

# Servicios y Aplicaciones en Redes de Ordenadores

## Grado en Sistemas de Telecomunicación

### Programa del curso 2014/2015

Jesús M. González Barahona, Gregorio Robles Martínez  
GSyC, Universidad Rey Juan Carlos

26 de febrero de 2015

## Índice

<b>1. Datos generales</b>	<b>8</b>
<b>2. Objetivos</b>	<b>9</b>
<b>3. Metodología</b>	<b>9</b>
<b>4. Evaluación</b>	<b>10</b>
<b>5. Programa de teoría</b>	<b>12</b>
5.1. 00 - Presentación . . . . .	12
5.1.1. Sesión del 13 de enero (2 horas) . . . . .	12
5.1.2. Sesión del 27 de enero (0.5 horas) . . . . .	12
5.2. 01 - Conceptos básicos de aplicaciones web . . . . .	12
5.2.1. Sesión del 15 de enero (2 horas) . . . . .	12
5.2.2. Sesión del 20 de enero (2 horas) . . . . .	12
5.2.3. Sesión del 29 de enero (2 horas) . . . . .	13
5.2.4. Sesión del 5 de febrero (2 horas) . . . . .	13
5.2.5. Sesión del 12 de febrero (0.5 horas) . . . . .	14
5.3. 02 - Servicios web que interoperan . . . . .	14
5.3.1. Sesión del 12 de febrero (1.5 horas) . . . . .	14
5.3.2. Sesión del 19 de febrero (2 horas) . . . . .	14
5.3.3. Sesión del 26 de febrero (2 horas) . . . . .	14
5.3.4. Sesión del 5 de marzo (2 horas) . . . . .	15

5.4.	03 - Modelo-vista-controlador . . . . .	15
5.4.1.	Sesión del ... (1 hora) . . . . .	15
5.5.	04 - Introducción a XML . . . . .	15
5.5.1.	Sesión del ... (1 hora) . . . . .	15
5.5.2.	Sesión del ... (2 horas) . . . . .	16
5.6.	05 - Hojas de estilo CSS . . . . .	16
5.6.1.	Sesión del ... (2 horas) . . . . .	16
5.6.2.	Sesión del ... (2 horas) . . . . .	17
5.7.	06 - AJAX . . . . .	17
5.7.1.	Sesión del ... (2 horas) . . . . .	17
<b>6.</b>	<b>Programa de prácticas</b>	<b>18</b>
6.1.	P1 - Introducción a Python . . . . .	18
6.1.1.	Sesión del 22 de enero (2 horas) . . . . .	18
6.1.2.	Sesión del 27 de enero (2 horas) . . . . .	18
6.1.3.	Sesión del 3 de febrero (2 horas) . . . . .	18
6.2.	P2 - Aplicaciones web simples . . . . .	19
6.2.1.	Sesión del 10 de febrero (2 horas) . . . . .	19
6.2.2.	Sesión del 17 de febrero (2 horas) . . . . .	19
6.2.3.	Sesión del 24 de febrero (2 horas) . . . . .	19
6.2.4.	Sesión del ... (0.5 horas) . . . . .	20
6.3.	P3 - Servidores simples de contenidos . . . . .	20
6.3.1.	Sesión del 5 de marzo (2 horas) . . . . .	20
6.4.	P4 - Introducción a Django . . . . .	20
6.4.1.	Sesión del 10 de marzo (2 horas) . . . . .	20
6.4.2.	Sesión del 17 de marzo (2 horas) . . . . .	21
6.4.3.	Sesión del 24 de marzo (2 horas) . . . . .	21
6.4.4.	Sesión del 7 de abril (2 horas) . . . . .	21
6.4.5.	Sesión del 14 de abril (2 horas) . . . . .	22
6.4.6.	Sesión del 21 de abril (2 horas) . . . . .	22
6.4.7.	Sesión del 28 de abril (2 horas) . . . . .	22
<b>7.</b>	<b>Entrega de prácticas incrementales</b>	<b>23</b>
<b>8.</b>	<b>Ejercicios 01: Conceptos básicos de aplicaciones web</b>	<b>24</b>
8.1.	Web 2.0 . . . . .	24
8.2.	Última búsqueda . . . . .	24
8.3.	Servicio horario . . . . .	25
8.4.	Última búsqueda: números aleatorios o consecutivos . . . . .	25
8.5.	Espía a tu navegador . . . . .	26
8.6.	Cookies en tu navegador . . . . .	26

8.7. Cookies en tu navegador avanzado . . . . .	26
8.8. Sumador simple con varios navegadores . . . . .	27
8.9. Sumador simple con varios navegadores intercalados . . . . .	27
8.10. Sumador simple con rearranques . . . . .	27
8.11. Traza de historiales de navegación por terceras partes . . . . .	27
8.12. Trackers en páginas web . . . . .	28
<b>9. Ejercicios 02: Servicios web que interoperan</b>	<b>28</b>
9.1. Arquitectura escalable . . . . .	28
9.2. Arquitectura distribuida . . . . .	29
9.3. Listado de lo que tengo en la nevera . . . . .	30
9.4. Sumador simple versión REST . . . . .	31
9.5. Calculadora simple versión REST . . . . .	31
9.6. Cache de contenidos . . . . .	33
9.7. Cache de contenidos anotado . . . . .	33
9.8. Gestor de contenidos multilingüe versión REST . . . . .	36
9.9. Sistema de transferencias bancarias . . . . .	36
9.10. Gestor de contenidos multilingüe preferencias del navegador . . . . .	37
9.11. Gestor de contenidos multilingüe con elección en la aplicación . . . . .	37
9.12. Sistema REST para calcular Pi . . . . .	38
<b>10. Ejercicios 04: Introducción a XML</b>	<b>39</b>
10.1. Chistes XML . . . . .	39
10.2. Modificación del contenido de una página HTML . . . . .	39
10.3. Titulares de BarraPunto . . . . .	39
10.4. Gestor de contenidos con titulares de BarraPunto . . . . .	40
10.5. Gestor de contenidos con titulares de BarraPunto versión SQL . . . . .	40
10.6. Gestor de contenidos con titulares de BarraPunto versión Django . . . . .	40
<b>11. Ejercicios 04: Hojas de estilo CSS</b>	<b>41</b>
11.1. Django cms.css simple . . . . .	41
11.2. Django cms.css elaborado . . . . .	41
<b>12. Ejercicios 05: AJAX</b>	<b>42</b>
12.1. SPA Sentences generator . . . . .	42
12.2. Ajax Sentences generator . . . . .	42
12.3. Gadget de Google . . . . .	43
12.4. Gadget de Google en Django cms . . . . .	43
12.5. EzWeb . . . . .	43
12.6. EyeOS . . . . .	43

<b>13.Ejercicios P1: Introducción a Python</b>	<b>44</b>
13.1. Uso interactivo del intérprete de Python . . . . .	44
13.2. Haz un programa en Python . . . . .	44
13.3. Tablas de multiplicar . . . . .	45
13.4. Ficheros y listas . . . . .	45
13.5. Ficheros, diccionarios y excepciones . . . . .	45
13.6. Calculadora . . . . .	46
<b>14.Ejercicios P2: Aplicaciones web simples</b>	<b>47</b>
14.1. Aplicación web hola mundo . . . . .	47
14.2. Variaciones de la aplicación web hola mundo . . . . .	47
14.3. Aplicación web generadora de URLs aleatorias . . . . .	48
14.4. Aplicación redirectora . . . . .	48
14.5. Sumador simple . . . . .	48
14.6. Clase servidor de aplicaciones . . . . .	49
14.7. Clase servidor de aplicaciones, generador de URLs aleatorias . . . . .	50
14.8. Clase servidor de aplicaciones, sumador . . . . .	50
14.9. Clase servidor de varias aplicaciones . . . . .	50
14.10Clase servidor, cuatro aplis . . . . .	50
14.11Herramientas de Web Developer . . . . .	51
<b>15.Ejercicios P3: Introducción a Django</b>	<b>51</b>
15.1. Instalación de Django . . . . .	51
15.2. Django Intro . . . . .	51
15.3. Django primera aplicación . . . . .	52
15.4. Django calc . . . . .	52
15.5. Django cms . . . . .	52
15.6. Django cms_put . . . . .	52
15.7. Django cms_users . . . . .	53
15.8. Django cms_users_put . . . . .	53
15.9. Django cms_templates . . . . .	54
15.10Django cms_post . . . . .	54
15.11Django cms_forms . . . . .	54
15.12Django feed_expander . . . . .	54
15.13Django feed_expander_db . . . . .	55
<b>16.Ejercicios P4: Servidores simples de contenidos</b>	<b>56</b>
16.1. Clase contentApp . . . . .	57
16.2. Instalación y prueba de Poster . . . . .	57
16.3. Clase contentPutApp . . . . .	57
16.4. Clase contentPostApp . . . . .	57

16.5. Clase contentPersistentApp . . . . .	58
16.6. Clase contentStorageApp . . . . .	58
16.7. Gestor de contenidos con usuarios . . . . .	58
16.8. Gestor de contenidos con usuarios, con control estricto de actualización . . . . .	59
<b>17.Ejercicios P5: Aplicaciones web con base de datos</b>	<b>59</b>
17.1. Introducción a SQLite3 con Python . . . . .	59
17.2. Gestor de contenidos con base de datos . . . . .	59
17.3. Gestor de contenidos con usuarios, con control estricto de actualización y base de datos . . . . .	60
<b>18.Prácticas de entrega voluntaria (curso 2013-2014)</b>	<b>61</b>
18.1. Práctica 1 (entrega voluntaria) . . . . .	61
18.2. Práctica 2 (entrega voluntaria) . . . . .	63
<b>19.Práctica final (2014, mayo)</b>	<b>64</b>
19.1. Arquitectura y funcionamiento general . . . . .	64
19.2. Funcionalidad mínima . . . . .	65
19.3. Funcionalidad optativa . . . . .	67
19.4. Entrega de la práctica . . . . .	68
19.5. Notas y comentarios . . . . .	70
<b>20.Ejercicios complementarios de varios temas</b>	<b>71</b>
20.1. Números primos . . . . .	71
20.2. Autenticación . . . . .	71
20.3. Recomendaciones . . . . .	72
20.4. Geolocalización . . . . .	73
<b>21.Prácticas de entrega voluntaria de cursos pasados</b>	<b>76</b>
21.1. Prácticas de entrega voluntaria (curso 2012-2013) . . . . .	76
21.1.1. Práctica 1 (entrega voluntaria) . . . . .	76
21.1.2. Práctica 2 (entrega voluntaria) . . . . .	77
21.2. Prácticas de entrega voluntaria (curso 2011-2012) . . . . .	78
21.2.1. Práctica 1 (entrega voluntaria) . . . . .	78
21.2.2. Práctica 2 (entrega voluntaria) . . . . .	79
21.3. Prácticas de entrega voluntaria (curso 2010-2011) . . . . .	80
21.3.1. Práctica 1 (entrega voluntaria) . . . . .	80
21.3.2. Práctica 2 (entrega voluntaria) . . . . .	81
21.3.3. Práctica 3 (entrega voluntaria) . . . . .	82
21.3.4. Práctica 4 (entrega voluntaria) . . . . .	83

<b>22.Prácticas finales de cursos pasados</b>	<b>84</b>
22.1. Práctica final (2013, mayo)	84
22.2. Arquitectura y funcionamiento general	84
22.3. Funcionalidad mínima	85
22.4. Funcionalidad optativa	88
22.5. Entrega de la práctica	88
22.6. Notas y comentarios	90
22.7. Práctica final (2012, diciembre)	90
22.7.1. Arquitectura y funcionamiento general	91
22.7.2. Funcionalidad mínima	92
22.7.3. Funcionalidad optativa	94
22.8. Práctica final (2011, diciembre)	95
22.8.1. Arquitectura y funcionamiento general	95
22.8.2. Funcionalidad mínima	96
22.8.3. Esquema de recursos servidos (funcionalidad mínima)	98
22.8.4. Funcionalidad optativa	98
22.8.5. Notas y comentarios	99
22.9. Práctica final (2012, mayo)	100
22.9.1. Arquitectura y funcionamiento general	100
22.9.2. Funcionalidad mínima	100
22.10 Práctica final (2010, enero)	101
22.10.1.Arquitectura y funcionamiento general	101
22.10.2.Funcionalidad mínima	102
22.10.3.Funcionalidad optativa	103
22.10.4.Entrega de la práctica	104
22.10.5.Notas y comentarios	105
22.11Práctica final (2010, junio)	105
22.12Práctica final (2010, diciembre)	106
22.12.1.Arquitectura y funcionamiento general	106
22.12.2.Funcionalidad mínima	108
22.12.3.Funcionalidad optativa	109
22.12.4.Entrega de la práctica	110
22.12.5.Notas y comentarios	111
22.12.6.Notas de ayuda	111
22.13Práctica final (2011, junio)	112
22.13.1.Arquitectura y funcionamiento general	113
22.13.2.Funcionalidad mínima	114
22.13.3.Funcionalidad optativa	115
22.13.4.Entrega de la práctica	116
22.13.5.Notas y comentarios	117

22.13.6. Notas de ayuda . . . . .	118
<b>23. Materiales de interés</b>	<b>119</b>
23.1. Material complementario general . . . . .	119
23.2. Introducción a Python . . . . .	119
23.3. Aplicaciones web mínimas . . . . .	119
23.4. SQL y SQLite . . . . .	120

## 1. Datos generales

<b>Título:</b>	Servicios y Aplicaciones en Redes de Ordena
<b>Titulación:</b>	Grado en Sistemas de Telecomunicación
<b>Cuatrimestre:</b>	Tercer curso, segundo cuatrimestre
<b>Créditos:</b>	6 (3 teóricos, 3 prácticos)
<b>Horas lectivas:</b>	4 horas semanales
<b>Horario:</b>	martes, 9:00–11:00 jueves, 9:00–11:00
<b>Profesores:</b>	Jesús M. González Barahona jgb @ gsync.es Despacho 003, Biblioteca, Campus de F Gregorio Robles Martínez grex @ gsync.es Despacho 110, Departamental III, Cam
<b>Sede telemática:</b>	<a href="http://campusvirtual.urjc.es/">http://campusvirtual.urjc.es/</a>
<b>Aulas:</b>	Laboratorio 209, Edif. Laboratorios III



## 2. Objetivos

En esta asignatura se pretende que el alumno obtenga conocimientos detallados sobre los servicios y aplicaciones comunes en las redes de ordenadores, y en particular en Internet. Se pretende especialmente que conozcan las tecnologías básicas que los hacen posibles.

## 3. Metodología

La asignatura tiene un enfoque eminentemente práctico. Por ello se realizará en la medida de lo posible en el laboratorio, y las prácticas realizadas (incluyendo especialmente la práctica final) tendrán gran importancia en la evaluación de la asignatura. Los conocimientos teóricos necesarios se intercalarán con los prácticos, en gran medida mediante metodologías apoyadas en la resolución de problemas. En las clases teóricas se utilizan, en algunos casos, transparencias que sirven de guión. En todos los casos se recomendarán referencias (usualmente documentos disponibles en Internet) para profundizar conocimientos, y complementarias de los detalles necesarios para la resolución de los problemas prácticos. En el desarrollo diario, las sesiones docentes incluirán habitualmente tanto aspectos teóricos como prácticos.

Se usa un sistema de apoyo telemático a la docencia (Moodle) para realizar actividades complementarias a las presenciales, y para organizar la documentación ofrecida a los alumnos.

Asimismo, se utiliza el servicio GitHub como repositorio, tanto de los materiales de la asignatura, como para entregar las prácticas por parte de los alumnos.

## 4. Evaluación

Parámetros generales:

- Teoría (obligatorio): 0 a 4.
- Práctica final (obligatorio): 0 a 2.
- Opciones y mejoras de la práctica final: 0 a 3
- Prácticas incrementales: 0 a 1
- Ejercicios en foro o en GitHub: 0 a 2
- Nota final: Suma de notas, moderada por la interpretación del profesor
- Mínimo para aprobar:
  - aprobado en teoría (2) y práctica final (1)
  - 5 puntos de nota final

Evaluación teoría: prueba escrita

Evaluación prácticas incrementales (evaluación continua):

- entre 0 y 2
- es muy recomendable hacerlas

Evaluación práctica final:

- posibilidad de examen presencial para práctica final
- tiene que funcionar en el laboratorio
- enunciado mínimo obligatorio supone 1, se llega a 2 sólo con calidad y cuidado en los detalles
- realización individual de la práctica

Opciones y mejoras práctica final:

- permiten subir la nota mucho

Evaluación ejercicios (evaluación continua):

- preguntas y ejercicios en foro

Evaluación extraordinaria:

- prueba escrita (si no se aprobó la ordinaria)
- nueva práctica final (si no se aprobó la ordinaria)
- entrega de ejercicios de evaluación continua (con penalización)

## 5. Programa de teoría

Programa de la asignatura (el detalle evoluciona según avanza el curso)

### 5.1. 00 - Presentación

#### 5.1.1. Sesión del 13 de enero (2 horas)

- **Presentación:** Presentación de la asignatura. Breve introducción y motivación de las aplicaciones web.
- **Material:** Transparencias, tema “Presentación”.

#### 5.1.2. Sesión del 27 de enero (0.5 horas)

- **Presentación:** Introducción a la entrega de prácticas en GitHub (sección 7).

### 5.2. 01 - Conceptos básicos de aplicaciones web

Sesión, mantenimiento de estado, persistencia.

#### 5.2.1. Sesión del 15 de enero (2 horas)

- **Ejercicio propuesto (discusión en clase):** “Última búsqueda” (ejercicio 8.2)  
Se introduce el problema, y se pide que trabajen sobre las interacciones HTTP involucradas. Para la próxima sesión se deja la discusión sobre el uso de cookies (u otros mecanismos) para conseguir la funcionalidad requerida.

#### 5.2.2. Sesión del 20 de enero (2 horas)

Páginas dinámicas (diferentes según cómo y cuándo se invocan). Cómo realizar sesiones en HTTP. Profundización en el concepto de sesión, y técnicas para conseguirla, incluyendo cookies y otros mecanismos.

- **Discusión de ejercicio (entrega en el foro):** “Última búsqueda” (ejercicio 8.2)  
Almacenamiento en el lado del servidor y en el lado del cliente. Relación entre peticiones HTTP. Cookies como herramienta para ambas situaciones. Entrega recomendada: antes del 27 de enero.

- **Discusión:** Usos de las cookies.  
 Uso de las cookies para identificación de visitantes (como en el ejercicio 8.2), para autenticación (interacción de autenticación y cookie de sesión posterior), para almacenamiento (como en el ejercicio 8.2, con última búsqueda en la cookie). Implicaciones de trasladar una cookie de identificación o de sesión de un ordenador a otro. Implicaciones de almacenar datos en el lado del navegador.
- **Ejercicio:** “Espía a tu navegador” (ejercicio 8.5)
- **Discusión de ejercicio:** “Servicio horario” (ejercicio 8.3).

### 5.2.3. Sesión del 29 de enero (2 horas)

Datos persistentes entre operaciones HTTP diferentes. Concepto de estado persistente frente a caídas del servidor.

- **Presentación:** Cookies
- **Material:** Transparencias, tema “Cookies”
- **Ejercicio propuesto:** “Cookies en tu navegador” (ejercicio 8.6)
- **Ejercicio propuesto:** “Cookies en tu navegador avanzado” (ejercicio 8.7)
- **Discusión de ejercicio:** “Última búsqueda: números aleatorios o consecutivos” (ejercicio 8.4)
- **Discusión:** Tres mecanismos básicos para mantenimiento de sesión (cookies, reescritura de URLs, campos ocultos en formularios)
- **Discusión de ejercicio (entrega en el foro):** “Traza de historiales de navegación por terceras partes” (ejercicio 8.11)  
 Entrega recomendada: antes del 12 de febrero.

### 5.2.4. Sesión del 5 de febrero (2 horas)

- **Ejercicio propuesto:** “Sumador simple con varios navegadores intercalados” (ejercicio 8.9)  
 Trabajo en grupo. Sólo se pretende llegar al pseudocódigo (lo más “estilo Python” que se pueda).

- **Discusión de ejercicio (entrega en el foro):** “Sumador simple con rearranques” (ejercicio 8.10).  
Entrega recomendada: antes del 12 de febrero.

#### 5.2.5. Sesión del 12 de febrero (0.5 horas)

- **Ejercicio (discusión en clase, entrega en el foro):** “Trackers en páginas web” (ejercicio 8.12)  
Entrega recomendada: antes del 19 de febrero.

### 5.3. 02 - Servicios web que interoperan

Invocaciones a aplicaciones web desde aplicaciones web. Servicios web como un conjunto de aplicaciones que interoperan.

#### 5.3.1. Sesión del 12 de febrero (1.5 horas)

- **Presentación:** Introducción breve a la arquitectura REST
- **Material:** Transparencias, tema “REST”
- **Ejercicio (discusión en clase, entrega en el foro):** “Listado de lo que tengo en la nevera” (ejercicio 9.3).  
Trabajo en grupos y discusión de los detalles del ejercicio.  
Entrega recomendada: antes del 19 de febrero.

#### 5.3.2. Sesión del 19 de febrero (2 horas)

- **Presentación:** Arquitectura REST
- **Material:** Transparencias, tema “REST”
- **Ejercicio (discusión en clase, entrega en el foro):** “Aplicación redirectora” (ejercicio 14.4).  
Entrega recomendada: antes del 26 de febrero.  
Repo GitHub: <https://github.com/CursosWeb/X-Serv-App-Redirectora>

#### 5.3.3. Sesión del 26 de febrero (2 horas)

- **Ejercicio (discusión en clase, entrega en el foro):** “Calculadora simple versión REST” (ejercicio 9.5).  
Entrega recomendada: antes del 5 de marzo.

- **Discusión de ejercicio:** “Cache de contenidos” (ejercicio 9.6).  
Trabajo en grupos y discusión de los detalles del ejercicio.
- **Ejercicio (entrega en el foro):** “Cache de contenidos anotado” (ejercicio 9.7)  
Entrega recomendada: antes del 5 de marzo.

#### 5.3.4. Sesión del 5 de marzo (2 horas)

- **Ejercicio (discusión en clase, entrega en el foro):** “Sistema de transferencias bancarias” (ejercicio 9.9).  
Entrega recomendada: antes del 12 de marzo.
- **Ejercicio (entrega en el foro):** “Sistema REST para calcular Pi” (ejercicio 9.12)  
Entrega recomendada: antes del 12 de marzo.

### 5.4. 03 - Modelo-vista-controlador

Explicación del patrón de diseño “modelo-vista-controlador”.

#### 5.4.1. Sesión del ... (1 hora)

- **Presentación:** “Modelo-vista-controlador”.
- **Material:** Transparencias “Modelo-vista-controlador”.
- **Presentación:** “Componentes de aplicaciones Django y MVC”. Repaso de los componentes principales de una aplicación Django y su relación con el patrón modelo-vista-controlador.

### 5.5. 04 - Introducción a XML

Uso de XML en aplicaciones web.

#### 5.5.1. Sesión del ... (1 hora)

- **Presentación:** “Introducción a XML”.  
Introducción a XML, sintaxis básica, formas de especificar gramáticas, reconocedores SAX y DOM.

- **Material:** Transparencias “Introducción a XML”.
- **Ejercicio (complementario):** “Chistes XML” (ejercicio 10.1).

#### 5.5.2. Sesión del ... (2 horas)

- **Presentación:** “Introducción a XML”.  
Usos de XML, canales semánticos usando RSS y similares. Complementos de XML, JSON.
- **Material:** Transparencias “Introducción a XML”.
- **Ejercicio (discusión en clase):** “Titulares de BarraPunto” (ejercicio 10.3).
- **Ejercicio (complementario):** “Gestor de contenidos con titulares de BarraPunto” (ejercicio 10.4).
- **Ejercicio (entrega en el foro):** “Gestor de contenidos con titulares de BarraPunto versión Django” (ejercicio 10.6)  
Entrega recomendada: antes del 3 de abril.

### 5.6. 05 - Hojas de estilo CSS

Hojas de estilo CSS, separación entre contenido y presentación.

#### 5.6.1. Sesión del ... (2 horas)

Hojas de estilo CSS, y su uso para manejar la apariencia de las páginas HTML.

- **Presentación:** “Hojas de estilo CSS”. Introducción a CSS. Principales elementos.
- **Material:** Transparencias, tema “CSS”.
- **Demo:** Inspección de datos de aspecto y hojas CSS con Firebug en Firefox.
- **Ejercicio (entrega en el foro):** “Django cms.css simple” (ejercicio 11.1).  
Entrega recomendada: antes del 10 de abril.



### 5.6.2. Sesión del ... (2 horas)

Hojas de estilo CSS, y su uso para manejar la apariencia de las páginas HTML.

