Servicios Telemáticos
Grado en Ingeniería Telemática
Guía de Estudio, curso 2023/2024

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

7 de febrero de 2024

Índice

1. Datos generales **12**

2. Objetivos **13**

3. Metodología **13**

4. Evaluación **14**
4.1. Criterios de evaluación 14

5. Calendario **16**

6. Programa de teoría **17**
6.1. 00 - Presentación 17
6.1.1. 25 de enero: Presentación (2 horas) 17
6.2. 01 - Conceptos básicos de aplicaciones web 17
6.2.1. 5 de febrero: Conceptos básicos I (2 horas) 17
6.2.2. 12 de febrero: Conceptos básicos II (2 horas) 18
6.2.3. 19 de febrero: Conceptos básicos III (2 horas) 18
6.2.4. 26 de febrero: Conceptos básicos IV (2 horas) 19
6.2.5. 4 de marzo: Conceptos básicos V (2 horas) 19
6.3. 02 - Servicios web que interoperan 20
6.3.1. 11 de marzo: Interoperación web I (2 horas) 20

6.3.2. 18 de marzo: Interoperación web II (2 horas) 20
6.4. 03 - Modelo-vista-controlador 20
6.4.1. 8 de abril: MVC (2 horas) 20
6.4.2. 15 de abril: MVC II (2 horas) 21
6.5. 04 - Introducción a XML y JSON 21
6.5.1. 22 de abril: XML, JSON I (2 horas) 21
6.5.2. 29 de abril: XML, JSON II (2 horas) 22
6.6. 05 - Hojas de estilo CSS 23
6.6.1. 6 de mayo: CSS (2 horas) 23

7. Programa de prácticas **24**
7.1. P1 - Introducción a Python 24
7.1.1. 25 de enero: Python y entorno de desarrollo I 24
7.1.2. 1 de febrero: Python y entorno de desarrollo II 24
7.1.3. 1 de febrero: Python II (1 hora) 24
7.1.4. 8 de febrero: Python III (2 horas) 25
7.2. P2 - Aplicaciones web simples 25
7.2.1. 15 de febrero: Aplicaciones web (2 horas) 25
7.3. P3 - Servidores simples de contenidos 25
7.3.1. 22 de febrero: Servidores con clase (2 horas) 26
7.3.2. 29 de febrero: Servidores con clase II (2 horas) 26
7.4. P4 - Introducción a Django 26
7.4.1. 7 de marzo: Django I (2 horas) 26
7.4.2. 14 de marzo: Django II (2 horas) 27
7.4.3. 21 de marzo: Django III (2 horas) 27
7.4.4. 4 de abril: Django IV (2 horas) 28
7.4.5. 11 de abril: Django V (2 horas) 28
7.4.6. 18 de abril: Django VI (2 horas) 28
7.4.7. 25 de abril: Django VII (2 horas) 28
7.4.8. 9 de mayo: Detalles finales I (2 horas) 29

8. Proyecto final: DeCharla (2023) **30**
8.1. Requisitos generales 30
8.2. Despliegue 35
8.3. Funcionalidad optativa 36
8.4. Entrega de la práctica 37
8.5. Notas y comentarios 40
8.6. Preguntas frecuentes 40

9. Prácticas de entrega voluntaria **43**
9.1. Entrega de microprácticas y miniprácticas . . . 43
9.2. Minipráctica 1 (entrega voluntaria) . . . 44
9.3. Minipráctica 2 (entrega voluntaria) . . . 46

10. Ejercicios 01: Conceptos básicos de aplicaciones web **48**
10.1. Web 2.0 . . . 48
10.2. Última búsqueda . . . 48
10.3. Espía a tu navegador (Firefox Developer Tools) . . . 49
10.4. Espía a tu navegador (Firebug) . . . 49
10.5. Explora tus cookies . . . 50
10.6. Explora tus cookies (2) . . . 50
10.7. Servidores que recuerdan . . . 51
10.8. Servicio horario . . . 52
10.9. Última búsqueda: números aleatorios o consecutivos . . . 52
10.10 Cookies en tu navegador . . . 52
10.11 Cookies en tu navegador avanzado . . . 53
10.12 Sumador simple con varios navegadores . . . 53
10.13 Sumador simple con varios navegadores intercalados . . . 53
10.14 Sumador simple con rearranques . . . 53
10.15 Contador simple . . . 54
10.16 cURL básico . . . 54
10.17 Distinto contenido según navegador . . . 55
10.18 Depurador básico . . . 55
10.19 Contador simple con varios navegadores . . . 55
10.20 Contador simple con varios navegadores intercalados . . . 55
10.21 Contador simple con rearranques . . . 56
10.22 Traza de historiales de navegación por terceras partes . . . 56
10.23 Trackers en páginas web . . . 57
10.24 Trackers en páginas web (Ghostery) . . . 58
10.25 Protección contra trackers en el navegador . . . 58
10.26 Transplante de cookies . . . 58
10.27 Transplante de cookies 2 . . . 58

11. Ejercicios 02: Servicios web que interoperan **59**
11.1. Arquitectura escalable . . . 59
11.2. Arquitectura distribuida . . . 59
11.3. Lista de la compra . . . 61
11.4. Listado de lo que tengo en la nevera . . . 61
11.5. Sumador simple versión REST . . . 64
11.6. Calculadora simple versión REST . . . 64

11.7. Calculadora simple versión REST (Django) . . . 66
11.8. Cache de contenidos . . . 66
11.9. Cache de contenidos versión Django . . . 66
11.10 Cache de contenidos anotado . . . 66
11.11 Gestor de contenidos miltilingüe versión REST . . . 69
11.12 Sistema de transferencias bancarias . . . 69
11.13 Gestor de contenidos multilingüe preferencias del navegador . . . 70
11.14 Gestor de contenidos multilingüe con elección en la aplicación . . . 70
11.15 Sistema REST para calcular Pi . . . 71

12. Ejercicios 03: Modelo-Vista-Controlador **72**
12.1. Red social muy simple . . . 72

13. Ejercicios 04: Introducción a XML **72**
13.1. Chistes XML . . . 73
13.2. Chistes XML (parser DOM) . . . 73
13.3. Modificación del contenido de una página HTML . . . 73
13.4. Titulares de BarraPunto . . . 74
13.5. Videos en canal de YouTube . . . 74
13.6. Videos en canal de YouTube (con descarga) . . . 75
13.7. Gestor de contenidos con titulares de BarraPunto . . . 76
13.8. Gestor de contenidos con titulares de BarraPunto versión SQL . . . 76
13.9. Gestor de contenidos con titulares de BarraPunto versión Django . . . 76
13.10 Gestor de contenidos con videos de YouTube (simple) . . . 77
13.11 Gestor de contenidos con videos de YouTube (2) . . . 79
13.12 Gestor de contenidos con videos de YouTube (tests) . . . 79
13.13 Gestor de contenidos con videos de YouTube (despliegue) . . . 80
13.14 Municipios JSON . . . 80
13.15 Municipios JSON via HTTP . . . 81
13.16 Forks de un repositorio GitLab . . . 81
13.17 Extractor de información de un documento HTML . . . 82

14. Ejercicios 05: Hojas de estilo CSS **83**
14.1. Django cms_css simple . . . 83
14.2. Django cms_css elaborado . . . 83
14.3. Django cms_bootstrap cuadrícula . . . 84
14.4. Django cms_bootstrap componentes . . . 84
14.5. Django cms_bootstrap componentes personalizados . . . 85

15.Ejercicios 06: AJAX **85**
15.1. SPA Sentences generator . . . 85
15.2. Ajax Sentences generator . . . 85
15.3. Gadget de Google . . . 86
15.4. Gadget de Google en Django cms . . . 86
15.5. EzWeb . . . 86
15.6. EyeOS . . . 87

16.Ejercicios P1: Introducción a Python **87**
16.1. Uso interactivo del intérprete de Python . . . 87
16.2. Subida de programa a repo de GitLab . . . 88
16.3. Asistente de IA generativa para PyCharm . . . 88
16.4. Haz un programa en Python . . . 88
16.5. Tablas de multiplicar . . . 89
16.6. Ficheros y listas . . . 89
16.7. Ficheros, diccionarios y excepciones . . . 89
16.8. Calculadora . . . 90
16.9. Descarga de documentos web . . . 90
16.10Descarga de documentos web con módulos . . . 92

17.Ejercicios P2: Aplicaciones web simples **93**
17.1. Aplicación web hola mundo . . . 93
17.2. Variaciones de la aplicación web hola mundo . . . 94
17.3. Aplicación web generadora de URLs aleatorias . . . 94
17.4. Aplicación redirectora . . . 95
17.5. Sumador simple . . . 95
17.6. Clase servidor de aplicaciones . . . 96
17.7. Clase servidor de aplicaciones, generador de URLs aleatorias . . . 96
17.8. Clase servidor de aplicaciones, sumador . . . 97
17.9. Clase servidor de varias aplicaciones . . . 97
17.10Clase servidor, cuatro aplis . . . 97
17.11Herramientas de Web Developer . . . 98

18.Ejercicios P3: Introducción a Django **98**
18.1. Instalación de Django . . . 98
18.2. Introducción a Django . . . 98
18.3. Django primera aplicación . . . 99
18.4. Django calc . . . 99
18.5. Django cms . . . 99
18.6. Django cms_put . . . 99
18.7. Django cms_put-post . . . 100

18.8. Django cms_users . . . 101
18.9. Django cms_users_put . . . 102
18.10Django cms_templates . . . 102
18.11Django cms_post . . . 102
18.12Django cms_forms . . . 103
18.13Django tests . . . 103
18.14Django tests en GitLab . . . 104
18.15Django feed_expander . . . 104
18.16Django feed_expander_db . . . 105
18.17Django Conciertos . . . 106

19.Ejercicios P4: Servidores simples de contenidos 106
19.1. Clase contentApp . . . 107
19.2. Instalación y prueba de Poster . . . 107
19.3. Clase contentPutApp . . . 107
19.4. Clase contentPostApp . . . 107
19.5. Clase contentPersistentApp . . . 108
19.6. Clase contentStorageApp . . . 108
19.7. Gestor de contenidos con usuarios . . . 108
19.8. Gestor de contenidos con usuarios, con control estricto de actualización . . . 109

20.Ejercicios P5: Aplicaciones web con base de datos 109
20.1. Introducción a SQLite3 con Python . . . 109
20.2. Gestor de contenidos con base de datos . . . 110
20.3. Gestor de contenidos con usuarios, con control estricto de actualización y base de datos . . . 110

21.Ejercicios complementarios de varios temas 111
21.1. Números primos . . . 111
21.2. Autenticación . . . 111
21.3. Recomendaciones . . . 112
21.4. Geolocalización . . . 113

22.Prácticas de entrega voluntaria de cursos pasados 116
22.1. Prácticas de entrega voluntaria (curso 2021-2022) . . . 116
22.2. Minipráctica 2 (entrega voluntaria) . . . 116
22.3. Prácticas de entrega voluntaria (curso 2020-2021) . . . 116
22.4. Minipráctica 1 (entrega voluntaria) . . . 116
22.5. Prácticas de entrega voluntaria (curso 2014-2015) . . . 118
22.5.1. Práctica 1 (entrega voluntaria) . . . 118

22.5.2. Práctica 2 (entrega voluntaria) . . . 120
22.6. Prácticas de entrega voluntaria (curso 2012-2013) . . . 121
22.6.1. Práctica 1 (entrega voluntaria) . . . 121
22.6.2. Práctica 2 (entrega voluntaria) . . . 122
22.7. Prácticas de entrega voluntaria (curso 2011-2012) . . . 123
22.7.1. Práctica 1 (entrega voluntaria) . . . 123
22.7.2. Práctica 2 (entrega voluntaria) . . . 124
22.8. Prácticas de entrega voluntaria (curso 2010-2011) . . . 125
22.8.1. Práctica 1 (entrega voluntaria) . . . 125
22.8.2. Práctica 2 (entrega voluntaria) . . . 126
22.8.3. Práctica 3 (entrega voluntaria) . . . 127
22.8.4. Práctica 4 (entrega voluntaria) . . . 128

23. Pruebas escritas pasadas **129**
23.1. Examen de ITT-SAT, 28 de junio de 2023 . . . 129
23.1.1. Enunciado . . . 129
23.1.2. Solución y rúbrica calificación . . . 132
23.2. Examen de ITT-SAT, 15 de mayo de 2023 . . . 141
23.2.1. Enunciado . . . 141
23.2.2. Solución y rúbrica calificación . . . 145
23.3. Examen de ITT-SAT, 23 de junio de 2022 . . . 151
23.4. Examen de ITT-SAT, 20 de mayo de 2022 . . . 154
23.4.1. Soluciones . . . 157
23.5. Examen de IST-SARO, 28 de junio de 2022 . . . 164
23.6. Examen de IST-SARO, 24 de mayo de 2022 . . . 167
23.7. Examen de IT-ST, 1 de julio de 2022 . . . 170
23.7.1. Ejercicio 1 (solución) . . . 173
23.7.2. Ejercicio 2 (solución) . . . 174
23.7.3. Ejercicio 3 (solución) . . . 175
23.7.4. Ejercicio 4 (solución) . . . 177
23.7.5. Ejercicio 5 (solución) . . . 177
23.8. Examen de IT-ST, 27 de mayo de 2022 . . . 178
23.8.1. Ejercicio 1 (solución) . . . 181
23.8.2. Ejercicio 2 (solución) . . . 182
23.8.3. Ejercicio 3 (solución) . . . 182
23.8.4. Ejercicio 4 (solución) . . . 184
23.8.5. Ejercicio 5 (solución) . . . 186
23.9. Examen de ITT-SARO, 6 de junio de 2021 . . . 186
23.9.1. Ejercicio 1 (solución) . . . 189
23.9.2. Ejercicio 2 (solución) . . . 191
23.9.3. Ejercicio 3 (solución) . . . 192

23.9.4. Ejercicio 4 (solución) . . . 192
23.9.5. Ejercicio 5 (solución) . . . 195
23.10 Examen de IST-SAT, 1 de junio de 2021 . . . 196
23.10.1 Ejercicio 1 (solución) . . . 198
23.10.2 Ejercicio 2 (solución) . . . 199
23.10.3 Ejercicio 3 (solución) . . . 201
23.10.4 Ejercicio 4 (solución) . . . 201
23.10.5 Ejercicio 5 (solución) . . . 204
23.11 Examen de ITT-SARO e IST-SAT, 6 de junio de 2020 . . . 204
23.11.1 Solución . . . 209
23.11.2 Rúbrica calificación . . . 212
23.12 Examen de ITT-SARO, 14 de mayo de 2019 . . . 214
23.12.1 Soluciones . . . 217
23.13 Examen de ITT-SAT, 15 de mayo de 2019 . . . 222
23.13.1 Soluciones . . . 224
23.14 Examen de ITT-SAT, 7 mayo de 2018 . . . 228
23.15 Examen de ITT-SARO, 17 mayo de 2018 . . . 237
23.16 Examen de ITT-SAT, 10 mayo de 2017 . . . 244
23.17 Examen de IST-SARO, 10 mayo de 2017 . . . 252

24. Proyecto final: MisPalabras (2022) 262
24.1. Arquitectura y funcionamiento general . . . 262
24.2. Información automática . . . 267
24.2.1. Wikipedia en español: definición resumida . . . 267
24.2.2. Wikipedia en español: imagen principal . . . 267
24.2.3. DRAE: definición . . . 268
24.2.4. Flickr: imagen de etiqueta . . . 268
24.2.5. Apimeme: imagen de meme . . . 269
24.2.6. Otra información automática . . . 269
24.3. Documentos con tarjeta embebida . . . 270
24.4. Despliegue . . . 271
24.5. Funcionalidad optativa . . . 272
24.6. Entrega de la práctica . . . 273
24.7. Notas y comentarios . . . 276
24.8. Preguntas frecuentes . . . 277

25. Proyecto final: LoVisto (2021) 281
25.1. Arquitectura y funcionamiento general . . . 281
25.2. Recursos reconocidos . . . 283
25.2.1. Predicción AEMET para un municipio . . . 283
25.2.2. Página Wikipedia en español . . . 284

25.2.3. Vídeo de YouTube . . . 285
25.2.4. Nota en Reddit . . . 286
25.2.5. Otros recursos reconocidos . . . 287
25.3. Recursos no reconocidos . . . 288
25.4. Funcionalidad mínima . . . 289
25.5. Despliegue . . . 292
25.6. Funcionalidad optativa . . . 294
25.7. Entrega de la práctica . . . 295
25.8. Notas y comentarios . . . 298
25.9. Preguntas frecuentes . . . 298

26. Proyecto final: MisCosas (2020, mayo) 302
26.1. Arquitectura y funcionamiento general . . . 302
26.2. Alimentadores . . . 304
26.3. Funcionalidad mínima . . . 310
26.4. Despliegue . . . 313
26.5. Funcionalidad optativa . . . 315
26.6. Entrega de la práctica . . . 316
26.7. Notas y comentarios . . . 319
26.8. Preguntas frecuentes . . . 319

27. Proyecto final: MiTiempo (2019, mayo) 326
27.1. Arquitectura y funcionamiento general . . . 326
27.2. Funcionalidad mínima . . . 328
27.3. Funcionalidad optativa . . . 331
27.4. Entrega de la práctica . . . 332
27.5. Notas y comentarios . . . 334
27.6. Preguntas y respuestas . . . 335

28. Proyecto final (2019, junio) 338

29. Proyecto final (2018, mayo) 339
29.1. Arquitectura y funcionamiento general . . . 339
29.2. Funcionalidad mínima . . . 341
29.3. Funcionalidad optativa . . . 343
29.4. Entrega de la práctica . . . 344
29.5. Notas y comentarios . . . 346

30. Proyecto final (2018, junio) 346

31.Proyecto final (2017, mayo) **348**
31.1. Arquitectura y funcionamiento general . . . 348
31.2. Funcionalidad mínima . . . 350
31.3. Funcionalidad optativa . . . 352
31.4. Entrega de la práctica . . . 353
31.5. Notas y comentarios . . . 355

32.Proyecto final (2017, junio) **355**

33.Proyecto final (2016, mayo) **357**
33.1. Arquitectura y funcionamiento general . . . 357
33.2. Funcionalidad mínima . . . 359
33.3. Funcionalidad optativa . . . 362
33.4. Entrega de la práctica . . . 362
33.5. Notas y comentarios . . . 364

34.Proyecto final (2016, junio) **365**

35.Proyecto final (2015, mayo y junio) **366**
35.1. Arquitectura y funcionamiento general . . . 366
35.2. Funcionalidad mínima . . . 367
35.3. Funcionalidad optativa . . . 369
35.4. Entrega de la práctica . . . 370
35.5. Notas y comentarios . . . 372

36.Proyecto final (2014, mayo) **373**
36.1. Arquitectura y funcionamiento general . . . 373
36.2. Funcionalidad mínima . . . 374
36.3. Funcionalidad optativa . . . 376
36.4. Entrega de la práctica . . . 377
36.5. Notas y comentarios . . . 379

37.Proyectos finales anteriores **379**
37.1. Proyecto final (2013, mayo) . . . 379
37.2. Arquitectura y funcionamiento general . . . 379
37.3. Funcionalidad mínima . . . 381
37.4. Funcionalidad optativa . . . 383
37.5. Entrega de la práctica . . . 384
37.6. Notas y comentarios . . . 385
37.7. Proyecto final (2012, diciembre) . . . 386
37.7.1. Arquitectura y funcionamiento general . . . 386
37.7.2. Funcionalidad mínima . . . 387

37.7.3. Funcionalidad optativa . . . 389
37.8. Proyecto final (2011, diciembre) . . . 390
37.8.1. Arquitectura y funcionamiento general . . . 390
37.8.2. Funcionalidad mínima . . . 392
37.8.3. Esquema de recursos servidos (funcionalidad mínima) . . . 393
37.8.4. Funcionalidad optativa . . . 394
37.8.5. Notas y comentarios . . . 395
37.9. Proyecto final (2012, mayo) . . . 395
37.9.1. Arquitectura y funcionamiento general . . . 395
37.9.2. Funcionalidad mínima . . . 395
37.10Proyecto final (2010, enero) . . . 396
37.10.1Arquitectura y funcionamiento general . . . 396
37.10.2Funcionalidad mínima . . . 397
37.10.3Funcionalidad optativa . . . 399
37.10.4Entrega de la práctica . . . 399
37.10.5Notas y comentarios . . . 400
37.11Proyecto final (2010, junio) . . . 400
37.12Proyecto final (2010, diciembre) . . . 401
37.12.1Arquitectura y funcionamiento general . . . 402
37.12.2Funcionalidad mínima . . . 403
37.12.3Funcionalidad optativa . . . 404
37.12.4Entrega de la práctica . . . 405
37.12.5Notas y comentarios . . . 406
37.12.6Notas de ayuda . . . 407
37.13Proyecto final (2011, junio) . . . 408
37.13.1Arquitectura y funcionamiento general . . . 408
37.13.2Funcionalidad mínima . . . 409
37.13.3Funcionalidad optativa . . . 411
37.13.4Entrega de la práctica . . . 412
37.13.5Notas y comentarios . . . 412
37.13.6Notas de ayuda . . . 413

38.Materiales de interés **415**
38.1. Material complementario general . . . 415
38.2. Introducción a Python . . . 415
38.3. Aplicaciones web mínimas . . . 415
38.4. SQL y SQLite . . . 416

39.Preguntas más fecuentes **416**
39.1. Django: Referencia a elementos en otro módulo . . . 416

1. Datos generales

Título:	Servicios Telemáticos
Titulación:	Grado en Ingeniería en Tecnologías de Telecomunicación
Cuatrimestre:	Tercer curso, segundo cuatrimestre
Créditos:	6 (3 teóricos, 3 prácticos)
Horas lectivas:	4 horas semanales
Horario:	lunes, 11:00–13:00 jueves, 11:00–13:00
Profesores:	Alberto Rafael Rodríguez Iglesias alberto.rodriguezi @ urjc.es David Moreno Lumbreras david.morenolu @ urjc.es Despacho 103, Biblioteca Sergio Montes sergio.montes @ urjc.es Despacho 103, Biblioteca
Sedes telemáticas:	https://aulavirtual.urjc.es https://cursosweb.github.io https://gitlab.etsit.urjc.es/cursosweb
Aulas:	Laboratorio 209, Edif. Laboratorios III (lunes y jueves)

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 medido de lo posible en el laboratorio, y las prácticas realizadas (incluyendo especialmente el proyecto final) tendrán gran importancia en la evaluación de la asignatura. Los conocimientos teóricos necesarios se intercalarán con los prácticos, en gran media 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 (aula virtual de la URJC) para realizar actividades complementarias a las presenciales, y para organizar parte de la documentación ofrecida a los alumnos. La mayoría de los contenidos utilizados en la asignatura están disponibles o enlazados desde el sitio web CursosWeb. Asimismo, se utiliza el servicio GitLab de la ETSIT como repositorio, tanto de los materiales de la asignatura, como para entregar las prácticas por parte de los alumnos.

4. Evaluación

4.1. Criterios de evaluación

Parámetros generales:

- Teoría (obligatorio): 0 a 4.
- Microprácticas diarias: 0 a 1
- Miniprácticas preparatorias: 0 a 1
- Proyecto final (obligatorio): 0 a 2.
- Opciones y mejoras del proyecto final: 0 a 3
- Nota final: Suma de notas, moderada por la interpretación del profesor
- Mínimo para aprobar:
  - aprobado en teoría (2) y proyecto final (1)
  - 5 puntos de nota final

Evaluación teoría: prueba escrita
Evaluación microprácticas diarias (evaluación continua):

- entre 0 y 1
- preguntas y ejercicios en foro y entragados en GitLab
- es muy recomendable hacerlas

Evaluación proyecto final:

- posibilidad de examen presencial para proyecto 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 proyecto final:

- permiten subir la nota mucho

Evaluación extraordinaria:

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

5. Calendario

Lunes	Jueves
Festivo	25 de enero: Presentación (2 horas) 25 de enero: Python y entorno de desarrollo I
Festivo	1 de febrero: Python y entorno de desarrollo I 1 de febrero: Python II (1 hora)
5 de febrero: Conceptos básicos I (2 horas)	8 de febrero: Python III (2 horas)
12 de febrero: Conceptos básicos II (2 horas)	15 de febrero: Aplicaciones web (2 horas)
19 de febrero: Conceptos básicos III (2 horas)	22 de febrero: Servidores con clase (2 horas)
26 de febrero: Conceptos básicos IV (2 horas)	29 de febrero: Servidores con clase II (2 horas)
4 de marzo: Conceptos básicos V (2 horas)	7 de marzo: Django I (2 horas)
11 de marzo: Interoperación web I (2 horas)	14 de marzo: Django II (2 horas)
18 de marzo: Interoperación web II (2 horas)	21 de marzo: Django III (2 horas)
Festivo	Festivo
Festivo	4 de abril: Django IV (2 horas)
8 de abril: MVC (2 horas)	11 de abril: Django V (2 horas)
15 de abril: MVC II (2 horas)	18 de abril: Django VI (2 horas)
22 de abril: XML, JSON I (2 horas)	25 de abril: Django VII (2 horas)
29 de abril: XML, JSON II (2 horas)	Festivo
6 de mayo: CSS (2 horas)	9 de mayo: Detalles finales I (2 horas)

6. Programa de teoría

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

6.1. 00 - Presentación

6.1.1. 25 de enero: Presentación (2 horas)

- **Presentación:** Presentación de la temática de la asignatura
- **Presentación:** Qué son las aplicaciones web del lado del servidor ("back-end") y del lado del cliente ("front-end"), y cómo se relacionan.
- **Presentación:** Detalles de la asignatura: teoría y prácticas, estructura de las clases, evaluación, etc.
- **Presentación:** Materiales de la asignatura: sitios web y documentos fundamentales que tenéis a vuestra disposición.
- **Material:** Transparencias, tema "Presentación".
- **Ejercicio propuesto (voluntario, entrega en el foro):** "Web 2.0" (ejercicio 10.1)
  Entrega recomendada: antes del 1 de febrero.

6.2. 01 - Conceptos básicos de aplicaciones web

Sesión, mantenimiento de estado, persistencia.

6.2.1. 5 de febrero: Conceptos básicos I (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.

- **Ejercicio (presentación en clase):** "Espía a tu navegador (Firefox Developer Tools)" (ejercicio 10.3)
  Centrado en las pestañas de "Red" e "Inspección".
- **Ejercicio propuesto (entrega en el foro):** "Explora tus cookies" (ejercicio 10.5)
  Entrega recomendada: antes del 12 de febrero.

6.2.2. 12 de febrero: Conceptos básicos II (2 horas)

- **Frikiminutos**: "De rebajas" (Markdown).
- Resolución de ejericios pendientes.
- **Ejercicio (discusión en clase):** "Explora tus cookies" (ejercicio 10.5). Explicación de cookies en interacciones HTTP y almacenadas, vistas en el depurador del navegador.
- **Presentación:** Cookies
- **Material:** Transparencias, tema "Cookies"
- **Ejercicio (entrega en el foro):** "Última búsqueda" (ejercicio 10.2)
  Entrega recomendada: antes del 19 de febrero.

6.2.3. 19 de febrero: Conceptos básicos III (2 horas)

- **Frikiminutos**: "Pregunta, que te responderán" (Stackoverflow).
- **Ejercicio (discusión en clase):** "Última búsqueda" (ejercicio 10.2)
  Se introducen las dos solucines típicas: cookie con la pregunta, y cookie con un identificador más tabla en el servidor. Discusión sobre el uso de cookies (u otros mecanismos) para conseguir la funcionalidad requerida. Se discute también el almacenamiento en el lado del servidor y en el lado del cliente. Relación entre peticiones HTTP. Cookies como herramienta para ambas situaciones. Analogía con la agencia de viajes física. Cómo tienen que ser las cookies de identificación para que no sean fácilmente "adivinables".
- **Discusión:** Usos de las cookies.
  Uso de las cookies para identificación de visitantes (como en el ejercicio 10.2), para autenticación (interacción de autenticación y cookie de sesión posterior), para almacenamiento (como en el ejercicio 10.2, 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 propuesto (entrega en el foro):** "Explora tus cookies (2)" (ejercicio 10.6)
  Entrega recomendada: antes del 26 de febrero.

6.2.4. 26 de febrero: Conceptos básicos IV (2 horas)

- **Frikiminutos**: "Lo importante es participar" (Google Summer of Code).
- **Ejercicio (discusión en clase):** "Explora tus cookies (2)" (ejercicio 10.6).
- **Discusión:** Datos persistentes entre operaciones HTTP diferentes. Concepto de estado persistente frente a caídas del servidor.
- **Ejercicio propuesto:** "Contador simple" (ejercicio 10.15)
- **Discusión:** Introducción al problema de los rearranques.
- **Discusión:** Medición de audiencias y visitas únicas por un sitio web.
- **Ejercicio propouesto (entrega en el foro):** "Traza de historiales de navegación por terceras partes" (ejercicio 10.22)
  Entrega recomendada: antes del 4 de marzo.
- **Ejercicio propuesto:** "cURL básico" (ejercicio 10.16)

6.2.5. 4 de marzo: Conceptos básicos V (2 horas)

- **Frikiminutos**: "¿Qué datos tuyos tienen?"
- **Discusión de ejercicio:** "Traza de historiales de navegación por terceras partes" (ejercicio 10.22)
- **Discusión:** Relación de traza de historias de navegación con identidades personales.
- **Discusión de ejercicio:** "cURL básico" (ejercicio 10.16)
- **Ejercicio (demo):** "Protección contra trackers en el navegador" (ejercicio 10.25).
- **Discusión de ejercicio:** "Distinto contenido según navegador" (ejercicio 10.17).
- **Discusión de ejercicio:** "Transplante de cookies" (ejercicio 10.26)
- **Ejercicio (discusión en clase):** "Contador simple con varios navegadores intercalados" (ejercicio 10.20)
- **Ejercicio (entrega en el foro):** "Contador simple con rearranques" (ejercicio 10.21).
  Entrega recomendada: antes del 11 de marzo.

6.3. 02 - Servicios web que interoperan

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

6.3.1. 11 de marzo: Interoperación web I (2 horas)

- Introducción al diseño de APIs HTTP
- **Ejercicio (discusión en clase)**: "Lista de la compra" (ejercicio 11.3). Trabajo en grupos y discusión de los detalles del ejercicio.
- **Presentación:** Arquitectura REST (1)
- **Material:** Transparencias, tema "REST"

6.3.2. 18 de marzo: Interoperación web II (2 horas)

- **Presentación:** Arquitectura REST (2)
- **Material:** Transparencias, tema "REST"
- **Discusión:** Introducción a las operaciones idempotentes.
- **Ejercicio (discusión en clase):** "Listado de lo que tengo en la nevera" (ejercicio 11.4).
- **Ejercicio (entrega en el foro):** "Listado de lo que tengo en la nevera" (ejercicio 11.4)
  Entrega recomendada: 8 de abril

6.4. 03 - Modelo-vista-controlador

Explicación del patrón de diseño "modelo-vista-controlador".

6.4.1. 8 de abril: MVC (2 horas)

- **Presentación:** "Tres implementaciones de una aplicación web simple: Counter". Comparación de tres formas de implementar una aplicación web muy sencilla, identificando los componentes y estructuras que se repiten, con el objetivo de ver cómo cuando pasamos a Django seguimos construyendo el mismo tipo de apicaciones, aunque el marco de programación nos proporcione ya muchos elementos que no tenemos que construir.
- **Código:**

- `counter-server-1.py`, directorio `Python-Web/counter`
- `counterapp.py`, directorio `Python-Web/http-server-classes/counterapp.py`
- Projecto Django `django-counter`, directorio `Python-Django`

6.4.2. 15 de abril: MVC II (2 horas)

- **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.
- **Ejercicio (discusión en clase y entrega voluntaria en el foro):** "Red social muy simple" (ejercicio 12.1).
- **Video:** "Arquitectura Modelo-Vista-Controlador"

6.5. 04 - Introducción a XML y JSON

Uso de XML en aplicaciones web.

6.5.1. 22 de abril: XML, JSON I (2 horas)

- **Presentación:** "XML: Conceptos fundamentales".
  Introducción a XML, sintaxis básica, equivalencia con el árbol XML correspondiente a un documento.
- **Video:** "XML: Conceptos fundamentales".
- **Presentación:** "XML: Lenguajes de definición de vocabularios XML".
  Formas de especificar vocabularios XML, ejemplo de DTD simple.
- **Video:** "XML: Lenguajes de definición de vocabularios XML".
- **Presentación:** "XML: Módulos habituales para trabajar con XML desde lenguajes de programación".
  Reconocedores SAX y DOM, y su uso en aplicaciones web.
- **Video:** "XML: Módulos habituales para trabajar con XML desde lenguajes de programación".

- **Presentación:** "XML: Usos en aplicaciones web".
  Usos habituales de XML en aplicaciones web, incluyendo el DOM de los navegadores y los canales RSS.
- **Video:** "XML: Usos en aplicaciones web".
- **Material:** Transparencias "Introducción a XML".
- **Ejercicio (discusión en clase):** "Chistes XML" (ejercicio 13.1).
- **Ejercicio (discusión en clase):** "Chistes XML (parser DOM)" (ejercicio 13.2).
- **Ejercicio (discusión en clase):** "Videos en canal de YouTube" (ejercicio 13.5).
- **Ejercicio (entrega en GitLab):** "Videos en canal de YouTube (con descarga)" (ejercicio 13.6)
  Repositorio: https://gitlab.etsit.urjc.es/cursosweb/practicas/server/youtube-descarga

6.5.2. 29 de abril: XML, JSON II (2 horas)

- **Presentación:** "JSON (JavaScript Object Nottion)".
  JSON como formato de intercambio de datos en aplicaciones web.
- **Video:** "JSON (JavaScript Object Nottion)".
- **Presentación:** "XML y JSON: Ejemplos reales".
  Ejemplo de canal XML de YouTube y documento JSON de GitLab.
- **Video:** "XML y JSON: Ejemplos reales".
- **Ejercicio (discusión en clase):** "Forks de un repositorio GitLab" (ejercicio 13.16)
- **Ejercicio (discusión en clase):** "Gestor de contenidos con videos de YouTube (simple)" (ejercicio 13.10)
- **Ejercicio (entrega en GitLab):** "Gestor de contenidos con videos de YouTube (2)" (ejercicio 13.11)
  Repositorio: https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-youtube
  Entrega: 9 de mayo
- **Ejercicio voluntario:** "Municipios JSON via HTTP" (ejercicio 13.15).

6.6. 05 - Hojas de estilo CSS

Hojas de estilo CSS, separación entre contenido y presentación.

6.6.1. 6 de mayo: CSS (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 el depurador de Firefox.
- **Ejercicio (discusión en clase):** "Django cms_css simple" (ejercicio 14.1).
- **Ejercicio (entrega en GitLab):** "Django cms_css elaborado" (ejercicio 14.2).
  Entrega recomendada: antes del 1 de mayo.

7. Programa de prácticas

Programa de las prácticas de la asignatura (tentativo).

7.1. P1 - Introducción a Python

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

7.1.1. 25 de enero: Python y entorno de desarrollo I

- **Presentación:** "Introducción a Python" (breve introducción a los materiales recomendados).
- **Material:** Transparencias "Introducción a Python"
- **Ejercicio (realizado en clase):** "Uso interactivo del intérprete de Python" (ejercicio 16.1)
- **Presentación y demo:** PyCharm
- **Presentación y demo:** El depurador de PyCharm

7.1.2. 1 de febrero: Python y entorno de desarrollo II

- **Presentación y demo:** GitLab, y git creación de repositorios, clonado de repositorios.
- **Ejercicio:** "Asistente de IA generativa para PyCharm" (ejercicio 16.3)
- **Presentación:** Mecanismos de conexión remota al laboratorio: VNC desde el navegador, ssh, etc.
- **Ejercicio (realizado en clase):** "Subida de programa a repo de GitLab" (ejercicio 16.2)
  Entrega recomendada: antes del 8 de febrero.

7.1.3. 1 de febrero: Python II (1 hora)

- **Presentación:** "Introducción a Python" (listas, bucles (for), ficheros, strings, argumentos)
- **Material:** Transparencias "Introducción a Python"
- **Presentación:** "Entornos virtuales en Python"

- **Ejercicio propuesto (entrega en GitLab):** "Calculadora" (ejercicio 16.8). Entrega recomendada: antes de 15 de febrero.

7.1.4. 8 de febrero: Python III (2 horas)

- **Frikiminutos Python:** "Despliegue de páginas web en GitHub/GitLab".
- **Ejercicio (hecho en clase):** Uso de ficheros (ejercicio 16.6). (Ligeramente modificado para que escriba en fichero tambien)
- **Presentación:** "Introducción a Python". Clases y orientación a objetos en Python.
- **Material:** Transparencias "Introducción a Python"
- **Ejercicio propuesto:** "Descarga de documentos web con módulos" (ejercicio 16.10). Entrega recomendada: antes del 15 de febrero.

7.2. P2 - Aplicaciones web simples

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

7.2.1. 15 de febrero: Aplicaciones web (2 horas)

- **Ejercicio:** "Aplicación web hola mundo" (ejercicio 17.1)
  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 17.2).
- **Explicación de ejercicio:** "Aplicación web generadora de URLs aleatorias" (ejercicio 17.3)
- **Ejercicio propuesto (entrega en GitLab):** "Aplicación redirectora" (ejercicio 17.4) Entrega recomendada: antes del 22 de febrero.

7.3. 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.

7.3.1. 22 de febrero: Servidores con clase (2 horas)

- **Frikiminutos Python:** "Eliminación de fondo en fotos".
- **Comentario de ejercicio:** "Aplicación web generadora de URLs aleatorias" (ejercicio 17.3)
- **Comentario de ejercicio:** "Aplicación redirectora" (ejercicio 17.4).
- **Comentario de ejercicio:** "Sumador simple" (ejercicio 17.5)
- **Trabajo y explicación del ejercicio:** "Clase servidor de aplicaciones" (ejercicio 17.6)
  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 GitLab):** "Clase contentApp" (ejercicio 19.1)
  Entrega recomendada: antes de 29 de febrero.

7.3.2. 29 de febrero: Servidores con clase II (2 horas)

- **Comentario de ejercicio:** "Clase contentApp" (ejercicio 19.1)
- **Realización de ejercicio:** "Clase contentPutApp" (ejercicio 19.3)
- **Ejercicio recomendado (entrega voluntaria):** "Clase contentPostApp" (ejercicio 19.4)
- **Presentación minipráctica (entrega en GitLab):** "Minipráctica 1" (ejercicio 9.2).
  Entrega recomendada: antes del 14 de marzo.

7.4. P4 - Introducción a Django

7.4.1. 7 de marzo: Django I (2 horas)

Presentación de Django como sistema de construcción de aplicaciones web.

- **Presentación:** Introducción a Django (primera parte)
- **Ejercicio:** "Instalación de Django" (ejercicio 18.1).
- **Ejercicio:** "Django Intro" (ejercicio 18.2).
- **Material:** Transparencias "Introducción a Django"

- **Ejercicio (discusión en clase:** "Django Primera Aplicación" (ejercicio 18.3).
- **Material:** Guión https://gsyc.urjc.es/grex/cursosweb/guion.html
- **Ejercicio (discusión en clase):** "Django Primera Aplicación" (ejercicio 18.3).

7.4.2. 14 de marzo: Django II (2 horas)

Primeros ejercicios con base de datos.
Usuarios, administración y autenticación con Django.

- **Presentación:** Introducción a Django (segunda parte)
- **Material:** Guión https://gsyc.urjc.es/grex/cursosweb/guion2.html
- **Ejercicio (discusión en clase):** "Django calc" (ejercicio 18.4).
- **Ejercicio (discusión en clase):** "Django cms" (ejercicio 18.5).
- **Ejercicio (discusión en clase):** "Django cms_put" (ejercicio 18.6).
- **Ejercicio (entrega en GitLab):** "Django cms_post" (ejercicio 18.7).
  Entrega recomendada: hasta el 21 de marzo.

7.4.3. 21 de marzo: Django III (2 horas)

- **Presentación:** Introducción a Django (tercera parte)
- **Material:** Guión https://gsyc.urjc.es/grex/cursosweb/guion3.html
- **Ejercicio (discusión en clase):** "Django cms_templates" (ejercicio 18.10).
- **Ejercicio (discusión en clase):** "Django cms_post" (ejercicio 18.11)
- Presentación de la **Práctica 2** (ejercicio 9.3)
  Entrega recomendada: antes del 11 de abril.

7.4.4. 4 de abril: Django IV (2 horas)

- **Presentación:** Tests con Django y GitLab
- **Material:** Repositorio https://gitlab.etsit.urjc.es/cursosweb/practicas/server/testing-example

7.4.5. 11 de abril: Django V (2 horas)

- **Presentación:** Introducción a Django (quinta parte)
- **Material:** Guión https://gsyc.urjc.es/grex/cursosweb/guion4.html
- **Material:** Transparencias "Introducción a Django"
- **Ejercicio (discusión en clase):** "Django cms_users" (ejercicio 18.8).
- **Ejercicio (discusión en clase):** "Django cms_users_put" (ejercicio 18.9). Entrega recomendada: antes del 18 de abril.

7.4.6. 18 de abril: Django VI (2 horas)

- **Material:** Guión https://gsyc.urjc.es/grex/cursosweb/guion5.html
- **Material:** Transparencias "Introducción a Django"
- **Ejercicio (discusión en clase):** "Django cms_post" (ejercicio 18.11). Entrega recomendada: antes del 25 de abril.
- **Presentación:** Práctica final (apartado 8).
  Entrega recomendada: día del examen de la asignatura.

7.4.7. 25 de abril: Django VII (2 horas)

- **Presentación:** Introducción a Django (formularios)
- **Ejercicio:** "Django cms_forms" (ejercicio 18.12)

- **Ejercicio:** "Django cms_bootstrap cuadrícula" (ejercicio 14.3)
- **Ejercicio:** "Django cms_bootstrap componentes" (ejercicio 14.4)
- **Ejercicio:** "Django cms_bootstrap componentes personalizados" (ejercicio 14.5)

7.4.8. 9 de mayo: Detalles finales I (2 horas)

- **Ejercicio:** "Django cms_bootstrap cuadrícula" (ejercicio 14.3)
- **Presentación:** Práctica final (apartado 25)

8. Proyecto final: DeCharla (2023)

Repositorio plantilla.
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/final-decharla

[**Nota importante:** Enunciado aún en proceso de elaboración, puede sufrir algunos cambios antes de su versión final. Por favor, avisa si detectas algún error o inconsistencia.]

Este es el enunciado del proyecto final del curso 2022-2023, tanto para la convocatoria ordinaria como para la extraordinaria.

La práctica final de la asignatura consiste en la creación de una aplicación web, llamada "DeCharla", que permitirá poner mensajes, y ver mensajes que hayan puesto otros. Algunos de los mensajes podrán ser obtenidos automáticamente de ciertas fuentes soportadas. Los mensajes podrán ser un texto o una imagen. Los mensajes se organizarán en "salas", que serán accesibles cada una en un recurso. El nombre del recurso de cada sala será decidido por quien la cree, en el momento de la creación. El "nombre de la sala" será el nombre del recurso de la sala, quitándole la "/" inicial. La "página de la sala" será al página HTML que se recibe cuando se invoca mediante GET el recurso de la sala. No habrá autenticación de usuarios para acceder a las salas: cada visitante podrá poner mensajes en una sala si conoce el nombre de su recurso. Cuando en este enunciado se indique que "hizo algo en el sitio" (por ejemplo, se diga "se verán los mensajes escritos anteriormente" o "desde que se visitó la página por última vez") se refiere a que se hizo algo en el sitio desde el mismo navegador en que se está ahora (por ejemplo "se verán los mensajes escritos anteriormente desde el mismo navegador" o "desde que se visitó la página por última vez desde el mismo navegador").

A continuación se describe el funcionamiento y la arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

8.1. Requisitos generales

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones (*apps*) Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de Django.
- Todas las bases de datos que contenga la aplicación tendrán que ser accesibles vía la interfaz que proporciona el "Admin Site" (además de lo que pueda hacer falta para que funcione al aplicación). Para acceder a este "Admin Site" habrá cuentas específicas, creadas vía el propio "Admin Site".

- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica tenga un aspecto similar) para definir las páginas que se servirán a los navegadores de los usuarios (excepto las páginas de "Admin Site" (ver "estructura de las páginas HTML, a continuación).

Se servirán al menos las siguientes páginas (recursos) con los contenidos descritos a continuación:

Página principal. En la página principal se ofrecerá un listado de todas las salas en las que se ha puesto algún mensaje. Para cada sala se indicará su nombre, que se presentará como un enlace a la página de la sala, y dos números: el número de mensajes total, y el número de mensajes desde que se visitó la sala por última vez.

Página de cada sala. En la página de cada sala se mostrarán:

- Formularios para poner un mensaje en la sala. Como el mensaje podrá ser un texto o una imagen, el formulario incluirá una caja de texto, en el que se podrá escribir el texto de un mensaje o la URL de una imagen, y un "checkbox"[1] que si está seleccionado indicará que el texto es una URL (y el texto de un mensaje si no es así).
- Un listado de todos los mensajes puestos en la sala desde cualquier navegador (primero los más recientes). Las imágenes se mostrarán con un tamaño del 50 % del ancho disponible, usando por ejemplo[2]:

```
img {
  width: 50%;
  height: auto;
}
```

Página dinámica de cada sala. En la página dinámica de cada sala se mostrará el mismo contenido que en la página de una sala, pero usando un script JavaScript que actualizará el contenido cada poco tiempo (ver más adelante, en la lista de preguntas frecuentes (apartado 8.6).

Página de configuración. Permitirá configurar el sitio para ese navegador. Incluirá:

[1] Input type checkbox:
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/checkbox

[2] Ver detalles sobre cómo definir el tamaños de la imágenes en:
https://imagekit.io/blog/how-to-resize-image-in-html/

- Un formulario para elegir el nombre con el que se pondrán los mensajes (nombre de charlador).
- Un formulario para elegir los elementos de apariencia personalizables del sitio (el tamaño y el tipo de la letra).

Página de ayuda. Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento y una brevísima documentación.

Elementos generales que aparecerán en todas las páginas HTML:

Cabecera. En la parte superior estará la cabecera del sitio, que consistirá de un texto (el nombre del sitio, "DeCharla") en un tipo de letra que resalte claramente, sobre una imagen de fondo. El texto será un enlace al recurso "/" del sitio.

Menú. En una barra debajo del elemento anterior tendremos un menú desde el que se podrá acceder, mediante enlaces, al menos a las siguientes páginas (salvo si la opción apunta a la página en la que se está, en cuyo caso no aparecerá en esa página):

- la página principal (con el texto "Principal")
- la página de configuración (con el texto "Configuración")
- la página de ayuda (con el texto "Ayuda")
- la página de acceso al "Admin Site" (con el texto "Admin").

Extra. En una columna a la derecha (en escritorio) o bajo los elementos anteriores (en móviles):

- El nombre de charlador que se está usando desde ese navegador (o "Anónimo" si no se ha especificado uno).
- Un formulario para acceder a una sala indicando su nombre. El formulario permitirá escribir el nombre, y tendrá un botón asociado que, al pulsarlo, pasará a mostrar la página de la sala (mediante una redirección). Es importante darse cuenta de que no hace falta que se hayan puesto mensajes previamente en esa sala para que pueda "verse": cualquier nombre puesto en el formulario se considerará el nombre de una sala.

Cuerpo. En una columna más ancha a la izquierda (en escritorio) o bajo los elementos anteriores (en móviles), el cuerpo de la página (los contenidos específicos de esa página, descritos más abajo).

Pie. Un pie de página con un resumen de métricas: "Mensajes: XXX (textuales), YYY (imágenes). Salas activas: ZZZ", siendo XXX el número de mensajes, YYY el número de imágenes, y ZZZ el número de salas con al menos un mensaje en todo el sitio.

En lo que tiene que ver con el aspecto de las páginas se tendrá en cuenta:

Marcado HTML. Cada uno de los elementos generales descritos anteriormente estará construido dentro de un elemento `div`, marcado con un atributo `id` en HTML, para poder ser referido fácilmente en hojas de estilo CSS. Cuando sea conveniente, se podrán utilizar en lugar de `div` elementos de HTML5 (`header`, `footer`, `nav`, etc).

CSS. Se utilizarán hojas de estilo CSS para determinar la apariencia del sitio. Estas hojas definirán al menos el color de fondo y de letra de cada una de las partes (elementos) marcadas con un *id* (ver "Marcado HTML"). Además, elementos que deban tener el mismo aspecto deberían estar en una misma clase CSS, para poder gestionarlo de forma común. También definirán los elementos personalizables del sitio: el tamaño de la letra y el tipo de la letra.

Bootstrap. Se utilizará Bootstrap para la maquetación (*layout*) de las páginas, de forma que funcionen adecuadamente tanto en navegadores de escritorio como en móviles.

El sitio también proporcionará:

Salas en formato JSON. Para cada sala, se ofrecerá un recurso donde el documento servido estará en formato JSON, e incluirá todos los mensajes puestos en la sala, en el siguiente formato:

```
[
  {
    "author": "XXX",
    "text": "....",
    isimg: false,
    date: "2023-04-16:14:04:01"
  },
  {
    ....
  }
  ...
]
```

Actualización de salas en XML. Para cada sala, se ofrecerá un recurso donde se admitirá recibir un documento XML en el cuerpo de una petición HTTP PUT. Los contenidos de este cuerpo se usarán para recibir nuevos mensajes para la sala. El formato del documento XML será como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>

<messages>
  <message isimg="false">
    <text>
       ...
    </text>
  </message>
  <message isimg="true">
    <text>
       https://yyy....
    </text>
  </message>
  ...
</messages>
```

Autenticación:

- Todos los recursos del sitio estarán "protegidos" de forma que sólo se podrá acceder a ellos (mediante cualquier comando HTTP) si se ha iniciado una sesión.
- Cuando un navegador acceda a cualquier recurso del sitio sin haber iniciado sesión, recibirá un formulario en el que se le solicitará una contraseña. El sitio mantendrá una lista de una o más contraseñas válidas (que serán cadenas alfanuméricas de cualquier longitud), en una tabla de la base de datos. Si la contraseña indicada en el formulario es una de ellas, se iniciará la sesión usando cookies (y se realizará la función correspondiente al recurso invocado). Si no es así, se enviará el formulario de nuevo, con código de error 401 (Unauthorized).
- Una vez un navegador ha recibido una cookie de autenticación, no tendrá que mostrar ya el formulario de autenticación a ese navegador.
- Los scripts que accedan al sitio para poner noticias mediante la interfaz XML podrán utilizar una contraseña válida, incluyéndola en su petición en la cabecera Authorization como la siguiente (siendo "xx34d23" la contraseña):

```
Autorization: Basic xx34d23
```

Tests:

Extremo a extremo. Para cada tipo de recurso de la práctica habrá que realizar al menos un test "extremo a extremo".

Unitarios. Será conveniente (pero no obligatorio) realizar también tests unitarios para distintas partes del código. Se se hace, es conveniente mencionarlo en el fichero de entrega como una parte opcional.

8.2. Despliegue

La práctica deberá estar desplegada en algún sitio de Internet, de forma que pueda accederse a ella. Deberá mantenerse desplegada y activa al menos desde el día de entrega de la práctica, hasta el día del cierre de actas.

Para el despliegue, se puede utilizar Python Anywhere[3], que proporciona un plan gratuito que incluye suficientes recursos como para poder desplegar la práctica.

Si el alumno así lo desea, puede considerarse desplegar en un ordenador dedicado (por ejemplo, una Raspberry Pi accesible directamente desde Internet, alojada en su hogar), o en servicios como Google Computing Engine[4]. En general, dado que este tipo de despliegues no podrá contar con una ayuda detallada por los profesores, estará algo más valorado.

En el caso de que la práctica se despliegue en Python Anywhere, hay que tener en cuenta que sus máquinas virtuales tienen cortado el acceso a todos los sitios de Internet salvo los que están en una "lista blanca"[5] (*whitelist*). Es de esperar que esto no cause problemas, porque el sitio GitLab de la ETSIT, donde está vuestro código fuente, está ya en la lista blanca, y por ello Python Anywhere debería poder acceder a él para clonar vuestro repositorio. Sin embargo, si vuestra aplicación se conectase a cualquier sitio para obtener información,si este sitio no está en la lista blanca, vuestra aplicación no se podrá conectar. Wikipedia y Flickr, por ejemplo, están ya en la lista blanca. Pero otros sitios que podríais estar usando, no.

Para lo tanto, si hacéis el despliegue en Python Anywhere y vuestra aplicación ha de obtener datos de un sitio tercero:

- Si está ya en la lista blanca de Python Anywhere, funcionará sin problemas sin hacer nada especial. Si os funcionaba ya en las pruebas locales, así que también deberían funcionar en el despliegue.

[3]Python Anywhere: https://pythonanywhere.com
[4]GCP Engine Free: https://cloud.google.com/free/
[5]Lista blanca de Python Anywhere: https://www.pythonanywhere.com/whitelist/

- Si no están en la lista blanca de Python Anywhere, aseguraos de que la base de datos que subís al despliegue de vuestra práctica incluya ya datos de ese sitio tercero..

Tened en cuenta que si usáis otras plataformas para el despliegue, puede que os encontréis problemas similares. Y tened en cuenta también que en cualquier caso, nosotros probaremos la práctica en otros despliegues, así que todos los recursos reconocidos que hayáis implementado deben funcionar correctamente si no hay restricciones de conexión.

Tenéis más detalles sobre cómo se hace un despliegue de una aplicación Django en Python Anywhere en el vídeo "Django: Despliegue en Python Anywhere"[6], que explica cómo desplegar allí la aplicación `django-youtube-4`[7] (ver también el fichero `README.md` de esa aplicación para más detalles).

8.3. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habéis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio (FÁCIL).
- Permitir que las sesiones autenticadas se puedan terminar. Esto es, que haya una opción para "terminar la sesión", de forma que al usarla, si se vuelve a acceder al sitio vuelva a recibirse el formulario para autenticarse (FÁCIL).
- Permitir votar las las salas. Para ello, en cada sala aparecerá un botón "Me gusta" para indicar que se da un voto a la sala. Cada sala, en el listado general, saldrá junto a los votos que tiene. (MEDIO)
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario de la aplicación tendrá en cuenta lo que especifique el navegador.

[6] https://www.youtube.com/watch?v=hlZPC5L2Itc

[7] https://github.com/CursosWeb/Code/tree/master/Python-Django/django-youtube-4

Para ello, pueden utilizarse las facilidades que proporciona Django para la localización[8] (MEDIO).

- Mejora de los tests de la práctica, incluyendo test de condiciones de error, test de escenarios con más de una invocación de recurso, tests de API Python, etc. En general, si se realizan tests más allá de los básicos "extremo a extremo" que se piden, serán considerados como opciones (MEDIO).
- Creación de un script para utilizar información de un sitio y subirla a la DeCharla como uno o varios mensajes, usando el formato XML descrito anteriormente. Por ejemplo, se puede usar información de la AEMET para subir a una cierta sala mensajes sobre el tiempo previsto en una localidad. Pueden realizarse varios scripts de este estilo, con diferentes sitios de información (DIFÍCIL).
- Igual que la anterior, pero incorporando directamente a DeCharla la funcionalidad de obtener información de sitios terceros e incluirla como mensajes en una sala. En este caso, en cada sala aparecerá uno o varios formularios, cada uno con los datos necesarios para incorporar esa funcionalidad. Por ejemplo, para incorporar previsiones de tiempo de la AEMET, el formulario ofrecerá una caja de texto para poner el nombre de la localidad (DIFÍCIL).

8.4. Entrega de la práctica

- **Fecha límite de entrega (convocatoria ordinaria):** día anterior al del examen de la asignatura a las 23:55 (hora española peninsular)
- **Fecha límite de entrega (convocatoria extraordinaria):** día anterior al del examen de la asignatura, a las 23:55 (hora española peninsular)
- **Notificación de alumnos que tendrán que realizar entrevista:** antes de la fecha de revisión de la asignatura.
- **Realización de entrevistas:** junto con la revisión de la asignatura.

La entrega de la práctica consiste en:

1. **Subir tu práctica a un repositorio en el GitLab de la Escuela**. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado una derivación (fork) del repositorio de plantilla de al asignatura, indicado al principio de este enunciado.

[8] Django, Internationalization and localization: https://docs.djangoproject.com/en/4.2/topics/i18n/

Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitLab.

Se recomienda mantener el repositorio como privado, hasta el momento en que se entregue la práctica.

2. Para considerar la práctica como entregada es fundamental que en el directorio raíz del repositorio de entrega haya un fichero `README.md` cuya primera línea sea:

   ```
   # ENTREGA CONVOCATORIA XXX
   ```

   Siendo `XXX` "MAYO" si se entrega en la convocatoria ordinaria de mayo, y "JUNIO" si se entrega en la convocatoria extraordinaria de junio.

   Atención: No se considerará entregada la práctica si no ve esta primera línea en el formato adecuado.

3. **Entregar un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional**, si se ha realizado parte optativa. Los vídeos 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). 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, Twitch, o YouTube). Los vídeos de más de tres minutos tendrán penalización.

   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). OBS Studio[9] está disponible para varias plataformas (Linux, Windows, MacOS). 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.

[9] OBS Studio: https://obsproject.com/

Sobre la entrega del repositorio:

- Se han de entregar los siguientes ficheros:

  - El repositorio en la instancia GitLab de la ETSIT deberá tener, en su directorio raíz, un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos. Deberá poder ejecutarse directamente con `python3 manage.py runserver` desde un entorno virtual en el que esté instalado Django 4.1. También ejecutará los tests con `python3 manage.py test`, desde el mismo entorno virtual.
  - La base de datos habrá de tener datos suficientes como para poder probarlo. Estos datos incluirán al menos mensajes puestos desde dos navegadores, con al menos diez mensajes en total, en al menos tres salas.
  - Un fichero `requirements.txt`, con un nombre de paquete Python por línea, para indicar cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, si es que fuera el caso, incluyendo Django. Si es posible, se recomienda escribir este fichero en el formato que entiende `pip install -r requirements.txt`
  - Cualquier fichero auxiliar que pueda hacer falta para que funcione la práctica, si es que fuera el caso.

- Se incluirán en el fichero README.md los siguientes datos (atención a lo que ya se han indicado sobre la primera línea de este fichero):

  - Nombre y titulación.
  - Nombre de cuenta en laboratorios Linux de la ETSIT.
  - Nombre de cuenta de la URJC.
  - URL del vídeo demostración de la funcionalidad básica
  - URL del vídeo demostración de la funcionalidad optativa, si se ha realizado funcionalidad optativa
  - URL de la aplicación desplegada
  - Al menos una contraseña que permita autenticarse para poder usar la aplicación.
  - Cuenta del superusuario para entrar en el Admin Site.
  - 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.

Estos datos se escribirán siguiendo estrictamente el siguiente formato:

```
# Entrega practica

## Datos

* Nombre:
* Titulación:
* Cuenta en laboratorios:
* Cuenta URJC:
* Video básico (url):
* Video parte opcional (url):
* Despliegue (url):
* Contraseñas:
* Cuenta Admin Site: usuario/contraseña

## Resumen parte obligatoria

## Lista partes opcionales

* Nombre parte:
* Nombre parte:
* ...
```

Asegúrate de que el fichero esté adecuadamente formateado en Markdown.

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 que se ha usado en prácticas.

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

8.6. Preguntas frecuentes

A continuación, algunas preguntas relacionadas con el enunciado de esta práctica, junto con sus respuestas:

- ¿Cómo puedo construir la "Página dinámica de cada sala"?

  Se hará apoyándose en el recurso "Sala en formato JSON", que ha de existir para cada sala. El recurso que sirva la página dinámica de cada sala utilizará código JavaScript para descargar periódicamente (por ejemplo cada 30 segundos) el documento en formato JSON, y actualizar con él un elemento `div`, en el que se mostrarán todos los mensajes que haya en ese documento JSON. Puede verse un ejemplo en el directorio `Ajax/messages` del repositorio de código de la asignatura. Puede probarse el contenido de ese directorio lanzando el servidor Python de una línea en él:

  ```
  python3 -m http.server
  ```

  y a continuación, apuntando el navegador a la URL que sirve.

  En general, bastará con utilizar el mismo fichero `messages.json` que puede encontrarse en ese directorio, referenciándolo en el documento HTML que sirva el recurso de la página dinámica de cada sala. Para que funcione, ha de haber un elemento `div` en la página, que será en el que el script JavaScript incluirá los mensajes del documento JSON. La URL del documento JSON de cada página está en el propio HTML, y podrá generarse dinámicamente con la plantilla de la página en cuestión.

- Cuando despliego mi práctica en Python Anywhere, algunos recursos reconocidos no me funcionan, pero otros (YouTube entre ellos), sí. Todo me funciona bien en mi versión local. ¿Qué está pasando?

  Las máquinas virtuales de Python Anywhere están limitadas en cuanto a los sitios a los que se pueden conectar: sólo se pueden conectar a aquellos que están en una cierta "lista blanca". Por eso, si el sitio al que tu programa se tiene que conectar para obtener recursos no está en la lista blanca, no va a poder descargarse el documento XML o JSON de esos recursos. Para evitar problemas, en el caso de despliegue en Python Anywhere pedimos que funcionen bien los recursos reconocidos que están en la lista blanca, y para los demás, que tengan recursos en la base de datos de despliegue. Más detalles en el apartado sobre despliegue de este enunciado (8.2).

- En la pagina de información que se menciona en el enunciado, ¿qué hay que incluir en el apartado de documentación?

  Casi que lo que queráis, lo importante es tener la página. Puede ser por ejemplo un resumen de un párrafo de lo que hace la aplicación.

- ¿Es necesario utilizar los mecanismos provistos por Django para el control de sesiones y autenticación?

  En principio, esa es la solución recomendada. El principal problema suele ser asegurarse de que cualquier mecanismo alternativo funciona al menos tan bien como el de Django, lo que no es en general trivial. De todas formas, salvo muy buenos motivos, la aplicación es una aplicación Django, y por lo tanto cuantas más facilidades de Django se usen (bien usadas), mejor.

- ¿Dónde puedo realizar el despliegue de la aplicación?

  El despliegue puede realizarse en cualquier ordenador que esté conectado permanentemente a Internet durante el periodo de corrección, en una dirección accesible desde cualquier navegador conectado a su vez a Internet. Esto puede ser por ejemplo un ordenador personal en un domicilio con acceso permanente a Internet, adecuadamente configurado (puede ser una Raspberry Pi o similar, si se busca una solución simple y de bajo coste). También puede ser un servicio en Internet, por ejemplo uno gratuito como los que ofrecen Google (instrucciones[10], precios[11]), o PythonAnywhere (instrucciones[12], precios[13]). Los profesores podremos ayudar de forma más detallada con PythonAnywhere.

- Algunos documentos no se descargan de PythonAnywhere. ¿Qué está pasando?

  Algunos documentos estáticos (por ejemplo, la hora de estilo CSS que estoy usando) no se carga en el navegador cuando despliego la práctica en PythonAnywhere. Sin embargo, cuando pruebo en mi ordenador, o en los ordenadores del laboratorio, todo parece ir bien. ¿Qué está pasando.

  Lo que ocurre es que para servir los ficheros estáticos (los que no genera tu aplicación Django, sino que simplemente los sirve a partir de ficheros ya existentes), hay que indicarle a PythonAnywhere qué ficheros son esos, y para qué recursos deben servirse. Es muy típico que esto ocurra, por ejemplo, con el fichero que tenga la hoja de estilo CSS. Puedes ver esto en la consola web, donde indica:

[10] GCP Quickstart Using a Linux VM:
https://cloud.google.com/compute/docs/quickstart-linux

[11] Google Compute Engine Pricing:
https://cloud.google.com/compute/pricing

[12] Capítulo "Deploy!" de Django Girls Tutorial:
https://tutorial.djangogirls.org/en/deploy/

[13] PythonAnywhere Plans and Pricing:
https://www.pythonanywhere.com/pricing/

```
Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or
```

Añade en esa tabla tus ficheros, y si no hay más problemas te funcionará.

9. Prácticas de entrega voluntaria

9.1. Entrega de microprácticas y miniprácticas

Para la entrega de prácticas incrementales se utilizarán repositorios git públicos alojados en el GitLab de la ETSIT. Para cada práctica entregable los profesores abrirán un repositorio público en el proyecto CursosWeb [14], 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"). En el caso de las miniprácticas, el nombre comenzará por "Mini–", seguido de un número (el de orden de entrega), y el identificador del ejercicio. 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 el GitLab de la ETSIT, 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 GitLab 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 GitLab, visitando (tras haberse autenticado con su cuenta de usuario de GitLab 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 GitLab usando `git push`.

[14] https://gitlab.etsit.urjc.es/CursosWeb

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

```
[En GitLab: 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 cera el directorio copia_propia, copia local del repositorio propio]

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

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

9.2. Minipráctica 1 (entrega voluntaria)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/mini-1-acortadora

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs. Los usuarios de la aplicación podrán especificar URLs, y la aplicación generará para ellos un recurso aleatorio que, a partir de ese momento, redireccionará a la URL correspondiente. La aplicación podrá realizarse según el esquema de clases explicado en clase (usando, si se quiere, el módulo `webapp.py`), o de cualquier otra forma.

