INGENIERÍA | DEPARTAMENTO
UNIVERSIDAD DE LOS ANDES | DE COMPUTACIÓN
MÉRIDA VENEZUELA | ESCUELA DE SISTEMAS

Laboratorio de Ingeniería del Software
Práctica Nro. 5
Frameworks

Prof. Solazver Solé
Prep. Yenifer Ramírez
Semestre B-2018

1. Framework

Es un conjunto de archivos y directorios, que incorporan funcionalidades ya desarrolladas y probadas que se implementa en un lenguaje de programación. Su objetivo principal es facilitar el desarrollo de aplicaciones, haciendo que nos centremos en las funcionalidades del usuario, esto gracias a que poseen herramientas de generación de código que evitan las tareas repetitivas para el desarrollador, además de estar constituida de una arquitectura y patrones de diseño ya probados.

1.1. Ventajas

- Uso de patrones de diseño para el desarrollo de la aplicación.
- Casi todos los frameworks utilizan el patrón arquitectónico MVC, el cual es un modelo que se divide en 3 capas:
  - **Modelo:** representa los datos de la aplicación y las reglas de negocio.
  - **Vista:** representa la capa de presentación, lo que se presenta al usuario.
  - **Controlador:** es el encargado de procesar las peticiones de los usuarios y controla el flujo de ejecución del sistema.
- El modelo MVC puede ser implementado sin la necesidad de usar un framework, pero la diferencia radica en que el framework nos obliga a utilizarlo.

- Viene con una estructura predefinida de la aplicación, lo que implica que el programador no necesita plantearse la estructura global de la aplicación.
- Todo el código que es parte del framework es testeado.
- Generalmente, según la antiguedad del framework, este cuente con un gran comunidad de usuarios, que ayudan uda para el desarrollo.

1.2. Características

- **Abstracción de URLs y sesiones:** no es necesario manejar directamente las URLs ni las sesiones, ya que el framework se encarga de hacerlo.
- **Acceso a datos:** incluyen herramientas e interfaces necesarias para comunicarse con bases de datos, independientemente del tipo que estemos utilizando.
- **Uso de controladores:** suelen implementar una serie de controladores para la gestión de los eventos y peticiones realizadas a la aplicación.
- **Autentificación y control de acceso:** incluyen mecanismos para la identificación de usuarios mediante el uso de un usuario y contraseña.
- **Internalización:** son mecanismos para poder mostrar la aplicación en todos aquellos idiomas que consideremos oportunos.

1.3. ¿Cómo elegir un framework?

- Conocimientos del equipo.
- Soporte y documentación.
- Existencia de proyectos desarrollados en él.
- Soporte del patrón arquitectónico MVC.
- Seguridad del framework.

1.4. Ejemplos de Frameworks Web

- **Ruby on Rails:** framework MVC basado en Ruby orientado al desarrollo de aplicaciones web.
- **CodeIgniter:** framework basado en PHP liviano y rápido.
- **Django:** framework para Python capaz de crear diseños muy limpios.
- **Zend Framework:** es un framework de código abierto en PHP para desarrollar aplicaciones web y servicios web con PHP 5.

- **Symfony:** completo framework en PHP diseñado para optimizar el desarrollo de las aplicaciones web basado en Modelo-Vista-Controlador.
- **Yii:** framework en PHP basado en componentes.
- **Struts:** herramienta de soporte para el desarrollo de aplicaciones Web bajo el MVC y la plataforma Java EE (Java Enterprise Edition).

2. Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–template. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt.
La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself).

