

Sistema de Gestion de Consultorios Medicos

Universidad Nacional de Salta



Facultad de Ciencias Exactas
Seminario de Computación

Alumno: Ricardo Daniel Quiroga
Director de Tesis: Ernesto Sanchez
Proyecto Especifico
Febrero de 2014

Escribiria una bonita frase copiada de algun lado pero no se me ocurre donde buscar algo que refleje todo el esfuerzo puesto en completar este proyecto, mejor solo doy las gracias a todos aquellos que de una u otra forma ayudaron durante el transcurso del mismo.

Índice general

1. Introduccion	1
1.1. Objetivos Generales	1
1.2. Resumen	1
1.3. Motivación	1
1.4. Descripción del Proyecto	1
1.5. Elección de la metodología de Programación	1
1.6. Elección de la Base de Datos	1
1.7. Elección de la Tecnología	1
2. Instalacion y Configuracion	3
2.1. Requerimientos	3
2.1.1. Requerimientos de Hardware	3
2.1.2. Requerimientos de Hardware	3
2.1.3. Requerimientos de Software	3
2.2. Apache	4
2.2.1. Instalacion en Limpio	4
2.2.2. Instalacion mediante WAMP, LAMP, MAMP, WAPP	4
2.2.3. Configuracion	5
2.3. PostgreSQL	5
2.3.1. Instalacion	6
2.3.2. Creacion de la Base de Datos	6
2.4. Python	7
2.4.1. La Filosofia detras de Python	7
2.4.2. Baterias Incluidas	8
2.4.3. Implementaciones	8
2.4.4. Instalacion	8
2.4.5. Probando Python	8
2.5. Django	10
2.5.1. MVC	10
2.5.2. Django y el MVT	10
2.5.3. El Modelo	11
2.5.4. La Vista	12
2.5.5. La Plantilla	13
2.5.6. La Configuracion de Rutas	15
2.5.7. Instalar Django	15
2.6. Instalando el Resto de Las Dependencias	15
2.6.1. psycopg2	15
2.6.2. ReportLab	16
2.6.3. easy_thumbnails	16
2.6.4. django_extensions	16
2.6.5. django_cron	16
2.7. mod_wsgi	17

2.7.1.	WSGI	17
2.7.2.	Descargar e Instalacion	17
2.7.3.	Cargando el Modulo en Apache	17
2.7.4.	Configuracion del Proyecto	17

Apéndices	21
------------------	-----------

Índice de figuras

2.1. Ejecutando Python en la Terminal	9
2.2. Diagrama del Patron MVC Modelo Vista Controlador	11
2.3. Generaciones de Herramientas de Desarrollo Web	11
2.4. El patron Modelo Vista Template de Django	12

Capítulo 1

Introduccion

1.1. Objetivos Generales

El siguiente proyecto de Tesis tiene como objetivo el diseño y desarrollo de un sistema de gestion para el area Salud, especificamente aplicandose al area de consultorios medicos, permitir un mejor seguimiento de la evolucion de los pacientes que ally se trata mediante su historia clinica, ademas facilitar la asignacion de turnos y pequeñas consultas.

Lo que se pretende es brindar un sistema modular y eficiente que permita su facil aplicacion y ademas de brindar la posibilidad de modificacion tanto para adecuacion para casos especificos como extension de sus funcionalidades.

1.2. Resumen

1.3. Motivación

1.4. Descripción del Proyecto

1.5. Elección de la metodología de Programación

1.6. Elección de la Base de Datos

1.7. Elección de la Tecnología

Capítulo 2

Instalacion y Configuracion

En este Capitulo Aparte de guiarle para realizar una exitosa implementacion Local del Servidor de Produccion se hara referencia a cada una de las Herramientas y librerias Utilizadas.

2.1. Requerimientos

2.1.1. Requerimientos de Hardware

Cualquier equipo que cumpla con las Caracteristicas para correr Windows 7 es suficiente en terminos de requerimientos minimos de Hardware siempre y cuando el numero de usuarios esperados no sea alto, despues el resto dependera de sus necesidades.

2.1.2. Requerimientos de Hardware

- Procesador x86, x64 de 1 Ghz o superior.
- Memoria Ram 1 GB o Superior

2.1.3. Requerimientos de Software

- Apache 2.2
- PostgreSQL 9.2
- Python 2.7.x o Python 2.6.x
- Django 1.3.x o Superior
- PGAdmin
- psycopg2
- mod_wsgi
- ReportLab
- easy_thumbnails
- django_extensions
- django_cron

2.2. Apache

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.12 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. Su nombre se debe a que Behelendorf quería que tuviese la connotación de algo que es firme y enérgico pero no agresivo, y la tribu Apache fue la última en rendirse al que pronto se convertiría en gobierno de EEUU, y en esos momentos la preocupación de su grupo era que llegasen las empresas y civilizasen.^{el} paisaje que habían creado los primeros ingenieros de internet. Además Apache consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. En inglés, a patchy server (un servidor "parcheado") suena igual que Apache Server.