- **Ejercicio (entrega en el foro):** “Django cms\_css elaborado” (ejercicio 11.2).  
Entrega recomendada: antes del 23 de abril.
  - **Presentación:** Práctica final.
- 

## 5.7. 06 - AJAX

Introducción a AJAX, *mash-ups* y otros tipos de aplicaciones web con código en el lado del cliente.

---

### 5.7.1. Sesión del ... (2 horas)

- **Presentación:** Aplicaciones web con código en el lado del cliente. DHTML, SPA, AJAX
- **Material:** Transparencias de la asignatura, tema “AJAX”.
- **Ejercicio:** “SPA Sentences generator” (ejercicio 12.1)
- **Ejercicio (entrega en foro):** “AJAX Sentences generator” (ejercicio 12.2)

## 6. Programa de prácticas

Programa de las prácticas de la asignatura (tentativo).

### 6.1. P1 - Introducción a Python

Introducción al lenguaje de programación Python, que se utilizará para la realización de las prácticas de la asignatura.

#### 6.1.1. Sesión del 22 de enero (2 horas)

- **Presentación:** “Introducción a Python” (introducción, entorno de ejecución, características básicas del lenguaje, ejecución en el intérprete, strings, listas, estructuras condicionales (if else), bucles for).
- **Material:** Transparencias “Introducción a Python”
- **Ejercicios:** “Uso interactivo del intérprete de Python” (ejercicio 13.1) y “Haz un programa en Python” (ejercicio 13.2).
- **Ejercicio propuesto (entrega en el foro):** “Tablas de multiplicar” (ejercicio 13.3). Entrega recomendada: antes del 27 de enero.

#### 6.1.2. Sesión del 27 de enero (2 horas)

- **Presentación:** “Introducción a Python”.
- **Material:** Transparencias “Introducción a Python”
- **Material:** Ejercicio “Ficheros y listas” (ejercicio 13.4).
- **Ejercicio propuesto (entrega en GitHub):** “Ficheros, diccionarios y excepciones” 13.5. Entrega recomendada: antes del 3 de febrero.

#### 6.1.3. Sesión del 3 de febrero (2 horas)

- **Presentación:** “Introducción a Python”.
- **Material:** Transparencias “Introducción a Python”
- **Ejercicio propuesto (entrega en GitHub):** “Calculadora” (ejercicio 13.6). Entrega recomendada: antes del 10 de febrero.

## 6.2. P2 - Aplicaciones web simples

Construcción de aplicaciones web mínimas sobre la biblioteca Sockets de Python.

### 6.2.1. Sesión del 10 de febrero (2 horas)

- **Ejercicio:** “Aplicación web hola mundo” (ejercicio 14.1)  
Se muestra la solución del ejercicio, y se comenta en clase. Se pide a los alumnos que lo ejecuten, lo modifiquen y se fijen en las cabeceras HTTP enviadas por el cliente y que el servidor muestra en pantalla (pero no hay entrega específica).
- **Ejercicio:** “Variaciones de la aplicación web hola mundo” (ejercicio 14.2).
- **Ejercicio propuesto (entrega en GitHub):** “Aplicación web generadora de URLs aleatorias” (ejercicio 14.3)  
Entrega recomendada: antes del 17 de febrero.

### 6.2.2. Sesión del 17 de febrero (2 horas)

- **Explicación de ejercicio:** “Aplicación web generadora de URLs aleatorias” (ejercicio 14.3)
- **Ejercicio en clase:** “Sumador simple” (ejercicio 14.5)
- **Trabajo y explicación del ejercicio:** “Clase servidor de aplicaciones” (ejercicio 14.6)  
Explicación de la estructura general que tienen las aplicaciones web, y fundamentos de cómo esta estructura se puede encapsular en una clase.
- **Ejercicio propuesto (entrega en GitHub):** “Clase servidor de aplicaciones, generador de URLs aleatorias” (ejercicio 14.7).  
Entrega recomendada: antes del 24 de febrero.

### 6.2.3. Sesión del 24 de febrero (2 horas)

- **Ejercicio propuesto:** “Clase servidor de aplicaciones, sumador” (ejercicio 14.8).
- **Ejercicio:** “Herramientas de Web Developer” (ejercicio 14.11).

- **Ejercicio propuesto:** “Clase servidor varias aplicaciones” (ejercicio 14.9)  
Explicación de la estructura principal de una clase que gestiona varias aplicaciones (o varios recursos, cada uno manejado por una aplicación)
  - **Ejercicio propuesto (entrega en GitHub):** “Clase servidor, cuatro aplis” (ejercicio 14.10).  
Entrega recomendada: antes del 5 de marzo.
- 

#### 6.2.4. Sesión del ... (0.5 horas)

- Presentación de la primera práctica de entrega voluntaria (18.1). Entrega en GitHub.

### 6.3. P3 - Servidores simples de contenidos

Construcción de algunos servidores de contenidos que permitan comprender la estructura básica de una aplicación web, y de cómo implementarlos aprovechando algunas características de Python.

#### 6.3.1. Sesión del 5 de marzo (2 horas)

- **Ejercicio:** “Clase contentApp” (ejercicio 16.1)  
Explicación de la estructura principal de una aplicación que sirve contenidos previamente almacenados.
- **Ejercicio:** “Instalación y prueba de Poster” (ejercicio 16.2).
- **Ejercicio propuesto (entrega en GitHub):** “Clase contentPutApp” (ejercicio 16.3).
- **Ejercicio propuesto (entrega en GitHub):** “Clase contentPostApp” (ejercicio 16.4). Entrega recomendada: antes del 10 de marzo.

### 6.4. P4 - Introducción a Django

#### 6.4.1. Sesión del 10 de marzo (2 horas)

Presentación de Django como sistema de construcción de aplicaciones web.

- **Presentación:** Introducción a Django (primera parte)
- **Ejercicio:** “Instalación de Django” (ejercicio 15.1).

- **Ejercicio:** “Django Intro” (ejercicio 15.2).
- **Material:** Transparencias “Introducción a Django”
- **Ejercicio (entrega en GitHub):** “Django Primera Aplicación” (ejercicio 15.3).  
Entrega recomendada: antes del 17 de marzo.

#### 6.4.2. Sesión del 17 de marzo (2 horas)

Primeros ejercicios con base de datos.

- **Presentación:** Introducción a Django (segunda parte)
- **Material:** Transparencias “Introducción a Django”
- **Ejercicio:** “Django cms” (ejercicio 15.5).
- **Ejercicio (entrega en GitHub):** “Django cms\_put” (ejercicio 15.6).  
Entrega recomendada: antes del 24 de marzo.

#### 6.4.3. Sesión del 24 de marzo (2 horas)

Usuarios, administración y autenticación con Django.

- **Presentación:** Introducción a Django (tercera parte)
- **Material:** Transparencias “Introducción a Django”
- **Ejercicio (discusión en clase):** “Django cms\_put” (ejercicio 15.6).
- **Ejercicio (discusión en GitHub):** “Django cms\_users” (ejercicio 15.7).  
Entrega recomendada: antes del 7 de abril.

#### 6.4.4. Sesión del 7 de abril (2 horas)

- **Presentación:** Introducción a Django (cuarta parte)
- **Material:** Transparencias “Introducción a Django”
- **Ejercicio (entrega en GitHub):** “Django cms\_users\_put” (ejercicio 15.8).
- **Ejercicio (entrega en GitHub):** “Django cms\_templates” (ejercicio 15.9).  
Entrega recomendada: antes del 14 de abril.

#### 6.4.5. Sesión del 14 de abril (2 horas)

POST y formularios en Django.

- **Presentación:** Introducción a Django (cuarta parte)
- **Material:** Transparencias “Introducción a Django”
- **Ejercicio (discusión en clase):** “Django cms\_templates” (ejercicio 15.9).
- **Ejercicio (entrega en GitHub):** “Django cms\_post” (ejercicio 15.10).  
Entrega recomendada: antes del 21 de abril.

#### 6.4.6. Sesión del 21 de abril (2 horas)

POST y formularios en Django (segunda parte)

- **Material:** Documentación de Django: “Working with forms”  
<http://docs.djangoproject.com/en/1.4/topics/forms>
- **Ejercicio (entrega en GitHub):** “Django cms\_forms” (ejercicio 15.11).  
Entrega recomendada: antes del 28 de abril.

#### 6.4.7. Sesión del 28 de abril (2 horas)

Uso de módulos externos con Django

- **Presentación:** Posibilidades de feedparser.py
- **Material:** <http://code.google.com/p/feedparser/>
- **Presentación:** Posibilidades de BeautifulSoup.py
- **Material:** <http://www.crummy.com/software/BeautifulSoup/>
- **Ejercicio (entrega en GitHub):** “Django feed\_expander” (ejercicio 15.12).

## 7. Entrega de prácticas incrementales

Para la entrega de prácticas incrementales se utilizarán repositorios git públicos alojados en GitHub. Para cada práctica entregable los profesores abrirán un repositorio público en el proyecto CursosWeb <sup>1</sup>, con un nombre que comenzará por “X-Serv-”, seguirá con el nombre del tema en el que se inscribe la práctica (por ejemplo, “Python” para el tema de introducción a Python) y el identificador del ejercicio (por ejemplo, “Calculadora”). Este repositorio incluirá un fichero README.md, con el enunciado de la práctica, y cualquier otro material que los profesores estimen conveniente.

Cada alumno dispondrá de una cuenta en GitHub, que usará a efectos de entrega de prácticas. Esta cuenta deberá ser apuntada en una lista, en el sitio de la asignatura en el campus virtual, cuando los profesores se lo soliciten. Si el alumno desea que no sea fácil trazar su identidad a partir de esta cuenta, puede elegir abrir una cuenta no ligada a sus datos personales: a efectos de valoración, los profesores utilizará la lista anterior. Si el alumno lo desea, puede usar la misma cuenta en GitHub para otros fines, además de para la entrega de prácticas.

Para trabajar en una práctica, los alumnos comenzarán por realizar una copia (fork) de cada uno de estos repositorios. Esto se realiza en GitHub, visitando (tras haberse autenticado con su cuenta de usuario de GitHub para entrega de prácticas) el repositorio con la práctica, y pulsando sobre la opción de realizar un fork. Una vez esto se haya hecho, el alumno tendrá un fork del repositorio en su cuenta, con los mismos contenidos que el repositorio original de la práctica. Visitando este nuevo repositorio, el alumno podrá conocer la url para clonarlo, con lo que podrá realizar su clon (copia) local, usando la orden `git clone`.

A partir de este momento, el alumno creará los ficheros que necesite en su copia local, los irá marcando como cambios con `git commit` (usando previamente `git add`, si es preciso, para añadirlos a los ficheros considerados por git), y cuando lo estime conveniente, los subirá a su repositorio en GitHub usando `git push`.

Por lo tanto, el flujo normal de trabajo de un alumno con una nueva práctica será:

[En GitHub: visita el repositorio de la práctica en CursosWeb,  
y le hace un fork, creando su propia copia del repositorio]

```
git clone url_copia_propia
```

[Se crea el directorio copia\_propia, copia local del repositorio propio]

```
cd copia_propia
```

---

<sup>1</sup><https://github.com/CursosWeb>

```
git add ... [ficheros de la práctica]
git commit .
git push
```

Conviene visitar el repositorio propio en GitHub, para comprobar que efectivamente los cambios realizados en la copia local se han propagado adecuadamente a él, tras haber invocado `git push`.

## 8. Ejercicios 01: Conceptos básicos de aplicaciones web

### 8.1. Web 2.0

#### **Enunciado:**

Seguramente has oído hablar muchas veces de la “web 2.0”. ¿Qué es lo que significa esta expresión? Si puedes, cita referencias en la Red al respecto.

### 8.2. Última búsqueda

#### **Enunciado:**

¿Cómo mostrar la última búsqueda en un buscador?

Se quiere que un cierto buscador web muestre a sus usuarios la última búsqueda que hicieron en él. Para ello, se utilizarán cookies. Son relevantes tres interacciones HTTP: la primera, en la que el navegador pide la página HTML con el formulario de búsquedas, la segunda, en la que el navegador envía la cadena de búsqueda que el usuario ha escrito en el navegador, y la tercera, que se realizará en cualquier momento posterior, en la que el navegador vuelve a pedir la página con el formulario de búsquedas, que ahora se recibe anotada con la cadena de la última búsqueda. Se pide indicar dónde van las cookies, cómo son éstas, y cómo solucionan el problema.

#### **Solución:**

Se puede hacer utilizando identificador de sesión en las cookies. Pero también es posible hacerlo sin que el servidor (el buscador) tenga que almacenar todos los identificadores de sesión junto con la última búsqueda realizada, lo que tiene varias ventajas.

Para identificador de sesión, basta con un número aleatorio grande que se almacena en la cookie. La cookie la envía el buscador al navegador en la respuesta al HTTP GET que se realiza para obtener la página del buscador. Luego, esa cookie va en cada POST que hace el navegador (para realizar una nueva búsqueda). Si no se quiere que el buscador almacena la última pregunta para cada sesión, se puede enviar la propia búsqueda en la cookie.



### Discusiones relacionadas:

- Ventajas y desventajas de utilizar identificadores de sesión, o de almacenar las preguntas en cookies en el navegador.
- ¿Serviría el mismo esquema para un servicio de banca electrónica? (en lugar de “recordar” la última pregunta, se quiere recordar qué usuario se autenticó.
- Cómo implementarlo usando el identificador de usuario y la contraseña en la cookie. Implicaciones para la seguridad. El problema de la salida de la sesión.

## 8.3. Servicio horario

### Enunciado:

Queremos construir una aplicación web que cuando se consulta, devuelva la hora actual. Además, queremos que cuando se consulta por segunda vez, devuelva la hora actual y la hora en que se consultó por última vez. Explicar cómo se pueden usar cookies para conseguirlo.

### Enunciado avanzado:

Igual que el anterior, pero se quiere que se muestre no sólo la hora en que se consultó por última vez, sino las horas de todas las consultas previas (además de la hora actual).

## 8.4. Última búsqueda: números aleatorios o consecutivos

### Enunciado:

En el ejercicio “Última búsqueda” (ejercicio 8.2) una de las soluciones pasa por usar cookies con identificadores de sesión. En principio, se han propuesto dos posibilidades para esos identificadores:

- Números enteros aleatorios sobre un espacio de números grande (por ejemplo entre 0 y  $2^{128} - 1$ )
- Números enteros consecutivos, comenzando por ejemplo por 0.

Comenta cuál de las dos soluciones te parece mejor, y si crees que alguna de ellas no sirve para resolver el problema. En ambos casos, indica las razones que te llevan a esa conclusión

## 8.5. Espía a tu navegador

### Enunciado:

El navegador hace una gran cantidad de tareas interesantes para esta asignatura. Es muy útil poder ver cómo lo hace, y aprender de los detalles que veamos. De hecho, también, en ciertos casos, se puede modificar su comportamiento. Para todo esto, se pueden usar herramientas específicas. En nuestro caso, vamos a usar el módulo “Firebug” de Firefox (también disponible para otros navegadores).

El ejercicio consiste en:

- Instalar el módulo Firebug en tu navegador
- Utilizarlo para ver la interacción HTTP al descargar una página web real.
- Utilizarlo para ver el árbol DOM de una página HTML real.

Más adelante, lo utilizaremos para otras cosas, así que si quieres jugar un rato con lo que permite hacer Firebug, mucho mejor.

### Referencias

Sitio web de Firebug: <https://getfirebug.com/>

## 8.6. Cookies en tu navegador

### Enunciado:

Busca dónde tiene tu navegador accesible la lista de cookies que mantiene, y mírala. ¿Cuántas cookies tienes? ¿Qué sitio te ha puesto más cookies? ¿Cuál es la cookie más antigua que tienes? Explica también qué navegador usas (nombre y versión), desde cuándo más o menos, y cómo has podido ver las cookies en él.

## 8.7. Cookies en tu navegador avanzado

### Enunciado:

Con el módulo adecuado, pueden editarse las cookies del navegador, lo que permite manejarlas con gran flexibilidad. Utiliza uno de estos módulos (por ejemplo, Cookies Manager+ para Firefox, o Firebug) para manipular las cookies que tenga tu navegador. Utilízalo para “traspasar” una sesión de un navegador a otro. Por ejemplo, puedes buscar las cookies que te autentican con un servicio (el campus virtual, una red social en la que tengas cuenta, etc.), guardarlas en un fichero, transferirlas a otro ordenador con otro navegador, e instalarlas en él, para comprobar cómo puedes continuar con la sesión desde él.

### Referencias

Sitio web de Cookies Manager+: <https://addons.mozilla.org/firefox/addon/cookies-manager-plus/>

## 8.8. Sumador simple con varios navegadores

### **Enunciado:**

Igual que el ejercicio “Sumador simple” (14.5), pero ahora puede haber varios navegadores invocando la aplicación web. Se supone que los navegadores no se interfieren (esto es, uno completa una suma antes de que otro la empiece).

### **Comentario:**

No hacen falta modificaciones al código del ejercicio “Sumador simple” (14.5).

## 8.9. Sumador simple con varios navegadores intercalados

### **Enunciado:**

Igual que “Sumador simple con varios navegadores” (8.8), pero ahora un navegador puede comenzar una suma en cualquier momento, incluyendo momentos en los que otro navegador no la haya terminado.

### **Comentarios:**

En una primera versión, se implementa con una cookie simple que incluye el primer operando, de forma que el servidor de aplicaciones no tiene que almacenar los operandos ni las cookies.

En una segunda versión, se utiliza una cookie de sesión más clásica, con un entero aleatorio, y se almacena el estado en un diccionario indexado por ese entero.

## 8.10. Sumador simple con rearranques

### **Enunciado:**

Igual que el ejercicio “Sumador simple con varios navegadores intercalados” (8.9), pero ahora desde que el navegador inicia la suma hasta que la completa, puede haberse caído la aplicación web.

### **Comentario:**

La aplicación web tendrá que almacenar su estado en almacenamiento estable. Hay que detectar cuál es ese estado, y almacenarlo en un fichero, en una base de datos, etc.

## 8.11. Traza de historiales de navegación por terceras partes

Cuando un navegador realiza un GET sobre una página web HTML lanza a continuación, de forma automática, otras operaciones GET sobre los elementos cargables automáticamente que contenga esa página, como por ejemplo, las imágenes empotradas. Cada vez que se realiza uno de estos GET, se pueden recibir una o

más cookies de los servidores que las sirven (y que en general pueden ser diferentes del que sirve la página HTML).

De esta forma, sirviendo imágenes para diferentes páginas HTML en diferentes sitios, una tercera parte puede trazar historiales de navegación, ligándolos a identificadores únicos. ¿Cómo?

Además, si la tercera parte en cuestión tiene acceso a información de un sitio web que permita identificar identidades, esos historiales pueden también ser ligados a identidades. ¿Cómo?

#### **Comentarios:**

La liga con identificadores únicos se puede lograr de varias formas. Por ejemplo se puede incluir en cada página HTML a trazar una imagen con nombre único, todas servidas por la tercera parte. La primera vez que sirve una imagen a un navegador dado, le envía también una cookie con identificador único. Todas las peticiones de imagen que se reciban serán escritas en un historial, junto con el identificador único de la cookie.

Para poder ligar este historial a una identidad, basta con que, en un servidor que ha identificado una identidad, sirva una imagen de la tercera parte con un nombre que permita posteriormente ligarlo a la identidad.

## **8.12. Trackers en páginas web**

Instala el *plug-in* Ghostery para tu navegador. Este *plug-in* permite detectar “trackers”, objetos incluidos en una página web para trazar a quienes descargan esa página. Utilízalo para encontrar páginas web con muchos trackers. Una vez lo hayas hecho, indica las dos páginas (de sitios distintos) en las que hayas encontrado más trackers.

#### **Referencias**

Sitio web de Ghostery: <http://www.ghostery.com/>

**Entrega recomendada:** hasta 10 de octubre de 2011

## **9. Ejercicios 02: Servicios web que interoperan**

### **9.1. Arquitectura escalable**

#### **Enunciado:**

Diseñar una arquitectura para una aplicación distribuida que cumpla las siguientes condiciones:

- Puede ser usada por millones de usuarios simultáneamente.

- Hay miles de equipos de desarrollo trabajando sobre ella. Entre los equipos hay poca comunicación pero no deben tener conflictos entre sí.
- Cada uno de los equipos podrían extender lo que habían hecho los otros sin que estos lo sepan y sin que la evolución de cada sistema rompiera la integración.

**Comentarios:**

Desde luego, hay otros sistemas, pero el web, entendido en sentido amplio, es uno que cumple bien estos requisitos.

## 9.2. Arquitectura distribuida

**Enunciado:**

Diseñar una arquitectura para una aplicación distribuida que cumpla las siguientes condiciones:

- Pueda gestionar elementos en mi casa desde remoto, en particular, mi comida. Por tanto, tendrá que gestionar los alimentos que se encuentran en la nevera, la despensa, el bol de frutas, etc.
- Pueda interactuar tanto con máquinas como con humanos
- Sea lo más sencilla posible
- Sea escalable

En particular, usando REST, define algunos recursos, y las operaciones que se podrían hacer sobre ellos. Explica también qué necesitaría para poder interoperar con los recursos correspondientes, por ejemplo, a la casa de tus amigos.

**Comentarios:**

Desde luego, hay muchas maneras de hacerlo y eso favorecerá el debate.

Una primera idea es modelar los elementos como objetos (la nevera, los alimentos, etc.) y hacerlo llegar de alguna manera al otro lado de la red, donde está mi portátil (esta es una solución que siguen muchos web services, o incluso CORBA). Hay que entender que esto hará que en el lado del portátil tengamos que conocer cómo funcionan los elementos (sus atributos y sus métodos). Es como tener que aprender el manual de instrucciones (los verbos) para cada cacharro que tengamos en la cocina.

Al ver esta solución, nos damos cuenta de que contamos en el otro lado con sustantivos. Éstos tienen una localización única, que especificamos mediante una URL. Asimismo, existe la URN, que permite especificar unívocamente un elemento

según su nombre, pero se ha de tener en cuenta de que puede haber un URN para muchos elementos (es como el ISBN, que hay uno para toda la edición, o sea para muchos libros). Dado la URL localizamos un URN de manera unívoca.

Mientras, en el otro lado (en el cliente) tendremos un número mínimo de acciones (los verbos). Vemos el primero: GET. Éste no obtiene el sustantivo, sino una representación del mismo. Los sustantivos son recursos. Y estos recursos pueden venir expresados de varias maneras. Así, por ejemplo, si pedimos manzanas desde un portátil la representación podría ser una imagen muy detallada; para el móvil, la imagen será más pequeña; y si el que lo pide es una máquina, podría ser un XML. Vemos los demás métodos: PUT, POST y DELETE.

Vistos los métodos discutimos si cambian el estado (vemos que sólo GET no lo hace) y si el resultado de realizar varios consecutivos es igual a hacerlo una vez (lo que llamamos idempotencia, vemos que sólo POST no lo es).

Introducimos el concepto de elemento y colección de elementos (cuando pedimos una colección, nos da un listado de los elementos que contiene; este listado contiene enlaces a los mismos) y qué pasa cuando aplicamos un método a cada uno. Hacemos especial hincapié en la diferencia entre PUT y POST.

Introducimos el concepto de REST y sus reglas. Hay varias las hemos visto ya: URLs, enlaces, representaciones y métodos. Nos falta por ver que las comunicaciones son sin estado. Los recursos pueden tenerlo, pero no la comunicación. Discutimos qué significa esto con respecto a lo que hemos visto en la asignatura hasta ahora, en particular con respecto a las sesiones (y las cookies).

Finalmente discutimos porque la web no es así, si en realidad los diseñadores de HTTP habían diseñado el protocolo para que todo fuera REST. Comentamos que con los navegadores sólo podemos hacer GETs y POSTs (y contamos que podemos utilizar los demás métodos mediante plug-ins como Poster (ver ejercicio 16.2). Mostramos que, más allá de los navegadores, ya estamos en disposición de crear programas para interactuar con servidores REST, de manera que podemos comunicar máquinas entre sí siguiendo estas reglas. Discutimos las ventajas de este enfoque en esos casos.

### 9.3. Listado de lo que tengo en la nevera

#### **Enunciado:**

Vamos a diseñar una API REST para una aplicación que concreta el ejercicio “Arquitectura distribuida” (9.2) en un caso bien simple: una aplicación web que mantenga la lista de lo que tengo en la nevera.

Supongamos que esta lista está compuesta únicamente por los items que tengo en la nevera en un momento dado (zanahorias, yogures, etc.) y un número natural para cada item que tengo, que expresa su cantidad. Por ejemplo, en un momento dado, la lista podría ser:

- Zanahorias: 5
- Yogures: 4
- Leche: 2

Si en este momento no tengo un determinado item en la nevera, no lo tengo en la lista. En otras palabras, no hay items con valor igual a 0.