2.1. Instalación

Es recomendable que Django se instale dentro de un entorno virtual de GNU-Linux, por lo que se debe proceder con los siguiente comandos para instalar los paquetes necesarios:

```
root@pc:/# apt-get update
```

```
root@pc:/# apt-get install python3-pip
```

```
root@pc:/# pip3 install virtualenv
```

Para iniciar un nuevo proyecto creamos y nos situamos en un nuevo directorio en la ruta deseada:

```
root@pc:/# mkdir proyecto && cd proyecto
```

Ahora podemos crear dentro del directorio del proyecto el entorno virtual, y procederemos a activarlo:

```
usuario@pc:~$ virtualenv -p python3 eV
```

```
usuario@pc:~$ cd eV
```

```
usuario@pc:~$ source bin/activate
```

E instalamos el Framework dentro del entorno:

```
(eV)usuario@pc:~$ pip install django
```

Nota: también es posible instalar Django en todo el sistema. Utilizando los paquetes que se encuentran en los repositorios de nuestro Sistema Operativo.

```
root@pc:/# apt-get install python python3 python-django
```

2.2. Creación de Proyectos

Para ejecutar los comandos de django usaremos el ejecutable *django-admin*, o el script *django-admin.py* que viene con django (luego de la instalacion de Django estos deben estar accesibles en el directorio */usr/bin,/usr/local/bin*) y si usamos el ambiente virtual se encuentra en */bin* del mismo).
Para crear un nuevo proyecto ejecutamos en nuestro terminal:

```
(eV)usuario@pc:~$ django-admin startproject mi_proyecto
```

Luego de ejecutado el comando se generan los archivos bases iniciales de nuestra aplicación.

2.2.1. Estructura inicial de un proyecto

En el directorio base:

- **manage.py:** realiza diferentes tareas especificas del proyecto como ejecutar el servidor, ejecutar las migraciones entre otros.

  En el directorio del proyecto:

- **___init___:** permite a los paquetes de python ser importados en los directorios donde se encuentre. Es un archivo generico usado en casi todas las aplicaciones de Python.
- **settings.py:** contiene los archivos de configuración para el proyecto en Django.
- **ulrs.py:** contiene los patrones URL para el proyecto Django.
- **wsgi.py:** contiene configuración de propiedades WSGI[1] para el proyecto en Django en etapa de producción.

2.2.2. Ejecutando el Servidor de Desarrollo

Para ejecutar el servidor de desarrollo ejecutamos el siguiente comando:

```
(eV)usuario@pc:~$ python manage.py runserver
```

Esto nos permitira observar nuestro desarrollo en `http:\\localhost:8000`.

[1] WSGI es el enfoque recomendado para la implantación de aplicaciones Django en producción.

2.2.3. Configuración de la Base de Datos

Por omisión django esta configurado con la base de datos *sqlite3*, por lo que crea automáticamente una base de datos para el proyecto, aunque también puede configurarse con los gestores de bases de datos mas populares entre esos: *PostrgreSQL, Oracle, Mysql*. Para configurar nuestro acceso a bases de datos editamos el archivo de nuestro directorio del proyecto **setting.py**.

```
1 DATABASES = {
2   'default': {
3   'ENGINE': 'django.db.backends.sqlite3',
4   'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
5   }
6 }
```

Database Django ENGINE value: el engine (motor) representa el gestor de bases de datos. En Django podemos utilizar:

- *MySQL:* django.db.backends.mysql
- *Oracle:* django.db.backends.oracle
- *PostgreSQL:* django.db.backends.postgresql psycopg2
- *SQLite:* django.db.backends.sqlite3

Database Django NAME value: el name (nombre) representa el nombre de la instancia de la base de datos. El valor de convención puede variar dependiendo del tipo de gestor de base de datos. Por ejemplo en *Sqlite3* representa la dirección del archivo de la base de datos, mientras que en Mysql indica el nombre lógico de la instancia. Todas las bases de datos a excepción de Sqlite3 necesitan configuración y deben descargarse los paquetes correspondientes.

- *Database Python Package:* `(eV)usuario@pc:~$ pip installation syntax`
- *PostgreSQL (psycopg2):* `(eV)usuario@pc:~$ pip install psycopg2`
- *MySQL (mysql-python):* `(eV)usuario@pc:~$ pip install mysql-python`
- *Oracle (cx Oracle):* `(eV)usuario@pc:~$ pip install cx Oracle`

2.2.4. Ejecutando migraciones iniciales

Las migraciones en Django se encargan de que toda la lógica asociada con la base de datos sea reflejada en la base de datos como tal. Cuando iniciamos un proyecto en Django, hay una serie de migraciones necesarias que Django necesita para mantener información de las sesiones y administradores. Esta siempre sera la primera migración que realizaremos:

```
(eV)usuario@pc:~$ python manage.py migrate
```

2.2.5. Urls, Templates y Apps

Los proyectos en django trabajan, con 3 bloques fundamentales de construccion: *urls, templates apps*. Estos se definen de manera separada aunque luego se conectan entre ellas, lo cual es uno de los principios de diseño de arquitectura débilmente acoplada de Django.