El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation.

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración.

Existen 2 caminos para instalar Apache La Primera Hacer una instalacion Limpia de Apache, la 2da es cuando no se quiere trastear con tanta configuracion por lo que opta por infraestructuras tipo WAMP, LAMP, WAPP, etc.

2.2.1. Instalacion en Limpio

Solo recomiendo este tipo de instalacion desde 0 para quienes ya poseen un conocimiento avanzado en cuanto a manejo de servidores.

Descargamos de [Apache.org](http://www.apache.org) la ultima version disponible, puedes utilizar el siguiente vinculo: <http://www.apachehaus.com/cgi-bin/download.plx>.

Crea dos carpetas en la unidad C, la primera de nombre **Apache** y la segunda **servidor**. Descomprime el archivo descargado y ejecútalo, sigue los pasos de la instalación y de los datos que te piden solo escoge el destino de la instalación, que será la carpeta que creaste en **C:\Apache**, los otros datos déjalos de la forma predeterminada para configurarlos más tarde. El programa al instalarse crea un icono en el área de notificación que te permitirá: iniciar, detener y reiniciar Apache; tienes que tener en cuenta que cualquier cambio que hagas en el archivo de configuración no tendrá efecto hasta que reinicies el servidor.

2.2.2. Instalacion mediante WAMP, LAMP, MAMP, WAPP

Existen una infinidad de Paquetes precompilados y configurados, con Apache, PHP, PosgreSQL o MySQL y mas. Dichas infraestructuras suelen nombrarse como el acronomico de las herramientas que agrupan por ejemplo:

- WAMP Windows Apache MySQL PHP
- WAPP Windows Apache PosgreSQL PHP
- LAMP Linux Apache MySQL PHP
- MAMP Mac OS Apache MySQL PHP

Algunas distribuciones mas usadas disponibles Para Windows son WAMP Server <http://www.wampserver.com/> (WAMP), XAMPP <http://sourceforge.net/projects/xampp/> (WAMP + Perl), Bitnami <http://bitnami.com/stack/wapp> (WAPP) solo nos resta elegir cualquiera de ellas e instalarlas, aparte de la ruta de instalacion nos pedirán el usuario y contraseña para acceder al motor de Base de Datos.

2.2.3. Configuracion

Toda la configuración para el funcionamiento de Apache se guarda en un archivo de texto nombrado: **httpd.conf** que se encuentra en la ruta **C:\Apache\conf** si realizamos una instalacion en limpio o **C:\wamp\bin\Apache\conf** si instalamos el paquete multiple preconfigurado no es necesario realizar este paso por lo que lo podremos saltar.

Al archivo **httpd.conf** lo podemos editar en cualquier editor de texto como Notepad. Buscamos la línea que dice

```
Listen LocalHost:80
```

y la Cambiamos por:

```
Listen 80
```

Ahora buscamos la instruccion:

```
DocumentRoot "C:\xxxxxxx"
```

y la Cambiamos por:

```
DocumentRoot "C:\Servidor"
```

Recordar que al inicio de la instalacion creamos una Carpeta llamada Servidor en la unidad C. Por ultimo solo nos queda reiniciar el servidor Apache e introducir la siguiente direccion <http://127.0.0.1> si nos aparece una pagina **It's Work!** felicidades Apache esta Funcionando.

2.3. PostgreSQL

PostgreSQL es un gestor de base de datos relacional que puede correr tanto bajo sistemas operativos Windows como en distribuciones Linux como Red Hat, Suse, Centos, etc.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

El nombre hace referencia a los orígenes del proyecto como la base de datos "post-Ingres", y los autores originales también desarrollaron la base de datos Ingres.

El proyecto post-ingres pretendía resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos. Se esforzaron en introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones - las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En Postgres la base de datos «comprendía» las relaciones y podía obtener información de tablas relacionadas utilizando reglas. Postgres usó muchas ideas de Ingres pero no su código.

2.3.1. Instalacion

La versión de PostgreSQL que he utilizado durante el desarrollo del sistema es la 9.2.x, quizás cuando leas esto haya salido una nueva versión la cual no debería generar inconvenientes además de que es posible que el proceso de instalación pueda variar.

