MIT DormPoll: Full Design (Revised)

Team Peanutbutter: Darius Bopp, Ron Dentiger, Nick Guo, Claudia Wu

---

Overview - Darius (Revised)

Description

The website will consist of a few different pages: a poll selection page, a page to answer a Poll, an Admin Portal page, **a page to add and delete other admins**, a page to create Polls, and a page to see results. ~~To facilitate this, the system will also need 2 databases: People and Results.~~ To facilitate this, the system will use multiple tables, requiring the admin-users to load in their own People table. This people table should include a roster table of the students living in a particular Dorm. **Only people in this table would be able to vote.** In order to work with the system, this database will need to have names, kerberoses, MIT ID's, and a column determining Admin status. The database could also include room number for extra functionality. ~~The Results database will consist of a table for every Poll, where results will be stored.~~ **Among many other tables to get the application to work, the other table useful for admin-users is the Results table, where admins can see saved result data.**

Purposes

- Be an easy-to-access way for students to send preferences and information to student government.
    - Dorm Student Governments ask for information from their constituents in three main ways: Voting, Surveys, and Guest Lists. It is vital that for any student, these interactions are easy-to-access as they serve has important way for students to have input or control in their Dorm community.
    - MIT DormPoll aims to meet this purpose by being online, and building off of MIT's OpenID. This way, all any student needs to do to access a Poll is navigate to the website while having a certificate installed, no account creation required.
- Be an easy-to-use way for students to send preferences and information to student government.

    - Along with being easy-to-access, this vital service also needs to be easy-to-use for similar reasons.
    - This purpose will be reached through a simple, and streamlined user experience. Once navigating to the website, they will see a page listing all current Polls. The user can then select one and fill it out.
- Be an easy-to-use (and easy-to-learn) way for student governments to send out these Polls and see / collect the information received
    - While the system needs to be easy-to-use for those voting, it also needs to be easy-to-use for those putting on the Polls. For the obvious reason of just being good software, it is especially important for MIT DormPoll. New student officials are elected every year, so the system will need to be able to be picked up easily for new admin.
    - This will be achieved through an Admin Portal built right into the site. This section of the website can only be reached by those given the Role of Admin, and is the part of the website where Polls are created and managed, and results can be observed. Similar to the rest of the website, the Admin Portal will work towards being simple and streamlined.
- Be flexible enough to match each Dorm's needs
    - Dorms at MIT are very unique, from both the rest of the world, and each other. Each of them have different culture, communities, people, and simply just a different way of doing things. That being said, they all do hold elections, but may require different styles of Polls or questions.
    - The Poll Creation part of the website will be a very open-ended tool for the administrators. A Poll will consist of Questions and Answers. The creator will make the text for the questions, and choose what answer-type they would like the Answer to be, for each question separately. It is worth noting that due to this requirement to be flexible, the system gets added functionality for free beyond just voting. Polls could be created to look for feedback or input, and for some dorms, be a way to get guest list information.

Issues With Current Solutions

At MIT, every Dorm has their own system for running elections, and not all of them online. Since there are in total nine dorms, we will only be discussing the problems with systems found at Maseeh and East Campus.

East Campus holds in-person elections. On a specified date, a polling box will be open for around 10 hours. Each hall in East Campus is required to man the poll box for a 1 hour shift. During this 10 hour window, students can come to the polling box and submit their vote. When they do, the people manning the box at that time will mark down that that student has voted. After the voting period is over, all the votes are collected and counted, and winners are announced.

This system, while it runs smoothly, has some subtle issues. First, from a voter's perspective, this system is not fully accessible. If you are not available to go to East Campus during the 10 hour window, you simply are unable to vote. From an administrator point of view, this system requires a lot of work and man-hours. It takes 20 man-hours for two people watch over the poll box at any given point. It also takes a non-negligible amount of time to hand count all the votes.

