TEXAS LAW DATABASE

Team: For-git about it

Members: Jimmy Kettler, Eliya Shabanov, Ian Hays, Keyon Mohebzad, Paul Strong, Michael Dong

Contents

Introduction

Problem

The democratic process works best if there is a well-informed citizenry. Given more information, voters

can make more informed decisions on their support for political candidates. However, there currently

does not exist a well-made central database containing information concerning the Texas Senate. There

is no place to see comprehensive information on bills, senators, committees and the relationship

between them. Without such a database, it is difficult for citizens to hold their government accountable.

The ability for voters to see a candidate's voting record allows them to decide for themselves whether a

candidate is suitab   qle for the job. While such databases exist for the federal government

(https://www.govtrack.us/ is an example), the only place where one can access a database for bills going

through the Texas legislature is through the official Texas legislature website, which is lacking in

information. The Texas Law Database exists to provide such a service, to open source democracy in

Texas, and to serve as a source of information so that voters can learn about bills, legislators, and

committees in order to stay informed about the democratic process in Texas.

Use Cases

The Texas Law Database contains the following things:

Information about specific bills that have gone through, and passed, the Texas Senate, including a brief summary, author, status, a link to the full text of the bill, the committee that the bill went through, and a breakdown of the votes for the bill.

Information about individual senators, including a brief biography, photos, mailing address, a list of committees they chair or are a member of, bills they have authored, and their Facebook and/or Twitter feeds, if they have them.

Information about Senate committees, including this session's interim charges (committee agendas), their members, and a list of all bills to have gone through that committee.

Users of the Texas Law Database can learn about the most recent bills to go through committee or a floor vote, or they can look through the lists of senators and committees and see information about the bills they have voted on.

Users can use the Texas Law Database to see Senator's voting records over time, and thus get a better understanding of a senator's position on issues. Voters can then use this information to affect their voting decision.

Users can see a list of bills that passed or failed and gain a greater understanding of the legal trends for the Senate as a whole. Users in the political field can use this information to make informed predictions on proposed bills or bills to be proposed.

The net effect of this information is to allow voters in Texas to contribute to and to learn about the Texas senate, to inject some much needed openness in the democratic process, and to give voters and citizens in Texas the tools to make informed decisions about their Senate.

Approach and Results

Design

Our main approach on the design of the Texas Law Database was user-focused. Every step of the way, our main focus was to present information that voters would like to see, in a way that voters would be able to access. To this end, our Django models contain a laundry list of attributes for all of the data, and twitter Bootstrap allows us to present that information in a clear, beautiful way that is accessible on any device.

Implementation

The key word for our approach to implementation is delegation. Using the Github issue tracker, we were able to easily assign issues to group members as they came up. As it was difficult for all six of us to find room in our schedule to consistently meet up, a large part of the website was not done as a group, but rather individually or in ad-hoc sub groups as necessary. Though, when it came to crunch time, we came together to crank things out on deadline day. Overall, we feel that this approach has been a success. Group members were responsible in timely completing their issues, and what needed to be done got done.

Limitations

There currently is no easy way for information to be added to the database beyond the Django console and the Django admin page. We feel that this is the biggest obstacle in our goal of an open democracy, because to securely edit the database means that the database can only be edited by trusted admins of the website, and this puts information that is important to many in the hands of the few. Furthermore, this is a cumbersome, manual process, and there is little check on human error. Still, this may not be the worst thing, as the Texas legislature only meets once every two years , and so a small dedicated team is able to timely update the database without compromising the integrity of the data by opening it up to insecure sources.

Delegation of Responsibilities

- Michael Dong – Was primarily responsible for technical report, wrote UML diagram, wrote SQL queries and wrote the page on which the results were displayed.

- Ian Hays – Worked on the initial API design and unit tests using Apiary for phase 1. For phase 2, API implementation and changes to the API and Django models. For phase 3, was project leader and worked on the API page and additional tests.

