**Xteam:** X3

**Lecture Number:** CS 400

**Xteam members:** Steven Qian, Chance Overberg, Ethiyal Raj Wilson

**Title**:

Polling Service with Secure and Efficient Access Management

**Problem:**

Username and password authentication continues to be an integral part of the day-to-day operations of many businesses. There is a need for an efficient and scalable way to store individual users and the access levels they should have.

The data storage method needs to be secure, as an easily compromised password will compromise the use case of this data storage method, and in extreme cases, lead to severe cybersecurity incidents.

At the same time, the data lookup needs to be efficient. Users will likely need to authenticate on at least a daily basis, and poor performance in this workflow will quickly decrease user satisfaction. In cases where the business grows, this also needs to be scalable to handle large increases in user volume.

Many companies rely on consistent market research to drive new projects. As part of that research, polling is a typical strategy for gathering the data necessary. This application would allow businesses to upload polls that would be distributed to users, and users would answer the polls using their individual credentials.

**Primary Stakeholder**:

There are likely many stakeholders at a leadership/administrative level. Those with most riding on the business (example: C-level executives) will want view the results of any given poll since they have the most use for the results. Those working in the technical operations of the organization will also be important to consider (example: sysadmins, Security leadership) as they will be responsible for keeping the underlying infrastructure secure and running by maintaining existing polls, validating data, and uploading new polls.

However, every individual at a business/organization utilizing this functionality will ultimately have a say in this process, including the users answering the polls. Best practices will ensure the process remains secure, and that individual accounts do not get compromised.

**Graphical Interface:**

The two core user interfaces would be a login page and a registration page. The login page would authenticate against existing data and the registration would create new entries. There is room for flexibility here on how many data elements we are collecting and the overall look and feel. Below are some sample designs, with the account creation also storing an email address.

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, Teams

Description automatically generated

Upon logging in, a typical user will be presented with a poll with a list of options for answers:

Table

Description automatically generated

The user will be able to fill the poll out and submit their answers to be aggregated alongside others’ answers.

A system administrator or leadership user, however, would be presented with a dashboard with the results of any given poll. The interface will look similar to the polling interface presented to typical users, but it would report the percentage of people who gave any given response:

Table

Description automatically generated

System administrators would also have a menu button that would allow them to upload a JSON file of the question and answers for a new poll.

**Data:**

For the user credentials, a hash data structure will allow both efficient lookup and secure storage of certain values and using a node to store those in combination with a lookup key will likely achieve our core workflow. This would be taking a bucket approach to storing index positions.

A base-level mockup would store this information in the following way:

* Key: Username (hashed + indexed for lookup purposes)
* Value: Inner class that stores a Password (private variable in the node and hashed from user input) and the user’s level of access (private variable that indicates typical user or system administrator/business user
  + Helper methods should only compare a user input to the password value, but not allow the raw hash value to be pulled

Our method for handling collisions can be flexible, but we will likely achieve the best performance results by storing as a LinkedList.

A hash data structure will also allow flexibility with resizing. An easy way we can do this is by setting a load factor threshold to know when we need to expand our number of indices available.

As for the poll results for any given poll, two structures will be used: a hash map and an array.  
The hash map will allow for efficient lookup of specific users’ answers. That way, if a user decides to look at a poll they previously answered, their previous answers will be defaulted in.  
The hash map will store information in the following way:

* Key: Username (hashed + indexed for lookup)
* Value: the answer given for the poll in question

The array will allow for efficient lookup of an answer’s frequency for the dashboards. If a system administrator or business user views the dashboard, this array will allow for quick loading of all answers and their frequencies.  
The array will store information in the following way:

* Each index will store an inner class object where each object contains:
  + A private string variable representing the answer
  + A private integer variable representing the total number of respondents who gave this answer

**Input Data File Format:**

A new poll will be input by a system administrator using a JSON format.  
The JSON file will have two main objects: the question object and the answers object. The question object will contain the poll question being posed while the answers object will contain inner objects that are each an answer possible.

An example would be:

{

**"question"**: "Which car name is the most appealing?",

**"answers"**: {

**"answer 1"**: "Mustang",

**"answer 2"**: "Corolla",

**"answer 3"**: "Juke",

**"answer 4"**: "Patriot"

},

}

**Output Example:**

The output to the dashboard will be as described the Graphical Interface section where each poll will have a table of its responses side-by-side with the percentage of respondents who provided that answer:

Table

Description automatically generated