Escrutinio Social

Desarrolladores y desarrolladoras de escrutinio-social

31 de mayo de 2021

Índice general

Escrutinio Social es una plataforma web para la realización de un escrutinio provisorio y auditoría ciudadana a partir de fotos documentos aportados por fiscales partidarios o extraídos del escrutinio oficial. Es software libre bajo la licencia BSD.

CAPÍTULO 1

Introducción

Escrutinio Social es una plataforma de código abierto para la realización de escrutinios electorales,

Es particularmente útil para que organizaciones de la sociedad civil, partidos politicos y/o ciudadanos particulares realicen operativos de «escrutinios paralelos» a manera de auditoría y contralor de un escrutinio provisorio oficial.

Sus características principales son:

- Recepción y procesamiento de imágenes de

documentos fuentes (actas de mesa) - Interfaz para la clasificación, reporte de problemas o carga de datos - Posibilidad de carga parcial para maximizar la velocidad de obtención de primeros resultados (Por ejemplo, sólo fuerzas mayoritarias y datos minimos que permitan calcular «Otros») - Versatilidad en la estructura de la elección. Una «categoría» puede existir en las mesas de un distrito y no en otras. - Interfaz de visualización de resultados, con distinto nivel de agregación - Carga resultados procedente de otras fuentes en formato CSV - Algoritmo de priorizacion de tareas configurable dinámicamente. - Sistema «antitrolling» para reducir cargas maliciosas - Consolidación de datos basada en coincidencia (método «doble-ciego», configurable a N coincidencias) - Cálculo de proyecciones - Soporte de datos históricos - API REST - Scripts para la importación de datos de la elección (estructura de mesas, opciones electorales, etc.) - Configuración y optimizaciones para un entorno de producción crítico

Diversas organizaciones y equipos técnicos de diferentes partidos políticos han contribuído al desarrollo de este software a lo largo de los años. Entre otras, podemos mencionar

- Open Data Argentina (anteriormente Open Data Córdoba)
- Mueve Latinoamerica
- Equipo técnico del Instituto Patria
- Equipo técnico del Frente Córdoba Ciudadana
- Equipo técnico de la Unión Cívica Radical de Córdoba

CAPÍTULO 2

Manual de usuaries y administración

Este capitulo es la documentación **no técnica** para usuaries y responsables del operativo.

2.1 Roles

2.1.1 Fiscales *dataentry*

Qué hacen.

2.1.2 Administradores y administradoras

2.2 El camino feliz

- Distintas maneras de que los documentos lleguen al sistema
- Identificación y carga *por categoría*
- Concepto de «consolidación»
- Agregación de resultados.
- Carga de datos parcial ¿como funciona?
- ¿Con qué tarea se sigue? El concepto de *scheduler*

2.3 La estructura de una elección

- Explicar estructura de «arbol». (distrito, sección, circuito, etc.)
- Elecciones internas abiertas: multiples opciones de las mismas fuerzas.
- Salvo para casos minimos, se genera programáticamente a traves de comandos (ver doc técnica)

2.4 Configuración

Hay muchas configuraciones que pueden realizarse desde la administración. Tienen valores por defecto que pueden cambiarse dinámicamente.

- Listar variables constance
- Importancia del orden de las opciones

CAPÍTULO 3

Historia

En el año 2013, durante las elecciones legislativas, el escrutinio provisorio resultó llamativo en la provincia de Córdoba, Argentina, ya que estuvio en disputa el resultado de la categoría Diputado/a Nacional: el partido minoritario parecía haber obtenido el resultado para obtener representación parlamentaria por primera vez, pero en un momento de la noche el resultado cambió y fue finalmente la Unión Cívica Radical (primera fuerza de aquella elección) la que se llevó esa banca.

Fue tal la polémica y tan estrecho el márgen que Martín Gaitán, a la postre uno de los creadores de Escrutinio Social, hizo un llamamiento desde su weblog para revisar «cargas sospechosas» del escrutinio provisorio.

Para esto obtuvo primero mediante un pequeño software (un «scraper») todas las imágenes y la carga de datos oficial correspondiente a cada telegrama. Aplicando algunas inferencias estadísticas básicas (por ejemplo, es sospechoso que el resultado en una mesa para un partido sea muy distinto que el promedio en las demas mesas en esa escuela o circuito), obtuvo listas de actas cuya carga era plausible de contener errores y debía ser revisada.

Pronto ese trabajo se viralizó y si bien finalmente no hubo modificaciones en los resultados, fue el puntapie inicial para diferentes iniciativas relacionadas a la transparencia electoral.

A finales de ese mismo año, un grupo de investigadores de la Facultad de Matemática, Física, Astronomía e Informática (FaMAF) de la Universidad Nacional de Córdoba propuso realizar una «hackatón» enfocada en la temática electoral que continuara y potenciara la idea germinal. Ese encuentro se llamó «Democracia con Códigos» y fue donde se discutieron y escribieron las primeras lineas de código de lo que sería «Escrutinio Social». La idea era profesionalizar y sistematizar la carga de datos de voluntaries (superar el «formulario de Google» del *post* original) y lograr que la detección de anomalías se haga más automática. En ese mismo encuentro se conocieron los fundadores (y de alguna manera dió origen) de la ONG Open Data Córdoba (ODC).

En 2017, durante otras elecciones legislativas nacionales, el Frente Córdoba Ciudadana (representación de Unidad Ciudadana, el partido fundado por Cristina Fernández de Kirchner, en la provincia de Córdoba) llevó a Pablo Carro como candidato a diputado nacional (que resultaría finalmente electo). El equipo de campaña necesitaba un software para su escrutinio interno y se recuperaron las ideas y bases de aquella hackatón, que eran de código abierto, y se lo llevó a un estado funcional. Ese software se llamó «Carreros». Además del sistema de escrutinio paralelo, incluía muchas otras funcionalidades para la optimización de la campaña y el operativo electoral.

Para ODC la transparencia electoral siempre fue parte principal de su agenda y fueron quienes se tomaron el trabajo de «extraer» de Carreros la parte estrictamente relacionada al escrutinio, volviendo a publicarlo bajo su nombre original.

El éxito de las experiencias en Córdoba permitió que el mismo software sea la base para el que se usaría en operativo electoral a nivel nacional durante las elecciones presidenciales del año 2019 del Frente de Todos, que llevó a la presidencia a Alberto Fernández. Muchísimas funcionalidades tendientes a la escala (doble carga, antitrolling, optimizaciones varias) fueron agregadas durante esta etapa, con un equipo conformodo por voluntarios de distintas agrupaciones políticas que apoyaban a esta formula.

Pasadas las elecciones, el software se liberó nuevamente adoptando ya el estatus de un proyecto de código abierto independiente (bajo la «organización» https://github.com/EscrutinioSocial), que tiene el afán de ser usado en elecciones de cualquier nivel en diversos lugares del mundo.

3.1 Apariciones en los medios

- https://www.cronista.com/economiapolitica/Cordoba-se-realizara-un-escrutinio-interpartidario-inedito-en-America-latina-20190
  html
- https://www.youtube.com/watch?v=51SX4LrdLJ0

CAPÍTULO 4

La funcionalidad *antitrolling*

4.1 Objetivo

El sistema puede ser usado por cualquiera que obtenga credenciales. Si alguna persona, o grupo de personas lo quisiera usar de forma maliciosa, entonces el sistema lo debe detectar y bloquear.

Entendemos que hay casos en los que las personas (sin intención de maldad) cargan mal algún resultado, por ejemplo en ciertos casos que la letra no es legible. Estos casos igual los queremos marcar como dudosos porque si la persona tiene muchas pequeñas fallas entonces puede causar que la carga de resultados nunca converja.

El mecanismo entonces, detecta a quienes intencional o no, cargan los resultados de forma incorrecta.

Las personas en el sistema son *fiscales*, al marcarlos como *troll* lo invalidamos, y se invalidan **todas** las cargas anteriores.

