Cole Newby
Nicolas Benavides Levin
Manuel Trevino
Kavi Thiagarajan

**Episkop: Final Report**

**Summary of Episkop Project:**

Episkop is a web application designed to improve online polling systems by implementing a dynamic polling process. Current polling platforms, like Google Surveys, are mostly separated into three parts: poll creation, response collection, and data analysis. While this format is useful, it is not entirely adaptable. For example, surveyors and respondents cannot modify polls to their liking during response collection. In general, the phases of the current online polling process seem very separated. The stakeholders of Episkop – Phillip Ritchey, Cole Newby, Nicolas Benavides Levin, Manuel Trevino and Kavi Thiagarajan – felt it necessary to change this.

The resulting product, Episkop, meets this goal by taking the three parts of the online polling format and forming them into one, fluid system. The platform allows users to construct and share polls that are modifiable at any time. Response data can be beautifully displayed in a variety of graphical formats. Furthermore, responses are collected and displayed dynamically, refreshing constantly in front of the user. An API for the application is also supported, with a fully fledged API key request system. Episkop is a successful implementation of an improved polling process.

**Team Roles:**

- Manuel Trevino: Team Leader, Scrum Master
- Nicolas Benavides Levin: Project Owner
- Cole Newby: Software Engineer
- Kavi Thiagarajan: Software Engineer

**User Stories:**

- **Create an account** (score: 3), implemented
    - As a user I want to be able to create an Episkop account so that I can login into the platform
    - 3 points were allotted to this user story because it required research third party oauth authentications apis, such as the google login, and integrating that into the project

- **Sign into account (**Score: 2), Implemented
    - As a user I want to be able to sign into an existing account.
    - 2 points were allotted to this user story because it only requiring grabbing the user information provided by the google oauth and creating a session variable for the user

- **Navigate to Home Page** (Score: 1), Implemented
    - As a user I want to be able to navigate to the homepage after logging in
    - 1 point were allocated to the user story because it only required redirecting the user to the homepage

- **Create new polls** (score: 4), implemented
    - As a surveyor I want to create polls with customizable options so I can receive feedback on my particular topic or question.
    - 4 points were allotted to this user story because creating polls is a foundational aspect of this product. This user story thus included defining what a poll was, how a user would interact with it, and implementing the design.

- **Send out polls for respondents through email** (score: 3), partially implemented
    - As a surveyor I want to send my respondents polls through email so they can access the poll to submit feedback.
    - 3 points were allotted to this user story because we needed to research mailers and their implementation. We did not have experience with constructing a mailing service, as well. Furthermore, this task was different from other tasks, as it is not related to the CRUD everything else is built upon.
    - While we initially implemented this feature

- **View poll** (score: 2), implemented
    - As a respondent I want to be taken to the poll answering form so that I can input my responses for the poll.
    - 2 points were allotted to this user story because, at this point, we would already have the framework for polls objects. This task would only consist of designing a front end, implementing methods to edit certain CRUD objects, and defining view access permissions.

- **Submit response** (score: 3), implemented
    - As a respondent I want to be able to click the submit button so that my responses are added to the poll's results.
    - 3 points were allotted to this user story because all that had to be done to submit a response was to iterate through each of the questions submitted, creating new poll votes for each of the questions and saving them on the database.

- **Edit response** (score: 2), implemented
    - As a respondent I want to be able to go back to the poll answering form so that I can change my responses and resubmit them.
    - 2 points were allotted to this user story because, at this point, all that had

to be done was graph all the poll votes for a particular poll and user and set the default values in the form to the previous submission of the user

- **Add additional responses to polls** (score: 3), not implemented
    - As a respondent I want to add additional response options to polls so that I can deliver new positions to a given question or topic.
    - This user story was allotted 3 points because it required modifying an existing CRUD object (a new poll question type). Many access cases would also need to be accounted for, such as whether or not a poll was opened to be modified.
    - We are confident that this user story could have been implemented in an efficient amount of time. We simply decided to focus on improving other user stories, as this story would affect previously implemented and future tasks.

- **See previously created polls** (score: 2), implemented
    - As a surveyor I want to be able to see my history of created polls so that I can see the results and download the data.
    - 2 points were allotted to this user story because it required a simple partial to be created to display and poll, and simply query the unique polls a user has participated through their poll votes.

- **Pull API data from my poll collection** (score: 4), implemented
    - As a software developer I want to be able to make a RESTful request for a poll collection of mine and receive a JSON response so I can utilize Episkop data in my product.
    - 4 points were allotted to this user story as this task required extensive research in an area that was unrelated to the rest of the project. This user story also required us to use third party apps for testing purposes. Testing for this was also a very lengthy and meticulous process.

