# Sistemas Web Desconectados

Defossé Nahuel, van Haaster Diego Marcos

1 de junio de 2009

# Índice general

# Índice de figuras

# Índice de cuadros

# Agradecimientos

A nuestros familiares

**Resumen**

# Capítulo 1

# Introducción

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 para obtener información, su constante expanción a convetido a esta red en un terreno muy atractivo para la implementación de sistemas de información.

Las aplicaciones web son populares debido a lo práctico que resulta el navegador web como cliente de acceso a las mismas. También resulta fácil actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. En la actualidad, existe una gran oferta de frameworks web para facilitar el desarrollo de aplicaciones web. 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. 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 ...

http://es.wikiquote.org/wiki/The$_M atrix$

## 1.3. Alcance

Aca ponemos hasta donde nos vamos a llegar.

# Capítulo 2

# Capítulo 3

# Tecnologías Web

## 3.1. Recursos

*Acá hablamos brevemente de Mozilla (cuentito de como evuoluciona Javascript), Javascript y como se perfila como estanadar de interconexión de sistemas (mashups), Python y CGI).—*

## 3.2. Lenguajes de programación para la web

## 3.3. El lenguaje de programación Python

El lenguaje de programación Python fue creado por Guido van Rossum en el año 1991. Python es un lenguaje de programación multiparadigma. Es decir, permite al programador utilizar diferentes formas de resolución de problemas

- programación orientada a objetos
- programación estructurada
- programación funcional

Otros paradigmas (como programación lógica) están soportados mediante el uso de extensiones [1][2].

Python usa tipo de dato dinámico y reference counting para el manejo de memoria. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Nuevos módulos se pueden escribir fácilmente en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. Aunque el diseño de Python es de alguna manera hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

[1] PySWIP http://code.google.com/p/pyswip/
[2] python-logic

### Filosofía del lenguaje

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico"(ünpythonic.[en] inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en *El Zen de Python*

1. Bello es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Ralo es mejor que denso.
7. La legibilidad cuenta.
8. Los casos especiales no son tan especiales como para quebrantar las reglas.

   Aunque lo práctico gana a la pureza.
9. Los errores nunca deberían dejarse pasar silenciosamente.

   A menos que hayan sido silenciados explícitamente.
10. Frente a la ambigüedad, rechaza la tentación de adivinar.
11. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.

    Aunque esa manera puede no ser obvia al principio a menos que usted sea Holandés
12. Ahora es mejor que nunca

    Aunque nunca es a menudo mejor que ya
13. Si la implementación es dificil de explicar, es una mala idea.
14. Si la implementacion es fácil de explicar, puede que sea una buena idea
15. Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Desde la versión 2.1.2, Python incluye estos puntos (en su versión original en inglés) como un huevo de pascua que se muestra al ejecutar

```
>>> import this
```

### 3.3.1. Desarrollo del lengauje

### 3.3.2. Tipos de datos y funciones

Python cuenta con los siguientes tipos de datos incorporados: *int*, *str*, *bool*, *float*, *complex*. Estos tipos de datos son denominados **inmutables**, es decir, para que su valor cambie, la instancia anterior es destruida.

Python no cuenta con el tipo de dato arreglo, sin embargo cuenta con 3 sustitutos. El tipo de dato tupla (*tuple()*) es un tipo de dato inmutable, que agrupa de manera *ordenada* un conjuntod de objetos del mismo o diferente tipo. El concepto de orden de los elementos es importante, ya que python tampoco provee la estrutura *for* tradicional de lenguajes procedurales como C o Pascal. Una tupla puede ser considreada como un arreglo fijo de valores que pueden tener diferentes tipos. Las tuplas pueden ser accedidas secuencialmente (*iteradas*) en orden ascendente o descendente, se puede acceder al n-ésimo item, pero no puede ser modificadas se generará una excepción debido a que es un tipo de datos estático.

```
a = (1, 2, 3) # Una tupla de enteros
b = (1, 1.0, 'pasta') # Una tupla con un conjuto dispar
                      # de elementos
c = ( (1, 2), (3, 4), (5, 6)) # Una tupla que tiene
                              # por elmentos otras tuplas
```

Python soprta asinganción a con la sintaxis de tupla, es decir, al colocar un conjunto de nombres separados por comas en el lado derecho de una asingación y un conjunto de valores o variables sobre la izquerda, la asingación es posicional.

Listing 3.1: Asignacion en Python

```
a, b = 1, 2     # Es equivalente a (a, b) = (1, 2)
a, b = b, a     # Operaci n de intercambio sin auxiliar
c = (1, 2, 3, 'bacalao')
w, x, y, z = c  # Asignaci n posicional
```

Listing 3.2: Asignacion en Python

```
function pepe(){}
```

Una lista es una coección ordenada de objetos del mismo o diferente tipo que pueden ser modificados. Básicamente la lista se comporta como una tupla mutable. a = [1, 2, 3, 4 ] b[0] = 4

Los *bloques* en Python son conjuntos de sentencias que se encuentran a bajo un cierto nivel de inetación. Un bloque comienza con dos puntos (:) y termina cuando se pierde el nivel de indetación que lo definía. Veremos ejemplos de bloques con las sentencias de control de flujo.