These problems can be solved with the digitalization of the system. An online voting platform would be accessible to any MIT student, and could be kept open longer without demanding more man power. Determining the result of an election can be done near-instantly through computation. However, an online system does not just magically fix everything, it is just as important that the online system is implemented well.

To exemplify this, we will look at Maseeh, which uses an online system. In this online system, students simply navigate to the Maseeh website, logging in with their certificate, chose a Poll to answer, fill out the poll and hit submit. From the user perspective, this is as easy as it gets. However, the backend is another story.

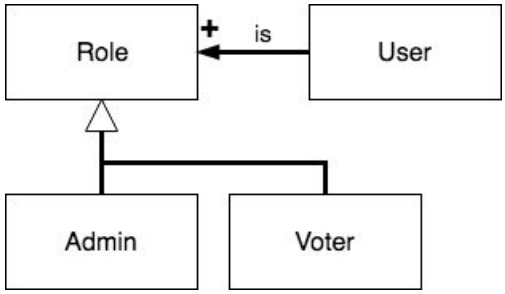
In order to facilitate this, the website grabs data from a few different databases and files. In one table, named Elections, each entry holds some amount of meta-data, specifically the name of the Poll, the file name of a specific Python file for this Poll, a table name to put the results into, and a start time and end time. The website then looks at this table to determine when to run a Poll, and what the content would be.

The content is stored in the python file names in the table. In this python file, html is written in string within an object structure. There is also conditional functions in the python file to determine who sees what part of the Poll, and special flags to mark what kind of Poll it is. Whenever an entirely new Poll needs to be made, a new Python file must be created, and a new entry in the Elections table. To update an existing Poll with new information, the text in a python file must be replaced, and reformatted with html in mind.

While Maseeh's system is great for the average user, for the admins of the polls, it is a headache. Creating or updating any poll requires swimming around in undocumented code and manually formatting plain text into html. Not only is this unwieldy to use, it is also difficult to learn for the new administors every year. We aim to keep the great voter usability of the Maseeh system while greatly improving the admin side.

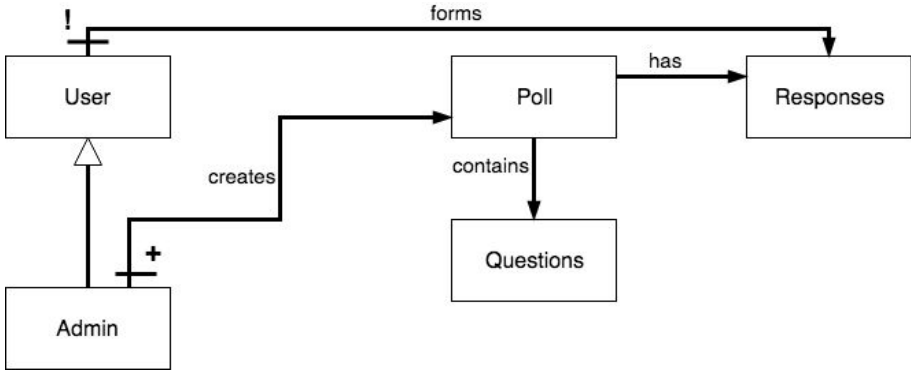
Conceptual Design - Darius

Role

- Purpose: To separate permission across users
- Structure:
    - 

    - Textual Constraints:
        - All Users are Votes
- Behaviors:
    - Note: Some initial Admin will need to be hard-coded
    - createRole()
        - Requires:
            - The User doing this action is an Admin
        - Effects:
            - That Role now exists
    - giveRole(User: U, Role: R)
        - Requires:
            - The User doing this action is an Admin
            - R not in U.is
        - Effects:
            - U.is := U.is+R
    - takeRole(User: U, Role: R)
        - Requires:
            - The User doing this action is an Admin
            - R in U.is
        - Effects
            - U.is :- U.is - R
    - deleteRole(User: U, Role: R)
        - Requires:
            - The User doing this action is an Admin
            - R to exist
        - Effects
            - For all U where R is in U.is, U.is := U.is - R
- Operational Principle:
    - A Student Government wants only certain Users to have the ability to create Polls and see the results. They create and give out the Admin Role to certain Users.

