



UNIVERSIDADE DA CORUÑA

Facultade de Informática
Departamento de Computación

PROXECTO FIN DE CARREIRA
Enxeñaría en Informática

**Servizo en liña para a publicación de gravacíons de radio e
podcasts**

Alumno: Fernando Liñares Varela
Director: José María Casanova Crespo
Data: 17 de junio de 2018

D. JOSÉ MARÍA CASANOVA CRESPO
Profesor, Facultade de Informática
Departamento de Computación
Universidade de A Coruña

CERTIFICA:

Que a memoria titulada “*Servizo en liña para a publicación de gravacíons de radio e podcasts*” foi realizada por FERNANDO LIÑARES VARELA conforme á descripción inicialmente proposta baixo a miña dirección e constitúe o seu Proxecto Fin de Carreira de Enxeñaría en Informática. Pola presente, autorizo a súa presentación para que o Proxecto sexa defendido nesta convocatoria.

En A Coruña, a 17 de junio de 2018

D. JOSÉ MARÍA CASANOVA CRESPO
Director do proxecto

Título

Servizo en liña para a publicación de gravacíons de
radio e podcasts

Servicio en línea para la publicación de grabaciones
de radio y podcasts

Online service for publishing radio broadcasting re-
cordings and podcasts

Clase: Proxecto clásico de enxeñaría

Autor: Fernando Liñares Varela

Director: José María Casanova Crespo

Data: 17 de junio de 2018

Tribunal

Data de

defensa:

Calificación:

Resumo

Lista de Palabras Clave

Radio, Podcast, Web, Django, Postgres, Python, Javascript, jQuery, CSS Grid.

Dedicatoria por hacer

Agradecimientos

A los profesores José María Casanova Crespo por sus consejos durante el desarrollo del proyecto. POR HACER

Índice general

1. Introdución	1
1.1. Marco do proxecto	2
1.2. Motivación	3
1.3. Obxectivos	3
2. Estado da arte	5
2.1. Alternativas Existentes	6
2.1.1. TuneIn	6
2.1.2. iTunes	7
2.1.3. Podomatic	7
2.1.4. Ivoox	8
2.1.5. RadioCo	9
2.2. Táboa comparativa	9
2.3. Conclusión	11
3. Tecnoloxía e ferramentas empregadas	13
3.1. Linguaxes	15
3.1.1. Python	15
3.1.2. HTML	16
3.1.3. CSS	17
3.1.4. JavaScript	18
3.1.5. SQL	19
3.1.6. XML	19
3.1.7. JSON	21
3.1.8. LaTeX	22
3.2. Django Framework	22
3.3. Celery	24

ÍNDICE GENERAL

3.4. RabbitMQ	24
3.5. Bootstrap	24
3.6. Ajax	25
3.7. Apache HTTP server	25
3.8. PostgreSQL	26
3.9. Ferramentas de desenvolvimento	27
3.9.1. Eclipse	27
3.9.2. Git	27
3.9.3. Dia	28
3.9.4. Fedora	28
3.9.5. TeXstudio	28
3.9.6. Mozilla Firefox	29
3.9.7. GIMP	29
3.9.8. Projectibre	30
4. Metodoloxía	31
4.1. TDD	32
4.2. XP	33
5. Planificación	35
5.1. Orixe do proxecto	36
5.2. Iteracións	37
5.2.1. Primeira reunión (Iteración 0)	37
5.2.2. Iteración 1	38
5.2.3. Iteración 2	39
5.2.4. Iteración 3	40
5.2.5. Iteración 4	41
5.2.6. Iteración 5	41
5.2.7. Iteración 6	42

ÍNDICE GENERAL

5.2.8. Iteración 7	43
5.2.9. Iteración 8	43
5.2.10. Iteración 9	44
5.2.11. Iteración 10	45
5.2.12. Iteración 11	46
5.3. Estudo de custos	47
5.4. Programación das tarefas	48
6. Análise	53
6.1. Requerimentos funcionais	54
6.2. Requerimentos non funcionais	55
6.2.1. Autenticación	55
6.2.2. Internacionalización	55
6.2.3. Rendimento	56
6.2.4. Seguridade	56
6.2.5. Adaptabilidade a dispositivos móveis	56
7. Deseño	57
7.1. Descripción do funcionamento	58
7.1.1. Actividades de consumo de contidos	58
7.1.2. Actividades de producción de contidos	58
7.2. Arquitectura	59
7.2.1. Capa Modelo	60
7.2.2. Capa Vista	64
7.2.3. Capa Template	67
7.3. Actualización dos datos	72
8. Implementación	75
8.1. Estrutura xeral do código fonte	76

ÍNDICE GENERAL

8.2. Ficheiros de imaxe	77
8.3. A vista de engadir programa	78
8.4. A tradución	79
8.5. O contador de escoitas	80
8.6. O panel de xestión	80
8.7. Execución dos procesos en Celery	82
8.8. Adaptabilidade a dispositivos móbiles	82
9. Probas do sistema	85
9.1. Probas de unidade	86
9.2. Probas de integración	91
10. Conclusiós	95
10.1. Coñecementos acadados	97
10.2. Futuros traballos	97
A. Apéndice: Dicionario de datos	99
B. Apéndice: Licenza	105
B.1. Licenzas das dependencias do proxecto	105
B.2. Conclusión	106

Índice de figuras

1.1.	Ouvintes de radio promedio diario do ano 2017 en España.	3
2.1.	Interface de TuneIn	6
2.2.	Interface de Podomatic	7
2.3.	Interface de Ivoox	8
2.4.	Interface de RadioCo	9
3.1.	Diagrama de interacción de tecnoloxías.	15
3.2.	Exemplo de árbore HTML DOM[1]	17
3.3.	Fragmento de ficheiro RSS real utilizado nas probas.	20
3.4.	Exemplo de sintaxe JSON[2]	21
3.5.	Cota de mercado dos servidores no Top million busiest sites (Netcraft, abril 2018)	26
3.6.	Cota de mercado dos navegadores web. (W3Counter, maio 2018)[3]	29
4.1.	Diagrama do ciclo de desenvolvemento coa metodoloxía TDD.	32
5.1.	Sección de audios da web da URCM que inspira o proxecto.	36
5.2.	It1: Diagrama de clases do primeiro borrador de deseño.	38
5.3.	It2: Interface web. Páxina dun episodio.	39
5.4.	It3: Interface web. Páxina de engadir programa.	40
5.5.	It4: Resultado de consulta á base de datos relacionando programas cos seus tags.	41
5.6.	It6: index.html para usuario identificado.	42
5.7.	It6: index.html para usuario anónimo.	42
5.8.	It8: Vista de detalles de emisora.	44
5.9.	It9: Vista de detalles de episodio.	45
5.10.	It10: Panel de xestión de programa.	46
5.11.	It11: Páxina de resultados de busca.	46

ÍNDICE DE FIGURAS

5.12. Diagrama de Gantt: Novembro 2017 - Decembro.	48
5.13. Diagrama de Gantt: Decembro 2017 - Xaneiro 2018.	48
5.14. Diagrama de Gantt: Xaneiro 2018 - Marzo 2018.	49
5.15. Diagrama de Gantt: Marzo 2018 - Maio 2018.	50
5.16. Diagrama de Gantt: Maio 2018 - Xuño 2018.	51
7.1. Árbore de módulos do aplicativo web.	59
7.2. Diagrama Entidade-Relación da Base de Datos.	61
7.3. Diagrama de clases da capa modelo.	62
7.4. Esquema do funcionamento das vistas.	65
7.5. Patrón Estratexia utilizado para o procesamiento de RSS.	66
7.6. Esquema dun elemento da cuadrícula.	67
7.7. Extracto do borrador do deseño da interface. Portada.	68
7.8. Extracto do borrador do deseño da interface. Detalles de programa.	70
7.9. Extracto do borrador do deseño da interface. Xestión de emisión.	71
7.10. Mapa de navegación entre as distintas páxinas.	72
8.1. Estrutura do código fonte do proxecto.	76
8.2. Formulario de engadir programa.	79
8.3. Fragmento do ficheiro .po para a tradución ao Galego.	80
8.4. Panel de xestión dunha emisora. Vista de xestión da emisión.	81
8.5. Páxina de administración da aplicación, sección de procesos de Celery.	82

Capítulo 1

Introducción

1.1.	Marco do proxecto	2
1.2.	Motivación	3
1.3.	Obxectivos	3

Nos últimos anos, Internet converteuse nunha peza clave para os medios de radiodifusión xa que permite un alcance global e o acceso baixo demanda aos contidos emitidos. Isto é especialmente interesante para as pequenas emisoras locais, a miúdo comunitarias, culturais e con orzamento limitado.

A estas últimas está orientado este proxecto. Consiste nun punto de encontro en liña para promover contidos radiofónicos e favorecer a súa redifusión por parte de distintos medios de comunicación (Emisoras, canles de podcasting...) así coma o seu consumo directo por parte dos visitantes da web. Para o seu desenvolvemento, utilizouse Django 1.11, un framework web de Python, ferramentas HTML5, Javascript e CSS-grid. Tamén se utilizaron ferramentas de sindicación RSS para o acceso aos contidos de terceiros.

Nesta memoria tratarase o proceso completo de desenvolvemento do proxecto desde as fases de análise e deseño até os detalles de implementación. Mencionaranse tamén as liñas de traballo que se pretenden seguir no futuro.

1.1. Marco do proxecto

A radiodifusión tradicional, entendendo esta coma a retransmisión de contidos de audio a través de ondas analóxicas, presenta a día de hoxe unha serie de limitacións. A máis importante, se cadra, é o feito de que a demanda de frecuencias é superior ao que o espectro radioeléctrico pode ofrecer. Cómpre, por iso, a existencia de unha autoridade que outorgue licenzas de emisión sendo ditas institucións, na nosa sociedade, a Secretaría de Estado de Telecomunicaciones y para la Sociedad de la Información e máis a Secretaría Xeral de Medios^[4]. Isto implica a imposibilidade de emitir contidos por parte de aqueles que ou ben non poidan fazer fronte á inversión que unha licenza supón ou ben non lles fose outorgada.

A medida que o acceso a Internet se fai máis cotián, a emisión por streaming eríxese coma solución aos problemas da radio en FM. A través deste medio, unha emisora pode emitir contidos sen necesidade de licenzas, acadando, ademais, unha cobertura global de xeito centralizado, sen necesidade de emitir mediante cadeas de radio enlace, como é costume nas grandes emisoras en FM do Estado Español, coa inversión en infraestruturas que tal cousa require.

Internet permite non só a emisión en directo mediante streaming, senón tamén o acceso a contidos baixo demanda co nacemento do podcasting a mediados da década dos 2000^[5]. A aparición de ferramentas de uso diario que permiten un acceso fácil a estes contidos está a afectar ao comportamento dos usuarios: Actualmente, un 7.5 % dos ouvintes escoitan a

radio por Internet; ánda lonxe dos ouvintes de FM, porén máis do dobre dos de Onda Media[6].

OYENTES DE RADIO PROMEDIO DIARIO								
EGM acumulado 2017	Total Población	Total Oyentes	FM	OM	Total Internet	Directo / Streaming	Diferido / Podcast	TDT
TOTAL OYENTES (000)	39.783	23.605	21.628	725	1.775	1.440	384	476
Penetración %	100,0	59,3	54,4	1,8	4,5	3,6	1,0	1,2
Cuota por onda		100,0	91,6	3,1	7,5	6,1	1,6	2,0

Figura 1.1: Ouvintes de radio promedio diario do ano 2017 en España.

1.2. Motivación

Ao entender que os suxeitos que atopan dificultades para emitir por FM son a miúdo medios do chamado terceiro sector. É dicir: entidades pequenas, comunitarias, sen ánimo de lucro, independentes e con finalidade maioritariamente cultural e social[7]. O impacto positivo na pluralidade informativa e no desenvolvemento da sociedade civil destes medios é explicitamente recoñecido pola UNESCO[8].

Se ben a emisión por streaming e o podcast se perfilan coma a alternativa nun futuro inmediato, tamén veñen acompañados de certa aura de incerteza. Hai que ter en conta que os intereses deste tipo de emisoras adoitan ser de ámbito local. Os eventos que unha radio comunitaria ten capacidade de cubrir rara vez son internacionais e os patrocinios aos que adoitan ter acceso son PEME's da propia localidade na que se atope a emisora.

Deste xeito, a vantaxe de de globalidade que ofrece Internet acaba por non ser tal mentres que, pola contra, a súa visibilidade si queda diluída pola numerosa oferta existente, esta si, a nivel global.

1.3. Obxectivos

Este proxecto pretende servir de axuda aos colectivos do terceiro sector da comunicación, favorecendo a colaboración entre eles e achegándolle as ferramentas necesarias para acadar unha maior presenza na rede.

O producto resultante consistirá nun portal web onde os usuarios poderán acceder a un catálogo de programas. O mencionado catálogo estará constituído por arquivos de son enlazados por outros usuarios que desexen compartirlos desde o seu propio almacenamento.

Os obxectivos a acadar son os seguintes:

Capítulo 1. Introdución

- Facilitar desde un portal web o acceso a ficheiros de audio mediante streaming por Internet e descarga directa desde servidores alleos ao servizo.
- Permitir a distintos usuarios a publicación, manual ou automatizada, do seu contido no sitio web de xeito organizado por programas e categorías.
- Ofrecer aos visitantes ferramentas de procura de contidos e a opción de subscribirse aos diferentes programas ou canais.
- Permitir a colaboración entre usuarios na xestión de contidos publicados.

Capítulo 2

Estado da arte

2.1.	Alternativas Existentes	6
2.1.1.	TuneIn	6
2.1.2.	iTunes	7
2.1.3.	Podomatic	7
2.1.4.	Ivoox	8
2.1.5.	RadioCo	9
2.2.	Táboa comparativa	9
2.3.	Conclusión	11

Neste capítulo, comentaranse os produtos ou ferramentas existentes na actualidade que máis se aproximan aos obxectivos deste proxecto. Existe certa variedade de portais onde se poden poñer arquivos de audio a disposición do público, algunas que tamén permiten o acceso á emisión en directo por streaming, pero ningunha delas cumpre a totalidade das metas a acadar.

2.1. Alternativas Existentes

2.1.1. TuneIn



Figura 2.1: Interface de TuneIn

TuneIn é un portal web que funciona coma un agregador de canles de streaming. Permite aos usuarios a procura de diversas emisoras, incluso de eido local, e acceder á súa emisión en directo. Aínda que non é o seu uso principal, tamén posibilita o acceso a escoita de podcasts.

Este aplicativo non contempla un concepto de “programa” coma tal. Está centrado a redor das emisoras e podemos saber o nome do programa que se está a emitir dependendo da información que o propio streaming proporcione, pero non obter, por exemplo, un listado de programas emitidos por unha emisora. Tampouco favorece a redifusión de programas entre emisoras. Pese a ser unha ferramenta útil, non parece estar pensada para os proxectos de radio comunitaria.

2.1.2. iTunes

iTunes é un software de reproducción multimedia propiedade de Apple Inc. É un aplicativo pioneiro na difusión dos podcasts, comezando a dar soporte a este tipo de emisión no ano 2005[9], e continúa, a día de hoxe, a ser unha das ferramentas máis populares para a escucha e xestión de subscrición de este formato radiofónico.

iTunes tamén ten unha funcionalidade para escutar radio por Internet, porén esta capacidade é limitada pois o catálogo de emisoras dispoñibles redúcese a aquelas posuídas por Apple. Un usuario si podería escutar a emisión por streaming dunha emisora independente, pero queda á súa discreción atopar o enlace de emisión e engadilo á súa biblioteca persoal. Tampouco ofrece ningunha información de emisión entre os programas e as emisoras más aló do que cada produtor poida ter escrito na descripción do seu podcast.

2.1.3. Podomatic

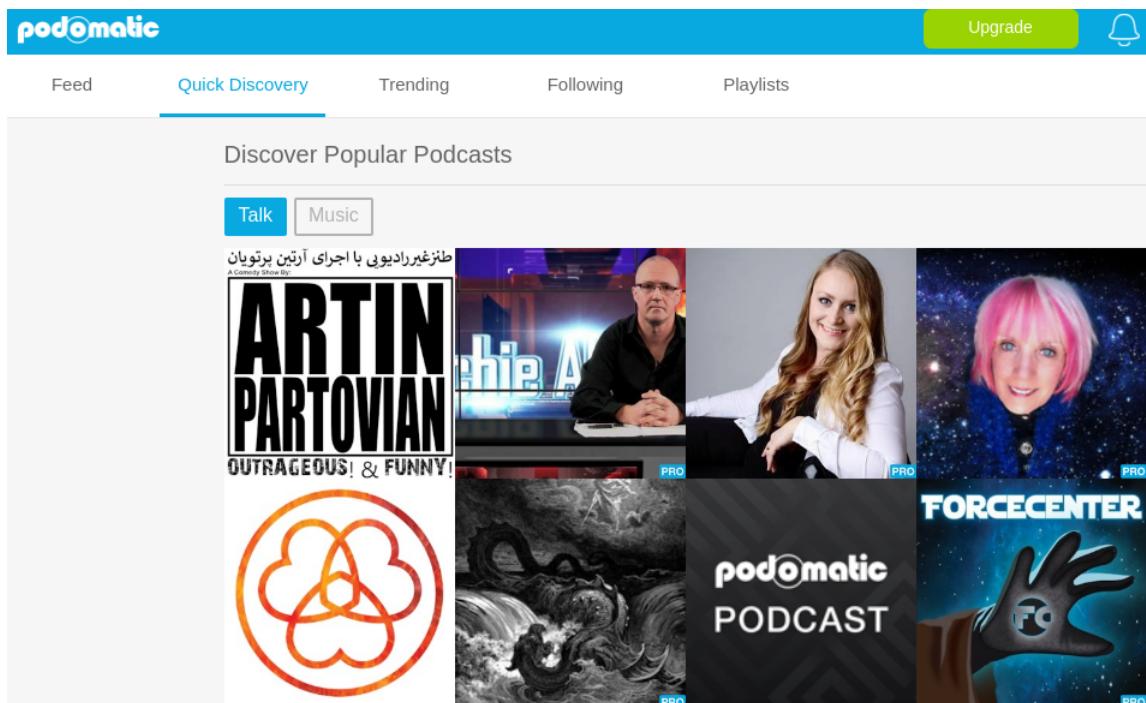


Figura 2.2: Interface de Podomatic

Podomatic é un servizo en liña de hosting de programas de radio e podcast. A diferenza das alternativas vistas con anterioridade, esta proporciona almacenamento propio para os arquivos de audio. O servizo ofrece aos produtores unha páxina de seu para o seu podcast, de xeito que non sería para eles necesario ningunha outra ferramenta para comenzar a

Capítulo 2. Estado da arte

emitir. Aos ouvintes, ofrécelles un completo taboleiro para xestionar as súas subscricións e formar listas de reprodución.

Esta web está íntegramente adicada ao podcasting, polo que non contempla a escoita de emisoras por streaming. Ao existir unha versión de pago, os servizos da versión gratuita están limitados a unha certa visibilidade e a un máximo de episodios publicados.

2.1.4. Ivoox



Figura 2.3: Interface de Ivoox

Ivoox é unha das meirandes comunidades de ouvintes de radio por Internet en lingua castelá. Dá uns servizos aos produtores de contenido e ouvintes semellantes aos de Podomatic (sección 2.1.3) mais, ao estar financiado principalmente por publicidade de terceiros, non presenta as limitacións deste último. Contempla o concepto de emisora de radio, puidendo escoitar a emisión en directo mediante streaming.

Permite, a creación de canles compartidas entre usuarios. Posibilitando o achegamento de produtores distintos a un mesmo proxecto, pero non así a redifusión de contidos entre emisoras, é dicir, non permite o tipo de colaboración requirido polos usuarios obxectivo.

2.1.5. RadioCo



Figura 2.4: Interface de RadioCo

RadioCo é un software libre de xestión da gravación de programas de radio. Encárgase de gravar os programas emitidos pola emisora na que estea instalado e da súa posterior publicación no respectivo podcast do programa. Mantén actualizado un catálogo de programas dispoñibles co seu horarios de emisión en directo ou redifusión en diferido.

O seu uso está orientado a radios comunitarias[10], pero o seu ámbito é de xestión dos contidos dunha emisora e non de ferramenta transversal entre distintos medios.

2.2. Táboa comparativa

A continuación defínense unha serie de obxectivos e amósase unha táboa (táboa 2.1) comparando o alcance das distintas alternativas vistas na sección 2.1.

- Escoita de episodios: Posibilidade de escoitar o ficheiro de audio desde a interface do software.
- Descarga directa: Posibilidade de descargar o ficheiro de audio desde a interface.
- Subscrepción a programas: Opción de manter unha lista de programas escoitados polo usuario e recibir información das novas publicacións destes.

Obxectivos	TuneIn	iTunes	Podomatic	Ivoox	RadioCo	Proxecto
Escoita de episodios	Si	Si	Si	Si	Si	Si
Descarga directa	Non	Si	Limit.	Si	Si	Si
Subscrepción a programas	Si	Si	Si	Si	Si	Si
Hosting de seu	Non	Non	Si	Si	Non	Non
Escoita en directo	Si	Limit.	Non	Si	Si	Si
Publicación de contidos	Limit.	Si	Limit.	Si	Si	Si
Ferramentas de procura	Si	Si	Si	Si	Si	Si
Acceso a distintas emisoras	Si	Limit.	Non	Si	Non	Si
Xerarquía emisora-programa	Non	Non	Non	Si	Non	Si
Redifusión entre emisoras	Non	Non	Non	Non	Non	Si
Xestión colaborativa	Non	Non	Non	Limit.	Si	Si
Código abierto	Non	Non	Non	Non	Si	Si

Táboa 2.1: Comparativa de alternativas fronte a requirimentos.

- Hosting de seu: Posibilidade de aloxar os ficheiros de audio nun almacenamento remoto propio do aplicativo.
- Escoita en directo: Opción de escuchar a emisión actual dunha emisora ou canle mediante streaming.
- Publicación de contidos: Posibilidade dos usuarios de engadir contidos (novos programas, novas emisoras...) á aplicación para que sexan visibles polos demais usuarios.
- Ferramentas de procura: Se o software facilita a busca de contidos de xeito amigable co usuario.
- Acceso a distintas emisoras: Posibilidade de escutar os programas ou a emisión de distintos colectivos desde unha única aplicación.
- Xerarquía emisora-programa: Se a aplicación explicita a relación entre as emisoras e os programas.
- Redifusión entre emisoras: Posibilidade, pola parte dun colectivo, de compartir os seus contidos e de redifundir os mesmos por parte dos demais.

- Xestión colaborativa: Opción de xestión dun mesmo elemento (emisora, programa...) por parte de máis dun usuario.
- Código aberto: Se o aplicativo en cuestión ten o seu código fonte publicado e o seu uso é libre.

2.3. Conclusión

Existen, na actualidade, produtos que cobren algunas das necesidades dos medios radiofónicos comunitarios de cara a Internet e, de feito, para calquera dos exemplos propostos na sección anterior, poderíamos atopar emisoras ou programas que os utilizan, moitas veces de xeito non excluínte. Porén, ningún deles satisfai a totalidade dos obxectivos marcados, en especial no tocante á compartición e redifusión de contidos entre emisoras, peza clave na filosofía comunitaria dos medios do terceiro sector.

É certo que algunas das alternativas propostas ofrece a posibilidade de albergar os ficheiros de audio, cousa fora do alcance deste proxecto. Non obstante, non se considerou esta funcionalidade coma algo crítico á hora de definir os obxectivos pois a pretensión non é substituír por completo os sistemas de acceso aos contidos que os usuarios xa posúan, senón complementalos.

Na actualidade, para as federacións nas que se adoitan agrupar os citados medios, non existe ningún software que lles dea a posibilidade de manter unha páxina de referencia de programas dispoñibles a modo de catálogo, unha páxina centralizada e privada para esa federación na que os pequenos proxectos puidesen acadar unha visibilidade maior da a que terían en portais globais coma, por exemplo, os antes mencionados iTunes ou Ivoox.

Capítulo 3

Tecnoloxía e ferramentas empregadas

3.1. Linguaxes	15
3.1.1. Python	15
3.1.1.1. Anaconda	16
3.1.2. HTML	16
3.1.3. CSS	17
3.1.3.1. CSS Grid	17
3.1.4. JavaScript	18
3.1.4.1. jQuery	18
3.1.5. SQL	19
3.1.6. XML	19
3.1.6.1. RSS	20
3.1.7. JSON	21
3.1.8. LaTeX	22
3.2. Django Framework	22
3.3. Celery	24
3.4. RabbitMQ	24
3.5. Bootstrap	24
3.6. Ajax	25
3.7. Apache HTTP server	25
3.8. PostgreSQL	26
3.9. Ferramentas de desenvolvimento	27
3.9.1. Eclipse	27
3.9.2. Git	27
3.9.2.1. GitHub	28
3.9.3. Dia	28
3.9.4. Fedora	28
3.9.5. TeXstudio	28
3.9.6. Mozilla Firefox	29

Capítulo 3. Tecnoloxía e ferramentas empregadas

3.9.7. GIMP	29
3.9.8. Projectibre	30

Neste capítulo, presentanse as linguaxes, ferramentas e frameworks utilizados no desenvolvemento deste proxecto. Á hora de elixir o uso destas ferramentas valorouse a súa natureza de código aberto xa que se pretende que o resultado dispoña dunha licenza de software libre compatible coa definición da Free Software Foundation.

Tamén se valoraron, dado que nos capítulos anteriores se insistiu na importancia dos dispositivos móveis, no consumo da radio a través de Internet, as posibilidades de ditas ferramentas de ofrecer un bo resultado en distintos dispositivos e distintas plataformas software.

