Agree

Project Documentation

Group 6
SDEV 495
December 2018

Hosted repositories
https://dev.azure.com/DOLIVER28/group6/_git/AgreeAPI
https://dev.azure.com/DOLIVER28/group6/_git/AgreeDesktop

Team members
Evans Ampomah, Shafro Batyrov, Mercymoy Gurmesa, Seth Kendrick, and Daniel Oliver

Table of Contents

Overview

This project, as part of the capstone to our degree programs in either computer science or software engineer and security, was meant to demonstrate our ability to plan and execute a software project as part of a small team.

At the end of this course, this team readily admits it did not achieve what it hoped to. Some of this is can be blamed, to varying degrees, on some personal matters or professional obligations, but on the whole we fell short on a few areas, specifically communication and collaboration.

We still produced enough planning and documentation that given more time and focused efforts, this team (or another) could produce a working product from it.

Individual Contributions

Each weekly submission was accompanied by a contribution report. Participation varied week to week, so a summary is below:

Evans Ampomah

<summary here>

Shafro Batyrov

Shafro primarily prepared the preliminary GUI's and, alongside Seth, worked on connecting the GUI to the API. She also worked on the test documentation and made sure the document was ready for submission each week.

Mercymoy Gurmesa

<summary here>

Seth Kendrick

Seth built and implemented the GUIs based on the GUI designs and connected the GUI's to the API. He also contributed to the project documentation.

Daniel Oliver

In an effort to ease distributed development and testing, Dan focused on putting the application's data in an online database accessible through a RESTful(ish) API. Earlier in the process, he also attempted to organize efforts using Microsoft's DevOps project-planning software.

Project Plan

Concept Approval and Requirements Review

We aim to develop an application that helps a group of people come to a decision while mitigating some of the social pressures that typically impede the process.

Requirements

To be successful, the application must:

1. Allow multiple users to offer their preference on a decision without exposing that user's preference.
2. Present a resulting decision without revealing how any individual participant voted.
3. Support some variety of ranked-choice voting. That is, the applications should take into account preferences beyond a single choice, anticipating that while one user's choice might have little support, their second choice might have broad support.

Should time permit, we would also like the application to:

1. Support negative votes, i.e., *voting against* an option.
2. Support across multiple devices instead of a single interface.

Example Scenario

Let's consider a group of dislocated sisters planning a wedding shower with some mundane decisions to coordinate:

1. Food options
2. Invitation theme
3. Game ideas

Communicating each of the available options would be tough enough. Then there's the trouble keeping track of who votes for what, and what they would support if nobody likes that idea, etc. And *then* there's the politics of announcing an opinion in a group text or a convoluted email chain.

Now, imagine the sisters have a tool to get these decisions made and move on. Anne creates a *Decision*: **"A Game for Judy's Shower"**

Then, she sets the *Quorum*: the minimum number of options needed to begin voting, and then the conditions for a decision. The conditions could be, say, a deadline ("5pm tonight"), open-ended ("Once everyone votes"), or some more demanding hybrid ("20 min after the most recent vote when at least 80% of invitees have voted").

Then, she invites all her sisters to participate in this *Decision*.

They can see available options, maybe add one of their own if Anne allowed it, and then begin doing ranked-choice voting. Just put the options in order of preference, with the option to Veto (or at least vote strongly against). Up until the end of voting, the ladies can adjust their votes, but they can't see what the current score is.

Once the deadline hits, the application delivers the decision. This is where the application can really mitigate the politics by obscuring how many votes something received. Instead, we get the one winning option or, in the case of a tie (though that chance should diminish with enough options and voters), the remaining contenders.

Minimum Viable Product

The Minimum Viable Product will include these fundamental features and capabilities:

- Allow a user to create a question with up to three choices
- Set a standard deadline of five minutes
- Allow at least five participants to vote upon proposals
- Display the resulting decision

These features will allow for a basic decision-based proposal to be created by a user. The user will be able to include up to three options to be voted upon by up to five participants. Each proposal will have a standard deadline of five minutes to be voted upon. After this standard deadline, the proposal will no longer be open for voting. The results of the decision will be displayed within the application to be accessed by all participants.

Platform

The desktop application will be written for a Windows 10 environment using the latest JDK. The web API will be in a LAMP environment using at least PHP 7 and MySQL 5.5.

Design Approval

The application design will be complete when the data structure is laid out, endpoints and their arguments are named, and the basic GUI is described..

Project Milestones

First Deliverable

November 25, 2018 - The first deliverable will meet the fundamental product features as described in the minimum viable product.