4.2 Scoring

Hay distintas métricas que el sistema de detección de trolls tiene en cuenta. La configuración de dichas métricas se encuentra en el archivo de *settings.py* o la interfaz de configuración

```
'SCORING_MINIMO_PARA_CONSIDERAR_QUE_FISCAL_ES_TROLL': (150000, 'Valor de scoring que␣
↪debe superar un fiscal para que la aplicación lo considere troll.', int),
'SCORING_TROLL_IDENTIFICACION_DISTINTA_A_CONFIRMADA': (1, 'Cuánto aumenta el scoring␣
↪de troll por una identificacion distinta a la confirmada.', int),
'SCORING_TROLL_PROBLEMA_MESA_CATEGORIA_CON_CARGA_CONFIRMADA': (1, 'Cuánto aumenta el␣
↪scoring de troll por poner "problema" en una MesaCategoria para la que se␣
↪confirmaron cargas.', int),
'SCORING_TROLL_PROBLEMA_DESCARTADO': (1, 'Cuánto aumenta el scoring de troll al␣
↪descartarse un "problema" que él reporto.', int),
'SCORING_TROLL_DESCUENTO_ACCION_CORRECTA': (1, 'Cuánto disminuye el scoring de troll␣
↪para cada acción aceptada de un fiscal.', int),
```

El primer valor es el umbral que se usará para marcar a una fiscal como troll. Esto se puede configurar si detectamos que durante la carga se nos escapan fiscales o estamos siendo muy crueles.

Por otro lado si una mesa se encuentra identificada por dos fiscales y una tercera la carga mal, entonces se le suman 200 puntos por error de cargar mal el dato de la mesa. En esta categoría están por ejemplo, el número de mesa, sección, distrito, etc.

El tercer parámetro es para evitar que marquen las fotos de las actas con problema cuando en realidad otras fiscales le cargan datos y estos son confirmados. Entendemos que puede haber fotos mal subidas, pero si dos otras fiscales la pueden leer y cargar, entonces restamos puntos a quien esta queriendo arruinar fotos que se ven bien.

Se puede consultar el puntaje de un *fiscal* mediante el método: *scoring_troll()*.

Desde la web se pueden admisnitrar los fiscales, y marcarlos o desmarcarlos.

4.3 ¿Cuánto pesan las cargas?

Si un acta tiene dos cargas coincidentes y una tercera que difiere, entonces al fiscal que cargo alguna categoría (o todas) mal, se le suma la diferencia de votos a modo de penalización.

Por ejemplo, si para presidente dos cargas coinciden en 40 votos y un tercer fiscal carga 30, entonces a ese tercer fiscal se le suman 10 puntos a su nivel de troll.

4.4 Invalidación de cargas

Al detectar a un fiscal como troll, todas sus cargas anteriores se invalidan: *Carga.objects.filter(invalidada=True)*.

Cuando se corra el comando asincrónico: *consolidar_identificaciones_y_cargas.py* se revisarán y cambiarán: * Cargas: los valores cargados de votos * Identificaciones: la identificación de mesa del acta

En el caso de las categorias, en el objeto: *MesaCategoria*, se puede ver que el **STATUS** puede degradarse de *consolidada_dc* a *sin_consolidar* o *sin_cargar*. Dependiendo el nivel de daño que haya hecho el fiscal.

La idea es que si un acta tiene una carga realizada por un fiscal que ahora es troll, entonces no podemos confiar en la doble carga que realizó. El sistema le enviará el acta a otro fiscal para que pueda volver a cargarla.

En el caso de las identificaciones, pueden pasar a no estar procesadas y esa identificacion ser invalida. En ese caso, también necesitamos que otro fiscal constate con una segunda carga los datos de identificación de la mesa.

4.5 Qué ve el Troll

Lo lindo es que nuestros trolls, seguirán jugando con el sistema y sus acciones no impactarán en nuestros resultados. No se les avisará que fueron degradados de fiscales a trolls.

CAPÍTULO 5

Preguntas frecuentes

5.1 ¿Qué tipo de archivos pueden ser «actas»?

Se pueden subir imágenes o archivos PDF.

En el caso de un PDF, cada página se convierte y se almacena como una imágen, pero sólo la primera es considerada para la identificación. Cuando esta se identifica de una manera válida (es decir, hay suficientes coincidencias), entonces todas las imágenes correspondientes a páginas del mismo PDF original son automaticamente indentificadas de la misma manera.

5.2 ¿Qué procesamiento se realizan a las imágenes?

Una imágen original subida al sistema puede tener problemas salvables como mala iluminación o perspectiva.

Para eso hay una pequeña barra de herramientas que permite editar la imágen original y guardar una copia mejorada que le permita al misme u otre *dataentry* una carga de datos más eficiente.

CAPÍTULO 6

Guía de administración técnica

6.1 Importación de datos para una elección

6.1.1 A. Para comenzar de cero con la base

Ejecutar los comandos::

```
$ make down
$ make build
$ make setup-dev-data
$ make shell-app
```

el último comando nos lleva al shell de la app, desde donde ejecutaremos el resto de los comandos

```
# python manage.py createcachetable
```

6.1.2 B. Pasos para cargar DISTRITOS, SECCIONES, CIRCUITOS, ESCUELAS Y MESAS

1. Borrar el distrito de prueba de la BD a través del admin de la app. Con eso borramos todos los datos de prueba, secciones, circuitos, mesas y resultados inclusive.
2. Crear todos las categorías de distribución geográfica de las mesas: Distritos, Secciones y Circuitos

```
# ./manage.py importar_circuitos <path>/circuitos.csv
```

Formato del archivo:

circuito_nro,circuito_name,seccion_nro,seccion_name,distrito_nro,distrito_name

1. Crear todos los lugares de votación

```
# ./manage.py importar_escuelas <path>/escuelas.csv
```

Formato del archivo:

escuela_nro,distrito_nro,seccion_nro,circuito_nro,escuela,direccion,localidad,latitud,longitud

1. Crear mesas

```
# ./manage.py importar_mesas <path>/mesas.csv
```

Formato del archivo:

mesa,circuito,lugar_votacion,seccion,distrito,electores

Pendiente Entiendo que se puede usar la misma función para setear los electores, pero no lo probé.

Aclaración: La diferenica con el comando importar_escuelas_y_mesas es el formato. Este último comando pide para cada escuela «desde-hasta» indicando nro de mesa de cada escuela.

6.1.3 C. Para configurar categorías electorales y opciones

Categorías Generales indica qué se vota en una determinada elección. Por ejemplo, Diputados Nacionales, Presidente, etc

Categorías relaciona las categorías generales con los lugares geográficos en los que se vota. Por ejemplo, Presidente se vota en todo el país, pero Diputados Nacionales, tiene candidatos diferenciados en cada provincia, por lo tanto una categorías sería Diputados Nacionales por Córdoba.

Partidos nos permite una clasificación más general de las opciones. Es muy útil para cuando hay varias opciones para un mismo partido.

Opciones las opciones «Positivas» son las correspondientes a cada partido, también se almacena entre las opciones los votos en blanco, votos impugnados, votos nulos y la metadata, cantidad de electores y de votantes.

1. Desde el admin de la app

- Borrar todas las «categorías generales» de la base
- Borrar todos los partidos de la base.
- Borrar todas las opciones de la base.

1. Crear las categorías generales

La categoría «Presidente y vice» debe tener slug «PV». Si se quiere cambiar el slug, es necesario cambiar la configuración.

```
# ./manage.py importar_categorias_generales <path>/categorias_generales.csv
```

Formato del archivo:

nombre,slug

1. Asociar la categoría general creada a la Eleccion en el admin de la app

Pendiente - habría que ver si hace falta o lo puede hacer el script.

1. Crear las categorías que se votan. A su vez, asociar a las mesas en las que se vota cada categoría.

```
# ./manage.py importar_categorias <path>/categorias.csv
```

1. Asociar las categorías creadas a Eleccion en el admin

Pendiente - habría que ver si hace falta o lo puede hacer el script.

1. Crear las opciones partidarias y asociarlas a las categorías correspondientes.

```
# ./manage.py importar_opciones <path>/opciones.csv
```

1. Crear blancos, nulos, etc. y los asociarlos a todas las categorías.

```
# ./manage.py setup_opciones_basicas
```

6.1.4 D. Más info

Quedan algunos comentarios mas de configuracion que se podrían revisar en https://github.com/EscrutinioSocial/escrutinio-social/wiki/Importaci %C3 %B3n-de-datos-y-deploy

o en la wiki original escrutinio/escrutinio-paralelo/wikis/importación-datos-general-2019

6.2 Comandos útiles

6.2.1 Importar datos desde email

El comando `importar_actas_desde_email` permite conectarse a una o más cuentas IMAP para bajar correos electrónicos y convertir sus archivos adjuntos en imágenes a clasificar.

Cada email puede tener una o más imágenes adjuntas.

Para esto es necesario configurar la variable de entorno `IMAPS` como un string json que contiene una lista de diccionarios con los datos de las cuentas.

Por ejemplo:

```
IMAPS=[{"email": "email_de_actas@gmail.com", "host": "imap.gmail.com", "user": "email_
↪de_actas@gmail.com", "pass": "xxxx", "mailbox": "INBOX"}]
```

Luego

```
$ python manage.py -v 3 --include-seen --only-images importar_actas_desde_email
```

Vea `python manage.py importar_actas_desde_email --help` para una ayuda sobre las opciones.