Poll

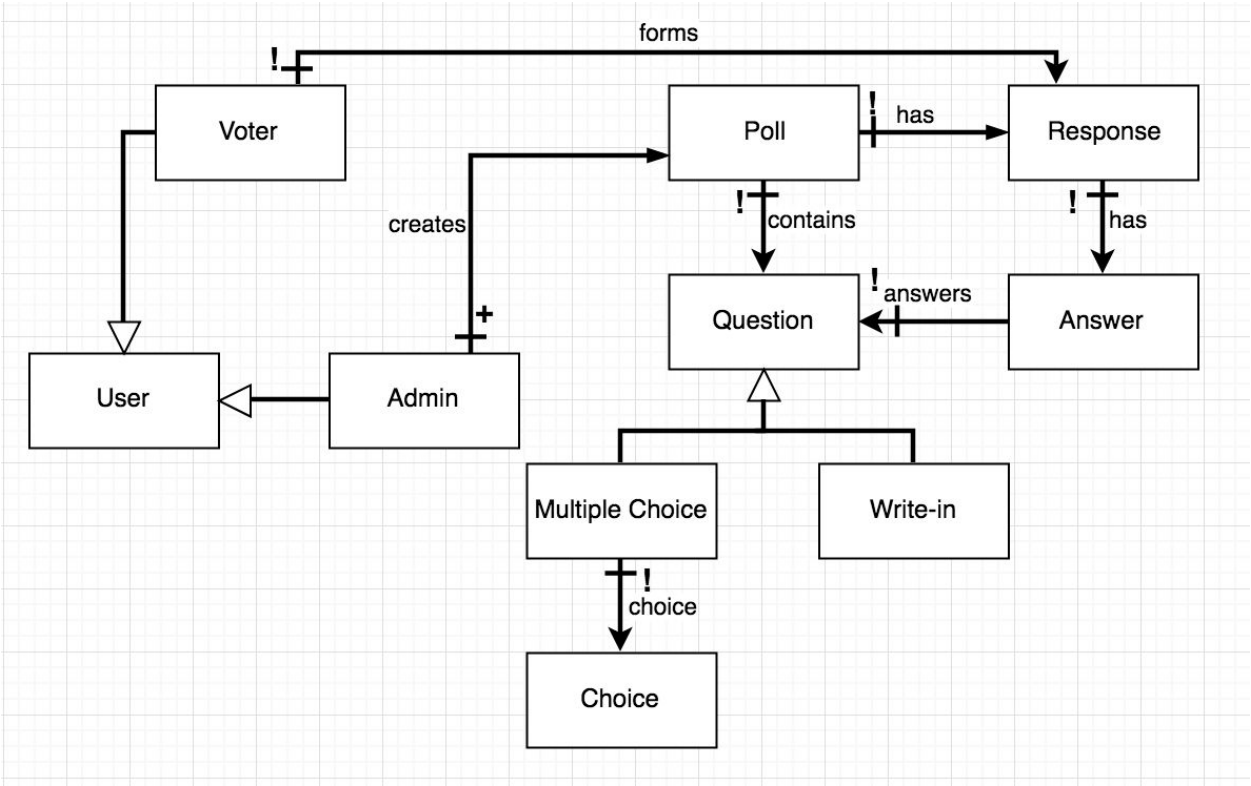
- Purpose: To collect opinions or preferences from students (users)
- Structure:
  - 