- Keyon Mohebzad - Implemented API with a combination of TastyPie and django serializers. Wrote unit tests for API and created an API page with a unique use of our API. Wrote search unit tests.

- Paul Strong - Modeling and database migrations, Votes Page, General Front-end work, Search Functionality, Picture uploading, Loading senator pictures from twitter.

- Eliya Shabanov - Worked on the initial API design and unit testing, front-end design, and collection of data and its input into the database. Also served as group leader in phase 2.

- Jimmy Kettler - Initial setup of the site, creating the project on github, setting up the framework for the displaying of pages using templates and dynamic data representation. Bug fixes, styling, twitter bootstrap feature such as typeahead and carousel. Installation and setup of south for database migrations, style guides for page styles. Bounced around a lot helping out with various problems as they came up.
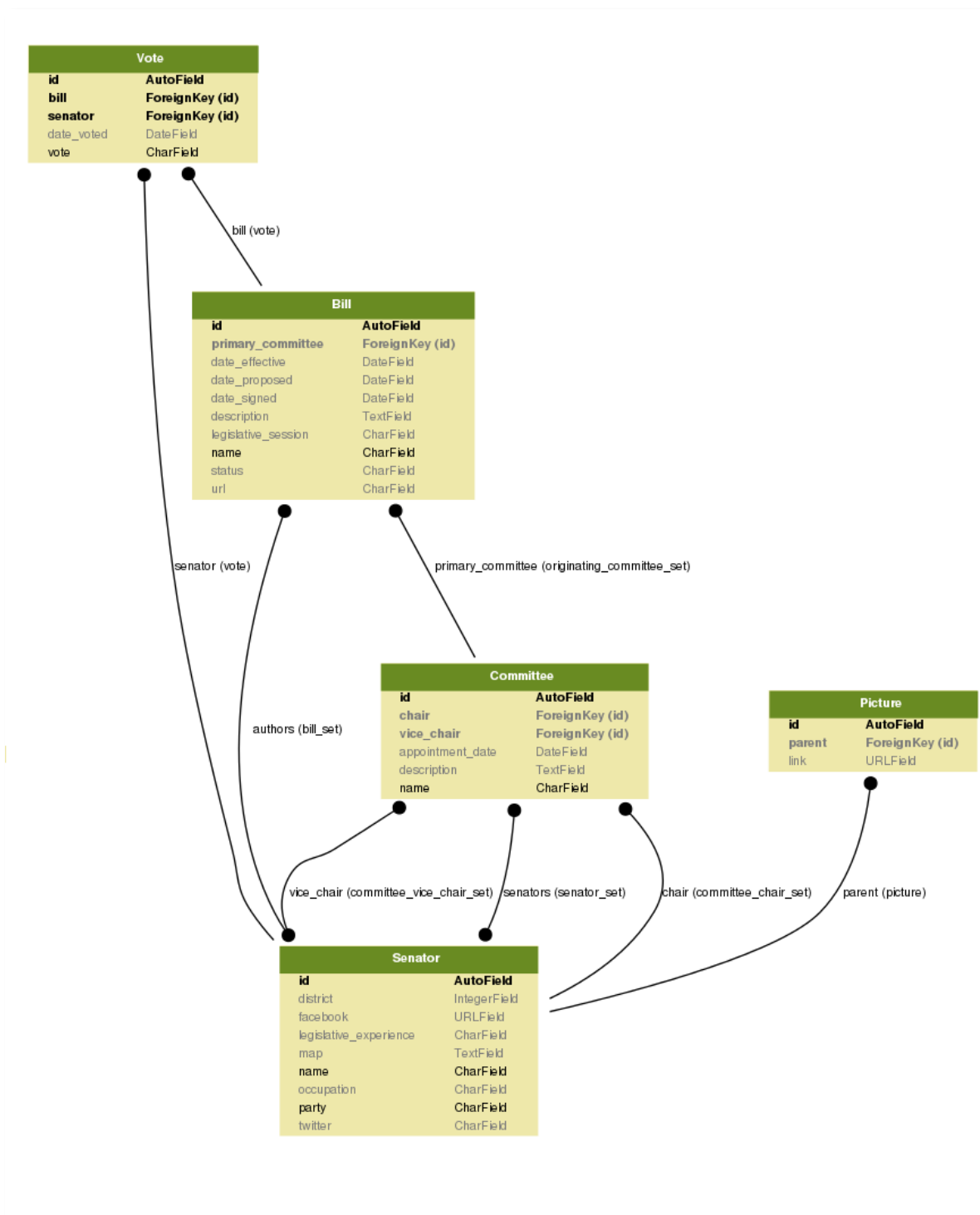
Backend Design
Design Overview