La aplicación web la gestiono desde el móvil, con una app que puede hacer GET, PUT, POST y DELETE (usando HTTP). Cada vez que abro la puerta de la nevera, y meto o saco cosas de ella, utilizo la app del móvil para actualizar la lista en la aplicación web. Al hacerlo, la app del móvil siempre indica a la aplicación web la nueva cantidad que queda en la nevera. Esto es, no indica “hay dos zanahorias más”, sino “ahora hay 7 zanahorias”.

Se pide diseñar la API REST de esta aplicación web, identificando cuáles son los recursos relevantes, sus nombres, y describiendo todas las interacciones HTTP que soporten.

## 9.4. Sumador simple versión REST

### Enunciado:

Desarrollar una versión RESTful de “Sumador simple” (ejercicio 14.5). ¿Plantea problemas si se usa simultáneamente desde varios navegadores? ¿Plantea problemas si se cae el servidor entre dos invocaciones por parte del mismo navegador?

### Comentarios:

Hay varias formas de hacer el diseño, pero por ejemplo, cada sumando podría ser un recurso, y el resultado obtenerse en un tercero (o bien como respuesta al actualizar el segundo sumando). Cada suma podría también realizarse en un espacio de nombres de recurso distinto (con su propio primer sumando, segundo sumando y resultado).

## 9.5. Calculadora simple versión REST

### Enunciado:

Realizar una calculadora de las cuatro operaciones aritméticas básicas (suma, resta, multiplicación y división), siguiendo los principios REST, a la manera del sumador simple versión REST (ejercicio 9.4).

### Comentarios:

Este ejercicio, más que para proponer una solución concreta, está diseñado para debatir sobre las posibles soluciones que se le podrían dar. Por ejemplo, tenemos primero la versiones donde se supone un único usuario:

- Versión con un recurso por tipo de operación (“/suma”, “resta”, etc.). Se actualiza con PUT, que envía los operandos (ej: 4,5), se consulta con GET, que devuelve el resultado (ej:  $4+5=9$ ).
- Versión con un único recurso, “/operacion”. Se actualiza con PUT, que envía en el cuerpo la operación (ej:  $4+5$ ), se consulta con GET, que devuelve el resultado (ej:  $4+5=7$ ).
- Versión actualizando por separado los elementos de la operación, con un único recurso “/operacion”. PUT podrá llevar en el cuerpo “Primero: 4” o “Segundo: 5”, o “Op: +”. Cada uno de ellos actualiza el elemento correspondiente de la operación. GET de ese recurso, devuelve el resultado de la operación con los elementos que tiene en este momento.
- Versión actualizando por separado los elementos de la operación, con un único recurso “/operación”. PUT podrá llevar en el cuerpo un número si es la primera o segunda vez que se invoca, un símbolo de operación si es la tercera. GET dará el resultado si se han especificado todos los elementos de la operación, error si no. Es “menos REST”, en el sentido que guarda más estado en el lado del servidor. Pero cumple los requisitos generales de REST si consideramos que el cliente es responsable de mantener su estado y saber en qué fase de la operación está en cada momento.
- Versión donde cada elemento se envía con un PUT a un recurso (“/operacion/primeroperando”, “/operacion/segundooperando”, “/operacion/signo”), y el resultado se obtiene con “GET /operacion/resultado”. No es REST, porque el estado del recurso “/operacion/resultado” depende del estado de los otros recursos .

También podemos extender el diseño a versiones con varios usuarios:

- Podría tenerse un identificador para cada operación. “POST /operaciones” podría devolver el enlace a una nueva operación creada, como “/operaciones/2af434ad3”. Cada una de estas sumas se comportaría como las “sumas con un usuario” que se han comentado antes. “DELETE /operaciones/2af434ad3” destruiría una operación.

### Material:

- `simplecalc.py`: Programa con una posible solución a este ejercicio. Proporciona cuatro recursos “calculadora”, uno para cada operación matemática (suma, resta, multiplicación, división). Cada calculadora mantiene un estado (operación matemática) que se actualiza con PUT y se consulta con GET.



- Vídeo que muestra el funcionamiento de `simplecalc.py`  
<http://vimeo.com/31427714>
- Vídeo que describe el programa `simplecalc.py`  
<http://vimeo.com/31430208>
- `multicalc.py`: Programa con otra posible solución a este ejercicio. Proporciona un recurso para crear calculadoras (mediante POST). Al crear una calculadora se especifica de qué tipo (operación) es. Cada calculadora mantiene un estado (operación matemática) que se actualiza con PUT y se consulta con GET. Se apoya en las clases definidas en `simplecalc.py` para implementar las calculadoras.
- `webappmulti.py`: Clase que proporciona la estructura básica para los dos programas anteriores (clase raíz de servicio web, de *aplis*, etc.)

## 9.6. Cache de contenidos

### Enunciado:

Vamos a construir una aplicación web que no sólo recibe peticiones de un cliente, sino que también hace peticiones a otros servicios web. El ejercicio consiste en construir una aplicación que, dada una URL (sin “http://”) como nombre de recurso, devuelve el contenido de la página correspondiente a esa URL. Esto es, si se le pide `http://localhost:1234/gsync.es/` devuelve el contenido de la página `http://gsync.es/`. Además, lo guarda en un diccionario, de forma que si se le vuelve a pedir, lo devuelve directamente de ese diccionario.

Puede usarse como base `ContentApp`, y si se quiere, el módulo estándar de Python `urllib`.

### Comentarios:

Pueden discutirse muchos detalles de esta aplicación. Por ejemplo, cómo gestionar las cabeceras, y en particular las cookies. También, cómo saber si la página ha cambiado en el sitio original antes de decidir volver a bajarla (cabeceras relacionadas con “cacheable”, peticiones “HEAD” para ver fechas, etc.)

Para la implementación de la aplicación sólo se pide lo más básico: no hay tratamiento de cabeceras, y no se vuelve a bajar el original una vez está en la cache.

## 9.7. Cache de contenidos anotado

### Enunciado:

Construir una aplicación como “Cache de contenidos” (ejercicio 9.6), pero que anote cada página, en la primera línea, con un enlace a la página original, y que

incluya también un enlace para “recargar” la página (volverla a refrescar a partir del original), otro enlace para ver el HTTP (de ida y de vuelta, si fuera posible) que se intercambió para conseguir la página original, y otro enlace para ver el HTTP de la consulta del navegador y de la respuesta del servidor al pedir esta página (de nuevo si fuera posible).

#### **Comentarios:**

Téngase en cuenta que por lo tanto cada página que sirva la aplicación, además de los contenidos HTML correspondientes (obtenidos de la cache o directamente de Internet) tendrá cuatro enlaces en la primera línea:

- Enlace a la página original.
- Enlace a un recurso de la aplicación que permita recargar.
- Enlace a un recurso de la aplicación que permita ver el HTTP que se intercambió con el servidor que tenía la página.
- Enlace a un recurso de la aplicación que permita ver el HTTP que se intercambió cuando se cargó en cache esa página.

Estos enlaces conviene introducirlos en el cuerpo de la página HTML que se va a servir. Así, por ejemplo, si la página que se bajó de Internet es como sigue:

```
<html>
  <head> ... </head>
  <body>
    Text of the page
  </body>
</html>
```

Debería servirse anotada como sigue:

```
<html>
  <head> ... </head>
  <body>
    <a href="original_url">Original webpage</a>
    <a href="/recurso1">Reload</a>
    <a href="/recurso2">Server-side HTTP</a>
    <a href="/recurso3">Client-side HTTP</a><br>
    Text of the page
  </body>
</html>
```

Para poder hacer esto, es necesario localizar el elemento `< body >` en la página HTML que se está anotando. Hay que tener en cuenta que este elemento puede venir tal cual o con atributos, por ejemplo:

```
<body class="all" id="main">
```

Por eso no basta con identificar dónde está la cadena “`< body >`” en la página, sino que habrá que identificar primero dónde está “`< body`” y, a partir de ahí, el cierre del elemento, “`>`”. Será justo después de ese punto donde deberán colocarse las anotaciones. Para encontrar este punto puede usarse el método `find` de las variables de tipo *string*, o expresiones regulares.

Para que los enlaces que se enlazan desde estas anotaciones funcionen, la aplicación tendrá que atender a tres nuevos recursos para cada página:

- `/recurso1`: Recarga de la página en la cache.
- `/recurso2`: Devuelve el HTTP con el servidor (que tendrá que estar previamente almacenado en, por ejemplo, un diccionario).
- `/recurso3`: Devuelve el HTTP con el navegador (que tendrá que estar previamente almacenado en, por ejemplo, un diccionario).

Naturalmente, cada página necesitará estos tres recursos, por lo tanto lo mejor será diseñar tres espacios de nombres donde estén los recursos correspondientes para cada una de las páginas. Por ejemplo, todos los recursos de recarga podrían comenzar por “`/reload/`”, de forma que “`/reload/gsync.es`” sería el recurso para recargar la página “`http://gsync.es`”.

Para poder almacenar el HTTP con el servidor, es importante darse cuenta de que el que se envía al servidor lo produce la propia aplicación. Si se usa `urllib`, no es posible acceder directamente a lo que se está enviando, pero se puede inferir a partir de lo que se indica a `urllib`. Por lo tanto, cualquier petición HTTP “razonable” para los parámetros dados será suficiente, aunque no sea exactamente lo que envíe `urllib`.

El HTTP que se recibe del servidor habrá que obtenerlo usando `urllib`, en la medida de lo posible.

Para poder almacenar el HTTP con el cliente, es importante darse cuenta de que el que se envía al navegador lo produce la propia aplicación, por lo que basta con almacenarlo antes de enviarlo. El que se recibe del navegador habrá que obtenerlo de la petición recibida.

## 9.8. Gestor de contenidos multilingüe versión REST

### Enunciado:

Diseño y construcción de “Gestor de contenidos multilingüe versión REST”. Retomamos la aplicación ContentApp, pero ahora vamos a proporcionarle una interfaz multilingüe simple. Para empezar, trabajaremos con español (“es”) e inglés (“en”). Siguiendo la filosofía REST, cada recurso lo vamos a tener ahora disponible en dos URLs distintas, según en qué idioma esté. Los recursos en español empezarán por “/es/”, y los recursos en inglés por “/en/”. Además, si a un recurso no se le especifica de esta forma en qué idioma está, se servirá en el idioma por defecto (si está disponible), o en el otro idioma (si no está en el idioma por defecto, pero sí en el otro). Como siempre, los recursos que no estén disponibles en ningún idioma producirán un error “Resource not available”.

### Comentarios:

Para construir esta aplicación puedes usar dos diccionarios de contenidos (uno para cada idioma), o quizás mejor un diccionario de diccionarios, donde para cada recurso tengas como dato un diccionario con los idiomas en que está disponible, que tienen a su vez como dato la página HTML a servir.

## 9.9. Sistema de transferencias bancarias

### Enunciado:

Diseñar un sistema RESTful sobre HTTP fiable para realizar una transferencia bancaria vía HTTP.

- Debe poder confirmarse que la transferencia ha sido realizada.
- Debe poder prevenirse que la transferencia se haga más de una vez.
- Datos de la transferencia: cuenta origen, cuenta destino, cantidad.
- También debe poder consultarse el saldo de la cuenta (datos: cuenta)
- En un segundo escenario, puede suponerse todo lo anterior, pero considerando que hay una contraseña que protege el acceso a operaciones sobre una cuenta data (una contraseña por cuenta), tanto transferencias como consultas de saldo .

Indica el esquema de recursos (URLs) que ofrecerá la aplicación, y los verbos (comandos) HTTP que aceptará para cada uno, y con qué semántica.

## 9.10. Gestor de contenidos multilingüe preferencias del navegador

### Enunciado:

Diseñar y construir la aplicación web “Gestor de contenidos multilingüe preferencias del navegador”. En la aplicación “Gestor de contenidos multilingüe versión REST” (ejercicio 9.8) se especificaban como parte del nombre e recurso el idioma en que se quiere recibir un recurso. Pero el navegador tiene habitualmente una forma de especificar en qué idioma quieres recibir las páginas cuando están disponibles en varios. Para ver cómo funciona esto, prueba a cambiar tus preferencias idiomáticas en Firefox, y consulta la página <http://debian.org>.

Implementa una aplicación web que sea como la anterior, pero que además, haga caso de las preferencias del navegador con que la invoca, al menos para el caso de los idiomas “es” y “en”.

### Material complementario:

- Descripción de “Accept-Language” en la especificación de HTTP (RFC 2616)  
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.4>

### Comentario:

¿Qué recibe el servidor para poder hacer la selección de idioma? Utiliza una de tus aplicaciones para ver lo que le llega al servidor. Verás que lo que utiliza el navegador para indicar las preferencias idiomáticas del usuario es la cabecera “Accept-Language”

## 9.11. Gestor de contenidos multilingüe con elección en la aplicación

### Enunciado:

Diseñar y construir la aplicación web “Gestor de contenidos multilingüe con elección en la aplicación”. Ahora vamos a construir un servidor de contenidos multilingüe que además de los dos mecanismos anteriores (interfaz REST y preferencias del navegador, ejercicios 9.8 y 9.10) permita que el usuario elija el idioma específicamente en la propia aplicación.

Para ello, el gestor de contenidos atenderá a peticiones GET sobre recursos de la forma “/language/es” (para indicar que se quieren recibir las páginas en español), “/language/en” (para indicar que se quieren recibir las páginas en inglés) o “language/browser” (para indicar que se quieren recibir las páginas en el idioma que indique las preferencias del navegador).

El mecanismo de especificación de idioma mediante nombre de recurso (“/en” o /es”) tendrá precedencia sobre el mecanismo de especificación en la aplicación, y éste sobre el de preferencias del navegador.

Cada página incluirá, además del contenido en el idioma especificado, una lista (con enlaces) de los idiomas en que está disponible esa página, y una lista (con enlaces) de los idiomas que se pueden elegir en la aplicación. Por ejemplo, si estamos consultando una página en español que está disponible también en inglés, veremos un enlace “This page in English” que apuntará a la URL REST de esa página en inglés. Además, habrá enlaces a “Ver páginas preferentemente en español” (que apuntará al recurso /language/es), “See pages preferently in English” (que apuntará al recurso /language/en) y “Ver páginas según preferencias del navegador” (que apuntará a /language/browser).

**Comentario:**

Para implementar la elección especificándolo en la propia aplicación se podrán usar cookies, aunque no haya sistema de cuentas en la aplicación, como es el caso.

## 9.12. Sistema REST para calcular Pi

**Enunciado:**

Diseñar un sistema RESTful sobre HTTP que permita calcular el número pi como una operación asíncrona.

- El usuario solicita el comienzo del cálculo indicando el número de decimales deseado
- El usuario debe poder consultar a partir de ese momento el estado del cálculo

Indica el esquema de recursos (URLs) que ofrecerá la aplicación, y los verbos (comandos) HTTP que aceptará para cada uno, y con qué semántica.

Háganse dos versiones: en la primera, se supone que hay un sólo usuario (navegador) del sistema. En la segunda, puede haber varios, pero no simultáneamente: si un usuario solicita el comienzo del cálculo mientras hay otro cálculo en curso, le devuelve un mensaje de error.

**Comentario:**

Quien esté interesado puede realizar una implementación de una aplicación web para este diseño. Puede usar, por ejemplo, el método Monte Carlo, aplicando incrementalmente números cada vez más altos de números aleatorios.

**Materiales:**

Explicación del cálculo de Pi mediante el método Monte Carlo, incluyendo ejemplo en Python:

<http://www.eveandersson.com/pi/monte-carlo-circle>

## 10. Ejercicios 04: Introducción a XML

### 10.1. Chistes XML

**Enunciado:**

Estudia y modifica el programa `xml-parser-jokes.py` (que funciona con el fichero `jokes.xml`), hasta que entiendas los rudimentos del manejo de reconocedores SAX con Python.

**Material:**

- `jokes.xml`. Fichero XML con descripciones de chistes.
- `xml-parser-jokes.py`. Programa que lee el fichero anterior, y usando un parser SAX lo reconoce y muestra en pantalla el contenido de los chistes.

### 10.2. Modificación del contenido de una página HTML

**Enunciado:**

Estudia y modifica el documento HTML `dom.html`, de forma que:

- Al pulsar con el ratón sobre un texto, se recargue la página (invocando para ello una función JavaScript). Este texto ha de estar disponible para poder pulsar sobre él una vez la página haya cambiado de contenido.
- Al pulsar con el ratón sobre un botón, se modificará alguna parte del contenido mostrando la hora y fecha del momento.

**Material:**

- `dom.html`. Documento HTML, que incluye algo de código JavaScript, y que hay que modificar.

### 10.3. Titulares de BarraPunto

**Enunciado:**

Descargar el fichero RSS de BarraPunto<sup>2</sup>, y construir un programa que produzca como salida sus titulares en una página HTML. Si se carga esa página en un navegador, picando sobre un titular, el navegador deberá cargar la página de BarraPunto con la noticia correspondiente. Como base puede usarse lo aprendido estudiando los programas `xml-parser-jokes.py` y `xml-parser-barrapunto.py`.

**Material:**

---

<sup>2</sup><http://barrapunto.com>

- `http://barrapunto.com/index.rss`: URL del fichero RSS de BarraPunto.
- `xml-parser-barrapunto.py`: Programa que muestre en pantalla los titulares y las URLs que se describen en el fichero `barrapunto.rss`.
- `barrapunto.rss`: Fichero con el contenido del canal RSS de BarraPunto en un momento dado.

## 10.4. Gestor de contenidos con titulares de BarraPunto

### Enunciado:

Partiendo de `contentApp` (“Gestor de contenidos”, ejercicio 16.1), realiza `contentAppBarraPunto`. Esta versión devolverá, para cada recurso para el cuál tenga un contenido asociado en el diccionario de contenidos, una página que incluirá el contenido en cuestión, y los titulares de BarraPunto (para cada uno, título y URL).

Para ello, podéis hacer por un lado una aplicación que sirva para bajar el canal RSS de la portada de BarraPunto, y lo almacene en un objeto persistente (usando, por ejemplo, `Shelve`). Por otro lado, `contentBarraPuntoApp` leerá, antes de devolver una página, ese objeto, y utilizará sus datos para componer esa página a devolver.

## 10.5. Gestor de contenidos con titulares de BarraPunto versión SQL

### Enunciado:

Realiza `contentDBAppBarraPuntoSQL`, con la misma funcionalidad que `contentAppBarraPunto` (ejercicio 10.4), pero usando una base de datos `SQLite` en lugar de un diccionario persistente gestionado con `Shelve`.

## 10.6. Gestor de contenidos con titulares de BarraPunto versión Django

### Enunciado:

Realiza una aplicación Django con la misma funcionalidad que “Django cms” (ejercicio 15.5), pero que devuelva para cada recurso para el cuál tenga un contenido asociado en su tabla de la base de datos una página que incluirá el contenido en cuestión, y los titulares de BarraPunto (para cada uno, título y URL).

Para reutilizar código, puedes partir de “Django cms” (ejercicio 15.5) o “Django cms\_put” (ejercicio 15.6).



En particular, puedes implementar la consulta a BarraPunto de una de las siguientes formas:

- Cada vez que se pida un recurso, se mostrará el contenido asociado a él, anotado con los titulares de BarraPunto, que se descargarán (vía canal RSS) en ese mismo momento.
- Habrá un recurso especial, “/update”, que se usará para actualizar una tabla con los contenidos de BarraPunto. Cuando se invoque este recurso, se bajarán los titulares (vía canal RSS) de BarraPunto, y se almacenarán en una tabla en la base de datos que mantiene Django. Cada vez que se pida cualquier otro recurso, se mostrará el contenido asociado a él, anotado con los titulares de BarraPunto, que se extraerán de esa tabla, sin volver a pedirlos a BarraPunto.

Basta con mostrar por ejemplo los últimos tres o cinco titulares de BarraPunto (cada uno como un enlace a la URL correspondiente).

## 11. Ejercicios 04: Hojas de estilo CSS

### 11.1. Django cms\_css simple

**Enunciado:**

Crea una hoja de estilo en la URL “/css/main.css” para manejar la apariencia de la página “/about” en “Django cms\_put” (ejercicio 15.6). La hoja tendrá el siguiente contenido:

```
body {  
    margin: 10px 20% 50px 70px;  
    font-family: sans-serif;  
    color: black;  
    background: white;  
}
```

La página “/about” tendrá el contenido que estimes conveniente. Ambos contenidos (el de “/about” y el de “/css/main.css”) se subirán al gestor de contenidos mediante un PUT, igual que cualquier otro contenido.

### 11.2. Django cms\_css elaborado

**Enunciado:**

Modifica tu solución para “Django cms\_put” (ejercicio 15.6) de forma que:

- Si el recurso está bajo “/css/”, se almacene tal cual al recibirlo (mediante PUT) y se sirva tal cual (cuando se recibe un GET).
- Si el recurso tiene cualquier otro nombre, se almacene de tal forma cuando se reciba (mediante PUT) que el contenido almacenado sea el cuerpo (lo que va en el elemento *< BODY >*) de las páginas que se sirvan (cuando se reciba el GET correspondiente). Para servir las páginas utiliza una plantilla (*template*) que incluya el uso de la hoja de estilo “/css/main.css” para manejar la apariencia de todas las páginas.

## 12. Ejercicios 05: AJAX

Ejercicios con AJAX y tecnologías relacionadas.

### 12.1. SPA Sentences generator

#### Enunciado:

Prueba el fichero `sentences_generator.html`, que incluye una aplicación SPA simple que genera frases de forma aleatoria, a partir de componentes de tres listas de fragmentos de frases. En particular, observa dónde se obtiene una referencia al nodo del árbol DOM donde se quiere colocar la frase, y cómo se manipula éste árbol para colocarla ahí, una vez está generada.

Una vez lo hayas entendido, modifícalo para que en lugar de usar tres fragmentos para cada frase, use cuatro, cogiendo cada uno, aleatoriamente, de una lista de fragmentos.

#### Material:

- `sentences_generator.html`: Aplicación SPA que muestra frases componiendo fragmentos.

### 12.2. Ajax Sentences generator

#### Enunciado:

Construye una aplicación con funcionalidad similar a “SPA Sentences generator” (ejercicio 12.1), pero realizada mediante una aplicación AJAX que pide los datos a un servidor implementado en Django.

El servidor atenderá GET sobre los recursos `/first`, `/second` y `/third`, dando para cada uno de ellos la parte correspondiente (primera, segunda o tercera) de una frase, devolviendo un fragmento de texto aleatorio de una lista con fragmentos que tenga para cada uno de ellos (esto es, habrá una lista para los “primeros” fragmentos, otra para los segundos, y otra para los terceros).

La aplicación AJAX solicitará los tres fragmentos que necesita, y los compondrá mostrando la frase resultante, de forma similar a como lo hace la aplicación “SPA Sentences generator”.

**Material:**

- `words_provider.tar.gz`: Proyecto Django que sirve como servidor que proporciona fragmentos de frases para la aplicación AJAX anterior. Incluye `apps/sentences_generator.html`, aplicación AJAX que muestra frases componiendo fragmentos que obtiene de un sitio web, utilizando llamadas HTTP síncronas, y `apps/async_sentences_generator.html` (similar, pero con llamadas asíncronas).

### 12.3. Gadget de Google

**Enunciado:**

Inclusión de un gadget de Google, adecuadamente configurado, en una página HTML estática.

**Referencias:**

<http://www.google.com/ig/directory?synd=open>

### 12.4. Gadget de Google en Django cms

**Enunciado:**

Crear una versión del gestor de contenidos Django (Django cms, ejercicio 15.5) con un gadget de Google en cada página (el mismo en todas ellas).

### 12.5. EzWeb

**Enunciado:**

Abrir una cuenta en el sitio de EzWeb, y crear allí un nuevo espacio de trabajo donde se conecten algunos gadgets.

**Referencia:**

<http://ezweb.tid.es>

### 12.6. EyeOS

**Enunciado:**

Abrir una cuenta en el sitio de EyeOS, y visitar el entorno que proporciona.

**Referencia:**

<http://www.eyeos.org/>

## 13. Ejercicios P1: Introducción a Python

Estos ejercicios pretenden ayudar a conocer el lenguaje de programación Python. Los ejercicios suponen que previamente el alumno se ha documentado sobre el lenguaje, usando las referencias ofrecidas en clase, u otras equivalentes que pueda preferir.

Aunque es fácil encontrar soluciones a los ejercicios propuestos, se recomienda al alumno que realice por si mismo todos ellos.

El primer ejercicio has de hacerlo directamente en el intérprete de Python (invocándolo sin un programa fuente como argumento). Para los demás, puedes usar un editor (Emacs, gedit, o el que quieras) o un IDE (Eclipse con el módulo PyDev, o el que quieras).