El código ha de guardarse en un fichero llamado *randomshort.py*.

El funcionamiento de la aplicación será el siguiente:

- Recurso "/", invocado mediante GET. Devolverá una página HTML con un formulario. En ese formulario habrá un campo para la url a acortar. El formulario se enviará al servidor mediante POST, y su campo se llamará `url`. Además, esa misma página incluirá un listado de todas las URLs reales y acortadas que ha configurado ese navegador hasta ese momento.

- Recurso "/", invocado mediante POST. Si el comando POST incluye una `qs` (query string) con un campo `url` se devolverá una página HTML con todas las URLs reales y acortadas que ha configurado ese navegador hasta ese momento (incluida esta), 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 "https://" 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 "https://gsyc.es".

Si se recibe una petición para un nombre de recurso que ya corresponde con una URL conocida por el acortdor, usará el mismo recurso aleatorio acortado que ya se le asignó.

Así, por ejemplo, si se quiere acortar `http://gsyc.urjc.es`, y la aplicación está en el puerto 1234 de la máquina "localhost", se invocará (mediante POST) la URL

```
https://localhost:1234/
```

y en el cuerpo de esa petición HTTP irá la `qs`

```
url=https://gsyc.urjc.es
```

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) una página HTML con el formulario, y la lista de URLs acortadas hasta el momento para este navegador, incluyendo esta.

- Recursos correspondientes a URLs acortadas. Estos serán los recursos aleatorios que se han ido construyendo para las urls a acortar, todos 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/sd34te
  ```

  y ese recurso aleatorio está configurado para redirigir a `http://gsyc.urjc.es`, la aplicación devolverá un HTTP REDIRECT a la URL

  ```
  https://gsyc.urjc.es
  ```

La aplicación funcionará con estado: se supone que cada vez que la aplicación muera y vuelva a ser lanzada, no perderá todo su estado anterior. Para ello, se

guardarán las URLs acortadas en un fichero con `shelve`. Al lanzar la aplicación, se leerá el fichero con las URLs acortadas. Y cada vez que se incluya una nueva URL acortada en el sistema, también se guardará esta información en el fichero.

Comentario

Se recomienda utilizar un diccionario para almacenar las URLs reales y los nombres de recurso correspondientes. La clave de búsqueda será el nombre de recurso, y el valor, la URL real.

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 a acortar).
- Reconocer "GET /recurso" (para cualquier recurso), y realizar la redirección correspondiente.
- Manejar las condiciones de error y realizar el resto de la funcionalidad.

9.3. Minipráctica 2 (entrega voluntaria)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/mini-2-acortadora

Esta práctica tendrá como objetivo la creación de una aplicación web (de nombre *acorta*) simple para acortar URLs utilizando Django (en un nuevo proyecto Django llamado *project*). Su enunciado será igual que el de la práctica 1 de entrega voluntaria (ejercicio 9.2), salvo en los siguientes aspectos:

- Se implementará utilizando Django.
- Se añadirá una nueva entrada en el formulario para elegir manualmente el recurso para la url acortada. Por lo tanto, en el recurso "/" invocado mediante POST, incluirá una qs que tenga campos `url` y `short`.

  Así, por ejemplo, si se quiere acortar `http://gsyc.urjc.es` con el nombre de recurso `gsyc`, y la aplicación está en el puerto 1234 de la máquina "localhost", se invocará (mediante POST) la URL

  `https://localhost:1234/`

  y en el cuerpo de esa petición HTTP irá la `qs`

  `url=https://gsyc.urjc.es&short=gsyc`

  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) una página HTML con el formulario, y la lista de URLs acortadas hasta el momento para este navegador, incluyendo esta.

- Además si el recurso para la url acortada está vacío, se deberá acortar mediante un número entero que irá aumentando un valor de manera secuencial.

  Así, por ejemplo, si se quiere acortar `http://gsyc.urjc.es` con el nombre de recurso vacío, y la aplicación está en el puerto 1234 de la máquina "localhost", se invocará (mediante POST) la URL

  `https://localhost:1234/`

  y en el cuerpo de esa petición HTTP irá la `qs`

  `url=https://gsyc.urjc.es&short=`

  Y se eligirá para acortar el número "1", por lo que la url acortada sería `https://localhost:1234/1`

  Seguidamente si se quiere acortar `http://www.urjc.es` con el nombre de recurso vacío de nuevo, se eligirá para acortar el número "2" por lo que la url acortada sería `https://localhost:1234/2`

  Así sucesivamente siempre y cuando el formulario para el nombre de recurso este vacío.

- 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.

- Utilizará plantillas, de manera que el código Python y el HTML estarán separados.

Repositorio de partida:
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/mini-2-acortadora

10. Ejercicios 01: Conceptos básicos de aplicaciones web

10.1. 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.

10.2. 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.

10.3. Espía a tu navegador (Firefox Developer Tools)

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 las "Firefox Developer Tools", que vienen ya preinstaladas en Firefox.

El ejercicio consiste en:

- Ojear las distintas herramientas de Firefox Developer Tools.
- Utilizarlas para ver la interacción HTTP al descargar una página web real.
- Utilizarlas 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 permiten hacer estas herramientas, mucho mejor.

Referencias

Sitio web de Firefox Developer Tools:
https://developer.mozilla.org/en/docs/Tools

10.4. Espía a tu navegador (Firebug)

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/

10.5. Explora tus cookies

Enunciado:

En este ejercicio vamos a ver las cookies que intercambia nuestro navegador con un servidor simple. El servidor que vamos a usar es `cookies-server-6.py` (en la carpeta `Python-Web/cookies`). Ejecuta el servidor, y luego, utilizando las herramientas de desarrollador de Firefox (ver ejercicio 10.3, observa las cookies que se intercambian entre este servidor y el navegador. En concreto, carga en el navegador la página principal del servidor, escribe algo en el formulario que te aparecerá, y contesta a este ejercicio escribiendo las cookies que observes en la interacción que se produce cuando le das al botón "Submit" para enviar al servidor el texto que has escrito.

Cuando lances el servidor, indicarás en qué puerto TCP escuchará (que tendrá que estar libre en la máquina donde se lance). Por ejemplo, para lanzarlo escuchando en el puerto 8000, usarás la línea:

```
python3 cookies-server-6.py -p 8000
```

En el navegador, tendrás que indicar la url correspondiente al puerto que hayas indicado. Por ejemplo, para acceder al servidor lanzado en el puerto 8000, la url será http://localhost:8000.

10.6. Explora tus cookies (2)

Enunciado:

Vamos a explorar las diferencias entre dos programas que tratan de "recordarte" lo último que escribiste en un formulario. Ambos son servidores HTTP, y están en el directorio `Python-Web/cookies` (en el repositorio de código de la asignatura), y son `cookies-server-8.py` y `cookies-server-9.py`. El ejercicio consiste en ejecutar cada uno de ellos de esta forma:

- Lanza el programa servidor que vas a probar.
- En el navegador, carga la página correspondiente al recurso principal de ese servidor.
- Borra las cookies que pueda haber para ese servidor.

- Recarga la página que tienes en el navegador
- En el formulario que tienes en la página, escribe "Primero", y envíalo al servidor. Llamaremos al resultado de este envío (la página que muestre el navegador al recibir la respuesta del servidor "Página 1".
- En el formulario que tienes ahora en la Página 1, escribe "Segundo", y vuelve a enviarlo al servidor. Llamaremos al resultado de este envío (la página que muestre el navegador al recibir la respuesta del servidor "Página 2".
- En el formulario que tienes ahora en la Página 2, escribe "Tercero", y vuelve a enviarlo al servidor. Llamaremos al resultado de este envío (la página que muestre el navegador al recibir la respuesta del servidor "Página 3".

Compara qué ves en Página 1, Página 2 y Página 3 en los dos casos (cuando lanzas cada uno de los dos servidores, y sigues el proceso con ellos). Trata de explicar lo que ocurre viendo en el navegador las cookies que envía el servidor en cada uno de los casos. A continuación, trata de explicarlo mirando el código de los dos servidores. Puedes utilizar la herramienta `diff` para ver las diferencias en el código de ambos, si eso te ayuda.

Escribe como respuesta:

- Las diferencias que observes entre Página 1, Página 2 y Página 3 (descritas, acompañadas de capturas de pantalla si lo ves útil).
- La explicación que hayas podido encontrar a las diferencias mirando las cookies en el navegador.
- La explicación que hayas podido encontrar a las diferencias mirando el código de los dos servidores.

10.7. Servidores que recuerdan

Enunciado:

En el directorio `Python-Web/cookies` (en el repositorio de código de la asignatura) puedes encontrar los programas `content-server-1.py` y `content-server-2.py`. Ambos utilizan cookies de datos para "recordar" el último texto que se introdujo en el formulario que proporciona el servidor. El primero, utiliza GET para enviar al servidor el contenido del formulario, y el segundo utiliza POST.

Este ejercicio consiste en entender el código de ambos programas, y escribir otros dos, `content-server-3.py` y `content-server-4.py`, que hagan lo mismo,

pero utilizando cookies de sesión (que identifican el navegador, y utilizan el identificador para buscar el último contenido del formulario en un diccionario que mantienen).

10.8. 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).

10.9. Última búsqueda: números aleatorios o consecutivos

Enunciado:

En el ejercicio "Última búsqueda" (ejercicio 10.2) 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

10.10. 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.

10.11. 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, Cookie Quick Manager para Firefox) 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

Cookie Quick Manager: https://addons.mozilla.org/en-US/firefox/addon/cookie-quick-manager/

10.12. Sumador simple con varios navegadores

Enunciado:

Igual que el ejercicio "Sumador simple" (17.5), 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" (17.5).

10.13. Sumador simple con varios navegadores intercalados

Enunciado:

Igual que"Sumador simple con varios navegadores" (10.12), 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.

10.14. Sumador simple con rearranques

Enunciado:

Igual que el ejercicio "Sumador simple con varios navegadores intercalados" (10.13), 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.

10.15. Contador simple

Enunciado:

Construir una aplicación web que funcione como contador inverso. Ofrecerá un recurso que, cuando sea invocado mediante un método GET, devolverá un número entero. La primera vez que se invoque, el número devuelto será un 5. Cuando se le invoque sucesivamente, el número obtenido se irá decrementando en uno (4, 3, 2...). Cuando se haya obtenido un 0, el siguiente número será de nuevo el 5 (esto es, el contador funciona como un contador inverso cíclico).

Comentario:

Es importante hacer énfasis en la solución en la estructura de la aplicación, tratando de estructurar el código de forma que las diferentes acciones que hará la aplicación web queden claras.

10.16. cURL básico

Enunciado:

Prueba el ejercicio "Contador simple" (10.15) con el programa `curl`. Utilizalo para ver el documento que se recibe de tu servidor, para ver las cabeceras que te envía, para ver tanto cabeceras (de ida y vuelta) como cabeceras...

Materiales:

- Understanding CURL and HTTP Headers (tutorial)
- cURL (sitio web)
- Everything curl (libro)

Solución:

```
curl -XGET http://localhost:1234/
curl -XGET -I http://localhost:1234/
curl -XGET -Iv http://localhost:1234/
```

10.17. Distinto contenido según navegador

Enunciado:

Realiza una versión del ejercicio "Contador simple" (10.15) que sirva distinto contenido según el navegador que lo pida. En particular, si el navegador que pide una página es Firefox, se responderá con una página HTML tal y como se hace en el ejercicio "Contador simple". Pero si la petición la hace `curl`, se responderá con un documento de texto plano sólo con el resultado.

Materiales:

- Cabecera User Agent (MDN): https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent

10.18. Depurador básico

Enunciado:

Prueba el ejercicio "Contador simple" (10.15) con el depurador de Python, por ejemplo, ejecutándolo desde PyCharm.

10.19. Contador simple con varios navegadores

Enunciado:

Igual que el ejercicio "Contador simple" (10.15), pero ahora puede haber varios navegadores invocando la aplicación web. Se supone que los navegadores no se interfieren (esto es, uno completa todas sus operaciones con el contador antes de que otro la empiece).

Comentario:

No hacen falta modificaciones al código del ejercicio "Contador simple" (10.15), si se asume que cuando se invoque el contador, éste puede empezar por cualquier número de su ciclo. Si por el contrario se quiere que comience como "la primera vez" (por 5) es preciso detectar que se está sirviendo a un nuevo navegador, y habrá que prever algún mecanismo al respecto.

Puede consultarse la implementación de referencia disponible en counter-server-1.py

10.20. Contador simple con varios navegadores intercalados

Enunciado:

Igual que "Contador simple con varios navegadores" (10.19), pero ahora un navegador puede comenzar a trabajar con el contador en cualquier momento, incluyendo momentos en los que otro navegador no haya terminado aún.

Comentarios:

En una primera versión, se implementa con una cookie simple que incluye el número que ha servido el contador de forma que la aplicación no tiene que almacenar el valor del contador para cada navegador.

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

En una tercera versión, se podría añadir una operación para crear un contador único para cada navegador, con un nombre de recurso propio. Cada navegador conocería su recurso, y sólo utilizaría ese. Se puede evitar que un navegador utilice un recurso que no le corresponde haciendo que su nombre no sea fácilmente descubrible.

Puede consultarse la implementación de referencia disponible en counter-server-2.py

10.21. Contador simple con rearranques

Repositorio plantilla (para entrega):
`https://gitlab.etsit.urjc.es/cursosweb/2022-2023/contador-simple-rearranques`

Enunciado:

Igual que el ejercicio "Contador simple con varios navegadores intercalados" (10.20), pero ahora desde que el navegador inicia el trabajo con el contador 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.

Puede consultarse la implementación de referencia disponible en counter-server-3.py y counter-server-4.py.

10.22. 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.

10.23. Trackers en páginas web

Instala algún *plug-in* similar a Lightbeam para tu navegador. Este *plug-in* permite detectar todos los sitios web que se acceden al descargar una página, incluyendo los forzados por "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 Lightbeam (sin mantenimiento, no funciona en versiones modernas de navegadores):
  https://www.mozilla.org/lightbeam/
- GreenBeam para Firefox:
  https://addons.mozilla.org/en-US/firefox/addon/greenbeam/
- Thunderbeam-Lightbeam para Chrome:
  https://chrome.google.com/webstore/detail/thunderbeam-lightbeam-for/hjkajeglckopdkbggdiajobpilgccgnj

10.24. Trackers en páginas web (Ghostery)

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/

10.25. Protección contra trackers en el navegador

Estudia las capacidades de protección contra trackers de tu navegador. Por ejemplo, en Firefox, están accesibles a partir del "escudo" que aparece junto a la barra de enlaces. Trata de entenderlas en el contexto de lo que has aprendido sobre cookies y otras formas de trazar historiales de navegación.

Referencias

Información sobre protección ante trackers de Firefox:
https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop

10.26. Transplante de cookies

En el ejercicio "Explora tus cookies (2)" (ejercicio 10.6) hemos explorado cómo se comportan las cookies con dos aplicaciones web simples. En este ejercicio se te pide que elijas una cualquiera de estas dos aplicaciones, y pruebes a transplantar las cookies que tenga un navegador después de haberla usado, a otro navegador diferente. Por ejemplo, puedes transplantarlas de Chrome a Firefox, o de un navegador que tengas en tu ordenador a un Firefox en un ordenador del laboratorio. Para responder al ejercicio, explica cómo has hecho para transplantar las cookies, y qué has observado al acceder a la aplicación desde el navegador al que las has transplantado.

10.27. Transplante de cookies 2

En el ejercicio "Contador Simple" (ejercicio 10.15) hemos explorado cómo realizar un contador general, que descuenta según cada petición recibida. En este ejercicio se pide modificar el código para que el contador sea particular de cada navegador (con una cookie). De esta manera, cada navegador contará con un contador particular, que sólo se decrementará con cada visita.

Finalmente, se puede que se pruebe a transplantar las cookies que tenga un navegador después de haberla usado, a otro navegador diferente. Por ejemplo,

puedes transplantarlas de Chrome a Firefox, o de un navegador que tengas en tu ordenador a un Firefox en un ordenador del laboratorio.

11. Ejercicios 02: Servicios web que interoperan

11.1. 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.

11.2. 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.

Mientas, 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 19.2). 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.

11.3. Lista de la compra

Enunciado:

Vamos a diseñar una API HTTP para un servicio que permite guardar una lista de la compra. Supongamos que esta lista está compuesta únicamente por los items que quiero comprar en un momento dado (zanahorias, yogures, etc.) y un número natural para cada item que tengo, que expresa la cantidad que quiero comprar. Por ejemplo, en un momento dado, la lista podría ser:

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

Puede haber items en la lista de la compra con valor 0, si se encuentra que es útil por algún motivo.

Las operaciones que permitirá la API del servicio son: consultar la lista, añadir un item (con su correspondiente cantidad) a la lista, modificar la cantidad de un item en la lista, y borrar un item de la lista.

Se pide diseñar una API HTTP para esta aplicación (servicio) web, identificando cuáles son los recursos relevantes, las operaciones HTTP válidas sobre ellas, y describiendo la semántica de cada una de ellas.

Podemos imaginar que el servicio web va a ser usado, por ejemplo, desde una aplicación en el móvil, que podrá hacer GET, PUT, POST y DELETE (usando HTTP). Cada vez que quiero apuntar o borrar algo de la lista de la compra, o quiero consultar la lista, utilizo la app del móvil para acceder, mediante HTTP, al servicio de lista de la compra.

11.4. Listado de lo que tengo en la nevera

Enunciado:

Este es un ejercicio muy similar a "Lista de la compra" (11.3). Vamos a diseñar una API REST para una aplicación web que mantenga la lista de lo que tengo en la nevera. El tipo de lista será el mismo que se indica para en el ejercicio de la lista de la compra, pero ahora trataremos de que la API cumpla los principios REST.

Comentarios:

Hay muchas soluciones posibles para este problema, que sobre todo pretende que se reflexione sobre las características que hacen de una interfaz HTTP una interfaz REST. Pero en cualquier caso, como mínimo hay que definir cuáles serán los recursos (y sus nombres), las operaciones sobre cada uno de ellos, y un breve comentario sobre su funcionamiento.

Una posible solución sería:

Recurso	Método	Descripción
/	GET	Lista de items en la nevera (enlaces a recursos)
	POST	Crea un nuevo item, `item=xx&cantidad=yy`
/[item]	GET	Obten el valor (número) del alimento "item"
	PUT	Actualiza el valor del alimento "item"
	DELETE	Borra el item (elimina el recurso)

Esta interfaz HTTP podría usarse desde la aplicación como se ve en los siguiente ejemplos.

La primera vez que se introducen zanahorias (supongamos que se introducen 5):

- Petición (para ver si hay zanahorias):

```
GET / HTTP/1.1
```

- Respuesta:

```
HTTP/1.1 200 OK

<a href="/leche"></a>
<a href="/chorizo"></a>
```

- Petición (para crear el recurso para las zanahorias):

```
POST / HTTP/1.1

item=zanahorias&cantidad=5
```

- Respuesta:

```
HTTP/1.1 200 OK

<a href="/chorizo"></a>
```

Si sacamos 3 zanahorias:

- Petición (para ver cuántas zanahorias hay):

  ```
  GET /zanahorias HTTP/1.1
  ```

- Respuesta:

  ```
  HTTP/1.1 200 OK

  5
  ```

- Petición (para actualizar al nuevo número de zanahorias):

  ```
  PUT /zanahorias HTTP/1.1

  2
  ```

- Respuesta:

  ```
  HTTP/1.1 200 OK

  2
  ```

Si sacamos otras 2 zanahorias:

- Petición (para ver cuántas zanahorias hay):

  ```
  GET /zanahorias HTTP/1.1
  ```

- Respuesta:

  ```
  HTTP/1.1 200 OK

  2
  ```

- Petición (para eliminar el recurso, porque quedaría a cero):

  ```
  DELETE /zanahorias HTTP/1.1
  ```

- Respuesta:

  ```
  HTTP/1.1 200 OK
  ```

11.5. Sumador simple versión REST

Enunciado:

Desarrollar una versión RESTful de "Sumador simple" (ejercicio 17.5). ¿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).

11.6. 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 11.5).

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 le 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.)

11.7. Calculadora simple versión REST (Django)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/calculadora-simple-rest-django

Enunciado:

Realizar el ejercicio 11.6 usando Django.

Comentarios:

Ver los comentarios del ejercicio 11.6.

11.8. 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/gsyc.es/ devuelve el contenido de la página http://gsyc.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.

11.9. Cache de contenidos versión Django

Enunciado:

Construir una aplicación que implemente una cache de contenidos, como la descrita en el ejercicio 11.8, pero sobre Django. Puede usarse como base "Django cms" (ejercicio 18.5), y si se quiere, el módulo estándar de Python "urllib".

11.10. Cache de contenidos anotado

Enunciado:

Construir una aplicación como "Cache de contenidos" (ejercicio 11.8), 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/gsyc.es" sería el recurso para recargar la página "http://gsyc.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.

11.11. Gestor de contenidos miltilingüe versión REST

Enunciado:

Diseño y construcción de "Gestor de contenidos miltilingüe 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.

11.12. 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.

11.13. 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 11.11) 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"

11.14. 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 11.11 y 11.13) 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.

11.15. 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

12. Ejercicios 03: Modelo-Vista-Controlador

12.1. Red social muy simple

Enunciado:

Vamos a construir una red social muy simple como aplicación web. En esta red, los usuarios podrán poner tantos comentarios como quieran. Estos comentarios quedarán almacenados a su nombre (esto es, cada comentario estará relacionado con el usuario que lo puso). Se supone que lo usuarios ya están autenticados por algún mecanismo que no es relevante para este ejercicio. Por lo tanto, cada navegador, cuando pone un comentario, ya lo pone a nombre de un usuario.

La funcionalidad de la aplicación web es la siguiente:

- En una cierta página podrán verse todos los comentarios que ha puesto cualquier usuario
- En una cierta página, diferente para cada usuario, podrán verse todos los comentarios de ese usuario.
- Los usuarios podrán poner, mediante un formulario, tantos comentarios como quieran (de uno en uno).

Supongamos que vamos a implementar esta aplicación sobre Django. Se pide:

- Descripción de la interfaz HTTP de la aplicación, siguiendo en la medida de lo posible los principios REST.
- Descripción del modelo de datos (si es posible, escribiendo el fichero `models.py` correspondiente).
- Descripción de las plantillas (templates) Django que se podrían utilizar. Para cada una describir qué mostraría la plantilla, y qué datos habría que utilizar para "renderizarla".
- Descripción de las vistas (contenido de `views.py`), en términos de qué interacción con la base de datos y con las plantillas habría que realizar en cada una de ellas, y qué relación con los recursos de la interfaz HTTP tendrían.
- Fichero `urls.py` que relacione los recursos de la interfaz HTTP con las vistas.

13. Ejercicios 04: Introducción a XML

Ejercicios sobre XML, JSON, y HTML (reconocedores, generadores, etc).

13.1. 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.

13.2. Chistes XML (parser DOM)

Enunciado:

Crea un programa que funcione como `xml-parser-jokes.py`, pero usando el parser DOM simple de Python, `xml.dom.minidom`.

Material:

- `jokes.xml`. Fichero XML con descripciones de chistes.
- `xml-dom-jokes.py`. Ejemplo de solución de este ejercicio.
- "A Roadmap to XML Parsers in Python"
  https://realpython.com/python-xml-parser/

13.3. 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.

13.4. Titulares de BarraPunto

Enunciado:

Descargar el fichero RSS de BarraPunto[15], 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:

- 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.

Repositorio de entrega en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/xml-barrapunto

13.5. Videos en canal de YouTube

Enunciado:

Descargar el fichero RSS con los videos del canal CursosWeb de Youtube[16], y construir un programa que produzca como salida sus títulos en una página HTML. Si se carga esa página en un navegador, picando sobre un titular, el navegador deberá cargar la página de Youtube con el video correspondiente. Como base puede usarse lo aprendido estudiando el programa `xml-parser-jokes.py`.

Ejemplo de ejecución:

Al ser ejecutado el programa, producirá la página HTML descrita anteriormente. Por lo tanto, podemos redirigir la salida estándar el programa a un fichero, que podrá ser visualizado mediante un navegador:

```
programa > pagina.html
```

Material:

[15] http://barrapunto.com
[16] https://www.youtube.com/channel/UC300utwSVAYOoRLEqmsprfg

- ytparser.py: Programa que muestre en pantalla los nombres y los enlaces de los videos de un fichero con el documento XML de un canal de YouTube, en formato HTML.

- youtube.xml: Fichero con un documento XML que describe un canal de YouTube, que se puede usar con el programa `ytparser.py`.

- https://www.youtube.com/feeds/videos.xml?channel_id=UC300utwSVAYOoRLEqmsprfg: URL del documento XML del canal CursosWeb de Youtube. El fichero anterior se generó descargando este documento. Si quieres, puedes descargarlo y probar tu programa también con él, debería funcionar igual que con `youtube.xml`.

13.6. Videos en canal de YouTube (con descarga)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/xml-youtube-descarga

Enunciado:

Realizar un programa con la funcionalidad descrita en "Videos en canal de Youtube" (ejercicio 13.5), pero realizanda la descarga del canal desde YouTube. El programa admitirá un único argumento, que será un identificador de canal de YouTube, y producirá como salida estándar la página HTML que producía el ejercicio mencionado anteriormente.

Ejemplo de ejecución:

Al ser ejecutado el programa, producirá la página HTML descrita anteriormente. Por lo tanto, podemos redirigir la salida estándar el programa a un fichero, que podrá ser visualizado mediante un navegador:

```
programa UC300utwSVAYOoRLEqmsprfg > pagina.html
```

Material:

- Módulo urllib de Python:
  https://docs.python.org/3/library/urllib.html

- Tutorial sobre urllib de Python:
  https://pythonspot.com/urllib-tutorial-python-3/

13.7. Gestor de contenidos con titulares de BarraPunto

Enunciado:

Partiendo de contentApp ("Gestor de contenidos", ejercicio 19.1), 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.

13.8. Gestor de contenidos con titulares de BarraPunto versión SQL

Enunciado:

Realiza contentDBAppBarraPuntoSQL, con la misma funcionalidad que contentAppBarraPunto (ejercicio 13.7), pero usando una base de datos SQLite en lugar de un diccionario persistente gestionado con Shelve.

13.9. Gestor de contenidos con titulares de BarraPunto versión Django

Enunciado:

Realiza una aplicación Django con la misma funcionalidad que "Django cms" (ejercicio 18.5), 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 18.5) o "Django cms_put" (ejercicio 18.6).

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).

Repositorio de entrega en GitLub:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-cms-barrapunto

13.10. Gestor de contenidos con videos de YouTube (simple)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-youtube

Enunciado:

Realiza una aplicación Django que utilice parte de la funcionalidad de "Django cms" (ejercicio 18.5) para construir un archivador de videos de un canal Youtube. Para ello, en el recurso "/", la aplicación ofrecerá, como documento HTML, dos listados:

- Listado de videos seleccionados
- Listado de videos no seleccionados (seleccionables)

Cualquier otro recurso devolverá una página de error.

El listado de videos seleccionables incluirá un listado de todos los videos del canal, junto con un botón "Seleccionar" a su lado. Si se pulsa el botón de "Seleccionar" para un video, se añadirá éste al listado de videos seleccionados.

El listado de videos seleccionados, que estará vacío inicialmente, incluirá los videos que hayan sido seleccionados, según se indica anteriormente. Junto a cada video, habrá un botón "Eliminar", que quitará el video del listado de videos seleccionados.

Un video dado aparecerá sólo en uno de los dos listados.

Para cada video, aparecerá su título, y un enlace al video en cuestión, tanto en el listado de seleccionables como de seleccionados.

La aplicación funcionará cargando el listado del canal cuando arranque, a partir del listado XML de ese canal. Puede usarse el canal "CursosWeb" como canal con el que funcionará la aplicación:

- HTML:
  https://www.youtube.com/channel/UC300utwSVAYOoRLEqmsprfg
- XML:
  https://www.youtube.com/feeds/videos.xml?channel_id=
  UC300utwSVAYOoRLEqmsprfg

Comentarios:

La descarga del documento XML con los contenidos del canal debería hacerse una vez. Esto debería hacerse en el momento en que la aplicación arranca. Una solución elegante para realizarlo sería utilizar el enganche (hook) `AppConfig.ready`.

Para usarlo, habría que escribir código siguiendo el siguiente esquema. En la app que esté implementando la solución al ejercicio (llamémosla `myapp`) escribiríamos un fichero `myapp/apps.py` con código de este estilo:

```
from django.apps import AppConfig
class MyAppConfig(AppConfig):
    name = 'myapp'
    def ready(self):
        url = 'https://www.youtube.com/feeds/videos.xml?channel_id=' \
            + sys.argv[1]
        xmlStream = urllib.request.urlopen(url)
```

Y luego, para que este código se ejecture, en el fichero `myapp/__init__.py`:

```
default_app_config = 'myapp.apps.MyAppConfig'
```

Si se hace la inicialización de esta manera, hay que tener en cuenta que no se podrá inicializar la base de datos desde el código en `ready`, por varios motivos (consultar la documentación sobre el enganche para ver detalles). Pero pueden usarse listas o diccionarios, dado que si la aplicación vuelve a inicializarse, se volverán a recoger los datos del canal, y dado el enunciado, no se require más persistencia.

De todas formas, si se quieren almacenar los datos en base de datos, alternativamente al mecanismo anterior se puede inicializar mediante migraciones.

Documentación:

- Appconfig.ready:
  https://docs.djangoproject.com/en/stable/ref/applications/
  #django.apps.AppConfig.ready
- Migraciones para inicializar datos:
  https://docs.djangoproject.com/en/3.0/howto/initial-data/
  #providing-initial-data-with-migrations

13.11. Gestor de contenidos con videos de YouTube (2)

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-youtube

Enunciado:

Vamos a extender la aplicación desarrollada en el ejercicio "Gestor de contenidos con videos de Youtube" (ejercicio 13.10), con la funcionalidad que se detalla a continuación.

Además del recurso "/", esta aplicación servirá recursos de la forma "/[id]", siendo "[id]" el identificador de un video seleccionado en la página principal (y sólo si no ha sido eliminado). Como identificador utilizaremos el que aparece en el elemento "yt:videoId" del documento XML que describe un canal. En cada uno de estos recursos se servirá una página HMTL con el siguiente contenido:

- Enlace a la página principal de la aplicación
- Título del video (que será un enlace a la url del video)
- Imagen del video (obtenida del elemento "media:thumbnail" del documento XML que describe el canal (que será un enlace a la url del video)
- Nombre del canal (que será un enlace a la url del canal)
- Fecha de publicación del video
- Descripción del video

El recurso "/", en el listado de videos seleccionados, en lugar de incluir un enlace al video en Youtube ofrecerá un enlace a la página del video en la aplicación, tal y como se ha indicado anteriormente.

13.12. Gestor de contenidos con videos de YouTube (tests)

Enunciado:

En este ejercicio, tienes que añadir tests al programa desarrollado para responder al ejercicio "Gestor de contenidos con videos de Youtube (2)" (ejercicio 13.11). Los tests deberán ejecutarse "a la manera de Django", usando `manage.py test`. Al menos, habrá que creat tests para:

- El parser que uses para extraer los datos del documento XML. Estos tests leerán uno o varios ficheros XML con el formato correcto, y comprobarán que el parser extrase los datos deseados.

- Una función auxiliar de cualquiera de las views. Si no tenías ninguna, crea al menos una función auxiliar para cualquier detalle de lo que haga una de las views, de forma que la puedas comprobar con un test.

- Cada uno de los tipos de recurso que atienda la aplicación. Si un recurso se atiende con GET y POST, harán falta al menos dos tests, uno para cada una de ellos.

13.13. Gestor de contenidos con videos de YouTube (despliegue)

Enunciado:

En este ejercicio, tienes que desplegar en PythonAnywhere[17] la implementación de la práctica "Gestor de contenidos con videos de Youtube (tests)" (ejercicio 13.12), o alguna otra de las relacionadas con este ejercicio.

PythonAnywyere ofrece un plan gartuito ("Beginner"), que proporciona recursos suficientes para realizar este ejercicio.

Materiales:

- Proyecto Django `django-youtube-4`, disponible en el repositorio de código de la asignatura. Incluye un fihero `README.md` con indicaciones detalladas sobre el despliegue en PythonAnywhere.

- "PythonAnywhere Help Pages":
  https://help.pythonanywhere.com/pages/

- "Deploying a web app on PythonAnywheere":
  https://www.pythonanywhere.com/task_helpers/start/4-deploy-local-web-app/

- "Deploying an existing Django project on PythonAnywhere":
  https://help.pythonanywhere.com/pages/DeployExistingDjangoProject

- Capítulo "Deploy!" del curso de Django de Django Girls:
  https://tutorial.djangogirls.org/en/deploy/

13.14. Municipios JSON

Enunciado:

Crea un programa Python que lea el fichero `municipios.json`, y muestra en pantalla el nombre de cada municipio y su id (campo "url"). El fichero

[17] https://pythonanywhere.com

`municipios.json` contiene una lista de diccionarios, uno por municipio, con varios campos.

Materiales:

Estos materiales pueden encontrarse en el directorio `Python-JSON` del repositorio de código de la asignatura.

- Fichero: `municipios.json`
  Originalmente, este fichero fue recogido de:
  https://opendata.aemet.es/opendata/api/maestro/municipios/?api_key=XXX
- Solución de referencia: `json-municipios.py`

13.15. Municipios JSON via HTTP

Enunciado:

Igual que "Municipios JSON" (ejercicio 13.14), pero recogiendo el documento JSON de la red, vía HTTP.

Materiales:

Estos materiales pueden encontrarse en el directorio `Python-JSON` del repositorio de código de la asignatura.

- Documento JSON con los municipios:
  https://raw.githubusercontent.com/CursosWeb/Code/master/Python-JSON/municipios.json
- Solución de referencia: `json-municipios-http.py`

13.16. Forks de un repositorio GitLab

Enunciado:

Escribe un programa en Python que lea la lista de forks de un repositorio de un sitio GitLab, y escriba un fichero JSON con la lista de urls de los forks.

Ejemplo de ejecución:

```
gitlab-forks.py gitlab.etsit.urjc.es 2799 <token>
```

El resultado será algo como:

```
[
  "https://gitlab.etsit.urjc.es/xxx/youtube-descarga.git",
  "https://gitlab.etsit.urjc.es/yyy/youtube-descarga.git",
  "https://gitlab.etsit.urjc.es/zzz/youtube-descarga.git"
]
```

Materiales:

- Documentación sobre la API de GitLab, apartado sobre forks de un repositorio:
  https://docs.gitlab.com/ee/api/projects.html#list-forks-of-a-project

- Ejemplo de acceso a lista de proyectos de GitLab:

```
curl "https://gitlab.com/api/v4/projects" | jq
```

- Ejemplo de acceso a la lista de forks de un repositorio:

```
curl --header "PRIVATE-TOKEN: <token>"
  "https://gitlab.etsit.urjc.es/api/v4/projects/2799/forks"
  | jq | grep http_url_to_repo
```

- "How to fetch Internet resources using the urllib package" (documentación de Python):
  https://docs.python.org/3/howto/urllib2.html

13.17. Extractor de información de un documento HTML

Enunciado:

Realiza un programa en Python que, dada la URL de un documento HTML, lo descarga, lo analiza, y muestra en pantalla parte de su contenido. En particular, ha de mostrar:

- Si hay disponibles propiedades Open Graph[18], se extraerán las propiedades `og:title` y `image` (o la que exista de ellas), y se mostrarán.

  Las propiedades Open Graph normalmente se encuentran como metadatos en la cabecera (`head`) del documento HTML. Por ejemplo:

```
<html>
  <head>
  <title>Este es el titulo</title>
  <meta property="og:title" content="Este es el titulo" />
  <meta property="og:image" content="https://..../imagen.jpg" />
  ...
</head>
...
```

[18] https://ogp.me/

- Si hay disponible un elemento `title`, se mostrará su contenido.

Comentarios:

Esta práctica puede realizarse usando el módulo `html.parser` de la bilbioteca estándar de Python, el módulo `BeautifulSoup4`[19] disponible en pypi (instalable con pip), o cualquier otro módulo de reconocimiento de HTML.

14. Ejercicios 05: Hojas de estilo CSS

14.1. Django cms_css simple

Enunciado:

Crea una hoja de estilo en la URL "/main.css" para manejar la apariencia de la página "/about" en "Django cms_put" (ejercicio 18.6). La hoja tendrá el siguiente contenido:

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

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

Explica en el fichero 'README.md' del repositorio de entrega cómo has solucionado la práctica.

Repositorio de entrega en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-cms-css

14.2. Django cms_css elaborado

Enunciado:

Modifica tu solución para "Django cms_put" (ejercicio 18.6) 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).

[19] https://www.crummy.com/software/BeautifulSoup/

- 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.

Repositorio de entrega en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-cms-css-2

14.3. Django cms_bootstrap cuadrícula

Enunciado:
Modifica tu solución para "Django css simple" (ejercicio 14.1) de forma que:

- Utilice Bootstrap (mediante CDN)
- El contenido se muestre en una rejilla (grid), con tres columnas:
  1. una con la llave,
  2. otra con el contenido, y
  3. una tercera con un enlace a la página para modificar el contenido

Repositorio de entrega en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/django-cms-cuadricula

14.4. Django cms_bootstrap componentes

Enunciado:
Modifica tu solución para "Django cms_bootstrap cuadrícula" (ejercicio 14.3) de forma que:

- Se añada una barra de navegación arriba, con un enlace a "home" y otras URLs a tu elección.
- Utilice el componente de tarjeta (*card*) para los contenidos de las rejillas

14.5. Django cms_bootstrap componentes personalizados

Enunciado:

Modifica tu solución para "Django cms_bootstrap componentes" (ejercicio 14.4) de forma que utilice uno de los siguientes componentes personalizados de Bootstrap (añadiendo donde fuera necesario imágenes, algunas de ellas servidas como ficheros estáticos desde Django):

- Carousel: https://getbootstrap.com/docs/5.0/examples/carousel/
- Jumbotron: https://getbootstrap.com/docs/5.0/examples/jumbotron/
- Album: https://getbootstrap.com/docs/5.0/examples/album/

Repositorio de entrega en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/django-cms-bootstrap

15. Ejercicios 06: AJAX

Ejercicios con AJAX y tecnologías relacionadas.

15.1. 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.

15.2. Ajax Sentences generator

Enunciado:

Construye una aplicación con funcionalidad similar a "SPA Sentences generator" (ejercicio 15.1), 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).

15.3. 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

15.4. Gadget de Google en Django cms

Enunciado:

Crear una versión del gestor de contenidos Django (Django cms, ejercicio 18.5) con un gadget de Google en cada página (el mismo en todas ellas).

15.5. 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

15.6. EyeOS

Enunciado:
Abrir una cuenta en el sitio de EyeOS, y visitar el entorno que proporciona.
Referencia:
http://www.eyeos.org/

16. 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).

16.1. 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`.

16.2. Subida de programa a repo de GitLab

Enunciado:

Crea un repositorio en GitLab, mediante fork del repositorio plantilla indicado en la sección de materiales, más abajo. Cuando esté creado, clónalo localmente, y haz en él un programa Python que calcule los factoriales de los números entre el 1 y 10 (incluidos), y los muestre en pantalla. A continuación, sincroniza el repositorio con el repositorio remoto en GitLab, y asegúrate de que ha quedado adecuadamente subido.

Materiales:

- Repositorio plantilla:
  https://gitlab.eif.urjc.es/cursosweb/repodeprueba

16.3. Asistente de IA generativa para PyCharm

Enunciado:

Instala el plugin de PyCharm "Sourcegraph Cody". Configúralo y pruébalo.
Puedes probar también otros plugins con funcionalidad similar, como Codeium.

Materiales:

- Información sobre el plugin Cody:
  https://plugins.jetbrains.com/plugin/9682-sourcegraph-cody--code-search
- Código fuente de Cody:
  https://github.com/sourcegraph/sourcegraph/tree/main/client/jetbrains
- Información sobre el plugin Codeium:
  https://plugins.jetbrains.com/plugin/20540-codeium-ai-autocomplete-and-chat-f
- Sitio web de Codeium:
  https://codeium.com/
- Código fuente de Codeium:
  https://github.com/Exafunction/codeium.vim

16.4. 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).

16.5. 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
```

16.6. 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.

Puedes partir del siguiente repositorio: `https://github.com/CursosWeb/X-Serv-Python-Fiche`

16.7. 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?

Puedes partir del siguiente repositorio: `https://github.com/CursosWeb/X-Serv-Python-FichD`

16.8. Calculadora

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2023-2024/calculadora

Enunciado:

Crea un programa que haga funciones de calculadora simple. El programa tendrá dos funciones (sumar, restar). Cada función aceptará dos parámetros, y devolverá su suma o su diferencia. También tendrá un programa principal que llamará a esas funciones para sumar primero 1 y 2, y luego 3 y 4, mostrando en cada caso el resultado en pantalla. A continuación restará primero 5 de 6, y luego 7 de 8, mostrando también los resultados en pantalla. Entrega la práctica en el GitLab de la ETSIT, utilizando el repositorio de plantilla, como sigue:

1. (Navegador) Entra en el GitLab de la ETSIT con tus credenciales de la URJC.
2. (Navegador) Haz un *fork* del repositorio de plantilla. Esto creará una copia del repositorio de la que tú serás dueño (como se puede comprobar a través de la URL, que ya no contendrá CursosWeb sino tu nombre de usuario).
3. (PyCharm) Clona tu repositorio, para tener una copia local.
4. (PyCharm) Crea en el directorio (proyecto) que has creado localmente el programa `calc.py`, con la funcionalidad que se pide más arriba.
5. (PyCharm) Realiza un "commit" con los cambios, incluyendo este nuevo fichero
6. (PyCharm) Haz un "push" para sincronizar tu repositorio en local con el tuyo en GitLab
7. (Navegador) Comprueba que tu repositorio en GitLab que se ha sincronizado correctamente (esto es, contiene el fichero tal y como lo has creado)

16.9. Descarga de documentos web

Enunciado:

Crea un fichero Python con clases que ayuden en la descarga de documentos web. Todas las clases estarán en el mismo fichero, que tendrá también un programa principal (que se ejecutará sólo cuando el fichero sea ejecutado directamente) que probará las clases.

Las clases serán las siguientes:

- Clase `Robot`. Proporcionará un "robot" que se encarga de descargar, y en su caso mostrar, un documento web, dada su url. Cumplirá la siguiente especificación:
  - Se instanciará indicando como argumento la url del documento del que se ocupará el robot.
  - Tendrá un método `retrieve`, sin argumentos, que se encargará de descargar el documento de la url de la que se ocupa el robot, sólo si no se lo ha descargado ya antes. Si se lo descarga, mostrará un mensaje en pantalla: "Descargando url" (siendo "url" la url en cuestión).
  - Tendra un método `show`, sin argumentos, que mostrará en pantalla el contenido del documento descargado. Para poder mostrarlo, `show` se encargará de usar `retrieve` cuando sea conveniente.
  - Tendrá un método `content`, sin argumentos, que devolverá una cadena de caracteres (*string*) con el contenido del documento descargado.
- Clase `Cache`. Proporcionará una cache de documentos, y cumplirá la siguiente especificación:
  - No aceptará ningún argumento para ser instanciada.
  - Tendrá un método `retrieve`, con una url como argumento, que se encargará de descargar el documento correspondiente a esa url, si no ha sido ya descargado antes, y si lo hubiera sido, no hará nada.
  - Tendrá un método `show`, con una url como argumento, que mostrará en pantalla el contenido del documento correspondiente con esa url, usando `retrieve` para descargarlo si es necesario.
  - Tendrá un método `show_all`, sin argumentos, que mostrará un listado de todas las urls cuyo documento se ha descargado ya.
  - Tendrá un método, `content`, con una url como argumento, que devolverá una cadena de caracteres (*string*) con el contenido del documento correspndiente a la url, usando `retrieve` para descargarlo si es necesario.

Se recomienda que la clase `Robot` use el módulo estándar de Python `urllib.request`[20][21] para descargar documentos, y se pide que la clase `Cache` use la clase `Robot` para descargar documentos.

[20] Módulo urllib.request:
https://docs.python.org/3/library/urllib.request.html

[21] Ejemplos del módulo urllib.request:
https://docs.python.org/3/library/urllib.request.html#examples

El programa principal creará al menos dos objetos de la clase Robot, llamará a varios de sus métodos, y creará al menos dos objetos de la clase Cache, llamando también a varios de sus métodos, para mostrar que todo funciona.

Solución:

Se ofrece una posible solución en el ficheros `cache_web.py` del directorio `Python-Intro` del repostorio de código de la asignatura.

16.10. Descarga de documentos web con módulos

Repositorio plantilla (para entrega):
https://gitlab.eif.urjc.es/cursosweb/2023-2024/descarga-documentos-web-con-modulos

Enunciado:

Realiza un programa que tenga la misma funcionalidad que "Descarga de documentos web" (ejercicio 16.9), pero usando un módulo (un fichero Python) para la clase Robot, y otro para la clase Cache. El programa principal, que será otro fichero, implementará la misma funcionalidad de prueba de las clases

Solución:

Se ofrece una posible solución en los siguientes ficheros del directorio `Python-Intro` del repostorio de código de la asignatura:

- `robot.py`: módulo con la clase Robot
- `cache.py`: módulo con la clase Cache
- `cache_web_modules.py`: programa principal

17. Ejercicios P2: Aplicaciones web simples

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

17.1. 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 tres soluciones en el directorio `Python-Web` del repositorio de código de la asignatura:

- `servidor-http-simple.py`: permite conexiones desde localhost
- `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.
- `servidor-http-simple-3.py` permite conexiones tanto desde localhost como desde fuera de la máquina huésped, y es capaz de reusar el puerto de forma que se puede rearrancar en cuanto muere.

17.2. Variaciones de la aplicación web hola mundo

Enunciado:

Basándose en la aplicación "Hola mundo" construida para el ejercicio 17.1, 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:

Soluciones de referencia, en el directorio `Python-Web` del repositorio de código de la asignatura:

- `servidor-http-simple-img.py`
- `servidor-http-simple-404.py`
- `servidor-http-simple-301.py`

17.3. 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".

Material:

Soluciones de referencia, en el directorio `Python-Web` del repositorio de código de la asignatura:

- `servidor-http-simple-random.py`

17.4. Aplicación redirectora

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/aplicacion-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), que bien puede ser de sí mismo (como en el ejercicio 17.3) o externo a partir de una lista con URLs.

Comentarios:

Para poder observar con más facilidad en el navegador lo que está ocurriendo, confgura el panel de Red de forma que mantenga el log de transacciones de forma permanente (en lugar de reiniciarlo cada vez que se pida una página nueva). En Firefox, esto se hace en la configuración del panel de Red, activando la opción "Log persistente".

Motivación:

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

17.5. Sumador simple

Enunciado:

a) En una fase: Construye una aplicación web que suma en una fases. Invocamos una URL del tipo `http://localhost:1234/sumar/1/2`, aportando la operación, el primer y el segundo operando. La aplicación nos devuelve el resultado de la suma.

b) En dos fases: Construye una aplicación web que suma en dos fases. En la primera, invocamos una URL del tipo `http://localhost:1234/5`, 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".

Repositorio de inicio: https://gitlab.etsit.urjc.es/grex/x-serv-14.5-sumador-simple

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.

17.6. 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`.

17.7. Clase servidor de aplicaciones, generador de URLs aleatorias

Enunciado:

Realiza el servidor especificado en el ejercicio "Aplicación web generadora de URLs aleatorias" (ejercicio 17.3) utilizando el esquema de clases definido en el ejercicio "Clase servidor de aplicaciones" (ejercicio 17.6).

Repositorio de inicio: https://gitlab.etsit.urjc.es/grex/x-serv-14.7-servurlaleat

17.8. Clase servidor de aplicaciones, sumador

Enunciado:

Realizar el servidor especificado en el ejercicio "Sumador simple" (ejercicio 17.5) utilizando el esquema de clases definido en el ejercicio "Clase servidor de aplicaciones" (ejercicio 17.6).

Repositorio de inicio: https://gitlab.etsit.urjc.es/grex/X-Serv-14.8-Servidor-Aplicaciones-Sumador

17.9. Clase servidor de varias aplicaciones

Enunciado:

Realizar una nueva clase, similar a la que se construyó en el ejercicio "Clase servidor de aplicaciones" (ejercicio 17.6), 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.

Repositorio de inicio: https://gitlab.etsit.urjc.es/grex/X-Serv-14.9-ServVariasApps

17.10. Clase servidor, cuatro aplis

Enunciado:

Utilizando la clase creada para "Clase servidor de varias aplicaciones" (ejercicio 17.9), crea una 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 17.5), 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 17.3).

Repositorio de inicio: https://gitlab.etsit.urjc.es/grex/X-Serv-14.10-CuatroAplis

17.11. Herramientas de Web Developer

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

18. Ejercicios P3: Introducción a Django

18.1. Instalación de Django

Enunciado:
Instala la versión de Django que utilizaremos en prácticas.
Comentarios:
Utilizaremos la versión Django 2.1.7.
Material:

- Transparencias "Introducción a Django"

- Descarga de Django: http://www.djangoproject.com/download/ ("Option 1: Get the latest official version")

18.2. Introducción a Django

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 sqlitebrowser.
Material:
Se puede encontrar un ejemplo de solución del ejercicio "Django intro" en el directorio `Python-Django/django-intro` del repositorio de codigo.
Además, se recomienda consultar:

- Django Getting Started:
  https://docs.djangoproject.com/en/2.1/intro/

18.3. 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`).

18.4. Django calc

Enunciado:

Realiza una calculadora con Django. Esta calculadora responderá a URLs de la forma "/suma/num1/num2", "/multi/num1/num2", "/resta/num1/num2", "/div/num1/num2", realizando las operaciones correspondientes, y devolviendo error "Not Found" para las demás.

Parte del repositorio en GitLab: `https://gitlab.etsit.urjc.es/cursosweb/x-serv-15.4-dj`
El proyecto Django se llamará `project` y la aplicación `calc`. Recuerda que sólo tendrás que modificar los siguientes ficheros: `urls.py` (modificando el del proyecto y creando el de la app) y `views.py`.

18.5. Django cms

Enunciado:

Realizar una sistema de gestión de contenidos muy simple con Django. Corresponderá con la funcionalidad de "contentApp" (ejercicio 19.1), almacenando los contenidos en una base de datos. La aplicación Django se ha de llamar `cms`.

18.6. Django cms_put

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-cms-put

Enunciado:

Realizar una sistema de gestión de contenidos muy simple con Django. Corresponderá con la funcionalidad de "contentPutApp" (ejercicio 19.3), almacenando los contenidos en una base de datos. En otras palabras, será como "Django cms" (ejercicio 18.5), añadiendo la funcionalidad de que el usuario pueda poner contenidos mediante PUT, tal y como se explicó en el ejercicio de "contentPutApp". La aplicación Django se ha de llamar `cms_put`.

