PropXdoesWHAT

Chris Renard Dustin Huang Eder Garza Jae Lee
Kevin Kuney

1 Motivation

We want to help users become aware of laws that affect them personally, particularly those in traditionally underrepresented groups. Since many laws can have complex side effects, due to both complexity in their primary purpose and the unfortunately common practice of including unrelated changes as riders, it can be difficult to keep up with legislation being worked on or voted on that could affect your life. Marginalized groups especially can find changes that affect programs they may rely on buried in otherwise innocuous laws, or be unsure of what groups may be able to advocate for them or help them navigate changes. Our goal is to allow groups to see what recent and upcoming laws may affect them and from there find groups which can better inform them and provide routes to take action against those which would harm them.

2 User Stories

The following are the user stories gathered for Phase 1:

- **As a user, I want to be able to identify which groups are affected by a law.**
  We solved this issue by mapping the laws to the affected group and displaying them within each law model. In our API model, we set up a relationship between the laws and the affected groups. From there, we were able to call our API and display the mappings in a well-structured format using Bootstrap and ReactJS.

- **As a user, I want to see what congressmen, senators, city-council, etc. are supporting a law.**
  We solved this issue by mapping the laws to the corresponding politician and

displaying them within each law model. In our API model, we set up a relationship between the laws and the policitians involved. From there, we were able to call our API and display the mappings in a well-structured format using Bootstrap and ReactJS.

- **As a user, I want to see where the politicians lie on the political spectrum.**
  We solved this issue by grabbing the politician's political affiliation from our API database and structuring the information using Bootstrap and ReactJS.

- **As a user, I want to see the implications of a law.**
  We solved this issue by having a description for each law. This was done by querying the description parameter data from our law API and displaying it onto the appropriate model page.

- **As a user, I want to know how I can contact my senator/representative.**
  We solved this issue by displaying each politician's bio, which includes their website and number. This was done by querying the relevant contact information parameters from our API and displaying it onto our website with front-end tools, such as Bootstrap and ReactJS.

The following are the user stories gathered for Phase 2:

- **As a user, I would like to see information about what the website does on the homepage.**
  We solved this issue by displaying the purpose of the website in the homepage.

- **As a user, I would like to see more laws, politicians associated with these laws, action groups, and affected groups.**
  We resolved this user story by having the front-end gather the data from the API and displaying it. Our models are linked with foreign keys creating relationships with each other. Within our JavaScript code, we displayed the necessary data which now displays more laws, politicians associated with these laws, action groups, and affected groups.

- **As a user, I would like each image on the front page carousel to lead me to a certain page.**
  We solved this issue by making each image a link to different pages.

- **As a user, I would like an easy way to contact a certain action group.**

We solved this issue by displaying the necessary contact information of a certain action group on the model page. This was done through querying the data from our API and displaying it to the website with front-end tools such as Bootstrap and ReactJS.

- **As a user, I would like the affected group's website pages to have a little more info (do a little more than just list laws affecting group and action groups associated)**
  We solved this issue by displaying more than just laws and action groups related to this affected group. From our API model for affected groups, we queried the data and displayed it to the model web page for affected groups with front-end tools such as Bootstrap and ReactJS.

The following are the user stories gathered for Phase 3:

- **As a user, for the laws model, I want to be able to filter by subject.**
  We implemented this user story by creating a filter option that has options of the subjects to select.

- **As a user, for the politicians model, I want to be able to filter by Legislator's state.**
  We implemented this user story by creating a filter option that has options of the Legislator's state to select.

- **As a user, I would like to search for laws, politicians, and action groups from the home page.**
  We implemented a search bar that will utilize full-text searching. We also have searching within our models that will take into account any filters applied.

- **As a user, for the politicians model, I want to be able to filter by republican or democrat.**
  We implemented this user story by creating a filter option that has options of the politicians's party to select.

- **As a user, for the laws model, I want to be able to sort by date introduced**
  We implemented this user story by creating a sort option that sorts by ascending or descending for date introduced.

3 Models

Each politician model shows each law and action group associated with them. Each law shows each politician and action group associated with them. Each affected group shows each politician and law associated with them. These associations lead to their respective model page. Each action group shows each law that affects the group.