#### Bloques y estructuras de control

La esentencia *if* ejecuta el if condicion: bloque$_s i_c ondicion_v erdaderaelse$ : $bloque_s i_c odicion_f alsa$

### 3.3.3. Espacio de nombres en Python

Cuando comenzamos a ejecutar el intérprete de Python de forma interactiva o cuando lo invocamos para ejecutar un script, están diponibles un conjuto de funciones, clases, clases de excepciones. El conjunto de estos nombres se conoce como ámbito de nombres __builtin__ (*incorporado*) y en este encontramos: int(), char(), str(), object, Exception, TypeError, ValueError, True, False, range, map, zip, locals, globals, entre otras. Pueden verse en el interprete interactivo mediante

```
>>> dir(__builtins__)
```

Un *ámbito de nombres* en Python es un mapeo de nombres a objetos. La mayoría de los ámbitos de nombres están implementados mediante diccionarios, but that's normally not noticeable in any way (except for performance), y podría cambiar en el futuro. Otros ejemplos de ámbitos de nombre son los nombres globales en un módulo; los valores locales en la invocación de una función y en cierta forma, los atributos de un objeto también forman un ámbito de nombres. Es importante tener en cuenta que no existe ninguna relación entre nombres de diferentes ámbitos de nobres.

Un *módulo* es un archivo con definiciones y sentencias en Python. El conjunto de todas las sentencias que no están indentadas, forman el ámbito de nombres global del modulo, las funciones, variables y clases que están suelen decirse que estan a "nivel módulo". Los elementos que pueden ser parete del ámbito de nombres del módulo son:

- una definición de una función
- una vairable
- una expresión lambda
- el nombre de otro módulo importado

Un número, el llamado a función o el lanzado de una excepción no alteran el ámbito de nombres. Un módulo puede ser cargado desde el intérprete interactivo en el ámbito de nombres global o desde otro módulo (en su propio espacio de nombres) a través de la sentencia *import* import os En este caso, el intérprete buscará secuencialmente en el PYTHONPATH un módulo llamado **os**, en caso de encontrarlo, lo cargará y en el ámbito de nombres actual generará una referencia con el nombre "os". A esta actividad se la llama importación y el interprete garantiza que un módulo se cargará de unicamente una sola vez, en otras palabras, existe una única instancia de un módulo. En este caso, si varios módulos importan a "os", solo la primer ocurrencia realiza la importación efetiva. El resto de las sentencias import solamente generan referencias a la primera "instancia" de un módulo.

La sentencia import permite a su vez hacer una importación selectiva de símbolos de un módulo, por ejemplo: from os import path En este caso, import crea la instancia del módulo en la máquina virtual, pero no expone todo el contenido al ámbito de nombres local, sino que únicamente genera una referencia al símbolo *path*.

El comodín * permite importar todos los símbolos definidos en un módulo al espacio de nombres local. from os import *

Está técnica debe ser utilizada con cuidado, debido a que "contamina" el ámbito de nombres local. Para mitigar este problma, puede definirse una variable **__all__** al comienzo del archivo, con una tupla en la cual se definen los símbolos que se desean exportar. Suponendo que el siguiente código se encuentra en el módulo promedio:

$^{a}ll_{=('calcular_promedio',)}$
def calular$_{p}romedio(numeros) : returnsuma(numeros)/len(numeros)defsuma(lista) : returnfloat(sum(lista))$

Al realizar from promedio import *

El único símbolo que encontraremos en nuestro espacio de nombres será, *calcular_promedio*.

### 3.3.4. Sobrecarga de operadores

### 3.3.5. Funciones en Python

En python, una función se define de la sieguiente manera: def funcion(argumento1, argumento2): "' Doc de la funcion "' print argumento1, argumento2 Python soporta lo que se ha dadon en llamar *empaquetado de argumentos* tanto en la definición como en la invocación de una función.

El empaquetado más simple es el del tipo lista, que consiste en la hablidad de una función de recibir argumentos variables en formato lista. Para usar esta caractrística es necesrio utilizar una signatura especial (agregando un asterisco).

def promedio(*lista): "' Calcula promedio "' return sum(lista) / float(len(lista)) utilizando varios builtins

Ejemplo de utilizacion ¿promedio(1, 2, 4, 5, 5) 3.3999999999999999

Esta signatura de argumentos se puede combinar con argumentos posicionales. def dividir$_{s}uma_{p}or(numero, *lista) : returnsum(lista)/numero$ La capiacidad de recibir listas puede ser utilizada a la inversa. Es decir, conociendo que una función recibe una cantidad de argumentos, "acomodar" en una lista los argumentos que se desean utilizar. Utilizanodo la funcion promedio anterior ¿lista = [1, 2, 4, 6, 7] ¿promedio( *lista ) 4.0

### 3.3.6. Clases en Python

En en lengauje Python, todos los tipos de datos son objetos. No existe el concepto de tipo de dato primitivo *(como en java)*. Las clases definidas por el usuario son a su vez objetos. A partir de la versión 2.2 del lenguaje, aparecen new-style-classes que permiten utilizar descriptores (base para implemetnación de propiedades) y metaclases. Una metaclase es una clase que define la estructura de otra clase y es una forma dinámica de definir la estructura de una clase. Esta característica es considerada compleja, pero es utilizada en varios componentes de Django (como la generación de formularios a partir de la definición de un modelo o la generación de interfases de CRUD en la administración integrada).

#### Instanciacion de una clase

En Python no existe el conceto de constructor como en los lenguajes C++, Objective-C o Java. La instanciación de una clase tiene dos fases: una fase de

creación estructural y otra de inicialización. Estos métodos utilizan los nombres reservados **__new__** e **__init__**.

