UNIVERSIDADE DA CORUÑA

Facultade de Informática
Departamento de Computación

Proxecto Fin de Carreira
Enxeñaría en Informática

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

Alumno: Fernando Liñares Varela
Director: José María Casanova Crespo
Data: 20 de xuño 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 gravacións de radio e podcasts"* foi realizada por Fernando Liñares Varela conforme á descrición inicialmente proposta baixo a miña direción 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 20 de xuño de 2018

D. José María Casanova Crespo
Director do proxecto

Título

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

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

Online service for publishing radio broadcasting recordings and podcasts

Clase: Proxecto clásico de enxeñaría

Autor: Fernando Liñares Varela
Director: José María Casanova Crespo
Data: 20 de xuño de 2018

Tribunal

Data de defensa:

Calificación:

Resumo

Este proxecto consiste no desenvolvemento dunha aplicación web que permite a escoita de gravacións de radio ou podcast.

O sistema conta con: emisoras, cuxa emisión en directo se pode escoitar mediante streaming, e programas, cuxos episodios quedan dispoñibles ou ben para escoitar por streaming baixo demanda, ou ben para a súa descarga directa. Estes contidos son engadidos e administrados polos propios usuarios, que poderán colaborar nas labores de xestión dos mesmos.

A web funciona tamén a modo de catálogo de programas para as emisoras. Unha emisora do sistema pode incorporar os programas existentes á súa emisión, sendo posible a difusión dun programa por parte de varios colectivos.

A motivación para este traballo foi a de proporcionar un punto de encontro en liña para as radios libres e comunitarias co obxectivo aumentar a visibilidade dos seus programas en Internet e favorecer a compartición de contidos e a colaboración entre elas.

O resultado do proxecto é un produto de software libre. Para a súa creación utilizouse Django, un framework de desenvolvemento web de Python e empregáronse técnicas propias das metodoloxías áxiles. Esta memoria explica o proceso completo desde a súa concepción, pasando polas fases de análise, deseño, planificación e dando detalles da súa implementación e validación.

Lista de Palabras Clave

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

Aos meus compañeiros de prácticas.

Agradecementos

Teño moito que agradecer ao profesor José María Casanova Crespo polos seus consellos, pola súa axuda e pola súa extraordinaria implicación, contestándome ao Hangouts incluso a altas horas da madrugada. Ao meu xefe e compañeiros de Gompute S.L.U. por apoiarme na miña decisión de retomar os estudos. A Cuac FM, por facer medrar en min o amor pola radio que inspirou este proxecto. Ao nefasto clima de Lugo, por afastarme de plans que puideran levarme a procrastinación. A Stack Overflow, o máis fiel dos compañeiros de prácticas.

Moitas grazas á miña parella e aos meus pais polo seu apoio constante e incondicional, aquí e en calquera eido da vida.

A todos eles, de corazón, moitas grazas.

Índice xeral

1. Introdución **1**

1.1. Marco do proxecto 2

1.2. Motivación 3

1.3. Orixe do proxecto 3

1.4. Obxectivos 4

1.5. Organización da memoria 5

2. Estado da arte **7**

2.1. Alternativas Existentes 8

2.1.1. audio.urcm.net 8

2.1.2. TuneIn 8

2.1.3. iTunes 9

2.1.4. Podomatic 9

2.1.5. Ivoox 10

2.1.6. RadioCo 11

2.2. Análise comparativa 12

2.3. Conclusión 13

3. Tecnoloxía e ferramentas empregadas **15**

3.1. Linguaxes 17

3.1.1. Python 17

3.1.2. HTML 18

3.1.3. CSS 19

3.1.4. JavaScript 20

3.1.5. SQL 21

3.1.6. XML 21

3.1.7. JSON 23

3.1.8. LaTeX 24
3.2. Django Framework 24
3.3. Celery 26
3.4. RabbitMQ 26
3.5. Bootstrap 26
3.6. Ajax 27
3.7. Apache HTTP server 27
3.8. PostgreSQL 28
3.9. Ferramentas de desenvolvemento 29
3.9.1. Eclipse 29
3.9.2. Git 29
3.9.3. Dia 30
3.9.4. Fedora 30
3.9.5. TeXstudio 30
3.9.6. Mozilla Firefox 31
3.9.7. GIMP 31
3.9.8. ProjectLibre 32

4. Metodoloxía **33**
4.1. Principios das metodoloxías áxiles 34
4.2. Extreme Programming 35
4.3. Test Driven Development 36

5. Análise **39**
5.1. Descrición do funcionamento 40
5.1.1. Actividades de consumo de contidos 40
5.1.2. Actividades de produción de contidos 40
5.2. Requirimentos funcionais 41
5.3. Requirimentos non funcionais 42

5.3.1. Autenticación 42
5.3.2. Protección de datos 42
5.3.3. Internacionalización 43
5.3.4. Rendemento 43
5.3.5. Concorrencia 43
5.3.6. Seguridade 43
5.3.7. Adaptabilidade a dispositivos móbiles 44
5.3.8. Usabilidade 44

6. Deseño **45**
6.1. Arquitectura 46
6.1.1. Capa Modelo 46
6.1.2. Deseño de interacción 51
6.1.3. Capa Vista 51
6.1.4. Capa Template 55
6.2. Actualización dos datos 60

7. Implementación **63**
7.1. Estrutura xeral do código fonte 64
7.2. Ficheiros de imaxe 65
7.3. A vista de engadir programa 66
7.4. A Internacionalización 67
7.5. O contador de escoitas 68
7.6. O panel de xestión 69
7.7. Execución dos procesos en Celery 70
7.8. Adaptabilidade a dispositivos móbiles 70

8. Planificación **73**
8.1. Concepción do proxecto 74

8.2. Iteracións 74
8.2.1. Primeira reunión (Iteración 0) 74
8.2.2. Iteración 1 75
8.2.3. Iteración 2 76
8.2.4. Iteración 3 77
8.2.5. Iteración 4 78
8.2.6. Iteración 5 78
8.2.7. Iteración 6 79
8.2.8. Iteración 7 80
8.2.9. Iteración 8 81
8.2.10. Iteración 9 82
8.2.11. Iteración 10 83
8.2.12. Iteración 11 84
8.3. Control da execución 85
8.3.1. Diagrama de Gantt 86
8.4. Avaliación de custos 98

9. Probas do sistema **101**
9.1. Plan de probas 102
9.2. Probas de unidade 102
9.3. Probas de integración 107
9.4. Probas de aceptación 110

10. Conclusións **111**
10.1. Coñecementos acadados 113
10.2. Traballo Futuro 113

A. Apéndice: Dicionario de datos **115**

B. Apéndice: Licenza **121**

B.1. Licenzas das dependencias do proxecto 121
B.2. Conclusión 122

C. Apéndice: Manual de usuario 123

C.1. Usuarios 124
C.1.1. Crear 124
C.1.2. Editar 124
C.2. Accións de ouvinte 127
C.2.1. Escoitar e seguir unha emisora 127
C.2.2. Subscribirse a un programa 128
C.2.3. Accións sobre os episodios 128
C.2.4. Contidos favoritos 132
C.3. Procura de contidos 133
C.4. Accións de produtor de contidos 134
C.4.1. Crear unha emisora 134
C.4.2. Xestionar unha emisora 134
C.4.3. Crear un programa 138
C.4.4. Xestionar un programa 140

Índice de figuras

1.1. Ouvintes de radio promedio diario do ano 2017 en España. 3

1.2. Sección de audios da web da URCM que inspira o proxecto. 4

2.1. Interface de TuneIn 8

2.2. Interface de Podomatic 10

2.3. Interface de Ivoox 10

2.4. Interface de RadioCo 11

3.1. Diagrama de interacción de tecnoloxías. 17

3.2. Exemplo de árbore HTML DOM[1] 19

3.3. Fragmento de ficheiro RSS real utilizado nas probas. 22

3.4. Exemplo de sintaxe JSON[2] 23

3.5. Cota de mercado dos servidores no Top million busiest sites (Netcraft, abril 2018) 28

3.6. Cota de mercado dos navegadores web. (W3Counter, maio 2018)[3] 31

4.1. Diagrama do ciclo de desenvolvemento coa metodoloxía TDD. 37

6.1. Árbore de módulos da aplicación web. 46

6.2. Diagrama Entidade-Relación da Base de Datos. 48

6.3. Diagrama de clases da capa modelo. 49

6.4. Mapa de navegación entre as distintas páxinas. 51

6.5. Esquema do funcionamento das vistas. 53

6.6. Patrón Estratexia utilizado para o procesamento de RSS. 54

6.7. Esquema dun elemento da cuadrícula. 55

6.8. Extracto do borrador do deseño da interface. Portada. 56

6.9. Deseño final da portada. 57

6.10. Deseño da portada para dispositivos móbiles. Á esquerda, a parte superior, á dereita, a parte inferior. 57

6.11. Extracto do borrador do deseño da interface. Detalles de programa. 59
6.12. Extracto do borrador do deseño da interface. Xestión de emisión. 60

7.1. Estrutura do código fonte do proxecto. 64
7.2. Formulario de engadir programa. 67
7.3. Fragmento do ficheiro .po para a tradución ao Galego. 68
7.4. Panel de xestión dunha emisora. Vista de xestión da emisión. 69
7.5. Páxina de administración da aplicación, sección de procesos de Celery. . . . 70

8.1. It1: Diagrama de clases do primeiro borrador de deseño. 76
8.2. It2: Interface web. Páxina dun episodio. 77
8.3. It3: Interface web. Páxina de engadir programa. 78
8.4. It4: Resultado de consulta á base de datos relacionando programas cos seus tags. 79
8.5. It6: index.html para usuario identificado. 80
8.6. It6: index.html para usuario anónimo. 80
8.7. It8: Vista de detalles de emisora. 81
8.8. It9: Vista de detalles de episodio. 82
8.9. It10: Panel de xestión de programa. 83
8.10. It11: Páxina de resultados de busca. 84
8.11. Diagrama de Gantt. Novembro-Decembro. 87
8.12. Diagrama de Gantt. Decembro. 88
8.13. Diagrama de Gantt. Decembro-Xaneiro 89
8.14. Diagrama de Gantt. Xaneiro-Febreiro 90
8.15. Diagrama de Gantt. Febreiro 91
8.16. Diagrama de Gantt. Febreiro-Marzo 92
8.17. Diagrama de Gantt. Marzo-Abril 93
8.18. Diagrama de Gantt. Abril 94
8.19. Diagrama de Gantt. Abril-Maio 95