Second Deliverable

December 2, 2018 - The second deliverable will build on the minimum viable product by implementing user accounts with corresponding login credentials.

Third Deliverable

December 9, 2018 - The third deliverable will include sent notifications to users about decision deadlines and decision results. The deliverable will also allow for a larger pool of users to vote on decisions.

Fourth Deliverable

December 16, 2018 - The fourth and final deliverable will provide a fully functional application delivering on all planned features without error or defect.

References

Whiston, Debbie. (2003). *Project Plan Odessa Mobile Technology Project*. Retrieved from Sample Project Planhttps://www.search.org/files/doc/Sample%20Project%20Plan.doc.

Meyer, B. Kolb, P. Software Project Plan Template. Retrieved from se.inf.ethz.ch/old/teaching/ws2005/0273/slides/Template%20Project%20Plan.pdf

Testing Plan

Purpose

This test plan describes the approach and overall framework that will guide the testing of *Agree*. The document introduces:

- Test Strategy: the objectives, approach, assumptions of the test.
- Test Execution: how the test will be carried out, the expected input for and output of the test, how to identify errors, and how to fix those errors.

Project Overview

Agree is an application that aids in solving pressurized decisions for any group of people. The application allows a user to create a list of options and invite other users to select options acceptable to them and rank them in order of preference, all in anonymity and without the worry and anxiety many group-based decisions entail. With the final results tallied and displayed for all invited guests, the way forward for the group is made.

Audience

- Project team members will perform the tasks specified in the test plan document. Team members will also provide input and any necessary recommendations on the test plan.
- Professor will oversee the project, review and comment upon the test plan, and advise on any necessary corrections to the plan or the overall direction of the project.
- Peer reviews from fellow classmates will review and comment upon the test plan and suggest any necessary corrections to the plan or to the overall direction of the project.

Test Strategy

Test Objectives

The object of the test is to validate the application's functionality and that it works as specified. The test will determine what works well and will identify, offer solutions to, fix, and retest all flaws in the minimum viable product. The test will produce a functional minimum viable product as specified in the project milestones, which includes:

- allowing a user to create a decision-based proposal with up to eight options to be voted on with ranked-choice voting
- setting a standard deadline of five minutes for all proposals to be voted on
- allowing up to 100 participants to vote
- displaying the resulting decision after certain conditions are met

Test Assumptions

The minimum viable product of the application will be complete.

Features To Be Tested

The initial testing will focus on the backend functionality of the application. This testing will also include a basic functionality of the user interface. Specifically, it will address how the user creates and submits a proposal within a specified deadline, how invited users view and interact with that proposal, and how all users view the results. These features are dictated by the minimum viable product.

Features Not To Be Tested

Upon delivery of the minimum viable product, later iterations of the application will incorporate the following features, which will not be included in the initial testing:

- user accounts with login credentials
- multiple users to create proposals

Test Execution

Test Expectations

For successful testing of the application, the following inputs are expected:

- User creates a decision-based proposal sentence
- User creates a minimum of two options to the proposal to be voted on
- User submits the proposal via the submit button
- Invited users select and order their votes from the created proposal
- Invited users submit their votes via the submit button

With the above inputs, the following outputs are expected during testing:

- Notification of a successfully created proposal
- Notification of a unsuccessfully created proposal
- Notification of a successful vote for a proposal option
- Notification of an unsuccessful vote for a proposal option
- Notification that voting time for the proposal has ended
- Display of decision results to all users

Test Cases

Test Case #1 - Open Application		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	Application displays the question-creation UI on opening a new instance.	

Test Case #2 - Create Decision-Based Proposal		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	User able to create decision-based question sentence.	
2	User adds the minimum of two options for the question.	
3	Additional options for the question are added if necessary.	
4	User successfully submits question to be voted on by other users.	

Test Case #3 - Voting Deadline Initiated and Completed		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	After user submits the question, a timer begins allowing other users five minutes to vote on the proposal.	
2	The question closes after the allotted five minutes.	
3	With the question closed, invited users can no longer gain access to the question.	

Test Case #4 - Invited Users Vote on the Proposal		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	Invited users can access the question.	
2	Invited users select acceptable options and rank	

	them in order of preference.	
3	Invited users submit their voted option.	
4	Uninvited users cannot gain access to the question.	

Test Case #5 - Display Results		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	When the decision conditions are met, be it through a hard deadline or the max-time-after-last-vote limit, the application's decision is calculated and displayed.	
2	If too few users voted, and the results would single out a user and make their choice obvious, the results *do not* display.	