### 13.1. Uso interactivo del intérprete de Python

#### Enunciado:

Invoca el intérprete de Python desde la shell. Crea las siguientes variables:

- un entero
- una cadena de caracteres con tu nombre
- una lista con cinco nombres de persona
- un diccionario de cuatro entradas que utilice como llave el nombre de uno de tus amigos y como valor su número de móvil

Comprueba con la sentencia `print nombre_variable` que todo lo que has hecho es correcto.

Fíjate en particular que la lista mantiene el orden que has introducido, mientras el diccionario no lo hace. Prueba a mostrar los distintos elementos de la lista y del diccionario con `print`.

### 13.2. Haz un programa en Python

#### Enunciado:

Haz un programa en Python que haga cualquier cosa, y escriba algo en la salida estándar (en el terminal, cuando lo ejecutes normalmente).

### 13.3. Tablas de multiplicar

**Enunciado:**

Utilizando bucles for, y funciones range(), escribe un programa que muestre en su salida estándar (pantalla) las tablas de multiplicar del 1 al 10, de la siguiente forma:

```
Tabla del 1
-----
1 por 1 es 1
1 por 2 es 2
1 por 3 es 3
...
1 por 10 es 10
Tabla del 2
-----
2 por 1 es 2
2 por 2 es 4
...
Tabla del 10
-----
...
10 por 10 es 100
```

### 13.4. Ficheros y listas

**Enunciado:**

Crea un script en Python que abra el fichero `/etc/passwd`, tome todas sus líneas en una lista de Python e imprima, para cada identificador de usuario, la shell que utiliza.

Imprime también el número de usuarios que hay en esta máquina. Utiliza para ello un método asociado a la lista, no un contador de la iteración.

### 13.5. Ficheros, diccionarios y excepciones

**Enunciado:**

Modifica el script anterior, de manera que en vez de imprimir para cada identificador de usuario el tipo de shell que utiliza, lo introduzca en un diccionario. Una vez introducidos todos, imprime por pantalla los valores para el usuario 'root' y para el usuario 'imaginario'. El segundo produce un error, porque no existe. ¿Sabrías evitarlo mediante el uso de excepciones?

## 13.6. Calculadora

### Enunciado:

Crea un programa que sirva de calculadora y que incluya las funciones básicas (sumar, restar, multiplicar y dividir). El programa ha de poder ejecutarse desde la línea de comandos de la siguiente manera: `python calculadora.py función operando1 operando2`. No olvides capturar las excepciones.

## 14. Ejercicios P2: Aplicaciones web simples

Estos ejercicios presentan al alumno unas pocas aplicaciones web que, aunque de funcionalidad mínima, van introduciendo algunos conceptos fundamentales.

### 14.1. Aplicación web hola mundo

#### **Enunciado:**

Construir una aplicación web, en Python, que muestre en el navegador “Hola mundo” cuando sea invocada. La aplicación usará únicamente la biblioteca socket. Construir la aplicación de la forma más simple posible, mientras proporcione correctamente la funcionalidad indicada.

#### **Motivación:**

Este ejercicio sirve para construir el primer ejemplo de aplicación web. Con ella se muestra ya la estructura típica genérica de una aplicación web: inicialización y bucle de atención a peticiones (a su vez dividido en recepción y análisis de petición, proceso y lógica y de aplicación, y respuesta). Todo está muy simplificado: no se hace análisis de la petición, porque se considera que todo vale, no se realiza proceso de la petición, porque siempre se hace lo mismo, y la respuesta es en realidad mínima.

Aunque no se usará mucho en la asignatura la biblioteca socket (pues trabajaremos a niveles de abstracción superiores), esta práctica sirve para ayudar a entender los detalles que normalmente oculta un marco de desarrollo de aplicaciones web.

La práctica también sirve para introducir el esquema típico de prueba (carga de la página principal de la aplicación con un navegador, colocación en un puerto TCP de usuario, etc.).

#### **Material:**

Se ofrecen dos soluciones en <https://github.com/CursosWeb/X-Serv-14.1-WebServer>. La más simple es `servidor-http-simple.py`. La otra, `servidor-http-simple-2.py`, permite conexiones desde fuera de la máquina huésped, y es capaz de reusar el puerto de forma que se puede rearrancar en cuanto muere.

### 14.2. Variaciones de la aplicación web hola mundo

#### **Enunciado:**

Basándose en la aplicación “Hola mundo” construida para el ejercicio 14.1, crear tres aplicaciones diferentes, con la siguiente funcionalidad cada una:

- Aplicación web que devuelva siempre la misma página HTML, que tendrá que tener al menos una imagen (usando un elemento IMG).

- Aplicación web que devuelva un código de error 404 y muestre un mensaje en el navegador.
- Aplicación web que produzca una redirección a la página `http://gsyc.es/`

**Material:**

### 14.3. Aplicación web generadora de URLs aleatorias

**Enunciado:**

Construcción de una aplicación web que devuelva URLs aleatorias. Cada vez que os conectéis al servidor, debe aparecer en el navegador “Hola. Dame otra”, donde “Dame otra” es un enlace a una URL aleatoria bajo `localhost:1234` (esto es, por ejemplo, `http://localhost:1234/324324234`). Esa URL ha de ser distinta cada vez que un navegador se conecte a la aplicación.

**Motivación:**

Explorar una aplicación web como extensión muy simple de “Aplicación web hola mundo”.

### 14.4. Aplicación redirectora

**Enunciado:**

Construir un programa en Python que sirva cualquier invocación que se le realice con una redirección (códigos de resultado HTTP en el rango 3xx) a otro recurso (aleatorio) de si mismo.

**Comentarios:**

Este programa se realiza muy fácilmente a partir de la solución de “Aplicación web generadora de URLs aleatorias” (ejercicio 14.3).

Para poder observar con más facilidad en el navegador lo que está ocurriendo, se puede hacer que la aplicación devuelva, en el cuerpo de la respuesta HTTP, un texto HTML indicando que se va a realizar una redirección, y a qué url va a realizarse. Para que este mensaje sea visible durante un tiempo razonable, se puede hacer que la aplicación, al recibir una petición, se quede “parada” durante unos segundos antes de contestar con la redirección.

**Motivación:**

Entender cómo funciona la redirección, y cómo reacciona un navegador ante ella.

### 14.5. Sumador simple

**Enunciado:**



Construir una aplicación web que suma en dos fases. En la primera, invocamos una URL del tipo `http://sumador.edu/5`, aportando el primer sumando (el número que aparece como nombre de recurso). En la segunda, invocamos una URL similar, proporcionando el segundo sumando. La aplicación nos devuelve el resultado de la suma. En esta primera versión, suponemos que la aplicación es usada desde un solo navegador, y que las URLs siempre le llegan “bien formadas”.

**Nota:**

Muchos navegadores, cuando se invoca con ellos una URL, lanzan un GET para ella, y a continuación uno o varios GET para el recurso `favicon.ico` en el mismo sitio. Por ello, hace falta tener en cuenta este caso para que funcione la aplicación web con ellos.

## 14.6. Clase servidor de aplicaciones

**Enunciado:**

Reescribe el programa “Aplicación web hola mundo” usando clases, y reutilizándolas, haz otro que devuelva “Adiós mundo cruel” en lugar de “Hola mundo”. Para ello, define una clase `webApp` que sirva como clase raíz, que al especializar permitirá tener aplicaciones web que hagan distintas cosas (en nuestro caso, `holaApp` y `adiosApp`).

Esa clase `webApp` tendrá al menos:

- Un método `Analyze` (o `Parse`), que devolverá un objeto con lo que ha analizado de la petición recibida del navegador (en el caso más simple, el objeto tendrá un nombre de recurso)
- Un método `Compute` (o `Process`), que recibirá como argumento el objeto con lo analizado por el método anterior, y devolverá una lista con el código resultante (por ejemplo, “200 OK”) y la página HTML a devolver
- Código para inicializar una instancia que incluya el bucle general de atención a clientes, y la gestión de sockets necesaria para que funcione.

Una vez la clase `webApp` esté definida, en otro módulo define la clase `holaApp`, hija de la anterior, que especializará los métodos `Parse` y `Process` como haga falta para implementar el “Hola mundo”.

El código `__main__` de ese módulo instanciará un objeto de clase `holaApp`, con lo que tendremos una aplicación “Hola mundo” funcionando.

Luego, haz lo mismo para `adiosApp`.

Conviene que en el módulo donde se defina la clase `webApp` se incluya también código para, en caso de ser llamado como programa principal, se cree un objeto de ese tipo, y se ejecute una aplicación web simple.

**Motivación:**

Explorar el sistema de clases de Python, y a la vez construir la estructura básica de una aplicación web con un esquema muy similar al que proporciona el módulo Python `SocketServer`.

## 14.7. Clase servidor de aplicaciones, generador de URLs aleatorias

**Enunciado:**

Realiza el servidor especificado en el ejercicio “Aplicación web generadora de URLs aleatorias” (ejercicio 14.3) utilizando el esquema de clases definido en el ejercicio “Clase servidor de aplicaciones” (ejercicio 14.6).

## 14.8. Clase servidor de aplicaciones, sumador

**Enunciado:**

Realizar el servidor especificado en el ejercicio “Sumador simple” (ejercicio 14.5) utilizando el esquema de clases definido en el ejercicio “Clase servidor de aplicaciones” (ejercicio 14.6).

## 14.9. Clase servidor de varias aplicaciones

**Enunciado:**

Realizar una nueva clase, similar a la que se construyó en el ejercicio “Clase servidor de aplicaciones” (ejercicio 14.6), pero preparada para servir varias aplicaciones (*aplis*). Cada *apli* se activará cuando se invoquen recursos que comiencen por un cierto prefijo.

Cada una de estas *aplis* será a su vez una instancia de una clase con origen en una básica con los dos métodos “parse” y “process”, con la misma funcionalidad que tenían en “Clase servidor de aplicaciones”. Por lo tanto, para tener una cierta *apli*, se extenderá la jerarquía de clases para *aplis* con una nueva clase, que redefinirá “parse” y “process” según la semántica de la *apli*.

Para especificar qué *apli* se activará cuando llegue una invocación a un nombre de recurso, se creará un diccionario donde para cada prefijo se indicará la instancia de *apli* a invocar. Este diccionario se pasará como parámetro al instanciar la clase que sirve varias aplicaciones.

## 14.10. Clase servidor, cuatro aplis

**Enunciado:**

Utilizando la clase creada para “Clase servidor de varias aplicaciones” (ejercicio 14.9), crea una aplicación web con varias aplis:

- Si se invocan recursos que comiencen por “/hola”, se devuelve una página HTML en la que se vea el texto “Hola”.
- Si se invocan recursos que comiencen por “/adios”, se devuelve una página HTML en la que se vea el texto “Adiós”.
- Si se invocan recursos que comiencen por “/suma/”, se proporciona la funcionalidad de “Sumador simple” (ejercicio 14.5), esperando que los sumandos se incluyan justo a continuación de “/suma/”.
- Si se invocan recursos que comiencen por “/aleat/”, se proporciona la funcionalidad de “Aplicación web generadora de URLs aleatorias” (ejercicio 14.3).

## 14.11. Herramientas de Web Developer

### Enunciado:

Introducción a las herramientas de *Web Developer*, que ayudan en el desarrollo y depuración de aplicaciones web en Firefox.

## 15. Ejercicios P3: Introducción a Django

### 15.1. Instalación de Django

#### Enunciado:

Instala la versión de Django que utilizaremos en prácticas.

#### Comentarios:

Utilizaremos la versión Django 1.7.4

#### Material:

- Descarga de Django: <http://www.djangoproject.com/download/>
- Transparencias “Introducción a Django”

### 15.2. Django Intro

#### Enunciado:

Realización de un proyecto Django de prueba (myproject), siguiendo el ejemplo de las transparencias “Introducción a Django”. Creación de las tablas de su base de datos, con Django, y consulta de la base de datos creada con sqllitebrowser.

#### Material:

myproject.tar.gz: Ejemplo de solución del ejercicio “Django intro”.

### 15.3. Django primera aplicación

**Enunciado:**

Realización de una aplicación Django que haga cualquier cosa, aún sin usar datos en almacenamiento estable. Por ejemplo, puede simplemente responder a ciertos recursos con páginas HTML definidas en el propio programa (en el correspondiente fichero `views.py`).

### 15.4. Django calc

**Enunciado:**

Realizar una calculadora con Django. Esta calculadora responderá a URLs de la forma `"/num1+num2"`, `"/num1*num2"`, `"/num1-num2"`, `"/num1/num2"`, realizando las operaciones correspondientes, y devolviendo error `"Not Found"` para las demás.

**Material:**

`calc.tar.gz`: Ejemplo de solución del ejercicio "Django calc"

### 15.5. Django cms

**Enunciado:**

Realizar una sistema de gestión de contenidos muy simple con Django. Corresponderá con la funcionalidad de `"contentApp"` (ejercicio 16.1), almacenando los contenidos en una base de datos.

**Material:**

`cms.tar.gz`: Ejemplo de solución del ejercicio "Django cms"

### 15.6. Django cms\_put

**Enunciado:**

Realizar una sistema de gestión de contenidos muy simple con Django. Corresponderá con la funcionalidad de `"contentPutApp"` (ejercicio 16.3), almacenando los contenidos en una base de datos. En otras palabras, será como "Django cms" (ejercicio 15.5), añadiendo la funcionalidad de que el usuario pueda poner contenidos mediante PUT, tal y como se explicó en el ejercicio de `"contentPutApp"`.

**Comentario:**

Para realizar este ejercicio, consultar el manual de Django, donde explica cómo se comporta el objeto `HttpRequest`, que es siempre primer argumento en los métodos que estamos definiendo en `views.py`. En particular, nos interesarán sus atributos `"method"` (que sirve para saber si nos está llegando un GET o un PUT) y `"body"`, que nos da acceso a los datos (cuerpo) de la petición. A pesar de su

nombre, este último atributo tiene esos datos tanto si la petición es un POST como si es un PUT.

**Material:**

cms\_put.tar.gz: Ejemplo de solución del ejercicio “Django cms\_put”.

## 15.7. Django cms\_users

**Enunciado:**

Realizar un proyecto Django con la misma funcionalidad que “Django cms\_put”, pero incluyendo un módulo de administración (lo que proporciona el “Admin site” de Django) y recursos para login y logout de usuarios. Además, cada página de contenidos (o cada mensaje indicando que una página no está disponible) deberá quedar anotada con la cadena “Not logged in. Login” (siendo “Login” un enlace al recurso de login) si no se está autenticado como usuario, o con la cadena “Logged in as name. Logout” (siendo “name” el nombre de usuario, y “Logout” un enlace al recurso de logout) si se está autenticado como usuario.

**Comentarios:**

- Cada página tendrá, por tanto, la misma funcionalidad que cms\_put, pero además una línea en la parte superior que dependerá de si el usuario que la visita está registrado o no.
- Se puede ver cómo realizar la funcionalidad de login y de logout en las páginas de Django, en particular, en <https://docs.djangoproject.com/en/dev/topics/auth/default/#auth-web-requests>.
- No hace falta tener una página de registro. Si queremos registrar un usuario, lo haríamos a través del interfaz de “admin”.

## 15.8. Django cms\_users\_put

**Enunciado:**

Realizar un proyecto Django con la misma funcionalidad que “Django cms\_users” (ejercicio 15.7), tratando de que el proceso de login y logout sea lo más razonable posible e incluyendo la funcionalidad de que sólo los usuarios que estén autenticados pueden cambiar el contenido de cualquier página, mientras que los que no lo están sólo pueden ver las páginas (funcionalidad similar a la de “Gestor de contenidos con usuarios”).

## 15.9. Django cms\_templates

### Enunciado:

Realizar un proyecto Django con la misma funcionalidad que “Django cms\_users\_put” (ejercicio 15.8), pero atendiendo a una nueva familia de recursos: “/annotated/”. Cualquier recurso que comience con “/annotated/” se servirá usando una plantilla, y por lo demás, con la misma funcionalidad que teníamos en “Django cms\_users\_put” al recibir un GET para el nombre de recurso.

## 15.10. Django cms\_post

**Enunciado:** Realizar un proyecto Django con la misma funcionalidad que “Django cms\_templates” (ejercicio 15.9), pero atendiendo a una nueva familia de recursos: “/edit/”. Cuando se acceda con un GET a un recurso que comience por “/edit/”, la aplicación web devolverá un formulario que permita editarlo (si se detecta un usuario autenticado, y si el nombre de recurso existe como página en la base de datos de la aplicación). Ese formulario tendrá un único campo que se precargará con el contenido de esa página. Si se accede con POST a un recurso que comience por “/edit/”, se utilizará el valor que venga en él para actualizar la página correspondiente, si el usuario está autenticado y la página existe. Además, volverá a devolver el formulario igual que con el GET, para que el usuario pueda continuar editando si así lo desea.

## 15.11. Django cms\_forms

### Enunciado:

Realizar el ejercicio “Django cms\_post” (ejercicio 15.10) utilizando la clase Forms de Django. Se ha de intentar que un cambio en el modelo (p.ej. añadir un campo nuevo) sólo afecte al modelo y a la clase Form derivada del mismo, y pueda realizarse sin modificar ni las vistas ni las plantillas.

## 15.12. Django feed\_expander

Utilizando Django y, en la medida que te parezca conveniente, `feedparser.py` y `BeautifulSoup.py`, realiza un servicio que expanda el contenido del canal de un usuario de Twitter. El servicio atenderá peticiones a recursos de la forma `/feed/user` (siendo `user` el identificador de un usuario de Twitter), devolviendo una página HTML con:

- Los cinco últimos *tweets* del usuario.

- Para cada uno de ellos, la lista de URLs que incluye (considerando como tales, por ejemplo, las subcadenas de caracteres delimitadas por espacios y que comiencen por “http://”).
- Para cada una de estas URLs:
  - El texto del primer elemento *< p >* de la página correspondiente, si existe.
  - Las imágenes (identificadas como elementos *< img >* que contenga la página correspondiente, si existen.

Los canales de usuarios de Twitter están disponibles en formato RSS en URLs como `https://api.twitter.com/1/statuses/user_timeline.rss?screen_name=user`, para el usuario `user`.

Pueden usarse también las bibliotecas `urllib2` para la descarga de páginas mediante HTTP, y `urlparse` para manipular URLs (ambos son módulos estándar de Python).

#### Referencias:

- Documentación sobre `feedparser.py` (vía WayBack Machine):  
<http://web.archive.org/web/20110625112257/http://feedparser.org/docs/>
- Presentación sobre `feedparser.py`:  
<http://www.slideshare.net/LindseySmith1/feedparser>
- Documentación sobre `BeautifulSoup.py`:  
<http://www.crummy.com/software/BeautifulSoup/documentation.html>

### 15.13. Django `feed_expander_db`

Realiza un servicio que proporcione la misma funcionalidad que “Django `feed_expander`” (ejercicio 15.12), pero almacenando los datos en tablas en una base de datos. Más en detalle:

- El recurso `/feed/user` seguirá haciendo lo mismo, para el usuario “user” de Twitter. Pero además de mostrar la página web resultante, almacenará entablas en la base de datos:
  - Los cinco últimos *tweets* del usuario, y el usuario al que se refieren
  - La lista de URLs de cada *tweet*

- El texto del primer elemento  $< p >$  de la página referenciada por cada URL.
- Las imágenes de dicha página.

En todos estos casos, la información se añadirá a la que haya ya previamente en las tablas correspondientes.

- El recurso `/db/user` mostrará la misma página que se muestra para `/feed/user`, pero incluyendo toda la información disponible en la base de datos para ese usuario (esto es, no limitado a los cinco últimos *tweets*, si hubiera más den la base de datos). Para mostrar la página mencionada, no se accederá a ningún recurso externo: sólo a la información en la base de datos.

### Comentarios:

Se pueden realizar varios diseños de tablas en la base de datos para este ejercicio. Entre ellos, se sugieren los basados en el siguiente esquema:

- Tabla de *tweets*, con dos campos: usuarios y *tweets*, ambos cadenas de texto (además, Django mantendrá un campo id para cada *tweet*).
- Tabla de URLs, con dos campos: id de *tweet* y URL, el primero un id, el segundo cadena de texto (además, Django mantendrá un campo id para cada URL).
- Tabla de textos, con dos campos: id de URL y texto (contenido de  $< p >$ , cadena de texto).
- Tabla de imágenes, con dos campos: id de URL e imagen (cadena de texto con la URL de la imagen).

Desde luego, este esquema se puede simplificar y complicar, pero quizás sea un buen punto medio para empezar a trabajar.

## 16. Ejercicios P4: Servidores simples de contenidos

Construcción de algunos servidores de contenidos que permitan comprender la estructura básica de una aplicación web, y de cómo implementarlos aprovechando algunas características de Python.



## 16.1. Clase `contentApp`

### Enunciado:

Esta clase, basada en el esquema de clases definido en el ejercicio “Clase servidor de aplicaciones” (ejercicio 14.6), sirve el contenido almacenado en un diccionario Python. La clave del diccionario es el nombre de recurso a servir, y el valor es el cuerpo de la página HTML correspondiente a ese recurso.

## 16.2. Instalación y prueba de Poster

### Enunciado:

Instalación y prueba de Poster, add-on de Firefox

### Referencias:

Poster Firefox add-on:

<https://addons.mozilla.org/en-US/firefox/addon/2691/>

## 16.3. Clase `contentPutApp`

### Enunciado:

Construcción de la clase “`contentPutApp`”, similar a `contentApp` (ejercicio 16.1). En este caso, la clase permite la actualización del contenido mediante peticiones HTTP PUT. Para probarla, se puede usar el add-on de Firefox llamado “Poster”. La clase será minimalista, basta con que funcione con “Poster”.

Opcionalmente, puede trabajarse en conseguir que un servidor construido con la clase anterior funcione con Bluefish. Bluefish es un editor de contenidos, que puede cargar una página especificando su URL, y que una vez modificada, puede enviarla, usando PUT, de nuevo a la misma URL. Aunque esto es exactamente lo que espera la clase “`contentPutApp`”, hay algunas peculiaridades de funcionamiento de Bluefish que hacen que probablemente la clase haya de ser modificada para que funcione correctamente con esta herramienta.

## 16.4. Clase `contentPostApp`

### Enunciado:

Construcción de la clase “`contentPostApp`”, similar a `contentApp` (ejercicio 16.1). En este caso, la clase permite la actualización del contenido mediante peticiones HTTP POST. Cuando se reciba un GET pidiendo cualquier recurso, se buscará en el diccionario de contenidos, y si existe, se servirá. En cualquier caso (exista o no exista el contenido en cuestión) se servirá en la misma página un formulario que permitirá actualizar el contenido del diccionario (o crear una nueva entrada, si no existía) mediante un POST.

**Referencias:**

Forms in HTML (HTML 4.01 Specification by W3C):  
<http://www.w3.org/TR/html4/interact/forms.html>

## 16.5. Clase `contentPersistentApp`

**Enunciado:**

Construcción de la clase `contentPersistentApp`, similar a `contentPutApp` (ejercicio 16.3), pero incluyendo almacenamiento del diccionario con los contenidos en almacenamiento persistente, de forma que la aplicación mantenga estado al recuperarse después de una caída. Para mantener estado, puede usarse el módulo “Shelve” de Python, que permite almacenar y recuperar objetos en ficheros.

Opcionalmente, puede usarse en otra versión el módulo “dbm” de Python, que sirve también para gestionar diccionarios persistentes, pero con más limitaciones.

## 16.6. Clase `contentStorageApp`

**Enunciado:**

Construcción de la clase `contentStorageApp` similar a `contentPersistentApp`, pero que use un objeto de clase `permanentContentStore` para almacenar el estado que ha de sobrevivir a caídas de la aplicación. Esta clase mantendrá variables internas con el estado a salvaguardar persistentemente, y métodos para consultar y actualizar los valores de ese estado.

## 16.7. Gestor de contenidos con usuarios

**Enunciado:**

Construye la clase `contentAppUsers`, que amplía el gestor de contenidos que estamos construyendo (clase `contentStorageApp`, ejercicio 16.6) con el concepto de usuarios registrados.

Cada usuario registrado tendrá un nombre y una contraseña (que puedes almacenar por ejemplo en un diccionario), y sólo si se ha mostrado al sistema que se es usuario registrado se podrá cambiar contenido del sitio (mediante un PUT). Para mostrar que se es usuario del sistema, se hará un GET a un recurso de la forma “/login,usuario,contraseña”, donde “usuario” y “contraseña” son el nombre de un usuario y su contraseña. A partir de ese momento, el sistema reconocerá que los accesos desde el mismo navegador son de ese usuario.