- **Present QR code to allow respondents to join poll** (score: 2), implemented
    - As a surveyor I want to be able to show a QR code for my poll so that respondents nearby can scan the code and respond to my poll.
    - 2 points were allotted to this user story because it required finding a way to create a qr code and then render into onto the DOM

- **Create Folders** (score: 4), implemented
    - As a surveyor I want to be able to group my created polls so that I can properly organize my data.
    - 4 points were allotted to this user story because it required creating the directory model, which would have an association among itself and also polls. Furthermore, the backend had to be modified to have a root directory and support changing and loading different directories

- **Add poll to folder** (score: 1), implemented
    - As a surveyor I want to be able to add a poll to an existing folder so I can organize my data.
    - 1 point was allotted to this user story because it simply required keeping track of what directory the poll was creating under and storying that when saving the poll

- **Move poll from folder to folder** (score: 3), not implemented
    - As a surveyor I want to be able to move polls from an existing folder to another existing folder to reorganize my data.
    - 3 points were allotted to this user story because moving polls requires complex front end and user interaction design. Specifically, there are many dependencies on how a user can move polls within their main directory.
    - We did not implement this user story because it was not necessary to construct a quality product. Specifically, one can “move” a poll by creating a new poll and destroying the old poll. Implementing this user story would be very time consuming and accomplish something that can technically be done already.

- **See participated (active) polls** (score: 3), implemented
    - As a respondent I want to be able to see the history of active polls I have participated in so that I can edit my responses.
    - This user story was allotted 2 points because it only required displaying polls while filtering for user ID and opened status attributes in our database.

- **See participated (expired) polls** (score: 2), implemented
    - As a respondent I want to be able to see the history of expired polls I have participated in so I can look at HTML results.
    - Similar to the previous user story, this user story was allotted 2 points because it only required displaying polls while filtering for user ID and opened status attributes in our database.

- **View active poll results** (score: 5), implemented
    - As a surveyor I want to be able to view the live results of active polls so I can see it being updated.
    - This user story was allotted 5 points because it required researching a library to use to render the graph, a new model used to store the type of graphs, a lot of javascript code, and a lot of front-end partials and views, and the codebase had to be restructured to use esbuild instead of importamaps.

- **Download poll data** (score: 3), implemented
    - As a surveyor I want to be able to download data from my polls so I can use them for other applications or view the results on my local machine.

    - 3 points were allotted to this user story because it required researching functions to convert our database data into downloadable formats. While this concept is similar to data formatting for HTML and JSON purposes, converting data to other formats was new to our team.

- **Create a poll form** (score: 2), implemented
    - As a surveyor I want to create a poll form to allow respondents to respond to polls.
    - This user story was allotted 2 points because it simply required piecing together other user stories, such as View a poll and Create a poll. Routing and front end was the main focus of this task.

- **Use URL invitations** (score: 2), implemented
    - As a respondent I want to use a custom URL invite to access polls that I want to respond to.
    - This user story was allotted 2 points because it only required research on randomly created URLs and small front end additions.

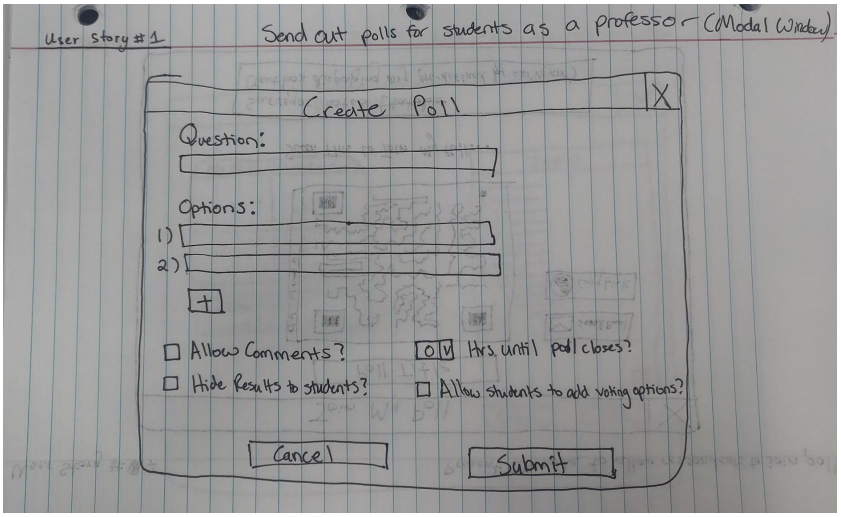
- **Administrate API Keys** (score: 3), implemented
    - As an admin I want to inspect and respond to API Key requests.
    - This user story was allotted 3 points because it required constructing a new CRUD object, modifying other CRUD objects, and setting up a variety of conditional statements. Not much new research would be required, so this user story was deemed a moderately difficult task.