CAPÍTULO 7

Despliegue en producción

Hay 4 componentes

- Aplicación web (wsgi)
- Base de datos Postgresql
- Demonios (scheduler, consolidador, importador de actas desde email)
- Archivos estáticos de la aplicacion y subidos por los usuarios

7.1 Despliege a Digital Ocean

Ingresar a Digital Ocean y habilitar la integración con Github para el repositorio y branch que corresponda.

7.1.1 Spaces

Crear una nueva instancia:

- Seleccionar la región (ej. San Francisco 3)
- Habilitar CDN
- Dejar restringido *File Listing*
- Elegir nombre único
- Seleccionar el proyecto

Ir a la configuración del *Space* y editar la sección de *CORS* para aceptar solicitudes desde el dominio de la aplicación

7.1.2 Database

Crear un cluster de base de datos:

- Seleccionar la base de datos (Postgres 12)
- Seleccionar la configuración del cluster: Basic nodes, 1GB RAM / 1vCPU / 10GB HD
- Seleccionar la región (ej. nyc)

Una vez que se termina de aprovisionar el cluster. Ir a *Users & Databases* y crear el usuario y base de datos *escrutinio-social*.

7.1.3 App Platform

Setup

```
doctl auth init --context <name>
doctl auth switch --context <name>

curl -L https://github.com/ko1nksm/shdotenv/releases/latest/download/shdotenv --
↪output /usr/local/bin/shdotenv
chmod +x /usr/local/bin/shdotenv
```

Crear un archivo .env-deploy y definir las siguientes variables:

```
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_STORAGE_BUCKET_NAME=
AWS_S3_ENDPOINT_URL=
DB_CLUSTER_NAME=
APP_REGION=
APP_DOMAIN=
APP_NAME=
DJANGO_SECRET_KEY=
GUNICORN_WORKERS=
GITHUB_REPO=
BRANCH_NAME=
IMAPS_CONFIG=
```

Test

Para inspeccionar la especificación luego del reemplazo de variables de entorno:

```
make test-app-platform-spec
```

Create

Para hacer el despliegue por primera vez:

```
make create-app-platform-deploy
```

Update

Para aplicar una actualización de la especificación, obtener el ID de la aplicación:

```
doctl apps list
```

luego, reemplazando el <app-id> con el valor que corresponda:

```
make update-app-platform-deploy app-id=<app-id>
```

CAPÍTULO 8

Cómo contribuir

8.1 Guía de estilo

La intención de tener una guía de estilo de codificación es lograr una consistencia que facilite la lectura y la comprensión del proyecto.

No nos limitamos a las minucias del formato, sino a fomentar la adopción de aquellos *idioms* que hacen que nuestro código resulte «pythónico» (y «djangoso» (?)), asumiendo las convenciones más comunes de la comunidad en general, y

> *"A language that doesn't affect the way you think about programming, is not worth knowing."* – Alan Perlis

Por supuesto, también fomentamos e intetamos aplicar las buenas prácticas del desarrollo de software que son agnósticas del lenguaje.

8.1.1 Algunas notas iniciales

- Esta guía de estilo es un `pep8.upgrade(esta_guia)`. Es decir, siempre que no estén especificadas, valen las recomendaciones de la guía de estilo oficial de Python. Y algunas que están aqui, actualizan o modifican las de PEP8.
- No queremos ser «dictadores del formato». Justamente la intención de tener esto es ganar tiempo y confort en el desarrollo, y si esa búsqueda nos obliga a lo contrario, pierde su sentido.

  Es válido pedir o recomendar cambios de formato en una revisión de código, pero no queremos bloquear una integración porque alguien dejó un espacio de menos o de más.
- Aceptamos usar herramientas de «autoformateado» como `yapf`, `black`, `isort`, o las que ofrezca tu editor preferido si considerás que te ahorran trabajo, pero siempre respetando las convenciones del proyecto.

  Usalas siempre sobre bloques de código específicos y no de manera automática sobre todo un módulo.

- Esta guía no está escrita en piedra y puede cambiar cuando la experiencia o los gustos del equipo lo dicten. Para proponer cambios, aplicaran los mismos criterios y herramientas que para el código: enviar un MR/PR y obtener los votos necesarios para su integración!

8.1.2 Convenciones

- El ancho máximo de linea recomendado es de 109 caracteres. (79 es demasiado poco para nuestro gusto por los nombres de variables expresivos)
- Nombres
  - de clases en `CamelCase`
  - de constantes en `UPPERCASE`
  - todo el resto (variables, funciones, argumentos, módulos, etc.), en `snake_case`
- Imports
  - Como recomienda PEP8, nunca hacer `from bla import *`.
  - Si la lista de objetos a importar de un determinado namespace realmente es muy grande, o sabemos que va a cambiar mucho, se puede importar el namespace. El ejemplo típico es `django.db.models` que contiene un montón de objetos usados en la definición de modelos

```
from django.db import models

class Foo(models.Model):
   campo = models.BooleanField ...
```

  - Si el espacio de nombre a importar es muy largo o puede colisionar con otro, usar un alias
- Si entran en el ancho máximo, hacer los imports en la misma línea. Si no entran, preferir un formato de sangrado vertical en orden alfabético

```
from namespace import (
   bar,
   foo,
   ...,
)
```

  No olvidar la coma en el último elemento, así el «diff» es menos verborrágico cuando la se agregue una nueva línea.
- El orden los imports es por procedencia y luego alfabeticamente.

```
<imports de la stdlibs>
...

<imports de requerimientos>
....

<imports del proyecto>
....
```

- Strings

- Para strings cortos. preferimos comilla simple (apóstrofes) sobre comillas doble, salvo, obviamente, que se necesiten apóstrofes dentro del texto.
- Para strings multilínea triple comilla doble, dejando las comillas solas en la primera y última línea siempre que se pueda
- Excepción con los docstrings, que preferimos aplicar la regla anterior de triple comilla aunque sean de una sola

```
    """
    Esta función recibe ``foo``
    """
```

- Preferimos f-strings por sobre otro tipo de interpolación. Pero la lógica «dentro» del f-string debe ser mínima, haciendo los precómputos necesarios.
- No nos gustan los backslashes. Si el texto es multilinea, usar comillas triples Si un texto es excepcionalmente largo, envolverlo en paréntesis

```
(
  "esto es un solo string "
  "repartido en muchas lineas"
)
```

- Docstrings
  - Nos gustan y sirven. Pero deben referirse a funcionalidad y no a explicar la implementación Para eso están los comentarios. . . y el código en sí .
  - Pueden formatearse en restructuredText, para que Sphinx los muestre bonitos.
- Objetos grandes
  - Aplica el mismo criterio de identación vertical de los imports

```
una_lista_larga = [
    'item1',
    'item2',
    [
        'item_de_item3',
        ...,
    ],
    ...,
]
```

  - Mismo criterio para diccionarios y otras estructuras de datos.
- Condiciones
  - No nos gustan los backslashes, preferimos envolver en paréntesis.
  - No nos gustan los paréntesis salvo que sean necesarios.
  - Cuando hay múltiples condiciones que requieren varias lineas y el costo de evaluación es despreciable, preferimos precalcular en variables que aporten semántica

```
requiere_parcial = self.mc.categoria.requiere_cargas_parciales
sin_consolidar = self.mc.status[:2] < MesaCategoria.STATUS.parcial_
↪consolidada_dc[:2])
if requiere_parcial and sin_consolidar:
    ...
```

Cuando la mera evaluación puede ser costosa computacionalmente, no hacer esto así usufructuamos los «cortocircuitos» del `if`

- La estructura ternaria (foo = "bar" if condicion else "baz") es útil siempre que la condicion no deba reusarse y todo entre en el ancho. En caso contrario
- Usar `foo is not bar` en vez de `not (foo is bar)`
- En caso de que inevitablemente una condición se compone de muchas condiciones parciales en varias lineas, preferir

