Texas Votes Technical Report

Phase II

Motivation

Texas Votes was designed to refute the idea that one vote makes no difference in American democracy. Many resources exist which effectively track federal elections, but few exist dedicated to state politicians of Texas. Our website will shed light on the American bi-annual horse race — giving users demographic data, fundraising information, district electoral history, and even more. With a clean UI and a clear view of the upcoming election season, we hope to give people the information and the motivation to definitively cast their ballots on November 3.

1. What will my ballot look like? (ideally, a person coming here can look up their personalized ballot)
2. What is the current state of an election? (Is one candidate out-fundraising another? can the challenger win, given the way a district has voted in the past?)
3. What does this election really mean? (Who supports what candidates? Is this election one I can sit out?)

User stories

Phase I

Our User Stories for Customer Team
1. Resources to help
    a. Story: As a concerned citizen of the Earth, I would like to know more about how I can help. It would be really useful if there were resources on your page about how I can help these creatures so that I don't have to go searching for them myself.
2. Add developer images bios
    a. Story: As a user, I would like to see who my developers are and a bio about themselves
3. Conservation
    a. Story: As an environmentalist, I would like to be able to sort/filter animals by their conservation status.
4. Filters
    a. Story: As an outdoor person, I would like to filter animals by habitat to know where I would encounter endangered animals and find out how I can protect them.
5. Splash Page
    a. Story: As a user, I would like an appealing Splash page with various designs

Customer User Stories for Our Team

1. Healthcare Plans for Politicians
    a. Story: Elderly voters are concerned about the security of their healthcare and would like to be able to easily determine which politicians have which stance on healthcare.
    b. We will hardcode this information on the Politician Detail page for now but will need to find and add this data to our API.
    c. Estimated: 0:05
    d. Actual: 0:10
2. District Based on Address
    a. Story: New voters want to know what district they are located in to be able to understand how to vote. They would also like information about voting.

     b. We will implement this next phase when we create a functioning search bar for the model pages.

     c. Estimated: Unable to do during this phase

     d. Actual: Unable to do during this phase

3. Tax Rates for Politicians

     a. Story: Wealthy and Conservative voters would like to be able to find out politicians' stances on tax breaks to determine which politicians would provide them with the most tax breaks.process as they are inexperienced.

     b. We will hardcode this information on the Politician Detail page for now but will need to find and add this data to our API.

     c. Estimated: 0:05

     d. Actual: 0:10

4. Address to where to go vote

     a. Story: Voters who have just recently moved would like to know where they can vote based on their new location.

     b. We will need to find out how to get the polling locations, and if possible, will add this data to our API.

     c. Estimated: Unable to do during this phase

     d. Actual: Unable to do during this phase

5. When and Where to Vote by Address

     a. Story: Busy voters would like a quick and easy way to determine the best polling location and time to fit their busy schedules.

     b. We will add Early Voting dates to the Election Detail pages. We will need to find out how to get the polling locations, and if possible, will add this data to our API.

     c. Estimated: 0:10

     d. Actual: 0:10

Phase II

Our User Stories for Customer Team

1. Mobile Friendly

     a. Story: As an individual who possesses a device that can communicate with people from across the world and do many more functions (Hi Alvin), also known as a mobile phone, I am struggling to see many of the slideshows and tables since they do not fit on my screen. I think this

would make it better since it would make your application more accessible for more users.

2. Organization Logo
   a. Story: As a visual person, I would like to see a picture/logo for all of the organizations. I love saving the environment by donating to organizations that will help it. This would allow me to remember them better so I can contribute to them instead of forgetting!

3. Navigation Bar
   a. Story: As a developer, I am concerned about the scalability of listing each animal/organization/habitat under each tab in the navigation bar. I think that the tab should link directly to each model page. This would allow the tab to not be overcrowded and for us to be able to see all the different animals, organizations, or habitats on only the model pages.

4. Homepage Image Slideshow Issues
   a. Story: For the slideshow of images on the home page, the arrows to move left and right and the position bar underneath the animal pictures change position depending on the height of the image. As a perfectionist, this is bothersome.

5. About Page
   a. Story: As a product manager, I would definitely like to see how many commits each person made and how many issues each person has, and I would like to see this without going through a carousel. This would help me to efficiently check the team members' contributions and make it clear who is participating in the project.