*necesitamos saber que son *largs y **kwargs*

### 3.3.7. Expresiones regulares en Python

Las expresiones reuglares proveen un medio conciso y flexible de identificar cadenas en un texto, como caracteres en particular, palabras o patrones de caracteres. Las expresiones regulares (a menudo abreviadas **regex**, o **regexp**, o en plural **regexes**, **regexps**, o incluso **regexen**) son escritas en un lenguaje formal que puede ser interpretado por un procesador de expresiones regulares. Las expresiones regulares se utilizan en muchos editores de texto, utilitarios[3] y lenguajes de programación.

*POSIX Prel-like*

El módulo *re* en python provee un motor de análisis de expresiones regulares. La sintaxis de definición de expresiones con captura nombrada de gurpos.

## 3.4. Frameworks web

*Acá tenemos que justificar por que django*

### 3.4.1. CGI

Interfaz de entrada común (en inglés Common Gateway Interface, abreviado CGI) es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME[4]. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGIs.

Las aplicaciones CGI fueron una de las primeras maneras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este programa puede estar hecho en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

CGI ha hecho posible la implementación de funciones nuevas y variadas en las páginas web, de tal manera que esta interfaz rápidamente se volvió un estándar, siendo implementada en todo tipo de servidores web.

#### Ejecución de CGI

A continuación se describe la forma de actuación de un CGI de forma esquemática:

1. En primera instancia, el servidor recibe una petición (el cliente ha activado un URL que contiene el CGI), y comprueba si se trata de una invocación de un CGI.

[3] Como las utilidades de consola UNIX *sed*, *grep*, etc.

[4] Ver apéndice sobre MIMEC

2. Posteriormente, el servidor prepara el entorno para ejecutar la aplicación. Esta información procede mayoritariamente del cliente.

3. Seguidamente, el servidor ejecuta la aplicación, capturando su salida estándar.

4. A continuación, la aplicación realiza su función: como consecuencia de su actividad se va generando un objeto MIME que la aplicación escribe en su salida estándar.

5. Finalmente, cuando la aplicación finaliza, el servidor envía la información producida, junto con información propia, al cliente, que se encontraba en estado de espera. Es responsabilidad de la aplicación anunciar el tipo de objeto MIME que se genera (campo CONTENT_TYPE), pero el servidor calculará el tamaño del objeto producido.

**Intercambio de información: Variable de Entorno**

Variables de entorno que se intercambian de cliente a CGI:

**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

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

### 3.4.2. WSGI

The Web Server Gateway Interface defines a simple and universal interface between web servers and web applications or frameworks for the Python programming language. The latest version 3.0 of Python, released in December 2008, is already supported by $\mathrm{mod}_wsgi(amodulefortheApacheWebserver)$.

### ¿Por qué WSGI?

Historically Python web application frameworks have been a problem for new Python users because, generally speaking, the choice of web framework would limit the choice of usable web servers, and vice versa. Python applications were often designed for either CGI, FastCGI, $mod_python$ or even custom API interfaces of specific web-servers.

WSGI[5] (sometimes pronounced 'whiskey' or 'wiz-gee') was created as a low-level interface between web servers and web applications or frameworks to promote common ground for portable web application development. WSGI is based on the existing CGI standard.

### Descripción general de la especificación

The WSGI has two sides: the "server.ºr "gateway"side, and the .ªpplication.ºr "framework"side. The server side invokes[clarification needed] a callable object (usually a function or a method) that is provided by the application side. Additionally WSGI provides middleware; WSGI middleware implements both sides of the API, so that it can be inserted "between.ª WSGI server and a WSGI application – the middleware will act as an application from the server's point of view, and as a server from the application's point of view.

Un módulo middleware puede realizar opreaciones como:

- Routing a request to different application objects based on the target URL, after changing the environment variables accordingly.
- Allowing multiple applications or frameworks to run side-by-side in the same process
- Load balancing and remote processing, by forwarding requests and responses over a network
- Perform content postprocessing, such as applying XSLT stylesheets

### Aplicación de ejemplo

A WSGI compatible "Hello World.ªpplication in Python syntax: *continuar*

## 3.5. Django

Django es un framework web escrito en Python[6] 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. Fue liberado[7] bajo la licencia BSD en Julio del 2005 como Django Web Framework en honor a Django Reinhart. 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.

[5] PEP 333, Python Web Server Gateway Interface v1.0

[6] Ver apartado sobre el lengauje Python 3.3

[7] En el ámbito del software libre, la liberación es la fecha en la cual se pone a disposición de la comunidad del software en cuestión

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 (**django.contrib.admin**), que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

La distribución principal de Django también aglutina 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.

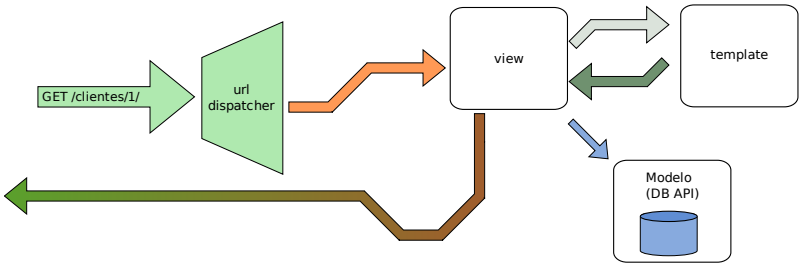


Figura 3.1: Estruccura báisca de Django

Django como framework de desarrollo consiste en un conjunto de utilidades de consola que permiten crear y manipular proyectos y aplicaciones.