- Funciones
  - No usar lambdas asignados a un nombre de variable. Definir con `def` en ese caso.
  - No nos gustan `map` y `filter`, preferimos comprenhensions. Especialmente si son generator expressions.
  - Sospechamos de los bloques de código que no entran en una pantalla Si una función se torna demasiado grande, preferimos factorizar pequeñas funciones auxiliares (que eventualmente pueden definirse de manera anidada para reutilizar definiciones del contexto).
- YAGNI
  - Evitamos la tentación de escribir o mantener código que no sirve pero «podemos llegar a necesitar». Borramos en vez de comentar código, con un commit prolijo acotado a ese fin y un claro mensaje, referencias en el issue tracker en los tickets relacionados, y otras maneras de dejar pistas si eventualmente hay que encontrarlo de nuevo.
  - El mismo criterio debería aplicarse a tests «skipeados». Salvo que sea evidente que alguien está trabajando sobre la funcionalidad relacionada a esos tests, si hay tests que no sirven (ni pasan) en el estado actual del proyecto, no deberian dejarse en el código.
- Django templates
  - Nos gusta dejar sangria de dos espacios para enfatizar los bloques. Es decir, que las etiquetas de django «sobresalgan» hacia la izquierda.

```
{% if incluir_votos %}
  <td id="votos_{{ partido.id|default:partido|lower }}" class="dato">{{ fila.
↪votos }}</td>
{% endif %}
...
```

  - Priorizar la legibilidad del template por sobre la del código (html o lo que sea) resultante. Por ejemplo, frecuentemente comentamos usando etiquetas de template en vez de etiquetas de html.
  - Siempre que aplique, usamos las mismas convenciones que para Python
- Querysets
  - En algunas APIs, como los queryset de django, es posible y útil «encadenar» llamadas a métodos. Las reglas de sangrado vertical aplican aqui.

```
query = qs.values(
    'mesa', 'status'
.annotate(
    total=Count('status')
).annotate(
    cuantos_csv=cuantos_csv
)
```

- Cuando las condiciones de filtrado son abundantes o dinámicas, se pueden definir con antelación utilizando diccionarios y/u objetos tipo `Q`.

CAPÍTULO 9

Instalación de un entorno de desarrollo

Escrutinio Social es un proyecto basado en Django (2.2), postgresql (9.6 o superior) y Python 3.7. Hemos puesto esfuerzo en simplificar todo lo posible el setup de un entorno de desarrollo

Para poner en marcha este entorno necesitamos contar con docker y docker-compose. Puedes seguir las instrucciones oficiales correspondientes a tu sistema operativo.

Para crear e inicializar los contenedores,

```
make build
make setup-dev-data
```

Para lanzar los servicios y la aplicación

```
make up
make shell-app
root@8e90b4b0175f:/src# python manage.py createcachetable
```

Luego podrás ingresar a http://localhost:8000/ y loguearte con `admin` / `admin`. Este usuario, además de ser fiscal (es decir, dataentry), tiene privilegios de superusuario, habilitándolo a subir actas.

Para detener los servicios de docker:

```
make stop
```

Los datos sintéticos que se cargan se tratan de una elección con tres opciones, 8 mesas (mesa 1 a 8) divididas en 2 secciones y 4 circuitos.

Una vez logueado, podés subir imágenes desde la opción «Subir actas» y asociarlas a alguna de las mesas. Eso te habilitará la opción de cargar actas y luego computar resultados.

Hay más comandos que pueden ser útiles.

- `make shell-app` y `make shell-db` para entrar a la consola de los contenedores
- `make log-app` y `make log-db` para ver el dump de outputs capturados
- `make down` para remover los contenedores y sus volúmenes de datos (para un fresh install)

- y más. Revisá el código de `Makefile`

9.1 Instalación local

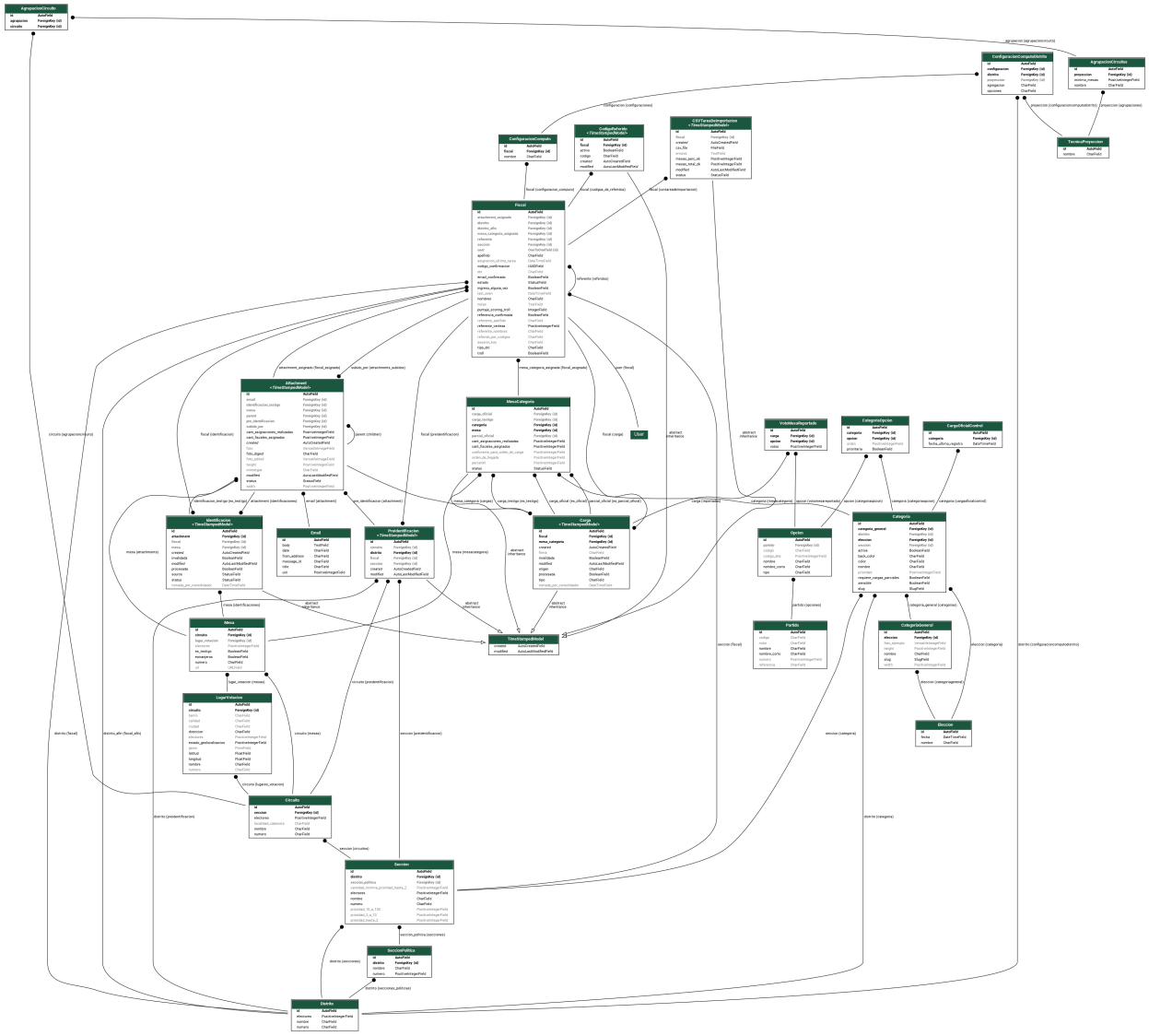
Si preferís no usar Docker podés seguir las instrucciones para armar un entorno de desarrollo local en nuestra wiki.

CAPÍTULO 10

Referencias para desarrollo

Esta sección contiene documentación autogenerada a partir del código fuente.

10.1 Modelos



class elecciones.models.**AgrupacionCircuito**(*id*, *circuito*, *agrupacion*)

Parámetros

- **id** (*AutoField*) – Id
- **circuito** (ForeignKey to *Circuito*) – Circuito
- **agrupacion** (ForeignKey to *AgrupacionCircuitos*) – Agrupacion

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**AgrupacionCircuitos**(**args*, ***kwargs*)

Representa un conjunto de circuitos que se computarán juntos a los efectos de una proyección.

Parámetros

- **id** (*AutoField*) – Id
- **nombre** (*CharField*) – Nombre
- **proyeccion** (ForeignKey to *TecnicaProyeccion*) – Proyeccion
- **minimo_mesas** (*PositiveIntegerField*) – Minimo mesas
- **circuitos** (*ManyToManyField*) – Circuitos

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**Carga**(**args*, ***kwargs*)

Es el contenedor de la carga de datos de un fiscal Define todos los datos comunes (fecha, fiscal, mesa, categoría) de una carga, y contiene un conjunto de objetos *VotoMesaReportado* para las opciones válidas en la mesa-categoría.