Entrega:

Incluye en el repositorio todos los ficheros con código Python del proyecto, incluido `manage.py` y los contenidos en el directorio de proyecto y en el directorio la app (`cms`), así como la base de datos en un fichero `db.sqlite3`). Comprueba también que el código en `views.py` sigue con las reglas de estilo de Python (PEP8), utilizando `pycodestyle`.

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 en bytes. A pesar de su nombre, este último atributo tiene esos datos tanto si la petición es un POST como si es un PUT.

18.7. Django cms_put-post

Repositorio plantilla (para entrega):

https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-cms-put

Enunciado:

Realizar una sistema de gestión de contenidos muy simple con Django. Corresponderá con la funcionalidad de "contentPostApp" (ejercicio 19.4), almacenando los contenidos en una base de datos. En otras palabras, será como "Django cms" (ejercicio 18.5), añadiendo la funcionalidad en la que cuando se realice un GET sobre un recurso que no exista en la base de datos, se muestre un formulario que permita añadir el nuevo contenido mediante POST, tal y como se explicó en el ejercicio de "contentPostApp". La aplicación Django se ha de llamar `cms_put_post`.

Entrega:

Incluye en el repositorio todos los ficheros con código Python del proyecto, incluido `manage.py` y los contenidos en el directorio de proyecto y en el directorio la app (`cms`), así como la base de datos en un fichero `db.sqlite3`). Comprueba también que el código en `views.py` sigue con las reglas de estilo de Python (PEP8), utilizando `pycodestyle`.

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 en bytes. A pesar de su nombre, este último atributo tiene esos datos tanto si la petición es un POST como si es un PUT.

18.8. Django cms_users

Enunciado:

Realizar un proyecto Django con la misma funcionalidad que "Django cms_put" (ejercicio 18.6 o "Django cms_put-post" (ejercicio 18.7, 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 "Authentication in web requests" https://docs.djangoproject.com/en/stable/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".

Las aplicación Django "Admin site", entre otras, utiliza el middleware CSRF (protección frente a "Cross Site Request Forgery"), que hay que tener en cuenta, especialmente en los formularios POST. En particular, es importante asegurarse de que se han referenciado los módulos de CSRF en settings.py:

```
MIDDLEWARE_CLASSES = (
    ...
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.middleware.csrf.CsrfResponseMiddleware',
    ...
```

Más información sobre este tema:

- https://docs.djangoproject.com/en/4.2/ref/csrf/
- https://docs.djangoproject.com/en/4.2/howto/csrf/#using-csrf

18.9. Django cms_users_put

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-users

Enunciado:

Realizar un proyecto Django con la misma funcionalidad que "Django cms_users" (ejercicio 18.8), 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").

Entrega:

Incluye en el repositorio todos los ficheros con código Python del proyecto, incluido `manage.py` y los contenidos en el directorio de proyecto y en el directorio la app (`cms`), así como la base de datos en un fichero `db.sqlite3`). Comprueba también que el código en `views.py` sigue con las reglas de estilo de Python (PEP8), utilizando `pycodestyle`.

18.10. Django cms_templates

Enunciado:

Realizar un proyecto Django con la misma funcionalidad que "Django cms_users_put" (ejercicio 18.9), 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.

18.11. Django cms_post

Enunciado: Realizar un proyecto Django con la misma funcionalidad que "Django cms_templates" (ejercicio 18.10), pero atendiendo a una nueva familiar 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.

Repositorio en GitLab:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-cms-post.

18.12. Django cms_forms

Enunciado:

Realizar el ejercicio "Django cms_post" (ejercicio 18.11) utilizando la clase Forms de Django.

Además, 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.

Repositorio en GitLab para entregar el ejercicio:
https://gitlab.etsit.urjc.es/cursosweb/practicas/server/django-cms-forms.

18.13. Django tests

Enunciado:

Realizar un juego de pruebas (tests) para el ejercicio "Django cms_forms" (ejercicio 18.12) utilizando los tests de Django (`django.test`. Para que las pruebas sean más completas, añádase un pie de página en todas las páginas que se sirven que incluya el número de páginas vistas del servidor hasta ese momento (sólo desde que se lanzó el servidor). Para anotar las páginas con el número de páginas vistas, se implementará un objeto contador, que tendrá un método para incrementar el contador, y que devolverá el valor del contador tras inscrementarlo.

Las pruebas deberán ser de dos tipos:

- Pruebas unitarias del objeto contador.
- Pruebas extremo a extremo del servidor recibiendo ciertas peticiones HTTP.

En este ejercicio no importa mucho qué pruebas se hagan, lo importante es experimenar haciendo pruebas.

Materiales:

- "Testing in Django":
  https://docs.djangoproject.com/en/4.0/topics/testing/
- "Writing your first Django app, part 5":
  https://docs.djangoproject.com/en/3.2/intro/tutorial05/
- Solución en el directorio `Python-Django/django-tests` del repositorio de código de la asignatura.

18.14. Django tests en GitLab

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/django-tests-gitlab

Enunciado:

Automatizar las pruebas de un projecto Django como el descrito en el ejercicio "Django tests" (ejercicio 18.13) utilizando el sistema de CI de GitLab, de forma que cada vez que se suba un nuevo commit al repositorio, se ejecuten esas pruebas.

Materiales:

- Repositorio con una solución a este ejercicio:
  https://gitlab.etsit.urjc.es/cursosweb/practicas/server/testing-example
- Video explicativo:
  https://www.youtube.com/watch?v=XjMseC6yGS8

18.15. Django feed_expander

Utilizando Django y, en la medida que te parezca conveniente, `feedparser.py` (https://github.com/kurtmckee/feedparser) y `BeautifulSoup.py` (http://www.crummy.com/software/BeautifulSoup/), 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 mediante el servicio Twitrss en URLs como https://twitrss.me/twitter_user_to_rss/?user=user, para el usuario `user`.

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

Referencias:

- Documentación sobre feedparser.py:
  https://pythonhosted.org/feedparser/
- Presentación sobre feedparser.py:
  http://www.slideshare.net/LindseySmith1/feedparser
- Documentación sobre BeautifulSoup.py:
  http://www.crummy.com/software/BeautifulSoup/documentation.html

Repositorio en GitLab para entregar el ejercicio:
https://gitlab.etsit.urjc.es/grex/X-Serv-15.12-Django-feedexpander.

18.16. Django feed_expander_db

Realiza un servicio que proporcione la misma funcionalidad que "Django feed_expander" (ejercicio 18.15), 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.

18.17. Django Conciertos

Realiza las siguientes modificaciones a la aplicación de Django llamada conciertos que encontrarás en https://gitlab.etsit.urjc.es/cursosweb/16.14-conciertos:

- Añade una imagen (estática) que se referencia en la plantilla base.
- Modifica el widget en el formulario de conciertos para que también se pueda indicar la hora de comienzo del concierto.
- Añade al menos dos tests a tests.py.

19. 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.

19.1. Clase contentApp

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/contentapp

Enunciado:

Esta clase, basada en el esquema de clases definido en el ejercicio "Clase servidor de aplicaciones" (ejercicio 17.6), 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.

19.2. Instalación y prueba de Poster

Enunciado:

Instalación y prueba de Poster, *add-on* de Firefox

Referencias:

Poster Firefox add-on:
https://addons.mozilla.org/es/firefox/addon/poster/

También se puede utilizar RestClient, que tiene funcionalidad parecida:
https://addons.mozilla.org/en-US/firefox/addon/restclient/

19.3. Clase contentPutApp

Enunciado:

Construcción de la clase "contentPutApp", similar a contentApp (ejercicio 19.1). 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.

19.4. Clase contentPostApp

Repositorio plantilla (para entrega):
https://gitlab.etsit.urjc.es/cursosweb/2022-2023/contentpostapp

Enunciado:

Construcción de la clase "contentPostApp", similar a contentApp (ejercicio 19.1). 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.

Repositorio de inicio: https://gitlab.etsit.urjc.es/cursosweb/X-Serv-17.4-ContentPostApp

Referencias:

- The Form element (MDN): https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form
- Forms in HTML (HTML 4.01 Specification by W3C): http://www.w3.org/TR/html4/interact/forms.html

19.5. Clase contentPersistentApp

Enunciado:

Construcción de la clase contentPersistentApp, similar a contentPutApp (ejercicio 19.3), 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.

19.6. 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.

19.7. Gestor de contenidos con usuarios

Enunciado:

Construye la clase contentAppUsers, que amplía el gestor de contenidos que estamos construyendo (clase contentStorageApp, ejercicio 19.6) 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.

19.8. Gestor de contenidos con usuarios, con control estricto de actualización

Enunciado:

Construye la clase contentAppUsersStrict, que implemente la misma funcionalidad de contentAppUsers (ejercicio 19.7), 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.

20. Ejercicios P5: Aplicaciones web con base de datos

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

20.1. 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.

20.2. Gestor de contenidos con base de datos

Enunciado:

Escribe y prueba la clase contentDBApp, que será una versión de contentApp (ejercicio 19.1), pero utilizando una base de datos SQLite3 para almacenar sus objetos persistentes.

20.3. 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 19.8), pero usando base de datos como almacenamiento permanente.

21. 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.

21.1. 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 funcionado 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 aparatados más arriba.

21.2. 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.

21.3. 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, diséñalo 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.

21.4. 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.

22. Prácticas de entrega voluntaria de cursos pasados

22.1. Prácticas de entrega voluntaria (curso 2021-2022)

22.2. Minipráctica 2 (entrega voluntaria)

Esta práctica tendrá como objetivo la creación de una aplicación web (de nombre *acorta*) simple para acortar URLs utilizando Django (en un nuevo proyecto Django llamado *project*). Su enunciado será igual que el de la práctica 1 de entrega voluntaria (ejercicio 9.2), 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.
- Utilizará plantillas, de manera que el código Python y el HTML estarán separados.

Repositorio de partida:
https://gitlab.etsit.urjc.es/cursosweb/x-serv-18.2-practica2

22.3. Prácticas de entrega voluntaria (curso 2020-2021)

22.4. Minipráctica 1 (entrega voluntaria)

Esta práctica tendrá como objetivo la creación de una aplicación web simple para acortar URLs. Los usuarios de la aplicación podrán especificar URLs y nombres de recurso, y a partir de ese momento, esos nombres de recurso redireccionarán a su URL correspondiente. La aplicación podrá realizarse según el esquema de clases explicado en clase (usando, si se quiere, el módulo `webapp.py`), o de cualquier otra forma.

El repositorio de partida es: https://gitlab.etsit.urjc.es/cursosweb/mini-1-acortadora/

El código ha de guardarse en un fichero llamado *shortener.py*.

El funcionamiento de la aplicación será el siguiente:

- Recurso "/", invocado mediante GET. Devolverá una página HTML con un formulario. En ese formulario habrá dos campos, uno para la url a acortar, y el otro para escribir el nombre de recurso para esa url. El formulario se enviará

al servidor mediante POST, y sus dos campos se llamarán, respectivamenet, `url` y `short`. 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 tenga campos `url` y `short` 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 "https://" 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 "https://gsyc.es".

  Si se recibe una petición para un nombre de recurso que ya corresponde con una URL, se cambiará la URL. Una URL podrá estar referenciada por más de un nombre de recurso.

  Así, por ejemplo, si se quiere acortar `http://gsyc.urjc.es` con el nombre de recurso `gsyc`, y la aplicación está en el puerto 1234 de la máquina "localhost", se invocará (mediante POST) la URL

  `https://localhost:1234/`

  y en el cuerpo de esa petición HTTP irá la `qs`

  `url=https://gsyc.urjc.es&short=gsyc`

  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) una página HTML con el formulario, y la lista de URLs acortadas, incluyendo esta.

- Recursos correspondientes a URLs acortadas. Estos serán los nombres de recurso (los que se escribieron en el campo `short`) 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/gsyc`

la aplicación devolverá un HTTP REDIRECT a la URL

`https://gsyc.urjc.es`

Comentario
Se recomienda utilizar un diccionario para almacenar las URLs reales y los nombres de recurso correspondientes. La clave de búsqueda será el nombre de recurso, y el valor, la URL real.

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 el nombre de recurso).
- Reconocer "GET /recurso" (para cualquier recurso), y realizar la redirección correspondiente.
- Manejar las condiciones de error y realizar el resto de la funcionalidad.

22.5. Prácticas de entrega voluntaria (curso 2014-2015)

22.5.1. Práctica 1 (entrega voluntaria)

Fecha recomendada de entrega: Antes del 15 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.

22.5.2. Práctica 2 (entrega voluntaria)

Fecha recomendada de entrega: Antes del 19 de abril.

Esta práctica tendrá como objetivo la creación de una aplicación web (de nombre *acorta*) simple para acortar URLs utilizando Django (proyecto *project*). Su enunciado será igual que el de la práctica 1 de entrega voluntaria (ejercicio 22.5.1), 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.

Repositorio GitLab de entrega:
https://gitlab.etsit.urjc.es/CursosWeb/X-Serv-18.2-Practica2

22.6. Prácticas de entrega voluntaria (curso 2012-2013)

22.6.1. 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.

22.6.2. 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 22.6.1), 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.

22.7. Prácticas de entrega voluntaria (curso 2011-2012)

22.7.1. 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

Comentario:

En algunas circunstancias, el servidor de Wikipedia puede devolver un código de redirección (por ejmeplo, 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 redirige.

22.7.2. Práctica 2 (entrega voluntaria)

Realiza lo especificado en la práctica 1 (ejercicio 22.7.1), 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.

22.8. Prácticas de entrega voluntaria (curso 2010-2011)

22.8.1. 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

22.8.2. 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 20.3)
- 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).

22.8.3. 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.

22.8.4. 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.

23. Pruebas escritas pasadas

23.1. Examen de ITT-SAT, 28 de junio de 2023

23.1.1. Enunciado

Se quiere construir un sitio web, CancionVerano, donde se pueden ver y comentar las canciones actuales y votarlas para ser la canción del verano. La funcionalidad básica del sitio es la siguiente:

1. En el sitio no hay cuentas para usuarios: toda la funcionalidad está disponible para cualquiera que lo visite.

2. De todas formas, cualquier visitante podrá reservar un nombre que no esté ya en uso para cuando suba información al sitio. Este nombre se mantendrá mientras el visitante utilice el mismo navegador.

3. La página principal del sitio mostrará a los visitantes un formulario para buscar canciones, como documento HTML. Ese formulario permitirá especificar el nombre de un artista o banda. Una vez especificado, el sitio devolverá un listado de las canciones de ese artista, y un nuevo formulario como el anterior (por si el visitante quiere buscar canciones de otro artista o banda). Este listado que aparecerá en la página principal incluirá, para cada canción, su título (como enlace a la página de la canción, ver más abajo), su vídeo musical y un listado de los artistas que participan en la canción. Las canciones aparecerán en orden cronológico inverso (esto es, cuanto más recientes, más arriba).

4. La página principal tendrá también, cuando el visitante no haya reservado nombre, un formulario para ponerlo. Si se rellena ese formulario, volverá a verse la página principal, pero ahora ya con el nombre reservado (salvo que el nombre ya estuviera reservado previamente, en ese caso volverá a ver el formulario). Desde ese momento en adelante, en la página principal se verá siempre el nombre reservado, mientras se conecte desde el mismo navegador.

5. Cada canción tendrá una página. En esa página aparecerá el título de la canción, su letra, su vídeo musical, los comentarios que ha habido para esa canción, los likes de la canción, un formulario para poner un comentario y un botón de "like". Los comentarios aparecerán en orden inverso de publicación (los más recientes primero).

6. Cada visitante puede poner un comentario utilizando el formulario indicado anteriormente. Cuando lo ponga, aparecerá su nombre (si ha reservado uno),

o como anónimo (si no lo ha hecho). Al poner un comentario, el visitante volverá a ver la página de la canción que estaba viendo, pero ahora con el comentario que acaba de poner.

7. Todas las páginas del sitio tendrán una imagen de cabecera (banner).

8. Los vídeos musicales están alojados en la plataforma de YouTube.

9. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio `tracking.com`. Cada una de estas imágenes se sirve con una URL diferente si está en una página diferente, pero igual si está en la misma página.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). (1 punto)

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la URL de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario para elegir un nombre. Pero el nombre elegido ya está reservado, y el visitante ha de rellenar el formulario por segunda vez (esta vez, el nombre no estará previamente reservado). El escenario termina cuando el visitante ve de nuevo la página principal, ya con el nombre que ha reservado. (1 punto)

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya ha reservado nombre y está viendo la página de una canción en su navegador. El visitante da "like" a esa canción. El visitante rellena el formulario de comentarios, poniendo uno. El escenario termina cuando el visitante ve la página de la canción con su "like" y su comentario. (1 punto)

5. Explica cómo debería ser la cookie o cookies que envíe el sitio a cada navegador, de forma que permita asegurar la funcionalidad indicada, pero nada más: (a) Explica qué campos tendría o tendrían (los mínimos posibles), (b) cuándo habría que enviarla (o enviarlas), siendo siempre el envío lo más tarde posible en las interacciones entre cada navegador (visitante) y la aplicación, (c) y para qué serviría. Explica también (d) si se enviarían cookies de otros sitios que no sean el sitio CancionVerano. Explica el porqué en todos los casos. En general, asegúrate de que la aplicación envía cookies sólo cuando lo necesita para cumplir su funcionalidad. (1 punto)

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

23.1.2. Solución y rúbrica calificación

Ejercicio 1

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Canciones de un artista	`artista="Nombre Artista"`
/	POST	Nombre de visitante	`nombre="Nombre Visitante"`
/{id_cancion}	GET	Página de la canción	
/{id_cancion}	POST	Nuevo comentario	`texto="..."`
/{id_cancion}	POST	"Like"	`like=True`
/banner	GET	Banner (imagen de cabecera)	

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index')
  path('<id_cancion>', views.cancion, name='cancion')
  path('banner', views.banner, name='banner')
]
```

Rúbrica

Para estar correcto, el ejercicio ha de incluir tanto una tabla con un documento `urls.py` correctos, completos, y sin recursos innecesarios.

Indicaciones aproximadas de errores:

- No hay columna de datos en la tabla, o está incorrecta: -0.3
- No hay recurso para banner: -0.3
- No hay recurso para pedir las canciones de un artista: -0.3
- No hay recurso para poner un nombre de visitante: -0.3
- No hay recurso para dar un "like" o poner comentario: -0.3
- No coincide la tabla con el fichero `urls.py`: -0.2

Ejercicio 2

Hay varias soluciones, una podría ser:

```
from django.db import models

class Visitante (models.Model):
  id = models.CharField(max_lenght = 50)
  nombre = models.CharField(max_lenght = 100)

class Artista (models.Model):
  nombre = models.CharField(max_lenght = 100)

class Cancion (models.Model):
  titulo = models.CharField(max_lenght = 200)
  letra: models.TextField()
  video: models.URLField()
  fecha: models.DateTimeField()

class ArtistaCancion(models.Model):
  artista = models.ForeignKey('Artista')
  cancion = models.ForeignKey('Cancion')

claas Like(models.Model):
  visitante = models.ForeignKey('Visitante')
  cancion = models.ForeignKey('Cancion')

class Comentario (models.Model):
  visitante = models.ForeignKey('Visitante')
  cancion = models.ForeignKey('Cancion')
  texto: models.CharField(max_lenght = 500)
  fecha: models.DateTimeField()
```

La tabla `Visitante` se usa para considerar los visitantes (navegadores distintos), y su id será el valor de la cookie que se les envíe.

La tabla `Artista` se usa para considerar los datos de un artista.

La tabla `Cancion` se usa para considerar las canciones que están almacenadas en la base de datos de la aplicación. El campo `fecha` hace falta para poder ordenar las canciones por fecha en la página principal.

La tabla `ArtistaCancion` se usa para expresar las relaciones entre artistas y sus canciones. Como una canción puede tener varios artistas, no se puede expresar con un campo "foreign key" apuntando a `Artista` en la tabla `Cancion`. De la misma forma, como un artista puede participar en varias canciones, no se puede expresar en la tabla `Artista` como un campo "foreign key" apuntando a la tabla `Cancion`. Sí podría expresarse también como un campo `ManyToManyField` entre

`Artista` y `Cancion`.

La tabla `Like` se usa para expresar la relación entre visitantes y las canciones a las que ha "dado un like". No se puede expresar con un contador en la tabla `Cancion` porque no habría forma de comprobar que se ha recibido más de un like del mismo visitante. Sí podría expresarse también como un campo `ManyToManyField` entre `Visitante` y `Cancion`.

La tabla `Comentario` almacena el texto de los comentarios, la canción en la que fueron puestos, el visitante que los puso, y la fecha en que fueron puestos. La canción se utilizará para mostrar en cada página de canción sus mensajes. La fecha se usará para poder ponerlos en orden. Para esto último se podría utilizar también el `Id` de la tabla, dado que Django lo autoincrementará, y por tanto también sirve para ordenar. En ese caso, no haría falta el campo `fecha`.

Rúbrica

- No están las tablas `Artista`, `Cancion` y `Comentario` (y no sé ve cómo cumplir el enunciado): -0.6
- No está la tabla `Visitante`, o alguna que cumpla su función: -0.3
- No están las tablas `ArtistaCancion` o `Like`, o alguna que cumpla su función: -0.3
- Hay campos innecesarios
- `Comentario` o `Cancion` no permiten ordenar por fecha: -0.2

Ejercicio 3

Las interacciones son entre el navegador y dos servidores web: el sitio y tracking.com. Como el navegador no ha accedido nunca al sitio, no puede tener una cookie para él. En este escenario se enviará cookie de sesión en cuanto hace falta: cuando se asigna un nombre válido al visitante. Antes no hace falta pues no se realiza ninguna acción que necesite identificar al visitante (ponerse un nombre, poner un comentario o dar un like). Como en este escenario no se ha hecho una consulta sobre las canciones de algún artista, la página principal no tendrá listado de canciones, y por tanto tampoco habrá que realizar consultas a YouTube. El escenario comenzará con una petición GET del navegador al recurso principal, que obtendrá como respuesta la página HTML principal. A continuación, se envía un POST, resultado de que el visitante haya rellenado el nombre del artista en el formulario de listado de canciones. Este POST falla, por lo que se vuelve a recibir la misma página HTML. Se repite el proceso, y esta vez se puede poner el nombre, por lo que se enviará cookie en respuesta al POST, que irá por tanto junto con la página HTML ya con el nombre puesto, y con eso termina el escenario.

- CancionVerano: Petición GET al recurso principal:

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, HTML]
```

- CancionVerano: Petición del banner:

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- tracking.com: Petición de la imagen:

```
GET /imagen-principal HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen]
```

- CancionVerano: POST para poner un nombre:

```
POST / HTTP/1.1
...
nombre="Pepita Grilla"
```

- Respuesta, suponiendo que `Pepita Grilla` es un nombre que ya existe.

```
HTTP/1.1 200 OK
...

[Página principal, HTML, indicando que el nombre ya existe]
```

- CancionVerano: Petición del banner (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).
- tracking.com: Petición de la imagen (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).
- CancionVerano: Nuevo POST para poner un nombre:

```
POST / HTTP/1.1
...
nombre="Pepito Grillo"
```

- Respuesta, suponiendo que `Pepito Grillo` es un nombre válido que no tenía ningún visitante.

```
HTTP/1.1 200 OK
...
Set-Cookie: vistante="4343fae54be...."; Expires=Fri, 6 Jul 2035 06:32:33 GMT

[Página principal, HTML, ya con el nombre Pepito Grillo en ella]
```

- CancionVerano: Petición del banner (suponemos que no se usa la cache):

```
GET /banner HTTP/1.1
...
Cookie: vistante="4343fae54be...."
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- tracking.com: Petición de la imagen (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).

Rúbrica

Se espera que las interacciones sean correctas, incluyan todos los campos indicados, y las cookies sean correctas y consistentes con el ejercicio anterior.

- Interacciones no consistentes con primer ejercicio: -0.3
- Faltan datos en la respuesta: -0.4
- Falta interacción para banner: -0.3
- Falta interacción para imagen: -0.3
- Faltan interacciones para segundo POST: -0.4
- Query string equivocada en algún POST: -0.2
- Cookies incorrectas: -0.3
- Cookie enviada demasiado pronto: -0.2
- Se violan reglas de funcionamiento de cookies (ej: se envía una cookie que no se ha recibido, o un navegador inicia una cookie): -0.4

Ejercicio 4

Las interacciones son entre el navegador y tres servidores web: el sitio, YouTube, y tracking.com. Como el navegador ya ha accedido al sitio para elegir nombre de visitante, tendrá una cookie para él, por lo que se enviará siempre que se conecte al sitio. Además, como la página HTML de la canción incluye el vídeo de la canción en YouTube, habrá al menos un acceso a YouTube para descargar la muestra del vídeo que éste ofrece. El escenario comenzará con una petición POST del navegador a la página de la canción, que se produce cuando el visitante pulsa el botón "Like". Esta petición obtendrá como respuesta la página HTML de la canción, ya con el "like" incluido. A continuación, se envía un nuevo POST, resultado de que el visitante decida poner un comentario. Esta petición recibe de nuevo la página HTML de la canción, ya con el comentario puesto, y con eso termina el escenario.

- CancionVerano: Petición POST al recurso correspondiente a la página de la canción, para hacer el "like" (suponemos que `/3452` es el recurso correspondiente a la página de la canción:

```
POST /3452 HTTP/1.1
...
Cookie: vistante="4343fae54be...."
...

like=True
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página de la canción, con el "like" contabilizado, HTML]
```

- CancionVerano: Petición del banner (suponemos que no se usa la cache):

```
GET /banner HTTP/1.1
...
Cookie: vistante="4343fae54be...."
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- tracking.com: Petición de la imagen:

```
GET /imagen-cancion-3452 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen]
```

- YouTube: Petición de muestra del vídeo (suponemos que el recurso de YouTube que la sirve es `/recurso-video.3452`. Podría haber varias peticiones si, por ejemplo, se descarga un script JavaScript que se encarga de preparar la muestra y éste a su vez se descarga más elementos de YouTube, pero suponemos que basta con una interacción, por simplificar:

```
GET /recurso-video-3452 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Muestra del video]
```

- CancionVerano: POST para poner el comentario:

```
POST /3452 HTTP/1.1
...
Cookie: vistante="4343fae54be...."
...

texto="Texto del comentario"
```

- Respuesta:

```
HTTP/1.1 200 OK
...

[Página de la canción, HTML, ya con el comentario]
```

- CancionVerano: Petición del banner (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).

- tracking.com: Petición de la imagen (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).
- YouTube: Petición de muestra del vídeo (suponemos que no se usa la cache): igual que la anterior (petición y respuesta).

Rúbrica

Se espera que las interacciones sean correctas, incluyan todos los campos indicados, y las cookies sean correctas y consistentes con el ejercicio anterior.

- Interacciones no consistentes con primer ejercicio: -0.3
- Interacciones no consistentes con ejercicio anterior: -0.3
- Faltan datos en la respuesta: -0.4
- Falta interacción para banner: -0.3
- Falta interacción para imagen: -0.3
- Falta interacción para YouTube: -0.3
- Faltan interacciones para alguno de los POST: -0.4
- Query string equivocada en algún POST: -0.2
- Cookies incorrectas: -0.3
- Se violan reglas de funcionamiento de cookies (ej: se envía una cookie que no se ha recibido, o un navegador inicia una cookie): -0.4

Ejercicio 5

Para mantener información asociado a los visitantes a CancionVerano, el sitio tendrá que enviar una cookie de sesión cuando el visitante haga alguna acción que haya que relacionar con un visitante concreto. Dado el enunciado, estas acciones son: ponerse un nombre, dar un "like" a una canción, o poner un comentario en una canción. Por lo tanto, la cookie se tendrá que enviar cuando ocurra una de estas tres cosas. Será una cookie de sesión, y es conveniente que sea persistente (se mantenga aunque el navegador rearranque): para ello tendrá que tener fecha de expiración. Como la cookie de sesión permite que un visitante se impersone como otro si tiene acceso a su cookie de sesión, es conveniente también que la cookie se envíe sólo sobre conexiones cifradas (HTTPS), para lo que tendrá que tener el campo `Secure`. Para mitigar algunos tipos de ataques, puede ser conveniente que la cookie sea inaccesible a JavaScript, para lo que se puede usar el campo `HttpOnly`

Un ejemplo completo de cómo enviar la cookie, por tanto, podría ser:

```
Set-Cookie: vistante="4343fae54be...."; Expires=Fri, 6 Jul 2035 06:32:33 GMT; Secu
```

Para el sitio tracking.com no se describe funcionalidad específica, pero es muy posible que para poder realizar sus funciones de trazado de páginas vistas, envié una cookie que en este caso no necesita ser segura, aunque sí será conveniente que sea persistente:

```
Set-Cookie: id="43334566750...."; Expires=Sat, 7 Jul 2035 16:32:33 GMT
```

Por su parte, aunque no hace falta para la funcionalidad descrita en la página, YouTube enviará sus propias cookies. En caso de que el usuario esté autenticado en YouTube (haya hecho login en una sesión), se habrán recibido sus cookies ligadas a un usuario concreto, y se enviarán éstas en cada interacción. Si no es así, típicamente YouTube enviará cookies al menos para trazar la sesión anónima.

Rúbrica

- Cada apartado mal: -0.3

23.2. Examen de ITT-SAT, 15 de mayo de 2023

23.2.1. Enunciado

Se quiere construir un sitio web, Chs.ts, para poder comunicarse mediante mensajes. La funcionalidad básica del sitio es la siguiente:

1. Toda la funcionalidad del sitio está disponible para cualquier visitante: no hay cuentas, ni hace falta acreditarse para tener acceso a la funcionalidad que proporciona.

2. La página principal del sitio mostrará enlaces a las salas en que el visitante ha entrado alguna vez (haya puesto mensajes o no) desde ese mismo navegador. También ofrecerá un botón para crear una nueva sala. Si se pulsa el botón, se creará una nueva sala, y como resultado de la operación el usuario quedará en la página de la sala.

3. Todas las salas del sitio se generan mediante el botón anterior (sólo se puede usar una sala si ha sido creada). Cada sala tiene una URL única (que mostrará la página de la sala), donde el nombre de recurso es una cadena de caracteres aleatorios de 50 caracteres.

4. La página de cada sala podrá ser utilizada por cualquier visitante, si conoce su URL. Al acceder a esta URL se verán los últimos 10 mensajes que se haya puesto en la sala por cualquier visitante. También habrá un formulario para poner nuevos mensajes: si se usa, el mensaje puesto será uno de los 10 mensajes que verá cualquier visitante en esta sala a partir de este momento, hasta que se pongan otros 10 después de él, momento en que ya no aparecerá si se accede a la URL de la sala.

5. El sitio también permite que quien quiera envíe mensajes desde un programa (usando el mismo mecanismo que usa el formulario de mensajes), y que los mensajes de una sala sean descargados en formato JSON (igualmente, solo los 10 últimos). Para esta descarga se utilizará una URL para cada sala, distinta de la que permite acceso "normal" a la sala desde el navegador.

6. Todas las páginas HTML del sitio usarán un único documento CSS, servido por el propio sitio, que definirá el estilo de todas ellas de forma uniforme.

En esta descripción, considérese que un "visitante" corresponde con un cierto navegador, con su propio almacén de cookies. Esto es, una misma persona que use dos navegadores distintos, cada uno con su propio almacén de cookies, será considerada como dos visitantes, mientras que varias personas usando el mismo navegador, con un único almacén de cookies para todas ellas, serán consideradas como un único visitante.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero `urls.py` de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos los campos o tablas necesarios para garantizar que la funcionalidad es la descrita, y que el sitio nunca guarda más información que la estrictamente necesaria para cumplir su función. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero `models.py` en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). (1 punto)

3. Explica cómo debería ser la cookie o cookies que envíe el sitio a cada navegador, de forma que permita asegurar la funcionalidad indicada, pero nada más: (a) Explica qué campos tendría o tendrían (los mínimos posibles), (b) cuándo habría que enviarla (o enviarlas), siendo siempre el envío lo más tarde posible en las interacciones entre cada navegador (visitante) y la aplicación, (c) y para qué serviría. Explica también (d) si se enviarían cookies a los programas (no navegadores) que lean y escriban mensajes en una cierta sala según se describe en el enunciado. Explica el porqué en todos los casos. En general, asegúrate de que la aplicación envía cookies sólo cuando lo necesita para cumplir su funcionalidad.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y la aplicación en el siguiente escenario. El escenario comienza cuando un visitante que no ha accedido nunca al sitio pone su URL en el navegador para acceder por primera vez. El visitante ve la página principal, crea una sala, y pone un mensaje en ella. El escenario termina cuando el visitante ve en su navegador la página de la sala con el mensaje que acaba de poner.

Para cada interacción HTTP indica claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando, y que el navegador acepta cualquier cookie que se le envíe, y nunca la borra salvo que la cookie expire. (1 punto)

5. Se quiere implementar, para cada sala del sitio, un botón para borrarla. Cuando un visitante pulse un botón, el contenido de la sala y la propia sala se destruirán, y el usuario pasará a ver la página principal (sin enlace para la sala que acaba de borrar, que ya no existe). A partir de ese momento cualquier visitante no podré acceder a la sala o a sus contenidos. Explica cómo implementarías la vista que reciba la acción de pulsar el botón, y en particular: (a) En qué nombre de recurso se activaría esta vista, (b) suponiendo que reciba una petición POST, que vendría como query string en ella, (c) qué acciones habría que hacer sobre la base de datos, (d) qué cookies habría que enviar al navegador, si hay que enviar alguna, y (e) qué código HTTP de resultado devolvería la vista en su respuesta, si la operación se ha realizado correctamente. Explica el porqué de tus respuestas (1 punto).

23.2.2. Solución y rúbrica calificación

Ejercicio 1

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Nueva sala	`sala=true`
/{id_sala}	GET	Página de la sala	
/{id_sala}	POST	Nuevo mensaje	`texto="..."`
/{id_sala}.json	GET	Contenido de sala como JSON	
/main.css	GET	Documento CSS	

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index')
  path('<id_sala>', views.sala, name='sala')
  path('<id_sala>.json', views.sala_json, name='sala_json')
  path('main.css', views.css, name='css')
]
```

Rúbrica

Para estar correcto, el ejercicio ha de incluir tanto una tabla con un documento `urls.py` correctos, completos, y sin recursos innecesarios.

Indicaciones aproximadas de errores:

- No hay columna de datos en la tabla, o está incorrecta: -0.3
- No hay recurso para documento CSS: -0.3
- No hay recurso para documento JSON: -0.3
- No hay recurso o método para crear sala o poner mensaje: -0.3
- El recurso para documento JSON no incluye id de sala: -0.2
- No coincide la tabla con el fichero `urls.py`: -0.2

Ejercicio 2

Hay varias soluciones, una podría ser:

```
from django.db import models

class Sesion (models.Model):
  id = models.CharField(max_lenght = 50)

class Sala (models.Model):
  id = models.CharField(max_lenght = 50)

class Visitada (models.Model):
  sala: models.ForeignKey('Sala')
  sesion: models.ForeignKey('Sesion')

class Mensaje (models.Model):
  texto: models.CharField(max_lenght = 500)
  sala: models.ForeignKey('Sala')
  fecha: models.DateTimeField()
```

La tabla `Sesion` se usa para considerar los visitantes (navegadores distintos), y su id será el valor de la cookie que se les envíe. También servirá, en combinación con la tabla `Visitada` para saber qué salas ha visitado un navegador.

La tabla `Sala` se usa para consdierar las salas que se han abierto, y poder detectar cuándo se pide una sala que no existe.

La tabla `Visitada` se usa para saber qué salas ha visitado un navegador, y se actulizará por la vista de cada sala, cuando detecte que un nuevo navegador está accediendo a ella.

La tabla `Mensaje` almacena el texto de los mensajes, la sala en la que fueron puestos, y la fecha en que fueron puestos. La sala se utilizará para mostrar en cada sala sus mensajes. La fecha se usará para poder ponerlos en orden. Para esto último se podría utilizar también el `Id` de la tabla, dado que Django lo autoincrementará, y por tanto también sirver para ordenar. En ese caso, no haría falta el campo `fecha`.

Rúbrica

- No están las tres tablas `Sesion`, `Sala` y `Mesaje` (y no sé ve cómo cumplir el enunciado): -0.6
- No está la tabla `Visitada`, o alguna que cumpla su función: -0.3
- Las referencias están al revés en `Sala` (ej: `Mensaje` desde `Sala`): -0.2
- Hay campos innecesarios (ej: `sesion` en `Mensaje`)

- `Mensaje` no permite ordenar por fecha: -0.2

Ejercicio 3

Basta con una cookie que se envíe a cada navegador cuando se visite una página de sala por primera vez. Antes, no hace falta, porque no hay ningún dato relevante que almacenar. La cookie ha de ser una cookie de sesión, por lo que puede ser una cadena de caracteres aleatorios. En esta solución utilizamos cadenas de 50 caracteres, como se vio en el campo `id` de la tabla `Sesion`. Suponiendo que se elija entre unos 60 caracteres diferentes para cada posición de la cadena (letras mayúsculas y minúsculas, y números, por ejemplo), tendríamos 60^{50} cadenas distintas, lo que es un número suficientemente grande como para poder manejar un número de navegadores incluso de varios miles de millones.

(a) Al enviar la cookie por primera vez, con `Set-Cookie`, bastaría con incluir un nombre y un valor de cookie (por ejemplo, `sesion`, y el valor sería la cadena aleatoria generada para ese navegador, que se almacenaría en la tabla `Sesion`).

Si se quiere que la cookie se almacene en almacenamiento estable (el enunciado no lo pide) basta con poner una fecha de expiración muy lejos en el futuro. Y para evitar que la cookie pueda ser espiada durante su tránsito por la red (el enunciado no lo pide, pero sería muy convenienet dado el enunciado), habría que incluir el campo `Secure`. Así, por ejemplo:

```
Set-Cookie: sesion=ae453...3rsv56; Expires=Mon, 22 May 2083 20:48:00 GMT; Secu
```

(b) Se enviaría cuando se reciba una petición sin cookie válida (o no la hay, o no se encuentra en la tabla `Sesion`) en el recurso que sirve la página de cualquier sala (la primera vez que un navegador visita una sala cualquiera del sitio).

(c) Serviría para poder ir apuntando qué salas va visitando el navegador (en el modelo de datos, en la tabla `Visitada`).

(d) No es preciso que los programas que descarguen los documentos JSON reciban o envíen cookies, ya que no hay nada que el servidor tenga que "recordar" de ellos, dado el enunciado. De todas formas, si el programa pone un mensaje en una sala, como realizará un POST al recurso de la sala, y el navegador no tiene forma de saber que es un programa y no un navegador, le enviará una cookie, como si fuera un navegador que visita una sala por primera vez. Pero el programa no tiene porqué volver a enviar esa cookie a la aplicación en futuras interacciones.

Rúbrica

- No se explican los campos (o no se mencionan): -0.3
- Se envían las cookies demasiado pronto: -0.3
- El diseño de cookies no funciona en general: -0.8
- El diseño de cookies no funciona para algún caso: -0.4
- El diseño de cookies no es óptimo: -0.2
- Respuesta incorrecta para el caso del programa: -0.3

Ejercicio 4

Todas las interacciones son entre el navegador y el sitio. Como el navegador no ha accedido nunca al sitio, no puede tener una cookie para él. El escenario comenzará con una petición GET del navegador al recurso principal, que obtendrá como respueta la página HTML principal. A continuación, el usuario pulsa el botón de crear una sala, y el navegador enviará un POST al recurso principal, que creará la sala y obtendrá como respuesta una redirección al recurso correspondiente a la página de la sala. Ante esta redirección, el navegador realizará un GET al recurso de la página de la sala, recibiendo como respuesta la página HTML de la sala. El usuario rellenará el formulario para poner un mensaje, y al pulsar el botón de enviar, el navegador realizará un POST sobre el recurso de la sala, para poner el mensaje. Como respuesta, obtendrá la página HTML de la sala, con el mensaje ya en ella.

- Petición GET al recurso principal:

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, HTML]
```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache):

```
GET /main.css HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- POST para crear una sala:

```
POST / HTTP/1.1
...
sala=true
```

- Respuesta, suponiendo que `/2df45..3gh` es el recurso de la nueva sala creada.

```
HTTP/1.1 303 See Other
...
Location: /2df45..3gh
...
```

- Nueva petición GET, en respuesta a la redirección:

```
GET /2df45..3gh HTTP/1.1
...
```

- Respuesta. Se incluye la cookie, como se ha indicado anteriormente, dado que la aplicación recibe una petición sin cookie en el recurso de una sala:

```
HTTP/1.1 200 OK
...
Set-Cookie: sesion=ae453...3rsv56; Expires=Mon, 22 May 2083 20:48:00 GMT; Se

[Página HTML de la sala]
```

- Petición del documento CSS `/main.css`, igual que antes, pero ahora el navegador envía la cookie, igual que en la interacción siguiente.

- POST para poner un mensaje. Se envía la cookie porque se recibió anteriormente:

```
POST /2df45..3gh HTTP/1.1
...
Cookie: sesion=ae453...3rsv56
...

mensaje="Este es mi primer mensaje"
```

- Respuesta:

```
HTTP/1.1 200 OK
...

[Página HTML de la sala, con el nuevo mensaje]
```

- Petición del documento CSS `/main.css`, igual que antes, también el navegador envía la cookie, igual que en la interacción anterior.

Rúbrica

Se espera que las interacciones sean correctas, incluyan todos los campos indicados, y las cookies sean correctas y consistentes con el ejercicio anterior.

- Interacciones no consistentes con primer ejercicio: -0.3
- Faltan datos en la respuesta: -0.4
- Falta interacción para CSS: -0.3
- Falta interacción para imagen: -0.3
- Faltan interacciones para segundo POST: -0.4
- Query string equivocada en algún POST: -0.2
- Cookies diferentes de ejercicio anterior, e incorrectas: -0.3
- Cookies diferentes de ejercicio anterior, y correctas: -0.2
- Cookie enviada demasiado pronto: -0.2

- Se violan reglas de funcionamiento de cookies (ej: se envía una cookie que no se ha recibido, o un navegador inicia una cookie): -0.4

Ejercicio 5

La vista que reciba la acción POST de pulsar el botón deberá borrar la sala de la tabla `Sala`. De esta forma, si se recibe una nueva petición GET o POST para esa sala, al no encontrarla en `Sala`, se tratará como si no existiera. Podría usarse el recurso de la sala (`/{id_sala}`) para atender esta pertición.

(a) El recurso de la sala (`/{id_sala}`).

(b) `borrar=True` (por ejemplo).

(c) Borrar la entrada correspondiete a la sala (obtenida a partir del nombre de recurso), asegurándose de que se borran las referencias a ella de la tabla `Mensaje` y `visitada`, para evitar inconsistencias. Django se podrá encargar automáticamente de ello si se ha configurado bien el modelo de datos. Si no es así, habría que realizar primero las queries correspndientes en esa tabla para borrar todas las entradas donde se referencie a la sala borrada.

(d) No habría que enviar ninguna cookie que no se hubiera enviado ya.

(e) Debería devolverse un código de redirección al recurso principal, dado el enunciado:

```
HTTP/1.1 303 See Other
...
Location: /
...
```

Rúbrica

- Cada apartado mal: -0.3

23.3. Examen de ITT-SAT, 23 de junio de 2022

Se quiere construir un sitio web, PelisOchenteras, donde se puede ver y comentar información sobre películas de los años 80. La funcionalidad básica del sitio es la siguiente:

1. En el sitio no hay cuentas para usuarios: toda la funcionalidad está disponible para cualquiera que lo visite.

2. De todas formas, cualquier visitante podrá reservar un nombre que no esté ya en uso para cuando suba información al sitio. Este nombre se mantendrá mientras el visitante utilice el mismo navegador.

3. La página principal del sitio mostrará a los visitantes un formulario para buscar películas, como documento HTML. Ese formulario permitirá especificar el nombre de una película. Una vez especificado, el sitio devolverá un listado de los actores de esa película, y un nuevo formulario como el anterior (por si el visitante quiere buscar actores de otra película). Este listado que aparecerá en la página principal incluirá, para cada película, su título (como enlace a la página de la película, ver más abajo), la imagen de su cartel y un listado delos actores que actúa en la película. Las películas aparecerán en orden aleatorio.

4. La página principal tendrá también, cuando el visitante no haya reservado nombre, un formulario para ponerlo. Si se rellena ese formulario, volverá a verse la página principal, pero ahora ya con el nombre reservado (salvo que el nombre ya estuviera reservado previamente, en ese caso volverá a ver el formulario). Desde ese momento en adelante, en la página principal se verá siempre el nombre reservado, mientras se conecte desde el mismo navegador.

5. Cada película tendrá una página. En esa página aparecerá el título de la película, una breve sinopsis, la imagen de su cartel, los comentarios que ha habido para esa película, y un formulario para poner un comentario. Los comentarios aparecerán en orden inverso de publicación (los más recientes primero).

6. Cada visitante puede poner un comentario utilizando el formulario indicado anteriormente. Cuando lo ponga, aparecerá a su nombre (si ha reservado uno), o como anónimo (si no lo ha hecho). Al poner un comentario, el visitante volverá a ver la página de la película que estaba viendo, pero ahora con el comentario que acaba de poner.

7. Todas las imágenes de los carteles que muestra PelisOchenteras se alojan en el propio PelisOchenteras.

8. Todas las páginas del sitio tendrán una imagen de cabecera (banner).

9. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio `tracking.com`. Cada una de estas imágenes se sirve con una URL diferente si está en una página diferente, pero igual si está en la misma página.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior.

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la URL de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario para elegir un nombre. Pero el nombre elegido ya está reservado, y el visitante ha de rellenar el formulario por segunda vez (esta vez, el nombre no estará previamente reservado). El escenario termina cuando el visitante ve de nuevo la página principal, ya con el nombre que ha reservado.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya ha reservado nombre y está viendo la página de una película en su navegador. El visitante rellena el formulario de comentarios, poniendo uno. El escenario termina cuando el visitante ve la página de la película ya con su comentario.

5. Se quiere añadir a PelisOchenteras la funcionalidad de "transferir" los nombres reservados a otro navegador. Describe un mecanismo que sirva para esto, basado en que el visitante pueda pedir alguna información a PelisOchenteras

para poder probar, desde otro navegador, que es la misma persona. El mecanismo no puede usar otra cosa que la interacción desde los dos navegadores (el navegador A, con el que tiene nombre reservado, y el navegador B, al que va a transferir el registro) con PelisOchenteras. Describe todas las interacciones HTTP en los dos navegadores desde que el visitante solicite transferencia del nombre en el navegador A, hasta que vea, desde el navegador B, la primera página del sitio con su nombre ya reservado.

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

23.4. Examen de ITT-SAT, 20 de mayo de 2022

Se quiere construir un sitio donde se puedan comentar programas de streaming, que llamaremos MiStreaming. El funcionamiento de MiStreaming seguirá el siguiente detalle:

1. Para ser usuario del sitio, habrá que tener una cuenta en él, y haber "entrado" en ella. Cuando un visitante intenta acceder al sitio sin haber entrado en una cuenta, se le muestran dos formularios: uno para acceder a una cuenta (introduciendo un nombre de usuario y una contraseña), y otro para crear una cuenta (con los mismos campos).

2. Si un usuario introduce en el formulario para acceder a una cuenta un nombre de usuario y una contraseña correctas, "entra" en esa cuenta.

3. Si un usuario introduce en el formulario para crear una cuenta un nombre de usuario que no existe y una contraseña, crea una nueva cuenta, y a la vez "entra" en ella.

4. Cada hora, MiStreaming descarga documentos JSON de los sitios web de las empresas de streaming, para actualizar la lista de programas disponibles. Entre la información que está incluida en esos documentos JSON se encuentra el nombre del programa, un identificador único de programa entre los de ese servicio de streaming, la descripción del programa, y la URL de una foto del programa. Se supone que cada servicio de streaming mantiene un documento JSON como el descrito. MiStreaming no descarga las fotos, sólo sus URLs.

5. Con la información que ha descargado de estos documentos JSON, MiStreaming mantiene una lista de programas, que se muestra a los usuarios en una página HTML. En esta página, los usuarios pueden ver todos los programas, y seleccionarlos (mediante un botón que realiza un POST sobre MiStreaming). Cuando un programa es seleccionado por un usuario, pasa a mostrarse en la página principal del sitio para ese usuario.

6. Los usuarios que han "entrado" en su cuenta pueden ver en la página principal del sitio una lista con los programas de streaming que han seleccionado. El nombre de cada uno de estos programas estará enlazado con la página del programa (descrita a continuación). Cada usuario sólo puede ver la página de los programas que ha seleccionado.

7. En la página de un programa se puede ver el nombre del programa, su descripción, su foto y los comentarios que cualquier usuario ha puesto sobre él. Cada comentario aparecerá acompañado del nombre de usuario de quien lo puso. Además, hay un formulario para poner un nuevo comentario (que tendrá un único campo: una cadena de texto).

8. En la página principal habrá además un recuadro con los cinco últimos comentarios a los programas que ha seleccionado el usuario (puestos por cualquier usuario), un enlace para salir de la cuenta (mediante un GET), y una imagen de cabecera con el logo del sitio.

9. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio `tracking.com`. Cada una de estas imágenes se sirve con una URL diferente si está en una página diferente, pero igual si está en la misma página.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior.

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos principales. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario: el escenario comienza cuando un visitante que no ha entrado aún en una cuenta, pone la URL de MiStreaming en su navegador, y accede a la página principal. En ella, introduce un usuario y una contraseña válidos, y pasa a ver el listado de sus programas elegidos en la página principal. El escenario termina cuando el usuario ve esta página con el listado en su navegador.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario: el escenario comienza con el usuario viendo en su navegador la página principal con sus programas elegidos. El usuario pulsa sobre uno de ellos, y pasa a ver la página de ese programa. En ella pone un comentario. El escenario termina cuando el usuario ve la página del programa con el nuevo comentario en ella.

5. Se quiere añadir un enlace en la página principal para que cada usuario pueda acceder a su lista de programas como un documento XML. Ese documento debe tener un listado con los nombres de todos los programas que ha elegido, y para cada programa, los cinco últimos comentarios que se han puesto para ese programa, y el nombre de usuario que puso el comentario. Escribe

un documento ejemplo de cómo sería ese XML, que incluya al menos dos programas y cuatro comentarios (en total).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

23.4.1. Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST Este podría ser el esquema REST:

Recurso	Método	Query String	Comentario
/	GET	–	Página principal (HTML)
/	POST	Registro (user=pepe&pass=pepon) o selección de programa (pid=123)	Permite registrarse o seleccionar programa, devuelve página principal (HTML)
/{id_programa}	GET	–	Página del programa (HTML)
/{id_programa}	POST	(comentario=Mola)	Envía comentario a programa
/banner.png	GET	–	Banner de la web

Para que el POST sobre el recurso principal, tendremos dos posibilidades. La primera es que se utilice para registrarse, en cuyo caso el query string tendrá dos variables (user y pass). La segunda es que se utilice para seleccionar un programa en la lista de usuario. En ese cas para que se pueda indentificar como datos para ese programa, se tendrá que incluir el identificador del programa en la query string. Para ello, incluiremos un campo oculto en el formulario de datos, justamente con ese dato.

No se incluye *path* para el XML del ejercicio 5.

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('banner.png', views.banner, name='banner'),
    path('<id_programa>', views.programa, name='programa'),
]
```

El recurso que sirve el banner podría servirse también configurando adecuadamente los mecanismos de Django para servirlo como fichero estático.

models.py Versión simplificada, que cumple el enunciado, aunque podría optimizarse:

```
from django.db import models

class Usuario(models.Model):
    usuario = models.CharField(max_length=16)
    contrasena = models.CharField(max_length=16)

class Plataforma(models.Model):
    plataforma = models.CharField(max_length=64)
    url_json = models.URLField()

class Programa(models.Model):
    plataforma = models.ForeignKey('Plataforma')
    nombre = models.CharField(max_length=64)
    descripcion = models.TextField()
    imagen = models.URLField()
```

```
class Comentario(models.Model):
    programa = models.ForeignKey('Programa')
    usuario = models.ForeignKey('Usuario')
    contenido = models.TextField()
    fecha = models.DateTimeField()

class Seleccion(models.Model):
    programa = models.ForeignKey('Programa')
    usuario = models.ForeignKey('Usuario')
```

Las plataformas las tenemos en la tabla "Plataforma", junto con la URL de donde podemos descargar su URL. A partir de esos JSON, obtenemos la información de los programas que se guardan en la tabla "Programa". Un usuario puede seleccionar programas, por lo que tendremos una tabla "Seleccion" donde podremos ver qué programas han sido seleccionados por qué usuarios. Lo mismo con la tabla "Comentarios".

Podríamos tener una tabla de "Session", pero nos vale con el usuario y mandar en la cookie el nombre de usuario.

No se incluyen los campos identificador único para cada tabla. Nótese que en este esquema utilizamos identificadores propios para los progarmas, no los proporcionados por el JSON de las plataformas.

Primer escenario Todas las interacciones son entre el navegador y el sitio.

- Petición GET de la página principal.

  ```
  GET / HTTP/1.1
  ...
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  ...

  [Página principal, HTML]
  ```

- Petición GET para el banner, originada debido a que está referenciada en la página HTML anterior.

```
GET /banner.png HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen con el banner]
```

- Petición GET para la imagen del sitio tracking.com, originada debido a que está referenciada en la página HTML anterior.

```
GET /principal.png HTTP/1.1
Host: tracking.com
...
```

- Respuesta

```
HTTP/1.1 200 OK
Set-Cookie: session_id=123456...
...

[Imagen de 1px]
```

- Petición POST para identificarse

```
POST /43 HTTP/1.1
...
user=pepe&pass=pepon
```

- Respuesta

```
HTTP/1.1 200 OK
Set-Cookie: usuario=pepe
...

[Página principal, HTML]
```

- Petición GET para el banner, originada debido a que está referenciada en la página HTML anterior.

```
GET /banner.png HTTP/1.1
Cookie: user=pepe
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen con el banner]
```

- Petición GET para la imagen del sitio tracking.com, originada debido a que está referenciada en la página HTML anterior.

```
GET /principal.png HTTP/1.1
Host: tracking.com
Cookie: session_id=123456...
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de 1px]
```

Segundo escenario A continuación, las interacciones son entre el navegador y el sitio. Suponemos que el visitante ya ha recibido la cookie al venir del primer escenario.

- Petición GET de la página de un programa (el de identificador 12).

```
GET /12 HTTP/1.1
Cookie: user=pepe
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, HTML]
```

- Petición GET para el banner, originada debido a que está referenciada en la página HTML anterior.

```
GET /banner.png HTTP/1.1
Cookie: user=pepe
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen con el banner]
```

- Petición GET para la imagen del sitio tracking.com, originada debido a que está referenciada en la página HTML anterior.

```
GET /12.png HTTP/1.1
Host: tracking.com
Cookie: session_id=123456...
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de 1px]
```

- Petición GET para la imagen del programa, originada debido a que está referenciada en la página HTML anterior (y que tenemos guardada en la tabla "Programa" del models.py).

```
GET /principal.png HTTP/1.1
Host: disneyplus.com
...
```

- Respuesta

```
HTTP/1.1 200 OK
[Potencialmente aquí disneyplus.com podria ponernos una cookie]
...

[Imagen]
```

- Petición POST para enviar un comentario

```
POST /12 HTTP/1.1
Cookie: user=pepe
...
comentario=Pedazo%20de%20pelicula
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página del programa 12 con el comentario recién incluido, HTML]
```

- Se repite la petición GET para el banner, originada debido a que está referenciada en la página HTML anterior.
- Se repite la petición GET para la imagen del sitio tracking.com, originada debido a que está referenciada en la página HTML anterior. s
- Se repite la ptición GET para la imagen del programa, originada debido a que está referenciada en la página HTML anterior (y que tenemos guardada en la tabla "Programa" del models.py).

Documento XML

```
<programas>
  <usuario>pepe</usuario>
  <programa>
    <nombre>La cenicienta</nombre>
    <comentarios>
      <comentario>
        <contenido>Me ha gustado mucho</contenido>
        <usuario>Jacinta</contenido>
      </comentario>
      <comentario>
        <contenido>Pichi, picha</contenido>
        <usuario>Gerardo</contenido>
      </comentario>
  </programa>
  <programa>
    <nombre>La Bella y la bestia</nombre>
    <comentarios>
      <comentario>
        <contenido>Me siento identificado... con la bestia</contenido>
        <usuario>Gumersindo</contenido>
      </comentario>
      <comentario>
        <contenido>Que bella es la bella</contenido>
        <usuario>Antonia</contenido>
      </comentario>
  </programa>
</programas>
```

23.5. Examen de IST-SARO, 28 de junio de 2022

Se quiere construir un sitio donde se puedan puntuar partidos de tenis, que llamaremos BestTennis. El funcionamiento de BestTennis seguirá el siguiente detalle:

1. Para ser usuario del sitio, habrá que tener una cuenta en él, y haber "entrado" en ella. Cuando un visitante intenta acceder al sitio sin haber entrado en una cuenta, se le muestran dos formularios: uno para acceder a una cuenta (introduciendo un nombre de usuario y una contraseña), y otro para crear una cuenta (con los mismos campos).

2. Si un usuario introduce en el formulario para acceder a una cuenta un nombre de usuario y una contraseña correctas, "entra" en esa cuenta.

3. Si un usuario introduce en el formulario para crear una cuenta un nombre de usuario que no existe y una contraseña, crea una nueva cuenta, y a la vez "entra" en ella.

4. BestTennis mantiene una lista de partidos históricos de tenis, que se muestra a los usuarios en una página HTML. En esta página, los usuarios pueden ver todos los partidos, y seleccionarlos (mediante un botón que realiza un POST sobre BestTennis). Cuando un partido es seleccionado por un usuario, pasa a mostrarse en la página principal del sitio para ese usuario.

5. Los usuarios que han "entrado" en su cuenta pueden ver en la página principal del sitio una lista con los partidos de tenis que han seleccionado. Cada uno de estos partidos estará enlazado con la página del partido (descrita a continuación). Cada usuario sólo puede ver la página de los partidos que ha seleccionado.