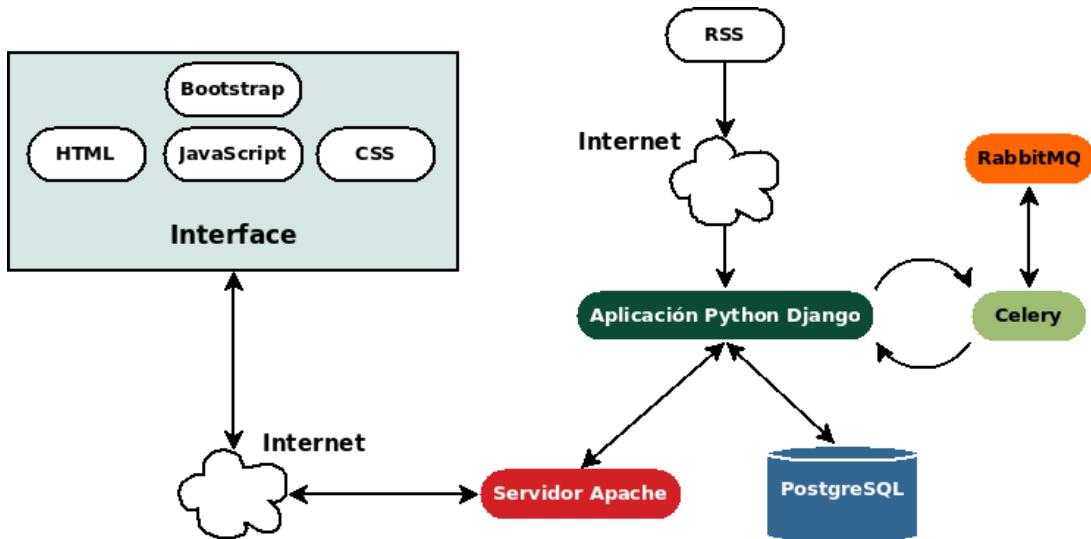


Figura 3.1: Diagrama de interacción de tecnoloxías.

3.1. Linguaxes

3.1.1. Python

Python é unha linguaxe de propósito xeral de alto nivel. Trátase dunha linguaxe interpretada polo que é necesario ter un intérprete para executar o código. Ao ser o intérprete unha capa intermedia de software entre o programa e o sistema, Python é unha linguaxe doadamente portable entre dispositivos de distinta natureza.

A súa sintaxe baseada na indentación está pensada para favorecer a comprensión entre distintos desenvolvedores e facilitar o mantemento do código.

O seu sistema de tipado implícito e a súa riqueza de bibliotecas para executar comandos de consola de xeito programático fan que sexa moitas veces descrito coma “unha linguaxe

de scripting orientada a obxectos”[11], o cal serve coma mostra da súa flexibilidade, un dos motivos polo que foi elixida para este proxecto.

3.1.1.1. Anaconda

Anaconda é unha distribución de Python, libre e de código aberto (licenza BSD), utilizada normalmente para análise estatística e machine learning. Inclúe, ademais dunha completa instalación do intérprete de Python, un rico conxunto de librarías de uso común e o xestor de paquetes conda, sendo esas dúas últimas melloras o motivo primordial polo que foi a elección para este proxecto. Conda utiliza os paquetes da comunidade de Conda Forge [12]

Anaconda é responsabilidade de Anaconda Incorporated, antigo Continuum Analytics.

A versión utilizada foi a Anaconda 4.4.0 de 64 bits para Python 3, a máis nova no momento de comezar o traballo. Inclúe o intérprete para Python versión 3.6.1.

3.1.2. HTML

HTML (abreviación de HyperText Markup Language) é a linguaxe estándar para a creación da estrutura das páxinas web. Os elementos presentes declaranse mediante bloques representados por etiquetas(tags), de aí que reciba o nome de “markup language” (lingua-xe de marcado). Estas etiquetas son interpretadas polos navegadores para renderizar os contidos[13].

A estrutura declarada no código HTML pode ser representada pola interface de DOM (Document Object Model), como se amosa na figura 3.2. O DOM é unha árbore creada polo navegador ao cargar a páxina e é moi útil para acceder aos elementos da páxina de xeito programático, como veremos máis adiante.

Dada a natureza multimedia da web realizada, utilizouse HTML5. Esta quinta revisión do estándar inclúe novas etiquetas para o tratamento das imaxes e dos contidos de audio e vídeo sen necesidade de utilizar outras tecnoloxías complementarias, simplificando así o desenvolvemento e o mantemento posterior do portal. Desde decembro do pasado ano 2017, a versión 5.2 é a última estable recomendada polo World Wide Web Consortium(W3C)[14].

Ao ser o uso de HTML un estándar tan lonxevo, consolidado e popular, non se consideraron alternativas para este proxecto.

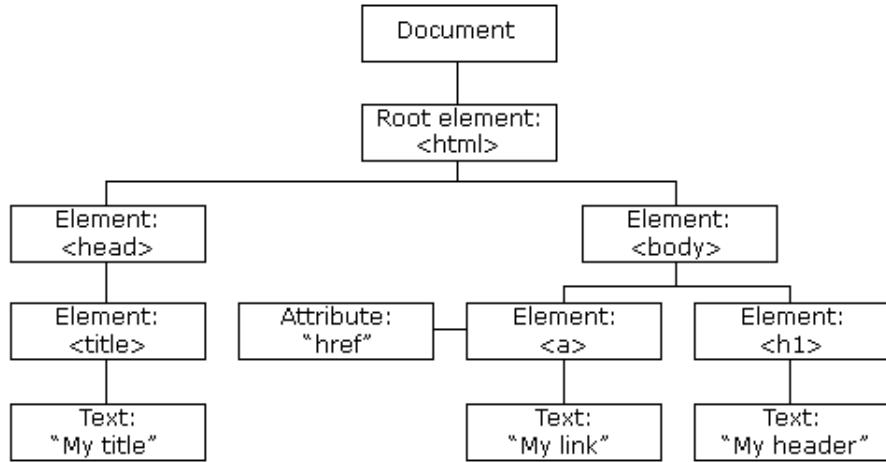


Figura 3.2: Exemplo de árbore HTML DOM[1]

3.1.3. CSS

CSS (Abreviatura de Cascading Style Sheets) ou “Follas de estilo” é unha linguaxe empregada para definir a estética dos elementos definidos anteriormente no código HTML: A súa cor, fonte, posición... Ao separar o estilo da estrutura, favorécese a reutilización de código xa que un mesmo ficheiro de CSS pode ser utilizado en diversas páxinas ao mesmo tempo. CSS permite tamén adaptar o contido das páxinas a dispositivos de distinto tamaño[15].

3.1.3.1. CSS Grid

O módulo de “Grid” úsase para definir un deseño de interface gráfica consistente nunha cuadrícula de dúas dimensións en CSS. Nun modelo deste tipo, declárarse un conxunto de elementos no HTML coma pertencentes a un “grid container” (contedor da grella ou da cuadrícula) e, á súa vez, zonas fillas que tomarán posición dentro dese contedor dependendo das características coas que este último fose definido na folla de estilos.

Unha cela nun contedor pode, á súa vez, consistir nun contedor en sí mesma, permitindo así deseños asimétricos. O tamaño das celas pode ser fixo, relativo á páxina ou automático dependendo do contido da cela. Por todo o dito, este módulo proporciona un nivel de flexibilidade superior ao que poderíamos obter utilizando outras alternativas populares coma CSS Flexbox.

O nivel primeiro de CSS Grid non acadou aínda, na data de entrega desta memoria, o status de “recomendación” da World Wide Web Consortium(W3C)[16], porén, xa é sopor-

tado polas últimas versións dos navegadores más populares[17] polo que non se considerou un risco utilizalo neste proxecto.

3.1.4. JavaScript

JavaScript é unha linguaxe de scripting interpretada e de alto nivel. Utilízase principalmente (e tamén neste proxecto) coma unha ferramenta para mellorar a interacción coa interface web ao permitir executar código orientado a eventos no lado do cliente, aforrando recargas da páxina e incluso accesos innecesarios á base de datos xa que permite o uso do disco local mediante cookies.

A pesares do seu nome, non garda relación algunha coa lingüxe Java e serven a propósitos claramente distintos. O núcleo desta linguaxe está regulado polo estándar ECMAScript®, na súa 8^a versión[18] no momento de entregar esta memoria.

Como se comentou no apartado 3.1.1, que sexa interpretada implica a necesidade dun intérprete para executar o código. Ese intérprete, comunmente chamado JavaScript Engine preséntase embebido nos navegadores web. Non obstante, non existe unha única implementación senón que distintos navegadores presentan distintas versións. Para o desenvolvemento deste proxecto utilizáronse os navegadores Mozilla Firefox 57.0.1 e Google Chrome 66.0, cuxos motores de JavaScript son SpiderMonkey e V8 respectivamente[19], ambos os dous libres e de código aberto.

Pese a que o seu uso nos navegadores continúa a ser a principal razón de ser desta linguaxe, tamén se utiliza a día de hoxe noutro tipo de produtos coma Node.js(para correr JavaScript no lado do servidor) ou Apache Couch DB (Para o manexo de bases de datos na nube)[20]

Ademais do código en JavaScript puro, este proxecto tamén fai uso nalgunhas partes do código de funcións de jQuery.

3.1.4.1. jQuery

JQuery é unha biblioteca de JavaScript creada co fin de simplificar o “scripting” no lado do cliente. Trátase dunha ferramenta libre, publicada baixo licenza MIT e é mantida pola comunidade jQuery Team. No 2015, xa era empregada polo 63 % do Top Million Websites[21], incluíndo sitios populares coma Netflix, Amazon ou Microsoft. O seu uso segue a ser moi xeralizado a día de hoxe.

Cunha API funcional e válida en gran parte dos navegadores web, evita moitos dos problemas de incompatibilidade que aparecen ao utilizar JavaScript puro e o código resultante

adoita ser moito máis conciso, mellorando a súa comprensibilidade e o seu mantemento. Isto débese en grande medida pola súa expresión “selector” que facilita o acceso aos distintos elementos da árbore do HTML DOM (ver figura 3.2).

A versión de jQuery utilizada foi a 2.1.4 por ser a más actual incluída na biblioteca django-static-jquery-2.1.4[22] do framework utilizado (ver sección 3.2) ao comezo do desenvolvemento.

3.1.5. SQL

SQL é a linguaxe utilizada para a manipulación dos datos dentro do eido das bases de datos relacionais. Nace coma un refinamento ou “secuela” (SQL é unha simplificación da verba inglesa “sequel”) da linguaxe SQUARE, que á súa vez consistía nunha simplificación da linguaxe DSI/Alpha proposta polo mesmo Edgar F. Codd no momento de presentar o modelo relacional de Bases de Datos. O primeiro estándar foi publicado no ano 1986 pola American National Standards Institute (ANSI)[23]. Permite definir a estruturación dos datos, inserilos, eliminálos, editalos e, por suposto, consultalos.

Neste proxecto, o seu uso explícito correspondeuse so coas primeiras fases do traballo debido á utilización do framework Django coma se detalla no apartado 3.2. É necesario especificar tamén que, pese á existencia dun estándar, existen distintas versións desta linguaxe que a fan “non completamente portable” entre distintos sistemas de xestión de bases de datos[24]. Neste proxecto utilizouse a variante correspondente a PostgreSQL.

3.1.6. XML

XML é a abreviatura de “eXtensible Markup Language” (Linguaxe de marcado extensible). Consiste nunha serie de normas que dividen un documento en distintas partes, asignándolle unha identidade a cada unha delas. A diferencia doutras linguaxes de marcado coma o HTML, non existe un conxunto de etiquetas válido. No XML queda á responsabilidade do autor do documento establecer as etiquetas necesarias dependendo do entorno no que ese documento vaia ser utilizado. Por suposto, existen unha serie de normas estruturais para que o etiquetado se poida considerar coma válido, pero desde o punto semántico, a natureza destas etiquetas é moi flexible. Isto fai que moitas veces se denomine o XML coma unha Meta-Linguaxe, é dicir, unha linguaxe para definir linguaxes[25].

A función de XML é definir a semántica e a estrutura dun documento, pero nunca o estilo. Ao igual que HTML, pódese asociar unha folla de estilos CSS.

Neste proxecto, XML é utilizado na súa modalidade RSS do xeito detallado no apartado 3.1.6.1.

3.1.6.1. RSS

```
- <rss version="2.0">
  - <channel>
    <itunes:author>Fer Lee</itunes:author>
    <itunes:explicit>yes</itunes:explicit>
    <title>Hasta Los Kinders</title>
    - <link>
      http://www.ivoox.com/podcast-hasta-los-kinders_sq_f14062_1.html
    </link>
    - <description>
      Legendario programa de humor anárquico que se emitió durante 2 temporadas
      (2006/2007-2007/2008) en Cuac fm 103.4, la radio comunitaria de A Coruña. Javier Casariego
      y Fer Liñares, con la colaboración de Estíbaliz Iglesias y Gabriel Rodríguez. Gavín y DJ
      Vázquez en el control técnico.
    </description>
    <language>es-ES</language>
    <generator>iVoox</generator>
    - <image>
      - <url>
        http://static-1.ivoox.com/canales/6/2/8/5/6431470915826_XXL.jpg
      </url>
      <title>Hasta Los Kinders</title>
    - <link>
      http://www.ivoox.com/podcast-hasta-los-kinders_sq_f14062_1.html
    </link>
  </image>
  <atom:link href="http://www.ivoox.com/hasta-los-kinders_fg_f14062_filtro_1.xml" rel="self"
  type="application/rss+xml"/>
  <itunes:image href="http://static-1.ivoox.com/canales/6/2/8/5/6431470915826_XXL.jpg"/>
  <itunes:explicit>yes</itunes:explicit>
  <itunes:type>episodic</itunes:type>
  <itunes:category text="Comedy"/>
```

Figura 3.3: Fragmento de ficheiro RSS real utilizado nas probas.

RSS é un tipo de formato XML utilizado para a “sindicación” web, isto é, a emisión de contidos actualizados dunha páxina web a un número indefinido de subscriptores. Utilízase para evitar a procura manual de novos contidos naquelas páxinas nas que a actualización é frecuente: Páxinas de novas, blogs, podcasts...

O seu funcionamento consiste nun ficheiro RSS (comunmente chamado “feed”) accesible desde a web polos programas clientes de rss que posúan os subscriptores. Ese feed mostra información resumida do contido publicado no sitio web ao que fai referencia. Cada vez que o contido se actualice, o feed actualizarase tamén e ese cambio será detectado polo cliente rss na súa seguinte comprobación mediante “polling”.

O nome procede do acrónimo de “Really Simple Syndication” (Sindicación Realmente Sinxela) e o estándar é mantido polo “RSS Advisory Board”. Na data de entrega desta memoria e desde o ano 2009, a última revisión é a 2.0.11 [26]

Neste proxecto, o interese céntrase na actualización dos podcasts. Existen alternativas a este formato de sindicación mais, dado o seu uso xeralizado entre os usuarios potenciais da aplicación web a desenvolver, non foron consideradas.