  - Textual Constraints:
    - A user may only vote in a poll once.
    - A poll has one response for each user who voted in that poll
      - Ie, there exists exactly one R in Poll P.has that is also in User U.forms.
- Behaviors
  - createPoll(Admin: A, Poll: P)
    - Requires:
      - a to be the admin creating the poll
    - Effects:
      - A.creates := A.creates + P
  - addQuestion(Poll: P, Question Q)
    - Requires:
      - Only Admin can do this
      - Q not in P.contains
    - Effects:
      - P.contains := P.contains + Q
  - deleteQuestion(Poll: P, Question Q)
    - Requires:
      - Only Admin can do this
      - Q in P.contains
    - Effects:
      - P.contains := P.contains - Q
  - editQuestion(Poll: P, Question Q)
    - Requires:
      - Only Admin can do this
      - Q in P.contains
    - Effects:
      - Edit the question to have different properties
  - deletePoll(Admin: A, Poll: P)
    - Requires:
      - any Admin can delete any poll
      - P is in A.creates
    - Effects:
      - A.creates := A.creates - P

    - voteInPoll(User: U, Poll: P, Response: R)
        - Requires:
            - R to be representative of U's Response
            - P not in U.votes
        - Effects:
            - U.votes := U.votes + P
            - P.has := P.has + R
    - getResponses(Admin: A, Poll P)
        - Requires:
            - Any Admin can get the results from any Poll
        - Effects:
            - Returns P.has to A for viewing
- Operational Principle
    - A Student Government (Admins) need to run an election for the upcoming year, and therefore need to collect students' (Users) votes (Responses). To do this, they createPoll() to make a Poll for students to vote in. The Admin can then addQuestion() to create the kind of Poll they want.
    - A student wants the Student Government to know their opinion or preference (Response) in the upcoming elections. They see the Student Government has created a Poll, and the student voteInPoll() so the Student Government get's their Response.
    - Interested in what their constituents think, the Student Government getResponses() to see all the responses from the election.

Data Model - Ron (Revised)

The final data model for DormPoll looks as follows:

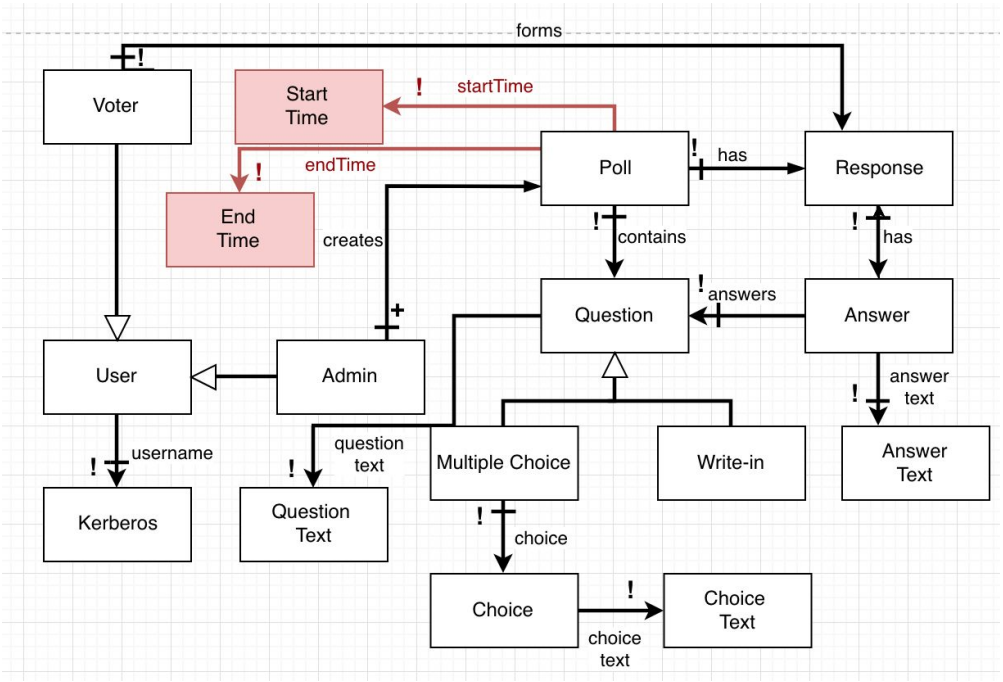


Textual Constraints

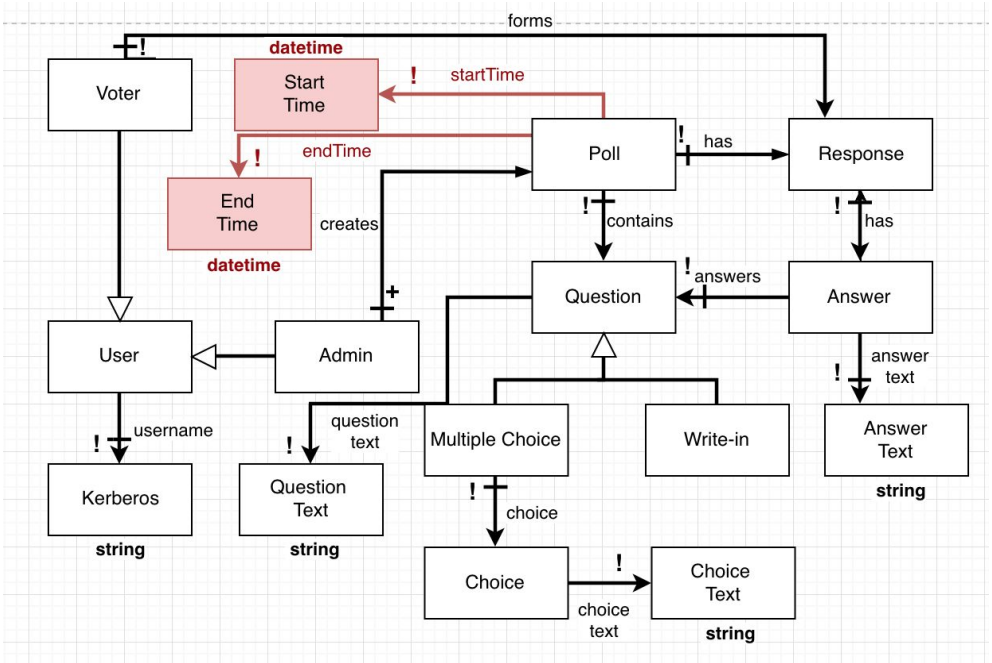
- For each poll, each voter has at most one response
- For each response, there is exactly one answer that points to answers each question of the poll that the response responds to.
- **A poll's endTime must FOLLOW a poll's startTime.**

Schema Design - Ron/Nick (Revised)

