Sistemas Web Desconectados

Release 1

van Haaster, Diego Marcos; Defossé, Nahuel

September 22, 2009

Índice general

CAPÍTULO 1

Introduccion

"Yo sólo puedo mostrarte la puerta. Tú eres quien debe atravesarla."

—Morfeo

1.1 Motivación

Hoy más que ayer, pero seguremente menos que mañana, Internet es "la red de redes". El alto contenido de información

Hoy en día Internet supone más que un medio de intercambio de información, su constante expansión la ha convetido un terreno muy atractivo para la implementación de sistemas de información.

Las tareas de mantenimiento y actualización de aplicaciones son posibles sin necesidad de distribuir software adicional a miles de usuarios potenciales.

Con la necesidad de acelerar y estandarizar la forma en la que se desarrollan las aplicaciones web han aparecido plataformas de desarrollo, o frameworks, para la mayoría de los lenguajes. A lo largo de los últimos 10 años ha existido una importante evolución de los mismos. Muchas tareas comunes son resueltas de una manera predefinida y modificable, permitendo al programador focalizarse en resolver el problema particular de su desarrollo.

Una ventaja significativa de las aplicaciones web es que funcionan independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para los múltiples sistemas operativos, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Las aplicaciones web tienen ciertas limitaciones en las funcionalidades que ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio, como dibujar en la pantalla o arrastrar y soltar, que no están soportadas por las tecnologías web estándar. Los desarrolladores web, generalmente, utilizan lenguajes interpretados o script en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez.Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con

tecnologías en el lado del servidor. Los sistemas operativos actuales de propósito general cuentan con un navegador web, con posibilidades de acceso a bases de datos y almacenamiento de código y recursos. La web, en el ámbito del software, es un medio singular por su ubicuidad y sus estándares abiertos. El conjunto de normas que rigen la forma en que se generan y transmiten los documentos a través de la web son regulados por la W3C (Consorcio World Wide Web). La mayor parte de la web está soportada sobre sistemas operativos y software de servidor que se rigen bajo licencias OpenSource1 (Apache, BIND, Linux, OpenBSD, FreeBSD). Los lenguajes con los que son desarrolladas las aplicaciones web son generalmente OpenSource, como e PHP, Python, Ruby, Perl y Java. Los frameworks web escritos sobre estos lenguajes utilizan alguna licencia OpenSource para su distribución; incluso frameworks basados en lenguajes propietarios son liberados bajo licencias OpenSource.

1.2 La word wide web

En 1994, el CERN y el MIT establecen el *Word Wide Web Consortium* (W3C), con la finalidad de promover el desarrollo de la Web, estandarizar protocolos y fomentar la interoperabilidad entre distintos sitios. El sitio oficial del consorcio es <http://www.w3.org>_ y en este se encuentran los principales documentos sobre la Web e información sobre la actividad de la organización.

1.3 La arquitectura web

1.4 Objetivos

Podemos decir que las aplicaciones tradicionales, que no hacen uso de la web, son más robustas ya que no dependen de una conexión. Por lo tanto, sería deseable poder dotar a las aplicaciones web de la capacidad de trabajar cuando no cuentan con conexión. Si bien los elementos necesarios para llevar a cabo esta tarea están disponibles actualmente, no están contemplados en los diseños de los frameworks web. Es decir, cuando una determinada aplicación web debe ser transportada al cliente, es necesario escribir el código de soporte específico para esa aplicación. Un framework no constituye un producto per sé, sino una plataforma sobre la cual construir aplicaciones. Consideramos que sería beneficioso aportar una extensión a un framework web OpenSource que brinde facilidades para transportar las aplicaciones web, basadas en éste, al cliente de manera que la aplicación que haga uso de nuestra extensión pueda ser ejecutada a posteriori en el navegador en el cual ha sido descargada. El framework web será elegido tras un estudio de las características que consideramos más importantes para el desarrollo veloz, como la calidad del mapeador de objetos (entre las características más importantes de éste buscaremos eficiencia en las consultas a la base de datos, ejecución demorada para encadenamiento de consultas, implementación de herencia, baja carga de configuración), la simplicidad para enlazar url's a funciones controladoras, extensibilidad del sistema de escritura de plantillas. Buscaremos frameworks que permitan la ejecución

transversal de cierto tipo de funciones, para ejecutar tareas como compresión de salida, sustitución de patrones de texto, caché, control de acceso, etc.

La World Wide Web, o web, durante los últimos años ha ganado terreno como plataforma para aplicaciones del variado tipo. Diversas tecnologías fueron formuladas para convertir el escenario inical, donde la web se limitaba a ser una gran colección de documentos enlazados (hipertexto), para llegar a ser...

Vamos a realizar un breve análisis sobre las tecnologías que son utilizadas en la web.

Luego un análisis de las tecnologías del cliente, haciendo hincapié en ...

En los primeros días de la Web, todo el contenido era estático, los clinetes accedian mediante URLs a archivos físico del sistema de archivos del servidor. Sin emabrgo, en los años recientes, cada vez mas conenido es diámico, es decir, se genera a solicitud en lugar de estar almancenado en disco. El presente documento se estructura en dos partes que explican como se genera contenido dinamico ya sea en el servidor o en el clinete.

1.5 Alcance

Aca ponemos hasta donde nos vamos a llegar.

CAPÍTULO 2

Tecnologías del servidor

2.1 Generación dinámica de páginas Web

Nota: Introduccion de la mano de Tanenbaum La parte estatica de la cuestion dejarla para la introduccion como cuentito, aca vamos a los bifes dinamicos.

Esta sección tiene como finalidad introducir los conceptos básicos concernientes a la *generación dinamica contenido* en el servidor.

En el enfoque dinámico, cuando un usuario realiza una solicitud, el mensaje enviado tiene como objetivo la ejecución de un programa o secuencia de comandos en el servidor. Por lo general, el procesamiento involucra el uso de la información proporcianda por el usuario para buscar registros en una base de datos y genrerar una página HTML personalizada para el cliente.

La forma tradicional de manejar páginas Web interactivas se conoce como *CGI*, éste es un estandar que consiste en delegar la generación de contenido a un programa. CGI se limita a definir la entrada y salida del programa.

Los servidores web primigenios y monolíticos avanzaron a una arquitectura modular [Apache-Mod2009] [MicrosoftIIS2009] , en la cual, un módulo brinda soporte para una tarea específica. Módulos comunes son autenticación, bitácora, balance de carga, entre otros como los de generación de contenido.

Un enfoque más moderno para la generación de contenido dinámico es la incrustación de secuencias de comandos dentro de la páginas HTML. Estas sentencias son leidas y ejecutadas por un módulo del servidor al momento de responder a la solicitud del cliente, como es el caso del lenguaje *PHP*, o *ASP*.

Más recientemente han aparecido mejoras sobre CGI, como FastCGI y SCGI, que suponen una simplificación y reducción de recursos a la hora de satisfacer una petición. Para satisfacer necesidades más complejas, que no fueron abordadas en la genericidad de CGI, se fueron desarrollando alternativas como servidores orientados a aplicaciones, como el servidor de aplicaciones Tomcat

para el lenguaje de progarmación Java [ApacheTomcat2009] [SunServlet2009] o módulos de lenguajes de programación específicos.

2.1.1 Servidor Web

Un servidor web, o *web server* es un software encargado de recibir solicitudes de un cliente, típicamente un *navegador web*, a través del protcolo *HTTP* y generar una respuesta a la solicitud. Mediante la especificación *MIME* que se incluye en el encabezado de la respuesta que es enviada al cliente, se puede identificar que tipo de archivo es devuelto, siendo el tipo más común *HTML* o *XHTML*.

El contenido que es enviado al cliente puede ser de origen *estático* o *dinámico*. El contendio estático es aquél que proviene desde un archivo en el sistema de archivos sin ninguna modificación. El contenido dinámico en contraposición al *contenido estático* es generado por algún programa, un *script* o algún tipo de API invocada por el web server, como SSI, *CGI*, SCGI, FastCGI, JSP, ColdFusion, NSAPI o ISAPI).

El clinete web accede a los recursos del web server mediante una dirección de recurso, u *URL*.

2.1.2 CGI

El estándar Common Gateway Interface (CGI) [1] surge alrededor del año 1998, como un estándar de comunicación o *API* entre un servidor web y un programa de usuario. Al permitir ejecutar un proceso, CGI permite generar contenido de manera dinámica a traves de un servidor web.

Es decir, el usuario puede recuperar datos de un programa que se ejecuta en el equipo en el que reside el servidor web, donde la salida puede ser un documento HTML entendible para el navegador, o culaquier otro tipo de archivo, como imágenes, contenido multimedial, sonidos, etc.

CGI fue la primera estandarización de un mecanismo para generar *contenido dinámico* en la web.

Los mecanismos de comunicación de entrada son las variables de entorno y la entrada estandar, mientras que para la salida se utiliza la salida estandard del proceso.

Los parámetros HTTP, como la *URL*, el método (GET, POST, PUSH, etc.), nombre del servidor o host, puerto, etc.) e información sobre el servidor son trasferidos a la aplicación CGI como variables de entorno.

Si existiese un cuerpo en la petición HTTP, como por ejemplo, el contenido de un formulario, bajo el método POST, la aplicación CGI accede a esta como entrada estandard.

El resultado de la ejecución de la aplicación CGI se escribe en la salida estándard, anteponiendo las cabeceras HTTP respuesta, para que el servidor responda al clinete. En los encabezados de respuesta, el tipo MIME determina como interpreta el clinete la respuesta. Es decir, la invocación

[1] A veces traducido como pasarela común de acceso.

de un CGI puede devolover diferentes tipos de contenido al clinete (html, imágenes, javascript, contenido multimedia, etc.)

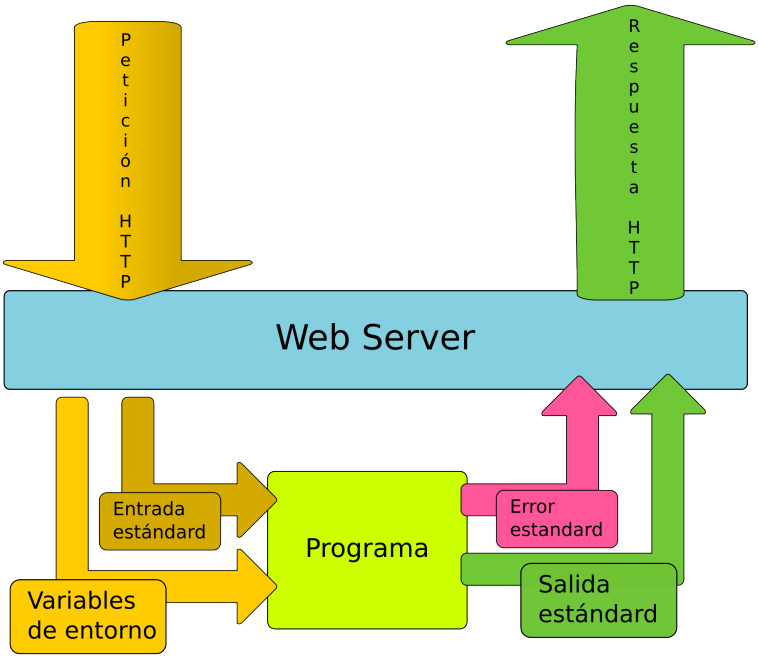


Figura 2.1: Poceso de una solicitud con CGI.

Dentor de las varaibles de entorno, la Wikipedia [WikiCGI2009] menciona:

- **QUERY_STRING** Es la cadena de entrada del CGI cuando se utiliza el método GET sustituyendo algunos símbolos especiales por otros. Cada elemento se envía como una pareja Variable=Valor. Si se utiliza el método POST esta variable de entorno está vacía.
- **CONTENT_TYPE** Tipo MIME de los datos enviados al CGI mediante POST. Con GET está vacía. Un valor típico para esta variable es: Application/X-www-form-urlencoded.
- **CONTENT_LENGTH** Longitud en bytes de los datos enviados al CGI utilizando el método POST. Con GET está vacía.
- **PATH_INFO** Información adicional de la ruta (el "path") tal y como llega al servidor en el URL.
- **REQUEST_METHOD** Nombre del método (GET o POST) utilizado para invocar al CGI.
- **SCRIPT_NAME** Nombre del CGI invocado.
- **SERVER_PORT** Puerto por el que el servidor recibe la conexión.
- **SERVER_PROTOCOL** Nombre y versión del protocolo en uso. (Ej.: HTTP/1.0 o 1.1)

Variables de entorno que se intercambian de servidor a CGI:

- **SERVER_SOFTWARE** Nombre y versión del software servidor de www.

- **SERVER_NAME** Nombre del servidor.
- **GATEWAY_INTERFACE** Nombre y versión de la interfaz de comunicación entre servidor y aplicaciones CGI/1.12

Debido a la popularidad de las aplicaciones CGI, los servidores web incluyen generalmente un directorio llamado **cgi-bin** donde se albergan estas aplicaciones.

Nota: Faltan referencias sobre la popularidad de los lenguajes

Históricamente las aplicaciones CGI han sido escritas en lenguajes interpretados, sinedo muy popular Perl y más recientemente el lenguaje PHP. En los *lenguajes interpretados*, el código ejecutable es texto plano, por lo que puede ser editado en una terminal directamente en el servidor. Otra ventaja importante de ser texto plano, es que es más sencillo de mantener con alguna herramienta de :term:SCM.

2.1.3 Lenguajes de programación para la web

Nota: Acá introducimos los conceptos de

- Propósito General vs DSL
- Compilado vs Intepretado, -> VMs
- Dinámico vs Estático
- Plataforma Específica vs Multiplataforma
- Porgramación de sistema y programación de aplicaciones

Una de las clasificaciones más generales que se suelen realizar sobre los lenguajes de programación es identificar su objetivo. Los lenguajes de programación de *propósito general* están orientados a resolver cualquier tipo de problema, mientras que los lenguajes de *propósito específico* o *DSL* están enfocados en reolver un tipo de problemas de manera más eficáz. Un ejemplo muy popular de DSL es la planilla de cálculo Microsoft Excel [DavidPollak2006].

Además de la división entre lenguaje de porposito general y DSL, exiten otras clasificaciones de interés en el estudio de los lenguajes de programación populares en la Web, como la divsión entre leguajes interpretados y compilados.

Un lenguaje de programación interpretado es aquel en el cual los programas son ejecutados por un intérprete, en vez de realizarse una traducción a lenguaje máquina, conocido como proceso de compilación.

En teoría cualquier lenguaje de programación podría ser compilado o interpretado.

En un lengauje de programación interpretado se puede decir que el código fuente es el código ejecutable (a través del intérprete). Para llegar a la ejecución de un programa escrito en un lenguaje compilado, se debe pasar generalmente por dos etapas, una de compilación, donde se traducen las sentencias del lenguaje por código máquina y otra de enlace, donde se ensambla el código objeto resultado de la compilación y se resuleven los enlaces entre los diferentes módulos compilados.

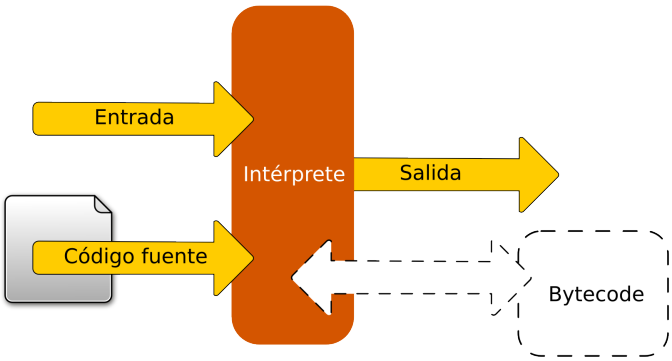


Figura 2.2: Lenguaje interpretado.

Existe un mecanismo intermedio de ejecución, que se conoce como máquina virtual, en el cual, existe un proceso de compilación de el código fuente a un lenguaje intermedio, comunmente denominado *bytecode*. Este bytecode es luego ejecutado sobre un intérprete, al cual se denomina comunmente como máquina virutal. La traducción del código fuente a *bytecode* puede ser explícita, como en el lenguaje Java, o implícita como en Python, donde se mezcla en el intérprete la funcionalidad de compilación a bytecode e interpretación en un solo programa.

Los lenguajes de programación interpreteados suelen ser de alto nivel y de *tipo dinámico*, es decir que la mayoría de las comprobaciones que se realizan en tiempo de compilación en otros lenguajes [2] , se realizan en tiempo de ejecución.

Si un lenguaje provee la posibilidad de ser ejecutado en varias plataformas [3] decimos que es multiplataforma. En el caso de los lenguaje compilados, es necesario disponer de un compilador para la plataforma, mientras que en un lenguaje interpretado, es necesario compilar el intérprete con el compilador de la plataforma.

Un lenguaje dinámico admite de manera directa en tiempo de ejecución, el agregado de código, extensión o redefinición de objetos o hasta inclusive modificar tipos de datos. Si bien estas tareas pueden ser realizadas en lenguajes no dinámicos, su implementación no es sencilla.

Jhon Cownan [Cowan2005] adempas hace una clasificación interesante sobre la utilización de los lenguajes de programación, diferenciando la *programación de sistema* de la *programación de aplicaciones*.

- *Programación de sistemas*

  En la programación de sistemas, el programador se centra en adaptar un nuevo dominio a interfases genéricas preexistentes.

- *Programación de aplicaciones*

[2] Estas comprobaciones comprenden el checkeo de tipos de datos, resolución de métodos entro otros.

[3] *Una plataforma es una combinación de hardware y software usada para ejecutar aplicaciones; en su forma más simple consiste únicamente de un sistema operativo, una arquitectura, o una combinación de ambos. La plataforma más conocida es probablemente Microsoft Windows en una arquitectura x86* [WikiPlataforma2009]

> En la programación de aplicaciones, por el contrario, modela el dominio de la aplicación de manera específica como paso inicial (análisis y diseño) y luego de la integración (si existe).

Podemos ejemplificar la programación de sistemas con la creación de un shellscript por parte de un administrador de sistemas basados en UNIX, donde existe un legnguaje de integración, shellscript [4] y las un set de utilidades de sistema con interfase conocida que le brindan funcionalidades como ordenamiento de caddenas, busqueda y reemplazo de patrones, inicio y detencción de procesos, interacción con el SO, etc.

Java es un lenguaje de progamación orientado a objetos, multiplataforma y multiproposito, basado en máquina virtual, de compilación explícita. Java ha definido algunos elementos importantes en cuanto a la web dinámica, como los applets [5] y la especificación J2EE.

Perl es un lenguaje de programación de propósito general diseñado por Larry Wall en 1987. Perl toma características del lenguaje C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación. Su uso principal es el procesamiento de texto, siendo muy popular en programación de sistemas. Muchos sistemas basados en CGI están escritos en Perl (sistemas de administración de servidores, correo, etc.). Perl esta disponible para muchas plataformas, incluyendo todas las variantes de UNIX.

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto.

Su sintaxis es compleja y ambigua, una tarea puede ser realizada de muchas maneras diferentes, dado lugar a confusión.

PHP es un lenguaje interpreteado, originalmente diseñado para ser embebido dentro del código HTML y procesado en el servidor (DSL) y que con los años ha intentado convertirse en un lenguaje de porpósito general. Toma elementos de Perl y shellscript, C, y recientemente Java. Cuando el cliente solicita a través de una URL un módulo php, el interprete busca los los tags del lenguaje, y los reemplaza por la salida del bloque. Presenta una ventaja importante sobre la escritura de CGI, ya que no es necesario confeccionar un programa de usuario, la resolución de URLs está dada por la estructura del sistema de archivos. Si bien es muy popular [6] y está disponible en la gran mayoría de los servidores UNIX, simplificando el *deployment* es criticado por no poseer ámbito de nombres para los módulos, promover el código desordenado, de dificil optimización y orientado a la web [BlogHardz2008] . Hoy es posible escribir aplicaciones gráficas mediante toolkits como Qt y GTK.

Ruby es un lenguaje orientado fuertemente objetos, multiplataforma, creado en 1995 por Yukihiro "Matz" Matsumoto, en Japón. A menudo comparado con *Smalltak*, se suele decir que Ruby es un lenguaje de objetos puro, ya que *todo* es un objeto. Posee muchas caracteríticas avanzadas como metaclases, clausuras, iteradores, integración de expresiones regulares en el lenguje, etc. Su sintaxis toma elementos de Perl, por lo cual, suelen existir convenciones de nombrado a las que

[4] Bash, Sh, Csh, Ksh, Zsh, etc.
[5] Pequeños programaas que se ejecutan en el navegaor web
[6] Utilización de PHP según php.net

cuesta acostumbrarse. Existen varios intérpretes de Ruby, siendo la oficial escrita en C, se concen YARV [7] , JRuby [8] , Rubinius [9] , IronRuby [10] , y MacRuby [11] .

