RCVis

Armin Samii

Mar 30, 2024

CONTENTS:

CHAPTER

ONE

RCVIS.COM

This is the code repository for rcvis.com. Unless you're a programmer, you probably want to be there instead of here!

RCVis has been used to visualize hundreds of polls and dozens of elections. It has been used as part of the official reporting of tabulation data in Colorado and Utah, as well as unofficially in New York City, Minnesota, Maine, California, and more. It is connected to both RankedChoices.com and RankedVote.co. It can import data from Opavote, ElectionBuddy, Dominion software, RCTab, and more. It was featured on a Ballotpedia page during the 2020 NYC election cycle, as well as in the Gothamist, NBC New York, Fox 5 NY, JD Forward, among others.

You may fork and run this code locally, though you can probably get what you want by just uploading your data directly to RCVis.

CHAPTER

TWO

RANKED CHOICE VOTING VISUALIZATION

Visualize the results of ranked-choice voting elections.

What is RCV? RCV allows you to have backup options. If your preferred candidate can't win, you still have a say: your vote gets *transferred* to your next-best pick.

Why RCV? In an RCV election, you can't spoil votes. Third-party candidates don't waste votes. Similar candidates help each other instead of hurting each other. They're less polarized and more fair. Multi-winner RCV elections mitigate the effects of gerrymandering. For more information, check out FairVote's guide to the benefits of RCV elections.

Why the visualizer? In a traditional election, the results are easy to understand: how many votes did each candidate get? In an RCV election, it can be a harder to understand how a candidate won, based on what happens in each round. Our goal is to create a series of visualizations which can work for a variety of audiences on a variety of mediums: print, web, and TV.

Learn more on our Medium post: An Illustrated Guide to Ranked-Choice Voting.

2.1 Installation

Install `python3`, `virtualenv`, and `npm` with your favorite package manager, then run `./scripts/install.sh`.

Create a .env file with your secrets and configuration options:

```
export RCVIS_SECRET_KEY=''
export RCVIS_DEBUG=True
export RCVIS_HOST=localhost

# Either have OFFLINE_MODE=True
export OFFLINE_MODE=True

# The following fields are optional, though you will
# have limited functionality without them.

# Or set up an AWS bucket and enter your credentials
# export OFFLINE_MODE=False
# export AWS_STORAGE_BUCKET_NAME=''
# export AWS_S3_REGION_NAME=''
# export AWS_ACCESS_KEY_ID=''
# export AWS_SECRET_ACCESS_KEY=''

# To send registration emails when OFFLINE_MODE is False:
# export SENDGRID_USERNAME=''
```

(continues on next page)

(continued from previous page)

```
# export SENDGRID_PASSWORD=''

# To clear cloudflare cache when models update:
# export CLOUDFLARE_ZONE_ID=''
# export CLOUDFLARE_AUTH_TOKEN=''

# To run the SauceLabs integration tests, you will need
export SAUCE_USERNAME=''
export SAUCE_ACCESS_KEY=''

# To generate videos (and to run movie tests), you will need:
export IMAGEIO_FFMPEG_EXE='/usr/bin/ffmpeg'
export SQS_QUEUE_NAME='default-queue'
# export MOVIE_FONT_NAME="Roboto"
# export AWS_POLLY_STORAGE_BUCKET_NAME="bucket-name-on-s3"

# To subscribe users to mailchimp upon registration, you need:
# export MAILCHIMP_API_KEY=''
# export MAILCHIMP_LIST_ID=''
# export MAILCHIMP_DC=''

# If you are updating a template, you'll need to clear the cache every time or set:
# export DISABLE_CACHE=True
```

To get moviepy working for Ubuntu 16.04 LTS users, comment out the following statement in `/etc/ImageMagick-6/policy.xml`:

```
<policy domain="path" rights="none" pattern="@*"/>
```

or, simply run `sudo ./scripts/fix-moviepy-on-ubuntu-1604.sh`

2.2 Running

To begin serving the website at localhost:8000:

```
./scripts/serve.sh
```

You may also need to run this whenever the npm dependencies change:

```
source .env
source venv/bin/activate

npm install  # this works for me
python3 manage.py npminstall  # this is purported to work but doesn't
```

To run workers to generate movies (optional - only needed to use the movie generation flow):

```
source .env
source venv/bin/activate
export DISPLAY=":0" # if not already set
celery -A rcvis worker --loglevel info
```

2.3 Examples

Check out rcvis.com for live examples, including:

2.4 REST API

The primary API documentation is in the form of example code, which is documented line-by-line. We recommend you start by looking over the example code. Addition documentation is available at rcvis.com/api/.

To get started with programmatic access to rcvis:

1. Create an account on RCVis
2. Email team@rcvis.com to enable API access
3. Submit a POST request to https://www.rcvis.com/api/auth/get-token to obtain an API Key, e.g. `POST https://www.rcvis.com/api/auth/get-token {"username": "username", "password": "password"}`.

With your API key, you may access two endpoints:

1. https://www.rcvis.com/api/visualizations/ requires field `jsonFile` with the body of a valid summary JSON.
2. https://www.rcvis.com/api/bp/ requires field `resultsSummaryFile` with the body of a valid summary JSON and allows four optional fields: `candidateSidecarFile`, `dataSourceURL` (string), `areResultsCertified` (boolean), and `isPrimary` (boolean).

For both endpoints, upload with POST and modify with PUT or PATCH. Authenticated users are limited to 1000 requests per hour.

