Dharmsinh Desai University, Nadiad
Faculty of technology, Department of Computer Engineering
Subject : Software Project
Lab Manual

Lab 8 : Developing authentication module for the project using Django's inbuilt authentication features and add .css file.

Aim : Create login module for the project

Requirements : Python, Django,

Prerequisit : Understanding of django framework , working with django projects and apps, creating views, urls, models and templates using django.

Step 1 : Setup the project

1.1 Create the app,

```
$python3.6 manage.py startapp loginmodule
```

1.2
Go to settings.py and check the following,

--> Base directory field should be as follows,

```
BASE_DIR=os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

--> Install app should show admin and auth. (uncomment it if it is commented)

```
INSTALLED_APPS = [
          'django.contrib.admin',
          'django.contrib.auth',
          'django.contrib.contenttypes',
          'django.contrib.sessions',
          'django.contrib.messages',
          'django.contrib.staticfiles',
          'loginmodule',
]
```

--> In template check path for context_processors.auth (this will be imported in views.py). Also check 'DIRS' field (If it is different from what is shown below, dont change it for now. You may later modify it if you observe template location related error)

```
TEMPLATES = [
            {
             'BACKEND':'django.template.backends.django.DjangoTemplates',
                'DIRS': [os.path.join(BASE_DIR, 'templates')],
                'APP_DIRS': True,
                'OPTIONS': {
                    'context_processors': [
                        'django.template.context_processors.debug',
                        'django.template.context_processors.request',
                        'django.contrib.auth.context_processors.auth',
                       'django.contrib.messages.context_processors.messages',
                    ],
                },
            },
        ]
```

1.3 Go to app folder and create app's *urls.py* and *templates* directory.

```
loginmodule$mkdir templates
loginmodule$touch urls.py //This creates empty file.
```

include app's ***urls.py*** in projects ***urls.py***. Go to project directory and open ***urls.py***. And update it as shown below.

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls import url
urlpatterns = [
     url(r'^admin/', admin.site.urls),
     path('loginmodule/', include('loginmodule.urls')),
]
```

Step 2 : Prepare specifications

Prepare requirement specification of login functionality. Following is a brief example of requirements for login module

R1 : System provides facility to authentic user.

R1.1 System allows user to enter username and password for authentication.

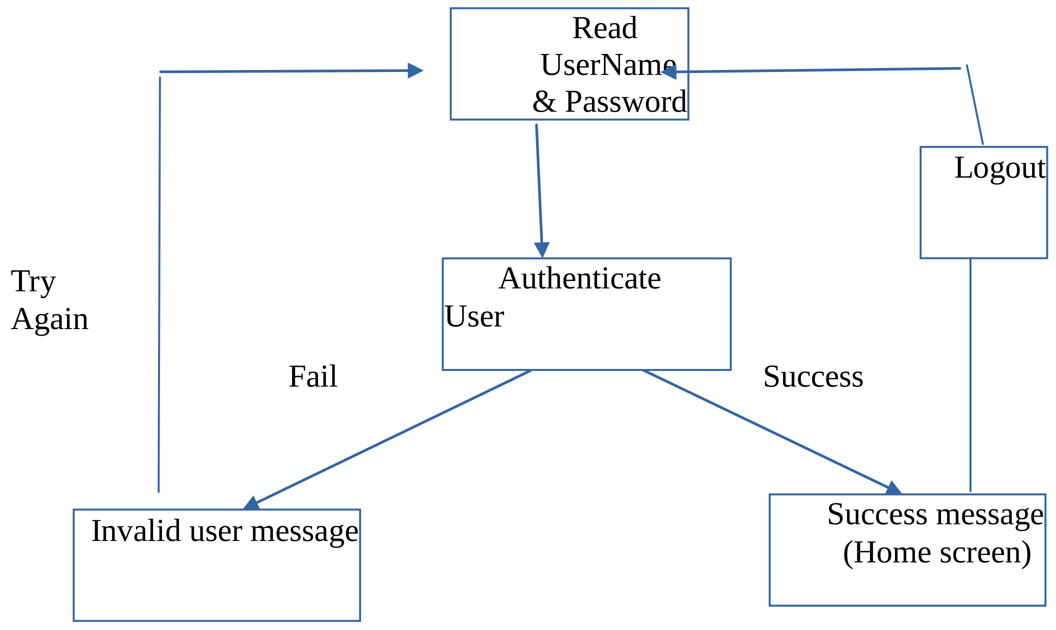
R1.2 System authenticates the user and generates success or failure message.