Test Reports

Report Table

The following table records the overall pass/fail of the test cases.

Test Case	Pass	Fail
Test Case #1		
Test Case #2		
Test Case #3		
Test Case #4		
Test Case #5		

Test Summary

(A summarization of the testing process - successes, failures, proposed fixes, etc)

References

Software Testing Fundamentals. *Test Plan*. Retrieved from http://softwaretestingfundamentals.com/test-plan/

Software Testing Help. *Test Plan Tutorial: A Guide to Write a Software Test Plan Document from Scratch*. Retrieved from https://www.softwaretestinghelp.com/how-to-write-test-plan-document-software-testing-training-day3/

User Guide

Terminology

Question

The decision your participants will be making. It should be a short *who*, *what*, or *when* question, like this:

Who should be the next club treasurer?

Or:

What should our team mascot be?

Option

One possible answer to the question. It should be unambiguous and not require any follow-up questions. For example, the club treasurer question above might have four options:

1. *Joey*
2. *Cathi*
3. *Fatima*
4. *Benjamin*

Ranked Choices

The options you find acceptable in order of your preference (in descending order). A difficult element of group decisions is tracking preferences and consolidating support around one choice. Here, you will choose any number of options you would be okay with and then rank them. Should your top choice be out of the running, your second choice becomes your vote.

Creating a Question

Open the Application

Click on the application. You should see two options: *Create new question* and *Participate in a question.*

Compose Your Question

Click on “Create new question.” Compose a short question (140 characters or less) in the text box and click “next.”

Create Your Options

You should see a form with two small text boxes. After you fill the second box, a third one will appear. Continue adding options as you need them, up to eight total. When you’re satisfied with your options, click “next.”

Set the Decision Conditions

You will be prompted to choose the conditions that will cause the question to end. You’ll have three settings to adjust. You can use them in any combination or not at all.

1. **Hard deadline**. You will choose a date and time.
2. **Participation minimum**. Whatever your other settings, if this is set and not met, no decision will be generated, and the question will be “dead” and a new one will have to be created. Users will see a message to this effect: “
3. **Moving time limit and participation threshold**. To encourage engagement in larger groups, this is a short time limit that begins to run after the participation minimum is hit, and is restarted after each subsequent vote is submitted. If the vote drags to the point where no vote is submitted before this moving time limit expires, the vote closes.

Participating in a Question

Open the Application

Click on the application. You should see two options: *Create new question* and *Participate in a question.* Click “Participate in a question.”

Choose a Question to Participate in

Because the questions are confined to the application instance (i.e., the window you have open), you will see the list of available questions. Choose one.

Examine the Options

Drag the options you'd be okay with to your choices field. From there, organize them by preference, top to bottom, with the top being your preferred option, and the last being your least favorite option that's still acceptable to you.

Review Your Choices and Submit

Once you submit your vote, there's no going back. When you've submitted your vote, you will see a screen confirming your vote has been accepted, and the option to return to the question list.

Deadline

Approaching the Deadline

From the time the deadline is set, be it through a hard deadline or a roving deadline as explained above, it will be visible to all participants in the bottom of the UI. For example, no matter what stage they are at in the decision-making process, there will be a little reminder on the bottom that says "Decision Due: xx/xx/xxxx at xx:xx AM/PM".

Final Decision

When the deadline has passed, the program will generate the highest-ranked option as the decision. This decision will be visible to all participants in that question. Once a decision is reached, that question will appear below the active questions. Once a question has been decided on, there is no way to edit the decision.

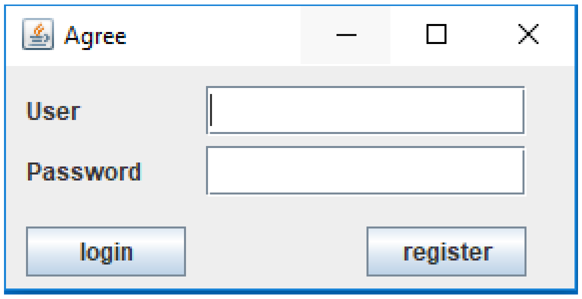
References

Giordano, Conne. 2018, July 25. "User Guide Template". Retrieved from https://techwhirl.com/user-guide-template/