Laws
- Name
- Title
- Description
- Sponsor
- Affected Groups (incoming table relation) (not built)
- Primary Subject
- Congress.gov Link
- GovTrack Link
- Date Introduced
- Date of last vote (if occurred)
- Date bill passed Senate (if occurred)
- Date bill passed House (if occurred)
- Date bill was vetoed (if occurred)
- Date bill was enacted (if occurred)

Politicians
- First Name
- Last Name
- State
- Party Affiliation
- Chamber (House or Senate)
- Phone
- Link to official site
- Link to contact page (if present)

Affected Groups
- Name
- Description

Action Groups
- Name
- Description

- Type/Category
- Link to official site
- Assisted groups (incoming table relation) (not built)
- Statements on laws (incoming table relation, text) (not built)

4 RESTful API

These are the API links we scraped from to fill our database.

http://docs.openstates.org/en/latest/api/
http://projects.propublica.org/api-docs/congress-api/
http://opensecrets.org/open-data/api

API Endpoints:

- api.propxdoeswhat.me/api/politicians
- api.propxdoeswhat.me/api/affected_groups
- api.propxdoeswhat.me/api/action_groups
- api.propxdoeswhat.me/api/laws

Full API Documentation at: https://documenter.getpostman.com/view/4704075/RWMCtUm6

5 Tools

GitLab
We used GitLab as our Git repository hosting to collaborate on our IDB project. Gitlab is easy to use because it has issue tracking and continuous integration.

Postman
Postman is a useful tool for testing,developing, and documenting our RESTful API. With Postman, building our API model for our website was easier. Postman also has a documentation feature for our API model. This makes things understandable for anyone who wishes to use our API.

Grammarly
Spelling/grammar feedback for this report

Piazza
Collecting User Stories from end users. Giving feedback and receiving criticism made easier with Piazza.

Slack
Slack is very useful in keeping up with team members and git commits. Slack is integrated with the GitLab repo.

Flask
Flask was used for our back-end framework for PropXdoesWHAT! In the Flask app, we used SQL-Alchemy to connect to our database and constructing our table models. Flask Restless was used create our API endpoints.

React
React was used to create a dynamic website by utilizing re-usable components. This made it easier to make our models, and add features such as filtering, sorting, and searching.

AWS
Server hosting for Flash, Node, MySQL, ElasticSearch instances (More Description in AWS section)

Selenium
Test GUI

React Router
React Router allowed for routing from URL on page to scripts. This allowed our front end to run without the need for flask.

npm
Node Package manager

Mocha
Test framework for JavaScript

Babel
JavaScript compiler that

Chai
Assertion library that paired with JavaScript testing framework

Docker
We used docker to contain our front-end (React) and back-end (Flask). This made it easier to isolate and work on each end without worrying about any interference.

Enzyme
Tests React

JavaScript
Programming language used to interact within web browsers

Bootstrap
Front-end framework used to style and design our website. Bootstrap made it easier to develop our front-end to be more appealing.

Python
Programming language used to code our back-end Flask App

ElasticSearch
Standalone full-text index server based on Lucene

6 Hosting

6.1 Back End

The back end is hosted on an AWS EC2 instance. The main file is "Back-end/app.py" and it is running in a docker container on the server. The back end utilizes flask, sqlalchemy and flask-restless to create an API. The API is hosted on "api.propxdoeswhat.me/api/".

6.2 Front End

The front end is also hosted on an AWS EC2 instance. The main file is "Front-end/app.py" and it is running in a docker container on the server. We used React and Bootstap to build the front end. The front end gets data from our back end and serves it to the client. The front end is hosted on "propxdoeswhat.me".

7 AWS

First, we launched an EC2 instance with Amazon Linux AMI(Amazon Machine Image). Since by default, all incoming ports are blocked, we added security group rules that allow incoming SSH and HTTP requests from anywhere. Next, we SSH'd into the EC2 instance by using the private key file we were given as we launched the instance, the username EC2-user, and the public IP address of the instance.

Frontend access:
`ssh -i front-end-private-key.pem ec2-user@propxdoeswhat.me`
Backend access:
`ssh -i back-end-private-key.pem ec2-user@api.propxdoeswhat.me`

Next, we updated the working Linux server running in the AWS cloud and installed docker on it. Then, we transferred our local files onto the server using FileZilla's SSH File Transfer Protocol. Finally, we built our Docker image and ran our web application on our Docker container.

8 Full-text Search

We use an ElasticSearch instance running on AWS as an external server to host the full-text indexes and perform searches. Indexes are built after database tables are loaded by scripts which dump the tables and load the indexes. Currently, fields are indexed as-is with no stemming or other prossessing. The Flask-Restless API manager was modified to first recognize a new parameter `search` (using a request pre-processor handler), call the ElasticSearch instance with the given query, and then adjust the query filter for the rest of that request to only include results with the returned ids (using a custom `query` model field).

9 Filter

The website implements filtering options for each of our models. The laws models filters based on the API parameter of subject, the politicians models filters based on party,state,and chamber, and the action group model filters by the API parameter "type". Whenever a user clicks on a filter on our page, we GET a JSON response and parse it. After parsing it, we generate the respective model with the applied filters. If another filter gets applied from the same filter category, we OR the two

filters and display the results. If another filter gets applied from a different filter category, we AND the two categories and display the results.

