django-politico-civic Documentation

Release 0.0.0-alpha

POLITICO

Jan 20, 2019

Contents:

CHAPTER 1

Why this?

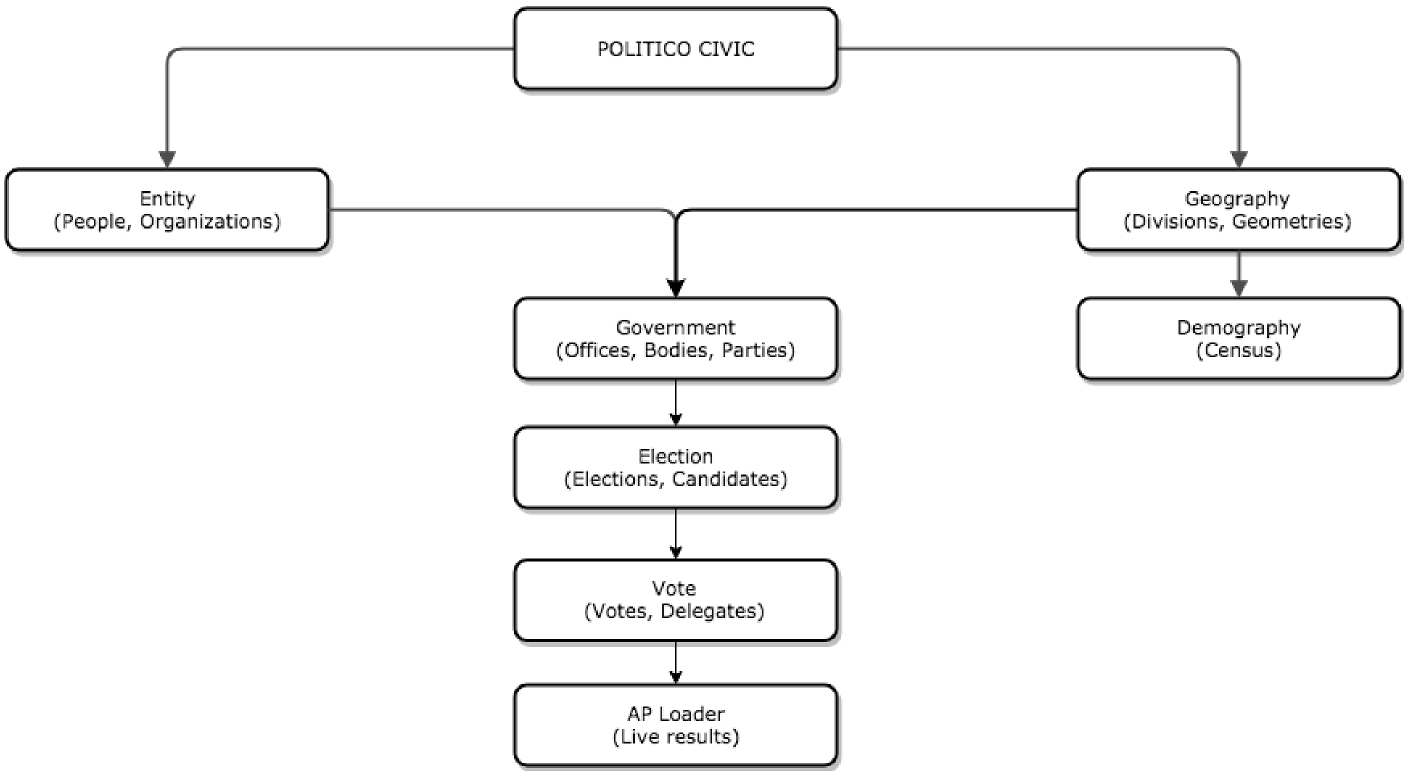
At POLITICO, civic data is a key component of our report. We record the results of every federal election in the country. We track the movements of prospective presidential candidates. We collect campaign finance data. All of these data tasks require the same foundation of data. Thus, the POLITICO Civic project was born.