2.5 oembed

RCVis implements the oembed protocol with discoverability, allowing you to embed files into your website with an iframe.

CHAPTER

THREE

MODULES

3.1 Django Visualizer

3.1.1 Subpackages

Graph Loading & Creation

Colors

Graph

Data that holds the entire Graph, as well as utilities for parsers to interactively build the graph. You probably don't want to use this directly, but instead, want to use the GraphSummary which is more user-friendly. To get the summary, use graph.summarize().

class visualizer.graph.graph.**Graph**(*title*)

Bases: object

Data about the entire graph, including nodes and links between thhem

create_graph_from_rounds(*rounds*)

Generates a graph with nodes and edges, where the nodes are a single Item at a specific Round, and the edges are Transfers

create_node(*item*, *count*, *round_i*)

Creates a node with the given count. Only meaningful while graph creation is in progress.

get_items_for_names(*listOfNames*)

Given a list of all names, returns the corresponding Item for each naem

property items

Returns all items present in this graph

property numRounds

Returns the number of rounds

set_date(*date*)

Sets the date of this election

set_elimination_order(*orderedItems*)

Given a list of Items, sets the elimination erder. Does no validation that the given order is complete, but will likely throw several errors here or elsewhere if you pass bad data.

set_threshold(*threshold*)

Sets the threshold for this election

summarize()

Returns the graph summary - or creates it if it hasn't been requested yet

class visualizer.graph.graph.**LinkData**(*source*, *target*, *value*)

Bases: object

Data about a single "link": a transfer from the source to target

class visualizer.graph.graph.**NodeData**(*item*, *label*, *count*, *roundNum*)

Bases: object

Data about a single "node": a candidate in a single round

mark_eliminated()

Marks the given node as the node in which this candidate was eliminated

mark_winner()

Marks the given node as the node in which this candidate won

GraphCreator

Data that holds the entire Graph, as well as utilities for parsers to interactively build the graph. You probably don't want to use this directly, but instead, want to use the GraphSummary which is more user-friendly. To get the summary, use graph.summarize().

class visualizer.graph.graph.**Graph**(*title*)

Bases: object

Data about the entire graph, including nodes and links between thhem

create_graph_from_rounds(*rounds*)

Generates a graph with nodes and edges, where the nodes are a single Item at a specific Round, and the edges are Transfers

create_node(*item*, *count*, *round_i*)

Creates a node with the given count. Only meaningful while graph creation is in progress.

get_items_for_names(*listOfNames*)

Given a list of all names, returns the corresponding Item for each naem

property items

Returns all items present in this graph

property numRounds

Returns the number of rounds

set_date(*date*)

Sets the date of this election

set_elimination_order(*orderedItems*)

Given a list of Items, sets the elimination erder. Does no validation that the given order is complete, but will likely throw several errors here or elsewhere if you pass bad data.

set_threshold(*threshold*)

Sets the threshold for this election

summarize()

Returns the graph summary - or creates it if it hasn't been requested yet

class visualizer.graph.graph.**LinkData**(*source*, *target*, *value*)

Bases: object

Data about a single "link": a transfer from the source to target

class visualizer.graph.graph.**NodeData**(*item*, *label*, *count*, *roundNum*)

Bases: object

Data about a single "node": a candidate in a single round

mark_eliminated()

Marks the given node as the node in which this candidate was eliminated

mark_winner()

Marks the given node as the node in which this candidate won

GraphSummary

Summarize the graph to provide helper functions to different visualizers

class visualizer.graph.graphSummary.**CandidateInfo**(*name*)

Bases: object

Summarizes a single candidate over each round

add_votes(*amount*)

Adds the given votes to the current round

class visualizer.graph.graphSummary.**GraphSummary**(*graph*)

Bases: object

A class which organizes a Graph into data that makes it easier to visualize

candidates: dict

linksByTargetNode: dict

numEliminated: int

numWinners: int

rounds: list

winners: list

class visualizer.graph.graphSummary.**RoundInfo**(*round_i*)

Bases: object

Summarizes a single round, with functions to build the round

add_eliminated(*item*)

Adds the name to the list of names eliminated this round

add_votes(*candidateItem*, *numVotes*)

Notes that the given Candidate received numVotes votes - unless they're not an "active" candidate.

add_winner(*item*)

Adds the name to the list of names elected this round

key()

Returns the "key" for this round (just the round number)

RCV Result

Helper methods to generate a Graph

class visualizer.graph.rcvResult.**Elimination**(*item*, *transfersByItem*)

Bases: *Transfer*

Syntactic sugar for an Elimination, which is kind of like a transfer

class visualizer.graph.rcvResult.**Item**(*name*)

Bases: object

A single Item, also known as a Candidate elsewhere in the Code.

class visualizer.graph.rcvResult.**Round**

Bases: object

A single Round, with data about who won and where votes were transferred

class visualizer.graph.rcvResult.**Transfer**(*item*, *transfersByItem*)

Bases: object

Transfers is a mapping from Item objects to a number of transferred votes.

class visualizer.graph.rcvResult.**WinTransfer**(*item*, *transfersByItem*)

Bases: *Transfer*

Syntactic sugar for a Win, which is kind of like a transfer

Read RCVRC JSON

Class which reads an RCVRC-formatted JSON file

class visualizer.graph.readRCVRCJSON.**FixIgnoreResidualSurplus**(*jsonData*)

Bases: *JSONMigrateTask*

Creates a "residual surplus" candidate in the first round if we find it in other rounds, since we look to the first round for all candidates (or places votes can be transferred)

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixNoTransfersTask**(*jsonData*)

Bases: *JSONMigrateTask*

The JSON prefers no key named "transfers" instead of an empty list. We do not.