R1.3 System shows success message and a link to logout if user credentials matches with the database.

R1.4 For unsuccessful authentication, Ssstem shows failure message and link to try again.

R1.5 System uses django's internal module for authentication. (non functional?)

Following flowchart describes the overall working of the system.



To implement this functionality, we will need following views and templates. Recall from our discussion of MVT architecture, we usually break every functionality into three parts, Model, View and Templates. (In this example, we are using Django's inbuilt user model, so we do not need to define models.) Role of views is to perform necessary logical operations and send and/or receive data to template.

(1) Read UserName & Password : login view and login.html template
(2) Authenticate user : Authentication view
(3) Success message (home screen) : loggedin view and loggedin.html template
(4) Invalid user message: invalidlogin view and invalidlogin.html template
(5) logout ; logout view and logout.html template.

Step 3 : Create views and update urls.py

Now let us first create views in (app's ***views.py***) as described in the specification (step 2)

We need to import the following,
- (i) 'HttpResponseRedirect' : to read and right from html,
- (ii) 'render_to_response' : to render the resulting web page.
- (iii) 'auth' : Django's authentication module to authenticate user.
- (iv) 'csrf' : a middleware template which protects from 'cross site request forgery' (we don't need to worry about this attach much. However, curious souls may visit, https://docs.djangoproject.com/en/2.0/ref/csrf/)
- (v) apart from above 4, lets also add usual TemplateView.

```
from django.shortcuts import render_to_response
```

```
        from django.views.generic import TemplateView
        from django.http import HttpResponseRedirect
        from django.contrib import auth
        from django.template.context_processors import csrf
```

(look at the last import csrf. The context_processors path should be same as what we saw in project's settings.py (step 1). Modify the path (here, in views) if it is not the same.)

Now let's start defining views,

(1) login view : It creates a csrf token and calls login.html (defined in templates)

```
                def login(request):
                     c = {}
                     c.update(csrf(request))
                     return render_to_response('login.html', c)
```

(2) auth_view : This is the core part where we are reading data from login.html and comparing it with database using django's auth module. On successful authentication we call 'loggedin' view and on failure we call 'invalidlogin'

```
     def auth_view(request):
          username = request.POST.get('username', '')
          password = request.POST.get('password', '')
          user = auth.authenticate(username=username,
password=password)

          if user is not None:
               auth.login(request, user)
               return HttpResponseRedirect('/loginmodule/loggedin/')
          else:
               return
HttpResponseRedirect('/loginmodule/invalidlogin/')
```

(3) loggedin view: This view simply renders loggedin.html.

```
     def loggedin(request):
          return render_to_response('loggedin.html', {"full_name":
request.user.username})
```

(4) invalidlogin view : This view renders invalidlogin.html

```
     def invalidlogin(request):
          return render_to_response('invalidlogin.html')
```

(5) loggedout view : This view renders loggedout.html

```
     def logout(request):
          auth.logout(request)
          return render_to_response('logout.html')
```

Now, update app's urls.py with views informations.

```
     from django.urls import path
     from loginmodule.views import login, auth_view, logout,
loggedin, invalidlogin
     from django.contrib.auth import views as auth_views
     from django.conf.urls import url

     urlpatterns = [
          url(r'^login/$', login),
          url(r'^auth/$', auth_view),
          url(r'^logout/$', logout),
          url(r'^loggedin/$', loggedin),
          url(r'^invalidlogin/$', invalidlogin),
     ]
```

Step 4 : go to applications templates directory and create html templates.