El primer paso es descargar el instalador de PostgreSQL para Windows, lo puedes descargar desde el enlace siguiente <http://www.postgresql.org/download/windows>, nos bajara un instalador similar a **postgresql-9.2.3-rc1-windows.exe** lo ejecutamos como administrador.

Si tenemos activado el control de cuentas de usuario nos mostrará una advertencia con el texto "¿Desea permitir que este programa realice cambios en el equipo?", pulsaremos "Sí" para continuar con la instalación de PostgreSQL.

Indicaremos la carpeta de instalación de PostgreSQL, donde se guardarán los ejecutables, librerías y ficheros de configuración de PostgreSQL en mi caso el directorio es **C: \PosgreSQL 9.2**, Indicaremos también la carpeta donde se guardarán los datos por defecto de PostgreSQL **C: \psql-data**.

Solo nos queda introducir la contraseña para el superusuario "postgres" que será con el que iniciemos sesión para administrar la base de datos, después podremos crear otros usuarios si es necesario. Además introduciremos el puerto de escucha para la conexión con el servidor PostgreSQL, por defecto el 5432.

Seleccionaremos la configuración regional y comenzará la instalación, con esto PostgreSQL quedará instalado. Si tenemos algún cortafuegos (firewall) deberemos abrir el puerto 5432.

2.3.2. Creacion de la Base de Datos

Junto con la instalación de PostgreSQL se instala el PGAdmin III que es una herramienta GUI para administrar el motor de base de datos. Iniciamos el Programa, desplegaremos "Server Groups", dentro desplegaremos "Servidores" dentro de éste pulsaremos con el botón derecho del ratón sobre "PostgreSQL 9.0 (localhost:5432)", en el menú emergente seleccionaremos "Conectar".

Introduciremos la contraseña para el superusuario postgres (la contraseña introducida en la instalación).

Pulsaremos con el botón derecho del ratón sobre "Bases de datos", seleccionaremos "Nueva Base de Datos", en la pestaña "Propiedades" introduciremos los siguientes datos:

- Nombre: nombre de la base de datos, en nuestro caso "BDSem".
- Propietario: seleccionaremos el usuario creado anteriormente "postgres".
- Codificado: seleccionaremos UTF8.
- Tablespace: seleccionaremos el tablespace creado anteriormente "pg_default".
- Colación: seleccionaremos "Spanish, Argentina".
- Tipo carácter: seleccionaremos "Spanish, Argentina".

Pulsaremos "OK" para crear la base de datos, con esto ya tendremos nuestra base de datos aunque vacía, el resto como creación de las Tablas correspondientes necesarias para el proyecto lo haremos más adelante mediante Django.

2.4. Python

Django esta escrito puramente en Python, por lo que Obiamente Necesitaremos Instalar Python. Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,¹ que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y conteo de referencias para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse en algunas situaciones hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

2.4.1. La Filosofía detras de Python

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" (ünpythonic.^{en} inglés).

Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en El Zen de Python, algunos de ellos son:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Aunque lo práctico gana a la pureza.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.

- Debería haber una y preferiblemente sólo una manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.¹⁵
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

2.4.2. Baterías Incluidas

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas. Esto viene de la filosofía "pilas incluidas" ("batteries included") en referencia a los módulos de Python. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en C como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como C y C++, los cuales son capaces de interactuar con otras bibliotecas, **Python es un lenguaje que combina su clara sintaxis con el inmenso poder de lenguajes menos elegantes.**

2.4.3. Implementaciones

En la actualidad existen diversas implementaciones de Python

- **CPython** es la implementación original, disponible para varias plataformas en el sitio oficial de Python.
- **IronPython** es la implementación para .NET
- **Stackless Python** es la variante de CPython que trata de no usar el stack de C www.stackless.com
- **Jython** es la implementación hecha en Java
- **Pippy** es la implementación realizada para Palm pippy.sourceforge.net
- **PyPy** es una implementación de Python escrita en Python y optimizada mediante JIT pypy.org

2.4.4. Instalacion

Para este proyecto se utilizo CPython pero no la version Oficial [urlhttp://www.python.org](http://www.python.org) sino la que distribuye Active State <http://www.activestate.com> llamada **Active Python** la cual provee características adicionales a version oficial, podremos descargar la ultima version desde <http://www.activestate.com/activepython/downloads> aunque se recomienda instalar la version 2.7.x para evitar cualquier posible problema.