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixRankitCombinedTallyResults**(*jsonData*)

Bases: *JSONMigrateTask*

Rankit includes eliminations and elected on the same tallyResult

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixRankitMissingTransfers**(*jsonData*)

Bases: *JSONMigrateTask*

Rankit often forgets to eliminate candidates, they just drop them

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixRankitMissingWinners**(*jsonData*)

Bases: *JSONMigrateTask*

Rankit stops including Winner in tally after they win

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixRankitNoElimOnLastRound**(*jsonData*)

Bases: *JSONMigrateTask*

Rankit incorrectly eliminates on the last round

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**FixUndeclaredUWITask**(*jsonData*)

Bases: *JSONMigrateTask*

Undeclared votes are sometimes marked as 'UWI' instead of 'Undeclared'

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**HideDecimalsTask**(*jsonData*)

Bases: *JSONMigrateTask*

If the config desired it - remove all decimal places

do()

Run the migration

class visualizer.graph.readRCVRCJSON.**JSONMigrateTask**(*jsonData*)

Bases: `object`

An abstract base class to "fix" JSONs. Each migration "task" should override this.

abstract do()

Run the migration. Put your subclass' logic here.

is_rankit_data()

Is the jsonData from RankIt?

rename(*fromStr*, *toStr*)

A helper function to rename a candidate s/fromStr/toStr throughout the JSON Returns false if toStr already existed in the results – will refuse to overwrite

class visualizer.graph.readRCVRCJSON.**JSONReader**(*data*)

Bases: object

The class which reads the JSON and performs migrations

self.graph is a Graph object which is partially initialized (TODO how partially?) self.rounds is a list of Round objects self.items is a list of Item objects

eliminationOrder: list

get_elimination_order()

Returns the elimination order: a list of names in the order in which they were eliminated

get_graph()

Returns the Graph object

get_rounds()

Returns the list of rounds

graph: object

items: list

parse_data(*data*)

Parses the JSON data, or raises an exception on failure

rounds: list

set_elimination_order(*rounds*, *items*)

Sets the elimination order given each round and a list of Items

class visualizer.graph.readRCVRCJSON.**MakeExhaustedAndSurplusACandidate**(*jsonData*)

Bases: *JSONMigrateTask*

If there are "exhausted" ballots, make them a first-class citizen candidate

do()

Run the migration, ensuring they are not already marked as candidates

class visualizer.graph.readRCVRCJSON.**MakeTalliesANumber**(*jsonData*)

Bases: *JSONMigrateTask*

Converts tally strings to numbers

do()

Run the migration

exception visualizer.graph.readRCVRCJSON.**MigrationError**

Bases: Exception

An error triggered during migration, with a message to be passed on to the user

class visualizer.graph.readRCVRCJSON.**NormalizeSpecialNames**(*jsonData*)

Bases: *JSONMigrateTask*

normalize names to their canonical, indexable names

do()

Run the migration

3.1.2 Admin

Admin page configuration

class visualizer.admin.**HomepageFeaturedElectionAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

Administer homepage featured links

list_display = ('title', 'order', 'column', 'jsonConfig')

property media

raw_id_fields = ('jsonConfig',)

class visualizer.admin.**HomepageFeaturedElectionColumnAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

Administer homepage featured link columns

list_display = ('title', 'order')

property media

class visualizer.admin.**JsonAdmin**(*model*, *admin_site*)

Bases: CursorPaginatorAdmin

The admin page to modify JsonConfig

actions = [<function make_movie>]

list_display = ('slug', 'title', 'owner', 'numRounds', 'numCandidates', 'uploadedAt', 'movieGenerationStatus')

property media

readonly_fields = ('slug', 'title', 'numRounds', 'numCandidates', 'uploadedAt', 'movieGenerationStatus')

search_fields = ['slug']

show_full_result_count = False

view_on_site = True

visualizer.admin.**make_movie**(*_*, *request*, *queryset*)

An action from the admin menu that creates a movie

3.1.3 Common

A set of common utilities to be used across modules

visualizer.common.**candidate_renames**()

A dictionary mapping how we should rename candidate names

visualizer.common.**get_host**(*request*)

Returns the HTTP_HOST, split for easy mocking

visualizer.common.**intify**(*notint*)

Turn into int if it's a round number

visualizer.common.**make_complete_url**(*request*, *urlWithoutDomain*)

Returns the complete URL, including domain

visualizer.common.**percentify**(*numerator*, *denominator*)

Turn a float into a percentage string. Or, if the denominator is zero, returns an empty string

3.1.4 Forms

Forms enable users to create Models

class visualizer.forms.**UploadByDataTableForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *UploadForm*