## 16.8. Gestor de contenidos con usuarios, con control estricto de actualización

### Enunciado:

Construye la clase `contentAppUsersStrict`, que implemente la misma funcionalidad de `contentAppUsers` (ejercicio 16.7), pero que además controle que sólo actualiza un contenido quien lo creó. En otras palabras, cuando la aplicación recibe un PUT, se comprueba que el recurso no existe, y en ese caso, si lo está subiendo un usuario autenticado, se crea. Pero si el recurso existe, sólo lo actualiza si el usuario que está invocando el PUT es el mismo que creó el recurso. Para implementar esta funcionalidad puedes utilizar un diccionario que “recuerde” quien creó cada recurso, o añadir, a los datos del diccionario de contenidos (donde sólo había la página HTML para el recurso en cuestión) un nuevo elemento (por ejemplo, usando una lista): el usuario que creó el recurso.

## 17. Ejercicios P5: Aplicaciones web con base de datos

Construcción de aplicaciones web con almacenamiento estable en base de datos.

### 17.1. Introducción a SQLite3 con Python

#### Enunciado:

Vamos a empezar a usar bases de datos relacionales con nuestras aplicaciones web. En particular, vamos a usar el módulo Python `sqlite3`, que proporciona enlace con el gestor de bases de datos SQLite3, que utiliza una interfaz SQL. Estudiar `test-db.py`, para entender cómo se hacen operaciones básicas sobre una base de datos con Python. Modificar ese programa para que añada más registros, y comprobar con `sqlitebrowser` la base de datos creada.

#### Material:

`test-db.py`. Programa que crea una base de datos simple SQLite3, y luego la muestra en pantalla.

### 17.2. Gestor de contenidos con base de datos

#### Enunciado:

Escribe y prueba la clase `contentDBApp`, que será una versión de `contentApp` (ejercicio 16.1), pero utilizando una base de datos SQLite3 para almacenar sus objetos persistentes.

### **17.3. Gestor de contenidos con usuarios, con control estricto de actualización y base de datos**

**Enunciado:**

Escribe y prueba la clase `contentDBAppUsersStrict`, que será igual que “Gestor de contenidos con usuarios, con control estricto de actualización” (`contentAppUsersStrict`, ejercicio 16.8), pero usando base de datos como almacenamiento permanente.

## 18. Prácticas de entrega voluntaria (curso 2013-2014)

### 18.1. Práctica 1 (entrega voluntaria)

**Fecha recomendada de entrega:** Antes del 13 de marzo.

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs. La aplicación funcionará únicamente con datos en memoria: se supone que cada vez que la aplicación muera y vuelva a ser lanzada, habrá perdido todo su estado anterior. La aplicación tendrá que realizarse según un esquema de clases similar al explicado en clase.

El funcionamiento de la aplicación será el siguiente:

- Recurso “/”, invocado mediante GET. Devolverá una página HTML con un formulario. En ese formulario se podrá escribir una url, que se enviará al servidor mediante POST. Además, esa misma página incluirá un listado de todas las URLs reales y acortadas que maneja la aplicación en este momento.
- Recurso “/”, invocado mediante POST. Si el comando POST incluye una qs (query string) que corresponda con una url enviada desde el formulario, se devolverá una página HTML con la url original y la url acortada (ambas como enlaces pinchables), y se apuntará la correspondencia (ver más abajo).

Si el POST no trae una qs que se haya podido generar en el formulario, devolverá una página HTML con un mensaje de error.

Si la url especificada en el formulario comienza por “http://” o “https://”, se considerará que ésa es la url a acortar. Si no es así, se le añadirá “http://” por delante, y se considerará que esa es la url a acortar. Por ejemplo, si en el formulario se escribe “http://gsyc.es”, la url a acortar será “http://gsyc.es”. Si se escribe “gsyc.es”, la url a acortar será “http://gsyc.es”.

Para determinar la url acortada, utilizará un número entero secuencial, comenzando por 0, para cada nueva petición de acortamiento de una URL que se reciba. Si se recibe una petición para una URL ya acortada, se devolverá la URL acortada que se devolvió en su momento.

Así, por ejemplo, si se quiere acortar

`http://docencia.etsit.urjc.es`

y la aplicación está en el puerto 1234 de la máquina “localhost”, se invocará (mediante POST) la URL

`http://localhost:1234/`

y en el cuerpo de esa petición HTTP irá la qs

`url=http://docencia.etsit.urjc.es`

si el campo donde el usuario puede escribir en el formulario tiene el nombre “url”. Normalmente, esta invocación POST se realizará rellenando el formulario que ofrece la aplicación.

Como respuesta, la aplicación devolverá (en el cuerpo de la respuesta HTTP) la URL acortada, por ejemplo

`http://localhost:1234/3`

Si a continuación se trata de acortar la URL

`http://docencia.etsit.urjc.es/moodle/course/view.php?id=25`

mediante un procedimiento similar, se recibirá como respuesta la URL acortada

`http://localhost:1234/4`

Si se vuelve a intentar acortar la URL

`http://docencia.etsit.urjc.es`

como ya ha sido acortada previamente, se devolverá la misma URL corta:

`http://localhost:1234/3`

- Recursos correspondientes a URLs acortadas. Estos serán números con el prefijo “/”. Cuando la aplicación reciba un GET sobre uno de estos recursos, si el número corresponde a una URL acortada, devolverá un HTTP REDIRECT a la URL real. Si no la tiene, devolverá HTTP ERROR “Recurso no disponible”.

Por ejemplo, si se recibe

`http://localhost:1234/3`

la aplicación devolverá un HTTP REDIRECT a la URL

`http://docencia.etsit.urjc.es`

## Comentario

Se recomienda utilizar dos diccionarios para almacenar las URLs reales y los números de las URLs acortadas. En uno de ellos, la clave de búsqueda será la URL real, y se utilizará para saber si una URL real ya está acortada, y en su caso saber cuál es el número de la URL corta correspondiente.

En el otro diccionario la clave de búsqueda será el número de la URL acortada, y se utilizará para localizar las URLs reales dadas las cortas. De todas formas, son posibles (e incluso más eficientes) otras estructuras de datos.

Se recomienda realizar la aplicación en varios pasos:

- Comenzar por reconocer “GET /”, y devolver el formulario correspondiente.
- Reconocer “POST /”, y devolver la página HTML correspondiente (con la url real y la acortada).
- Reconocer “GET /num” (para cualquier número num), y realizar la redirección correspondiente.
- Manejar las condiciones de error y realizar el resto de la funcionalidad.

## 18.2. Práctica 2 (entrega voluntaria)

**Fecha recomendada de entrega:** Antes del 3 de abril.

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs utilizando Django. Su enunciado será igual que el de la práctica 1 de entrega voluntaria (ejercicio 18.1), salvo en los siguientes aspectos:

- Se implementará utilizando Django.
- Tendrá que almacenar la información relativa a las URLs que acorta en una base de datos, de forma que aunque la aplicación sea rearrancada, las URLs acortadas sigan funcionando adecuadamente.

## 19. Práctica final (2014, mayo)

La práctica final de la asignatura consiste en la creación de una aplicación web que aglutine información sobre el estado de las carreteras y relacionada. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener. Llamaremos a la aplicación DeLorean, como tributo a los casi 30 años de la primera película “Regreso al Futuro”.

La aplicación consiste en descargarse datos de tráfico (disponibles públicamente en formato XML) y ofrecer estos datos a los usuarios de la aplicación para que puedan gestionar la información de la manera con consideren más conveniente. De esta manera, un escenario típico es el de un usuario que indique una provincia (o incluso una carretera) en la que está interesado; en su página personal aparecerán todas las incidencias de tráfico que cumplan esos requisitos, en tiempo real.

### 19.1. Arquitectura y funcionamiento general

Arquitectura general:

- DeLorean se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin Site” para crear cuenta a los usuarios en el sistema, y para la gestión general de las bases de datos necesarias. Todas las bases de datos que mantenga DeLorean tendrá que ser accesible vía este “Admin Site”.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que DeLorean tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones.
  - Una caja para entrar (hacer login en el sitio), o para salir (si ya se ha entrado). En caso de que no se haya entrado en una cuenta, esta caja permitirá al visitante introducir su identificador de usuario y su contraseña. En caso de que ya se haya entrado, esta caja mostrará el identificador del usuario y permitirá salir de la cuenta (logout).



- Un pie de página con una nota de copyright.

Cada una de estas partes estará marcada con propiedades “id” en HTML, para poder ser referenciadas en hojas de estilo CSS.

- Se utilizarán hojas de estilo CSS para determinar la apariencia de DeLorean. Estas hojas definirán al menos el color y el tamaño de la letra, y el color de fondo de cada una de las partes (elementos) marcadas con id que se indican en el apartado anterior.

Funcionamiento general:

- Los usuarios serán dados de alta en DeLorean mediante el módulo “Admin Site” de Django. Una vez estén dados de alta, serán considerados “usuarios registrados”.
- Los usuarios registrados podrán crear su selección de estados de carretera de DeLorean. Para ello, dispondrán de una página personal. Llamaremos a esta página la “página del usuario”.
- La selección de incidencias en su página personal la realizará cada usuario a partir de información sobre incidencias ya disponibles en el sitio.
- Las incidencias se actualizarán sólo cuando un usuario indique que quiere que se actualicen.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la página personal de cada usuario, para todos los usuarios del sitio.

## 19.2. Funcionalidad mínima

Interfaz pública: recursos a servir como páginas HTML completas (pensadas para ser vistas en el navegador) para cualquier visitante (sea usuario registrado o no):

- /: Página principal de DeLorean. Mostrará un listado de las últimas diez incidencias y posteriormente otro listado con las páginas personales disponibles. Para cada página personal mostrará el título (como un enlace a la página personal), el nombre de su usuario y una pequeña descripción. Si a una página personal aún no se le hubiera puesto título, este título será “Página de usuario”, donde “usuario” es el identificador de usuario del usuario en cuestión.

- /usuario: Página personal de un usuario. Si la URL es “/usuario”, es que corresponde al usuario “usuario”. Mostrará las incidencias seleccionadas por ese usuario (aunque no puede haber más de 10 a la vez, como se indicará más adelante). Para cada incidencia se mostrará la “información pública de cada incidencia”, ver más adelante.
- /usuario/rss: Canal RSS para las incidencias seleccionadas por ese usuario.
- /ayuda: Página con información HTML explicando el funcionamiento de DeLorean.
- /todas: Página con todas las incidencias. En la parte superior de la página, existirá un formulario que permite filtrar las incidencias según varios campos, como, por ejemplo, provincia, tipo, longitud.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todas las incidencias (URL /todas) con el texto “Todas” y a la ayuda (URL /ayuda) con el texto “Ayuda”. Todas las página que no sean la principal tendrán otra opción de menú para la URL /, con el texto “Inicio”.

Interfaz privada: recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado).

- Todos los recursos de la interfaz pública.
- /incidencias: Página con la lista de incidencias disponibles en DeLorean:
  - Las incidencias se toman de interpretar la información pública ofrecida por la Dirección General de Tráfico (DGT), y que es la siguiente:
    - Información de incidencias en carreteras (canal de incidencias):  
`http://www.dgt.es/incidencias.xml`
  - Se mostrará el número de incidencias disponibles para el canal, y la fecha en que fue actualizado por última vez.
  - Existirá un botón para actualizar las incidencias a partir del canal de incidencias. Si se pulsa este botón, se tratarán de actualizar las incidencias accediendo al canal de incidencias de la DGT. Al terminar la operación se volverá a mostrar esta misma página, actualizada.
  - La lista de incidencias disponibles en el canal de incidencias, incluyendo para cada una la “información pública”, ver más adelante.
  - Junto a cada incidencia de la lista, se incluirá un botón que permitirá elegir la incidencia para la página personal del usuario autenticado. Tras añadir una incidencia a la página del usuario, se volverá a ver en el navegador la página /incidencias.

- En la página /usuario que corresponde al usuario autenticado se mostrará, además de lo ya mencionado para la interfaz pública, un formulario en el que se podrá especificar la siguiente información:
  - Los parámetros CSS para el usuario autenticado (al menos los indicados anteriormente para ser manejados por un documento CSS). Si el usuario los cambia, a partir de ese momento deberá verse el sitio con los nuevos valores, y para ello deberá servirse un nuevo documento CSS.
  - El título de su página personal.

Si es preciso, se añadirán más recursos (pero sólo si es realmente preciso) para poder satisfacer los requisitos especificados.

Dados los recursos mencionados anteriormente, no se permitirán los nombres de usuario “incidencias”, “ayuda” ni “todas” (pero no hay que hacer ninguna comprobación para esto: se asume que no se darán de alta esos usuarios en el Admin Site).

Como información pública de cada incidencia se mostrará:

- El tipo de incidencia
- La provincia de la incidencia y la carretera
- La fecha en que fue publicada la incidencia en el sitio original (junto al texto “publicada en”).
- La fecha en que fue seleccionada para la página personal del usuario (junto al texto “elegida en”).
- La información detallada de la incidencia (toda la demás información de la incidencia que se puede extraer del XML)

### 19.3. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.

- Generación de un canal RSS para los contenidos que se muestran en la página principal.
- Uso de AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar una incidencia para una página de usuario).
- Puntuación de incidencias. Cada visitante (registrado o no) puede dar un “+1” a cualquier incidencia del sitio. La suma de “+” que ha obtenido una incidencia se verá cada vez que se vea la incidencias en el sitio.
- Comentarios a incidencias. Cada usuario registrado puede comentar cualquier incidencia del sitio. Estos comentarios se podrán ver luego en la página personal.

## 19.4. Entrega de la práctica

**Fecha límite de entrega de la práctica:** sábado, 24 de mayo de 2014 a las 03:00 (hora española peninsular).

**Fecha de publicación de notas:** lunes, 26 de mayo de 2014, en la plataforma Moodle.

**Fecha de revisión:** miércoles, 28 de mayo de 2014 a las 12:00. Se requerirá a algunos alumnos que asistan a la revisión **en persona**; se informará de ello en el mensaje de publicación de notas.

La práctica se entregará subiéndola al recurso habilitado a tal fin en el sitio Moodle de la asignatura. Los alumnos que no entreguen la práctica de esta forma serán considerados como no presentados en lo que a la entrega de prácticas se refiere. Los que la entreguen podrán ser llamados a realizar también una entrega presencial, que tendrá lugar en la fecha y hora exacta se les comunicará oportunamente. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

Para entregar la práctica en el Moodle, cada alumno subirá al recurso habilitado a tal fin un fichero tar.gz con todo el código fuente de la práctica. El fichero se habrá de llamar practica-user.tar.gz, siendo “user” el nombre de la cuenta del alumno en el laboratorio.

El fichero que se entregue deberá constar de un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos con datos suficientes como para poder probarlo. Estos datos incluirán al menos dos usuarios con sus datos correspondientes, y con al menos cinco incidencias en su página personal. Se incluirá también un fichero README con los siguientes datos:

- Nombre de la asignatura.

- Nombre completo del alumno.
- Nombre de su cuenta en el laboratorio.
- Nombres y contraseñas de los usuarios creados para la práctica. Éstos deberán incluir al menos un usuario con cuenta “marty” y contraseña “marty” y otro usuario con cuenta “doc” y contraseña “doc”.
- Resumen de las peculiaridades que se quieran mencionar sobre lo implementado en la parte obligatoria.
- Lista de funcionalidades opcionales que se hayan implementado, y breve descripción de cada una.
- URL del vídeo demostración en YouTube que muestre la funcionalidad básica
- URL del vídeo demostración en YouTube con la funcionalidad optativa, si se ha realizado funcionalidad optativa

Además, parte de la información del fichero README se incluirá a su vez en un formulario web a la hora de realizar la entrega.

Los vídeos de demostración serán de una duración máxima de 3 minutos (cada uno), y consistirán en una captura de pantalla de un navegador web utilizando la aplicación, y mostrando lo mejor posible la funcionalidad correspondiente (básica u opcional). Se valorará negativamente que los vídeos duren más de 3 minutos (de la experiencia de cursos pasados, tres minutos es un tiempo más que suficiente si uno no entra en detalles que no son importantes). Siempre que sea posible, el alumno comentará en el audio del vídeo lo que vaya ocurriendo en la captura. Los vídeos se colocarán en YouTube y deberán ser accesibles públicamente al menos hasta el 31 de mayo, fecha a partir de la cual los alumnos pueden retirar el vídeo (o indicarlo como privado).

Hay muchas herramientas que permiten realizar la captura de pantalla. Por ejemplo, en GNU/Linux puede usarse Gtk-RecordMyDesktop o Istanbul (ambas disponibles en Ubuntu). Incluso hay alguna aplicación web como Screen-O-Matic. Es importante que la captura sea realizada de forma que se distinga razonablemente lo que se grabe en el vídeo.

En caso de que convenga editar el vídeo resultante (por ejemplo, para eliminar tiempos de espera) puede usarse un editor de vídeo, pero siempre deberá ser indicado que se ha hecho tal cosa con un comentario en el audio, o un texto en el vídeo. Hay muchas herramientas que permiten realizar esta edición. Por ejemplo, en GNU/Linux puede usarse OpenShot o PiTiVi.

## 19.5. Notas y comentarios

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura con la versión de Django que se ha usado en prácticas (Django 1.6.2).

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos de los canales que alimentarán las revistas.

Los canales (feeds) RSS que produce la aplicación web realizada en la práctica deberán funcionar al menos con el navegador Firefox (considerándolos como canales RSS) disponibles en el laboratorio.

## 20. Ejercicios complementarios de varios temas

A continuación, algunos ejercicios relacionados con el temario de la asignatura. Algunos de ellos han sido propuestos en exámenes de ediciones previas, o en asignaturas con temarios similares.

### 20.1. Números primos

Se pide realizar una aplicación web que, dado un número, calcule si es primo o no. El número se indica como recurso, con URLs de la forma `http://primos.org/34` (si el número a probar es “34”). Para esta aplicación:

1. Escribir la petición y la respuesta HTTP que se podría observar para el caso de que se pruebe el número 34.
2. Escribir el código de la aplicación (sin usar un entorno de desarrollo de aplicaciones web). Escribir el código en Python, pseudo-Python o pseudocódigo. Puede usarse un método “IsPrime”, que acepta un número como parámetro, y devuelve True si ese número es primo, y False en caso contrario.
3. Se quiere que la aplicación mantenga una caché de los números ya probados, para evitar volver a probar un número si ya se calculó si era primo. Explicar las modificaciones que se verán en el intercambio HTTP, y en el código de la aplicación.
4. Se quiere que la aplicación, tal y como la describía el enunciado al principio de este ejercicio, siga funcionando en presencia de caídas y posteriores recuperaciones del servidor. ¿Qué cambios habrá que hacerle?
5. Lo mismo, en el caso de la aplicación con caché, tal y como se describe dos apartados más arriba.

### 20.2. Autenticación

Una aplicación web dada permite el acceso a cierto recurso, “/resource”, sólo a usuarios que se hayan autenticado previamente. Los usuarios se autentican mediante nombre de usuario y contraseña. La autenticación se realiza mediante POST a un recurso “/login”. Ese mismo recurso, si recibe un GET, sirve un formulario para poder realizar la autenticación. En este caso, se plantean las siguientes preguntas:

1. Describir (indicando las cabeceras relevantes y el contenido del cuerpo de los mensajes) las interacciones HTTP, desde que un usuario se quiere autenticar, y pincha en la URL para recibir el formulario, hasta que este usuario recibe un mensaje de bienvenida indicando que está ya autenticado.

2. Escribe el código de una vista (view) que de servicio al recurso “/login”. Escríbelo como se haría en una view Django (pero si prefieres, usando pseudo-Python o pseudocódigo).
3. Describe la interacción HTTP que se producirá desde que un navegador invoca la un GET sobre “/resource” hasta que recibe la pertinente respuesta de la aplicación web. Hazlo primero en el caso de que el navegador se haya autenticado previamente como usuario, y luego en caso de que no lo haya hecho.

### 20.3. Recomendaciones

Te han pedido que diseñes un servicio en Internet para elegir, comentar y recibir recomendaciones sobre lugares para pasar las vacaciones. Las características principales del sistema serán:

1. La información, comentarios y recomendaciones siempre estarán referidos a un lugar (un pueblo, una playa, una zona).
2. Cualquier usuario del servicio podrá “abrir” un nuevo lugar, simplemente indicando su nombre y subiendo una descripción del mismo. A partir de ese momento, habrá una URL en el servicio que mostrará esa información. Nos referiremos a esa URL como “la página del lugar”.
3. Cualquier usuario del servicio (incluyendo el que lo abrió) podrá modificar la descripción de un lugar y/o añadir un comentario. Los comentarios y los cambios en la descripción se reflejarán inmediatamente en la página del lugar correspondiente.
4. Cualquier usuario del servicio podrá “elegir” un lugar. Para ello, tendrá un botón que podrá pulsar en la página de ese lugar.
5. Cualquier usuario del servicio podrá pedir que se le recomiende un lugar, según las elecciones pasadas propias y de otros usuarios. El algoritmo que el servicio use para realizar estas recomendaciones no es objeto del diseño.
6. No se quieren mantener cuentas de usuarios, pero sí se quiere poder diferenciar entre usuarios diferentes al menos para las elecciones y las recomendaciones (para que el algoritmo pueda diferenciar entre elecciones propias y elecciones de otros).
7. El sitio ofrecerá, para cada lugar, un canal RSS con los comentarios sobre ese lugar. También habrá un canal RSS, único para todo el sitio, con los últimos lugares sobre los que se ha comentado.



Salvo cuando se indique otra cosa, se supone que un usuario corresponde con un navegador en un ordenador concreto.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Detalla un esquema de URLs que permita nombrar, siguiendo en lo posible el diseño REST, todos los elementos del servicio. Procura no usar URLs innecesarias.
2. Describe todas las interacciones HTTP que tendrán lugar en el sistema para abrir un lugar. Detalla las URLs implicadas, e indica las cabeceras más relevantes.
3. Ídem para realizar un comentario sobre un lugar. En la página del lugar habrá un formulario para poner comentarios, el usuario lo rellenará y a continuación lo verá en esa misma página del lugar (no se usa AJAX en este apartado).
4. Ídem para elegir un lugar. El usuario habrá de estar a la vista del lugar que quiere elegir, y una vez elegido, tendrá que verlo como elegido en esa misma página (no se usa AJAX en este apartado).
5. Cuando un usuario cambie de navegador, querrá seguir siendo reconocido por el sistema. Diseña un mecanismo, lo más simple posible, que le permita hacerlo, manteniendo garantías de que quien no tenga acceso a su navegador no podrá colocarse en su lugar desde otro. Si es posible, diseñalo sin usar el correo electrónico.
6. Describe los cambios que habría que hacer al sistema para que en la página de cada lugar cualquier usuario pueda, además de comentarios, subir fotos.
7. Describe los cambios que habría que hacer en el sistema para que la elección de un lugar se pudiera expresar sin que se produzca una recarga de página, usando AJAX.
8. ¿Se podría construir un gadget, para integrar en un mashup, que mostrase los últimos comentarios que se están poniendo en el servicio? Explica qué partes del servicio especificado en la primera parte del ejercicio usarías, y si es caso, qué modificaciones del servicio harían falta.

## 20.4. Geolocalización

Se decide construir un sitio para permitir que sus usuarios realicen anotaciones geolocalizadas que puedan ser consultadas por otros usuarios. Las características principales del sistema serán:

1. Cualquier usuario del sitio podrá subir una anotación geolocalizada. Para ello, rellenará un formulario en su navegador en el que especificará el texto que constituirá la anotación y sus coordenadas (latitud y longitud).
2. Cualquier usuario podrá consultar información geolocalizada, de varias formas:
  - Especificando unas coordenadas (latitud y longitud) y una distancia en un formulario en el navegador. El sistema devolverá una página HTML con todas las anotaciones (incluyendo sus coordenadas y el texto correspondiente) que estén cerca de las coordenadas especificadas (a menos de la distancia indicada).
  - Especificando unas coordenadas (latitud y longitud) y una distancia como parte de una URL del servicio, y obteniendo como respuesta un canal GeoRSS con todas las anotaciones (incluyendo sus coordenadas y el texto correspondiente) que estén cerca de las coordenadas especificadas (a menos de la distancia indicada).
  - Especificando unas coordenadas (latitud y longitud) y una distancia como parte de una URL del servicio, y obteniendo como respuesta un mapa con los puntos anotados (en formato PNG).
  - Especificando una cadena de texto en un formulario en el navegador. El sistema devolverá una página HTML con todas las anotaciones (incluyendo sus coordenadas y el texto correspondiente) que incluyan ese texto.
3. Los usuarios podrán usar el sitio sin tener que abrir cuenta (de hecho, el sitio no mantendrá cuentas).
4. Cualquier anotación podrá ser editada (para modificarla o eliminarla) las veces que se quiera, si se hace desde el mismo navegador desde el que se creó.