## Estructura de un proyecto

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

Una aplicación es un paquete que contiene al menos los módulos **models.py** y **views.py**, y generalmente suele agregarse un módulo **urls.py**.

Un proyecto Django consite en 3 módulos[8] básicos:

### Módulo manage.py

Esta es la interfase con el framework. Este módulo permite crear aplicaciones, testear que los modelos de una aplicación estén bien definidos (validación), iniciar el servidor de desarrollo, crar volcados de la base de datos y restaurarlos restaurarlos (*fixtures*, utilizados en casos de pruebas y para precarga de datos conocidos).

[8] Un módulo en Python, es un archivo con extensión .py

#### Módulo settings.py

El módulo *settings* define la configuración transversal a las aplicaciones de usuario. En este módulo no se suelen definir más que constantes. Dentro de estas constantes encontramos la base de datos sobre la cual trabaja el ORM, el(los) directorio(s) de las plantillas, las clases middleware, ubicación de los medios estáticos[9]. En este módulo se defnien la lista de aplicaciones instaladas.

#### Módulo urls.py

Este módulo define las asociacines entre las URL y las funciones (vistas) que las atienden. Para generar código más modular, django permite delegar urls que cumplan con cierto patrón a un otro módulo. Típicamente este módulo se llama también urls y es parte de una aplicación (Ej: tratar todo lo lo que comience con **/clientes/** con el módulo **mi_proyecto.mi_aplicacion.urls**).

Una *expresión regular* es una forma de definir un patrón en una cadena. Mediante ésta técnica se realizan validaciones y búsquedas de elementos en cadenas. En python se define además una forma de otorgarle un alias a los elementos buscaods (en contraposición a la forma tradicional que utiliza un índice numérico). Esta particularidad de las expresiones regulares ha sido explotada para la asociación de las URLs a las funciones que las atienden. Cuando el cliente realiza accede a una URL dentro de un proyecto django, esta es checkeada contra cada patrón definido como url, en caso de éxito, se ejecuta la función asociada o vista. Si la expresión regualar tiene definido gurpos nombrados, cada subcadena pasa a ser argumento

pasan a ser argumentos de la vista, es decir, cada grupo nombrado, pasa a ser argumento de la función asociada.

### Elementos de una aplicación Django

Una aplicación consiste en 2 módulos fundamentales.

#### Módulo models.py

En este módulo se definien los modelos.

#### Módulo views.py

En este módulo se definen las viastas. Una vista es una función que recibe como primer argumento un objeto HttpRequest[10], el cual encapsula la información preveniente del request, como el método (GET, POST), los elementos de la query http.

[9] Un medio estático es todo contenido que no se genera dinámicamente, como imágenes, liberías de javascript, contenido para embeber como archivos multimedia u elementos

[10] Documentación oficial sobre HttpRequest en djangoproject.com

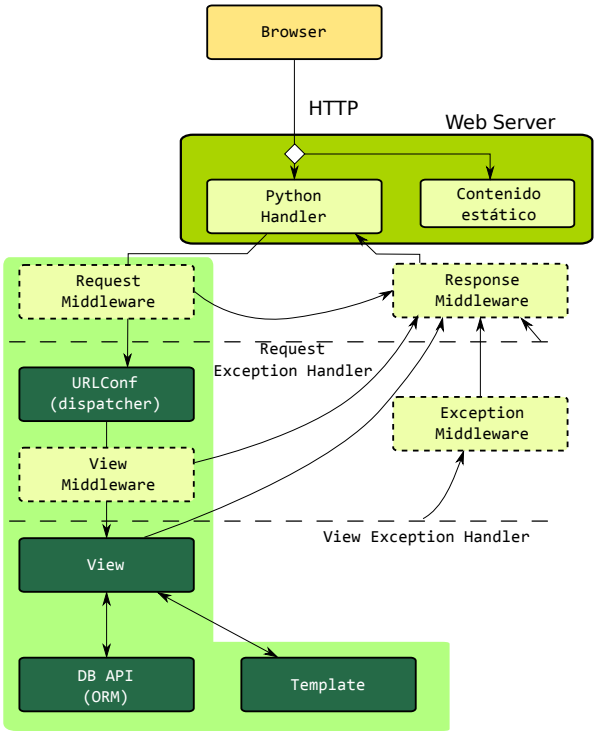


Figura 3.2: Esquema de flujo de una aplicación Django

# Capítulo 4

# Tecnologías Web del lado del Cliente

# Capítulo 5

# Integración y metodología de desarrollo con framework web

## 5.1. Mecanismos de extension de Django

### 5.1.1. Comandos Personalizados

Para poder realizar extensiones sobre la funcionalidad de Django sus desarrolladores han añadido la posibilidad de crear comandos personalizados. Un comando personalizado se define en un módulo, dentro de una aplicación y es ejecutado de manrera similar a cualquer otro comando provisto por el framework(como *syncdb*, *runserver*, *dumpdata*).

De igual manera que los comandos provistos por el framework, los comandos personalizados son ejecutados mediante el script de administración de proyecto *manage.py* o *django-admin.py*.

Un comando se define dentro de una aplicación. No existen comandos a nivel proyecto. Para que estos comandos sean detectados por el módulo de administración de poryectos deben respetar la siguiente estructura dentro del directorio de la aplicación

```
management/
    __init__.py
    commands/
        __init__.py
        mi_comando.py
```

Además la aplicación debe estar incluida en **INSTALLED_APPS** dentro de el módulo de configuración de proyecto **settings.py**.