6. En la página de un partido se puede ver el nombre del campeonato, los contendientes, un breve relato, una foto y la puntuación que cualquier usuario ha puesto sobre él. Cada puntuación aparecerá acompañado del nombre de usuario de quien lo puso. Además, hay un formulario para poner una nueva puntuación (que tendrá un único campo: un valor de uno a cinco).

7. En la página principal habrá además un recuadro con las cinco últimas puntuaciones a los partidos que ha seleccionado el usuario (puestos por cualquier usuario), un enlace para salir de la cuenta (mediante un GET), y una imagen de cabecera con el logo del sitio.

8. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio `tracking.com`. Cada una de estas imágenes se sirve con una URL diferente si está en una página diferente, pero igual si está en la misma página.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos

en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior.

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos principales. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario: el escenario comienza cuando un visitante que no ha entrado aún en una cuenta, pone la URL de BestTennis en su navegador, y accede a la página principal. En ella, introduce un usuario y una contraseña válidos, y pasa a ver el listado de sus programas elegidos en la página principal. El escenario termina cuando el usuario ve esta página con el listado en su navegador.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario: el escenario comienza con el usuario viendo en su navegador la página principal con sus programas elegidos. El usuario pulsa sobre uno de ellos, y pasa a ver la página de ese programa. En ella pone una puntuación. El escenario termina cuando el usuario ve la página del programa con el nueva puntuación en ella.

5. Se quiere añadir un enlace en la página principal para que cada usuario pueda acceder a su lista de programas como un documento XML. Ese documento debe tener un listado con los nombres de todos los programas que ha elegido, y para cada programa, las cinco últimas puntuaciones que se han puesto para ese partido, y el nombre de usuario que puso la puntuacioń. Escribe un documento ejemplo de cómo sería ese XML, que incluya al menos dos partidos y cuatro puntuaciones (en total).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

23.6. Examen de IST-SARO, 24 de mayo de 2022

Se quiere construir un sitio web, ComerFinoFino, donde se puede ver y comentar información sobre restaurantes de lujo. La funcionalidad básica del sitio es la siguiente:

1. En el sitio no hay cuentas para usuarios: toda la funcionalidad está disponible para cualquiera que lo visite.
2. De todas formas, cualquier visitante podrá reservar un nombre que no esté ya en uso para cuando suba información al sitio. Este nombre se mantendrá mientras el visitante utilice el mismo navegador.
3. La página principal del sitio mostrará a los visitantes un formulario para buscar restaurantes, como documento HTML. Ese formulario permitirá especificar el nombre de una ciudad. Una vez especificado, el sitio devolverá un listado de los restaurantes de lujo en esa ciudad, y un nuevo formulario como el anterior (por si el visitante quiere buscar restaurantes en otra ciudad). Este listado que aparecerá en la página principal incluirá, para cada restaurante, su nombre (como enlace a la página del restaurante, ver más abajo), y una foto. Los restaurantes aparecerán en orden aleatorio.
4. La página principal tendrá también, cuando el visitante no haya reservado nombre, un formulario para ponerlo. Si se rellena ese formulario, volverá a verse la página principal, pero ahora ya con el nombre reservado (salvo que

el nombre ya estuviera reservado previamente, en ese caso volverá a ver el formulario). Desde ese momento en adelante, en la página principal se verá siempre el nombre reservado, mientras se conecte desde el mismo navegador.

5. Cada restaurante tendrá una página. En esa página aparecerá el nombre del restaurante, una breve descripción, una foto, los comentarios que ha habido para ese restaurante, y un formulario para poner un comentario. Los comentarios aparecerán en orden inverso de publicación (los más recientes primero).

6. Cada visitante puede poner un comentario utilizando el formulario indicado anteriormente. Cuando lo ponga, aparecerá a su nombre (si ha reservado uno), o como anónimo (si no lo ha hecho). Al poner un comentario, el visitante volverá a ver la página del restaurante que estaba viendo, pero ahora con el comentario que acaba de poner.

7. Todas las fotos que muestra ComerFinoFino se alojan en el propio ComerFinoFino.

8. Todas las páginas del sitio tendrán una imagen de cabecera (banner).

9. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio `tracking.com`. Cada una de estas imágenes se sirve con una URL diferente si está en una página diferente, pero igual si está en la misma página.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior.

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la URL de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario para elegir un nombre. Pero el nombre elegido ya está reservado, y el visitante ha de rellenar el formulario por segunda vez (esta vez, el nombre no estará previamente reservado). El escenario termina cuando el visitante ve de nuevo la página principal, ya con el nombre que ha reservado.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya ha reservado nombre y está viendo la página de un restaurante en su navegador. El visitante rellena el formulario de comentarios, poniendo uno. El escenario termina cuando el visitante ve la página del restaurante ya con su comentario.

5. Se quiere añadir a ComerFinoFino la funcionalidad de "transferir" los nombres reservados a otro navegador. Describe un mecanismo que sirva para esto, basado en que el visitante pueda pedir alguna información a ComerFinoFino para poder probar, desde otro navegador, que es la misma persona. El mecanismo no puede usar otra cosa que la interacción desde los dos navegadores (el navegador A, con el que tiene nombre reservado, y el navegador B, al que va a transferir el registro) con ComerFinoFino. Describe todas las interacciones HTTP en los dos navegadores desde que el visitante solicite transferencia del nombre en el navegador A, hasta que vea, desde el navegador B, la primera página del sitio con su nombre ya reservado.

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP

- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

23.7. Examen de IT-ST, 1 de julio de 2022

Se quiere construir un sitio web, Encuestam.os, donde cualquier persona pueda proponer encuestas simples, que otros puedan contestar. La funcionalidad básica del sitio es la siguiente:

1. Una encuesta en Encuestam.os será una pregunta (como mucho de 200 caracteres) a la que se puede contestar "Sí", "No" o "No sé". Por ejemplo, una encuesta podría ser "¿Ha sido difícil el exámen de ST?". Ante esta encuesta, las respuestas se ofrecerían como tres botones (uno para "Sí", otro para "No", otro para "No sé"), que los visitantes podrían pulsar para responder la encuesta.

2. Cualquier visitante puede proponer encuestas, ver sus propias encuestas y las que hayan propuesto otros visitantes, y contestar las encuestas que hayan propuesto otros visitantes (pero no puede contestar las encuestas que ha propuesto).

3. Una vez que un visitante haya respondido una encuesta, ya no puede volver a responderla, sólo ver sus resultados.

4. Cada encuesta se mostrará a un visitante mostrando su pregunta y:

    - Si el visitante puede responder, los tres botones para responder.
    - Si el visitante no puede responder, el número de respuestas para las tres categorías ("Sí", "No" y "No sé").

5. La página principal del sitio mostrará el listado con todas las encuestas, mostradas tal y como se indica en el apartado anterior.

6. Cuando un visitante responda una encuesta, pulsando el botón correspondienet, se contabilizará su respuesta y se le volverá a presentar la página principal, con todas las encuestas (incluyendo la que acaba de responder, pero ya con resultados en lugar de botones para contestar).

7. Además, el sitio ofrecerá una página (recurso) de comentarios. Esta página incluirá un formulario para enviar comentarios, y todos los comentarios que se hayan enviado, por cualquier visitante, hasta ese momento. Un comentario será un texto de menos de 100 caracteres. Cuando un usuario rellene el formulario con un nuevo comentario, y lo envíe, recibirá como respueta la misma página de comentarios, con su comentario ya puesto (y todos los demás)

8. La página principal incluirá un enlace a la página de comentarios, y la página de comentarios un enlace a la página principal.

9. Las dos páginas (principal y de comentarios) incluirán una imagen de promoción del sitio "CosasRic.As", que servirá el sitio web "CosasRic.As". La URL de esa imagen será la misma para las dos páginas.

10. Las dos páginas HTML del sitio usarán un único documento CSS, servido por el propio sitio, que definirá su estilo de forma uniforme.

En esta descripción, considérese que un "visitante" corresponde con un cierto navegador, con su propio almacén de cookies. Esto es, una misma persona que use dos navegadores distintos, cada uno con su propio almacén de cookies, será considerada como dos visitantes, mientras que varias personas usando el mismo navegador, con un único almacén de cookies para todas ellas, serán consideradas como un único visitante.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la

primera línea de la cabecera). Escribe también un fichero similar al fichero `urls.py` de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación, incluyendo la clase "Usuario" que se describe más adelante. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos los campos o tablas necesarios para garantizar que los visitantes no pueden responder a sus propias encuestas, o a una encuesta que ya han respondido, y en general, que se cumplen las condiciones del enunciado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero `models.py` en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). (1 punto)

3. Describe las interacciones HTTP que ocurrirán entre el navegador, el servidor web (la aplicación Encuestam.os) y cualquier otro servidor que pueda ser necesario en el siguiente escenario. El escenario comienza cuando un visitante que nunca ha visitado Preguntam.os escribe en la barra de URLs de su navegdor la dirección del sitio. A continuación, ve en su navegador la página principal del sitio, y el formulario que hay en esta página para poner una encuesta. Escribe en él la pregunta, y pulsa el botón para enviarlo. El escenario termina cuando el visitante recibe de nuevo la página principal, ya con la nueva encuesta en ella. (1 punto)

4. Se quiere ofrecer, en la página principal, un enlace con el texto "Nuevo visitante", que cuando se pulse, vuelva a mostrar la página principal, pero como si estuviera accediendo a ella un nuevo visitante (ahora no se considerará que este navegador haya creado ninguna encuesta, ni respondido a ninguna, ni puesto ningún comentario). ¿Qué cambios habría que hacer al esquema REST que se respondió en la primera pregunta para que esto funcionase? ¿Qué ocurrará desde que se pulse ese enalce hasta que se vuelva a ver la página principal, ya como nuevo visitante? Importante: para el resto de los ejercicios de este examen, hay que suponer que este enlace no existe. (1 punto)

5. ¿Podría el sitio CosasRic.as llevar una cuenta de cuántos visitantes distintios diarios están viendo en sus navegadores la página principal de Encuestam.os, y cuántos visitantes dsitintos diarios están viendo la página de comentarios?. ¿Podría llevar una cuenta de cuántos visitantes distintos diarios están viendo en sus navegdores al menos una de las dos páginas? En cualquiera de los

dos casos, si crees que sí, explica cómo, y si crees que no, explica porqué. Importante: para el resto de los ejercicios de este examen, hay que suponer que CosasRic.as no trata de llevar ninguna cuenta en particular, sólo sirve la imagen. (1 punto)

En las dos preguntas sobre interacciones HTTP, para cada interacción HTTP indica claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita en qué parte de la petición HTTP van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando, y que el navegador acepta cualquier cookie que se le envíe, y nunca la borra salvo que la cookie expire.

23.7.1. Ejercicio 1 (solución)

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Nueva encuesta	`question="..."`
/comments	GET	Página de comentarios	
/comments	POST	Nuevo comentario	`text="..."`
/style.css	GET	Documento CSS	

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.main, name='main')
  path('comments', views.comments, name='comments')
  path('style.css', views.css, name='css')
]
```

El documento `style.css` puede servirse también directamente como fichero estáticos, con las facilidades que Django proporciona para ello.

23.7.2. Ejercicio 2 (solución)

Hay varias soluciones, una podría ser:

```
from django.db import models

class Visitante (models.Model):
  cookie = models.CharField(max_lenght = 60)

class Encuesta (models.Model):
  pregunta =  models.TextField(max_lenght = 200)
  autor =     models.ForeignKey('Visitante')

class Respuesta (models.Model):
  encuesta  = models.ForeignKey('Encuesta')
  respuesta = models.CharField(
    max_length=1,
    choices=[('S', 'Sí'), ('N', 'No'), ('X', 'No sé')]
  )
  autor     = models.ForeignKey('Visitante')

class Comentario (models.Model):
  texto =    models.TextField(max_lenght = 100)
```

La tabla `Visitante` se usa para identificar a los visitantes que hace falta identificar, almacenando la cookie que se les ha enviado.

La tabla `Encuesta` necesita un campo `autor` para evitar que un visitante pueda responder a una encuesta que ha creado. Igualmente, el campo `autor` en `Respuesta` se necesita para evitar que un visitante responda varias veces a una misma encuesta.

El campo `respuesta` en `Respuesta` puede ser simplemente un campo de texto "normal". En esta respuesta se fuerzan valores concretos (con `choices`) porque eso permite simplificar el código.

`Comenario` no requiere un campo que relacione con el visitante que lo puso, porque no hay ninguna limitación sobre la puesta de comentarios.

23.7.3. Ejercicio 3 (solución)

Como según el enunciado el usuario no ha visitado nunca el sitio, no se le ha enviado nunca una cookie, por lo que la primera interacción claramente no va a incluir una cookie en la petición. El escenario consistirá en un GET para obtener la página principal, y un POST para crear la encuesta. La respuesta al POST deberá incluir una cookie para identificar al navegador, pues habrá que evitar ya que este visitante responda a esta encuesta. A partir de este momento, todas las peticiones del navegador a Encuestam.os incluirán la cookie (pero no las que se hacen a CosasRic.as). Además, serán necesarias interacciones para obtener los demás elementos de la página principal (documento CSS de Encuestam.os e imagen de CosasRic.as).

Si no se dice otra cosa, cada una de las siguientes interacciones son entre el navegador y el servidor Encuestam.os.

- Petición GET a la página principal, para verla.

  ```
  GET / HTTP/1.1
  ...
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  ...

  [Página principal, con la nueva pregunta, HTML]
  ```

- Petición GET para obtener el documento CSS.

  ```
  GET /style.css HTTP/1.1
  ...
  ```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición GET para obtener la imagen (esta petición se hará a CosasRic.as). Se supone que el nombre de recurso de la imagen es `/imagen`.

```
GET /imagen HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen]
```

- Petición POST a la página principal, para crear la encuesta. Suponemos que `yyy` es el texto de la pregunta.

```
POST / HTTP/1.1
...

question="yyy"
```

- Respuesta

```
HTTP/1.1 200 OK
...
Set-Cookie: id=fad32-dff132-54weee; Expires=Wed, 11 Oct 2022 07:29:00 GMT
...

[Página principal, con la nueva encuesta, HTML]
```

- Petición GET para obtener el documento CSS.

```
GET /style.css HTTP/1.1
...
Cookie: id=fad32-dff132-54weee
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición GET para obtener la imagen (esta petición se hará a CosasRic.as). Exactamente igual que antes.

23.7.4. Ejercicio 4 (solución)

El enlace que se ve como "Nuevo visitante" ha de apuntar a algún sitio, de forma que cuando el visitante pulse sobre él, vaya a ese recurso. Por lo tanto, añadimos al esquema REST el recurso `/nuevo`.

Cuando nuestra aplicación reciba una petición sobre ese recurso, reconocerá que se quiere funcionar como nuevo visitante, y borrará la entrada correspondiente a la cookie que le ha llegado (si le ha llegado alguna) de la tabla `Visitantes`. En la respuesta puede enviar una cookie `id` vacía, para que el navegador la anule, pero eso estrictaemnte no hace falta (si recibe la cookie antigua, la aplicación la ignorará, porque ya no la tiene en la tabla `Visitantes`.

Como después de pulsar el enlace el visitante tiene que volver a ver la página rprincipal, pero la petición se ha hecho al recurso `/nuevo`, la respuesta que enviará la aplicación después de borrar la entrada correspondiente a la cookie tendrá que ser un código de redirección, por ejemplo "303 See Other".

La tabla de recursos de nuestra aplicación, por tanto, tendría una nueva entrada:

Recurso	Método	Semántica
...	...	...
/nuevo	GET	Borrado de cookie. Responde "303 See Other" con cabecera "Location: /"

23.7.5. Ejercicio 5 (solución)

Primera pregunta. No podría llevar una cuenta distinta para las dos páginas, porque a CosasRic.as le van a pedir siempre el mismo nombre de recurso, y por tanto no va a poder discriminar cuándo el visitante está viendo la página principal, y cuándo la de comentarios.

Segunda pregunta. Sí puede hacerlo. Basta con que cada vez que reciba una petición al recurso de la imagen, si esta petición no lleva cookie, devuelva una que sea única. Para llevar la cuenta de los visitantes diarios (que hayan visto cualquiera de los dos recursos de Encuestam.os) basta con que apunte en una tabla cada

cookie que va viendo (bien porque le llegan en peticiones, bien porque las devuelve en respuestas), si no la tiena ya en la tabla. Esa tabla se borra completamente al comenzar el día, y el número de cookies que tenga al terminar el día será el número de visitantes que se quiere calcular para ese día.

23.8. Examen de IT-ST, 27 de mayo de 2022

Se quiere construir un sitio web, Pregunt.as, donde cualquier persona pueda hacer preguntas, que otros puedan contestar. La funcionalidad básica del sitio es la siguiente:

1. Cualquier visitante puede hacer preguntas, pero sólo los usuarios autenticados pueden poner respuestas.

2. La página principal del sitio mostrará el listado con las 10 últimas preguntas realizadas. Para cada una de ellas se mostrará el título (que será un enlace a la página de la pregunta en Pregunt.as), y el número de respuestas que ha recibido.

3. Además, la página principal también tendrá un formulario para subir una nueva pregunta, con un solo campo de texto (para escribir la pregunta).

4. La página principal tendrá también un formulario, si el visitante no se ha autenticado, para autenticarse (se supone que el registro en el sitio se ha realizado previamente por algún mecanismo que no es de interés para este ejercicio). Ese formulario tendrá un campo para el nombre de usuario, otro para una contraseña, y un botón para enviarlos: si se envían correctamente, el usuario ya registrado pasará a ver la misma página principal, pero ya como usuario autenticado. Si el usuario ya está autenticado, en lugar de ese formulario verá un botón para salir de la sesión ("desautenticarse"): si lo pulsa, volverá a ver la página principal, pero ya como visitante sin autenticar.

5. La página de cada pregunta incluye el texto de la pregunta, y el listado de las respuestas que se han puesto (cada una, junto con el autor de la respuesta). En caso de que la página se muestre a un usuario autenticado, además aparecerá un formulario para poner una respuesta, sólo si ese usuario no ha puesto ya una respuesta (cada usuario registrado sólo podrá poner una respuesta a una pregunta dada). Ese formulario tendrá un campo de texto (para poner la respuesta) y un botón para enviar la respuesta. Cuando se pulse el botón, se obtendrá como resultado la misma página de la pregunta que se estaba viendo, pero ya con la nueva respuesta.

6. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el propio sitio. La imagen será siempre la misma (el recurso del sitio que sirve la imagen será siempre el mismo).

7. Todas las páginas HTML del sitio usarán un único documento CSS, servido por el propio sitio, que definirá el estilo de todas ellas de forma uniforme.

8. Pregunt.as servirá también un documento JSON con el listado completo de preguntas, indicando para cada pregunta el texto de esa pregunta, y la URL de la página de la pregunta correspondiente.

En esta descripción, considérese que un "visitante" corresponde con un cierto navegador, con su propio almacén de cookies. Esto es, una misma persona que use dos navegadores distintos, cada uno con su propio almacén de cookies, será considerada como dos visitantes, mientras que varias personas usando el mismo navegador, con un único almacén de cookies para todas ellas, serán consideradas como un único visitante. Un "usuario autenticado" será un visitante que se ha autenticado vía el formulario de autenticación correspondiente.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas URLs (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero `urls.py` de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación, incluyendo la clase "Usuario" que se describe más adelante. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos los campos o tablas necesarios para garantizar que usuario sólo responde a cada pregunta como mucho con una respuesta. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero `models.py` en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). Se supondrá que hay

ya una tabla de usuarios registrados con el nombre de usuario y la contraseña para cada uno de ellos (esa tabla se llamará "usuarios", correspondiente a la clase "Usuario"). Para este modelo de datos, supóngase que no se usan las facilidades de Django para gestión de usuarios, sino que todo se basa en esta tabla de usuarios y las demás tablas que incluyas en el modelo de datos. (1 punto)

3. Describe las interacciones HTTP que ocurrirán entre el navegador y el servidor web (la aplicación Pregunt.as) en el siguiente escenario. El escenario comienza cuando un visitante sin autenticar está viendo en su navegador la página principal del sitio. El visitante ve el formulario que hay en esta página para poner una pregunta, escribe en él una pregunta, y pulsa el botón para enviarlo. El escenario termina cuando el visitante recibe de nuevo la página principal, ya con la nueva pregunta en ella.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y el servidor web (la aplicación Pregunt.as) en el siguiente escenario. El escenario comienza cuando el visitante anterior está viendo la página principal con la pregunta que acaba de poner. Rellena el formulario de autenticación (usuario y contraseña), y pulsa el botón para enviarlo. Cuando recibe de nuevo la página principal, ya autenticado, pulsa la página de la primera pregunta que aparece. El escenario termina cuando el visitante recibe la página de esa pregunta.

5. Escribe cómo podría ser un documento JSON como los que sirve Pregunt.as, si tenemos tres preguntas almacenadas en su base de datos. (1 punto)

En las dos preguntas sobre interacciones HTTP, para cada interacción HTTP indica claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando, y que el navegador acepta cualquier cookie que se le envíe, y nunca la borra salvo que la cookie expire.

23.8.1. Ejercicio 1 (solución)

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Nueva pregunta	`text="..."`
/	POST	Autenticación	`user=...&passwd=...`
/	POST	Cierre sesión	`cerrar=true`
/{id_pregunta}	GET	Página de una pregunta	
/{id_pregunta}	POST	Nueva respuesta	`texto="..."`
/pixel.png	GET	Imagen de un pixel	
/main.css	GET	Documento CSS	
/main.json	GET	Documento JSON	

Como no se indica mucho sobre cómo se cierra la sesión, podría hacerse sobre un recurso separado también, en lugar de sobre el recurso principal, con un valor que iría en un campo escondido del formulario con el botón para cerrar la sesión.

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index')
  path('<id_pregunta>', views.pregunta, name='pregunta')
  path('pixel.png', views.pixel, name='pixel')
  path('main.css', views.css, name='css')
  path('main.json', views.json, name='json')
]
```

Tanto `pixel.json` como `main.css` pueden servirse también como ficheros estáticos, con las facilidades que Django proporciona para ello.

El documento JSON podría servirse también si se invoca el recurso / con una query string (por ejemplo, `/?format=json`).

23.8.2. Ejercicio 2 (solución)

Hay varias soluciones, una podría ser:

```
from django.db import models

class Usuario (models.Model):
  nombre =  models.CharField(max_lenght = 20)
  palabra = models.CharField(max_lenght = 20)

class Sesion (models.Model):
  cookie = models.CharField(max_lenght = 60)
  usuario = models.ForeignKey('Usuario')

class Pregunta (models.Model):
  texto = models.TextField()
  fecha = models.DateTimeField()

class Respuesta (models.Model):
  texto =    models.TextField()
  autor =    models.ForeignKey('Usuario')
  pregunta = models.ForeignKey('Pregunta')
```

La tabla `Sesion` se usa para llevar cuenta de los usuarios registrados, y darles una cookie a cada uno de ellos. No se puede incluir la cookie en la tabla `Usuario` porque en ese caso un usuario no se podría autenticar simultáneamente desde varios navegadores, algo que no prohibe el enunciado.

La tabla `Pregunta` necesita un campo `fecha` para poder ordenar las preguntas para la página principal.

Como nombre de recurso para cada pregunta se podría utilizar el valor del campo `id` de la tabla `Pregunta`, que Django genera automáticamente como un entero único, al no habe especificado ningún campo como clave.

23.8.3. Ejercicio 3 (solución)

Como según el enunciado el usuario no está autenticado, no ha hecho falta hasta el momento enviarle cookie de sesión, así que no se habrá hecho, y por lo tanto no habrá cookies en la interacción. Como ya está viendo la página principal, el escenario consistirá solamenete en un POST para subir los datos del formulario

con la pregnta, su respueta, y las interacciones derivadas del documento HTML que recibirá en el cuerpo de esta respuesta.

- Petición POST a la página principal, para subir la pregunta. Suponemos que `yyy` es el texto de la pregunta.

```
POST / HTTP/1.1
...

text="yyy"
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, con la nueva pregunta, HTML]
```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache):

```
GET /main.css HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición de la imagen de un pixel (suponemos que no se usa la cache):

```
GET /pixel.png HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen, PNG]
```

23.8.4. Ejercicio 4 (solución)

Como según el enunciado el usuario no está autenticado, no ha hecho falta hasta el momento enviarle cookie de sesión, así que no se habrá hecho, y por lo tanto no habrá cookies en la interacción. Como ya está viendo la página principal, el escenario consistirá en un POST para subir los datos de autenticación, su respueta, que incluirá el envío de una cookie de sesión para que las siguientes interacciones con el navegador se puedan identificar como autenticads, las interacciones derivadas del documento HTML que recibirá en el cuerpo de esta respuesta. A continuación, habrá un GET para acceder a la página de la noticia, que igualmente provocará interacciones derivadas del documento HTML que recibirá en el cuerpo de la respuesta

- Petición POST a la página principal, para autenticarse. Suponemos que `xxx` es el nombre de usuario y `yyy` es la contraseña.

  ```
  POST / HTTP/1.1
  ...

  user=xxx&passwd=yyy
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  ...
  Set-Cookie: id=32fad-32dff1-weee54; Expires=Wed, 11 Oct 2022 07:28:00 GMT
  ...

  [Página principal, sesión ya autenticada, HTML]
  ```

  (la cookie debería enviarse con el atributo "Secure", para evitar que se envíe por conexiones sin cifrar, por lo que esa cabecera sería más correcta así:

  ```
  Set-Cookie: id=32fad-32dff1-weee54; Secure; Expires=Wed, 11 Oct 2022 07:28:0
  ```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache):

  ```
  GET /main.css HTTP/1.1
  Cookie: id=32fad-32dff1-weee54
  ...
  ```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición de la imagen de un pixel (suponemos que no se usa la cache):

```
GET /pixel.png HTTP/1.1
Cookie: id=32fad-32dff1-weee54
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen, PNG]
```

- Petición GET de la página de la pregunta. Suponemos que el identificado rde la pregunta es "56443"..

```
GET /56443 HTTP/1.1
Cookie: id=32fad-32dff1-weee54
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página de la pregunta, sesión ya autenticada, HTML]
```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache): igual que la anterior petición de hoja de estilo, con la misma respuesta.

- Petición de la imagen de un pixel (suponemos que no se usa la cache): igual que la anterior petición de imagen, con la misma respuesta.

  Después de la primera respuesta, las demás respuestas no llevan cabecera `Set-Cookie` porque no es necesario reenviar la cookie, ya la tiene el navegador. Podrían volver a enviarla si se quisiera actualizar su fecha de expiración, o realizar alguna otra acción de refresco. Las peticiones, una vez el navegador recibió la cookie, incluyebn siempre una cabecera `Cookie` porque el navegador la reenviará siempre al servidor que se la envió.

23.8.5. Ejercicio 5 (solución)

Hay varias formas en que se puede construir el documento JSON, una de ellas puede ser:

```
[
  {"text": "¿Cuál es el sentido de la vida, el universo y todo lo demás?",
   "url": "https://pregunt.as/1"},
  {"text": "¿Cuál es la pregunta cuya respuesta es 42?",
   "url": "https://pregunt.as/2"},
  {"text": "¿Cuál es al especie más inteligente de la Tierra?",
   "url": "https://pregunt.as/3"}
]
```

23.9. Examen de ITT-SARO, 6 de junio de 2021

Se quiere construir un sitio web, Vide.os, donde se puede votar cualquier vídeo de YouTube. La funcionalidad básica del sitio es la siguiente:

1. Toda la funcionalidad del sitio está disponible para cualquier visitante: no hay cuentas, ni hace falta acreditarse para tener acceso a la funcionalidad que proporciona.

2. La página principal del sitio mostrará el listado con los 10 vídeos con más votos en cada momento. Para cada una de ellos se mostrará el título (que será un enlace a la página del vídeo en Vide.os), y el número de votos que ha recibido ese vídeo.

3. Además, la página principal también tendrá un formulario para subir el enlace a un nuevo vídeo, con dos campos: un enlace a un vídeo en YouTube, y un nombre de visitante. A partir del enlace, Vide.os obtendrá de YouTube el título del vídeo en cuestión, usando la API de YouTube. También habrá un formulario con un campo para poner un nombre de visitante, y un botón para

votar el video. El campo para poner el nombre de visitante sólo aparecerá si el visitante no ha rellenado un campo de nombre de visitante antes: en caso de que lo haya hecho, aparecerá ese nombre, y no podrá cambiarse. Al pulsar el botón para votar se enviará el voto, y en su caso, el nombre. Si un visitante pulsa el botón en un video que ya ha sido votado por ese mismo visitante, no se contabilizará el voto. No se aceptarán enlaces a vídeos de visitantes que no hayan rellenado el campo de nombre de visitante (o ya tuvieran nombre).

4. La página de cada vídeo incluye el título del vídeo (que será un enlace a el vídeo en YouTube), el listado de los nombres de visitante de todos los que han votado el vídeo, y un formulario para poner el nombre de visitante sólo aparecerá si el visitante no ha rellenado un campo de nombre de visitante antes: en caso de que lo haya hecho, aparecerá ese nombre, y no podrá cambiarse. No se aceptarán votos de visitantes que no hayan rellenado el campo de nombre de visitante.

5. Puede ocurrir que dos o más visitantes elijan el mismo nombre de visitante: simplemente, tendremos un nombre repetido.

6. No se permitirá que el mismo enlace a un vídeo sea subido por más de un visitante. Tampoco que un mismo visitante vote más de una vez a un vídeo.

7. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio Trazad.or. Cada una de estas imágenes se sirve con una url diferente si está en una página diferente, pero igual si está en la misma página.

8. Todas las páginas HTML del sitio usarán un único documento CSS, servido por el propio sitio, que definirá el estilo de todas ellas de forma uniforme.

9. Vide.os servirá también un documento XML con el listado completo de vídeos, indicando para cada uno de ellos su url, su título, y el número de votos, ordenador por fecha de último voto.

En esta descripción, considérese que un "visitante" corresponde con un cierto navegador, con su propio almacén de cookies. Esto es, una misma persona que use dos navegadores distintos, cada uno con su propio almacén de cookies, será considerada como dos visitantes, mientras que varias personas usando el mismo navegador, con un único almacén de cookies para todas ellas, serán consideradas como un único visitante.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos los campos o tablas necesarios para garantizar que cada video sólo se sube una vez, y que cada visitante vota como mucho una vez un video dado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). Para este modelo de datos, supóngase que el nombre de visitante se gestionará en esta base de datos. (1 punto)

3. Explica si sería posible usar una cookie para gestionar el nombre de visitante en lugar de almacenarlo en la base de datos. Esto es, si usando la cookie como almacén de datos, podríamos poner el nombre de visitante en él, en lugar de en uno o varios registros de la base de datos. Si crees que sería posible, explica cómo y cuándo se crearía esta cookie, y qué cambios habría que hacer al modelo de datos que has diseñado para el apartado anterior. Si crees que no, explica por qué. En cualquier caso, indica claramente si crees que se puede, o que no se puede.

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante está viendo en su navegador la página principal del sitio. El visitante ve el formulario que hay en esta página, escribe en él la url de un vídeo de YouTube, y pulsa el botón para enviarlo. El visitante recibe de nuevo la página principal, ya con el nuevo vídeo en ella. A continuación, pulsa sobre el enlace de ese vídeo. El escenario termina cuando el usuario ve en su

navegador la nueva página que le ha llegado del servidor, con la página del vídeo.

Para cada interacción HTTP indica claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando, y que el navegador acepta cualquier cookie que se le envíe, y nunca la borra salvo que la cookie expire.

Para este apartado, considera que el nombre de visitante se almacena en la base de datos, no en una cookie. (1 punto)

5. ¿Podría el sitio Trazad.or conocer qué páginas de Vide.os está visitando cada visitante de Vide.os, aunque no sepa a qué persona concreta corresponde? Explica cómo podría hacerlo, incluyendo (si fuera relevante) el intercambio de cookies, las cabeceras HTTP imprescindibles, y cualquier otro detalle que te parezca relevante. (1 punto)

23.9.1. Ejercicio 1 (solución)

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Nuevo video	`nombre="..."&url="..."` (*)
/{id_video}	GET	Página de un video	
/{id_video}	POST	Nuevo voto	`nombre="..."` (*)
/todo.xml	GET	Docuento XML	
/main.css	GET	Documento CSS	

(*) El campo `nombre` sólo es necesario la primera vez que un visitante invoque POST, pero tampoco hace daño si se incluye el nombre siempre (en ese caso deberá coincidir con el que el visitante especificó la primera vez, pero el enunciado no dice qué hacer si no coinciden).

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index')
  path('<id_video>', views.video, name='video')
  path('main.css', views.css, name='css')
  path('todo.xml', views.todo, name='xml')
]
```

No tiene que haber un recurso para la imagen que sirve Trazad.or, precismaente porque la sirve ese otro sitio web.

El documento XML podría servirse también si se invoca el recurso / con una query string (por ejemplo, `/?format=xml`).

Rúbrica

Para estar correcto, el ejercicio ha de incluir tanto una tabla con un documento `urls.py` correctos, completos, y sin recursos innecesarios.

Indicaciones aproximadas de errores:

- No hay columna de datos en la tabla, o está incorrecta: -0.2
- No hay recurso para documento CSS: -0.3
- No hay recurso para documento XML: -0.3
- Hay recurso para la imagen servida por Trazad.or: -0.3
- No coincide la tabla con el fichero `urls.py`: -0.2

23.9.2. Ejercicio 2 (solución)

Hay varias soluciones, una podría ser:

```
from django.db import models

class Visitante (models.Model):
  id = models.CharField(max_lenght = 50)
  nombre = models.CharField(max_lenght = 30)

class Video (models.Model):
  titulo: models.CharField(max_lenght = 80)
  url: models.URLField()

class Voto (models.Model):
  video: models.ForeignKey('Video')
  autor: models.ForeignKey('Visitante')
  fecha: models.DateTimeField()
```

La tabla `Sesion` se usa para llevar cuenta de los visitantes (navegadores distintos), y su id será el valor de la cookie que se les envíe. El campo `nombre` será el nombre del visitante, si ya lo ha especificado.

La tabla `Video` almacena el título y la url de todos los videos. Nonecesita un campo que apunte a `Visitante` porque no se usa para nada el visitante que subió un video. Podría incluirse uno `fecha`, con la fecha del último voto, como se explica a continuación (si no se incluye fecha en `Voto`. Se podría poner un campo `votos`, con el número de votos del video, pero este número se puede obtener fácilmente de la tabla `Voto`, contando los que apunten (tengan como "foreign key") el video en cuestión.

La tabla `Voto` incluye referencias al video al que se vota, y al visitante que lo ha votado. También incluye un campo `fecha` que sería necesario para poder ordenar los votos por la fecha del último voto en el documento XML. También se podria utilizar un campo similar en `Video`, que se actualizaría cada vez que se vote un video.

Rúbrica

- No están las tres tablas (y no sé ve cómo cumplir el enunciado): -0.7
- Las referencias están al revés en `Voto` (ej: `Voto` desde `Video`): -0.2
- `Voto` no permite discriminar autor: -0.2

- `Voto` no permite discriminar video: -0.2
- `Voto` o `Video` no permitem saber fecha de último voto, o está repetido en los dos: -0.2

23.9.3. Ejercicio 3 (solución)

Sí sería posible. Bastaría con que, tras el POST donde Vide.os recibe el nombre de un visitante, éste le devuelva una cookie `nombre="..."`. Cada vez que, a partir de ese momento, ese navegador haga una petición a Vide.os, enviará (además de la cookie de visitante), esta cookie, con lo que Vide.os podrá saber que el visitante ya tiene nombre, y cuál es éste.

Esta cookie podría ser enviada por Vide.os en una cabecera similar a:

```
Set-Cookie: nombre=xxx; Expires=Thu, 31 Dec 2200 23:59:59 GMT;
```

siendo xxx el nombre elegido por el visitante.

El campo `Expires` se añade para que la cookie no sea "cookie de sesión" (que dejaría de estar almacenada en en navegador cuando el navegador se cerrara), y dure lo suficiente en el navegador para que sirva a efectos prácticos.

En caso de que se incluyese esa cookie, no haría falta el campo `nombre` del modelo `Visitante`. Seguiría haciendo falta la cookie de visitante, ya que el nombre no nos sirve para identificar un visitante de forma única (dos visitantes podrían elegir el mismo nombre).

Rúbrica

- Se indica que no se puede, sin explicación que lo pueda mitigar: -1.0
- No se explican el campo que se elimina del modelo (o no se mencionan): -0.3
- El diseño de cookies no funciona en general: -0.8
- El diseño de cookies no funciona para algún caso: -0.4
- El diseño de cookies no es óptimo: -0.2

23.9.4. Ejercicio 4 (solución)

Todas las interacciones son entre el navegador y el sitio, salvo la que pide la imagen de un pixel, que tendrá lugar entre el navegador y el sitio Trazad.or. Como según el enunciado el usuario sólo pone en el formulario la url, y el video correspondiente es aceptado, tenemos que entender que ya había puesto antes el nombre, por lo que ya tendrá una cookie de visitante, que se le dio en ese momento.

Por lo tanto, la envía en todas las interacciones con Vide.os. No hace falta que el servidor envíe cookies, pues esta de visitante será suficiente, y no hace falta cambiarle el valor.

- Petición POST a la página principal, para subir la url de un video. Suponemos que `yyy` es la url del video, y `/435` es el recurso que sirve la página de este video una vez ha sido aceptado.

```
POST / HTTP/1.1
Cookie: Visitante=xxx
...

url="yyy"
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, con el nuevo video, HTML]
```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache):

```
GET /main.css HTTP/1.1
Cookie: Visitante=xxx
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición de la imagen de un pixel, a Trazad.or. Podría llevar cookies, pero como no son necesarias para satisfacer el enunciado, no se incluyen:

```
GET /principal.png HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen, PNG]
```

- Petición GET, ahora a la página del video, para verla.

```
GET /435 HTTP/1.1
Cookie: Visitante=xxx
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página del video, HTML]
```

- Petición del documento CSS `/main.css`, igual que antes.
- Petición de la imagen a Trazad.or, será a un nuevo recurso (según el enunciado, cada página de Vide.os tiene una imagen de Trazad.or distinta):

```
GET /435.png HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen, PNG]
```

Hay también una interacción entre Video.os y YouTube, para obtener el título del video a partir de su url. Pero no se incluye, porque el enunciado pide sólo las interacciones HTTP entre el navegador y cualquier servidor, y en esta no interviene el navegador.

Rúbrica

Se espera que las interacciones sean correctas, incluyan todos los campos indicados, y las cookies sean correctas y consistentes con el ejercicio anterior.

- Interacciones no consistentes con primer ejercicio: -0.3
- Faltan datos en la respuesta: -0.4
- Falta interacción para CSS: -0.3
- Falta interacción para imagen: -0.3
- Faltan interacciones para segundo GET: -0.3
- Cookies diferentes incorrectas: -0.3
- No hay cookies enviadas por el navegador a Comentam.os: -0.3
- Hay cookies enviadas por Comentam.os: -0.2
- Hay cookie en intercción con Trazad.or: -0.2
- Las dos peticiones a Trazad.or son del mismo recurso: -0.3
- Se violan reglas de funcionamiento de cookies (ej: se envía una cookie que no se ha recibido, o un navegador inicia una cookie): -0.4

23.9.5. Ejercicio 5 (solución)

Sí. Cada vez que el visitante recibe una página de Vide.os el navegador hace una petición de un recurso de Trazad.or, y el recurso solicitado es distinto para cada recurso distinto solicitado de Vide.os. Si Trazad.or envía una cookie a cada navegador que le permita identificarle si vuelve a hacerle una petición, Trazad.or puede apuntar en una tabla cada vez que recibe una petición de un cierto navegador, apuntando también qué recurso de Vide.os había solicitado (lo sabrá mirando el recurso que le han pedido a él). De esa tabla puede extraer la información que pide el enunciado.

Rúbrica

- Se responde que no se puede, y la explicación no lo mitiga: -1.0.
- Se explica que sí, pero que sólo se puede saber el número de páginas pedidas, o que se han pedido algunas páginas pero no se sabe cuál: -0.5
- No se menciona que Trazad.or ha de enviar su propia cookie, o no se explica bien: -0.5

23.10. Examen de IST-SAT, 1 de junio de 2021

Se quiere construir un sitio web, Comentam.os, donde se puede comentar cualquier tema de actualidad. La funcionalidad básica del sitio es la siguiente:

1. Toda la funcionalidad del sitio está disponible para cualquier visitante: no hay cuentas, ni hace falta acreditarse para tener acceso a la funcionalidad que proporciona.

2. La página principal del sitio mostrará el listado con las noticias subidas durante las últimas 24 horas. Para cada una de ellas se mostrará el título (que será un enlace a la página de la noticia en Comentam.os), y el número de comentarios que ha recibido esa noticia.

3. Además, la página principal también tendrá un formulario para subir una nueva noticia, con dos campos: uno para poner el título de la noticia, y otro para poner el enlace a la noticia en el sitio web donde aparece.

4. La página de cada noticia incluye el título de la noticia (que será un enlace a la noticia en el sitio web donde aparece), y todos los comentarios puestos para esa noticia, junto con la fecha en que fueron puestos. Además, tendrá también un formulario para poner un comentario, con un solo campo de texto, para escribir el comentario. También incluirá un enlace para descargarse el mismo contenido (título de la noticia, enlace al sitio web donde aparece, contenido y fecha de todos los comentarios de la noticia) en formato JSON.

5. Cuando un visitante suba una noticia, no podrá subir otra hasta que pasen 24 horas. Cada visitante podrá poner sólo un comentario por noticia (pase el tiempo que pase).

6. Todas las páginas HTML del sitio incluyen una imagen de 1 pixel, servida por el sitio Trazad.or. Todas estas imágenes se sirven desde la misma url.

7. Todas las páginas HTML del sitio usarán un único documento CSS, servido por el propio sitio, que definirá el estilo de todas ellas de forma uniforme.

En esta descripción, considérese que un "visitante" corresponde con un cierto navegador, con su propio almacén de cookies. Esto es, una misma persona que use dos navegadores distintos, cada uno con su propio almacén de cookies, será considerada como dos visitantes, mientras que varias personas usando el mismo navegador, con un único almacén de cookies para todas ellas, serán consideradas como un único visitante.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). **Muy importante:** Coloca la información en una **tabla**, con los nombres de los recursos en una columna, los métodos en otra, la descripción de lo que realizará la aplicación al recibirlos en la tercera, y el contenido de los datos de la petición, si los hay en la cuarta (se consideran datos de la petición a los que vayan, normalmente como *query string*, en el cuerpo de la petición HTTP o tras el nombre de recurso en la primera línea de la cabecera). Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior. (1 punto)

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos los campos o tablas necesarios para garantizar que cada visitante sólo sube las noticias o comentarios que tiene permitido subir. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones). (1 punto)

3. Explica cómo debería ser la cookie o cookies que envíe Comentam.os a cada navegador, de forma que permita asegurar la funcionalidad indicada. Explica qué campos tendría o tendrían (los mínimos posibles), cuándo habría que enviarla (o enviarlas), siendo siempre el envío lo más tarde posible en las interacciones entre cada navegador (visitante) y Comentam.os, y para qué serviría. Sólo se ha de enviar cookies cuando sirvan para la funcionalidad descrita en este enunciado.(1 punto)

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante está viendo en su navegador la página de una noticia. El visitante ve el formulario que hay en esta página, escribe en él un comentario, y pulsa el botón para enviarlo. El visitante recibe de nuevo la página de la noticia con el comentario en él. A continuación vuelve a rellenar el formulario, y lo vuelve a enviar. El escenario termina cuando el usuario ve en su navegador la nueva página que le ha llegado del servidor, con un mensaje indicando que no puede poner un comentario porque ya ha puesto uno en esa noticia.

Para cada interacción HTTP indica claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando, y que el navegador acepta cualquier cookie que se le envíe, y nunca la borra salvo que la cookie expire. (1 punto)

5. Escribe el documento JSON que devolverá la página JSON de una noticia titulada "La noticia más importante", con url "`https://notici.as/noticia-importante`", y dos comentarios: "Qué interesante" y "Pues a mi no me lo parece". Para campo del documento JSON, indica de qué campo (y de qué tabla) procede, entre los que describiste en el apartado anterior sobre el modelo de datos. (1 punto)

23.10.1. Ejercicio 1 (solución)

Tabla de recursos:

Recurso	Método	Semántica	Datos
/	GET	Página principal	
/	POST	Nueva noticia	`noticia="..."&url="..."`
/{id_noticia}	GET	Página de noticia	
/{id_noticia}	POST	Nuevo comentario	`comentario="..."`
/{id_noticia}.json	GET	Docuento JSON de la noticia	
/main.css	GET	Documento CSS	

Fichero `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index')
  path('<id_noticia>', views.noticia, name='noticia')
  path('<id_noticia>.json', views.noticia_json, name='noticia_json')
  path('main.css', views.css, name='css')
]
```

No tiene que haber un recurso para la imagen que sirve Trazad.or, precismaente porque la sirve ese otro sitio web.

El documento JSON podría servirse también si se invoca el recurso `/{id_noticia}` con una query string (por ejemplo, `/{id_noticia}?format=json`).

Rúbrica

Para estar correcto, el ejercicio ha de incluir tanto una tabla con un documento `urls.py` correctos, completos, y sin recursos innecesarios.

Indicaciones aproximadas de errores:

- No hay columna de datos en la tabla, o está incorrecta: -0.2
- No hay recurso para documento CSS: -0.3
- No hay recurso para documento JSON: -0.3
- Hay recurso para la imagen servida por Trazad.or: -0.3
- No coincide la tabla con el fichero `urls.py`: -0.2

23.10.2. Ejercicio 2 (solución)

Hay varias soluciones, una podría ser:

```
from django.db import models

class Sesion (models.Model):
  id = models.CharField(max_lenght = 50)

class Noticia (models.Model):
  titulo: models.CharField(max_lenght = 80)
  url: models.URLField()
  autor: models.ForeignKey('Sesion')
```

```
  fecha: models.DateTimeField()

class Comentario (models.Model):
  texto: models.charField(max_lenght = 500)
  noticia: models.ForeignKey('Noticia')
  autor: models.ForeignKey('Sesion')
  fecha: models.DateTimeField()
```

La tabla `Sesion` se usa para llevar cuenta de los visitantes (navegadores distintos), y su id será el valor de la cookie que se les envíe.

La tabla `Noticia` incluye un campo `autor` para saber qué visitante puso la noticia, y un campo `fecha` para poder comprobar que un mismo visitante no puso una noticia durante las últimas 24 horas. Alternativamenete, se podría poner un campo `ultima_publicacion` en `Sesion`, pero dado que necesitamos un campo fecha para saber qué noticias se han publicado durante las últimas 24 horas (necesario para la página principal), es más "económico" organizarlo así.

La tabla `Comentario` incluye referencias a la noticia en la que está (para poderlo mostrar en su página) y al visitante que lo puso (para poder comprobar que no vuelva a poner un comentario en la misma noticia). También incluye un campo fecha que sería necesario para poder ordenar los comentarios por antiguedad en la página de la noticia, pero esto no está estrictamente pedido por el enunciado, así que es correcto no ponerlo.

Rúbrica

- No están las tres tablas (y no sé ve cómo cumplir el enunciado): -0.7
- Las referencias están al revés en `Comentario` (ej: `Comentario` desde `Noticia`): -0.2
- `Noticia` no permite discriminar autor: -0.2
- `Noticia` no permite controlar 24 horas por autor: -0.2
- `Noticia` no permite seleccionar 24 horas: -0.1
- `Comentario` no permite discriminar autor: -0.2
- `Comentario` no permite discriminar noticia: -0.2

23.10.3. Ejercicio 3 (solución)

Basta con una cookie que se envíe a cada navegador cuando hagan la primera acción que modifique la base de datos (que será un POST). No hay que enviarla antes, porque las dos acciones que hay que controlar (no más de una noticia cada 24 horas, no más de un comentario por noticia) sólo se hacen en ese momento.

Esta cookie podría tener una forma similar (como cabecera) similar a:

```
Set-Cookie: Visitante=xxx; Expires=Thu, 31 Dec 2200 23:59:59 GMT;
```

siendo xxx un identificador único, aleatorio, y suficientemente grande.

El campo `Expires` se añade para que la cookie no sea "cookie de sesión" (que dejaría de estar almacenada en en navegador cuando el navegador se cerrara), y dure lo suficiente en el navegador para que sirva a efectos prácticos.

El valor de la cookie es el que se almacenará en el campo `id` del modelo `Sesion`.

Rúbrica

- No se explican los campos (o no se mencionan): -0.3
- El diseño de cookies no funciona en general: -0.8
- El diseño de cookies no funciona para algún caso: -0.4
- El diseño de cookies no es óptimo: -0.2

23.10.4. Ejercicio 4 (solución)

Todas las interacciones son entre el navegador y el sitio, salvo la que pide la imagen de un pixel, que tendrá lugar entre el navegador y el sitio Trazad.or. Como el servidor ha podido enviar una cookie antes, si este navegador ha puesto ya una noticia o un comentario (y no lo sabemos, pues el enunciado no lo dice), sirven tanto soluciones donde en el POST (primera acción en este escenario) se envía ya una cookie que se tenía, o no se envía, y en ese caso se recibe. En esta solución, se supone que la cookie ya se tenía.

- Petición POST a la página de una noticia, para subir un comentario. Suponemos que `/435` es el recurso que sirve la página de la noticia que está viendo el visitante.

  ```
  POST /435 HTTP/1.1
  Cookie: Visitante=xxx
  ...

  comentario="Este es el comentario"
  ```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página de la noticia, HTML]
```

- Petición de la hoja de estilo CSS que incluye la página CSS (suponemos que no se usa la cache):

```
GET /main.css HTTP/1.1
Cookie: Visitante=xxx
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición de la imagen de un pixel, a Trazad.or. Podría llevar cookies, pero como no son necesarias para satisfacer el enunciado, no se incluyen:

```
GET /imagen.png HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen, PNG]
```

- Nueva petición POST a la página de una noticia, para subir otro comentario.

```
POST /435 HTTP/1.1
Cookie: Visitante=xxx
...

comentario="Este es otro comentario"
```

- Respuesta

```
HTTP/1.1 403 Forbidden
...