3.1.7. JSON

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "isAlive": true,  
    "age": 27,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": "10021-3100"  
    },  
    "phoneNumbers": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "office",  
            "number": "646 555-4567"  
        },  
        {  
            "type": "mobile",  
            "number": "123 456-7890"  
        }  
    ],  
    "children": [],  
    "spouse": null  
}
```

Figura 3.4: Exemplo de sintaxe JSON[2]

JSON é un estándar de sintáctico utilizado para o intercambio de datos lixeiros de texto entre distintas linguaxes. A pesares de ser a abreviatura de JavaScript Object Notation pola súa orixe inspirada nos tipos desta linguaxe[27] o seu uso por parte doutras tecnoloxías é común. Esta linguaxe non define un sistema completo de intercambio de datos, mais si un marco sintáctico sobre o que definir un.

Baséase nunha xerarquía definida por chaves, corchetes, comas, dous puntos e parénteses no xeito descrito na figura 3.4, onde se amosa un exemplo de codificación dos datos persoais dun home.

Neste proxecto, utilizouse de forma implícita ao ser a codificación utilizada polos obxectos de contexto de Django.

3.1.8. LaTeX

LaTeX é un sistema de preparación de documentos para os que sexa necesaria unha tipografía de alta cualidade, habitualmente, documentos medios ou longos para publicacións científicas. A motivación de LaTeX é a máxima separación posible entre o contido e o estilo do documento, facendo depender o segundo de modelos preexistentes para que os autores podan concentrarse na creación do primeiro.

Este sistema non é un editor de texto. Consiste nun conxunto de macros e un programa procesador que interpreta os mesmos. É un software libre e de código aberto (publicado baixo licenza LaTeX Project Public License[28]). Na actualidade, pódese escoller entre distintas distribucións libres de LaTeX que inclúen unha ampla variedade de paquetes de macros adicionais. Para a confección desta memoria, utilizouse TexLive na súa versión de 2016 por ser a máis actual dispoñible nos repositorios do Sistema Operativo no momento de comezar a escritura.

3.2. Django Framework

O proxecto Django nace no ano 2005 coma un conxunto de ferramentas Python para o desenvolvemento web publicadas baixo licencia BSD. A motivación dos seus creadores, Simon Wilson e Adrian Holovaty, naquel tempo programadores nun medio periodístico, era a homoxeneización e a reutilización de código. Isto daba resposta á necesidade de reducir o alto custo que supón manter código ad-hoc para cada novo artigo ou función, especialmente nunha páxina que requira unha constante actualización de contidos coma a dun diario en liña. É por iso que se adoita utilizar o termo “newsroom schedule” cando se fala da rapidez de desenvolvemento que permite o framework[29].

Django ofrece un conxunto de bibliotecas que dan soporte ás mecánicas máis comúns do desenvolvemento web:

- Persistencia: Django permite crear a estrutura da base de datos sen necesidade de escribir SQL. Isto conséguese mediante a superclase Model da librería “db” de Django. Ao declarar unha clase de Python coma extensión de “Model”, a biblioteca ocúpase automaticamente de manter a coherencia entre as súas instancias en memoria e as táboas da base de datos.
- Peticións web: A lectura e a interpretación destas peticións son levadas a cabo polo conxunto de bibliotecas de HTTP do framework, encapsulando as peticións e as

respostas en obxectos para formar un estándar sinxelo e garantir a correcta formación das mesmas.

- Enrutamento e Validación: O framework ofrece un estándar para asignar as distintas vistas definidas no código ás URL's desexadas. Os posibles formularios presentes nesas vistas poden ser abstraídos en obxectos de Python simplificando a validación dos datos introducidos polos usuarios. A utilización destes sistemas é non obstante, opcional, podendo o programador optar por utilizar sinxelos formularios HTML no caso de que isto axilizase o desenvolvemento.
- Sistema para embeber datos dinámicos no HTML: O chamado “template system” de Django marca uns procedementos sinxelos para acceder aos datos xerados no código. Tamén ofrece ferramentas para implementar certa lóxica na presentación do front end.
- Interface de administración nativa: Por defecto, este framework dispón dun panel de administración de acceso reservado aos superusuários que evita os accesos directos á base de datos no caso de que un administrador necesite facer cambios manuais nos datos.
- Soporte por defecto para a autenticación e autorización: Django permite ao desenvolvedor invocar a clase nativa Usuario e efectuar con ela accións de rexistro, autenticación e concesión/revogación de permisos sen máis programación.

Para este proxecto utilizouse concretamente Django 1.11, a versión estable máis avanzada no tempo en que se comezou o traballo. Django sufriu unha actualización importante desde entón que rematou coa publicación de Django 2.0 en Decembro do pasado ano 2017. Estas dúas versións non son totalmente compatibles de modo que unha actualización implicaría cambios no código. Tendo en conta que a Django Software Foundation, responsable do mantemento do framework, non prevee retirar o soporte á versión utilizada nun futuro próximo e que a versión de Python utilizada é soportada tamén por Django 2[30], considerouse aceptable o grao de actualidade da tecnoloxía utilizada.

Ademais do mencionado anteriormente, confiouuse en Django para este proxecto pola súa madurez e pola súa popularidade, sendo utilizado en sitios coma Instagram, Disqus, Pinterest ou Open Knowledge Foundation.

3.3. Celery

Celery é unha cola de traballos asíncrona baseada na mensaxería distribuída (distributed message passing). Utilízase para distribuír traballos entre máquinas ou fíos. Celery levanta un conxunto de procesos chamados “workers” nos que se executarán de xeito concorrente os diferentes traballos ou “tasks”. Eses workers consultan constantemente a cola de traballos para executar os traballos que podan estar pendentes[31].

Neste proxecto, utilizouse Celery para controlar a execución dos demos que corren no lado do servidor. Eses dous demos son: O encargado de actualizar a información do programa e os episodios e o encargado de calcular a popularidade dos programas. Utilizouse a versión de Celery 4.1, a máis actual no repositorio de Anaconda no momento de comezar o traballo.

Django ofrece bibliotecas para traballar con Celery de xeito máis sinxelo, por exemplo, permitíndolle gardar información na base de datos do proxecto e engadindo ferramentas de administración dos procesos de Celery ao menú de administración de Django. Utilizáronse as bibliotecas django_celery_results 1.0.1 e django-celery-beat 1.1.1

Como xa se mencionou, Celery baséase na mensaxería, mais non inclúe un sistema de seu senón que lle hai que proporcionar un. Neste caso, optouse pola utilización de RabbitMQ (ver sección 3.4), un dos recomendados na documentación de Celery.

3.4. RabbitMQ

RabbitMQ é un software de “message passing” tipo “broker” mensaxes. Isto é, un sistema onde diferentes aplicacións se conectan ao broker coa fin de enviar a ou lelos deste, funcionando o dito broker coma unha cola. As mensaxes enviadas á cola por un aplicativo permanecerán aí até que sexan lidos por outro aplicativo. Unha mensaxe pode ser calquera cousa, desde meta información dos procesos até texto plano[32].

Neste proxecto utilizouse a versión 3.6.10-1 por seres esta a más actual dispoñible nos repositorios do Sistema Operativo no momento de comezar o traballo.

3.5. Bootstrap

Bootstrap é un framework libre e de código aberto (publicado baixo licenza MIT) para a construción de interfaces web. O Framework ofrece unha serie de modelos ou “templa-

tes”, cada unha cunha estrutura HTML, declaracíons en CSS e, nalgúns casos, extensíons JavaScript.

Neste proxecto, utilizouse a través da biblioteca de django-bootstrap4 para facilitar o uso dos elementos da versión 4 de Bootstrap desde os templates do framework de Django.

3.6. Ajax

Ajax é a abreviación de “Asynchronous JavaScript and XML”. Trátase dunha técnica combinada desas tecnoloxías, aínda que a miúdo substituíndo XML por JSON[33], utilizado para manter unha comunicación entre o lado cliente e o lado servidor nunha páxina web sen necesidade dunha recarga completa.

Utilizando Ajax pódese facer unha petición ao servidor e recibir información de xeito asíncrono mediante JSON ou XML para, mediante a manipulación do HTML DOM con JavaScript, actualizar só partes da páxina amosada ao usuario. Utilízase para diminuir o tráfico entre os dous lados, facendo a navegación máis ágil e mellorando a interactividade.

Nesta aplicación utilizouse a través dos métodos definidos por jQuery (ver apartado 3.1.4.1)

3.7. Apache HTTP server

Apache é un proxecto colaborativo para o desenvolvemento dun servidor HTTP robusto, con cualidade suficiente para o seu uso comercial, libre e de código aberto (publicado baixo licenza Apache, versión 2.0[34]). Apache implementa o protocolo HTTP/1.1 (RFC2616) xunto con varias funcionalidades frecuentemente requiridas pola web coma a personalización das mensaxes de erro (incluso mensaxes que conteñan de CGI scripts), a posibilidade de redirección e declaración de “alias” para as URLs de xeito ilimitado ou tamén soporte para hosts virtuais.

A pesar de que o seu dominio coma servidor máis utilizado decae nos últimos anos, segue a ser a opción libre máis popular, estando presente nun 36.34 % dos sitios web pertencentes ao “Top million busiest sites” segundo datos do mes de abril do presente ano 2018 (ver figura 3.5). Na actualidade é utilizado en sitios ben coñecidos coma o a páxina de novas da BBC, o sistema de pago PayPal ou a tenda de videoxogos en liña Steam[35]. Esta popularidade é a razón principal pola que se utilizou este software no proxecto.

A versión utilizada foi Apache 2.4.27 por ser a más actual no momento de comezar o traballo. Foi necesaria tamén a instalación das seguintes extensíons:

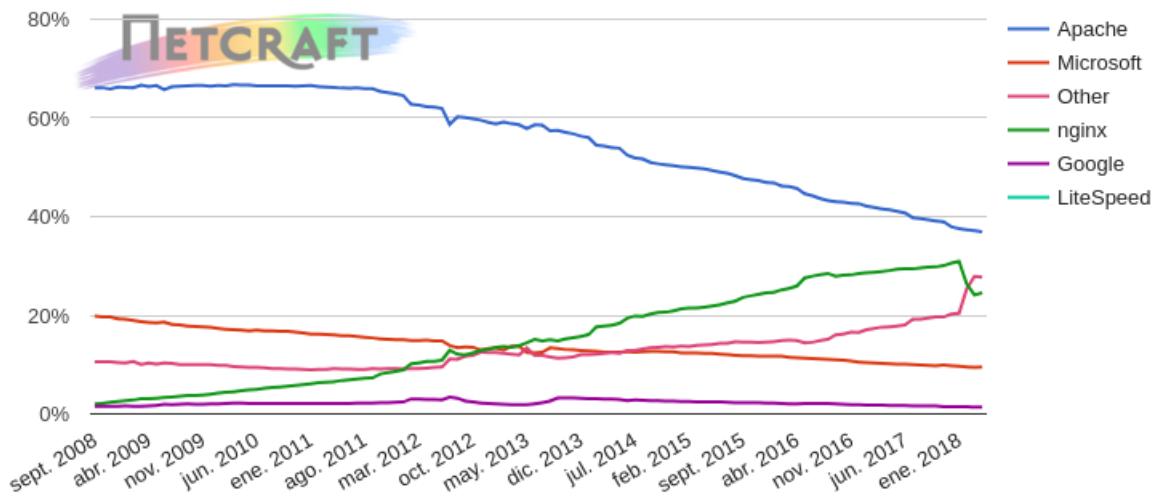


Figura 3.5: Cota de mercado dos servidores no Top million busiest sites (Netcraft, abril 2018)

- Apache Portable Runtime (APR): Biblioteca que inclúe unha serie de API's para garantir o funcionamento de Apache sexa cal sexa a plataforma na que se execute. Utilizouse a versión 1.6.2 e a 1.6.0 da biblioteca complementaria APR-util.
- mod_wsgi: Módulo de Apache que poporciona unha WSGI (Web Server Gateway Interface), necesaria para albergar aplicativos web baseados en Python. Utilizouse a versión 4.5.17.

3.8. PostgreSQL

PostgreSQL é un sistema de xestión de bases de datos (SXBD) relacionais libre e de código aberto (Licenza PostgreSQL, permisiva) desenvolvido polo PostgreSQL Global Development Group. A linguaxe SQL soportada pretende ser o máis fiel posible ao estándar e as súas operacións cumpren coas regras ACID (Acrónimo inglés para Atomicidade, Consistencia, Illamento e Durabilidade)[36]. A súa orixe remóntase a 1996, polo que existe unha grande disponibilidade de recursos de documentación. Ese feito unido á certa experiencia persoal no seu uso xa antes de comenzar o proxecto foron valorados á hora de elixir este sistema.

A versión utilizada é a 9.6.3 por ser a máis actual no momento de comenzar o traballo. Para facer posible o acceso mediante Python, utilizouse o driver psycopg2 na súa versión 2.7.1

3.9. Ferramentas de desenvolvemento

3.9.1. Eclipse

Eclipse é un IDE(Integrated Development Environment ou Entorno de Desenvolvemento Integrado), isto é, un aplicativo consistente nunha serie de ferramentas de desenvolvemento de software dispoñibles arredor dun editor de texto tales coma, por exemplo, unha consola de saída estándar e ferramentas para compilar e depurar código desde a interface proporcionada.

É principalmente empregado nos proxectos de Java, C e C++, porén, dada a súa popularidade, existen extensións dispoñibles no seu propio “marketpace” que nos permiten utilizalo para outras linguaxes. No caso deste proxecto, ese plugin é PyDev, que proporciona as ferramentas necesarias para o desenvolvemento non só de proxectos de Python en xeral senón especificamente de Django.

Outra extensión utilizada foi EGit na súa versión 4.8. Esta serve para manexar o control de versións desde a interface gráfica de Eclipse. Egit utiliza JGit, unha implementación puramente Java do sistema Git (ver apartado 3.9.2).

Optouse pola versión de Eclipse máis nova no momento de comezar o traballo pese a que áinda non estaba dispoñible daquela nos repositorios do sistema operativo do equipo de desenvolvemento: Eclipse Oxygen 4.7.0. A versión de PyDev utilizada foi a 5.9.2. Valorouse PyCharm, un IDE específico para Python, coma alternativa; mais a versión de comunidade non tiña soporte para desenvolvemento específico de Django[37].

3.9.2. Git

Git é un sistema de control de versións libre e de código aberto (licenza GPL v2). Un sistema de control de versións utilízase para gardar os diferentes estados (versións) do código nas distintas fases de desenvolvemento. Permite a creación de ramas para a división do traballo e de etiquetas para marcar certos hitos no proceso. Estas versións pódense gardar no equipo local ou nun repositorio remoto. No caso de Git, mantéñense copias tanto no repositorio coma nos equipos dos programadores, polo cal adoita cualificarse coma un sistema de control de versións distribuído[38].

Como se mencionou no apartado 3.9.1, utilizouse sobre todo na súa implementación de EGit, mais as veces recorreuse ao paquete de sistema para realizar operacións sen necesidade de acceder a Eclipse. Neses casos, utilizouse a versión 2.9.5.

Existen outros sistemas coma, por exemplo, Apache Subversion, pero elixiuse Git pola posibilidade de publicar o código en GitHub

3.9.2.1. GitHub

GitHub é un servizo de hosting web para repositorios de control de versións. É moi popular para o desenvolvemento colaborativo de proxectos de código aberto. Neste proxecto, utilizouse ata o momento coma alternativa gratuita para obter un repositorio en liña pero, idealmente, tamén será utilizado para colaborar con outros desenvolvedores no futuro.

3.9.3. Dia

Dia é un editor de Diagramas libre e de código aberto(Licenza GPL). Utilizáronse neste proxecto as súas extensións de UML para o diagrama de clases do sistema e a de Entidade Relación para o esquema de deseño da base de datos. A versión utilizada foi a 0.97.3.

Consideráronse outros editores coma, por exemplo, Umbrello, pero a elección foi DIA por criterios de estética e comodidade de uso debido á súa interface sinxela.

3.9.4. Fedora

Fedora é un Sistema Operativo que utiliza o kernel de Linux. É desenvolvido pola comunidade do Fedora Project, patrocinada desde os seus inicios por Red Hat Incorporated. Se ben as súas distintas compoñentes non están publicadas baixo a mesma licenza, estas si se enmarcan na definición de “Licenza de Software Libre” da FSF (Free Software Foundation) a excepción dalgúns ficheiros de firmware que, por especificación dos fabricantes de hardware, teñan que estar presentes únicamente coma archivos binarios. Nese último caso, han de cumplimentar unha serie de requisitos coma estar libres de pago polo seu dereito de uso[39]. Podemos entender este, polo tanto, coma un Sistema Operativo libre.

Todo o desenvolvemento deste proxecto así coma a escritura desta memoria realizouse sobre Fedora 25 (64 bits), utilizando Xfce 4.12 coma entorno de escritorio.

3.9.5. TeXstudio

TeXstudio é un editor de documentos LaTex libre e de código aberto (Licenza GPL v2[40]). Proporciona, entre outras funcionalidades, un editor gráfico con soporte de marcado ortográfico para distintos idiomas, un visor de ficheiros PDF integrado e un corrector

de referencias. Para este proxecto, utilizouse a versión 2.12.6 co paquete ortográfico de lingua galega de Libre Office v13-10.

Trátase unicamente dun editor, non inclúe un procesador de macros de LaTex. Utilizouse para isto a distribución TeX Live 2016 (ver sección 3.1.8) e, para a exportación da memoria a formato PDF, pdfTex na súa versión 3.14159265 (a incluída no paquete).

3.9.6. Mozilla Firefox

Firefox é un navegador web libre e de código aberto (Licenza Mozilla Public License v2[41]). Soporta unha grande variedade de estándares e está disponible en gran variedade de sistemas operativos, incluíndo os dos dispositivos móveis más populares: iOS e Android. Aínda que a súa popularidade foi en retroceso nos últimos anos, segue a ser un dos navegadores más populares e o máis utilizado entre as alternativas de software libre (ver figura 3.6).

Valorouse, á hora de decidir utilizarlo, a súa importancia no mundo do software libre, a súa disponibilidade de serie no Sistema Operativo utilizado e as súas ferramentas de desenvolvemento, incluído o seu modo de deseño “responsive”. Utilizouse a versión 57.0.1 de 64 bits, a máis actual disponible nos repositorios de Fedora 25.

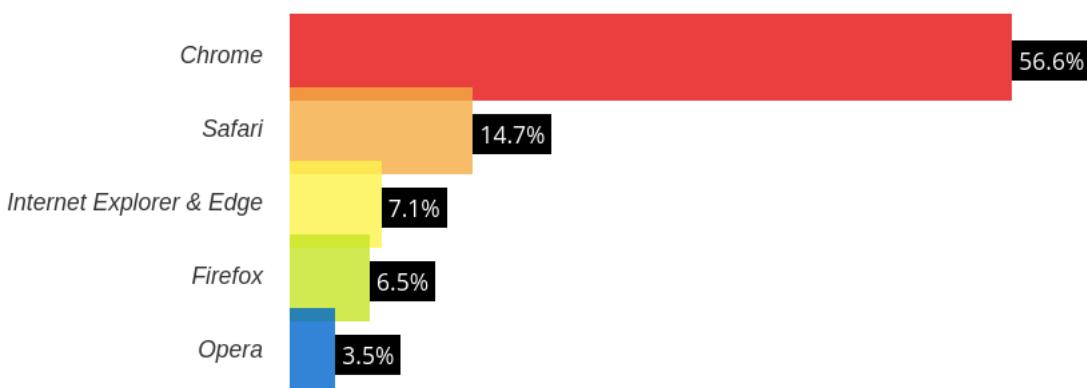


Figura 3.6: Cota de mercado dos navegadores web. (W3Counter, maio 2018)[3]

3.9.7. GIMP

GIMP é a abreviatura de GNU Image Manipulation Program (Programa de manipulación de imaxes de GNU). É un software de edición de imaxes libre e de código aberto (Licenza GPL v3[42]). Neste proxecto utilizouse para facer postprocesados sinxelos das imaxes incluídas na memoria. A versión utilizada foi a 2.8.22 por ser a más actual nos repositorios do Sistema Operativo no momento de comenzar a escritura da memoria.

3.9.8. Projectibre

Projectibre é unha ferramenta de xestión e dirección de proxectos, libre e de código aberto (licenza CPAL[43]). Naceu coma alternativa libre a Microsoft Project, sendo compatible con este. Ofrece diversas funcionalidades coma os diagramas de Gantt, histogramas de recursos ou esquemas RBS (Resource Breakdown Structure)

Capítulo 4

Metodoloxía

4.1. TDD	32
4.2. XP	33

Este capítulo pretende explicar as metodoloxías aplicadas durante as diferentes fases do traballo. Nas primeiras etapas seguiuse a metodoloxía TDD (Test Driven Development) e máis adiante optouse por unha metodoloxía XP (Extreme Programming) aínda que, ao ser este un traballo individual, non se aplicaron todas as súas características. Ámbalas dúas poden considerarse metodoloxías áxiles de desenvolvemento iterativo e incremental.

4.1. TDD

Test Driven Development é unha metodoloxía ágil de desenvolvemento que segue a filosofía “probar primeiro”. Esta baséase na conversión dos requisitos en probas antes da propia existencia de código que as avale. Isto obriga ao desenvolvedor a pensar nos requisitos con detemento, identificar de antemán os posibles fallos e aumentar o alcance da validación de datos. Unha vez o test está definido, pasaremos a implementación da funcionalidade posta a prueba. De pasar a prueba, repítese co seguinte requisito. Este ciclo pode verse no esquema da figura 4.1.

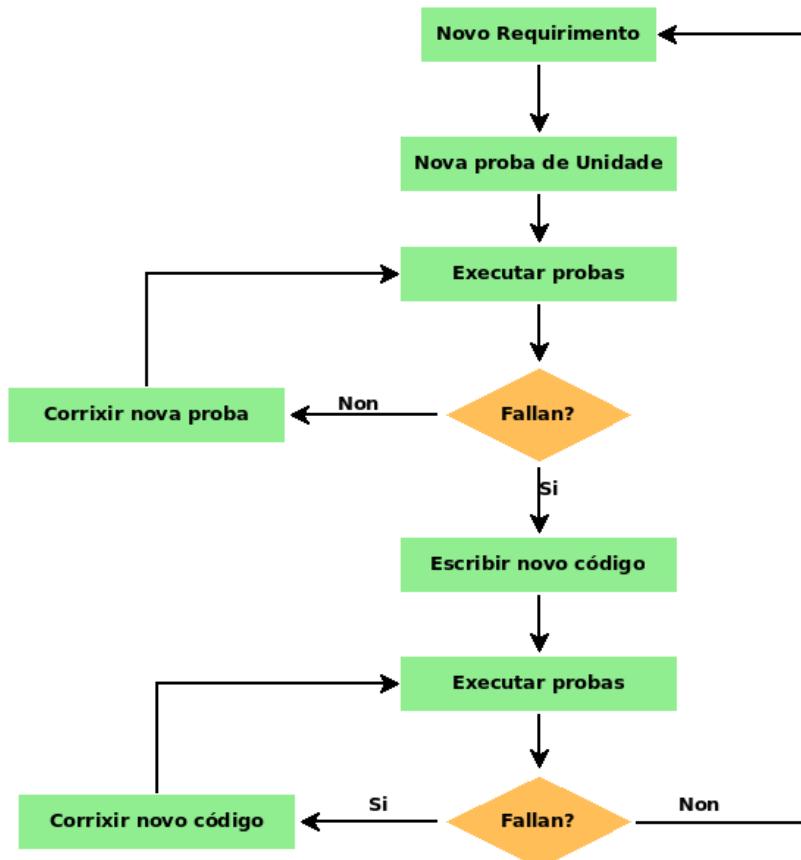


Figura 4.1: Diagrama do ciclo de desenvolvemento coa metodoloxía TDD.

Existen distintas estratexias á hora de seguir o citado ciclo [44] que se poden utilizar en función das características do requisito:

- **Fake it ('til you make it):** Ou “falséao (ata que o fagas)”. Baséase en escribir primeiro un método falso que devolva un valor constante que satisfaga o test. A partir de aí, ilo modificando aos poucos sempre vixiando que o test sexa positivo até que o método estea completo. Esta práctica é moi complexa de utilizar se os métodos son grandes, polo que pode ser unha boa forma de forzar a división, algo desexable para a cualidade do código.
- **Triangulación:** Consiste en facer as mesmas comprobación 2 ou máis veces con parámetros distintos dos que esperemos unha saída distinta coa fin de abstraer a función a implementar a raíz dos resultados esperados. Esta implementación é útil para evitar os parámetros superfluos que poidan aparecer nas funcións complexas.
- **Implementación Obvia:** Para as funcionalidades sinxelas, non paga a pena utilizar estratexias de abstracción. Podemos simplemente implementar o método e agardar a que o test non falle. Queda a discreción do desenvolvedor definir que funcións son sinxelas e complexas.
- **Un a Moitos:** No caso daqueles métodos que actúan sobre unha pluralidade de obxectos, esta estratexia avoga por dividir o requisito en dous: Habería que validar primeiro o un método que actuaría sobre unicamente un obxecto e, a continuación, validar que se execute sobre dita colección.

Neste proxecto, seguiuse maioritariamente a terceira estratexia da anterior lista por resultar más intuitiva para o traballo a realizar.

4.2. XP

Extreme Programming, ou Programación Extrema, é un estilo de desenvolvemento de software que promove unha serie de valores de cara ao traballo en equipo e unha serie de técnicas que supoñen unha simplificación e unha alternativa flexible en contraposición a outras metodoloxías más tradicionais coma, por exemplo, o “desenvolvemento en fervenza”.

Eses valores enuméranse coma[45]:

- **Comunicación:** Tanto entre programadores coma co cliente.

- **Simplicidade:** Que o código sexa flexible e a documentación concisa.
- **Retroalimentación:** Realizar iteracións curtas para obter opinións rápidas sobre os progresos.
- **Coraxe:** Aceitar que a aparición de novos requisitos é natural e poden redefinir o deseño.
- **Respecto:** Estrutura horizontal do equipo de desenvolvemento.

A posta en práctica dos anteriores valores lévanos a unha metodoloxía de traballo baseada no contacto continuo co cliente para poder obter correccións e ideas novas (retroalimentación). Isto acádase mediante ciclos curtos de desenvolvemento, ao final dos cales volveremos reunirnos co usuario obxectivo e o ciclo volve a comezar.

Referímonos a estes ciclos coma “iteracións”. Poden ser tomadas coma unha planificación do traballo a curto prazo a entender non coma unha improvisación, senón coma unha peza dun plan global suxeito á evolución do proxecto.

Debido á aceptación de que os sucesivos cambios no deseño son inevitables (coraxe) e que a refactorización de código é algo necesario para asegurar a súa calidad (simplicidade), faise necesaria a automatización de probas e a utilización dun control de versións.

A metodoloxía XP ten unha serie de características aplicables ao traballo en equipo que non se levaron a cabo neste proxecto: A programación por parellas, a revisión de código entre desenvolvedores ou a división do traballo en función da especialización do persoal (Testers, deseñadores de interacción, directores do proxecto, programadores...)