Used by the upload form using the DataTables entry

class Meta

Bases: *Meta*

Metadata is all we need here

fields = ['candidateSidecarFile', 'rotateNames', 'showRoundNumbersOnSankey', 'onlyShowWinnersTabular', 'doUseDescriptionInsteadOfTimeline', 'isPreferentialBlock', 'hideSankey', 'hideTabular', 'doDimPrevRoundColors', 'excludeFinalWinnerAndEliminatedCandidate', 'hideDecimals', 'colorTheme', 'eliminationBarColor', 'dataSourceURL', 'areResultsCertified', 'textForWinner']

base_fields = {'areResultsCertified': <django.forms.fields.BooleanField object>, 'candidateSidecarFile': <django.forms.fields.FileField object>, 'colorTheme': <django.forms.fields.IntegerField object>, 'dataSourceURL': <django.forms.fields.URLField object>, 'doDimPrevRoundColors': <django.forms.fields.BooleanField object>, 'doUseDescriptionInsteadOfTimeline': <django.forms.fields.BooleanField object>, 'eliminationBarColor': <django.forms.fields.IntegerField object>, 'excludeFinalWinnerAndEliminatedCandidate': <django.forms.fields.BooleanField object>, 'hideDecimals': <django.forms.fields.BooleanField object>, 'hideSankey': <django.forms.fields.BooleanField object>, 'hideTabular': <django.forms.fields.BooleanField object>, 'isPreferentialBlock': <django.forms.fields.BooleanField object>, 'jsonFile': <django.forms.fields.FileField object>, 'onlyShowWinnersTabular': <django.forms.fields.BooleanField object>, 'rotateNames': <django.forms.fields.BooleanField object>, 'showRoundNumbersOnSankey': <django.forms.fields.BooleanField object>, 'textForWinner': <django.forms.fields.IntegerField object>}

clean_jsonFile()

Converts the datatables json to URCVT json

declared_fields = {'colorTheme': <django.forms.fields.IntegerField object>, 'eliminationBarColor': <django.forms.fields.IntegerField object>, 'jsonFile': <django.forms.fields.FileField object>, 'textForWinner': <django.forms.fields.IntegerField object>}

property media

Return all media required to render the widgets on this form.

class visualizer.forms.**UploadForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: ModelForm

Used by the upload form

class Meta

Bases: object

Metadata is all we need here

fields = ['jsonFile', 'candidateSidecarFile', 'rotateNames', 'showRoundNumbersOnSankey', 'onlyShowWinnersTabular', 'doUseDescriptionInsteadOfTimeline', 'isPreferentialBlock', 'hideSankey', 'hideTabular', 'doDimPrevRoundColors', 'excludeFinalWinnerAndEliminatedCandidate', 'hideDecimals', 'colorTheme', 'eliminationBarColor', 'dataSourceURL', 'areResultsCertified', 'textForWinner']

model

alias of *JsonConfig*

base_fields = {'areResultsCertified': <django.forms.fields.BooleanField object>, 'candidateSidecarFile': <django.forms.fields.FileField object>, 'colorTheme': <django.forms.fields.IntegerField object>, 'dataSourceURL': <django.forms.fields.URLField object>, 'doDimPrevRoundColors': <django.forms.fields.BooleanField object>, 'doUseDescriptionInsteadOfTimeline': <django.forms.fields.BooleanField object>, 'eliminationBarColor': <django.forms.fields.IntegerField object>, 'excludeFinalWinnerAndEliminatedCandidate': <django.forms.fields.BooleanField object>, 'hideDecimals': <django.forms.fields.BooleanField object>, 'hideSankey': <django.forms.fields.BooleanField object>, 'hideTabular': <django.forms.fields.BooleanField object>, 'isPreferentialBlock': <django.forms.fields.BooleanField object>, 'jsonFile': <django.forms.fields.FileField object>, 'onlyShowWinnersTabular': <django.forms.fields.BooleanField object>, 'rotateNames': <django.forms.fields.BooleanField object>, 'showRoundNumbersOnSankey': <django.forms.fields.BooleanField object>, 'textForWinner': <django.forms.fields.IntegerField object>}

declared_fields = {'colorTheme': <django.forms.fields.IntegerField object>, 'eliminationBarColor': <django.forms.fields.IntegerField object>, 'textForWinner': <django.forms.fields.IntegerField object>}

property media

Return all media required to render the widgets on this form.

3.1.5 JS Utils

A helper function for javascript-generating python

visualizer.jsUtils.**approx_length**(*stringToMeasure*)

c/o https://stackoverflow.com/a/16008023/1057105 - measure the approximate pixels of the given string

3.1.6 Models

The django object models

class visualizer.models.**ColorTheme**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntegerChoices

Describes the status of movie generation for this model

ALTERNATING = 2

PURPLE_TO_ORANGE = 1

RAINBOW = 0

class visualizer.models.**EliminationBarColor**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntegerChoices

Describes the status of movie generation for this model

GRAY = 0

HIDDEN = 1

LAST_ROUND_COLOR = 2

class visualizer.models.**HomepageFeaturedElection**(**args*, ***kwargs*)

Bases: Model

Represents a single link on the homepage list of featured elections.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

column

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

column_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

jsonConfig

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

jsonConfig_id

objects = <django.db.models.manager.Manager object>

order

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class visualizer.models.**HomepageFeaturedElectionColumn**(**args*, ***kwargs*)

Bases: Model

Represents a column of links on the homepage.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

links_in_column

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

objects = <django.db.models.manager.Manager object>

order

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class visualizer.models.**JsonConfig**(**args*, ***kwargs*)

Bases: Model

A Json file representing a single election, and its configuration

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

areResultsCertified

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

candidateSidecarFile

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

colorTheme

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

dataSourceURL

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

detail_views = ('visualizer.views.Visualize',)

doDimPrevRoundColors

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

doUseDescriptionInsteadOfTimeline

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

electionpage_set

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

eliminationBarColor

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

excludeFinalWinnerAndEliminatedCandidate

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_absolute_url()

