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.

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: documenter.getpostman.com/view/4704075/RWEmKHJp

5 Tools

- Git — Version Control
- GitLab — Git repository hosting, issue tracking
- Postman — API design and testing
- Grammarly — Spelling/grammar feedback for this report
- Piazza — Collecting User Stories from end users
- Slack — Team communication and collaboration
- Flask —
- React —
- AWS — Server
- Selenium —
- React Router —
- npm —
- Mocha —
- Babel —
- Chai —
- Docker —
- Enzyme —
- JavaScript —
- Bootstrap —
- Python —

6 Setup

HOW WE SETUP TOOLS

7 Hosting

7.1 Database

Our database is hosted on Amazon's Relational Database service. We used MySQL to design and populate our database.

7.2 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/".

7.3 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".

8 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.

9 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`

10 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.

11 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.