Capítulo 5

Planificación

5.1.	Orixе do proxecto	36
5.2.	Iteracións	37
5.2.1.	Primeira reunión (Iteración 0)	37
5.2.2.	Iteración 1	38
5.2.3.	Iteración 2	39
5.2.4.	Iteración 3	40
5.2.5.	Iteración 4	41
5.2.6.	Iteración 5	41
5.2.7.	Iteración 6	42
5.2.8.	Iteración 7	43
5.2.9.	Iteración 8	43
5.2.10.	Iteración 9	44
5.2.11.	Iteración 10	45
5.2.12.	Iteración 11	46
5.3.	Estudo de custos	47
5.4.	Programación das tarefas	48

Capítulo 5. Planificación

Neste capítulo explicarase a planificación do traballo realizado e a avaliación de custes.

5.1. Orixe do proxecto

Produciese un encontro informal con membros de Cuac FM (a radio comunitaria da Coruña) e a URCM onde se entrou en contacto cos usuarios finais do proxecto a realizar. A URCM (Unión de Radios Comunitarias de Madrid) está composta por unha serie de emisoras independentes e con programas de seu; todos eles listados nunha sección da web (ver figura 5.1) que á súa vez da acceso ao directorio dos ficheiros de audio (aos que a partir de agora nos referiremos coma “audios”) publicados por ditas emisoras. Esta sección da web, pese a súas limitacións, leva funcionando uns anos e resultou ser moi positiva para a redifusión dos programas por distintos colectivos.

Diagonal Periódico, boletín radiofónico



Desde que cumple su primer año de vida, la Unión de Radios Libres y Comunitarias de Madrid realiza junto con el periódico Diagonal, un boletín radiofónico con los contenidos más relevantes de la publicación.

Este boletín se produce quincenalmente y participan en él los dos colectivos, además de otros voluntarios de radios federadas (Radio Almenara y Radio Ritmo)

DIAGONAL es un periódico de información de actualidad, debate, investigación y análisis que, desde Madrid y gracias a la participación de decenas de personas en diferentes partes del mundo, se edita cada quince días. Anclado en la realidad de los movimientos sociales que luchan por transformar el actual orden de cosas, de donde nace y se nutre, se presenta como una alternativa comunicativa seria, de calidad y con vocación de tener una amplia difusión social.

Diagonal Periódico, boletín radiofónico: <http://audio.urcm.net/-Boletin-Diagonal-Periodico->

Dridam, radio cultural



Dridam-Radio Cultural es un proyecto de la Unión de Radios Libres y Comunitarias de Madrid y de la Dirección General de Promoción Cultural de la Consejería de Cultura y Turismo de la Comunidad de Madrid.

El proyecto se crea para la realización y difusión de producciones de contenido cultural a través del medio radiofónico.

Se pretende una mayor difusión de la cultura y las manifestaciones artísticas a través de radios locales.

El proyecto se desarrolla del 1 de noviembre de 2003 al 31 de octubre de 2004 y está prevista su continuidad y un mayor desarrollo para el 2007.

Este proyecto es una continuidad del llevado a cabo desde 2003 y que ya contó con la colaboración de la Dirección General de Promoción Cultural de la Consejería de Las Artes.

Dridam, radio cultural: <http://audio.urcm.net/-Dridam-Radio-Cultural->

Figura 5.1: Sección de audios da web da URCM que inspira o proxecto.

Inspirados nesta idea, propúxose crear unha ferramenta semellante, esta vez para a ReMC (Red de medios comunitarios), unha federación de medios comunitarios do Estado

Español. Defínironse uns requisitos iniciais, más centrados naquel entón na redifusión e a organización que na escoita.

A seguinte lista é unha transcripción das notas tomadas durante ese encontro:

- Os usuarios dos sistema son as propias emisoras.
- As emisoras teñen que poder engadir audios.
- As emisoras poden acceder aos audios das outras.
- As emisoras teñen que poder saber quen está a emitir os seus programas.

5.2. Iteracións

Como se explicou no capítulo 4, o desenvolvemento executouse de forma iterativa e incremental onde unha iteración son os obxectivos a cumplir entre dúas reunións co cliente. Ao ser isto un proxecto de final de carreira, contarase o tempo entre as sesións de revisión de progresos entre o director do proxecto e o alumno. Estas reunións tiveron unha periodicidade bisemanal na súa meirande parte, reducindo a duración dos ciclos nas derradeiras fases da implementación.

5.2.1. Primeira reunión (Iteración 0)

Na primeira reunión estableceuse a lista de obxectivos xerais do proxecto repasados xa na sección 1.3. Estes supoñen unha aproximación ao problema máis ambiciosa que a idea orixinal de ter un simple directorio web compartido entre emisoras pois ten en conta as necesidades dos ouvintes e a propiedade dos programas por parte dos seus autores e non necesariamente da emisora, cousa bastante común no mundo da radio comunitaria. Foi neste momento cando se tomou a decisión de utilizar Django coma framework de desenvolvemento e PostgreSQL coma sistema de xestión de bases de datos.

Os obxectivos de cara a primeira iteración foron:

- Facer un primeiro borrador do deseño do modelo de datos.
- Investigar a manipulación de ficheiros RSS utilizando Python. Desde o principio, a consulta dos ficheiros de RSS intuíuse coma o xeito de manter actualizados os programas mais neste momento ánda non se tomara unha decisión firme de como facelo.

5.2.2. Iteración 1

Cumpríronse os obxectivos marcados, entregando un deseño preliminar do modelo de datos na data estimada (figura 5.2)

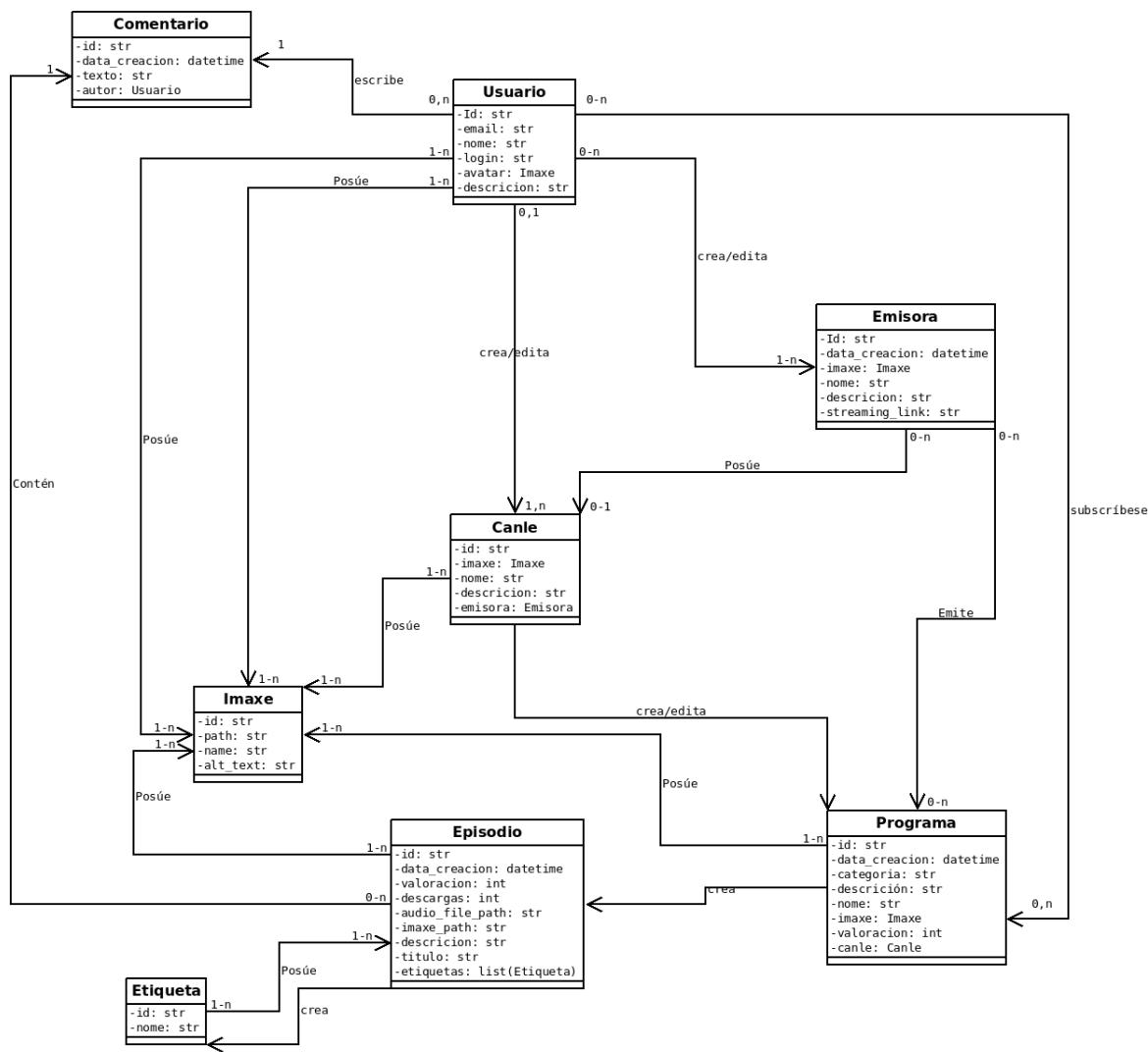


Figura 5.2: It1: Diagrama de clases do primeiro borrador de deseño.

Unha cousa a comentar neste primeiro modelo é a existencia dunha clase “canle” intermediaria entre a emisora e o programa que, como veremos, quedou finalmente desbotada. A idea era que os colectivos que realmente teñen unha emisión por onda e aqueles que publican podcasts de escucha baixo demanda terían necesidades distintas e debían diferenciarse, sempre sen bloquear a posibilidade de que unha radio emitise podcasts.

En canto ao RSS, fixéronse os primeiros scripts de lectura. Ese primeiro código era capaz de descargar un ficheiro RSS e transformalo nun obxecto cos datos do programa que á súa vez contiña unha lista de obxectos cos datos dos episodios, demostrando que a consulta de RSS si era a forma axeitada de afrontar a inserción e actualización dos contidos.

Novos obxectivos:

- Instalar a infraestrutura desenvolvemento: Instalación do entorno Django-PostgreSQL.
- Implementación de Programa e Episodio.
- Implementación dunha interface web básica.
- Ampliar o código do script de lectura dos ficheiros RSS.

5.2.3. Iteración 2

Configurouse o entorno de django e as ferramentas de desenvolvemento. Creáronse as clases Program e Episode coas súas correspondentes táboas nunha base de datos controlada mediante PostgreSQL. Esas primeiras clases aínda non contiñan información das imaxes nin das etiquetas e os datos non extraíbles do RSS contiñan valores nulos. O sistema executábase sobre o servidor de desenvolvemento de Django, algo que se mantivo durante todo o proceso de desenvolvemento por comodidade e eficiencia. Creouse tamén un repositorio para o control de versións utilizando a plataforma GitHub (ver sección 3.9.2)

Conseguiuse a medias o obxectivo de servir unha interface web sinxela: Amosábanse os datos gardados, pero non servía para engadir novo contido aínda. Desprazouse ese obxectivo á seguinte iteración.



Figura 5.3: It2: Interface web. Páxina dun episodio.

Os traballos de ampliación do código de lectura de RSS leváronse a cabo: Os programas e episodios recollían toda a información requirida no momento agás a información de imaxe e de etiqueta (tag). Os traballos neste código revelaron a necesidade de dispor de distintos tipos de lector dependendo da fonte da que o ficheiro RSS fose extraído.

Novos obxectivos::

- Engadir formularios á interface para crear novos programas desde ela.
- Implementar novos tipos de lector de RSS.
- Aplicar “plantillas” de estilos ao frontend.

5.2.4. Iteración 3

Creouse o formulario de envío do enlace RSS. Instalouse o soporte de Django para Bootstrap 4, podendo así usar eses modelos. Utilizándoos, creouse unha páxina de engadir programa co aspecto amosado na figura 5.4.

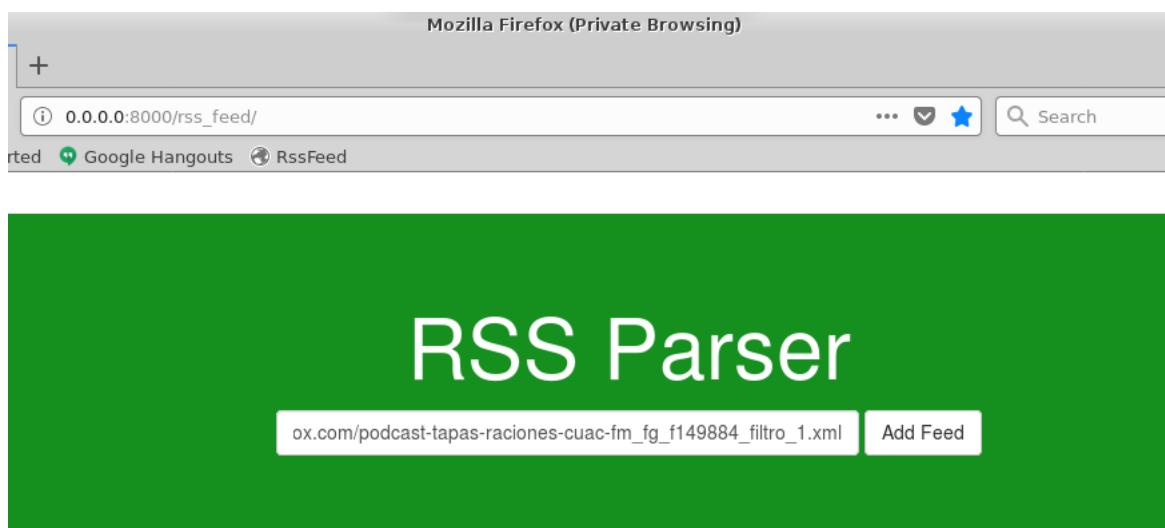


Figura 5.4: It3: Interface web. Páxina de engadir programa.

Implementouse o soporte para os distintos formatos de ficheiro RSS: Ao principio da iteración só se podían engadir ficheiros extraídos de RadioCo. Ao final, os parsers para Ivoox e Podomatic quedaron tamén implementados. Pese a que os 3 funcionaban, a calidade do código non era a satisfactoria.

Novos obxectivos:

- Refactorización dos lectores de RSS.
- Implementación das clases de imaxe e etiqueta.
- Redefinir a estrutura das páxinas.

5.2.5. Iteración 4

Nesta iteración créase un módulo específico para as funcións de lectura de RSS. Aplícase un patrón estratexia para dar soporte a distintos parsers cunha interface única (máis detalles no capítulo 7 sobre o deseño). Para completar o proceso de inserción de programas e episodios, implementáronse as clases da imaxe e etiqueta (Image e Tag).

p_id	p_name	tag
1	Que no es Poco	comedy
2	Spoiler	tv & film
3	Alegría CUAC FM 103.4	comunicacion
3	Alegría CUAC FM 103.4	radio
3	Alegría CUAC FM 103.4	coruña
3	Alegría CUAC FM 103.4	solidaridad
3	Alegría CUAC FM 103.4	cuac
3	Alegría CUAC FM 103.4	izquierda
3	Alegría CUAC FM 103.4	politica
3	Alegría CUAC FM 103.4	society & culture
3	Alegría CUAC FM 103.4	esquerda
3	Alegría CUAC FM 103.4	galiza
4	El Desinformativo	news & politics
(13 rows)		

Figura 5.5: It4: Resultado de consulta á base de datos relacionando programas cos seus tags.

No tocante á interface, introducíronse nesta iteración as follas de estilo con CSS-Grid, definindo un deseño que serviu coma base do que máis tarde sería o deseño definitivo.

Novos obxectivos:

- Crear un proceso que corra no servidor para manter os programas actualizados.
- Engadir usuarios ao sistema.

5.2.6. Iteración 5

Levouse a cabo a instalación de Celery e as súas dependencias para correr procesos sobre os seus fíos. Reutilizouse o código do módulo que contén os parsers de RSS.

No tocante aos usuarios, escribiuse unha primeira versión da autenticación e o rexistro, o cal deu pé a relacionar os usuarios cos seus programas engadidos. Porén, a clase usuario por defecto de Django non satisfacía as necesidades previstas.

Novos obxectivos:

- Estender a clase Usuario.
- Crear páxina de edición de Ususario.
- Facer as vistas dependentes da autenticación.

5.2.7. Iteración 6

A partir desta iteración, as reunións pasan a ser semanais.

Introduciuse unha clase de apoio para outorgarlle ao usuario de Django unha serie de atributos necesarios no sistema. Introduciuse a comprobación de autenticación naquelhas vistas que o requirisen, por exemplo, engadir programa, como se ve nas figuras 5.6 e 5.7.

Fíxose unha primeira implementación da vista de edición de usuario, pero sen soporte para o cambio de contrasinal. Quedaría para a iteración seguinte.

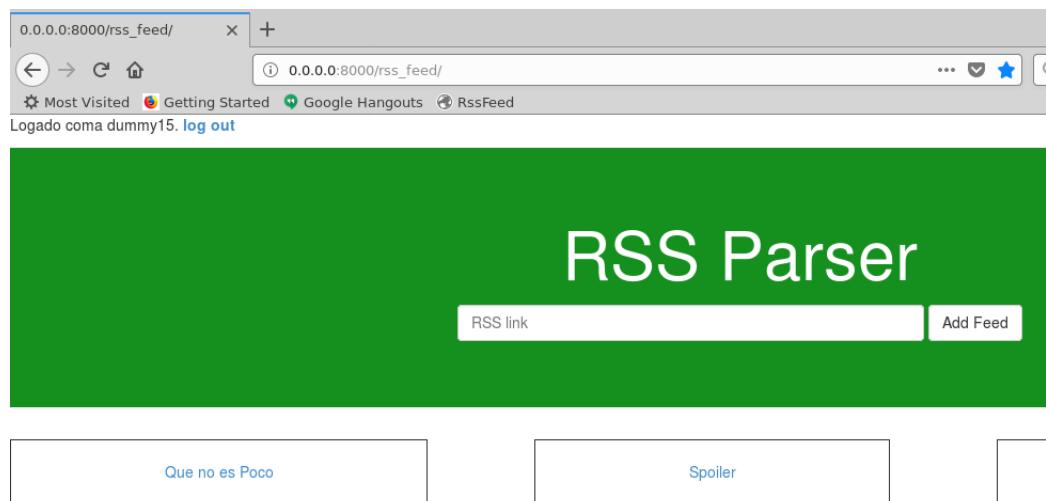


Figura 5.6: It6: index.html para usuario identificado.

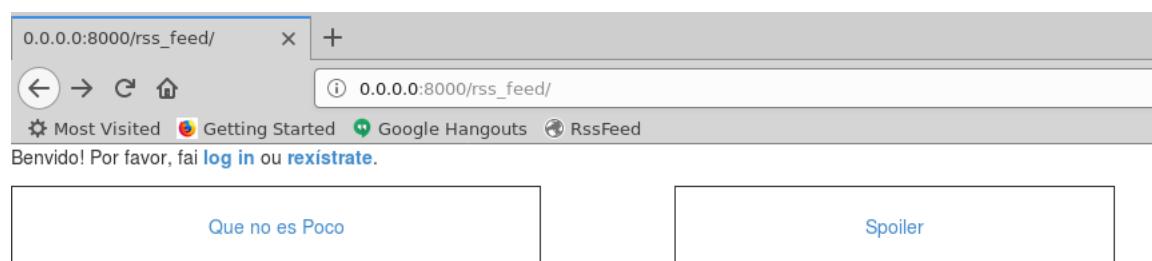


Figura 5.7: It6: index.html para usuario anónimo.

Novos obxectivos:

- Completar páxina de edición de usuario.
- Implementar clases para os votos e os comentarios dos episodios.
- Implementar a clase de emisora e canle.
- Crear deseño de interacción das páxinas.

5.2.8. Iteración 7

Debuxáronse uns deseños esquemáticos das distintas páxinas de interface web e definiuse a navegación entre elas. Este deseño de interface foi xa o definitivo. Valería, en diante, coma guía para confeccionar as distintas vistas. Poden verse algúns borradores no capítulo 7 sobre o deseño.

O resto de obxectivos foron cumplidos. Nesta iteración decidiu desbotarse a clase de Canle por ter unhas funcións moi limitadas e solaparse en gran parte con Emisora.

Novos obxectivos:

- Dar soporte á internacionalización.
- Crear vista para engadir novas emisoras.

5.2.9. Iteración 8

Engadiuse o soporte de Django para a internacionalización e fixose unha primeira tradución da web a lingua galega. Non obstante, o seu funcionamento non era o axeitado. Dado que se estaba empregar moito tempo nesta tarefa sen acadar ningún resultado, decidiuse pospoñela.

Co gallo de situar o selector de idioma, fixose unha cabeceira para situar as ferramentas transversais ás vistas.

Integráronse a vista de engadir programa e a nova de crear emisora nunha única, chamada “engadir contido”. Debido ao bloqueo sufrido pola tarefa de internacionalización, sobrou tempo que se empregou en implementar a vista da páxina principal (index) e a vista de detalles de emisora (figura 5.8).

Novos obxectivos:

- Corrixir internacionalización (Posposto)
- Crear edición de emisora e programa.

- Implementar accións de ouvinte.
- Completar vistas de detalle de emisora e programa

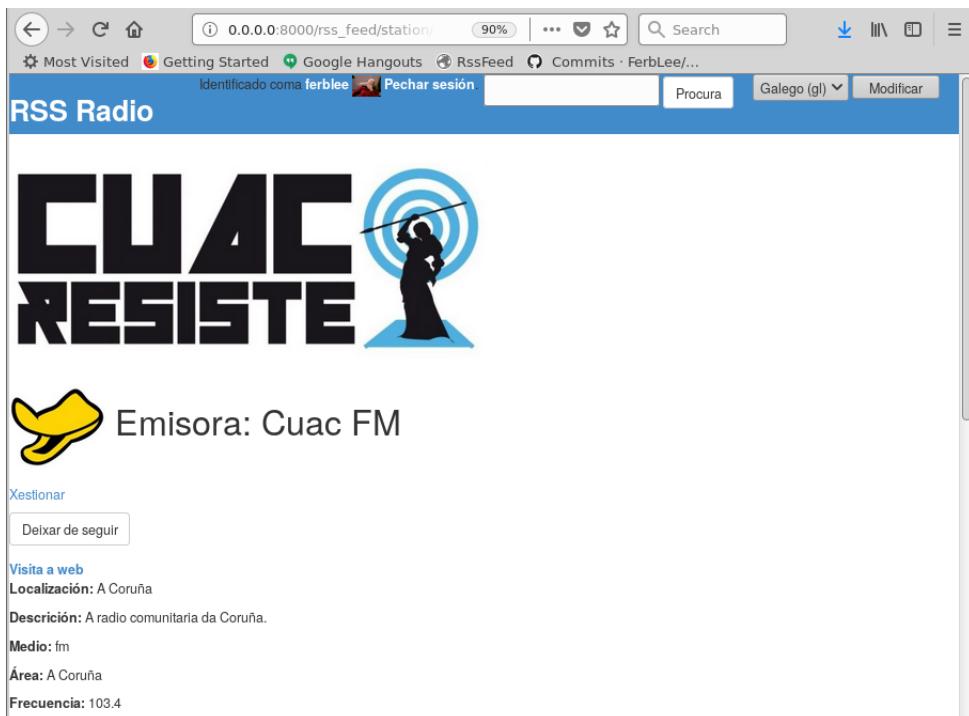


Figura 5.8: It8: Vista de detalles de emisora.

5.2.10. Iteración 9

Actualizáronse as vistas de detalle de programa e episodio (ver figura 5.9). Implementáronse as accións dos ouvintes: Votar os episodios (Positivo, negativo e desfacer voto), subscribirse, seguir emisora e engadir comentarios.

Ao afrontar a implementación das vistas de edición, discutiuse a cerca da relación entre os usuarios e os seu contido. Chegouse a conclusión de que sería necesaria a creación de roles de administración para programas e emisoras.

Descubríronse unha serie de errores na vista de engadir contidos que foron corrixidos. Non se realizou ningunha actividade no tocante á internacionalización.

Novos obxectivos:

- Corrixir internacionalización.
- Crear roles de usuario.

- Crear panel de edición e xestión de emisora e programa.
- Implementar as funcións de procura de contidos.

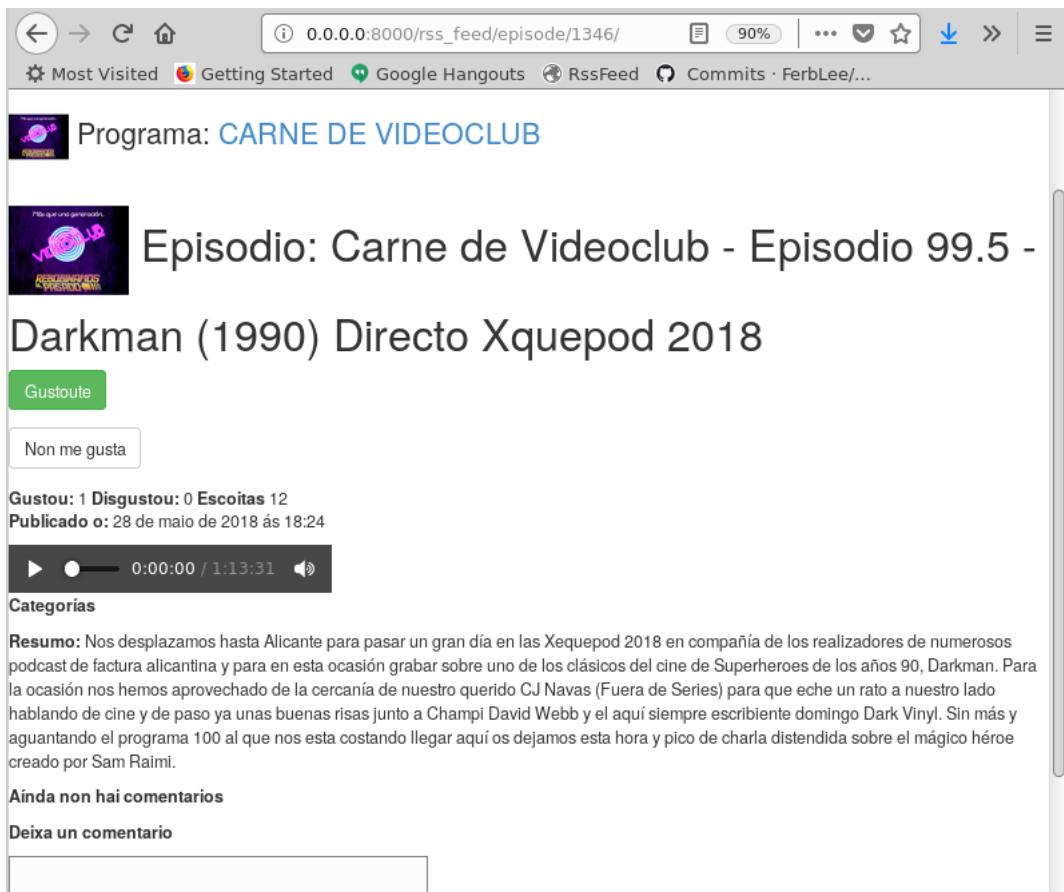


Figura 5.9: It9: Vista de detalles de episodio.

5.2.11. Iteración 10

Creáronse os roles de usuario e fixose o panel de edición e xestión de programas e emisoras. A implementación da procura quedou posposta por falta de tempo.

Coméntose a posibilidade de adaptar o estilo da páxina para os dispositivos móveis.

Novos obxectivos:

- Comezar a redacción da documentación.
- Corrixir internacionalización.
- Implementar as funcións de procura de contidos.

Capítulo 5. Planificación

- Adaptar o estilo para dispositivos móviles.

Panel de xestión de Que no es Poco

Editar Perfil

Xestión de Emisión

Administración

Borrar Programa

Editar Perfil

Enlace ao RSS

Sítio web

Compartición

Permitir comentarios

Actualizar Cancelar

Figura 5.10: It10: Panel de xestión de programa.

5.2.12. Iteración 11

Resultados da procura de: A Coruña

Episodios atopados:

- Alegria en CUAC FM del 17-11-09
- Alegria CUAC FM 103.4
- Tapas y Raciones Especial Navidad (Parte 2 de 3)
- Podcast Tapas y Raciones - Cuac FM
- Tapas y Raciones - Programa 95 - Cuac FM
- Podcast Tapas y Raciones - Cuac FM
- Alegria en CUAC FM do 27-10-2009
- Alegria CUAC FM 103.4

Programas atopados:

- Que no es Poco
- Hasta Los Kinders

Emisoras atopadas:

- audio audio/mpeg barcelona
- canada canadianj china clubdj
- cocopere comedy
- comedy danko dj djjane
- djmismi españa european
- femaledj humor itunes
- itunesstore ivoox jpod
- losdanko miss missm
- mixshow mixtape mmadictos music
- openformatdj partydj peñisbal
- pelos podcast
- podcaster podcasting
- podcasts power sam spain

Figura 5.11: It11: Páxina de resultados de busca.

Todos os obxectivos restantes foron cumplidos, sendo o máis destacable a nova vista de resultados de busca (Figura 5.11). Implementáronse ferramentas de busca por texto e busca por tag.

A partir de aquí, as sucesivas reunións consistiron en revisións desta documentación. Demos, polo tanto, o desenvolvemento por finalizado.

5.3. Estudo de custos

A intención desta sección é facer unha estimación do custo do proxecto. Dado que este foi realizado por unha única persoa levando a cabo distintas tarefas en cada fase do proxecto, estimarase un custe uniforme de 19€/h para todas elas.

Na táboa 5.1 pódese ver o gasto por iteración e o custo total do proxecto.

Iteración	Horas de traballo	Custo(€)
Primeira reunión	1	19
Iteración 1	3	57
Iteración 2	12	228
Iteración 3	10	190
Iteración 4	20	380
Iteración 5	25	475
Iteración 6	23	437
Iteración 7	27	513
Iteración 8	32	608
Iteración 9	26	494
Iteración 10	30	570
Iteración 11	18	342
Escritura da documentación	45	855
Total	272	5168

Táboa 5.1: Custo das distintas etapas de traballo.

5.4. Programación das tarefas

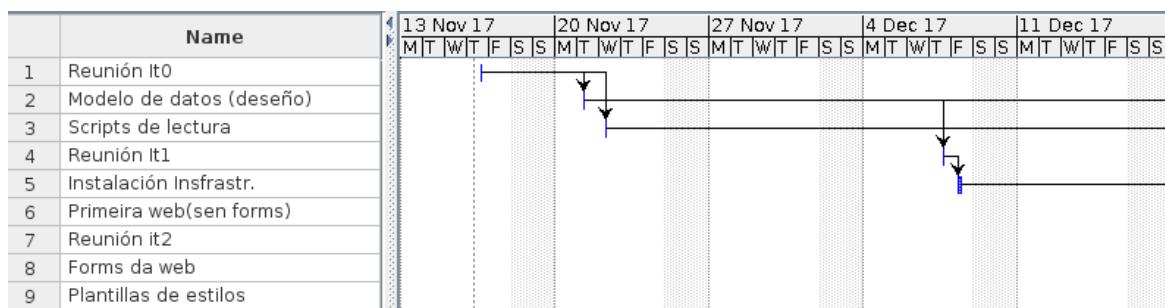


Figura 5.12: Diagrama de Gantt: Novembro 2017 - Decembro.

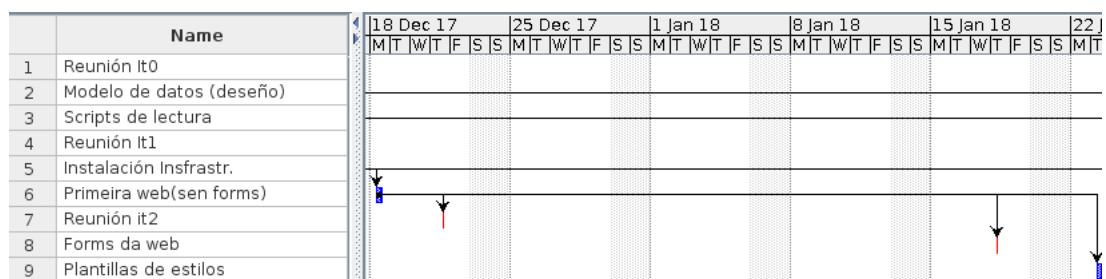


Figura 5.13: Diagrama de Gantt: Decembro 2017 - Xaneiro 2018.

5.4. Programación das tarefas

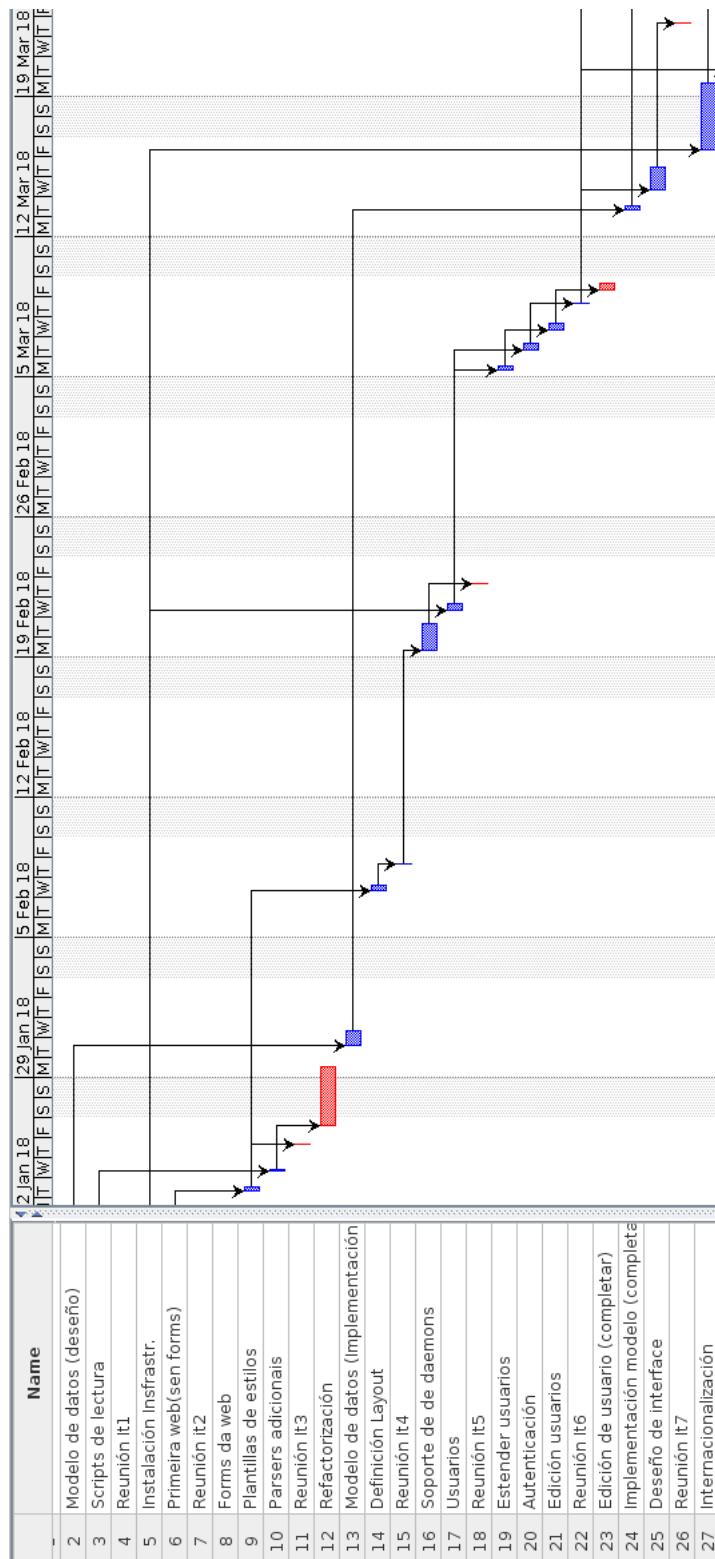


Figura 5.14: Diagrama de Gantt: Xaneiro 2018 - Marzo 2018.

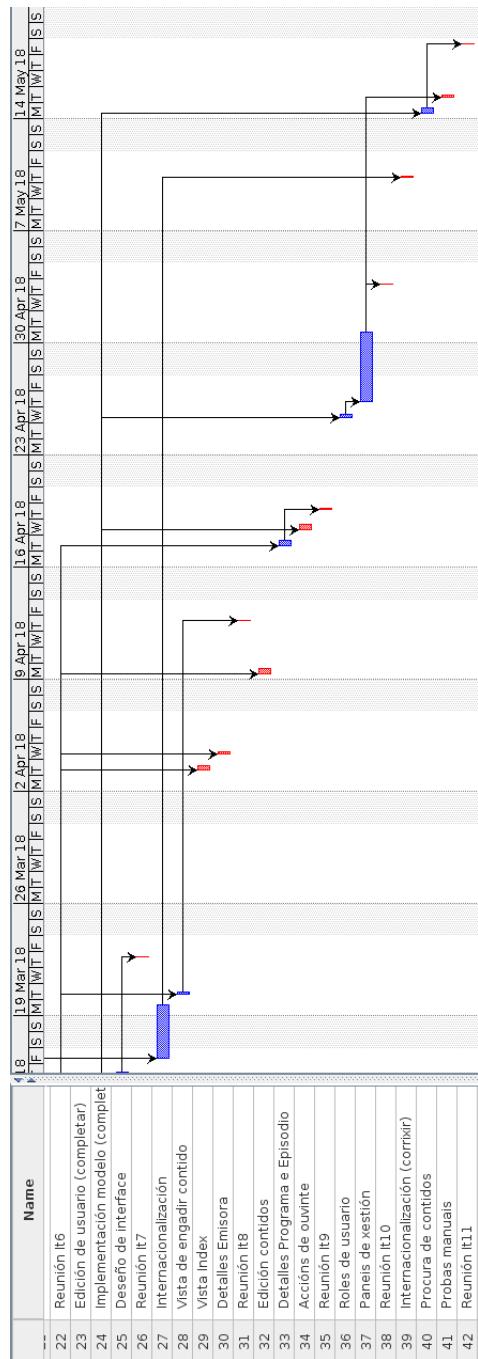


Figura 5.15: Diagrama de Gantt: Marzo 2018 - Maio 2018.

5.4. Programación das tarefas

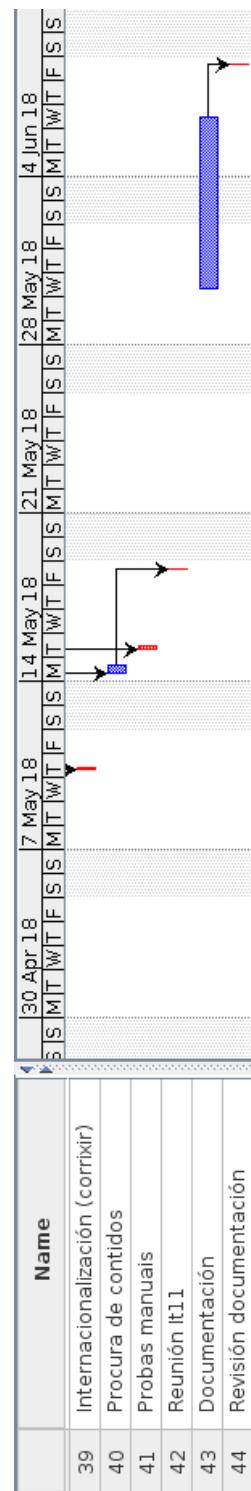


Figura 5.16: Diagrama de Gantt: Maio 2018 - Xuño 2018.

Capítulo 6

Análise

6.1.	Requerimentos funcionais	54
6.2.	Requerimentos non funcionais	55
6.2.1.	Autenticación	55
6.2.2.	Internacionalización	55
6.2.3.	Rendimento	56
6.2.4.	Seguridade	56
6.2.5.	Adaptabilidade a dispositivos móbiles	56

Neste capítulo repasaránse os requisitos identificados ao abordar os obxectivos do proxecto. Distinguimos entre:

- **Requerimentos funcionais:** Son aqueles que describen as accións que o sistema debe efectuar, isto é: a colección de capacidades e características que o software pon a disposición do usuario.
- **Requerimentos non funcionais:** Son aqueles que especifican as propiedades do sistema no referente á manexabilidade, seguridade e robustez[46].

6.1. Requerimentos funcionais

Presentaranse estes en forma de “historias de usuario”. Unha historia de usuario é unha frase concisa onde o usuario obxectivo expresa unha necesidade que quere que o sistema cubra.

Se ben é certo que houbo encontros con membros de colectivos do terceiro sector coma a U.R.C.M.(Unión de radios libres e comunitarias de Madrid) e Cuac FM (A radio comunitaria da Coruña), parte delas están extraídas da experiencia propia coma colaborador nestes medios e coma moderador de comunidades en liña.

- Ter un punto de encontro onde ver os programas que están a emitir as emisoras federadas na ReMC (Red de Medios Comunitarios)
- Que unha emisora poda acceder aos ficheiros de audio das demais.
- Que os “autores” dun programa poidan saber en que emisoras está a ser emitido e a que hora.
- Que os programas se poidan dar de alta e de baixa.
- Poder visualizar información das emisoras e dos programas: Nome, información de emisión, datos de contacto...
- Poder subscribirse aos programas.
- Poder compartir contido entre usuarios.
- Que os ficheiros de audio se asocien de xeito automático a cada emisora.
- Que non sexa necesario engadir manualmente os novos ficheiros.
- Que os usuarios podan colaborar na xestión das emisoras.

- Agrupar os programas por categorías.

Eses “ficheiros de audio” nos que se pensaba nunha primeira aproximación foron o que finalmente derivou no concepto de “episodios de programa” que se explicará máis a fondo no capítulo 7 sobre o deseño.

Gran parte destas historias veñen dos usuarios que producen o contido. A medida que avanzaba o deseño e os primeiros pasos da implementación, fóreronse perfilando novas historias de usuario pensando máis nos ouvintes:

- Que os ouvintes poidan manifestar a súa opinión sobre os programas mediante votos e comentarios.
- Que os ouvintes poidan manter unha lista de programas e emisoras favoritas.
- Ter ferramentas de procura de contidos.
- Recibir recomendacións.

E tamén outros requerimentos propios da xestión dos contidos.

- Crear diferentes roles de administración.
- Que o administrador dun programa poida restrinxir a emisión do seu contido.
- Que o administrador dun programa poida moderar os comentarios no seu contido.

6.2. Requerimentos non funcionais

6.2.1. Autenticación

O sistema ten que permitir o rexistro de usuarios cun login único e un contrasinal. Dito contrasinal ha de gardarse cifrado na base de datos. A información de sesión da autenticación debe protexerse de manipulacións para evitar impostores.

6.2.2. Internacionalización

O sistema debe ter soporte para a tradución a distintos idiomas. As datas deben gardarse nun formato común e zona horaria UTC(Universal Time Coordinated) para adecuarse na interface ás preferencias do usuario.

6.2.3. Rendemento

A carga de obxectos desde a base de datos debe realizarse de xeito “lazy” (preguiceiro) de modo que non se sobrecargue a memoria de forma innecesaria. Isto é especialmente eficiente en comprobacións de existencia de obxectos ou na navegación entre clases relacionadas.

O tráfico entre o cliente e o servidor debe minimizarse mediante, por exemplo, un sistema de paxinación para aquelas vistas que amosen unha grande cantidade de datos.

6.2.4. Seguridade

Os novos datos que se engadan ao sistema han de ser validados para evitar comportamentos maliciosos coma ataques de inxección de código. Cómpre tamén facer validación das URL's para evitar accesos non permitidos.

6.2.5. Adaptabilidade a dispositivos móbiles

Xa que nos capítulos anteriores mencionamos a importancia dos dispositivos móbiles no crecemento da radio por Internet, a páxina debe ser amigable con eles adaptando a interface cando a pantalla na que se execute sexa pequena.

Capítulo 7

Deseño

7.1. Descripción do funcionamento	58
7.1.1. Actividades de consumo de contidos	58
7.1.2. Actividades de producción de contidos	58
7.2. Arquitectura	59
7.2.1. Capa Modelo	60
7.2.1.1. Usuarios	60
7.2.1.2. Programas e episodios	61
7.2.1.3. Votos e comentarios	63
7.2.1.4. A entidade Station	63
7.2.1.5. Administración de contenido	63
7.2.2. Capa Vista	64
7.2.2.1. Engadir contenido	66
7.2.3. Capa Template	67
7.2.3.1. Portada	68
7.2.3.2. Páxinas de engadir e editar contenido	69
7.2.3.3. Páxinas de detalle	69
7.2.3.4. Páxina de resultados de busca	69
7.2.3.5. Páxinas de xestión	70
7.3. Actualización dos datos	72
7.3.0.1. Novos episodios	72
7.3.0.2. Popularidade e cualificación dos programas	73

7.1. Descripción do funcionamento

O sistema consiste nunha páxina web con apartados adicados, por un lado, ao que chamamos actividades de consumo, e por outro ás actividades de produción.

7.1.1. Actividades de consumo de contidos