6. React Production Mode
   a. Story: As a developer, I am concerned about the fact that your website is running the development build of React instead of the production build of React - this is because production mode obscures component information, which tightens security, and gets better performance. Please properly redeploy so we don't see warnings in the console.

Customer User Stories for Our Team

1. Commas for large numbers
   a. Story: This is a very simple issue, but as an elderly person, it's hard to interpret the numbers on your pages such as population count. Instead of displaying a number like 818807, it would make more sense to make it a comma-number such as 818,807 just to make it easier to read and

compare to other numbers. Hopefully this is an easy fix but will make your page significantly more user friendly!

    b. We implemented a function that converted integers to a string with the appropriate commas. We then were able to use this function in all of the places that had numbers on the page.

    c. Estimated: 0:05

    d. Actual: 0:05

2. Colors of party for politician grid view

    a. Story: As a very liberal Democrat, and I want to clearly see if the politicians that are on the politicians page are Republican or Democrat. I can barely tell right now because there is just a small D or R on their header, but if they were color coded red/blue, it would be much more clear. Thanks!

    b. We were able to do this by adding a colored border to each politician on the model page. This lets users easily determine what party each politician is at a first glance.

    c. Estimated: 0:05

    d. Actual: 0:05

3. Increased font size

    a. Story: I am an elderly person trying to use your website. As I am reading about the different politicians, I am finding a hard time reading because the font is so small. If you can bold key points on your page or make the font bigger, this would be much more user friendly. Thanks!

    b. We were able to fix this by manually going through our site and changing the font size.

    c. Estimated: 0:30

    d. Actual: 0:20

4. Filtering by party for politician

    a. Story: I am a conservative Republican, and I only care about Republican candidates. If on your politicians page, you made it possible to filter out the Democratic candidates and only show me Republican candidates, this would make it easier to look at information of people I actually care about.

    b. We will implement this next phase when we implement filtering

    c. Estimated: Unable to do during this phase

    d. Actual: Unable to do during this phase

5. Default sort election list view by date

a. Story: I am more interested in upcoming elections rather than previous elections. I am wondering if upcoming elections will show up on the elections page. I also think it makes sense to display the previous elections in chronological order to make it easier to find past election results.
b. We will implement this next phase when implement sorting
c. Estimated: Unable to do during this phase
d. Actual: Unable to do during this phase

RESTful API

The RESTful APIs we used were:
- https://developers.google.com/civic-information/docs/v2
- https://www.opensecrets.org/open-data/api
- https://v3.openstates.org/

Other sources we used to build the database are:
- https://www.ethics.state.tx.us/search/cf/
- https://www.sos.state.tx.us/elections/historical/elections-results-archive.shtml
- https://redistricting.capitol.texas.gov/Current-districts

Postman API link:
- https://documenter.getpostman.com/view/12817007/TVYC8zTg

We made a RESTful API as a preliminary backbone for our website with Postman. We defined paths for politicians, districts, and elections, and created models and schemas for them. These models and schemas did the initial querying and formatting of our database information. We performed further formatting after converting the data into JSON.

Models

The three models used for this website are politicians, districts, and elections. Each model has 10 attributes with some attributes being used to filter the data. Each model is tightly intertwined. A politician serves their constituents in a district and runs in an election. A district's demographics and electoral history can explain whether a politician can win or lose in an upcoming election. And an election decides which politician keeps or loses their job in their district.

Model 1: Politicians
- ID
- Name
- Incumbent
- Offices (what office do they currently hold or are running for)
- Party
- District
- Contact Information
    - Image
    - Social Media (Facebook, Twitter, etc)
    - Website
- Upcoming elections
- Fundraising
    - Raised
    - Spent
    - Remaining Cash
    - Contributors
    - Industries
    - Debt

Model 2: Districts:
- ID
- Open Civic Data (OCD) ID
- Type
- Party
- Counties

- Number
- Upcoming Elections
- Elected Officials
- Demographics
    - Total Population
    - Age
    - Race
    - Education
    - Income

Model 3: Elections:
- ID
- Type
- District
- Candidates
- Office
- Dates
    - Election Day
    - Early Voting Start
    - Early Voting End

Tools

We utilized React and Ant Design for the front-end development of our website and Postman for back-end API design and documentation.

We utilized Git flow to ensure proper development of features, minimization of merge conflicts, and to ensure that development and production deployments occur smoothly.