If you work in a newsroom or with any civic data project, you might have similar problems to solve. Newsrooms across the United States spend many months every two years (at least) building the same piece of technology: a system to ingest election results as quickly as possible and display them with fancy data visualizations. By sharing POLITICO Civic, we hope to deescalate that arms race.

1.1 What is it?

POLITICO Civic is a Django project composed of a number of pluggable Django apps. Each of the pluggable apps contains models around a particular facet of civic data and standard serializers that allow us to pass data around through JSON. At the bottom of the app tree, the apps contain front-end applications for live results, election calendars and other data-driven displays.

To give an illustration of how POLITICO Civic works, here is the election results app tree and its dependency structure described in a big scary dependency diagram.


POLITICO CIVIC
Entity (People, Organizations)
Geography (Divisions, Geometries)
Government (Offices, Bodies, Parties)
Demography (Census)
Election (Elections, Candidates)
Vote (Votes, Delegates)
AP Loader (Live results)

Don't run away! I promise this makes sense!

1.2 Benefits

Modularity: We designed our project this way to make each component of civic data easier to reason about. And when we start supporting other types of civic data, we don't have to add bloat to another Django application. We've designed in a way that allows us to start that app from scratch and pull in the dependencies we need.

Predictable structure: During high-stress live events such as election nights, having a strong schema and foundation for all of the underlying data that goes into an election night — information about political offices, geography, election cycles, primary conditions, and more — lets us breathe a little easier and focus more on what is new about that night: the results.

Reusability: Some of these applications are useful outside the context of POLITICO Civic. For example, Geography contains all of the geographic data for political divisions in the United States. It comes with a bootstrap process built-in that grabs the latest geodata for states, counties and Congressional districts from the U.S. Census Bureau. It can also compress that data into topojson and dump it to Amazon S3. That is useful for any newsroom that might make maps of the United States.

1.3 Core technologies

POILTICO Civic is based on several key pieces of technology:

- Python (3.6+)
- Django (2.0+)
- PostgreSQL

- Django REST Framework
- Celery
- Elex
- Fabric

Our model architecture took inspiration from a couple inspired projects:

- opencivicdata
- Popolo

CHAPTER 2

Using component apps in Django

The component apps in politico-civic are designed to be plug-and-play. You can install any of them in your own Django project and they should work within your project and install all their necessary dependencies. Each app contains its own bootstrap management command that will seed your models with real data.

For example, let's install `politico-civic-vote` in a Django project. You can follow these steps for any of POLITICO Civic's component apps.

First, you need to set up your Django project with a PostgreSQL database. Read the Django docs on databases if you don't know how to do this.

Then, install the component application.

```
$ pip install politico-civic-vote
```

In your Django settings, add the app *and its dependencies* to your `INSTALLED_APPS` section. Consult the dependency diagram in the quickstart section to determine your dependencies.

```
INSTALLED_APPS = [
    ...
    "rest_framework",
    "entity",
    "geography",
    "government",
    "election",
    "vote",
]
```

Then, migrate your database.

```
$ python manage.py migrate
```

No matter which component app you choose to install, you can use a Django management command to seed your database with real data. For `politico-civic-vote`, the command is `bootstrap_vote`. The naming convention extends to whichever app you isntalled. Each component app will seed its own data and the data of the apps it depends on.

Run the management command like this:

```
$ python manage.py bootstrap_vote
```

Note: If you use anything depending on `politico-civic-government`, you will need an API key from the ProPublica Congress API. Export it into your environment as `PROPUBLICA_CONGRESS_API_KEY`.

That's it! Open your Django admin and see your seed data.

CHAPTER 3

Quickstart

Use these docs if you're trying to install the entire `politico-civic` project. If you don't work at POLITICO, you probably don't want this. Instead, install the component apps you want in your own Django project.

1. Install global dependencies for the project:

```
$ brew install jq
$ pip install pipenv
```

Get Terraform from the project website.