Una de las Killer App [12] según el autor es el framework para contenido web denominado "Ruby on Rails" que alcanzó su versión 1.0 en el año 2005, que poroponía un cambio radical al enfoque complejo de *J2EE*.

Su principales desventajas son la baja aceptación, quizás debido a que la documentación oficial solía estar en idioma Japonés (auque la situación se ha venido revirtiendo ultimamente). Otra desventaja importante es que la velocidad del intérprete oficial es bastante baja cuando se la compara con otros lenguajes de programación y muestra variaciones importantes entre plataformas. **Python** es un lenguaje de programación interpretado multiparadigma, de propósito general. Fue creado por por Guido van Rossum en el año 1991. Se trata de un lenguaje dinámico, y toma elmentos de varios lenguajes, como C, Java, Scheme, entre otros. Gracias a su naturaleza dinámica, toda resolución de nombres (o símbolos) se realiza en tiempo de ejecución (*dynamic binding*). En la jerga del lenguaje se suele llamar a este hecho, *duck typing* y se lo asocia con el siguiente broma, *si tiene pico y hace cuac, se trata de un pato*.

Python puede ser extendido meidante módulos escritos en C o C++, y también se puede embeber el intérprete en otros lenguajes. Python permite actualmente cargar bibliotecas de enlace dinámico, de manera dinámica y realizar llamadas incluso con callbacks escritos en Pyhton.

Python es considerado por parte de la comunidad UNIX, como una evolución de Perl, de sintaxis limpia y potente. Un caso sitable de esta "evolución" es el árticulo escrito por Eric Raymond, "Why Python?", donde explica su conversión de Perl a Python [EricRaymon2000].

Muchos de las sistemas webs basados en CGI, están escritos en Perl, por lo cual no es sorpresa encontrar una buena cantidad de proyectos del lenguaje Python orientados a la Web [13].

2.1.4 Selección de un lenguaje de programación

Jhon Crockford .. Con la masificación de la World Wide Web se hizo necesario encontrar formas de desarrollo

En el *apéndice* se encuentra una referecia detallada del lenguaje.

[7] Yet Another Ruby VM, escrita por Sacasada Kiochi http://www.atdot.net/yarv/

[8] JRuby es una máquina virtual de Ruby escrita sobre la máquina virtual de **Java**, http://jruby.codehaus.org/

[9] Rubinius es una máquina vritual de Ruby escrita en **C++** http://rubini.us/

[10] IronRuby es una implementación de Ruby sobre la plataforma **.Net** http://www.ironruby.net/

[11] MacRuby es una implementación de Ruby sobre **Objective-C** para el sistema Mac OS X, http://www.macruby.org/

[12] Aplicación que populariza un lenguaje.

[13] En el repositorio de proyectos del lenguaje se encuentran más 1100 resultados para paquetes relacionados con el término "web". http://pypi.python.org/pypi?%3Aaction=search&term=web&submit=search

WSGI

WSGI o Web Server Gateway Interfase es una especidicación para que un web server y una aplicación se comuniquen. Es un estándar del lenguaje Python, descrito en el PEP [14] 333. Si bien WSGI es similar en su concepción a CGI, su objetivo es estandarizar la aparición de estructras de software cada vez más complejas (frameworks *servidor-frameworks_*)

Python, son albergados en el sitio oficial http://www.python.org

WSGI propone que una aplicación es una función que recibe 2 argumentos. Como primer argumento, un diccionario con las variables de entorno, al igual que en CGI, y como segundo argumento una función (u *objeto llamable*) al cual se invoca para iniciar la respuesta.

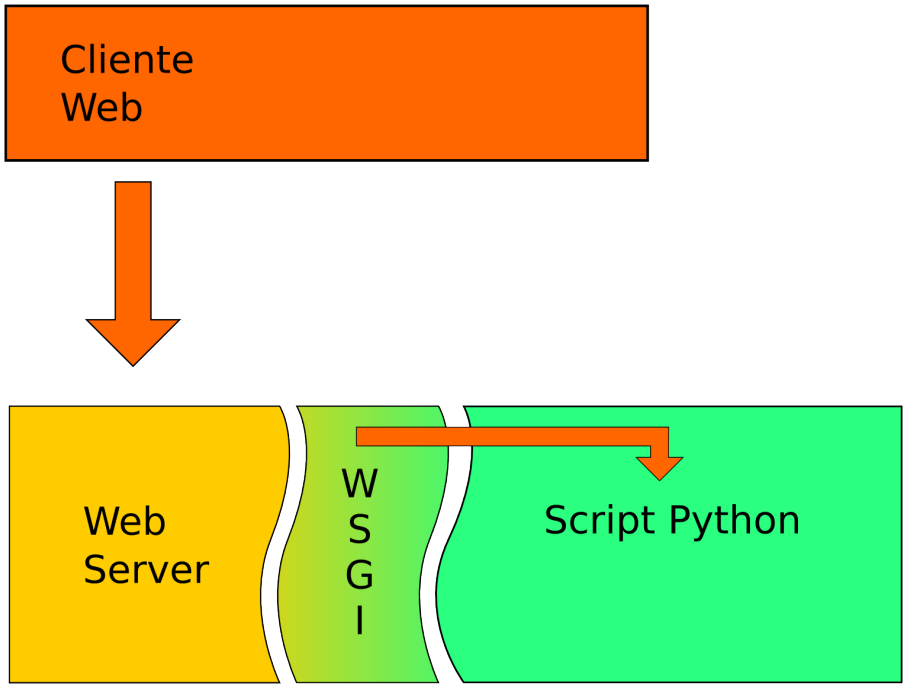


Figura 2.3: Esquema WSGI.

En el siguente ejemplo, la función app devuelve *Hello World* informandole al navegador web, que el contenido se trata de texto plano.

[14] PEP *Python Enhancement Proposals* son documentos en los que se proponen mejoras para el lenguaje

```
def app(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return ['Hello World\n']
```

2.2 Frameworks Web

2.2.1 Frameworks

Nota: Poner CLI

Según la la wikipedia [WIK001] un framework de software es *una abstracción en la cual un código común, que provee una funcionalidad genérica, puede ser personalizado por el programador de manera selectiva para brindar una funcionalidad específica*.

Además agrega que los frameworks son similares a las bibliotecas de software (a veces llamadas librerías) dado que proveen abstracciones reusables de código a las cuales se accede mediante una API bien definida. Sin embargo, existen ciertas características que diferencian al framework de una librería o aplicaciones normales de usuario:

- Inversion de control

  Al contrario que las bibliotecas en las aplicaciones de usuario, en un framework, el flujo de control no es manejado por el llamador, sino por el framework. Es decir, cuando se utilizan bibliotecas o programas de usuario como soporte para brindar funcionalidad, estas son llamados o invocados en el código de aplicación principal que es definido por el usuario. En un framework, el flujo de control principal está definido por el framework.

- Comportamiento por defecto definido

  Un framework tiene un comportamiento por defecto definido. En cada componete del framework, existe un comportamineto genérico con alguna utilidad, que puede ser redefinido con funcionalidad del usuario.

- Extensibiliad

  Un framework suele ser extendido por el usuario mediante redefinición o especialización para proveer una funcionalidad específica.

- No modificabilidad del código del framework

  En general no se permite la modificación del código del framework. Los programadores pueden extender el framework, pero no modificar su código.

Los diseñadores de frameworks tienen como objetivo facilitar el desarrollo de software, permitiendo a los programadores enfocarse en cumplimentar los requermientos del análisis y diseño, en vez

de dedicar tiempo a resolver los detalles comunes de bajo nivel. En general la utilización de un framework reduce el tiempo de desarrollo.

Por ejemplo, en un equipo donde se utiliza un framework web para desarrollar un sitio de banca electrónica, los desarrolladores pueden enfocarse en la lógica necesaria para realizar las extraciones de dinero, en vez de la mecánica para preserver el estado entre las peticiones del navegador.

Sin embargo, se suele argumentar que los frameworks pueden ser una carga, debido a la complejidad de sus APIs o la incertidumbre que generar la existencia de varios frameworks para un mismo tipo de aplicación. A pesar de tener como objetivo estandarizar y reducir el tiempo de desarrollo, el aprendizaje de un framework suele requerir tiempo extra en el desarrollo, que debe ser tenido en cuenta por el quipo de desarrollo. Trás completar el desarrollo en un framework, el equipo de desarrllo no debe volver a invertir tiempo en aprendizaje en sucesivos desarrollos.

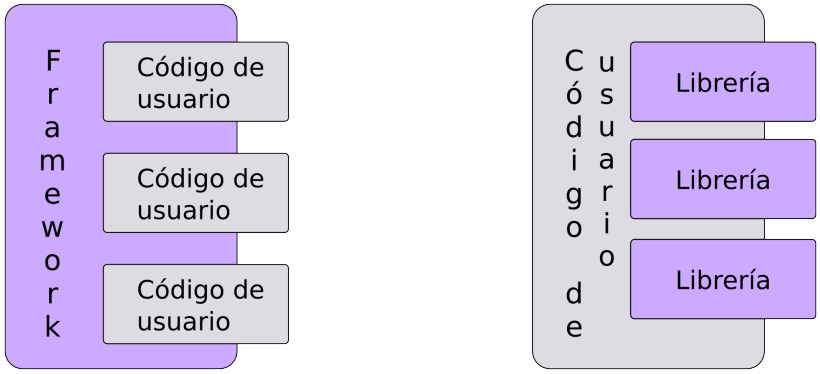


Figura 2.4: Libreria vs Framework.

Nota: Breve introducción a patrones arquitecturales y mvc

2.2.2 Model View Controler

En aplicaciones complejas que impliquen sofisticadas interfaces, como las aplicaciones web, la lógica de la interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Si se realiza un diseño mezclando los componentes de interfaz y de negocio, entonces las consecuencias serán que, cuando se necesite cambiar la interfaz, se tendra que modificar trabajosamente los componentes de negocio, teniendo de esta forma mayor trabajo y mayor riesgo de error.

El patrón arquitectural MVC, *Modelo Vista Controlador* trata de realizar un diseño que desacople la interfaz o vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.

Este patrón fue descrito por primera vez en 1979 por Trygve Reenskaug [Tryg1979], entonces trabajando en Smalltalk en laboratorios de investigación de Xerox. La implementación original está descrita a fondo en Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador [SmallMVC].

Descripción del patrón:

- Modelo

  Esta es la capa de datos, una representación de la información con la cual el sistema opera. La lógica de datos asegura la integridad y permite derivar nuevos datos.

- Vista

  Esta es la capa de presentación del modelo, seleccionando qué mostrar y cómo mostrarlo, usualmente la interfaz de usuario.

- Controlador

  Esta capa responde a eventos, usualmente acciones del usuario, e invoca cambios en el modelo y probablemente en la vista.

El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el sistema de gestión de base de datos y la lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Framework Web

Nota: Ver diferencia entre sitio y aplicación http://www.javahispano.org/contenidos/es/comparativa_de_framewor

Un framework Web, es un framework de software que permite implementar aplicaciones Web, brindando soporte para tareas comunes. Su objetivo es facilitar el desarrollo de aplicaciones Web.

Para entender mejor esto, se presenta a continuación un ejemplo de aplicación Web escrita usando CGI; este fragmento de código en Python, muestra los diez libros más recientemente publicados de una base de datos:

```
#!/usr/bin/python

import MySQLdb

# Se imprime el Content-Type y una linea en blanco, tal como requiere CGI
```

```
print "Content-Type: text/html"
print

# Algo de HTML introductorio que de forma al documento
print "<html><head><title>Libros</title></head>"
print "<body>"
print "<h1>Los ultimos 10 libros</h1>"
print "<ul>"

# Contectar con la base de datos y obtener los utlimos libros publicados
conexion = MySQLdb.connect(user='yo', passwd='dejame_entrar', db='mi_base')
cursor = conexion.cursor()
cursor.execute("SELECT nombre FROM libros ORDER BY fecha_pub DESC LIMIT 10")

# Iterar sobre la lista de libros e imprimir HTML con los datos obtenidos
for fila in cursor.fetchall():
    print "<li> %s</li>" % fila[0]

# Cerrar el documento HTML
print "</ul>"
print "</body></html>"

# Cerrar la conexion con la base de datos
conexion.close()
```

Si bien el código es sencillo de comprender y utilizar, cuando la aplicación Web comienza a crecer más allá de lo trivial, este enfoque se desmorona y surgen una serie de problemas:

- ¿Qué sucede cuando múltiples páginas necesitan conectarse a la base de datos? Seguro que ese código de conexión a la base de datos no debería estar duplicado en cada uno de los scripts CGI, así que la forma pragmática de hacerlo sería refactorizarlo en una función compartida.
- ¿Debería un desarrollador *realmente* tener que preocuparse por imprimir la línea de "Content-Type" y acordarse de cerrar la conexión con la base de datos? Este tipo de código repetitivo reduce la productividad del programador e introduce la oportunidad para que se cometan errores. Estas tareas de configuración y cierre estarían mejor manejadas por una infraestructura común.
- ¿Qué sucede cuando este código es reutilizado en múltiples entornos, cada uno con una base de datos y contraseñas diferentes? En ese punto, se vuelve esencial alguna configuración específica del entorno.
- ¿Qué sucede cuando un diseñador Web que no tiene experiencia programando en Python desea rediseñar la página? Lo ideal sería que la lógica de la página – la búsqueda de libros en la base de datos – esté separada del código HTML de la página, de modo que el diseñador pueda hacer modificaciones sin afectar la búsqueda.

Precisamente estos son los problemas que un framework Web intenta resolver. Un framework Web provee una infraestructura de programación para las aplicaciones, para que el desarrollador se pueda concentrar en escribir código limpio y de fácil mantenimiento sin tener que reinventar la rueda.

En Wikipeida [WIKI002]

- **Acceso a datos** ORM
- Seguridad
- Mapeo de URLs
- Sistema de plantillas
- Caché
- AJAX
- Configuración mínima y simplificada

Nota: completar tema framework un poco, antes de caer en django

2.2.3 Mapeador Objeto-Relacional

En las aplicaciones modernas, la lógica arbitraria a menudo implica interactuar con una base de datos. Detrás de escena, un *programa impulsado por una base de datos* se conecta a un servidor de base de datos, recupera algunos datos de esta, y los presenta al usuario con un formato agradable para su interpretacion. Una aplicacion web no escapa a esta aseveración, solo que presenta los datos representados en HTML, así mismo un sitio puede proporcionar funcionalidad que permita a los visitantes del sitio poblar la base de datos por su propia cuenta.

Amazon.com, por ejemplo, es un buen ejemplo de un sitio que maneja una base de datos. Cada página de un producto es esencialmente una consulta a la base de datos de productos de Amazon formateada en HTML, y cuando se envian datos al servidor, como opiniónes de cliente, estos son insertadados en la base de datos de opiniones.

La forma simple de interactuar con una base de datos, es mediante el uso de bibliotecas provistas por los lenguajes para ejecutar consultas SQL y una vez obtenidos los datos, procesarlos.

En este ejemplo se usa la biblioteca `MySQLdb` para conectar con una base de datos MySQL, recuperar algunos registros:

```
import MySQLdb

db = MySQLdb.connect(user='me', db='mydb', passwd='secret', host='localhost')
cursor = db.cursor()
cursor.execute('SELECT name FROM books ORDER BY name')
names = [row[0] for row in cursor.fetchall()]
db.close()
```

Este enfoque funciona, pero presenta algunos problemas:

- **Los parametros de la conexión a la base de datos estan codificandos** *en duro* (*hard-coding*).
- **Se debe escribir una cantidad de código estereotípico: crear una** conexión, un cursor, ejecutar una sentencia, y cerrar la conexión.
- **Ata a las aplicaciones a MySQL. Si, en el camino, se quiere cambiar MySQL** por PostgreSQL por ejemplo, se deben alterar todas las lineas que hagan falta para la nueva biblioteca o conector, parámetros de conexión, posiblemente reescribir el SQL, etc.

Por otro lado y quiza mas importante a la hora de desarrollar un programador que trabajé con programación orientada a objetos y bases de datos relacionales, debe realizar un cambio de contexto cada vez que requiera interactuar con la base de datos, escribiendo consultas en SQL y luego lidear con los resultados obtenidos de las consultas entre los objetos. Este *cambio de contexto* es debido a una diferencia que existe entre los dos paradigmas involucrados. Mientras que el modelo relacional trata con relaciones, conjuntos y la logica matemática correspondiente, el paradigma orientado a objetos trata con objetos, atributos

y asociaciones de unos con otros. Tan pronto como se quieran persistir los

objetos utilizando una base de datos relacional esta desaveniencia resulta evidente.

Las primeras aproximaciones al mapeo relacional de objetos, surgen de convertir los valores de los objetos en grupos de valores simples para almacenarlos en la base de datos (y volverlos a convertir luego de recuperarlos de la base de datos). Sin embargo, esta traduccion simple dista mucho del concepto de *objetos persistentes*, la idea de estos es la traducción automatica de objetos en formas almacenables en la base de datos y su posterior recuperación conservando las propiedades y las relaciones ente los mismos.

Con la finalidad de lograr *objetos persistentes* un buen número de sistemas de mapeo objeto-relacional se han desarrollado a lo largo de los años y aunque su efectividad es muy discutida la realidad es que estos permiten agilizar el proceso de desarrollo, paleando mucho de los problemas presentados con anterioridad.

Desde el punto de vista de un programador, un ORM debe lucir como un almacén de objetos persistentes. Uno puede crear objetos y trabajar normalmente con ellos, los cambios que sufran terminarán siendo reflejados en la base de datos.

2.3 Django

2.3.1 Introducción

Nota: Acá tenemos que justificar por que django

Django es un framework web escrito en Python el cual sigue vagamente el concepto de Modelo Vista Controlador. Ideado inicialmente como un adminsitrador de contenido para varios sitios de

noticias, los desarrolladores encontraron que su CMS era lo sufcientemente genérico como para curbir un ámbito más aplio de aplicaciones.

En honor al músico Django Reinhart, fue liberado el código base bajo la licencia *BSD* en Julio del 2005 como Django Web Framework. El slogan del framework fue "Django, Él framework para perfeccionistas con fechas límites" [15].

En junio del 2008 fue anuncidada la cereación de la Django Software Fundation, la cual se hace cargo hasta la fecha del desarrollo y mantenimiento.

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos que puede incluirse como parte de cualquier proyecto; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno (sistema de logging o bitácora), y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

Con Django también se distribuyen aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido via RSS y/o Atom, "páginas planas" que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

Django como framework de desarrollo consiste en un conjunto de utilidades de consola que permiten crear y manipular proyectos y aplicaciones. Este sigue el patrón MVC y como el controlador "C" es manejado por el mismo sistema los desarrolladores dieron a conocer a Django como un *Framework MTV*.

- *M* significa "Model" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- *T* significa "Template" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento.
- *V* significa "View" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelos y las plantillas.

MVC o MTV la realidad es que ninguna de las interpretaciones es más "correcta" que otra. Lo importante es entender los conceptos subyacentes.

Nota: Justificacción

- completitud

[15] Del ingles "The Web framework for perfectionists with deadlines"

- popularidad
- simplicidad

2.3.2 Estructuración de un proyecto en Django

Durante la instalación del framework en el sistema del desarrollador, se añade al PATH un comando con el nombre django-admin.py. Mediante este comando se crean proyectos y se los administra.

Un proyecto se crea mediante la siguiente orden:

```
$ django-admin.py startproject mi_proyecto # Crea el proyecto mi_proyecto
```

Un proyecto es un paquete Python que contiene 3 módulos:

- **manage.py** Interfase de consola para la ejecución de comandos
- **urls.py** Mapeo de URLs en vistas (funciones)
- **settings.py** Configuración de la base de datos, directorios de plantillas, etc.

En el ejemplo anterior, un listado gerárquico del sistema de archivos mostraría la siguiente estructura:

```
mi_proyecto/
    __init__.py
    manage.py
    settings.py
    urls.py
```

El proyecto funciona como un contenedor de aplicaciones que ser rigen bajo la misma base de datos, los mismos templates, las mismas clases de middleware entre otros parámetros.

Analicemos a continuación la función de cada uno de estos 3 módulos.