FIGURE 1: UML OVERVIEW OF MODELS – AUTO-GENERATED FROM MODELS.PY

Django Models
Senator Model

Senator	
id	**AutoField**
district	IntegerField
facebook	URLField
legislative_experience	CharField
map	TextField
name	**CharField**
occupation	CharField
party	**CharField**
twitter	CharField

The senator model models each of the 30 individual senators in the Texas Senate (1 seat is vacant).

These are the legislators in the Texas Senate who author, sponsor, and vote on bills, whether in

committees or on the floor. The primary key is an auto-generated integer ID supplied by Django.

Attributes include their name, district, party, occupation, legislative experience, a map of where their

office is located, pictures, and their Facebook and Twitter feeds if they have one.

Bill Model

Bill	
id	**AutoField**
primary_committee	**ForeignKey (id)**
date_effective	DateField
date_proposed	DateField
date_signed	DateField
description	TextField
legislative_session	CharField
name	**CharField**
status	CharField
url	CharField

The bill model models specific bills submitted to committees or the senate floor at large. These bills are

bills that have gone to a vote in committees or on the floor. As with senators, their primary key is an

auto-generated integer supplied by Django. Attributes include the name of the bill, the legislative

session in which the bill was proposed, the date it was proposed, and if passed, the date it was signed

into law and the date it was effective. Attributes also include its status, a link to its full text, and a brief description on the contents of the bill.

Committee Model

Committee	
id	**AutoField**
chair	ForeignKey (id)
vice_chair	ForeignKey (id)
appointment_date	DateField
description	TextField
name	CharField

The committee model models the 18 standing committees in the Texas Senate. Every bill must go through committee, where it is debated, amended, and voted on before it goes to the floor where it is voted on by the Senate as a whole. As with senators and bills, the primary key is an auto-generated integer supplied by Django. Attributes include its name, a description, and the date that the committee was first formed.

Vote Model

Vote	
id	**AutoField**
bill	**ForeignKey (id)**
senator	**ForeignKey (id)**
date_voted	DateField
vote	CharField

The vote model models the votes that senators cast for bills. Senators may vote Aye or Nay, or they may be present but not voting, or absent from the vote. We store the actual value of the vote itself as a tuple of strings, a 3 letter abbreviation for the value of the vote, and its full English value. We also store as an attribute the date of the vote.

Picture Model

The picture model is there to represent the one to many relationship between senators and pictures. Instead of having one picture link per senator, the picture model allows us to display a slideshow of pictures for each senator.

Senator and Bill Relationship

Bills are authored by Senators, and although there are co-authors, we have chosen to only include the main author, for simplicity's sake. A bill can have a long list of co-authors, some of which may not be senators, and thus, as an initial step, we have decided only to include main authors. Senators can author multiple bills. Therefore, we have a many to one relationship between bills and senators.

Senator and Committee Relationship

There are 18 standing committees in the Texas Senate. Each committee is chaired by a senator and vice chaired by a senator, though those positions may be vacant at any given time. Each committee also has some number of senators as members, usually less than 10. A senator may be part of any number of committees. Therefore, there exists a many to many relationship between senators and committees. Since there are multiple types of relationship between senators and committees: members, vice chairs, and chairs, a senator's membership of a committee is modeled by the three different sets a senator to committee relation can belong to: committee_chair_set, committee_vice_chair_set, and committee_senators_set.