Used in the admin panel to have a "Visit Site" link

classmethod get_all_non_auto_fields()

All editable fields of JsonConfig - must be kept up to date with the list of fields above. (I'm sure there's a way to do this automatically…)

classmethod get_all_public()

Returns users whose data can be included in sitemap & index

get_colorTheme_display(*, *field=<django.db.models.fields.IntegerField: colorTheme>*)

get_eliminationBarColor_display(*, *field=<django.db.models.fields.IntegerField: eliminationBarColor>*)

get_movieGenerationStatus_display(*, *field=<django.db.models.fields.IntegerField: movieGenerationStatus>*)

get_next_by_uploadedAt(*, *field=<django.db.models.fields.DateTimeField: uploadedAt>*, *is_next=True*, ***kwargs*)

get_previous_by_uploadedAt(*, *field=<django.db.models.fields.DateTimeField: uploadedAt>*, *is_next=False*, ***kwargs*)

get_textForWinner_display(*, *field=<django.db.models.fields.IntegerField: textForWinner>*)

hideDecimals

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

hideSankey

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

hideTabular

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

isPreferentialBlock

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

jsonFile

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

movieGenerationStatus

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

movieHorizontal

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Restaurant.place is a ForwardOneToOneDescriptor instance.

movieHorizontal_id

movieVertical

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Restaurant.place is a ForwardOneToOneDescriptor instance.

movieVertical_id

multiscraper

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

numCandidates

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

numRounds

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

onlyShowWinnersTabular

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

owner

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

owner_id

rawDownloadedBy

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

rotateNames

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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.

scraper

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

showRoundNumbersOnSankey

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

slug

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

textForWinner

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

uploadedAt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class visualizer.models.**MovieGenerationStatuses**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntegerChoices`

Describes the status of movie generation for this model

COMPLETE = 4

FAILED = 5

LANDSCAPE_COMPLETE = 3

NOT_REQUESTED = 0

NOT_STARTED = 1

PICKED_UP_BY_TASK = 2

class visualizer.models.**TextForWinner**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntegerChoices

Describes the status of movie generation for this model

ELECTED = 0

LEAD = 3

PRIMARY = 2

WON = 1

3.1.7 Template Tags

Helper functions for reversing and getting absolute URLs, including the domain

visualizer.templatetags.reverse.**get_as_complete_url**(*context*, *url*)

Takes a URL and returns the full URL, including domain, if it's not already

visualizer.templatetags.reverse.**get_reverse_as_complete_url**(*context*, *name*, **args*)

Runs reverse and then returns the full URL, including domain

Tag to get a share icon, to be used in tandem with django_social_share

visualizer.templatetags.share_icon.**get_icon**(*shareTo*)

Pass in the text label. Will return an image from /share_icons/ of the same name.

3.1.8 Tests

3.1.9 URLs

URLs for the primary upload and visualization app

3.1.10 Views

The django views file

class visualizer.views.**BallotpediaViewSet**(**args*, ***kwargs*)

Bases: LoggingMixin, ModelViewSet

API endpoint with all ballotpedia fields

basename = None

description = None

detail = None

name = None

perform_create(*serializer*)

permission_classes = [<class 'accounts.permissions.HasAPIAccess'>, <class 'accounts.permissions.IsOwnerOrReadOnly'>]

queryset

serializer_class

alias of *BallotpediaSerializer*

suffix = None

class visualizer.views.**DownloadRawData**(***kwargs*)

Bases: LoginRequiredMixin, DetailView

Download raw data - don't just share the presigned AWS URL, we want a fresh URL for each download.

get_context_data(***kwargs*)

Insert the single object into the context dict.

login_url = 'login'

model

alias of *JsonConfig*

redirect_field_name = 'redirect_to'

template_name = 'visualizer/rawdata.html'

class visualizer.views.**Index**(***kwargs*)

Bases: TemplateView

The homepage

build_path = 'index.html'

get_context_data(***kwargs*)

template_name = 'visualizer/index.html'

class visualizer.views.**JsonOnlyViewSet**(**args*, ***kwargs*)

Bases: LoggingMixin, ModelViewSet

API endpoint that allows tabulated JSONs to be viewed or edited.

basename = None

description = None

detail = None

name = None

perform_create(*serializer*)

permission_classes = [<class 'accounts.permissions.HasAPIAccess'>, <class 'accounts.permissions.IsOwnerOrReadOnly'>]

queryset

serializer_class
alias of *JsonOnlySerializer*

suffix = None

class visualizer.views.**Oembed**(***kwargs*)
Bases: View

The oembed protocol, pointing to VisualizeEmbedded

dispatch(*request*, **args*, ***kwargs*)

get(*request*)
Overriding the getter for this class-based view

class visualizer.views.**Upload**(***kwargs*)
Bases: LoginRequiredMixin, CreateView

The upload page

build_path = 'upload.html'

form_class
alias of *UploadForm*

form_invalid(*form*)
If the form is invalid, render the invalid form.

form_valid(*form*)
If the form is valid, save the associated model.

include = ['jsonFile', 'candidateSidecarFile', 'rotateNames', 'showRoundNumbersOnSankey', 'onlyShowWinnersTabular', 'doUseDescriptionInsteadOfTimeline', 'isPreferentialBlock', 'hideSankey', 'hideTabular', 'doDimPrevRoundColors', 'excludeFinalWinnerAndEliminatedCandidate', 'hideDecimals', 'colorTheme', 'eliminationBarColor', 'dataSourceURL', 'areResultsCertified', 'textForWinner']

login_url = 'login'

model
alias of *JsonConfig*

redirect_field_name = 'redirect_to'

success_url = 'v/{slug}'

template_name = 'visualizer/uploadFile.html'

class visualizer.views.**UploadByDataTable**(***kwargs*)