[Página de la noticia, con mensaje de error, HTML]
```

- Petición del documento CSS `/main.css`, igual que antes.
- Petición de la imagen `/imagen.png` a Trazad.or, igual que antes.

Rúbrica

Se espera que las interacciones sean correctas, incluyan todos los campos indicados, y las cookies sean correctas y consistentes con el ejercicio anterior.

- Interacciones no consistentes con primer ejercicio: -0.3
- Faltan datos en la respuesta: -0.4
- Falta interacción para CSS: -0.3
- Falta interacción para imagen: -0.3
- Faltan interacciones para segundo POST: -0.4
- Cookies diferentes de ejercicio anterior, e incorrectas: -0.3
- Cookies diferentes de ejercicio anterior, y correctas: -0.2
- No hay cookies enviadas por el navegador a Comentam.os: -0.3
- Hay cookies enviadas por Comentam.os: -0.3
- Hay cookie en intercción con Trazad.or: -0.2
- Se violan reglas de funcionamiento de cookies (ej: se envía una cookie que no se ha recibido, o un navegador inicia una cookie): -0.4

23.10.5. Ejercicio 5 (solución)

```
{"titulo": "La noticia mas importante",
 "url": "https://notici.as/noticia-importante",
 "comentarios": [{"texto": "Que interesante",
                  "fecha": "..."}
                 {"texto": "Pues a mi no me lo parece",
                  "fecha": "..."}]
}
```

`titulo` y `url` se obtienen del modelo `Noticia`, `texto` y `fecha` de `Comentario`.

Rúbrica

- Uso incorrecto de listas o diccionarios: -0.3
- Anidamiento excesivo: -0.3
- No es JSON: -0.6

23.11. Examen de ITT-SARO e IST-SAT, 6 de junio de 2020

[**Atención:** Este curso se realizó una prueba escrita que hace referencia al proyecto final de la asignatura. Normalmente, el enunciado de la prueba escrita es independiente del proyecto final, pero este curso no lo fue. Esta prueba escrita tenía también varias preguntas entre las que se componían conjuntos de exámenes diferentes para cada alumno, algo que tampoco es habitual. Estas preguntas "alternativas" están marcadas en el enunciado como "ALTER".]

Teniendo en cuenta el sistema construido para la práctica final de la asignatura (apartado 26), tal y como lo entregues, se pide:

1. Describe la interfaz HTTP que proporciona el sistema, incluyendo todos los recursos accesibles mediante GET o POST. Coloca la información en una tabla, con los nombres de recurso en una columna, los métodos HTTP en otra, y la descripción de lo que realizará la aplicación al recibirlos en la tercera. (0.5 puntos).

2. ¿Qué le falta a la la interfaz HTTP que acabas de describir para cumplir los principios REST? Explica brevemente cómo la modificarías (respetando la funcionalidad del enunciado de la práctica final) para que los cumpliera (si crees que no los cumple) o por qué los cumple (si crees que sí los cumple). (0.5 puntos)

3. En el enunciado de la práctica final se pide que cuando un visitante cualquiera elimina un alimentador en la página principal, éste deje de salir en el listado de alimentadores de esa página. ¿Qué cambios en el modelo de datos tendrías que hacer para que sólo los usuarios (autenticados) pudieran eliminar un alimentador, y si lo eliminan sólo se eliminase de la lista que ve el usuario (autenticado) que lo eliminó? Explica los cambios sobre el fichero `models.py` de tu práctica. (0.75 puntos)

4. ALTER. En el enunciado de la práctica final se pide que cuando un visitante cualquiera elimina un alimentador en la página principal, éste deje de salir en el listado de alimentadores de esa página. ¿Qué cambios en el modelo de datos tendrías que hacer para que si lo elimina sólo se eliminase de la lista que ve el visitante (esté autenticado o no) que lo eliminó? Explica los cambios sobre el fichero `models.py` de tu práctica. Si el visitante no está autenticado, se entiende que sigue usando el mismo navegador desde el que eliminó el alimentador. (0.75 puntos)

5. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en los siguientes escenarios:

   a) El escenario comienza cuando un visitante que ha accedido por primera vez al sitio tiene ya cargada en su navegador la página principal del sitio. A continuación, el visitante escribe en el formulario correspondiente el identificador de un canal de YouTube, para seleccionarlo como alimentador (ese canal no ha sido seleccionado previamente nunca). El escenario termina cuando el visitante está viendo en su navegador la página del alimentador que acaba de seleccionar. (0.5 puntos)

   b) El escenario comienza cuando un visitante que no ha accedido nunca al sitio, y tiene una página en blanco en el navegador, escribe en su navegador la url de la página principal del sitio concatenada con `?format=xml` al final. El escenario termina cuando el visitante ve la página resultante en su navegador (o su navegador le propone guardarla o mostrarla con una aplicación, según esté configurado). (0.5 puntos)

   c) El escenario comienza cuando un visitante que ha accedido, estando ya autenticado, a varias páginas del sitio, está viendo la página de un canal de YouTube (esto es, la página de un alimentador de YouTube). A continuación, el visitante pulsa el botón, que no estaba seleccionado en ese momento, para seleccionarlo como alimentador (ese canal ya había sido seleccionado previamente al menos en una ocasión). El escenario termina cuando el visitante está viendo de nuevo en su navegador la página de este canal de YouTube. (0.5 puntos)

En todas las respuestas indica para cada interacción HTTP, claramente y en este orden:

- La primera línea de la petición HTTP
- La línea Content-Type (si no la hubiera en la petición HTTP real, la que debería haber)
- Si lo hay, el contenido (cuerpo) de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y cómo.

Asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario. Para todas las cookies que aparezcan en las interacciones, explica para qué sirven, y si para el caso de la interacción concreta en que aparecen, hacen falta o no. Incluye sólo las cookies que tengan que tengan que ver con la práctica final que entregues. Para todas las interacciones descritas, considérese que la cache del navegador no se está usando.

6. Describe las interacciones que ocurrirán entre tu aplicación y la base de datos en el siguiente escenario. El escenario comienza cuando un usuario ya autenticado en un cierto navegador, y que está viendo en ese navegador su página de usuario, decide rellenar el formulario para cambiar el tamaño de letra. Una vez relleno, envía el contenido. El escenario termina cuando el usuario vuelve a ver su misma página de usuario, pero ya con el nuevo tamaño de letra.

   Para cada interacción, indica al menos la tabla (o tablas) de la base de datos a las que se accede, si se escriben o se leen datos, y para qué se realiza la interacción (para seleccionar un registro que cumpla tales condiciones, para escribir tal campo de tal registro, etc.) (0.75 puntos)

7. ALTER. Describe las interacciones que ocurrirán entre tu aplicación y la base de datos en el siguiente escenario. El escenario comienza cuando un usuario ya autenticado en un cierto navegador, y que está viendo en ese navegador la página de un item, decide votarlo positivamente. Para eso, pulsa el botón correspondiente. El escenario termina cuando el usuario vuelve a ver la página del item, ya con su voto registrado.

Para cada interacción, indica al menos la tabla (o tablas) de la base de datos a las que se accede, si se escriben o se leen datos, y para qué se realiza la interacción (para seleccionar un registro que cumpla tales condiciones, para escribir tal campo de tal registro, etc.) (0.75 puntos)

8. ALTER. Describe las interacciones que ocurrirán entre tu aplicación y la base de datos en el siguiente escenario. El escenario comienza cuando un usuario ya autenticado en un cierto navegador, y que está viendo en ese navegador su página de usuario, decide rellenar el formulario para cambiar el estilo (se supone que estaba en "ligero" y pasa a "oscuro"). Una vez relleno, envía el contenido. El escenario termina cuando el usuario vuelve a ver su misma página de usuario, pero ya con el nuevo estilo.

   Para cada interacción, indica al menos la tabla (o tablas) de la base de datos a las que se accede, si se escriben o se leen datos, y para qué se realiza la interacción (para seleccionar un registro que cumpla tales condiciones, para escribir tal campo de tal registro, etc.) (0.75 puntos)

9. Un usuario de tu sitio está especialmente preocupado por que nadie vote items en su nombre. Te dice que suele usar el sitio en el navegador de su móvil, mientras está conectado mediante redes en las que es posible que alguien pueda ver los paquetes de datos en tránsito desde su navegador hasta tu sitio (redes inseguras). Dado que tu aplicación no funciona sobre HTTPS, y por tanto la conexión entre el navegador y tu sitio no está cifrado, ¿cuál o cuáles de las siguientes acciones le recomendarías que no hiciera, para evitar que alguien pueda acceder a su cuenta y votar en su nombre? (asegúrate que respondes en el caso concreto de lo que hace tu práctica tal y como la entregues):

   a) Rellenar el formulario de autenticación con usuario y contraseña, y entrar así en la cuenta, mientras está conectado a una de esas redes inseguras

   b) Usar el sitio desde redes inseguras, ya autenticado, pero habiendo hecho la autenticación previamente, cuando estaba conectado con su móvil desde una red que consideras segura (en la que nadie tiene acceso a los paquetes en tránsito).

   c) Usar el sitio desde redes inseguras, ya autenticado, pero habiendo hecho la autenticación previamente, cuando estaba conectado con su móvil desde una red que consideras segura, con la precaución de no votar ningún item mientras está conectado mediante redes inseguras.

Tanto si le recomiendes que no lo haga, como si le dices que lo puede hacer sin problemas especiales, explica por qué. (0.75 puntos)

10. Acordamos con un sitio tercero (Trazador.com) que sirva el banner (la imagen que aparece en la cabecera de todas las páginas HTML de nuestro sitio) en lugar de servirlo nuestro sitio directamente. Para ello, en todas las páginas HTML que sirva nuestra aplicación irá la misma url (la del banner que sirve Trazador.com) como atributo `src` de un elemento `img`. Explica si Trazador.com puede, gracias a que está sirviendo el banner, y sin hacer ninguna otra modificación a la práctica:

    a) Saber cuántas páginas HTML está sirviendo tu sitio.

    b) Saber cuántas veces se está sirviendo la página principal HTML de tu sitio.

    c) Saber cuántas veces se está sirviendo el fichero JSON que se sirve desde la página principal de tu sitio cuando se incluye la query string adecuada.

    d) Saber cuántos navegadores únicos visitan tu sitio.

    e) Saber cuántos navegadores únicos visitan la página principal de tu sitio.

    En cada caso, si crees que no puede, explica porqué, y si crees que sí puede, explica cómo. En todos los casos suponemos que los navegadores almacenan y envían cookies de forma normal, sin tener instalado ningún sistema que las bloquee. (0.75 puntos)

11. ALTER. Acordamos con un sitio tercero (Trazador.com) que sirva el banner (la imagen que aparece en la cabecera de todas las páginas HTML de nuestro sitio) en lugar de servirlo nuestro sitio directamente. Para ello, en todas las páginas HTML que sirva nuestra aplicación irá la misma url (la del banner que sirve Trazador.com) como atributo `src` de un elemento `img`. Explica si Trazador.com puede, gracias a que está sirviendo el banner, y sin hacer ninguna otra modificación a la práctica:

    a) Saber si un cierto navegador, que está visitando una página HTML de tu sitio ahora mismo, estuvo visitando alguna página HTML del sitio también ayer.

    b) Saber cuántas páginas HTML distintas de tu sitio está visitando un cierto navegador, que está visitando tu una página HTML de tu sitio ahora mismo.

c) Saber cuántos días del mes pasado visitó alguna página HTML de tu sitio un cierto navegador, que está visitando una página HTML de tu sitio ahora mismo.

d) Saber si un cierto navegador, que está visitando una página HTML de tu sitio ahora mismo, ha visitado alguna vez el documento JSON que sirve el recurso principal.

En cada caso, si crees que no puede, explica porqué, y si crees que sí puede, explica cómo. En todos los casos suponemos que los navegadores almacenan y envían cookies de forma normal, sin tener instalado ningún sistema que las bloquee. (0.75 puntos)

En todas las preguntas tu respuesta debe considerar cómo se comporta la práctica que entregues, salvo donde se diga expresamente otra cosa.

23.11.1. Solución

API HTTP

Esta podría ser una descripción de la API HTTP:

Recurso	Método	Comentario
`/`	GET	Página principal. Si incluye query string `?format=json` o `?format=xml` devuelve los formatos correspondientes
`/alimentadores`	GET	
`/alimentadores/{id}`	GET	
	POST	Actualización y desactualización de alimentadores, con qs: `alimentador=id&acción=[add/remove]`
`/alimentadores/{id}/{item\_id}`	GET	
	POST	Voto de item, con qs: `vote=[plus/minus]`, comentario de item, con qs: `comment="Comentario"`
`/users`	GET	Lista de usuarios
`/users`	POST	Registro de usuarios nuevos, y autenticación de usuarios. Query strings: `user=usuario&passwd=palabra&action=[ne`
`/users/{id}`	GET	Página de usuario
	POST	Configuración de usuario y salida de cuenta, con varias qs: `picture={foto}`, `style=[light`dark]—, `font=[small,medium,large]`, `action=logout`
`/users/{id}/foto`	GET	Foto de usuario
`/info`	GET	Página de información
`/banner.jpg`	GET	Banner

REST

Algunos aspectos básicos para evaluar si la API HTTP cumple mejor o peor los principios REST:

- Los recursos corresponden realmente con recursos, que tienen su propio estado, y no con funciones o acciones. Por ejemplo, tener recursos como `/login`, `/logout` o `/register` es más orientado a acciones que a recursos. Si lo orientamos a recursos, tendríamos un recurso para usuarios, en el que podríamos tanto obtener un nuevo recurso (registrar un usuario) como autenticarse, en ambos casos mediante POST. La salida de la cuenta podría hacerse, por ejemplo, con un POST sobre `/users/{id}`, con los parámetros adecuados (a falta de poder usar DELETE.

- Dado que la API ha de funcionar desde un navegador, sin uso de JavaScript, estamos limitados a lo que permite HTML, y en particular los formularios

HTML, que serán la forma fundamental de subir datos a la aplicación. Por eso tendremos que sustituir PUT y DELETE por POST.

- El uso de colecciones para agrupar recursos homogéneos. Por ejemplo, usuarios, alimentadores, items... Estas colecciones se mostrarán en la estructura jerárquica de la API (por ejemplo, todos los usuarios accesibles como `/users/{id}`, bajo `/users`).

- En general, se van a usar cookies para mantener estado (sesión) entre llamadas a la API. Hay que explicar la relación de REST con esto, dado que en REST se supone que no hay estado entre llamadas a la API (no hay sesión). Cuando se usan cookies, lo que ocurre es que cada llamada lleva un parámetro (la cookie, como cabecera) que permite que el recurso que recibe la llamada pueda dirigirse al estado adecuado (el que corresponde al navegador que hace la llamada). De esta forma, en realidad cada recurso proporciona distintos estados según la cookie.

Sobre esta base, puede verse que, por ejemplo, la API descrita en el apartado anterior cumple bastante bien los principios REST (en general está diseñada en torno a recursos, con recursos homogéneos agrupados en colecciones, evitando que los recursos correspondan con acciones). Sin embargo, habría que explicar al menos cómo no se usa PUT y DELETE, y el papel de las cookies.

Modelos. Escenario usuarios autenticados

Para que sólo los usuarios autenticados puedan eliminar un alimentador de la lista de alimentadores que ven, habría que tener una tabla que ligue los alimentadores con los usuarios que los han eliminado. Para ello, pueden usarse al menos dos estrategias (supongamos que la tabla de usuarios está representada por el modelo User, y la de alimentadores, Feeder):

- Nuevo modelo (Removed), con dos campos. Ambos serán ForeignKeyField, apuntando a User y Feeder.

- Nuevo campo (removed), que sera un ManyToManyField, en la tabla Feeder, apuntando al modelo User.

En ambos casos, esto supondrá que se creará una nueva tabla como la descrita anteriormente. Para saber qué alimentadores habría que mostrar a un usuario:

- En el primer caso, se filtraría la tabla Feeder excluyendo aquellos alimentadores que estén también en la tabla Removed apuntando al usuario en cuestión.

- En el segundo caso, se filtraría la tabla Feeder excluyendo aquellos alimentadores cuyo campo removed apunte al usuario en cuestión.

Modelos. Escenario visitantes

Para hacer algo similar con visitantes, se puede seguir la misma estrategia anterior, pero con la tabla de sesiones, en lugar de con la tabla de usuarios (por lo tanto, habría al menos las dos mismas soluciones que se explican anteriormente). De todas formas, para contestar completamente bien el ejercicio, hay que tener en cuenta que si el visitantes se ha autenticado, sus selecciones pasan a ser del usuario. Por ello, en realidad habría que realizar también lo descrito en el apartado anterior. Así pues, al eliminar un alimentador:

- Si se elimina por un usuario no autenticado, se añadiría una relación desde la sesión hacia ese alimentador.
- Si se elimina por un usuario autenticado, se añadiría una relación desde el usuario hacia ese alimentador (igual que en el apartado anterior).

De esta manera, los alimentadores a mostrar serían:

- Si el visitante no está autenticado, los que no hayan sido eliminados desde su sesión.
- Si el visitante está autenticado, los que no hayan sido eliminados desde su usuario.

23.11.2. Rúbrica calificación

En general, se descuenta por errores de concepto, por explicaciones que muestran errores de comprensión de conceptos de la asignatura, o por elementos importantes que faltan en la respuesta. El siguiente detalle es sólo orientativo.

- API HTTP:
  - Faltan recursos necesarios de la API (/info, /banner, /style.css, foto del usuario, etc): -0.1 (cada uno)
  - Faltan descripciones, o son incorrectas: -0.1 (cada una)
  - Faltan acciones (GET o POST), o son incorrectas: -0.1 (cada una)
- REST:
  - No se explica la parte que cumple REST: entre -0.1 y -0.2

- No se explica el uso de POST en lugar de PUT o DELETE: -0.1
- No se explica el papel de las colecciones: -0.1
- No se explica el papel de las cookies en el mantenimiento de estado entre llamadas: -0.1
- No se analizan los recursos que corresponden más a funciones que a recursos: -0.1

- Modelos:
  - Esquema que no puede funcionar. Por ejemplo, usar one2one donde debería ser foreignkey: -0.3
  - No usar modelos, sino otras cosas (listas, cookies, etc.) para la solución: -0.3
  - No tener en cuenta algún detalle (por ejemplo, las sesiones sin autenticar) cuando eso es relevante: -0.2
  - No explicar al menos un poco cómo se usarían los modelos o cambios a los modelos propuestos: -0.2
- Todos los escenarios HTTP:
  - Falta banner, CSS: -0.1 cada uno
  - No coincide con API HTTP: -0.2
  - Faltan interacciones "importantes": -0.2
  - Content-Type mal: -0.1
  - No funciona: -0.4
  - Faltan cookies "auxiliares" (ej: CSRF): -0.1
  - Faltan (o sobran) cookies importantes (ej: autenticación): -0.2
  - Falta la descripción del cuerpo, qs, etc: -0.2
  - No se consideran errores relativos al favicon.
- Escenarios de acceso a base de datos:
  - No acceder a usuario cuando hace falta: -0.3
  - No actualizar o consultar datos fundamentales: -0.4
  - No actualizar o consultar datos accesorios: -0.2
  - No acceder para comprobar sesión (lo hace Django): -0.2
  - No se considera error si se llevan cuentas de totales a mano.

- Acceso desde redes inseguras:
  - Se considera seguro poner usuario y contraseña: -0.55
  - Se considera seguro exponer la cookie: -0.55
- Escenarios de trazador:
  - No se tienen en cuenta las caches del navegador: -0.3
  - No se tiene en cuenta que la url del banner no cambia (cuando es el caso): -0.3
  - No se explica el papel de las cookies para las visitas únicas: -0.3
  - no se tiene en cuenta que el JSON no supone descarga del banner: -0.3
  - Todo ello matizado por el conocimiento general sobre el mecanismo de trazado con elementos HTML.

23.12. Examen de ITT-SARO, 14 de mayo de 2019

Se quiere construir un sitio web, ElTiempecito, donde se puedan compartir comentarios sobre el tiempo en diversas localidades. La funcionalidad básica del sitio es la siguiente:

1. El sitio sólo permite acceso a usuarios registrados (registrados previamente, mediante un sistema no considerado en este enunciado, y que no es parte de ElTiempecito), mediante autenticación.

2. Si un visitante sin autenticar visita el sitio, cualquier página que visite le redirigirá a la página principal. Y en esta página principal verá un formulario para poder autenticarse mediante usuario y contraseña (que será lo que se ha registrado previamente). Una vez autenticado, el visitante podrá acceder los servicios del sitio, que se describen a continuación.

3. La página principal mostrará las 20 poblaciones con más comentarios durante las últimas 24 horas, ordenadas por número de comentarios durante ese periodo. Para cada población, se mostrará la previsión de temperatura, y un enlace a la página de la población.

4. La página principal del sitio tendrá también un banner (imagen en formato PNG). Este banner se mostrará también cuando el visitante no se haya autenticado.

5. Cada población tendrá una página (la "página de la población"), con los datos meteorológicos de la población, y los 20 comentarios mas recientes que se han puesto sobre ella, ordenados de más moderno a más antiguo.

6. La página de cada población tendrá también un formulario para poner comentarios. Este formulario constará sólo de una caja de texto, donde se escribirá el comentario, y un botón, para enviarlo. Tras enviar el comentario, se volverá a ver la misma página de la población, ya con el comentario, y sin que para ello se haya hecho ninguna redirección.

7. La página de cada población mostrará también un icono (una nube o un sol), en formato PNG, que resumirá la previsión del tiempo.

8. Todas las imágenes (iconos de nube o sol, banner) serán servidos por el sitio.

9. El sitio ofrecerá también un canal XML con el listado de las mismas poblaciones que aparecen en la página principal, y para cada una el dato de temperatura prevista para mañana, y un enlace a la página de la población.

10. El sitio obtendrá todos sus datos meteorológicos de su propia base de datos. Este enunciado no describe cómo se actualizan esos datos, ni es eso relevante para lo que se pregunta.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos la tabla o tablas necesarias para saber el número de páginas vistas por cada navegador, gracias al uso de la imagen transparente que se describe en en enunciado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de una página del sitio que no es la principal. Al no estar autenticado, es redirigido a la página principal donde, después de verla, rellena el formulario de autenticación con un nombre de usuario y contraseña validos (previamente registrados), y ve de nuevo la página principal. El escenario termina cuando el visitante está viendo de nuevo la página principal de nuevo, tal y como se ofrece ya a usuarios autenticados. (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante autenticado que está que viendo en su navegador la página de una población. El visitante rellena al caja de comentario, y pulsa el botón para enviarlo. El escenario termina cuando el visitante vuelve a ver la página de la población, ya con el nuevo comentario en ella.

5. El enunciado indicaba que el banner que se muestra en la página principal, se haya autenticado el visitante o no, es servido por el propio sitio. En caso de que fuera servido por un sitio tercero, ¿podría este sitio tercero aprovechar esto de alguna manera para saber cuańtos visitantes únicos (navegadores únicos) visitan la página principal del sitio, lleguen estos a autenticarse, o no? ¿Cómo?

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo

(incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando.

23.12.1. Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST

Este podría ser el esquema REST:

Recurso	Método	Comentario
/	GET	Página principal (HTML)
/	POST	Autenticación, devuelve página principal (HTML)
/{id_poblacion}	GET	Página de población (HTML)
/{id_poblacion}	POST	Comentario, devuelve página de población (HTML)
/img/nube	GET	Icono de "nube"
/img/sol	GET	Icono de "sol"
/img/banner	GET	Banner del sitio
/xml	GET	Canal XML

Todos los recursos del sitio devolverán una redirección 303 (See Other) al recurso "/" cuando no se reciba cookie de autenticación válida (el visitante no se ha autenticado).

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('xml', views.xml, name='xml'),
    path('img/nube', views.image, {'file': 'nube'}, name='image'),
    path('img/sol', views.image, {'file': 'sol'}, name='image'),
    path('img/banner', views.image, {'file': 'banner'}, name='image'),
    path('<id_poblacion>', views.poblacion, name='poblacion')
```

```
]
```

Los recursos que sirven imágenes podrían servirse también configurando adecuadamente los mecanismos de Django para servir ficheros estáticos.

models.py Versión simplificada, que cumple el enunciado, aunque podría optimizarse:

```
from django.db import models

class Usuario(models.Model):
    nombre: models.CharField(max_length=20)
    contrasena: models.CharField(max_length=20)

class Sesion(models.Model):
    cookie = models.CharField(max_length=20)
    user = models.ForeignKey('Usuario', null==True)

class Poblacion(models.Model):
    id = models.IntegerField()
    nombre = models.TextField()
    temperatura = models.FloatField()
    soleado = models.BooleanField()

class Comentario(models.Model):
    poblacion = models.ForeignKey('Poblacion')
    comentario = models.TextField()
    fecha = models.DateTimeField()
```

No se incluyen los campos identificador único para cada tabla.

Primer escenario Todas las interacciones son entre el navegador y el sitio.

- Petición GET a una página distinta de la principal. Suponemos que se accede a una población, cuyo identificador es "43".

  ```
  GET /43 HTTP/1.1
  ...
  ```

- Respuesta

```
HTTP/1.1 303 See Other
...
Location: /
```

- Petición GET de la página principal, fruto de la redirección anterior.

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página con el formulario para autenticarse, HTML]
```

- Petición GET para el banner, originada debido a que está referenciado en la página HTML anterior.

```
GET /img/banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de banner, PNG]
```

- Petición POST para enviar datos del formulario de autenticación.

```
POST / HTTP/1.1
...

usuario=Maria&contrasena=PalabraDePaso
```

- Respuesta

```
HTTP/1.1 200 OK
...
Set-Cookie: sesion=323ddfd3323243hhhh323

[Página principal con toda la funcionalidad, HTML]
```

- Petición GET para el banner, originada debido a que está referenciado en la página HTML anterior (se considera que no se usa la cache).

```
GET /img/banner HTTP/1.1
...
Cookie: sesion=323ddfd3323243hhhh323
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de banner, PNG]
```

Segundo escenario A continuación, las interacciones son entre el navegador y el sitio. Como el visitante se ha autenticado ya, se le habrá enviado (con una cabecera "Set–Cookie") una cookie, como se indica en el apartado anterior. Suponemos que el identificador de la población cuya página está viendo el usuario al comenzar el escenario es "34".

- Petición POST /34 (para subir un comentario):

```
POST /34 HTTP/1.1
...
Cookie: sesion=323ddfd3323243hhhh323

comentario="Me gusta el tiempo de esta poblacion"
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página de la poblacion, con el comentario ya puesto, HTML]
```

- Petición GET para el banner, originada debido a que está referenciado en la página HTML anterior (se considera que no se usa la cache).

```
GET /img/banner HTTP/1.1
...
Cookie: sesion=323ddfd3323243hhhh323
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de banner, PNG]
```

- Petición GET para el icono, originada debido a que está referenciado en la página HTML anterior (se considera que no se usa la cache).

```
GET /img/nube HTTP/1.1
...
Cookie: sesion=323ddfd3323243hhhh323
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de la nube, PNG]
```

Trazado de navegadores únicos desde servidor de banner Cada vez que se sirva la página principal del sitio, el navegador que la realice hará una petición al sitio Tercero para pedirle el banner (que ya está presente incluso antes de autenticarse). De esta manera, dado que se considera que no se usa la cache del navegador, Tercero recibirá una petición por cada petición de la página principal del sitio. Para poder saber si el navegador que está realizando la visita es el mismo que ya la realizó antes, Tercero enviará junto con la imagen una cookie de sesión (usando una cabecera "Set–Cookie"). De esta manera, cada vez que un navegador vuelva a visitar la página principal, y por tanto a pedir el banner, enviará también a Tercero, con esta petición, la cookie que recibió. Tercero la usará para saber que no tiene que considerar esta petición, pues corresponde con una cookie que ya sirvió, y por tanto con un navegador que ya visitó la página principal del sitio.

23.13. Examen de ITT-SAT, 15 de mayo de 2019

Se quiere construir un sitio web, ElTiempazo, donde se puedan compartir datos meteorológicos (temperaturas, por ejemplo) en diversas localidades. La funcionalidad básica del sitio es la siguiente:

1. El sitio no precisa de registro previo. Cualquier visitante puede utilizar toda su funcionalidad.

2. La página principal del sitio mostrará un listado de todas las localidades sobre las que se pueden aportar datos, como enlaces a las páginas de esas localidades en el sitio (ver más abajo).

3. La página principal mostrará también un listado de los datos meteorológicos que se han aportado desde este mismo navegador en el pasado (junto con un enlace a la página de la población para la que se aportaron cada uno de esos datos).

4. Cada población tendrá una página (la "página de la población"), con los datos meteorológicos que se hayan aportado para la población durante las últimas 24 horas, por cualquier visitante.

5. La página de cada población tendrá también un formulario para aportar nuevos datos. Este formulario constará de dos cajas, una para poner números, donde se escribirá la temperatura, otra para poner texto, donde se escribirá una descripción del tiempo (ejemplo: "Nubes, claros y chubascos esporádicos"), y un botón, para enviar su contenido. Tras enviar los datos, se verá la página principal, ya con los nuevos datos en el listado de datos suministrados desde este navegador.

6. Todas las páginas tendrán su estilo determinado por una hoja de estilo CSS, la misma para todo el sitio, que se sirve por el propio sitio.

7. Para cada población sobre la que se puedan poner datos, se ofrecerá también un canal XML con los datos subidos para ella durante las últimas 24 horas y un enlace a la página de la población.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará

la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos la tabla o tablas necesarias para saber el número de páginas vistas por cada navegador, gracias al uso de la imagen transparente que se describe en en enunciado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página principal del sitio. En ella, pulsa sobre la página de una población. El escenario termina cuando el visitante está viendo esa página de población. (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que está que viendo en su navegador la página de una localidad. El visitante rellena las cajas de datos, y pulsa el botón para enviarlos. El escenario termina cuando el visitante ve la página principal, ya con los nuevos datos en ella.

5. Supongamos que un sitio tercero llega a un acuerdo con ElTiempazo para trazar todas las acciones de subida de datos meteorológicos. Para ello, va a servir una imagen transparente, que se puede colocar en cualquier página HTML. Diseña, si crees que es posible, un sistema por el que este sitio tercero pueda saber (y apuntar) cada vez que un navegador cualquiera suba nuevos datos sobre una población a ElTiempazo, explicando qué tendría que hacer la aplicación web de ElTiempazo para que esto sea posible. Es importante que el sistema permita al sitio tercero llevar su propia contabilidad basada en las descargas de esa imagen, no en datos que ElTiempazo le pueda enviar. Si crees que no puede hacerse en este caso concreto, explica por qué, y qué habría que cambiar para que se pudiera.

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse.

Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.
- Si hubiera envío de datos de formulario, recuerda indicar de forma explícita dónde van esos datos, y si es posible, en qué formato.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

En general, para todas las interacciones descritas, considérese que la cache del navegador no se está usando.

23.13.1. Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST Este podría ser el esquema REST:

Recurso	Método	Comentario
/	GET	Página principal (HTML)
/	POST	Nuevos datos, devuelve página principal (HTML)
/{id_poblacion}	GET	Página de población (HTML) Incluirá formularion, con `action=/` y campo oculto con `poblacion=id_poblacion`
/main.css	GET	Hoja de estilo CSS
/{id_poblacion}.xml	GET	Documento XML para cada población

Para que el POST sobre el recurso principal, con los datos de una población, se pueda indentificar como datos para esa población, tendrá que incluir el identificador de la población en la query string. Para ello, incluiremos un campo oculto en el formulario de datos, justamente con ese dato.

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('main.css', views.css, name='css'),
    path('<id_poblacion>', views.poblacion, name='poblacion'),
    path('<id_poblacion>.xml', views.poblacion_xml, name='poblacion_xml')
]
```

El recurso que sirve el documento CSS podría servirse también configurando adecuadamente los mecanismos de Django para servirlo como fichero estático.

models.py Versión simplificada, que cumple el enunciado, aunque podría optimizarse:

```
from django.db import models

class Sesion(models.Model):
    cookie = models.CharField(max_length=20)

class Poblacion(models.Model):
    id = models.IntegerField()
    nombre = models.TextField()

class Datos(models.Model):
    poblacion = models.ForeignKey('Poblacion')
    sesion = models.ForeignKey('Sesion')
    temperatura = models.FloatField()
    descripcion = models.TextField()
    fecha = models.DateTimeField()
```

No se incluyen los campos identificador único para cada tabla.

Primer escenario Todas las interacciones son entre el navegador y el sitio.

- Petición GET de la página principal.

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal, HTML]
```

- Petición GET para el documento CSS, originada debido a que está referenciado en la página HTML anterior.

```
GET /main.css HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

- Petición GET a una página de población. Suponemos que su identificador es "43".

```
GET /43 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página de la población, HTML]
```

- Petición GET para el documento CSS, originada debido a que está referenciado en la página HTML anterior.

```
GET /main.css HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

Segundo escenario A continuación, las interacciones son entre el navegador y el sitio. Suponemos que el visitante no ha puesto un dato anteriormente, por lo que no ha recibido aún cookie (si lo hubiera hecho, enviaría una cabecera "Cookie", porque habría recibido una cuando lo puso, y por tanto no recibirá la cabecera "Set-Cookie"). Suponemos que el identificador de la población cuya página está viendo el usuario al comenzar el escenario es "34".

- Petición POST / (para subir un dato):

```
POST / HTTP/1.1
...

poblacion=34&temperatura=14&descricion="Me gusta el tiempo de esta poblacion
```

- Respuesta

```
HTTP/1.1 200 OK
...
Set-Cookie: sesion=323ddfd3323243hhhh323

[Página principal, con los nuevos datos ya puestos, HTML]
```

- Petición GET para el documento CSS, originada debido a que está referenciado en la página HTML anterior.

```
GET /main.css HTTP/1.1
...
Cookie: sesion=323ddfd3323243hhhh323
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Documento CSS]
```

Trazado de navegadores únicos desde un sitio tercero Los datos se suben con un POST, por lo que en principio no es posible incluir ninguna imagen para trazarlo. Sin embargo, podemos aprovecharnos de que como respuesta al POST, nuestro servidor enviará una página HTML: será en esa página en la que incluyamos la imagen transparente.

De esta manera, cada vez que un navegador realice un POST con nuevos datos, recibirá la página HTML con la imagen del sitio tercero, y realizará una petición a éste para obtenerla. Por lo tanto, este sitio tercero podrá contabilizar las respuestas a estos POST. Además, si envía una cookie a cada navegador, podrá saber si el navegador en cuestión ya ha subido más datos anteriormente.

Naturalmente este mecanismo no funcionará si se hiciera un POST, y se recibiera una respuesta, pero luego no se tratara de obtener los elementos de la página HTML que se recibe. Esto no ocurrirá si la subida de datos se realiza de forma normal desde un navegador.

23.14. Examen de ITT-SAT, 7 mayo de 2018

Se quiere construir un sitio web, MisMuseos, donde se puedan compartir valoraciones sobre museos. La funcionalidad básica del sitio es la siguiente:

1. Para poder utilizar el sitio hace falta un código de acceso. Los códigos de acceso son cadenas alfanuméricas de 20 caracteres, que se consiguen en los museos. Una vez se ha introducido un código de acceso correcto desde un navegador se puede acceder desde ese navegador a toda la funcionalidad del sitio. Si no, cualquier recurso del sitio devolverá un documento HTML con un formulario para introducir un código de acceso.

2. Una vez se ha introducido un código válido (que llamaremos, a partir de ese momento, "código activo" en ese navegador), el sitio sólo proporcionará dos recursos que devuelvan un documento (salvo que haga falta alguno más para proporcionar la funcionalidad descrita en este enunciado):

    - El recurso (página) principal, que devolverá el documento HTML que se describe más adelante.

- El recurso de valoraciones realizadas, que devolverá un documento XML con un listado de las valoraciones realizadas usando el código de acceso activo en el navegador, ordenadas por fecha de valoración, e incluyendo para cada una de ellas el nombre del museo y la valoración dada.

3. Cualquier otro recurso que se pida desde el navegador causará que se envié una redirección a la página principal.

4. La página principal del sitio mostrará a los visitantes (una vez se ha introducido un código válido):

   - Un formulario para elegir un nombre, o el nombre si ya se usó el formulario para elegirlo
   - Un enlace que, si se pulsa, hará que el código de acceso quede "desactivado" (dejando por tanto de ser un "código activo" en ese navegador). Cualquier nueva acción en el sitio devolverá el formulario para introducir el código de acceso.
   - Un listado con todos los museos que tienen MisMuseos, incluyendo para cada museo una foto, el nombre del museo, la puntuación media que le han dado los visitantes del sitio, y un formulario para valorarlo. Este formulario tendrá un botón ("Valorar") y una caja para poner la valoración (un número entero entre 0 y 4).

5. Además, todas las páginas HTML (incluido el formulario para introducir el código de acceso) incluirán una imagen transparente, de un píxel, que se utilizará para que MisMuseos pueda trazar el número de páginas vistas desde un mismo navegador, esté activo un código en ese navegador o no.

6. Las fotos de cada museo son servidas por el sitio web de cada museo, no por MisMuseos.

7. Al poner un valor en el formulario de valoración de un museo, y pulsar "Valorar", se añadirá una nueva valoración al museo en cuestión, si el código de acceso activo en ese navegador nunca había valorado ese museo, o cambiará la valoración anterior, si ya lo había valorado.

8. Un mismo código de acceso puede ser utilizado desde varios navegadores (estar activo en ellos), en periodos diferentes o simultáneos.

9. En un mismo navegador pueden estar activos varios códigos de acceso, pero no simultáneamente. Sólo si se "olvida" (deja de estar activo) el que se está usando, se podrá activar otro, introduciéndolo en el formulario que se recibirá tras pulsar el enlace de "olvidar".

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos la tabla o tablas necesarias para saber el número de páginas vistas por cada navegador, gracias al uso de la imagen transparente que se describe en en enunciado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario que recibe con un código de acceso válido. El escenario termina cuando el visitante vuelve a ver la página principal del sitio, pero ahora ya con el código activo, y por lo tanto viendo la lista de museos. (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya tiene un código activo está viendo la página principal del sitio, con la lista de museos. El visitante rellena el formulario de valoración de un museo, que nunca había valorado antes con ese código, y pulsa el botón "Valorar". El escenario termina cuando el visitante vuelve a ver la página principal, con la lista de museos (1 punto).

5. Escribe cómo podría ser el documento XML para un visitante que está usando un código con el que se han realizado valoraciones para los museos "El Campo", "Reina Margarita" y "Tisten" (una valoración para cada uno) (1 punto).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST

Este podría ser el esquema REST una vez se ha introducido un código válido:

Recurso	Método	Comentario

/	GET
	POST
		Página principal (HTML)
		`museo=id&val=val`
		(valoración de museo)
		o
		`nombre=nombre`
		(poner nombre)
		Devolverá el mismo HTML que si se invoca con GET

/val.xml	GET	Página XML con valoraciones para el código (XML)
/pixel	GET	Pixel para trazar páginas vistas (GIF)
/salir	GET	Desactivación de código Devolverá el formulario para introducir código (HTML) (según el enunciado, se invoca con un enlace, luego ha de ser GET)

Antes de introducirlo, todos los recursos devolverán, ante un GET, el formulario para introducir el código, salvo "/pixel" (que funcionaría igual) y / que para POST admitiría un código, `codigo=codigo_museo` (para GET devolvería el formulario también).

El recurso / podrá discriminar, si recibe un POST, si se está valorando un museo o si se está poniendo un nombre por el nombre de los campos en la query string recibida.

Nota: Alternativamente, podría haber un recurso para valorar para cada muse, por ejemplo "/valorar/{museo}", sobre el que se haría el POST de valoración. Pero en ese caso, sería recomendable que este recurso, además de aceptar la valoración, devolviese una redirección sobre el recurso /.

urls.py

Lo escribimos sólo para el esquema REST una vez se ha introducido el código:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.pagina_principal),
    url(r'^val.xml$', views.valoraciones),
    url(r'^pixel$', views.trazado),
    url(r'^salir$', views.salir)
]
```

models.py

Versión simplificada, que cumple el enunciado, aunque podría optimizarse:

```
from django.db import models

class Codigos(models.Model):
    codigo = models.CharField(max_length=20)
    nombre = models.CharField(max_length=100, null==True)

class Museos(models.Model):
    nombre = models.TextField()
    id = models.IntegerField()
    foto = models.CharField(max_length=100)

class Navegadores(models.Model):
    cookie = models.CharField(max_length=32)
    vistas = models.IntegerField()

class Activos(models.Model):
    codigo = models.ForeignKey('Codigos')
    navegador = models.ForeignKey('Navegadores')

class Valoraciones(models.Model):
    codigo = models.ForeignKey('Codigos')
    museo = models.ForeignKey('Museos')
    valoracion = models.IntegerField()
    fecha = models.DateTimeField()
```

No se incluyen los campos identificador único para cada tabla.

La tabla Codigos tendrá todos los códigos válidos (que se han repartido a los museos). Esta tabla es fuente de ineficiencias, porque la mayoría de los nombres estarán vacíos (dado que corresponderán a códigos no usados o a los que no se les ha puesto nombre), por lo que en producción sería conveniente tener una tabla separada para los nombres. Pero tal y como está definida aquí, funcionaría.

Primer escenario

Todas las interacciones son entre el navegador y el sitio MisMuseos, salvo cuando se indica otra cosa.

- Petición GET /

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK

[Formulario para codigo, HTML]
```

- Petición GET /pixel. El navegador, al cargar el documento HTML recibido, encontrará en él la referencia al pixel para trazado, y lo pedirá mediante otra interacción HTTP:

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...
Set-Cookie: ....; navegador=12345ABCDE12345ABCDE12345ABCDE12

[Imagen para trazado, GIF]
```

`navegador` es un identificador de navegador, que servirá para trazar las páginas vistas (suponemos que este se envía con cabeceras que lo hagan no-cacheable, de forma que pueda realizar su misión). También se utilizará, cuando se haya enviado un código válido, para saber que este navegador se ha autenticado (anotándolo en la tabla Activos).

- Petición POST / (para proporcionar el código que se ha introducido en el formulario):

```
POST / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12

codigo=ABCDE12345ABCDE12345
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal con la lista de museos, HTML]
```

Al recibir esta petición y comprobar que el código es correcto (utilizando la tabla Codigos), MisMuseos apuntará este navegador con este código en la tabla Activos, donde seguirá apuntado hasta que el usuario decida desactivar este código en su navegador.

- Petición GET /pixel (igual que la anterior):

```
GET / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12
```

- Respuesta

```
HTTP/1.1 200 OK

[Imagen para trazado, GIF]
```

En esta ocasión, ya no se recibe una cookie, sino que se envía (ya se recibió, y MisMuseos, al detectar que ya viene con la petición, no la vuelve a enviar). Al recibir esta petición, MisMuseos apuntará una nueva página vista para este navegador.

- Petición GET de la foto del museo museo (una por cada museo). Cada una de estas interacciones son **con el sitio web de los museos en cuestión**.

```
GET /url_foto HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK

[Foto, GIF]
```

- ...

Segundo escenario

A continuación, las interacciones son entre el navegador y el sitio MisMuseos. Suponemos, como ya se ha indicado, que la imagen se sirve como no-cacheable..

- Petición POST / (para realizar una valoración):

```
POST / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12

museo=5&val=3
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página principal con la lista de museos, HTML]
```

  Al recibir esta petición, MisMuseos comprobará que el código está activo, buscando el identificador de navegador en la tabla Activos, y consiguiendo a partir de la entrada correspondiente el código. A continuación, utilizará el código para añadir una entrada a la tabla Valoraciones con el identificador del museo, el código, y la valoración.

- Petición GET /pixel (igual que las anteriores):

```
GET / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12
```

- Respuesta

```
HTTP/1.1 200 OK

[Imagen para trazado, GIF]
```

En este caso no se han incluido las peticiones de las fotos de los museos, porque se hace la suposición razonable de que serán imágenes cacheables. En cualquier caso, si no se hace esta suposición, se pueden incluir, de la misma forma que se incluyeron anteriormente

Canal XML

```
<?xml version="1.0" encoding="UTF-8" ?>

<valoraciones>
  <codigo>ABCDE12345ABCDE12345</codigo>
  <lista_valoraciones>
    <valoracion>
      <museo>El Campo</museo>
      <val>3</val>
      <fecha>2018-05-03 12:20:21</fecha>
    </valoracion>
    <valoracion>
      <museo>Reina Margarita</museo>
      <val>4</val>
      <fecha>2018-05-02 11:10:31</fecha>
    </valoracion>
    <valoracion>
      <museo>Tisten</museo>
      <val>1</val>
      <fecha>2018-05-01 13:23:11</fecha>
    </valoracion>
  </lista_valoraciones>
</valoraciones>
```

En general, hay que cuidar que la valoración de cada museo sea claramente identificable que corresponde a ese museo, y las convenciones sintácticas de XML.

23.15. Examen de ITT-SARO, 17 mayo de 2018

Se quiere construir un sitio web, MisMuseos, donde se puedan compartir comentarios sobre museos. La funcionalidad básica del sitio es la siguiente:

1. El sitio está públicamente accesible para cualquiera que lo quiere consultar, sin necesidad de abrir cuenta ni ningún otro trámite.

2. Además, cualquiera podrá dar un "me gusta" a los museos que quiera, en la página del museo (ver más abajo), pero no más de una vez desde el mismo navegador (ver página de museo, más abajo).

3. Para poder poner un comentario sobre un museo, hace falta un código de acceso, disponible en ese museo. Los códigos de acceso son cadenas alfanuméricas de un solo uso, en el sentido de que quien tiene un código puede

poner un comentario sobre el museo correspondiente y editarlo cuantas veces quiera desde cualquier navegador, usando ese código. Pero una vez usado para poner un comentario, ese código sólo permitirá cambiar el comentario.

4. La funcionalidad "en modio consulta" del sitio es la siguiente:

   - El recurso (página) principal del sitio tendrá un listado de todos los museos que se pueden consultar. Para cada museo se mostrará el nombre del museo (que será un enlace a la página del museo, ver más abajo), el último comentario para ese museo, y un icono (en formato PNG) con el número de "me gusta" que ha recibido ese museo.
   - La página de cada museo tendrá el nombre y dirección del museo, un formulario con un botón (sin icono) para indicar "me gusta" si no se ha pulsado ya ese botón desde ese mismo navegador, y un formulario para escribir un código de museo, si no se ha utilizado ya desde ese navegador para ese mismo museo (en ese caso, el museo estará "en modo comentario" (ver más abajo). Además, tendrá el listado de todos los comentarios que se hayan puesto, con cualquier código válido y desde cualquier navegador, para ese museo.

5. La funcionalidad "en modo comentario" del sitio es igual, salvo que en la página de los museos donde se haya introducido un código válido desde ese navegador, en lugar del formulario para introducir el código aparecerá un formulario para introducir un comentario. Si ese código ya se ha usado (desde cualquier navegador) para introducir un comentario, ese comentario aparecerá "precargado" en el formulario. El resto de la página es exactamente igual que en "modo consulta".

6. Los iconos que se ven en las páginas del sitio están servidas por el propio sitio.

7. Todas las páginas del sitio (página principal y páginas de museos) tendrán un banner (imagen en formato PNG), que será servido por un sitio tercero que llamaremos "Servidor del banner".

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará

la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Asegúrate de que incluyes en el modelo de datos la tabla o tablas necesarias para saber el número de páginas vistas por cada navegador, gracias al uso de la imagen transparente que se describe en en enunciado. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página principal del sitio. A continuación, después de ver esta página principal, pulsa sobre el enlace de un museo, y ve la página de ese museo. El escenario termina cuando el visitante está viendo la página principal de ese museo en su navegador (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya ha rellenado, en la página de un museo, el formulario de código con un código válido, y está viendo la página de ese museo. A continuación, rellena el formulario con un comentario (no lo había rellenado nunca antes), y lo envía a MisMuseos. Y a continuación, pone un "me gusta" para ese mismo museo (pulsando en el botón correspondiente). El escenario termina cuando el visitante vuelve a ver la página del museo, ya sin el botón de "me gusta" y con el comentario pre-relleno en el formulario de comentarios (1 punto).

5. Describe qué tendrá que hacer el "Servidor del banner" para poder saber, de forma independiente de MisMuseos, el número de navegadores únicos que están visitando MisMuseos, sin más colaboración por parte de MisMuseos que colocar un banner que él sirva en todas sus páginas (como ya se comentó en la descripción de funcionalidad de MisMuseos).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST

Este podría ser el esquema REST:

Recurso	Método	Comentario
/	GET	Página principal (HTML)

/{id_museo}	GET
	POST
		Página de un museo
		`codigo=codigo`
		(introducción de código de museo)
		`megusta=True`
		(“me gusta” al museo)
		`comentario=texto`
		(poner un comentario sobre el museo)

/iconos/{num}	GET	Iconos para los números de “me gusta”

Las opciones POST para “me gusta” y para poner comentario a un museo devolverán 401 (no autorizado) cuando no se haya introducido un código válido para ese museo, y “funcionarán” de acuerdo al enunciado cuando se haya introducido.

urls.py

Lo escribimos sólo para el esquema REST una vez se ha introducido el código:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.pagina_principal),
    url(r'^(\d+)$', views.museo),
    url(r'^iconos/(\d+)$', views.trazado),
]
```

models.py

Versión simplificada, que cumple el enunciado, aunque podría optimizarse:

```
from django.db import models

class Codigos(models.Model):
    codigo = models.CharField(max_length=20)
    museo = models.ForeignKey('Museos')
    navegador = models.ForeignKey('Navegadores', null==True)

class Museos(models.Model):
    nombre = models.TextField()
    id = models.IntegerField()
    direccion = models.TextField()

class Navegadores(models.Model):
    cookie = models.CharField(max_length=32)

class MeGusta(models.Model):
    navegador = models.ForeignKey('Navegadores')
    museo = models.ForeignKey('Museos')

class Comentarios(models.Model):
    codigo = models.ForeignKey('Codigos')
    comentario = models.TextField()
    fecha = models.DateTimeField()
```

No se incluyen los campos identificador único para cada tabla.

La tabla Codigos tendrá todos los códigos válidos (que se han repartido a los museos), cada uno con su museo correspondiente. Cuando un navegador use un código, se anotará en esta tabla.

Primer escenario

Todas las interacciones son entre el navegador y el sitio MisMuseos, salvo cuando se indica otra cosa.

- Petición GET /

```
GET / HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK

[Página principal, HTML]
```

- Petición GET de los iconos de número de "me gusta" para cada uno de los museos mostrados. Habrá tantas de estas interacciones como iconos con el número de "me gusta" haya en los museos de la página principal. La primera de estas interacciones supone que el número de "me gusta" que muestra el icono es 3:

```
GET /iconos/3 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen con 3 "me gusta", PNG]
```

- Petición GET para el banner, realizada no a MisMuseos sino al "Servidor del banner"

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen de banner, PNG]
```

- Petición GET de la página de museo, una vez el visitante ha pulsado sobre el enlace correspondiente (suponemos que el museo en cuestión tiene el identificador "12"):

```
GET /12 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK

[Página de museo, HTML]
```

En este escenario no han hecho falta cookies, porque no es necesario identificar al navegador (no se introducen códigos, no se pulsa sobre "me gusta"...)

Segundo escenario

A continuación, las interacciones son entre el navegador y el sitio MisMuseos. Suponemos que el banner no es preciso volver a pedirlo, porque estará en la cache del navegador. Además, como el visitante ha introducido ya un código válido de museo, se le habrá enviado (con una cabecera "Set–Cookie") una cookie, para poder identificarle. Suponemos que el museo al que corresponde la página que está viendo el visitante es el "12".

- Petición POST /12 (para subir un comentario):

```
POST / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12

comentario="Me ha gustado mucho este museo"
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página del museo, con el comentario ya puesto, HTML]
```

- Petición POST /12 (para indicar "me gusta"):

```
POST / HTTP/1.1
...
Cookie: navegador=12345ABCDE12345ABCDE12345ABCDE12

megusta=True
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Página del museo, con el comentario y sin el boton "me gusta", HTML]
```

Trazado de navegadores únicos desde servidor de banner

Basta con que, cada vez que sirve por primera vez un banner de MisMuseos a un navegador, le envíe (con cabecera "Set-Cookie") una cookie con un identificador de navegador único. Cuando le llegue una petición que ya tenga cookie, no hará más que servir el banner. El número de navegadores únicos será el número de cookies servidas.

23.16. Examen de ITT-SAT, 10 mayo de 2017

Se quiere construir un sitio web, Mensajitos, donde se pueden poner mensajes para que los vean otras personas. La funcionalidad básica del sitio es la siguiente:

1. En el sitio no hay cuentas para usuarios: toda la funcionalidad está disponible para cualquiera que lo visite.

2. De todas formas, cualquier visitante podrá reservar un nombre que no esté ya en uso para cuando suba información al sitio. Este nombre se mantendrá mientras el visitante utilice el mismo navegador. Para ello se usará el formulario que aparece en la página principal (ver a continuación).

3. La página principal del sitio mostrará a los visitantes un botón para crear un canal de mensajes, y un formulario para elegir un nombre (si se ha elegido ya, en lugar del formulario aparecerá el nombre elegido). El botón permitirá crear un nuevo canal (cada visitante puede crear tantos como quiera), según se indica más abajo. Además, en esta página principal cada visitante verá la lista de los canales que ha creado previamente. Tras crear un nuevo canal, o elegir un nombre, el visitante volverá a ver la página principal del sitio.

4. Cada canal tendrá un nombre de recurso único, que se generará aleatoriamente cuando se cree. Cualquiera que conozca el nombre de recurso de un canal, podrá leer y escribir en él, simplemente accediendo a ese recurso (lo haya creado quien lo haya creado).

5. El recurso correspondiente a cada canal mostrará una página HTML (la "página del canal") con los 10 últimos mensajes en el canal, un formulario para poner un nuevo mensaje, y un formulario para poner la url de una imagen (que puede estar en cualquier sitio de Internet, mientras la haga visible mediante HTTP). Cada mensaje que se muestre, se mostrará con el formato:

   ```
   Nombre: mensaje
   ```

   Donde "Nombre" es el nombre del visitante (o "Anónimo", si no lo ha elegido), y "mensaje" es el mensaje en cuestión.

   Las urls de imágenes se considerarán también como mensajes, pero antes de mostrarlos como tales (y de almacenarlos en la base de datos), se convertirán a un elemento IMG de HTML. Por ejemplo, la imagen de url `http://fotos.com/123345.jpg` se convertirá en el HTML siguiente (que se considerará el "mensaje" en el formato descrito anteriormente):

   ```
   <img src="http://fotos.com/123345.jpg" style="width:200px;height:150px;">
   ```

   Tras poner un nuevo mensaje (o una imagen) en un canal, el visitante vuelve a ver de nuevo la página del canal.

6. Cada canal tendrá también un recurso asociado donde se podrán descargar todos sus mensajes (incluyendo aquellos que se especificaron como imágenes) en formato XML. Este recurso aparecerá también como enlace en la página del canal. El documento XML correspondiente incluirá al menos todos los mensajes que se han escrito en el canal, el nombre del visitante que puso cada uno de ellos, la fecha en que se puso cada uno de ellos, el enlace a la página

HTML del canal, la fecha en que se creó el canal, y el nombre del visitante que creó el canal (si alguno de los visitantes implicados no ha especificado un nombre, se usará "Anónimo").

7. Todas las páginas HTML del sitio incluirán una imagen de cabecera (banner) que se alojará en el propio sitio.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario para elegir un nombre. En este momento, el navegador vuelve a mostrar la página principal, ya con el nombre elegido en lugar del formulario para elegirlo. A continuación el visitante crea un nuevo canal. El escenario termina cuando el visitante vuelve a ver la página principal del sitio, con el nuevo canal ya creado. (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza con un visitante que ya ha reservado nombre y está viendo la página de un canal que aún no tiene mensajes. El visitante rellena el formulario de imagen, poniendo la url de una imagen válida. El escenario termina cuando el visitante vuelve a ver la página del canal, ya con el nuevo mensaje generado a partir de la url de la imagen (1 punto).

5. Escribe cómo podría ser el documento XML para un canal que tiene tres mensajes, uno de los cuales corresponde a la url de una imagen, para un usuario que tiene nombre (1 punto).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST

Recurso	Método	Comentario
/	GET	Página principal (HTML)

/	POST	Creación de canal
		`canal=True`
		Reserva de nombre
		`nombre=nombre_visitante`
		Devolverá el mismo HTML que si se invoca con GET
		Este HTML incluirá ya un enlace al nuevo canal

/{id_canal}	GET	Página del canal `id_canal` (HTML)

/{id_canal}	POST	Subir mensaje al canal `id_canal`
		`mensaje=texto`
		Subir imagen al canal `id_canal`
		`imagen=url`
		Devolverá el mismo HTML que si se invoca con GET

/{id_canal}.xml	GET	Página del canal `id_canal` (XML)
/banner	GET	Imagen que se usará como banner del sitio

Nota: No se indica en el enunciado, pero convendrá que la página HTML que se reciba como respuesta a un POST para crear un canal incluya, de forma prominente, un enlace a dicho canal, dado que el usuario necesita saber cuál es.

urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.pagina_principal),
    url(r'^banner$', views.banner),
    url(r'^(.+\.xml)$', views.canal_xml),
    url(r'^(.+)$', views.canal)
]
```

models.py

Versión simplificada, que cumple el enunciado:

```
from django.db import models

class Visitante(models.Model):
```

```
    nombre = models.CharField(max_length=20, null==True)
    cookie = models.CharField(max_length=64)

class Canal(models.Model):
    recurso = models.CharField(max_length=50)
    creador = models.ForeignKey('Visitante')

class Mensaje(models.Model):
    canal = models.ForeignKey('Canal')
    autor = models.ForeignKey('Visitante')
    texto = models.TextField()
    fecha = models.DateTimeField() # Para fecha en XML
```

Primer escenario

Todas las interacciones son entre el navegador y el sitio Mensajitos.

- Petición GET /

  ```
  GET / HTTP/1.1
  ...
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  Set-Cookie: ....; session=session_id

  [Pagina principal, HTML]
  ```

  session_id es un identificador de sesión (o de visitante), que se puede enviar también más adelante. Como identificador de sesión que es, será normalmente una cadena de caracteres larga, generada aleatoriamente, y por tanto difícil de adivinar para quien no la conozca.

- Petición GET /banner (para cargar la imagen del banner)

  ```
  GET /banner HTTP/1.1
  ...
  Cookie: session=session_id
  ```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- Petición POST / (para enviar los datos del formulario de nombre)

```
POST / HTTP/1.1
...
Cookie: session=session_id

nombre=Nombre_Usado
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina principal, ya sin formulario para elegir nombre, HTML]
```

  La cookie que se envió anteriormente, en realidad se podría enviar aquí, pues hasta este momento no hay nada que asociar a la sesión.

- Petición POST / (para enviar los datos del botón de crear canal)

```
POST / HTTP/1.1
...
Cookie: session=session_id

canal=True
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina principal, ahora con el nuevo canal, HTML]
```

Segundo escenario

A continuación, las interacciones son entre el navegador y el sitio Mensajitos. Suponemos que la imagen del banner ya está en la caché del navegador, y por tanto no se pide. Como el navegador ya ha estado visitando el sitio y tiene nombre, ha de haber recibido la cookie de sesión. Suponemos que "/2732434232" es el nombre de recurso correspondiente al canal.

- Petición POST /2732434232 (para poner el mensaje)

```
POST / HTTP/1.1
...
Cookie: session=session_id

imagen="url"
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina del canal, ahora con un nuevo mensaje con el img correspondiente, HT

 <img src="url" style="width:200px;height:150px;">]
```

Ahora, el navegador tendrá que pedir la imagen que se haya incluido anteriormente (url "url"). Esta interacción será por lo tanto entre el navegador y el sitio al que apunte la url de la imagen. Suponiendo que la url sear `http://sitio.com/imagen`:

- Petición GET /imagen (para cargar la imagen)

```
GET /imagen HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Imagen]
```

Canal XML

```
<?xml version="1.0" encoding="UTF-8" ?>

<canal>
  <recurso>http://mensajitos.com/3443344453</recurso>
  <creado>20 de marzo de 2016 23:05:05</creado>
  <creador>Flor de Loto</creador>
  <mensaje>
    <texto>Este es un mensaje</texto>
  </mensaje>
    <texto>
      <img src="url" style="width:200px;height:150px;">
    </texto>
  <mensaje>
  </mensaje>
  <mensaje>
    <texto>Este es el último mensaje</texto>
  </mensaje>
</canal>
```

El texto para el mensaje de la imagen tendría que ponerse "codificado" para que no se confunda con texto XML, pero esto no se ha tenido en cuenta en esta solución..

23.17. Examen de IST-SARO, 10 mayo de 2017

Se quiere construir un sitio web, Fotogram, donde se pueden poner fotos para que las vean otras personas. La funcionalidad básica del sitio es la siguiente:

1. En el sitio no hay cuentas para usuarios: toda la funcionalidad está disponible para cualquiera que lo visite.

2. La página principal del sitio mostrará a los visitantes un formulario para crear un nuevo canal de fotos. Este formulario permitirá elegir un nombre para el canal, y el nombre del recurso en que se servirá, que deberá ser (el nombre del recurso) uno que no esté ya en uso en el sitio. Además, en esta página principal cada visitante verá la lista de los canales que ha creado previamente, y junto a cada uno habrá un botón para borrarlo. Tras crear un nuevo canal, o borrarlo, el visitante volverá a ver la página principal del sitio.

3. Cualquiera que conozca el nombre de recurso de un canal, podrá ver sus fotos, y poner fotos en él, simplemente accediendo a ese recurso (lo haya creado quien lo haya creado).

4. El recurso correspondiente a cada canal mostrará una página HTML (la "página del canal") con las fotos puestas en ese canal y el comentario asociado a cada foto (si lo hay), y un formulario para poner una nueva foto. Este formulario permitirá especificar la url de una foto (que puede estar en cualquier sitio de Internet, mientras la haga visible mediante HTTP), y opcionalmente un comentario asociado a esa foto. Para cada foto que se muestre se mostrará la foto, el comentario asociado a ella (si lo hay) y la fecha en que se subió la foto. Tras poner una nueva foto en un canal, el visitante vuelve a ver de nuevo la página de ese canal.

5. El sitio aceptará en un recurso (uno para todo el sitio, no uno por canal), no enlazado en ninguna página del mismo, un documento XML con un listado de fotos a subir a un canal, que se recibirá en el cuerpo de un POST de HTTP. El documento XML incluirá el nombre de recurso del canal donde se subirán las fotos (que deberá existir), y un listado de las fotos a subir. Para cada foto, se incluirá su url y (opcionalmente) su comentario asociado.

6. Todas las páginas HTML del sitio incluirán una imagen de cabecera (banner) que se alojará en el sitio de url `http://banners.com`.

7. Se supone que la cache del navegador está deshabilitada.

Teniendo en cuenta los requisitos anteriores, se pide:

1. Diseña un esquema REST para proporcionar el servicio descrito. Se habrán de especificar los nombres de recurso empleados, y cómo reaccionará la aplicación cuando reciba los métodos POST o GET sobre esas urls (no se usarán los métodos PUT o DELETE). Coloca la información en una tabla, con las urls en una columna, los métodos en otra, y la descripción de lo que realizará la aplicación al recibirlos en la tercera. Escribe también un fichero similar al fichero urls.py de Django (aunque no es importante que se respete la sintaxis mientras se entienda y la estructura sea similar a la de Django), que refleje el esquema REST anterior (1 punto).

2. Describe el modelo de datos que necesitará esta aplicación. Define las tablas necesarias y los campos necesarios para la funcionalidad descrita. Hazlo de forma lo más similar posible a lo que tendrías que escribir en el fichero models.py en Django (aunque no es importante que se respete la sintaxis mientras se entienda el modelo de datos que propones) (1 punto).

3. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página principal del sitio. A continuación, después de ver esta página principal, rellena el formulario para crear un canal. El sitio le devuelve una página donde aparecerá ya el canal recién creado en la lista de canales. El escenario termina cuando el visitante ve en su navegador la página de ese canal, tras haber pulsado sobre él en la lista (1 punto).

4. Describe las interacciones HTTP que ocurrirán entre el navegador y cualquier servidor web en el siguiente escenario. El escenario comienza cuando un visitante que accede por primera vez al sitio pone en el navegador la url de la página de un canal en el que ya hay una foto. A continuación, después de ver la página del canal en cuestión con esa foto, rellena el formulario para poner una nueva foto, indicando su url (no incluye comentario). El escenario termina cuando el visitante ve en su navegador de nuevo la página del canal, ya con la foto que acaba de poner (1 punto).

5. Escribe cómo podría ser el documento XML para subir un listado de fotos a un canal (1 punto).

En todos los escenarios, ten en cuenta que tu respuesta debe considerar toda la funcionalidad que ofrece el servicio, y permitir que ésta pueda proporcionarse. Diseña la aplicación de forma que envíe cookies al navegador sólo cuando sea necesario.

En las respuestas donde describas interacciones HTTP indica para cada una de ellas claramente y en este orden:

- La primera línea de la petición HTTP
- Si lo hay, el contenido de la petición
- La primera línea de la respuesta HTTP
- Si lo hay, el contenido de la respuesta
- Una brevísima explicación de para qué se usa la interacción
- Tanto en la petición como en la respuesta, las cabeceras con cookies, si es que fueran necesarias para la funcionalidad del escenario que se está describiendo (incluyendo el aspecto que han de tener esas cabeceras). Si la cabecera con cookie va o no dependiendo de algún factor ajeno a tu aplicación, explica cuando irá y cuándo no, y cuál es ese factor.

Además, asegúrate de que describes las interacciones HTTP en el orden en que ocurrirían en el escenario.

Soluciones

Hay muchas soluciones posibles. A continuación, una de ellas.

Esquema REST

Recurso	Método	Comentario
/	GET	Página principal (HTML)
/	POST	Creación de canal `canal=Nombre&recurso=Recurso` Para borrar canal, misma qs, con nombre de canal vacío Devolverá el mismo HTML que si se invoca con GET Este HTML incluirá ya un enlace al nuevo canal cuando se haya creado
/{rec_canal}	GET	Página del canal `rec_canal` (HTML)
/{rec_canal}	POST	Subir foto al canal `rec_canal` `foto=url&comentario=Texto` Devolverá el mismo HTML que si se invoca con GET
/subir	POST	Página del canal `id_canal` (XML)

urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.pagina_principal),
    url(r'^(subir)$', views.subir_xml),
    url(r'^(.+)$', views.canal)
]
```

models.py

Versión simplificada, que cumple el enunciado:

```
from django.db import models

class Visitante(models.Model):
    cookie = models.CharField(max_length=64)

class Canal(models.Model):
    recurso = models.CharField(max_length=50)
```

```
    nombre = models.CharField(max_length=50)
    creador = models.ForeignKey('Visitante')

class Foto(models.Model):
    canal = models.ForeignKey('Canal')
    url = models.CharField(max_length=50)
    comentario = models.TextField()
    fecha = models.DateTimeField()
```

Primer escenario

Las interacciones son entre el navegador y el sitio que se indica.