Bill and Committee Relationship

Each bill must go through a committee. While a bill may go through multiple committees before going to the floor for a vote, and indeed, every bill must go through the calendar committee to be placed on the

floor for a vote, each bill mainly falls under the purview of one committee. This is the committee which

we have chosen to model. Each committee oversees many bills. Therefore, there exists a many to one

relationship between bills and committees.

RESTful API
GET
Example snippets of JSON requests and replies are shown below for the Senator class. Similar requests

and replies are analogous for the other classes.

/api/senators - GET

Gets all of the senators. Example:

```
+ Response 200 (application/json)


        [
            {
                "id": 1,
                "name": "Jane Nelson",
                "party": "Republican",
                "occupation": "Businesswoman, former teacher",
                "legistlative_experience": "Disaster Relief",
                "district": "12",
                "twitter": "https://twitter.com/SenJaneNelson",
                "facebook": "https://www.facebook.com/SenatorJaneNelson",
                "committees": [1,2]
            },
            {
                "id": 2,
                "name": "John Whitmire",
                "party": "Democratic",
                "occupation": "Attorney",
                "legistlative_experience": "",
                "district": "15",
                "twitter": "",
                "facebook": "",
                "committees": [2]
            }

        ]
```

/api/senators/{id} - GET

Gets one of the senators. Example:

```
+ Response 200 (application/json)
    + Body

        {
            "id": 1,
            "name": "Jane Nelson",
            "party": "Republican",
            "occupation": "Businesswoman, former teacher",
            "legistlative_experience": "Disaster Relief",
            "district": "12",
            "twitter": "https://twitter.com/SenJaneNelson",
            "facebook": "https://www.facebook.com/SenatorJaneNelson",
            "committees": [1,2]
        }
```

/api/senators/{id}/bills - GET

```
All of the Bills that this senator has authored
###List the bills [GET]
+ Response 200 (application/json)
        [
            {
                "id": 1,
                "name": "SB 63",
                "authors": [1],
                "legislative_session": "83(R)",
                "date_proposed": "11/12/2012",
                "date_signed": "6/14/2013",
                "date_effective": "6/14/2013",
                "status": "Signed into law",
                "url":
"http://www.legis.state.tx.us/BillLookup/History.aspx?LegSess=83R&Bill=SB63",
                "primary_committee": 1,
                "Description": "Relating to consent to the immunization of certain children.",
                "voters": [2, 3]
            }
        ]
```

Others

- /api/bills – GET – Gets a collection of all of the bills in the database

- /api/bills/{id} – GET – Gets a specific bill, where {id} is the identification key of the desired bill

- api/bills/{id}/senators – GET – Gets a list of senators and their corresponding votes for the bill with id {id}

- api/bills/{id}/authors – GET – Gets a list of the senators who authored the bill with id {id}

- /api/committees – GET – Gets a collection of all of the committees in the database

- /api/committees/{id} – GET – Gets a specific committee, where {id} is the identification key of

  the desired committee

- /api/committees/{id}/senators – GET – Get all the senators in the committee with id {id}

- /api/committees/{id}/bills – GET – Get all the bills originating in the committee with id {id}

PUT

/api/senators/{id} - PUSH

Update an existing senator

```
+ Request (application/json)


    {
        "id": 1,
        "name": "Jane Nelson",
        "party": "Republican",
        "occupation": "Businesswoman, former teacher",
        "legistlative_experience": "Disaster Relief",
        "district": "12",
        "twitter": "https://twitter.com/SenJaneNelson",
        "facebook": "https://www.facebook.com/SenatorJaneNelson",
        "picture": "none",
        "committees": [1,2]
    }
+ Response 204
```