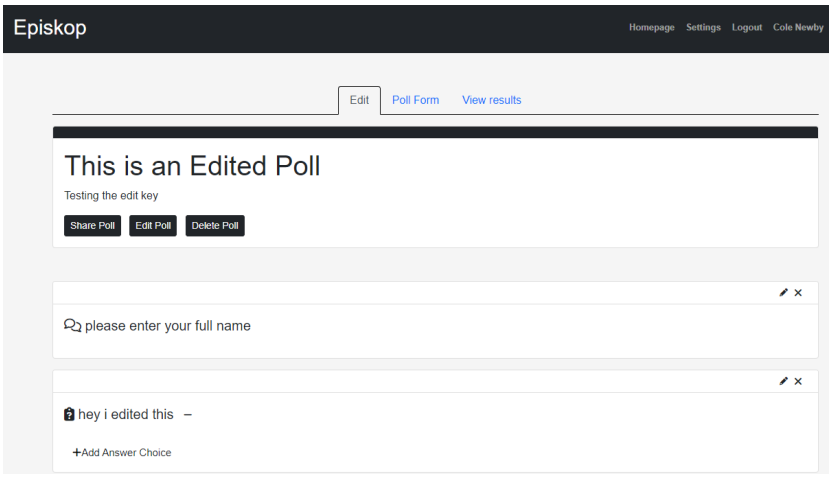
- **Remaining CRUD file directory** (score: 3), implemented
    - As a user I want to Create, Read, Update, and Destroy folders and polls within the file directory
    - 3 points were allotted to this question because a javascript context menu was used to display the renaming and deletion of folders and confirmations modals were implemented to prevent the user from accidentally deleting non-empty folders

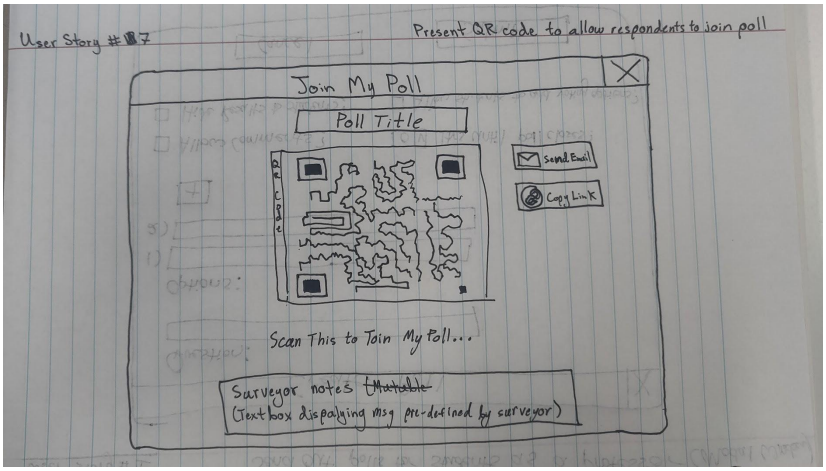
- **Edit the poll/include settings page** (score: 3), implemented
    - As a surveyor I want to go to a settings page to edit my poll specifications.
    - 3 points were allotted to this question because it required many front-end modifications and four types of restrictions were created, non published polls, the poll closed by date / show results / allow resubmits

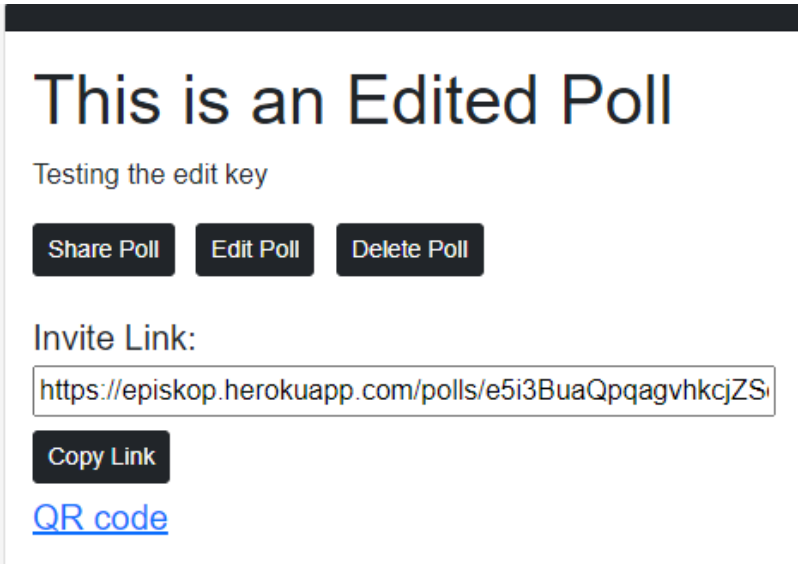
- **Anonymous answer choices** (score: 1), implemented
    - As a surveyor I want to be able to choose a setting that makes responses anonymous.
    - 1 point was allotted because it simply required adding an anonymous attribute to a poll vote which would be set to the current anonymity of a poll.