Design

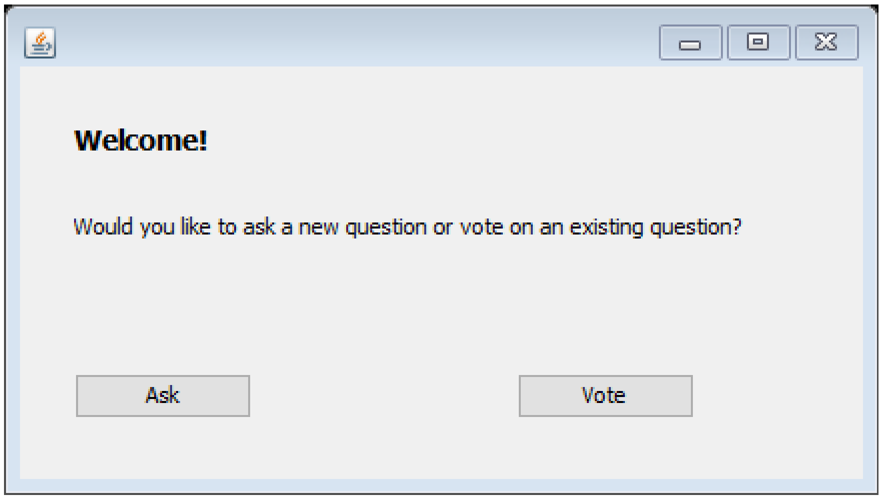
Desktop Interface

Initial Login GUI



The application opens with the login window. Users are presented with two options: 1) the user may enter a username and password and press “login” to enter the application; or 2) the user may press “register” to create an account and then may access the application.

Choosing to Ask or Vote

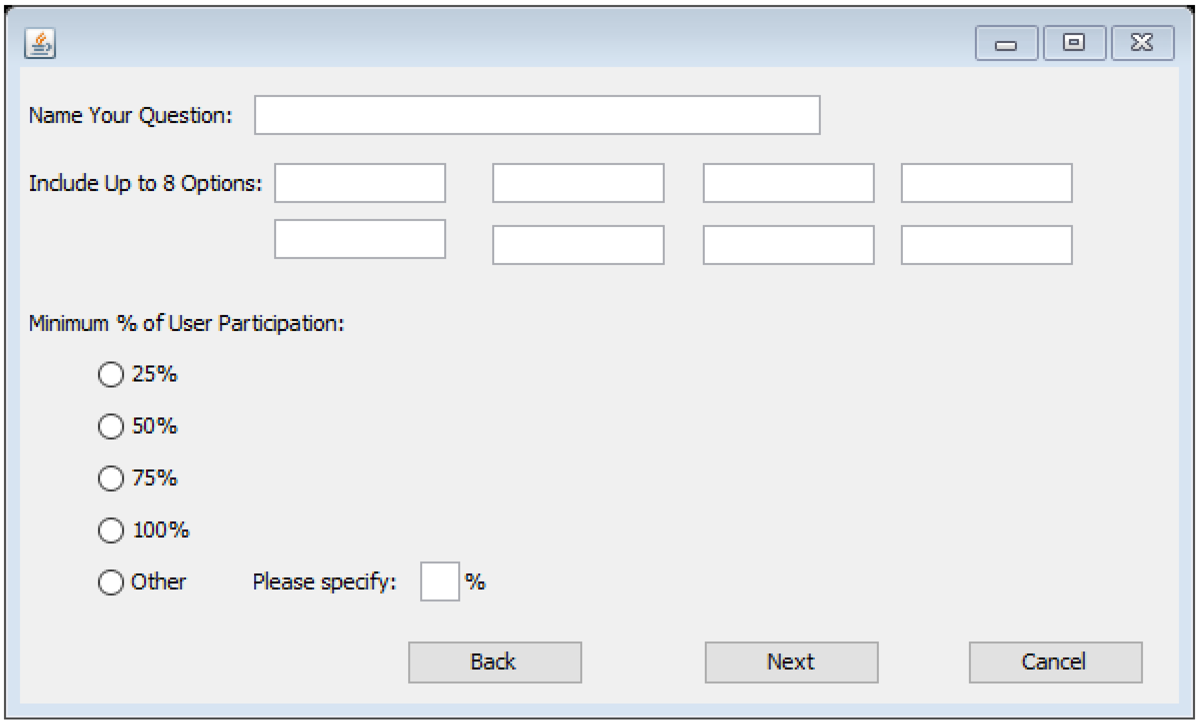


Welcome!

Would you like to ask a new question or vote on an existing question?

The welcome window greets users after gaining access to the application. Users are presented with two options: 1) to create a decision-based question, users are directed to press “Ask”; or 2) to vote on an already created question, users are directed to press “Vote”.

