INTRODUCCIÓN A DJANGO

Framework Web.
Un framework web, provee una infraestructura de programación para el desarrollo de sitios web dinámicos, aplicaciones web y servicios web.

Patrón de diseño MVC.
El patrón de diseño **Modelo-Vista-Controlador(MVC)** define una manera de desarrollar software en la que el código para definir y acceder a los datos (el **modelo**) está separado del pedido lógico de asignación de ruta (el **controlador**), que a su vez está separado de la interfaz del usuario (la **vista**). Este enfoque tiene la ventaja de que los componentes tienen un acoplamiento débil, por lo cual cada parte de la aplicación web tiene un único propósito, con esto es posible modificarlas de manera independiente sin afectar a las demás.

Historia.
Django nació en otoño de 2003, de la mano de mano de los programadores web del diario *Lawrence Journal-World*, ***Adrian Holovaty*** *y* ***Simon Willison***, cuando empezarón a usar **Python** para crear sus aplicaciones. Debido a la necesidad creciente por parte de varios sitios de noticias de agregar nuevas características y aplicaciones enteras de manera inmediata, Adrian y Simon desarrollarón un framework de desarrollo web que les ahorrara tiempo a la hora de crear aplicaciones mantenibles. El equipo creador de este framework decidió liberarlo como software de código abierto, fue liberado en Julio de 2005 y lo llamarón **Django**, por el guitarrista de jazz **Django Reinhardt**.

EMPEZANDO CON DJANGO

Instalar Python.
Django está 100% escrito en código puro de Python, por lo cual para poder trabajar con este framework es necesario tenerlo debidamente instalado en tu computadora.

Instalar Django(método distutils).

(HACER EJEMPLO)

Instalar Django(Subversion).
Para trabajar sobre la versión de desarrollo o contribuir con el código de Django, se recomienda instalar Django desde el repositorio de Subversion. Para obtener Django desde el repositorio de Subversion debes utilizar un cliente de Subversion para bajar el código fuente más actual y en cualquier momento actualizar tu copia local del código, *checkout local.* Al último código de desarrollo de Django alojado en el repositorio de Subversion se le hace referencia como el *trunk*.

- Tener debidamente instalado un cliente de Subversion.
- Hacer *check out* del *trunk* usando el comando *svn co http://code.djangoproject.com/svn/django/trunkdjtrunk*
- Crear *site-packages/django.pth* y agrega el directorio *djtrunk* a este, o actualiza tu *PYTHONPATH* agregando *djtrunki*
- Incluye *djtrunk/django/bin* en el PATH de tu sistema. Este directorio incluye utilidades de administración como *django-admin.py*

No es necesario ejecutar *python setup.py install*, esto se realiza por default.

Configurando la base de datos.
Django cuenta con soporte para PostgreSQL, SQLite, MySQL y Oracle.

Django con PostgreSQL.
Si se está utilizando PostgreSQL, necesitarás el paquete *pyscopg*.

Django con SQLite 3.
Si estás usando Python 2.5 ya tienes integrado el soporte modular para SQLite, si tienes instalada una versión inferior de Python necesitas SQLite 3, la obtienes desde http://www.djangoproject.com/r/sqlite/ y el paquete *pysqlite* desde http://www.djangoproject.com/r/python-sqlite/. Asegúrate de tener *pysqlite* en una versión 2.0.3 o superior.

Django con MySQL.
Django requiere MySQL 4.0 o superior, también se necesita instalar el paquete *MySQLdb* desde http://www.djangoproject.com/r/python-mysql/.

Comenzando un proyecto con Django.
Un *proyecto* es una colección de configuraciones para una instancia de Django, incluyendo la configuración de la base de datos, opciones específicas de Django y configuraciones específicas de aplicaciones.

(HACER EJEMPLO)

El servidor de desarrollo.
Django incluye un servidor web ligero que puedes usar mientras estás desarrollando tu sitio. Este servidor de desarrollo vigila tu código a la espera de cambios y se reinicia automáticamente, ayudándote a hacer algunos cambios rápidos en tu proyecto sin necesidad de reiniciar nada.

(HACER EJEMPLO)

El servidor de desarrollo puede manejar fiablemente una sola petición a la vez, y no ha pasado por una auditoría de seguridad de ningún tipo.
Ahora que el servidor está corriendo, puedes ir a tu navegador y visitar http://127.0.0.1:8000/.
En caso de que le puerto asignado por defecto se encuentre en uso, debemos indicar en la instrucción que puerto deseamos utilizar.