8.20. Diagrama de Gantt. Maio 96
8.21. Diagrama de Gantt. Maio-Xuño 97

C.1. Portada da web para usuario anónimo 123
C.2. Rexistro de usuario. 125
C.3. Detalles de usuario. Información do perfil. 126
C.4. Editar usuario. 127
C.5. Detalles de emisora. 129
C.6. Detalles de programa. 130
C.7. Detalles de episodio. 131
C.8. Contidos favoritos na portada. 132
C.9. Contidos favoritos na vista de perfil de usuario. 132
C.10. Páxina de resultados de procura. Exemplo de busca pola palabra radio. . . 133
C.11. Engadir emisora. 135
C.12. Vista de detalles de emisora: Nova emisora. 136
C.13. Panel de xestión de emisora: Editar perfil. 136
C.14. Panel de xestión de emisora: Xestionar emisión. 137
C.15. Panel de xestión de emisora: Xestionar administradores. 138
C.16. Panel de xestión de emisora: Borrar emisora. 138
C.17. Engadir programa. 139
C.18. Vista de detalles de program: Novo programa. 139
C.19. Panel de xestión de programa: Editar perfil. 140
C.20. Panel de xestión de programa: Xestionar emisión. 141
C.21. Panel de xestión de programa: Xestionar administradores. 141
C.22. Panel de xestión de programa: Borrar programa. 142

Capítulo 1

Introdución

1.1. Marco do proxecto 2
1.2. Motivación 3
1.3. Orixe do proxecto 3
1.4. Obxectivos 4
1.5. Organización da memoria 5

Nos últimos anos, Internet converteuse nunha peza importante 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 ás que comunmente se fai referencia coma **medios do terceiro sector**.

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, 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, neste país, a *Secretaría de Estado de Telecomunicaciones y para la Sociedad de la Información* e máis a *Secretaría Xeral de Medios* da Xunta de Galicia[4]. Isto implica a imposibilidade de emitir contidos por parte de aqueles que ou ben non poidan facer 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 unha cobertura global. Este tipo de emisión, ademais, non require de cadeas de radio enlace, como é costume nas grandes emisoras en FM, coa inversión en infraestruturas que tal cousa implica.

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; aínda 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

Por consecuencia da falta de regulamentación específica e a desgana das institucións públicas, os medios do terceiro sector atopan a miúdo serias dificultades para realizar a súa actividade a través da FM. Para comprender a gravidade desta situación, cómpre recordar que se está a falar de 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 está 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 pois os asuntos que unha radio comunitaria ten capacidade de tratar rara vez son internacionais .

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

1.3. Orixe do proxecto

Produciuse un encontro informal con membros de **Cuac FM** (a radio comunitaria da Coruña) e a **URCM**(Unión de Radios Comunitarias de Madrid) onde se entrou en contacto cos usuarios finais do proxecto a realizar.

A URCM está composta por unha serie de emisoras independentes e con programas de seu. Conta cunha páxina web onde os usuarios teñen acceso a un **directorio cos ficheiros de audio** publicados por ditas emisoras (ver figura 1.2)[9]. 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.

Figura 1.2: Sección de audios da web da URCM que inspira o proxecto.

Inspirados nesta idea, propúxose crear unha ferramenta semellante, pero cun conxunto de funcionalidades máis ambicioso. Esta vez para uso potencial da **ReMC** (Red de Medios Comunitarios), unha federación de medios comunitarios do Estado Español.

1.4. Obxectivos

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

O produto 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 compartilos desde o seu propio almacenamento.

Os obxectivos a acadar son os seguintes:

- 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.

1.5. Organización da memoria

Esta memoria consta de dez capítulos, incluído este mesmo. A continuación explícase brevemente cada un deles:

- **Capítulo 1, Introdución:** Explicación das motivacións e a orixe do proxecto. Defínense aquí, tamén os obxectivos a cumprir.
- **Capítulo 2, Estado da arte:** Comparación do proxecto presentado con outros produtos software existentes na actualidade. Arguméntase por que o primeiro cumpre de xeito máis amplo e preciso cos obxectivos marcados.
- **Capítulo 3, Tecnoloxía:** Mostra das ferramentas, bibliotecas, linguaxes e "frameworks" utilizados no desenvolvemento do proxecto e máis na confección desta memoria.
- **Capítulo 4, Metodoloxía:** Comentario das diferentes técnicas e metodoloxías para o desenvolvemento de software que se seguiron neste proxecto.
- **Capítulo 5, Análise:** Exposición do funcionamento ideal xeral do sistema e os requirimentos funcionais e non-funcionais que este debe cumprir.
- **Capítulo 6, Deseño:** Percorrido pola arquitectura do sistema explicando a interacción entre as súas compoñentes e os patróns de deseño utilizados.
- **Capítulo 7, Implementación:** Mostra das solucións de implementación aplicadas a certas partes do sistema especialmente complexas ou que non fosen tratadas nos capítulos anteriores.

- **Capítulo 8, Planificación e avaliación de custos:** Diario de traballo que explica a planificación de historias de usuario e os avances do desenvolvemento a través do tempo. Inclúe a avaliación de custes.
- **Capítulo 9, Probas do sistema:** Revisión do traballo de validación realizado sobre as distintas partes do sistema.
- **Capítulo 10, Conclusións:**Valoración persoal do traballo, comparando o resultado final cos obxectivos previos. Inclúe unha breve reflexión sobre os coñecementos acadados e unha guía de traballos que se poderían realizar no futuro co fin de mellorar o produto desenvolvido.

Tamén se inclúen, a maiores, nesta memoria, tres apéndices:

- **Apéndice A, Dicionario de datos:** Lista das entidades do modelo de datos detallando dos seus atributos.
- **Apéndice B, Licenza:** Relación das licenzas das dependencias do proxecto e determinación das licenzas baixo as que se distribúe a aplicación creada e esta documentación.
- **Apéndice C, Manual de usuario:** Instrucións de uso da aplicación web.

Capítulo 2

Estado da arte

2.1. Alternativas Existentes **8**
2.1.1. audio.urcm.net 8
2.1.2. TuneIn 8
2.1.3. iTunes 9
2.1.4. Podomatic 9
2.1.5. Ivoox 10
2.1.6. RadioCo 11
2.2. Análise comparativa **12**
2.3. Conclusión **13**

Existe certa variedade produtos software onde se poden poñer arquivos de audio a disposición do público, algúns que tamén permiten o acceso á emisión en directo por streaming e a subscrición ás diferentes fontes de contido. Neste capítulo analizaranse as seguintes solucións co fin de comparalas cos nosos obxectivos: audio.urcm.net, TuneIn, iTunes, Podomatic, Ivoox e RadioCo.

2.1. Alternativas Existentes

2.1.1. audio.urcm.net

Portal que serviu de inspiración para o proxecto (ver sección 1.3) pola súa función de páxina centralizada de contidos para distintas emisoras que forman unha federación. Esta ferramenta non posibilita a creación de perfís de usuario para os ouvintes e o seu concepto de emisora non é máis que un atributo de programa. Non permite a escoita da emisión en directo e o reprodutor de episodios é unha ferramenta Flash un tanto obsoleta.

A superación destas limitacións son parte da motivación deste traballo.

2.1.2. 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.

2.1.3. iTunes

iTunes é un software de reprodución 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[10], e continúa, a día de hoxe, a ser unha das ferramentas máis populares para a escoita e xestión de subscrición de este formato radiofónico.

iTunes tamén ten unha funcionalidade para escoitar radio por Internet, porén esta capacidade é limitada pois o catálogo de emisoras dispoñibles redúcese a aquelas posuídas por Apple. Un usuario si podería escoitar 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 máis aló do que cada produtor poida ter escrito na descrición do seu podcast.

2.1.4. 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 comezar a 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 gratuíta están limitados a unha certa visibilidade e a un máximo de episodios publicados.

Figura 2.2: Interface de Podomatic

2.1.5. 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 contido e ouvintes semellantes aos de Podomatic (sección 2.1.4) 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.

2.1.6. 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[11], pero o seu ámbito é de xestión dos contidos dunha emisora e non de ferramenta transversal entre distintos medios.

2.2. Análise 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. Non se incluiu audio.urcm.net (ver seccións 1.3 e 2.1.1) nela por considerarse redundante volver a incidir nas súas limitacións.

- **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.
- **Subscrición a programas:** Opción de manter unha lista de programas escoitados polo usuario e recibir información das novas publicacións destes.
- **Hosting de seu:** Posibilidade de aloxar os ficheiros de audio nun almacenamento remoto propio do aplicación.
- **Escoita en directo:** Opción de escoitar 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 escoitar 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.
- **Software Libre:** Se a aplicación en cuestión ten unha licenza de software libre avalada pola "Free Software Foundation"[12].

Obxectivos	TuneIn	iTunes	Podomatic	Ivoox	RadioCo	Proxecto
Escoita de episodios	Si	Si	Si	Si	Si	Si
Descarga directa	Non	Si	Limit.	Si	Si	Si
Subscrición 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
Licenza de software libre	Non	Non	Non	Non	Si	Si

Táboa 2.1: Comparativa de alternativas fronte a requirimentos.

2.3. Conclusión

Existen, na actualidade, produtos que cobren algunhas das necesidades dos medios radiofónicos comunitarios de cara a Internet e, de feito, para calquera dos exemplos propostos na sección anterior, poderiamos 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 algunhas 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 **17**
3.1.1. Python 17
3.1.1.1. Anaconda 18
3.1.2. HTML 18
3.1.3. CSS 19
3.1.3.1. CSS Grid 19
3.1.4. JavaScript 20
3.1.4.1. jQuery 20
3.1.5. SQL 21
3.1.6. XML 21
3.1.6.1. RSS 22
3.1.7. JSON 23
3.1.8. LaTeX 24
3.2. Django Framework **24**
3.3. Celery **26**
3.4. RabbitMQ **26**
3.5. Bootstrap **26**
3.6. Ajax **27**
3.7. Apache HTTP server **27**
3.8. PostgreSQL **28**
3.9. Ferramentas de desenvolvemento **29**
3.9.1. Eclipse 29
3.9.2. Git 29
3.9.2.1. GitHub 30
3.9.3. Dia 30
3.9.4. Fedora 30
3.9.5. TeXstudio 30
3.9.6. Mozilla Firefox 31

3.9.7. GIMP 31
3.9.8. ProjectLibre 32

Neste capítulo, preséntanse as linguaxes, ferramentas e frameworks utilizados no desenvolvemento deste proxecto. Á hora de elixir o uso destas, valorouse a súa natureza libre e de código aberto xa que se pretende que o resultado dispoña dunha licenza de software libre compatible coa definición da Free Software Foundation[12] (ver apéndice B).