Listing 5.1: Comando personalizado básico en Django

```
1
2 from django.core.management.base import NoArgsCommand
3
```

```
4 class Command(NoArgsCommand):
5   help = '''Descripci n del comando '''
6   def handle_noargs(self, **options):
7     print "Soy un comando personalizado"
```

Desde el punto de vista del lenguaje, la creación de un comando es la creación de un módulo management.commands.nombre_comando, en el cual se extiende a alguna clase base Command (que se encuentran en django.core.management.base).

Las clases base diponible son NoArgsCommand, para la imeplentación de comandos que no reciben argumentos; AppLabelCommand para la implementación de comandos que reciben como argumento el nombre de una aplicación instalada y

## 5.2. Integración

## 5.3. Convivencia

### 5.3.1. Modos de trabajo

Los distinotos modos de trabajo de la aplicacion son:

#### Estado online sin soporte offline

En este modo la aplicacion trabaja directamente sobre el servidor de aplicacion y depende completamente de una conexion. Al activar e implementar el soporte offline para nuestra aplicacion, en el servidor; el cliente debe ser conciente de este cambio y pasar a soportar los modos.

#### Estado online con soporte offline

#### Estado offline con soporte offline

# Capítulo 6

# Lineas futuras

## 6.1. Sitio de administración

## 6.2. Historial de navegación

## 6.3. Workers con soporte para Javascript 1.7

# Apéndice A

# Protopy

El desarrollo de un framework en javascript que funcione del lado del cliente presupone gran cantidad de codigo viajando de un lado al otro de la conexion. Previendo basicamente este postulado desarrollamos una libreria que brinde el soporte a los requerimentos mas basicos. Esta librería contituye la base para posteriores contrucciones mas complejas en el ciente y auna herramientas que simplifican el desarrollo client-side.

## A.1. Modulos

Undo de los principales inconvenientes a los que protopy da solucion es a la inclucion dinamica de funcionalidad bajo demanda, esto es logrado mediante los modulos. Basicamente un modulo en un archivo con codigo javascript que recide en el servidor y es obtenido y ejecutado en el cliente.

Listing A.1: Estructura de un modulo

```
1  //Archivo: tests/module.js
2  require('event');
3
4  var h1 = $('titulo');
5
6  function set_texto(txt) {
7      h1.update(txt);
8  }
9
10 function get_texto() {
11     return h1.innerHTML;
12 }
13
14 event.connect($('titulo'), 'click', function(event) {
15     alert('El texto es: ' + event.target.innerHTML);
16 });
17
18 publish({
19     set_texto: set_texto,
20     get_texto: get_texto
```

```
21 });
```

Probamos el modulo:

```
>>> require('tests.module')
>>> module.get_texto()
"Test de modulo"
>>> module.set_texto('Un titulo')
>>> require('tests.module', 'get_texto')
>>> get_texto()
"Un titulo"
>>> require('tests.module', '*')
>>> set_texto('Hola luuu!!!')
>>> get_texto()
"Hola luuu!!!"
```

## A.2. Modulos incluidos en el nucleo de protopy

### A.2.1. builtin

#### publish, publicando modulos

`publish(object)`

#### require, importando modulos

`require(module[, object...]) -> module or object`

#### type, constructor de tipos

`type(name, [base...] [, forType ], forPrototype) -> newType`

Como ya se menciono anteriormente javascript es un lenguaje orientado a prototipos, para acercarnos un poco a la programacion de objetos, utilizamos una funcion constructora de tipos o clases a la que denominamos "type".

Listing A.2: Definicion de tipos

```
1  var Animal = type('Animal', object, {
2      contador: 0,
3  }, {
4      __init__: function(especie) {
5          this.especie = especie;
6          this.orden = Animal.contador++;
7      }
8  });
9
10 var Terrestre = type('Terrestre', Animal, {
11     caminar: function() {
12         console.log(this.especie + ' caminando');
13     }
14 });
15
16 var Acuatico = type('Acuatico', Animal, {
17     nadar: function() {
```

```
18          console.log(this.especie + ' nadando');
19      }
20  });
21
22  var Anfibio = type('Anfibio', [Terrestre, Acuatico]);
23
24  var Piton = type('Piton', Terrestre, {
25      __init__: function(nombre) {
26          super(Terrestre, this).__init__(this.__name__);
27          this.nombre = nombre;
28      },
29      caminar: function() {
30          throw new Exception(this.especie + ' no camina');
31      },
32      reptar: function() {
33          console.log(this.nombre + ' la ' + this.especie.
                toLowerCase() + ' esta reptando');
34      }
35  });
36
37  var doris = new Piton('Doris');
38  var ballena = new Acuatico('Ballena');
39  var rana = new Anfibio('Rana');
```

Jugamos con los tipos:

```
>>> doris
window.Piton especie=Piton orden=0 nombre=Doris __name__=Piton
>>> rana
window.Anfibio especie=Rana orden=2 __name__=Anfibio
>>> isinstance(rana, Terrestre)
true
>>> isinstance(doris, Animal)
true
>>> issubclass(Anfibio, Acuatico)
true
>>> issubclass(Piton, Animal)
true
>>> doris.caminar()
Exception: Piton no camina args=[1] message=Piton no camina
>>> doris.reptar()
Doris la piton esta reptando
```