En particular, y teniendo en cuenta los requisitos anteriores, se pide:

1. Describe todas las interacciones HTTP que tendrán lugar en el sistema para crear una anotación. Detalla las URLs implicadas, e indica las cabeceras más relevantes.
2. Ídem para ver como página HTML las anotaciones cercanas a una posición dada por sus coordenadas.
3. Ídem para editar una anotación previamente creada desde el mismo navegador.

4. Se quiere que si un usuario pierde su ordenador, y pasa a usar uno nuevo, pueda seguir editando las anotaciones que creó. Describe un mecanismo que lo permita, sin obligar al usuario a crear una cuenta en el sistema.
5. Se quiere utilizar el servicio de consulta de anotaciones desde un programa de gestión de mapas. El programa ya tiene funcionalidad de mostrar mapas, y de mostrar información asociada con un punto cualquiera del mapa. Se pretende que se utilice esta funcionalidad de mostrar información para mostrar las anotaciones. Explicar cómo se podría usar el servicio descrito en la primera parte de este ejercicio. Indica las URLs y las transacciones HTTP involucradas (indicando sus principales cabeceras) para que la aplicación pueda mostrar las anotaciones cercanas a un punto del mapa.
6. Indica cómo se podría usar el servicio descrito en la primera parte del ejercicio para que desde la aplicación del apartado anterior se puedan también crear anotaciones. ¿Puede decirse que la parte del servicio que has usado sigue las directrices REST?
7. Pasado un tiempo se plantea la posibilidad de incorporar cuentas de usuario para que estos puedan autenticarse en el sitio web. Describe brevemente 2 mecanismos (en cuanto a interacción navegador-servicio) que podrían usarse con HTTP para realizar la autenticación y las principales ventajas e inconvenientes de cada uno.

## 21. Prácticas de entrega voluntaria de cursos pasados

### 21.1. Prácticas de entrega voluntaria (curso 2012-2013)

#### 21.1.1. Práctica 1 (entrega voluntaria)

**Fecha recomendada de entrega:** Antes del 12 de marzo.

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs. La aplicación funcionará únicamente con datos en memoria: se supone que cada vez que la aplicación muera y vuelva a ser lanzada, habrá perdido todo su estado anterior. La aplicación tendrá que realizarse según un esquema de clases similar al explicado en clase.

El funcionamiento de la aplicación será el siguiente:

- Recursos que comienzan por el prefijo “/acorta/” (invocados mediante GET). Estos recursos se utilizarán para devolver URLs acortadas, por el procedimiento de proporcionar un número entero secuencial, comenzando por 0, para cada nueva petición de acortamiento de una URL que se reciba. Si se recibe una petición para una URL ya acortada, se devolverá la URL acortada que se devolvió en su momento. La URL a acortar se especificará como parte del nombre de recurso, justo a partir de “/acorta/” (quitando la parte “http://” de la URL.

Así, por ejemplo, si se quiere acortar

`http://docencia.etsit.urjc.es`

y la aplicación está en el puerto 1234 de la máquina “localhost”, se invocará (mediante GET) la URL

`http://localhost:1234/acorta/docencia.etsit.urjc.es`

Como respuesta, la aplicación devolverá (en el cuerpo de la respuesta HTTP) la URL acortada, por ejemplo

`http://localhost:1234/3`

Si a continuación se trata de acortar la URL

`http://docencia.etsit.urjc.es/moodle/course/view.php?id=25`

se invocará para ello la URL

`http://localhost:1234/acorta/docencia.etsit.urjc.es/moodle/course/view.php?id=`

y se recibirá como respuesta la URL acortada

`http://localhost:1234/4`

Si se vuelve a intentar acortar la URL

`http://docencia.etsit.urjc.es`

como ya ha sido acortada previamente, se devolverá la misma URL corta:

`http://localhost:1234/3`

- Recursos correspondientes a URLs acortadas. Estos serán números con el prefijo “/”. Cuando la aplicación reciba un GET sobre uno de estos recursos, si el número corresponde a una URL acortada, devolverá un HTTP REDIRECT a la URL real. Si no la tiene, devolverá HTTP ERROR “Recurso no disponible”.

Por ejemplo, si se recibe

`http://localhost:1234/3`

la aplicación devolverá un HTTP redirect a la URL

`http://docencia.etsit.urjc.es`

- Recurso “/”. Si se invoca este recurso con GET, se obtendrá un listado de todas las URLs reales y acortadas que maneja la aplicación en este momento.

### Comentario

Se recomienda utilizar dos diccionarios para almacenar las URLs reales y los números de las URLs acortadas. En uno de ellos, la clave de búsqueda será la URL real, y se utilizará para saber si una URL real ya está acortada, y en su caso saber cuál es el número de la URL corta correspondiente.

En el otro diccionario la clave de búsqueda será el número de la URL acortada, y se utilizará para localizar las URLs reales dadas las cortas. De todas formas, son posibles (e incluso más eficientes) otras estructuras de datos.

#### 21.1.2. Práctica 2 (entrega voluntaria)

**Fecha recomendada de entrega:** Antes del 9 de abril.

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs utilizando Django. Su enunciado será igual que el de la práctica 1 de entrega voluntaria (ejercicio 21.1.1), salvo en los siguientes aspectos:

- Se implementará utilizando Django.
- Tendrá que almacenar la información relativa a las URLs que acorta en una base de datos, de forma que aunque la aplicación sea rearrancada, las URLs acortadas sigan funcionando adecuadamente.

## 21.2. Prácticas de entrega voluntaria (curso 2011-2012)

### 21.2.1. Práctica 1 (entrega voluntaria)

Esta práctica tendrá como objetivo la creación de una aplicación web para acceso a los artículos de Wikipedia con almacenamiento en cache.

La aplicación servirá dos tipos de recursos:

- “/decorated/article”: servirá la página correspondiente al artículo “article” de la Wikipedia en inglés, decorado con las cajas auxiliares.
- “/raw/article”: servirá la página correspondiente al artículo “article” de la Wikipedia en inglés, sin decorar con las cajas auxiliares.

La página “decorada” es accesible mediante URLs de la siguiente forma (para el artículo “pencil” de la Wikipedia en inglés):

`http://en.wikipedia.org/w/index.php?title=pencil&action=view`

El contenido que sirven estas URLs está previsto para ser directamente mostrado, como página HTML completa, por un navegador.

La página “no decorada” es accesible mediante URLs de la siguiente forma (para el artículo “pencil” de la Wikipedia en inglés):

`http://en.wikipedia.org/w/index.php?title=pencil&action=render`

El contenido que sirven estas URLs está previsto para ser directamente empotrable en una página HTML, dentro del elemento “body” (y por lo tanto la aplicación web tendrá que aportar el HTML necesario para acabar teniendo una página HTML correcta).

Cualquiera de los dos tipos de recursos se comportará de la misma forma. Si es invocado mediante GET, usará para responder el artículo que tenga en cache. Si no lo tiene, lo bajará previamente accediendo a la URL adecuada, que se indicó anteriormente, lo almacenará en la cache, y lo usará para responder.

La respuesta, en cada caso, será una página HTML que contenga en la parte superior la siguiente información:

- Nombre del artículo, junto con la indicación “(decorated)” o “(non decorated)”, según corresponda. Por ejemplo, “Pencil (decorated)”.
- Enlaces a las páginas con el artículo en la Wikipedia (versiones decorada y no decorada)
- Enlace a la historia de modificaciones del artículo en la Wikipedia

- Enlace al último artículos de la Wikipedia que ha servido la aplicación (al navegador que le hizo la petición, o a cualquier otro).
- Línea de separación (elemento “hr”).

Y a continuación el texto correspondiente del artículo de la Wikipedia (decorado o no decorado, según sea el nombre del recurso invocado).

En caso de que se pida un artículo que no exista en la Wikipedia, se devolverá el código de error correspondiente, y se marcará en la cache, de alguna forma, que ese artículo no existe, para no tener que buscarlo en caso de que vuelva a ser pedido. En general, puede usarse algún texto que aparezca en la página que devuelve Wikipedia cuando sirve la página de un artículo que no existe, como por ejemplo:

```
<div class="noarticletext">
```

### **Materiales de apoyo:**

- Parámetros de index.php en Wikipedia (MediaWiki): [http://www.mediawiki.org/wiki/Manual:Parameters\\_to\\_index.php#View\\_and\\_render](http://www.mediawiki.org/wiki/Manual:Parameters_to_index.php#View_and_render)

### **Comentario:**

En algunas circunstancias, el servidor de Wikipedia puede devolver un código de redirección (por ejemplo, un “301 Moved permanently”). Téngase en cuenta que la aplicación ha de reconocer esta situación, y repetir el GET en la URL a la que se dirige.

## **21.2.2. Práctica 2 (entrega voluntaria)**

Realiza lo especificado en la práctica 1 (ejercicio 21.2.1), pero usando el entorno de desarrollo Django. En particular, utiliza plantillas (templates) para la generación de las páginas HTML, tablas en base de datos para almacenar las páginas de Wikipedia descargada, y añade la siguiente funcionalidad:

- Utilizando el módulo correspondiente de Django, añade usuarios, que se autenticarán en el recurso “/login”. Las cuentas de usuario estarán dadas de alta por el administrador (vía módulo Admin de Django). Si una página es bajada por un usuario autenticado se incluirá en la parte superior el mensaje “Usuario: user (logout)”, siendo “user” el identificador de usuario correspondiente, y “logout” un enlace al recurso que puede utilizar el usuario para salir de su cuenta. Si la página es bajada sin haberse autenticado previamente, en lugar de ese mensaje se incluirá “Usuario anónimo (login)”, siendo “login” un enlace al recurso “/login”.

- La aplicación atenderá el recurso “/”, en el que ofrecerá (si se invoca con “GET”) una lista de los artículos de Wikipedia disponibles en la base de datos, junto al enlace correspondiente (bajo “/decorated” o bajo “/raw”) para descargarla, y el mensaje “decorated” o “raw”, según el tipo de artículo descargado.

## **21.3. Prácticas de entrega voluntaria (curso 2010-2011)**

### **21.3.1. Práctica 1 (entrega voluntaria)**

Esta práctica tendrá como objetivo la creación de una aplicación web para acceso a los artículos de Wikipedia, con almacenamiento en cache, y con consulta en varios idiomas.

La aplicación servirá dos tipos de recursos:

- “/article”: servirá la página correspondiente al artículo “article” de la Wikipedia en inglés.
- “/language/article”: servirá la página correspondiente al artículo “article” correspondiente al idioma “language”, expresado mediante el código ISO de dos letras. Bastará con que funcione con los idiomas inglés (en) y español (es).

La página que se bajará de la Wikipedia para cada artículo será la “no decorada”, accesible mediante URLs de la siguiente forma (para el artículo “pencil” de la Wikipedia en inglés):

`http://en.wikipedia.org/w/index.php?action=render&title=pencil`

El contenido que sirve esta URL está previsto para ser directamente empotrable en una página HTML, dentro del elemento “body”.

Cualquiera de los dos tipos de recursos se comportará de la misma forma. Si es invocado mediante GET, usará para responder el artículo que tenga en cache. Si no lo tiene, lo bajará previamente accediendo a la URL de página no decorada, que se indicó anteriormente, lo almacenará en la cache, y lo usará para responder.

La respuesta será una página HTML que contenga:

- Título de la página (nombre del artículo).
- Enlace a la página con el artículo en la Wikipedia (versión decorada)
- Enlace a la historia de modificaciones del artículo en la Wikipedia



- Enlace a los tres últimos artículos de la Wikipedia que ha servido la aplicación (al navegador que le hizo la petición, o a cualquier otro).
- Texto de la página no decorada del artículo de la Wikipedia.

En caso de que se pida un artículo que no exista en la Wikipedia, se devolverá el código de error correspondiente, y se marcará en la cache, de alguna forma, que ese artículo no existe, para no tener que buscarlo en caso de que vuelva a ser pedido. En general, puede usarse algún texto que aparezca en la página que devuelve Wikipedia cuando sirve la página de un artículo que no existe, como por ejemplo:

```
<div class="noarticletext">
```

### **Materiales de apoyo:**

- Parámetros de index.php en Wikipedia (MediaWiki): [http://www.mediawiki.org/wiki/Manual:Parameters\\_to\\_index.php#View\\_and\\_render](http://www.mediawiki.org/wiki/Manual:Parameters_to_index.php#View_and_render)

### **21.3.2. Práctica 2 (entrega voluntaria)**

Esta práctica consistirá en la realización de un gestor de contenidos que tenga las siguientes características:

- Funcionalidad de “Gestor de contenidos con usuarios, con control estricto de actualización y uso de base de datos” (ejercicio 17.3)
- Implementación de HEAD para todos los recursos.
- Terminación de una sesión autenticada. Para ello se usará el recurso “/logout”.
- Además, cada página que se obtenga con un GET irá anotada con la siguiente información:
  - Sólo si la página no la está viendo un usuario autenticado. Enlace que permita la autenticación del usuario que creó la página (a falta de la contraseña). Aparecerá con la cadena “Autor: user”, siendo “user” el nombre de usuario que creó la página, y estando enlazado a “/login,user,”.
  - Enlace que permita ver el mensaje HTTP que envió el navegador para poder ver esa página (se puede suponer que esa fue la última página descargada desde este navegador).

- Enlace que permita ver la respuesta HTTP que envió el servidor para poder ver esa página (se puede suponer que esa fue la última página descargada desde este navegador).

Además, opcionalmente, podrá tener:

- Creación de cuentas de usuario. Para ello se usará un recurso “/signin,user,passwd”, sobre el que un GET creará el usuario “user” con la contraseña “passwd”, si ese usuario no existía ya.
- Subida de páginas con POST. en lugar de PUT. Se usará un POST para subir una nueva página. No hace falta implementar un formulario HTML que invoque el POST, pero también se podría hacer.
- Una implementación que no tenga la limitación de que los enlaces al mensaje HTTP del navegador y del servidor sean de la última página descargada, sino de los de la descarga de la página que los tiene, sea la última o no.

Realizar la entrega en un fichero tar.gz o .zip, incluyendo además del código fuente los ficheros de SQLite3 necesarios, y un fichero README que resuma la funcionalidad exacta que se ha implementado (en particular, que detalle la funcionalidad opcional implementada).

### 21.3.3. Práctica 3 (entrega voluntaria)

Realiza lo especificado en la práctica 2, pero usando el entorno de desarrollo Django. Donde lo creas oportuno, interpreta las especificaciones en el contexto de las facilidades que proporciona Django. Por ejemplo, la autenticación de usuarios se puede hacer vía un formulario de login (con el POST correspondiente) usando los módulos que proporciona Django para ello.

Igualmente, extiende las especificaciones en lo que te sea simple al usar las facilidades de Django. Por ejemplo, la gestión de usuarios (creación y borrado de usuarios) puede hacerse fácilmente usando módulos Django.

En la medida que sea razonable, usa POST (con los correspondientes formularios) en lugar de PUT. Opcionalmente, mantén ambas funcionalidades (subida de contenidos vía PUT, como se indicaba en la práctica 2, y vía POST, como se está recomendando para ésta).

#### Notas:

Parte de la especificación requiere almacenar las cabeceras de la respuesta del servidor al navegador. En Django, las cabeceras se van añadiendo al objeto `HTTPResponse` (o similar), y por tanto será necesario extraerlas de él. La forma más simple, y suficiente para estas prácticas, es simplemente convertir el objeto

HTTPResponse en string: “str(response)”. Si se quiere, se puede manipular el string resultante, para obtener las cabeceras en un formato más parecido al de la práctica 1, pero esto no será necesario para la versión básica.

#### **21.3.4. Práctica 4 (entrega voluntaria)**

Realización de lo especificado en la práctica 3 de entrega voluntaria, utilizando para la implementación las posibilidades avanzadas de Django, incluyendo especialmente las plantillas, y si es posible el sitio de administración, los usuarios y las sesiones Django. La parte básica seguirá siendo básica, y la opcional, opcional (más la adición, opcional, que se comenta más adelante).

La funcionalidad de esta práctica es, por lo tanto, la misma que la de la práctica 3. Pero a diferencia de la práctica 3, en este caso sí se pide usar los módulos “de alto nivel” de Django.

La URL que se usaba en las prácticas 2 y 3 para autenticarse pasa a ser “/login”, que en el caso de recibir un GET devolverá el formulario para autenticarse, y en caso de recibir un POST gestionará la autenticación.

A la parte opcional de las prácticas 2 y 3, que sigue siendo opcional, se añade la de modificar el contenido de las páginas con formularios (usando métodos POST para la actualización), y de crear nuevas páginas también mediante formularios y POST. Para la actualización se sugiere que se usen nombres de recurso de la forma “/edit/name”, siendo “name” el nombre de la página. Para la creación se sugiere que se use un nombre de recurso de la forma “/create”.

Con respecto a la opción de crear usuarios, ahora la opción cambia a servir una URL “/signin” que devuelva el formulario para crearse una cuenta, y que cuanto reciba un POST gestione la creación de la cuenta.

## 22. Prácticas finales de cursos pasados

### 22.1. Práctica final (2013, mayo)

La práctica final de la asignatura consiste en la creación de un selector de noticias a partir de canales, como aplicación web. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener. Llamaremos a la aplicación MiRevista.

### 22.2. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- MiRevista se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin Site” para mantener los usuarios con cuenta en el sistema, y para la gestión general de las bases de datos necesarias. Todas las bases de datos que mantenga MiRevista tendrá que ser accesible vía este “Admin Site”.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que MiRevista tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones.
  - Una caja para entrar (hacer login en el sitio), o para salir (si ya se ha entrado). En caso de que no se haya entrado en una cuenta, esta caja permitirá al visitante introducir su identificador de usuario y su contraseña. En caso de que ya se haya entrado, esta caja mostrará el identificador del usuario y permitirá salir de la cuenta (logout).
  - Un pie de página con una nota de copyright.

Cada una de estas partes estará marcada con propiedades “id” en HTML, para poder ser referenciadas en hojas de estilo CSS.

- Se utilizarán hojas de estilo CSS para determinar la apariencia de MiRevista. Estas hojas definirán al menos el color y el tamaño de la letra, y el color de fondo de cada una de las partes (elementos) marcadas con id que se indican en el apartado anterior.

Funcionamiento general:

- Los usuarios serán dados de alta en MiRevista mediante el módulo “Admin Site” de Django. Una vez estén dados de alta, serán considerados “usuarios registrados”.
- Los usuarios registrados podrán crear su selección de noticias en MiRevista. Para ello, dispondrán de una página personal, en la que trabajarán. Llamaremos a esta página la “revista del usuario”.
- La selección de noticias de su revista la realizará cada usuario a partir de canales RSS de sitios web ya disponibles en el sitio.
- Además, si hay un canal no disponible en el sitio, un usuario podrá indicar sus datos para que pase a estar disponible.
- Los contenidos de cada canal se actualizarán sólo cuando un usuario indique que quiere que se actualicen (esta indicación se hará por separado para cada canal que se quiera actualizar).
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la revista de cada usuario, para todos los usuarios del sitio.

### 22.3. Funcionalidad mínima

Interfaz pública: recursos a servir como páginas HTML completas (pensadas para ser vistas en el navegador) para cualquier visitante (sea usuario registrado o no):

- /: Página principal de MiRevista. Mostrará la lista de las revistas disponibles, ordenadas por fecha de actualización, en orden inverso (las revistas actualizadas más recientemente, primero). Para cada revista se mostrará su título (como un enlace a la página de la revista), el nombre de su usuario y la fecha de su última actualización (fecha en que se añadió una noticia a esa revista por última vez). Si a una revista aún no se le hubiera puesto título, este título será “Revista de usuario”, donde “usuario” es el identificador de usuario del usuario en cuestión.

- /usuario: Página de la revista de un usuario. Si la url es “/usuario”, es que corresponde al usuario “usuario”. Mostrará las 10 noticias de la revista de ese usuario (no puede haber más de 10, como se indicará más adelante). Para cada noticia se mostrará la “información pública de noticia”, ver más adelante.
- /usuario/rss: Canal RSS para la revista de ese usuario.
- /ayuda: Página con información HTML explicando el funcionamiento de MiRevista.

Además, todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder la ayuda (url /ayuda) con el texto “Ayuda”.

Además, todas las página que no sean la principal tendrán otra opción de menú para la url /, con el texto “Revistas”.

Interfaz privada: recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado).

- Todos los recursos de la interfaz pública.
- /canales: Página con la lista de los canales disponibles en MiRevista:
  - Para cada canal se mostrará el nombre del canal (apuntando a la página de ese canal en MiRevista, ver más adelante), el logo del canal, el número de mensajes disponibles para el canal, y la fecha en que fue actualizado por última vez.
  - Además, en esta página se mostrará un formulario en el que se podrá introducir una url, que se interpretará como la url de un nuevo canal. Esta será la forma de añadir un nuevo canal para que esté disponible en el sitio. Cuando se añada un nuevo canal se tratarán de actualizar sus contenidos a partir de la url indicada: si esta operación falla (bien porque la url no está disponible, bien porque no se puede interpretar su contenido como un documento RSS), no se añadirá el canal como disponible. En cualquier caso, tras tratar de añadir un nuevo canal se volverá a ver la página /canales en el navegador.
- /canales/num: Página de un canal en MiRevista. “num” es el número de orden en que se hizo disponible (si fue el segundo canal que se hizo disponible en el sitio, será /canales/2). Mostrará:
  - El nombre del canal (según venga como titulo en el canal RSS correspondiente) como enlace apuntando al sitio web donde se puede ver el contenido del canal (ojo: el contenido original, no el canal RSS)

- Junto a él pondrá entre paréntesis “canal”, como enlace al canal RSS correspondiente en el sitio original
  - Un botón para actualizar el canal. Si se pulsa este botón, se tratarán de actualizar las noticias de ese canal accediendo al documento RSS correspondiente en su sitio web de origen. Al terminar la operación se volverá a mostrar esta misma página /canales/num.
  - La lista de noticias de ese canal, incluyendo para cada una la “información pública de noticia”, ver más adelante.
  - Junto a cada noticia de la lista, se incluirá un botón que permitirá elegir la noticia para la revista del usuario autenticado. Si al añadirla la lista de noticias de esa revista fuera de más de 10, se eliminarán las que se eligieron hace más tiempo, de forma que no queden más de 10. Tras añadir una noticia a la revista del usuario, se volverá a ver en el navegador la página /canales/num correspondiente al canal en que se seleccionó.
- En la página /usuario que corresponde al usuario autenticado se mostrará, además de lo ya mencionado para la interfaz pública, un formulario en el que se podrá especificar la siguiente información:
    - Los parámetros CSS para el usuario autenticado (al menos los indicados anteriormente para ser manejados por un documento CSS). Si el usuario los cambia, a partir de ese momento deberá verse el sitio con los nuevos valores, y para ello deberá servirse un nuevo documento CSS.
    - El título de la revista del usuario autenticado.

Si es preciso, se añadirán más recursos (pero sólo si es realmente preciso) para poder satisfacer los requisitos especificados.

Dados los recursos mencionados anteriormente, no se permitirán los nombres de usuario “canales” ni “ayuda” (pero no hay que hacer ninguna comprobación para esto: se asume que no se darán de alta esos usuarios en el Admin Site).

Como información pública de noticia se mostrará:

- El título de la noticia, como enlace a la noticia en el sitio web original.
- La fecha en que fue publicada la noticia en el sitio original (junto al texto “publicada en”).
- La fecha en que fue seleccionada para esta revista (junto al texto “elegida en”).

- El contenido de la noticia.
- El nombre del canal de donde viene la noticia, como enlace a la página de ese canal en MiRevista.

## 22.4. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.
- Generación de un canal RSS para los contenidos que se muestran en la página principal.
- Uso de AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar una noticia para una revista).
- Puntuación de noticias. Cada visitante (registrado o no) puede dar un “+1” a cualquier noticia del sitio. La suma de “+” que ha obtenido una noticia se verá cada vez que se vea la noticia en el sitio.
- Comentarios a revistas. Cada usuario registrado puede comentar cualquier revista del sitio. Estos comentarios se podrán ver luego en la página de la revista.
- Autodescubrimiento de canales. Dada una url (de un blog, por ejemplo), busca si en ella hay algún enlace que parece un canal. Si es así, ofrécelo al usuario para que lo pueda elegir. Esto se puede usar, por ejemplo, en la página que muestra el listado de canales, como una opción más para elegir canales (“especifica un blog para buscar sus canales”).

## 22.5. Entrega de la práctica

**Fecha límite de entrega de la práctica:** 22 de mayo de 2013.

La práctica se entregará subiéndola al recurso habilitado a tal fin en el sitio Moodle de la asignatura. Los alumnos que no entreguen la práctica de esta forma



serán considerados como no presentados en lo que a la entrega de prácticas se refiere. Los que la entreguen podrán ser llamados a realizar también una entrega presencial, que tendrá lugar en la fecha y hora exacta se les comunicará oportunamente. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

Para entregar la práctica en el Moodle, cada alumno subirá al recurso habilitado a tal fin un fichero tar.gz con todo el código fuente de la práctica. El fichero se habrá de llamar practica-user.tar.gz, siendo “user” el nombre de la cuenta del alumno en el laboratorio.