Son aquellas levadas a cabo polos ouvintes das emisoras e os programas presentes no sistema. Un usuario, no seu papel de “consumidor”, pode, mediante a interface web, acceder ao catálogo de programas, xa sexa mediante as ferramentas de busca proporcionadas ou mediante as recomendacións que a propia páxina amosa en portada.

Dentro da páxina correspondente a unha emisora, o usuario atopará un reprodutor HTML5 desde o que escoitar a súa emisión en directo mediante streaming. O usuario terá opción de facerse “seguidor” da emisora para acceder a ela de xeito máis rápido desde a páxina principal.

Tamén obterá acceso a lista de programas emitidos por dita emisora co seu horario de emisión. Na páxina de cada programa verá a lista completa de episodios dispoñibles e terá a opción de subscribirse para poder acceder de xeito máis rápido aos novos episodios publicados.

Na páxina propia de episodio atopará outro reprodutor HTML que lle posibilitará ou ben a escoita por streaming ou ben a descarga directa do ficheiro de audio correspondente. Poderá votar o programa a favor ou en contra (isto repercutirá na popularidade do programa, concepto explicado máis adiante) e deixar un comentario no caso de que esta opción esté habilitada polos administradores do programa.

7.1.2. Actividades de producción de contidos

Son aquellas levadas a cabo polos donos e administradores dos programas e emisoras. Un usuario, no seu papel de produtor, poderá engadir ao sistema unha nova emisora cubrindo o formulario habilitado a tal efecto na interface web no que deberá introducir manualmente os datos.

Poderá tamén engadir un programa. Para isto, o usuario só necesitará proporcionarlle ao sistema un enlace ao ficheiro de RSS do podcast que queira engadir e o sistema encargarase de extraer a información tanto dos programas coma dos episodios correspondentes. Este programa e os seus episodios manteranse actualizados xa que o propio backend do sistema

comprobará periodicamente se houbo algún cambio no RSS e actualizará a base de datos de xeito acorde.

Un usuario que posúa un programa ou emisora pode invitar a outro usuario a colaborar na xestión desta ou deste.

7.2. Arquitectura

A elección de Django coma framework de desenvolvemento obriga a seguir o patrón Modelo-Vista-Template. Este patrón é unha pequena variación do máis coñecido Modelo-Vista-Controlador, que define 3 capas interconectadas coa fin de separar a lóxica da aplicación(Vista) do almacenamento dos datos(Modelo) e estas dúas, á súa vez, da interface de usuario(Controlador). A vantaxe que ofrecen os “templates” de Django é a posibilidade de utilizar os tipos propios do framework para implementar certa lóxica presentacional no renderizado do template.

Neste proxecto, os modelos definíronse no módulo `models.py` e as vistas en `views.py`. Trataranse coma paquetes nos seguintes diagramas. A capa template ven definida no paquete de templates.



Figura 7.1: Árbore de módulos do aplicativo web.

7.2.1. Capa Modelo

Á hora de definir esta cuestión, tratáronse de respectar as regras do modelo relacional de bases de datos, porén, fixeronse algunhas excepcións motivadas por esixencia da elección do framework Django:

- Claves primarias subrogadas: Django encárgase da identificación das tuplas engadindo ás táboas un campo numérico enteiro de incremento automático. Isto utilizarase en todas as táboas sen excepción.
- Relación 1-1: Como se ve no diagrama Entidade-Relación da figura 7.2, existe unha relación 1-1 entre a entidade User e a súa entidade feble UserProfile. O motivo da existencia desta última é que se decidiu utilizar a entidade de usuario nativa de Django, co cal fíxose necesaria unha nova táboa para cubrir os atributos de usuario necesarios especificamente para o proxecto.

No diagrama Entidade-Relación da figura 7.2 pódense ver as táboas empregadas sen incluir aquelas automáticamente xeradas para o correcto funcionamento de Django e Celery a excepción da xa mencionada User. Por motivos de claridade, non se incluíron os atributos agás aqueles adicionais nas táboas correspondentes ás relacións N-N.

A estrutura da BD tradúcese na aplicación ás clases definidas no paquete `models.py` que se ve na figura 7.3. Por claridade, nese diagrama de clases só se incluíron os atributos máis representativos ou explicativos das relacións entre clases.

7.2.1.1. Usuarios

Os usuarios están definidos por dúas táboas: `User` e `UserProfile` que se corresponden coas clases `User`, do paquete `django.contrib.auth.models`, e `UserProfile`, definida polo desenvolvedor e, polo tanto, includida no módulo `models.py`. Esta segunda considerámola coma feble de `User` pois a existencia dun perfil está vencellada de xeito ineludible á existencia dun usuario (ver figura 7.2). Esta entidade auxiliar contén os atributos de usuario: avatar, descripción e localización. Non se utiliza para máis nada.

A clase `User` inclúe os atributos necesarios para a autenticación: `username`, `email` e `password`, quedando este último encriptado en base de datos mediante o algoritmo PBKDF2. De entre os campos utilizados para o funcionamento de Django, cómpre destacar `is_staff`, que é o que concede acceso do usuario ao panel de administración do aplicativo que provee o propio framework (como se mencionou no apartado 3.2)

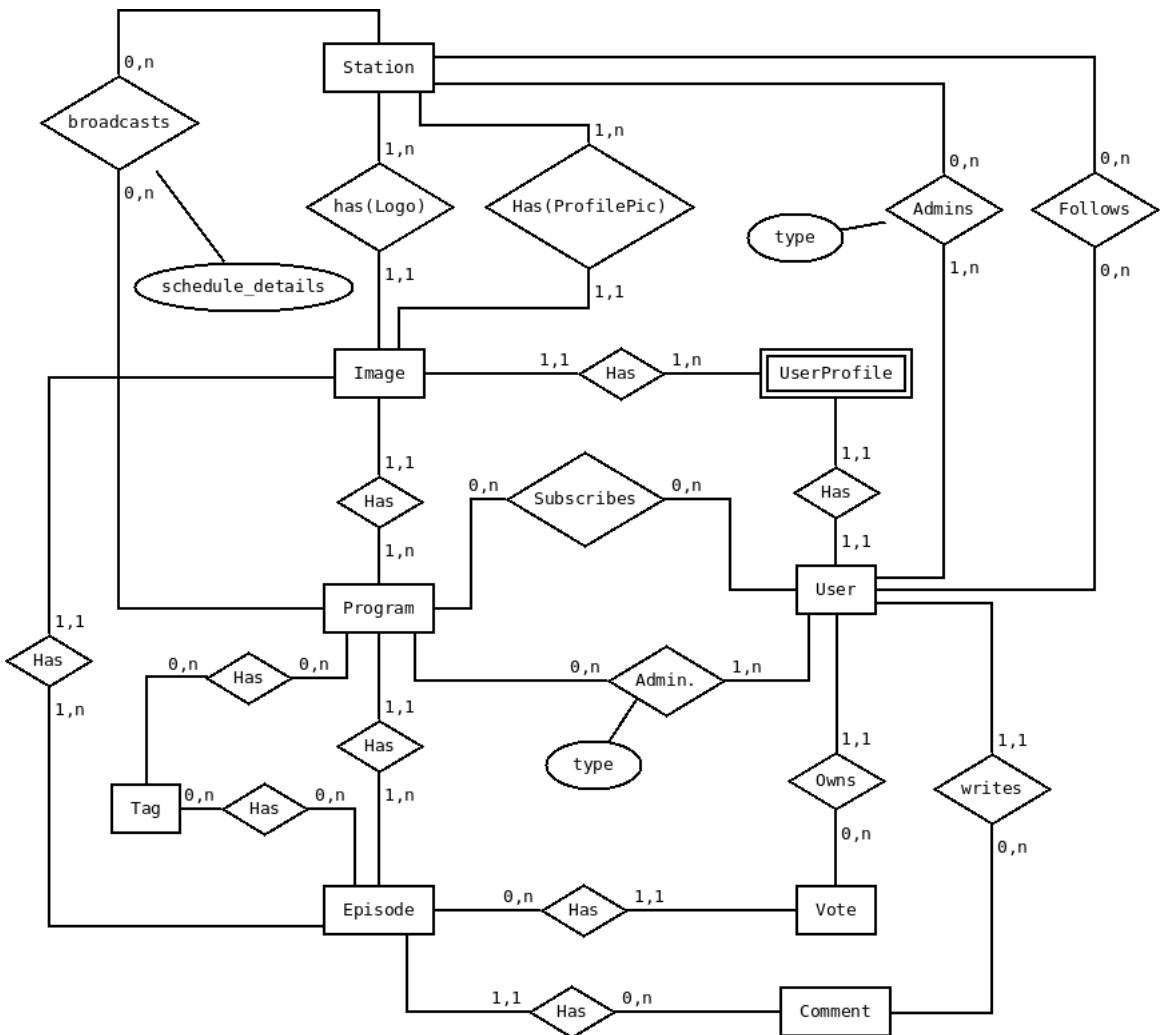


Figura 7.2: Diagrama Entidade-Relación da Base de Datos.

Outros atributos de User reflectidos na figura 7.3 coma followers ou station_admins son engadidos automáticamente polo framework ao declarar as relacións coa fin de facilitar a navegación entre clases.

7.2.1.2. Programas e episodios

Entendemos, neste sistema, un programa coma un producto radiofónico do cal se publican entregas periódicamente. Están almacenados na táboa Program da base de datos. Os programas teñen, entre outros atributos, un nome, unha descripción, unha imaxe, un conxunto de categorías (táboa Tag) e un enlace a un ficheiro RSS dado polo usuario.

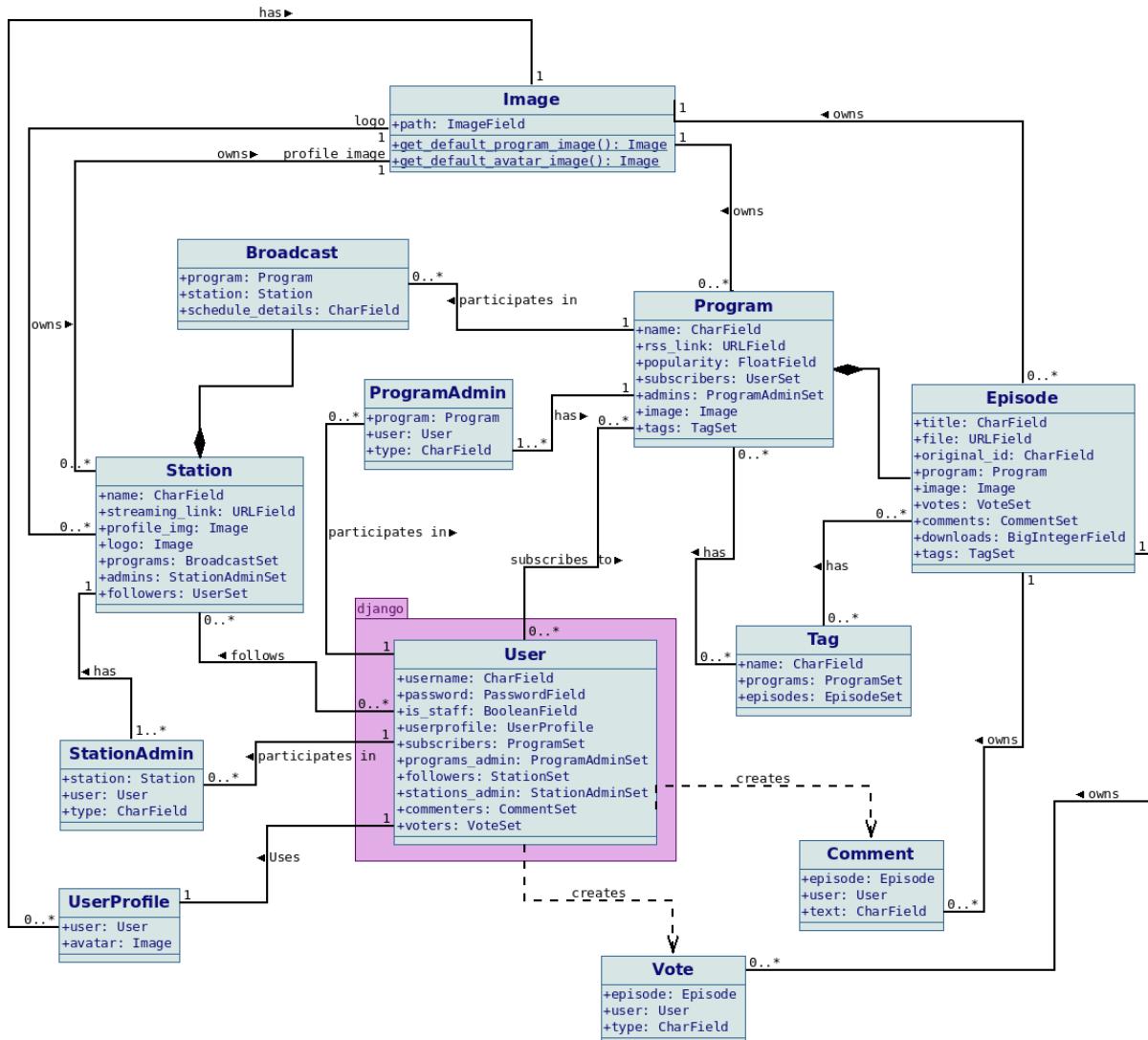


Figura 7.3: Diagrama de clases da capa modelo.

A subscrición dos usuarios aos programas modelouse coma unha relación N-N. As instances de Program poden acceder ao conxunto dos seus subscriptores mediante o atributo subscribers.

Episodio é como chamamos ás entregas dun programa. Cada un ten, entre outros atributos, un enlace a un ficheiro de audio que será o que se poña a disposición dos usuarios na interface, un resumo, unha imaxe e un conxunto de etiquetas (tags). Un episodio ten que formar parte de 1 e só 1 programa. Se un programa é borrado, os episodios deixan de ter sentido no sistema e son borrados en cascada. Debido ao anterior, considérase a relación destas clases coma unha composición.

7.2.1.3. Votos e comentarios

Nunha primeira aproximación ao deseño da base de datos, entendéronse estas dúas entidades coma simples relacións N-N entre as entidades Episode e User. Porén, decidiuse finalmente que representan conceptos de seu que non perden o sentido semántico fóra da relación e, polo tanto, modeláronse coma entidades (Vote e Comment nos diagramas 7.2 e 7.3).

O atributo type de Vote serve para lle dar significado ao voto: Positivo, negativo ou neutro.

7.2.1.4. A entidade Station

Identícanse coa entidade Station os colectivos aos que se poidan asociar os programas: Emisoras de radio por ondas, emisoras de radio por internet ou canles de podcasting. O atributo mais destacable de Station é o de streaming_link, que garda o enlace á canle de emisión por internet da emisora que será logo utilizado polo reproductor na web. Este campo pode ser nulo para aqueles colectivos que non dispoñan de emisión en directo.

A emisión dos programas por parte das emisoras queda modelada coma unha relación N-N entre Program e Station á que chamamos broadcasts. Engadiuse o atributo adicional de schedule_details, un campo de texto para os detalles de emisión: Horario, periodicidade... A clase correspondente en models.py é Broadcast. Dada a natureza de Station coma aglutinador de programas e dado que non ten sentido a existencia dunha emisión sen emisora, entendeuse que a relación entre Broadcast e Station é de composición.

7.2.1.5. Administración de contido

Existe unha relación N-N entre os usuarios e as emisoras e unha semellante entre os usuarios e os programas: A de administración. Entendemos esta coma a posibilidade de editar, actualizar e borrar os contidos preexistentes. Isto queda reflectido nas clases ProgramAdmin e StationAdmin. As súas instancias relacionan, respectivamente, aos usuarios cos programas e emisoras que poden administrar e establecen os permisos de administración que posúen, estes últimos, expresados polo atributo type.

Tanto para a administración de emisoras como de Programas, existen dous roles: Propietario e administrador. O propietario ten permisos completos: Edición, actualización, borrado e xestión de administradores. O administrador só ten permisos de edición e actualización.

Nótese que tal relación de administración non existe no caso dos episodios. Isto débese a que, ao ser engadidos automaticamente segundo a información recibida polo ficheiro RSS do programa (explicado máis en detalle na sección 7.2.2.1), non son editables. Aquelas opcións que poidan afectar en bloque aos episodios dun programa considéranse xa opcións do programa.

Os superusuários do sistema, pola súa parte, poden editar e borrar calquera contido mediante ferramentas de Django coma o panel de administración ou o IPython shell.

7.2.2. Capa Vista

A capa vista é na que se atopa a lóxica funcional do aplicativo. Fai uso dos obxectos da capa modelo para, ou ben extraer e compñer os datos que serán enviados ao cliente ou ben recoller os datos enviados desde o cliente para facer as conseguintes modificacións na base de datos. O módulo central desta capa é `views.py`. Nel, defínense funcións e clases que reciben un obxecto de Django `HttpRequest` e unha serie de parámetros opcionais e, tras realizar as operacións necesarias, retornan un obxecto `HttpResponse`.

Un obxecto `HttpRequest` contén metadatos da petición que se executou desde o cliente: Información da sesión do usuario, tipo de petición (GET, POST...), datos necesarios para os posibles “middlewares” e datos que o cliente envía a través dos formularios.

Os middlewares, en Django, son plugins que alteran globalmente a entrada e saída do procesamento das peticións e as respuestas. O framework oferta certa variedade de plugins de serie e a posibilidade de crear middlewares propios. Comentaranse os utilizados no capítulo de Implementación.

Un obxecto `HttpResponse` contén os datos que han de ser devoltos ao cliente, incluíndo a páxina á que se ha de redirixir ao usuario por causa da petición. Django require gardar os datos nun dicionario clave-valor ao que chama `context`. Esas claves valerán para acceder aos valores no template á hora de renderizar a resposta.

A decisión de a qué instancia ou función se lle pasa o obxecto `HttpRequest` tómase en base á url destino enviada polo cliente xunto co propio obxecto. É necesario manter un módulo que relate as urls coas funcións e clases correspondentes á operación a realizar. Comunmente en Django e tamén neste proxecto, este ficheiro chámase `urls.py`.

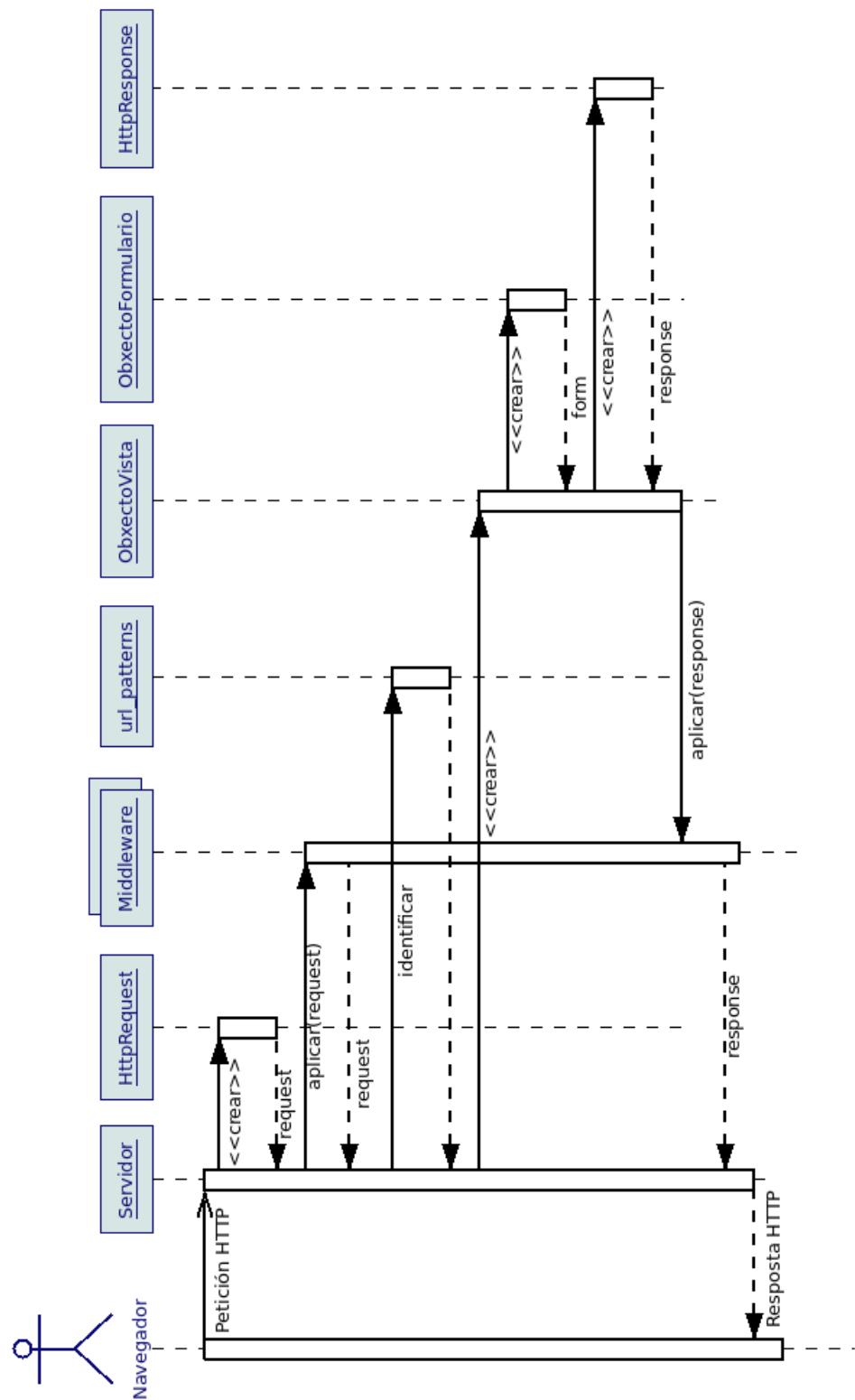


Figura 7.4: Esquema do funcionamento das vistas.

Na figura 7.4 amósase un esquema do funcionamento da capa vista. Para unha maior claridade, condensáronse as distintas clases de middleware nun único paso do diagrama. Tamén nesa figura pode verse unha referencia ao obxectos Form de Django que son utilizados neste proxecto. Estes obxectos consisten nunha abstracción a obxecto de Python da información procedente dos formularios despregados no template e dos contedores de dita información (selectores, caixas de texto...). O seu uso é opcional, pero simplifican a definición valores por defecto e a validación dos datos.

7.2.2.1. Engadir contenido

Explicarase en detalle esta vista por ser, a posibilidade de que os usuarios engadan os seus propios programas e os episodios, a parte principal deste proxecto. Un usuario, pode engadir un programa e todos os seus episodios simplemente proporcionándolle ao aplicativo o enlace ao seu ficheiro de RSS.

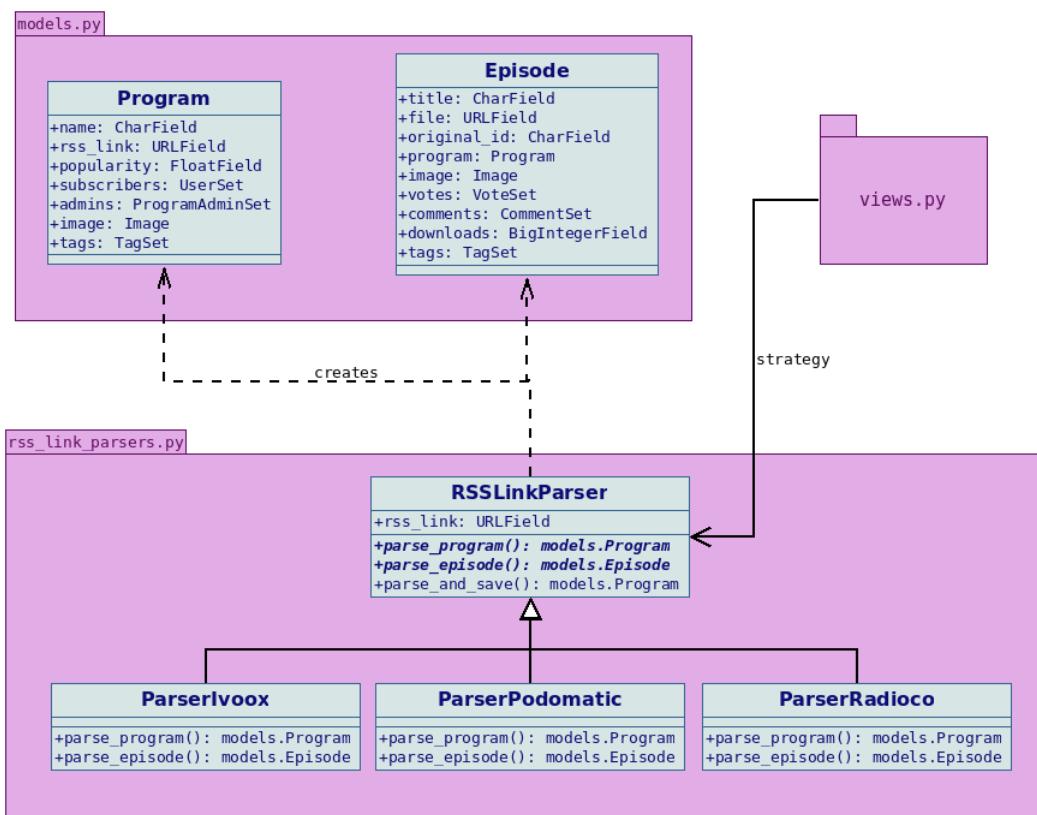


Figura 7.5: Patrón Estratexia utilizado para o procesamento de RSS.

A función de vista encargada de realizar este traballo recibe o obxecto de request e carga os datos desde nun formulario de Django. Unha vez validado, accede ao ficheiro RSS e extrae os datos do programa e os episodios. Enfrontámonos aquí a unha limitación do

proxecto: Non existe un estándar de campos de RSS. Distintos servidores de podcasting poden ter distintos formatos.

Debido a isto, debía deseñarse o sistema de xeito que puidésemos contar con distintos algoritmos de interpretación do RSS e, de cara a unha continuación do desenvolvemento, que engadir novos algoritmos fose sinxelo. De modo que se decidiu aplicar o patrón de deseño “estratexia”, como se ve no diagrama 7.5. A superclase, RSSLinkParser, implementa o método parse_and_save, encargado da creación das novas instancias. Esta función utiliza os métodos de lectura da información de programa e episodio, pero deixa a súa implementación ás clases fillas. Actualmente, o proxecto soporta 3 tipos ficheiro RSS dos máis populares entre os usuarios obxectivo.

7.2.3. Capa Template

A capa template define a forma na que os datos obtidos na capa vista serán amosados ao usuario e tamén os métodos de entrada de datos que poremos a disposición deste. A idea xeral foi a dunha interface na que os datos simples (atributos) se amosan en forma de lista e os complexos (Entidades) en forma de cuadrícula. Cada elemento dessa cuadrícula representa un programa, episodio ou emisora da forma representada na figura 7.6 e constitúe un enlace á páxina de detalles dessa entidade.

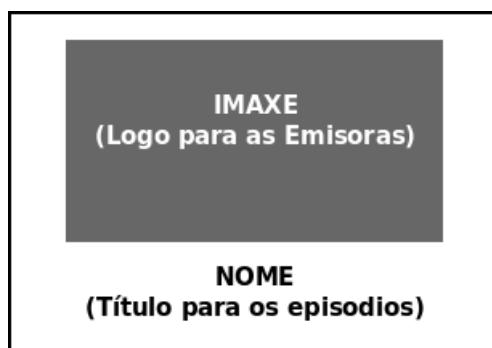


Figura 7.6: Esquema dun elemento da cuadrícula.

Pensando no uso da web en dispositivos móbiles, o número de elementos por fila da cuadrícula varía dependendo do tamaño da pantalla. Aquellas páxinas divididas de xeito horizontal (esquerda e dereita) pasan a verticais (arriba e abaixo) pois considerase o “scroll” vertical más cómodo para o usuario destes dispositivos.

Todas as páxinas levan unha cabeceira amosando a información transversal á páxina: Un enlace á portada, botón de inicio/peche de sesión, unha caixa de procura por texto e máis o selector de linguaxe.

A continuación coméntase brevemente as diferentes partes da interface:



Figura 7.7: Extracto do borrador do deseño da interface. Portada.

7.2.3.1. Portada

Pensouse na portada coma unha páxina que ofrece información breve e xeral que o usuario queira consultar con máis frecuencia, por exemplo, as novidades nas súas subscricións e o acceso rápido ás súas emisoras favoritas. Tamén, pensando nos usuarios anónimos,

considerouse engadir información de interese xeral coma os episodios máis recentes ou os programas más “populares”, concepto que se explica na sección 7.3.

Na figura 7.7 amósase un primeiro borrador desta páxina. Móstrase a información dos programas e os episodios en forma de grella na parte esquerda e das emisoras en forma de lista no lado derecho. Isto cambia nos pequenos dispositivos, onde a información das emisoras pasa a situarse na parte máis baixa da páxina.

7.2.3.2. Páxinas de engadir e editar contenido

As páxinas de engadir contenido (inclusa a de rexistrar usuario) consisten na lista de atributos da entidade a engadir xunto cos seus respectivos elemento HTML de entrada de datos. As de edición son semellantes, pero amosando os valores actuais da entidade.

7.2.3.3. Páxinas de detalle

Ás páxinas de detalle son aquelas onde se amosa a información completa dunha instancia dalgúnha das 4 grandes clases deste proxecto: Usuario, emisora, programa e episodio. Tamén debería dar acceso á páxina de edición de ter, o usuario, permisos para iso.

Na figura 7.8 amosamos un borrador da páxina de detalles de programa que nos ha valer coma exemplo xeral: Encabézase coa imaxe da instancia e o seu nome. Debaixo, lístanse os seus atributos incluíndo o reprodutor de audio para os casos de emisora e episodio. Finalmente, as cuadrículas dos obxectos cos que se relacionan.

A máis distinta sería, se cadra, a de episodio. Esta incluiría ao final unha sección de comentarios en lugar dunha grella de obxectos.

7.2.3.4. Páxina de resultados de busca

A páxina de resultados de busca é á que se accede a través da caixa de procura da cabecera (busca por texto) ou a través de “clicar” nunha etiqueta de categoría (busca por tag). Divídese en dúas partes en proporcións semellantes á de portada: A parte da esquerda amosa as entidades atopadas en forma de cuadrícula e a da dereita unha nube de etiquetas. Ao igual que na páxina de portada, a parte da dereita trasládase abaixo nos dispositivos pequenos.

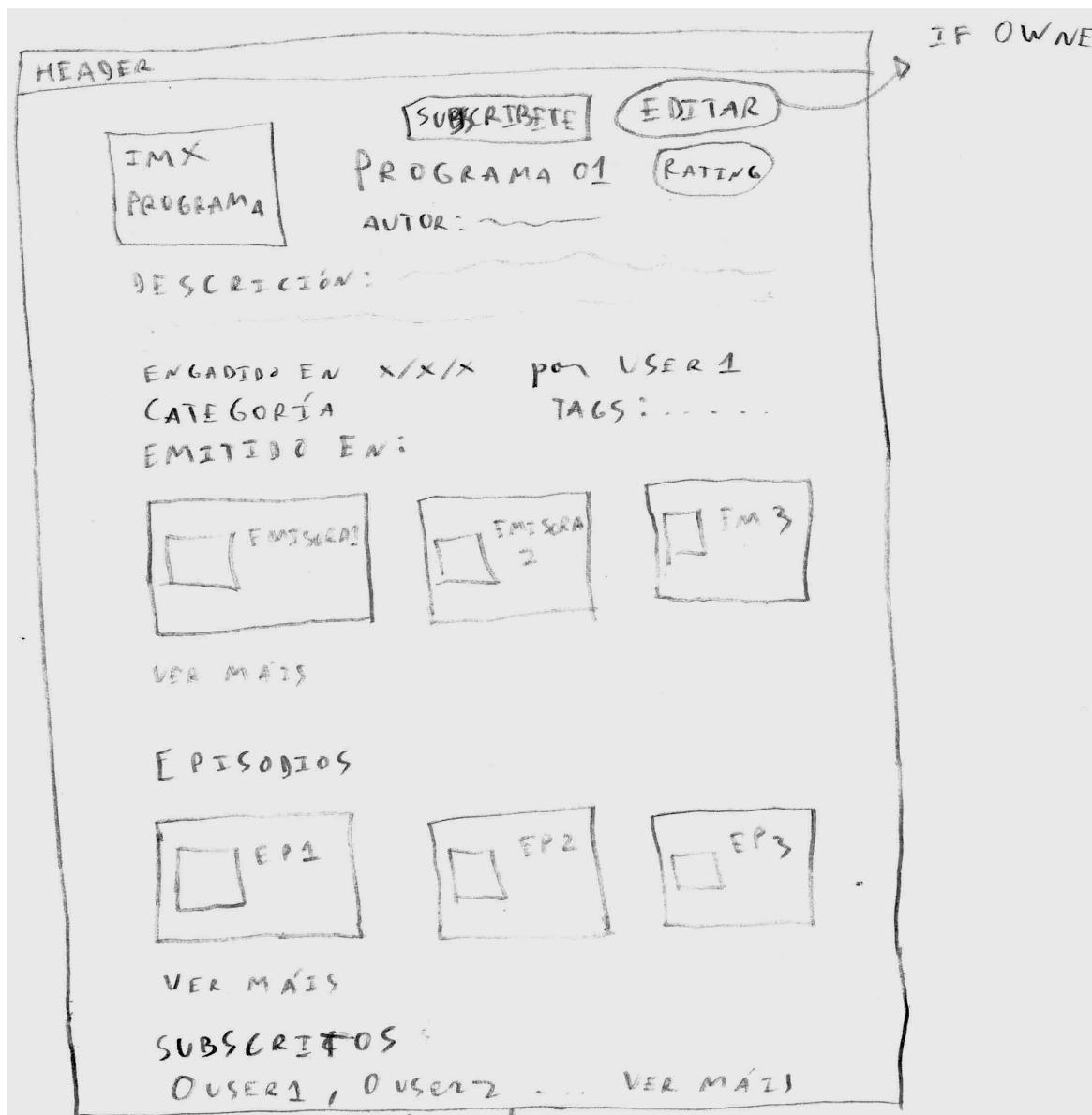


Figura 7.8: Extracto do borrador do deseño da interface. Detalles de programa.

7.2.3.5. Páxinas de xestión

As páxinas de xestión de emisión e xestión de administradores para programas e emisoras son semellantes. A figura 7.9 mostra un esquema da páxina que permitiría xestionar os programas emitidos por unha emisora. Aínda que o aspecto aí debuxado cambiou moito durante a implementación, serve para dar unha idea das necesidades de deseño: O usuario xestor ten que poder ver os programas xa emitidos para poder eliminarlos da emisión e ver os non emitidos para poder engadilos ao catálogo da emisora.

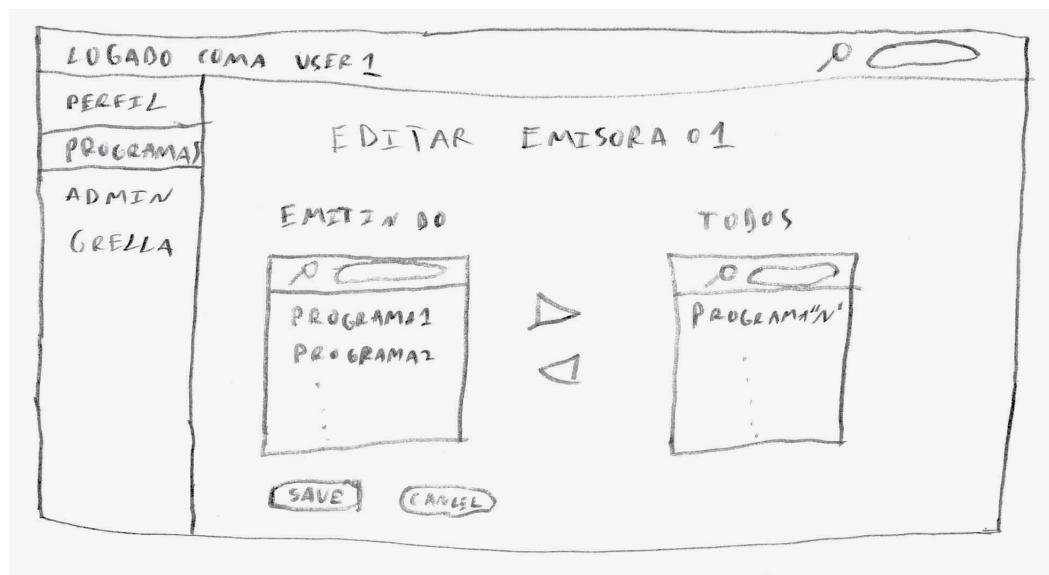


Figura 7.9: Extracto do borrador do deseño da interface. Xestión de emisión.

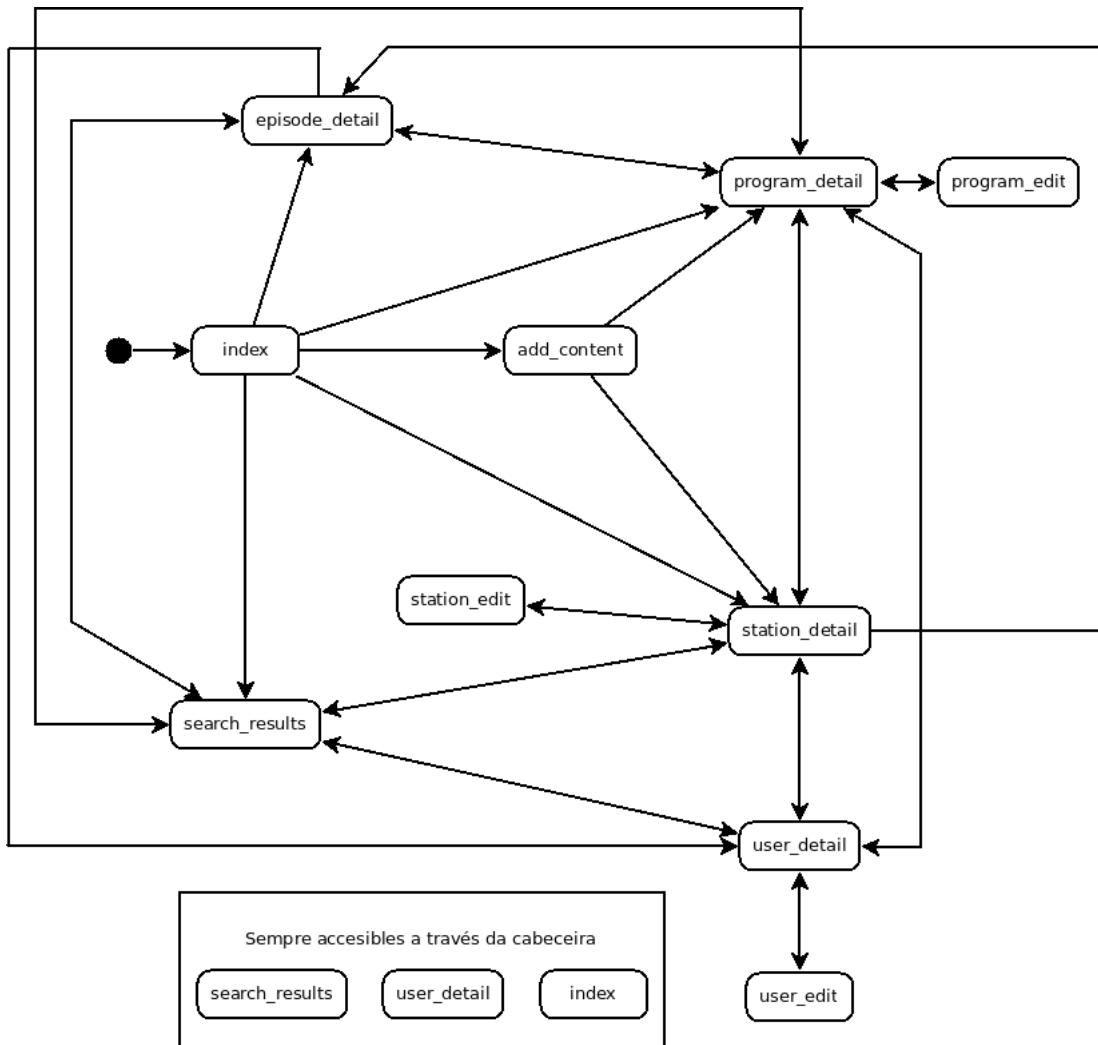


Figura 7.10: Mapa de navegación entre as distintas páxinas.

7.3. Actualización dos datos

Existen dous procesos executados periodicamente no lado do servidor coa fin de manter actualizados os datos dos programas no sistema.

7.3.0.1. Novos episodios

Cando un programa é engadido, todos os episodios presentes no ficheiro RSS gárdanse tamén. A partir de entón, é responsabilidade do sistema facer “polling” do RSS para

detectar as novas publicacións e engadilas á base de datos. Este proceso reutiliza o código do módulo `rss_link_parsers` visto na sección 7.2.2.1.

7.3.0.2. Popularidade e cualificación dos programas

A cualificación (rating) dun programa calcúlase en base aos votos positivos e negativos. Só se teñen en conta os episodios con data de publicación entre os últimos 365 días. Se non hai votos, asúmese unha cualificación do 50 %:

$$cualificacion_p = 100 \frac{\sum_{episodios_p} vpositivos}{\sum_{episodios_p} vpositivos + vnegativos}$$

A popularidade é unha medida do impacto que os programas teñen no sistema e, coma tal, inflúe na súa visibilidade en portada e resultados de procura. Depende das subscricións, dos votos ponderados dos episodios, das escoitas dos episodios e do número de emisoras que o emitan. Tampouco aquí se teñen en conta os episodios con máis dun ano de antigüidade.

$$vponderados_e = (vpositivos - vnegativos) * (vpositivos + vnegativos)$$
$$popularidade_p = 5 * subscricions + 10 * emisoras + escoitas + \sum_{episodios_p} vponderados_e$$

A ponderación de emisoras e subscricóns decidiuse en base a diferencia de magnitud esperada. A ponderación dos votos é en base ao número de votos de cada capítulo.

Ámbolos dous valores, ao depender de actividades dos usuarios, han de ser actualizados periodicamente.

Capítulo 8

Implementación

8.1.	Estrutura xeral do código fonte	76
8.2.	Ficheiros de imaxe	77
8.3.	A vista de engadir programa	78
8.4.	A tradución	79
8.5.	O contador de escoitas	80
8.6.	O panel de xestión	80
8.7.	Execución dos procesos en Celery	82
8.8.	Adaptabilidade a dispositivos móbiles	82

Neste capítulo comentarase a implementación daquelas partes do proxecto que merezan unha explicación máis detallada.

8.1. Estrutura xeral do código fonte

Seguiuse a estrutura xenérica dos proxectos de Django que se amosa na figura 8.1. Creáronse dous paquetes: radio01 e rss_feed, dos cales só se amosa expandido o primeiro. Para máis detalles do segundo, pódese ver a figura 7.1 .

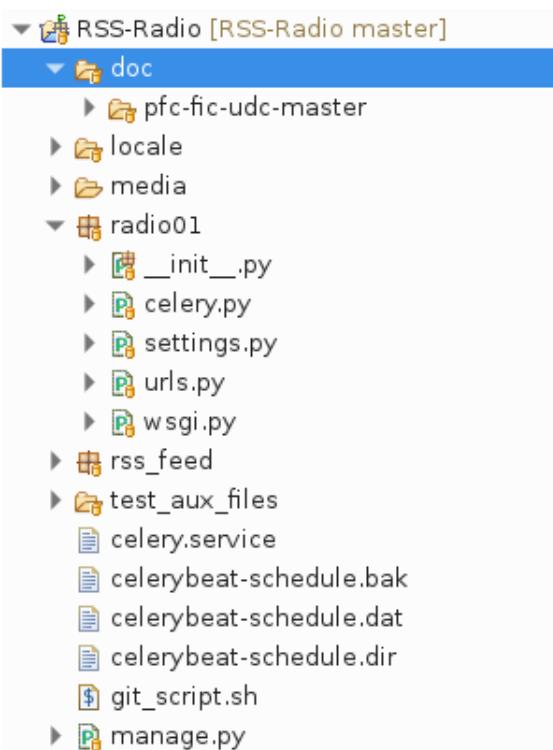


Figura 8.1: Estrutura do código fonte do proxecto.

A motivación destes dous paquetes é separar o código propio da aplicación, no de rss_feed, e os ficheiros de configuración do servizo, no de radio01. Deste modo sería posible, de desexalo, crear novas aplicacións e poñelas a funcionar sobre o mesmo servidor sen necesidade de facer cambios nas que xa están a funcionar.

En radio01 atópanse os seguintes ficheiros:

- **settings.py:** Neste defínense, entre outras cousas, as aplicacións utilizadas por django (celery, django-bootstrap4...), os middlewares utilizados, os datos de conexión á base de datos e os subdirectorios que o sistema ha de coñecer.

- **urls.py:** Relaciona cada aplicación coa súa URL. Nos paquetes de aplicación tamén hai un urls.py que define, dentro da aplicación, URL's relativas á definida no do paquete xeral. Neste proxecto, por exemplo, definiuse *servername[:Porto]/rss_feed* coma URL da aplicación, polo tanto, para acceder unha vista definida coma “vista1” de rss_feed teríase que ir a *servername[:Porto]/rss_feed/vista1*
- **celery.py:** Ficheiro de configuración de Celery.
- **wsgi.py:** Configuración da interface WSGI para aplicacóns de Python (ver capítulo 3)

En rss_feed atópanse os módulos propios da aplicación, incluíndo o código dos procesos periódicos do servidor dos que xa se falou en capítulos anteriores.

8.2. Ficheiros de imaxe

A diferenza dos ficheiros de audio, as imaxes amosadas si son gardadas en almacenamento propio. Serializar os ficheiros de imaxe para gardalos en base de datos non ofrece a penas vantaxes e sí incrementan o custo en almacenamento de xeito sensible, polo que se decidiu gardar os ficheiros nun directorio do servidor e, na base de datos, os seus metadatos e a ruta ao ficheiro.

As imaxes son recollidas de dúas formas:

- **Subida manual:** Utilízase para os avatares de usuario e as imaxes da emisora.
- **Descarga de un servidor externo:** Ao engadir un programa, as imaxes asociadas a eles e aos seus episodios descárganse do enlace que da o campo de imaxe de RSS (se existe). Dado que moitos servidores, por motivos de seguridade, bloquean as peticóns de axentes descoñecidos, utilizáronse as cabeceiras de Firefox. Pode verse o código no listado 8.1.

En calquera dos dous casos, as imaxes gárdanse no directorio “media”, agrupadas en subcarpetas por ano e mes. Nese mesmo directorio hai outra subcarpeta “default” onde se gardan as imaxes por defecto para aquelas entidades sen unha de seu.