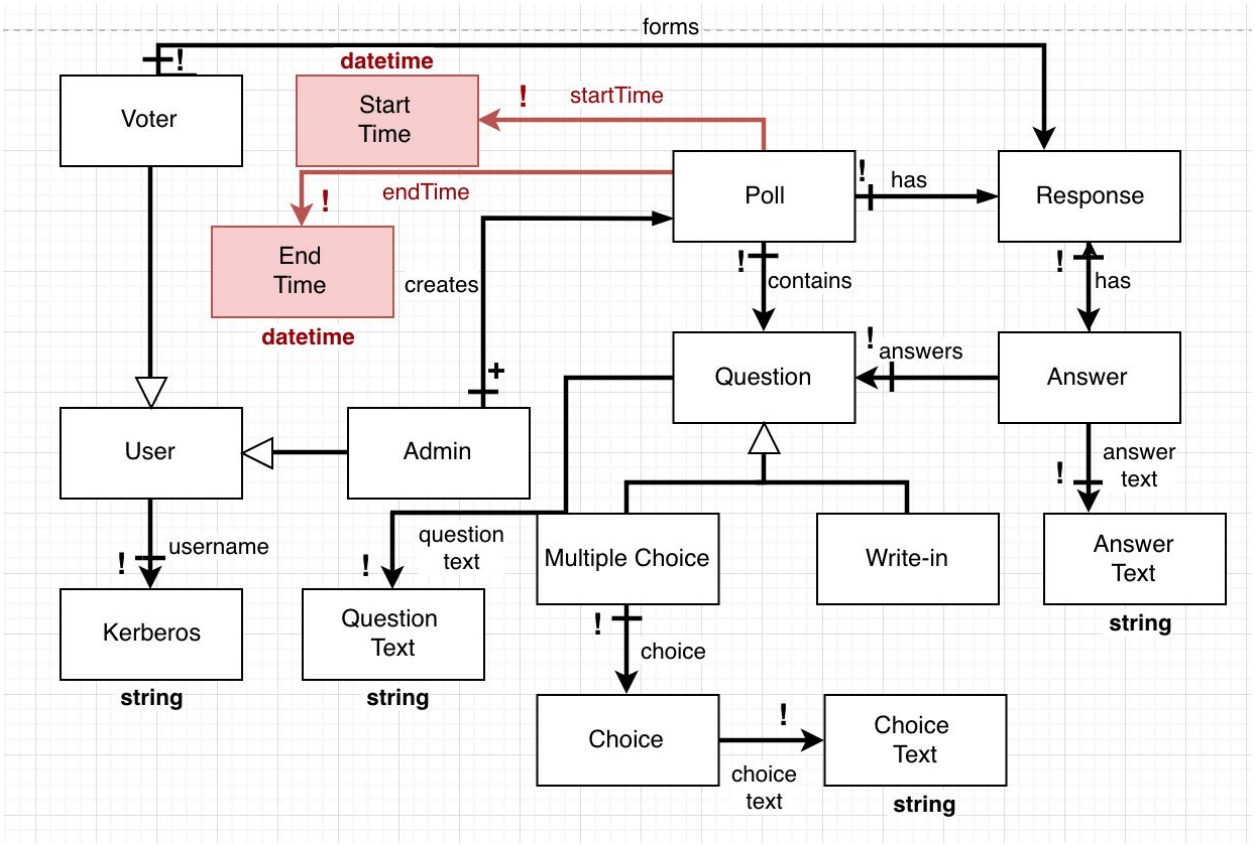
Step 1: Add Attributes



Step 2: Pick Primitive Types



Step 3: Reverse 1:Many Relations



Step 4: Break into/Create Tables - Nick/Ron

```
CREATE TABLE IF NOT EXISTS users (
        userId INT PRIMARY KEY AUTO_INCREMENT,
        kerberos TEXT NOT NULL UNIQUE,
        isAdmin BOOLEAN NOT NULL,
        isVoter BOOLEAN NOT NULL
);

CREATE TABLE IF NOT EXISTS polls (
        pollId INT PRIMARY KEY AUTO_INCREMENT,
        startTime DATETIME NOT NULL,
        endTime DATETIME NOT NULL
);

CREATE TABLE IF NOT EXISTS admins (
        pollId INT NOT NULL REFERENCES polls(pollId),
        adminId INT NOT NULL REFERENCES users(userId),
        CONSTRAINT PK_Admin PRIMARY KEY (pollId, adminId)
);

CREATE TABLE IF NOT EXISTS questions (
        questionId INT PRIMARY KEY AUTO_INCREMENT,
```

```
        pollId INT NOT NULL REFERENCES polls(pollId),
        questionType ENUM('multiplechoice', 'writein') NOT NULL,
        questionText TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS choices (
        choiceId INT PRIMARY KEY AUTO_INCREMENT,
        questionId INT NOT NULL REFERENCES questions(questionId),
        choiceText TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS responses (
        responseId INT PRIMARY KEY AUTO_INCREMENT,
        pollId INT NOT NULL REFERENCES questions(questionId),
        voterId INT NOT NULL REFERENCES users(userId),
);

CREATE TABLE IF NOT EXISTS answers (
        answerId INT PRIMARY KEY AUTO_INCREMENT,
        responseId INT NOT NULL REFERENCES responses(responseId),
        questionId INT NOT NULL REFERENCES questions(questionId),
        answerText TEXT
);
```

Security Concerns - Nick

Since our web application is intended to be used for polling students for elections, it is extremely important for security and integrity of votes to maintained.