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **invalidada** (*BooleanField*) – Invalidada
- **tipo** (*CharField*) – Tipo
- **origen** (*CharField*) – Origen
- **mesa_categoria** (ForeignKey to *MesaCategoria*) – Mesa categoria
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal
- **firma** (*CharField*) – Firma
- **tomada_por_consolidador** (*DateTimeField*) – Tomada por consolidador
- **procesada** (*BooleanField*) – Procesada

exception DoesNotExist

exception MultipleObjectsReturned

actualizar_firma(*forzar=False*)

A partir del conjunto de reportes de la carga se genera una firma como un string

<id_opcion_A>-<votos_opcion_A>|<id_opcion_B>-<votos_opcion_B>...

Si esta firma iguala o coincide con la de otras cargas se marca consolidada.

listado_de_opciones()

Devuelve una lista de los ids de las opciones de esta carga.

opcion_votos()

Devuelve una lista de los votos para cada opción.

save(**args*, ***kwargs*)

si el fiscal es troll, la carga nace invalidada y ya procesada

class elecciones.models.**CargaOficialControl**(**args*, ***kwargs*)

Este modelo se agrega para guardar la fecha y hora del último registro de carga parcial oficial obtenido desde la planilla de cálculo de gdocs

Parámetros

- **id** (*AutoField*) – Id
- **fecha_ultimo_registro** (*DateTimeField*) – Fecha ultimo registro
- **categoria** (ForeignKey to *Categoria*) – Categoria

exception DoesNotExist

exception MultipleObjectsReturned

exception elecciones.models.**CargasIncompatiblesError**
Error que se produce si se pide la resta entre dos cargas incompatibles

class elecciones.models.**Categoria**(**args*, ***kwargs*)
Representa una categoría electiva, es decir, una «columna» del acta. Por ejemplo: Presidente y Vicepresidente, Intendente de La Matanza, etc)

Una categoría tiene habilitadas diferentes *opciones* que incluyen las partidarias (boletas) y blanco, nulo, etc.

Parámetros

- **id** (*AutoField*) – Id
- **eleccion** (ForeignKey to *Eleccion*) – Eleccion
- **categoria_general** (ForeignKey to *CategoriaGeneral*) – Categoria general
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **seccion** (ForeignKey to *Seccion*) – Seccion
- **slug** (*SlugField*) – Slug
- **nombre** (*CharField*) – Nombre
- **color** (*CharField*) – Color. Color para CSS (ej, red o #FF0000)
- **back_color** (*CharField*) – Back color. Color para CSS (ej, red o #FF0000)
- **activa** (*BooleanField*) – Activa. Si no está activa, no se cargan datos para esta categoría y no se muestran resultados.
- **sensible** (*BooleanField*) – Sensible. Solo pueden visualizar los resultados de esta cagtegoría con permisos especiales.
- **requiere_cargas_parciales** (*BooleanField*) – Requiere cargas parciales
- **prioridad** (*PositiveIntegerField*) – Prioridad
- **opciones** (*ManyToManyField*) – Opciones

exception DoesNotExist

exception MultipleObjectsReturned

property electores
Devuelve la cantidad de electores habilitados para esta categoría

opciones_actuales(*solo_prioritarias=False*, *excluir_optativas=False*)
Devuelve las opciones asociadas a la categoría en el orden correspondiente. Determina el orden de la filas a cargar, tal como se definen en el acta.

classmethod para_mesas(*mesas*)
Devuelve el conjunto de categorías que son comunes a todas las mesas dadas.

Por ejemplo, permite mostrar links válidos a las distintas categorías para una sección o circuito. Por ejemplo, si filtramos el circuito 1J o cualquiera se sus subniveles (escuela, mesa) se debe mostrar la categoría a Intentendente de La Matanza, pero no a intendente de San Isidro.

class elecciones.models.**CategoriaGeneral**(**args*, ***kwargs*)

A diferencia del modelo *Categoria*, éste representa una categoría sin asociación geográfica. Por ejemplo «Intendente», sin decir de dónde.

Sirve para poder de alguna manera agrupar a todas las categorías «Intendende de X», además de para permitir meter aquí atributos de visualización comunes a ellas, así como también nombres de columnas en, por ejemplo, el CSV.

Parámetros

- **id** (*AutoField*) – Id
- **eleccion** (ForeignKey to *Eleccion*) – Eleccion
- **slug** (*SlugField*) – Slug
- **nombre** (*CharField*) – Nombre
- **foto_ejemplo** (*VersatileImageField*) – Foto ejemplo
- **height** (*PositiveIntegerField*) – Image height
- **width** (*PositiveIntegerField*) – Image width

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**CategoriaOpcion**(*id*, *categoria*, *opcion*, *orden*, *prioritaria*)

Parámetros

- **id** (*AutoField*) – Id
- **categoria** (ForeignKey to *Categoria*) – Categoria
- **opcion** (ForeignKey to *Opcion*) – Opcion
- **orden** (*PositiveIntegerField*) – Orden. Orden en el acta
- **prioritaria** (*BooleanField*) – Prioritaria

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**Circuito**(**args*, ***kwargs*)

Define el circuito, perteneciente a una sección

Distrito -> Sección -> **Circuito** -> Lugar de votación -> Mesa

Parámetros

- **id** (*AutoField*) – Id
- **seccion** (ForeignKey to *Seccion*) – Seccion
- **localidad_cabecera** (*CharField*) – Localidad cabecera
- **numero** (*CharField*) – Numero
- **nombre** (*CharField*) – Nombre
- **electores** (*PositiveIntegerField*) – Electores

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**ConfiguracionComputo**(**args*, ***kwargs*)
Definición de modos de computar resultados por distrito.

Parámetros

- **id** (*AutoField*) – Id
- **nombre** (*CharField*) – Nombre
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**ConfiguracionComputoDistrito**(**args*, ***kwargs*)
Definición de modos de computar resultados para un distrito, de acuerdo a un usuario.

Parámetros

- **id** (*AutoField*) – Id
- **configuracion** (ForeignKey to *ConfiguracionComputo*) – Configuracion
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **agregacion** (*CharField*) – Agregacion
- **opciones** (*CharField*) – Opciones
- **proyeccion** (ForeignKey to *TecnicaProyeccion*) – Proyeccion

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**Distrito**(**args*, ***kwargs*)
Define el distrito o circunscripción electoral. Es la subdivisión más grande en una carta marina. En una elección provincial el distrito es único.

Distrito -> Sección -> Circuito -> Lugar de votación -> Mesa

Parámetros

- **id** (*AutoField*) – Id
- **numero** (*CharField*) – Numero
- **nombre** (*CharField*) – Nombre
- **electores** (*PositiveIntegerField*) – Electores

exception DoesNotExist

exception MultipleObjectsReturned

get_secciones()
Todas las secciones del distrito ordenadas por número.

class elecciones.models.**Eleccion**(**args*, ***kwargs*)
Es un modelo contenedor que representa, basicamente, un dia de elecciones.

Contiene *categorías*

La finalidad que tiene es permitir la persistencia de resultados de multiples elecciones (por ejemplo PASO, primarias, balotaje) para hacer análisis

Parámetros

- **id** (*AutoField*) – Id
- **fecha** (*DateTimeField*) – Fecha
- **nombre** (*CharField*) – Nombre

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**LugarVotacion**(**args*, ***kwargs*)

Define el lugar de votación (escuela) que pertenece a un circuito y contiene mesas. Tiene un representación geoespacial (point).

Distrito -> Sección -> Circuito -> **Lugar de votación** -> Mesa

Parámetros

- **id** (*AutoField*) – Id
- **circuito** (ForeignKey to *Circuito*) – Circuito
- **nombre** (*CharField*) – Nombre
- **direccion** (*CharField*) – Direccion
- **barrio** (*CharField*) – Barrio
- **ciudad** (*CharField*) – Ciudad
- **numero** (*CharField*) – Numero. Número de escuela
- **electores** (*PositiveIntegerField*) – Electores
- **geom** (*PointField*) – Geom
- **estado_geolocalizacion** (*PositiveIntegerField*) – Estado geolocalizacion. Indicador (0-10) de que confianza hay en la geolozalización
- **calidad** (*CharField*) – Calidad. calidad de la geolocalizacion
- **latitud** (*FloatField*) – Latitud
- **longitud** (*FloatField*) – Longitud

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The "force_insert" and "force_update" parameters can be used to insist that the «save» must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class elecciones.models.**Mesa**(**args*, ***kwargs*)

Define la mesa de votación que pertenece a un class:*LugarDeVotación*.

Sección -> Circuito -> Lugar de votación -> **Mesa**

Está asociada a una o más categorías electivas para los cuales el elector habilitado debe elegir.

Por ejemplo, la mesa 12 del circuito 1J de La Matanza, elige Presidente y Vice, Diputado de Prov de Buenos Aires e Intendente de La Matanza.

