, it is used for internationalization, but we are not going to learn that for now.

@form(action=…) it is self-explained, isn't it? When we send the data to the server pressing the submit button, we called the function signUp() which is defined in the Application controller. It must be defined in the routes file, we'll do that soon.

@inputText(param) the parameter must be a field of the userForm defined in the controller. We'll see that soon.

Routes

Just add this:

```
POST    /                       controllers.Application.signUp
```

Controller

This is not going to be as easy as before, so let's do it step by step.

First, we are going to define the form we'll send to the view.

```scala
private val userForm: Form[User] = Form(
  mapping(
    "username" -> nonEmptyText, //fields of the form.
    "password" -> nonEmptyText //nonEmptyText forces the field to not to be empty
  )(User.apply)(User.unapply)
)
```

and import the following:

```scala
import play.api.data.Form
import play.api.data.Forms.{mapping, nonEmptyText}
```

User.apply is a function already defined by Scala as User is a case class. The same happens with unapply. If user wasn't a case class, you would have to defined them.

Now, we can create the function for signing up.

```
def signUp = Action { implicit request =>
  val newUser = userForm.bindFromRequest()
  newUser.fold(
    hasErrors = { form =>
      Redirect(routes.Application.index())
    },
    success = { user =>
      User.signUp(user)
      Redirect(routes.Application.list())
    }
  )
}
```

newUser has the data introduced by the user in the form. NewUser can have correct and incorrect data. If it has errors, then we redirect to index, otherwise, we call to the signUp function of the companion User object and we redirect to the page which shows all users.

We haven't finished yet! As we are using in the view @form we had to use an implicit parameter for internationalization. In the controller, we must modify the header of the controller.

```
class Application @Inject()(val messagesApi: MessagesApi) extends Controller with
I18nSupport {
```

And do not forget to import the libraries:

```
import play.api.i18n.{I18nSupport, MessagesApi}
```

We've changed the index view, so we must change the function index in the Applicaiton controller.

```
def index = Action {
  Ok(views.html.index(userForm))
}
```

We're sending now its parameter.

Now, we have finished!

Login

View

Write the login.scala.html file in views.users.

```
@(userForm: Form[User])(implicit messages: Messages)
@import helper._
@main("Welcome to Play") {
    @form(action=routes.Application.login()) {
        @inputText(userForm("username"))
        @inputText(userForm("password"))
        <input type="submit" value="Submit">
    }
}
```

In this view we have a form as in the sign up view but the action of the form is different.

Let's define that action in the routes file.

GET /login controllers.Application.loginPage #When we enter in the browser /login, the function loginPage in the Application controller is executed. This function renders the page.
POST /login controllers.Application.login #When we press the submit button, the function login in the Application controller is executed. This function let the user login.

Controller

Let's write the functions we talked about before.

```
def loginPage = Action {
  Ok(views.html.users.login(userForm))
}
```

```
def login = Action { implicit request =>

  val newUser = userForm.bindFromRequest()
  newUser.fold(
    hasErrors = { form =>
      Redirect(routes.Application.loginPage())
    },
    success = { user =>
      if(User.login(user)){
        Redirect(routes.Application.show(user.username))
      } else {
        Redirect(routes.Application.loginPage())
      }
    }
  )
}
```

In the login function, we are using two undefined funcitons: `User.login(user)` and `routes.Application.show(user.username).`

Model

Let's define the login(user: User) function in the companion object.

```
def login(user: User): Boolean = {
  try{
    findByUsername(user.username).get.password == user.password
  } catch {
    case e: Exception => false
  }
}
```

Now, let's define `findByUsername(username: String)`

```
def findByUsername(u: String) = users.find(_.username == u)
```

View

If the user logins successfully, he must be redirected to the view show that we are going to implement right now.

In views.users create the file user.scala.html

```
@(user: User)
@main(user.username){
    Welcome @user.username
}
```

In the application controller, write this function:

```
def show(username: String) = Action { implicit request =>
  User.findByUsername(username).map { user =>
    Ok(views.html.users.user(user))
  }.getOrElse(NotFound)
}
```

And finally, in the routes file add this line:

```
GET     /users/:user              controllers.Application.show(user: String)
```

Hey! There's something different! That's right, we're adding a parameter. This parameter will appear in the URL.

That's all. I hope you've learned a little bit about Play Framework :)