Bases: *Upload*

Upload form when using the datatables input

form_class

alias of *UploadByDataTableForm*

class visualizer.views.**UserViewSet**(**args*, ***kwargs*)

Bases: LoggingMixin, ReadOnlyModelViewSet

API endpoint that allows you to view but not edit Users.

basename = None

description = None

detail = None

name = None

permission_classes = [<class 'rest_framework.permissions.IsAdminUser'>]

queryset

serializer_class

alias of *UserSerializer*

suffix = None

class visualizer.views.**ValidateDataEntry**(***kwargs*)

Bases: LoginRequiredMixin, View

Validation AJAX view: would the current input succeed in creating a graph?

post(*request*)

Doesn't render a webpage - just text

class visualizer.views.**Visualize**(***kwargs*)

Bases: DetailView

Visualizing a single JsonConfig

get(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)

Insert the single object into the context dict.

model

alias of *JsonConfig*

template_name = 'visualizer/visualize.html'

class visualizer.views.**VisualizeBallotpedia**(***kwargs*)

Bases: DetailView

The embedded ballotpedia visualization

dispatch(*request*, **args*, ***kwargs*)

get(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *JsonConfig*

template_name = 'visualizer/visualize-ballotpedia.html'

class visualizer.views.**VisualizeEmbedded**(***kwargs*)
Bases: DetailView

The embedded visualization, to be used in an iframe.

dispatch(*request*, **args*, ***kwargs*)

get(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *JsonConfig*

template_name = 'visualizer/visualize-embedded.html'

class visualizer.views.**VisualizeEmbedly**(***kwargs*)
Bases: RedirectView

VisualizeEmbedded, but without any custom arguments so it can be supported by embedly. Since embedly doesn't allow custom arguments, we cannot use ?vistype in oembed. We have replaced this with /vo/slug/vistype instead,

Further, we simplify vistype so it reads more easily. The changed vistypes are: barchart-interactive –> bar barchart-fixed –> bar-static tabular-candidate-by-round –> table tabular-by-round-interactive –> table-by-round tabular-by-round –> table-by-round-static tabular-by-candidate –> table-by-candidate

get_redirect_url(**args*, ***kwargs*)
Return the URL redirect to. Keyword arguments from the URL pattern match generating the redirect request are provided as kwargs to this method.

pattern_name = 'visualizeEmbedded'

permanent = True

3.2 Automatic Movie Generation

3.2.1 Subpackages

Movie Creator

Movie Creator

Movie generation entry point. Allows creation of movies from a jsonConfig at various resolutions.

class movie.creation.movieCreator.**MovieCreationFactory**(*browser*, *domain*, *jsonconfig*)

Bases: object

Holds expensive-to-create resources necessary to create a movie

make_one_movie_at_resolution(*width*, *height*)

Create a movie at a specific resolution

classmethod save_and_upload(*movie*, *slug*, *mp4FileObject*, *gifFileObject*, *titleImageFileObject*)

Saves data to the given movie object and uploads the video and image to the movie model.

exception movie.creation.movieCreator.**ProbablyFailedToLaunchBrowser**

Bases: Exception

A common error when the browser has an issue.

class movie.creation.movieCreator.**SingleMovieCreator**(*browser*, *textToSpeechFactory*, *jsonconfig*, *size*)

Bases: object

Class for creation of a single movie at a single resolution.

make_gif(*gifFilename*)

Creates a gif without titles or captions, just the rounds

make_movie(*mp4Filename*, *staticPngFilename*)

Create a movie at a specific resolution

Text-to-Speech

Text-to-speech via Amazon Polly.

exception movie.creation.textToSpeech.**AudioGenerationFailedException**

Bases: Exception

AWS Polly returned an audio generation failure.

exception movie.creation.textToSpeech.**AudioGenerationTimedOutException**

Bases: Exception

Waited too long without a response

class movie.creation.textToSpeech.**GeneratedAudioWrapper**(*pollyClient*, *s3Client*, *text*)

Bases: object

To facilitate asynchronous waiting for Polly audio generation. Initializaton spawns the AWS job, and there are various methods to poll for the result. Always checks TextToSpeechCachedFile first.

download_if_ready(*toFilename*)

Download the result if it's ready. Can only be called once, then deletes the result from S3.

download_synchronously(*timeoutSeconds=20*)

Wait up to timeoutSeconds, waiting for the task to complete. @return a tempfile object: the file will be deleted once the object is destructed.

prefix = 'generated_speech'

region = None

class movie.creation.textToSpeech.**TextToSpeechFactory**

Bases: object

Holds on to boto clients, initializing an AWS session once and allowing reuses of that session for text-to-speech.

text_to_speech(*text*)

Returns a GeneratedAudioWrapper which you can poll for the result.