**$**

$(\text{id}) \rightarrow HTMLElement\$(id...) \rightarrow [HTMLElement...]$

```
>>> $('content')
<div id="content">
>>> $('content body')
>>> $('content', 'body')
[div#content, div#body]
>>> $('content', 'body', 'head')
[div#content, div#body, undefined]
```

**$$**

$\$\$(\text{cssRule}) \rightarrow [HTMLElement...]$

Ejemplo:

```
>>> $$('div')
[div#wrap, div#top, div#content, div.header, div.breadcrumbs, div.middle, div
    , div.right, div#clear, div#footer, div#toolbar]
>>> $$('div#toolbar')
[div#toolbar]
>>> $$('div#toolbar li')
[li, li.panel, li.panel, li, li]
>>> $$('div#toolbar li.panel')
[li.panel, li.panel]
>>> $$('a:not([href~=google])')
[a add_post, a add_tag, a removedb, a syncdb]
>>> $$('a:not([href=google])')
[a add_post, a add_tag, a#google www.google.com, a removedb, a syncdb]
>>> $$('div:empty')
[div#logger.panel, div#dbquery.panel, div#clear, div#top]
```

**extend**

`extend(destiny, source) -> alteredDestiny`

```
>>> a = {perro: 4}
>>> b = {gato: 4}
>>> c = extend(a, b)
>>> c
Object perro=4 gato=4
>>> a
Object perro=4 gato=4
>>> b
Object gato=4
```

**super**

super(destiny, source) → *alteredDestiny*

**isundefined**

isundefined(destiny, source) → *alteredDestiny*

**isinstance**

isundefined(destiny, source) → *alteredDestiny*

**issubclass**

issubclass(destiny, source) → *alteredDestiny*

**Arguments**

new Arguments(destiny, source) → *alteredDestiny*

**Template**

new Template(destiny, source) → *alteredDestiny*

**Dict**

new Dict(destiny, source) → *alteredDestiny*

**Set**

new Set(destiny, source) → *alteredDestiny*

**hash**

hash(destiny, source) → *alteredDestiny*

**id**

id(destiny, source) → *alteredDestiny*

**getattr**

getattr(destiny, source) → *alteredDestiny*

**setattr**

setattr(destiny, source) → *alteredDestiny*

**hasattr**

hasattr(destiny, source) → *alteredDestiny*

**assert**

assert(destiny, source) → *alteredDestiny*

**bool**

bool(destiny, source) → *alteredDestiny*

**callable**

callable(destiny, source) → *alteredDestiny*

**chr**

chr(number) → *character*

**ord**

ord(character) → *number*
Ejemplo:

```
>>> ord(chr(65))
65
>>> chr(ord("A"))
"A"
```

**bisect**

bisect(seq, element) → *position*

```
>>> a = [1,2,3,4,5]
>>> bisect(a,6)
5
>>> bisect(a,2)
2
>>> a[bisect(a,3)] = 3
>>> a
[1, 2, 3, 3, 5]
```

**equal**

equal(destiny, source) → *alteredDestiny*

**nequal**

nequal(destiny, source) → *alteredDestiny*

**number**

number(destiny, source) → *alteredDestiny*

**flatten**

flatten(destiny, source) → *alteredDestiny*

**include**

include(destiny, source) → *alteredDestiny*

**len**

len(destiny, source) → *alteredDestiny*

**array**

array(iterable) → [*element*...]

**print**

print(destiny, source) → *alteredDestiny*

**string**

string(destiny, source) → *alteredDestiny*

**values**

values(object) → [*value*...]

**keys**

keys(object) → [*key*...]

**items**

items(object) → [[*key*, *value*]...]
Ejemplo:

```
>>> items({'perro': 1, 'gato': 7})
[["perro", 1], ["gato", 7]]
```

**inspect**

inspect(destiny, source) → *alteredDestiny*

**unique**

unique(destiny, source) → *alteredDestiny*

**range**

range([begin = 1, ] end[, step = 1]) → [*number*...]

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(4, 10)
[4, 5, 6, 7, 8, 9]
>>> range(4, 10, 2)
[4, 6, 8]
```

**xrange**

xrange(destiny, source) → *alteredDestiny*

**zip**

zip(seq1 [, seq2 [...]]) → [[*seq*1[0], *seq*2[0]...], [...]]