Tamén se valoraron, dado que nos capítulos anteriores se insistiu na importancia dos dispositivos móbiles 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 este 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"[13], 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 [14]

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 decláranse mediante bloques representados por etiquetas(tags), de aí que reciba o nome de "markup language" (linguaxe de marcado). Estas etiquetas son interpretadas polos navegadores para renderizar os contidos[15].

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)[16].

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**[17].

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, declárase 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)[18], porén, xa é sopor-

tado polas últimas versións dos navegadores máis populares[19] 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 linguxe Java e serven a propósitos claramente distintos. O núcleo desta linguaxe está regulado polo estándar ECMAScript®, na súa 8ª versión[20] 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[21], 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)[22]

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[23], 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 máis actual incluída na biblioteca django-static-jquery-2.1.4[24] 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)[25]. Permite definir a estruturación dos datos, inserilos, eliminalos, 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[26]. 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[27].

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 subscritores. 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 subscritores. 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 [28]

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 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[29] 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[30]). 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[31].

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 superusuarios 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[32], considerouse aceptable o grao de actualidade da tecnoloxía utilizada.

Ademais do mencionado anteriormente, confiouse 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[33].

Neste proxecto, utilizouse Celery para controlar a **execución dos daemons** 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 proxectos 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 co fin de enviar a ou lelos deste, funcionando o dito broker coma unha cola. As mensaxes enviadas á cola por unha aplicación permanecerán aí até que sexan lidos por outra aplicación. Unha mensaxe pode ser calquera cousa, desde meta información dos procesos até texto plano[34].

Neste proxecto utilizouse a versión 3.6.10-1 por seres esta a máis 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

"templates", cada unha cunha estrutura HTML, declaracións en CSS e, nalgúns casos, extensións 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[35], 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 diminuír o tráfico entre os dous lados, facendo a navegación máis áxil 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[36]). 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 decaeu 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[37]. 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 máis actual no momento de comezar o traballo. Foi necesaria tamén a instalación das seguintes extensións:

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 aplicacións web baseadas 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)[38]. A súa orixe remóntase a 1996, polo que existe unha grande dispoñibilidade de recursos de documentación. Ese feito unido á certa experiencia persoal no seu uso xa antes de comezar 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 comezar 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, unha aplicación 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 aínda 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[39].

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[40].

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 gratuíta 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 unicamente coma arquivos binarios. Nese último caso, han de cumprimentar unha serie de requirimentos coma estar libres de pago polo seu dereito de uso[41]. 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[42]). 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[43]). Soporta unha grande variedade de estándares e está dispoñible en gran variedade de sistemas operativos, incluíndo os dos dispositivos móbiles máis populares: iOS e Android. Aínda que a súa popularidade foi en retroceso nos últimos anos, segue a ser un dos navegadores máis populares e o máis utilizado entre as alternativas de software libre (ver figura 3.6).

Valorouse, á hora de decidir utilizalo, a súa importancia no mundo do software libre, a súa dispoñibilidade 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 dispoñible 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[44]). Neste proxecto utilizouse para facer postprocesados sinxelos das

imaxes incluídas na memoria. A versión utilizada foi a 2.8.22 por ser a máis actual nos repositorios do Sistema Operativo no momento de comezar a escritura da memoria.

3.9.8. ProjectLibre

ProjectLibre é unha **ferramenta de xestión e dirección de proxectos**, libre e de código aberto (licenza CPAL[45]). 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. Principios das metodoloxías áxiles **34**
4.2. Extreme Programming **35**
4.2.0.1. Aporte da metodoloxía ao proxecto 36
4.3. Test Driven Development **36**
4.3.0.1. Aporte da técnica ao proxecto 38

A metodoloxía de traballo baseouse nos principios das **metodoloxías áxiles** aplicando, concretamente, aquelas prácticas e técnicas de **"Extreme Programming"** (XP) adaptables ao traballo individual. Destacarase, pola súa especificidade, o uso da técnica de "Test Driven Development" (TDD) nalgunhas partes do desenvolvemento.

4.1. Principios das metodoloxías áxiles

Considéranse áxiles aquelas metodoloxías motivadas por unha serie de valores e que respectan unha serie principios, ambos recollidos no "manifesto áxil" do ano 2001 [46]. Eses valores son:

- **Os individuos e súa interacción** valórase máis que as ferramentas e os procesos.
- **O correcto funcionamento do software** valórase máis que a documentación extensiva.
- **A colaboración co cliente** valórase máis que a negociación contractual.
- **A resposta ante o cambio** valórase máis que o seguimento dun plan.

Que se traducen nos chamados **"Doce principios do Software Áxil"**:

1. A nosa maior prioridade é satisfacer ao cliente mediante a entrega temperá e continua de software con valor.
2. Aceptamos que os requirimentos cambien, incluso en etapas tardías do desenvolvemento. Os procesos Áxiles aproveitan o cambio para proporcionar vantaxe competitiva ao cliente.
3. Entregamos software funcional frecuentemente, entre dúas semanas e dous meses, con preferencia ao período de tempo máis corto posible.
4. Os responsables de negocio e os desenvolvedores traballamos xuntos de forma cotiá durante todo o proxecto.
5. Os proxectos desenvólvense en torno a individuos motivados. Hai que darlles o entorno e o apoio que necesitan, e confiarlles a execución do traballo.
6. O método máis eficiente e efectivo de comunicar información ao equipo de desenvolvemento e entre os seus membros é a conversación cara a cara.
7. O software funcionando é a medida principal de progreso.

8. Os procesos Áxiles promoven o desenvolvemento sostible. Los promotores, desenvolvedores e usuarios debemos ser capaces de manter un ritmo constante de forma indefinida.
9. A atención continua á excelencia técnica e ao bo deseño mellora a Axilidade.
10. A simplicidade, a arte de maximizar a cantidade de traballo non realizado, é esencial.
11. As mellores arquitecturas, requirimentos e deseños emerxen de equipos auto-organizados.
12. A intervalos regulares o equipo reflexiona sobre como ser máis efectivo para a continuación axustar e perfeccionar o seu comportamento en consecuencia.

4.2. Extreme Programming

Extreme Programming (XP), 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 máis tradicionais coma, por exemplo, o "desenvolvemento en fervenza".

Eses valores enuméranse coma[47]:

- **Comunicación:** Tanto entre programadores coma co cliente.
- **Simplicidade:** Que o código sexa lexible e a documentación concisa.
- **Retroalimentación:** Realizar iteracións curtas para obter opinións rápidas sobre os progresos.
- **Coraxe:** Aceptar que a aparición de novos requirimentos 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.

A esas "ideas novas" que aparecen en cada iteración chamámoslles **historias de usuario**. Unha historia de usuario consiste nunha frase concisa onde o usuario obxectivo expresa unha necesidade que ha de ser cuberta polo sistema.

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 cualidade (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...)

4.2.0.1. Aporte da metodoloxía ao proxecto

Ao ser este un Proxecto de Final de Carreira, as reunións co cliente substituíronse por **reunións co director do proxecto de periodicidade quincenal**. Nelas, revisábanse os progreso e xurdían novas historias de usuario (ou se actualizaban as vellas). Grazas a este modo de traballo conseguíronse identificar varios **requirimentos non valorados a priori** coma, por exemplo, a necesidade de establecer roles de usuario.

Para afondar individualmente en cada iteración, ver o capítulo 8 sobre a planificación.

4.3. Test Driven Development

Test Driven Development (TDD) é unha técnica utilizada nas metodoloxías áxiles de desenvolvemento que segue a filosofía **probar primeiro**. Esta baséase na conversión dos requirimentos en probas antes da propia existencia de código que as avale. Isto obriga ao desenvolvedor a pensar nos requirimentos con detemento, identificar de antemán os posibles fallos e aumentar o alcance da validación de datos. Unha vez o test está definido, pasarase a implementación da funcionalidade posta a proba. De pasar a proba, repítese co seguinte requirimento. Este ciclo pode verse no esquema da figura 4.1.

Existen distintas estratexias á hora de seguir o citado ciclo [48] que se poden utilizar en función das características do requirimento:

- **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

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

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 requirimento 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 máis intuitiva para o traballo a realizar.

4.3.0.1. Aporte da técnica ao proxecto

Considerouse a utilización de TDD durante a **implementación do modelo de datos** (ver primeiras iteracións no capítulo 8), ao predicirse que sería un código moi sensible aos cambios podería dar pé a regresións. Tamén axudou ao análise de requirimentos ao forzarnos a pensar nos resultados posibles de cada método de antemán.

O uso de TDD forzou a temperá implementación de probas automatizadas, que se poden ver detalladas no capítulo 9 sobre as probas do sistema.

Capítulo 5

Análise

5.1. Descrición do funcionamento 40
5.1.1. Actividades de consumo de contidos 40
5.1.2. Actividades de produción de contidos 40
5.2. Requirimentos funcionais 41
5.3. Requirimentos non funcionais 42
5.3.1. Autenticación 42
5.3.2. Protección de datos 42
5.3.3. Internacionalización 43
5.3.4. Rendemento 43
5.3.5. Concorrencia 43
5.3.6. Seguridade 43
5.3.7. Adaptabilidade a dispositivos móbiles 44
5.3.8. Usabilidade 44

Neste capítulo descríbese o funcionamento ideal da aplicación. Exporanse tamén os requirimentos funcionais e non funcionais identificados ao abordar os obxectivos do proxecto.

5.1. Descrición 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.

5.1.1. Actividades de consumo de contidos

Son aquelas **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 a través das 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 á 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 HTML5 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.

5.1.2. Actividades de produción de contidos

Son aquelas levadas a cabo polos **propietarios 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 encar-

garase 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.

5.2. Requirimentos 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. Presentaranse estes en forma de **historias de usuario** (ver capítulo 4 sobre a metodoloxía).

Se ben é certo que houbo **encontros con membros de colectivos do terceiro sector** coma a URCM (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 6 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, fóronse 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 requirimentos 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.

5.3. Requirimentos non funcionais

Son aqueles que **especifican as propiedades do sistema** no referente á manexabilidade, seguridade e robustez[49].

5.3.1. 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.

5.3.2. Protección de datos

Os usuarios deben saber en todo momento que datos persoais deles son recompilados polo sistema e para que fin serán utilizados. Ademáis, débese garantir o seguinte:

- **Dereito ao esquecemento:** Os datos dun usuario han de ser borrados do sistema cando este así o requira.