10 Sort

For the models, we allow the users to sort the displayed results by ascending or descending order.This is done by utilizing the API calls of *order_by*. In the Laws model, the laws are sorted by the date introduced. In the Politicians model, the politicians are sorted by name.In the Action Groups model, the action groups are sorted by type. The sort component is implemented in the same way as filter by GET of JSON response and parsing it.

11 Database

We use a MySQL 5.6 instance running on Amazon RDS. The Amazon default settings use `latin1` for text (this is bad), we had to manually adjust them to use `utf8mb4` and rebuild tables. Note: `utf8` in MySQL does not meet the utf8 spec, as it will not work with astral glyphs (four-byte characters). While this is unlikely to occur in our dataset, for future-proofing we use `utf8mb4` which correctly implements the standard.

Table: `'laws'`

- `'id' INT UNSIGNED NOT NULL AUTO_INCREMENT` – internal id, auto-generated
- `'bill_id' VARCHAR(32) NOT NULL` – unique bill name across legislatures, ie: hr5515-115
- `'name' VARCHAR(32) NOT NULL` – name of bill, ie: H.R.5515
- `'title' VARCHAR(256) NOT NULL` – short title of bill
- `'subject' VARCHAR(256) NOT NULL` – primary subject of bill
- `'sponsor_id' INT UNSIGNED NOT NULL` – foreign key to 'politicians'
- `'sponsor_bio_id' VARCHAR(16) NOT NULL` – Biography of Congress unique id for sponsor, ie: T000238
- `'cdotgov_url' VARCHAR(256),` – url of bill on congress.gov
- `'govtrack_url' VARCHAR(256),` – url of bill on govtrack
- `'introduced' DATE`
- `'last_vote' DATE`
- `'house_pass' DATE`

- `'senate_pass' DATE`
- `'enacted' DATE`
- `'vetoed' DATE`
- `'desc' TEXT` – propublica summary
- `'raw' TEXT` – raw json from source for this entry

Table: `'politicians'`

- `'id' INT UNSIGNED NOT NULL AUTO_INCREMENT` – internal id, auto-generated
- `'first_name' VARCHAR(64) NOT NULL`
- `'last_name' VARCHAR(64) NOT NULL`
- `'dob' DATE NOT NULL` – date of birth
- `'bio_id' VARCHAR(16) NOT NULL` – Biography of Congress unique id, ie: T000238
- `'chamber' ENUM('house', 'senate') NOT NULL`
- `'state' CHAR(2) NOT NULL`
- `'party' ENUM('R', 'D', 'I') NOT NULL`
- `'site' VARCHAR(256)` – url for official site
- `'contact_form' VARCHAR(256)` – url for official contact form
- `'phone' VARCHAR(32)`
- `'raw' TEXT` – raw json from source for this entry

Table: `'action_groups'`

- `'id' INT UNSIGNED NOT NULL AUTO_INCREMENT` – internal id, auto-generated
- `'name' VARCHAR(256) NOT NULL`
- `'url' VARCHAR(256) NOT NULL` – url of official site
- `'type' VARCHAR(256) NOT NULL` – startguide.org category of group
- `'desc' TEXT`

12 Pagination

For our models, the data in our database contains a large number of instances of laws, politicians, action groups, and affected groups. To give the user a better experience navigating through all of our data, we implemented a traditional pagination number system. Our API paginates all the data. Using React components and Bootstrap, we created functionality to the buttons. Our pagination number system contains a previous page, next page, page items, and a page footer.

13 Testing

We are using Selenium Python to test the GUI(Graphical User Interface) of our website. We installed Selenium with pip and downloaded the required interface driver for the Firefox browser, geckodriver, and then added geckodriver.exe in the system PATH. This let us use instances of Firefox WebDriver in our Python unit test to navigate around our website and test our GUI.

Since we are using Flask-Restless and SQLAlchemy to create our RESTFul API, we did not test our back-end because those tools handle it.

We are using Postman for API testing. The Postman test requests GET methods to our API and we make sure that all the fields we need in the database are present and have correct number of pages.

We are using Mocha, Chai, and Enzyme to test our javascript. We installed Mocha, Chai, Enzyme, babel-preset-env, babel-core, babel-preset-react, and react in npm and ran mocha with arguments –require babel-core/register.

14 Obstacles

In Phase 3 of our IDB project we ran into a huge problem. When we were trying to implement filter, sort and search we had an issue with our code because of the way we setup our Front-end in phase I and phase II. Because of that we had to scrap all of our front-end code we developed in the first two phases and refactor for phase 3. Although phase 3 was suppose to be more lightweight on the amount of work needed to be done, that was not the case for us. We had to turn in our project a day late to make sure we had a working and well developed website.