## Lo-Fi Mockups & Storyboards

- Poll Creation and Editing
  - 

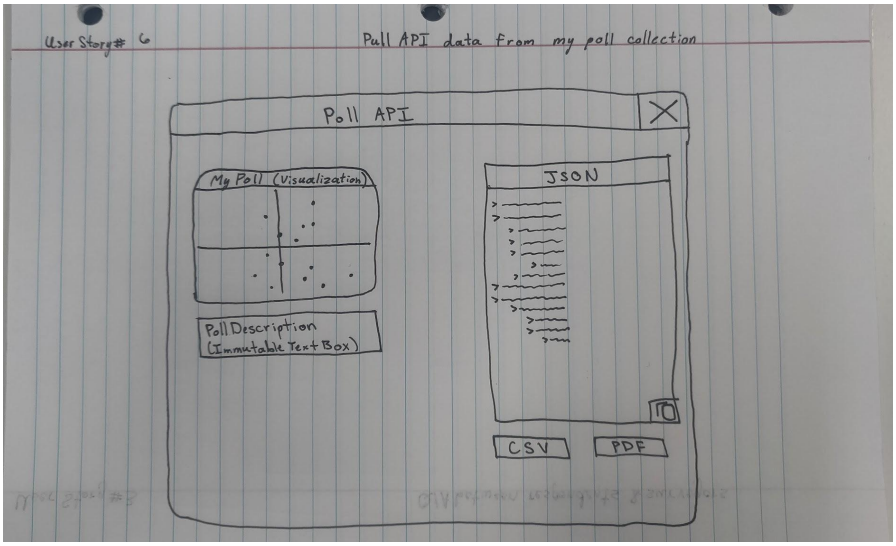
  - 

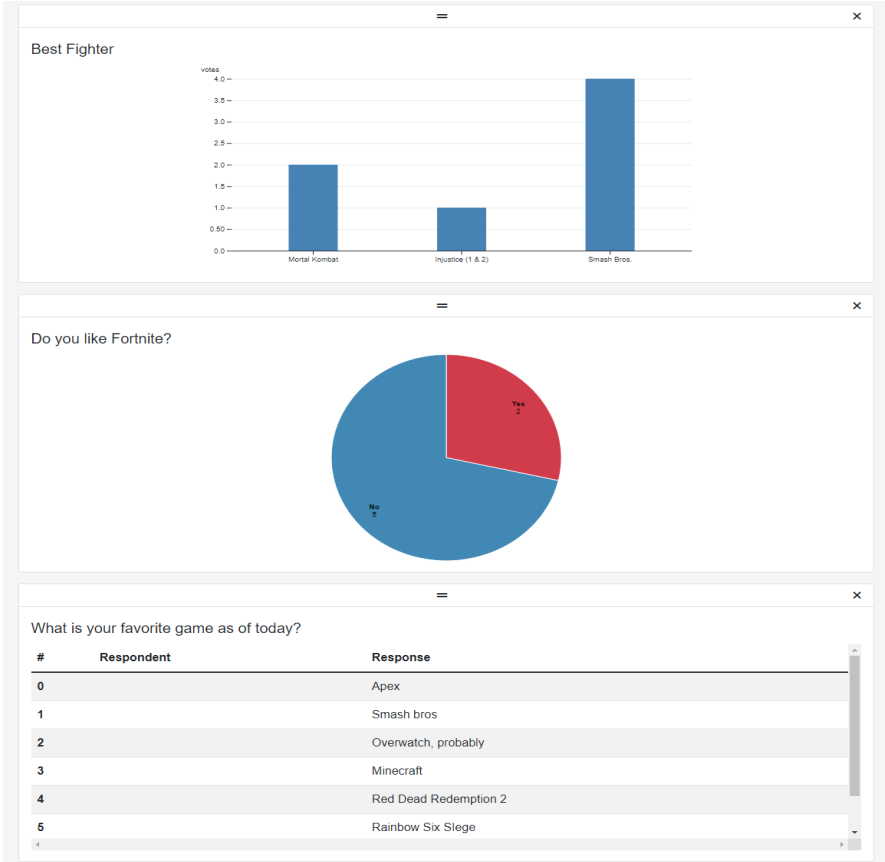
- Sharing a Poll
  - 

- 

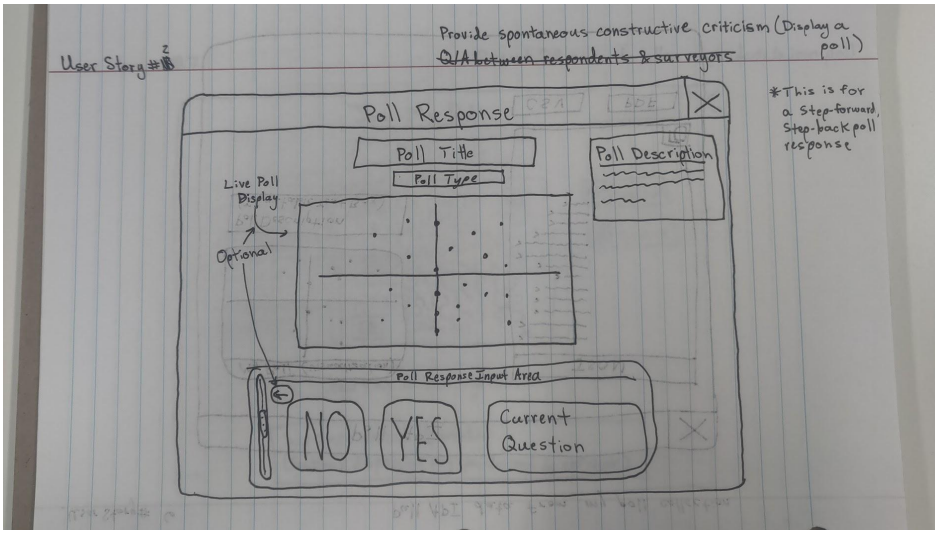
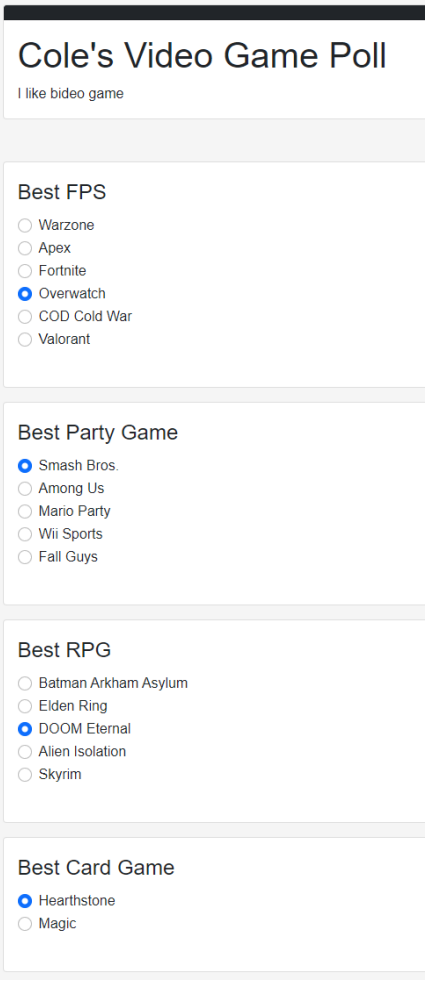
- 

- Viewing Poll Data

- 

- 


- Responding to a Poll

- 
User Story #2
Provide spontaneous constructive criticism (Display a poll)
Q/A between respondents & surveyors
Poll Response
*This is for a Step-forward, Step-back poll response
Poll Title
Poll Type
Poll Description
Live Poll Display
Optional
Poll Response Input Area
NO
YES
Current Question
- 
Cole's Video Game Poll
I like bideo game
Best FPS
Warzone
Apex
Fortnite
Overwatch
COD Cold War
Valorant
Best Party Game
Smash Bros.
Among Us
Mario Party
Wii Sports
Fall Guys
Best RPG
Batman Arkham Asylum
Elden Ring
DOOM Eternal
Alien Isolation
Skyrim
Best Card Game
Hearthstone
Magic