(i) base template : base template are place holders where other htmls can put content. Let's first create a base.html as follows,

```
     <!DOCTYPE html>
     <html>
         <head>
             <meta charset="utf-8">
             <title>MyLogin!</title>
         </head>
         <body>
             <h1>Django Login test!</h1>

          {% block content %}
          {% endblock %}
         </body>
     </html>
```

Here, {% block content %}   {% endblock %} is the part where other htmls can put their content. Apart from that base.html is not doing much here. You may beutify the base html further and glorify your project. :)

(2) login.html : Write login.html as follows,

```
{% extends 'base.html' %}

   {% block content %}
    {% if form.errors %}
          <p class="error"> Invalid Username or Password </p>
    {% endif %}

    <form action="/loginmodule/auth/" method = "post">{% csrf_token %}
          <label for="username">user name : </label>
          <input type="text" name = "username" value = "" id="username">
          <label for="password"> Password:</label>
          <input type="password" name="password" value="" id="password">
          <input type="submit" value="login"/>
    {% endblock %}
```

Notice that, the login.html extends base.html and puts content between {%block content%} and {% endblock %}. Apart from that the html has two text fields to enter user name and password and a submit button.

(3) invalidlogin.html :

```
{% extends 'base.html' %}
  {% block content %}
     <h2> Invalid user name or password! </h2>
     <p> Click <a href="/loginmodule/login/"> here </a> to login
           again. </p>
  {% endblock %}
```

(4) logout.html:

```
{%extends "base.html" %}
{% block content %}
     <h2> You are logged out! </h2>
     <p> Click <a href="/loginmodule/login/"> here </a> to login
          again. </p>
{% endblock %}
```

(5) loggedin.html:

```
{%extends "base.html" %}
{% block content %}
     <h2> Welcome {{fullname}} you are logged in! </h2>
     <p> Click <a href="/loginmodule/logout/"</a> to logout </p>
{% endblock %}
```

Step 5 : Testing the app

(1) Run usual makemigration and migrate commands from project's root directory. (where manage.py is located)

```
$ python3.6 manage.py makemigrations
$ python3.6 manage.py migrate
```

(2) Before running the server, let's create some user data in django's user database. Start django shell as follows,

```
      $python manage.py shell
      >>> from django.contrib.auth.models import User
      >>> user = User(username = 'test', email = 'test@check.com',
password='test123')
      >>> user.full_name = "teeessst"
      >>> user.save()
```

(3) Now run server: **`$python manage.py runserver`**

(4) go to ***127.0.0.1:8000/loginmodule/login*** and test the app.

- **Add .css File to application**

First, create a directory called static in your loginmodule directory. Django will look for static files there, similarly to how Django finds templates inside loginmodule/templates.

Django's STATICFILES_FINDERS setting contains a list of finders that know how to discover static files from various sources. One of the defaults is AppDirectoriesFinder which looks for a "static" subdirectory in each of the INSTALLED_APPS, like the one in loginmodule we just created. The admin site uses the same directory structure for its static files.

Within the static directory you have just created, create another directory called loginmodule and within that create a file called style.css. In other words, your stylesheet should be at loginmodule/static/loginmodule/style.css. Because of how the AppDirectoriesFinder staticfile finder works, you can refer to this static file in Django simply as loginmodule/style.css, similar to how you reference the path for templates.

In style.css put following code :

Style.css

```
h1
{
        color:purple;
}

h2
{
        color:white;
}

p
{
        color:yellow;
}
```

and modify **base.html** as below:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset = "utf-8" >
		<title>MyLogin!</title>
	</head>
	<body>
	{% load static %}
	<link rel="stylesheet" type="text/css" href="{% static 'loginmodule/style.css'    %}" />
		<h1 align="center">Django Login test !</h1>

		{%block content %}
		{%endblock %}
	</body>
</html>
```

To add background image, create a subdirectory for images(named images) in **loginmodule/static/loginmodule** and put a background-image in that directory. Then add following code in **style.css** file.

```
	body {
	   background: white url("images/background.gif") no-repeat;
	}
```