3.2.2 Models

Models for storing data about a movie

class movie.models.**Movie**(**args*, ***kwargs*)

Bases: Model

An automatically-generated Movie showing the interaction.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

generatedOnApplicationVersion

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

gifFile

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

movieFile

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

objects = <django.db.models.manager.Manager object>

resolutionHeight

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

resolutionWidth

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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.

titleImage

Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

class movie.models.**SpeechSynthStorage**

Bases: DefaultStorage

Speech synth is stored in a separate bucket. No-op when using offline mode.

bucket_name = None

class movie.models.**TextToSpeechCachedFile**(**args*, ***kwargs*)

Bases: Model

A mapping from a text to an audio file of the text-to-speech mp3

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

audioFile

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

get_next_by_lastUsed(**, field=<django.db.models.fields.DateTimeField: lastUsed>, is_next=True, **kwargs*)

get_previous_by_lastUsed(*, *field=<django.db.models.fields.DateTimeField: lastUsed>*, *is_next=False*, ***kwargs*)

lastUsed

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

text

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

3.2.3 Tasks

Tasks fail to generate docs on readthedocs.io, skipping...

3.2.4 Tests

Unit and integration tests for automatic movie creation

class movie.tests.**MovieCreationTestsIntegration**(*methodName='runTest'*)

Bases: StaticLiveServerTestCase

Integration tests - no mocking here to test everything above that was mocked, but with short text

test_caching_polly()

Make sure that TextToSpeechCachedFile reads and writes properly

class movie.tests.**MovieCreationTestsMocked**(*methodName='runTest'*)

Bases: StaticLiveServerTestCase

Tests for the Movie Creation module. Currently, these do not test the celery connection.

setUp()

Hook method for setting up the test fixture before exercising it.

tearDown()

Hook method for deconstructing the test fixture after testing it.

test_avoid_upload_collision()

Ensure that a unique filename is created for each upload. Regression for the vertical upload immediately overriding the horizontal.

test_captions_all_as_expected(*mockSpawnAudio*, *mockGenerateImage*, *mockWriteVideoFile*)

Integration test to verify the end-to-end script

test_failure_status(*mockTraceback*)

Test that the failure status is accurately set

test_long_text_doesnt_fail()

Make sure that very long text requests don't crash, but don't save a cache either.

test_movie_task_by_url(*mockCreateMovie*)

Test create_movie_task() is called with .delay() when accessing /createMovie

test_movie_task_without_celery(*mockGetNumRounds*)

Test that the movie gets created when calling create_movie_task() directly

3.2.5 URLs

URLs for movie generation and viewing

3.2.6 Views

Views for movie generation and viewing

class movie.views.**CreateMovie**(***kwargs*)

Bases: LoginRequiredMixin, RedirectView

Create a movie. Admin access required for this long-running process.

get_redirect_url(**args*, ***kwargs*)

Return the URL redirect to. Keyword arguments from the URL pattern match generating the redirect request are provided as kwargs to this method.

login_url = '/admin/login/'

permanent = False

query_string = False

class movie.views.**MovieGenerationView**(***kwargs*)

Bases: DetailView

The view used by movie generation - not intended to be user-facing, but no harm done by exposing it either.

dispatch(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)

Insert the single object into the context dict.

model

alias of *JsonConfig*

template_name = 'movie/movie-generation.html'

class movie.views.**VisualizeMovie**(***kwargs*)

Bases: DetailView

Temporary view to see just the movie visualization. Delete once it's integrated into a share button.

dispatch(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)

Insert the single object into the context dict.

model

alias of *JsonConfig*

template_name = 'movie/only-movie.html'

3.3 REST API

The REST API usage documentation can be found at https://www.rcvis.com/api/ This documentation only covers the internal modules used to support the REST API.

3.3.1 Serializers

Data serializers - used for the REST API

class visualizer.serializers.**BallotpediaSerializer**(**args*, ***kwargs*)

Bases: *BaseVisualizationSerializer*

A serializer for specifying the Ballotpedia data.

Instead of specifying jsonFile (which is confusing because the sidecar is also JSON), this serializer uses "resultsSummaryFile" and "candidateSidecarFile", along with the other options bp wants: 1. dataSourceURL (URL) 2. areResultsCertified (bool) 3. isPrimary (bool)

class Meta

Bases: *Meta*

```
bp_fields = ('jsonFile', 'textForWinner', 'candidateSidecarFile',
'dataSourceURL', 'areResultsCertified', 'owner')

fields = ('slug', 'id', 'movieHorizontal', 'movieVertical',
'movieGenerationStatus', 'numRounds', 'numCandidates', 'title', 'jsonFile',
'textForWinner', 'candidateSidecarFile', 'dataSourceURL', 'areResultsCertified',
'owner')

resultsSummaryFile = SerializerMethodField()
```

to_internal_value(*data*)

Instead of jsonFile, this endpoint uses the more-descriptive "resultsSummaryFile". Rename resultsSummaryFile to jsonFile here.

class visualizer.serializers.**BaseVisualizationSerializer**(**args*, ***kwargs*)

Bases: HyperlinkedModelSerializer

The rest_framework serializer for a JsonConfig Model. DRF expects a fixed set of options, so this uses the model defaults and nothing more.

class Meta

Bases: object

The meta class to simplify construction of the serializer

```
fields = ('slug', 'id', 'movieHorizontal', 'movieVertical',
'movieGenerationStatus', 'numRounds', 'numCandidates', 'title')
```

model

alias of *JsonConfig*

```
owner = ReadOnlyField(source='owner.username')

read_only_but_validate_fields = ('numRounds', 'numCandidates', 'title')
```

read_only_fields = ('slug', 'id', 'movieHorizontal', 'movieVertical', 'movieGenerationStatus')

check_for_superfluous_fields_before_modification(*data*)

Raises a ValidationError if the data does not have superfluous fields.