```
1 | def create_image(image_url):  
2 |     creation_date = timezone.now()  
3 |     original_image_name = os.path.basename(image_url)  
4 |     image_name = creation_date.strftime("%d %H %M %S") + '-' + original_image_name.  
5 |             lower()
```

```
6      opener = urllib.request.build_opener()
7      opener.addheaders = [('User-Agent', 'Mozilla/5.0')]
8      urllib.request.install_opener(opener)
9      image_file = urllib.request.urlretrieve(image_url)
10
11     with open(image_file[0], 'rb') as ifd:
12
13         new_image_instance = Image()
14         new_image_instance.path.save( image_name, File(ifd) )
15
16         new_image_instance.creation_date = creation_date
17         new_image_instance.name = original_image_name
18         new_image_instance.alt_text = original_image_name.lower()
19         new_image_instance.original_url = image_url
20
21         new_image_instance.save()
22
23
24     return new_image_instance
```

Listado 8.1: Código da función `create_image` do módulo `rss_link_parsers.py`

8.3. A vista de engadir programas

Como se mencionou no capítulo 7, empregouse un patrón **estratexia** para o deseño dos lectores de RSS. Implementouse, para isto, unha superclase **RSSLinkParser** cun método *parse_and_save* que crea os novos programas e episodios. Ese método apóiase en dous auxiliares que serán implementados polas súas clases filhas:

- **parse_program:** Identifica os campos do programa do RSS e devolve un obxecto Program.
- **parse_episode:** Identifica os campos de episodio e devolve un obxecto Episode. Execútase unha vez por elemento na lista de episodios do RSS.

Ningún dos dous métodos crea entradas novas na base de datos, iso corre da conta da superclase.

Desta forma, basta con instanciar o tipo de *RSSLinkParser* axeitado antes de aplicar o método de creación dos obxectos. A decisión de qué subclase utilizar, tómase na función *add_content* do módulo `views.py` en base a palabras clave contidas no enlace RSS introducido polo usuario (ver formulario de creación de programa na figura 8.2). De non atopar ningunha delas, probará con todos os parsers existentes ata atopar o primeiro que non levante unha excepción.

Se ningún parser da interpretado o RSS, o usuario será redirixido a unha páxina de erro.

The screenshot shows a web interface for adding new content to an RSS feed. At the top, there's a blue header bar with the text "RSS Radio" and user information "Logged as ferblee". Below the header, the main title is "Add New Content:". There are two buttons: "Add New Program" and "Add New Station". The "Add New Program" section contains a field labeled "New Program:" with a placeholder "RSS Link:" and a note "This field is required." Below it is a "Website:" field. Under "Sharing mode:", there's a dropdown menu set to "share free". Under "Enable episode comments:", there's another dropdown menu set to "Enable". At the bottom of the form are two buttons: "Send" and "Cancel".

Figura 8.2: Formulario de engadir programa.

8.4. A tradución

A tradución implementouse mediante as ferramentas que Django prové: *i18n* para a tradución de texto estático dos templates e *gettext* para o texto xerado no código Python. Ambas válense do paquete *gettext* de GNU.

O procedemento consiste elixir un idioma base e etiquetar aquelas cadeas de texto que sexan sensibles de seren traducidas. Unha vez estas estean marcadas, utilizaranse as ferramentas de Django para xerar un **ficheiro de extensión .po** por cada lingua que se queira habilitar. Estes ficheiros créanse no directorio marcado no ficheiro de `settings.py`, no caso deste proxecto, a carpeta “`locale`” que se ve na figura 8.1.

Os arquivos `.po` son ficheiros de texto que relacionan as cadeas de texto no idioma de base coa súa tradución no idioma elixido. Móstrase un exemplo na figura 8.3. Neste proxecto elixiuse o Inglés coma lingua base e escribiuse unha tradución ao Galego.