```
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f','g','h'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, undefined], [6, undefined]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f'], [10,11,12,13,14,15,16])
[[1, "a", 10], [2, "b", 11], [3, "c", 12], [4, "d", 13], [5, "e", 14], [6, "f
    ", 15]]
```

### A.2.2. sys

**version**

version: 0.8,

**browser**

browser: IE: !!(window.attachEvent navigator.userAgent.indexOf('Opera') === -1), Opera: navigator.userAgent.indexOf('Opera') ¿-1, WebKit: navigator.userAgent.indexOf('AppleWebKit/') ¿-1, Gecko: navigator.userAgent.indexOf('Gecko') ¿-1 navigator.userAgent.indexOf('KHTML') === -1, MobileSafari: !!navigator.userAgent.match(/Apple.*Mobile.*Safari/), features: XPath: !!document.evaluate, SelectorsAPI: !!document.querySelector, ElementExtensions: !!window.HTMLElement, SpecificElementExtensions: document.createElement('div')['$_{proto_{]document.createElement('div')['_{proto_{]!==document.crea$

,

**get_gears**

get_gears: get$_{g}ears$,

**register_path**

register_path: function(module, path)

**module_url**

module_url: function(name, postfix)

**modules**

modules: ModuleManager.modules,

**paths**

paths: ModuleManager.paths

### A.2.3. exception

var exception = ModuleManager.create('exceptions', 'built-in', Exception: Exception, AssertionError: type('AssertionError', Exception), AttributeError: type('AttributeError', Exception), LoadError: type('LoadError', Exception), KeyError: type('KeyError', Exception), NotImplementedError: type('NotImplementedError', Exception), TypeError: type('TypeError', Exception), ValueError: type('ValueError', Exception), );

### A.2.4. event

**connect**

connect(object, event, context, method)

**disconnect**

disconnect(handle)

**subscribe**

subscribe(topic, context, method)

**unsubscribe**

unsubscribe(handle)

**publish**

publish(topic, args)

**connectpublisher**

connectpublisher(topic, obj, event)

**fixevent**

fixevent()

**stopevent**

stopevent()

**keys**

keys: BACKSPACE: 8, TAB: 9, CLEAR: 12, ENTER: 13, SHIFT: 16, CTRL: 17, ALT: 18, PAUSE: 19, CAPS$_LOCK$ : 20, $ESCAPE$ : 27, $SPACE$ : 32, $PAGE_UP$ : 33, $PAGE_DOWN$ : 34, $END$ : 35, $HOME$ : 36, $LEFT_ARROW$ : 37, $UP_ARROW$ : 38, $RIGHT_ARROW$ : 39, $DOWN_ARROW$ : 40, $INSERT$ : 45, $DELETE$ : 46, $HELP$ : 47, $LEFT_WINDOW$ : 91, $RIGHT_WINDOW$ : 92, $SELECT$ : 93, $NUMPAD_0$ : 96, $NUMPAD_1$ : 97, $NUMPAD_2$ : 98, $NUMPAD_3$ : 99, $NUMPAD_4$ : 100, $NUMPAD_5$ : 101, $NUMPAD_6$ : 102, $NUMPAD_7$ : 103, $NUMPAD_8$ : 104, $NUMPAD_9$ : 105, $NUMPAD_MULTIPLY$ : 106, $NUMPAD_PLUS$ : 107, $NUMPAD_ENTER$ : 108, $NUMPAD_MINUS$ : 109, $NUMPAD_PERIOD$ : 110, $NUMPAD_DIVIDE$ : 111, $F1$ : 112, $F2$ : 113, $F3$ : 114, $F4$ : 115, $F5$ : 116, $F6$ : 117, $F7$ : 118, $F8$ : 119, $F9$ : 120, $F10$ : 121, $F11$ : 122, $F12$ : 123, $F13$ : 124, $F14$ : 125, $F15$ : 126, $NUM_LOCK$ : 144, $SCROLL_LOCK$ : 145);

### A.2.5. timer

**setTimeout**

setTimeout: window.setTimeout,

**setInterval**

setInterval: window.setInterval,

**clearTimeout**

clearTimeout: window.clearTimeout,

**delay**

delay: function(f)

**defer**

defer: function(f)

### A.2.6. ajax

**Request**

new Request()

**Response**

new Response()

**toQueryParams**

toQueryParams(string, separator) → *object*

**toQueryString**

toQueryString(params) → *string*

### A.2.7. dom

**query**

query(cssRule) → [*HTMLElement*...]

## A.3. Extendiendo Javascript

### A.3.1. String

**gsub**

gsub(pattern, replacement) → *string*

**sub**

sub(pattern, replacement[, count = 1]) → *string*

**subs**

subs(pattern, replacement) → *string*

**format**

format(pattern, replacement) → *string*

**inspect**

inspect(use$_d$ouble$_q$uotes) → *string*

**truncate**

truncate(length, truncation) → *string*

**strip**

strip() → *string*

**striptags**

striptags() → *string*

**stripscripts**

stripscripts() → *string*

**extractscripts**

extractscripts() → *string*

**evalscripts**

evalscripts() → *string*

**escapeHTML**

escapeHTML() → *string*

**unescapeHTML**

unescapeHTML() → *string*

**succ**

succ() → *string*

**times**

times(count[, separator = ”]) → *string*

**camelize**

camelize() → *string*

**capitalize**

capitalize() → *string*

**underscore**

underscore() → *string*

**dasherize**

dasherize() → *string*

**startswith**

startswith(pattern) → *string*

**endswith**

endswith(pattern) → *string*

**blank**

blank() → *string*

### A.3.2. Number

**format**

format(f, radix) → *string*

### A.3.3. Date

**toISO8601**

toISO8601() → *string*

### A.3.4. Element

**visible**

visible()

**toggle**

toggle()

**hide**

hide()

**show**

show()

**remove**

remove()

**update**

update(content)

**insert**

insert(insertions)

**select**

select(selector)

```
>>> $('PostForm').select('input')
[input#id_title, input guardar]
>>> $('content').select('div')
[div.header, div.breadcrumbs, div.middle, div, div.right, div#clear]
>>> $('content').select('div.middle')
[div.middle]
```

### A.3.5. Forms

**disable**

disable()

**enable**

enable()

**serialize**

serialize() → *object*

### A.3.6. Forms.Element

**serialize**

`serialize()`

**get_value**

`get_value()`

**set_value**

`set_value(value)`

**clear**

`clear()`

**present**

`present()`

**activate**