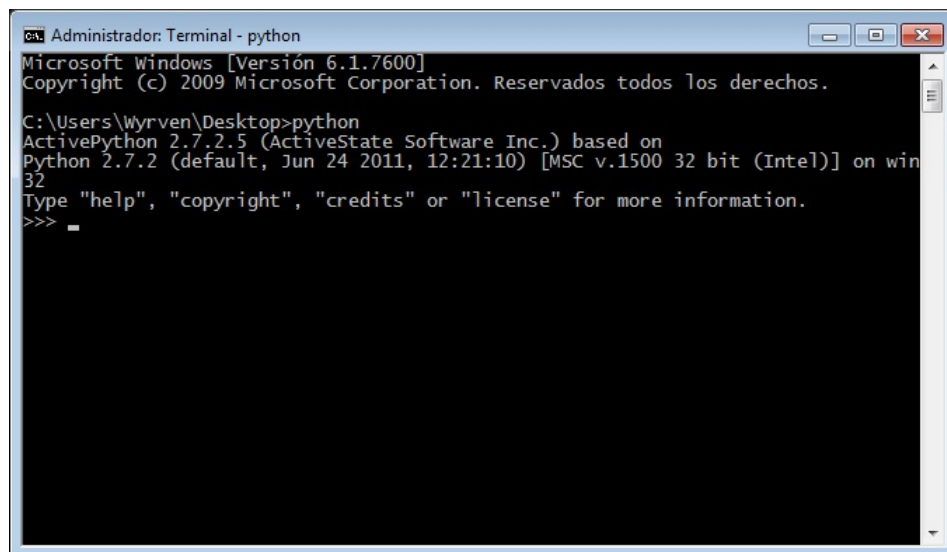
2.4.5. Probando Python

Para probar que la instalacion haya sido correcta abriremos la Terminal cmd.exez escribiremos:

```
python
```

Si todo va bien nos debera aparecer algo similar a:

En caso contrario deberias revisar que la ruta de Python este dentro de la variable PATH del sistema.



```
ca. Administrador: Terminal - python
Microsoft Windows [Versión 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Wyrven\Desktop>python
ActivePython 2.7.2.5 (ActiveState Software Inc.) based on
Python 2.7.2 (default, Jun 24 2011, 12:21:10) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Figura 2.1: Ejecutando Python en la Terminal

2.5. Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el paradigma conocido como **Model Template View**. 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 http://es.wikipedia.org/wiki/Django_Reinhardt.

En junio del 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro.

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). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

2.5.1. MVC

Antes de Explicar como funciona Django empezare por una breve explicacion de el patrón (MVC) Modelo Vista Controlador el cual es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' actuando como Middleware¹.

La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

2.5.2. Django y el MVT

Si hicieramos una clasificacion de Herramientas de desarrollo web, podriamos clasificar a Django como parte de la tercera generacion:

Sin embargo más alla de las clasificaciones que podrían existir, está el entender como funciona realmente, al entenderlo se puede llegar a dominarlo.

Dijimos que era un framework MTV (una modificación de MVC, nada que ver con un canal de música), esto se debe a que los desarrolladores no tuvieron la intención de seguir algún patron de desarrollo, sino hacer el framework lo más funcional posible.

¹Middleware es el software que proporciona un enlace entre aplicaciones de software independientes. Middleware a veces se llama a la vía que conecta dos aplicaciones y pasa los datos entre ellas. Los Middleware permiten que los datos contenidos en una base de datos puedan ser accedidos a través de otra. Ahorra el tiempo a los programadores.

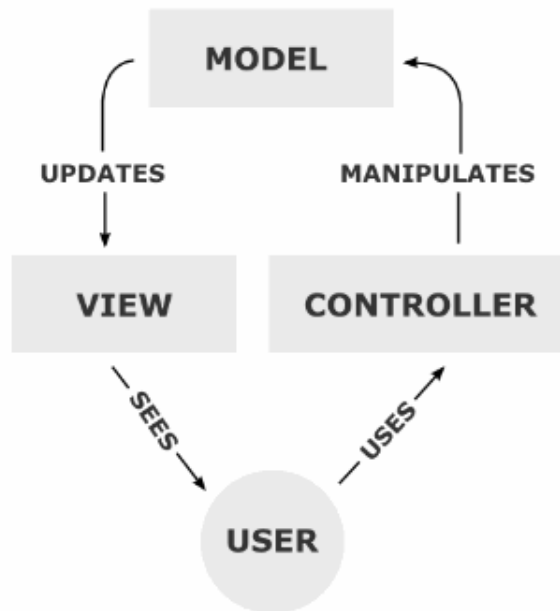


Figura 2.2: Diagrama del Patron MVC Modelo Vista Controlador

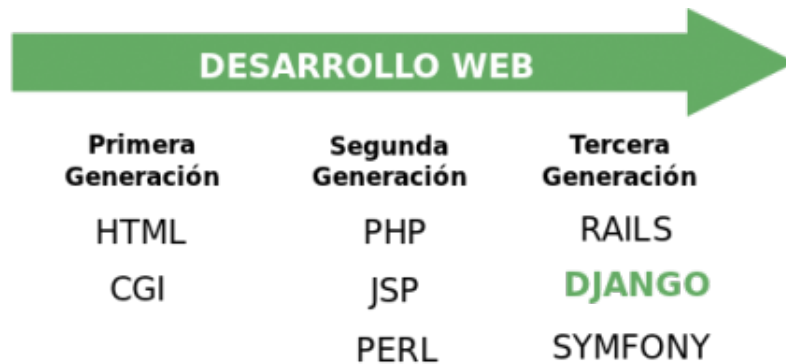


Figura 2.3: Generaciones de Herramientas de Desarrollo Web

- **El Modelo** en Django sigue siendo el modelo
- **La Vista** en Django se llama Plantilla (Template)
- **El controlador** en Django se llama Vista

Una imagen nos hará entender mejor esta relación:

2.5.3. El Modelo

El modelo define los datos almacenados, se encuentra en forma de clases de Python, las clases definidas son traducidas por Django y este genera las Tablas necesarias para el funcionamiento del modelo dentro de la base de datos, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

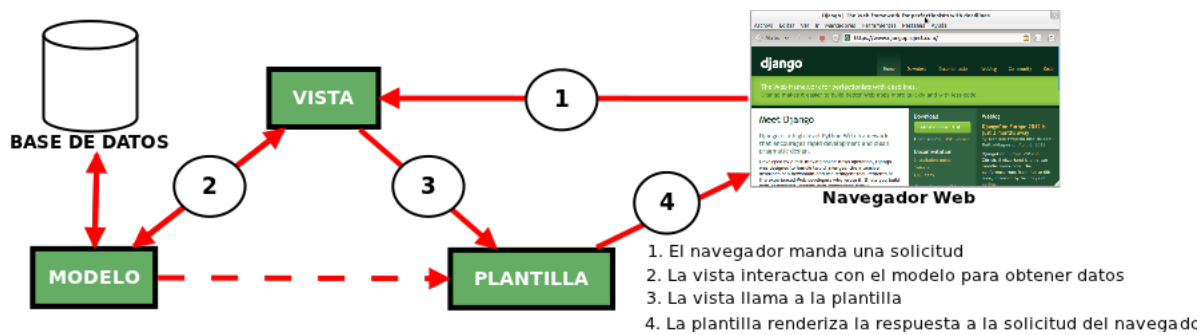


Figura 2.4: El patrón Modelo Vista Template de Django

Aquí un extracto del código mostrando como se implementa uno de los tantos modelos con los que trabaja el Sistema

```

1
2 class Message(models.Model):
3     """
4     Clase Para Manejar mensajes entre usuarios
5     """
6     from_user = models.ForeignKey(User, related_name='from_user')
7     to_user = models.ForeignKey(User, related_name='to_user')
8     date = models.DateTimeField("Fecha y Hora", auto_now_add=True)
9     issue = models.CharField("Asunto", max_length=125, default='')
10    content = models.TextField("Cuerpo del Mensaje")
11    read = models.BooleanField("Leido", default=False)
12
13
14    class Meta:
15        db_table = "Messages"
16        verbose_name = "InboxMessage"
17        verbose_name_plural = "InboxMessages"

```

2

3

2.5.4. La Vista

La vista se presenta en forma de funciones en Python, su propósito es determinar que datos serán visualizados, entre otras cosas más que iremos viendo conforme avanzamos con el curso. El ORM 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. Lo mas 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.

Aquí muestro una vista sencilla que realiza una consulta base de datos que listara todos los usuarios que sean medicos.

²Como algunos lo notaran la variable from_user del modelo internamente es una relacion 1:M dentro de la base de datos.

³La Clase interna Meta define atributos especiales como 'db_name' que hace referencia a como se llamara la tabla dentro de la Bases de Datos.