2. Create local PostgreSQL database

```
$ createdb civic
```

3. Fill out your .env file

```
DATABASE_URL="postgresql://username:password@localhost:5432/civic"
...
(get all of our API keys from someone on the team)
```

4. Install local dependencies for the project:

```
$ pipenv install
$ pipenv shell
$ python setup.py develop
```

5. Bootstrap database

```
$ python manage.py bootstrap_electionnight
```

6. Check it out!

```
$ python manage.py runserver
```

CHAPTER 4

Models

4.1 aploader

AP Loader leverages elex, a tool created by NPR and the New York Times, to get election results from the AP Elections API.

`APElectionMeta`

class `aploader.models.`**`APElectionMeta`**`(`**args*, ***kwargs*`)`

Election information corresponding to AP election night.

Parameters

- **`id`** (*`AutoField`*) – Id
- **`election`** (OneToOneField to `Election`) – Election
- **`ballot_measure`** (OneToOneField to `BallotMeasure`) – Ballot measure
- **`ap_election_id`** (*`CharField`*) – Ap election id
- **`called`** (*`BooleanField`*) – Called
- **`tabulated`** (*`BooleanField`*) – Tabulated
- **`override_ap_call`** (*`BooleanField`*) – Override ap call
- **`override_ap_votes`** (*`BooleanField`*) – Override ap votes
- **`precincts_reporting`** (*`PositiveIntegerField`*) – Precincts reporting
- **`precincts_total`** (*`PositiveIntegerField`*) – Precincts total
- **`precincts_reporting_pct`** (*`DecimalField`*) – Precincts reporting pct

`ChamberCall`

class `aploader.models.`**`ChamberCall`**`(`**args*, ***kwargs*`)`

Calls for chambers of Congress

Parameters

- **id** (*AutoField*) – Id
- **body** (ForeignKey to Body) – Body
- **cycle** (ForeignKey to ElectionCycle) – Cycle
- **party** (ForeignKey to Party) – Party
- **call_time** (*DateTimeField*) – Call time

4.2 demography

Demography collects and aggregates Census variables by the political divisions defined in Geography.

4.2.1 CensusEstimate

class demography.models.**CensusEstimate**(**args*, ***kwargs*)
Individual census series estimates.

Parameters

- **id** (*AutoField*) – Id
- **division** (ForeignKey to Division) – Division
- **variable** (ForeignKey to CensusVariable) – Variable
- **estimate** (*FloatField*) – Estimate

4.2.2 CensusLabel

class demography.models.**CensusLabel**(**args*, ***kwargs*)
Custom labels for census variables that allow us to aggregate variables.

Parameters

- **id** (*AutoField*) – Id
- **label** (*CharField*) – Label
- **aggregation** (*CharField*) – Aggregation
- **table** (ForeignKey to CensusTable) – Table

4.2.3 CensusTable

class demography.models.**CensusTable**(**args*, ***kwargs*)
A census series.

Parameters

- **id** (*AutoField*) – Id
- **series** (*CharField*) – Series
- **year** (*CharField*) – Year
- **code** (*CharField*) – Code
- **title** (*CharField*) – Title

4.2.4 `CensusVariable`

class `demography.models.`**`CensusVariable`**(**args*, ***kwargs*)

Individual variables on census series to pull, e.g., "001E" on ACS table 19001, the total for household income.

Parameters

- **`id`** (*`AutoField`*) – Id
- **`code`** (*`CharField`*) – 3 digit code for variable and 'E', e.g., 001E.
- **`table`** (ForeignKey to `CensusTable`) – Table
- **`label`** (ForeignKey to `CensusLabel`) – Label

4.3 election

Election models information about races for particular offices. It also models candidate information, which inherits people from Entity and attaches them to races in Election.

4.3.1 `BallotAnswer`

class `election.models.`**`BallotAnswer`**(**args*, ***kwargs*)

An answer to a ballot question.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`answer`** (*`TextField`*) – Answer
- **`winner`** (*`BooleanField`*) – Winner
- **`ballot_measure`** (ForeignKey to `BallotMeasure`) – Ballot measure

4.3.2 `BallotMeasure`

class `election.models.`**`BallotMeasure`**(**args*, ***kwargs*)

A ballot measure.