classmethod load_graph_or_errors(*jsonFile*, *candidateSidecarFile*)

Returns the graph, or raises an error if it cannot.

classmethod populate_dict_with_json_data(*model*, *graph*)

Adds title, num rounds, num candidates to the model. Does not save the model.

classmethod populate_model_with_json_data(*model*, *graph*)

Same as populate_dict_with_json_data, but using dot notation instead of bracket

to_internal_value(*data*)

Before saving from the REST API, populates all required data

to_representation(*instance*)

Object instance -> Dict of primitive datatypes.

class visualizer.serializers.JsonOnlySerializer(**args*, ***kwargs*)

Bases: *BaseVisualizationSerializer*

A serializer for specifying just the JSON file. All other settings are set to their default values.

class Meta

Bases: *Meta*

fields = ('slug', 'id', 'movieHorizontal', 'movieVertical', 'movieGenerationStatus', 'numRounds', 'numCandidates', 'title', 'jsonFile', 'owner')

writeable_fields = ('jsonFile', 'owner')

class visualizer.serializers.UserSerializer(**args*, ***kwargs*)

Bases: ModelSerializer

The rest_framework serializer for a User Model

class Meta

Bases: object

The meta class to simplify construction of the serializer

fields = ['id', 'username', 'this_users_jsons']

model

alias of User

ordering = ['-id']

3.3.2 Validators

Data validation - to be used across REST and HTTP access

visualizer.validators.**ensure_file_is_under_2_mb**(*jsonFileObj*)

Limit file size to 2mb

visualizer.validators.**ensure_title_is_under_256_chars**(*graph*)

Limit title length 256 chars

visualizer.validators.**try_to_load_jsons**(*jsonFileObj*, *sidecarJsonFileObj*)

Checks that the JSON can be loaded and is under 2mb. Raises: - BadJSONError: Summary JSON cannot be loaded - BadSidecarError: Sidecar JSON cannot be loaded - ValidationError: 2mb limit is reached - Anything else: unknown error Returns: - Loaded graph

3.4 Common cross-app utils

3.4.1 Common

Helper functions for unit and integration tests

class common.testUtils.**TestHelpers**

Bases: object

Helper function for various test classes below

classmethod copy_with_new_name(*jsonFileToCopy*, *newName*, *newFilenamePrefix=None*)

Copies the file to a tempfile, but changes the name. Returns the tempfile

classmethod create_non_admin_api_user()

Basic setup for tests: create a non-admin API user

classmethod create_request_mock(*data*, *statusCode*)

Creates a mock function call for the requests library

classmethod does_fieldfile_equal_file(*fsFilePath*, *fieldFile*)

Does the FieldFile (django field) equal the file at fsFilePath?

classmethod generate_random_valid_json_of_size(*numBytes*)

Generates a valid but strange JSON of size num_bytes and returns the filename

classmethod get_email_reg_link(*outbox*)

Gets the auth regisration link from the email outbox

classmethod get_headless_browser()

Returns a headless browser

classmethod get_latest_upload()

Returns the last-uploaded json config

classmethod get_multiwinner_upload_response(*client*)

Uploads the multiwinner json file and returns a response

classmethod give_auth(*user*, *auths*)

Gives auth or list of auths to the current user, then refetches user from the db

classmethod login(*client*)

Forces a login. Creates a user as needed, and returns that user

classmethod login_with_scrape_permissions(*client*)

Logs in and creates permissions needed to run scraper

classmethod logout(*client*)

Logs out (if logged in)

classmethod make_multi_scraper()

Creates a multi-scraper object. You must mock out requests.get if you plan to scrape.

classmethod make_scraper()

Creates a scraper object. You must mock out requests.get if you plan to scrape.

classmethod mock_scraper_url_with_file(*requestMock*, *url='mock://scrape'*, *filename='testData/oneRound.json'*)

Creates a valid response from the server

classmethod modify_json_with(*jsonFilename*, *modifierFunc*)

Given a modifierFunc which modifies the data in the given file, updates the json file and returns a Tempfile holding the new data

classmethod setup_host_mocks(*testClass*)

Mock out get_host (since tests don't set it but visualizer depends on it

classmethod silence_logging_spam()

Clean up output of misc libraries

Utility functions shared across views, in either movie or visualizer apps

class common.viewUtils.**DefaultConfig**

Bases: object

A simplified JsonConfig with just the default values. Use this to pass to functions that require a config if you do not have or need a config (e.g. in validators).

common.viewUtils.**default_iframe_height**(*numCandidates*)

How tall should the iframe be for a vis with this # of candidates?

common.viewUtils.**get_data_for_graph**(*graph*, *config*)

Helper function for get_data_for_view: convert the graph to data to be passed on to JS.

common.viewUtils.**get_data_for_round_describer**(*graph*, *config*)

Helper function for get_data_for_view: convert the round describer to data to be passed on to JS

common.viewUtils.**get_data_for_view**(*config*)

All data needed to pass on to the visualize or visualizeembedded view

common.viewUtils.**get_embed_html**(*embedlyUrl*, *maxwidth*, *maxheight*)

Given the absolute URL to the visualizeEmbedly page, returns the UTF-8 encoded HTML needed to embed that page in an iframe.

common.viewUtils.**get_script_to_disable_animations**()

Disables transitions on the current page

common.viewUtils.**request_to_domain**(*request*)

Gets the domain name from the request

3.5 Descriptors

3.5.1 Round Describer

3.5.2 Dynamic FAQ Generator

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

m

v

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W