Módulo settings

Este módulo define la configuración del proyecto, siendo sus atributos principales la configuración de la base de datos a utilizar, la ruta en la cual se encuentran los médios estáticos, cuál es el nombre del archivo raíz de urls (generalmente urls.py). Otros atributos son las clases middleware, las rutas de los templates, el idioma para las aplicaciones que soportan *i18n*, etc.

Al ser un módulo del lengauje python, la confguración se puede editar muy facilmente a diferencia de configuraciones realizadas en XML, además de contar con la ventaja de poder configurar en caliente algunos parametros que así lo requieran.

Un parametro fundamental es la lista denominada INSTALLED_APPS que contiene los nombres de las aplicaciones instaladas en le proyecto.

Módulo manage

Esta es la interfase con el framework. Éste módulo es un script ejecutable, que recibe como primer argumento un nombre de comando de django.

Los comandos de django permiten entre otras cosas:

- startapp <nombre de aplicación>

  Crear una aplicación

- runserver

  Correr el proyecto en un servidor de desarrollo.

- syncdb

  Generar las tablas en la base de datos de las aplicaciones instaladas

Módulo urls

Este nombre de módulo aparece a nivel proyecto, pero también puede aparecer a nivel aplicación. Su misión es definir las asociaciones entre URLs y vistas, de manera que el framework sepa que vista utilizar en función de la URL que está requiriendo el clinete. Las URLs se ecriben mediante expresiones regulares del lenguaje Python. Este sistema de URLs aprovecha muy bien el modulo de expreciones regulares del lenguaje permitiendo por ejemplo recuperar gurpos nombrados (en contraposición al enfoque ordinal tradicional).

La asociación url-vistas se define en el módulo bajo el nombre *urlpatterns*. También es posible derivar el tratado de una parte de la expresión reuglar a otro módulo de urls. Generalmente esto ocurre cuando se desea delegar el tratado de las urls a una aplicación particular.

Ej: Derivar el tratado de todo lo que comience con la cadena personas a al módulo de urls de la aplicación personas.

```
(r'^personas', include('mi_proyecto.personas.urls'))
```

2.3.3 Mapeando URLs a Vistas

Con la estructura del proyecto así definida y las herramientas que provee Django, es posible ya ver resultados en el navegador web corriendo el servidor de desarrollo incluido en el framework para tal fin.

Es posible tambien en este momento definir algo de logica de negocios implementando vistas dentro del proyecto para dotar al sitio de algo de funcionalidad dinamica. Una función vista, es una simple función de Python que toma como argumento una petición Web y retorna una respuesta Web. En el momento de procesar una peticion HTTP Django seleccionara y ejecutara la vista. Lo

importante de este punto es como decirle a Django que vista ejecutar ante determinada url, es en este punto donde surgen las *URLconfs*.

La *URLconf* es como una tabla de contenido para el sitio web. Básicamente, es un mapeo entre los patrones URL y las funciones de vista que deben ser llamadas por esos patrones URL. Es como decirle a Django, "Para esta URL, llama a este código, y para esta URL, llama a este otro código".

En el apartado de modulos del proyecto se observo el modulo sobre el cual el objeto URLconf es creado automáticamente: el archivo `urls.py`, este modulo tiene como requisito indispensable la definicion de la variable `urlpatterns`, la cual Django espera encontrar en el módulo `ROOT_URLCONF` definido en `settings`. Esta es la variable que define el mapeo entre las URLs y el código que manejan esas URLs.

2.3.4 El sistema de plantillas

Las vistas son las encargadas de retornar respuestas Web, entre estas respuestas esta el codigo HTML que debe ser enviado al cliente o navegador, Django separar el diseño de la página del código Python correspondiente a la logica de negocio usando un *sistema de plantillas* para generar el HTML.

Una plantilla en Django es una cadena de texto que separar la presentación de un documento de sus datos. Una plantilla define rellenos y diversos bits de lógica básica (esto es, etiquetas de plantillas) que regulan como el documento debe ser mostrado. Normalmente, las plantillas son usadas para producir HTML, pero las plantillas de Django son igualmente capaces de generar cualquier formato basado en texto.

2.3.5 Sistema básico de plantillas

Una plantilla de Django es una cadena de texto que pretende separar la presentación de un documento de sus datos. Una plantilla define rellenos y diversos bits de lógica básica (esto es, etiquetas de plantillas) que regulan como el documento debe ser mostrado. Normalmente, las plantillas son usadas para producir HTML, pero las plantillas de Django son igualmente capaces de generar cualquier formato basado en texto.

A continuación se presenta una simple plantilla de ejemplo:

```
<html>
<head><title>Ordering notice</title></head>

<body>

<p>Dear {{ person_name }},</p>

<p>Thanks for placing an order from {{ company }}. It's scheduled to
ship on {{ ship_date|date:"F j, Y" }}.</p>
```

```
<p>Here are the items you've ordered:</p>

<ul>
{% for item in item_list %}
<li>{{ item }}</li>
{% endfor %}
</ul>

{% if ordered_warranty %}
<p>Your warranty information will be included in the packaging.</p>
{% endif %}

<p>Sincerely,<br />{{ company }}</p>

</body>
</html>
```

Esta plantilla es un HTML básico con algunas variables y etiquetas de plantillas agregadas.

- **Cualquier texto encerrado por un par de llaves es una *variable*.** Esto significa "insertar el valor de la variable con el nombre tomado".
- **Cualquier texto que esté rodeado por llaves y signos de porcentaje** es una *etiqueta de plantilla*. Una etiqueta le indica al sistema de plantilla que debe hacer algo.

  Este ejemplo de plantilla contiene dos etiquetas: la etiqueta `{% for item in item_list %}` (una etiqueta `for`) y la etiqueta `{% if ordered_warranty %}` (una etiqueta `if`).

  Una etiqueta `for` actúa como un simple constructor de bucle. Una etiqueta `if` actúa como una cláusula lógica "if".
- **Finalmente, el segundo párrafo de esta plantilla tiene un ejemplo de un *filtro*,** con el cual se alteran la exposición de una variable.

Cada plantilla de Django tiene acceso a varias etiquetas y filtros incorporados.

Nota: quiza completar un poco

2.3.6 Estructura de una aplicación Django

Una aplicación es un paquete python que consta de un módulo models y un módulo views, para crear una aplicacion se utiliza el comando **startapp** del modulo *manage* de la siguiente manera:

```
$ python manage.py startapp mi_aplicacion # Crea la aplicacion
```

El resultado de este comando genera la siguiente estructura en el proyecto:

```
mi_proyecto/
    mi_aplicacion/
        __init__.py
        models.py
        views.py
    ...
```

Módulo models

Cada vez que se crea una aplicación, se genera un módulo models.py, en el cual se le permite al programador definir modelos de objetos, que luego son transformados en tablas relacionales [16].

Módulo views

Cada aplicacion posee un módulo views, donde se definen las funcinoes que atienden al cliente y son activadas gracias a el mapeo definido en el módulo urls del proyecto o de la aplicación.

Las funciones que trabajan como vistas deben recibir como primer parámetro el request y opcionalmente parámetros que pueden ser recuperados del mapeo de urls.

Dentro del módulo de urls

```
# Tras un mapeo como el sigunete
(r'^persona/(?P<id_persona>\d)/$', mi_vista)
# la vista se define como
def mi_vista(request, id_persona):
    persona = Personas.objects.get(id = id_persona)
    datos = {'persona':persona, }
    return render_to_response('plantilla.html', datos)
```

2.3.7 Contenido dinámico y estático en Django

Django está orientado a la generación de *solo* contenido dinámico. Pero generalmente se reuiqre en una aplicación web la inclusión de recursos de tipo estático, como imágenes, javascript u hojas de estilo.

Django delega esta tarea al servidor web [DjangoDoc2009], sin embargo para desarrollo se prové un servidor de contenido estático.

La puesta en producción de una aplicación desarrollada en Django consta de la configuración del servidor web en 2 aspectos:

[16] Mediante el comando syncdb del módulo manage del proyecto

- **Configuración del soporte para ejecución de Python** En la presente tesis nos focalizamos en WSGI ya que es un estándard Pythonico, pero existen toros como mod_python para Apache. Independientemente de que tipo de soporte se utilize es necesario asegurarse que tanto Django, como así aplicaciones de terceros estén incluidas en el PythonPath con el cual se invoca al script con el entry point de la aplicación.
- **Configuración de los medios estáticos.** Generalmente suele ser una ruta, o dos, pero no está limitado.

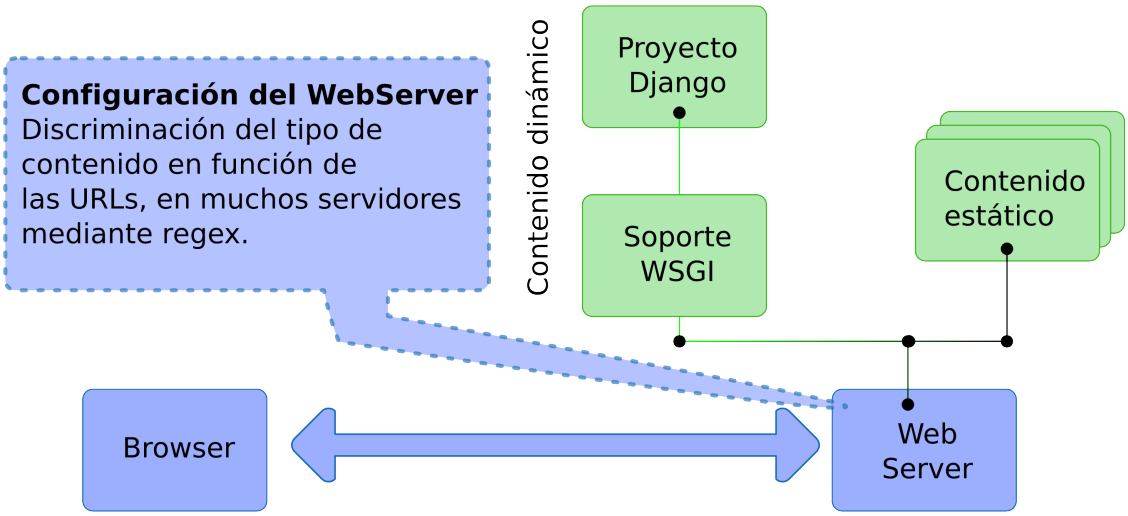


Figura 2.5: Deployment de Django

Una aplicación escrita en Django puesta en producción consta de contenido dinámico manejado por WSGI y contenido estático servido por el web server.

2.3.8 El cliclo de una petición

Cada vez que un browser realiza una petición a un proyecto desarrollado en django, la petición HTTP pasa por varias capas.

Inicialmente atraviesa los Middlewares, en la cual, el middleware de Request, empaqueta las variables del request en una instancia de la calse Request.

Luego de atravesar los middlewares de request, mediante las definciones de URLs, se selecciona la vista a ser ejecutada.

Una vista es una función que recibe como primer argumento el request y opcionalmente una serie de parámetros que puede recuperar de la propia URL.

Dentro de la vista se suelen hacer llamadas al ORM, para realizar consultas sobre la base de datos. Una vez que la vista a completado la lógica, genera un mapeo que es transferido a la capa de templates.

El template rellena sus comodines en función de los valores del mapeo que le entrega la vista. Un template puede poseer lógica muy básica (bifurcaciones, bucles de repetición, formateo de datos, etc).

El template se entrega como un HttpResponse. La responsabilidad de la vista es entregar una instancia de esta clase.

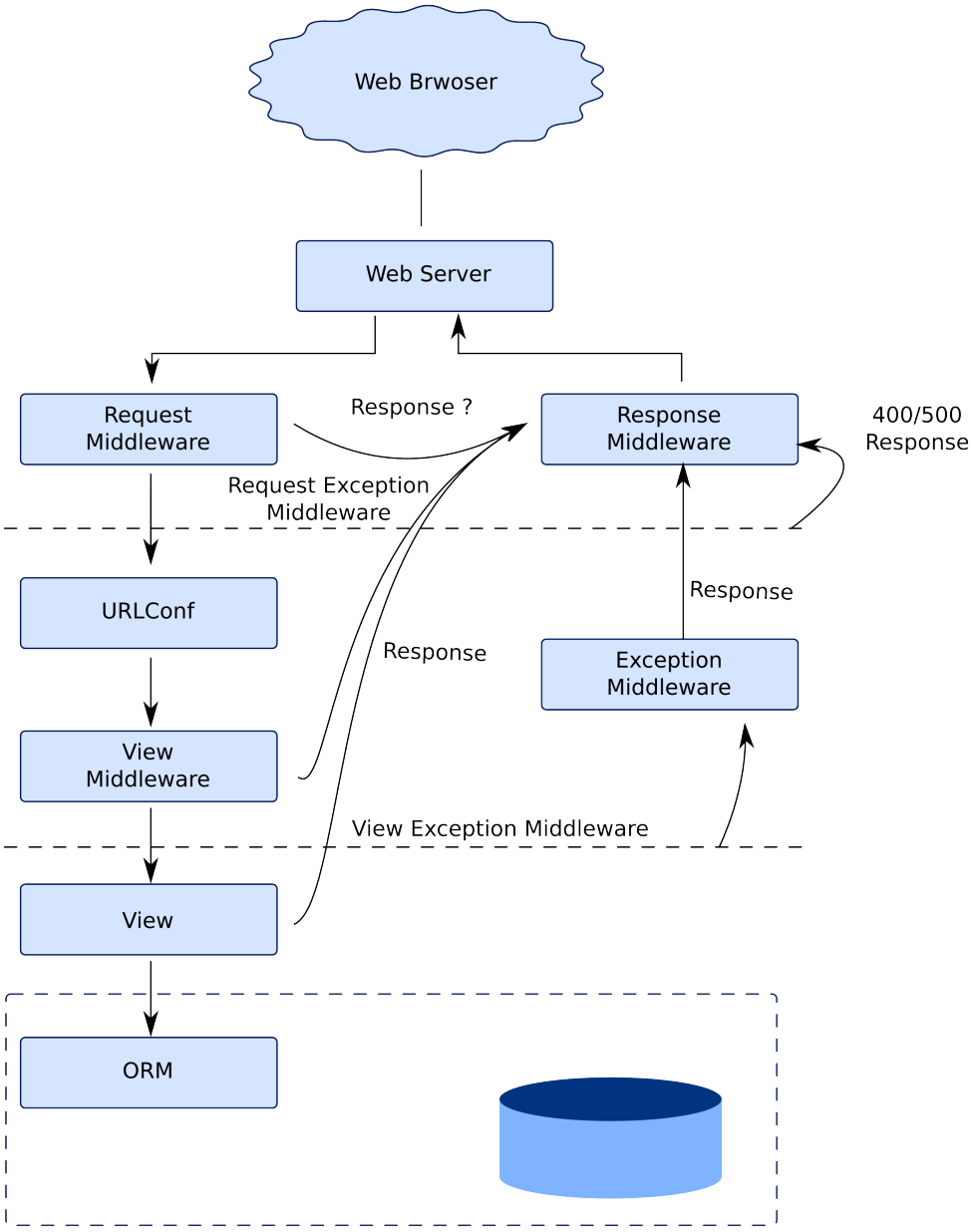


Figura 2.6: Proceso de un request

2.3.9 Interactuar con una base de datos

Django incluye una manera fácil pero poderosa de realizar consultas a bases de datos utilizando Python.

Una ves configurada la conexión a la base de datos en el módulo de configuración *settings* se esta condiciones de comenzar a usar la capa del sistema de Mapeo Objeto-Relacional del framework.

Si bien existen pocas reglas estrictas sobre cómo desarrollar dentro de Django, existe un requisito respecto a la convención de la aplicación: "si se va a usar la capa de base de datos de Django (modelos), se debe crear una aplicación de Django. Los modelos deben vivir dentro de una aplicaciones". Para crear una aplicación se debe proceder con el procedimiento ya mencionado en *manage*.

2.3.10 Modelos

Un modelo de Django es una descripción de los datos en la base de datos, representada como código de Python.

Esta es la capa de datos – lo equivalente a sentencias SQL – excepto que están en Python en vez de SQL, e incluye más que sólo definición de columnas de la base de datos. Django usa un modelo para ejecutar código SQL detrás de las escenas y retornar estructuras de datos convenientes en Python representando las filas de las tablas base de datos. Django también usa modelos para representar conceptos de alto nivel que no necesariamente pueden ser manejados por SQL.

Django define los modelos en Python por varias razones:

- **La introspección requiere overhead y es imperfecta. Django necesita** conocer la capa de la base de datos para porveer una buena API de consultas y hay dos formas de lograr esto. Una opción sería la introspección de la base de datos en tiempo de ejecución, la segunda y adoptada por Django es describir explícitamente los datos en Python.
- **Escribir Python es divertido, y dejar todo en Python limita el número de** veces que el cerebro tiene que realizar un "cambio de contexto".
- **El codigo que describe a los modelos se puede dejar fácilmente bajo un** control de versiones.
- **SQL permite sólo un cierto nivel de metadatos y tipos de datos basicos,** mientras que un modelo puede contener tipos de datos especializado. La ventaja de un tipo de datos de alto nivel es la alta productividad y la reusabilidad de código.
- SQL es inconsistente a través de distintas plataformas.

Una contra de esta aproximación, sin embargo, es que es posible que el código Python quede fuera de sincronía respecto a lo que hay actualmente en la base. Si se hacen cambios en un modelo Django, se necesitara hacer los mismos cambios dentro de la base de datos para mantenerla consistente con el modelo.

Finalmente, Django incluye una utilidad que puede generar modelos haciendo introspección sobre una base de datos existente. Esto es útil para comenzar a trabajar rápidamente sobre datos heredados.

Este modelo de ejemplo define una `Persona` que encapsula los datos correspondientes al `nombre` y el `apellido`.

```
from django.db import models

class Persona(models.Model):
    nombre = models.CharField(max_length = 30)
    apellido = models.CharField(max_length = 30)
```

`nombre` y `apellido` son atributos de clase

```
CREATE TABLE miapp_persona (
    "id" serial NOT NULL PRIMARY KEY,
    "nombre" varchar(30) NOT NULL,
    "apellido" varchar(30) NOT NULL
);
```

En el ejemplo presentado se observa que un modelo es una clase Python que hereda de `django.db.models.Model` y cada atributo representa un campo requerido por el módelo de datos de la aplicación. Con esta informacion Django genera automaticamente la *API* de acceso a los datos en la base.

Usando la API

Luego de crear los modelos y sincronizar la base de datos generando de esta manera el SQL correspondiente, se esta en condiciones de usar la API de alto nivel en Python que Django provee para accesar los datos:

```
>>> from models import Persona
>>> p1 = Persona(nombre='Pablo', apellido='Perez')
>>> p1.save()
>>> personas = Persona.objects.all()
```

En estas lineas se ven algunos detalles de la interaccion con los modelos:

- **Para crear un objeto, se importa la clase del modelo apropiada y se crea** una instancia pasándole valores para cada campo.
- Para guardar el objeto en la base de datos, se usa el método `save()`.
- Para recuperar objetos de la base de datos, se usa `Persona.objects`.

Internamente Django traduce todas las invocaciones que afecten a los datos en secuencias `INSERT`, `UPDATE`, `DELETE` de SQL

Django provee tambien una forma de seleccionar, filtrar y obtener datos de la base a traves de los administradores de consultas representado en el ejemplo anterior por `Persona.objects`.

Administradores de consultas

Los managers o administradores de consultas son los objetos que representan la interfase de comunicacion con la base de datos. Cada modelo tiene por lo menos un administrador para acceder a los datos almacenados. Cada entidad presente en el modelo, tiene al menos un *Manager*. Este *Manager* encapsula en una semántica de objetos las opreaciones de consulta (*query*) de la base de datos [17] . Un *Manager* consiste en una instancia de la clase *django.db.models.manager.Manager* donde se definen, entre otros métodos, *all()*, *filter()*, *exclude()* y *get()*.

Cada uno de éstos métodos genera como resultado una instancia de la calse *QuerySet*. Un *QuerySet* envuelve el "resultado" de una consulta a la base de datos. Se dice que envuelven el "resultado" porque la estrategia de acceso a la base de datos es *evaluación retardada* [18], es decir, que la consulta que representa el *QuerySet* no será evaluada hasta que no sea necesario acceder a los resultados.

Un *QuerySet*, además de presentar la posibilidad de ser iterado, para recuperar los datos, también posee una colección de métodos orientados a conuslta, como *all()*, *filter()*, *exclude()* y *get()*. Cada uno de estos métodos, al igual que en un mánager, devuelven instancias de *QuerySet* como resultado. Gracias a esta característica recursiva, se pueden generar consultas medinate encadenamiento.