El fichero que se entregue deberá constar de un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos con datos suficientes como para poder probarlo. Estos datos incluirán al menos tres usuarios con sus datos correspondientes, y con al menos cinco noticias en su revista, y al menos tres canales RSS diferentes. Se incluirá también un fichero README con los siguientes datos:

- Nombre de la asignatura.
- Nombre completo del alumno.
- Nombre de su cuenta en el laboratorio.
- Nombres y contraseñas de los usuarios creados para la práctica. Éstos deberán incluir al menos un usuario con cuenta “marta” y contraseña “marta” y otro usuario con cuenta “pepe” y contraseña “pepe”.
- Canales disponibles en el sitio, incluyendo su URL
- Resumen de las peculiaridades que se quieran mencionar sobre lo implementado en la parte obligatoria.
- Lista de funcionalidades opcionales que se hayan implementado, y breve descripción de cada una.
- URL del vídeo demostración de la funcionalidad básica
- URL del vídeo demostración de la funcionalidad optativa, si se ha realizado funcionalidad optativa

El fichero README se incluirá también como comentario en el recurso de subida de la práctica, asegurándose de que las URLs incluidas en él son enlaces “pinchables”.

Los vídeos de demostración serán de una duración máxima de 2 minutos (cada uno), y consistirán en una captura de pantalla de un navegador web utilizando la

aplicación, y mostrando lo mejor posible la funcionalidad correspondiente (básica u opcional). Siempre que sea posible, el alumno comentará en el audio del vídeo lo que vaya ocurriendo en la captura. Los vídeos se colocarán en algún servicio de subida de vídeos en Internet (por ejemplo, Vimeo o YouTube).

Hay muchas herramientas que permiten realizar la captura de pantalla. Por ejemplo, en GNU/Linux puede usarse Gtk-RecordMyDesktop o Istanbul (ambas disponibles en Ubuntu). Es importante que la captura sea realizada de forma que se distinga razonablemente lo que se grabe en el vídeo.

En caso de que convenga editar el vídeo resultante (por ejemplo, para eliminar tiempos de espera) puede usarse un editor de vídeo, pero siempre deberá ser indicado que se ha hecho tal cosa con un comentario en el audio, o un texto en el vídeo. Hay muchas herramientas que permiten realizar esta edición. Por ejemplo, en GNU/Linux puede usarse OpenShot o PiTiVi.

## **22.6. Notas y comentarios**

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura con la versión de Django que se ha usado en prácticas (Django 1.4.5).

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos de los canales que alimentarán las revistas.

Los usuarios registrados pueden, en principio, hacer disponible cualquier canal de cualquier blog. Sin embargo, para la funcionalidad mínima es suficiente que MiRevista funcione con blogs de WordPress.com.

Los canales (feeds) RSS que produce la aplicación web realizada en la práctica deberán funcionar al menos con el navegador Firefox (considerándolos como canales RSS) disponibles en el laboratorio.

## **22.7. Práctica final (2012, diciembre)**

La práctica final de la asignatura consiste en la creación de un planeta, o agregador de canales, como aplicación web. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener. Llamaremos a la aplicación MiPlaneta.

### 22.7.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- MiPlaneta se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin Site” para mantener los usuarios con cuenta en el sistema, y para la gestión general de las bases de datos necesarias (todas las bases de datos que mantenga MiPlaneta tendrá que ser accesible vía este “Admin Site”).
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que MiPlaneta tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones.
  - Un pie de página con una nota de copyright.

Cada una de estas partes estará marcada con propiedades “id” en HTML, para poder ser referenciadas en hojas de estilo CSS.

- Se utilizarán hojas de estilo CSS para determinar la apariencia de MiPlaneta. Estas hojas definirán al menos el color de fondo y del texto, y alguna propiedad para las partes marcadas que se indican en el apartado anterior.

Funcionamiento general:

- Los usuarios serán dados de alta en MiPlaneta mediante el módulo “Admin Site” de Django. Una vez estén dados de alta, serán considerados “usuarios registrados”.
- Los usuarios registrados podrán especificar en MiPlaneta su número de usuario en el Moodle de la ETSIT. Por ejemplo, si la página de perfil de un usuario en el Moodle de la ETSIT es <http://docencia.etsit.urjc.es/moodle/user/profile.php?id=8> (llamaremos a la página a la que apunta

esta url la “página del usuario en el Moodle de la ETSIT”) su número de usuario es 8. Puede obtenerse el número de usuario en el Moodle de la ETSIT a través de los enlaces a ese usuario en los mensajes que pone en sus foros, por ejemplo.

- Cada usuario registrado podrá indicar el blog que le representa en MiPlaneta. Para ello, especificará la url del canal RSS correspondiente a ese blog.
- Habrá una url para actualizar los contenidos.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá los artículos en la base de datos e información pública para cada usuario.

### 22.7.2. Funcionalidad mínima

Interfaz pública: recursos a servir como páginas HTML completas (pensadas para ser vistas en el navegador) para cualquier visitante (sea usuario registrado o no):

- /: Página principal de MiPlaneta. Lista de los últimos 20 artículos, por fecha de publicación, en orden inverso (más nuevos primero). Para cada artículo se mostrará la “información pública de artículo”, ver más abajo.
- /users: Lista de usuarios registrados de MiPlaneta. Para cada usuario se mostrará la “información resumida de usuario”, ver más abajo.
- /users/alias: Información sobre el usuario que tiene el alias “alias” en MiPlaneta. El alias es el nombre de usuario que tiene como usuario registrado, fijado con el módulo “Admin Site”. Se incluirá la “información completa de usuario”, ver más abajo.
- /update: Actualización de los artículos de todos los blogs. Cuando sea invocada, se bajarán todos los canales y se almacenarán en la base de datos los artículos correspondientes. Si un artículo ya estaba en la base de datos, no debe almacenarse dos veces. Al terminar, enviará una redirección a la página principal.

Además, todas las páginas de la interfaz pública incluirán un formulario para poder autenticarse si se es usuario registrado, y un menú desde el que se podrá acceder a / (con el texto “página principal”), a /users (con el texto “listado de usuarios”) y a /update (con el texto “actualizar”).

Interfaz privada: recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado).

- Todos los recursos de la interfaz pública.
- /conf: Configuración de usuario. Tendrá un formulario en el que se podrá especificar:
  - Un número de usuario del Moodle de la ETSIT
  - La url del canal RSS de un blog
  - El color de fondo de todas las páginas del blog
  - El color del texto normal de todas las páginas del blog

Además, todas las páginas de la interfaz privada incluirán el nombre y la foto del usuario registrado (según aparecen en su perfil en Moodle de la ETSIT), una opción para cerrar la sesión y un menú que incluirá las mismas opciones que el menú público más otra que permita acceder a /conf con el texto “configuración”.

Tanto el color de fondo como el del texto normal de las páginas deberán recibirse en el navegador como parte de un documento CSS.

Detalles de las distintas informaciones mencionadas:

- Información pública de artículo. Se mostrará:
  - Del artículo: su título (que será un enlace al artículo en su blog original) y su contenido (tal y como venga especificado en el canal).
  - Del blog original que lo contiene: el nombre del blog, un enlace al blog, y otro a su canal RSS.
  - Del usuario del Moodle de la ETSIT correspondiente: el nombre, que será un enlace a “/users/alias” (el alias en MiPlaneta) y la foto.
- Información resumida de usuario. Se mostrará:
  - Del usuario del Moodle de la ETSIT correspondiente: el nombre, la foto, el enlace a su sitio web. El nombre será un enlace a “/users/alias” (el alias en MiPlaneta).
  - Del blog original que lo contiene: el nombre del blog, que será un enlace a ese mismo blog.
- Información completa de usuario. Se mostrará:
  - Del usuario del Moodle de la ETSIT correspondiente: el nombre, la foto, el enlace a su sitio web, y un enlace a su perfil en Moodle de la ETSIT.

- Del blog original que lo contiene: el nombre del blog, un enlace al blog, y otro a su canal RSS, todos los artículos almacenados para ese blog.
- De cada artículo: su título (que será un enlace al artículo en su blog original) y su contenido (tal y como venga especificado en el canal).

Además de estos recursos, se atenderá a cualquier otro que sea necesario para proporcionar la funcionalidad indicada.

### 22.7.3. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.
- Generación de un canal RSS para los contenidos que se muestran en la página principal.
- Uso de AJAX para algún aspecto de la práctica (por ejemplo, en los formularios de /conf)
- Puntuación de artículos. Cada usuario registrado puede puntuar cualquier artículo del sitio, por ejemplo entre 1 y 5. Estas puntuaciones se podrán ver luego junto al artículo en cuestión.
- Comentarios a artículos. Cada usuario registrado puede comentar cualquier artículo del sitio. Estos comentarios se podrán ver luego junto al artículo en cuestión (en la página de ese artículo).
- Soporte para logos. Cada blog o artículo de un blog se presentará junto con un logo que represente al blog en cuestión.
- Autodescubrimiento de canales. Dada una url (de un blog, por ejemplo), busca si en ella hay algún enlace que parece un canal. Si es así, ofrécelo al usuario para que lo pueda elegir. Esto se puede usar, por ejemplo, en la página de configuración de usuario, como una opción más para elegir canales (“especifica un blog para buscar sus canales”).

## 22.8. Práctica final (2011, diciembre)

La práctica final de la asignatura consiste en la creación de un sitio web de creación de revistas con resúmenes de información procedente de sitios terceros, MetaMagazine. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

### 22.8.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- La aplicación web se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin site” para mantener los usuarios con cuenta en el sistema, y para la gestión general de las bases de datos necesarias.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que toda la aplicación tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones también en la parte superior.
  - Un pie de página con una nota de copyright.
- Se utilizarán hojas de estilo CSS para determinar la apariencia de la aplicación.

Funcionamiento general:

- El sitio MetaMagazine ofrece como servicio la construcción de revistas con resúmenes de información obtenidos a partir de canales RSS de ciertos sitios terceros. Para construir una revista, primero se extraerán noticias de los canales correspondientes. Para cada noticia, se buscarán las URLs incluidas en su texto. Para cada URL, se visitará la página correspondiente, y se

extraerá de ella la información (texto, imágenes, etc.) deseada. Con esta información se compondrá una página HTML que será la que se sirva a los navegadores que visiten la revista de ese usuario.

- Cada usuario autenticado podrá construir una revista indicando en qué información de sitios terceros están interesados (eligiendo los canales RSS correspondientes), e indicando cuántas noticias de cada uno se tomarán como máximo cuando se actualice la revista. Cuando un usuario autenticado indica un nuevo canal en el que está interesado, el sistema genera una revista para ese sitio a partir de su canal (usando el número de noticias que ha seleccionado), y se lo muestra al usuario. Si el usuario lo acepta, se toma nota del sitio y de los contenidos de la revista en la base de datos.
- Cuando cualquier visitante de MetaMagazine acceda a la revista creada por un usuario, podrá ver la información almacenada para esa revista. Además, la página de la revista incluirá un mecanismo para actualizarla, bajando información de los sitios correspondientes. En la actualización, para cada canal sólo se considerará el número de noticias más actuales que haya seleccionado el creador de la revista (y se ignorarán las más antiguas, salvo que ya estén en la base de datos). No se eliminarán las noticias antiguas de la base de datos al actualizar las revistas.
- Cuando esté visitando MetaMagazine un visitante sin autenticar, le aparecerá una caja para autenticarse. Si es un usuario autenticado, le aparecerá un mecanismo para salir de la cuenta (“desautenticarse”).

### 22.8.2. Funcionalidad mínima

Esta es la funcionalidad mínima que habrá de proporcionar la aplicación:

- Para cada revista (correspondiente a un usuario registrado del sitio) se mostrará a cualquier visitante:
  - El título de la revista.
  - Un enlace a los canales y sitios web correspondientes a esos canales, y la fecha de última actualización (para cada uno de ellos).
  - Para cada canal, un mecanismo para actualizar en la base de datos la información extraída las páginas web que referencie.
  - El texto de las noticias de los sitios elegidos para esa revista.
  - Para cada noticia, un mecanismo para desplegar la información extraída las páginas web que referencie.



- Un mecanismo para desplegar (de una vez) la información extraída de todas las noticias.
- Para cada noticia, la información que se mostrará será:
  - Enlace a la página de la noticia en MetaMagazine.
  - Los enlaces a las páginas web cuya URL aparezca en la noticia.
  - Para cada una de esas páginas, las primeras 50 palabras que incluyan (basta con considerar, por ejemplo, las primeras 50 palabras incluidas en elementos  $\langle p \rangle$ ).
  - Para cada una de esas páginas, 5 de las imágenes que incluyan, si las hubiera.
  - Para cada una de esas páginas, los vídeos de Youtube, si los hubiera.
- Para cada revista (correspondiente a un usuario registrado del sitio) se mostrará al usuario que la construye:
  - Toda la información anterior, que se muestra también para cualquier visitante.
  - El título de la revista de forma que se pueda cambiar.
  - Una zona para incluir nuevos canales en la revista, que incluirá:
    - Un menú con la opción de sitios de los que se podrán incluir canales.
    - Un formulario para indicar qué canal del sitio elegido se incluirá.
  - Para cada canal de la revista, un mecanismo para eliminarlo.
- Como mínimo, se podrán seleccionar los siguientes tipos de canales:
  - Canales RSS correspondientes a usuarios de Twitter<sup>3</sup>.
  - Canales RSS correspondientes a usuarios de Identi.ca<sup>4</sup>.
  - Canales RSS correspondientes a usuarios de Youtube<sup>5</sup>.

---

<sup>3</sup>Para el usuario “jgbarah”:

[https://twitter.com/statuses/user\\_timeline/jgbarah.rss](https://twitter.com/statuses/user_timeline/jgbarah.rss)

<sup>4</sup>Para el usuario “jgbarah”:

<http://identi.ca/jgbarah/rss>

<sup>5</sup>Para el usuario “user”:

<http://gdata.youtube.com/feeds/api/videos?max-results=5&alt=rss&author=user>

### 22.8.3. Esquema de recursos servidos (funcionalidad mínima)

Recursos a servir como páginas HTML completas (pensadas para ser vistas en el navegador):

- `/`: Página principal de MetaMagazine, con texto de bienvenida y contenidos de una de las revistas (aleatoriamente, se elegirá una cada vez que se reciba una nueva visita, y se incluirán sus contenidos, que deberán ser iguales a los que se verían en la página de esa revista).
- `/channels`: Lista de canales activos, con enlace a los RSS correspondientes
- `/magazines`: Lista de revistas disponibles, con enlace a cada una de ellas.
- `/magazines/user`: Revista del usuario “user”
- `/news/id`: Página de la noticia “id” en MetaMagazine: título de la noticia y elementos a mostrar (enlaces de la noticia, primeras palabras de los sitios web en esos enlaces, imágenes en esos enlaces, etc.)

Recursos a servir con texto HTML listo para empotrar en otras páginas (estos, texto que pueda ir dentro de un elemento `< body >`):

- `/api/news/id`: Para la noticia “id”, elementos a mostrar (enlaces de la noticia, primeras palabras de los sitios web en esos enlaces, imágenes en esos enlaces, etc.)

Además de estos recursos, se atenderá a cualquier otro que sea necesario para proporcionar la funcionalidad indicada.

### 22.8.4. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Recurso `/conf`: Configuración del usuario, para usuarios registrados. Incluirá campos para editar su nombre público, su contraseña (dos veces, para comprobar).

- Recurso /conf/skin: Configuración del estilo (skin), para usuarios registrados. Mediante un formulario, el usuario podrá editar el fichero CSS que codificará su estilo, o podrá copiar el de otro usuario. Cada usuario tendrá un estilo (fichero CSS) por defecto, que el sistema le asignará si no lo ha configurado.
- Recurso /rss/user: Canal RSS para la revista del usuario “user”, con las 20 últimas entradas (del canal que sea).
- Uso de AJAX para otros aspectos de la aplicación. Por ejemplo, para indicar qué canales se quieren.
- Puntuación de revistas. Cada usuario registrado puede puntuar cualquier revista del sitio, por ejemplo entre 1 y 5. Estas puntuaciones se podrán ver luego junto a la revista en cuestión.
- Puntuación de noticias. Cada usuario registrado puede puntuar cualquier noticia del sitio, por ejemplo entre 1 y 5. Estas puntuaciones se podrán ver luego junto a la noticia en cuestión.
- Comentarios a noticias. Cada usuario registrado puede comentar cualquier noticia del sitio. Estos comentarios se podrán ver luego junto a la noticia en cuestión.
- Soporte para avatares. Cada canal se presentará junto con el avatar (el logo que ha elegido el usuario en el sitio original, como por ejemplo Twitter) del canal.
- Mejoras en la identificación de la información de las páginas web enlazadas. Por ejemplo, seleccionar las imágenes descartando las que probablemente son pequeños iconos (analizando el tamaño de la imagen), o identificando otros elementos relevantes.

#### **22.8.5. Notas y comentarios**

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura, con la versión de Django instalada en /usr/local/django (Django 1.3.1).

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos de los canales que alimentarán el planeta.

## 22.9. Práctica final (2012, mayo)

La práctica final a entregar en la convocatoria extraordinaria (mayo de 2012) será como la de la entrega ordinaria (práctica 22.8), con las diferencias que se indican en los siguientes apartados.

### 22.9.1. Arquitectura y funcionamiento general

Con respecto a las de la práctica de la convocatoria ordinaria, el enunciado tiene los siguientes cambios:

- En lugar de canales RSS se utilizarán canales Atom para descargar las noticias de los sitios terceros.
- Para construir una revista, en lugar de indicar qué información se quiere de cada sitio tercero, se indicarán cadenas de texto. Estas cadenas se utilizarán como hashtags en los sitios terceros que los soporten, o como cadenas de búsqueda en los que no. Por lo tanto, el usuario especificará una cadena, que se usará para definir qué canales Atom de los sitios terceros habrá que considerar (ver funcionalidad mínima más adelante).
- Al definir su revista, un usuario podrá por tanto especificar cadenas, igual que antes especificaba canales de un sitio tercero. Ahora, cada cadena indicará qué canales de todos los sitios terceros hay que considerar para esa revista.

El resto queda igual.

### 22.9.2. Funcionalidad mínima

Con respecto a la de la práctica de la convocatoria ordinaria, el enunciado tiene los siguientes cambios:

- Para cada cadena que un usuario especifique en su revista se bajará información de, como mínimo, los siguientes canales (los ejemplos serían para la cadena “urjc”):
  - Canal Atom correspondiente al hashtag de Twitter definido por esa cadena<sup>6</sup>.
  - Canal Atom correspondientes al hashtag de Identi.ca definido por esa cadena<sup>7</sup>.

---

<sup>6</sup>Para el hashtag “#urjc”:

<http://search.twitter.com/search.atom?q=%23urjc>

<sup>7</sup>Para el hashtag “#urjc”:

<http://identi.ca/api/statusnet/tags/timeline/urjc.atom>

- Canal Atom correspondientes a la búsqueda en Youtube de esa cadena<sup>8</sup>.

El resto queda igual.

## 22.10. Práctica final (2010, enero)

La práctica final de la asignatura consiste en la creación de un planeta, o agregador de canales, como aplicación web. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

### 22.10.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- La aplicación web se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin site” para mantener los usuarios con cuenta en el sistema, y para la gestión general de las bases de datos necesarias.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que toda la aplicación tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones también en la parte superior.
  - Un pie de página con una nota de copyright.
- Se utilizarán hojas de estilo CSS para determinar la apariencia de la aplicación.

Funcionamiento general:

---

<sup>8</sup>Para la búsqueda “urjc”:  
<http://gdata.youtube.com/feeds/api/videos?q=urjc&max-results=5&alt=atom>

- Los usuarios indicarán en qué canales (blogs) están interesados. Para ello, cada usuario podrá especificar un número en principio ilimitado de URLs, cada una correspondiente a un canal que le interesa.
- Cuando un usuario indica que le interesa un blog, se baja el canal correspondiente y se almacenan en la base de datos los artículos referenciados en él.
- Cuando un usuario acceda a la URL de actualización de sus blogs, se bajan los canales correspondientes a todos ellos, y se almacenan en la base de datos los artículos correspondientes. Si un artículo ya estaba en la base de datos, no debe almacenarse dos veces.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá los artículos en la base de datos e información pública para cada usuario.
- Sólo los navegadores con un usuario autenticado podrán personalizar en qué blogs están interesados.

#### **22.10.2. Funcionalidad mínima**

- Para cada artículo en la base de datos del planeta, se mostrarán (salvo que se indique lo contrario) su título (que será un enlace al artículo en su blog original), un enlace al blog original que lo incluye, y su contenido (tal y como venga especificado en el canal).
- El planeta mostrará en una interfaz pública (visible por cualquiera que no tenga cuenta en el sitio) todos los artículos que tenga en la base de datos, organizados en los siguientes recursos:
  - /: Lista de los últimos 20 artículos, por fecha de publicación, en orden inverso (más nuevos primero).
  - /blog: Lista de los últimos 20 artículos del blog “blog”, por fecha de publicación, en orden inverso (más nuevos primero).
  - /blog/num: Artículo número “num” del blog “blog”, siendo “0” el artículo más antiguo de ese blog que se tiene en la base de datos.
- Además, el planeta mostrará en una interfaz privada (visible sólo para un usuario concreto cuando se autentica como tal) los artículos que éste haya seleccionado, organizados en los siguientes recursos:

- /custom: Lista de los últimos 20 artículos, por fecha de publicación, en orden inverso (más nuevos primero), de los blogs seleccionados por el usuario.
- Además, habrá ciertos recursos donde los usuarios registrados podrán (una vez autenticados) configurar ciertos aspectos del sitio:
  - /conf: Configuración del usuario. Incluirá campos para editar su nombre público, su contraseña (dos veces, para comprobar), y los blogs en los que está interesado. Estos blogs se podrán elegir bien de un menú desplegable (en el que estarán los que ya se están bajando) o indicando sus datos (la URL de su canal correspondiente).
  - /conf/skin: Configuración del estilo (skin) con el que el usuario quiere ver el sitio. Mediante un formulario, el usuario podrá editar el fichero CSS que codificará su estilo, o podrá copiar el de otro usuario. Cada usuario tendrá un estilo (fichero CSS) por defecto, que el sistema le asignará si no lo ha configurado.
  - /update: Actualizará los artículos de los blogs en los que está interesado el usuario.
- Para cada usuario, se mantendrán ciertos recursos públicos con información relacionada con ellos:
  - /user: Nombre de usuario y lista de blogs que interesan al usuario “user”.
  - /user/feed: Canal RSS con los 20 últimos artículos que interesan al usuario “user”.
- El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador, y podrá ser especificado también en la URL /conf para los usuarios registrados (entre opciones para indicar un idioma particular, o “por defecto”, que respetará lo que indique el navegador).

### 22.10.3. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Uso de AJAX para algún aspecto de la práctica (por ejemplo, para elegir un nuevo blog, o para subir comentarios)
- Puntuación de artículos. Cada usuario registrado puede puntuar cualquier artículo del sitio, por ejemplo entre 1 y 5. Estas puntuaciones se podrán ver luego junto al artículo en cuestión.
- Comentarios a artículos. Cada usuario registrado puede comentar cualquier artículo del sitio. Estos comentarios se podrán ver luego junto al artículo en cuestión (en la página de ese artículo).
- Soporte para logos. Cada blog o artículo de un blog se presentará junto con un logo que represente al blog en cuestión.
- Autodescubrimiento de canales. Dada una URL (de un blog, por ejemplo), busca si en ella hay algún enlace que parece un canal. Si es así, ofrécelo al usuario para que lo pueda elegir. Esto se puede usar, por ejemplo, en la página de configuración de usuario, como una opción más para elegir canales (“especifica un blog para buscar sus canales”).

#### **22.10.4. Entrega de la práctica**

La práctica se entregará el día del examen de la asignatura, o un día posterior si así se acordase. La entrega se realizará presencialmente, en el laboratorio donde tienen lugar las clases de la asignatura habitualmente.

Cada alumno entregará su práctica en un fichero tar.gz, que tendrá preparado antes del comienzo del examen, y cuya localización mostrará al profesor durante el transcurso del mismo. El fichero se habrá de llamar practica-user.tar.gz, siendo “user” el nombre de la cuenta del alumno en el laboratorio.

El fichero que se entregue deberá constar de un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos con datos suficientes como para poder probarlo. Estos datos incluirán al menos tres usuarios, y cinco blogs con sus noticias correspondientes. Se incluirá también un fichero README con los siguientes datos:

- Nombre de la asignatura.
- Nombre completo del alumno.
- Nombre de su cuenta en el laboratorio.
- Nombres y contraseñas de los usuarios creados para la práctica.