**Iterations:**

- Iteration 0 (Points completed - 0) :
    - For this iteration, we defined the scope of our project as well as the features we wanted to implement. Since this was the planning stage for our project, no code or user stories were implemented.
- Iteration 1 (Points completed - 6):
    - In this iteration, we created the landing page for our project and deployed it to Heroku. We also implemented Google OAuth 2.0 so a user can login to our application using their Google account.
- Iteration 2 (Points completed - 8):
    - In this iteration we added the CRUD functionality to create polls, create poll forms for submission, and generate URL invitation links for the poll forms.
- Iteration 3 (Points completed - 17) :
    - For this iteration we implemented two primary features: directories and email invitations. We also implemented the ability to respond to polls and be able to see these updates live.
- Iteration 4 (Points completed - 15) a lot of rework was done, which was not part of user stories:
    - For this iteration, we redesigned a lot of our front-end structure and made major improvements to the overall style and presentation of the website. Instead of having multiple different HTML pages, we utilized AJAX calls to insert views into the DOM, giving the user a seamless experience while using the app.
    - Second, we created an API for our app, allowing users to access all of the functionality of Episkop through API requests. On top of this, we constructed an administration system so Episkop admins can respond to key requests.
    - Lastly, we added the results feature. The results page displays all of the votes collected in a neat way for the user. Each question is created alongside a graph. However, the user has full control of what graphs to use in each question. There is an add graph button which allows the user to create graphs based on which question they select. Furthermore, there are two special types of graphs for "yes/no" type questions where the graph shows where a respondent stands in regard to a set of questions.
- Iteration 5 (Points - 13):
    - For this iteration, we first added the functionality of downloading results for a question in JSON or CSV format. We also added a settings page for each poll so that a user can utilize and update a variety of options for their poll. Furthermore, we added the ability to rename and delete folders

within our file directory system. Finally, we further improved upon our front-end by touching up typos.

**Customer Meetings:**

- Iteration 0 Meeting: 12/30/2021 @ 9:30 AM
    - In this meeting we presented our idea for Episkop with the client and received feedback on wanted features and changes to our UI Mockups.
- Iteration 1 Meeting: 1/4/2022 @ 2:30 PM
    - During this meeting, the general goals and preliminary thoughts for Episkop were created. It was determined that deploying a basic landing page and allowing a user to create an account to login to our platform would be our goals for Iteration 2.
- Iteration 2 Meeting: 2/3/2022 @ 2:45 PM
    - The purpose of this meeting was to show our progress after Iteration 2. We discussed the user stories that were implemented during Iteration 2, and the client verified the completion of the user stories. Furthermore, we went over the rspec tests for the application, and verified the planned user stories for Iteration 3 with the client.
- Iteration 3 Meeting: 3/10/2022 @ 2:45 PM
    - The purpose of this meeting was to show our progress after Iteration 3. We discussed our implementation of two primary features: directories and email invitations. We also demoed live responses to polls. Finally, we discussed our goals for the next iteration, which includes appending features and pages, as well as changing CSS and page layout.
- Iteration 4 Meeting: 4/21/2022 @ 2:45 PM
    - The purpose of this meeting was to show our progress after Iteration 4. We discussed our implementation of three primary features: a functional API, front end restructure, and our poll results feature. We also demoed live responses to polls to demonstrate that our results graphs update live without refreshing the page. Finally, we discussed our goals for the next iteration, which includes downloading poll results, creating a poll question that users can add options to, allow polls to be anonymous, and closing polls based on time.
- Iteration 5 Meeting: 5/4/2022 @ 2:00 PM
    - The purpose of this meeting was to show our progress after Iteration 5. For this iteration, we showed improvements for the poll API as well as demonstrated our new settings page, anonymous polling, and poll results downloading. We discussed small front-end improvements and ways we could demonstrate our application during the final demonstration. Overall,

this iteration was a success and resulted in the completion of Episkop for Spring 2022.

**BDD/TDD Process:**

We were unable to get Cucumber to work with our application due to versioning constraints in Cucumber. After researching the issue and experimenting with Cucumber, we determined that Cucumber has an issue with the latest version of rails and is not usable. Our team decided that it would be best to carry on without the use of Cucumber rather than rework our entire codebase, as we came to this realization around iteration 3. We did achieve behavior driven testing by using rspec features combined with Capybara instead. This was very useful in the first 3 iterations, as we needed to test routing with our Goggle-based login functionality. We were able to ensure that our login system was 100% functional using this testing method.

**Configuration Management Approach:**

We utilized a spike to define our CRUD organization for polls, poll questions, and answer choices. Specifically, we implemented the CRUD for polls and set up a basic, CSS-less front end to display a poll and its questions. This allowed us to easily identify mistakes in our object design, as well as gave us insight into where we should begin when implementing new features.

In total, our project had 22 branches. These branches mainly consisted of feature implementations, but we also created branches to work on small tasks like CSS and HTML fixups. We deployed our code a total of five times.

**Heroku Deployment Process:**

Our team did not experience any major issues when deploying our product to Heroku. The only issue we encountered was a CSS issue, in which some of our icons would not load properly after deployment and one of the entry point sass files was not being preproccessed on heroku, but that was fixed by adding another script.