Docker and GitLab pipelines were used for the automatic deployment of our websites on target branches, as well as setting up development environments.

For developing the Python back-end, we used Flask and SQLAlchemy.

For the database, we used PostgreSQL hosted on AWS RDS, as well as pgAdmin for monitoring the state of the database.

For data collection, we drew from Google Civics API, Open Secrets API, Open States API, and Ballotpedia.org, and we also got data from CSVs on census.gov and the Texas Secretary of State website. Querying the data involved requests.py, PyQuery, and Pandas, and exporting the data involved making use of Pandas again as well as python's json library.

Hosting

We originally used AWS S3 with AWS CloudFront for our static website hosting, but switched over to AWS Amplify. AWS Amplify automatically includes a TLS/SSL certificate in the domain registration process, while AWS CloudFront requires a more hands-on approach for manual registration and verification (Amplify looks like it actually uses CloudFront for this as well). Amplify also has a built in feature to deploy separate stage and production websites, which was useful for QA and assuring that our application was properly working before fully deploying to production.

NameCheap was used to register and configure our domain name, and to verify ownership with AWS services in order to register a TLS/SSL certificate.

For our back-end API, we utilized AWS Elastic Beanstalk, and AWS RDS for our database hosting. We were able to split up our deployments for Elastic Beanstalk using GitLab's CI/CD, so that we could have independent development and production APIs. We also created two separate databases in our RDS instances for the same purpose.

Environment variables were used to connect the front-end, back-end, and databases in a manner that was both secure, and allowed proper linking between development and production units. This allows us to make changes across the entire stack in our development deployment without the possibility of adversely affecting our production deployment.

In order to bypass GitLab's 400 minute monthly quota for pipeline runner usage, we outsourced our GitLab runners onto a GCP Kubernetes cluster.

Pagination

Pagination is mostly handled in the API back end. We use the database models (explained more below) to create model schemas using Marshmallow. These schemas can be used to convert our model objects into JSON with only the desired information.

To get a page of a model's information, we query the database and call SQLAlchemy's paginate() function on the query which defaults to 20 records per page. We can then use the corresponding schema's dump() function to turn the query object's items into JSON. After that, we just format the JSON and pass it to the front end.

For the front-end, we utilized the pagination component provided by Ant Design, as well as some hooks to determine the current page. When the current page is changed, an API call for the new page is sent to the Flask server for the next page, the current page count was set as a hook, with a dependency for it in the useEffect() so that API calls for new pages were done at appropriate instances.

Database

In designing the database, we had to first decide what columns would be present in each table for all the models, what the relationships are between the models, and if we needed to break out any of our info into different tables.

The first part of deciding what columns would be needed for each table was relatively straightforward. We mainly referenced the already constructed Postman API and the information we had collected after programmatically scraping our APIs.

For the relationships, we decided that we would have one-to-many relationships between politicians to districts and elections to districts, and a many-to-many relationship between politicians to elections.

We also decided to break out counties into its own table, as there is a many-to-many relationship between counties and districts. This also allows a user to possibly query the districts they are a part of based on what county they live in.

We were able to construct a Python script using SQLAlchemy to convert all of our scraped data in its raw JSON form into SQLAlchemy models that were then pushed onto the database.

Testing

Front-end JavaScript testing used Jest as the testing framework. Enzyme was also used to help create assertions for React components. We were able to quite easily integrate the testing scripts into GitLab's CI/CD script.

Front-end GUI testing used Selenium as the testing framework. We were able to test navigation through the links displayed on the page, and confirm that pages from these links are valid. We found it much harder to get end-to-end testing on a local deployment in the GitLab CI/CD pipelines, so we chose to point the Selenium tester at our development and production deployments for now in our pipelines. We are able to run it on local deployments outside of the GitLab CI/CD environment.

Back-end unit tests were done through the built in Python unittests library. It was relatively easy to set up and get working in our GitLab CI/CD pipelines. We used the unittest examples provided in Collatz as a reference point in writing our tests.

Postman tests were done by refining our API design on Postman and downloading the JSON schema. We then pointed the JSON at our production and development API endpoints, and were able to run it inside our GitLab CI/CD pipelines. We might move this test after deployments of development and production, as this is more of an acceptance test to make sure that our deployments are working correctly.