```

1 def patient_show_medics_list(request):
2     """
3     Muestra el listado de Medicos
4     """
5     mi_template = get_template('Patients/GestionTurnos/medics-list.html')
6     dict = generate_base_keys(request)
7
8     if is_patient(request.user):
9         dict['medics'] = UserInformation.objects.filter( \
10             user__groups__name='Medico')
11         html_cont = mi_template.render(Context(dict))
12         return HttpResponse(html_cont)
13
14     else:
15         #si hay un usuario logueado intentanto acceder sera enviado a una
16         # pagina de error
17         path = request.META['PATH_INFO']
18         return HttpResponseRedirect("/restricted-access%s" %path)

```

Aunque es un ejemplo sencillo podemos apreciar el potencial de Django, como vemos no vemos ningun codigo SQL, pues bien dicho codigo SQL se ejecuta internamente nos aleja del problema de las restricciones de la Base de Datos ya sea que usemos PostgreSQL (como en este sistema), MySQL, SQLServer o SQLite nosotros solo escribiremos codigo Python, El framework se encargara de traducir esa instruccion al motor de bases de datos correspondiente que estemos usando.

```

dict['medics'] = UserInformation.objects.filter( \
    user__groups__name='Medico')

```

Traducido a SQL terminariamos con algo tan horrible como esto:

```

SELECT * FROM UserInformation as Info
INNER JOIN User ON Info.username = User.username
INNER JOIN GroupsByUsers ON User.username = GroupsByUsers.username
...

```

2.5.5. La Plantilla

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en si no solamente crea contenido en HTML (también 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, pueden Extenderse a partir de otras plantillas incluso tiene estructuras de datos como if, por por si es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python.

Esto permite que la lógica del sistema siga permaneciendo en la vista. Aqui la vista para Iniciar Session:

```

{% extends 'base.html' %}

{% block style %}
<link type="text/css" rel="stylesheet" media="all"
      href="/media/css/fancy-forms.css" />
<link type="text/css" rel="stylesheet" media="all"
      href="/media/css/gradient-buttons.css" />
<link type="text/css" rel="stylesheet" media="all"
      href="/media/css/messages.css" />
{% endblock %}

{% block contenido %}
<br /><br /><br />
    {% if not_login %}
        <div class="fancy-form-white" style="width: 350px;
            margin: 0 auto;">
            <h3 class="title">Inciar Session</h3><br />
            <form action="." method="POST">
            <table style="margin: 0 auto; width: 330px;" >
            <tr>
                <td><label for="username">Usuario:</label></td>
                <td><input type="text" name="username" value=""
                    tabindex="1" id="username"></td>
                <td rowspan="2">
                    <input type="submit" value="Login" tabindex="3"
                    class="grad-button-blue" style="height: 50px;">
                </td>
            </tr>
            <tr>
                <td><label for="password">Contrase~na:</label></td>
                <td><input type="password" name="password" value=""
                    tabindex="2" id="password"></td>
            </tr>
            </table>
            </form>
            <br />
        </div>

        {% if login_error %}
            <br />
            <br />
            <div class="alert">Alerta: Error Usuario y/o Contrase~na
                Incorrectos</div>
        {% endif %}

        {% else %}
            <div class="alert">Alerta: Usted ya ha iniciado session con el
                usuario <strong>{{ username }}</strong></div>
            <br />
            <a href="/logout">Cerrar Session</a>
        {% endif %}
    {% endblock %}

```

2.5.6. La Configuración de Rutas

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta 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 y sigue la filosofía de Python: Explicito es mejor que implícito. Este URLConf permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

Fragmento del archivo urls.py del Proyecto

```
(r'^$', base_views.index),
(r'^index/$', base_views.index),
(r'^login/$', base_views.login),
(r'^logout/$', base_views.logout),
(r'^change-password/$', base_views.change_password),
(r'^restricted-access/$', base_views.restricted_access),
(r'^restricted-access/(.+)/$', base_views.restricted_access),
```

2.5.7. Instalar Django

Puedes bajarte Django desde el siguiente enlace <https://www.djangoproject.com/download/1.3.7/tarball/> ⁴ te descargara un paquete llamado Django-1.3.7.tar.gz lo descomprimes en algun directorio luego abres la Terminal y te posicionas sobre el directorio donde descomprimiste y ejecutas:

```
$ python setup.py install
```

Sino mediante el instalador de Paquetes de Python de manera mas automatica escribes en la terminal