Asking a Question


Name Your Question:

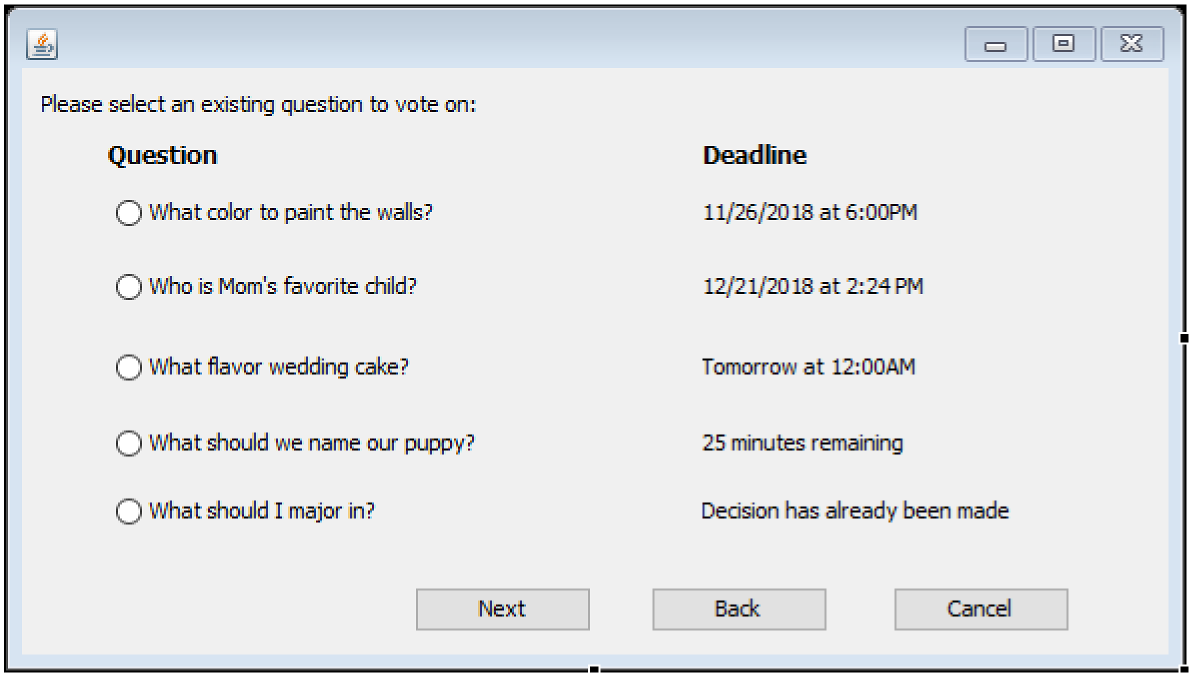
Include Up to 8 Options:

Minimum % of User Participation:

- 25%
- 50%
- 75%
- 100%
- Other Please specify: %

After selecting “Ask” from the welcome window, users are directed to this window to create a question. The user enters any question they want in the first textfield. Next, the user may enter up to eight options to be voted and ranked by fellow users. A minimum of two options are required for the question to be submitted. The user may also select the percentage of users invited must participate for the decision to be valid. If the user has completed creating the question with options, the user presses “Next” to submit the question for voting. To go back to the welcome window, the user presses “Back”. To exit the application, the user presses “Cancel” where the user will be prompted to confirm their decision to exit.