```
#: rss_feed/forms.py:92 rss_feed/forms.py:157
msgid "Location"
msgstr "Localización"

#: rss_feed/forms.py:93 rss_feed/forms.py:110 rss_feed/forms.py:156
msgid "Description"
msgstr "Descripción"

#: rss_feed/forms.py:96 rss_feed/forms.py:113
msgid "Avatar"
msgstr "Avatar"

#: rss_feed/forms.py:124
msgid "Old password"
msgstr "Antigo password"

#: rss_feed/forms.py:125
msgid "New password"
msgstr "Novo password"
[]

#: rss_feed/forms.py:126
msgid "New password confirmation"
msgstr "Confirmar novo password"

#: rss_feed/forms.py:140
msgid "RSS Link"
```

40,0 -1

2%

Figura 8.3: Fragmento do ficheiro .po para a tradución ao Galego.

8.5. O contador de escoitas

Implementouse un contador das escoitas de cada capítulo. Cando un usuario comeza a reproducir o ficheiro de audio, mándase unha petición asíncrona ao servidor mediante **Ajax** para incrementar en 1 as escoitas do episodio na base de datos. Ese dato non é refrescado na interface automaticamente para non facer este proceso excesivamente transparente ao usuario.

Ao realizar esta acción, engádese o identificador do episodio a unha lista que se mantén nos **datos da sesión**, xa sexa un usuario identificado ou anónimo, de forma que nunha única sesión non se poida incrementar en máis de 1 as escoitas a un mesmo episodio.

Isto faise para evitar, na medida do posible, as escoitas fraudulentas xa que este valor é un parámetro para o cálculo da popularidade.

8.6. O panel de xestión

O panel de xestión de emisora e programa comparten gran parte do código, así que poremos o primeiro coma exemplo dos dous. O panel de xestión componse de 4 vistas:

- **Edición de perfil:** Para cambiar o logo da emisora, o nome, a localización...
- **Xestión da emisión:** Mostrada na figura 8.4. É na que se dan de alta/baixa os programas a emitir ou se cambia o seu horario.
- **Administración:** Só visible a administradores de nivel “propietario”. Serve para dar de alta/baixa os administradores ou se cambian os seus permisos.
- **Borrado:** Só visible a administradores de nivel “propietario”. Serve para borrar a emisora, para o que se require unha confirmación previa.

Para o caso da xestión de emisión atopámonos co problema de como darlle ao usuario unha ferramenta para seleccionar calquera programa existente na base de datos, pois pode ser unha lista moi extensa. Solucionouse isto aplicando unha ferramenta de selección de jQuery consistente nunha lista despregable que permite a busca por texto sobre a lista de opcións coa que sexa cargada (ver figura 8.4).

Unha solución semellante se aplicou na vista de administración, pois a lista de usuarios a seleccionar tamén pode ser moi longa.

Figura 8.4: Panel de xestión dunha emisora. Vista de xestión da emisión.

8.7. Execución dos procesos en Celery

Para correr procesos de actualización sobre Celery, declaráronse os “endpoints” deses procesos no ficheiro `celery_endpoints.py` do paquete `rss_feed`. Os endpoints chámense `update_rss_info_daemon`, para o proceso de actualización de episodios e `update_popularity_rating_daemon` para o do cálculo da popularidade.

Eses endpoints serán utilizados polo worker de Celery para identificar os procesos e para poder configurar a súa periodicidade na sección de Celery do panel de administración de Django, como se ve na figura 8.5.

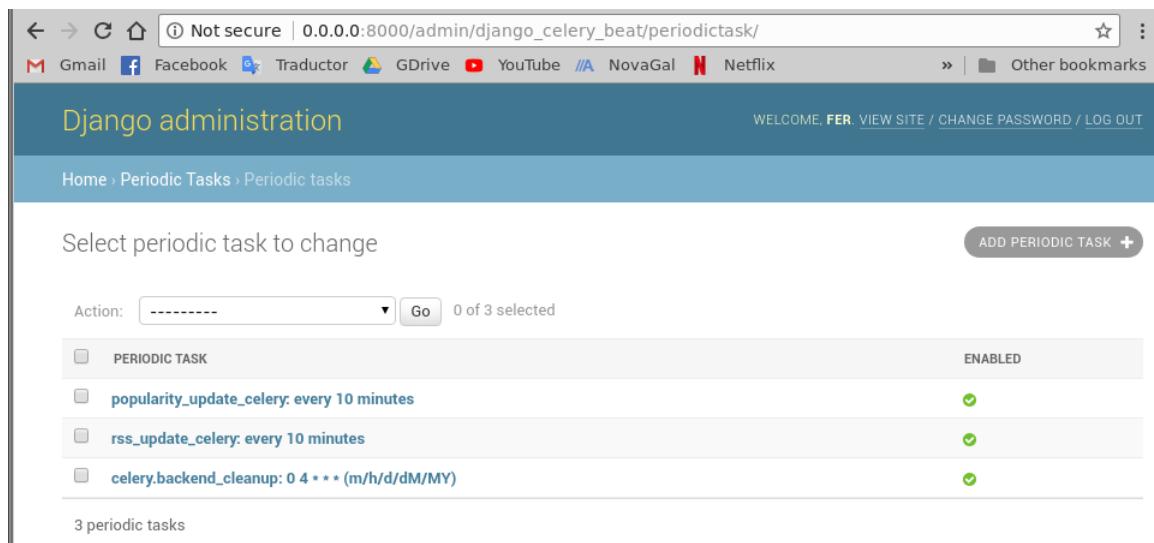


Figura 8.5: Páxina de administración da aplicación, sección de procesos de Celery.

8.8. Adaptabilidade a dispositivos móveis

Utilizouse a regra de CSS `@media` para definir diferentes estilos dependendo do tamaño da pantalla na que se amose o portal web. O listado 8.2 amosa o código para que un CSS-grid de 4 columnas pase a ser de 2 se a pantalla do dispositivo ten menos de 700 píxeles de ancho.

```
1 @media only screen and (min-width: 700px) {  
2     .grid-item{  
3         border: 1px solid rgba(0, 0, 0, 0.8);  
4         text-align: center;  
5         padding: 20px;  
6     }  
7     .grid-container{  
8 }
```

```
9   display: grid;
10  grid-template-columns: [col1-start]1fr [col2-start]1fr [col3-start]1fr [
11    col4-start]1fr ;
12  grid-gap: 20px;
13 }
14 }
15 @media only screen and (max-width: 700px) {
16
17 .grid-item{
18   border: 1px solid rgba(0, 0, 0, 0.8);
19   text-align: center;
20   padding: 5px;
21 }
22 .grid-container{
23   display: grid;
24   grid-template-columns: [col1-start]1fr [col2-start]1fr;
25   grid-gap: 2px;
26 }
27 }
```

Listado 8.2: Extracto da folla de estilos de index.html

Capítulo 9

Probas do sistema

9.1. Probas de unidade	86
9.2. Probas de integración	91

Neste capítulo explicarase o proceso de probas ao que se someteu o sistema desenvolvido. Distinguimos entre probas de unidade e probas de integración.

9.1. Probas de unidade

Son aquelas que verifican o correcto funcionamento individual das componentes. Para realizar este tipo de probas, cómpre definir un entorno independente para cada unha de xeito que os resultados das anteriores non inflúan nas posteriores. Como consecuencia da utilización da metodoloxía TDD (ver capítulo 4) os casos de proba definidos estarán automatizados. Isto fai posible executalos cando se fagan cambios no código co fin de detectar a aparición de efectos non desexados. O código correspondente pode atoparse no ficheiro “tests.py” incluido no proxecto (ver figura 7.1)

Utilizouse a biblioteca “django.test” de Django, a cal, mediante o uso do paquete “unit-test” estándar de Python, permite a escritura das probas de unidade. Defínense, para isto, conxuntos de casos de proba en forma de clase que ha ser herdeira da clase TestCase, incluída na biblioteca. Confecciónanse, a continuación, as probas en forma de métodos cuxo nome ha levar o prefixo “test”.

Para asegurar a independencia entre probas, Django creará unha base de datos temporal que será borrada automaticamente unha vez os tests finalicen, independentemente do seu éxito. Isto implica que cada instancia filla de TestCase teña que popular a base de datos coma paso previo á execución dos tests, podendo isto facerse en cada proba ou declarando un método setUp.

```
1 class ProgramModelTests(TestCase):
2
3
4     def setUp(self):
5
6         rssl = 'http://dummy_link.xml'
7         p_image = Image.objects.create(name='p_image1', path='path/to/p_image1')
8
9         p1 = Program.objects.create(name='TestProgram1', rss_link=rssl, image=p_image
10             )
11
12         u1 = User.objects.create(username='userp1')
13         u2 = User.objects.create(username='userp2')
14         User.objects.create(username='userp3')
15
16         p1.programadmin_set.create(user=u1, type=ADMT_OWNER[0])
17         p1.programadmin_set.create(user=u2, type=ADMT_ADMIN[0])
18
19         t1 = Tag.objects.create(name='pt1', times_used=1)
20         t2 = Tag.objects.create(name='pt2', times_used=2)
```

```

20
21     p1.tag_set.add(t1)
22     p1.tag_set.add(t2)
23
24
25     def test_check_user_is_admin(self):
26
27         p1 = Program.objects.get(name='TestProgram1')
28         u1 = User.objects.get(username='userp1')
29         u2 = User.objects.get(username='userp2')
30         u3 = User.objects.get(username='userp3')
31
32         self.assertTrue(p1.check_user_is_admin(u1, ADMT_OWNER[0]))
33         self.assertFalse(p1.check_user_is_admin(u1, ADMT_ADMIN[0]))
34         self.assertTrue(p1.check_user_is_admin(u2, ADMT_ADMIN[0]))
35         self.assertFalse(p1.check_user_is_admin(u2, ADMT_OWNER[0]))
36         self.assertFalse(p1.check_user_is_admin(u3))
37
38
39     def test_delete(self):
40
41         p1 = Program.objects.get(name='TestProgram1')
42         p1_id = p1.id
43         img_id = p1.image.id
44
45         p1.delete()
46
47         p2 = list(Program.objects.filter(pk=p1_id))
48         self.assertEqual(p2, [])
49
50         img2 = list(Image.objects.filter(pk=img_id))
51         self.assertEqual(img2, [])
52
53         t1 = Tag.objects.get(name='pt1')
54         t2 = Tag.objects.get(name='pt2')
55
56         self.assertEqual(t1.times_used, 0)
57         self.assertEqual(t2.times_used, 1)

```

Listado 9.1: Probas de unidade dos métodos de Program

O código amosado no exemplo 9.1 é o utilizado nas probas de unidade da clase Program. Primeiro, o método *setUp* crea na base de datos temporal o programa *p1*, 3 usuarios dos cales lle asigna 2 como administradores e más 2 tags que tamén lle asigna. Nótese que se utiliza un método de creación propio dun obxecto imaxe, iso é posible porque xa ten a súa propia proba de unidade executada anteriormente no mesmo ficheiro. O código xa probado pasa a considerarse seguro e pode ser utilizado nos tests seguintes.

O primeiro test comproba que o método *check_user_is_admin(user,[type])* de Program responda correctamente á pregunta de se o usuario dado é administrador dese programa e cos permisos especificados. O segundo comproba se o método de borrado borra tamén

Capítulo 9. Probas do sistema

a imaxe asignada e fai decrecer o contador de uso dos tags. Probas semellantes se fixeron para as clases do módulo models.py que posúen métodos propios.

```
1 class ParserIvooxRSSLPTests(TestCase):
2
3
4     def initialize_test(self):
5
6         RSS_file = TEST_AUX_FILE_PATH + 'hasta-los-kinders_original.xml'
7         feed_dict = feedparser.parse(RSS_file)
8         u1 = User.objects.create(username='userRSSLP1')
9
10    return ParserIvoox(RSS_file,u1),feed_dict
11
12
13    def test_get_entry_list(self):
14
15        rlp,fd = self.initialize_test()
16        el = rlp.get_entry_list(fd)
17
18        self.assertIsInstance(el,list)
19        self.assertIsInstance(el[0],dict)
20
21
22    def test_parse_program(self):
23
24        program_web = 'http://www.ivoox.com/podcast-hasta-los-kinders_sq_f14062_1.
25        html'
26        rlp,fd = self.initialize_test()
27
28        p1 = rlp.parse_program(fd)
29
30        self.assertEqual(p1.name,'Hasta Los Kinders')
31        self.assertEqual(p1.language,'es-ES')
32        self.assertEqual(p1.original_site,program_web)
33        self.assertEqual(p1.rss_link_type,IVOOX_TYPE[0])
34
35
36    def test_parse_episode(self):
37
38        rlp,fd = self.initialize_test()
39
40        #Expected info
41        exp_titlte = 'CiudadanoKinders: Como hacer un monólogo'
42        exp_pub_date = datetime.datetime(2010, 9, 17, 20, 45, 24, tzinfo=pytz.utc)
43        exp_file = 'http://www.ivoox.com/ciudadanokinders-como-hacer-
44        monologo_mf_368919_feed_1.mp3'
45        exp_web = 'http://www.ivoox.com/ciudadanokinders-como-hacer-monologo-audios
46        -mp3_rf_368919_1.html'
47        exp_original_id = 'http://www.ivoox.com/368919'
48
49        p1 = rlp.parse_program(fd)
50        entry_dict = rlp.get_entry_list(fd)[0]
```

```

49     e1 = rlp.parse_episode(entry_dict,p1)
50
51     self.assertIsInstance(e1,Episode)
52     self.assertEquals(e1.title,exp_titlte)
53     self.assertEquals(e1.publication_date,exp_pub_date)
54     self.assertEquals(e1.file,exp_file)
55     self.assertEquals(e1.original_site,exp_web)
56     self.assertEquals(e1.original_id,exp_original_id)
57
58
59 # From superclass
60 def test_parse_and_save(self):
61
62     rlp,_ = self.initialize_test()
63
64     p1 = rlp.parse_and_save()
65     self.assertIsInstance(p1,Program)
66
67     p1_id = p1.id
68     p2 = list(Program.objects.filter(pk=p1_id))
69     self.assertNotEqual(p2,[])
70
71     p2 = p2[0]
72
73     self.assertEqual(p2.tag_set.count(),1)
74     self.assertEqual(p2.tag_set.all()[0].name,'comedy')
75     self.assertIsInstance(p2.image,Image)
76
77
78     self.assertEqual(p2.episode_set.count(),20)
79
80     e1 = p2.episode_set.filter(title='CiudadanoKinders: Como hacer un monologo',
81         )
82     self.assertNotEqual(e1,[])
83
84     e1 = e1[0]
85     self.assertIsInstance(e1,Episode)
86
87     self.assertEqual(e1.image,p2.image)
88
89     # Clean copied image
90     p2.image.delete()

```

Listado 9.2: Probas de unidade dos métodos de ParserIvoox

No exemplo 9.2 atópase o código que valeu para probar o funcionamento dun dos intérpretes de RSS do módulo rss_link_parser.py (ver sección 7.2.2.1), concretamente o correspondente aos ficheiros co formato de Ivoox. Tanto con este coma cos outros “parsers” probáronse as subrutinas auxiliares e máis os métodos *parse_program* e *parse_episode*. No amosado, inclúese tamén o test da función *parse_and_save* herdada da superclase.

Para realizar as anteriores probas utilizáronse ficheiros RSS reais descargados a disco local. Dado que o obxectivo último desta clase é a creación de obxectos a partir da información do RSS, as probas enfócanse en comprobar se o gardado en base de datos coincide co esperado.

```
1  class UpdatePopularityDaemonTests(TestCase):
2
3
4      def setUp(self):
5
6          u1 = User.objects.create(username='userUPD1')
7          u2 = User.objects.create(username='userUPD2')
8          s1 = Station.objects.create(name='RadioTest2')
9
10         RSS1 = TEST_AUX_FILE_PATH + 'hasta-los-kinders_original.xml'
11         rlp = ParserIvoox(RSS1,u1)
12         hlk = rlp.parse_and_save()
13
14         hlk_ep1 = hlk.episode_set.all()[0]
15         hlk_ep1.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u1,episode=
16             hlk_ep1))
17
18         hlk_ep2 = hlk.episode_set.all()[1]
19         hlk_ep2.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u1,episode=
20             hlk_ep2))
21
22         hlk_ep3 = hlk.episode_set.all()[2]
23         hlk_ep3.vote_set.add(Vote.objects.create(type=DISLIKE_VOTE[0],user=u1,
24             episode=hlk_ep3))
25         hlk_ep3.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u2,episode=
26             hlk_ep3))
27
28         RSS2 = TEST_AUX_FILE_PATH + 'falacalado_podomatic.xml'
29         rlp = ParserPodomatic(RSS2,u1)
30         fc = rlp.parse_and_save()
31         fc.subscribers.add(u1)
32         fc.broadcast_set.add(Broadcast.objects.create(station=s1,program=fc,
33             schedule_details='Friday 21:00'))
34
35         fc_ep1 = fc.episode_set.all()[0]
36         fc_ep1.downloads = 1000
37         fc_ep1.save()
38
39
40     def test_update_pop_rating_all_programs(self):
41
42         hlk = Program.objects.get(name='Hasta Los Kinders')
43         fc = Program.objects.get(name="fala calado's podcast")
44
45         self.assertEqual(hlk.rating,50)
46         self.assertEqual(fc.popularity,0)
47
48         #Ignore 365 days limitation
```

```

44     update_pop_rating_all_programs(days=0)
45
46     hlk2 = Program.objects.get(name='Hasta Los Kinders')
47     fc2 = Program.objects.get(name="fala calado's podcast")
48
49     exp_pop = program_popularity_formula(1,1,0,1000)
50
51     self.assertEqual(hlk2.rating,75)
52     self.assertEqual(fc2.popularity,exp_pop)

```

Listado 9.3: Probas de unidade do proceso de actualización de popularidade

As funcións que componen os procesos executados polo servidor para actualizar os programas (ver sección 7.3) tamén foron sometidos a probas. No exemplo 9.3 vese o código do test de unidade do proceso que actualiza os campos de *rating* (cualificación) e *popularity* (popularidade) dos programas. Para iso, créase en *setUp* os programas *hlk* e *fc* mediante os parsers de Ivoox e Podomatic (previamente probados) respectivamente. Despois, manipúlanse as características valorables para o *rating* no primeiro e as valorables para *popularity* no segundo.

Ao comenzar o test, compróbase se os valores de ámbolos dous atributos son os iniciais, logo executase o código de actualización e, finalmente, compróbase se os valores cambiaron da forma esperada.

9.2. Probas de integración

Son aquelas que serven para verificar o traballo conxunto de distintas componentes. Estes casos de test, por regra xeral, tratan de probar as conexións entre componentes ignorando o funcionamento interno[47].

No módulo *views.py* atópanse as funcións que reciben os datos entrantes e pasan as respostas ao cliente. Para probalas foi necesario o uso da clase Client, disponible na biblioteca de Django utilizada para os tests. Esta clase interactúa co sistema a modo de navegador web sinxelo permitindo simular operacións de GET e POST, ver a cadea redirección entre os distintos templates e comprobar que os datos presentes no contexto son os axeitados.