Parameters

- **`uid`** (*`CharField`*) – Uid
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`question`** (*`TextField`*) – Question
- **`division`** (ForeignKey to `Division`) – Division
- **`number`** (*`CharField`*) – Number
- **`election_day`** (ForeignKey to `ElectionDay`) – Election day

4.3.3 `Candidate`

class election.models.**Candidate**(**args*, ***kwargs*)

A person who runs in a race for an office.

Parameters

- **id** (*UUIDField*) – Id
- **uid** (*CharField*) – Uid
- **race** (ForeignKey to `Race`) – Race
- **person** (ForeignKey to `Person`) – Person
- **party** (ForeignKey to `Party`) – Party
- **ap_candidate_id** (*CharField*) – Ap candidate id
- **incumbent** (*BooleanField*) – Incumbent
- **top_of_ticket** (ForeignKey to `Candidate`) – Top of ticket
- **prospective** (*BooleanField*) – The candidate has not yet declared her candidacy.

4.3.4 `CandidateElection`

class election.models.**CandidateElection**(**args*, ***kwargs*)

A CandidateElection represents the abstract relationship between a candidate and an election and carries properties like whether the candidate is uncontested or whether we aggregate their vote totals.

Parameters

- **id** (*UUIDField*) – Id
- **candidate** (ForeignKey to `Candidate`) – Candidate
- **election** (ForeignKey to `Election`) – Election
- **aggregable** (*BooleanField*) – Aggregable
- **uncontested** (*BooleanField*) – Uncontested

4.3.5 `Election`

class election.models.**Election**(**args*, ***kwargs*)

A specific contest in a race held on a specific day.

Parameters

- **uid** (*CharField*) – Uid
- **election_type** (ForeignKey to `ElectionType`) – Election type
- **race** (ForeignKey to `Race`) – Race
- **party** (ForeignKey to `Party`) – Party
- **election_day** (ForeignKey to `ElectionDay`) – Election day
- **division** (ForeignKey to `Division`) – Division
- **candidates** (*ManyToManyField*) – Candidates

4.3.6 `ElectionCycle`

class election.models.**ElectionCycle**(*uid*, *slug*, *name*)

Parameters

- **uid** (*CharField*) – Uid
- **slug** (*SlugField*) – Slug
- **name** (*CharField*) – Name

4.3.7 `ElectionDay`

class election.models.**ElectionDay**(**args*, ***kwargs*)

A day on which one or many elections can be held.

Parameters

- **uid** (*CharField*) – Uid
- **slug** (*SlugField*) – Slug
- **date** (*DateField*) – Date
- **cycle** (ForeignKey to ElectionCycle) – Cycle

4.3.8 `ElectionEvent`

class election.models.**ElectionEvent**(**args*, ***kwargs*)

A statewide election event

Parameters

- **id** (*AutoField*) – Id
- **slug** (*SlugField*) – Slug
- **label** (*CharField*) – Label
- **event_type** (*CharField*) – Event type
- **dem_primary_type** (*CharField*) – Dem primary type
- **gop_primary_type** (*CharField*) – Gop primary type
- **election_day** (ForeignKey to ElectionDay) – Election day
- **division** (ForeignKey to Division) – Division
- **early_vote_start** (*DateField*) – Early vote start
- **early_vote_close** (*DateField*) – Early vote close
- **vote_by_mail_application_deadline** (*DateField*) – Vote by mail application deadline
- **vote_by_mail_ballot_deadline** (*DateField*) – Vote by mail ballot deadline
- **online_registration_deadline** (*DateField*) – Online registration deadline
- **registration_deadline** (*DateField*) – Registration deadline
- **poll_closing_time** (*DateTimeField*) – Poll closing time

4.3.9 `ElectionType`

class `election.models.`**`ElectionType`**`(`**args*, ***kwargs*`)`

e.g., "General", "Primary"

Parameters

- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`ap_code`** (*`CharField`*) – Ap code
- **`number_of_winners`** (*`PositiveSmallIntegerField`*) – Number of winners
- **`winning_threshold`** (*`DecimalField`*) – Winning threshold

4.3.10 `Race`

class `election.models.`**`Race`**`(`**args*, ***kwargs*`)`

A race for an office comprised of one or many elections.