Clasificación de aplicaciones Django

Nota: Mejorar esto

Podemos hacer una clasificación entre los tipos de aplicaciones en función de su objetivo:

- **Las aplicaciones de usuario** Son aquellas aplicaciones que resuelven un problema específico. Suelen

estar

dispuestas en un subdirectorio del proyecto.

- **Las aplicaciones de soporte** Las aplicaciones de soporte son las aplicaciones que resuelven un problema

general, como autenticación, envío de correo, interacción con servicos provistos por

sitios de terceros (Google Maps, Google Charts, AdSense, Yahoo Maps, Yahoo Pipes,

Feedburner, etc.), etc. Django provee varias aplicaciones que están disponibles por defecto

[17] En el lenguaje SQL, las consultas se realizan miediante la sentecia SELECT.
[18] También conocida como *Lazy Evaluation*

ante la

creación de un proyecto con el comando *create-app*.

- **Aplicaciones template** Esta caracterización se puede aplicar tanto a proyectos como a aplicaciones

y consite en una aplicación distribuida como plantilla para lograr una aplicación de

usuario, pero que tiene varios factores engorrosos resueltos. Podemos citar en este caso,

a el proyecto *Pinax* [19] que provee un conjunto de aplicaciones de soporte y

aplicaciones de template.

[19] *Pinax Project* http://pinaxproject.com/

CAPÍTULO 3

Tecnologías del cliente

3.1 Generación dinámica de páginas Web

Desde la perspectiva de un usuario, la Web consiste en una enorme cantidad de documentos interconectados a nivel mundial llamados *paginas Web*. En el cliente las paginas se ven mediante un programa llamado navegadores

JavaScript ha tenido durante mucho tiempo la reputación de lenguaje torpe e inadecuado para el desarrollo serio. Esto ha sido en gran parte debido a las implementaciones incompatibles del lenguaje y del DOM en varios navegadores, y al uso extenso de código "pegado" lleno de errores por parte de los aficionados. Los errores en tiempo de ejecución eran tan frecuentes y difíciles de solucionar que pocos programadores intentaban corregirlos.

La reciente aparición de navegadores con un soporte adecuado de los estándares web, frameworks para JavaScript tales como Prototype, y herramientas de depuración de alta calidad han facilitado enormemente la creación de código organizado y escalable en JavaScript, y la aparición de AJAX lo ha hecho esencial. Mientras hasta hace poco JavaScript era solo utilizado para las tareas relativamente simples y no críticas (validación de formularios y decoraciones llamativas), actualmente se está utilizando para escribir código complejo que a menudo es responsable de buena parte de la funcionalidad básica de un sitio. Los errores en tiempo de ejecución y el comportamiento imprevisible ya no son molestias de menor importancia; son errores fatales.

3.2 Estructura de un navegador

3.2.1 Navegador Web

Nota: Acá abordamos desde la perspectiva del navegador HTTP, HTML y JavaScript

Un navegador web o *web browser*, es un software encargado de representar documentos de hipertexto al usuario, siendo los lenguajes de codificación de hipertexto más populares HTML y XHTML. Un navegador no solo representa los documentos de hipertexto, sino que puede representar otros tipos de documentos, como imágenes (:term:JPEG, :term:GIF, :term:PNG, etc.), sonido (:term:WAV, :term:MP3, :term:OGG) y contenido multimedia como vídeo (:term:MPEG, :term:H264, :term:RM, :term:MOV), e interactivos como es el caso de Macromedia Flash, applets Java o controles ActiveX en la plataforma Windows. Debido a la cantidad de recursos que debe manejar un navegador, el servidor web agrega a cada respuesta al cliente una cabecera donde le indica el tipo de recurso que está entregando. Esta especificación se realiza con el estándar :term:MIME.

Un navegador web acepta como entrada del usuario una URL, comúnmente conocida como dirección de Internet. Una vez validada la URL, el navegador web descarga el recurso apuntado por la URL mediante el protocolo HTTP.

Una URL tiene el siguiente esquema, donde podemos diferenciar varios componentes

Figura 3.1: Forma de una URL.

Los componentes de una URL son:

- **esquema** Especifica el mecanismo de comunicación. Generalmente HTTP y HTTPS en

una comunicación

asegurada mediante TLS [1] .

- **anfitrión** Especifica el nombre de dominio del servidor en Internet, por ejemplo:

***google.com*,** *nasa.gov*, *wikipedia.com*, etc. Se popularizó la utilización de el subdominio "www" para identificar el

anfitrión que ejecuta el servidor web, dando lugar a direcciones del tipo

***www.google.com*,** *www.nasa.gov*, etc.

El *puerto* es un parámetro de conexión TCP, y suele ser omitido debido

a que

el esquema suele determinarlo, siendo 80 para HTTP y 443 para HTTPS.

- **recurso** Especifica dentro del servidor, la ruta para acceder al recurso
- **query** El parámetro query tiene sentido cuando el recurso apuntado por la ruta

[1] *Transport Security Layer* es el sucesor de *Secure Socket Layer* (SSL),

no se trata de una página estática y sirve para el pasaje de parámetros. El programa que genera el recurso puede recibir como argumentos estos

parámetros, por ejemplo, cuando se ingresa la palabra *foo* en el buscador google,

la

URL que provee el resultado de la búsqueda es la siguiente:

```
http://www.google.com/search?*q=foo*
```

- **enlace** Dentro de un documento de hipertexto pueden existir destinos de

enlaces, o enlaces internos. Gracias a este parámetro se puede enlazar a una sección específica de un documento, permitiendo al navegador ubicarse visualmente.

es un protocolo criptográfico que provee conexiones seguras a través de una red, típicamente Internet [WikiSSL2009] .

Agosto 2009.

Cuando el recurso apuntado se encuentra en Internet, el navegador realiza una conexión hacia el servidor web indicado por el dominio y mediante el protocolo HTTP le informándole a que recurso debe acceder.

3.2.2 HTTP

Para acceder al recurso *~ndefosse/introducion_lenguaje_python.html* en el servidor *students.unp.edu.ar*, de la url http://students.unp.edu.ar/~ndefosse/introducion_lenguaje_python.html, el navegador conforma la siguiente consulta:

```
GET /~ndefosse/introduccion-lenguaje-python.html HTTP/1.1
Host: students.unp.edu.ar
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ar,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cookie: user_id=G7NVG5YY51I9DZAIJDEDQIXYQSRF0CTL
```

Esto conforma lo que se conoce como una consulta HTTP o *HTTP Request*. En la primera linea se conforma de el método HTTP y el nombre del recurso, finalizando con la versión del prtocolo que soporta el navegador (o cliente):

```
GET /~ndefosse/introduccion-lenguaje-python.html HTTP/1.1
```

La sgunda linea es el host al cual se accede. Un mismo servidor web puede estar publicado en varios dominos, mediante esta linea se puede discriminar desde cual se intenta acceder al recurso:

```
Host: students.unp.edu.ar
```

El siguiente componente del *request* es la linea que identifica al cliente, en este caso el navegador informa que se trata de Mozila, versión 5:

```
User-Agent: Mozilla/5.0
```

Una vez que el servidor web a localizado y accedido al recurso, precede a enviar la respuesta

3.2.3 HTML

HTML es un lenguaje de marcado que teiene como objetivo describir un documento de hipertexto. Un documento HTML se conforma por una serie de *tags* o etiquetas.

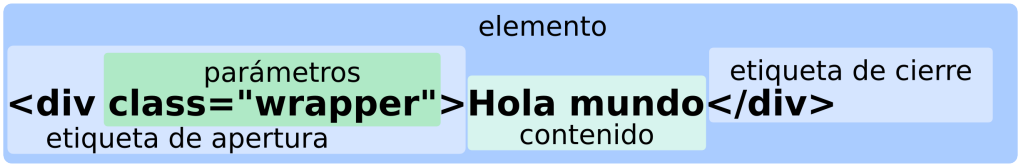


Figura 3.2: Una etiqueta HTML.

Un documento HTML está delimitado por las etiquetas o *tags* html y contiene una cabecera delimitada por *head* y un cuerpo, delmitado por *body*.

```
<html>
        <head>
                <title>Mi pagina</title>
        </head>
        <body>
                <h1>Título principal</h1> <!-- comentario -->
                <p>Párrafo</p>
        </body>
</html>
```

Un documento HTML contener enlaces a recursos entendibles para el navegador, como los enlaces a hojas de estilos o código javascript.

La inclusión de una hoja de estilo se realiza mediante el tag *link*, de la siguiente manera:

```
<link type="text/css" rel="stylesheet" href="hoja_de_estilos.css">
```

Se puede además embeber en la página el estilo, de la siguente manera

```
<style type="text/css">

        BODY {
                font-family: "Verdana";
                font-size: 12pt;
                padding: 2px 2px 3px 2px;
        }
</style>
```

Al haberse incrustado el estilo en una página en particular, este solo tiene validés para ese recurso en particular.

Otro tipo de recurso entendible para el navegador constituye la inclusión de código de ejecución en el clinete, mediante el tag script.

```
<script type="text/javscript" src="mi_codigo.js"></script>
```

Además de la inclusión de recursos javascript externos, el código javascript se puede embeber en el código html de varias maneras [StephenChapmanJS2009], entre ellas:

```
<script type="text/javascript">
<!-- // Javascript en un comentario, para pasar validadores XHTML y ocultar
```

de navegadores

// que no soporten Javascript var x = 2; var y = 4;

// –> </script>

último acceso Agosto 2009, http://javascript.about.com/library/blxhtml.htm

3.2.4 CSS

Una hoja de estilo en cascada especfica como se va a mostrar un documento en pantalla, como se va a imprimir o inclusive como se realiza la pronunciación a través de un dispositivo de lectura [W3cCSS2009] .

El objetivo de CSS es separar el contendio de la presentación de un documento HTML o XML. Una hoja de estilo puede ser enlazada desde varias paginas, permitiendo mantener coherencia y consistencia en el estilo.

http://www.w3c.es/divulgacion/guiasbreves/HojasEstilo

3.2.5 JavaScript

Javascript es un lenguaje de programación interpretado creado originalmente por Brendan Eich para la empresa Netscape con el nombre de Mocha. El lenguaje surge a principios de 1996, como un lenguaje de scripting para la Web y apuntado a la interacción directa con el usuario.

Con una sintaxis semejante a la de Java y C, JavaScript dista mucho de ser Java y debe su nombre más a cuestiones de marketing que a principios de diseño, de hecho se reconocen mayores influencias de lenguajes como Self, Scheme, Perl e incluso en versiones modernas de Python.

En junio de 1997 la European Computer Manufacturers' Association ECMA, adopta como un estándar a JavaScript, con el nombre de ECMAScript. Este importante paso marco un inicio para la compatibilidad entre navegadores y tras él las empresas comenzaron a desarrollar sus propias versiones del lenguaje, JScript es la implementación de Microsoft, muy similar al la de Netscape, pero con ciertas diferencias en el modelo de objetos del navegador [2]; con la finalidad de evitar la incopatibilidad que se avecinaba, el World Wide Web Consortium diseñó el estándar Document Object Model (DOM) que en conjunto con el lenguaje resuelven la modificación de la página Web en el cliente. La batalla entre los navegadores se continua luchando y probablemente de para unos cuantos años más, pero estos dos estandares representan la base para un desarrollo web compatible.

En la actualidad, todo computador personal en el mundo tiene, al menos, ún interprete de JavaScript instalado y activo en el. A pesar de su popularidad, solo algunos saben que JavaScript es un muy buen lenguaje de programación dinámico, orientado a objetos y de proposito general.

Incluir JavaScript en un documento

Cualquier documento HTML puede incluir JavaScript y el método correcto que define la W3C es incluir javascript como un archivo externo, tanto por cuestiones de accesibilidad, como practicidad y velocidad en la navegación. Para esto basta con escribir en el documento HTML:

```
<script type="text/javascript" src="[URL]"></script>
```

Siendo [URL] la URL relativa o absoluta del recurso con código JavaScript. Cuando el navegador descarga el documento y comienza su lectura al encontrar esta etiqueta solicita al servidor el archivo referenciado por URL y lo interpreta, para continuar luego con la lectura del resto de las etiquetas.

El otro método para incluir código, es hacerlo directamente en el documento entre los elementos <script> y </script>, aunque esta practica no es la recomendada:

```
<script type="text/javascript">
<!--
    // código JavaScript
```

[2] Por que sera?.

```
-->
</script>
```

De forma similar a la anterior, el navegador comienza con la interpretacion del codigo al encontrar la etiqueta y luego continua con la lectura del resto de las etiquetas.

Orientado a Objetos

JavaScript es un lenguaje orientado a objetos, es más, todo en este lenguaje es un objeto. Normalmente a muchos programadores de Java o C++ les cuesta comprender la implementación de objetos que provee JavaScript y esto es así porque el lenguaje esta orientado a prototipos.

Algunos argumentan que JavaScript no es verdaderamente orientado a objetos porque no provee ocultación de la información. Pero resulta ser que los objetos de JavaScript pueden tener variables privadas y métodos privados gracias a las clausuras.

A continuacion se explica como crear un objeto:

```
uno = new Object();
dos = {};
```

En este primer ejemplo se ver que hay dos formas de crear objetos, en particular la segunda forma hace uso de una característica propia del lenguaje para declarar estructuras de datos y resulta de mucha utilidad en los casos de intercambio de mensajes mediante el estándar *JSON*. Otros tipos de objetos pueden ser declarados de esta manera, como:

```
cadenas = "";
arreglos = [];
funciones = function() {};
```

Las caracteristicas dinámicas de lenguaje posibilitan la modificación de los objetos en tiempo de ejecución, esto quiere decir que a un objeto, luego de ser creado, se le pueden agregar o eliminar métodos y propiedades:

```
uno.variable = “hola”;
dos.metodo = new Function();
```

De esta forma se tienen dos objetos personalizados uno con un “atributo” y otro con un “metodo”, de esta manera se pueden continuar agregando miembros de cualquier tipo y hacer crecer los objetos. Sin embargo, crear objetos así no es cómodo, ya que si se quisiéra crear una copia de este objeto se deberían declarar nuevamente todos sus miembros.

Creando ‘Clases’:

La forma en que los lenguajes orientados a objetos comúnmente resuelven el problema anterior es mediante el uso de clases, en JavaScript no es posible declarar clases, pero si es posible instanciar objetos a partir de un constructor.

El objeto Function se utiliza como objeto instanciable en JavaScript, y el cuerpo de la función es el constructor del objeto. Para emular el comprortamiento propio de una clase el método mas difundido es aprovechar el funcionamiento de la plabra reservada *this* dentro del constructor y las funciones miembro. Cuándo una función es llamada con el operador *new*, *this* hace referencia al objeto que será retornado.

```
/*clase de ejemplo*/

Clase = function() {
    this.propiedad = "hola!";
    this.metodo = function(){
        alert(this.propiedad);
    }
}
objeto = new Clase(); // instanciamos 'Clase'
```

Podria parecer que aquí se terminan los problemas pero no es así, esta forma de crear clases provoca que cada objeto tenga una versión unica de sus 'metodos', cuando lo ideal seria que todos los objetos compartieran la misma función.

La propiedad 'Prototype':

Todas las funciones tienen una propiedad llamada prototype, esta propiedad es un objeto que será utilizado como 'modelo' inicial de todos los objetos que sean creados con esta función cuando sea utilizada como constructor.

Reescribiendo el ejemplo anterior el código quedaría así:

```
/* creamos un constructor limpio */

Clase = new Function();
Clase.prototype.propiedad = "hola!";
Clase.prototype.metodo = function(){
    alert(this.propiedad);
}
objeto = new Clase(); // instanciamos 'miClase'
```

Ahora todos los objetos creados a partir de 'Clase' comparten inicialmente las mismas referencias en todas sus propiedades, esto significa que, todos los objetos van a compartir la misma versión de 'metodo'.

Adicionalmente, al utilizar la propiedad prototype, se obtiene otra ventaja y es poder usar la palabra reservada instanceof para determinar si un objeto es instancia de un constructor.

> http://javis.wordpress.com/2006/10/23/javascript-orientado-a-objetos/

3.3 Evolución del JavaScript

3.3.1 Herencia

JavaScript no puede tener herencia orientada a clases, pero tiene herencia "prototipal". Existen muchos autores que implementan su forma de emular la herencia en JavaScript y entre los métodos mas utilizados se detallan a continuacion dos prinsipios basicos ampliamente difundidos.

Object masquerading

Este método saca provecho del comportamiento de la palabra reservada this dentro de los constructores. El funcionamiento es el siguiente: Un constructor asigna propiedades y métodos a un objeto referenciándolo con la palabra clave this, como un constructor es simplemente una función, se puede usar el constructor de una clase A como método de una clase B.

```
function ClaseA(nombre){
    this.nombre = nombre;
    this.identificarse = function(){
        alert(this.nombre);
    }
}
```

Recordemos que en un constructor, this hace referencia al nuevo objeto que será retornado. Pero en un método, this hace referencia al objeto desde el cual fue llamado.

```
function ClaseB(nombre){
    this.superClase = ClaseA;
    this.superClase(nombre);
    delete this.superClase;
}
```

En el código anterior, el constructor 'ClaseA', es llamado como método del nuevo objeto que se esta creando en 'ClaseB', por lo tanto, todas las propiedades y métodos que se crean en ClaseA se van a agregar al nuevo objeto de ClaseB.

Quizás el punto mas interesante del Object masquerading, frente a otros métodos de emular herencia, es que soporta la herencia múltiple, esto significa que un objeto puede heredar de mas de una clase al mismo tiempo. Solo basta con llamar a cuantos constructores sean necesarios dentro del constructor de la clase hija.

Prototype chaining

Anteriormente se mostro como definir clases utilizando el objeto prototype. Prototype chaining se basa en este objeto y es el método recomendado por el standard ECMA Script. Prototype es una

propiedad del objeto Function, que actúa como un template sobre el cual se van a crear nuevos objetos. Mas precisamente, las propiedades y métodos del objeto prototype van a ser pasados a todas las instancias de esa clase.

El ejemplo anterior utilizando prototype chaining quedaría de la siguiente manera:

```
function ClassA() {}
ClassA.prototype.nombre = "";
ClassA.prototype.identificarse = function() {
    alert(this.nombre);
}
function ClassB() {}
ClassB.prototype = new ClassA();
```

La última línea del ejemplo muestra el funcionamiento del prototype chaining. Lo que ocurre es que asignamos al prototipo de 'ClassB' una nueva instancia de 'ClassA'. De ahora en adelante, todos los objetos creados con 'ClassB' van a tener, también, los mismos métodos y propiedades de la instancia de 'ClassA'. Y si queremos agregar mas métodos y propiedades, lo único que tenemos que hacer es agregárselos al prototype de 'ClassB'.

Lo malo de este método para emular herencia, es que no se puede pasar parámetros a la clase base, como hicimos en el ejemplo de Object masquerading.

Lo bueno, es que el operador instanceof funciona de una manera única: por cada instancia de ClaseB, instanceof nos retorna true tanto con 'ClaseA' como con 'ClaseB':

```
var miobjeto = new ClassB();
alert(miobjeto instanceof ClassB); // true
alert(miobjeto instanceof ClassA); // true
```

3.3.2 AJAX

La expansión y masificación de la Web, llevó a que el ciclo de petición respuesta en el navegador resultara insuficiente para proveer al usuario de aplicaciones interactivas o :term:´RIA´, por este motivo se comenzo a trabajar sobre nuevas técnica de desarrollo que culminaron el en nacimiento de AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML).

Una aplicacion Web que trabaje con AJAX se ejecutan en el cliente o navegador y mantiene una comunicación asíncrona con el servidor en segundo plano. De esta forma le es posible realizar cambios sobre las páginas sin necesidad de recargarla, lo que significa un aumento de la interactividad, velocidad y usabilidad en las aplicaciones.

AJAX es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje en el que normalmente se efectúan las funciones de llamada de

AJAX mientras que el acceso a los datos se realiza mediante XMLHttpRequest. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

En la practica AJAX no constituye una tecnología en sí, sino que es un término que engloba a éstas cuatro tecnologías:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que

acompaña a la información.

- Document Object Model (DOM) accedido con un lenguaje de scripting por

parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.

- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el

servidor web. En algunos frameworks y en algunas situaciones concretas, se usa un objeto iframe en lugar del XMLHttpRequest para realizar dichos intercambios.