Others
- /api/bills/{id} – PUT – Updates a specific bill, where {id} is the identification key of the desired
  bill
- /api/committees/{id} – PUT – Updates a specific committee, where {id} is the identification key
  of the desired committee

POST

/api/senators/{id} - POST

Create a new senator entry

```
+ Request (application/json)

    [
        {
            "name": "Jane Nelson",
            "party": "Republican",
            "occupation": "Businesswoman, former teacher",
            "legistlative_experience": "Disaster Relief",
            "district": "12",
            "twitter": "https://twitter.com/SenJaneNelson",
            "facebook": "https://www.facebook.com/SenatorJaneNelson",
            "committees": [1,2]
        }
    ]

+ Response 201 (application/json)

    {
        "id": 1
    }
```

Others

- /api/bills – POST – Adds a new bill to the collection of bills

- /api/committees – POST – Adds a new committee to the collection of committees

DELETE

/api/senators/{id} – DELETE

```
+ Response 204
```

Others

- /api/bills/{id} – DELETE – Delete a bill from the database, where {id} is the identification key of

  the bill to remove

- /api/committees/{id} – DELETE – Delete a committee from the database, where {id} is the

  identification key of the committee to remove

Search

There are several approaches to implementing full-text searching, including full-text search engines such as Spinx and Elastic search, but instead we opted for a simple multi-table full-text index. Watson(https://github.com/etianen/django-watson), a Django package, provides and manages the index for us. Given our parameters as to what models and fields to index it creates a table with a full-text index that we can search. The reason we chose Watson instead of some of the more full-fledged solutions is that it fit our search requirements, and did not come with the added burden of managing and setting up a search engine. It also comes with out-of-the-box integration with Django which made it very easy to integrate it with our application.

Watson is setup to index all the text fields of the Bill, Committee, and Senator models. It automatically updates the index when one of these three models are created, updated or deleted. The index consists of a single table `watson.searchentry` which contains the index data. Watson then uses PostgreSQL's full-text search functionality to find entries in this table and match them to the appropriate models.

Frontend Design
Overview

The driving idea behind the UI design of the Texas Law Database is clarity. We believe that clear, readily available information best serve our goals for not only a functional, usable website, but also serve our goal to presenting information to voters in an easily accessible way. We believe that an open democracy is best served when citizens have full access to information, and the best way to achieve that is to not hide information, but to display it in the open. To that end, we have, wherever possible, eschewed complicated menus and hierarchal displays of data and instead opted for a clear, concise view of information. Furthermore, we recognize that our website is a tool for the 21st century voter, and with that comes the expectation that our website looks modern, feels modern, and is usable on modern devices, such as phones and tablets. To that end, we have used Twitter Bootstrap as our front end

framework. Not only does Twitter Bootstrap provide a sleek, intuitive, powerful framework for our

frontend, it also provides features such as responsive CSS to allow our website to be used by a variety of

devices at a variety of resolutions. With these goals in mind, the following pages will attempt to provide

a user perspective on the frontend design of the Texas Law Database

Splash Page

FIGURE 2: DESKTOP SPLASH PAGE

The first thing a user sees when navigating to the Texas Law Database is the splash page. There, as

required, one can see a list of contributors to the project. More importantly, one can see the header,

which lists the categories for which the Texas Law Database provides information about, each of which

is represented by a Django model. This allows the user to easily and quickly select what they are

immediately interested in, allowing easy navigation to the site's full capabilities. This header is placed on

every page of the Texas Law Database, so that a user can return to the splash page or navigate to any category from any page.