Parameters

- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`description`** (*`TextField`*) – Description
- **`office`** (ForeignKey to `Office`) – Office
- **`cycle`** (ForeignKey to `ElectionCycle`) – Cycle
- **`special`** (*`BooleanField`*) – Special

4.4 electionnight

Election Night builds live results pages based on AP data and models the text content needed on those pages.

4.4.1 `PageContent`

class `electionnight.models.`**`PageContent`**`(`**args*, ***kwargs*`)`

A specific page that content can attach to.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`content_type`** (ForeignKey to `ContentType`) – Content type
- **`object_id`** (*`CharField`*) – Object id

- **election_day** (ForeignKey to `ElectionDay`) – Election day
- **division** (ForeignKey to `Division`) – Division
- **special_election** (*BooleanField*) – Special election
- **parent** (ForeignKey to `PageContent`) – Parent
- **featured** (*ManyToManyField*) – Featured

4.4.2 PageContentBlock

class electionnight.models.**PageContentBlock**(**args*, ***kwargs*)

A block of content for an individual page.

Parameters

- **id** (*UUIDField*) – Id
- **page** (ForeignKey to `PageContent`) – Page
- **content_type** (ForeignKey to `PageContentType`) – Content type
- **content** (*MarkdownField*) – Content
- **created** (*DateTimeField*) – Created
- **updated** (*DateTimeField*) – Updated

4.4.3 PageContentType

class electionnight.models.**PageContentType**(**args*, ***kwargs*)

The kind of content contained in a content block. Used to serialize content blocks.

Parameters

- **slug** (*SlugField*) – Slug
- **name** (*CharField*) – Name

4.4.4 PageType

class electionnight.models.**PageType**(**args*, ***kwargs*)

A type of page that content can attach to.

Parameters

- **id** (*UUIDField*) – Id
- **model_type** (ForeignKey to `ContentType`) – Model type
- **election_day** (ForeignKey to `ElectionDay`) – Election day
- **division_level** (ForeignKey to `DivisionLevel`) – Set for all page types except generic election day
- **jurisdiction** (ForeignKey to `Jurisdiction`) – Only set jurisdiction for federal pages
- **body** (ForeignKey to `Body`) – Only set body for senate/house pages
- **office** (ForeignKey to `Office`) – Only set office for the presidency

4.5 entity

Entity houses models for people and organizations. For example, the Republican Party is an organization, and Mitt Romney is a person. Their roles as political parties and candidates will come in downstream apps, but Entity houses the base level information about them.

4.5.1 `ImageTag`

class `entity.models.`**`ImageTag`**(**args*, ***kwargs*)

Tags represent a type of image, which is used to serialize it.

Parameters

- **`id`** (*`AutoField`*) – Id
- **`name`** (*`SlugField`*) – Name

4.5.2 `Organization`

class `entity.models.`**`Organization`**(**args*, ***kwargs*)

An org.

Generally follows the Popolo spec: http://www.popoloproject.com/specs/organization.html

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`name`** (*`CharField`*) – Name
- **`identifiers`** (*`JSONField`*) – Identifiers
- **`classification`** (ForeignKey to `OrganizationClassification`) – Classification
- **`parent`** (ForeignKey to `Organization`) – Parent
- **`national_headquarters`** (*`CountryField`*) – National headquarters
- **`founding_date`** (*`DateField`*) – Founding date
- **`dissolution_date`** (*`DateField`*) – Dissolution date
- **`summary`** (*`CharField`*) – A one-line biographical summary.
- **`description`** (*`TextField`*) – A longer-form description.
- **`links`** (*`ArrayField`*) – External web links, comma-separated.
- **`created`** (*`DateTimeField`*) – Created
- **`updated`** (*`DateTimeField`*) – Updated

4.5.3 `OrganizationClassification`

class `entity.models.`**`OrganizationClassification`**(*id*, *name*)

Parameters

- **`id`** (*`AutoField`*) – Id
- **`name`** (*`CharField`*) – Name

4.5.4 `OrganizationImage`

class `entity.models.`**`OrganizationImage`**(**args*, ***kwargs*)

Image attached to a person, which can be serialized by a tag.