- Resumen de las peculiaridades que se quieran mencionar sobre lo implementado en la parte obligatoria.
- Lista de funcionalidades opcionales que se hayan implementado, y breve descripción de cada una.

#### **22.10.5. Notas y comentarios**

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura, con la versión de Django instalada en `/usr/local/django` (Django 1.1.1).

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos de los canales que alimentarán el planeta.

Los canales (feeds) RSS que produce la aplicación web realizada en la práctica deberán funcionar al menos con el agregador Liferea.

#### **22.11. Práctica final (2010, junio)**

La práctica final para entrega en la convocatoria extraordinaria de junio será similar a la especificada para la convocatoria ordinaria de enero. En particular, deberá cumplir las siguientes condiciones:

- La arquitectura general será la misma, salvo:
  - En lugar de incluir en las plantillas Django un menú de opciones en la parte superior de las páginas, ese menú estará en una columna en la parte derecha de cada página.
- El funcionamiento general será el mismo, salvo:
  - Cuando un usuario indica que le interesa un blog, no se almacenan en la base de datos los artículos de ese blog.
  - No habrá URL de actualización de los blogs de un usuario.
  - Los artículos correspondientes a un blog se actualizarán sólo cuando se visualice una página del planeta que incluya artículos de ese blog. En ese momento, los artículos nuevos (los que no estaban ya en la base de datos) se bajarán a dicha base de datos.
- La funcionalidad mínima será la misma, salvo:

- No se implementará el recurso “/update”, dado que el funcionamiento de la actualización será diferente, como se ha indicado anteriormente.
  - El recurso “/user” incluirá la lista de los últimos 20 artículos, por fecha de publicación, en orden inverso (más nuevos primero), de los blogs seleccionados por el usuario, además del nombre de usuario.
  - Cada usuario registrado podrá puntuar cualquier artículo del sitio entre 1 y 5. Estas puntuaciones se podrán ver junto al artículo en cuestión, en todos los sitios donde aparece un enlace a él en el planeta.
- La funcionalidad optativa será la misma, salvo la puntuación de artículos, que ya ha sido mencionada como funcionalidad mínima.

El resto de condiciones serán iguales que en la convocatoria de enero de 2010.

## 22.12. Práctica final (2010, diciembre)

La entrega de esta práctica será necesaria para poder optar a aprobar la asignatura. Este enunciado corresponde con la convocatoria de diciembre.

La práctica final de la asignatura consiste en la creación de una aplicación web de resumen y cache de micronotas (microblogs). En este enunciado, llamaremos a esa aplicación “MiResumen”, y a los resúmenes de micronotas para cada usuario, “microresumen”.

Los sitios de microblogs permiten a sus usuarios compartir notas breves (habitualmente de 140 caracteres o menos). Entre los más populares pueden mencionarse Twitter<sup>9</sup> e Identi.ca<sup>10</sup>. La aplicación web a realizar se encargará de mostrar las micronotas que se indiquen, junto con información relacionada. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

### 22.12.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- La aplicación web se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.

---

<sup>9</sup><http://twitter.com>

<sup>10</sup><http://identi.ca>

- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- Se usará la aplicación Django “Admin site” para mantener los usuarios con cuenta en el sistema, y para la gestión general de las bases de datos necesarias.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que toda la aplicación tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones también en la parte superior, a la derecha del banner del sitio.
  - Un pie de página con una nota de copyright.
- Se utilizarán hojas de estilo CSS para determinar la apariencia de la aplicación.

Funcionamiento general:

- Se considerarán sólo micronotas en Identi.ca. Llamaremos a los usuarios de Identi.ca “micronoteros”.
- MiResumen mantendrá usuarios, que habrán de autenticarse para poder configurar la aplicación.
- Cada usuario de MiResumen indicará qué micronoteros de Identi.ca le interesan, configurando una lista de micronoteros.
- Cuando un usuario indica que le interesa un micronotero, MiResumen bajará el canal RSS correspondiente, y se almacenarán en la base de datos las micronotas referenciadas en él.
- Cuando un usuario acceda a la URL de actualización de su microresumen, se bajan los canales correspondientes a todos los micronoteros que tiene especificados, y se almacenan en la base de datos las micronotas correspondientes. Si una micronota ya estaba en la base de datos, no debe almacenarse dos veces.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá los microresúmenes de cada usuario.

### 22.12.2. Funcionalidad mínima

- Para cada micronota en la base de datos del planeta, se mostrarán (salvo que se indique lo contrario) su texto, un enlace a la micronota en Identi.ca, el nombre del micronotero que la puso (con un enlace a su página en Identi.ca), y la fecha en que la puso,
- MiResumen mostrará en una interfaz pública (visible por cualquiera que no tenga cuenta en el sitio) todas las micronotas que tenga en la base de datos, organizadas en los siguientes recursos:
  - /: Microresumen de las últimas 50 micronoticias, ordenadas por fecha inversa de publicación, en orden inverso (más nuevos primero).
  - /noteros/micronotero: Microresumen de las últimas 50 micronoticias del micronotero “micronotero”, ordenadas por fecha inversa de publicación, en orden inverso (más nuevos primero).
  - /usuarios/usuario: Microresumen de las últimas 50 micronoticias de los micronoteros que sigue el usuario “usuario”, ordenadas por fecha inversa de publicación.
  - /usuarios/usuario/feed: Canal RSS con las 50 últimas micronotas que interesan al usuario “usuario”.
- Además MiResumen proporcionará ciertos recursos donde los usuarios registrados podrán (una vez autenticados) configurar ciertos aspectos del sitio:
  - /conf: Configuración del usuario. Incluirá campos para editar su nombre público, su contraseña (dos veces, para comprobar), y el idioma que prefiere (al menos deberá poder elegir entre español e inglés).
  - /conf/skin: Configuración del estilo (skin) con el que el usuario quiere ver el sitio. Mediante un formulario, el usuario podrá editar el fichero CSS que codificará su estilo, o podrá copiar el de otro usuario. Cada usuario tendrá un estilo (fichero CSS) por defecto, que el sistema le asignará si no lo ha configurado.
  - /micronoteros: Lista de los micronoteros seleccionados por el usuario, junto con enlace a su página en Identi.ca. El usuario podrá eliminar un micronotero de la lista, o añadir uno nuevo mediante POST sobre ese recurso. Los micronoteros se podrán elegir bien de un menú desplegable (en el que estarán los que tiene seleccionados cualquier usuario de MiResumen) o indicando su nombre de micronotero en Identi.ca.

- /update: Actualizará las micronotas de los micronoteros en los que está interesado el usuario.
- El idioma de la interfaz de usuario del planeta será el especificado en la URL /conf para los usuarios registrados. Para los visitantes no registrados, será español.

Para la generación de canales RSS y para la internacionalización se podrán usar los mecanismos que proporciona Django, o no, según el alumno considere que le sea más conveniente.

### 22.12.3. Funcionalidad optativa

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Uso de Ajax para algún aspecto de la práctica (por ejemplo, para solicitar actualización de la lista de micronotas, o para suscribirse a un micronotero picando sobre una micronota suya).
- Promoción de micronotas. Cada usuario registrado puede promocionar (indicar que le gusta) cualquier micronota del sitio. Cada micronota se verá en el sitio junto con el número de promociones que ha recibido.
- Soporte para avatares. Cada micronota se presentará junto con el avatar (imagen) correspondiente al micronotero que la ha puesto.
- Soporte para Twitter y/o otros sitios de microblogging (micronotas) además de Identi.ca
- Enlace a URLs. Se identificarán en las micronotas los textos que tengan formato de URL, y se mostrará esa URL como enlace.
- Enlace a micronoteros referenciados. Se identificarán en las micronotas los textos que tengan formato de identificador de micronotero (@nombre), y se mostrarán como enlace a la página del micronotero en cuestión.
- Suscripción a los mismos micronoteros a los que esté suscrito otro usuario. Un usuario podrá indicar que quiere suscribirse a la misma lista de micronoteros que otro, indicando sólo su identificador de usuario.

#### 22.12.4. Entrega de la práctica

La práctica se entregará electrónicamente en una de las dos fechas indicadas:

- El día anterior al examen de la asignatura, esto es, el 12 de diciembre, a las 18:00.
- El 30 de diciembre a las 23:00.

Además, los alumnos que hayan presentado las prácticas podrán tener que realizar una entrega presencial en una de las dos fechas indicadas:

- El día del examen, esto es, el 14 de diciembre, al terminar el examen de teoría. La lista de alumnos que tengan que hacer la entrega presencial se indicará durante el examen de teoría.
- El día 10 de enero, a las 16:00. La lista de alumnos que tengan que hacer la entrega presencial se indicará con anterioridad en el sitio web de la asignatura.

La entrega presencial se realizará en el laboratorio donde tienen lugar habitualmente las clases de la asignatura.

Cada alumno entregará su práctica colocándola en un directorio en su cuenta en el laboratorio. El directorio, que deberá colgar directamente de su directorio hogar (\$HOME), se llamará “pf\_django.2010”.

El directorio que se entregue deberá constar de un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos con datos suficientes como para poder probarlo. Estos datos incluirán al menos tres usuarios, y cinco micronoteros con sus micronotas correspondientes. Entre los usuarios, habrá en la base de datos al menos los dos siguientes.

- Usuario “pepe”, contraseña “XXX”
- Usuario “pepa”, contraseña “XXX”

Cada uno de estos usuarios estará ya siguiendo al menos dos micronoteros.

Se incluirá también en el directorio que se entregue un fichero README con los siguientes datos:

- Nombre de la asignatura.
- Nombre completo del alumno.
- Nombre de su cuenta en el laboratorio.

- Nombres y contraseñas de los usuarios creados para la práctica.
- Nombres de al menos cinco micronoteros cuyas noticias estén en la base de datos de la aplicación.
- Resumen de las peculiaridades que se quieran mencionar sobre lo implementado en la parte obligatoria.
- Lista de funcionalidades opcionales que se hayan implementado, y breve descripción de cada una.

Es importante que estas normas se cumplan estrictamente, y de forma especial lo que se refiere al nombre del directorio, porque la recogida de las prácticas, y parcialmente su prueba, se hará con herramientas automáticas.

[Las normas de entrega podrán incluir más detalles en el futuro, compruébalas antes de realizar la entrega.]

#### **22.12.5. Notas y comentarios**

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura, con la versión 1.2.3 de Django.

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos de los canales que alimentarán MiResumen.

Los canales (feeds) RSS que produce la aplicación web realizada en la práctica deberán funcionar al menos con el agregador Liferea y el que lleva integrado Firefox.

#### **22.12.6. Notas de ayuda**

A continuación, algunas notas que podrían ayudar a la realización de la práctica. Gracias a los alumnos que han contribuido a ellas, bien preguntando sobre algún problema que han encontrado, o incluso aportando directamente una solución correcta.

- **Conversión de fechas:**

La conversión de fechas, tal y como vienen en el formato de los canales RSS de Identi.ca, al formato de fechas datetime adecuado para almacenarlas en una tabla de la base de datos se puede hacer así:

```

from email.utils import parsedate
from datetime import datetime

dbDate = datetime(*(parsedate(rssDate)[:6]))

```

El uso de “\*” permite, en este caso, obtener una referencia a la tupla de siete elementos que contiene los parámetros que espera datetime() (que son siete parámetros).

Más información sobre parsedate() en la documentación del módulo email.utils de Python.

#### ■ Envío de hojas CSS:

Para que el navegador interprete adecuadamente una hoja de estilo, puede ser conveniente fijar el tipo de contenidos de la respuesta HTTP en la que la aplicación la envía al navegador. En otras palabras, asegurar que cuando el navegador reciba la hoja CSS, le venga adecuadamente marcada como de tipo “text/css” (y no “text/html” o similar, que es como vendrá marcado normalmente lo que responda la aplicación).

En código, bastaría con poner la cabecera “Content-Type” adecuada al objeto que tiene la respuesta HTTP que devolverá la función que atiende a la URL para servir la hoja CSS (normalmente en `views.py`):

```

myResponse = HttpResponse(cssText)
myResponse['Content-Type'] = 'text/css'
return myResponse

```

## 22.13. Práctica final (2011, junio)

La entrega de esta práctica será necesaria para poder optar a aprobar la asignatura. Este enunciado corresponde con la convocatoria de junio.

La práctica final de la asignatura consiste en la creación de una aplicación web de resumen y cache de micronotas (microblogs). En este enunciado, llamaremos a esa aplicación “MiResumen2”, y a los resúmenes de micronotas para cada usuario, “microresumen”.

Los sitios de microblogs permiten a sus usuarios compartir notas breves (habitualmente de 140 caracteres o menos). Entre los más populares pueden mencionarse Twitter<sup>11</sup> e Identi.ca<sup>12</sup>. La aplicación web a realizar se encargará de mostrar las

---

<sup>11</sup><http://twitter.com>

<sup>12</sup><http://identi.ca>



micronotas que se indiquen, junto con información relacionada. A continuación se describe el funcionamiento y arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

### **22.13.1. Arquitectura y funcionamiento general**

Arquitectura general:

- La práctica consistirá en una aplicación web que servirá los datos a los usuarios.
- La aplicación web se construirá como un proyecto Django, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas según modelos en Django.
- No se mantendrán usuarios con cuenta, ni usando la aplicación Django “Admin site” ni de otra manera. Por lo tanto, para usar el sitio no hará falta registrarse, ni entrar en una cuenta.
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que toda la aplicación tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios. Estas plantillas incluirán en todas las páginas al menos:
  - Un banner (imagen) del sitio, en la parte superior.
  - Un menú de opciones justo debajo del banner, formateado en una línea.
  - Un pie de página con una nota de copyright.
- Se utilizarán hojas de estilo CSS para determinar la apariencia de la aplicación. Estas hojas se almacenarán en la base de datos.

Funcionamiento general:

- Se considerarán sólo micronotas en Identi.ca. Llamaremos a los usuarios de Identi.ca “micronoteros”.
- MiResumen2 recordará a todos sus visitantes. A estos efectos, consideraremos como sesión de un visitante todas las interacciones que se hagan con el sitio desde el mismo navegador (por lo tanto, se podrán usar cookies de sesión para mantener esta relación).

- MiResumen2 mostrará notas de Identi.ca, que se irán actualizando según se indica en el apartado siguiente.
- Los visitantes de MiResumen2 podrán seleccionar cualquier micronota que aparezca en él.
- Cada visitante podrá ver las micronotas que ha seleccionado, por orden inverso de publicación en Identi.ca, en un listado que incluirá también la fecha en que seleccionó cada micronota.

### 22.13.2. Funcionalidad mínima

- Para cada micronota en la base de datos del planeta, se mostrarán (salvo que se indique lo contrario):
  - el texto de la micronota
  - un enlace a la micronota en Identi.ca
  - el nombre del micronotero que la puso (con un enlace a su página en Identi.ca)
  - la fecha en que se publicó en Identi.ca
  - un botón para que cualquier visitante pueda seleccionar esta nota (o deseccionarla si ya la había seleccionado)
  - si el usuario ha seleccionado la micronota, la fecha en que la había seleccionado
  - un número que representará el número de visitantes que han seleccionado esta micronota
- MiResumen2 mostrará en una interfaz pública (visible por cualquiera que visite el sitio) todas las micronotas que tenga en la base de datos, organizadas en los siguientes recursos:
  - /: Microresumen de las últimas 30 micronoticias almacenadas en MiResumen2, ordenadas por fecha inversa de publicación (más nuevos primero). Además, incluirá un enlace al recurso de actualización (ver más abajo), y al microresumen de las 30 siguientes micronoticias (/30, ver más abajo)
  - /nnn: Microresumen de las micronoticas entre la nnn y la nnn+29, según el orden de fecha inversa de publicación (más nuevos primero, con números más bajos). Se considerará que la micronota más reciente es la micronota 0. Así, /0 mostrará lo mismo que / , /30 mostrará

las 30 micronotas siguientes a las mostradas en / y /40 mostrará las micronotas de la 40 a la 67.

- /update: Recurso de actualización: cuando se acceda a él, MiResumen2 accederá al RSS de la página principal de Identi.ca y extraerá de él las últimas 20 micronotas (o menos, si no hay tantas micronotas en el canal que no estén ya en la base de datos), almacenándolas en la base de datos y mostrándolas.
- /selected: Listado de todas las micronotas seleccionadas por el visitante actual, ordenadas por fecha de publicación inversa (más nuevas primero).
- /feed: Canal RSS con las 10 micronotas más recientes (por fecha de publicación) que ha seleccionado el visitante actual.
- /conf: Configuración del visitante. Incluirá campos para editar el nombre del visitante, que se mostrará en todas las páginas del sitio que se sirvan a ese visitante.
- /skin: Configuración del estilo (skin) con el que el visitante quiere ver el sitio. Mediante un formulario, el visitante podrá editar el fichero CSS que codificará su estilo (y que se almacenará en la base de datos). Si no lo han cambiado, los visitantes tendrán el estilo CSS por defecto del sitio.
- /cookies: Página HTML que incluirá un listado de las cookies que se están usando con cada uno de los visitantes conocidos para la aplicación, en formato listo para que cada cookie pueda ser copiada y pegada en un editor de cookies.

Para la generación de canales RSS y la gestión de sesiones y/o cookies se podrán usar los mecanismos que proporciona Django, o no, según el alumno considere que le sea más conveniente.

### **22.13.3. Funcionalidad optativa**

De forma optativa, se podrá incluir cualquier funcionalidad relevante en el contexto de la asignatura. Se valorarán especialmente las funcionalidades que impliquen el uso de técnicas nuevas, o de aspectos de Django no utilizados en los ejercicios previos, y que tengan sentido en el contexto de esta práctica y de la asignatura.

Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Uso de Ajax para algún aspecto de la práctica (por ejemplo, para seleccionar y deseleccionar una micronota).
- Votación de micronotas. Cada visitante podrá dar una puntuación entre 0 y 10 a cada micronota. Cuando se muestre cada micronota en el sitio, además de los demás datos que se han mencionado, se incluirá la media de las votaciones que ha tenido, y el número de votaciones que ha tenido esa micronota. Una vez que un visitante ha votado una micronota, no puede volver a votarla, ni cambiar su votación.
- Soporte para avatares. Cada micronota se presentará junto con el avatar (imagen) correspondiente al micronotero que la ha puesto.
- Soporte para Twitter y/o otros sitios de microblogging (micronotas) además de Identi.ca
- Enlace a URLs, etiquetas y micronoteros referenciados. Se identificarán en las micronotas los textos que tengan formato de URL, y se mostrará esa URL como enlace, los que tengan formato de etiqueta (tag, nombres que comienzan por #), mostrándolos como enlace a la página Identi.ca para ese tag, y los micronoteros referenciados (nombres que comienzan por @), mostrándolos como enlace a la página del micronotero en cuestión en Identi.ca.
- Recomendación de micronotas. En una página, se mostrarán las micronotas que probablemente interesen al micronotero, basada en la historia de elecciones pasadas. El algoritmo a usarse puede ser: busca los tres visitantes que más notas hayan elegido en común con las del visitante actual, y muestra todas las micronotas que hayan elegido esos visitantes y el visitante actual aún no ha elegido.

#### **22.13.4. Entrega de la práctica**

La práctica se entregará electrónicamente como muy tarde el día 17 de junio a las 23:00.

Además, los alumnos que hayan presentado las prácticas podrán tener que realizar una entrega presencial el día que esté fijado el examen de teoría de la asignatura. La entrega presencial se realizará en el laboratorio donde tienen lugar habitualmente las clases de la asignatura.

Cada alumno entregará su práctica colocándola en un directorio en su cuenta en el laboratorio. El directorio, que deberá colgar directamente de su directorio hogar (\$HOME), se llamará “pf\_django.2010\_2”.

El directorio que se entregue deberá constar de un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos con

datos suficientes como para poder probarlo. Estos datos incluirán al menos cinco visitantes diferentes, cada uno con al menos 3 micronotas elegidas, y un total de al menos 50 micronotas en la base de datos de MiResumen2

Se incluirá también en el directorio que se entregue un fichero README con los siguientes datos:

- Nombre de la asignatura.
- Nombre completo del alumno.
- Nombre de su cuenta en el laboratorio.
- Resumen de las peculiaridades que se quieran mencionar sobre lo implementado en la parte obligatoria.
- Lista de funcionalidades opcionales que se hayan implementado, y breve descripción de cada una.

Es importante que estas normas se cumplan estrictamente, y de forma especial las que se refieren al nombre del directorio, porque la recogida de las prácticas, y parcialmente su prueba, se hará con herramientas automáticas.

[Las normas de entrega podrán incluir más detalles en el futuro, compruébalas antes de realizar la entrega.]

### **22.13.5. Notas y comentarios**

La práctica deberá funcionar en el entorno GNU/Linux (Ubuntu) del laboratorio de la asignatura, con la versión 1.2.3 de Django.

La práctica deberá funcionar desde el navegador Firefox disponible en el laboratorio de la asignatura.

Se recomienda construir una o varias aplicaciones complementarias para probar la descarga y almacenamiento en base de datos del canal que alimentarán MiResumen.

Los canales (feeds) RSS que produce la aplicación web realizada en la práctica deberán funcionar al menos con el agregador Liferea y el que lleva integrado Firefox.

Se recomienda utilizar alguna extensión para Firefox que permita manipular cookies para poder probar la aplicación simulando varios visitantes desde el mismo navegador.

### 22.13.6. Notas de ayuda

A continuación, algunas notas que podrían ayudar a la realización de la práctica. Gracias a los alumnos que han contribuido a ellas, bien preguntando sobre algún problema que han encontrado, o incluso aportando directamente una solución correcta.

- **Conversión de fechas:**

La conversión de fechas, tal y como vienen en el formato de los canales RSS de Identi.ca, al formato de fechas `datetime` adecuado para almacenarlas en una tabla de la base de datos se puede hacer así:

```
from email.utils import parsedate
from datetime import datetime

dbDate = datetime(*(parsedate(rssDate)[:6]))
```

El uso de “\*” permite, en este caso, obtener una referencia a la tupla de siete elementos que contiene los parámetros que espera `datetime()` (que son siete parámetros).

Más información sobre `parsedate()` en la documentación del módulo `email.utils` de Python.

- **Envío de hojas CSS:**

Para que el navegador interprete adecuadamente una hoja de estilo, puede ser conveniente fijar el tipo de contenidos de la respuesta HTTP en la que la aplicación la envía al navegador. En otras palabras, asegurar que cuando el navegador reciba la hoja CSS, le venga adecuadamente marcada como de tipo “text/css” (y no “text/html” o similar, que es como vendrá marcado normalmente lo que responda la aplicación).

En código, bastaría con poner la cabecera “Content-Type” adecuada al objeto que tiene la respuesta HTTP que devolverá la función que atiende a la URL para servir la hoja CSS (normalmente en `views.py`):

```
myResponse = HttpResponse(cssText)
myResponse['Content-Type'] = 'text/css'
return myResponse
```

## 23. Materiales de interés

### 23.1. Material complementario general

- Philip Greenspun, *Software Engineering for Internet Applications*:  
<http://philip.greenspun.com/seia/>  
utilizado en un curso del MIT  
<http://philip.greenspun.com/teaching/one-term-web>

### 23.2. Introducción a Python

- <http://www.python.org/doc>  
Documentación en línea de Python (incluyendo un Tutorial, los manuales de referencia, HOWTOS, etc. Usa la versión para Python 2.x)
- <http://www.diveintopython.org/>  
“Dive into Python”, por Mark Pilgrim. Libro para aprender Python, orientado a quien ya sabe programa con lenguajes orientados a objetos.
- <http://wiki.python.org/moin/BeginnersGuide/Programmers>  
Otros textos sobre Python, de interés especialmente para quien ya sabe programar en otros lenguajes.
- [http://en.wikibooks.org/wiki/Python\\_Programming](http://en.wikibooks.org/wiki/Python_Programming)  
“Python Programming”, Wikibook sobre programación en Python.
- [http://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Python_(programming_language))  
Python en la Wikipedia
- <http://www.python.org/dev/peps/pep-0008/>  
Style Guide for Python Code (PEP 8). Esta es la guía de estilo que se puede comprobar con el programa pep8.

### 23.3. Aplicaciones web mínimas

- <http://docs.python.org/dev/howto/sockets.html>  
“Socket Programming HOWTO”. Programación de sockets en Python, guía rápida.
- <http://docs.python.org/library/socket.html>  
Documentación de la biblioteca de sockets de Python.

- <https://addons.mozilla.org/en-US/firefox/>  
Lista de add-ons y plugins para Firefox.

## **23.4. SQL y SQLite**

- <http://www.shokhirev.com/nikolai/abc/sql/sql.html>  
“SQLite / SQL Tutorials: Basic SQL”, por Nikolai Shokhirev