- XML es el formato usado generalmente para la transferencia de datos

solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Funcionamiento de AJAX

Se crea y configura un objeto XMLHttpRequest. # El objeto XMLHttpRequest realiza una llamada al servidor. # La petición se procesa en el servidor. # El servidor retorna un documento XML que contienen el resultado. # El objeto XMLHttpRequest llama a la función callback() y procesa el resultado. # Se actualiza el DOM (Document Object Model)de la página asociado con la petición con el resultado devuelto

Las ventajas de AJAX

- Mayor interactividad
- Recuperación asíncrona de datos, reduciendo el tiempo de espera del usuario
- Facilidad de manejo del usuario
- El usuario tiene un mayor conocimiento de las aplicaciones de escritorio
- Se reduce el tamaño de la información intercambiada
- Portabilidad entre plataformas
- No requieren instalación de complementos, applets de Java, ni ningún otro

elemento * Código público

Las desventajas de AJAX

- Usabilidad: Comportamiento del usuario ante la navegación

- Botón de volver atrás del navegador
- Empleo de iframes ocultos para almacenar el historial
- Empleo de fragmento identificador del URL ('#') y recuperación mediante

JavaScript * Problema al agregar marcadores/favoritos en un momento determinado de la aplicación * Problemas al imprimir páginas renderizadas dinámicamente * Tiempos de respuesta entre la petición del usuario y la respuesta del servidor * Empleo de feedback visual para indicar el estado de la petición al usuario * Requiere que los usuarios tengan el JavaScript activado en el navegador * En el caso de Internet Explorer 6 y anteriores, que necesita tener activado el ActiveX

Nota: terminar este tema

A pesar de que el término «AJAX» fuese creado en 2005, la historia de las tecnologías que permiten AJAX se remonta a una década antes con la iniciativa de Microsoft en el desarrollo de Scripting Remoto. Sin embargo, las técnicas para la carga asíncrona de contenidos en una página existente sin requerir recarga completa remontan al tiempo del elemento iframe (introducido en Internet Explorer 3 en 1996) y el tipo de elemento layer (introducido en Netscape 4 en 1997, abandonado durante las primeras etapas de desarrollo de Mozilla). Ambos tipos de elemento tenían el atributo src que podía tomar cualquier dirección URL externa, y cargando una página que contenga javascript que manipule la página paterna, pueden lograrse efectos parecidos al AJAX.

El Microsoft's Remote Scripting (o MSRS, introducido en 1998) resultó un sustituto más elegante para estas técnicas, con envío de datos a través de un applet Java el cual se puede comunicar con el cliente usando JavaScript. Esta técnica funcionó en ambos navegadores, Internet Explorer versión 4 y Netscape Navigator versión 4. Microsoft la utilizó en el Outlook Web Access provisto con la versión 2000 de Microsoft Exchange Server.

La comunidad de desarrolladores web, primero colaborando por medio del grupo de noticias microsoft.public.scripting.remote y después usando blogs, desarrollaron una gama de técnicas de scripting remoto para conseguir los mismos resultados en diferentes navegadores. Los primeros ejemplos incluyen la librería JSRS en el año 2000, la introducción a la técnica imagen/cookie[1] en el mismo año y la técnica JavaScript bajo demanda (JavaScript on Demand)[2] en 2002. En ese año, se realizó una modificación por parte de la comunidad de usuarios[3] al Microsoft's Remote Scripting para reemplazar el applet Java por XMLHttpRequest.

Frameworks de Scripting Remoto como el ARSCIF[4] aparecieron en 2003 poco antes de que Microsoft introdujera Callbacks en ASP. NET.[5]

Desde que XMLHttpRequest está implementado en la mayoría de los navegadores, raramente se usan técnicas alternativas. Sin embargo, todavía se utilizan donde se requiere una mayor compatibilidad, una reducida implementación, o acceso cruzado entre sitios web. Una alternativa, el Terminal SVG[6] (basado en SVG), emplea una conexión persistente para el intercambio continuo entre el navegador y el servidor.

3.3.3 JSON

JSON brinda un buen soporte al intercambio de datos, resultando de fásil lectura/escritura para las personas y de un rapido interpretacion/generacion para las maquinas. Se basa en un subconjunto del lenguaje de programación JavaScript, estándar ECMA-262 3ª Edición - Diciembre de 1999. Este formato de texto es completamente independiente del lenguaje de programacion, pero utiliza convenciones que son familiares para los programadores de lenguajes de la familia ‘‘C’‘, incluyendo C, C + +, C#, Java, JavaScript, Perl, Python y muchos otros.

JSON se basa en dos estructuras: * Una colección de pares nombre / valor. En varios lenguajes esto se

representa mediante un objeto, registro, estructura, diccionario, tabla hash, introducido lista o matriz asociativa.

- Una lista ordenada de valores. En la mayoría de los lenguajes esto se representa como un arreglo, matriz, vector, lista, o secuencia.

Estas son estructuras de datos universales. Prácticamente todos los lenguajes de programación modernos las soportan de una forma u otra. Tiene sentido que un formato de datos que es intercambiable con los lenguajes de programación también se basan en estas estructuras.

Para mas informacion sobre JSON http://www.json.org/

Nota: Iteradores

3.4 Google Gears

3.4.1 Introducción

Google Gears es un software de código abierto distribuido por Google que añade una nueva capa de aplicación a los navegadores.

Una vez instaldo como una extensión en el navegador, el producto agrega una API que permite programar en JavaScript interacciones con los componentes que contiene.

Los tres componentes principales que incorpora gears son:

- Local Server

  Permite almacenar localmente datos correspondientes a las páginas webs. Tanto HTML, JavaScript e imágenes entre otros, pueden ser almacenados localmente por el cliente e interponerse entre el requerimiento del navegador al servidor en conslutas posteriores, evitando asi la solicitud HTTP y optimizando el tiempo de respuesta de la aplicacion.

  Pese a que su funcionamiento es muy similar al de la cache del navegador la diferencia fundamental esta en que la actualizacion de los recursos que almacena

es mantenida y realizada por el desarrollador.

- Database

  Permite almacenar localmente datos que no correspondan a una página web pero son parte de la logica de la aplicacion y requieren de un almacenamiento presistente.

  El motor de base de datos utilizado es SQLite con algunos agregados y restricciones para brindar seguridad y formas de busqueda.

  Luego de que el usuario de la aplicacion web otroge el permiso explicito de creacion de la base, el desarrollador, disponer de un almacenamiento del tipo relacional en la máquina huésped.

- Worker Pool

  De manera similar a los "hilos" del sistema operativo, el manejador de hilos permite ejecutar acciones en segundo plano sin bloquear la ejecución del hilo principal del navegador.

  Hay que destacar que el manejador no corre en forma paralela a la ejecución del navegador, sino que se ejecuta cuando la página web se mantiene activa, por lo cual el refresco de página o la salida de la misma provoca que este se detenga o no se ejecute directemente.

Basicamente Gears y sus principales componentes estan enfocados en permitir al programador ejecutar sus aplicaciones cuando el navegador no está conectado al servidor. El lider del grupo de desarrollo Bret Taylor dijo que buscaba ser capaz de acceder al Google Reader mientras usaba la conexión de la compañía, la cual frecuentemente tenia un acceso defectuoso a Internet.

Gears esta incluido en el nuevo navegador de Google (Google Chrome) y posee extensiones para instalarse en Internet Explorer 6.0+, Mozilla Firefox, Safari y Opera Mini, y funciona en los sistemas operativos Windows 2000, XP y Vista, Windows Mobile 5 y 6, MacOS y Linux de 32 bits.

Con susecivos lanzamientos el producto sea visto mejorado y favorecido en varios aspectos a partir de la versión 0.4 del Gears se puede hablar ya de:

- API para GIS, que permite acceder a la posición geográfica del usuario.
- El API Blob, que permite gestionar bloques de datos binarios.
- Accede a archivos en el equipo cliente a través del API de Google Desktop.
- Permite enviar y recibir Blobs con el API HttpRequest.
- Localización de los cuadros de diálogo de Gears en varios idiomas.
- API para canvas, que permite manipular imagenes desde JavaScript.

En todos los casos que se requiera tener instalada una aplicacion soportada por Gears vale la pena aclarar que el cliente debe haber accedido al menos una vez al servidor de la aplicacion web y

otorgado los permisos de instalacion correspondiente.

CAPÍTULO 4

Introducción al desarrollo

Nota: Un mejor titulo no vendría mal

4.1 Soporte de lenguajes de programación en el browser

Tras la elección de Django como framework web, se emprendió el análisis de las posibilidades de ejecución de una aplicación web desarrollada en Django en el browser. Surgen puntos de interés como:

- ¿Es posible ejecutar el esquema de servidor web en un cliente?
- ¿Es posible ejecutar Python y mod_wsgi en el cliente?
- ¿Que posibilidades hay de brindar seguridad sobre los datos que se transfieran al cliente?
- Una vez lograda la transferencia de una aplicación web al cliente, ¿Que posibilidad existe de sincronizar los datos con la aplicación "madre"?

entre otras que se analizaran más adelante en el texto.

El enfoque inicial fue realizar un "espejo" de la aplicación que se ejecuta en el servidor en el cliente.

Django está escrito en Python y si bien la ejecución de Python es posible en un browser, la solución es engorrosa y no es cross-browser. En la plataforma Mozilla, la integración se puede realizar mediante PyXPCOM [1] , PyShell [2] y también existe una extensión para XUL [3]

[1] *PyXPCOM*, conexión del modelo de objetos multiplataforma de Mozilla con Python, https://developer.mozilla.org/en/PyXPCOM

[2] *PyShell*, consola interactiva

[3] *Luxor, Python for XUL* http://mail.python.org/pipermail/python-announce-list/2003-March/002084.html

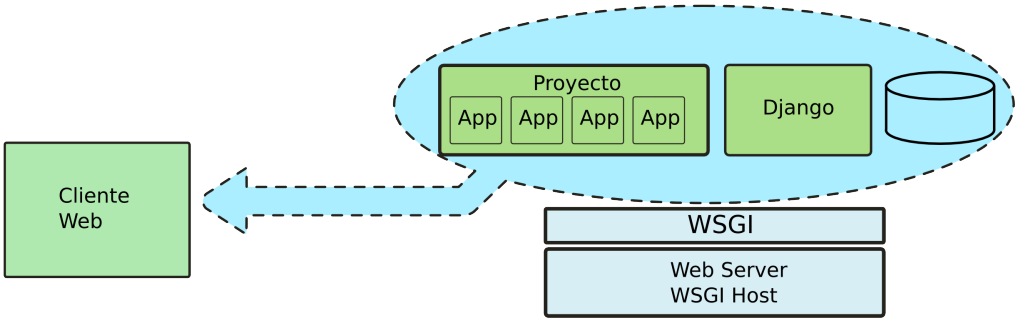


Figura 4.1: Elementos a transferir desde la aplicación online

4.2 Soluciones existentes

Sobre la plataforma Windows, existen dos formas de ejecutar Python en el navegador, la primera consiste en la ejecución de un control ActiveX sobre el browser que cuente con el intérprete de Python embebido. Un control ActiveX es un componente ejecutable empotrable, que puede ser dibujado en una página web. Los controles ActiveX son peligrosos en el ámbito de la web debido a que fueron ideados para ser utilizados como elemento incrustable entre aplicaciones o para el uso en entornos confiables. Un control ActiveX cuenta con privilegios similares a los de una aplicación tradicional sobre el equipo del cliente. La mayoría de los antivirus y herramientas de seguridad los eliminan o hacen responsable de la seguridad al usuario a partir de la ejecución del ActiveX. Si bien esta técnica se presenta atractiva gracias a que Python es un lenguaje que ha sido diseñado para ser embebido, los controles ActiveX no cumplen con las garantías de seguridad necesarias para el desarrollo de aplicaciones para la web. Es posible considerar esta solución "cross-browser" gracias a proyectos como un *host para ActiveX sobre la plataforma mozilla* [4] pero no es multiplataforma.

La segunda alternativa es utilizar el framework Silverlight de Microsoft, que permite generar aplicaciones para browsers basados en la plataforma *.Net*. Silverlight es un runtime similar al popular Adobe Flash Player, pero las aplicaciones pueden ser creadas en cualquier lenguaje de la plataforma .Net, incluyendo Python y Ruby [5] [6] a partir de la versión 4, bajo sus implementaciones sobre CLR.

Nota: Preguntar en el IRC de *pyar* desde cuando tiene la API standard IronPython si no volar el párrafo

Esta tecnología exige que el navegador cuente con un plugin para la plataforma, siendo el más completo el desarrollado por Microsoft, propulsor de la plataforma .Net. IronPython es una implementación de Python sobre .Net que no cuenta con la API standard [7] de la versión CPython, por lo

[4] *ActiveX para Mozilla* http://www.iol.ie/~locka/mozilla/plugin.htm

[5] *Silverlight, a new way for Python?* http://mail.python.org/pipermail/python-list/2007-May/610021.html

[6] *Sitio de Michael Frood, donde explica como ejecutar IronPython sobre .Net* http://www.voidspace.org.uk/ironpython/silverlight/index.shtml#id2

[7] *Listado de Módulos de la API standard* http://docs.python.org/modindex.html

que no se puede ejecutar django de manera nativa.

Aunque en un principio no era posible ejecutar Django sobre la IronPython, gracias a la popularidad del framework, con la versión 2.0 se superó esta dificultad [8].

En

Gracias a la posibilidad de acceso a DOM por medio de una aplicación construida con Silverlight [9] y al almacenamiento local en el cliente introducido en en Silverlight 2.0, se hace posible ejecutar Django en el cliente con acceso a una base de datos local, sin embargo, la arquitectura de software necesaria para desplegar este tipo de aplicaciones se hace compleja, que en cierta forma apunta contra los ideales de Python y Django:

- **Plugin necesario** Es necesario un plugin en el browser que no se encuentra disponible para todas las plataformas *no cross-browser*. O al menos, no en su estado más maduro
- **Madurez de la herramientas fuera de la plataforma Windows** Las herramientas de desarrollo solo están en su estado más maduro sobre la plataforma Windows Si bien existen compiladores gratuitos, las herramientas son propietarias y las IDEs que permiten un desarrollo más eficiente son pagas y propietarias.
- No existe soporte para IronPython en la IDE de facto, VisualStudio.
- La implementación de Python no es la estándar, y por ahora poco soportada [IronPython-FAQ2009] .

Tras el análisis de Silverlight como tecnología de soporte, se decide analizar las posibilidades nativas de los navegadores web. En particular se focalizó el análisis sobre las implementaciones de Javascript como lenguaje de soporte para la programación del lado del cliente.

4.3 Un lenguaje adecuado para ejecución de la aplicación en el cliente

En principio, Javascript y Python parecen lenguajes bastante diferentes en su sintaxis, sin embargo, comparten ciertas características como ser orientados a objetos basados en prototipos y permitir la definición de clausuras [AtulVarma2009] .

Más allá de la ejecución de Python, el servidor necesita un medio para almacenar los datos, típicamente una base de datos y un mecanismo para servir estáticamente los archivos de medios. Transportar la aplicación a un cliente web requiere un mecanismo de almacenamiento que brinde estos dos elementos en conjunto.

Google desarrolló un plugin multiplataforma y cross-browser llamado *Google Gears* que tiene como objetivo facilitar el sporte offline de aplicaciones web. Gears se compone de 3 componentes

[8] *Django On IronPython* http://www.infoq.com/news/2008/03/django-and-ironpython

[9] *Silverlight Tutorial - Interaction With The DOM* http://www.switchonthecode.com/tutorials/silverlight-tutorial-interaction-with-the-dom

básicos:

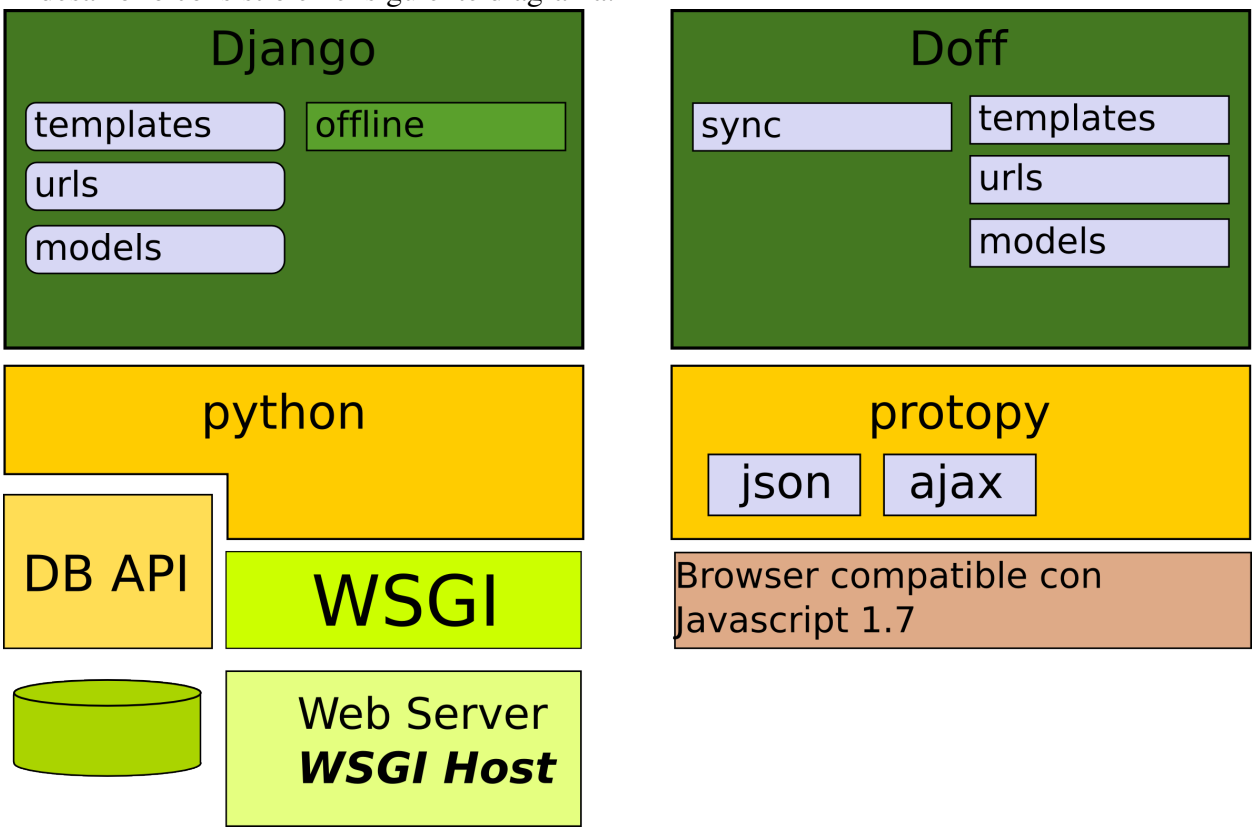
- **LocalServer** Un servidor de archivos estáticos en el clinete
- **DataBase** Una base de datos relacional
- **WorkerPool** Un sistema de procesos, para realizar tareas concurrentes de manera aislada y no provocar retrasos en el dibujado de la pantalla.

Generar este espejo conlleva poseer un equivalente al framework Django que pueda ser ejecutado en un browser, brindando los componentes básicos de Django:

- Mapeador Objeto Relacional
- Renderización de templates
- Asociación de expresiones regulares a funciones.

Esta idea surge en parte gracias al proyecto "Gars On Rails" [10] , un equivalente al objetivo de la presente tesis sobre el framework Rails.

El desarrollo consistió en el siguiente diagrama:



[10] *Isolated Storage in Silverlight 2.0* http://www.ddj.com/windows/208300036

CAPÍTULO 5

Protopy

5.1 Introducción

Nota: Ver donde poner estos dos parrafos, que estan buenos pero creo que no van aca :)

En apartados teóricos se menciono, entre otras cosas, que la creación de un framework generalmente surge de la identificación de objetos reusables en el desarrollo de software. Posteriormente los objetos identificadas decantan en componentes que forman parte de la arquitectura del framework y a los cuales se accede mediante la extención y la configuración. Esta forma de obtener un producto que permita arquitecturar proyectos de caracteristicas similares a los que lé dieron origen, implica pasar por varias etapas de maduración en el desarrollo, pero requieren de un punto de partida primordial y es justamente un proyecto que oriente la identificacion de las partes reusables. Nuestro caso se complica en este punto, ya que la intención panteada desde el principio es la de "desarrollar un framework" y la realidad del desarrollo es que muchas de las partes surguieron en paralelo y de la mano de aplicaciones de prueba.