**AWS & GitHub Issues:**

In order to have a common environment for development, we decided to use AWS EC2. Initially, we ran into major connectivity issues due to our use of the free version of EC2 while having a poor internet connection in Greece. We ended up purchasing a more powerful EC2 instance in order to have better performance and connectivity during our development of Episkop.

Any issues regarding GitHub were user error and easily solvable.

**Other Tools:**

Postman was utilized to assist in the implementation and testing of our API. The application allowed for us to test a variety of keys, test individual queries and accessibility responses, and finalize our API documentation.

**API Documentation:**

Documentation for our API syntax and key structure is included in our Git repo: https://github.com/manueltr/Episkop/blob/main/Episkop%20API%20Documentation.pdf

**Codebase description and structure:**

## Javascript

The Episkop app is a javascript heavy application. By default, rails generates a new project using the importmaps bundler, a way of importing all the javascript files into one entry point file. Instead, due to troubles importing the library used for the results page, the episkop codebase is configured using esbuild, an extremely fast JavaScript Bundler. There are two main files that act as entry points in the code base: “application.js” and “graph.js”. The entry point “Application.js” contains all the event handlers used for the functionality of the website, which are further broken down into their own feature files under the **app/javascript/features** folder. The entry point “Graph.js” contains all the code to support the chart/results feature of the project. The codebase uses the **D3** library to create the charts https://d3js.org/, which are then attached onto the DOM through javascript.The logic for that is contained in “graphs.js” under the function load_graphs(). Depending on the graph type, a switch case is used to render the correct graph.

## Front-End

The front-end framework used for the styling and structure of the project is **bootstrap 5**. The majority html elements in the partial and html files use bootstrap classes. Furthermore, there is unique css styling used throughout the project. The codebased is configured to use **Sass**, a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets. There are two main entry point Sass files in the code base: “application.bootstrap.scss” and “logged.bootstrap.scss”, both under app/assets/stylesheets. The file “application.bootstrap.scss” is the entry point for the styling used in the homepage. “Logged.bootstrap.scss” is the entry point for the styling used in the remainder of the application.

## Back-End

The majority of the site is rendered through javascript request to the server, which then returns a javascript snippet that is used to render the html partials into the DOM (.js.erb files). The server also supports many JSON requests, which are again used to render information onto the website (.json.jbuilder files).

## API

Our API relies on the rails framework, with the api-key controller and the poll, question and answer controllers checking for API keys and rendering appropriate JSON responses for queries. Whereas our site utilizes session data, defined when a user logs in, to render poll data, the API only uses data from a user's request (specifically the API key value and the body of the request). The app's application controller is responsible for differentiating between an API request and a session request. This is accomplished by running a function to check for a key value titled ApiKey in the request header. The app's poll, question and answer controllers are responsible for filtering the database data for valid requests and for denying invalid requests. This is done via JSON renders and nested conditional checks.

**Deployment Instructions:**

## EC2 Setup

The team worked on an AWS EC2 instance for the majority of the project's development, using VSCode to directly connect and work on the codebase using an SSH Key. The following script gets you up and running within minutes on a new EC2 instance or linux machine:

```
#Ruby installation
sudo apt update
sudo apt install git curl libssl-dev libreadline-dev zlib1g-dev autoconf bison
build-essential libyaml-dev libreadline-dev libncurses5-dev libffi-dev libgdbm-dev
curl -fsSL https://github.com/rbenv/rbenv-installer/raw/HEAD/bin/rbenv-installer |
bash
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
source ~/.bashrc
type rbenv
git clone https://github.com/rbenv/ruby-build.git
PREFIX=/usr/local sudo ./ruby-build/install.sh
rbenv install -l
rbenv install 3.0.3
```

```
rbenv global 3.0.3

# Gem installations
echo "gem: --no-document" > ~/.gemrc
gem install bundler
gem env home

#Heroku
sudo snap install --classic heroku
```

Run each line one by one, do not run the entire script at once.

**PostgreSQL Setup**

In order to set up the PostgreSQL database you must first configure the database.yml file under the config folder appropriately as such using our example:

```
default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see Rails configuration guide
  # https://guides.rubyonrails.org/configuring.html#database-pooling
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>

development:
  <<: *default
  database: MyDatabase
  host: localhost
  username: postgres
  password: episkop
test: &test
  <<: *default
  database: test
  host: localhost
  username: postgres
  password: episkop
production:
  <<: *default
  database: Episkop_production
  username: Episkop
  password: <%= ENV["EPISKOP_DATABASE_PASSWORD"] %>

cucumber:
```

```
  <<: *test
```

Then in your terminal you should run the following set of commands to create your database:

```
#Postgresql setup
sudo apt install libpq-dev postgresql

sudo -u postgres psql
create database MyDatabase;
alter user postgres with encrypted password 'MyPassword(can be anything)';
\q
```