Things that could raise potential issues / need to be taken care of:

- Setting up MIT OpenId authentication: We plan to authenticate users by using MIT's OpenId. The problem here is that none of us have worked with OpenId, so we will need to figure out how to integrate that with our react app and how to make sure login tokens are protected properly against any attacks
- SQL injection: Since we are basically storing all of our user data in mysql tables, it is important to prevent attackers from doing anything with that data, including accessing, modifying, or deleting. Any outside actions would damage the privacy of users and disrupt our response collection.
- XSS: Users creating the polls or answering polls could potentially add html to their responses that could potentially execute on others' browsers. The intent could be malicious and we would like to prevent this.
- CSRF: Attackers can create an external site that relies on cookies in the browser to make api calls to our server using the credentials of users who are currently logged into our website on that browser. This way attackers could fake their identity and perform many different malicious actions that are extremely detrimental to a voting platform.
- Double-write attacks: Users might be able to find a way to submit two votes simultaneously and bypass our checker to see if users have voted before and thus insert two votes into the database. Of course, we do not want to have people voting twice.

Means of mitigating these risks:

- Securing SQL injection: To prevent this vulnerability, we will sanitize user inputs by using node mysql's escape function.
- Preventing XSS: We will be using react which sanitizes html tags and other html. We will also strip html tags from any user inputs by using node's sanitizer package.
- CSRF: We will use the csurf package from node to remember browser sessions and only allow api calls to be done in the same webpage that they signed in on. This way foreign api calls with the right login credentials will be blocked if they are not made through our given web interface.
- Double-write attacks: To block users from voting multiple times, we will write DB queries such that when retrieving anyone's vote/response, we only accept the latest one. Also when retrieving all responses for a poll, we will ensure that only the latest response from each user is provided. This way, regardless of attacks, each user will have one response submitted. Additionally, they will be able to see which one we have on record for them.

Dependencies - Darius and Nick (New)

Material-UI

We have decided to use the Material-UI package instead of styling everything ourselves. This allows our website to have consistent styling all around without creating our own global css style and reduces repeated css. Material-UI also supplies a lot of basic element functionality (radio buttons, text inputs, etc) that we no longer need to implement ourselves, saving time for the more complicated parts of our website. Finally, the Material-UI style is consistent with Google's UI design, ensuring our website looks up-to-date and well-made.

OpenIDConnect

We used mit's openidconnect to have a means of authenticating users such that they can just use their kerberos / password combination instead of having to go through the hassle of creating accounts and profiles. When a user logs into our site, they are redirected to mit's openid site where they log in with their kerberos or mit certificate. If they provide the proper credentials there, the openid site sends a get request with the authentication token to our server which then gives them access to our site logged in as that particular user.

Wireframes - Claudia (Revised)

https://app.moqups.com/wclaudia888@gmail.com/Flk8sfK751/view

Further Comments:

~~For navigating between pages (i.e. Elections -> Guest List or Guest List -> Home), a user would be able to open the hamburger icon in the top right to click on their desired destination. For the sake of simplicity for the wireframes, this hamburger icon is not clickable for each page, but a sample of what it would look and behave like is shown in page 2 (User Portal - Hamburger).~~ **There is no longer a distinction between election, guest list, and feedback polls. They are now just categorized all as polls, so the intermediate step of choosing between types of polls is removed. Navigation between pages will be handled by a navigation bar at the top of the page with links to the Home and Poll Page (and maybe a Create Poll page if you are logged in as an admin). Since there are only two links, a hamburger icon is no longer needed as much.**