```
activate()
```

**disable**

```
disable()
```

**enable**

```
enable()
```

# Apéndice B

# Doff

# Apéndice C

# MIME

MIME (*Multipurpose Internet Mail Extensions*), (Extensiones de Correo de Internet Multipropósito), son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos. En sentido general las extensiones de MIME van encaminadas a soportar:

- texto en conjuntos de caracteres distintos de US-ASCII
- adjuntos que no son de tipo texto
- cuerpos de mensajes con múltiples partes (multi-part)
- información de encabezados con conjuntos de caracteres distintos de ASCII.

Prácticamente todos los mensajes de correo electrónico escritos por personas en Internet y una proporción considerable de estos mensajes generados automáticamente son transmitidos en formato MIME a través de SMTP. Los mensajes de correo electrónico en Internet están tan cercanamente asociados con el SMTP y MIME que usualmente se les llama mensaje SMTP/MIME.[1]

En 1991 la IETF (Internet Engineering Task Force) comenzó a desarrollar esta norma y desde 1994 todas las extensiones MIME están especificadas de forma detallada en diversos documentos oficiales disponibles en Internet.

MIME está especificado en seis RFCs (acrónimo inglés de Request For Comments) : RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 y RFC 2077.

Los tipos de contenido definidos por el estándar MIME tienen gran importancia también fuera del contexto de los mensajes electrónicos. Ejemplo de esto son algunos protocolos de red tales como HTTP de la Web. HTTP requiere que los datos sean transmitidos en un contexto de mensajes tipo e-mail aunque los datos pueden no ser un e-mail propiamente dicho.

En la actualidad ningún programa de correo electrónico o navegador de Internet puede considerarse completo si no acepta MIME en sus diferentes facetas (texto y formatos de archivo).

## C.1. Introducción

El protocolo básico de transmisión de mensajes electrónicos de Internet soporta solo caracteres ASCII de 7 bit (véase también 8BITMIME). Esto limita los mensajes de correo electrónico, ya que incluyen solo caracteres suficientes para escribir en un número reducido de lenguajes, principalmente Inglés. Otros

lenguajes basados en el Alfabeto latino es adicionalmente un componente fundamental en protocolos de comunicación como HTTP, el que requiere que los datos sean transmitidos como un e-mail aunque los datos pueden no ser un e-mail propiamente dicho. Los clientes de correo y los servidores de correo convierten automáticamente desde y a formato MIME cuando envían o reciben (SMTP/MIME) e-mails.

# Encabezados MIME

### MIME-Version

La presencia de este encabezado indica que el mensaje utiliza el formato MIME. Su valor es típicamente igual a "1.0"por lo que este encabezado aparece como:

```
MIME-Version: 1.0
```

Debe señalarse que los implementadores han intentado cambiar el número de versión en el pasado y el cambio ha tenido resultados imprevistos. En una reunión de IETF realizada en Julio 2007 se decidió mantener el número de versión en *‘‘1.0’’* aunque se han realizado muchas actualizaciones a la versión de MIME.

### Content-Type

Este encabezado indica el tipo de medio que representa el contenido del mensaje, consiste en un tipo: type y un subtipo: subtype, por ejemplo:

```
Content-Type: text/plain
```

A través del uso del tipo multiparte (multipart), MIME da la posibilidad de crear mensajes que tengan partes y subpartes organizadas en una estructura arbórea en la que los nodos hoja pueden ser cualquier tipo de contenido no multiparte y los nodos que no son hojas pueden ser de cualquiera de las variedades de tipos multiparte. Este mecanismo soporta:

- mensajes de texto plano usando text/plain (este es el valor implícito para el encabezado Çontent-type:")
- 
- texto más archivos adjuntos (multipart/mixed con una parte text/plain y otras partes que no son de texto, por ejemplo: application/pdf para documentos pdf, application/vnd.oasis.opendocument.text para OpenDocument text). Un mensaje MIME que incluye un archivo adjunto generalmente indica el nombre original del archivo con un encabezado Çontent-disposition:.º por un atributo name de Content-Type, por lo que el tipo o formato del archivo se indica usando tanto el encabezado MIME content-type y la extensión del archivo (usualmente dependiente del SO).

  ```
  Content-Type: application/vnd.oasis.opendocument.text;
      name="Carta.odt"
  Content-Disposition: inline;
      filename="Carta.odt"
  ```

- reenviar con el mensaje original adjunto (multipart/mixed con una parte text/plain y el mensaje original como una parte message/rfc822)

- contenido alternativo, un mensaje que contiene el texto tanto en texto plano como en otro formato, usualmente HTML (multipart/alternative con el mismo contenido en forma de text/plain y text/html)
- muchas otras construcciones de mensaje

# Apéndice D

# Plataforma Mozilla

- Porque desarrollaron e implementaron Javascript 1.7
- Porque javascript 1.7 tomo semántica (y sintaxis???) de Python
- Porque es código abierto
- Porque es extensible mediante plugins
  - Tiene firebug
  - Gears y firebug = muy compardor para el desarrollador.
-

Protopy persigue acercar la semántica del Javascript 1.7 a la del lenguaje Python. Las funcionalidades principales son las siguientes:

- Ámbito de nombres
- Semántica de objetos, dentro de la cual se hace una adaptación de
  - Iteradores
  - Generadores
- Iteradores
- Generadores
- Tipos básicos de la librería estándar de python, entre los que se encuentran: *bool*,

**type** Type sirve para defninir clases.

# Bibliografía

[1] Versiones de Javascript, Jhon Reisig