- **Disponibilidade:** O usuario ha de poder acceder aos seus datos sen posibilidade de ocultación ou demora por parte do sistema.

5.3.3. 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.

5.3.4. 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.

5.3.5. Concorrencia

O servidor ten que ser capaz de servir á páxina a un número de usuarios simultáneos aceptable. Isto depende en grande medida do hardware no que o servidor sexa executado. En calquera caso, sendo as emisoras comunitarias colectivos pequenos e baseándonos en datos de visitas á web de Cuac FM, non se espera que o cumprimento deste requisito sexa un risco debido ao baixo volume de usuarios e as tecnoloxías seleccionadas.

5.3.6. 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.

5.3.7. 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.

5.3.8. Usabilidade

O sistema debe poder ser utilizado por usuarios cun perfil non técnico. As ferramentas de entrada e saída de datos han de ser o suficientemente comprensibles e coñecidas para o público xeral.

Capítulo 6

Deseño

6.1. Arquitectura 46

6.1.1. Capa Modelo 46

6.1.1.1. Usuarios 47

6.1.1.2. Programas e episodios 48

6.1.1.3. Votos e comentarios 49

6.1.1.4. A entidade Station 50

6.1.1.5. Administración de contido 50

6.1.2. Deseño de interacción 51

6.1.3. Capa Vista 51

6.1.3.1. Engadir contido 54

6.1.4. Capa Template 55

6.1.4.1. Portada 55

6.1.4.2. Páxinas de engadir e editar contido 58

6.1.4.3. Páxinas de detalle 58

6.1.4.4. Páxina de resultados de busca 58

6.1.4.5. Páxinas de xestión 59

6.2. Actualización dos datos 60

6.2.0.1. Novos episodios 60

6.2.0.2. Popularidade e cualificación dos programas 60

6.1. 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 co 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 6.1: Árbore de módulos da aplicación web.

6.1.1. 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 6.2, 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 6.2 pódense ver as táboas empregadas sen incluír aquelas automaticamente 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 6.3. Por claridade, nese diagrama de clases só se incluíron os atributos máis representativos ou explicativos das relacións entre clases.

Nota: As entidades e os seus atributos amósanse en detalle no dicionario de datos do anexo A.

6.1.1.1. 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, incluída 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 6.2). Esta entidade auxiliar contén os atributos de usuario: avatar, descrición 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. 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 da aplicación que provee o propio framework (como se mencionou no apartado 3.2)

Outros atributos de User reflectidos na figura 6.3 coma followers ou station_admins son engadidos automáticamente polo framework ao declarar as relacións co fin de facilitar a navegación entre clases.

Figura 6.2: Diagrama Entidade-Relación da Base de Datos.

6.1.1.2. Programas e episodios

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

A **subscrición** dos usuarios aos programas modelouse coma unha relación N-N. As instancias de Program poden acceder ao conxunto dos seus subscritores 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

Figura 6.3: Diagrama de clases da capa modelo.

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**.

6.1.1.3. 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 6.2 e 6.3).

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

6.1.1.4. A entidade Station

Identifícanse 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 emsión: 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**.

6.1.1.5. 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 6.1.3.1), non son editables. Aquelas opcións que poidan afectar en bloque aos episodios dun programa considéranse xa opcións do programa.

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

6.1.2. Deseño de interacción

A figura 6.4 mostra o mapa de navegación entre vistas. Isto serviu como guía do deseño das capas vista e template que se analizan a continuación.

Figura 6.4: Mapa de navegación entre as distintas páxinas.

6.1.3. Capa Vista

A capa vista é na que se atopa a **lóxica funcional da aplicación**. Fai uso dos obxectos da capa modelo para, ou ben extraer e compoñer 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 respostas. 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 relacione as urls coas funcións e clases correspondentes á operación a realizar. Comunmente en Django e tamén neste proxecto, este ficheiro chámase **urls.py**.

Na figura 6.5 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 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**.

Figura 6.5: Esquema do funcionamento das vistas.

6.1.3.1. Engadir contido

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 á aplicación o enlace ao seu **ficheiro de RSS**.

Figura 6.6: 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 deste 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: Distintos servidores de podcasting poden ter **distintos formatos de ficheiro RSS**.

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 6.6. 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.

6.1.4. 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 desa cuadrícula representa un programa, episodio ou emisora da forma representada na figura 6.7 e constitúe un enlace á páxina de detalles desa entidade.

Figura 6.7: 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. Aquelas páxinas divididas de xeito horizontal (esquerda e dereita) pasan a verticais (arriba e abaixo) pois considerase o "scroll" vertical máis 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:

6.1.4.1. Portada

Pensouse na portada coma unha páxina que ofreza **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 máis "populares", concepto que se explica na sección 6.2.

Figura 6.8: Extracto do borrador do deseño da interface. Portada.

Figura 6.9: Deseño final da portada.

Figura 6.10: Deseño da portada para dispositivos móbiles. Á esquerda, a parte superior, á dereita, a parte inferior.

Na figura 6.8 amósase un **primeiro borrador do deseño da interface** 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 dereito. Isto cambia nos pequenos dispositivos, onde a información das emisoras pasa a situarse na parte máis baixa da páxina. Pódese ver este cambio comparando as figuras 6.9 e 6.10.

6.1.4.2. Páxinas de engadir e editar contido

As páxinas de engadir contido (incluída 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.

6.1.4.3. Páxinas de detalle

Ás páxinas de detalle son aquelas onde se amosa a **información completa dunha instancia** dalgunha 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 6.11 amosamos un borrador do deseño da interface 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.

6.1.4.4. Páxina de resultados de busca

A páxina de resultados de busca é á que se accede a través da **caixa de procura da cabeceira** (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**.

Figura 6.11: Extracto do borrador do deseño da interface. Detalles de programa.

6.1.4.5. Páxinas de xestión

As páxinas de xestión de emisión e xestión de administradores para programas e emisoras son semellantes. A figura 6.12 mostra un esquema de interface 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 eliminalos da emisión e ver os non emitidos para poder engadilos ao catálogo da emisora.

Figura 6.12: Extracto do borrador do deseño da interface. Xestión de emisión.

6.2. Actualización dos datos

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

6.2.0.1. 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 6.1.3.1.

6.2.0.2. 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 subscricións decidiuse en base a **diferencia de magnitude** 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 7

Implementación

7.1. Estrutura xeral do código fonte 64
7.2. Ficheiros de imaxe 65
7.3. A vista de engadir programa 66
7.4. A Internacionalización 67
7.5. O contador de escoitas 68
7.6. O panel de xestión 69
7.7. Execución dos procesos en Celery 70
7.8. Adaptabilidade a dispositivos móbiles 70

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

O código fonte pode atoparse no CD que se adxunta con esta memoria. Tamén está publicado de xeito íntegro en **GitHub** (Ver sección 3.9.2.1) no seguinte repositorio: https://github.com/FerbLee/RSS-Radio

7.1. Estrutura xeral do código fonte

Seguiuse a estrutura xenérica dos proxectos de Django que se amosa na figura 7.1. Creáronse dous paquetes: radio01 e rss_feed, dos cales só se amosa expandido o primeiro. Para máis detalles do segundo, pódese ver a figura 6.1 .

Figura 7.1: 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 aplicacións 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.

7.2. 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 peticións de axentes descoñecidos, utilizáronse as cabeceiras de Firefox. Pode verse o código no listado 7.1.

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.

```
def create_image(image_url):

  creation_date = timezone.now()
  original_image_name = os.path.basename(image_url)
  image_name = creation_date.strftime("%d%H%M%S") + '-' + original_image_name.
      lower()

  opener = urllib.request.build_opener()
  opener.addheaders = [('User-Agent','Mozilla/5.0')]
  urllib.request.install_opener(opener)
  image_file = urllib.request.urlretrieve(image_url)

  with open(image_file[0],'rb') as ifd:

  new_image_instance = Image()
  new_image_instance.path.save( image_name, File(ifd) )

  new_image_instance.creation_date = creation_date
  new_image_instance.name = original_image_name
  new_image_instance.alt_text = original_image_name.lower()
  new_image_instance.original_url = image_url

  new_image_instance.save()

  return new_image_instance
```

Listado 7.1: Código da función create_image do módulo rss_link_parsers.py

7.3. A vista de engadir programa

Como se mencionou no capítulo 6, 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 fillas:

- **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 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 7.2).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.

Figura 7.2: Formulario de engadir programa.

7.4. A Internacionalización

A tradución implementouse mediante as ferramentas que Django prové: *i18n* para a tradución de texto estático dos templates e *ugettext* 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 7.1.

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 7.3. 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 "Descrición"

#: 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 7.3: Fragmento do ficheiro .po para a tradución ao Galego.

7.5. 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.

7.6. 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 7.4. 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.

Figura 7.4: Panel de xestión dunha emisora. Vista de xestión da emisión.

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 7.4).

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

7.7. 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 chámanse *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 7.5.

Figura 7.5: Páxina de administración da aplicación, sección de procesos de Celery.

7.8. Adaptabilidade a dispositivos móbiles

Utilizouse a regra de CSS **@media** para definir diferentes estilos dependendo do tamaño da pantalla na que se amose o portal web. O listado 7.2 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
3   .grid-item{
4     border: 1px solid rgba(0, 0, 0, 0.8);
```

```
5       text-align: center;
6       padding: 20px;
7     }
8     .grid-container{
9       display: grid;
10      grid-template-columns: [col1-start]1fr [col2-start]1fr [col3-start]1fr [
            col4-start]1fr ;
11      grid-gap: 20px;
12    }
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 7.2: Extracto da folla de estilos de index.html

Pódense comparar os dous estilos definidos polo código anterior vendo as figuras 6.9 e 6.10 no capítulo de deseño.

Capítulo 8

Planificación e avaliación de custos

8.1. Concepción do proxecto 74

8.2. Iteracións 74

8.2.1. Primeira reunión (Iteración 0) 74

8.2.2. Iteración 1 75

8.2.3. Iteración 2 76

8.2.4. Iteración 3 77

8.2.5. Iteración 4 78

8.2.6. Iteración 5 78

8.2.7. Iteración 6 79

8.2.8. Iteración 7 80

8.2.9. Iteración 8 81

8.2.10. Iteración 9 82

8.2.11. Iteración 10 83

8.2.12. Iteración 11 84

8.3. Control da execución 85

8.3.1. Diagrama de Gantt 86

8.4. Avaliación de custos 98

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

8.1. Concepción do proxecto

Como xa se comentou na sección 1.3, o proxecto nace dun **encontro con membros de radios comunitarias**. Naquel mesmo momento obtivéronse unha serie de primitivas historias de usuario, máis centradas entón na redifusión e a organización que na escoita. A seguinte lista é unha **transcrición 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.