(VER EJEMPLO)

COMO TRABAJA DJANGO

Django(MTV).
Django es un framework MTV (Model-Template-View), una modificación del clasico MVC (Model-View-Controller). Esto se debe a que sus desarrolladores no tenian la intención de seguir algún patrón de desarrollo en particular, sino hacer el framework lo más funcional posible.

Analogía MVC-MTV:
- El modelo en Django sigue siendo modelo.
- La **vista** en Django se llama **plantilla(Template).**
- El **controlador** en Django se llama **vista.**

Configuración de las rutas.
Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, ésta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf esta construido con expresiones regulares en Python.

***El modelo.**
El modelo define los datos almacenados de la aplicación, contiene los campos básicos y el comprotamiento de los datos. Se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Por lo general cada modelo se convierte en un tabla de la base de datos.
- Cada modelo es una subclase de *django.db.models.Model*.
- Cada atributo de un modelo representa a un campo de una tabla.
- Django automáticamente nos da acceso a la base de datos.

Referencia de los campos.

Referencia del Modelo

Tipos de campos	Opciones de campo
AutoField **BigIntegerField** **BooleanField** **NullBooleanField** **CharField** max_length **CommaSeparateIntegerField** **DateField** auto_now = False auto_now_add = False **DateTimeField** auto_now = False auto_now_add = False **DecimalField** max_digits = 10 decimal_places = 2 **IntegerField** **SlugField** max_length = 50 **SmallIntegerField** **TextField** auto_now = False auto_now_add = False **URLField** verify_exists = True max_length = 200 **FloatField** **IPAddressField** **GenericIPAddressField** **EmailField** max_length = 75 **FileField** upload_to = 'cargas/' storage = FileSystemStorage **FilePathField** path = '/home/archivos' match = r'\.pngS' recursive = False **ImageField** upload_to = 'cargas/' height_field = 'nombre_campo' width_field = 'nombre_campo' **PositiveIntegerField** **PositiveSmallIntegerField** **ForeignKey** related_name = 'modelo' limit_choices_to = query_kwargs to_field = 'llave_campo' **ManyToManyField** related_name = 'modelo' limit_choices_to = query_kwargs symmetrical = True through = 'ModeloIntermedio' **OneToOneField** parent_link = 'campo'	**null** = False **blank** = False **choices** = tupla_de_opciones **db_column** = 'nombre de columna' **db_index** = False **db_tablespace** = 'nombre_tablespace' **default** = 'valor' **editable** = True **error_messages** = diccionario_de_mensajes **help_text** = 'text' **primary_key** = True **unique** = True **unique_for_date** = 'campo_fecha' **unique_for_month** = 'campo_fecha' **unique_for_year** = 'campo_fecha' **verbose_name**='nombre' **validators** = lista_de_validadores

Elaborado por: **Sergio Infante (@neosergio)**

***) El shell de Django.**

Los modelos nos permiten manipular los datos: registrarlos, editarlos, actualizarlos, consultarlos, eliminarlos y realizar procesos con ellos. Toda esta manipulación se reflejará en las vistas y posteriormente en las plantillas para mostrar los resultados en el navegador, esta manipulación se le conoce generalmente como: Consultas.

El shell.

Es el intérprete interactivo de Python, que nos permitirá probar los modelos, hacer consultas, analizar resultados, antes de elaborar las vistas. (VER EJEMPLO)

Las consultas.

Las consultas en base a los modelos de Django son la base de todo el desarrollo en este framework, estas consultas nos permiten saber, por ejemplo, la lista de usuarios, los correos electrónicos de los que hacen comentarios, el primer comentario de un artículo, etc.

Metodos para hacer consultas

Metodos que retornan nuevos conjuntos de consultas	Metodos que no retornar conjuntos de consultas
filter(búsquedas) exclude(búsquedas) annotate(anotaciones) order_by(campos) reverse( ) distinct( ) values(campos) values_list(campos) dates(campos) none( ) all( ) select_related(campos) prefetch_related(búsquedas) extra( ) defer(campos) only(campos) using(alias) select_for_update(nowait=False)	get(búsquedas) create(atributos) get_or_create(atributos) bulk_create(objetos) count( ) in_bulk(id_list) iterator( ) latest(campo) aggregate(agregaciones) exists( ) update(atributos) delete( )
Búsquedas	**Funciones de Agregaciones / Anotaciones**
exact iexact contains icontains in gt gte lt lte startswith istartswith endswith iendswith range year month day week_day isnull search regex iregex	Avg Count Max Min StdDev Sum Variance

Elaborado por: **Sergio Infante (@neosergio)**

***La vista.**

La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados. El ORM (Modelo-Objeto-Relacional) de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios, la vista contiene toda la lógica necesaria para devolver una respuesta. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla. Las consultas son parte fundamental de las vistas, permiten elegir qué tipo de contenido se visualizará.