Si bien la intención primaria es la de brindar un prodcuto que permita ejecutar "aplicaciones desconectadas" en los navegadores, existen aspectos complejos en el ambiente de un navegador que dificultan el desarrollo y que requieren de una capa soporte. Entre estas podemos mencionar:

- Organizar y obtener código.

  Un framework con funciones minimas, como una API de base de datos y un motor de plantillas, requiere de varias lineas de código para su implementación. Es por esto que resulta deseable que existan formas de organizar y obtener código en el cliente; liberando así al desarrollador de estas tareas tediosas y permitiendole enfocarse en la funcionalidad.

- Reuso y extensión.

  No es una buena practica modificar un framework y su utilizacion debe estar basa en la extención y la configuración. JavaScript es el lenguaje para el framework y

si se quiere promover el reuso y la extensión es conveniente proveer al lenguaje de objetos extensibles o contrucciones que fasilmente sean asimiladas por el desarrollador.

- Guerra de los navegadores.

  Si bien se han logrado muchos avanzes con la estandarizacion de APIs, en general existen muchas diferencias entre navegadores para trabajar con el DOM o con los objetos de JavaScript. Una capa uniforme de acceso al DOM, funciones y objetos que trabajen sobre APIs heterogeneas, entre otras cosas, resultan buenas herramientas para el desarrollo client-side.

- Interacción entre cliente servidor.
- La cara visible o vista debe ser fasilmente manipulable por la aplicacion de

usuario. * Como los datos generados en el cliente son informados al servidor. * El framework debe brindar soporte a la aplicacion de usuario de una forma natural y transparente. * Como se ponen en marcha los mecanimos o acciones que la aplicacion de usuario define.

En este capitulo se introducen las ideas principales que motivaron la creacion de una biblioteca en JavaScript, que brinde el soporte necesario al framework y a buena parte de los items expuestos.

5.1.1 Biblioteca

Protopy es una biblioteca JavaScript para el desarrollo de aplicaciones web dinamicas. Aporta un enfoque modular para la inclusión de código, orientación a objetos, manejo de AJAX, DOM y eventos.

Para una referencia completa de la API de Protopy remitase al apandice *Protopy*

5.1.2 Historia

Si bien el desarrollo de la biblioteca se mantuvo en paralelo a la del Framework, existen aspectos basicos a los que esta brinda soporte y permiten presentarla en un apartado separado como "Una Biblioteca en JavaScript", esta constituye la base para posteriores construcciones y auna herramientas que simplifican el desarrollo client-side.

proto type + **py** thon = **protopy**

"La creación nace del caos", la libreria "Protopy" no escapa a esta afirmacion e inicialmente nace de la integracion de Prototype con las primeras funciones para lograr la modularizacion; con el correr de las lineas de codigo [1] el desarrollo del framework torna el enfoque inicial poco sustentable, requiriendo este de funciones más Python-compatibles se desecha la libreria base y se continua con un enfoque "pythonico", persiguiendo de esta forma acercar la semántica de JavaScript 1.7 a la del lenguaje de programacion Python.

[1] Forma en que los informaticos miden el paso del tiempo.

No es arbitrario que el navegador sobre el cual corre Protopy sea Firefox y mas particularmente sobre la version 1.7 de JavaScript. El proyecto mozilla esta acercando, con cada nueva versiones del lenguaje, la semantica de JavaScript a la de Python, incluyendo en esta version generadores e iteradores los cuales son muy bien explotados por Protopy y el Framework.

5.2 Organizando el codigo

Como ya se vio en la sección dedicada a *JavaScript*, una de las formas tradicionales y recomendada de incluir funcionalidad en un documento HTML es mediante el tag *script*, haciendo una referencia en el atributo *src* a la url del archivo que contiene el codigo; en una instancia posterior, cuando el cliente accede al recurso, carga el archivo con las sentencias JavaScript y las interpreta en el contexto del documento.

El enfoque tradicional resulta sustentable para pequeños proyectos, donde el lenguaje brinda mayormente soporte a la interacción con el usuario (validacion, accesibilidad, etc) y los fragmentos de código que se pasan al cliente son bien conocidos por el desarrollador, pero en proyectos que implican mayor cantidad de funcionalidad JavaScript, con grandes cantidades de código, este enfoque resulta complejo de mantener y evolucionar en el tiempo. Es por esto que para Protopy se busco como primera medida una forma de organizar y obtener el código del servidor que resulte sustentable y escalable.

Similar al concepto de *modulos en Python*, el desarrollo de Protopy se oriento en pequeñas unidades funcionales llamadas **modulos**.

Además de la sanidad mental que implica organizar el código en distintos archivos, los módulos representan un cambio muy importante en la obtención de funcionalidad; ya no es el documento HTML el que dice al cliente que archivo cargar del servidor, sino que el mismo código interpretado va obteniendo la funcionalidad a medida que la requiere.

El enfoque modular no es nuevo en programación y basicamente, la implementación de Protopy, implica llevar el concepto de “divide y vencerás” ó “análisis descendente (Top-Down)” al hambito de JavaScript.

Un módulo resuelve un problema especifico y define una interfaz de comunicación para accesar y utilizar la funcionalidad que contiene. Por más simple que resulte de leer, esto implica que existe una manera de **obtener** un modulo y una manera de **publicar** la funcionalidad de un modulo, logrando de esta forma que interactuen entre ellos.

En su forma mas pedestre un módulo es un archivo que contiene definiciones y sentencias de JavaScript. El nombre del archivo es el nombre del módulo con el sufijo .js pegado y dentro de un módulo, el nombre del módulo está disponible como el valor de la variable *__name__*.

5.2.1 Obtener un módulo

La función *require* es la encargada de obtener un módulo del servidor e incorporarlo al *espacio de nombres* del llamador. Por ejemplo, cuando un módulo llamado spam es requerido, Protopy busca un archivo llamado spam.js en la url base [2], de no encontrar el archivo el error `LoadError` es lanzado a la función que requirio el módulo.

Otra forma de obtener módulos es usando nombres de **paquetes**. A diferencia de Python un paquete no incluye funcionalidad en si mismo y su funcionalidad principal es la de establecer las bases en la cual buscar modulos. De forma similar a la anterior cuando un módulo llamado foo.spam es importado, Protopy busca en el objeto sys.paths si existe una url asociada al paquete foo, de encontrar la url base para foo el archivo spam.js es buscado en esa ubicación, por otra parte si sys.paths no contiene una url asociada a foo el archivo foo/spam.js es buscado en la url base.

El uso del objeto sys.paths permite a los modulos de JavaScript que saben lo que están haciendo modificar o reemplazar el camino de búsqueda para los módulos. Nótese que es importante que el script no tenga el mismo nombre que un *módulo estandar*.

Las formas en que el modulo obtenido es presentado al llamador difiere en funcion de los parametros pasados a *require*. Estas formas son:

- Un modulo puede ser obtenido como un objeto,
- Se puede obtener solo determinada funcion de un modulo,
- O se pueden importar todas las definiciones del módulo en el espacio de

nombres del llamador.

Para ver más *Apendice Protopy*.

5.2.2 Publicar un módulo

La acción de publicar un modulo implica exponer la funcionalidad que este define. En Python no es necesario explicitar que funcionalidad del módulo se expone a los llamadores, ya que todo lo definido en él es público; pero los módulos en Protopy se evaluan dentro de una clausura y los llamadores no podran acceder a sus funciones si no son publicadas.

La funcion *publish* es la encargada de relizar la tarea de publicar el contenido del modulo en Protopy.

Un modulo puede contener sentencias ejecutables y definición de funciones, generalmente las sentencias son para inicializar el módulo ya que estas se evaluan solo la primera vez que el módulo es requerido a alguna parte y las definiciones son las que efectivamente seran publicadas como funcionalidad.

[2] Ruta base desde la cual la biblioteca Protopy carga los módulos, por defecto esta es la url base del archivo protopy.js más el sufijo *packages*.

A continuación se presenta un fragmento de código que ejemplifica el uso de las dos funciones presentadas.

```
 /* Obtengo la funcion copy y deepcopy del modulo copy
    estas funciones son para copia de objetos
    superficiales y en profundidad respectivamente. */
require('copy', 'copy', 'deepcopy');

/* Representa la cantidad maxima de caches */
var MAX = 1000;

/* Objeto para guardar las cache */
var cache = {};

...

/* Estructura de datos que representa a una cache */
var Cache = type('Cache', [ Dict ], {
    ...
});

/* Funcion que retorna una cache */
function get_cache(name) {
    if (len(cache) > 1000)
        throw new Exception('%s caches creadas'.subs(MAX));
    var c = cache[name];
    if (!isundefined(c)) {
        c = new Cache();
        cache[name] = c;
    }
    return d;
}

/* Publico la cantidad maxima de caches
   la funcion para obtener caches. */
publish({
    MAX: MAX,
    get_cache: get_cache
});
```

Asumiendo que el código presentado está en un archivo llamado *caches.js* en la url base, éste representa un módulo en sí mismo y el acceso se obtiene mediante la invocación de 'require('caches')'.

Nota: describir de forma rápida los módulos principales

5.3 Creando tipos de objeto

En el apartado teórico se toco el tema de construcción de objetos en base a "clases" y de la emulación de herencia en JavaScript. Aunque muchos autores cuestionan estas practicas alegando, con justa razon, que no tiene sentido emular un paradigma dentro de otro; en la practica tener una implementación de objetos tipados en JavaScript ayuda al reuso de código y en gran medida a que los programadores se acerquen al lenguaje.

La forma de crear nuevos "tipos de objetos" en Protopy es a traves de la función *type*. Esta funcion no fue parte de la biblioteca hasta que no se observo la necesidad de otrogar mayor "poder" al constructor de clases que brindaba Prototype, y su aparción posibilito nuevas formas de construir tipos, similares a las construcciones de tipos que brinda la funcion homónima en Python, a la cual debe su nombre.

A continuación se presenta un fragmento de código que ejemplifica la creación de tipos en Protopy.

```
var Dict = type('Dict', object, {
    ...
});

var SortedDict = type('SortedDict', [ Dict ], {
    __init__: function(object) {
        this.keyOrder = (object && isinstance(object, SortedDict))? copy(object.key
        super(Dict, this).__init__(object);
    },
    __iter__: function() {
        for each (var key in this.keyOrder) {
            var value = this.get(key);
            var pair = [key, value];
            pair.key = key;
            pair.value = value;
            yield pair;
        }
    },
    __deepcopy__: function() {
        var obj = new SortedDict();
        for (var hash in this._key) {
            obj._key[hash] = deepcopy(this._key[hash]);
            obj._value[hash] = deepcopy(this._value[hash]);
        }
        obj.keyOrder = deepcopy(this.keyOrder);
        return obj;
    },
    __str__: function() {
        var n = len(this.keyOrder);
        return "%s".times(n, ', ').subs(this.keyOrder);
    },
```

```
32      set: function(key, value) {
33          this.keyOrder.push(key);
34          return super(Dict, this).set(key, value);
35      },
36      unset: function(key) {
37          without(this.keyOrder, key);
38          return super(Dict, this).unset(key);
39      }
40  });
```

Rapidamente, este ejemplo presenta la definición del tipo SortedDict, el cual es una especialización del tipo base Dict. Como se observa la función constructora recibe, como primer argumento el nombre para el nuevo tipo, seguidamente un arreglo con los tipo base y para terminar un objeto con los atributos y metodos para los objetos de ese tipo.

Con estos dos nuevos constructores ya es posible crear "objetos tipados" que se comporten en función de sus respectivas definiciones.

```
>>> d = new SortedDict({'uno': 1})
>>> d.set('dos', 2)
>>> d.get('dos')
2
>>> d.items()
[["uno", 1 ], ["dos", 2 ]]
```

Como se observa, para instanciar un nuevo tipo se utiliza el operador *new* de JavaScript, este operador crea el nuevo objeto e invoca a la funcion __init__.

En los métodos la palabra reservada *this* tiene el comportamiento esperado, presentado en la parte teorica, el cual es hacer referencia al objeto instanciado con *new*.

Internamente *type* utiliza el objeto *prototype* para la construcción, con lo cual el operador *instanceof* presenta un comportamiento coherente, pese a esto se recomienda usar la funcion *isinstance* que trae Protopy, ya que esta permite navegar por la cadena de herencia.

```
>>> d instanceof SortedDict
true
>>> d instanceof Dict
false
>>> isinstance(d, Dict)
true
```

5.3.1 Inicialización

En el ejemplo se muestra la inicialización del tipo SorteDict usando una funcion llamada __init__. Este método es llamado por el operador *new* inmediatamente despues de crear una instancia. Sería

tentador decir que es el "constructor" de la clase. Si bien se parece un constructor de Java, actúa como si lo fuese ya que el objeto ya esta instanciado cuando se llama a esta función.

Los métodos __init__ pueden tomar cualquier cantidad de argumentos, e igual que las funciones en JavaScript estos son opcionales para quien invoca.

Los métodos __init__ son opcionales, pero cuando se define uno, se debe recordar llamar explícitamente al método __init__ del ancestro.

5.3.2 Otros métodos

- __str__

  Este metodo se llama cuando es necesario proveer de una reprecentacion en texto del objeto, JavaScript provee para este objetivo el metodo toString, pero por cuestiones de nombres se prefirio usar este metodo en su lugar y hacer internamente una relacion entre las funciones.

- __iter__

  Protopy se vale de versiones modernas de JavaScript y brinda soporte a iteradores, este es el método que el desarrollador debe definir si quiere objetos iterable, la funcion debe retornar un objeto que implemente el metodo next (un iterador o un generado), los bucles for hacen esto automáticamente, pero también se puede hacer manualmente.

- __deepcopy__
- __copy__
- __json__
- __html__

Los primeros tipos que surgen para la organizacion de datos dentro de la librerias con los "Sets" y los "Diccionarios", hambos aproximan su estructura a las estructuras homonimas en python, brindando una funcionalidad similar. Si bien la estructura "hasheable" nativa a JavaScript en un objeto, los diccionarios de Protopy permiten el uso de objetos como claves en lugar de solo cadenas.

Al igual que muchas partes de Protopy *type* cambio a lo largo del desarrollo, sumando soporte para herencia multiple, metodos magicos e incluso "metatipos".

Arguments

isinstance

5.3.3 Herencia

La función contrcuctora de tipos en la biblioteca surgue como herramienta para proveer al Framework de mecanismos de extensión.

issubclass

super

5.3.4 Metodos y atributos de tipo

5.4 Extendiendo DOM y JavaScript

Si bien el *DOM* ofrece ya una *API* muy completa para acceder, añadir y cambiar dinámicamente el contenido del documento HTML, existen funciones muy utiles y comunes en los desarrollos que los programadores incorporan al HTML y se intentaron englobar y mantener dentro del nucleo de Protopy. Funciones para modificar e incorporar elementos al documento son muy comunes y estan disponibles en Protopy, en conjunto con otras para el manejo de formularios, como serializacion, obtencion de valores, etc.

Otra extension interesante de mencionar es la de los tipos de datos en JavaScript, en manejo de cadenas incorpora nuevas funciones que simplifican tareas comunes, los numeros y fechas tambien tiene su aporte.

Los eventos estan uniforamdos bajo un modulo de manejo de eventos, que permite conectar eventos del :term:DOM con funciones en JavaScript, así tambien como funciones entre si.

Nota: Poner los nombres de las funciones para destacar el laburo

5.5 Envolviendo a gears

La biblioteca puede funcionar independiente de la instalación de Google Gears, debido a que su principal funcionalidad como ya se menciono es la de extención de JavaScript y posterior soporte al framework, pese a esto Protopy provee mecanismos para uniformar el acceso y extender los objetos de Gears cuando éste se encuentra instalado en el navegador.

El acceso al *Factory* de Gears esta centralizado y controlado en el objeto *gears* dentro del modulo *sys*, mediante este objeto es posible conocer el estado de Gears y sus permisos. El objeto informa al desarrollador si Gears esta instalado, la versión, si los permisos son corrector e incluso simplifica el proceso de instalación de la extensión de no estar presente en el navegador entre otras cosas.

El metodo *create* del objeto *sys.gears* es el encargado de crear y retornar los objetos Gears. Esta función se ayuda de módulos presentes en el paquete *gears* para asistir la creación de los objetos. El objeto que se retorne dependera de la precencia del módulo con el mismo nombre en dicho

paquete; de no encontrar un modulo que asista la cración de un objeto Gears el objeto en sus estado "puro" es retornado al llamador.

Si bien no es necesario que los módulos obtengan el acceso a Gears a traves de Protopy, es recomendable que asi se haga; ya que la biblioteca provee los mecanimos de extención para los objetos en *create*. Esto no fue así desde el comienzo de del desarrollo y fue una idea que se maduro luego de observar que resultaba complejo y costoso requerir los módulos que involucraban a Gears desde distintos lugares.

El desarrollo del framework implico extender algunos objetos Gears, concretamente al paquete *gears* se incorporaron los siguientes módulos:

- desktop

  El objeto *desktop* de Gears permite interactuar con el escritorio del **cliente.** Aquí se extendio la creacción de accesos directos para simplificar la **generación** de los mismos y agregar la posibilidad de manejar *Icon* y algunos *IconTheme*.

- database

  Sobre el objeto *database* se agrego funcionalidad uniformar el acceso a **la base** de datos por los módulos de Protopy y encapsular los *ResulSet* en **cursores a** los que se incorporo iteradores, registro de funciones para tipos de datos, etc.

5.6 Auditando el codigo

Una queja recurrente de los desarrolladores que trabajan con JavaScript es lo complejo que resulta el lenguaje para depurar errores. Encontrar errores en el código resulta molesto y mas todavía si la salida de los mismos no esta en un formato adecuado y encausada en un lugar especifico.

Tradicionalmente lo que se hace para detectar errores es valerse de funciones *alert* diseminadas por el código con textos del estilo "Paso por aquí", pero luego de cerrar unas diez o quince ventanas de este estilo, generalmente se pierde referencia de donde esta ocurriendo el error y se cae en la tentación de comenzar a comentar alerts a mansalva esperando dar con el indicado. Este es un claro ejemplo de que tanto la salida como el sistema de detección de errores es molesto e infructifero.

Una clara ventaja sobre el sistema tradicional y poderosa herramienta de desarrollo es el plug-in Firebug, que integra entre otras cosas una consola JavaScript y un debuger al ambinete del navegador, permitiendo a los desarrolladores depurar el código mediante puntos de corte, inspección de variables, etc. En firebug la consola pasa a ser por defecto la principal salida de errores, gracias a la funcion *console.log* los desarrolladores pueden redirigir todos los "Paso por aquí" a una salida uniforme e incluso inspeccionar el valor de las variables.

Protopy lleva la depuración y auditoria del código un paso más lejos, integrando un sistema de logging propio altamente configurable y con posibilidades de escribir en diferentes salidas y con diferentes formatos.

Similar a log4j el logger de Protopy trabaja con niveles de prioridad para los mensajes, distintos manejadores o *Handlers* y salidas en varios formatos. Todo esto configurable por el desarrollador.

Una vez configurado el sistema de logging, los módulos que requieran auditar el codigo solo deben requerir un logger en su sespacio de nombre e invocar a sus funciones.

```
var logging = require('logging.base');
var logger = logging.get_logger(__name__);

...

logger.debug('La query: %s\n Los parametros: %s', query, params, {});
```

En este ejemplo se requiere el módulo *logging* y posteriormente un logger para el módulo con el nombre __name__, de no encontrar configuración para este módulo se adopta la configuracion del modulo inmediato superior y asi consecutivamente hasta tomar la configuración del root logger.

Suponiendo que el módulo del ejemplo se llama doff.db.models.sql el siguiente archivo de configuracion perapraria este logger en modo DEBUG para auditar el código en la consola de firebug y en una url con distintos formatos.

```
{
    'loggers': {
        'root': {
            'level': 'DEBUG',
            'handlers': 'firebug'
        },
        'doff.db.models.sql': {
            'level': 'DEBUG',
            'handlers': [ 'firebug', 'remote'],
            'propagate': true
        },
    },
    'handlers': {
        'firebug': {
            'class': 'FirebugHandler',
            'level': 'DEBUG',
            'formatter': '%(time)s %(name)s(%(levelname)s):\n%(message)s',
            'args': []
        },
        'remote': {
            'class': 'RemoteHandler',
            'level': 'DEBUG',
            'formatter': '%(levelname)s:\n%(message)s',
```

```
            'args': ['/loggers/audit']
        },
        'alert': {
            'class': 'AlertHandler',
            'level': 'DEBUG',
            'formatter': '%(levelname)s:\n%(message)s',
            'args': []
        }
    }
}
```

Nota: Poner referencia a firebug

5.7 Interactuando con el servidor

Protopy permite al desarrollador trabajar con AJAX de forma simple y segura, encapuslando en objetos la lógica de petición y los valores de retorno del servidor.