Welcome to the Texas Law Database
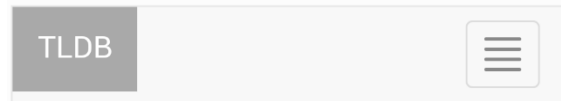
Welcome to the Texas Law Database
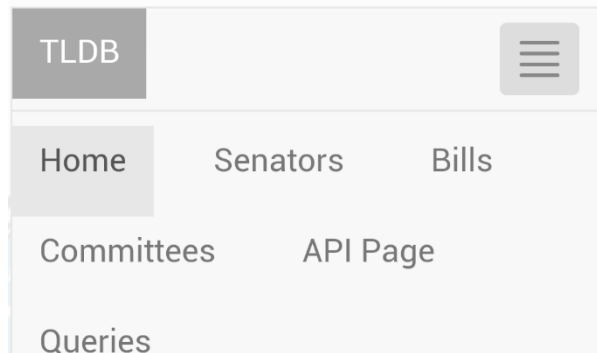
Here already, we can see the power of Twitter Bootstrap. The header is preserved in function on mobile devices, and we see that the menu collapses into a touchable element. In this way, mobile users can still have the full functionality of the header without sacrificing valuable mobile screen real estate.

Senators View
Overview Page
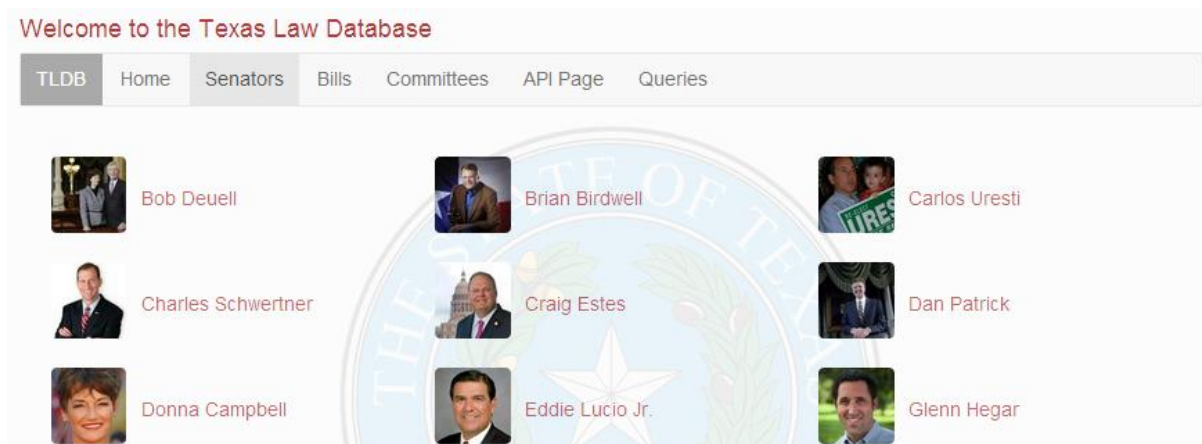
FIGURE 5: SENATOR OVERVIEW PAGE

Navigating to the senators page (at texaslawdb.heroku.com/senators or from clicking on the senators link in the header) brings us to the senator overview page. Here, we list the members of the Texas Senate of the current legislature and display their picture, taken from their Twitter profile. Not only is

this source of pictures programmatically easier, this allows voters to visually search for their senator in a way that they are already used to through twitter. This way, voters already familiar with their senator can easily, through recognition of familiar visual cues, navigate to the page of the senator they're interested in.

Individual Page

Navigating to an individual senator's page from the overview page brings us to this view. Here, we can see an individual senator's district, party, legislative experience, occupation, committee membership, and address. These attributes are all contained in the Django model, and provide valuable information to voters interested in a legislator. The committee membership in particular is modeled entirely through Django, and links to those committees from a senator's individual page will bring you to the individual page for the committee.

Underneath the senator's information, we have a carousel of photos for each senator. This allows voters to connect further with their legislators, as seeing them in different situations, ones not necessarily picked by the legislators themselves as in their Twitter pictures, allows voters to more deeply connect with their legislators. This is modeled by our photo model in the Django database.