***La plantilla.**

Django posee un componente conocido como “el motor de plantillas”, este motor brinda un poderoso mini-lenguaje para definir detalles de la capa de la aplicación, que visualizará el usuario. Esto refuerza la separación de la lógica de programación y de presentación.

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django. Las plantillas son usadas para producir HTML, pero en Django las plantillas son igualmente capaces de generar cualquier formato basado en texto (XML, CSS, Javascript, CSV, etc). La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del frontend (interacción con los usuarios).

Etiquetas y filtros.

Plantillas en Django

Etiquetas	Filtros
{% autoescape on %} {% endautoescape %} {% block nombre %} {% endblock%} {% comment %} {% endcomment %} {% csrf_token %} {% cycle 'elemento1' 'elemento2' %} {% debug %} {% extends 'base.html' %} {% filter force_escape|lower %} {% endfilter %} {% firstof variable1 variable2 variable 3 %} {% for i in lista %} {% empty %} {% endfor %} {{ forloop.counter }} {{ forloop.counter0 }} {{ forloop.revcounter }} {{ forloop.revcounter0 }} {{ forloop.first }} {{ forloop.last }} {{ forloop.parentloop }} {% if condicion %} {% elif condicion %} {% else %} {% endif %} {% ifchanged %} {% else %} {% endifchanged %} {% ifequal %} {% endifequal %} {% ifnotequal %}{% endifnotequal %} {% include 'otra_plantilla.html' %} {% load %} {% now %} {% regroup %} {% spaceless %} {% endspaceless %} {% ssi %} {% templatetag openblock / closeblock / openvariable / closevariable / openbrace / closebrace / opencomment / closecomment %} {% url %} {% widthradio %} {% with %} {% endwith %}	add addslashes capfirst center cut date default default_if_none dictsort dictsortreversed divisibleby escape escapejs filesizeformat first fix_ampersands floatformat force_escape get_digit iriencode join last length length_is linebreaks linebreaksbr linenumbers ljust lower make_list phone2numeric pluralize pprint random removetags rjust safe safeseq slice slugify stringformat striptags time timesince timeuntil title truncatechars truncatewords unordered_list upper urlencode urlize urlizetrunc wordcount wordwrap yesno

Elaborado por: **Sergio Infante (@neosergio)**

***Los formularios**

Los formularios permiten el ingreso de datos para su procesamiento, ya sea para crear nuevos contenidos, para modificar el contenido que ya esta registrado previamente y hasta para realizar búsquedas. Django permite:

- Mostrar un formulario HTML generado a partir de reglas básicas.
- Generar formularios HTML validados a partir de modelos ya declarados.
- Validar información que se desa registrar a través del formulario.
- Mostrar nuevamente el formulario, haciendo notar los errores que ha producido la validadción.
- Convertir la información subida en tipos de datos de Python, para procesarlos de acuerdo a las vistas.

Campos, atributos y opciones de validadción.

Campos

BooleanField
CharField
- max_length
- min_length

ChoiceField
- choices

TypedChoiceField
- choices
- coerce
- empty_value

DateField
- input_formats

DateTimeField
- input_formats

DecimalField
- max_value
- min_value
- max_digits
- decimal_places

EmailField
FileField
FilePathField
- path
- recursive
- match
- allow_files
- allow_folders

FloatField
ImageField
IntegerField
- max_value
- min_value

IPAddressField
GenericIPAddressField
MultipleChoiceField
- choices

TypedMultipleChoiceField
- choices
- coerce
- empty_value

NullBooleanField
RegexField
- regex

SlugField
TimeField
- input_formats

URLField
- max_length
- min_length

Campos ligeramente complejos

ComboField
- fields

MultiValueField
- fields

SplitDateTimeField
- input_date_formats
- input_time_formats

Argumentos comunes para los campos

required
label
initial
widget
help_text
error_messages
validators
localize

Widgets

Widget
- attrs

TextInput
PasswordInput
- render_value

HiddenInput
MultipleHiddenInput
- choices

FileInput
ClearableFileInput
DateInput
- format

DateTimeInput
- format

TimeInput
- format

Textarea
CheckboxInput
- check_test

Select
- choices

NullBooleanSelect
SelectMultiple
RadioSelect
CheckboxSelectMultiple
MultiWidget
- widgets