Cómpre clarificar que o listado anterior só se trata de impresións que, finalmente, inspiraron este traballo. En **modo algún se deben tomar coma obxectivos do proxecto**, pois estes non foron definidos até a primeira reunión ou iteración 0 (ver sección 8.2.1)

8.2. Iteracións

Como se explicou no capítulo 4 a cerca da metodoloxía, o desenvolvemento executouse de de forma **iterativa e incremental**. Unha iteración consiste nun ciclo de desenvolvemento cuns obxectivos a cumprir (historias de usuario) e unha reunión co cliente onde se fai unha valoración dos progresos e se recollen novas historias de usuario.

Ao ser isto un proxecto de final de carreira, contaranse coma reunións co cliente as sesións de **revisión de progresos** entre o director do proxecto e o alumno. Estas reunións tiveron unha periodicidade quincenal (aproximadamente, dependendo da dispoñibilidade)

8.2.1. Primeira reunión (Iteración 0)

Na primeira reunión estableceuse a **lista de obxectivos xerais do proxecto** repasados xa no listado da sección 1.4. 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 aínda non se tomara unha decisión firme de como facelo.

8.2.2. Iteración 1

Cumpríronse os obxectivos marcados, entregando un **deseño preliminar do modelo de datos** na data estimada (figura 8.1)

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 escoita 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.

Figura 8.1: It1: Diagrama de clases do primeiro borrador de deseño.

8.2.3. Iteración 2

Configurouse o entorno de Django e as ferramentas de desenvolvemento. Utilizouse a **técnica TDD** (ver capítulo 4 sobre a metodoloxía) 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 8.2: 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 á 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.

Cos obxectivos de instalación e configuración cumpridos, comezou a recompilación de historias de usuario:

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

8.2.4. 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 8.3.

Implementouse o soporte para os distintos formatos de ficheiro RSS: Ao principio da iteración só se podían engadir ficheiros extraidos 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.

Novas historias de usuario:

- **Refactorización** dos lectores de RSS.

Figura 8.3: It3: Interface web. Páxina de engadir programa.

- Implementación das clases de **imaxe e etiqueta**.
- Redefinir a **estrutura** das páxinas.

8.2.5. 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 6 sobre o deseño). Para completar o proceso de inserción de programas e episodios, implementáronse as clases da imaxe e etiqueta (Image e Tag).

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.

8.2.6. 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.

```
 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 8.4: It4: Resultado de consulta á base de datos relacionando programas cos seus tags.

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.

Novas historias de usuario:

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

8.2.7. Iteración 6

Introduciuse unha **clase de apoio** para outorgarlle ao usuario de Django unha serie de atributos necesarios no sistema. Introduciuse a comprobación de autenticación naquelas vistas que o requirisen, por exemplo, engadir programa, como se ve nas figuras 8.5 e 8.6.

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

Novas historias de usuario:

- Completar páxina de **edición de usuario**.
- Implementar clases para os **votos e os comentarios** dos episodios.

Figura 8.5: It6: index.html para usuario identificado.

Figura 8.6: It6: index.html para usuario anónimo.

- Implentar a clase de **emisora e canle**.
- Crear **deseño de interacción** das páxinas.

8.2.8. 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 6 sobre o deseño.

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

Novas historias de usuario:

- Dar soporte á **internacionalización**.

- Crear vista para engadir **novas emisoras**.

8.2.9. Iteración 8

Engadiuse o soporte de Django para a internacionalización e fíxose 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, fíxose 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 8.7).

Figura 8.7: It8: Vista de detalles de emisora.

Novas historias de usuario:

- Corrixir internacionalización (Posposto)

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

8.2.10. Iteración 9

Actualizáronse as vistas de detalle de programa e episodio (ver figura 8.8). 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 erros na vista de engadir contidos que foron corrixidos. Non se realizou ningunha actividade no tocante á internacionalización.

Figura 8.8: It9: Vista de detalles de episodio.

Novas historias de usuario:

- Corrixir **internacionalización**.
- Crear **roles** de usuario.
- Crear **panel de edición** e xestión de emisora e programa.
- Implementar as funcións de **procura** de contidos.

8.2.11. Iteración 10

Creáronse os roles de usuario e fíxose o panel de edición e xestión de programas e emisoras (figura 8.9). A implementación da procura quedou posposta por falta de tempo.

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

Novas historias de usuario:

- Comezar a redacción da **documentación**.
- Corrixir internacionalización.
- Implementar as funcións de procura de contidos.
- Adaptar o estilo para **dispositivos móbiles**.

Figura 8.9: It10: Panel de xestión de programa.

8.2.12. Iteración 11

Todos os obxectivos restantes foron cumpridos, sendo o máis destacable a nova vista de **resultados de busca** (Figura 8.10). 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.

Figura 8.10: It11: Páxina de resultados de busca.

8.3. Control da execución

Gardouse un rexistro das tarefas realizadas e a súa duración e considerouse un hipotético custo por hora de traballo. Clasifícanse nos seguintes tipos:

- **Organización:** Englóbanse aquí as tarefas de selección das ferramentas e as metodoloxías, o seu aprendizaxe e a instalación e configuración do entorno de desenvolvemento.
- **Análise:** Actividades de idear o funcionamento xeral da aplicación, identificando os distintos requirimentos a cumprir.
- **Deseño:** Tarefas de deseño do modelo de datos e da interfaz de usuario. Tamén entra a aplicación de patróns de deseño e formulación dos parámetros calculados.
- **Implementación:** Programación do definido nas etapas anteriores.
- **Probas:** Confección das probas de unidade e de integración. Probas manuais da interface de usuario.
- **Documentación:** Escritura da presente memoria.

Na táboa 8.1 amósase o tempo empregado polo total das tarefas pertencentes a cada tipo do listado anterior. Do mesmo xeito, gardouse un rexistro do tempo que se adicou ao traballo por iteración. Pode verse na táboa 8.2.

Tipo de tarefa	Horas de traballo	Horas de traballo(%)
Organización	16	2.6
Análise	55	9.1
Deseño	41	6.8
Implementación	271	44.7
Probas	55	9.1
Documentación	133	21.9
Total	607	100.0

Táboa 8.1: Tempo empregado acumulado por tipo de tarefa.

Iteración	Horas de traballo	Horas de traballo(%)
Primeira reunión	2	0.3
Iteración 1	8	1.3
Iteración 2	12	2.0
Iteración 3	10	1.7
Iteración 4	20	3.3
Iteración 5	63	10.4
Iteración 6	60	9.9
Iteración 7	66	10.9
Iteración 8	75	12.4
Iteración 9	55	9.1
Iteración 10	66	10.9
Iteración 11	48	7.9
Iteracións so de documentación	122	20.1
Total	607	100.0

Táboa 8.2: Tempo empregado nas distintas etapas de traballo.

8.3.1. Diagrama de Gantt

A continuación proporciónase un diagrama de Gantt dividido, por claridade, en distintas imaxes. Para a súa correcta interpretación cómpre ter en conta que o representado non é o resultado dunha planificación predictiva, senón a evolución real das tarefas a través do tempo.

Isto é así debido á utilización das metodoloxías áxiles. Como se explicou no capítulo 4 sobre a metodoloxía, o desenvolvemento é iterativo e incremental, de xeito que o desenvolvemento está orientado ás historias de usuario definidas en cada iteración.

Figura 8.11: Diagrama de Gantt. Novembro-Decembro.

Figura 8.12: Diagrama de Gantt. Decembro.

Figura 8.13: Diagrama de Gantt. Decembro-Xaneiro

Figura 8.14: Diagrama de Gantt. Xaneiro-Febreiro

Figura 8.15: Diagrama de Gantt. Febreiro

Figura 8.16: Diagrama de Gantt. Febreiro-Marzo

Figura 8.17: Diagrama de Gantt. Marzo-Abril

Figura 8.18: Diagrama de Gantt. Abril

Figura 8.19: Diagrama de Gantt. Abril-Maio

Figura 8.20: Diagrama de Gantt. Maio

Figura 8.21: Diagrama de Gantt. Maio-Xuño

8.4. Avaliación 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, suporemos un custo por hora común para todos os tipos de tarefa vistos na sección anterior. Nun proxecto en equipo, sería posible que os distintos tipos de tarefas se levasen a cabo por persoal de distinta cualificación e salario, polo que o cálculo sería máis complexo.

Segundo a legalidade vixente no momento da entrega desta memoria, o custo social para un traballador con contrato temporal, por ser dunha duración menor de un ano, incluíndo as continxencias comúns da empresa (23'60 %), o desemprego por contrato de duración determinada a tempo completo (6,70 %) o FOGASA (0,20 %) e a formación profesional (0,60 %), supón un 31,10 % de custos socias que debe abonar unha empresa sobre o salario bruto[50].

Para a estimación desta contía, suponse un **salario bruto anual de 30.000€** (trinta mil euros), polo tanto, seguindo o cálculo anterior, os custos sociais ascenderán a 9330€.

$$CustoEmpresa = SalarioAnual + CustosSociais = 30000 + 9330 = 39330euros$$

O calendario laboral do ano 2018 en España consta de 244 días laborables. Un empregado acollido ao convenio colectivo de analista-programador conta con 23 días de vacacións anuais e unha xornada de 8 horas, polo tanto, o **custo por hora** para a empresa calcúlase da seguinte forma:

$$\frac{CustoEmpresa}{(DiasLaborables - DiasVacacions)HorasXornada} = \frac{39330}{(244 - 23)8} = 22{,}25euros/hora$$

Para a realización deste proxecto **non se adquiriu ningún hardware** específico nin se contrataron servizos de terceiros. As **ferramentas** utilizadas, expostas no capítulo 3 sobre a tecnoloxía utilizada, foron na súa totalidade **libres e gratuítas**, polo que non se fixo gasto algún en licenzas.

Non se terán en conta os custos en **luz eléctrica**, conexión a Internet nin material de oficina (folios, bolígrafos...) empregado. Tampouco aqueles gastos derivados do uso de equipos informáticos propios.

Tendo en conta todo o anterior e, baseándonos nos datos da táboa 8.1, faise a **estimación de custo total deste proxecto** que se pode ver na táboa 8.3.

Tipo de tarefa	Custo(€)
Organización	356.00
Análise	1223.75
Deseño	2024.75
Implementación	6029.75
Probas	912.25
Documentación	2959.25
Total	**13505.75**

Táboa 8.3: Custo total estimado do proxecto.

Capítulo 9

Probas do sistema

9.1. Plan de probas 102

9.2. Probas de unidade 102

9.3. Probas de integración 107

9.4. Probas de aceptación 110

Neste capítulo explicarase o proceso de probas ao que se someteu o sistema desenvolvido.

9.1. Plan de probas

Durante as primeiras fases do desenvolvemento utilizouse a técnica **TDD** (ver capítulo 4 sobre metodoloxía) polo que se comezou realizando **probas de unidade automatizadas** para as clases do modelo de datos e as do módulo rss_link_parsers.py.

As **probas de integración** comezaron á vez que o desenvolvemento das funcións e clases de vista do módulo views.py. Son tamén tests de unidade pero utilizando un simulador de peticións GET e POST.

Finalmente, realizouse un conxunto de **tests de aceptación** de xeito manual. Utilizáronse para validar o correcto funcionamento da **interface**.

9.2. Probas de unidade

Son aquelas que verifican o **correcto funcionamento individual das compoñentes**. 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. Os casos de proba están 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** incluído no proxecto (ver figura 6.1)

Utilizouse a biblioteca *django.test* de Django, a cal, mediante o uso do paquete *unittest* 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      u1 = User.objects.create(username='userp1')
12      u2 = User.objects.create(username='userp2')
13      User.objects.create(username='userp3')
14
15      p1.programadmin_set.create(user=u1,type=ADMT_OWNER[0])
16      p1.programadmin_set.create(user=u2,type=ADMT_ADMIN[0])
17
18      t1 = Tag.objects.create(name='pt1',times_used=1)
19      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 metodos 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 coma administradores e máis 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 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.
         html'
25     rlp,fd = self.initialize_test()
26
27     p1 = rlp.parse_program(fd)
28
29     self.assertEqual(p1.name,'Hasta Los Kinders')
30     self.assertEqual(p1.language,'es-ES')
31     self.assertEqual(p1.original_site,program_web)
32     self.assertEqual(p1.rss_link_type,IVOOX_TYPE[0])
33
34
35   def test_parse_episode(self):
36
37     rlp,fd = self.initialize_test()
```

```
38
39      #Expected info
40      exp_titlte = 'CiudadanoKinders: Como hacer un monologo'
41      exp_pub_date = datetime.datetime(2010, 9, 17, 20, 45, 24, tzinfo=pytz.utc)
42      exp_file = 'http://www.ivoox.com/ciudadanokinders-como-hacer-
            monologo_mf_368919_feed_1.mp3'
43      exp_web = 'http://www.ivoox.com/ciudadanokinders-como-hacer-monologo-audios
            -mp3_rf_368919_1.html'
44      exp_original_id = 'http://www.ivoox.com/368919'
45
46      p1 = rlp.parse_program(fd)
47      entry_dict = rlp.get_entry_list(fd)[0]
48
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      self.assertNotEqual(e1,[])
82
83      e1 = e1[0]
84      self.assertIsInstance(e1,Episode)
85
86      self.assertEquals(e1.image,p2.image)
87
88      # Clean copied image
```

```
    p2.image.delete()
```