```

1  class IndexViewTests(TestCase):
2
3
4  @classmethod
5  def setUpTestData(cls):
6
7      u1 = User.objects.create(username='userIV1')
8
9      RSS1 = TEST_AUX_FILE_PATH + 'hasta-los-kinders_original.xml'
10     rlp = ParserIvoox(RSS1,u1)

```

```
11     hlk = rlp.parse_and_save()
12     hlk.popularity = 10 # Check if first
13     hlk.save()
14
15     RSS2 = TEST_AUX_FILE_PATH + 'falacalado_podomatic.xml'
16     rlp = ParserPodomatic(RSS2,u1)
17     rlp.parse_and_save()
18
19     # Total of 10 + 2 programs
20     for i in range(1,11):
21
22         rssl = 'http://dummy_link_iv_' + str(i) + '.xml'
23         pname = 'ProgramIV_' + str(i)
24         Program.objects.create(name=pname,rss_link=rssl)
25
26     # Create 30 stations
27     for i in range(1,31):
28
29         sname = 'StationIV_' + str(i)
30         Station.objects.create(name=sname)
31
32
33     def test_view_uses_correct_template(self):
34
35         resp = self.client.get(reverse('rss_feed:index'))
36         self.assertEqual(resp.status_code, 200)
37
38         self.assertTemplateUsed(resp, 'rss_feed/index.html')
39
40
41     def test_program_list(self):
42
43         resp = self.client.get(reverse('rss_feed:index'))
44         self.assertEqual(resp.status_code, 200)
45
46         self.assertEqual( len(resp.context['program_list']),4)
47         self.assertTrue(resp.context['program_list'].has_next())
48         self.assertFalse(resp.context['program_list'].has_previous())
49
50         p1 = resp.context['program_list'][0]
51         p2 = Program.objects.get(name='Hasta Los Kinders')
52         self.assertEqual(p1,p2)
53
54         resp = self.client.get('/rss_feed/?p_page=3')
55         self.assertEqual(resp.status_code, 200)
56
57         self.assertEqual( len(resp.context['program_list']),4)
58         self.assertFalse(resp.context['program_list'].has_next())
59         self.assertTrue(resp.context['program_list'].has_previous())
```

Listado 9.4: Fragmento das probas da vista index

No exemplo 9.4 véñse algunas probas realizadas para a vista de “index”, que se corresponde coa páxina principal do aplicativo. Créanse primeiro, na función de clase *setUpTestData*, 12 programas e 20 emisoras co obxectivo de comprobar que o contexto está a pasar o número deles correcto, na orde correcta e cos datos axeitados a cerca da paxinación. Isto realiza no segundo test, o do método *test_program_list*, só para o caso dos programas (as emisoras tamén se proban, pero incluílo semella redundante). O primeiro test, *test_view_uses_correct_template*, comproba que se accede á vista index a través do template axeitado.

```

1  class AddContentViewTests(TestCase):
2
3
4  def setUp(self):
5
6      u2 = User.objects.create(username='userIV2')
7      u2.set_password('12345678A')
8      u2.save()
9
10
11     def test_login(self):
12
13         self.client.login(username='userIV2', password='12345678A')
14         resp = self.client.get(reverse('rss_feed:add_content'))
15
16         self.assertEqual(resp.status_code, 200)
17         self.assertEqual(str(resp.context['user']), 'userIV2')
18
19         self.assertTemplateUsed(resp, 'rss_feed/add_content.html')
20
21
22     def test_add_station(self):
23
24         self.client.login(username='userIV2', password='12345678A')
25
26         form_station = {'form_station-name': ['StationIV1'], 'form_station-logo': [
27             ''], 'form_station-profile_img': [''], 'form_station-broadcasting_method': ['fm'],
28         'form_station-broadcasting_area': [''], 'form_station-broadcasting_frequency': [''],
29         'form_station-streaming_link': [''], 'form_station-description': [''],
30         'form_station-website': [''], 'form_station-location': ['']}
31
32         self.client.post(reverse('rss_feed:add_content'), form_station)
33
34         s1 = Station.objects.get(name='StationIV1')
35
36         self.assertIsInstance(s1, Station)
37         self.assertEqual(s1.logo, Image.get_default_program_image())

```

Listado 9.5: Fragmento das probas da vista add_content

Aquelas vistas que requiran que o usuario estea identificado tamén poden ser probadas, como se ve no exemplo 9.5. O primeiro caso proba que o login funciona e que se utiliza o template correcto. O segundo, proba a inserción dunha nova emisora a través do formulario que se enviaría desde a interface web.

O código puramente de “front-end” (Javascript, HTML...) foi probado de xeito manual e non automatizado.

Capítulo 10

Conclusións

10.1. Coñecementos acadados	97
10.2. Futuros traballos	97

Neste capítulo porase en balance o traballo realizado e darase unha breve guía de melloras que poderían implementarse no futuro.

A aplicación web presentada nesta memoria cumpre cos obxectivos iniciais do proxecto (ver sección 1.3)

- Creouse un portal web que da **acceso a ficheiros de audio**, ou ben mediante streaming por Internet, ou ben por descarga directa, estando estes aloxados nun servidor alleo.
- Implementáronse funcionalidades para que os **usuarios engadan o seu propio contido**. Poden engadir as súas emisoras ou programas de forma manual. No caso dos episodios, son creados automaticamente mediante os algoritmos de lectura de ficheiros RSS. Estes mesmos algoritmos, agrupan os arquivos de audio por programa e categoría.
- Puxérонse a disposición dos usuarios **ferramentas de procura** de contidos por texto e más por etiqueta.
- Habilitouse, para os usuarios, un **sistema de subscrición** aos programas e de seguimento das emisoras.
- Deseñouse un esquema de roles e permisos para facilitar a **colaboración entre os usuarios** nas tarefas de xestión de contidos.
- Logrouse o anterior utilizando ferramentas e bibliotecas de software libre, o cal permite que **o software teña unha licenza de software libre** que satisfai os requisitos da Free Software Foundation.

A medida que o traballo avanzaba, fóreronse intuíndo novas necesidades que os usuarios poderían ter. Destas, implementáronse as seguintes:

- Panel de administración: Provese dunha interface web de administración para que un superusuário do sistema poida manipular os datos presentes na base de datos.
- Visualización por méritos: Creouse o concepto de “popularidade” dos programas. Canto máis popular sexa un programa, máis posibilidades terá de saír en portada e aparecerá antes nos resultados de busca.
- Comentarios e votos: Os usuarios poden dar “feedback” aos autores dos programas mediante votos e comentarios nos episodios.

- Limitación de redifusión dos programas: Un dos fins deste proxecto é facilitar o intercambio de programas por parte de distintas emisoras, porén, poden darse situacións nas que se queira limitar esa posibilidade. As “Opcións de compartición” poden ser seleccionadas no panel de xestión do programa.

Persoalmente, considero que se desenvolveu unha ferramenta útil para os medios do terceiro sector. A aplicación web creada dá resposta a unha serie de necesidades reais que coñezo de primeira man, non so polos contactos con membros deste tipo de medios, senón tamén pola miña experiencia persoal colaborando en Cuac FM. O aumento das sinerxias entre colectivos é unha necesidade de supervivencia para estes e creo que este proxecto é un modesto aporte.

10.1. Coñecementos acadados

Durante os meus anos coma estudiante e coma profesional, o deseño web nunca foi unha predilección debido ao caóticas que me resultaban as ferramentas de desenvolvemento de **frontend**. Forzarme a utilizar JavaScript e afondar no meu coñecemento de CSS e HTML foi unha forma de tirar eses prexuízos e diversificar os meus coñecementos.

Aprender **Django** foi un aspecto moi positivo deste traballo, pois completa bastante a miña experiencia de uso de Python, linguaxe que me esperta especial interese.

O proxecto tamén me valeu para refrescar conceptos teóricos de **enxeñaría do software** (metodoloxías, xestión de proxectos, patróns de deseño...) e como aplicalos.

10.2. Futuros traballos

A continuación, exponse unha lista de melloras que se poderían levar a cabo no futuro:

- **Rediseño da interface de xestión de Programas e Emisoras:** Tras amosar a interface a xente allea ao proxecto, parece que a configuración actual das ferramentas de entrada de datos nesas vistas poden levar a certa confusión.
- **Enriquecer información de emisión:** O horario de emisión dun programa por unha emisora é información en texto. Poderíase modelar ese dato de xeito que o sistema puidese responder a preguntas coma: “Que programas se emiten os luns?”, “Que programa se está a emitir agora?”

- **Crear caixa de mensaxes para os usuarios:** Actualmente, os usuarios poden ver os novos episodios das súas subscricións na portada, pero estaría ben que se lles avisase cunha mensaxe directa. Poderíanse enviar mensaxes para máis eventos, por exemplo: “A emisora *RadioX* quere emitir o teu programa *ProgramaY*”
- **Melloras en seguridade:** O rexistro de usuario podería ter un CAPTCHA para evitar a un atacante crear usuarios de xeito automatizado. Tamén se podería pedir un número de teléfono aos usuarios para implementar unha “verificación de dous pasos”.

Apéndice A

Dicionario de datos

Neste apéndice figura, por orde alfabética, unha descripción dos modelos de datos utilizados. Inclúense so as entidades definidas de forma propia, non aquelas dadas polo framework (por exemplo, User). Do mesmo xeito, non se inclúen os atributos automaticamente definidos pola declaración relacóns en Django.

Broadcast: Clase que representa a relación N-N de emisión entre programas e emisoras.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
program	ForeignKey	Clave foránea da entidade Program
station	ForeignKey	Clave foránea da entidade Station
schedule_details	Char(100)	Texto do comentario

Comment: Entidade que garda os comentarios que os usuarios deixan nos episodios.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
episode	ForeignKey	Clave foránea da entidade Episode
user	ForeignKey	Clave foránea da entidade User
text	Text	Texto do comentario
publication_date	DateTime	Data de publicación (GMT)
removed	Boolean	Marcado coma borrado

Episode: Entidade que garda cada unha das entregas (episodios) dos programas.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
program	ForeignKey	Clave foránea da entidade Program
title	Char(200)	Título do episodio
summary	Text	Resumo do episodio
publication_date	DateTime	Data de publicación (GMT)
insertion_date	DateTime	Data de inserción na base de datos (GMT)
file	URL	Enlace ao ficheiro de audio
file_type	Char(40)	Enlace ao ficheiro de audio

downloads	BigInt	Contador de descargas e escoitas do episodio
original_id	Char(200)	Id do episodio no sistema orixinal de almacenamento
original_site	URL	Páxina do episodio no sistema orixinal de almacenamento
removed	Boolean	Marcado coma borrado
image	ForeignKey	Clave foránea da entidade Image
votes	ManyToMany	Referencia ás instancias de Vote relacionadas
comments	ManyToMany	Referencia ás instancias de Comment relacionadas

Program: Entidade que garda os programas engadidos polos usuarios.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
image	ForeignKey	Clave foránea da entidade Image
name	Char(200)	Nome do programa
description	Text	Texto descriptivo sobre o programa
creation_date	DateTime	Data de inserción na base de datos (GMT)
author_email	Email	Correo electrónico do autor do programa
author	Char(200)	Autor orixinal extraído do ficheiro RSS
language	Char(10)	Código de linguaxe do programa
rss_link	URL	Enlace ao ficheiro RSS do programa
rss_link_type	Char[ivoox radioco podomatic]	Tipo de RSSLinkParser utilizado na súa creación
rating	PositiveSmall Integer[0:100]	Cualificación calculada para o programa
original_site	URL	Enlace ao podcast orixinal
popularity	Float	Popularidade calculada para o programa
website	URL	Páxina web do programa
sharing_options	Char[share_free no_share]	Condicións de compartición
comment_options	Char[enable disable]	Opción de activar ou desactivar comentarios
subscribers	ManyToMany	Referencia ás instancias de User relacionadas. Representa os subscriptores do programa

admins	ManyToMany	Referencia ás instancias de ProgramAdmin relacionadas
---------------	------------	---

ProgramAdmin: Clase que representa a relación N-N de administración entre programas e usuarios.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
program	ForeignKey	Clave foránea da entidade Program
user	ForeignKey	Clave foránea da entidade User
type	Char[owner admin]	Permisos do usuario sobre o programa
date	DateTime	Data na que se concedeu o permiso (GMT)

Station: Entidade que garda os colectivos de emisión (radios por ondas, radios por internet, canles de podcast...)

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
logo	ForeignKey	Clave foránea da entidade Image para o logotipo da emisora
profile_img	ForeignKey	Clave foránea da entidade Image para a imaxe de cabeceira do perfil
name	Char(200)	Nome da emisora
broadcasting_method	Char[RadioFM RadioAM RadioDigital TV-Channel Radio-Internet PodcastingChannel Others]	Método de emisión dos programas
broadcasting_area	Char(200)	Área de emisión (No caso de RadioFM, RadioAM e RadioDigital)
broadcasting_frequency	Char(50)	Frecuencia de emisión (No caso de RadioFM, RadioAM e RadioDigital)
streaming_link	URL	Enlace á emisión en directo por streaming
website	URL	Enlace á páxina web do colectivo

location	Char(200)	Localización da emisora
programs	ManyToMany	Referencia ás instancias de Broadcast relacionadas
admins	ManyToMany	Referencia ás instancias de ProgramAdmin relacionadas
followers	ManyToMany	Referencia ás instancias de User relacionadas. Representa os seguidores da emisora.

StationAdmin: Clase que representa a relación N-N de administración entre emisoras e usuarios.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
station	ForeignKey	Clave foránea da entidade Station
user	ForeignKey	Clave foránea da entidade User
type	Char[owner admin]	Permisos do usuario sobre a emisora
date	DateTime	Data na que se concedeu o permiso (GMT)

Tag: Entidade que garda os etiquetas de categoría dadas polos ficheiros RSS.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
name	Char(50)	Nome do tag en minúsculas. Ten que ser único
times_used	Positive Integer-Field	Cantidad de veces presente en programas e episodios
programs	ManyToMany	Referencia ás instancias de Program relacionadas
episodes	ManyToMany	Referencia ás instancias de Episode relacionadas

UserProfile: Entidade que extende a clase User para asignarlle novos atributos.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
user	OneToOne	Referencia á instancia de User que extende
description	Text	Texto de presentación do usuario
avatar	ForeignKey	Clave foránea da entidade Image
location	Char(100)	Localización do usuario

Vote: Entidade que representa os votos cos que os usuarios cualifican os episodios.

Atributo	Tipo	Descripción
@id	Serial	Clave primaria
user	ForeignKey	Clave Foránea da entidade User
episode	ForeignKey	Clave foránea da entidade Episode
location	Char(100)	Localización do usuario
date	DateTime	Data na que se concedeu o permiso (GMT)

Apéndice B

Licenza

Este apéndice ten o propósito de determinar a licenza baixo a que se distribúen este proxecto e maila súa documentación. Un dos obxectivos a priori era que o software resultante fose compatible coa definición de software libre que da a Free Software Foundation, segundo a cal, un programa ten que cumplir o que comunmente se chama “As catro liberdades esenciais”^[48]:

- **Liberdade 0:** Liberdade de executar o programa con calquera propósito desexable.
- **Liberdade 1:** Liberdade para estudar o funcionamento do programa e de modificalo sen limitacións. Para garantir esta liberdade, o código fonte debe estar dispoñible ao acceso público.
- **Liberdade 2:** Liberdade de redistribución de copias do programa orixinal.
- **Liberdade 3:** Liberdade de distribución de copias de versións modificadas do software sempre e cando ese código modificado estea á súa vez dispoñible ao público.

B.1. Licenzas das dependencias do proxecto

A continuación móstrase unha táboa das dependencias do software deste proxecto. Centrarémonos en dúas características:

- **Liberdade:** Se a licenza é compatible coa definición de software libre vista con anterioridade.
- **Copyleft:** Dise daquela licencia que esixa preservar as súas liberdades na distribución do produto ou derivados. Un software con licenza libre non copyleft podería ser utilizado noutro software de natureza privativa.

Dependencia	Versión	Licenza	Libre	Copyleft
Anaconda	4.4.0 (64bits)	BSD (Berkeley Software Distribution)	Si	Non
Django	1.11	BSD	Si	Non
psycopg2	2.7.1	LGPL (GNU Lesser General Public License)	Si	Si
django-bootstrap4	0.0.6	Apache Software License 2.0	Si	Si
celery	4.1.0	BSD	Si	Non
django-celery-results	1.0.1	BSD	Si	Non
django-celery-beat	1.1.1	BSD	Si	Non
django-static-jquery	2.1.4	MIT (Massachusetts Institute of Technology)	Si	Non
gettext	0.19.8.1-3	GPLv3 (GNU General Public License)	Si	Si
select2	4.0.6	MIT	Si	Non

B.2. Conclusión

O software creado neste proxecto públicase baixo a licenza **GPLv3**. Esta foi orixinalmente creada para o proxecto GNU pola propia Free Software Foundation, sendo a versión 3, publicada en 2007, a máis recente. Trátase dunha licencia copyleft, o cal significa que a distribución das copias deste software e calquera traballo derivado han de ser tamén un proxecto de software libre. Esta licenza permite non so que, no futuro, outros desenvolvedores podan colaborar na mellora e mantemento deste software, senón tamén que outros proxectos o reutilicen.

A documentación, é dicir, a presente memoria, queda publicada baixo licenza **GFDL** (GNU Free Documentation License). É unha licenza libre creada tamén pola FSF para o proxecto GNU. Este feito implica que o presente documento pode ser copiado e modificado por quien o deseche. Ao ser GFDL unha licenza copyleft, a redistribución de copias e obras derivadas desta documentación está suxeita ao mantemento dos termos da licencia por parte das mesmas.

Bibliografía

- [1] W3Schools. Javascript html dom, Extraído o 22/05/2018. URL https://www.w3schools.com/js/js_htmldom.asp. v, 17
- [2] Wikipedia. JSON, Data types, syntax and example, Extraído o 05/05/2018. URL <https://en.wikipedia.org/wiki/JSONs>. v, 21
- [3] W3Counter. Browser & Platform Market Share May 2018, Extraído o 10/06/2018. URL <https://www.w3counter.com/globalstats.php?year=2018&month=5>. v, 29
- [4] España. Lei 7/2010, do 31 de marzo, Xeral da Comunicación Audiovisual. *Boletín Oficial Del Estado, xoves 1 de abril de 2010, suplemento en lingua galega ao núm. 79, sec I, pág. 30157.* URL https://www.boe.es/boe_gallego/dias/2010/04/01/pdfs/BOE-A-2010-5292-G.pdf. 2
- [5] Ben Hammersley. Audible revolution: Online radio is booming thanks to ipods, cheap audio software and weblogs. *The Guardian*, 12/04/2004. URL <https://www.theguardian.com/media/2004/feb/12/broadcasting.digitalmedia>. 2
- [6] Asociación para la investigación de medios de comunicación (AIMC). El streaming supera a la onda media en la escucha de radio. *3ª Ola EGM*, 30/11/2017. URL https://www.aimc.es/a1mc-c0nt3nt/uploads/2017/11/171130_NP_EGM_2017ola3.pdf. 3
- [7] Investigan a España por no dar licencias a medios comunitarios. *Federación de Sindicatos de Periodistas (FeSP)*, 14/03/2018. URL <http://www.fesp.org/index.php/noticias/item/8136-investigan-a-espana-por-no-dar-licencias-a-medios-comunitarios>. 3
- [8] Frances J. Berrigan. *La Comunicación Comunitaria: Cometido de los medios de comunicación comunitaria en el desarrollo*. Editorial de la Unesco, 1981. URL <http://unesdoc.unesco.org/images/0013/001343/134355so.pdf>. 3
- [9] Cyrus Farivar. iTunes 4.9 First Look: Apple takes on Podcasting, 28/06/2005. URL <https://www.macworld.com/article/1045522/podcastingfirstlook.html>. 7
- [10] Iago Veloso Abalo. Radioco, Extraído o 30/05/2018. URL <http://radioco.org/es>.
9
- [11] Mark Lutz. *Learning Python*. O'Reilly, 5 edition, 2013. ISBN 978-1-449-35573-9. 16
- [12] Anaconda Inc (Continuum Analytics). Anaconda distribution, Extraído o 22/05/2018. URL <https://www.anaconda.com/distribution>. 16

BIBLIOGRAFÍA

- [13] W3Schools. HTML Introduction, Extraído o 22/05/2018. URL https://www.w3schools.com/html/html_intro.asp. 16
- [14] Steve Faulkner, Aaron Eicholz, Travis Leithead, Alex Danilo, Sangwhan Moon. HTML 5.2. Technical report, W3C, 14/12/2017. URL <https://www.w3.org/TR/2017/REC-html52-20171214>. 16
- [15] W3Schools. What is CSS?, Extraído o 23/05/2018. URL https://www.w3schools.com/css/css_intro.asp. 17
- [16] Tab Atkins Jr., Elika J. Etemad, Rossen Atanassov. CSS Grid layout module level 1. Technical report, W3C, 14/12/2017. URL <https://www.w3.org/TR/2017/CR-css-grid-1-20171214>. 17
- [17] W3Schools. CSS Grid layout module, Extraído o 23/05/2018. URL https://www.w3schools.com/css/css_grid.asp. 18
- [18] VV. AA. ECMAScript® 2017 Language Specification (ECMA-262, 8th edition). Technical report, ECMA International, xuño 2017. URL <https://www.ecma-international.org/ecma-262/8.0>. 18
- [19] javascript.info. An Introduction to JavaScript, Extraído o 24/05/2018. URL <https:////javascript.info/intro>. 18
- [20] MDN Web docs. JavaScript, Extraído o 24/05/2018. URL <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. 18
- [21] Bear Bibeault, Yehuda Katz, Aurelio De Rosa. *jQuery in action*. Manning, 3 edition, 2015. ISBN 978-1-617-29207-1. 18
- [22] Python Software Foundation. django-static-jquery 2.1.4, 06/05/2015. URL <https:////pypi.org/project/django-static-jquery>. 19
- [23] Alan Beaulieu. *Learning SQL*. O'Reilly, 2005. ISBN 978-0-596-00727-0. 19
- [24] W3Schools. Introduction to SQL, Extraído o 25/05/2018. URL https://www.w3schools.com/sql/sql_intro.asp. 19
- [25] Elliotte Rusty Harold. *XML Bible*. IDG Books Worldwide, 1999. ISBN 0-7645-3236-7. 19
- [26] RSS Advisory Board. RSS 2.0 Specification, 30/03/2009. URL <http://www.rssboard.org/rss-specification>. 21

- [27] VV. AA. The JSON data interchange syntax (Standard ECMA-404, 2nd edition). Technical report, ECMA International, decembro 2017. URL <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. 21
- [28] The LaTeX project. An introduction to LaTeX, Extraído o 05/06/2018. URL <https://www.latex-project.org/about>. 22
- [29] James Bennett. *Practical Django projects*. Apress, 2009. ISBN 978-1-4302-1938-5. 22
- [30] Django Software Foundation. Django 2.0 release notes, 02/12/2017. URL <https://docs.djangoproject.com/en/2.0/releases/2.0>. 23
- [31] The Celery Project. Introduction to Celery, Extraído o 28/05/2018. URL <http://docs.celeryproject.org/en/latest/getting-started/introduction.html>. 24
- [32] Lovisa Johansson. *Getting started with RabbitMQ and CloudAMQP*. Codes AB, 2 edition, 2017. 24
- [33] Edmond Woichowsky. *Ajax, Creating web pages with asynchronous JavaScript and XML*. Prentice Hal, 2006. ISBN 0-13-227267-9. 25
- [34] The Apache Software Foundation. Licensing of Distributions, Extraído o 29/05/2018. URL <https://www.apache.org/licenses/>. 25
- [35] Netcraft. April 2018 Web Server Survey, 26/04/2018. URL <https://news.netcraft.com/archives/2018/04/26/april-2018-web-server-survey.html>. 25
- [36] The PostgreSQL Global Development Group. About PostgreSQL, Extraído o 25/05/2018. URL <https://www.postgresql.org/about>. 26
- [37] Jet Brains S.R.O. PyCharm Editions Comparison, Extraído o 27/05/2018. URL https://www.jetbrains.com/pycharm/features/editions_comparison_matrix.html. 27
- [38] Scott Chacon, Ben Straub. *Pro Git*. Apress, 2 edition, 2018. 27
- [39] The Fedora Project. Licensing FAQ, Extraído o 29/05/2018. URL <https://fedoraproject.org/wiki/Licensing:FAQ>. 28
- [40] Benito van der Zander. Welcome to TeXstudio, Extraído o 29/05/2018. URL <https://www.texstudio.org>. 28
- [41] Mozilla Foundation. Mozilla Foundation End-User Licensing Agreements, Extraído o 10/06/2018. URL <https://www.mozilla.org/en-US/about/legal/eula>. 29

BIBLIOGRAFÍA

- [42] The GIMP Team. About GIMP, Extraído o 11/06/2018. URL <https://www.gimp.org/about>. 29
- [43] ProjectLibre. ProjectLibre Blog, Extraído o 15/06/2018. URL <https://www.projectlibre.com/blog/license>. 30
- [44] Kent Beck. *Test-Driven Development By Example*. Addison Wesley, 2002. ISBN 978-0321146533. 33
- [45] Kent Beck, Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2 edition, 2012. ISBN 0-321-27865-8. 33
- [46] Ralph R. Young. *The Requirements Engineering Handbook*. Artech House, 2004. ISBN 1-58053-266-7. 54
- [47] Mozilla Development Network. Testing a Django web application, Extraído o 12/06/2018. URL <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing>. 91
- [48] The Free Software Foundation. ¿Qué es el software libre?, Extraído o 12/06/2018. URL <http://www.gnu.org/philosophy/free-sw.html>. 105