Todo lo relacionado con AJAX se encuentra en el modulo *ajax*. El objeto de transporte para ajax es *XMLHttpRequest* e internamente se salvan las diferencias que existen entre los distintos navegadores. La forma de realizar una peticion es creando una instancia del objeto ajax.Request.

```
new ajax.Request('una/url', {method: 'get'});
```

La primera opcion es la url de la solicitud, y el segundo parametro es el hash de opciones, en este caso el metodo a utilizar es GET, si no se especifica, el metodo por defecto es POST.

Por defecto la respuesta del servidor es asincrona, para este caso se debe explicitar en el hash de opciones las funciones que manejaran los eventos disparados.

```
new ajax.Request('una/url', {
    method:'get',
    onSuccess: function(transport){
        var response = transport.responseText || "sin texto";
        alert("Success! \n\n" + response); },
    onFailure: function(){
        alert('Algo esta mal...'); }
});
```

En el ejemplo se presentan dos funciones, onSuccess y onFailure, para manejar los eventos correspondientes. A cada manejador se le pasa un objeto que representa la respuesta obtenida y que esta en relación con el evento capturando.

Otros manejadores que se pueden definir son:

- onUninitialized

- onLoading
- onLoaded
- onInteractive
- onComplete
- onException

Todos estos dependen de un estado del objeto *XMLHttpRequest*.

De igual manera que el resto de las opciones es posible agregar parametros a la peticion, estos pueden ser pasados como un objeto "hasheable" o como una cadena clave-valor.

5.8 Soporte para json

La idea detras del soporte para JSON en Protopy es la transimisión de datos generados offline por el cliente, en el momento de recuperar la conexion con el servidor el cliente debe serializar los datos y enviarlos al servidor; otro uso para es el intercambio de mensajes de control.

La transferencia de datos involucra varios temas, uno de ellos y que compete a este apartado, es el formato de los datos que se deben pasar por la conexcion; este formato debe ser "comprendido" tanto por el cliente como por el servidor. Desde un primer momento se penso en JSON como el formato de datos a utilizar, es por esto que Protopy incluye un modulo para trabajar con el mismo.

El soporte para JSON se encuentra en el modulo "json" entre los módulos estandar de Protopy. Este brinda soporte al pasaje de estructuras de datos JavaScript a JSON y viceversa.

Los tipos base del lenguje JavaScript estan soportados y tienen su reprecentacion correspondiente, object, array, number, string, etc. pero este modulo interpreta ademas de una forma particular a aquellos objetos que implementen el metodo __json__, dejando de este modo en manos del desarrollador la reprecentacion en JSON de determinado objetos.

Con el soporte de datos ya establecidos en la libreria, el framework solo debe limitarse a hacer uso de él y asegurar la correcta sincronizacion de datos entre el cliente y el servidor web, este tema se retomara en el capitulo de sincronizacion.

5.8.1 XML

No existe una razon concreta por la cual se deja de lado el soporte en Protopy para XML como formato de datos; aunque se puede mencionar la simplicidad de implementacion de un parser JSON contra la implementacion de uno en XML. Para el leector interesado agregar el sporte para XML en Protopy consta de escribir un modulo que realize esa tarea y agregarlo al paquete base.

5.9 Ejecutando codigo remoto

El RPC (del inglés Remote Procedure Call, Llamada a Procedimiento Remoto) es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. El protocolo es un gran avance sobre los sockets usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando éstas encapsuladas dentro de las RPC.

Las RPC son muy utilizadas dentro del paradigma cliente-servidor. Siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.

Hay distintos tipos de RPC, muchos de ellos estandarizados como pueden ser el RPC de Sun denominado ONC RPC (RFC 1057), el RPC de OSF denominado DCE/RPC y el Modelo de Objetos de Componentes Distribuidos de Microsoft DCOM, aunque ninguno de estos es compatible entre sí. La mayoría de ellos utilizan un lenguaje de descripción de interfaz (IDL) que define los métodos exportados por el servidor.

Hoy en día se está utilizando el XML como lenguaje para definir el IDL y el HTTP como protocolo de red, dando lugar a lo que se conoce como servicios web. Ejemplos de éstos pueden ser SOAP o XML-RPC. XML-RPC es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.[1]

Es un protocolo muy simple ya que sólo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión. La simplicidad del XML-RPC está en contraste con la mayoría de protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso.

Fue creado por Dave Winer de la empresa UserLand Software en asociación con Microsoft en el año 1998. Al considerar Microsoft que era muy simple decidió añadirle funcionalidades, tras las cuales, después de varias etapas de desarrollo, el estándar dejó de ser sencillo y se convirtió en lo que es actualmente conocido como SOAP. Una diferencia fundamental es que en los procedimientos en SOAP los parámetros tienen nombre y no interesan su orden, no siendo así en XML-RPC.[2

CAPÍTULO 6

Doff

6.1 Introducción

————————- OJO CON ESTO QUE TIENE PONIES ————————

Este libro es sobre Django, un framework de desarrollo Web que ahorra tiempo y hace que el desarrollo Web sea divertido. Utilizando Django puedes crear y mantener aplicaciones Web de alta calidad con un mínimo esfuerzo.

En el mejor de los casos, el desarrollo web es un acto entretenido y creativo; en el peor, puede ser una molestia repetitiva y frustrante. Django te permite enfocarte en la parte divertida – el quid de tus aplicaciones Web – al mismo tiempo que mitiga el esfuerzo de las partes repetitivas. De esta forma, provee un alto nivel de abstracción de patrones comunes en el desarrollo Web, atajos para tareas frecuentes de programación y convenciones claras sobre cómo solucionar problemas. Al mismo tiempo, Django intenta no entrometerse, dejándote trabajar fuera del ámbito del framework según sea necesario.

El objetivo de este libro es convertirte en un experto de Django. El enfoque es doble. Primero, explicamos en profundidad lo que hace Django, y cómo crear aplicaciones Web con él. Segundo, discutiremos conceptos de alto nivel cuando se considere apropiado, contestando la pregunta "¿Cómo puedo aplicar estas herramientas de forma efectiva en mis propios proyectos?" Al leer este libro, aprenderás las habilidades necesarias para desarrollar sitios Web poderosos de forma rápida, con código limpio y de fácil mantenimiento.

En este capítulo ofrecemos una visión general de Django.

6.1.1 Framework

Doff en un Framework Web escrito en JavaScript que provee un marco de desarrollo para las aplicaciones Web cliente-side.

Para una referencia completa de la API de Doff remitase al apandice *Doff*

6.1.2 Historia

> **d** jango + **off** line = **doff**

Mientras que un cliente se encuentre sin conexion con el servidor web, es capaz de generar y almacenar datos usando su base de datos local. Al reestablecer la conexion con el servidor web, estos datos deben ser transimitos a la base de datos central para su actulizacion y posterior sincronizacion del resto de los clinetes.

El framework fue construido con el objeto de ser compatible con los modelos y las plantillas de django, intentando acercar así al desarrollador que tenga experiencia en este framework.

Django nació naturalmente de aplicaciones de la vida real escritas por un equipo de desarrolladores Web en Lawrence, Kansas. Nació en el otoño boreal de 2003, cuando los programadores Web del diario *Lawrence Journal-World*, Adrian Holovaty y Simon Willison, comenzaron a usar Python para crear sus aplicaciones. El equipo de The World Online, responsable de la producción y mantenimiento de varios sitios locales de noticias, prosperaban en un entorno de desarrollo dictado por las fechas límite del periodismo. Para los sitios – incluidos LJWorld.com, Lawrence.com y KUsports.com – los periodistas (y los directivos) exigían que se agregaran nuevas características y que aplicaciones enteras se crearan a una velocidad vertiginosa, a menudo con sólo días u horas de preaviso. Es así que Adrian y Simon desarrollaron por necesidad un framework de desarrollo Web que les ahorrara tiempo – era la única forma en que podían crear aplicaciones mantenibles en tan poco tiempo – .

En el verano boreal de 2005, luego de haber desarrollado este framework hasta el punto en que estaba haciendo funcionar la mayoría de los sitios World Online, el equipo de World Online, que ahora incluía a Jacob Kaplan-Moss, decidió liberar el framework como software de código abierto. Lo liberaron en julio de 2005 y lo llamaron Django, por el guitarrista de jazz Django Reinhardt.

A pesar de que Django ahora es un proyecto de código abierto con colaboradores por todo el mundo, los desarrolladores originales de World Online todavía aportan una guía centralizada para el crecimiento del framework, y World Online colabora con otros aspectos importantes tales como tiempo de trabajo, materiales de marketing, y hosting/ancho de banda para el Web site del framework (http://www.djangoproject.com/).

Esta historia es relevante porque ayuda a explicar dos cuestiones clave. La primera es el "punto dulce" de Django. Debido a que Django nació en un entorno de noticias, ofrece varias características (en particular la interfaz admin, tratada en el **'Capítulo 6'_**) que son particularmente apropiadas para sitios de "contenido" – sitios como eBay, craigslist.org y washingtonpost.com que ofrecen información basada en bases de datos –. (De todas formas, no dejes que eso te quite las ganas – a pesar de que Django es particularmente bueno para desarrollar esa clase de sitios, eso no significa que no sea una herramienta efectiva para crear cualquier tipo de sitio Web dinámico –. Existe una diferencia entre ser *particularmente efectivo* para algo y *no ser efectivo* para otras cosas).

La segunda cuestión a resaltar es cómo los orígenes de Django le han dado forma a la cultura de su

comunidad de código abierto. Debido a que Django fue extraído de código de la vida real, en lugar de ser un ejercicio académico o un producto comercial, está especialmente enfocado en resolver problemas de desarrollo Web con los que los desarrolladores de Django se han encontrado – y con los que continúan encontrándose –. Como resultado de eso, Django es activamente mejorado casi diariamente. Los desarrolladores del framework tienen un alto grado de interés en asegurarse de que Django les ahorre tiempo a los desarrolladores, produzca aplicaciones que son fáciles de mantener y rindan bajo mucha carga. Aunque existan otras razones, los desarrolladores están motivados por sus propios deseos egoístas de ahorrarse tiempo a ellos mismos y disfrutar de sus trabajos. (Para decirlo sin vueltas, se comen su propia comida para perros).

6.2 Modelos

La forma de interactuar con la base de datos es a traves de los modelos, siendo consistente con Django, Doff abstrae la logica de negocios dentro de los modelos.

Continuando con la linea de ejemplos se presenta a continuación una configuración de datos básica sobre libro/autor/editor, esta estructura debe estar contenida en un archivo models.js dentro de una aplicacion:

```
var models = require('doff.db.models.base');

var Publisher = type('Publisher', [ models.Model ], { name: new models.CharField({ maxlength: 30 }), address: new models.CharField({ maxlength: 50 }), city: new models.CharField({ maxlength: 60 }), state_province: new models.CharField({ maxlength: 30 }), country: new models.CharField({ maxlength: 50 }), website: new models.URLField()

});

var Author = type('Author', [ models.Model ], { salutation: new models.CharField({ maxlength: 10 }), first_name: new models.CharField({ maxlength: 30 }), last_name: new models.CharField({ maxlength: 40 }), email: new models.EmailField(), headshot: new models.ImageField({ upload_to: '/tmp' })

});

var Book = type('Book', [ models.Model ], { title: new models.CharField({ maxlength: 100 }), authors: new models.ManyToManyField(Author), publisher: new models.ForeignKey(Publisher), publication_date: new models.DateField()

});
```

Cada modelo es representado por un tipo de Protopy que es un subtipo de `doff.db.models.model.Model`. El tipo `Model` contiene toda la maquinaria necesaria para hacer que los nuevos tipos sean capaces de interactuar con la base de datos.

Cada modelo se corresponde con una tabla única de la base de datos, y cada atributo de un modelo con una columna en esa tabla. El nombre de atributo corresponde al nombre de columna, y el tipo de campo corresponde al tipo de columna de la base de datos. Por ejemplo, el modelo `Publisher` es equivalente a la siguiente tabla:

> **CREATE TABLE "books_publisher"** ("id" serial NOT NULL PRIMARY KEY, "name" varchar(30) NOT NULL, "address" varchar(50) NOT NULL, "city" varchar(60) NOT NULL, "state_province" varchar(30) NOT NULL, "country" varchar(50) NOT NULL, "website" varchar(200) NOT NULL
>
>);

La excepción a la regla una-clase-por-tabla es el caso de las relaciones muchos-a-muchos. En el ejemplo, `Book` tiene un `ManyToManyField` llamado `authors`. Esto significa que un libro tiene uno o más autores, pero la tabla de la base de datos `Book` no tiene una columna `authors`. En su lugar, se crea una tabla adicional que maneja la correlación entre libros y autores.

Para una lista completa de tipos de campo y opciones de sintaxis de modelos, ver el Apéndice B.

Finalmente, no se define explícitamente una clave primaria en ninguno de estos modelos. A no ser que se le indique lo contrario, Doff dará automáticamente a cada modelo un campo de clave primaria entera llamado `id`.

Para activar los modelos en el proyecto, la aplicación que los contine debe estar incluida en la lista de aplicaciones instaladas de Doff. Esto es, edita el archivo `settings.js`, y examina la variable de configuración `INSTALLED_APPS`

Posteriormente, cuando el usuario instala la aplicación en su navegador, el sistema recorre las aplicaciones en `INSTALLED_APPS` y genera el SQL para cada modelo, creando las tablas en la base de datos.

Doff provee entre las herramientas del desarrollador, un interprete de SQL sobre la base de datos del cliente para conslutas.

6.2.1 Acceso básico a datos

Una vez que se creo el modelo, Doff provee automáticamente una API JavaScript de alto nivel para trabajar con estos modelos:

```
>>> require('books.models', 'Publisher');
>>> p1 = new Publisher({ name: 'Addison-Wesley', address: '75 Arlington Street',
...     city: 'Boston', state_province: 'MA', country: 'U.S.A.',
...     website: 'http://www.apress.com/'});
>>> p1.save();
>>> publisher_list = Publisher.objects.all();
>>> print(array(publisher_list));
[<Publisher: Publisher object>]
```

Se puede hacer mucho con la API de base de datos de Doff y para mejorar la interactividad se recomienda implementar __str__ de Protopy. Con este método los objetos tendran su representación en "string", es imporante que eso sea así ya que el framework utiliza esta representacion en muchos lugares, como templates y salidas por consola.

6.2.2 Insertando y actualizando datos

En el ejemplo presentado anteriormente se ve cómo se hace para insertar una fila en la base de datos, primero se crea una instancia del modelo pasando argumentos nombrados y luego se llama al método `save()` del objeto:

En el caso de `Publisher` se usa una clave primaria autoincremental `id`, por lo tanto la llamada inicial a `save()` hace una cosa más: calcula el valor de la clave primaria para el registro y lo establece como el valor del atributo `id` de la instancia.

Las subsecuentes llamadas a `save()` guardarán el registro en su lugar, sin crear un nuevo registro (es decir, ejecutarán una sentencia SQL `UPDATE` en lugar de un `INSERT`).

6.2.3 Seleccionar objetos

La forma de seleccionar y tamizar los datos se consige a traves de los administradores de consultas, en el ejmplo la linea `Publisher.objects.all()` pide al administrador objects de Publisher que obtenga todos los registros, internamente esto genera una consulta SQL:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher;
```

Todos los modelos automáticamente obtienen un administrador `objects` que debe ser usado cada vez que se quiera consultar sobre una instancia del modelo. El metodo `all()` es un método del administrador `objects` que retorna todas las filas de la base de datos. Aunque este objeto se *parece* a una lista, es actualmente un *QuerySet* – un objeto que representa algún conjunto de filas de la base de datos. El Apéndice C describe QuerySets en detalle.

Cualquier búsqueda en base de datos va a seguir esta pauta general.

6.2.4 Filtrar datos

Aunque obtener todos los objetos es algo que ciertamente tiene su utilidad, la mayoría de las veces lo que vamos a necesitar es manejarnos sólo con un subconjunto de los datos. Para ello usaremos el método `filter()`:

```
>>> Publisher.objects.filter(name="Apress Publishing")
[<Publisher: Apress Publishing>]
```

`filter()` toma argumentos de palabra clave que son traducidos en las cláusulas SQL `WHERE` apropiadas. El ejemplo anterior sería traducido en algo como:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
WHERE name = 'Apress Publishing';
```

Puedes pasarle a `filter()` múltiples argumentos para reducir las cosas aún más:

```
>>> Publisher.objects.filter(country="U.S.A.", state_province="CA")
[<Publisher: Apress Publishing>]
```

Esos múltiples argumentos son traducidos a cláusulas SQL `AND`. Por lo tanto el ejemplo en el fragmento de código se traduce a lo siguiente:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
WHERE country = 'U.S.A.' AND state_province = 'CA';
```

Notar que por omisión la búsqueda usa el operador SQL = para realizar búsquedas exactas. Existen también otros tipos de búsquedas:

```
>>> Publisher.objects.filter(name__contains="press")
[<Publisher: Apress Publishing>]
```

Notar el doble guión bajo entre `name` y `contains`. Del mismo modo que Python, Django usa el doble guión bajo para indicar que algo "mágico" está sucediendo – aquí la parte `__contains` es traducida por Django en una sentencia SQL `LIKE`:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
WHERE name LIKE '%press%';
```

Hay disponibles varios otos tipos de búsqueda, incluyendo `icontains` (`LIKE` no sensible a diferencias de mayúsculas/minúsculas), `startswith` y `endswith`, y `range` (consultas SQL `BETWEEN`). El Apéndice C describe en detalle todos esos tipos de búsqueda.

6.2.5 Obteniendo objetos individuales

En ocasiones desearás obtener un único objeto. Para esto existe el método `get()`:

```
>>> Publisher.objects.get(name="Apress Publishing")
<Publisher: Apress Publishing>
```

En lugar de una lista (o más bien, un QuerySet), este método retorna un objeto individual. Debido a eso, una consulta cuyo resultado sean múltiples objetos causará una excepción:

```
>>> Publisher.objects.get(country="U.S.A.")
Traceback (most recent call last):
    ...
AssertionError: get() returned more than one Publisher -- it returned 2!
```

Una consulta que no retorne objeto alguno también causará una excepción:

```
>>> Publisher.objects.get(name="Penguin")
Traceback (most recent call last):
    ...
DoesNotExist: Publisher matching query does not exist.
```

6.2.6 Ordenando datos

A medida que juegas con los ejemplos anteriores, podrías descubrir que los objetos so devueltos en lo que parece ser un orden aleatorio. No estás imaginándote cosas, hasta ahora no le hemos indicado a la base de datos cómo ordenar sus resultados, de manera que simplemente estamos recibiendo datos con algún orden arbitrario seleccionado por la base de datos.

Eso es, obviamente, un poco ***silly*** (tonto), no querríamos que una página Web que muestra una lista de editores estuviera ordenada aleatoriamente. Así que, en la práctica, probablemente querremos usar `order_by()` para reordenar nuestros datos en listas más útiles:

```
>>> Publisher.objects.order_by("name")
[<Publisher: Apress Publishing>, <Publisher: Addison-Wesley>, <Publisher: O'Reilly>
```

Esto no se ve muy diferente del ejemplo de `all()` anterior, pero el SQL incluye ahora un ordenamiento específico:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
ORDER BY name;
```

Podemos ordenar por cualquier campo que deseemos:

```
>>> Publisher.objects.order_by("address")
[<Publisher: O'Reilly>, <Publisher: Apress Publishing>, <Publisher: Addison-Wesley>

>>> Publisher.objects.order_by("state_province")
[<Publisher: Apress Publishing>, <Publisher: Addison-Wesley>, <Publisher: O'Reilly>
```

y por múltiples campos:

```
>>> Publisher.objects.order_by("state_provice", "address")
 [<Publisher: Apress Publishing>, <Publisher: O'Reilly>, <Publisher: Addison-Wesley
```

También podemos especificar un ordenamiento inverso antecediendo al nombre del campo un prefijo – (el símbolo menos):

```
>>> Publisher.objects.order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress Publishing>, <Publisher: Addison-Wesley>
```

Aunque esta flexibilidad es útil, usar `order_by()` todo el tiempo puede ser demasiado repetitivo. La mayor parte del tiempo tendrás un campo particular por el que usualmente desearás ordenar. Es esos casos Django te permite anexar al modelo un ordenamiento por omisión para el mismo:

Este fragmento `ordering = ["name"]` le indica a Django que a menos que se proporcione un ordenamiento mediante `order_by()`, todos los editores deberán ser ordenados por su nombre.

6.2.7 Encadenando búsquedas

Has visto cómo puedes filtrar datos y has visto cómo ordenarlos. En ocasiones, por supuesto, vas a desear realizar ambas cosas. En esos casos simplemente "encadenas" las búsquedas entre sí:

```
>>> Publisher.objects.filter(country="U.S.A.").order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress Publishing>, <Publisher: Addison-Wesley>
```

Como podrías esperar, esto se traduce a una consulta SQL conteniendo tanto un `WHERE` como un `ORDER BY`:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
WHERE country = 'U.S.A'
ORDER BY name DESC;
```

Puedes encadenar consultas en forma consecutiva tantas veces como desees. No existe un límite para esto.