Listado 9.2: Probas de unidade dos metodos 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 6.1.3.1), 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.

```
class UpdatePopularityDaemonTests(TestCase):


  def setUp(self):

    u1 = User.objects.create(username='userUPD1')
    u2 = User.objects.create(username='userUPD2')
    s1 = Station.objects.create(name='RadioTest2')

    RSS1 = TEST_AUX_FILE_PATH + 'hasta-los-kinders_original.xml'
    rlp = ParserIvoox(RSS1,u1)
    hlk = rlp.parse_and_save()

    hlk_ep1 = hlk.episode_set.all()[0]
    hlk_ep1.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u1,episode=
        hlk_ep1))

    hlk_ep2 = hlk.episode_set.all()[1]
    hlk_ep2.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u1,episode=
        hlk_ep2))

    hlk_ep3 = hlk.episode_set.all()[2]
    hlk_ep3.vote_set.add(Vote.objects.create(type=DISLIKE_VOTE[0],user=u1,
        episode=hlk_ep3))
    hlk_ep3.vote_set.add(Vote.objects.create(type=LIKE_VOTE[0],user=u2,episode=
        hlk_ep3))

    RSS2 = TEST_AUX_FILE_PATH + 'falacalado_podomatic.xml'
    rlp = ParserPodomatic(RSS2,u1)
    fc = rlp.parse_and_save()
    fc.subscribers.add(u1)
    fc.broadcast_set.add(Broadcast.objects.create(station=s1,program=fc,
        schedule_details='Friday 21:00'))

    fc_ep1 = fc.episode_set.all()[0]
    fc_ep1.downloads = 1000
```

```
32        fc_ep1.save()
33
34
35      def test_update_pop_rating_all_programs(self):
36
37        hlk = Program.objects.get(name='Hasta Los Kinders')
38        fc = Program.objects.get(name="fala calado's podcast")
39
40        self.assertEqual(hlk.rating,50)
41        self.assertEqual(fc.popularity,0)
42
43        #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 compoñen os procesos executados polo servidor para actualizar os programas (ver sección 6.2) 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 comezar 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.

9.3. Probas de integración

Son aquelas que serven para **verificar o traballo conxunto de distintas compoñentes**. Estes casos de test, por regra xeral, tratan de probar as conexións entre compoñentes ignorando o funcionamento interno[51].

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**, dispoñible 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.

```
class IndexViewTests(TestCase):


  @classmethod
  def setUpTestData(cls):

    u1 = User.objects.create(username='userIV1')

    RSS1 = TEST_AUX_FILE_PATH + 'hasta-los-kinders_original.xml'
    rlp = ParserIvoox(RSS1,u1)
    hlk = rlp.parse_and_save()
    hlk.popularity = 10 # Check if first
    hlk.save()

    RSS2 = TEST_AUX_FILE_PATH + 'falacalado_podomatic.xml'
    rlp = ParserPodomatic(RSS2,u1)
    rlp.parse_and_save()

    # Total of 10 + 2 programs
    for i in range(1,11):

    rssl = 'http://dummy_link_iv_' + str(i) + '.xml'
    pname = 'ProgramIV_' + str(i)
    Program.objects.create(name=pname,rss_link=rssl)

    # Create 30 stations
    for i in range(1,31):

    sname = 'StationIV_' + str(i)
    Station.objects.create(name=sname)


    def test_view_uses_correct_template(self):

    resp = self.client.get(reverse('rss_feed:index'))
    self.assertEqual(resp.status_code, 200)

    self.assertTemplateUsed(resp, 'rss_feed/index.html')


  def test_program_list(self):

    resp = self.client.get(reverse('rss_feed:index'))
    self.assertEqual(resp.status_code, 200)

    self.assertEqual( len(resp.context['program_list']),4)
    self.assertTrue(resp.context['program_list'].has_next())
    self.assertFalse(resp.context['program_list'].has_previous())

    p1 = resp.context['program_list'][0]
    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 vénse algunhas probas realizadas para a vista de *index*, que se corresponde coa páxina principal da aplicación. 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 realízase 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': [
            ''], 'form_station-profile_img': [''],
27      'form_station-broadcasting_method': ['fm'], 'form_station-broadcasting_area
            ': [''],
28      'form_station-broadcasting_frequency': [''], 'form_station-streaming_link':
             [''],
```

```
29      'form_station-description': [''], 'form_station-website': [''], '
            form_station-location': [''']}
30
31      self.client.post(reverse('rss_feed:add_content'),  form_station)
32
33      s1 = Station.objects.get(name='StationIV1')
34
35      self.assertIsInstance(s1,Station)
36      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.

9.4. Probas de aceptación

O código puramente de **frontend** (Javascript, HTML...) foi probado de xeito manual e non automatizado. Tamén se contou cun usuario alleo ao proxecto para realizar un pequeno conxunto de probas.

Capítulo 10

Conclusións

10.1. Coñecementos acadados **113**
10.2. Traballo Futuro **113**

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 1.4)

- 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.
- Puxéronse a disposición dos usuarios **ferramentas de procura** de contidos por texto e máis 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, fóronse intuíndo novas necesidades que os usuarios poderían ter. Destas, implementáronse as seguintes:

- Panel de administración: Provese dunha interface web de administración para que un superusuario 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 estudante 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. Traballo Futuro

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 descrición 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 relacións en Django.

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

Atributo	Tipo	Descrición
@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	Descrición
@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	Descrición
@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	Descrición
@id	Serial	Clave primaria
image	ForeignKey	Clave foránea da entidade Image
name	Char(200)	Nome do programa
description	Text	Texto descritivo 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 subscritores 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	Descrición
@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	Descrición
@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 | TVChannel | RadioInternet | 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	Descrición
@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	Descrición
@id	Serial	Clave primaria
name	Char(50)	Nome do tag en minúsculas. Ten que ser único
times_used	PositiveIntegerField	Cantidade 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	Descrición
@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	Descrición
@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 cumprir o que comunmente se chama "As catro liberdades esenciais"[12]:

- **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 publícase 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 quen o desexe. 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.

Apéndice C

Manual de usuario

Neste apéndice apórtase un manual de referencia para os usuarios da web.

Figura C.1: Portada da web para usuario anónimo

Na figura C.1 amósase o aspecto da portada da web para un usuario anónimo con números identificando as distintas áreas:

- **1:** Enlace á páxina de login.
- **2:** Enlace á páxina de rexistro.
- **3:** Caixa de procura por texto.
- **4:** Selector de idioma.

- **5:** Enlace ao menú de creación de novo programa e emisora. Require identificación.

- **6:** Sección de episodios recentes. Amósanse os 4 últimos episodios engadidos ao sistema independentemente do seu programa. Para ver os 4 seguintes, pódese premer o botón marcado co **8**.

- **7:** Lista de emisoras presentes no sistema.

- **8:** Botón para cargar a seguinte páxina.

- **9:** Sección de programas ordenados por popularidade. Amósanse os 4 primeiros tamén con opción de cargar a seguinte páxina.