Parameters

- **`id`** (*`AutoField`*) – Id
- **`organization`** (ForeignKey to `Organization`) – Organization
- **`tag`** (ForeignKey to `ImageTag`) – Used to serialize images.
- **`image`** (*`ImageField`*) – Image
- **`created`** (*`DateTimeField`*) – Created
- **`updated`** (*`DateTimeField`*) – Updated

4.5.5 `Person`

class `entity.models.`**`Person`**(**args*, ***kwargs*)

A real human being.

Generally follows the Popolo spec: http://www.popoloproject.com/specs/person.html

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`last_name`** (*`CharField`*) – Last name
- **`first_name`** (*`CharField`*) – First name
- **`middle_name`** (*`CharField`*) – Middle name
- **`suffix`** (*`CharField`*) – Suffix
- **`full_name`** (*`CharField`*) – Full name
- **`identifiers`** (*`JSONField`*) – Identifiers
- **`gender`** (*`GenderField`*) – Gender
- **`race`** (*`RaceField`*) – Race
- **`nationality`** (*`CountryField`*) – Nationality
- **`state_of_residence`** (*`StateField`*) – If U.S. resident.
- **`birth_date`** (*`DateField`*) – Birth date

- **death_date** (*DateField*) – Death date
- **summary** (*CharField*) – A one-line biographical summary.
- **description** (*TextField*) – A longer-form description.
- **links** (*ArrayField*) – External web links, comma-separated.
- **created** (*DateTimeField*) – Created
- **updated** (*DateTimeField*) – Updated

4.5.6 `PersonImage`

class `entity.models.`**`PersonImage`**(**args*, ***kwargs*)

Image attached to a person, which can be serialized by a tag.

Parameters

- **id** (*AutoField*) – Id
- **person** (ForeignKey to `Person`) – Person
- **tag** (ForeignKey to `ImageTag`) – Used to serialize images.
- **image** (*URLField*) – Image
- **created** (*DateTimeField*) – Created
- **updated** (*DateTimeField*) – Updated

4.6 geography

Geography houses models for all of the geographic political divisions in the United States. It contains bootstrap scripts that get shapefiles from the Census Bureau for states, counties and congressional districts and load them into your database. It also creates a simplified geography for each of those objects.

4.6.1 `Division`

class `geography.models.`**`Division`**(**args*, ***kwargs*)

A political or administrative geography.

For example, a particular state, county, district, precinct or municipality.

Parameters

- **id** (*UUIDField*) – Id
- **uid** (*CharField*) – Uid
- **slug** (*SlugField*) – Slug
- **name** (*CharField*) – Name
- **label** (*CharField*) – Label
- **short_label** (*CharField*) – Short label
- **parent** (ForeignKey to `Division`) – Parent
- **level** (ForeignKey to `DivisionLevel`) – Level

- **code** (*CharField*) – Code representing a geography: FIPS code for states and counties, district number for districts, precinct number for precincts, etc.
- **code_components** (*JSONField*) – Component parts of code
- **effective** (*BooleanField*) – Effective
- **effective_start** (*DateTimeField*) – Effective start
- **effective_end** (*DateTimeField*) – Effective end
- **intersecting** (*ManyToManyField*) – Intersecting divisions intersect this one geographically but do not necessarily have a parent/child relationship. The relationship between a congressional district and a precinct is an example of an intersecting relationship.