```
pip install django==1.3.7
```

Con esto ya tendremos instalado Django.

2.6. Instalando el Resto de Las Dependencias

Ademas de Django en el Proyecto se utilizaron otras Librerias de Python las cuales algunas vienen instaladas y Otras Requieren ser instaladas de manera similar a como instalamos Django.

2.6.1. psycopg2

psycopg2 es un adaptador de base de datos PostgreSQL para el lenguaje de programación Python. psycopg2 fue escrito con el objetivo de ser muy pequeño y rápido y estable.

psycopg2 es diferente del otro adaptador de base de datos, ya que fue diseñado para aplicaciones en gran medida de subprocesos múltiples que crean y destruyen un montón de cursores y hacen que un número notable de inserciones o actualizaciones concurrentes. psycopg2 también proporcionan operaciones asincrónicas completos y apoyo a las bibliotecas de co-rutinas.

Para instalar descargue el precompilado desde <http://www.stickpeople.com/projects/python/win-psycopg/> Ejecutelo con permisos de administrador, nos pedira que seleccionemos la version de python con que se instalar.

⁴la version 1.3.7 no es la ultima version disponible a la hora de crear este informe estaba por la 1.6.2 ya que Django se actualiza constantemente.

2.6.2. ReportLab

ReportLab es la ultra-robusto motor de código abierto a prueba de tiempo para la creación de documentos PDF y gráficos vectoriales personalizado. Escrito en Python, ReportLab es rápido, flexible y una plataforma cruzada.

Proporciona un completo conjunto de herramientas de programación para la creación de documentos y gráficos complejos. Ofrecemos una serie de componentes de forma gratuita y de código abierto, además de un paquete comercial con características adicionales.

Para Instalar descargue el instalado desde <http://www.reportlab.com/software/installation/> y proceda de manera similar a como hizo con la instalacion de psycpg2.

2.6.3. easy_thumbnails

Easy_Thumbnails es Una potente aplicación thumbnailing ⁵, pero fácil de implementacion para Django.

Para Instalar solo ejecute el siguiente comando en terminal, no Se necesita configurar nada en el proyecto el mismo esta previamente configurado.

```
pip install easy-thumbnails
```

2.6.4. django_extensions

Django.Extensions es una coleccion de Extensiones (utilidades) Personalizadas de diferentes autores no relacionados con el Proyecto Django, para extender las capacidades del Framework.

Para Instalar solo ejecute el siguiente comando en terminal ⁶

```
pip install django-extensions
```

2.6.5. django_cron

Django-cron permite ejecutar código de Django de manera recurrente para el seguimiento y ejecución de las tareas. En este caso no es Necesario Instalar Nada, viene junto con el Código Fuente del Proyecto. Igualmente si tiene curiosidad puede visitar la pagina del proyecto <https://github.com/Tivix/django-cron>

⁵Cuando hablamos de thumbnails nos referimos a las diferentes miniaturas que son versiones en distintos tamaños de una imagen y son usadas para ayudar a su organización y reconocimiento.

⁶Importante, no todas las funcionalidades estan soportadas en Windows, pero en cuanto al proyecto no hay problemas.

2.7. mod_wsgi

mod_wsgi es un módulo de Apache que provee una interfaz WSGI para correr Web en Python sobre Apache. Y esto, es todo lo que necesitas para que tus archivos *.py se ejecuten por medio de un navegador Web.

2.7.1. WSGI

WSGI es el acronimico de Web Server Gateway Interface que es una especificacion para una simple y universal interfaz entre una aplicacion web (en nuestro caso una aplicacion escrita en Django) y un servidor Web para el lenguaje de programacion Python. Es un estandar de Python el cual se describe con detalle en la PEP 333 ⁷

2.7.2. Descargar e Instalacion

Asumiendo que ya tienes instalado Python y Apache, solo debes descargar el paquete libapache2-mod-wsgi ,la ultima version de mod_wsgi se puede descargar desde su pagina oficial <https://code.google.com/p/modwsgi/> descargaran un archivo similar a "mod_wsgi-win32-ap22py27-3.3.so"la version que descarguen de mod_wsgi depende como se ve, de la plataforma asi como de la version de python que correrar en el servidor. luego por cuestiones de practicidad renombraremos el archivo de la siguiente manera:

```
mod_wsgi-win32-ap22py27-3.3.so -> mod_wsgi.so
```

Realizado dicho cambio copiamos el modulo dentro de la siguiente carpeta: APACHE_FOLDER\modules\APACHE_FOLDER vendria a ser el directorio donde tenemos la instalacion de WAMP en mi caso es: C:\Apache.

2.7.3. Cargando el Modulo en Apache

Una vez que el módulo de Apache ha sido instalado en el directorio de módulos de su instalación de Apache, todavía es necesario configurar Apache para cargar el módulo en realidad.

Abrimos el archivo "httpd.conf" agregamos la siguiente linea en el mismo punto donde se cargan el resto de los modulos. ⁸