C.1. Usuarios

C.1.1. Crear

Para crear un novo usuario, prémese un botón **2** da figura C.1. Cargarase a páxina de rexistro de usuario. Basta con cubrir os campos como se amosa no exemplo da figura C.2. Se non hai ningún erro, o usuario quedará autenticado e será redirixido á súa páxina de detalles (figura C.3)

C.1.2. Editar

Para acceder ao menú de edición de usuario, prémese o botón de edición marcado co **10** na figura C.3. O menú de edición pódese ver na figura C.4. Nótese que o nome de usuario non é editable e que o cambio de password é un formulario aparte.

Se a edición se completa de forma correcta, vólvese á vista de detalles de usuario.

Benvido, por favor identifícate ou rexístrate.

RSS Radio

Procura

Galego (gl)

Cambiar

Rexístrate

Nome de usuario:

peter

Campo requirido. Non pode ser maior de 150 caracteres. Aceptanse Letras, números e os símbolos @/./+/-/_

Nome:

Peter

Apelidos:

Griffin

Enderezo de correo electrónico:

peter@familyguy.com

Campo requirido

Contrasinal:

•••••••••

- O teu password non pode ser semellante ao resto dos datos persoais.
- O teu password ter alomenos 8 caracteres.
- O teu password non pode ser un utilizado comunmente.
- O teu password non pode ser completamente numérico

Confirmación do contrasinal:

•••••••••

Introduza o mesmo password que antes, para a verificación.

Localización:

Quahog, Rhode Island

Descrición:

Pai de familia. Estrela dos debuxos animados.

Avatar:

Choose file pg.jpeg

New Image:

Rexístrate

Volver

Figura C.2: Rexistro de usuario.

Figura C.3: Detalles de usuario. Información do perfil.

Figura C.4: Editar usuario.

C.2. Accións de ouvinte

C.2.1. Escoitar e seguir unha emisora

Para realizar accións sobre unha emisora cómpre ir á súa páxina de detalles. As emisoras poden ser "seguidas" polos usuarios como forma de engadila a "favoritas". A continuación explícanse os elementos da páxina de detalles de emisora marcados na figura C.5.

- **11:** Botón de seguimento.
- **12:** Reprodutor de emisión en directo.
- **13:** Lista dos últimos episodios publicados ordenada por data.
- **14:** Lista de programas emitidos pola emisora.
- **15:** Usuarios seguidores da emisora.

C.2.2. Subscribirse a un programa

Para subscribirse ou cancelar unha subscrición previa, o usuario debe ir á páxina de detalles do programa en cuestión. A continuación explícanse os elementos da páxina de detalles de programa marcados na figura C.6.

- **16:** Botón de subscrición.
- **17:** Enlace á páxina de resultados de busca polo tag "comedy".
- **18:** Lista dos últimos episodios ordenada por data de publicación dos audios.
- **19:** Lista de emisoras nas que se emite o programa.
- **20:** Usuarios subscritos ao programa.

C.2.3. Accións sobre os episodios

Accedendo á vista de detalles de episodio pódese escoitar mediante streaming ou descargar o ficheiro de audio. Os episodios poden votarse e comentarse. A continuación explícanse os elementos da páxina de detalles de episodio marcados na figura C.7.

- **21:** Botóns de voto positivo ou negativo. O usuario sempre pode desfacer ou cambiar o seu voto.
- **22:** Reprodutor do ficheiro de audio do episodio.
- **23:** Enlace á páxina de resultados de busca por tag (Neste exemplo, o episodio non ten tags de seu.)
- **24:** Sección de comentarios do episodio.

RSS Radio
Identificado coma peter Pechar sesión.
Procura
Galego (gl)
Cambiar

CUAC RESISTE

Emisora: Cuac FM

Seguir 11

Visita a web
Localización: A Coruña
Descrición: A radio comunitaria da Coruña.
Medio: fm
Área: A Coruña
Frecuencia: 103.4
En directo:
0:00 12
Últimos episodios: 13

15x34 El Desinformativo 11-6-2018
El Desinformativo

15x33 El Desinformativo 4-6-2018
El Desinformativo

5x18 Spoiler
Spoiler

15x32 El Desinformativo 29-5-2018
El Desinformativo

seguinte

Programas asociados: 14

El Desinformativo
Martes 20:00

Que no es Poco
Luns-Venres 08:00

Spoiler
Mércores 21:00

Seguidores: 15
fer ferblee atreides ash ozzy

Volver

Figura C.5: Detalles de emisora.

RSS Radio

Identificado coma **peter** Pechar sesión.

Procura

Galego (gl) Cambiar

Programa: Que no es Poco

Subscribirte 16

Autor: Fer Lee

Engadido o: 25 de abril de 2018 ás 11:23 **por** ferblee

Lingua: es-ES **Calificación:** 50%

Categorías comedy 17

Descrición: Antiguo morning show humorístico de CUAC FM, A Coruña. Actuales perpetradores de especiales de Nochevieja. Monguer Wappers a tiempo parcial. Presentado por Diego de la Vega, con Fer Liñares, Víctor Grande, Anchonio, Guito y Marta Chil. A los controles Iverson y Mr.Borji

Podcast: http://www.ivoox.com/podcast-que-no-es-poco_sq_f110383_1.html

Sitio Web: https://www.facebook.com/quenoespoco/

Episodios 18

Nochevieja 2017-18 QNEP: Programa Completo
1 de xaneiro de 2018 ás 18:17
Que no es Poco

QNEP 2017-18 AVANCE: ¡Todos a la cárcel!
28 de decembro de 2017 ás 18:57
Que no es Poco

QNEP 2017-18 Villancico: El españolismo es total
21 de decembro de 2017 ás 18:30
Que no es Poco

Nochevieja 2016-17 QNEP: Programa Completo
1 de xaneiro de 2017 ás 13:09
Que no es Poco

seguinte

Emitido en: 19

Cuac FM

Subscritores: 20

ash atreides fer ferblee

Volver

Figura C.6: Detalles de programa.

RSS Radio

Identificado coma peter Pechar sesión

Procura

Galego (gl) Cambiar

Programa: Que no es Poco

Episodio: Nochevieja 2017-18 QNEP: Programa Completo

Gústame 21

Non me gusta

Gustou: 2 **Disgustou:** 0 **Escoitas** 3
Publicado o: 1 de xaneiro de 2018 ás 18:17

0:00 / 2:39:43 22

Categorías 23

Resumo: En octubre de 2017, Cuac FM cesó forzosamente en su presencia en la 103.4 a causa de una injusta sanción de la Xunta de Galicia. Sin embargo, los incorregibles locutores de Que No Es Poco decidieron continuar sus emisiones en un estudio clandestino en el desván de la lujosa Mansión De La Vega's. A las 2 horas de empezar a emitir, ya tenían a la policía irrumpiendo en su casa porque los muy imbéciles gustaban de decir desde donde emitían para que el de las pizzas no se perdiera. Fueron encerrados en el penal de máxima seguridad de Zapateira en régimen de aislamiento. Sin embargo, un rumor ha empezado a extenderse entre los presos del módulo: Alguien está organizando una fuga para la mismísima noche de Fin de Año... #freequenoespoco #respetec Escucha YA el tradicional especial de nochevieja de Que no es poco!!! Este año contaremos con Diego de la Vega, el señor Iverson, Fer Liñares, Walter "la bestia" López, Isa Lema, Iago Gordillo y muchos mas. Con Mariano Fernandez a los controles.

Comentarios: 24

Comentario:

atreides o 17 de xuño de 2018 ás 16:59

Gustoume máis o episodio anterior, pero non estivo mal

Deixa un comentario

Publicar

Volver

Figura C.7: Detalles de episodio.

C.2.4. Contidos favoritos

Nos exemplos anteriores, o usuario seguiu unha emisora e subscribiuse a un programa. Este contido está agora dispoñible de xeito máis fácil desde a portada e máis desde o seu perfil como se pode ver nas figuras C.8 e C.9.

Figura C.8: Contidos favoritos na portada.

Figura C.9: Contidos favoritos na vista de perfil de usuario.

C.3. Procura de contidos

O sistema proporciona 2 formas de procura de contidos: Por texto, mediante a caixa marcada co **3** na figura C.1 , ou por tag, premendo nos enlaces correspondentes (Ver número **17** na figura C.6). Ambas levan á páxina de resultados de procura, onde se amosarán os episodios, programas, emisoras e usuarios que cumpran os criterios da busca.

Na figura C.10 móstrase un exemplo de procura pola palabra "radio".

Figura C.10: Páxina de resultados de procura. Exemplo de busca pola palabra radio.

C.4. Accións de produtor de contidos

C.4.1. Crear unha emisora

Para engadir unha emisora, o usuario ten que ir á vista de "engadir contido", mediante o enlace marcado co **5** na figura C.1. Unha vez alí, cóbrense os campos do formulario como se ve na figura C.11

Se os datos introducidos son correctos, rediríxese ao usuario á páxina de detalles da nova emisora (figura C.12). O usuario creador da emisora ten permisos de "propietario" sobre a citada emisora.

C.4.2. Xestionar unha emisora

Se o usuario é propietario ou xestor dunha emisora, poderá ver o botón marcado coma **25** na figura C.12. Este levaralle ao panel de xestión da emisora, no que se poden realizar as seguintes operacións:

C.4.2.1. Editar perfil

Vista amosada na figura C.13. Desde ahí pódense editar os datos do perfil da emisora.

C.4.2.2. Xestionar emisión

Vista amosada na figura C.14. Desde aí pódense engadir programas á emisión e retiralos da mesma.

- **26:** Selector de programa. Úsase para seleccionar o programa que se queira engadir á emisión.
- **27:** Caixa de texto para especificar o horario de emisión do programa a engadir.
- **28:** Área de programas xa emitidos. Ás filas seleccionadas aplicaráselles á acción definida en **29**.
- **29:** Botóns de acción. Determinan se os programas seleccionados son para borrar ou para actualizar os seus datos de emisión.

RSS Radio
Identificado coma **ferblee** **Pechar sesión**
Procura
Galego (gl)
Cambiar

Engadir Novo Contido:

Engadir Programa | Engadir Emisora

Nova Emisora:

Nome:

Lume FM

Logo:

Choose file lume_logo.png

New Image:

Imaxe de Perfil:

Choose file RQ_pp2.jpeg

New Image:

Medio:

Radio FM/AM

Área:

A Coruña

Frecuencia:

90.6

Enlace ao streaming:

http://radiolume.org/streami

Descrición:

Radio de proba

Sitio web:

http://radiolume.org

Localización:

A Coruña

Enviar | Cancelar

Volver

Figura C.11: Engadir emisora.

Figura C.12: Vista de detalles de emisora: Nova emisora.