Parámetros

- **id** (*AutoField*) – Id

- **numero** (*CharField*) – Numero
- **es_testigo** (*BooleanField*) – Es testigo
- **circuito** (ForeignKey to *Circuito*) – Circuito
- **lugar_votacion** (ForeignKey to *LugarVotacion*) – Lugar de votacion
- **url** (*URLField*) – Url. url al telegrama
- **electores** (*PositiveIntegerField*) – Electores
- **extranjeros** (*BooleanField*) – Extranjeros
- **categorias** (*ManyToManyField*) – Categorias

exception DoesNotExist

exception MultipleObjectsReturned

fotos()
Devuelve una lista de tuplas (titulo, foto) asociados a la mesa, incluyendo cualquier version editada de una foto, para aquellos attachements que esten consolidados

Este método se utiliza para alimentar las pestañas en la pantalla de carga de datos.

invalidar_asignacion_attachment()
Efecto de que esta mesa tenía un attachment asociado y ya no lo tiene. Hay que: invalidar todas las cargas, y borrar el orden de carga de las MesaCategoria para que no se tengan en cuenta en el scheduling

metadata()
Las opciones metadatas comunes a las distintas categorías de la misma mesa reúsan el valor reportado. Se cargan hasta que se consolide en alguna categoría y las siguientes cargas reusarán sus valores reportados.

Este método devuelve la lista de tuplas de (opción metadata, número) para alguna de las cargas consolidadas testigo de la mesa. El número es la cantidad de «votos».

classmethod obtener_mesa_en_circuito_seccion_distrito(*mesa*, *circuito*, *seccion*, *distrito*)
Valida si existe una mesa con dicho codigo en el circuito y seccion indicados

class elecciones.models.**MesaCategoria**(**args*, ***kwargs*)
Modelo intermedio para la relación m2m `Mesa.categorias` mantiene el estado de las *cargas*

Parámetros

- **id** (*AutoField*) – Id
- **status** (*StatusField*) – Status
- **mesa** (ForeignKey to *Mesa*) – Mesa
- **categoria** (ForeignKey to *Categoria*) – Categoria
- **carga_testigo** (ForeignKey to *Carga*) – Carga testigo
- **carga_oficial** (ForeignKey to *Carga*) – Carga oficial
- **parcial_oficial** (ForeignKey to *Carga*) – Parcial oficial
- **percentil** (*PositiveIntegerField*) – Percentil
- **orden_de_llegada** (*PositiveIntegerField*) – Orden de llegada
- **coeficiente_para_orden_de_carga** (*PositiveIntegerField*) – Coeficiente para orden de carga

- **cant_fiscales_asignados** (*PositiveIntegerField*) – Cant fiscales asignados
- **cant_asignaciones_realizadas** (*PositiveIntegerField*) – Cant asignaciones realizadas

exception DoesNotExist

exception MultipleObjectsReturned

actualizar_coeficiente_para_orden_de_carga()
Actualiza *self.coeficiente_para_orden_de_carga* a partir de las prioridades por sección y categoría.

datos_previos(*tipo_carga*)
Devuelve un diccionario que contiene informacion confirmada (proveniente de campos de metadata o prioritarios compartidos) para «pre completar» una carga.

{opcion.id: cantidad_votos, . . . }

Si una opcion está en este diccionario, su campo votos se inicilizará con la cantidad de votos y será de sólo lectura, validando además que coincidan cuando la carga se guarda.

firma_count()
Devuelve un diccionario que agrupa por tipo de carga y firmas. Este método se usa para testing solamente. Por ejemplo:

```
{'total': {
    '1-10|2-2': 1,
    '1-10|2-1': 1'
 },
 'parcial': {
    '1-10': 1}
}
```

invalidar_cargas()
Por alguna razón, hay que marcar todas las cargas que se hicieron para esta MesaCategoria como inválidas.

recalcular_coeficiente_para_orden_de_carga()
Actualiza el valor de *self.coeficiente_para_orden_de_carga* a partir de las prioridades por sección y categoría, **sin** disparar el *save* correspondiente.

classmethod recalcular_coeficiente_para_orden_de_carga_para_categoria(*categoria*)
Recalcula el coeficiente_para_orden_de_carga de las MesaCategoria correspondientes a la categoría indicada que estén pendientes de carga. Se usa como acción derivada del cambio de prioridades en la categoría.

classmethod recalcular_coeficiente_para_orden_de_carga_para_seccion(*seccion*)
Recalcula el coeficiente_para_orden_de_carga de las MesaCategoria correspondientes a la sección indicada que estén pendientes de carga. Se usa como acción derivada del cambio de prioridades en la categoría.

class elecciones.models.**MesaCategoriaQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

anotar_prioridad_status()
Dados los status posibles anota el querset `prioridad_status` con un valor entero (0, 1, 2, . . .) que se corresponde con el índice del status en la constante `config.PRIORIDAD_STATUS`, que es configurable vía Constance. Sirve para poder ordenar por «prioridad de status»:

```
>>> qs.anotar_prioridad_status().order_by('prioridad_status')
```

Por defecto, esta prioridad sale del orden definido en `settings.MC_STATUS_CHOICE`.

con_carga_sensible_y_parcial_pendiente()
Devuelve las mesas categorías que tengan cargas parciales pendientes y que correspondan a categorías sensibles (que son las realmente prioritarias).

debug_mas_prioritaria()
Esta función invoca a ordenadas_por_prioridad() y además imprime los campos de orden. Sirve sólo para debuggear.

identificadas()
Filtra instancias que tengan orden de carga definido (que se produce cuando hay un primer attachment consolidado).

mas_prioritaria()
Devuelve la intancia más prioritaria del queryset.

redondear_cant_fiscales_asignados_y_de_asignaciones()
Redondea la cantidad de fiscales asignados y de asignaciones a múltiplos de `settings.MIN_COINCIDENCIAS_CARGAS` para que al asignar mesas no se pospongan indefinidamente mesas que fueron entregadas ya a algún fiscal.

siguiente()
Devuelve mesacat con carga pendiente más prioritaria

siguiente_para_ub()
Devuelve mesacat con carga pendiente más prioritaria

sin_cargas_del_fiscal(*fiscal*)
Excluye las instancias que tengan alguna carga del fiscal indicado.

sin_consolidar_por_csv()
Excluye las instancias consolidadas por CSV.

sin_consolidar_por_doble_carga()
Excluye las instancias consolidadas con doble carga.

sin_problemas()
Excluye las instancias que tengan problemas.

solo_de_cats_activas()
Excluye a las instancias de categorías no activas.

class elecciones.models.**Opcion**(**args*, ***kwargs*)
Una opción es lo que puede elegir hacer el elector con voto para una categoría.

Incluye las opciones partidarias (que redundan en votos positivos) o blanco, nulo, etc, que son opciones no positivas y no asociadas a partidos. También pueden ser opciones de «metainformación» del acta, como totales de votos positivos, votos válidos, etc, que si bien pueden calcularse indirectamente a partir de otros datos, a veces se prefiere cargar para minimizar las decisiones en quien carga datos.

Más de una opción puede estar asociada al mismo partido, (por ejemplo varias listas de un espacio en una PASO) pero actualmente sus votos se computan agregados.

Parámetros

- **id** (*AutoField*) – Id
- **tipo** (*CharField*) – Tipo
- **nombre** (*CharField*) – Nombre
- **nombre_corto** (*CharField*) – Nombre corto
- **codigo** (*CharField*) – Codigo. Código de opción

- **partido** (ForeignKey to *Partido*) – Partido
- **codigo_dne** (*PositiveIntegerField*) – Codigo dne. Nº asignado en la base de datos de resultados oficiales

exception DoesNotExist

exception MultipleObjectsReturned

property color
Devuelve el color del partido si existe o blanco. Permite destacar con un color la fila en la tabla de resultados

class elecciones.models.**Partido**(**args*, ***kwargs*)
Representa un partido político o alianza, que contiene *opciones*.