~~Also, since Elections and Feedback polls behave pretty similarly, only Election Items have been sketched out with their "closed" and "open" views. Feedback items would largely be the same with a further option of having text inputs instead of just radio buttons with candidate names (i.e. a text input for suggestions students can add with an "Other" option).~~ **Since everything is just categorized as polls now, they all have similar UI, except for guest lists. To follow current guest list protocol, instead of being able to add items to a guest list, just have 5 guest list items (not shown on the page though because of space) that a user fills in as needed**

For admin to create polls, only the action of creating an election has been sketched out on the CreatePoll page, but creating the other kinds of polls would behave similarly. **The UI for creating a poll is more flushed out with fields for name, dates, and questions, which are also available to be edited by admin as well. There is now a new Admin portal to add/delete admins by clicking on the "Admin" button at the navbar.**

When a user logs in through OpenID, the backend will determine whether that user matches any person on the admin list, which will start off with 1 person manually inputted. For the sake of simplicity for the wireframes, there are two buttons to login as a user and admin to show the distinction between the two flows. The "User" or "Admin" button at the navbar acts as a link to a settings page, with basic info or functionality, like adding/deleting admins for an admin.

For the Polls page, the only difference between what a user and admin sees is that admin has additional functionality: edit, view results, and delete. If an admin clicks on a closed or open poll, they will see the same things a user sees. If they click edit, that will bring them to a page where they can edit individual questions on a poll. If they click view results, that will bring them to a page where they can view the results of a poll.

Design Commentary - Claudia (Revised)

Starting with our idea of dorm polling, we discussed the current ways polls are conducted in various dorms across campus. We agreed that having it digitalized was the best option, but differed in opinion on how this should be done.

One option was to adopt the way Maseeh polling is currently done, by having a Python input file with specific JSON objects and then render it into a user interface. The advantage of going with this option was that it was already shown to be working and was relatively simple for students to use. However, it would require some training and technical knowledge on the part of the administrators to make sure they were able to successfully create polls and view their results. The main challenge here would be how to render these JSON objects from the input file into a functional design that is easy-to-use.

The other option would be to go in a completely different direction from what dorm polling is now and create a UI that both users and administrators can interact with directly. This would scrap the need for a Python input file, which previously was a hassle for administrators to deal with, so creating or editing a poll would be made much easier. The problem though was trying to understand the specific needs of each dorm and try to find a way to combine them all into a UI that would be all-encompassing, while still being easy-to-use and easy-to-learn.

In the end, we decided that having everything be done directly through UI made more sense for users and administrators. However, we were in unknown territory and had to really examine everything we knew to determine the appropriate scope of the problem. Should we only allow polls to be made up of simple questions and options in the form of multiple choice? Or should we allow polls to consist of multiple types of inputs (i.e. text, dropdown menus, checkboxes, date/time)? While including more options would give more freedom for administrators to structure their polls, it could easily make the problem of polling unnecessarily overcomplicated.

To start off, there will be at least one admin hardcoded into the database through PHPAdmin. We can make this assumption because the admin of the website will control what polls are released and who these polls will be directed towards. We need to make this assumption because there is no way to validate the dorm student government through DormPoll, so we should do this through the database.

We ~~picked out~~ **came up** with three of the most common polls a dorm would need (elections, guest lists, and feedback) and based our design on the functionality ~~each poll~~ **a poll on DormPoll would require to meet the basic needs of these already existing types of polls**. We also decided to limit the response options to just multiple choice and text inputs if necessary, like for guest lists or places for users to share their ideas in feedback forms. This compromise between a simple and complex design allows DormPoll to meet the various functionalities a poll needs to serve a specific dorm, while still being straightforward.

Polls can be edited, whether they are closed or opened, by admin. The name, dates, and questions of a poll can be changed as needed. If questions are changed in any way, then we decided that all the previous user responses will be discarded. While this also helps us as developers, this also keeps everything as valid and consistent as possible, because a change to question #1 of 3 questions could greatly impact a user's answer to the rest of the questions. The name and dates can be changed as needed without affecting user responses. Changing dates gives the option for admin to reuse or extend the length of a poll.