```
LoadModule wsgi_module modules/mod_wsgi.so
```

Con todo esto echo solo tenemos que reiniciar el servidor Apache, en nuestro caso clic en el icono en la barra de notificaciones luego las opciones Apache-¿Service-¿Reiniciar Servicio.

2.7.4. Configuracion del Proyecto

Bueno Ahora solo tenemos que crear un alias en Apache ⁹ para nuestra carpeta donde colocaremos en mi caso la carpeta destino sera:

```
C:\Servidor\SGCM
```

SGCM es la carpeta contenedora del proyecto, y el alias que usaremos sera:

```
/sgcm/
```

tendremos que agregar las siguientes lineas al final del archivo httpd.conf de apache.

⁷<http://www.python.org/dev/peps/pep-0333/>

⁸El archivo httpd.conf esta en la siguiente ruta en el caso de mi instalacion: C:\Apache\conf\httpd.conf

⁹Para mayor informacion de como crear alias en Apache consulte <http://httpd.apache.org/docs/2.2/urlmapping.html>

```
Alias /sgcm/ "C:/Servidor/SGCM/"
WSGIScriptAlias /sgcm "C:/Servidor/SGCM/handle.wsgi"

<Directory "C:/Servidor/SGCM">
    Options Indexes FollowSymLinks MultiViews
    AllowOverride all
    Order allow,deny
    Allow from all
</Directory>
```

Hay un número de maneras en que una aplicación WSGI organizada por `mod_wsgi`¹⁰ puede montarse contra una URL específica. Estos métodos son similares a cómo se podría configurar las aplicaciones CGI tradicionales.

El principal enfoque implica declarar explícitamente en el archivo de configuración principal de Apache el punto de montaje URL y una referencia al archivo de comandos de aplicaciones WSGI. En este caso, el mapeo se fija, con cambios sólo ser capaz de ser hecho mediante la modificación de la configuración principal de Apache y reiniciar Apache.

Al utilizar `mod_cgi` para alojar aplicaciones CGI, esto se haría mediante la directiva `ScriptAlias`. Para `mod_wsgi`, la directiva en su lugar se llama `WSGIScriptAlias`.

```
WSGIScriptAlias /wsgi "C:/Servidor/SGCM/handle.wsgi"
```

Esta directiva sólo puede aparecer en los principales archivos de configuración de Apache. La directiva se puede utilizar en el ámbito del servidor, pero normalmente se coloca en el contenedor `VirtualHost` para un sitio en particular. No se puede utilizar en cualquiera de las directivas de contenedores ubicación, directorios o archivos, ni puede ser utilizada dentro de un archivo `".htaccess"`. El primer argumento de la directiva `WSGIScriptAlias` debe ser el punto de montaje URL para la aplicación WSGI. En este caso, la URL no debe contener una barra diagonal. La única excepción a esto es si la aplicación WSGI es para ser montado en la raíz del servidor web, en cuyo caso `/` sería utilizado.

El segundo argumento de la directiva `WSGIScriptAlias` debe ser una ruta absoluta para el archivo de comandos de aplicaciones WSGI. Es en este archivo que la muestra de código de la aplicación WSGI debe colocarse.

Tenga en cuenta que una ruta absoluta debe ser utilizado para el archivo de comandos de aplicaciones WSGI suministrado como segundo argumento. No es posible especificar una aplicación por sí sola Python nombre de módulo. Una ruta de acceso completa se utiliza para una serie de razones, la principal de las cuales por lo que todos los controles de acceso de Apache todavía pueden aplicarse para indicar que en realidad puede acceder a la aplicación WSGI.

Porque se aplicarán los controles de acceso de Apache, si la aplicación WSGI se encuentra fuera de los directorios que ya están configurados para ser accesible a Apache, habrá que decirle a Apache que los archivos dentro de ese directorio se pueden utilizar. Para ello se debe utilizar la directiva `Directory`.

Hasta aquí tenemos `mod_wsgi` y nuestro directorio listo, ahora probaremos que todo va bien para ello dentro del directorio crearemos un archivo llamado `"handle.wsgi"` que tendrá el siguiente contenido:

```
1 # -*- coding: utf-8 -*-
2
3 import os, sys
4 import django.core.handlers.wsgi
5
6 sys.path.append('C:/Servidor/SGCM')
7 sys.path.append('C:/Servidor')
8
9 os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
```

¹⁰Puede consultar <https://code.google.com/p/modwsgi/wiki/QuickInstallationGuide> si desea explorar otras opciones de configuración.


```
10  
11 application = django.core.handlers.wsgi.WSGIHandler()
```

Con esto nuestro servidor de aplicacion ya deberia funcionar aunque como veran no se cargan los archivos estaticos como imagenes y hojas de estilo por lo que necesitamos agregarlo.

Apéndices