Voting on an Existing Question

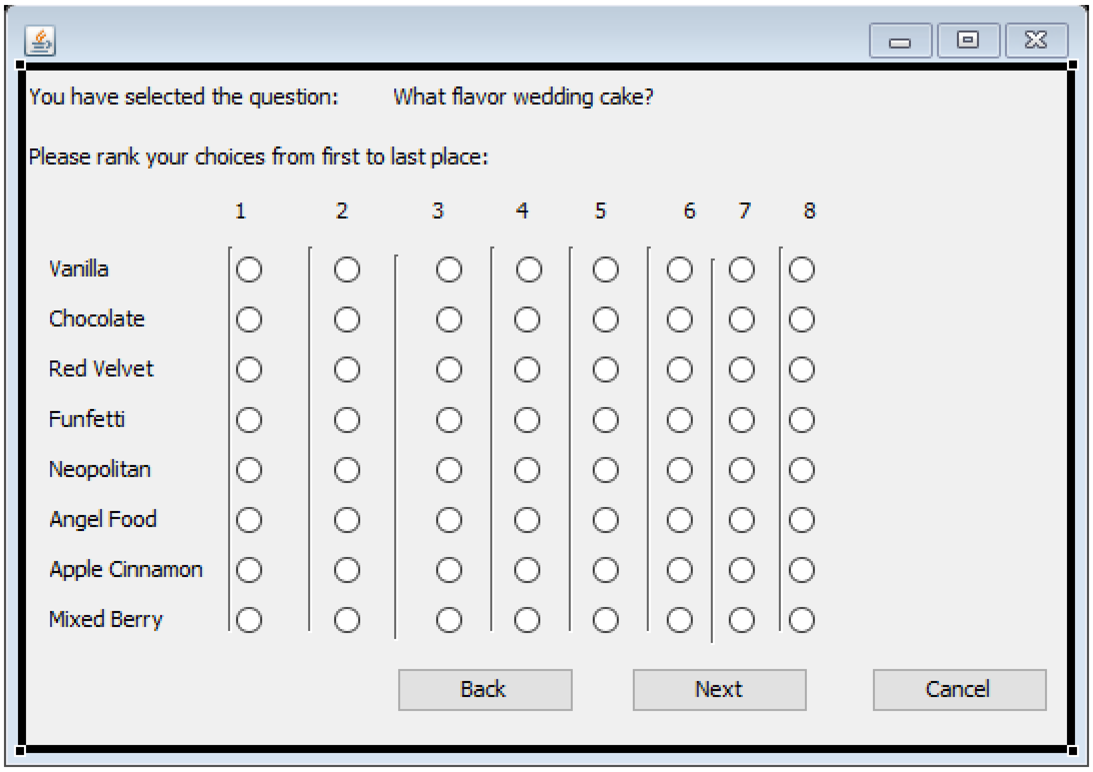


Please select an existing question to vote on:

Question	Deadline
○ What color to paint the walls?	11/26/2018 at 6:00PM
○ Who is Mom's favorite child?	12/21/2018 at 2:24 PM
○ What flavor wedding cake?	Tomorrow at 12:00AM
○ What should we name our puppy?	25 minutes remaining
○ What should I major in?	Decision has already been made

After successfully submitting the question for vote, the user may vote from a list of questions that the user created or that the user was invited to participate. Each question in the list includes its deadline, that is, when the voting period will end for that specific question. After selecting the desired question to vote, the user may vote on the question by pressing “Next”. If the user wishes to return to the previous page, the user may press “Back”. If the user wishes to exit the application, the user presses “Cancel” where the user will be prompted to confirm their decision to exit.

Choosing an Option

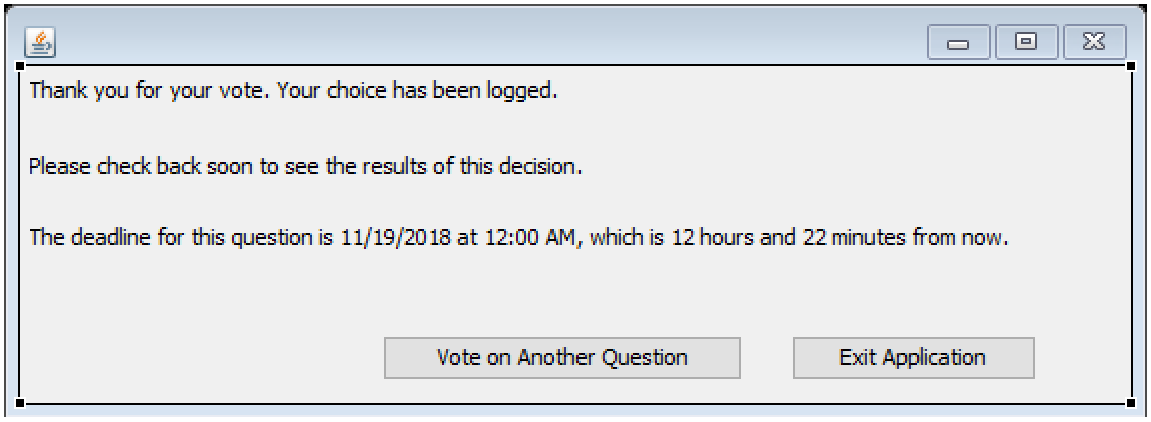

You have selected the question: What flavor wedding cake?