4.6.2 DivisionLevel

class geography.models.**DivisionLevel**(**args*, ***kwargs*)

Level of government or administration at which a division exists.

For example, federal, state, district, county, precinct, municipal.

Parameters

- **id** (*UUIDField*) – Id
- **uid** (*CharField*) – Uid
- **slug** (*SlugField*) – Slug
- **name** (*CharField*) – Name
- **parent** (ForeignKey to DivisionLevel) – Parent

4.6.3 Geometry

class geography.models.**Geometry**(**args*, ***kwargs*)

The spatial representation (in topoJSON) of a Division.

Parameters

- **id** (*UUIDField*) – Id
- **division** (ForeignKey to Division) – Division
- **subdivision_level** (ForeignKey to DivisionLevel) – Subdivision level
- **simplification** (*FloatField*) – Minimum quantile of planar triangle areas for simplfying topojson.
- **topojson** (*JSONField*) – Topojson
- **source** (*URLField*) – Link to the source of this geography data.
- **series** (*CharField*) – Year of boundary series, e.g., 2016 TIGER/Line files.
- **effective** (*BooleanField*) – Effective
- **effective_start** (*DateField*) – Effective start
- **effective_end** (*DateField*) – Effective end

4.6.4 `IntersectRelationship`

class `geography.models.`**`IntersectRelationship`**(**args*, ***kwargs*)

Each IntersectRelationship instance represents one side of a paired relationship between intersecting divisions.

The intersection field represents the decimal proportion of the to_division that intersects with the from_division. It's useful for apportioning counts between the areas, for example, population statistics from census data.

Parameters

- **`id`** (*`AutoField`*) – Id
- **`from_division`** (ForeignKey to `Division`) – From division
- **`to_division`** (ForeignKey to `Division`) – To division
- **`intersection`** (*`DecimalField`*) – The portion of the to_division that intersects this division.

4.7 government

Government contains information about political jurisdictions, bodies, and offices. For example, the United States Federal Government is a jurisdiction, the U.S. Senate is a body, and the Class 1 Senate seat from Texas is an office. It also contains the modeling for political parties.

4.7.1 `Body`

class `government.models.`**`Body`**(**args*, ***kwargs*)

A body represents a collection of offices or individuals organized around a common government or public service function.

For example: the U.S. Senate, Florida House of Representatives, Columbia City Council, etc.

Note: Duplicate slugs are allowed on this model to accomodate states, for example:

- florida/senate/
- michigan/senate/

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Customizable slug. Defaults to Org slug without stopwords.
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`organization`** (OneToOneField to `Organization`) – Organization
- **`jurisdiction`** (ForeignKey to `Jurisdiction`) – Jurisdiction
- **`parent`** (ForeignKey to `Body`) – Parent

4.7.2 `Jurisdiction`

class `government.models.`**`Jurisdiction`**(**args*, ***kwargs*)

A Jurisdiction represents a logical unit of governance, comprised of a collection of legislative bodies, administrative offices or public services.

For example: the United States Federal Government, the Government of the District of Columbia, Columbia Missouri City Government, etc.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`name`** (*`CharField`*) – Name
- **`division`** (ForeignKey to `Division`) – Division
- **`parent`** (ForeignKey to `Jurisdiction`) – Parent

4.7.3 `Office`

class `government.models.`**`Office`**(**args*, ***kwargs*)

An office represents a post, seat or position occuppied by an individual as a result of an election.

For example: Senator, Governor, President, Representative.

In the case of executive positions, like governor or president, the office is tied directly to a jurisdiction. Otherwise, the office ties to a body tied to a jurisdiction.

Note: Duplicate slugs are allowed on this model to accomodate states, for example:

- florida/house/seat-2/
- michigan/house/seat-2/

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`uid`** (*`CharField`*) – Uid
- **`slug`** (*`SlugField`*) – Slug
- **`name`** (*`CharField`*) – Name
- **`label`** (*`CharField`*) – Label
- **`short_label`** (*`CharField`*) – Short label
- **`senate_class`** (*`CharField`*) – Senate class
- **`division`** (ForeignKey to `Division`) – Division
- **`jurisdiction`** (ForeignKey to `Jurisdiction`) – Jurisdiction
- **`body`** (ForeignKey to `Body`) – Body

4.7.4 `Party`

class government.models.**Party**(**args*, ***kwargs*)
A political party.