6.2.8 Rebanando datos

Otra necesidad común es buscar sólo un número fijo de filas. Imagina que tienes miles de editores en tu base de datos, pero quieres mostrar sólo el primero. Puedes hacer eso usando la sintaxis estándar de Python para el rebanado de listas:

```
>>> Publisher.objects.all()[0]
<Publisher: Addison-Wesley>
```

Esto se traduce, someramente, a:

```
SELECT
    id, name, address, city, state_province, country, website
FROM book_publisher
ORDER BY name
LIMIT 1;
```

6.2.9 Eliminando objetos

Para eliminar objetos, simplemente se debe llamar al método `delete()` del objeto:

```
>>> p = Publisher.objects.get({ name: "Addison-Wesley" });
>>> p.delete();
>>> array(Publisher.objects.all());
[]
```

Se pueden borrar objetos al por mayor llamando a `delete()` en el resultado de una búsqueda:

```
>>> publishers = Publisher.objects.all();
>>> publishers.delete();
>>> array(Publisher.objects.all());
[]
```

Nota: Mixin en RemoteSite, los modelos se registran en RemoteSite o se pueden hacer a mano. Hacer la referencia correspondiente al anexo de modelos

6.3 Plantillas

Doff brinda soporte al sistema de plantillas de Django. La idea detras de esto es que las plantillas escritas para una apilcacion on-line en Django pueda ser utilizada en la aplicación off-line de una forma consistente.

Nota: Existen sin emabargo algunos factores a tener en cuenta a la hora de realizar plantillas para aplicaciones offline. Cuando una aplicación se ejecuta de manera desconectada, el browser solo realiza una carga de documento y la navegación se basa en la inserción y supresión de nodos sobre el elemento document. Esto implica que el estado de la aplicación

Retomando el ejemplo presentado en la seccion de plantillas en Django, continuamos desde aquí describiendo como es el trabajo con esta plantilla en Doff.

Rapidamente, la forma de obtener un producto de la plantilla es:

 # Crea un objeto `Template` brindando el código crudo de la plantilla como una cadena.

 # Llama al método `render()` del objeto `Template` con un conjunto de variables (o sea, el contexto). Este retorna una plantilla totalmente renderizada como una cadena de caracteres, con todas las variables y etiquetas de bloques evaluadas de acuerdo al contexto.

6.3.1 Creación de objetos Template

Doff provee su versión del objeto `Template` para crear plantillas, y esté puede ser importado del módulo `doff.template.base`, el argumento para la construccion del objeto es el texto en crudo de la plantilla.

```
>>> require('doff.template.base', 'Template');
>>> t = new Template("Mi nombre es {{ name }}.");
>>> print(t);
```

En este ejemplo *t* es un objeto template listo para ser renderizado. Si se obtubo el objeto es porque la plantilla esta correctamente analizada y no se econtraron errores en la misma, algunos errores por los que puede fallar la construccion son:

- Bloques de etiquetas inválidos
- Argumentos inválidos de una etiqueta válida
- Filtros inválidos
- Argumentos inválidos para filtros válidos
- Sintaxis de plantilla inválida
- Etiquetas de bloque sin cerrar (para etiquetas de bloque que requieran la etiqueta de cierre)

En todos los casos el sistema lanza una excepción `TemplateSyntaxError`.

6.3.2 Renderizar una plantilla

Una vez que se tiene el objeto `Template`, se esta en condiciones de obtener una salida procesada en un determinado *contexto*. Un contexto es simplemente un conjunto de variables y sus valores asociados. Una plantilla usa las variables para poblar la plantilla evaluando las etiquetas de bloque.

El contexto esta representado en el tipo `Context`, el cual se encuentra en el módulo `doff.template.base`. La construccion del objeto toma un argumento opcional: un hash mapeando nombres de variables con valores. La llamada al método `render()` del objeto `Template` con el contexto "rellena" la plantilla:

```
>>> requiere('doff.template.base', 'Context', 'Template');
>>> t = new Template("Mi nombre es {{ name }}.")
>>> c = new Context({"name": "Pedro"})
>>> t.render(c)
'My name is Pedro.'
```

El objeto `Template` puede ser renderizado con múltiples contextos, obteniendo así salidas diferentes para la misma plantilla. Por cuestiones de eficiencia es conveniente crear un objeto `Template` y luego llamar a `render()` sobre este muchas veces:

```
# Mal chico
for each (var name in ['John', 'Julie', 'Pat']) {
    var t = new Template('Hello, {{ name }}');
    print(t.render(new Context({'name': name})));
}

# Buen chico
t = new Template('Hello, {{ name }}');
for each (var name in ['John', 'Julie', 'Pat'])
    print(t.render(new Context({'name': name})));
```

Al igual que en Django el objeto contexto puede contener variables mas complejas y la forma de inspeccionar dentro de estas es con el operador .. Usando el punto se puede acceder a objetos, atributos, índices, o métodos de un objeto.

Cuando un sistema de plantillas encuentra un punto en una variable el orden de busqueda es el siguiente:

- Diccionario (por ej. `foo["bar"]`)
- Atributo (por ej. `foo.bar`)
- Llamada de método (por ej. `foo.bar()`)
- Índice de lista (por ej. `foo[bar]`)

El sistema utiliza el primer tipo de búsqueda que funcione. Es la lógica de cortocircuito.

Para terminar con este objeto, si una variable no existe en el contexto, el sistema de plantillas renderiza este como un string vacío, fallando silenciosamente. Es posible cambiar este comportamiento modificando el valor de la variable de configuración *TEMPLATE_STRING_IF_INVALID* en el módulo *settings*.

6.3.3 Cargador de plantillas

Se ve a continuación un ejemplo de una vista que retorna HTML generado por una plantilla:

```
require('doff.template.base', 'Template', 'Context');
require('doff.utils.http', 'HttpResponse');
require('doff.template.loader', 'get_template');

function current_datetime(request) {
    var t = get_template('mytemplate.html');
    html = t.render(new Context({'current_date': new Date()}))
    return new HttpResponse(html);
}
```

En esta vista se utiliza la API para cargar plantillas, a la cual se accede mediante la funcion *get_template*, antes de poder utilizar esta funcion es necesario indicarle al framework donde están las plantillas. El lugar para hacer esto es en el *archivo de configuración*.

Existen varios cargadores de plantillas que se pueden habilitar en el archivo de configuración, este se vera en profundidad en el Apéndice E, por ahora se vera el cargador realacionado con la variable de configuración `TEMPLATE_URL`. Esta variable le indica al mecanismo de carga de plantillas dónde buscar las plantillas. Por omisión, ésta es una cadena vacia. El valor para esta varible es la url del servidor en donde se sirven los templates que se renderizan localmente, por defecto si no se especifica la variable los archivos se buscan en la base del soporte off-line /templates/.

Nota: Terminar esto. con el tema de la base del soporte off-line Introducir el concepto al inicio de doff Hacer la referencia correspondiente al anexo de plantillas

6.4 Formularios

CAPÍTULO 7

La aplicación

Una vez lograda la implementación de Django en el clinete sobre el framework de aplicaciones Doff, aparece la necesidad de conectar el proyecto online con el browser. Surge la necesidad de crear un equivalente al proyecto para ser instalado en el clinete.

El desarrollo de los doff.models sigue el esquma de aplicaciones de Django.

7.1 Definición de un proyecto en el cliente

Nota: hacer la referencia al capítulo de django

Doff fue desarrollado con el objetivo de realizar la menor cantidad de reescritura posible de las aplicaciones Django para ser ejecutadas de manera desconectada. La transferencia del proyecto al clinete conlleva la transferencia de la configuración, las aplicaciones y la base de datos.

Una aplicación está conformada por al menos un módulo de vistas, uno de urls y un módulo de modelos.

En el caso de las vistas, estas deben ser reescritas debido a que si bien Porotpy está diseñado para que los dearrolladores encunetren en Javascript 1.7 una sintaxis y semántica cercana a la de Python, pueden existir módulos de la API de Python que Protopy no soprte. Así como también pueden hacer uso de módulos de terceros, el caso de ReportLab para la genreación de PDF, MatplotLib para la generación de gráficas, o alguna otra librería.

Por lo tanto vemos que no podemos realizar una reescritura uno a uno de las vistas. Además las vists que se rendericen sobre templates que accedan a datos o servicios de otros dominios tampoco pueden ser transformadas de modo directo, sinedo un caso dificilmente salvable.

En cuanto a la base de datos por razones de seguirad y eficiencia, es posible que no se desee realizar una transferencia total de los datos al cliente.

Por ejemplo, tablas que almacenan datos con información privada o confidencial, también es posible que un usuario o grupo tenga diferentes niveles de accesso sobre los datos que otros.

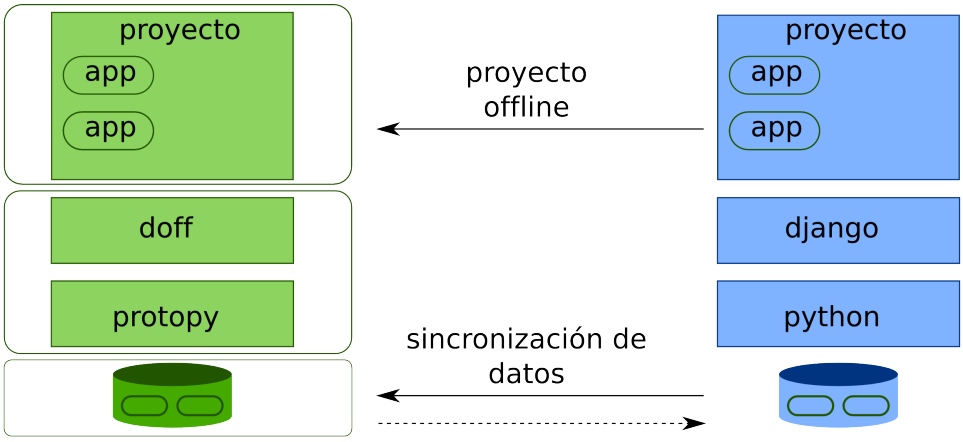


Figura 7.1: Análisis inicial de elementos a transferir para lograr una versión offline de un proyecto

Se arriba a la conclusión de que una aplicación offline puede diferir tanto en funcionalidad como en datos de una versión en linea. Por lo tanto se hace necesario definir un proyecto offline, analizando como se realiza la equivalencia de cada elemento de la versión offline.

7.2 Módulo de vistas y urls en una aplicación offline

Como hemos se analizó en el apartado anterior, las vistas deben ser reescritas para la versión offline.

7.3 Bootstrap

7.4 Transferencia de los modelos

Debido a que la API de base de datos posee diferencias mínimas, la transferencia de la definición de los modelos al cliente se ideó de tal forma que sea posible mediante introspección.

Para que el clinete conozca la defínción de un modelo, realiza una

7.5 Sitio Remoto

Se creó la entidad RemoteSite para definir un proyecto desconectado. Un proyecto puede tener uno o más RemoteSites. Cada sitio remoto está publicado en una URL.

Un sitio remoto consiste en una instancia de una clase que posee un nombre. Este nombre coincide con el nombre del directorio que almacena las vistas.

El sitio remoto tiene la responsabilidad de servir el código del framework doff y el proyecto offline (vistas, modelos, urls, templates).

CAPÍTULO 8

Sincronizacion

Se ha analizado hasta aquí la transferencia de un proyecto Django a un equivalente en un browser con soprte para Javascript 1.7 y GoogleGears. Tras la instalación de la aplicación en el cliente, se requiere la transferencia de los datos.

Muchas aplicaciones requieren cierto contendio de datos iniciales o de trabajo para poder funcionar. Estos datos residen en la base de datos del servidor y para que la aplicación offline pueda trabajar, es necesario proveer un al menos un mecanismo de transferencia de datos desde el servidor hacia el cliente.

Debido a que el desarrollo se realizó apegandose al las estructuras de Django, tanto para la definición de proyecto, como de las aplicaciones y sus componentes, se decide realizar una sincronización a nivel ORM y no a bajo nivel, es decir, se utiliza la API de acceso a datos de Django, en vez de plantear sincronización mediante SQL.

En el framework Django, el acceso a la base de datos se realiza mediante el mapeador objeto relacional, en particular para la las consultas, mediante las instancias de Managers definidas en cada modelo. En Doff se birnda la misma API, de menara que el acceso a datos en cualquiera de las aplicaciones es transparente.

8.1 Sincronización simple de servidor a cliente

Cada entidad del modelo definido en la aplicación del servidor posee al menos el manager *objects*, que equivale a un consulta por todas las filas de la tabla a la cual está asociada la entidad.

Django provee un mecanismo de serialización de QuerySets, que son los objetos que encapsulan las consultas, en varios formatos.

Figura 8.1: Esquema de sincronización simple

8.2 Identificación de instancias en el servidor

Nota: No se donde está la aclaración de pk e id para hacer la aclaración

Django provee un sub frameowork llamado *Content Types Framework* que permite generar relaciones genéricas en los instancias arbitrarias del modelo. Cada modelo posee un identificador único entero en Conent Types y en conjunción con la clave del objeto (id o pk) se obtiene una calve única para cualquier instancia, sin importar su tipo.

8.2.1 Modelos

Los modelos que son sincronizables, en el cliente deben extender al tipo *SyncModel*.

A estos modelos se los provee con un Proxy que interactua con el servidor para service de los datos

CAPÍTULO 9

Conclusiones y lineas futuras

9.1 Conclusiones

El desarrollo consistió en el siguiente diagrama:

9.2 Lineas futuras

9.2.1 Sitio de administración

Django se caracteriza por brindar una aplicación `django.contrib.admin` de admnistración que permite realizar CRUD (Create, Retrieve, Update, Delete) sobre los modelos de las aplicaciones de usuario, interactuando con la aplicación `django.contrib.auth` que provee usuarios, grupos y permisos.

9.2.2 Historial de navegación

9.2.3 Workers con soporte para Javascript 1.7

Google Gears provee un mecanismo de ejecución de código en el cliente de manera concurrente llamado Worker Pool. De esta manera tareas que demandan tiempo de CPU pueden ser envaidas a segundo plano, de manera de no entorpecer el refresco de la GUI. Una caractersística de los worker pools, es que se ejecutan en un ámbito de nombres diferente al del ‘‘hilo principal’‘. Es decir, existe encapsulamiento de su estado.

CAPÍTULO 10

Glossary

.net Plataforma de desarrollo creada por Microsoft.

API Application-Programming-Interface; conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

BSD ve ese de

CGI Common Gateway Interfase

deployment Etapa en el dearrollo de sistemas en la cual el producto es puesto en producción. El deployment involucra todas las actividades necesarias para poner el sistema en funcionamiento para el usuario final.

Deplpyment Etapa del desarrollo que consiste en la puesta en producción de una aplicación

DOM Document-Object-Model; interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

field An attribute on a *model*; a given field usually maps directly to a single database column.

generic view A higher-order *view* function that abstracts common idioms and patterns found in view development and abstracts them.

HTML Lenguaje de hipertexto basado en etiquetas de marcado

HTTP Hyper Text Transfer Protocol

i18n La internacionalización es el proceso de diseñar software de manera tal que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código. La localización es el proceso de adaptar el software para una región específica mediante la adición de componentes específicos de un locale y la traducción de los textos,

por lo que también se le puede denominar regionalización. No obstante la traducción literal del inglés es la más extendida.

JSON JavaScript-Object-Notation; formato ligero para el intercambio de datos.

MIME Estandar de especificación de tipo de contenido para correo electrónico utilizado también en encabezados HTTP.

model Models store your application's data.

MTV hola

MVC Model-view-controller; a software pattern.

PHP Lenguaje de programación diseñado para ser incrustado en documentos HTML.

project A Python package – i.e. a directory of code – that contains all the settings for an instance of Django. This would include database configuration, Django-specific options and application-specific settings.

property Also known as "managed attributes", and a feature of Python since version 2.2. From the property documentation:

> Properties are a neat way to implement attributes whose usage resembles attribute access, but whose implementation uses method calls. [...] You could only do this by overriding `__getattr__` and `__setattr__`; but overriding `__setattr__` slows down all attribute assignments considerably, and overriding `__getattr__` is always a bit tricky to get right. Properties let you do this painlessly, without having to override `__getattr__` or `__setattr__`.

queryset An object representing some set of rows to be fetched from the database.

RPC Remote-Procedure-Call; es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

Script Porgrama escrito en un lenguaje interpretado

slug A short label for something, containing only letters, numbers, underscores or hyphens. They're generally used in URLs. For example, in a typical blog entry URL:

```
http://www.djangoproject.com/weblog/2008/apr/12/spring/
```

the last bit (`spring`) is the slug.

template A chunk of text that separates the presentation of a document from its data.

URL Localizador universal de recursos, especificada en la RFC 2396

view A function responsible for rending a page.

XHTML Lenguaje de hipertexto basado en etiquetas de marcado que no viola la esepcificación HTML

Bibliografía

[ApacheMod2009] *Módulos del servidor Apache 2.2*, último acceso, Septiembre de 2009, http://httpd.apache.org/docs/2.2/mod/

[MicrosoftIIS2009] *Módulos en Microsoft IIS*, último acceso Septiembre 2009, http://msdn.microsoft.com/en-us/library/bb757040.aspx

[ApacheTomcat2009] http://tomcat.apache.org/index.html

[SunServlet2009] http://java.sun.com/products/servlet/

[WikiCGI2009] *Interfaz de entrada común*, Wikipedia, 2009, último acceso Agosto 2009, http://es.wikipedia.org/wiki/Common_Gateway_Interface#Intercambio_de_informaci.C3.B3n:_Variables_de

[DavidPollak2006] *Ruby Sig:How To Design A Domain Specific Language*, Google Tech Talk, 2:44, http://video.google.com/videoplay?docid=-8103284744220333344

[WikiPlataforma2009] *Multiplataforma*, Wikipedia, 2009, último acceso, Agosto de 2009, http://es.wikipedia.org/wiki/Multiplataforma

[Cowan2005] *RESTful Web Services, An introduction to building Web Services without tears (i.e., without SOAP or WSDL)*, 2005, http://home.ccil.org/~cowan/restws.pdf

[BlogHardz2008] http://hardz.wordpress.com/2008/02/07/php-hipertexto-pre-procesado/

[EricRaymon2000] *Why Python*, Linux Journal, publicado el 1° de Mayo de 2000, http://www.linuxjournal.com/article/3882

[WIK001] *Software Framework*, Wikipedia, 2009, http://en.wikipedia.com/software_framework, última visita Agosto de 2009.

[Tryg1979] Trygve Reenskaug, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[SmallMVC] Steve Burbeck, Ph.D. http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[WIKI002] *Web Framework*, Wikipedia, 2009, http://en.wikipedia.org/wiki/Web_application_framework, última visita Agosto de 2009.

[DjangoDoc2009] *Sirviendo archivos esáticos con Django*, Django Wiki, http://docs.djangoproject.com/en/dev/howto/static-files/#the-big-fat-disclaimer

[WikiSSL2009] *Transport Layer Security*, Wikipedia, 2009, último acceso,

[StephenChapmanJS2009] *Javascript and XML*, Stephen Chapman, About.com,

[W3cCSS2009] *Guia breve de CSS*, W3C, español, último acceso Agosto 2009,

[IronPythonFAQ2009] *Diferencias entre IronPython y CPython* http://ironpython.codeplex.com/Wiki/View.aspx?title=IPy2.0.xCPyDifferences&referringTitle=Home

[AtulVarma2009] *Python For Javascript Programmers*, Atul Varma http://hg.toolness.com/python-for-js-programmers/raw-file/tip/PythonForJsProgrammers.html

Índice