- Petición GET / (a Fotogram)

  ```
  GET / HTTP/1.1
  ...
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  Set-Cookie: ....; session=session_id

  [Pagina principal, HTML]
  ```

  session_id es un identificador de sesión (o de visitante), que se puede enviar también más adelante. Como identificador de sesión que es, será normalmente una cadena de caracteres larga, generada aleatoriamente, y por tanto difícil de adivinar para quien no la conozca.

- Petición GET /banner (a Banners)

  ```
  GET /banner HTTP/1.1
  ...
  ```

- Respuesta

  ```
  HTTP/1.1 200 OK
  ...

  [Banner]
  ```

- Petición POST / (para enviar los datos del formulario de canal)

```
POST / HTTP/1.1
...
Cookie: session=session_id

canal=Nombre&recurso=Recurso
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina principal, ya con enlace al canal recién creado, HTML]
```

  La cookie que se envió anteriormente, en realidad se podría enviar aquí, pues hasta este momento no hay nada que asociar a la sesión.

- Petición GET /banner (a Banners)

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- Petición GET al recurso del canal (para ver la página del canal, a Fotogram)

```
GET /af34df45aed33 HTTP/1.1
...
Cookie: session=session_id
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina del canal, HTML]
```

- Petición GET /banner (a Banners)

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

Segundo escenario

Las interacciones son entre el navegador y el sitio que se indica.

- Petición GET al recurso del canal (a Fotogram)

```
GET /af34df45aed33 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...
Set-Cookie: ....; session=session2_id

[Pagina del canal, que lleva una foto, HTML]
```

session2_id es un identificador de sesión (o de visitante), que se puede enviar también más adelante (o incluso no enviar en este escenario, porque según enunciado no se traza qué visitante sube las fotos).

- Petición GET /banner (a Banners)

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- Petición GET /Foto (al sitio donde está la foto)

```
GET /Foto HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Foto, JPEG, PNG, GIF, etc.]
```

- Petición POST al recurso del canal (a Fotogram)

```
POST /af34df45aed33 HTTP/1.1
...
Cookie: session=session2_id

foto=url_foto2&comentario=
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Pagina del canal, que lleva una foto, HTML]
```

- Petición GET /banner (a Banners)

```
GET /banner HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Banner]
```

- Petición GET /Foto (al sitio donde está la foto)

```
GET /Foto HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Foto, JPEG, PNG, GIF, etc.]
```

- Petición GET /Foto2 (al sitio donde está la segunda foto)

```
GET /Foto2 HTTP/1.1
...
```

- Respuesta

```
HTTP/1.1 200 OK
...

[Foto, JPEG, PNG, GIF, etc.]
```

Documento XML

```
<?xml version="1.0" encoding="UTF-8" ?>

<fotos>
  <recurso>/Canal</recurso>
  <foto>
    <url>http://sitiodefotos.com/foto1</url>
  </foto>
  <foto>
    <url>http://sitiodefotos2.com/foto2</url>
    <comentario>Esta es una foto</comentario>
  </foto>
  <foto>
    <url>http://sitiodefotos3.com/foto3</url>
    <comentario>Esta es otra foto</comentario>
  </foto>
</fotos>
```

24. Proyecto final: MisPalabras (2022)

Este es el enunciado del proyecto final del curso 2021-2022, tanto para la convocatoria ordinaria como para la extraordinaria.

La práctica final de la asignatura consiste en la creación de una aplicación web, llamada "MisPalabras", que permitirá gestionar información sobre palabras, obtenida automáticamente de ciertas fuentes soportadas, o aportada por usuarios. La información aportada por usuarios podrá consistir en comentarios e imágenes, o en enlaces. En este caso, si la página apuntada por el enlace tiene una "tarjetas embebidas", se mostrará también su contenido. Los usuarios podrán añadir aportaciones con enlaces a páginas que hayan visto, comentarlos, puntuarlos, compartirlos, etc. La aplicación mantendrá una única lista de palabras (con su información correspondiente) para todos los usuarios. A continuación se describe el funcionamiento y la arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

24.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones (*apps*) Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de Django.
- Para implementar usuarios, cuando sea necesario, se usará como base el sistema de autenticación de usuarios que proporciona Django[22].
- Todas las bases de datos que contenga la aplicación tendrán que ser accesibles vía la interfaz que proporciona el "Admin Site" (además de lo que pueda hacer falta para que funcione al aplicación).
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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:

[22] User Authentication in Django:
https://docs.djangoproject.com/en/3.0/topics/auth/

- Una cabecera del sitio, preferentemente en la parte superior, que consistirá de un texto (el nombre del sitio, "MisPalabras") con una imagen de fondo.
- Un formulario 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, este formulario permitirá al visitante introducir su identificador de usuario y su contraseña y o bien autenticarse (logearse) con esos datos, o bien crearse una cuenta con esos datos.
  - En caso de que ya se haya entrado, este formulario mostrará el identificador del usuario y permitirá salir de la cuenta (logout). Este formulario aparecerá preferentemente como una línea, justo debajo de la cabecera.
- Un menú de opciones, como barra, preferentemente debajo de los dos elementos anteriores (banner y caja de entrada o salida). Los contenidos de este menú se indican más adelante en el enunciado.
- Un formulario para ir a la página de una palabra, y un listado de las 10 palabras más votadas. Este formulario y esta lista aparecerán en una columna en el lado derecho de la página.
- Un pie de página con una nota de atribución, indicando "Esta aplicación tiene información sobre XXX palabras (ejemplo: YYY)", siendo XXX el número de palabras sobre las que tiene información en ese momento, e YYY una de esas palabras, como enlace a la página de esa palabra.

Cada una de estas partes estará construida dentro de un elemento `div`, marcada con un atributo `id` en HTML, para poder ser referidas fácilmente en hojas de estilo CSS. Cuando sea conveniente, se podrán utilizar en lugar de `div` elementos de HTML5 (`header`, `footer`, `nav`, etc).

- Menú de opciones. En todas las páginas habrá un menú desde el que se podrá acceder, mediante enlaces, al menos a la página principal (con el texto "Inicio"), a la de ayuda (con el texto "Ayuda"), a la de usuario si el usuario está autentica (con el texto "Mi página"), al documento de contenidos en formato XML ("Contenidos XML") y JSON ("Contenidos JSON"). Estas opciones de menú estarán en cada página, salvo si la opción apunta a la página en la que se está, en cuyo caso no saldrá la opción correspondiente.
- Se utilizarán hojas de estilo CSS para determinar la apariencia de la práctica. Estas hojas definirán al menos el tamaño y el tipo de la letra, y el color de fondo de cada una de las partes (elementos) marcadas con un *id*, tal como se

indica en el apartado anterior. Además, elementos que deban tener el mismo aspecto deberían estar en una misma clase CSS, para poder gestionarlo de forma común.

- Se utilizará Bootstrap para la maquetación (*layout*) de las páginas, de forma que funcionen adecuadamente tanto en navegadores de escritorio como en móviles.

- En la página principal se ofrecerá un listado de todas las palabras almacenadas en el sitio, paginado de 5 en 5 palabras, ordenadas por orden inverso de creación (las páginas más nuevas primero). Junto a cada palabra se mostrará su descripción resumida e imagen principal, según se obtuvo automáticamente de Wikipedia en español, el nombre de usuario que la creó, y su número de votos.

- Para ver la página de información de una palabra, un usuario simplemente incluirá esa palabra en el formulario de palabras (que, según se ha indicado, aparecerá en todas las páginas del sitio). Tanto si la palabra ya existe en MisPalabras como si no, se mostrará su página de información (ver más abajo cómo es esta página).

- En la página de una palabra aparecerá, si no está aún almacenada en el sitio:

  - La palabra, en tipo de letra grande.
  - La definición resumida de la palabra según Wikipedia en español (si no existiera, una nota indicando "esta palabra no está en Wikipedia en español).
  - La imagen principal de la palabra en Wikipedia en español (si no existiera, una nota indicando "esta palabra no tiene imagen en Wikipedia en español).
  - Un botón para almacenar la palabra.

  Ver detalles en "Información automática" (apartado 24.2), más adelante, sobre cómo conseguir la definición y la imagen de Wikipedia. La palabra sólo se almacenará (junto con la definición y la imagen) si el usuario pulsa el botón de almacenar.

- En la página de una palabra aparecerá, si ya está almacenada en el sitio:

  - La misma información que aparece si la palabra aún no está almacenada, salvo el botón para almacenar la palabra, que no aparecerá.

    - Los comentarios y enlaces de usuario, y la información automática (ver detalle más adelante) que se hayan almacenado para esa palabra.
    - Las opciones de información automática que no se hayan almacenado para esa palabra, como botones que al pulsarlos, obtenga y almacene la información automática correspondiente.
    - Una caja de texto para introducir un comentario. Si se introduce un comentario en ella, y se pulsa el botón correspondiente, el comentario quedará almacenado, y se volverá a ver la página de la palabra.
    - Una caja de texto para introducir un enlace relacionado. Si se introduce un enlace (URL) en ella, y el pulsa el botón correspondiente, el enlace quedará almacenado, y si la página correspondiente tiene "tarjeta embebida", se obtendrá y se almacenará también. Al terminar, se volverá a ver la página de la palabra.
    - El número de votos que ha recibido la palabra, junto a un botón para dar un voto a la palabra (sólo si el usuario no se lo ha dado ya).
    - Los comentarios, los enlaces de usuario y la información automática aparecerán junto al nombre del usuario que los almacenó.

- Para obtener información automáticamente de sitios soportados (incluida la Wikipedia en español), se utilizará la API del sitio al que corresponda, o quizás en algunos casos, ser hará un análisis de las páginas HTML del sitio. Ver detalles en "Información automática" (apartado 24.2), más adelante.

- En general, la forma de funcionamiento para obtener información automáticamente de sitios soportados será la siguiente:

    - El usuario pulsará, en la página de una palabra, el botón correspondiente al sitio soportado.
    - La aplicación, vía HTTP, recibirá como resultado la palabra y un identificador del sitio soportado, obteniendo a partir de ellas la URL que tendrá que usar para acceder a la información en el sitio soportado.
    - La aplicación obtendrá del sitio correspondiente (vía API, o de otras formas) la información extendida ese recurso.
    - La aplicación almacenará al información obtenida para la palabra indicada.
    - La aplicación devolverá al navegador la página de la palabra, ya con esta información.

- Para obtener información de un enlace especificado por el usuario:

    - El usuario especificará el enlace (URL) en la página de la palabra, escribiéndola en la caja de texto correspondiente, y pulsando el botón correspondiente.
    - La aplicación obtendrá el documento correspondiente a esa URL, y buscará si hay en ella una "tarjeta embebida". Ver detalles en "Tarjetas embebidas" (apartado 24.3), más adelante.
    - Si hay tarjeta, almacenará la URL y los campos indicados de la tarjeta.
    - Si no hay tarjeta, almacenará la URL.

- Cada palabra que se almacene, cada comentario, enlace o sitio soportado que almacene, quedará registrado a nombre del usuario que lo realizó.

- Página de usuario. Si el usuario está autenticado podrá acceder a la página de usuario, donde verá, en orden cronológico inverso (lo más reciente primero) el listado de todo lo que ha almacenado (palabras o comentarios, enlaces o sitios soportados para palabras). En caso de que sea una palabra, aparecerá la palabra y su definición resumida. En caso de que sea un comentario, aparecerá el comentario. En caso de que sea un enlace, aparecerá el enlace y la información de la tarjeta embebida correspondiente, si la hubiera. En caso de que sea información automática, aparecerá esa información. En todos los casos aparecerá también un enlace apuntando a la página de la palabra correspondiente.

- Documentos con todas las palabras del sitio. El sitio ofrecerá un documento JSON y un documento XML con todas las palabras del sitio, incluyendo, para cada palabra, la página de esa palabra. Este documento se ofrecerá cuando se pida la página principal, concatenando al final `?format=xml` o `?format=json`.

- Página de ayuda: Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento y una brevísima documentación.

Funcionamiento general:

- En general se podrá consultar el sitio sin haberse autenticado con una cuenta. En este caso se podrá ver toda la información, pero no se podrán añadir palabras ni ningún tipo de información para una palabra ya almacenada (ni se verán las opciones correspondientes).

- Cuando un visitante quiera, podrá crearse una cuenta (con lo que quedará autenticado con ella), o autenticarse en una cuenta ya existente. En este

caso, la funcionalidad quedará ligada a su cuenta. En ese momento podrá almacenar nuevas palabras, o añadir comentarios, enlaces o información de sitios soportados a cada palabra. También podrá acceder a la página de usuario.

- La aplicación se encargará de controlar que no haya más de un voto por usuario para cada aportación. Por lo tanto, si un usuario ya ha votado una aportación, y vuelve a votarlo, se ignorará su voto.
- Además, la práctica incluirá tests, que se ejecutarán con `python3 manage.py test`, y que incluirán al menos un test de API HTTP para cada recurso que sirva la aplicación, y para cada método (GET, POST) que admita cada recurso. Además, al menos la mitad de los test incluirán comprobar algo distinto del código HTTP retornado por la petición.

24.2. Información automática

La aplicación será capaz de recoger y almacenar, de forma automática, información para una palabra de lo siguientes sitios.

24.2.1. Wikipedia en español: definición resumida

Se recogerá la definición resumida de la palabra tal y como aparece en la Wikipedia en español. Por ejemplo, para la palabra "silla":

- URL: https://es.wikipedia.org/w/api.php?action=query&format=xml&titles=silla&prop=extracts&exintro&explaintext
- Formato: XML
- Elemento a recoger: `api¿query¿pages¿page¿extract`

Más información en la página API de Mediawiki[23].

24.2.2. Wikipedia en español: imagen principal

Se recogerá la imagen principal tal y como aparece en la Wikipedia en español. Por ejemplo, para la palabra "silla":

[23] API de Mediawiki:
https://www.mediawiki.org/wiki/API:Main_page

- URL: https://es.wikipedia.org/w/api.php?action=query&titles=silla&prop=pageimages&format=json&pithumbsize=200
- Formato: JSON
- Elemento a recoger: `query¿pages¿<id>¿thumbnail¿source`

Más información en la página API de Mediawiki[24].

24.2.3. DRAE: definición

Se recogerá la definición del Diccionario de la RAE. Por ejemplo, para la palabra "silla":

- URL: https://dle.rae.es/silla
- Formato: HTML
- Elemento a recoger: `meta (property=.°g:description")¿content`

Más información: Documentos con tarjeta embebida (subsección 24.3). Importante: para poder acceder a DRAE es preciso usar una cabecera `User-Agent` adecuada. Consulta la subsección 24.8 para más detalles.

24.2.4. Flickr: imagen de etiqueta

Se recogerá la primera imagen de Flickr, usando la palabra como etiqueta. Por ejemplo, para la palabra "silla":

- URL: https://www.flickr.com/services/feeds/photos_public.gne?tags=silla
- Formato: XML
- Elemento a recoger: `feed¿entry¿link (rel=.enclosure")¿href`

Más información en la página Public Feed de Flickr[25].

[24] API de Mediawiki:
https://www.mediawiki.org/wiki/API:Main_page

[25] Public Feed de Flickr:
https://www.flickr.com/services/feeds/docs/photos_public/

24.2.5. Apimeme: imagen de meme

Se ofrecerá al usuario un formulario con una caja de texto y un menú donde pueda elegir entre al menos tres memes. La palabra se usará para el texto superior del meme, y el texto introducido por el usuario para el texto inferior. Por ejemplo, para la palabra "silla", habiendo elegido el meme "Afraid-To-Ask-Andy" y el texto "¿o no?":

- URL: http://apimeme.com/meme?meme=Afraid-To-Ask-Andy&top=mesa&bottom=%C2%BFo+no%3F
- Formato: Imagen
- Elemento a recoger: la imagen generada

Más información en la página API Meme Generator[26].

24.2.6. Otra información automática

Además de lo sitios anteriores, puedes proponer otros sitios para descargar información automática. La información extendida de estos recursos reconocidos ha de ser accesibles públicamente (el acceso mediante un token de aplicación se considera público), y proporcionar datos en formato XML, JSON, texto o imagen. Si hay algún tipo de sitio para información automática que querrías utilizar, coméntalo con los profesores para que te indiquen si es válido. En caso de ser aceptado como válido, estos sitios para información automática serán puntuados positivamente, teniendo en cuenta la iniciativa del alumno que los propuso. En este caso, documéntalos en el documento de descripción de tu práctica, de forma similar a como se han documentado los anteriores (ejemplo de URL, formato, elementos a recoger, etc.). Ten en cuenta que sirven sitios para información automática que sólo funcionen con ciertos tipos de palabras (nombres de comidas, nombres de ciudades, etc.) o en ciertos idiomas (por ejemplo, sólo funcionen en inglés).

Si quieres buscar servicios que ofrezcan APIs que podrían ser utilizadas, puedes buscarlos en Internet. Algunas listas que te pueden resultar interesantes:

- Programmable Web API Directory: https://www.programmableweb.com/apis/directory.
- List of Free and Open Public APIs: https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/

[26] API Meme Generator: http://apimeme.com/?ref=apilist.fun

También puedes usar sitios que proporcionan una API de acceso a sitios terceros, como por ejemplo (ten en cuenta que en estos casos muy habitualmente necesitarás token de acceso, que normalmente se puede conseguir de forma gratuita):

- RapidAPI: https://rapidapi.com

24.3. Documentos con tarjeta embebida

En el caso de los enlaces indicados por el usuario, se comprobará si tienen "tarjeta embebida" en su HTML, y en ese caso se extraerá información de ella para almacenarla con el enlace. Consideraremos como "tarjeta embebida" al marcado con propiedades de Open Graph[27]. Se atenderá específicamente a las siguientes propiedades (las dos, o la que exista entre ellas):

- `og:description`. Descripción del documento. Si no existiera, se puede usar también `og:title`
- `og:image`. Enlace (URL) de una imagen relacionada con el documento.

Las propiedades Open Graph normalmente se encuentran como metadatos en la cabecera (`head`) del documento HTML. Por ejemplo:

```
<html>
  <head>
  <title>Este es el titulo</title>
  <meta property="og:description" content="Esta es una descipción del documento" />
  <meta property="og:image" content="https://..../imagen.jpg" />
  ...
</head>
...
```

Para obtener los elementos interesantes del documento HTML puede usarse el módulo `html.parser`[28][29] de Python3, o algún otro módulo que ayude en la identificación de elementos HTML, como BeautifulSoup4[30].

Muchos sitios incluyen tarjetas embebidas basadas en Open Graph. Se puede probar, por ejemplo, alguno de los siguientes:

[27] https://ogp.me/
[28] `html.parser`: https://docs.python.org/3/library/html.parser.html
[29] Ejemplo de uso de `html.parser`: https://stackoverflow.com/a/36650753/2075265
[30] https://pypi.org/project/beautifulsoup4/

- Imágenes en Pixabay. Incluyen campos `og:title` y `og:image`. Ejemplo: https://pixabay.com/photos/chair-couch-furniture-road-1840011/

- Palabras en el Diccionario de la RAE. Incluyen campos `og:description` y `og:image`. Ejemplo: https://dle.rae.es/silla

24.4. Despliegue

La práctica deberá estar desplegada en algún sitio de Internet, de forma que pueda accederse a ella. Deberá mantenerse desplegada y activa al menos desde el día de entrega de la práctica, hasta el día del cierre de actas.

Para el despliegue, se puede utilizar Python Anywhere[31], que proporciona un plan gratuito que incluye suficientes recursos como para poder desplegar la práctica.

Si el alumno así lo desea, puede considerarse desplegar en un ordenador dedicado (por ejemplo, una Raspberry Pi accesible directamente desde Internet, alojada en su hogar), o en servicios como Google Computing Engine[32]. En general, dado que este tipo de despliegues no podrá contar con una ayuda detallada por los profesores, estará algo más valorado.

En el caso de que la práctica se despliegue en Python Anywhere, hay que tener en cuenta que sus máquinas virtuales tienen cortado el acceso a todos los sitios de Internet salvo los que están en una "lista blanca"[33] (*whitelist*). Esto afectará a vuestro despliegue de dos formas:

- Al clonar vuestro repositorio git dentro de la máquina virtual, para tener el código de vuestra aplicación. No debería dar problemas, porque el sitio GitLab de la ETSIT, donde está vuestro código fuente, está ya en la lista blanca.

- Cuando vuestra aplicación se conecte para obtener la información extendida de un recurso, si el sitio al que la aplicación se tiene que conectar para conseguir el documento JSON o XML no está en la lista blanca, vuestra aplicación no se podrá conectar. Wikipedia y algunos otros sitios para recibir información automática (como Flickr) están ya en la lista blanca. Pero otros sitios que podéis estar usando, no. Entre ellos, no está (ni van a poner) el sitio de la RAE.

[31] Python Anywhere: https://pythonanywhere.com
[32] GCP Engine Free: https://cloud.google.com/free/
[33] Lista blanca de Python Anywhere: https://www.pythonanywhere.com/whitelist/

Para evitar los problemas con los sitios que no estén en la lista blanca, os pedimos que si hacéis el despliegue en Python Anywhere:

- Tiene que funcionar al menos con recursos reconocidos de Wikipedia y Flickr, recogiendo los documentos correspondientes, como indica el enunciado.
- Para los demás recursos reconocidos que hayáis implementado, tenéis dos opciones:
  - Si están en la lista blanca de Python Anywhere, funcionarán sin problemas sin hacer nada especial, si os funcionaban ya en las pruebas locales, así que también deberían funcionar en el despliegue.
  - Si no están en la lista blanca de Python Anywhere, aseguraos de que la base de datos que subís al despliegue de vuestra práctica incluya información automática de esos sitios para algunas palabras.

Tened en cuenta que si usáis otras plataformas para el despliegue, puede que os encontréis problemas similares. Y tened en cuenta también que en cualquier caso, nosotros probaremos la práctica en otros despliegues, así que todos los recursos reconocidos que hayáis implementado deben funcionar correctamente si no hay restricciones de conexión.

Tenéis más detalles sobre cómo se hace un despliegue de una aplicación Django en Python Anywayre en el vídeo "Django: Despliegue en Python Anywhere"[34], que explica cómo desplegar allí la aplicación `django-youtube-4`[35] (ver también el fichero `README.md` de esa aplicación para más detalles).

24.5. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habéis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio

[34] https://www.youtube.com/watch?v=hlZPC5L2Itc

[35] https://github.com/CursosWeb/Code/tree/master/Python-Django/django-youtube-4

- Visualización de la página de palabra en formato JSON y/o XML, de forma similar a como se ha indicado para la página principal.
- Permitir que un usuario autenticado pueda eliminar su cuenta.
- Permitir que el voto a una página se pueda revertir (quitar), y que no cuente más.
- Generación de un documento XML y/o JSON para los comentarios puestos en el sitio.
- Incorporación de otros sitios de información automática. Se valorará especialmente la búsqueda de otros tipos de recurso no descritos en este enunciado, y la implementación de tipos de recurso que requieran token de autenticación (en este caso, atención a no subir el token de autenticación a GitLab).
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario de la aplicación tendrá en cuenta lo que especifique el navegador.
- Inclusión de imágenes (no solo texto) en los comentarios. Esto puede hacerse de dos formas: quien suba un comentario, además de rellenar una caja de texto con el comentario, puede indicar también la URL de una imagen, que se mostrará junto al comentario, o bien subiendo una imagen a la aplicación, que se mostrará junto al comentario (se valorará más la segunda opción, y se pueden implementar las dos).
- Mejora de los tests de la práctica, incluyendo test de condiciones de error, test de escenarios con más de una invocación de recurso, tests de API Python, etc.

24.6. Entrega de la práctica

- **Fecha límite de entrega (convocatoria ordinaria):** domingo, 29 de mayo de 2022 a las 23:55 (hora española peninsular)
- **Fecha límite de entrega (convocatoria extraordinaria):** domingo, 3 de julio de 2022, a las 23:55 (hora española peninsular)

La entrega de la práctica consiste en:

1. **Rellenar un formulario** enlazado en el sitio de la asignatura en el aula virtual (es el formulario "Usuarios en el GitLab de la ETSIT", que normalmene habrás rellenado ya para la entrega de las miniprácticas y microprácticas).

2. **Subir tu práctica a un repositorio en el GitLab de la Escuela**. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado una derivación (fork) del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://gitlab.etsit.urjc.es/cursosweb/practicas/server/final-mispalabras/

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitLab.

   Se recomienda mantener el repositorio como privado, hasta el momento en que se entregue la práctica.

3. **Entregar un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional**, si se han realizado opciones avanzadas. Los vídeos 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). 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, Twitch, o YouTube). Los vídeos de más de tres minutos tendrán penalización.

   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). OBS Studio[36] está disponible para varias plataformas (Linux, Windows, MacOS). 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.

Sobre la entrega del repositorio:

- Se han de entregar los siguientes ficheros:

[36] OBS Studio: https://obsproject.com/

- El repositorio en la instancia GitLab de la ETSIT deberá contener un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos. Deberá poder ejecutarse directamente con `python3 manage.py runserver` desde un entorno virtual en el que esté instalado Django 3.0.3. También ejecutará los tests con `python3 manage.py test`, desde el mismo entorno virtual.
- La base de datos habrá de tener datos suficientes como para poder probarlo. Estos datos incluirán al menos dos usuarios, con al menos diez aportaciones en total, cinco comentarios puestos en total, y al menos seis aportaciones votadas por cada usuario.
- Un fichero `requirements.txt`, con un nombre de paquete Python por línea, para indicar Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, si es que fuera el caso. Este fichero no ha de incluir Django, dado que ya se supone que hace falta. Si es posible, se recomienda escribir este fichero en el formato que entiende `pip install -r requirements.txt`
- Cualquier fichero auxiliar que pueda hacer falta para que funcione la práctica, si es que fuera el caso.

- Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica: se recomienda hacer un copia y pega de estos datos en el formulario):

  - Nombre y titulación.
  - Nombre de su cuenta en el laboratorio del alumno.
  - URL del vídeo demostración de la funcionalidad básica
  - URL del vídeo demostración de la funcionalidad optativa, si se ha realizado funcionalidad optativa
  - URL de la aplicación desplegada
  - Cuenta (login) y contraseña de los usuarios que están dados de alta en 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.

  Estos datos se escribirán siguiendo estrictamente el siguiente formato:

```
# Entrega practica

## Datos

* Nombre:
* Titulación:
* Despliegue (url):
* Video básico (url):
* Video parte opcional (url):
* Despliegue (url):
*

## Cuenta Admin Site

* usuario/contraseña

## Cuentas usuarios

* usuario/contraseña
* usuario/contraseña
* ...

## Resumen parte obligatoria

## Lista partes opcionales

* Nombre parte:
* Nombre parte:
* ...
```

Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitLab los incluya como enlaces "pinchables".

24.7. 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.

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

Los documentos XML que genere la práctica deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el del módulo xml.sax de Python. Los documentos JSON generados deberán ser correctos desde el punto de vista de la sintaxis JSON, y por lo tanto reconocibles por un reconocedor JSON, como por ejemplo el del módulo json de Python

24.8. Preguntas frecuentes

A continuación, algunas preguntas relacionadas con el enunciado de esta práctica, junto con sus respuestas:

- Cuando despliego mi práctica en Python Anywhere, algunos recursos reconocidos no me funcionan, pero otros (YouTube entre ellos), sí. Todo me funciona bien en mi versión local. ¿Qué está pasando?

  Las máquinas virtuales de Python Anywhere están limitadas en cuanto a los sitios a los que se pueden conectar: sólo se pueden conectar a aquellos que están en una cierta "lista blanca". Por eso, si el sitio al que tu programa se tiene que conectar para obtener recursos no está en la lista blanca, no va a poder descargarse el documento XML o JSON de esos recursos. Para evitar problemas, en el caso de despliegue en Python Anywhere pedimos que funcionen bien los recursos reconocidos que están en la lista blanca, y para los demás, que tengan recursos en la base de datos de despliegue. Más detalles en el apartado sobre despliegue de este enunciado (25.5).

- En la pagina de información que se menciona en el enunciado, ¿qué hay que incluir en el apartado de documentación?

  Casi que lo que queráis, lo importante es tener la página. Puede ser por ejemplo un resumen de un párrafo de lo que hace la aplicación.

- ¿Qué es la "API key" en la API de algunos sitios, como Last.fm y GitLab?

  Algunas API de servicio, entre ellas la de Last.fm y GitLab, requieren el uso de una "API key" (clave de API) para poder usarla. Normalmente, estas claves las usa el servicio para evitar abusos, o para limitar lo que se puede hacer con su API. El caso es que si no se incluye la clave de API en cada GET que se hace al servicio, no se reciben los datos (el documento XML o JSON).

  Es habitual que estas claves se obtengan creándose una cuenta en el servicio en cuestión, y luego obteniendo la clave en una página al efecto, estando autenticados con el servicio.

Por ejemplo, en el caso de Last.fm, hay que ir a la página de petición de claves de API[37], donde (una vez autenticados con una cuenta de Last.fm), rellenaremos los dato que nos pide: "contact email", "application name" (cualquier nombre de aplicación, por ejemplo MisPalabras), "application description" (cualquier descripción por ejemplo "App to manage Last.fm artists"). En este caso, puedes ignorar los campos "callback url" y "application homepage". Cuando se hayan enviado estos datos, te devolverá entre otros datos tu "API key". Esa es la que tendrá que usar en tus llamadas a Last.fm.

Como las claves de API son personales, mantenlas en secreto. En particular, no las subas a repositorios públicos, pues cualquiera podrá verlas (y usarlas). Si quieres que el repositorio de tu práctica sea público, incluye la clave en un fichero que tengas sólo en tu disco, y no subas al repositorio git. Por ejemplo, puedes poner la clave en un fichero `apikeys.py` del estilo de este:

```
LASTFM_APIKEY = "012345678"
```

Luego, en el módulo Python que la uses (por ejemplo `views.py`), pondrás algo como:

```
from .apikeys import LASTFM_APIKEY
```

Y ya puedes usar la clave en tu código. Este fichero `apikeys.py` no lo subirás al repositorio git público.

Si usas claves de API en tu práctica, indícalo claramente en el fichero de entrega de la práctica, y o bien sube una clave de API válida (si el repositorio de entrega es privado) para que la podamos probar, o bien indica en qué fichero hay que ponerla, y cómo se consigue una clave API válida para el servicio que estés usando, de forma que la podamos conseguir y ejecutar tu práctica.

- ¿Es necesario utilizar los mecanismos provistos por Django para el control de sesiones y autenticación?

  En principio, esa es la solución recomendada. El principal problema suele ser asegurarse de que cualquier mecanismo alternativo funciona al menos tan bien como el de Django, lo que no es en general trivial. De todas formas, salvo muy buenos motivos, la aplicación es una aplicación Django, y por lo tanto cuantas más facilidades de Django se usen (bien usadas), mejor.

[37] Lastfm Create API Account: https://www.last.fm/api/account/create

- ¿Dónde puedo realizar el despliegue de la aplicación?

  El despliegue puede realizarse en cualquier ordenador que esté conectado permanentemente a Internet durante el periodo de corrección, en una dirección accesible desde cualquier navegador conectado a su vez a Internet. Esto puede ser por ejemplo un ordenador personal en un domicilio con acceso permanente a Internet, adecuadamente configurado (puede ser una Raspberry Pi o similar, si se busca una solución simple y de bajo coste). También puede ser un servicio en Internet, por ejemplo uno gratuito como los que ofrecen Google (instrucciones[38], precios[39]), o PythonAnywhere (instrucciones[40], precios[41]). Los profesores podremos ayudar de forma más detallada con PythonAnywhere.

- Al acceder a la página en DRAE recibo un error. ¿Qué puedo hacer?

  Si accedes a la página de una palabra en el DRAE desde el navegador, normalmente verás esa página sin problemas. Pero si lo haces desde un programa (por ejemplo, desde un script Python o desde tu aplicación Django) obtienes un error de acceso no permitido ("Forbidden", o similar). El problema al parecer tiene que ver con que el servidor que sirve las páginas de DRAE comprueba la cabecera de identificación del cliente (`User-Agent`) y espera que corresponda a la de un navegdor. Por ello, podrás acceder a la página si usas una cabecera como:

  ```
  User-Agent: Mozilla/5.0 (X11) Gecko/20100101 Firefox/100.0
  ```

  Por ejemplo, puedes usar un código similar a este:

  ```
  import urllib.request

  url = "https://dle.rae.es/silla"
  user_agent = "Mozilla/5.0 (X11) Gecko/20100101 Firefox/100.0"
  headers = {'User-Agent': user_agent}
  req = urllib.request.Request(url, headers=headers)
  ```

[38] GCP Quickstart Using a Linux VM:
https://cloud.google.com/compute/docs/quickstart-linux

[39] Google Compute Engine Pricing:
https://cloud.google.com/compute/pricing

[40] Capítulo "Deploy!" de Django Girls Tutorial:
https://tutorial.djangogirls.org/en/deploy/

[41] PythonAnywhere Plans and Pricing:
https://www.pythonanywhere.com/pricing/

```
with urllib.request.urlopen(req) as response:
    html = response.read().decode('utf8')
```

- Algunos documentos no se descargan de PythonAnywhere. ¿Qué está pasando?

  Algunos documentos estáticos (por ejemplo, la hora de estilo CSS que estoy usando) no se carga en el navegador cuando despliego la práctica en PythonAnywhere. Sin embargo, cuando pruebo en mi ordenador, o en los ordenadores del laboratorio, todo parece ir bien. ¿Qué está pasando.

  Lo que ocurre es que para servir los ficheros estáticos (los que no genera tu aplicación Django, sino que simplemente los sirve a partir de ficheros ya existentes), hay que indicarle a PythonAnywhere qué ficheros son esos, y para qué recursos deben servirse. Es muy típico que esto ocurra, por ejemplo, con el fichero que tenga la hoja de estilo CSS. Puedes ver esto en la consola web, donde indica:

  ```
  Static files:

  Files that aren't dynamically generated by your code, like CSS, JavaScript or
  ```

  Añade en esa tabla tus ficheros, y si no hay más problemas te funcionará.

25. Proyecto final: LoVisto (2021)

Este es el enunciado del proyecto final del curso 2020-2021, tanto para la convocatoria ordinaria como para la extarordinaria.

La práctica final de la asignatura consiste en la creación de una aplicación web, llamada "LoVisto", que permitirá gestionar aportaciones de los usuarios, que serán enlaces (URLs) a vídeos, noticias y otra información que los usuarios vayan viendo por la red y les resulte interesante. Cuando las aportaciones correspondan con ciertos patrones de recurso, de ciertos sitios (patrones reconocidos), se mostrará cierta información ofrecida por esos sitios, a modo de una "previsualización". Los usuarios podrán añadir aportaciones con enlaces a páginas que hayan visto, comentarlos, puntuarlos, compartirlos, etc. A continuación se describe el funcionamiento y la arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

25.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones (*apps*) Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de Django.
- Para implementar usuarios, cuando sea necesario, se usará como base el sistema de autenticación de usuarios que proporciona Django[42].
- Todas las bases de datos que contenga la aplicación tendrán que ser accesibles vía la interfaz que proporciona el "Admin Site" (además de lo que pueda hacer falta para que funcione al aplicación).
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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, preferentemente en la parte superior.

[42] User Authentication in Django:
https://docs.djangoproject.com/en/3.0/topics/auth/

- Un formulario 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, este formulario permitirá al visitante introducir su identificador de usuario y su contraseña.
  - En caso de que ya se haya entrado, este formulario mostrará el identificador del usuario y permitirá salir de la cuenta (logout). Este formulario aparecerá preferentemente en la parte superior derecha.
- Un menú de opciones, como barra, preferentemente debajo de los dos elementos anteriores (banner y caja de entrada o salida). Los contenidos de este menú se indican más adelante en el enunciado.
- Un pie de página con una nota de atribución, indicando "Esta aplicación enlaza a XXX, YYY, ZZZ y otros muchos sitios", siendo XXX, YYY y ZZZ los tres últimos sitios que los usuarios han referenciado en sus aportaciones, y siendo cada uno de ellos un enlace al sitio en cuestión.

Cada una de estas partes estará construida dentro de un elemento `div`, marcada con un atributo `id` en HTML, para poder ser referenciadas fácilmente en hojas de estilo CSS. Cuando sea conveniente, se podrán utilizar en lugar de `div` elementos de HTML5 (`header`, `footer`, `nav`, etc).

- Se utilizarán hojas de estilo CSS para determinar la apariencia de la práctica. 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 un *id*, tal como se indica en el apartado anterior. Además, elementos que deban tener el mismo aspecto deberían estar en una misma clase, para poder gestionarlo de forma común.

- Se utilizará Bootstrap para la maquetación (*layout*) de las páginas, de forma que funcionen adecuadamente tanto en navegadores de escritorio como en móviles.

- Para obtener información correspondiente a un recurso reconocido, se utilizará la API del sitio al que corresponda, o quizás en algunos casos, ser hará un análisis de las páginas HTML del sitio. En general, la forma de funcionamiento será la siguiente:

  - El usuario que quiera realizar una aportación, rellenará un formulario con la URL de la aportación que realiza, y un título para ella.
  - La aplicación recibirá esta URL, y la analizará, obteniendo a partir de ella el nombre del sitio, y el nombre del recurso.

- Si el recurso es uno de los reconocidos, se obtendrá del sitio correspondiente (vía API, o de otras formas) la información extendida ese recurso. Ver detalles en "Recursos reconocidos" (apartado 25.2), más adelante.
- Si no es uno de los recursos reconocidos, se extraerá información directamente de la página HTML del recurso. Ver detalles en "Recursos no reconocidos" (apartado 25.3).
- En ambos casos, esa información se almacenará en la base de datos (normalmente como string HTML), y se mostrará junto a la URL cuando se especifique que se muestra la "información extendida" para esa URL.

Funcionamiento general:

- En general, para utilizar el sitio, no hará falta autenticarse con una cuenta. Si no se está autenticado, se podrá ver toda la información, salvo la página de usuario, y no se podrán realizar aportaciones, votar ni poner comentarios.
- Cuando un visitante quiera, podrá autenticarse en una cuenta ya existente. En este caso, la funcionalidad quedará ligada a su cuenta. En ese momento podrá ya realizar aportaciones, votar, o poner comentarios. También podrá acceder a la página de usuario, que mostrará la inforamción de su usuario.

25.2. Recursos reconocidos

La aplicación reconocerá ciertos patrones recurso, que se describen a continuación. Si la URL que se incluye con una aportación es uno de estos recursos reconocidos, la aplicación tendrá que recoger información adicional, y almacenarla en la base de datos.

La práctica tendrá que funcionar con al menos tres patrones de recursos reconocidos, entre los siguientes, uno de los cuales deberá ser la predicción AEMET para un municipio.

25.2.1. Predicción AEMET para un municipio

- Ejemplo:
  `http://www.aemet.es/es/eltiempo/prediccion/municipios/getafe-id28065`
- Patrones reconocidos:
  - Sitio: `www.aemet.es`, `aemet.es`
  - Patrón de recurso:
    `/es/eltiempo/prediccion/municipios/{municipio}-id{num}`

- Información extendida:
  - Ejemplo: https://www.aemet.es/xml/municipios/localidad_28065.xml
  - Patrón de url: `https://www.aemet.es/xml/municipios/localidad_{num}.xml`
  - Formato: XML
  - Datos mínimos:
    - Municipio, provincia
    - Para todos los días que tengan predicción, temperatura máxima y mínima, sensación térmica máxima y mínima, humedad relativa máxima y mínima.
    - Nota de copyright
  - Ejemplo:

```
<div class="aemet">
  <p>Datos AEMET para Getafe (Madrid)</p>
  <ul>
  <li>2021-05-08. Temperatura: 11/22, sensación: 16/25, humedad: 60/75.</li>
  <li>2021-05-09. Temperatura: 10/18, sensación: 19/21, humedad: 50/66.</li>
  ...
  <li>2021-05-14. Temperatura: 9/12, sensación: 10/15, humedad: 90/95.</li>
  </ul>
  <p>Copyright AEMET.
    <a href="http://www.aemet.es/es/.../getafe-id28065">Página
      original en AEMET</p>
</div>
```

25.2.2. Página Wikipedia en español

- Ejemplo: https://es.wikipedia.org/wiki/Astronauta
- Patrones reconocidos:
  - Sitio: `es.wikipedia.org`
  - Patrón de recurso: `/wiki/{articulo}`
- Información extendida:
  - Ejemplo texto: https://es.wikipedia.org/w/api.php?action=query&format=xml&titles=Astronauta&prop=extracts&exintro&explaintext

- Ejemplo imagen:
  https://es.wikipedia.org/w/api.php?action=query&titles=Astronauta&prop=pageimages&format=json&pithumbsize=100
- Patrón de URL (texto):
  `https://es.wikipedia.org/w/api.php?action=query&format=xml&titles={articulo}&prop=extracts&exintro&explaintext`
- Patrón de URL (imagen):
  `https://es.wikipedia.org/w/api.php?action=query&titles={articulo}&prop=pageimages&format=json&pithumbsize=100`
- Formato: XML (text) y JSON (image)
- Datos mínimos:
  - Primeros 400 caracteres del texto.
  - Imagen.
  - Nota de copyright
- Ejemplo:

```
<div class="wikipedia">
  <p>Artículo Wikipedia: Astronauta</p>
  <img src="url de la imagen principal">
  <p>[Texto del principio de la página]</p>
  <p>Copyright Wikipedia.
    <a href="https://es.wikipedia.org/wiki/Astronauta">Artículo
      original</a>.</p>
</div>
```

25.2.3. Vídeo de YouTube

- Ejemplo:
  https://www.youtube.com/watch?v=IfoSqaxJsAM
- Patrones reconocidos:
  - Sitio: `www.youtube.com`, `youtube.com`
  - Patrón de recurso:
    `/watch?v={video}`
- Información extendida:

- Ejemplo:
  https://www.youtube.com/oembed?format=json&url=https://www.youtube.com/watch?v=IfoSqaxJsAM
- Patrón de URL:
  `https://www.youtube.com/oembed?format=json&url=https://www.youtube.com/watch?v={video}`
- Formato: JSON
- Datos mínimos:
  - Título.
  - Autor.
  - Vídeo embebido en iframe.
- Ejemplo:

```
<div class="youtube">
  <p>Video YouTube: Presentación de la asignatura</p>
  <iframe width="560" height="315"
    src="https://www.youtube.com/embed/IfoSqaxJsAM"
    title="YouTube video player" frameborder="0"
    allow="accelerometer; encrypted-media; gyroscope; picture-in-picture"
    allowfullscreen>
  </iframe>
  <p>Autor: CursosWeb.
    <a href="https://www.youtube.com/watch?v=IfoSqaxJsAM">Video
      en YouTube</a></p>
</div>
```

25.2.4. Nota en Reddit

- Ejemplo:
  https://www.reddit.com/r/django/comments/n842st/is_there_a_public_repo_that_shows_productionlevel/
- Patrones reconocidos:
  - Sitio: `www.reddit.com`, `reddit.com`
  - Patrón de recurso:
    `/r/{subreddit}/comments/{id}/{titulo}/`
- Información extendida:
  - Ejemplo:
    https://www.reddit.com/r/django/comments/n842st/.json

- Patrón de URL:
  `https://www.reddit.com/r/django/comments/{id}/.json`

- Formato: JSON
- Datos mínimos (obtenidos de [0][data][children][0][data]):
  - Subreddit
  - Título (title)
  - Texto (selftext)
  - Aprobación (upvote_ratio)
  - URL (url)
- Comentarios. Si la URL es del sitio `https://i.redd.it/`, normalmente se refiere a una imagen, y por lo tanto habrá que presentarla como tal en la información extendida. Si no aparecen los campos anteriores, puede considerarse que la información extendida sólo tiene los campos que se hayan encontrado. Si se mejora el código para que se reconozcan otros casos, se considerará una mejora a la práctica: documentar esos cambios en el documento de descripción de la práctica.
- Ejemplos:

```
<div class="reddit">
  <p>Nota Reddit: Is there a public repo that shows...</p>
  <p>[texto]</p>
  <p><a href="https://www.reddit.com/r/django/comments/n842st">Publicado
    en [subreddit]</a>. Aprobación: [aprobacion].</p>
</div>

<div class="reddit">
  <p>Nota Reddit: Distribution of the surname Ryan...</p>
  <img src="[imagen_url]">
  <p><a href="https://www.reddit.com/r/dataisbeautiful/comments/n8azu6/">Publicado
    en [subreddit]</a>. Aprobación: [aprobacion].</p>
</div>
```

25.2.5. Otros recursos reconocidos

Además de las anteriores, puedes proponer otros tipos de recursos reconocidos para tu aplicación. La información extendida de estos recursos reconocidos ha de ser accesibles públicamente (el acceso mediante un token de aplicación se considera público), y proporcionar datos en formato XML o JSON. Si hay algún tipo de recurso reconocido querrías utilizar, coméntalo con los profesores para que te indiquen si es válido. En caso de ser aceptado como válido, estos tipos de recursos

reconocidos serán puntuados positivamente, teniendo en cuenta la iniciativa del alumno que los propuso. En este caso, documéntalos en el documento de descripción de tu práctica, de forma similar a como se han documentado los anteriores (ejemplo de URL, parones reconocidos, información extendida, etc.).

Si quieres buscar servicios que ofrezcan APIs que podrían ser recursos reconocidos, puedes buscarlos en Internet. Una lista por la que puedes comenzar es la que mantiene ProgrammableWeb[43].

25.3. Recursos no reconocidos

En el caso de que el recurso que indique el usuario no sea uno de los recursos reconocidos, habrá que extraer información extendida del propio documento HTML, si esto es posible. Para ello se utilizarán dos estrategias (primero una, y si no funciona, la otra):

- Si hay disponibles propiedades Open Graph[44], se extraerán las propiedades `og:title` y `image` (o la que exista de ellas), y se usarán para componer una información extendida que incluya el titulo y la imagen.

  Las propiedades Open Graph normalmente se encuentran como metadatos en la cabecera (`head`) del documento HTML. Por ejemplo:

  ```
  <html>
    <head>
    <title>Este es el titulo</title>
    <meta property="og:title" content="Este es el titulo" />
    <meta property="og:image" content="https://..../imagen.jpg" />
    ...
  </head>
  ...
  ```

  Ejemplo de información extendida generada de esta forma:

  ```
  <div class="og">
    <p>Este es el titulo</p>
    <img src="https://..../imagen.jpg">
  </div>
  ```

[43] Programmable Web API Directory:
https://www.programmableweb.com/apis/directory

[44] https://ogp.me/

- Si hay disponible un elemento `title`, se usará su contenido para componer la información extendida. Por ejemplo:

```
<div class="html">
  <p>Este es el titulo</p>
</div>
```

En ambos casos, se usará el módulo `html.parser`[45][46] de Python3, o algún otro módulo que ayude en la identificación de elementos HTML, como BeautifulSoup4[47].

Si no funciona ninguna de esas estrategias, o la URL no corresponde con una página HTML (pero se puede descargar), se incluirá una información extendida que indique no hay información extendida:

```
<div class="html">
  <p>Información extendida no disponible</p>
</div>
```

25.4. Funcionalidad mínima

La aplicación servirá las siguientes páginas:

- Página principal de la aplicación:

  1. Listado con los 10 últimas aportaciones del el sitio. Para cada una se mostrará su información en formato completo (ver más abajo).

  Si el visitante está autenticado como usuario, se mostrará también:

  - Para cada aportación que aparezca en la página se mostrarán también dos botones para votar (positivo, negativo), resaltando de alguna forma que el valor que se haya votado, si se hubiera votado ya ese item.
  - Formulario para realizar una aportación. Tendrá campos para un título, un texto de descripción, y un enlace (que será el enlace a la aportación en el sitio donde se ha visto). Al enviar el formulario, se creará una nueva aportación a nombre de usuario que la ha enviado
  - Listado con las últimos 5 aportaciones (formato resumido) realizadas por el usuario.

[45] `html.parser`: https://docs.python.org/3/library/html.parser.html
[46] Ejemplo de uso de `html.parser`: https://stackoverflow.com/a/36650753/2075265
[47] https://pypi.org/project/beautifulsoup4/

    - Enlace a la página del usuario.

- Página de la aportación (para cada aportación):

    - Datos de la aportación (formato completo).
    - Comentarios que haya recibido la aportación. Para cada comentario se mostrará el texto del comentario, el identificador de quien lo puso, y la fecha en que se puso.

  Si el visitante está además autenticado como usuario, se mostrará también:

    - Dos botones para votar (positivo, negativo), resaltando de alguna forma que el valor que se haya votado, si se hubiera votado ya esa aportación.
    - Formulario para poner un comentario. Tras poner el comentario, se volverá a ver la misma página de la aportación.

- Página del usuario (siempre el mismo nombre de recurso, sólo para usuarios autenticados):

    - Listado de todas las aportaciones del usuario. Cada aportación, en formato resumido.
    - Listado de todos los comentarios del usuario. Cada comentario, junto con la aportación a la que hizo ese comentario, en formato resumido.
    - Listado de todas las aportaciones votadas por el usuario. Cada voto, junto con la aportación a la que votó, en formato resumido, y una indicación de si el voto fue positivo o negativo.

- Página de todas las aportaciones:

    - Listado de todas las aportaciones realizadas al sitio, en formato resumido, ordenadas por fecha de aportación (primero las más recientes).

  La página de todas las aportaciones se ofrecerá también como un documento XML y como un documento JSON, que incluiría la misma información (el mismo listado de todas las aportaciones, cada una con su título y su enlace a la página de la aportación). Este documento se ofrecerá cuando se pida la página principal, concatenando al final `?format=xml` o `?format=json`.

- Página de información: Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento y una brevísima documentación.

Información para una aportación (formato resumido):

- La información para una aportación estará compuesta por:
  - Título
  - Enlace a la página de la aportación
- Por ejemplo, esta información se podrá presentar como:

```
<p class="resumida"><a href="[enlace_pagina]">[titulo]</a></p>
```

Información para una aportación (formato completo):

- La información completa para una aportación estará compuesta por:
  - Título
  - Enlace a la página de la aportación
  - Descripción
  - Usuario
  - Votos positivos
  - Votos negativos
  - Fecha de la aportación
  - Número de comentarios
  - Información extendida
- Por ejemplo, esta información se podrá presentar como:

```
<div class="aportacion">
  <h2>[titulo]</h2>
  <p class="descripcion">[descripcion]</p>
  <p class="datos">Contribución de [usuario], enviada en [fecha],
    [num_comentarios] comentarios,
    <a href="[enlace_pagina]">más info</p>
  <p class="votos">Positivos: [positivos]. Negativos: [negativos]</p>
  [info_extendida]
</div>
```

La aplicación se encargará de controlar que no haya más de un voto (positivo o negativo) por usuario para cada aportación. Por lo tanto, si un usuario ya ha votado una aportación, y vuelve a votarlo, se ignorará su voto (si es igual que el que

está almacenado) o se anotará el nuevo (si es distinto). Por ejemplo, si había votado una aportación con positivo, y ahora vuelve a votarla con positivo, se ignorará el segundo voto. Si vuelve a votarla, pero ahora con negativo, se cambiará el voto a negativo.

En todos los casos en que se vote, tras votar se volverá a ver la misma página en que se estaba, ahora con el voto contabilizado.

En todas las páginas habrá un menú desde el que se podrá acceder a la página principal (con el texto "Inicio"), a la de información (con el texto "Información"), a la de todas las aportaciones en formato "normal" (HTML) (con el rexto "Todo"), y descargar el listado de todas las aportaciones en formato XML ("Descarga como fichero XML") y JSON ("Descarga como fichero JSON") (que apuntará a la página de todas las aportaciones en el formato correspondiente). Estas opciones de menú estarán en cada página, salvo si la opción apunta a la página en la que se está, en cuyo caso no saldrá la opción correspondiente.

Desde este menú también se podrá cambiar, si se está autenticado como usuario, a un "modo oscuro" (se se está en el modo normal), o volver al modo normal (si se está en el modo oscuro). Este cambio se realizara cambiando el CSS que servirá para todas las páginas del sitio, mientras el visitante no vuelva a cambiar de modo.

Además, la práctica incluirá tests, que se ejecutarán con `python3 manage.py test`, y que incluirán al menos un test de API HTTP para cada recurso que sirva la aplicación, y para cada método (GET, POST) que admita cada recurso. Además, al menos la mitad de los test incluirán comprobar algo distinto del código HTTP retornado por la petición.

25.5. Despliegue

La práctica deberá estar desplegada en algún sitio de Internet, de forma que pueda accederse a ella. Deberá mantenerse desplegada y activa al menos desde el día de entrega de la práctica, hasta el día del cierre de actas.

Para el despliegue, se puede utilizar Python Anywhere[48], que proporciona un plan gratuito que incluye suficientes recursos como para poder desplegar la práctica.

Si el alumno así lo desea, puede considerarse desplegar en un ordenador dedicado (por ejemplo, una Raspberry Pi accesible directamente desde Internet, alojada en su hogar), o en servicios como Google Computing Engine[49]. En general, dado que este tipo de despliegues no podrá contar con una ayuda detallada por los profesores, estará algo más valorado.

En el caso de que la práctica se despliegue en Python Anywhere, hay que tener

[48]Python Anywhere: https://pythonanywhere.com
[49]GCP Engine Free: https://cloud.google.com/free/

en cuenta que sus máquinas virtuales tienen cortado el acceso a todos los sitios de Internet salvo los que están en una "lista blanca"[50] (*whitelist*). Esto afectará a vuestro despliegue de dos formas:

- Al clonar vuestro repositorio git dentro de la máquina virtual, para tener el código de vuestra aplicación. No debería dar problemas, porque el sitio GitLab de la ETSIT, donde está vuestro código fuente, está ya en la lista blanca.
- Cuando vuestra aplicación se conecte para obtener la información extendida de un recurso, si el sitio al que la aplicación se tiene que conectar para conseguir el documento JSON o XML no está en la lista blanca, vuestra aplicación no se podrá conectar. YouTube y algunos otros sitios para los que se pueden implementar recursos reconocidos (como Flickr) están ya en la lista blanca. Pero otros sitios que podéis estar usando, no.

Para evitar los problemas con los sitios que no estén en la lista blanca, os pedimos que si hacéis el despliegue en Python Anywhere:

- Tiene que funcionar al menos con recursos reconocidos de YouTube, recogiendo los documentos XML correspondientes, como indica el enunciado.
- Para los demás recursos reconocidos que hayáis implementado, tenéis dos opciones:
  - Si están en la lista blanca de Python Anywhere, funcionarán sin problemas sin hacer nada especial, si os funcionaban ya en las pruebas locales, así que también deberían funcionar en el despliegue.
  - Si no están en la lista blanca de Python Anywhere, aseguraos de que la base de datos que subís al despliegue de vuestra práctica incluya datos de recursos reconocidos de la plataforma en cuestión. Por ejemplo, si tenéis implementados recursos reconocidos de Last.fm, basta con que tengáis en la base de datos algunas aportaciones de un par de recursos de este sitio, que muestren que en la versión local os funcionó.

Tened en cuenta que si usáis otras plataformas para el despliegue, puede que os encontréis problemas similares. Y tened en cuenta también que en cualquier caso, nosotros probaremos la práctica en otros despliegues, así que todos los recursos reconocidos que hayáis implementado deben funcionar correctamente si no hay restricciones de conexión.

[50] Lista blanca de Python Anywhere: https://www.pythonanywhere.com/whitelist/

Tenéis más detalles sobre cómo se hace un despliegue de una aplicación Django en Python Anywayre en el video "Django: Despliegue en Python Anywhere"[51], que explica cómo desplegar allí la aplicación `django-youtube-4`[52] (ver también el fichero `README.md` de esa aplicación para más detalles).

25.6. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habéis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio
- Visualización de cualquier página en formato JSON y/o XML, de forma similar a como se ha indicado para la página principal.
- Generación de un documento XML y/o JSON para los comentarios puestos en el sitio.
- Incorporación de datos de otros tipos de recurso además de los obligatorios. Se valorará especialmente la búsqueda de otros tipos de recurso no descritos en este enunciado, la implementación de tipos de recurso que requieran token de autenticación (en este caso, atención a no subir el token de autenticación a GitLab).
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario de la aplicación tendrá en cuenta lo que especifique el navegador.
- Inclusión de imágenes (no solo texto) en los comentarios. Esto puede hacerse de dos formas: quien suba un comentario, además de rellenar una caja de texto con el comentario, puede indicar también la URL de una imagen, que se mostrará junto al comentario, o bien subiendo una imagen a la aplicación, que se mostrará junto al comentario (se valorará más la segunda opción, y se pueden implementar las dos).

[51] https://www.youtube.com/watch?v=hlZPC5L2Itc

[52] https://github.com/CursosWeb/Code/tree/master/Python-Django/django-youtube-4

- Mejora de los tests de la práctica, incluyendo test de condiciones de error, test de escenarios con más de una invocación de recurso, tests de API Python, etc.