FIGURE 8: SENATOR SOCIAL MEDIA VIEW

Scrolling down further brings us to the social media feeds of the legislators. We believe that there is no better tool in the 21st century for voters to interact with their legislators than through social media. An open social media platform allows voters to connect with legislators on an unprecedented basis. In

many cases, legislators themselves will run their own social media operations, allowing voters ready access to their legislators' thoughts and actions in a nearly unrestricted way. Since social media is so important to our goal of furthering an open-source democracy, we have given it great prominence on our senators' pages. We hope that voters can use our website to learn about their legislators' actions from us, and interact with them without having to leave our website. This way, we give users all the tools they need to interact with their legislators, and open up democracy beyond what information they can get from traditional media sources.

Bills

FIGURE 9: INDIVIDUAL BILL VIEW

From the bill view, voters can learn about individual bills. They can see the legislative session to which the bill belongs to, the primary committee through which the bill was passed, the status of the bill, dates regarding the bill, the author to the bill, and a link to the full text of the bill. Of these, the committee and author are modeled through Django, and clicking on those links will bring you to the appropriate page in the Texas Law Database. Furthermore, one can see a vote summary for the bill, which summarizes the votes, as well as displaying the date of the vote.

FIGURE 10: BILL VOTE BREAKDOWN

Clicking on this vote summary will display the vote breakdown pane, where one can see a detailed graphical presentation of the bill's votes. From here, one can click on the name of each of the senators to navigate to the senator's page in the Texas Law Database, where they can learn more detailed information for each senator. Each of these votes is modeled by the vote model, and each senator is modeled through the Senator model.

Committees
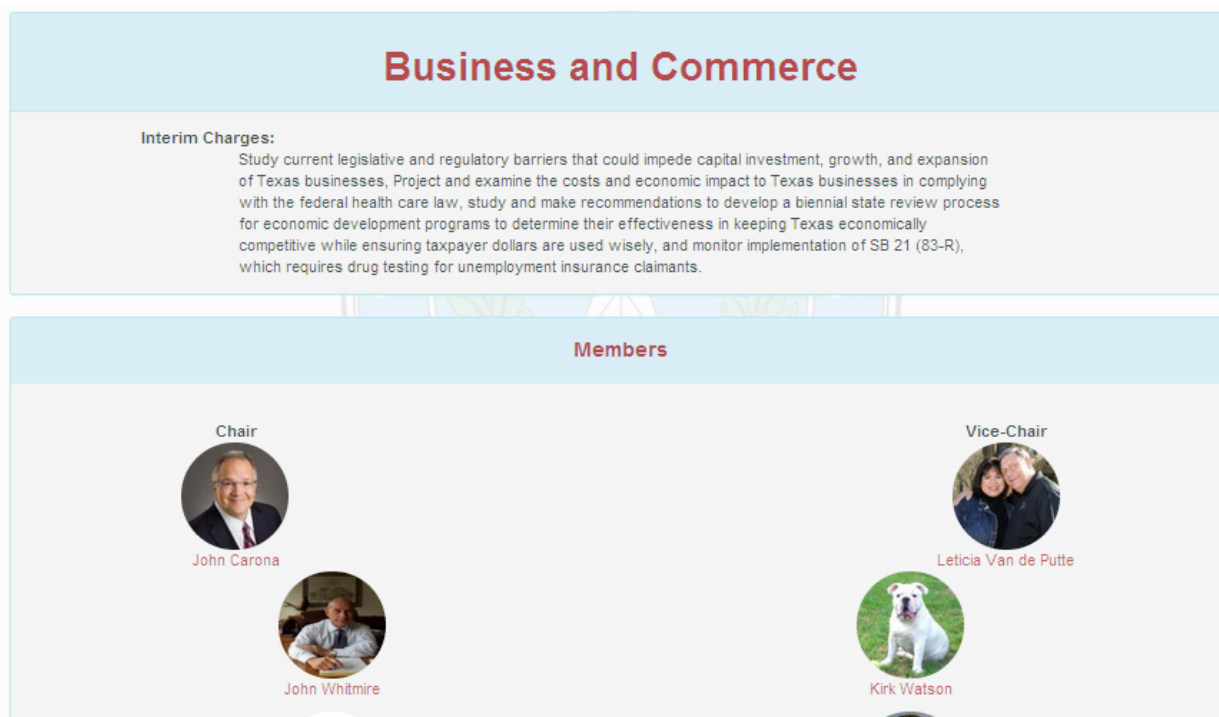
FIGURE 11: INDIVIDUAL COMMITTEE VIEW