Figura C.13: Panel de xestión de emisora: Editar perfil.

Figura C.14: Panel de xestión de emisora: Xestionar emisión.

C.4.2.3. Xestionar administradores

Vista amosada na figura C.15. Desde aí pódenselle conceder e revogar aos usuarios permisos sobre a emisora. Só os usuarios de nivel propietario poden acceder a esta vista.

Importante: O propietario orixinal da emisora é o usuario que a crea. Este pode nomear outros propietarios pero, unha vez nomeados, non os poderá "degradar" xa que estarán ao seu mesmo nivel. Recoméndase prudencia á hora de conceder os permisos.

- **30:** Selector de usuario. Úsase para seleccionar o usuario ao que se lle queiran dar permisos.

- **31:** Selector de permisos. Poden ser de administrador (limitados) ou de propietario (totais)

- **32:** Área de usuarios xa con permisos. Ás filas seleccionadas aplicaráselles á acción definida en **33**.

- **33:** Botóns de acción. Determinan se os usuarios seleccionados son para borrar ou para actualizar os seus permisos.

C.4.2.4. Borrar emisora

Vista amosada na figura C.16. Desde aí pódese eliminar a emisora. So os propietarios teñen acceso a esta vista.

Figura C.15: Panel de xestión de emisora: Xestionar administradores.

Figura C.16: Panel de xestión de emisora: Borrar emisora.

C.4.3. Crear un programa

Ao igual que a emisora, é necesario acceder ao menú de engadir contido a través do botón **5** na figura C.1 e cubrir o formulario que se ve na figura C.17.

Se os datos introducidos son correctos, rediríxese ao usuario á páxina de detalles da novo programa (figura C.18), onde tamén poderemos ver todos os seus episodios, creados automaticamente. O usuario creador do programa ten permisos de "propietario" sobre el e mailos seus episodios.

- **34:** Opcións de compartición. As opcións son *libre* e *desactivada* segundo se desexe que as emisoras poidan emitir libremente o novo programa ou non.

- **35:** Opcións de comentarios. As opcións son *Permitir* e *Deshabilitar* os comentarios nos episodios do novo programa.

Figura C.17: Engadir programa.

Figura C.18: Vista de detalles de program: Novo programa.

C.4.4. Xestionar un programa

Se o usuario é propietario ou xestor dun programa, poderá ver o botón marcado coma **36** na figura C.18 que leva ao panel de xestión do programa. O sistema non ofrece ferramentas de xestión non edición dos episodios de xeito individual. As opcións definidas para o programa aplicaranse a todos os seus episodios sen excepcións.

As ferramentas que se utilizan neste panel funcionan do mesmo xeito que as do panel de xestión de emisora. Ante calquera dúbida, consultar as seccións C.4.2.2 e C.4.2.3.

Neste panel pódense realizar as seguintes operacións:

C.4.4.1. Editar perfil

Esta vista é semellante á de editar o perfil da emisora, porén, podemos ver na C.19 que as opcións son, aquí, moito máis limitadas debido a que a información é obtida dunha fonte externa (o ficheiro RSS) e non da entrada manual de datos por parte do usuario.

Figura C.19: Panel de xestión de programa: Editar perfil.

C.4.4.2. Xestionar emisión

Vista amosada na figura C.20. O sistema dá a oportunidade ao propietario do programa de cesar a emisión nunha emisora, pero, como é lóxico, non de modificar o seu horario de emisión nesa emisora.

Figura C.20: Panel de xestión de programa: Xestionar emisión.

C.4.4.3. Xestionar administradores

Vista amosada na figura C.21. Desde aí pódenselle conceder e revogar aos usuarios permisos sobre a emisora. Só os usuarios de nivel propietario poden acceder a esta vista.

Importante: O propietario orixinal do programa é o usuario que o crea. Este pode nomear outros propietarios pero, unha vez nomeados, non os poderá "degradar" xa que estarán ao seu mesmo nivel. Recoméndase prudencia á hora de conceder os permisos.

Figura C.21: Panel de xestión de programa: Xestionar administradores.

C.4.4.4. Borrar programa

Vista amosada na figura C.22. Desde aí pódese eliminar o programa. So os propietarios teñen acceso a esta vista.

Importante: O borrado dun programa implica a eliminación de todos os seus episodios así como das súas imaxes relacionadas. Os datos non serán recuperables.

Figura C.22: Panel de xestión de programa: Borrar programa.

Bibliografía

[1] W3Schools. Javascript html dom, Extraído o 22/05/2018. URL https://www.w3schools.com/js/js_htmldom.asp. VII, 19

[2] Wikipedia. JSON, Data types, syntax and example, Extraído o 05/05/2018. URL https://en.wikipedia.org/wiki/JSONs. VII, 23

[3] W3Counter. Browser & Platform Market Share May 2018, Extraído o 10/06/2018. URL https://www.w3counter.com/globalstats.php?year=2018&month=5. VII, 31

[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, páx. 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] Unión de Radios Comunitarias de Madrid. audio.urcm.net, Extraído o 18/06/2018. URL http://audio.urcm.net. 3

[10] Cyrus Farivar. iTunes 4.9 First Look: Apple takes on Podcasting, 28/06/2005. URL https://www.macworld.com/article/1045522/podcastingfirstlook.html. 9

[11] Iago Veloso Abalo. Radioco, Extraído o 30/05/2018. URL http://radioco.org/es. 11

[12] The Free Software Foundation. ¿Qué es el software libre?, Extraído o 12/06/2018. URL http://www.gnu.org/philosophy/free-sw.html. 12, 17, 121

[13] Mark Lutz. *Learning Python*. O'Reilly, 5 edition, 2013. ISBN 978-1-449-35573-9. 18

[14] Anaconda Inc (Continuum Analytics). Anaconda distribution, Extraído o 22/05/2018. URL https://www.anaconda.com/distribution. 18

[15] W3Schools. HTML Introduction, Extraído o 22/05/2018. URL https://www.w3schools.com/html/html_intro.asp. 18

[16] 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. 18

[17] W3Schools. What is CSS?, Extraído o 23/05/2018. URL https://www.w3schools.com/css/css_intro.asp. 19

[18] 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. 19

[19] W3Schools. CSS Grid layout module, Extraído o 23/05/2018. URL https://www.w3schools.com/css/css_grid.asp. 20

[20] 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. 20

[21] javascript.info. An Introduction to JavaScript, Extraído o 24/05/2018. URL https://javascript.info/intro. 20

[22] MDN Web docs. JavaScript, Extraído o 24/05/2018. URL https://developer.mozilla.org/en-US/docs/Web/JavaScript. 20

[23] Bear Bibeault, Yehuda Katz, Aurelio De Rosa. *jQuery in action*. Manning, 3 edition, 2015. ISBN 978-1-617-29207-1. 20

[24] Python Software Foundation. django-static-jquery 2.1.4, 06/05/2015. URL https://pypi.org/project/django-static-jquery. 21

[25] Alan Beaulieu. *Learning SQL*. O'Reilly, 2005. ISBN 978-0-596-00727-0. 21

[26] W3Schools. Introduction to SQL, Extraído o 25/05/2018. URL https://www.w3schools.com/sql/sql_intro.asp. 21

[27] Elliotte Rusty Harold. *XML Bible*. IDG Books Worldwide, 1999. ISBN 0-7645-3236-7. 21

[28] RSS Advisory Board. RSS 2.0 Specification, 30/03/2009. URL http://www.rssboard.org/rss-specification. 23

[29] 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. 23

[30] The LaTeX project. An introduction to LaTeX, Extraído o 05/06/2018. URL https://www.latex-project.org/about. 24

[31] James Bennett. *Practical Django projects*. Apress, 2009. ISBN 978-1-4302-1938-5. 24

[32] Django Software Foundation. Django 2.0 release notes, 02/12/2017. URL https://docs.djangoproject.com/en/2.0/releases/2.0. 25

[33] The Celery Project. Introduction to Celery, Extraído o 28/05/2018. URL http://docs.celeryproject.org/en/latest/getting-started/introduction.html. 26

[34] Lovisa Johansson. *Getting started with RabbitMQ and CloudAMQP*. Codes AB, 2 edition, 2017. 26

[35] Edmond Woychowsky. *Ajax, Creating web pages with asynchronous JavaScript and XML*. Prentice Hal, 2006. ISBN 0-13-227267-9. 27

[36] The Apache Software Foundation. Licensing of Distributions, Extraído o 29/05/2018. URL https://www.apache.org/licenses/. 27

[37] Netcraft. April 2018 Web Server Survey, 26/04/2018. URL https://news.netcraft.com/archives/2018/04/26/april-2018-web-server-survey.html. 27

[38] The PostgreSQL Global Development Group. About PostgreSQL, Extraído o 25/05/2018. URL https://www.postgresql.org/about. 28

[39] Jet Brains S.R.O. PyCharm Editions Comparison, Extraído o 27/05/2018. URL https://www.jetbrains.com/pycharm/features/editions_comparison_matrix.html. 29

[40] Scott Chacon, Ben Straub. *Pro Git*. Apress, 2 edition, 2018. 29

[41] The Fedora Project. Licensing FAQ, Extraído o 29/05/2018. URL https://fedoraproject.org/wiki/Licensing:FAQ. 30

[42] Benito van der Zander. Welcome to TeXstudio, Extraído o 29/05/2018. URL https://www.texstudio.org. 30

[43] Mozilla Foundation. Mozilla Foundation End-User Licensing Agreements, Extraído o 10/06/2018. URL https://www.mozilla.org/en-US/about/legal/eula. 31

[44] The GIMP Team. About GIMP, Extraído o 11/06/2018. URL https://www.gimp.org/about. 31

[45] ProjectLibre. ProjectLibre Blog, Extraído o 15/06/2018. URL https://www.projectlibre.com/blog/license. 32

[46] V.V.A.A. Manifesto polo Desenvolvemento Áxil de Software, 2001. URL http://agilemanifesto.org/iso/gl/manifesto. 34

[47] Kent Beck, Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2 edition, 2012. ISBN 0-321-27865-8. 35

[48] Kent Beck. *Test-Driven Development By Example*. Addison Wesley, 2002. ISBN 978-0321146533. 36

[49] Ralph R. Young. *The Requirements Engineering Handbook*. Artech House, 2004. ISBN 1-58053-266-7. 42

[50] Gobierno de España Ministerio de empleo y seguridad social. Bases y tipos de cotización 2018, Extraído o 19/06/2018. URL http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm. 98

[51] 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. 107