25.7. Entrega de la práctica

- **Fecha límite de entrega (convocatoria extraordinaria):** domingo, 11 de julio de 2021 a las 23:55 (hora española peninsular)

La entrega de la práctica consiste en:

1. **Rellenar un formulario** enlazado en el sitio de la asignatura en el aula virtual.

2. **Subir tu práctica a un repositorio en el GitLab de la Escuela**. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado una derivación (fork) del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://gitlab.etsit.urjc.es/cursosweb/practicas/server/final-lovisto/

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitLab.

   Se recomienda mantener el repositorio como privado, hasta el momento en que se entregue la práctica.

3. **Entregar un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional**, si se han realizado opciones avanzadas. Los vídeos 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). 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, Twitch, o YouTube). Los vídeos de más de tres minutos tendrán penalización.

   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). OBS Studio[53] está disponible para varias plataformas (Linux, Windows, MacOS). 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.

Sobre la entrega del repositorio:

- Se han de entregar los siguientes ficheros:
  - El repositorio en la instancia GitLab de la ETSIT deberá contener un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos. Deberá poder ejecutarse directamente con `python3 manage.py runserver` desde un entorno virtual en el que esté instalado Django 3.0.3. También ejecutará los tests con `python3 manage.py test`, desde el mismo entorno virtual.
  - La base de datos habrá de tener datos suficientes como para poder probarlo. Estos datos incluirán al menos dos usuarios, con al menos diez aportaciones en total, cinco comentarios puestos en total, y al menos seis aportaciones votadas por cada usuario.
  - Un fichero `requirements.txt`, con un nombre de paquete Python por línea, para indicar Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, si es que fuera el caso. Este fichero no ha de incluir Django, dado que ya se supone que hace falta. Si es posible, se recomienda escribir este fichero en el formato que entiende `pip install -r requirements.txt`
  - Cualquier fichero auxiliar que pueda hacer falta para que funcione la práctica, si es que fuera el caso.
- Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica: se recomienda hacer un copia y pega de estos datos en el formulario):
  - Nombre y titulación.
  - Nombre de su cuenta en el laboratorio del alumno.

[53] OBS Studio: https://obsproject.com/

- URL del vídeo demostración de la funcionalidad básica
- URL del vídeo demostración de la funcionalidad optativa, si se ha realizado funcionalidad optativa
- URL de la aplicación desplegada
- Cuenta (login) y contraseña de los usuarios que están dados de alta en 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.

Estos datos se escribirán siguiendo estrictamente el siguiente formato:

```
# Entrega practica

## Datos

* Nombre:
* Titulación:
* Despliegue (url):
* Video básico (url):
* Video parte opcional (url):
* Despliegue (url):
*

## Cuenta Admin Site

* usuario/contraseña

## Cuentas usuarios

* usuario/contraseña
* usuario/contraseña
* ...

## Resumen parte obligatoria

## Lista partes opcionales
```

```
* Nombre parte:
* Nombre parte:
* ...
```

Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitLab los incluya como enlaces "pinchables".

25.8. 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.

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

Los documentos XML que genere la práctica deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el del módulo xml.sax de Python. Los documentos JSON generados deberán ser correctos desde el punto de vista de la sintaxis JSON, y por lo tanto reconocibles por un reconocedor JSON, como por ejemplo el del módulo json de Python

25.9. Preguntas frecuentes

A continuación, algunas preguntas relacionadas con el enunciado de esta práctica, junto con sus respuestas:

- Cuando despliego mi práctica en Python Anywhere, algunos recursos reconocidos no me funcionan, pero otros (YouTube entre ellos), sí. Todo me funciona bien en mi versión local. ¿Qué está pasando?

  Las máquinas virtuales de Python Anywhere están limitadas en cuanto a los sitios a los que se pueden conectar: sólo se pueden conectar a aquellos que están en una cierta "lista blanca". Por eso, si el sitio al que tu programa se tiene que conectar para obtener recursos no está en la lista blanca, no va a poder descargarse el documento XML o JSON de esos recursos. Para evitar problemas, en el caso de despliegue en Python Anywhere pedimos que funcionen bien los recursos reconocidos que están en la lista blanca, y para los demás, que tengan recursos en la base de datos de despliegue. Más detalles en el apartado sobre despliegue de este enunciado (25.5).

- En la pagina de información que se menciona en el enunciado, ¿qué hay que incluir en el apartado de documentación?

Casi que lo que queráis, lo importante es tener la página. Puede ser por ejemplo un resumen de un párrafo de lo que hace la aplicación.

- ¿Qué es la "API key" en la API de algunos sitios, como Last.fm y GitLab?

  Algunas API de servicio, entre ellas la de Last.fm y GitLab, requieren el uso de una "API key" (clave de API) para poder usarla. Normalmente, estas claves las usa el servicio para evitar abusos, o para limitar lo que se puede hacer con su API. El caso es que si no se incluye la clave de API en cada GET que se hace al servicio, no se reciben los datos (el documento XML o JSON).

  Es habitual que estas claves se obtengan creándose una cuenta en el servicio en cuestión, y luego obteniendo la clave en una página al efecto, estando autenticados con el servicio.

  Por ejemplo, en el caso de Last.fm, hay que ir a la página de petición de claves de API[54], donde (una vez autenticados con una cuenta de Last.fm), rellenaremos los dato que nos pide: "contact email", "application name" (cualquier nombre de aplicación, por ejemplo LoVisto), "application description" (cualquier descripción por ejemplo "App to manage Last.fm artists"). En este caso, puedes ignorar los campos "callback url" y "application homepage". Cuando se hayan enviado estos datos, te devolverá entre otros datos tu "API key". Esa es la que tendrá que usar en tus llamadas a Last.fm.

  Como las claves de API son personales, mantenlas en secreto. En particular, no las subas a repositorios públicos, pues cualquiera podrá verlas (y usarlas). Si quieres que el repositorio de tu práctica sea público, incluye la clave en un fichero que tengas sólo en tu disco, y no subas al repositorio git. Por ejemplo, puedes poner la clave en un fichero `apikeys.py` del estilo de este:

  ```
  LASTFM_APIKEY = "012345678"
  ```

  Luego, en el módulo Python que la uses (por ejemplo `views.py`), pondrás algo como:

  ```
  from .apikeys import LASTFM_APIKEY
  ```

  Y ya puedes usar la clave en tu código. Este fichero `apikeys.py` no lo subirás al repositorio git público.

  Si usas claves de API en tu práctica, indícalo claramente en el fichero de entrega de la práctica, y o bien sube una clave de API válida (si el repositorio

[54] Lastfm Create API Account: https://www.last.fm/api/account/create

de entrega es privado) para que la podamos probar, o bien indica en qué fichero hay que ponerla, y cómo se consigue una clave API válida para el servicio que estés usando, de forma que la podamos conseguir y ejecutar tu práctica.

- ¿Qué quiere decir "vídeo empotrado", en el caso de YouTube? ¿Cómo se hace?

  "Vídeo empotrado" es en ese caso la traducción de "embedded video", y quiere decir que el video aparezca directamente en la página, normalmente en un elemento `iframe`. En el caso de YouTube, puedes obtener el código HTML para empotrar cualquier video si, cuando lo estás viendo en el navegador, pulsas el botón de compartir ("Share"), y eliges la opción "Embed". Esto muestra un código HTML como el siguiente:

  ```
  <iframe width="560" height="315"
    src="https://www.youtube.com/embed/HZOodD84MR8"
    frameborder="0"
    allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-pictu
    allowfullscreen></iframe>
  ```

  Basta con que en él cambies el código del video (en este caso "HZOodD84MR8" por el del video que quieres empotrar, y ya está. Puedes ver también esta información, incluyendo el HTML para el iframe de un video, en el fichero de información sobre un video (ver el enunciado cuando se especifican los recursos reconocidos de YouTube).

- ¿Es necesario utilizar los mecanismos provistos por Django para el control de sesiones y autenticación?

  En principio, esa es la solución recomendada. El principal problema suele ser asegurarse de que cualquier mecanismo alternativo funciona al menos tan bien como el de Django, lo que no es en general trivial. De todas formas, salvo muy buenos motivos, la aplicación es una aplicación Django, y por lo tanto cuantas más facilidades de Django se usen (bien usadas), mejor.

- Los archivos CSS para "modo oscuro" y "modo normal", ¿dónde y cómo debemos guardarlos?

  La forma recomendada de hacerlo es mediante plantillas:

  - En el directorio de plantillas incluirías una para la hoja CSS del sitio. Esa plantilla tendría como variables de plantilla los valores que cambien de estilo normal a estilo oscuro (color de tipo de letra, color de fondo, etc.).

- Además, para cada usuario, tendrás una tabla en la base de datos donde se indicará el modo (normal o oscuro).
- Tendrás una vista en `views.py` que se encargará de generar la hoja CSS a partir de la plantilla. Esa vista es la que comprobará si la petición que está atendiendo corresponde a un usuario (en cuyo caso tendrá que obtener los valores para ese usuario de la tabla anterior), o no (en cuyo caso usará valores por defecto, siempre para el modo "normal"). Con los valores que obtenga, generará la hoja CSS a partir de la plantilla anterior.
- Por último, en `urls.py` tendrás una línea para indicar que si te piden el recurso que sirve la hoja de estilo, llamas a la vista anterior.

- ¿Dónde puedo realizar el despliegue de la aplicación?

  El despliegue puede realizarse en cualquier ordenador que esté conectado permanentemente a Internet durante el periodo de corrección, en una dirección accesible desde cualquier navegador conectado a su vez a Internet. Esto puede ser por ejemplo un ordenador personal en un domicilio con acceso permanente a Internet, adecuadamente configurado (puede ser una Raspberry Pi o similar, si se busca una solución simple y de bajo coste). También puede ser un servicio en Internet, por ejemplo uno gratuito como los que ofrecen Google (instrucciones[55], precios[56]), o PythonAnywhere (instrucciones[57], precios[58]). Los profesores podremos ayudar de forma más detallada con PythonAnywhere.

[55] GCP Quickstart Using a Linux VM:
https://cloud.google.com/compute/docs/quickstart-linux
[56] Google Compute Engine Pricing:
https://cloud.google.com/compute/pricing
[57] Capítulo "Deploy!" de Django Girls Tutorial:
https://tutorial.djangogirls.org/en/deploy/
[58] PythonAnywhere Plans and Pricing:
https://www.pythonanywhere.com/pricing/

26. Proyecto final: MisCosas (2020, mayo)

La práctica final de la asignatura consiste en la creación de una aplicación web, llamada "MisCosas", que permitirá gestionar vídeos, noticias y otra información que los usuarios vayan encontrando por la red y les resulte interesante. Los usuarios podrán ver los contenidos de sitios preseleccionados, añadir otros, elegir los que más les interesen, y compartir los que han seleccionado de distintas maneras. A continuación se describe el funcionamiento y la arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

Por un lado, la aplicación se encargará de descargar información de varios sitios de Internet para permitir a los usuarios que puedan elegirla. Por otro, permitirá a los usuarios elegir, entre ellos, qué información quieren que se les muestre para realizar su selección, y podrán compartir estas selecciones con otros usuarios.

26.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones (apps) Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de Django.
- Para implementar usuarios, cuando sea necesario, se usará como base el sistema de autenticación de usuarios que proporciona Django[59].
- Todas las bases de datos que contenga la aplicación tendrán que ser accesibles vía la interfaz que proporciona el "Admin Site" (además de lo que pueda hacer falta para que funcione al aplicación).
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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, preferentemente en la parte superior izquierda.

[59] User Authentication in Django:
https://docs.djangoproject.com/en/3.0/topics/auth/

- 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, o crearse una cuenta.
  - En caso de que ya se haya entrado, esta caja mostrará el identificador del usuario y permitirá salir de la cuenta (logout). Esta caja aparecerá preferentemente en la parte superior derecha.
- Un menú de opciones, como barra, preferentemente debajo de los dos elementos anteriores (banner y caja de entrada o salida).
- Un pie de página con una nota de atribución, indicando "Esta aplicación utiliza datos proporcionados por XXX, YYY y ZZZ", siendo XXX, YYY y ZZZ los sitios desde donde se descarga información, y siendo cada uno de ellos un enlace al sitio en cuestión.

Cada una de estas partes estará construida dentro de un elemento "div", marcada con un atributo "id" en HTML, para poder ser referenciadas fácilmente en hojas de estilo CSS.

- Se utilizarán hojas de estilo CSS para determinar la apariencia de la práctica. 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 un *id*, tal como se indica en el apartado anterior. Además, elementos que deban tener el mismo aspecto deberían estar en una misma clase, para poder gestionarlo de forma común.

- Para obtener información de cada sitio de Internet soportado por la aplicación, se utilizará la API de ese sitio, o quizás en algunos casos, ser hará un análisis de las páginas HTML del sitio. En general, la forma de funcionamiento será la siguiente:

  - Llamaremos "alimentador" a cada una de las fuentes de datos del sitio. Por ejemplo, en YouTube, cada alimentador será un canal.
  - Llamaremos "item" a cada uno de los elementos de un alimentador. Por ejemplo, en YouTube, cada item será un vídeo.
  - Se ofrecerá un elemento HTML que permita al usuario elegir qué alimentador se va a obtener del sitio.
  - La información obtenida de ese alimentador se organizará como una lista de items, que se almacenará en la base de datos.

- A partir de lo almacenado en la base de datos, se ofrecerá al usuario la lista de items para la selección.
- Se ofrecerá una forma para actualizar la información.

Puede verse más información sobre los alimentadores en la sección 26.2.

Funcionamiento general:

- En general, para utilizar el sitio, no hará falta autenticarse con una cuenta. Toda la funcionalidad estará disponible para cualquier visitante, mientras use el sitio desde el mismo navegador y tenga las cookies habilitadas.
- Cuando un visitante quiera, se podrá abrir una cuenta (y quedará autenticado en ella), o autenticarse en una cuenta ya existente. En este caso, la funcionalidad quedará ligada a su cuenta.
- Cada usuario podrá elegir cualquier ítem que se le presente, y realizar dos acciones fundamentales con él (comentarlo, y votarlo):
  - Comentar un ítem quiere decir escribir un pequeño mensaje (menos de 256 caracteres) que quedará relacionado con el item.
  - Votarlo significa darle un voto positivo o uno negativo. El resultado de las votaciones quedará relacionado con el ítem.

26.2. Alimentadores

La práctica tendrá que funcionar con al menos dos alimentadores entre los que se describen a continuación. El número 0 (YouTube canal XML) será obligatorio para todos. Además, cada alumno tendrá que implementar al menos otro, según la primera letra de su primer apellido: alimentador 1 para las letras A-D, alimentador 2 para las letras E-L, alimentador 3 para las letras M-Q, alimentador 4 para las letras R-Z.

Alimentadores descritos:

- **Alimentador 0**. YouTube (canal XML). En este caso el alimentador será el canal de YouTube, y el item será un vídeo en particular. Los últimos vídeos de un canal están disponibles como documentos XML (RSS)[60], donde el identificador del canal se puede obtener del enlace que tenemos en el navegador cuando estamos viendo el canal. Funcionamiento:

[60] Ejemplo: Para el canal *UC300utwSVAYOoRLEqmsprfg*, la url es: https://www.youtube.com/feeds/videos.xml?channel_id=UC300utwSVAYOoRLEqmsprfg

- Alimentador: canal de Youtube.
- Ítem: vídeo de YouTube.
- Elemento HTML para elegir el alimentador: formulario que permita escribir el identificador del canal.
- Elemento HTML para actualizar el alimentador: botón que actualiza con los vídeos disponibles en el canal RSS.
- Datos mostrados para el alimentador cuando se muestra resumido: nombre (título) del canal, enlace del canal, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
- Datos mostrados para el alimentador cuando se muestra con detalle: nombre (título) del canal, enlace del canal, y lista de vídeos (con información resumida).
- Datos mostrados del ítem (cuando se muestra resumido): título del vídeo, enlace del vídeo
- Datos mostrados del ítem (cuando se muestra con detalle): título del vídeo, enlace del vídeo, descripción del vídeo, vídeo empotrado, nombre del canal, enlace del canal.

- **Alimentador 1**. Reddit (Subreddit). En este caso, el alimentador será un Subreddit (una sección de Reddit, como por ejemplo `r/memes`), y el item una noticia en el Subreddit. Las últmas noticias de un Subreddit están disponibles como documento XML (RSS)[61]. Más información en el wiki de Reddit[62]. Funcionamiento:

  - Alimentador: Subreddit (sección) de Reddit.
  - Ítem: noticia del Subreddit.
  - Elemento HTML para elegir el alimentador: formulario que permita escribir el nombre del Subreddit.
  - Elemento HTML para actualizar el alimentador: botón que actualiza con las noticias disponibles en el canal RSS.
  - Datos mostrados para el alimentador cuando se muestra resumido: nombre (título) del Subreddit, enlace del Subreddit, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).

[61] Ejemplo: Para el Subreddit *memes*, la url es:
https://www.reddit.com/r/memes.rss
[62] What features does reddit have?:
https://www.reddit.com/wiki/rss

- Datos mostrados para el alimentador cuando se muestra con detalle: nombre (título) del Subreddit, enlace del Subreddit, y lista de noticias (con información resumida).
- Datos mostrados del ítem (cuando se muestra resumido): título de la noticia, enlace de la noticia.
- Datos mostrados del ítem (cuando se muestra con detalle): título de la noticia, enlace de la noticia, descripción de la noticia, nombre del Subreddit, enlace del Subreddit.

- **Alimentador 2**. Last.fm (artista). En este caso, el alimentador será un artista de Last.fm (como por ejemplo `Cher`), y el item un álbum. Los álbumes de un artista están están disponibles como documento XML o JSON[63]. Más información en la documentación "Last.fm Web Services[64]. **Atención:** Para el uso de este servicio hace falta conseguir una "clave de API"[65], que tendrás que poner en las llamadas en lugar de `YOUR_API_KEY`. Consulta la lista de preguntas frecuentes (apartado 26.8), hay una que trata justamente sobre este tema (página 320).

  Funcionamiento:

  - Alimentador: Artista en Last.fm.
  - Ítem: álbum de un artista en Last.fm.
  - Elemento HTML para elegir el alimentador: formulario que permita escribir el nombre del artista.
  - Elemento HTML para actualizar el alimentador: botón que actualiza con los álbumes disponibles para el artista.
  - Datos mostrados para el alimentador cuando se muestra resumido: nombre del artista, enlace al artista, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
  - Datos mostrados para el alimentador cuando se muestra con detalle: nombre del artista, enlace del artista, y lista de álbumes (con información resumida).

[63] Ejemplo: Para *Cher*, la url es:
http://ws.audioscrobbler.com/2.0/?method=artist.gettopalbums&artist=cher&api_key=YOUR_API_KEY

[64] Last.fm Web Services, artist.getTopAlbums:
https://www.last.fm/api/show/artist.getTopAlbums

[65] Para conseguir la clave de API en Last.fm (mira también en las preguntas frecuentes):
https://www.last.fm/api/account/create

- Datos mostrados del ítem (cuando se muestra resumido): título del álbum, enlace del álbum.
- Datos mostrados del ítem (cuando se muestra con detalle): título del álbum, enlace del álbum, portada del álbum, nombre del artista, enlace del artista.

- **Alimentador 3**. Flickr (etiqueta). En este caso, el alimentador será una eitqueta (tag) de Flickr (como por ejemplo "Fuenlabrada"), y el item una foto con esa etiqueta. Las fotos que tienen una etiqueta están disponibles como documento XML[66]. Más información en la página Public Feed de Flickr[67]. Funcionamiento:

  - Alimentador: Etiqueta de Flickr.
  - Ítem: foto de Flickr.
  - Elemento HTML para elegir el alimentador: formulario que permita escribir la etiqueta.
  - Elemento HTML para actualizar el alimentador: botón que actualiza con las fotos de la etiqueta.
  - Datos mostrados para el alimentador cuando se muestra resumido: etiqueta, enlace a las fotos con esa etiqueta en Flickr[68], total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
  - Datos mostrados para el alimentador cuando se muestra con detalle: etiqueta, enlace a la página de la etiqueta en Flickr, y lista de fotos para esa etiqueta (con información resumida).
  - Datos mostrados del ítem (cuando se muestra resumido): título de la foto, enlace a la página de la foto en Flickr.
  - Datos mostrados del ítem (cuando se muestra con detalle): título de la foto, enlace a la página de la foto en Flickr, foto, etiqueta, enlace a la página de la etiqueta en Flickr.

- **Alimentador 4**. Wikipedia (historia de artículos). En este caso, el alimentador será la historia de un artículo de Wikipedia (como por ejemplo "Madrid"), y el item la descripción de un cambio en esa historia. La historia de

[66] Ejemplo: Para la etiqueta "Fuenlabrada" la url es:
https://www.flickr.com/services/feeds/photos_public.gne?tags=fuenlabrada
[67] Public Feed de Flickr:
https://www.flickr.com/services/feeds/docs/photos_public/
[68] Por ejemplo, para la etiqueta "Fuenlabrada", el enlace sería:
https://www.flickr.com/search/?tags=fuenlabrada

un artículo está disponible como documento XML[69]. Más información en la página Wikipedia Syndication[70] (sección "RSS Feeds"). Funcionamiento:

- Alimentador: historia de un artículo de Wiipedia.
- Ítem: cambio en la historia de un artículo.
- Elemento HTML para elegir el alimentador: formulario que permita escribir el nombre del artículo.
- Elemento HTML para actualizar el alimentador: botón que actualiza con la historia de un artículo.
- Datos mostrados para el alimentador cuando se muestra resumido: nombre del artículo, enlace al articulo en la Wikipedia, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
- Datos mostrados para el alimentador cuando se muestra con detalle: nombre del artículo, enlace al articulo en la Wikipedia, y lista de cambios para ese articulo (con información resumida).
- Datos mostrados del ítem (cuando se muestra resumido): título del cambio, enlace al cambio.
- Datos mostrados del ítem (cuando se muestra con detalle): título del cambio, enlace al cambio, autor del cambio, fecha del cambio, nombre del artículo, enlace al articulo en la Wikipedia.

Otros alimentadores, entre ellos algunos sugeridos por alumnos de la asignatura, por si te interesa implementarlos:

- TodoLiteratura (sección). En este caso, el alimentador será una sección de TodoLiteratura (como por ejemplo "Actualidad"), y el item un artículo de la sección. Las noticias de una sección están disponibles como documento XML[71]. Más información en la página de canales RSS de TodoLiteratura[72]. Funcionamiento:

  - Alimentador: Sección en TodoLiteratura.

[69] Ejemplo: Para la página "Fuenlabrada" la url es:
https://en.wikipedia.org/w/index.php?title=Fuenlabrada&action=history&feed=rss

[70] Wikipedia Syndication:
https://en.wikipedia.org/wiki/Wikipedia:Syndication

[71] Ejemplo: Para la sección "Actualidad" se usa el número "127", y la url es:
https://www.todoliteratura.es/rss/seccion/127/

[72] Página de canales RSS de Todo Literatura:
https://www.todoliteratura.es/rss/

- Ítem: noticia en una sección de TodoLiteratura.
- Elemento HTML para elegir el alimentador: formulario que permita escribir el identificador de la seccion (número de la sección). Alternativamente, se puede usar un menú que de cómo opción varias de las secciones.
- Elemento HTML para actualizar el alimentador: botón que actualiza con las noticias disponibles en una sección.
- Datos mostrados para el alimentador cuando se muestra resumido: título de la sección, enlace a la sección, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
- Datos mostrados para el alimentador cuando se muestra con detalle: título de la noticia, enlace a la sección, descripción de la sección, y lista de noticias (con información resumida).
- Datos mostrados del ítem (cuando se muestra resumido): título de la noticia, enlace a la noticia.
- Datos mostrados del ítem (cuando se muestra con detalle): título de la noticia, enlace a la noticia, descripción de la noticia, título de la sección, enlace a la sección.

- TuCanaldeSalud (sección). En este caso, el alimentador será una sección de TuCanaldeSalud (como por ejemplo "Tecnología"), y el item un artículo de la sección. Las noticias de una sección están disponibles como documento XML[73]. Más información en la página de canales RSS de TuCanaldeSalud[74]. Funcionamiento:

  - Alimentador: Sección en TuCanaldeSalud.
  - Ítem: noticia en una sección de TuCanaldeSalud.
  - Elemento HTML para elegir el alimentador: formulario que permita escribir el identificador de la seccion (número de la sección). Alternativamente, se puede usar un menú que de cómo opción varias de las secciones.
  - Elemento HTML para actualizar el alimentador: botón que actualiza con las noticias disponibles en una sección.

[73]Ejemplo: Para la sección "Tecnología" se usa el número "70038", y la url es: https://www.tucanaldesalud.es/idcsalud-client/cm/tucanaldesalud/rss?locale=es_ES&rssContent=70038

[74]Página de canales RSS de TuCanaldeSalud: https://www.tucanaldesalud.es/es/feed-rss

- Datos mostrados para el alimentador cuando se muestra resumido: título de la sección, enlace a la sección, total de items disponibles para este alimentador, puntuación (total de votos positivos menos votos negativos para todos sus items).
- Datos mostrados para el alimentador cuando se muestra con detalle: título de la noticia, enlace a la sección, descripción de la sección, y lista de noticias (con información resumida).
- Datos mostrados del ítem (cuando se muestra resumido): título de la noticia, enlace a la noticia.
- Datos mostrados del ítem (cuando se muestra con detalle): título de la noticia, enlace a la noticia, descripción de la noticia.

Además de las anteriores, puedes proponer otros alimentadores. Los requisitos fundamentales son que sean accesibles públicamente (el acceso mediante un token de aplicación se considera público), y que proporcionen datos en formato XML o JSON. Si hay algún alimentador que querrías utilizar, coméntalo con los profesores para que te indiquen si es un alimentador válido. En caso de ser aceptado como válido, estos alimentadores serán puntuados positivamente, teniendo en cuenta la iniciativa del alumno que los propuso.

Si quieres buscar servicios que ofrezcan APIs que podrían ser alimentadores, puedes buscarlos en Internet. Una lista por la que puedes comenzar es la que mantiene ProgrammableWeb[75].

26.3. Funcionalidad mínima

La aplicación servirá las siguientes páginas:

- Página principal de la aplicación:

    1. Listado con los 10 items (formato resumido) que han conseguido más puntuación (votos positivos menos votos negativos) en el sitio. Para cada uno se mostrarán sus votos positivos y negativos, y un enlace a la página del item (ver a continuación).
    2. Formulario para elegir alimentador, para cada uno de los sistemas de alimentación (por ejemplo, canales de YouTube) disponibles. Tras elegir un alimentador vía el formulario, se recibirá la página del alimentador elegido (ver a continuación), con información actualizada, y se almacenarán sus datos en la base de datos (todos los recibidos, si es la primera

[75] Programmable Web API Directory:
https://www.programmableweb.com/apis/directory

vez que se le ha elegido, o lo que no estuvieran ya en la base de datos, si ya se hubiera elegido anteriormente).

3. Listado de alimentadores elegidos en el pasado (formato resumido), por cualquier usuario. Cada alimentador aparecerá con un botón para poder elegirlo (si se elige de esta forma, la aplicación se comportará igual que si se hubiera elegido vía el formulario), y otro para eliminarlo (si se pulsa, el alimentador dejará de salir en este listado en el futuro). Cualquier visitante o usuario podrá eliminar un canal de este listado, pero eso no supondrá que sus datos desaparezcan de la base de datos, y en cualquier caso el alimentador seguirá saliendo en la página de alimentadores.

Si el visitante está además autenticado como usuario, se mostrará también:

- Listado con los últimos 5 items (formato resumido) votados por el usuario (tanto positiva como negativamente).
- Para cada item que aparezca en la página se mostrarán también dos botones para votar (positivo, negativo), resaltando de alguna forma que el valor que se haya votado, si se hubiera votado ya ese item, y un enlace a la página del item (ver a continuación).

- Página del ítem (para cada ítem):

  - Datos del ítem (formato detallado).
  - Datos del alimentador al que pertenece el ítem (formato resumido), incluyendo un enlace a la página del alimentador.
  - Comentarios que haya recibido el ítem. Para cada comentario se mostrará el texto del comentario, el identificador de quien lo puso, y la fecha en que se puso.

  Si el visitante está además autenticado como usuario, se mostrará también:

  - Dos botones para votar (positivo, negativo), resaltando de alguna forma que el valor que se haya votado, si se hubiera votado ya ese item.
  - Formulario para poner un comentario. Tras poner el comentario, se volverá a ver la misma página del ítem.

- Página de alimentadores:

  - Listado de todos los alimentadores de los que se ha podido descargar datos alguna vez (formato resumido). Esto es, todos los que se han "seleccionado" alguna vez, por cualquier visitante, aunque no salgan en la página principal.

- Página de alimentador (para cada alimentador):

  - Datos del alimentador (formato detallado)
  - Botón para poder elegir o dejar de tener elegido el alimentador. Si se pulsa, y no estaba elegido, el alimentador pasará a estar elegido, con los mismos efectos que si se hubiera elegido en el formulario de la página principal. Si se pulsa, y estaba elegido, pasa a dejar de estar elegido, con el mismo efecto que se hubiera pulsado el botón de "eliminar" del listado de alimentadores elegidos de la pagina principal. El botón tendrá que indicar de alguna manera (por ejemplo, con dos textos distintos, o con colores distintos) si el alimentador está o no elegido, antes de pulsarlo. En ningún caso si un alimentador pasa a dejar de estar elegido, se eliminarán sus datos de la base de datos: sólo dejará de salir en el listado de elegidos.
  - Lista de items del alimentador (formato resumido).
  - Para cada ítem, enlace a la página del ítem.

- Página de usuario (para cada usuario "con cuenta"). Página "pública", que verá cualquiera que cargue el recurso apropiado:

  - Datos públicos del usuario (identificador, foto)
  - Lista de items votados por ese usuario (formato resumido)
  - Lista de items comentados por ese usuario (formato resumido)

  Si el visitante está autenticado, cuando acceda a su propia página, se mostrará también:

  - Formulario para cambiar la foto
  - Formulario para cambiar de estilo. Se ofrecerán al menos dos estilos: "ligero" y "oscuro".
  - Formulario para cambiar el tamaño de la letra. Se ofrecerán al menos tres tamaños: "pequeña", "normal" y "grande".

- Página de usuarios:

  - Listado de todos los usuarios "con cuenta". Para cada usuario, aparecerá su identificador, su foto, el número de items votados, el número de comentarios que ha hecho, y un enlace a su página de usuario.

La página principal se ofrecerá también como un documento XML y como un documento JSON, que incluiría la misma información (los mismos listados de items y alimentadores). Este documento se ofrecerá cuando se pida la página principal, concatenando al final `?format=xml` o `?format=json`.

La página principal en formato HTML incluirá un enlace a la página principal en formato XML ("Descarga como fichero XML") y JSON ("Descarga como fichero JSON").

- Página de información: Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento y una brevísima documentación.

La aplicación se encargará de controlar que no haya más de un voto (positivo o negativo) por usuario para cada ítem. Por lo tanto, si un usuario ya ha votado un ítem, y vuele a votarlo, se ignorará su voto (si es igual que el que está almacenado) o se anotará el nuevo (si es distinto). Por ejemplo, si había votado un ítem con positivo, y ahora vuelve a votarlo con positivo, se ignorará el segundo voto. Si vuelve a votarlo, pero ahora con negativo, se cambiará el voto a negativo.

En todos los casos en que se vote, tras votar se volverá a ver la misma página en que se estaba, ahora con el voto contabilizado.

Todas las páginas un menú desde el que se podrá acceder a la página principal (con el texto "Inicio"), a la de alimentadores (con el texto "Alimentadores"), a la de usuarios (con el texto "Usuarios") y a la de información (con el texto "Información"), salvo que ya estés en esa página, en cuyo caso no saldrá el elemento de menú correspondiente.

Además, la práctica incluirá tests, que se ejecutarán con `python3 manage.py test`, y que incluirán al menos un test de API HTTP para cada recurso que sirva la aplicación, y para cada método (GET, POST) que admita cada recurso. Además, al menos la mitad de los test incluirán comprobar algo distinto del código HTTP retornado por la petición.

26.4. Despliegue

La práctica deberá estar desplegada en algún sitio de Internet, de forma que pueda accederse a ella. Deberá mantenerse desplegada y activa al menos desde el día de entrega de la práctica, hasta el día del cierre de actas.

Para el despliegue, se puede utilizar Python Anywhere[76], que proporciona un plan gratuito que incluye suficientes recursos como para poder desplegar la práctica.

[76] Python Anywhere: https://pythonanywhere.com

Si el alumno así lo desea, puede considerarse desplegar en un ordenador dedicado (por ejemplo, una Raspberry Pi accesible directamente desde Internet, alojada en su hogar), o en servicios como Google Computing Engine[77]. En general, dado que este tipo de despliegues no podrá contar con una ayuda detallada por los profesores, estará algo más valorado.

En el caso de que la práctica se despliegue en Python Anywhere, hay que tener en cuenta que sus máquinas virtuales tienen cortado el acceso a todos los sitios de Internet salvo los que están en una "lista blanca"[78] (*whitelist*). Esto afectará a vuestro despliegue de dos formas:

- Al clonar vuestro repositorio git dentro de la máquina virtual, para tener el código de vuestra aplicación. No debería dar problemas, porque el sitio GitLab de la ETSIT, donde está vuestro código fuente, está ya en la lista blanca.
- Cuando vuestra aplicación se conecte para actualizar un alimentador. Si el sitio al que la aplicación se tiene que conectar para conseguir el documento JSON o XML no está en la lista blanca, vuestra aplicación no se podrá conectar. YouTube (que está en la parte obligatoria para todo el mundo) y algunos otros sitios de alimentadores (como Flickr) están ya en la lista blanca. Pero otros sitios que podéis estar usando, no.

Para evitar los problemas con los sitios que no estén en la lista blanca, os pedimos que si hacéis el despliegue en YouTube:

- La actualización ("selección") de alimentadores tiene que funcionar al menos con YouTube, recogiendo los documentos XML correspondientes, como indica el enunciado.
- Para los demás alimentadores que hayáis implementado, tenéis dos opciones:
  - Si están en la lista blanca de Python Anywhere, funcionarán sin problemas sin hacer nada especial, si os funcionaban ya en las pruebas locales, así que también deberían funcionar en el despliegue.
  - Si no están en la lista blanca de Python Anywhere, aseguraos de que la base de datos que subís al despliegue de vuestra práctica incluya datos de alimentadores de la plataforma en cuestión. Por ejemplo, si tenéis implementados alimentadores de Last.fm, basta con que tengáis en la base de datos items de un par de artistas, que muestren que en la versión local os funcionó.

[77] GCP Engine Free: https://cloud.google.com/free/
[78] Lista blanca de Python Anywhere: https://www.pythonanywhere.com/whitelist/

Tened en cuenta que si usáis otras plataformas para el despliegue, puede que os encontréis problemas similares. Y tened en cuenta también que en cualquier caso, nosotros probaremos la práctica en otros despliegues, así que todos los alimentadores que hayáis implementado deben funcionar correctamente si no hay restricciones de conexión.

26.5. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habéis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio
- Visualización de cualquier página en formato JSON y/o XML, de forma similar a como se ha indicado para la página principal.
- Generación de un canal RSS, XML libre y/o JSON para los comentarios puestos en el sitio.
- Incorporación de datos de otros alimentadores además de los obligatorios. Se valorará especialmente la búsqueda de otros alimentadores no descritos en este enunciado, la implementación de alimentadores no basados en RSS (o derivados), y la implementación de alimentadores que requieran de token de autenticación (en este caso, atención a no subir el token de autenticación a GitLab).
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario de la aplicación tendrá en cuenta lo que especifique el navegador.
- Utilización de Bootstrap[79] para la maquetación del sitio web.
- Inclusión de imágenes (no solo texto) en los comentarios. Esto puede hacerse de dos formas: quien suba un comentario, además de rellenar una caja de texto con el comentario, puede indicar también la url de una imagen, que se mostrará junto al comentario, o bien subiendo una imgen a la aplicación,

[79]Bootstrap: https://getbootstrap.com/

que se mostrará junto al comentario (se valorará más la segunda opción, y se pueden implementar las dos).

- Mejora de los tests de la práctica, incluyendo test de condiciones de error, test de escenarios con más de una invocación de recurso, tests de API Python, etc.

26.6. Entrega de la práctica

- **Fecha límite de entrega de la prueba teórica:** viernes, 12 de junio de 2020 a las 15:00 (hora española peninsular)
- **Fecha límite de entrega de la práctica:** domingo, 14 de junio de 2020 a las 23:59 (hora española peninsular)
- **Notificación de alumnos que tendrán que realizar entrevista:** martes, 16 de junio, en el aula virtual.
- **Realización de entrevistas:** miércoles, 17 de junio, en la aplicación Teams. Si es necesario, se realizarán también los días 18 y 19.
- **Fecha de publicación de notas:** viernes, 19 de junio, en el aula virtual.
- **Fecha de revisión:** lunes, 22 de junio, a las 12:00, en la aplicación Teams.

La entrega de la práctica consiste en:

1. **Rellenar un formulario** enlazado en el sitio de la asignatura en el aula virtual.
2. **Subir tu práctica a un repositorio en el GitLab de la Escuela**. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado una derivación (fork) del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   `https://gitlab.etsit.urjc.es/cursosweb/practicas/server/final-miscosas/`

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitLab.

   Se recomeinda mantener el repositorio como privado, hasta el momento en que se entregue la práctica.

3. **Entregar un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional**, si se han realizado opciones avanzadas. Los vídeos 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). 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, Twitch, o YouTube). Los vídeos de más de tres minutos tendrán penalización.

   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). OBS Studio[80] está disponible para varias plataformas (Linux, Windows, MacOS). 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.

Sobre la entrega del repositorio:

- Se han de entregar los siguientes ficheros:

  - El repositorio en la instancia GitLab de la ETSIT deberá contener un proyecto Django completo y listo para funcionar en el entorno del laboratorio, incluyendo la base de datos. Deberá poder ejecutarse directamente con `python3 manage.py runserver` desde un entorno virtual en el que esté instalado Django 3.0.3. Tmbién ejecutará los tests con `python3 manage.py test`, desde el mismo entorno virtual.
  - La base de datos habrá de tener datos suficientes como para poder probarlo. Estos datos incluirán al menos dos usuarios con sus datos correspondientes, con al menos cinco alimentadores elegidos en total, cinco comentarios puestos en total, y al menos 10 items votados por cada usuario.
  - Un fichero `requirements.txt`, con un nombre de paquete Python por línea, para indicar Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, si es que fuera el caso. Este fichero

[80] OBS Studio: https://obsproject.com/

no ha de incluir Django, dado que ya se supone que hace falta. Si es posible, se recomienda escribir este fichero en el formato que entiende `pip install -r requirements.txt`

- Cualquier fichero auxiliar que pueda hacer falta para que funcione la práctica, si es que fuera el caso.

- Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica: se recomienda hacer un copia y pega de estos datos en el formulario):

  - Nombre y titulación.
  - Nombre de su cuenta en el laboratorio del alumno.
  - URL del vídeo demostración de la funcionalidad básica
  - URL del vídeo demostración de la funcionalidad optativa, si se ha realizado funcionalidad optativa
  - URL de la aplicación desplegada
  - Cuenta (login) y contraseña de los usuarios que están dados de alta en 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.

  Estos datos se escribirán siguiendo estrictamente el siguiente formato:

```
# Entrega practica

## Datos

* Nombre:
* Titulación:
* Despliegue (url):
* Video básico (url):
* Video parte opcional (url):
* Despliegue (url):
*
```

```
## Cuenta Admin Site

* usuario/contraseña

## Cuentas usuarios

* usuario/contraseña
* usuario/contraseña
* ...

## Resumen parte obligatoria

## Lista partes opcionales

* Nombre parte:
* Nombre parte:
* ...
```

Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitLab los incluya como enlaces "pinchables".

26.7. 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.

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

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. Los documentos XML deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el del módulo xml.sax de Python. Los documentos JSON generados deberán ser correctos desde el punto de vista de la sintaxis JSON, y por lo tanto reconocibles por un reconocedor JSON, como por ejemplo el del módulo json de Python

26.8. Preguntas frecuentes

A continuación, algunas preguntas relacionadas con el enunciado de esta práctica, junto con sus respuestas:

- Cuando despliego mi práctica en Python Anywhere, algunos alimentadores no me funcionan, pero otros (YouTube entre ellos), sí. Todo me funciona bien en mi versión local. ¿Qué está pasando?

  Las máquinas virtuales de Python Anywhere están limitadas en cuanto a los sitios a los que se pueden conectar: sólo se pueden conectar a aquellos que están en una cierta "lista blanca". Por eso, si el sitio al que tu programa se tiene que conectar para obtener items de un alimentador no está en la lista blanca, no va a poder descargarse el documento XML o JSON con esos items. Para evitar problemas, en el caso de despliegue en Python Anywhere pedimos que funcionen bien los alimentadores que están en la lista blanca, y para los demás, que tengan items en la base de datos de despliegue. Más detalles en el apartado sobre despliegue de este enunciado (26.4).

- En la pagina de información que se menciona en el enunciado, ¿qué hay que incluir en el apartado de documentación?

  Casi que lo que queráis, lo importante es tener la página. Puede ser por ejemplo un resumen de un párrafo de lo que hace la aplicación.

- ¿Qué es la "API key" en los alimentadores de Last.fm, y en otros alimentadores?

  Algunas API de servicio, entre ellas la de Last.fm, requieren el uso de una "API key" (clave de API) para poder usarla. Normalmente, estas claves las usa el servicio para evitar abusos, o para limitar lo que se puede hacer con su API. El caso es que si no se incluye la clave de API en cada GET que se hace al servicio, no se reciben los datos (el documento XML o JSON).

  Es habitual que estas claves se obtengan creándose una cuenta en el servicio en cuestión, y luego obteniendo la clave en una página al efecto, estando autenticados con el servicio.

  Por ejemplo, en el caso de Last.fm, hay que ir a la página de petición de claves de API[81], donde (una vez autenticados con una cuenta de Last.fm), rellenaremos los dato que nos pide: "contact email", "aplication name" (cualquier nombre de aplicación, por ejemplo MisCosas), "appication description" (cualquier descripción por ejemplo "App to manage Last.fm artists"). En este caso, puedes ignorar los campos "callback url" y "application homepage". Cuando se hayan enviado estos datos, te devolverá entre otros datos tu "API key". Esa es la que tendrá que usar en tus llamadas a Last.fm.

  Como las claves de API son personales, mantenlas en secreto. En particular, no las subas a repositorios públicos, pues cualquiera podrá verlas (y usarlas).

[81] Lastfm Create API Account: https://www.last.fm/api/account/create

Si quieres que el repositorio de tu práctica sea público, incluye la clave en un fichero que tengas sólo en tu disco, y no subas al repositorio git. Por ejemplo, puedes poner la clave en un fichero `apikeys.py` del estilo de este:

```
LASTFM_APIKEY = "012345678"
```

Luego, en el módulo Python que la uses (por ejemplo `views.py`), pondrás algo como:

```
from .apikeys import LASTFM_APIKEY
```

Y ya puedes usar la clave en tu código. Este fichero `apikeys.py` no lo subirás al repositorio git público.

Si usas claves de API en tu práctica, indícalo claramente en el fichero de entrega de la práctica, y o bien sube una clave de API válida (si el repositorio de entrega es privado) para que la podamos probar, o bien indica en qué fichero hay que ponerla, y cómo se consigue una clave API válida para el servicio que estés usando, de forma que la podamos conseguir y ejecutar tu práctica.

- ¿Qué tiene que haber en la página de usuarios?

  En el enunciado tenemos una página de usuarios, con un listado de todos los usuarios (visitantes con cuenta). En la primera versión de este enunciado, se incluía, para cada usuario en este listado, un "número de alimentadores que ha elegido". Pero al simplificar el enunciado para que la lista de seleccionados sea común a todo el sitio, y no particular a un visitante, esto ya no tiene sentido. Así que lo hemos sustituido por un "número de comentarios que ha hecho" (el usuario). Por lo tanto, el enunciado actual indica que esta página mostrará:

  > *Listado de todos los usuarios "con cuenta". Para cada usuario, aparecerá su identificador, su foto, el número de items votados, el número de comentarios que ha hecho, y un enlace a su página de usuario.*

- ¿Qué tiene que haber en la página de alimentadores?

  En esta página tiene que esta la lista de alimentadores que se han elegido alguna vez, aunque luego se hayan "deseleccionado". Esto es, será la lista de todos los alimentadores que tenemos en la base de datos.

- ¿Qué alimentadores tienen que tener una página de alimentador?

  Todos los alimentadores que se hayan elegido alguna vez tienen que tener pagina de alimentador, aunque luego se hayan "deseleccionado". Esto es, todos los alimentadores que tenemos en la base de datos tienen que tener página de alimentador.

- ¿Cómo se comporta el botón que tengo en la página de un alimentador?

  En la página de alimentadot hay un botón "para poder elegir o dejar de tener elegido el alimentador". Este botón se muestra siempre, esté el visitante autenticado o no. Si el alimentador no ha sido "deselegido" nunca, al pulsar el botón se "deselegirá", con el efecto que dejará de salir en el listado de alimentadores elegidos (en la página principal). Pero no será borrado de la base de datos, seguirá saliendo en el listado de la página de alimentadores, y seguirá teniendo su página de alimentador.

  Si el alimentador fue "deselegido" y no ha vuelto a ser elegido (bien en la página principal, poniéndolo en el formulario de alimentadores, o bien en la págian de alimentadores, eligiéndolo), por cualquier usuario, el botón servirá para volver a alegirlo. Como esto tiene los misms efectos que si se hubiera puesto su nombre en el formulario de alimentadores de la página principal, si se pulsa el botón el alimentador quedará elegido, sus datos se actualizarán descargándolos del sistema de alimentadores correspondiente (YouTube, por ejemplo), y se recibirá la página de ese alimentador, con información actualizada.

- ¿Tiene que haber una lista de alimentadores en la página de usuario?

  En una primera versión del enunciado teníamos una lista de alimentadores en la página de usuario. Pero como esa lista ya está en varias otras páginas, y no tiene mucho sentido en esa (porque serían los alimentadores elegidos por cualquier usuario), en la versión final del enunciado no hay que incluir lista de alimentadores en la página de usuario. Sin embargo, puede ser una buena mejora opcional, si te interesa implementarla. Si lo haces, sería conveniente que sea la lista de alimentadores que ha seleccionado ese usuario en particular.

- ¿Cómo puedo implementar la subida de la foto del usuario?

  Para que el usuario pueda cambiar su foto, tendrás que implementar una vista que recoja la foto que se le envíe desde un formulario, y la almacene. El manual de Django indica cómo hacer eso para ficheros en general[82].

[82] "File uploads" en la documentación de Django:

Y puedes encontrar varias referencias en la red que explican cómo hacerlo específicamente para imágenes[83].

Resumiendo mucho, convendrá que utilices un campo de tipo `ImageField`[84] en la base de datos, especificando a qué directorio se subirán las imágenes (parámetro `upload_to`). Para subir los ficheros, puedes utilizar un Django Form (heredando de `forms.Form`) en el que especifiques un campo `forms.ImageField()`, y una plantilla de formulario (elemento HTML `form`) en la que especifiques que el método es POST y además un `enctype` (tipo de codificación) `multipart/form-data`. Cuando instancies el formulario Django, además del parámetro habitual (`request.POST`), le pasarás también `request.FILES`, que es donde vendrá la imagen (dado que usas `multipart/form-data`). Después de validar la respuesta que te ha venido del formulario, extraes de ella el campo donde venga la imagen, y lo guardas en la base de datos. Al hacerlo, se guardará un descriptor en la base de datos, y la imagen se guardará (automáticamente) en el directorio que hayas especificado.

Trata de entender todo el proceso antes de ponerte a implementarlo, y cuidado con las recetas que podrás encontrar en Internet, porque puede que no te sirvan exactamente "tal cual".

- ¿Qué quiere decir "vídeo empotrado", que se menciona en la información detallada para un vídeo de un canal de YouTube? ¿Cómo se hace?

  "Vídeo empotrado" es en ese caso la traducción de "embedded video", y quiere decir que el video aparezca directamente en la página, normalmente en un elemento `iframe`. En el caso de YouTube, puedes obtener el código HTML para empotrar cualquier video si, cuando lo estás viendo en el navegador, pulsas el botón de compartir ("Share"), y eliges la opción "Embed". Esto muestra un código HTML como el siguiente:

```
<iframe width="560" height="315"
  src="https://www.youtube.com/embed/HZOodD84MR8"
  frameborder="0"
  allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-pictu
  allowfullscreen></iframe>
```

https://docs.djangoproject.com/en/3.0/topics/http/file-uploads/
Mira especialmente el apartado "Handling uploaded files with a model", pero en general será conveniente leer toda la página.

[83] Ejemplo de referencia que indica cómo subir imágenes a Django:
https://coderwall.com/p/bz0sng/simple-django-image-upload-to-model-imagefield

[84] Campo "ImageField" en la documentación de Django:
https://docs.djangoproject.com/en/3.0/ref/models/fields/#imagefield

Basta con que en él cambies el código del video (en este caso "HZOodD84MR8" por el del video que quieres empotrar, y ya está.

- En la página de cada alimentador, ¿hay que mostrar los datos en formato resumido o detallado?

  Había una errata en el enunciado que no dejaba claro este extremo. Ahora debería quedar claro: en la página de cada alimentador habrá un listado de los items de ese alimentador, en formato resumido, incluyendo también para cada item un enlace a la página del item.

- En el formulario para elegir alimentador, ¿qué hay que introducir? (el nombre de los alimentadores que ya hay, dar opciones de los que ya hay, directamente el identificador de alimentador...).

  Habrá un formulario por cada tipo de alimentador. Por ejemplo, uno para canales de YouTube, otro para etiquetas de Flickr, etc. Cada uno de ellos será un formulario en el que se podrá poner lo que haga falta para elegir un alimentador para ese tipo de alimentador. Por ejemplo, en el caso de YouTube, el formulario tendrá una caja de texto para poder poner el id del canal, y un botón para enviarlo. Lo normal, es que todos los formularios para los tipos de alimentadores que hayas implementado estén juntos.

  Alternativamente, y esto es opcional, se puede tener un único formulario para todos los tipos de alimentador. En ese caso, tendrás que tener algo parecido a un menú desplegable para que puedas elegir qué tipo de alimentador vas a indicar, algún elemento para poner el identificador (id, etiqueta, nombre de sección... lo que sea), y un botón para enviar.

  Para algunos alimentadores (por ejemplo, el de TuCanaldeSalud), puede tener sentido que el formulario incluya un menú par elegir el alimentador, pero este no es el caso de ninguno de los alimentadores obligatorios. Puede ocurrir esto cuando el número de alimentadores a elegir sea pequeño.

- ¿Es necesario utilizar los mecanismos provistos por Django para el control de sesiones y autenticación?

  En principio, esa es la solución recomendada. El principal problema suele ser asegurarse de que cualquier mecanismo alternativo funciona al menos tan bien como el de Django, lo que no es en general trivial. De todas formas, salvo muy buenos motivos, la aplicación es una aplicación Django, y por lo tanto cuantas más facilidades de Django se usen (bien usadas), mejor.

- Los archivos CSS que pueden modificar los usuarios, ¿dónde y cómo debemos guardarlos?

La forma recomendada de hacerlo es mediante plantillas:

- En el directorio de plantillas incluirías una para la hoja CSS del sitio. Esa plantilla tendría como variables de plantilla los valores que quieras que los usuarios puedan cambiar (color de tipo de letra, tamaño de tipo de letra, etc.).
- Además, para cada usuario, tendrás una tabla en la base de datos donde se almacenarán los valores para ese usuario (normalmente, una fila de la tabla por usuario).
- Tendrás una vista en views.py que se encargará de generar la hoja CSS a partir de la plantilla. Esa vista es la que comprobará si la petición que está atendiendo corresponde a un usuario (en cuyo caso tendrá que obtener los valores para ese usuario de la tabla anterior), o no (en cuyo caso usará valores por defecto). Con los valores que obtenga, generará la hoja CSS a partir de la plantilla anterior.
- Por último, en urls.py tendrás una línea para indicar que si te piden el recurso que sirve la hoja de estilo, llamas a la vista anterior.

- ¿Dónde puedo realizar el despliegue de la aplicación?

  El despliegue puede realizarse en cualquier ordenador que esté conectado permanentemente a Internet durante el periodo de corrección, en una dirección accesible desde cualquier navegador conectado a su vez a Internet. Esto puede ser por ejemplo un ordenador personal en un domicilio con acceso permanente a Internet, adecuadamente configurado (puede ser una Raspberry Pi o similar, si se busca una solución simple y de bajo coste). También puede ser un servicio en Internet, por ejemplo uno gratuito como los que ofrecen Google (instrucciones[85], precios[86]), o PythonAnywhere (instrucciones[87], precios[88]). Los profesores podremos ayudar de forma más detallada con PythonAnywhere.

[85] GCP Quickstart Using a Linux VM:
https://cloud.google.com/compute/docs/quickstart-linux
[86] Google Compute Engine Pricing:
https://cloud.google.com/compute/pricing
[87] Capítulo "Deploy!" de Django Girls Tutorial:
https://tutorial.djangogirls.org/en/deploy/
[88] PythonAnywhere Plans and Pricing:
https://www.pythonanywhere.com/pricing/

27. Proyecto final: MiTiempo (2019, mayo)

El proyecto final de la asignatura consiste en la creación de una aplicación web, llamada "MiTiempo", que aglutine información sobre municipios de España, y especialmente información meteorológica sobre ellos. A continuación se describe el funcionamiento y la arquitectura general de la aplicación, la funcionalidad mínima que debe proporcionar, y otra funcionalidad optativa que podrá tener.

La aplicación se encargará de descargar información sobre las condiciones meteorológicas de los municipios, disponibles públicamente en en sitio web de la AEMET, y de ofrecerla a los usuarios para que puedan monitorizar con facilidad las previsiones para aquellos municipios que les parezcan más interesantes, y comentar sobre ellos. De esta manera, un escenario típico es el de un usuario que elija los municipios que le parezcan de interés, y comente lo que le quiera sobre ellos.

27.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones (apps) Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de Django.
- Se usará la aplicación Django "Admin Site" para crear cuentas a los usuarios en el sistema, y para la gestión general de las bases de datos necesarias. Todas las bases de datos que contenga la aplicación tendrán que ser accesibles vía este "Admin Site".
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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, preferentemente en la parte superior izquierda.
  - 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). Esta caja aparecerá preferentemente en la parte superior derecha.
  - Un menú de opciones, como barra, preferentemente debajo de los dos elementos anteriores (banner y caja de entrada o salida).
  - Un pie de página con una nota de atribución, indicando "Esta aplicación utiliza datos proporcionados por la AEMET", y un enlace al sitio web de AEMET[89].

  Cada una de estas partes estará construida dentro de un elemento "div", marcada con un atributo "id" en HTML, para poder ser referenciadas fácilmente en hojas de estilo CSS.

- Se utilizarán hojas de estilo CSS para determinar la apariencia de la práctica. 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 un *id*, tal como se indica en el apartado anterior.

- Para obtener la información sobre previsión meteorológica de cada municipio se utilizará la información disponible en AEMET:

  - Ejemplo de información para un municipio, en formato XML (para cada municipio, el número de cinco cifras que finaliza la URL se obtiene del documento descrito más abajo):
    `http://www.aemet.es/xml/municipios/localidad_28058.xml`
  - Documento JSON con listado de municipios, incluyendo su nombre y su identificador para localizar los documentos anteriores (campo "id_old"):
    `https://raw.githubusercontent.com/CursosWeb/Code/master/Python-JSON/municipios.json`

Funcionamiento general:

- Los usuarios serán dados de alta en la práctica mediante el módulo "Admin Site" de Django. Una vez estén dados de alta, serán considerados "usuarios registrados".

[89] Sitio web de AEMET: `https://aemet.es`

- El listado de municipios se cargará de nuevo cada vez que arranque la aplicación, a partir de un fichero que será parte del proyecto Django. El listado se mantendrá en un diccionario en memoria, y no se guardará en almacenamiento persistente en la base de datos.
- Los usuarios registrados podrán crear su selección de municipios. Para ello, dispondrán de una página personal. Llamaremos a esta página la "página del usuario".
- La selección de municipios en su página personal la realizará cada usuario rellenando un formulario que estará en su página de usuario. Este formulario permitirá elegir un nombre de municipio. Si el municipio coincide con uno en el listado de municipios, se considerará válido, y se añadirá a la lista de municipios seleccionados por ese usuario. Si no es así, se le indicará que el nombre del municipio es erróneo.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la página personal de cada usuario, para todos los usuarios del sitio.

27.2. Funcionalidad mínima

La información para cada municipio se obtendrá a partir de la información pública ofrecida por AEMET, en forma de ficheros XML, como se indicaba anteriormente.

La **interfaz pública** contiene los recursos a servir como páginas HTML completas (pensadas para ser vistas en el navegador) para cualquier visitante (sea usuario registrado o no), excepto donde se indica que se servirá una página XML:

- /: Página principal de la práctica. Constará de un listado de poblaciones que han sido elegidas por algún usuario, y otro con enlaces a páginas de usuarios:

    1. Mostrará un listado de los 10 municipios con más comentarios. Si no hubiera 10 municipios con comentarios, se mostrarán sólo los que tengan comentarios. Para cada municipio, incluirá información sobre:
        - su nombre (que será un enlace que apuntará a la URL del municipio en el sitio de AEMET)[90],
        - su altitud, latitud y longitud,
        - su previsión de tiempo para mañana: probabilidad de precipitación (0 a 24), temperatura máxima y mínima, y descripción (0 a 24).

[90] Por ejemplo http://www.aemet.es/es/eltiempo/prediccion/municipios/fuenlabrada-id28058 (donde el identificador "fuenlabrada-id28058" puede encontrarse en el docuento JSON con el listado de municipios como campo "url"

    - y un enlace, “Más información”, que apuntará a la página del municipio en la aplicación (ver más adelante).
2. También se mostrará un listado, en una columna lateral, con enlaces a las páginas personales disponibles. Para cada página personal mostrará el título que le haya dado su usuario (como un enlace a la página personal en cuestión) y el nombre del usuario. 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.
3. Tambień se mostrará un botón, que al pulsarlo se verán sólo los municipios con probabilidad de precipitacion mayor que cero. Si se vuelve a pulsar, se verán los que tengan ptobabilidad de precipitación igual a cero. Si se vuelve a pulsar una vez más, se volverán a ver todos los municipios.