Parámetros

- **id** (*AutoField*) – Id
- **numero** (*PositiveIntegerField*) – Numero
- **codigo** (*CharField*) – Codigo. Código de partido
- **nombre** (*CharField*) – Nombre
- **nombre_corto** (*CharField*) – Nombre corto
- **color** (*CharField*) – Color
- **referencia** (*CharField*) – Referencia

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**Seccion**(**args*, ***kwargs*)
Define la sección electoral:

Distrito -> **Sección** -> Circuito -> Lugar de votación -> Mesa

Parámetros

- **id** (*AutoField*) – Id
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **seccion_politica** (ForeignKey to *SeccionPolitica*) – Seccion politica
- **numero** (*CharField*) – Numero
- **nombre** (*CharField*) – Nombre
- **electores** (*PositiveIntegerField*) – Electores
- **prioridad_hasta_2** (*PositiveIntegerField*) – Prioridad hasta 2
- **prioridad_2_a_10** (*PositiveIntegerField*) – Prioridad 2 a 10
- **prioridad_10_a_100** (*PositiveIntegerField*) – Prioridad 10 a 100
- **cantidad_minima_prioridad_hasta_2** (*PositiveIntegerField*) – Cantidad minima prioridad hasta 2

exception DoesNotExist

exception MultipleObjectsReturned

get_circuitos()
Todos los circuitos de las sección ordenados por número.

class elecciones.models.**SeccionPolitica**(**args*, ***kwargs*)
Define la sección política, que es una agrupación de nuestras secciones electorales, que se usa con fines políticos y para mostrar resultados.

En términos políticos, en especial para la PBA, nuestra Sección es un Municipio (eg, «La Matanza»), y esta sección política es «la tercera sección electoral».

Distrito -> **Sección política** -> Sección -> Circuito -> Lugar de votación -> Mesa

Parámetros

- **id** (*AutoField*) – Id
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **numero** (*PositiveIntegerField*) – Numero
- **nombre** (*CharField*) – Nombre

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**TecnicaProyeccion**(**args*, ***kwargs*)
Representa una estrategia para agrupar circuitos para hacer proyecciones. Contiene una lista de AgrupacionCircuitos que debería en total cubrir a todos los circuitos correspondientes a la categoria que se desea proyectar.

Parámetros

- **id** (*AutoField*) – Id
- **nombre** (*CharField*) – Nombre

exception DoesNotExist

exception MultipleObjectsReturned

class elecciones.models.**VotoMesaReportado**(**args*, ***kwargs*)
Representa una «celda» del acta a cargar, es decir, dada una carga que define mesa y categoria, existe una instancia de este modelo para cada opción y su correspondiente cantidad de votos.

Parámetros

- **id** (*AutoField*) – Id
- **carga** (ForeignKey to *Carga*) – Carga
- **opcion** (ForeignKey to *Opcion*) – Opcion
- **votos** (*PositiveIntegerField*) – Votos

exception DoesNotExist

exception MultipleObjectsReturned

elecciones.models.**actualizar_electores**(*sender*, *instance=None*, *created=False*, ***kwargs*)
Actualiza las denormalizaciones de cantidad de electores a nivel circuito, seccion y distrito cada vez que se crea o actualiza una instancia de mesa.

En general, esto sólo debería ocurrir en la configuración inicial del sistema.

elecciones.models.**canonizar**(*valor*)
Tomado prestado de adjuntos/csv_import, también está en managament/commands Pasa a mayúsculas y elimina espacios. Si se trata de un número, elimina los ceros previos.

class fiscales.models.**CodigoReferido**(*id*, *created*, *modified*, *fiscal*, *codigo*, *activo*)

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal
- **codigo** (*CharField*) – Codigo. Código con el que el fiscal puede referir a otres
- **activo** (*BooleanField*) – Activo

exception DoesNotExist

exception MultipleObjectsReturned

classmethod fiscales_para(*codigo*, *nombre=None*, *apellido=None*)
Devuelve una lista de fiscales candidatos

save(**args*, ***kwargs*)
Genera un código único de 4 dígitos alfanuméricos

class fiscales.models.**Fiscal**(**args*, ***kwargs*)
Representa al usuario «data-entry» del sistema. Es una extensión del modelo `auth.User`

Parámetros

- **id** (*AutoField*) – Id
- **estado** (*StatusField*) – Estado
- **notas** (*TextField*) – Notas. Notas internas, no se muestran
- **codigo_confirmacion** (*UUIDField*) – Codigo confirmacion
- **email_confirmado** (*BooleanField*) – Email confirmado
- **apellido** (*CharField*) – Apellido
- **nombres** (*CharField*) – Nombres
- **tipo_dni** (*CharField*) – Tipo dni
- **dni** (*CharField*) – Dni
- **puntaje_scoring_troll** (*IntegerField*) – Puntaje scoring troll
- **troll** (*BooleanField*) – Troll
- **user** (OneToOneField to `User`) – User
- **seccion** (ForeignKey to *Seccion*) – Seccion
- **session_key** (*CharField*) – Session key
- **last_seen** (*DateTimeField*) – Last seen
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **referente** (ForeignKey to *Fiscal*) – Referente
- **referente_certeza** (*PositiveIntegerField*) – Referente certeza. El código no era exacto?
- **referencia_confirmada** (*BooleanField*) – Referencia confirmada

- **referente_nombres** (*CharField*) – Referente nombres
- **referente_apellido** (*CharField*) – Referente apellido
- **referido_por_codigos** (*CharField*) – Referido por codigos
- **ingreso_alguna_vez** (*BooleanField*) – Ingreso alguna vez
- **asignacion_ultima_tarea** (*DateTimeField*) – Asignacion ultima tarea
- **attachment_asignado** (ForeignKey to *Attachment*) – Attachment asignado
- **mesa_categoria_asignada** (ForeignKey to *MesaCategoria*) – Mesa categoria asignada
- **distrito_afin** (ForeignKey to *Distrito*) – Distrito afin
- **datos_de_contacto** (*GenericRelation*) – Datos de contacto

exception DoesNotExist

exception MultipleObjectsReturned

asignar_attachment(*attachment*)
Asigna al fiscal un attachment para que trabaje en él.

Tiene como prerrequisito que se hayan hecho un llamado previo a limpiar_asignacion_previa() para desasignar la asignación anterior. No se hace aquí para evitar deadlocks (ver #317), se hace desde acciones.py en una transacción independiente.

asignar_mesa_categoria(*mesa_categoria*)
Asigna al fiscal una mesa_categoria para que trabaje en ella.

Tiene como prerrequisito que se hayan hecho un llamado previo a limpiar_asignacion_previa() para desasignar la asignación anterior. No se hace aquí para evitar deadlocks (ver #317), se hace desde acciones.py en una transacción independiente.

classmethod liberar_mesacategorias_y_attachments()
Toma a los fiscales cuya última tarea haya sido asignada más de *settings.TIMEOUT_TAREAS* minutos atrás y: - No se la desasigna para no perder el trabajo que el fiscal puede estar haciendo y "presentará" cuando haga el submit. - Pero sí le baja la cantidad de asignaciones a la mesacategoría y los attachments para que queden postergados por demasiado tiempo.

limpiar_asignacion_previa()
Este método se utiliza para que las mesa-categorías o attachments que tenga asignados el fiscal sean liberados cuando corresponda (por timeout o cuando se asigna uno nuevo).

No se la desasigna para no perder el trabajo que el fiscal puede estar haciendo y "presentará" cuando haga el submit.

marcar_como_troll(*actor*)
Un UE decidió, explícitamente, marcarme como troll

marcar_ingreso_alguna_vez()
Marca que el Fiscal efectivamente ya ingresó alguna vez. En principio la marca de ingreso_alguna_vez se usa para capacitar al validador.

quitar_marca_troll(*actor*, *nuevo_scoring*)
Un UE decidió, explícitamente, indicar que no soy troll

ultimo_codigo()
devuelve el último código activo

ultimo_codigo_url()
devuelve la url absoluta con último código activo

fiscales.models.**crear_user_y_codigo_para_fiscal**(*sender*, *instance=None*, *created=False*, *update_fields=None*, ***kwargs*)

Cuando se crea o actualiza una instancia de `Fiscal` en estado confirmado que no tiene usuario asociado, automáticamente se crea uno `auth.User` utilizando el DNI como *username*.

class adjuntos.models.**Attachment**(**args*, ***kwargs*)

Guarda las fotos de ACTAS y otros documentos fuente desde los cuales se cargan los datos. Están asociados a una imágen que a su vez puede tener una versión editada.

Los attachments están asociados a mesas una vez que se clasifican.

No pueden existir dos instancias de este modelo con la misma foto, dado que el atributo digest es único.

