Play 02

# Views - Dust Templates

Let’s create a simple Dust template in demo1-frontend/public/templates/hellodust.tl:

Hello {name}!

name is a template variable. We will supply a model which will provide a value for this variable. To do that, we create a new controller using the LinkedIn Dust plugin. In demo1-frontend/app/controllers/ProfileStrengthController.scala add:

def helloDust = Action { implicit request =>

val basePage = DustFizzyPlugin.defaultBasePage

val data = DefaultBasePageData(

clientTemplate = "templates/hellodust",

clientData = Map("name" -> "eric"))

val options = DustOptionsScala()

DustFizzyPlugin.getInstance.result(basePage, data, options)

}

Here basePage is the page that will be loaded in the browser which contains the dust template and the model data. The defaultBasePage is the page that’s used during development for prototyping. data packages the template and the model data, which is represented as a Map. options is for providing additional information to Dust. For now we leave it empty. Finally, we return a dust result, passing in the basePage, the data (containing the template and the model), and the options.

Add a route to conf/routes

# @noPageKeyNeeded

GET /hellodust controllers.ProfileStrengthController.helloDust

In your browser, visit <http://localhost:9000/demo1/hellodust>

View the source of the page. You’ll see a lot of Javascript. The dust template is compiled into Javascript, the model is serialized into JSON, and the template is rendered on the client using the dust javascript.

# Add an image

Download the image from <http://goo.gl/rAcP7v> and move it into demo1-frontend/public/images (you can drag it directly into the folder in the IDE).

Modify the template to include the image. At the top of hellodust.tl, add

{@img path="lilogo.png"/}

<p>

Refresh <http://localhost:9000/demo1/hellodust>. You should see an image.

# Generate a Link

In the ProfileStrengthController.helloDust action, modify DustOptions to the following:

val javascriptRoutes: Seq[JavascriptReverseRoute] = Seq(routes.javascript.ProfileStrengthController.index)

val options = DustOptionsScala(javascriptRoutes = javascriptRoutes)

ProfileStrengthController.index is the controller that sends back “it works” to the browser. The routes.javascript expression is a reverse route to that index, allowing the javascript to generate a URL that corresponds to the controller. Play is able to do this by consulting the routing table defined in the conf/routes file.

To generate a link, add the following to hellodust.tl:

<a href="{@url alias="controllers.ProfileStrengthController.index"/}">A specialmessage is here</a>

Refresh <http://localhost:9000/demo1/hellodust>. You should see a link at the bottom. Click the link.

# Create a Form

Let’s create a form which accepts input for a memberId and fetches the profile strength for that member. Create a new template in demo1-frontend/public/templates/profilestrengthform.tl:

{@img path="lilogo.png"/}

<p>

{@form alias="controllers.ProfileStrengthController.processForm"}

<fieldset>

<label for="memberid">Input the member's ID: </label><input name="memberid" type="text">

</fieldset>

{/form}

The controllers.ProfileStrengthController.processForm reverse route is the endpoint data will be sent to when the form is submitted. In ProfileStrengthController, create processForm. For now we just return a simple string. Later we’ll process actual data.

def processForm = Action { implicit request =>

Ok("submitted!")

}

Create a route to the action in routes:

# @noPageKeyNeeded

POST /processProfileStrengthForm controllers.ProfileStrengthController.processForm

Now let’s create a method method renderForm which sets up the dust template and renders the form.

def renderForm(templateData: Map[String, Any])(implicit request: RequestHeader) = {

val basePage = DustFizzyPlugin.defaultBasePage

val data = DefaultBasePageData(

clientTemplate = "templates/profilestrengthform",

clientData = templateData)

val javascriptRoutes: Seq[JavascriptReverseRoute] =

Seq(routes.javascript.ProfileStrengthController.processForm)

val options = DustOptionsScala(javascriptRoutes = javascriptRoutes)

DustFizzyPlugin.getInstance.result(basePage, data, options)

}

Let’s create an action which will do the initial rendering of the form:

def showForm = Action { implicit request =>

renderForm(Map())

}

Create a route to showForm in conf/routes:

# @noPageKeyNeeded

GET /showprofilestrengthform controllers.ProfileStrengthController.showForm

Hit the page: <http://localhost:9000/demo1/showprofilestrengthform>

Enter 3 and hit return.

# Process Form Data

When the form is submitted, we want the processForm action to process the data, call the rest.li service, and return the result.

We’re going to refactor our restliProfileStrengthForMember action to extract the code which fetches the profileStrength.

Modify it to:

def fetchProfileStrength(memberId: Long)(implicit request: RequestHeader) = {

val memberUrnString = s"urn:li:member:$memberId"

val restliRequest = new ProfileStrengthsBuilders().get().id(new Urn(memberUrnString)).build()

val futureResponse = RestliPlugin.getInstance.sendRequest(restliRequest)

val profileStrengthInfo = Await.result(futureResponse, 5 seconds).getEntity()

profileStrengthInfo.getScore()

}

def restliProfileStrengthForMember(memberId: Long)= Action { implicit request =>

val score = fetchProfileStrength(memberId)

val profileStrength = new ProfileStrength().setScore(score)

Ok(JsonUtil.toJsValue(profileStrength))

}

Revisit <http://localhost:9000/demo1/profilestrength/30> to ensure it works still:

Now let’s hardcode the memberId to ensure it works. Modify processForm to

def processForm = Action { implicit request =>

val score = fetchProfileStrength(30)

Ok(s"submitted. The score for member 30 is $score")

}

Hit the page: <http://localhost:9000/demo1/showprofilestrengthform>

Enter 3 and hit return.

Now, we need a data structure to capture the form data. Create a case class in ProfileStrengthController:

case class Member(memberid: Long)

Now create a Play Form which specifies how to map the incoming HTTP request to the Form data structure.

import play.api.data.Forms.\_

val memberForm = Form[Member] {

mapping(

"memberid" -> longNumber

)(Member.apply)(Member.unapply)

}

Now modify processForm to process the data:

def processForm = Action { implicit request =>

val member = memberForm.bindFromRequest.get

val memberId = member.memberid

val score = fetchProfileStrength(memberId)

Ok(s"submitted. The score for member $memberId is $score")

}

Hit the page: <http://localhost:9000/demo1/showprofilestrengthform>

Enter 3 and hit return.

# Render Form after Submit

We’d like to re-display the form after submission, so we can try again with new members. To do that, just call renderForm on the last line of processForm:

def processForm = Action { implicit request =>

val member = memberForm.bindFromRequest.get

val memberId = member.memberid

val score = fetchProfileStrength(memberId)

renderForm(Map("mid" -> member.memberid, "score" -> score))

}

and modify the profilestrengthform.tl template to display the model data:

{@img path="lilogo.png"/}

<p>

{#score}

Score for member {mid} is {score}.

{/score}

{@form alias="controllers.ProfileStrengthController.processForm"}

<fieldset>

<label for="memberid">Input the member's ID: </label><input name="memberid" type="text">

</fieldset>

{/form}

Hit the page: <http://localhost:9000/demo1/showprofilestrengthform>

Enter 3 and hit return.

Enter 30 and hit return.

# Play Console

Open up a terminal. Cd to ~/projects/demo1 then enter

play console

When the console starts, enter

import controllers.ProfileStrengthController.\_

Member(34)

You have access to all the code you wrote. Have fun playing with it.