   La página principial se ofrecerá también como un documento XML, que incluirá la misma lista de municipios, y un enlace al fichero XML que proporciona para cada uno de ellos la AEMET. Este documento se ofrecerá cuando se pida la URL de municipios, concatenando al final `?format=xml`.

   La página principal en formato HTML includirá un enlace a la página principal en formato XML (“Descarga como fichero XML”).

- /usuario: Página personal de un usuario. Si la URL es “/usuario”, es que corresponde al usuario “usuario”. Mostrará los municipios seleccionados por ese usuario. Para cada municipio se mostrará la misma información que en la página principal. Los municipios deben aparecer en el orden en que los ha seleccionado el usuario (primero el que fue seleccionado más recientemente).

  La página de cada usuario se ofrecerá también como un documento XML, que incluirá la lista de municipios seleccionados, y un enlace al fichero XML que proporciona para cada uno de ellos la AEMET. Este documento se ofrecerá cuando se pida la URL del usuario, concatenando al final `?format=xml`.

  La página de cada usuario en formato HTML includirá un enlace a la página de ese mismo usuario en formato XML (“Descarga como fichero XML”).

- /municipios: Página con todos los municipos que han sido seleccionados por algún usuario (aunque hayan sido luego “desseleccionados”. Para cada uno de ellos aparecerá sólo el nombre, como un enlace a su página (ver más abajo), y el número de comentarios que se han puesto sobre él. En la parte superior de la página, habrá un formulario que permita filtrar según la temperatura máxima para mañana: se mostrarán solo los municipios que para mañana

tengan previsión de temperatura máxima entre las dos que se indiquen, si se indican.

La página de municipios se ofrecerá también como un documento XML, que incluirá la misma lista de municipios, y un enlace al fichero XML que proporciona para cada uno de ellos la AEMET. Este documento se ofrecerá cuando se pida la URL de municipios, concatenando al final `?format=xml`.

La página de municipios en formato HTML includirá un enlace a la página de municipios en formato XML ("Descarga como fichero XML").

- /municipios/id: Página de un municipio en la aplicación. Mostrará toda la información razonablemente posible del documento XML obtenido de AEMET (en cuanto a predicción para mañana, en el rango 0 a 24 horas), incluyendo también al menos la que se menciona en otros apartados de este enunciado. También se incluirá un enlace a la página de este municipio en el sitio de AEMET. Además, se mostrarán todos los comentarios que se hayan puesto para este municipio. Esta información se actualizará cuando se consulte esta página de un minicipio, y a partir de este momento se mostrará actualizada en cualquier otra página del sitio. La información no se actualizará en ningún otro momento.

- /info: Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento y una brevísima documentación.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todos los municipios (URL /municipios) con el texto "Todos" y a la ayuda (URL /info) con el texto "Info". Todas las página que no sean la principal tendrán otra opción de menú para la URL /, con el texto "Inicio".

La **interfaz privada** contiene los recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado):

- Todos los recursos de la interfaz pública.

- /municipios/id: Además de la información que se muestra de manera pública:

    1. Un formulario para poner comentarios sobre este municipio. Los comentarios quedarán a nombre del usuario que los ponga, y sólo se podrán poner por los usuarios registrados, una vez se han autenticado. Por tanto, bastará con que este formulario esté compuesto por una caja de texto, donde se podrá escribir el comentario, y un botón para enviarlo. El sistama anotará automáticamente quién está poniendo el comentario, y mostrará esa información cada vez que muestre el comentario (con el texto "Comentado por").

- /usuario: Además de la información que se muestra de manera pública:

    1. Un formulario para cambiar el estilo CSS de todo el sitio para ese usuario. Bastará con que se pueda cambiar el tamaño y el color de la letra y el color de fondo. Si se cambian estos valores, quedará cambiado el documento CSS que utilizarán todas las páginas del sitio para este usuario. Este cambio será visible en cuanto se suba la nueva página CSS.
    2. Un formulario para elegir el título de su página personal.
    3. Un formulario para seleccionar un nuevo municipio. En este formulario se podrá poner el nombre de un municipio, que si existe, quedará seleccionado para este usuario.
    4. Un botón "Quitar" que aparecerá asociado a cada municipio seleccionado, que permitirá al usuario "deseleccionar" el municipio de su lista.

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

27.3. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habeis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio
- Visualización de las páginas en formato JSON, de forma similar a como el enunciado indica para XML.
- Generación de un canal RSS, XML libre y/o JSON para los comentarios puestos en el sitio.
- Incorporación de datos del canal RSS de avisos de AEMET[91] a la página principal y/o a otras páginas ofrecidas por la aplicación.

[91] Canales RSS de AEMET: http://www.aemet.es/es/rss_info

- Funcionalidad para acceder a datos ofrecidos por AEMT via su API de datos abiertos[92]
- Funcionalidad de registro de usuarios: que la aplicación proporcione la funcionalidad de registrarse en el sitio.
- Uso de Javascript o AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar un municipio para una página de usuario).
- Puntuación de municipios. Cada visitante (registrado o no) puede dar un "+1" a cualquier municipio que aparezca en el sitio. La suma de "+" que ha obtenido un municipio se verá cada vez que se vea el municipio en el sitio.
- Uso de elementos HTML5 (especificar cuáles al entregar)
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.
- Despliegue de la práctica en algún sitio de Internet, de forma que pueda accederse a ella. Por ejemplo, puede considerarse desplegar en un ordenador dedicado (por ejemplo, Raspberry Pi accesible directamente desde Internet), o en servicios como Google Computing Engine[93] o Heroku[94].

27.4. Entrega de la práctica

- **Fecha límite de entrega de la práctica:** viernes, 24 de mayo de 2019 a las 03:00 (hora española peninsular)[95]
- **Fecha de publicación de notas de prácticas:** sábado 25 de mayo, en el aula virtual.
- **Fecha de revisión de prácticas:** martes 28 de mayo, 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 entrega de la práctica consiste en **rellenar un formulario** (enlazado en el Moodle de la asignatura) y en seguir las instrucciones que se describen a continuación.

[92] AEMET open data: https://opendata.aemet.es
[93] GCP Engine Free: https://cloud.google.com/free/
[94] Heroku Free: https://www.heroku.com/free
[95] Entiéndase la hora como jueves por la noche, ya entrado en viernes.

1. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado una derivación (fork) del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://gitlab.etsit.urjc.es/cursosweb/practicas/server/final-mitiempo/

   Los alumnos que no entreguen las 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 de la revisión. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitLab.

2. Un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional, si se han realizado opciones avanzadas. Los vídeos 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). 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, Twitch, o YouTube). Los vídeos de más de tres minutos tendrán penalización.

   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). OBS Studio[96] está disponible para varias plataformas (Linux, Windows, MacOS). 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.

3. Se han de entregar los siguientes ficheros:

[96] OBS Studio: https://obsproject.com/

- Un fichero README.md que resuma las mejoras, si las hay, y explique cualquier peculiaridad de la entrega (ver siguiente punto).
- El repositorio en el GitLab de la ETSIT deberá contener 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, con al menos seis municipios en su página personal, al menos 12 municipios distintos seleccionados, y con al menos cinco comentarios en total.
- Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, junto con los ficheros auxiliares que utilice, si es que los utiliza.

4. Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica - se recomienda hacer un corta y pega de estos datos en el formulario):

   - Nombre y titulación.
   - Nombre de su cuenta en el laboratorio del alumno.
   - 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
   - Cuenta (login) y contraseña de los usuarios que están dados de alta en la aplicación.
   - URL de la aplicación desplegada (si es que se ha desplegado)

   Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitLab los incluya como enlaces "pinchables".

27.5. 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.

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

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. Los documentos XML deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el del módulo xml.sax de Python. Los documentos JSON generados deberán ser correctos desde el punto de vista de la sintaxis JSON, y por lo tanto reconocibles por un reconocedor JSON, como por ejemplo el del módulo json de Python

27.6. Preguntas y respuestas

A continuación, algunas preguntas relacionadas con el enunciado de esta práctica, junto con sus respuestas:

- ¿Es necesario utilizar los mecanismos provistos por Django para el control de sesiones y autenticación?

  En principio, esa es la solución recomendada. El principal problema suele ser asegurarse de que cuaquier mecanismo alternativo funciona al menos tan bien como el de Django, lo que no es en general trivial. De todas formas, salvo muy buenos motivos, la aplicación es una aplicación Django, y por lo tanto cuantas más facilidades de Django se usen (bien usadas), mejor.

- ¿Puedo guardar en la base de datos los datos referentes a latitud, altitud, etc (datos que no varian nunca) y precipitación, temperatura, descripción, etc y cambiarlos cuando sea necesario (ya que estos si cambian)?

  Pueden almacenarse en tablas en la base de datos los datos correspondientes a poblaciones que han sido seleccionadas por al menos un usuario. En otras palabras, cada vez que un usuario seleccione un municipio, puedes guardar en una tabla en la base de datos los datos sobre ese municipio (includos latitud y longitud). Pero no puedes analizar todos los municipios que hay en el fichero JSON e incorporar su información a la base de datos.

  La información sobre un municipio que puedas almacenar en la base de datos deberá actualizarse cuando se acceda al fichero XML para ese municipio, según indica el enunciado (por ejemplo, porque un usuario selecciona ese municipio, o porque hay un acceso a su página de municipio).

- Los archivos CSS que pueden modificar los usuarios, ¿dónde y cómo debemos guardarlos?

  La forma recomendada de hacerlo es mediante plantillas:

- En el directorio de plantillas incluirías una para la hoja CSS del sitio. Esa plantilla tendría como variables de plantilla los valores que quieras que los usuarios puedan cambiar (color de tipo de letra, tamaño de tipo de letra, etc.).
- Además, para cada usuario, tendrás una tabla donde se almacenarán los valores para ese usuario (normalmente, una fila de la tabla por usuario).
- Tendrás una vista en views.py que se encargará de generar la hoja CSS a partir de la plantilla. Esa vista es la que comprobará si la petición que está atendiendo corresponde a un usuario (en cuyo caso tendrá que obtener los valores para ese usuario de la tabla anterior), o no (en cuyo caso usará valores por defecto). Con los valores que obtenga, generará la hoja CSS a partir de la plantilla anterior.
- Por último, en urls.py tendrás una línea para indicar que si te piden el recurso que sirve la hoja de estilo, llamas a la vista anterior.

- ¿Qué partes de la página tiene que modificar el CSS "customizable" del usuario? En el enunciado de la práctica dice "se usarán hojas CSS para cambiar al menos el tamaño y color de la letra, y el color del fondo, para los elementos marcados con un id, tal y como se especifica en el apartado anterior". En el "apartado anterior" lo que se especifica es que el banner, caja de login, menú y pie de página tienen que ir cada uno en un elemento div con una id. ¿Significa esto que el CSS que personaliza el usuario se aplica solo a esos cuatro elementos, o aplica a toda la página? ¿En el caso de ser a cada uno de los cuatro elementos, debería el usuario poder modificar el color y letra de cada uno de ellos por separado, o aplicaría para los cuatro el mismo estilo?

  Creemos que el enunciado no es ambiguo. Debe haber, por un lado, estilos CSS que afecten, como mínimo, al tamaño y color de la letra, y al color de fondo, de los elementos que es obligatorio marcar con un id, según indica el enunciado (efectivamente, el banner, la caja de login, etc.) Pueden llevar todos los mismos valores, o valores diferentes, como quiera quien realice la práctica, pero los estilos tienen que estar aplicados específicamente a esos ids.

  Por otro lado, el usuario puede especificar unos cuantos valores para toda la página (según indica el enunciado: "Un formulario para cambiar el estilo CSS de todo el sitio para ese usuario. Bastará con que se pueda cambiar el tamaño y el color de la letra y el color de fondo. Si se cambian estos valores, quedará cambiado el documento CSS que utilizarán todas las páginas del sitio para este usuario." Esto es, al indicar en el formulario valores para lo que puede

personalizar el usuario (como mínimo el color de la letra y el color de fondo) estos valores se cambiarán para todo el sitio. Este color de letra y de fondo pueden aplicarse a todos los elementos que se muestren en el sitio, o sólo a algunos de ellos (por ejemplo, a todos los que no se ven afectados por los id mencionados anteriormente), según quiera el alumno. Lo importante es que el cambio afecte, en los elementos que se vean afectados, a todas las páginas del sitio. Naturalmente, si se decide cambiar por ejemplo la apariencia de todos los elementos del sitio, eso afectará también a los que tengan id Por eso quizás no sea una buena idea cambiar también estos elementos, desde el punto de vista estético, dado que quizás sea mejor que aparezcan con u color de letra y/o de fondo diferente. Pero eso queda como decisión del alumno.

- Si decido trabajar en la opción de despliegue de la aplicación, ¿dónde puedo realizar este despliegue?

  El despliegue puede realizarse en caulquier ordenador que esté conectado permanentemente a Internet durante el periodo de correción, en una dirección accesible desde cualquier navegador conectado a su vez a Internet. Esto puede ser por ejemplo un ordenador personal en un domicilio con acceso permanente a Intener, adecaudamente configurado (puede ser una Raspberry Pi o similar, si se busca una solución simple y de bajo coste). También puede ser un servicio en Internet, por ejemplo uno gratuito como los que ofrecen Google (instrucciones[97], precios[98]), Heroku (instrucciones[99], características[100]) o PythonAmywhere (instrucciones[101], precios[102]).

- Para las URLs de los documentos XML que ofrece MiTiempo, ¿puedo usar la terminación `format=xml` en lugar de `?format=xml` ?

  Sí. Debido a un error, los primeros enunciados mencionaban la terminación `format=xml` para estos ficheros. Por ello, el alumno puede elegir entre servirlos con ese nombre de recurso, o con el que indica la versión final del enunciado, `?format=xml`. Si aún no se ha realizado la implementación de

[97] GCP Quickstart Using a Linux VM:
https://cloud.google.com/compute/docs/quickstart-linux
[98] Google Compute Engine Pricing:
https://cloud.google.com/compute/pricing
[99] Heroku Deploying with Git:
https://devcenter.heroku.com/categories/deploying-with-git
[100] Heroku Free Dyno Hours:
https://devcenter.heroku.com/articles/free-dyno-hours
[101] Capítulo "Deploy!" de Django Girls Tutorial:
https://tutorial.djangogirls.org/en/deploy/
[102] PythonAnywhere Plans and Pricing:
https://www.pythonanywhere.com/pricing/

ninguna de las dos formas, se recomienda hacerlo como indica el enunciado definitivo, porque eso permitirá utilizar la misma vista (view) que se utiliza para el documento HTML correspondiente, simplemente comprobando si la petición incluye una "query string" (utilizando los mecanismos pertinentes de Django). Pero como se ha dicho, si el alumno prefiere implementarlo de la otra forma, se considerara de la misma manera.

28. Proyecto final (2019, junio)

El proyecto final para la convocatoria de junio de 2019 será la misma que la descrita para la convocatoria de mayo de 2019, con las siguientes consideraciones:

- En vez de "*La página principal se ofrece rá también como un documento XML, que incluirá la misma lista de municipios, y un enlace al fichero XML que proporciona para cada uno de ellos la AEMET. Este documento se ofrecerá cuando se pida la URL de municipios, concatenando al final* `?format=xml`' ahora será "*La página principal se ofrecerá también como un documento JSON, que incluirá la misma lista de municipios, y un enlace al fichero XML que proporciona para cada uno de ellos la AEMET. Este documento se ofrecerá cuando se pida la URL de municipios, concatenando al final* `?format=json`".
- En vez de "*Mostrará un listado de los 10 municipios con más comentarios. Si no hubiera 10 municipios con comentarios, se mostrarán sólo los que tengan comentarios*" ahora será "*Mostrará un listado de los 10 municipios más seleccionados por los usuarios. Si no hubiera 10 municipios seleccionados, se mostrarán sólo los que se hayan seleccionado*".

Las fechas de entrega, publicación y revisión de esta convocatoria quedan como siguen:

- **Fecha límite de entrega de la práctica:** lunes, 1 de julio de 2019 a las 05:00 (hora española peninsular)[103].
- **Fecha de publicación de notas:** miércoles, 3 de julio de 2019, en la plataforma Moodle.
- **Fecha de revisión:** viernes, 5 de julio de 2019 a las 10:00.

[103] Entiéndase la hora como domingo por la noche, ya entrado el lunes.

29. Proyecto final (2018, mayo)

El proyecto final de la asignatura consiste en la creación de una aplicación web que aglutine información sobre museos de la ciudad de Madrid. 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.

La aplicación se encargará de descargar información sobre los mencionados museos, disponibles públicamente en varios formatos en el portal de datos abiertos de Madrid, y de ofrecerlos a los usuarios para que puedan seleccionar los que les parezca más interesantes, y comentar sobre ellos. De esta manera, un escenario típico es el de un usuario que a partir de los museos disponibles, elija los que le parezca más adecuados, y comente sobre los que quiera.

29.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de 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 contenga la aplicación tendrá que ser accesible vía este "Admin Site".
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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 izquierda.
  - 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). Esta caja aparecerá en la parte superior derecha.

- Un menú de opciones, como barra, debajo de los dos elementos anteriores (banner y caja de entrada o salida).
- Un pie de página con una nota de atribución, indicando "Esta aplicación utiliza datos del portal de datos abiertos de la ciudad de Madrid", y un enlace a la página con los datos, y a la descripción de los mismos (ver enlaces más abajo).

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 la práctica. 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.
- Se utilizará, para componer la información sobre museos, la disponible en el portal de datos abiertos de la ciudad de Madrid:
- Fichero con los datos abiertos de museos proporcionado por el Ayuntamiento de Madrid:
  https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=118f2fdbecc63410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default
- Copia del fichero anterior en el repositorio CursosWeb/Code de GitHub:
  https://github.com/CursosWeb/CursosWeb.github.io/tree/master/etc

Funcionamiento general:

- Los usuarios serán dados de alta en la práctica mediante el módulo "Admin Site" de Django. Una vez estén dados de alta, serán considerados "usuarios registrados".
- El listado de museos se cargará a partir del fichero XML cuando un usuario indique que se carguen. Hasta que algún usuario indique por primera vez que se carguen los datos, no habrá listado de museos en la base de datos te la aplicación.
- Los usuarios registrados podrán crear su selección de museos. Para ello, dispondrán de una página personal. Llamaremos a esta página la "página del usuario".

- La selección de museos en su página personal la realizará cada usuario a partir de información sobre museos ya disponibles en el sitio.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la página personal de cada usuario, para todos los usuarios del sitio.
- Cualquier usuario podrá indicar que quiere una vista del sitio que incluya sólo los museos (los que en XML tienen el atributo de nombre "Accesibilidad" con valor "1").

29.2. Funcionalidad mínima

Los museos se obtendrán a partir de la información pública ofrecida por el Ayuntamiento de Madrid en el Portal de Datos Abiertos, en forma de ficheros XML, como se indicaba anteriormente.

La **interfaz pública** contiene los 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 la práctica. Constará de un listado de museos y otro con enlaces a páginas personales:

  1. Mostrará un listado de los cinco museos con más comentarios. Si no hubiera 5 museos con comentarios, se mostrarán sólo los que tengan comentarios. Para cada museo, incluirá información sobre:
     - su nombre (que será un enlace que apuntará a la URL del museo en el portal esmadrid),
     - su dirección,
     - y un enlace, "Más información", que apuntará a la página del museo en la aplicación (ver más adelante).
  2. También se mostrará un listado, en una columna lateral, con enlaces a las páginas personales disponibles. Para cada página personal mostrará el título que le haya dado su usuario (como un enlace a la página personal en cuestión) y el nombre del usuario. 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.
  3. Tambień se mostrará un botón, que al pulsarlo se pasará a ver en todos los listados los museos accesibles, y sólo estos. Si se vuelve a pulsar, se volverán a ver todos los museos.

- /usuario: Página personal de un usuario. Si la URL es "/usuario", es que corresponde al usuario "usuario". Mostrará los museos seleccionados por ese usuario (aunque no puede haber más de 5 a la vez; si hay más debería haber un enlace para mostrar las 5 siguientes y así en adelante, siempre de 5 en 5). Para cada museo se mostrará la misma información que en la página principal. Además, para cada museo se deberá mostrar la fecha en la que fue seleccionada por el usuario.

- /museos: Página con todos los museos. Para cada uno de ellos aparecerá sólo el nombre, y un enlace a su página. En la parte superior de la página, existirá un formulario que permita filtrar estos museos según el distrito. Para poder filtrar por distrito, se buscará en la base de datos cuáles son los distritos con museos.

- /museos/id: Página de un museo en la aplicación. Mostrará toda la información razonablemente posible de XML del portal de datos abierto del Ayuntamiento de Madrid, incluyendo al menos la que se menciona en otros apartados de este enunciado, la dirección, la descripción, si es accesible o no, el barrio y el distrito, y los datos de contacto. Además, se mostrarán todos los comentarios que se hayan puesto para este museo.

- /usuario/xml: Canal XML para los museos seleccionados por ese usuario. El documento XML tendrá una entrada para cada museo seleccionado por el usuario, y tendrá una estructura similar (pero no necesariamente igual) a la del fichero XML del portal del Ayuntamiento.

- /about: Página con información en HTML indicando la autoría de la práctica, explicando su funcionamiento.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todos los museos (URL /museos) con el texto "Todos" y a la ayuda (URL /about) con el texto "About". Todas las página que no sean la principal tendrán otra opción de menú para la URL /, con el texto "Inicio".

La **interfaz privada** contiene los recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado):

- Todos los recursos de la interfaz pública.

- /museos/id: Además de la información que se muestra de manera pública:

    1. Un formulario para poner comentarios sobre este museo. Los comentarios serán anónimos, pero sólo se podrán poner por los usuarios registrados, una vez se han autenticado. Por tanto, bastará con que este

formulario esté compuesto por una caja de texto, donde se podrá escribir el comentario, y un botón para enviarlo.

- /usuario: Además de la información que se muestra de manera pública:

  1. Un formulario para cambiar el estilo CSS de todo el sitio para ese usuario. Bastará con que se pueda cambiar el tamaño de la letra y el color de fondo. Si se cambian estos valores, quedará cambiado el documento CSS que utilizarán todas las páginas del sitio para este usuario. Este cambio será visible en cuanto se suba la nueva página CSS.
  2. Un formulario para elegir el título de su página personal.

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

29.3. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habeis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio
- Generación de un canal XML para los contenidos que se muestran en la página principal.
- Generación de canales, pero con los contenidos en JSON
- Generación de un canal RSS para los comentarios puestos en el sitio.
- Funcionalidad para leer los datos del Ayuntamiento en otros formatos diferentes a XML: CSV, JSON...
- Funcionalidad de registro de usuarios
- Uso de Javascript o AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar un museo para una página de usuario).

- Puntuación de museos. Cada visitante (registrado o no) puede dar un "+1" a cualquier museo del sitio. La suma de "+" que ha obtenido un museo se verá cada vez que se vea el museo en el sitio.
- Uso de elementos HTML5 (especificar cuáles al entregar)
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.

29.4. Entrega de la práctica

- **Fecha límite de entrega de la práctica:** lunes, 21 de mayo de 2018 a las 03:00 (hora española peninsular)[104]
- **Fecha de publicación de notas:** miércoles, 23 de mayo de 2018, en la plataforma Moodle.
- **Fecha de revisión:** jueves, 24 de mayo de 2018 a las 13:00.

La entrega de la práctica consiste en rellenar un formulario (enlazado en el Moodle de la asignatura) y en seguir las instrucciones que se describen a continuación.

1. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado un fork del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://github.com/CursosWeb/X-Serv-Practica-Museos

   Los alumnos que no entreguen las 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 de la revisión. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitHub.

[104] Entiéndase la hora como domingo por la noche, ya entrado en lunes.

2. Un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional, si se han realizado opciones avanzadas. Los vídeos 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). 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). Los vídeos de más de tres minutos tendrán penalización.

   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.

3. Se han de entregar los siguientes ficheros:

   - Un fichero README.md que resuma las mejoras, si las hay, y explique cualquier peculiaridad de la entrega (ver siguiente punto).
   - El repositorio GitHub deberá contener 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, con al menos cinco museos en su página personal, y con al menos cinco comentarios en total.
   - Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, junto con los ficheros auxiliares que utilice, si es que los utiliza.

4. Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica - se recomienda hacer un corta y pega de estos datos en el formulario):

   - Nombre y titulación.
   - Nombre de su cuenta en el laboratorio del alumno.

- Nombre de usuario en GitHub.
- 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

Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitHub los incluya como enlaces "pinchables".

29.5. 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.

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

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. Los documentos XML deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el reconocedor del módulo xml.sax de Python.

30. Proyecto final (2018, junio)

El proyecto final para la convocatoria de junio de 2018 será la misma que la descrita para la convocatoria de mayo de 2018, con las siguientes consideraciones:

- En vez de /usuario/xml: Canal XML para los museos seleccionados por ese usuario, se ofrecerá el canal en formato JSON. El documento JSON tendrá una entrada para cada museo seleccionado por el usuario, y tendrá una estructura similar (pero no necesariamente igual) a la del fichero JSON del portal del Ayuntamiento.
- La página principal no mostrará los cinco museos más comentados, sino que mostrará los cinco museos más seleccionados por usuarios para sus páginas personales.

Las fechas de entrega, publicación y revisión de esta convocatoria quedan como siguen:

- **Fecha límite de entrega de la práctica:** viernes, 28 de junio de 2018 a las 05:00 (hora española peninsular)[105].
- **Fecha de publicación de notas:** domingo, 1 de julio de 2018, en la plataforma Moodle.
- **Fecha de revisión:** martes, 3 de julio de 2018 a las 12:00.

[105] Entiéndase la hora como jueves por la noche, ya entrado el viernes.

31. Proyecto final (2017, mayo)

El proyecto final de la asignatura consiste en la creación de una aplicación web que aglutine información sobre aparcamientos en la ciudad de Madrid. 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.

La aplicación se encargará de descargar información sobre los mencionados aparcamientos, disponibles públicamente en formato XML en el portal de datos abiertos de Madrid, y de ofrecerlos a los usuarios para que puedan seleccionar los que les parezca más interesantes, y comentar sobre ellos. De esta manera, un escenario típico es el de un usuario que a partir de los aparcamientos disponibles, elija los que le parezca más adecuados, y comente sobre los que quiera.

31.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica se construirá como un proyecto Django/Python3, que incluirá una o varias aplicaciones Django que implementen la funcionalidad requerida.
- Para el almacenamiento de datos persistente se usará SQLite3, con tablas definidas en modelos de 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 contenga la aplicación tendrá que ser accesible vía este "Admin Site".
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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 izquierda.
  - 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). Esta caja aparecerá en la parte superior derecha.
  - Un menú de opciones, como barra, debajo de los dos elementos anteriores (banner y caja de entrada o salida).
  - Un pie de página con una nota de atribución, indicando "Esta aplicación utiliza datos del portal de datos abiertos de la ciudad de Madrid", y un enlace al XML con los datos, y a la descripción de los mismos (ver enlaces más abajo).

  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 la práctica. 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.

- Se utilizará, para componer la información sobre aparcamientos disponibles, la disponible en el portal de datos abiertos de la ciudad de Madrid:

- Fichero con los datos abiertos de aparcamientos para residentes proporcionado por el Ayuntamiento de Madrid:
  http://datos.munimadrid.es/portal/site/egob/menuitem.ac61933d6ee3c31cae77ae7784f1a5a0/?vgnextoid=00149033f2201410VgnVCM100000171f5a0aRCRD&format=xml&file=0&filename=202584-0-aparcamientos-residentes&mgmtid=e84276ac109d3410VgnVCM2000000c205a0aRCRD&preview=full

- Descripción del fichero:
  http://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=e84276ac109d3410VgnVCM2000000c205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

- Copia del fichero anterior en el repositorio CursosWeb/Code de GitHub:

Funcionamiento general:

- Los usuarios serán dados de alta en la práctica mediante el módulo "Admin Site" de Django. Una vez estén dados de alta, serán considerados "usuarios registrados".

- El listado de aparcamientos se cargará a partir del fichero XML cuando un usuario indique que se carguen. Hasta que algún usuario indique por primera vez que se carguen los datos, no habrá listado de aparcamientos en la base de datos te la aplicación.
- Los usuarios registrados podrán crear su selección de aparcamientos. Para ello, dispondrán de una página personal. Llamaremos a esta página la "página del usuario".
- La selección de aparcamientos en su página personal la realizará cada usuario a partir de información sobre aparcamientos ya disponibles en el sitio.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la página personal de cada usuario, para todos los usuarios del sitio.
- Cualquier usuario podrá indicar que quiere una vista del sitio que incluya sólo los aparcamientos accesibles (los que en XML tienen "accesibility" con valor "1").

31.2. Funcionalidad mínima

Los aparcamientos se obtendrán a partir de la información pública ofrecida por el Ayuntamiento de Madrid en el Portal de Datos Abiertos, en forma de ficheros XML, como se indicaba anteriormente.

La **interfaz pública** contiene los 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 la práctica. Constará de un listado de aparcamientos y otro con enlaces a páginas personales:

    1. Mostrará un listado de los cinco aparcamientos con más comentarios. Si no hubiera 5 aparcamientos con comentarios, se mostrarán sólo los que tengan comentarios. Para cada aparcamiento, incluirá información sobre:
        - su nombre (que será un enlace que apuntará a la url del aparcamiento en el portal esmadrid),
        - su dirección,
        - y un enlace, "Más información", que apuntará a la página del aparcamiento en la aplicación (ver más adelante).

2. También se mostrará un listado, en una columna lateral, con enlaces a las páginas personales disponibles. Para cada página personal mostrará el título que le haya dado su usuario (como un enlace a la página personal en cuestión) y el nombre del usuario. 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.
3. Tambień se mostrará un botón, que al pulsarlo se pasará a ver en todos los listados los aparcamientos accesibles, y sólo estos. Si se vuelve a pulsar, se volverán a ver todos los aparcamientos.

- /usuario: Página personal de un usuario. Si la URL es "/usuario", es que corresponde al usuario "usuario". Mostrará los aparcamientos seleccionados por ese usuario (aunque no puede haber más de 5 a la vez; si hay más debería haber un enlace para mostrar las 5 siguientes y así en adelante, siempre de 5 en 5). Para cada aparcamiento se mostrará la misma información que en la página principal. Además, para cada aparcamiento se deberá mostrar la fecha en la que fue seleccionada por el usuario.

- /aparcamientos: Página con todos los aparcamientos. Para cada uno de ellos aparecerá sólo el nombre, y un enlace a su página. En la parte superior de la página, existirá un formulario que permita filtrar estos aparcamientos según el distrito. Para poder filtrar por distrito, se buscará en la base de datos cuáles son los distritos con aparcamientos.

- /aparcamientos/id: Página de un aparcamiento en la aplicación. Mostrará toda la información razonablemente posible de XML del portal de datos abierto del Ayuntamiento de Madrid, incluyendo al menos la que se menciona en otros apartados de este enunciado, la información de latitud y longitud, la descripción, si es accesible o no, el barrio y el distrito, y los datos de contacto. Además, se mostrarán todos los comentarios que se hayan puesto para este aparcamiento.

- /usuario/xml: Canal XML para los aparcamientos seleccionados por ese usuario. El documento XML tendrá una entrada para cada aparcamiento seleccionado por el usuario, y tendrá una estructura similar (pero no necesariamente igual) a la del fichero XML del portal del Ayuntamiento.

- /about: Página con información en HTML indicando la autoría de la práctica y explicando su funcionamiento.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todos los aparcamientos (URL /aparcamientos) con el texto "Todos" y

a la ayuda (URL /about) con el texto "About". Todas las página que no sean la principal tendrán otra opción de menú para la URL /, con el texto "Inicio".

La **interfaz privada** contiene los recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado):

- Todos los recursos de la interfaz pública.
- /aparcamientos/id: Además de la información que se muestra de manera pública:

    1. Un formulario para poner comentarios sobre este aparcamiento. Los comentarios serán anónimos, pero sólo se podrán poner por los usuarios registrados, una vez se han autenticado. Por tanto, bastará con que este formulario esté compuesto por una caja de texto, donde se podrá escribir el comentario, y un botón para enviarlo.

- /usuario: Además de la información que se muestra de manera pública:

    1. Un formulario para cambiar el estilo CSS de todo el sitio para ese usuario. Bastará con que se pueda cambiar el tamaño de la letra y el color de fondo. Si se cambian estos valores, quedará cambiado el documento CSS que utilizarán todas las páginas del sitio para este usuario. Este cambio será visible en cuanto se suba la nueva página CSS.
    2. Un formulario para elegir el título de su página personal.

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

31.3. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habeis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio

- Generación de un canal XML para los contenidos que se muestran en la página principal.
- Generación de canales, pero con los contenidos en JSON
- Generación de un canal RSS para los comentarios puestos en el sitio.
- Funcionalidad de registro de usuarios
- Uso de Javascript o AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar un aparcamiento para una página de usuario).
- Puntuación de aparcamientos. Cada visitante (registrado o no) puede dar un "+1" a cualquier aparcamiento del sitio. La suma de "+" que ha obtenido un aparcamiento se verá cada vez que se vea el aparcamiento en el sitio.
- Uso de elementos HTML5 (especificar cuáles al entregar)
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.

31.4. Entrega de la práctica

- **Fecha límite de entrega de la práctica:** miércoles, 24 de mayo de 2017 a las 03:00 (hora española peninsular)[106]
- **Fecha de publicación de notas:** sábado, 27 de mayo de 2017, en la plataforma Moodle.
- **Fecha de revisión:** lunes, 29 de mayo de 2017 a las 13:00.

La entrega de la práctica consiste en rellenar un formulario (enlazado en el Moodle de la asignatura) y en seguir las instrucciones que se describen a continuación.

1. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado un fork del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://github.com/CursosWeb/X-Serv-Practica-Aparcamientos/

   Los alumnos que no entreguen las práctica de esta forma serán considerados como no presentados en lo que a la entrega de prácticas se refiere. Los que

[106] Entiéndase la hora como miércoles por la noche, ya entrado el jueves.

la entreguen podrán ser llamados a realizar también una entrega presencial, que tendrá lugar en la fecha y hora de la revisión. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitHub.

2. Un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional, si se han realizado opciones avanzadas. Los vídeos 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). 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.

3. Se han de entregar los siguientes ficheros:

   - Un fichero README.md que resuma las mejoras, si las hay, y explique cualquier peculiaridad de la entrega (ver siguiente punto).
   - El repositorio GitHub deberá contener 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, con al menos cinco aparcamientos en su página personal, y con al menos cinco comentarios en total.
   - Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, junto con los ficheros auxiliares que utilice, si es que los utiliza.

4. Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica - se recomienda hacer un corta y pega de estos datos en el formulario):

   - Nombre y titulación.
   - Nombre de su cuenta en el laboratorio del alumno.
   - Nombre de usuario en GitHub.
   - 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

   Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitHub los incluya como enlaces "pinchables".

31.5. 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.

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

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. Los documentos XML deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el reconocedor del módulo xml.sax de Python.

32. Proyecto final (2017, junio)

El proyecto final para la convocatoria de junio de 2017 será la misma que la descrita para la convocatoria de mayo de 2017, con las siguientes consideraciones:

- La puntuación de aparcamientos será requisito de la práctica básica.

- El formulario para poner comentarios deja de ser un requisito de la práctica básica.

Las fechas de entrega, publicación y revisión de esta convocatoria quedan como siguen:

- **Fecha límite de entrega de la práctica:** jueves, 29 de junio de 2017 a las 03:00 (hora española peninsular)[107].
- **Fecha de publicación de notas:** sábado, 1 de julio de 2017, en la plataforma Moodle.
- **Fecha de revisión:** lunes, 4 de julio de 2017 a las 13:00.

[107] Entiéndase la hora como jueves por la noche, ya entrado el viernes.

33. Proyecto final (2016, mayo)

El proyecto final de la asignatura consiste en la creación de una aplicación web que aglutine información sobre alojamientos en la ciudad de Madrid. 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.

La aplicación se encargará de descargar información sobre los mencionados alojamientos, disponibles públicamente en formato XML en el portal de datos abiertos de Madrid, y de ofrecerlos a los usuarios para que puedan seleccionar los que les parezca más interesantes, y comentar sobre ellos. De esta manera, un escenario típico es el de un usuario que a partir de los alojamientos disponibles, elija los que le parezca más adecuados, y comente sobre los que quiera.

33.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica 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 en modelos de 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 contenga la aplicación tendrá que ser accesible vía este "Admin Site".
- Se utilizarán plantillas Django (a ser posible, una jerarquía de plantillas, para que la práctica 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 izquierda.
  - 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). Esta caja aparecerá en la parte superior derecha.
    - Un menú de opciones, como barra, debajo de los dos elementos anteriores (banner y caja de entrada o salida).
    - Un pie de página con una nota de atribución, indicando "Esta aplicación utiliza datos del portal de datos abiertos de la ciudad de Madrid", y un enlace al XML con los datos, y a la descripción de los mismos (ver enlaces más abajo).

  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 la práctica. 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.

- Se utilizará, para componer la información sobre alojamientos disponibles, la disponible en el portal de datos abiertos de la ciudad de Madrid:

    - Descripción:
      http://bit.ly/1T24Zsq
    - Fichero XML con los datos (en español):
      http://www.esmadrid.com/opendata/alojamientos_v1_es.xml
      http://cursosweb.github.io/etc/alojamientos_es.xml
    - Fichero XML con los datos (en inglés):
      http://www.esmadrid.com/opendata/alojamientos_v1_en.xml
      http://cursosweb.github.io/etc/alojamientos_en.xml
    - Fichero XML con los datos (en francés):
      http://www.esmadrid.com/opendata/alojamientos_v1_fr.xml
      http://cursosweb.github.io/etc/alojamientos_fr.xml
    - Hay ficheros XML con los datos en otros idiomas

Funcionamiento general:

- Los usuarios serán dados de alta en la práctica mediante el módulo "Admin Site" de Django. Una vez estén dados de alta, serán considerados "usuarios registrados".

- El listado de alojamientos se cargará a partir del XML con los datos en español sólo cuando un usuario indique que quiere que se carguen. Hasta que algún usuario indique por primera vez que se carguen los datos, no habrá listado de alojamientos en la base de datos te la aplicación.
- Los usuarios registrados podrán crear su selección de alojamientos. Para ello, dispondrán de una página personal. Llamaremos a esta página la "página del usuario".
- La selección de alojamientos en su página personal la realizará cada usuario a partir de información sobre alojamientos ya disponibles en el sitio.
- Cualquier navegador podrá acceder a la interfaz pública del sitio, que ofrecerá la página personal de cada usuario, para todos los usuarios del sitio.
- Cualquier usuario, al ver la página de alojamientos de cualquier usuario (incluido él mismo), podrá pedir verla en otro de los idiomas disponibles. En ese caso, la aplicación descargará el documento XML con el listado de alojamientos en el idioma elegido, buscará los alojamientos en cuestión, y usará sus datos para mostrar la misma página, pero con los datos sobre los alojamientos en ese idioma. La aplicación no almacenará estos datos en otro idioma en la base de datos, de forma que si se le vuelve a pedir lo mismo, volverá a descargar el fichero XML.

33.2. Funcionalidad mínima

Los alojamientos se obtendrán a partir de la información pública ofrecida por el Ayuntamiento de Madrid en el Portal de Datos Abiertos, en forma de ficheros XML, como se indicaba anteriormente.

La **interfaz pública** contiene los 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 la práctica. Constará de un listado de alojamientos y otro con enlaces a páginas personales:

    1. Mostrará un listado de los diez alojamientos con más comentarios. Si no hubiera 10 alojamientos con comentarios, se mostrarán sólo los que tengan comentarios. Para cada alojamiento, incluirá información sobre:
        - su nombre (que será un enlace que apuntará a la url del alojamiento en el portal esmadrid),
        - su dirección,

    - una imagen suya en pequeño formato,
    - y un enlace, "Más información", que apuntará a la página del alojamiento en la aplicación (ver más adelante).
  2. También se mostrará un listado, en una columna lateral derecha, con enlaces a las páginas personales disponibles. Para cada página personal mostrará el título que le haya dado su usuario (como un enlace a la página personal en cuestión) y el nombre del usuario. 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 los alojamientos seleccionados por ese usuario (aunque no puede haber más de 10 a la vez; si hay más debería haber un enlace para mostrar las diez siguientes y así en adelante, siempre de diez en diez). Para cada alojamiento se mostrará la misma información que en la página principal. Además, para cada alojamiento se deberá mostrar la fecha en la que fue seleccionada por el usuario.

- /alojamientos: Página con todos los alojamientos. Para cada uno de ellos aparecerá sólo el nombre, y un enlace a su página. En la parte superior de la página, existirá un formulario que permita filtrar estos alojamientos según varios campos, como, por ejemplo, por su categoría (por ejemplo, "Hoteles") y su subcategoría (por ejemplo, "4 estrellas") .

- /alojamientos/id: Página de un alojamiento en la aplicación. Mostrará toda la información de los elementos "basicData" y "geoData" obtenida del XML del portal de datos abierto del Ayuntamiento de Madrid. Además, se mostrarán cinco fotos entre las que se pueden obtener del mismo documento XML (o menos, si en el documento no hay tantas), y todos los comentarios que se hayan puesto para este alojamiento.

- /usuario/xml: Canal XML para los alojamientos seleccionados por ese usuario. El documento XML tendrá una entrada para cada alojamiento seleccionado por el usuario, y tendrá una estructura similar (pero no necesariamente igual) a la del fichero XML del portal del Ayuntamiento.

- /about: Página con información en HTML indicando la autoría de la práctica y explicando su funcionamiento.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todos los alojamientos (URL /alojamientos) con el texto "Todos"

y a la ayuda (URL /about) con el texto "About". Todas las página que no sean la principal tendrán otra opción de menú para la URL /, con el texto "Inicio".

La **interfaz privada** contiene los recursos a servir como páginas HTML completas para usuarios registrados (una vez se han autenticado):

- Todos los recursos de la interfaz pública.
- /alojamientos/id: Además de la información que se muestra de manera pública:

    1. Un formulario para poner comentarios sobre este alojamiento. Los comentarios serán anónimos, pero sólo se podrán poner por los usuarios registrados, una vez se han autenticado. Por tanto, bastará con que este formulario esté compuesto por una caja de texto, donde se podrá escribir el comentario, y un botón para enviarlo.
    2. Un botón para cada uno de los idiomas en que está disponible el documento XML en el portal del Ayuntamiento. En caso de que el usuario pulse uno de esos botones, la aplicación descargará el XML correspondiente al idioma seleccionado, buscará en él la información sobre el alojamiento en cuestión, y si está disponible, la mostrará en pantalla en ese idioma (además de la información que ya estaba disponible). Si el alojamiento no está disponible en ese idioma, se pondrá un mensaje indicándolo. Esta información en otros idiomas no se guardará en la base de datos.

- /usuario: Además de la información que se muestra de manera pública:

    1. Un formulario para cambiar el estilo CSS de todo el sitio para ese usuario. Bastará con que se pueda cambiar el tamaño de la letra y el color de fondo. Si se cambian estos valores, quedará cambiado el documento CSS que utilizarán todas las páginas del sitio para este usuario. Este cambio será visible en cuanto se suba la nueva página CSS.
    2. Un formulario para elegir el título de su página personal.

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

33.3. 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.

En el formulario de entrega se pide que se justifique por qué se considera funcionalidad optativa lo que habeis implementado. Sólo a modo de sugerencia, se incluyen algunas posibles funcionalidades optativas:

- Inclusión de un *favicon* del sitio
- Generación de un canal XML para los contenidos que se muestran en la página principal.
- Generación de canales, pero con los contenidos en JSON
- Generación de un canal RSS para los comentarios puestos en el sitio.
- Funcionalidad de registro de usuarios
- Uso de Javascript o AJAX para algún aspecto de la práctica (por ejemplo, para seleccionar un alojamiento para una página de usuario).
- Puntuación de alojamientos. Cada visitante (registrado o no) puede dar un "+1" a cualquier alojamiento del sitio. La suma de "+" que ha obtenido un alojamiento se verá cada vez que se vea el alojamiento en el sitio.
- Uso de elementos HTML5 (especificar cuáles al entregar)
- Atención al idioma indicado por el navegador. El idioma de la interfaz de usuario del planeta tendrá en cuenta lo que especifique el navegador.

33.4. Entrega de la práctica

- **Fecha límite de entrega de la práctica:** lunes, 23 de mayo de 2016 a las 02:00 (hora española peninsular)[108]
- **Fecha de publicación de notas:** martes, 24 de mayo de 2016, en la plataforma Moodle.
- **Fecha de revisión:** miércoles, 25 de mayo de 2016 a las 13:30.

[108] Entiéndase la hora como domingo por la noche, ya entrado el lunes.

La entrega de la práctica consiste en rellenar un formulario (enlazado en el Moodle de la asignatura) y en seguir las instrucciones que se describen a continuación.

1. El repositorio contendrá todos los ficheros necesarios para que funcione la aplicación (ver detalle más abajo). Es muy importante que el alumno haya realizado un fork del repositorio que se indica a continuación, porque si no, la práctica no podrá ser identificada:

   https://github.com/CursosWeb/X-Serv-Practica-Hoteles/

   Los alumnos que no entreguen las 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 de la revisión. Esta entrega presencial podrá incluir una conversación con el profesor sobre cualquier aspecto de la realización de la práctica.

   Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podéis cambiar el nombre del repositorio desde el interfaz web de GitHub.

2. Un vídeo de demostración de la parte obligatoria, y otro vídeo de demostración de la parte opcional, si se han realizado opciones avanzadas. Los vídeos 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). 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.

3. Se han de entregar los siguientes ficheros:

- Un fichero README.md que resuma las mejoras, si las hay, y explique cualquier peculiaridad de la entrega (ver siguiente punto).
- El repositorio GitHub deberá contener 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, con al menos cinco alojamientos en su página personal, y con al menos cinco comentarios en total.
- Cualquier biblioteca Python que pueda hacer falta para que la aplicación funcione, junto con los ficheros auxiliares que utilice, si es que los utiliza.

4. Se incluirán en el fichero README.md los siguientes datos (la mayoría de estos datos se piden también en el formulario que se ha de rellenar para entregar la práctica - se recomienda hacer un corta y pega de estos datos en el formulario):

   - Nombre y titulación.
   - Nombre de su cuenta en el laboratorio del alumno.
   - Nombre de usuario en GitHub.
   - 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

   Asegúrate de que las URLs incluidas en este fichero están adecuadamente escritas en Markdown, de forma que la versión HTML que genera GitHub los incluya como enlaces "pinchables".

33.5. 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.

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

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. Los documentos XML deberán ser correctos desde el punto de vista de la sintaxis XML, y por lo tanto reconocibles por un reconocedor XML, como por ejemplo el reconocedor del módulo xml.sax de Python.

34. Proyecto final (2016, junio)

El proyecto final para la convocatoria de junio de 2016 será la misma que la descrita para la convocatoria de mayo de 2016, salvo la siguiente cuestión:

Los comentarios incluirán información sobre quién los ha introducido, y cada hotel sólo podrá tener un comentario por cada usuario.

Además, las fechas de entrega, publicación y revisión quedan como siguen:

- **Fecha límite de entrega de la práctica:** lunes, 27 de junio de 2016 a las 02:00 (hora española peninsular)[109]. Se ha de entregar el código en GitHub y rellenar el formulario de entrega (incluyendo los enlaces a los vídeos de presentación).
- **Fecha de publicación de notas:** martes, 28 de junio de 2016, en la plataforma Moodle.
- **Fecha de revisión:** jueves, 30 de junio de 2016 a las 13:00.

[109] Entiéndase la hora como domingo por la noche, ya entrado el lunes.

35. Proyecto final (2015, mayo y junio)

El proyecto final de la asignatura consiste en la creación de una aplicación web que aglutine información sobre actividades culturales y de ocio que tienen lugar en el municipio de Madrid. 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.

La aplicación consiste en descargarse datos de actividades culturales (disponbiles 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 a partir de las actividades existentes, incluya en su perfil las que le interesen.

35.1. Arquitectura y funcionamiento general

Arquitectura general:

- La práctica 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 la práctica 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 la práctica. 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 la práctica 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 actividades de cultura y de ocio. Para ello, dispondrán de una página personal. Llamaremos a esta página la "página del usuario".
- La selección de actividades en su página personal la realizará cada usuario a partir de información sobre actividades de ocio y cultura ya disponibles en el sitio.
- Las actividades de ocio y cultura 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.

35.2. Funcionalidad mínima

Las actividades de ocio y de cultura se toman de interpretar la información pública ofrecida por el Ayuntamiento de Madrid en el Portal de Datos Abiertos, y que es la siguiente:

- Actividades Culturales y de Ocio Municipal en los próximos 100 días: http://goo.gl/809BPF

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 la práctica. Mostrará un listado de las diez actividades de ocio y cultura más próximas en el tiempo, que incluya información sobre su título, el tipo de evento y la fecha del mismo. También se mostrará un

listado, probablemente en un lateral, 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 actividades de ocio y de cultura seleccionadas por ese usuario (aunque no puede haber más de 10 a la vez; si hay más debería haber un enlace para mostrar las diez siguientes y así en adelante, siempre de diez en diez). Para cada actividad de ocio y de cultura se mostrará al menos el título y la fecha de los eventos (con un enlace a la página /actividad de cada evento, ver más adelante). Además, para cada actividad se deberá mostrar la fecha en la que fue seleccionada por el usuario.

- /actividad/id: Página de una actividad de cultura o de ocio. Mostrará toda la información obtenida del XML del portal de datos abierto del Ayuntamiento de Madrid. Además, se mostrará su "información adicional", conseguida a partir de seguir la URL con información adicional. Esta información adicional es la que se puede encontrar si seguimos el enlace justo debajo de "Amplíe información". Se puede hacer uso del módulo *Beautiful Soup* para llevar a cabo esta funcionalidad.

- /usuario/rss: Canal RSS para las actividades seleccionadas por ese usuario.

- /ayuda: Página con información HTML explicando el funcionamiento de la práctica.

- /todas: Página con todas las actividades de ocio y de cultura. En la parte superior de la página, existirá un formulario que permite filtrar estas actividades según varios campos, como, por ejemplo, la fecha, la duración, el precio o el título.

Todas las páginas de la interfaz pública incluirán un menú desde el que se podrá acceder a todas las actividades (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.

- /todas: Además de la información que se muestra de manera pública:
  - Se mostrará el número de actividades de ocio y de cultura disponibles para el canal, y la fecha en que fue actualizado por última vez.
  - Existirá un botón para actualizar las actividades a partir del canal de actividades. Si se pulsa este botón, se tratarán de actualizar las actividades accediendo al canal de actividades del Ayuntamiento de Madrid. Al terminar la operación se volverá a mostrar esta misma página /todas, actualizada.
  - La lista de actividades disponibles en el canal de actividades.
  - Junto a cada actividad de la lista, se incluirá un botón que permitirá elegir la actividad para la página personal del usuario autenticado. Tras añadir una actividad a la página del usuario, se volverá a ver en el navegador la página /todas.
- 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.

Dados los recursos mencionados anteriormente, no se permitirán los nombres de usuario "actividad", "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).

35.3. 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 actividad para una página de usuario).
- Puntuación de actividades. Cada visitante (registrado o no) puede dar un "+1" a cualquier actividad del sitio. La suma de "+" que ha obtenido una actividad se verá cada vez que se vea la actividad en el sitio.
- Comentarios a actividades. Cada usuario registrado puede comentar cualquier actividad del sitio. Estos comentarios se podrán ver luego en la página personal.

35.4. Entrega de la práctica

Fecha límite de entrega de la práctica: domingo, 24 de mayo de 2015 a las 23:59 (hora española peninsular). **Convocatoria de junio:** miércoles, 24 de junio de 2015 a las 23:59 (hora peninsular española).

Fecha de publicación de notas: martes, 26 de mayo de 2015, en la plataforma Moodle. **Convocatoria de junio:** viernes, 26 de junio, en la plataforma Moodle.

Fecha de revisión: viernes, 29 de mayo de 2014 a las 12:00. **Convocatoria de junio:** martes, 30 de junio a las 13:30. 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 realizando **dos** acciones:

1. Rellenando un formulario web, que pedirá la siguiente información:
   - 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

2. Subiendo la práctica a un repositorio GitHub. El nombre del repositorio se dará al entregar la práctica. Así, para ir realizando la práctica se recomienda crearse un repositorio en GitHub con el nombre que queráis, e ir haciendo commits. Recordad que es importante ir haciendo commits de vez en cuando y que sólo al hacer push estos commits son públicos. Antes de entregar la práctica, haced un push. Y cuando la entreguéis y sepáis el nombre del repositorio, podeis cambiar el nombre del repositorio desde el interfaz web de GitHub.

   El repositorio GitHub deberá contener 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 actividades en su página personal.

Los vídeos de demostración serán de una duración máxima de tres 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.

Los alumnos que no entreguen las práctica de esta forma serán considerados como no presentados en lo que a la entrega de prácticas se refiere.

35.5. 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 1.7.*).

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

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.

36. Proyecto final (2014, mayo)

El proyecto 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 (disponbiles 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.

36.1. 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.

36.2. 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 (aunuque 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 incidiencias (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)

36.3. 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.

36.4. 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 las 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.

36.5. 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 1.7.*).

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.

37. Proyectos finales anteriores

37.1. Proyecto final (2013, mayo)

El proyecto 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.

37.2. 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.

37.3. 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.

37.4. 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").

37.5. 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 las 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.

37.6. 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 1.7.*).

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.

37.7. Proyecto final (2012, diciembre)

El proyecto 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.

37.7.1. 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.

37.7.2. 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 articulo", ver más abajo.
- /users: Lista de usuarios registrados de MiPlaneta. Para casa 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 el 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.

37.7.3. 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").

37.8. Proyecto final (2011, diciembre)

El proyecto 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.

37.8.1. 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").

37.8.2. 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 $< p >$).
  - 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[110].
    - Canales RSS correspondientes a usuarios de Identi.ca[111].
    - Canales RSS correspondientes a usuarios de Youtube[112].

37.8.3. 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 (esto es, 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.

[110] Para el usuario "jgbarah":
https://twitter.com/statuses/user_timeline/jgbarah.rss
[111] Para el usuario "jgbarah":
http://identi.ca/jgbarah/rss
[112] Para el usuario "user":
http://gdata.youtube.com/feeds/api/videos?max-results=5&alt=rss&author=user

37.8.4. 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.

37.8.5. 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.

37.9. Proyecto final (2012, mayo)

El proyecto final a entregar en la convocatoria extraordinaria (mayo de 2012) será como la de la entrega ordinaria (práctica 37.8), con las diferencias que se indican en los siguientes apartados.

37.9.1. 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.

37.9.2. 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[113].
  - Canal Atom correspondientes al hashtag de Identi.ca definido por esa cadena[114].
  - Canal Atom correspondientes a la búsqueda en Youtube de esa cadena[115].

El resto queda igual.

37.10. Proyecto final (2010, enero)

El proyecto 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.

37.10.1. 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.

[113] Para el hashtag "#urjc":
http://search.twitter.com/search.atom?q=%23urjc
[114] Para el hashtag "#urjc":
http://identi.ca/api/statusnet/tags/timeline/urjc.atom
[115] Para la búsqueda "urjc":
http://gdata.youtube.com/feeds/api/videos?q=urjc&max-results=5&alt=atom

- 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:

- 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.

37.10.2. 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).

37.10.3. 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").

37.10.4. 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.

37.10.5. 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.

37.11. Proyecto final (2010, junio)

El proyecto 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.

37.12. Proyecto 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.

El proyecto 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[116] e Identi.ca[117]. 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.

37.12.1. 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.
- 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.

[116] http://twitter.com
[117] http://identi.ca

- 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.

37.12.2. 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.

37.12.3. 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.

37.12.4. 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 sito 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.]

37.12.5. 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.

37.12.6. 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
  ```

37.13. Proyecto 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.

El proyecto 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[118] e Identi.ca[119]. 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.

37.13.1. 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.

[118] http://twitter.com
[119] http://identi.ca

- Se utilizarán hojas de estilo CSS para determinar la apariencia de la aplicación. Estas hojas se almacenará 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.

37.13.2. 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 deseleccionarla 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.

37.13.3. 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.

37.13.4. 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.]

37.13.5. 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.

37.13.6. 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
```

38. Materiales de interés

38.1. 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

38.2. 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

  "Python Programming", Wikibook sobre programación en Python.

- 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.

38.3. 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.

38.4. SQL y SQLite

- http://www.shokhirev.com/nikolai/abc/sql/sql.html

  "SQLite / SQL Tutorials: Basic SQL", por Nikolai Shokhirev

39. Preguntas más fecuentes

39.1. Django: Referencia a elementos en otro módulo

Esta pregunta se plantea de muchas formas, pero la más habitual es cuando queremos usar un módulo Python que hemos construido para proporcionar una cierta funcionalidad (llamémoslo `modulo.py`), y necesitamos referirnos a alguno de sus elementos (variables, funciones, clases) desde `views.py`. ¿Cómo podemos hacerlo?

Respuesta

Hay varias formas de hacerlo, pero una de las que pueden ser más habituales consiste en colocar el módulo en el mismo directorio donde tenemos `views.py` (normalmente, el directorio de la app Django que estamos construyendo), y luego importar el módulo desde `views.py` usando el formato relativo de importación (donde "." se refiere a módulos en el directorio del que importa).

Si lo hacemos así, nos quedarán en el directorio de la app Django los siguientes ficheros:

- `urls.py`
- `views.py`
- `modulo.py`
- ...

Y el fichero `views.py`, si en él queremos utilizar por ejemplo una función `funcion` y una variable `variable` del módulo `modulo.py`, se escribirá así:

```
...
from .modulo import funcion, variable

...
```

```
... = variable
...
funcion()
...
```