SplitDateTimeWidget
- date_format
- time_format

SplitHiddenDateTimeWidget
SelectDateWidget
- years

Campos que manejan relaciones

ModelChoiceField
- queryset
- empty_label

ModelMultipleChoiceField
- queryset

Validadores
(también sirven para modelos)

RegexValidator
- regex
- message
- code

URLValidator
validate_email
validate_slug
validate_ipv4_address
validate_ipv6_address
validate_ipv46_address
validate_comma_separated_integer_list
MaxValueValidator
- max_value

MinValueValidator
- min_value

MaxLengthValidator
- max_length

MinLengthValidator
- min_length

Objetos formulario.
Un objeto formulario en Django es una secuencia de campos y reglas de validación, que permiten depurar la información requerida y procesarla eficientemente. Estos campos y reglas deben ser declarados en el orden que se desea que aparezcan. Las clases formulario son creadas como subclases de *django.forms.Form* y tienen un estilo de declaración muy similar a los modelos de Django.

Formularios a partir de modelos.
Si se esta construyendo una aplicación que gestiona una base de datos, lo más apropiado es usar los modelos ya declarados como formularios y así evitar estar repitiendo las mismas reglas para procesar los datos. Por esta razón, Django provee una clase de ayuda que permite crear un formulario a partir de un modelo, esta clase se llama ModelForm.

***Los archivos estáticos.**
Cuando se trabaja con Django, a los elementos que son imagenes, hojas de estilo y códigos de javascript se les conoce generalmente como contenido estático. Mediante *django.contrib.staticfiles*, Django gestiona el contenido estático para las aplicaciones y los ordena en una sola ubicación.

El primer lugar donde inicia el manejo de los archivos estáticos reside en el archivo de configuraciones del proyecto: *settings.py*, en este archivo tenemos 4 elementos exclusivamente dedicados al manejo del contenido estático: *STATIC_ROOT, STATIC_URL, STATICFILES_DIRS y STATICFILES_FINDERS*

Posteriormente se debe crear el directorio *static*, este debe crearse dentro del directorio del proyecto. Dentro del directorio *static*, se debe tener una carpeta por cada tipo de contenido estático que se vaya a incluir; *css, img, js*.

Para hacer uso del contenido estático en las plantillas, cada vista debe finalizar con *context_instance=RequestContext(request)* para poder usar *{{STATIC_URL}}* y así en caso de cambiar el nombre o ubicación de la carpeta static en producción, no afecte el proyecto.

Archivos predeterminados.
Archivos del proyecto:

- __init__.py: Es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python.
- manage.py: Contiene una porción de código que permite interactuar con el proyecto de Django de muchas formas.
- settings.py: Contiene todas las opciones/configuraciones para este proyecto.
- urls.py: Contiene las rutas que están disponibles en el proyecto, manejado por URLConf.

Archivos de la aplicación:

- __init__.py: Es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python.
- models.py: Se declaran las clases del modelo.
- views.py: Se declaran las funciones de la vista.
- test.py: Se declaran las pruebas necesarias para la aplicación.

Creando una aplicación(VER EJEMPLO).

Settings.py

Una parte muy importante del proyecto es el archivo settings.py, este archivo permite configurar la conexión a la base de datos, la zona horaria, el idioma, los directorios principales del proyecto, las aplicaciones del proyecto, entre otras cosas.

Codificación de caracteres (VER EJEMPLO)

Ruta del proyecto.

Es importante la configuración de la ruta del proyecto, esto permitirá lanzar la aplicación desde cualquier directorio y mover el proyecto a cualquier computador con Django instalado. (VER EJEMPLO)

Administradores.

Cuando Django tiene la opción de DEBUG=False, las notificaciones de error de código deben ser enviadas vía correo electrónico a los administradores, junto con los detalles completos del error. **(VER EJEMPLO)**

Configuración de la base de datos. (VER EJEMPLO)

Zona Horaria.

Django permite configurar la zona horaria del proyecto. **(VER EJEMPLO)**

Configuración del idioma.

Django también permite configurar el idioma que usará de manera predeterminada
para su funcionamiento. **(VER EJEMPLO)**

Creación de la base de datos. (VER EJMPLO)

Direcciones del proyecto.

Para visualizar los cambios que hicimos y la interfaz administrativa de Django, debemos modificar el archivo urls.py. **(VER EJEMPLO)**