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **status** (*StatusField*) – Status
- **mesa** (ForeignKey to *Mesa*) – Mesa
- **email** (ForeignKey to *Email*) – Email
- **mimetype** (*CharField*) – Mimetype
- **parent** (ForeignKey to *Attachment*) – Parent
- **foto** (*VersatileImageField*) – Foto
- **foto_edited** (*VersatileImageField*) – Foto edited
- **foto_digest** (*CharField*) – Foto digest
- **height** (*PositiveIntegerField*) – Image height
- **width** (*PositiveIntegerField*) – Image width
- **subido_por** (ForeignKey to *Fiscal*) – Subido por
- **identificacion_testigo** (ForeignKey to *Identificacion*) – Identificacion testigo
- **pre_identificacion** (ForeignKey to *PreIdentificacion*) – Pre identificacion
- **cant_fiscales_asignados** (*PositiveIntegerField*) – Cant fiscales asignados
- **cant_asignaciones_realizadas** (*PositiveIntegerField*) – Cant asignaciones realizadas

exception DoesNotExist

exception MultipleObjectsReturned

crear_pre_identificacion_si_corresponde()

Le asocia al attachment una PreIdentificacion con los datos del fiscal que la subió si no hay una previa.

save(**args*, ***kwargs*)

Actualiza el hash de la imágen original asociada antes de guardar. Notar que esto puede puede producir una excepción si la imágen (el digest) ya es conocido en el sistema.

Además, crea una PreIdentificacion con los datos del Fiscal que lo subió si no hay una.

status_count(*estado*)

A partir del conjunto de identificaciones del attachment que tienen el estado parámetro devuelve una lista de tuplas (mesa_id, cantidad, cantidad que viene de csv). Sólo cuenta las no invalidadas.

Cuando status == "problema" el id de mesa es None

Por ejemplo (cuando estado == "identificada"):

[(3, 2, 0), (4, 1, 1),

]

Hay 2 identificaciones para la mesa id==3 y 1 para la id==4, pero ésa tiene una identificación por CSV.

class adjuntos.models.**AttachmentQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

priorizadas()

Ordena por cantidad de fiscales trabajando en el momento, luego por cantidad de personas que alguna vez trabajaron, y por último por orden de llegada.

redondear_cant_fiscales_asignados_y_de_asignaciones()

Redondea la cantidad de fiscales asignados y de asignaciones a múltiplos de `settings.MIN_COINCIDENCIAS_IDENTIFICACION` para que al asignar mesas no se pospongan indefinidamente mesas que fueron entregadas ya a algún fiscal.

sin_identificar(*fiscal_a_excluir=None*, *for_update=True*)

Devuelve un conjunto de Attachments que no tienen identificación consolidada.

Se excluyen attachments que ya hayan sido clasificados por *fiscal_a_excluir*

class adjuntos.models.**CSVTareaDeImportacion**(*id*, *created*, *modified*, *csv_file*, *status*, *errores*, *fiscal*, *mesas_total_ok*, *mesas_parc_ok*)

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **csv_file** (*FileField*) – Csv file
- **status** (*StatusField*) – Status
- **errores** (*TextField*) – Errores
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal
- **mesas_total_ok** (*PositiveIntegerField*) – Mesas total ok
- **mesas_parc_ok** (*PositiveIntegerField*) – Mesas parc ok

exception DoesNotExist

exception MultipleObjectsReturned

class adjuntos.models.**Email**(**args*, ***kwargs*)

Almacena la información de emails que entran al sistema y contienen attachments La persistencia de estos objetos no es estrictamente necesaria.

Ver `elecciones.management.commands.importar_actas`

Parámetros

- **id** (*AutoField*) – Id

- **date** (*CharField*) – Date
- **from_address** (*CharField*) – From address
- **body** (*TextField*) – Body
- **title** (*CharField*) – Title
- **uid** (*PositiveIntegerField*) – Uid
- **message_id** (*CharField*) – Message id

exception DoesNotExist

exception MultipleObjectsReturned

class adjuntos.models.**Identificacion**(**args*, ***kwargs*)

Es el modelo que guarda clasificaciones de actas para asociarlas a mesas

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **status** (*StatusField*) – Status
- **source** (*StatusField*) – Source
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal
- **mesa** (ForeignKey to *Mesa*) – Mesa
- **attachment** (ForeignKey to *Attachment*) – Attachment
- **tomada_por_consolidador** (*DateTimeField*) – Tomada por consolidador
- **procesada** (*BooleanField*) – Procesada
- **invalidada** (*BooleanField*) – Invalidada

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*)

Si el fiscal es troll, la identificación nace invalidada y ya procesada.

class adjuntos.models.**PreIdentificacion**(**args*, ***kwargs*)

Este modelo se usa para asociar a los attachment información de identificación que no es completa. No confundir con Identificacion ni con el status de identificación de una mesa.

Parámetros

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Creado
- **modified** (*AutoLastModifiedField*) – Modificado
- **fiscal** (ForeignKey to *Fiscal*) – Fiscal
- **distrito** (ForeignKey to *Distrito*) – Distrito
- **seccion** (ForeignKey to *Seccion*) – Seccion
- **circuito** (ForeignKey to *Circuito*) – Circuito

exception DoesNotExist

exception MultipleObjectsReturned

adjuntos.models.**hash_file**(*file*, *block_size=65536*)

Dado un objeto file-like (en modo binario), devuelve un hash digest único de 128 digitos hexadecimales

Utiliza el algoritmo de hashing blake2

```
>>> hash_file(open('messi.jpg', 'rb'))

↪'90554e1d519e0fc665fab042d7499a1bc9c191f2a13b0b2c369753dcb23b181866cb116007fc37a445421270e0491
↪'
```

CAPÍTULO 11

Tests unitarios

El proyecto utiliza pytest via pytest-django para escribir y ejecutar pruebas unitarias.

Para corer los tests usando Docker

```
$ make test
```

Alternativamente, si usás un entorno local (o desde el shell del contenedor de la app, `make shell-app`) podes correr

```
$ pytest
```

Podés correr un test en particular

```
$ pytest path_to/test_file.py::test_a_correr
```

O más en general

```
$ pytest -k test_a_correr
```

El test runner de pytest tiene muchas opciones útiles. Ver `--sw`, `--lf`, `-s` y más.

Tené en cuenta que por defecto la base de datos de tests se reusa (setting `--reuse-db`, definido en `pytest.ini`) Eventualmente, la base puede divergir y los tests fallan porque faltan o sobran columnas.

Para reconstruir la base al estado actual, ejecutar

```
$ pytest --create-db
```

Hay un threshold de **cobertura de tests** mínimo definido en `pytest.ini` con el parámetro `--cov-fail-under`. Esto significa que cualquier código que se agrega debe incoporar las pruebas suficientes para no bajar de este nivel de cobertura.

No se aceptarán integraciones de cambios que modifiquen este cambio a menos, salvo una muy justificada explicación.

Si en cambio, tu branch aumenta la cobertura (porque hiciste tests que faltaban o borraste código muerto) por favor, aumentá ese valor al entero menor más próximo de la cobertura que lograste. ¡Gracias!

Si necesitás saber exáctamente qué código del proyecto falta cubrir (especialmente en las partes donde realizaste modificaciones), podés obtener un reporte navegable donde verás el código coloreado con distinto fondo las líneas ejecutadas y las que no.

```
$ pytest --cov-report=html
$ sensible-browser htmlcov/index.html
```

11.1 Pruebas de integación «end to end»

También hay pruebas «end to end» basadas en Cypress

Primero hay que levantar la app (Ver `Instalación <./INSTALL.md>`).

Para correr cypress c/ide:

```
make test-e2e
```

Para correr cypress headless:

```
make test-e2e-headless
```

CAPÍTULO 12

Escribir documentación

Le damos total relevancia a la documentación de Escrutinio Social. Es un sistema complejo y requiere detalles tanto para su uso, configuración, administración en condiciones de «misión crítica» y desarrollo.

Utilizamos el software Sphinx y podés escribir nuevos capitulos tanto en markdown como en restructuredText. Para eso creá tu archivo en la carpeta `docs/` y declaralo en alguna directiva `..toctree::`, por ejemplo en los indices principales de `docs/index.rst`

Usamos el servicio Read the Docs para compilar automaticamente y tener al día la documentación en https://escrutinio-social.readthedocs.io/

En cada *pull request* genera una compilación de la documentación sobre ese branch para cersiorarse de que no hay errores antes de mergear. Por favor, revisá el resultado, especialmente si cambiaste algo.

También podes compilar la documentación localmente

```
$ pip install -r requirements/docs.txt
$ cd docs/
$ make html
$ xdg-open _build/html/index.html
```

12.1 Diagrama de modelos

En cada compilación, se incluye *un diagrama* de la definición de los modelos principales. Esto se realiza ejecutando `make update-models-diagram` al inicio de la compilación, que utiliza a su vez un comando de django-extensions para crear la imágen inspeccionando el código.

CAPÍTULO 13

Glosario

voto nulo Un voto nulo es un voto no positivo que . . .

voto no positivo: Es todo aquel voto que no suma a una fuerza política particular.

CAPÍTULO 14

Indices y tablas

- genindex
- modindex
- search

Índice de Módulos Python

Índice

P

Q

R

S

T

U

V