Parameters

- **id** (*UUIDField*) – Id
- **uid** (*CharField*) – Uid
- **slug** (*SlugField*) – Customizable slug. Defaults to slugged Org name.
- **label** (*CharField*) – Label
- **short_label** (*CharField*) – Short label
- **organization** (OneToOneField to `Organization`) – All parties except Independent should attach to an Org.
- **ap_code** (*CharField*) – Ap code
- **aggregate_candidates** (*BooleanField*) – Determines whether to globally aggregate vote totals of this party's candidates during an election.

4.8 vote

Vote models various types of voting that happens in elections.

4.8.1 `Delegates`

class vote.models.**Delegates**(**args*, ***kwargs*)
Pledged delegates.

Parameters

- **id** (*UUIDField*) – Id
- **division** (ForeignKey to `Division`) – Division
- **count** (*PositiveIntegerField*) – Count
- **pct** (*DecimalField*) – Pct
- **total** (*PositiveIntegerField*) – Total
- **candidate_election** (ForeignKey to `CandidateElection`) – Candidate election
- **superdelegates** (*BooleanField*) – Superdelegates

4.8.2 `ElectoralVotes`

class vote.models.**ElectoralVotes**(**args*, ***kwargs*)
Electoral votes.

Parameters

- **id** (*UUIDField*) – Id
- **division** (ForeignKey to `Division`) – Division

- **count** (*PositiveIntegerField*) – Count
- **pct** (*DecimalField*) – Pct
- **total** (*PositiveIntegerField*) – Total
- **candidate_election** (ForeignKey to CandidateElection) – Candidate election
- **winning** (*BooleanField*) – Winning

4.8.3 Votes

class vote.models.**Votes**(**args*, ***kwargs*)
Popular votes.

Parameters

- **id** (*UUIDField*) – Id
- **division** (ForeignKey to Division) – Division
- **count** (*PositiveIntegerField*) – Count
- **pct** (*DecimalField*) – Pct
- **total** (*PositiveIntegerField*) – Total
- **candidate_election** (ForeignKey to CandidateElection) – Candidate election
- **ballot_answer** (ForeignKey to BallotAnswer) – Ballot answer
- **winning** (*BooleanField*) – Winning
- **runoff** (*BooleanField*) – Runoff

CHAPTER 5

Servers

Civic provides a cli called `onespot` that handles server management for you.

To get it installed on your path, make sure your virtual environment is activated, and run `python setup.py develop`.

IMPORTANT: Each `onespot` command takes a `--target=production` argument in order to make these commands run on the production server. By default, the commands go to staging.

You will also need to ensure that you have environment files for the servers in your project. These are gitignored because they contain API keys that we cannot leak to the public. In both the `terraform/staging` and `terraform/production` folders, you will need both a `.env` file and a `terraform.tfvars` file. Talk to Tyler if you don't have these.

You can always run `onespot help` for information on the command line.

5.1 Provisioning

Run these commands when you need to create new servers or push new code to the servers.

5.1.1 Destroy server

```
onespot server destroy
```

This command will completely remove the server and its corresponding security groups from AWS.

5.1.2 Provision new server

```
onespot server launch
```

This command will create a new EC2 instance according to the size defined in *terraform.tfvars*, and associate it with the elastic IP defined in *terraform.tfvars*.

5.1.3 Setup new server

```
onespot server setup
```

This command will install an SSL certificate, setup logging, and install your nginx and uwsgi configuration files to an existing server. Run this after you have launched a new server.

5.1.4 Updating existing server

```
onespot server update
```

This command will grab the latest from the master branch of this repo on Github and put it on the server. Then, it will reinstall requirements, migrate the database if necessary, and collect static files.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index