Please rank your choices from first to last place:

	1	2	3	4	5	6	7	8
Vanilla	○	○	○	○	○	○	○	○
Chocolate	○	○	○	○	○	○	○	○
Red Velvet	○	○	○	○	○	○	○	○
Funfetti	○	○	○	○	○	○	○	○
Neopolitan	○	○	○	○	○	○	○	○
Angel Food	○	○	○	○	○	○	○	○
Apple Cinnamon	○	○	○	○	○	○	○	○
Mixed Berry	○	○	○	○	○	○	○	○

After selecting the question to vote upon, the user may vote by ranking the preselected options. Only one option may be selected for each rank or number choice, that is, an option cannot be selected twice. Also, only one rank may be selected for each option, that is, a rank can only be used once. After the user has ranked all of the options, the user may submit the vote by pressing “Next”. If the user no longer wants to vote on this question but vote on another, the user presses “Back” to return to the list of available questions. If the user wishes to exit the application, the user presses “Cancel” where the user will be prompted to confirm their decision to exit.

Vote Logged Page


Thank you for your vote. Your choice has been logged.

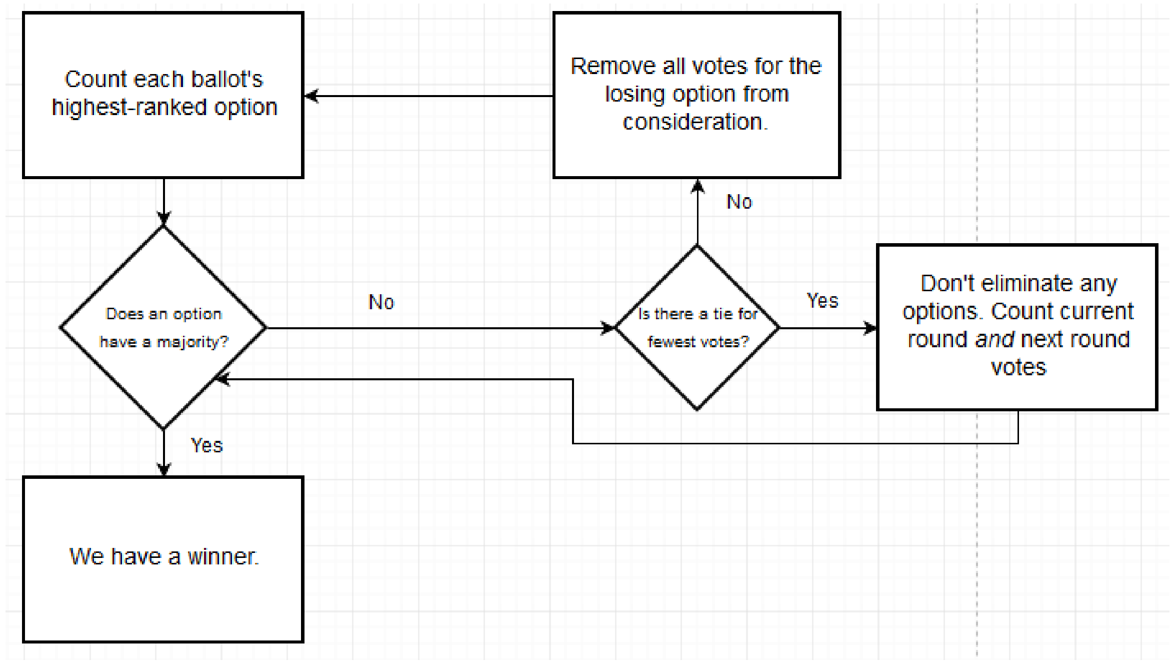
Please check back soon to see the results of this decision.

The deadline for this question is 11/19/2018 at 12:00 AM, which is 12 hours and 22 minutes from now.

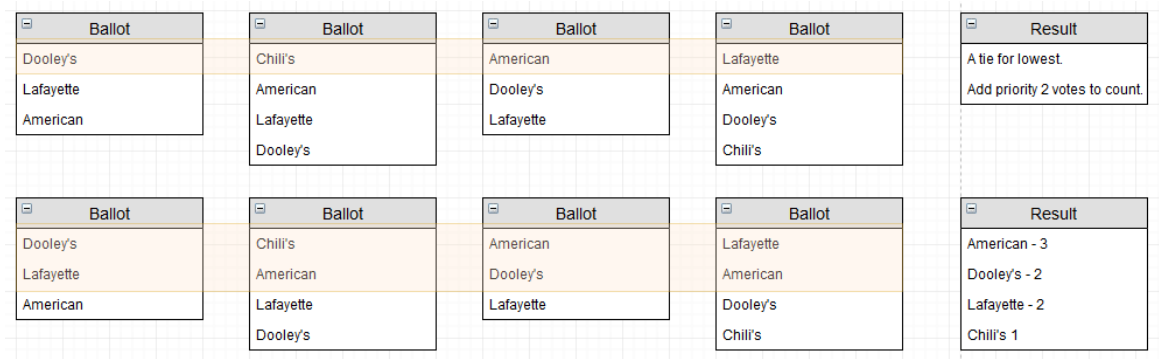
After the user submits the votes to the selected question, this window notifies the user that their vote has been successfully submitted. The user is reminded of the deadline for the question. To vote on another question, the user presses the "Vote on Another Question". If the user wishes to exit the application, the user presses "Cancel" where the user will be prompted to confirm their decision to exit.