**Heroku**

Once you have the repository cloned, deploying to heroku is surprisingly easy. Simply login into your heroku account and do the following:

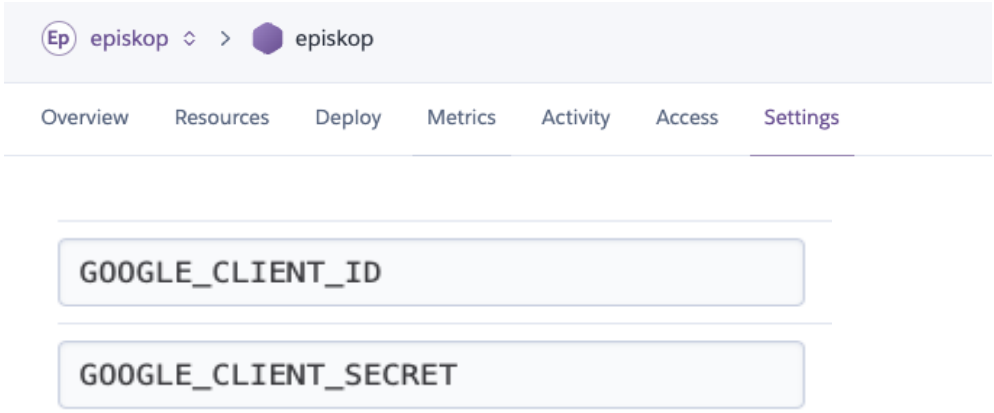
```
#Heroku Setup
 heroku login
 heroku git:remote -a example-app
 git push heroku main
```

Heroku takes care of the rest, it installs the gems, bundles the project, and runs the scripts specified in the package.json file. The three important ones are below:

```
 "scripts": {

   "build": "esbuild app/javascript/*.* --bundle --outdir=app/assets/builds",

   "build:css1": "sass ./app/assets/stylesheets/application.bootstrap.scss
./app/assets/builds/application.css --no-source-map --load-path=node_modules",

   "build:css2": "sass ./app/assets/stylesheets/logged.bootstrap.scss
./app/assets/builds/logged.css --no-source-map --load-path=node_modules"

 },
```

“Build”: runs esbuild and bundles all the imported files specified in the two entry point files (application.js) and (graph.js). And “build:css1” and “build:css” run sass which preprocesses and bundles the css files imported in the two files: “application.bootstrap.scss” and “logged.bootstrap.scss”

Lastly, you have to define the following ENVIRONMENT variables through the heroku site under the apps settings tab.



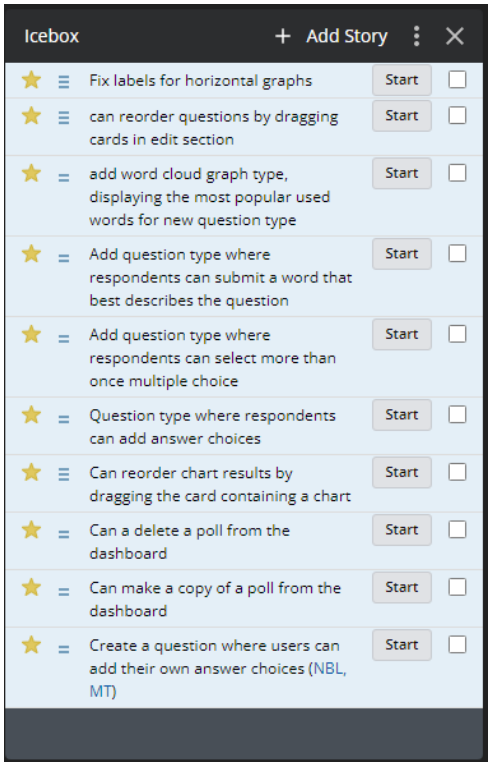

Our suggestion is to create your own set of keys through https://developers.google.com/identity/protocols/oauth2. The current keys used for the project will be included in an encrypted file on the repository, Professor Ritchey will have the key to decrypt the file.

## IMPORTANT FOR RUNNING EPISKOP LOCALLY

The package manager used for the codebase is **yarn.** Install yarn and run "**yarn install**" while in the root folder in order to install all of the javascript dependencies.

In order to run all the scripts in the package.json (i.e. run the project locally), use the command **bin/dev** while in the root folder

## Icebox of User Stories for Future Team




**Pivotal Tracker:**
https://www.pivotaltracker.com/n/projects/2547060

**GitHub Repo:**
https://github.com/manueltr/Episkop

**Heroku Deployment:**
https://episkop.herokuapp.com/

**Presentation & Demo Link:**
https://youtu.be/_P892s7a3u8