Clicking on an individual committee will bring us to this view, where users can learn about the charge of

a committee, and see a list of its members. These members are all senators as modeled by our senators

model, and clicking on a link to a senator will bring us to the individual page for the senator.
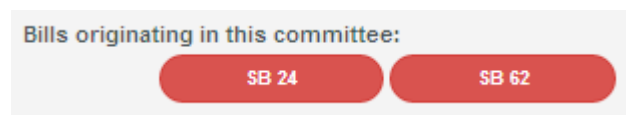
FIGURE 12: COMMITTEE BILL VIEW

Furthermore, for committees with associated bills
in the database, we display a list of bills in the
database that have gone through the committee.
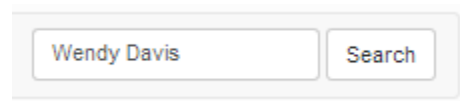
Here, users can click on the bills to navigate to the page for the bill in the Texas Law Database, where

they can learn more detailed information about the bill.

Search

Our search view is fairly simple and straightforward. Watson provides some search results views out of the box, but we found them to be inadequate for our site and rolled our own implementation. Watson is responsible for taking the query, performing the search, and returning a list of a result objects. We take those result objects and display them appropriately.

On the top of each page within the header there is a site-wide search box. The user is able to make a search at any time, and is then directed to a list of bills, committees and senators that match the user's query. The user can click on any of the results and be taken to the corresponding entity.

The display of each result varies based on its related entity. Our view in combination with Watson displays different templates based on the object type of the related entity. The different views are as follows:

FIGURE 14: SENATOR SEARCH RESULT VIEW

FIGURE 15: BILL SEARCH RESULT VIEW

FIGURE 16: COMMITTEE SEARCH RESULT VIEW

Implementation

Frameworks

- Django
- Twitter Bootstrap

Tools

- Git
- Github
- Heroku
- Heroku Postgres
- Cloudinary
- South
- Tastypie
- Watson

Datastore

- PostgreSQL

Tests

API Unit Tests

Unit tests for the API, found in cs373-idb/tests.py, use the Python unittest library to individually test each endpoint and the associated HTTP methods. The idea is to comprehensively test the RESTful API on the live Heroku server. To do this, at the start of each test method we open a connection to the app on Heroku which we close at the end of the method. In each test, the response status of the HTTP request is checked. In the methods that return a response body, the body is in byte form and must be decoded to a string and then loaded into JSON using the "json.loads" function.

In the tests for the POST and PUT HTTP methods, the variable "values" is used with the "json.dumps" function to convert a python list/dictionary into a JSON object to be passed to the client. For the POST method, the response body is a JSON object containing only the id of the new object. For the PUT method, there is no response body check.

In the tests for the GET HTTP methods, the variable "desired_body" contains the expected response body in the form of a python object that is checked against the response returned from the client.

In the tests for the DELETE HTTP methods, there is no "desired_body" or "values" variables as no JSON objects are passed to or from the client.

Django Unit Tests

Django unit tests, found at cs373-idb/idb/tests, aims to comprehensively test the capabilities of the Django model. For each of the models, we first set up a test case for the model by creating, using Django methods, a test member with the desired attributes and relationships. This simulates the way Django itself uses these methods to populate the database. We then use Django's methods to retrieve a member, and we check that the values in Django's database match what is expected. In this way, we are able to test Django's ability to add to and retrieve from the database.