- **urls:** definen los puntos de acceso o donde acceder el contenido.
- **templates:** definen los puntos finales que dan forma al contenido final.
- **apps:** sirven de middleware entre las urls y las plantillas. Añadiendo o alterando contenido desde una base de datos o interacciones de usuario.

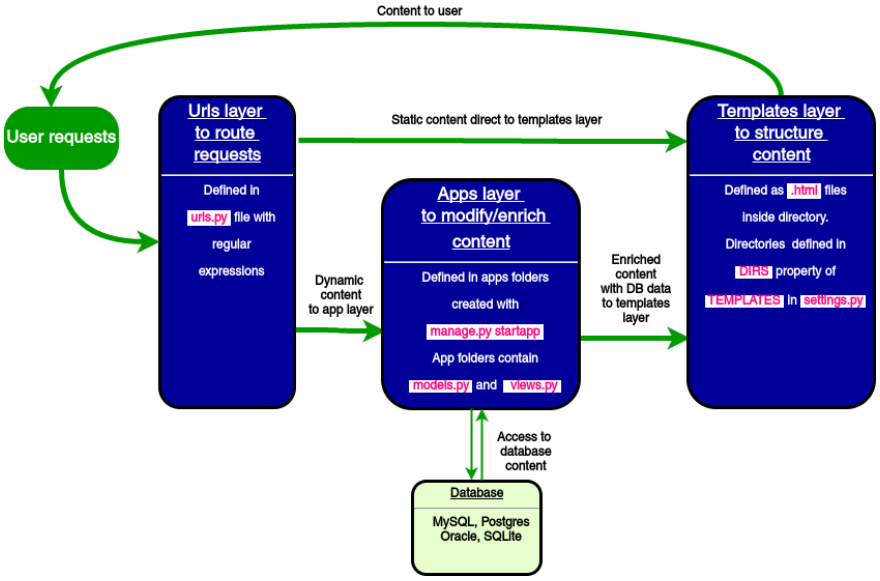

Figura 1: Workflow Django

2.3. Utilizando la Aplicación de Autenticación

Luego de hacer nuestra primera migración Django instala automáticamente la aplicación de autenticación. Si observamos el archivo **settings.py** en INSTALLED_APPS podemos ver que auth es una de las varias aplicaciones integradas, que Django ha instalado para nosotros.
Para usar la aplicación de autenticación necesitamos agregarla a nuestro archivo **urls.py** a nivel de proyecto. Asegúrese de agregar `include` en la segunda línea. Elegimos incluirla en la aplicación `accounts/`, aunque se puede usar cualquier patrón de URL.

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/',
          include('django.contrib.auth.urls')),
]
```

La aplicación de autenticación que ahora hemos incluido nos proporciona varias vistas de autenticación y URL para administrar el inicio de sesión, el cierre de sesión y la administración de contraseñas. Las URLs proporcionadas por *auth* son:

```
accounts/login/ [name='login']
accounts/logout/ [name='logout']
accounts/password_change/ [name='password_change']
accounts/password_change/done/ [name='password_change_done']
accounts/password_reset/ [name='password_reset']
accounts/password_reset/done/ [name='password_reset_done']
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']
accounts/reset/done/ [name='password_reset_complete']
```

2.3.1. Inicio de Sesión

Django por defecto buscará dentro de una carpeta de plantillas llamada **registration** para plantillas de autenticación. La plantilla de inicio de sesión se llama **login.html**.
Creamos un nuevo directorio llamado **registration** y el archivo **login.html** requerido dentro de él. Ingresamos en línea de comandos, los siguientes comandos:

```
(eV)usuario@pc:~$ mkdir templates
```

```
(eV)usuario@pc:~$ mkdir templates/registration
```

```
(eV)usuario@pc:~$ touch templates/registration/login.html
```

Luego incluimos este código en nuestro archivo **login.html**:

```html
<h2>Login</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
```

Este es un formulario estándar de Django que utiliza `POST` para enviar datos y etiquetas `{% csrf_token %}` por motivos de seguridad, es decir, para evitar

un ataque XSS. El contenido del formulario se imprime entre las etiquetas de párrafo gracias a `form.as_p` y luego agregamos un botón de enviar.
Ahora actualizamos el archivo **settings.py** para decirle a Django que busque una carpeta de plantillas en el proyecto. Actualice la configuración DIRS dentro de TEMPLATES de la siguiente manera:

```
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        ...
    },
]
```

Nuestra funcionalidad de inicio de sesión ahora funciona, ahora solo debemos especificar dónde redirigir al usuario después de un inicio de sesión exitoso. Usamos la configuración LOGIN_REDIRECT_URL para especificar esta ruta. En la parte inferior del archivo **settings.py**, agregamos lo siguiente para redirigir al usuario a la página de inicio:

```
#mi_proyecto/settings.py
LOGIN_REDIRECT_URL = '/'
```

Ahora iniciamos nuestro servidor en Django con `(eV)usuario@pc:~$ python manage.py runserver` y navegamos en `http://127.0.0.1:8000/accounts/login`

2.3.2. Crear Usuario Administrador

Para poder iniciar sesión debemos crear un usuario, por lo que creamos una cuenta de superusuario desde nuestro terminal. Salimos del servidor con `control + c` y luego ejecutamos el siguiente comando:

```
(eV)usuario@pc:~$ python manage.py createsuperuser
```

2.3.3. Crear Página de Inicio

Antes de iniciar sesión con nuestro superusuario debemos crear una página de inicio simple, que muestre un mensaje para los usuarios que han cerrado la sesión y otro para los usuarios que han iniciado sesión. Primero salimos del servidor local con `control + c` y luego creamos los nuevos archivos **base.html** y **home.html**.

```
(eV)usuario@pc:~$ touch templates/base.html
```

```
(eV)usuario@pc:~$ touch templates/home.html
```

Ahora agregamos el siguiente código a cada archivo:

```
<!--templates/base.html-->
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>{% block title %}Django Auth Tutorial
                {% endblock %}</title>
</head>
<body>
  <main>
    {% block content %}
    {% endblock %}
  </main>
</body>
</html>
```

```
<!--templates/home.html-->
{% extends 'base.html' %}

{% block title %}Home{% endblock %}

{% block content %}
{% if user.is_authenticated %}
  Hi {{ user.username }}!
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'login' %}">login</a>
{% endif %}
{% endblock %}
```

También podemos actualizar a **login.html** para agregar nuestro nuevo archivo **base.html**:

```
<!-- templates/registration/login.html -->
{% extends 'base.html' %}

{% block title %}Login{% endblock %}

{% block content %}
<h2>Login</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
```

```
{% endblock %}
```

Por último actualizamos nuestro archivo **urls.py** para mostrar la página de inicio. En la tercera línea, importe `TemplateView` y luego agregue un `urlpattern` para ello.

```
#my_project/urls.py
from django.contrib import admin
from django.urls import path, include
from django.views.generic.base import TemplateView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/',
          include('django.contrib.auth.urls')),
    path('', TemplateView.as_view(template_name=
          'home.html'), name='home'),
]
```

Hecho esto corremos nuevamente nuestro servidor en Django e ingresamos a nuestra página para iniciar sesión.

Recomendación: para completar el conocimiento adquirido con esta práctica realice el curso de este enlace: `https://wsvincent.com/django-user-authentication-tutorial-signup/`

Referencias

[1] Django Login/Logout Tutorial (Part 1) - William S. Vincent. (2019). Tomado de https://wsvincent.com/django-user-authentication-tutorial-login-and-logout.