Application Logic

The key feature in the application logic is the actual decision-making process, where the votes are counted in a ranked-choice or "instant runoff" scheme.


Count each ballot's highest-ranked option
Remove all votes for the losing option from consideration.
No
Does an option have a majority?
No
Is there a tie for fewest votes?
Yes
Don't eliminate any options. Count current round *and* next round votes
Yes
We have a winner.

Because we can anticipate small groups of people using this, the chances of ties in a given round are very high. That is why we have the unusual variation on the right side of the diagram above, where we can occasionally count two of all users' votes in the same round. Here is an example of what we can reasonably expect to be a common scenario:



Ballot	Ballot	Ballot	Ballot	Result
Dooley's	Chili's	American	Lafayette	A tie for lowest.
Lafayette	American	Dooley's	American	Add priority 2 votes to count.
American	Lafayette	Lafayette	Dooley's	
	Dooley's		Chili's	

Ballot	Ballot	Ballot	Ballot	Result
Dooley's	Chili's	American	Lafayette	American - 3
Lafayette	American	Dooley's	American	Dooley's - 2
American	Lafayette	Lafayette	Dooley's	Lafayette - 2
	Dooley's		Chili's	Chili's 1

The table structure is below. The key to understanding the terminology is that for this application, a single ballot is a collection of several votes. Ballots are not meant to be connected to an individual user. Rather, a valid, unused invitation, which is connected to the user, allows a user to submit a ballot. While this leaves a potential gap where we cannot tie a single ballot to a single user, it preserves fundamental anonymity while allowing at least some validation in the form of comparing the raw number of invitations to the number of ballots cast.

```
Users
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id              | varchar(40)  | NO   | PRI | NULL    |       |
| email           | varchar(64)  | NO   | UNI | NULL    |       |
| password        | varchar(255) | NO   |     | NULL    |       |
| token           | varchar(40)  | YES  |     | NULL    |       |
| lastActivity    | datetime     | YES  |     | NULL    |       |
| activityProfile | text         | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
Questions
+-----------------------+--------------+------+-----+---------+-------+
| Field                 | Type         | Null | Key | Default | Extra |
+-----------------------+--------------+------+-----+---------+-------+
| id                    | varchar(40)  | NO   | PRI | NULL    |       |
| title                 | varchar(240) | NO   |     | NULL    |       |
| userId                | varchar(40)  | NO   | MUL | NULL    |       |
| startTime             | datetime     | YES  |     | NULL    |       |
| hardDeadline          | datetime     | YES  |     | NULL    |       |
| movingDeadlineSeconds | int(4)       | YES  |     | NULL    |       |
| trashed               | int(1)       | YES  |     | NULL    |       |
+-----------------------+--------------+------+-----+---------+-------+
Choices
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| id         | varchar(40)  | NO   | PRI | NULL    |       |
| questionId | varchar(40)  | NO   | MUL | NULL    |       |
| title      | varchar(240) | YES  |     | NULL    |       |
| trashed    | int(1)       | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
Invitations;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | varchar(40) | NO   | PRI | NULL    |       |
| questionId | varchar(40) | NO   | MUL | NULL    |       |
| email      | varchar(64) | YES  |     | NULL    |       |
| used       | bit(1)      | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
Ballots;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | varchar(40) | NO   | PRI | NULL    |       |
| questionId | varchar(40) | NO   | MUL | NULL    |       |
| rejected   | int(3)      | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
Votes
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| ballotId   | varchar(40) | NO   | MUL | NULL    |                |
| choiceId   | varchar(40) | NO   | MUL | NULL    |                |
| preference | int(2)      | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
Decisions;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| questionId | varchar(40)  | NO   | UNI | NULL    |                |
| choiceId   | varchar(40)  | YES  | MUL | NULL    |                |
| narrative  | varchar(255) | YES  |     | NULL    |                |
| endTime    | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

Application Programming Interface

To support multiple devices and platforms, the central functionality of this application is accessed through a web API.

Anatomy of a request

Request type

The API does not distinguish between different HTTP verbs. All arguments sent through GET, POST, or PUT are collected and routed according to the *action* parameter without regard to the verb. This means the application will accept passwords sent via GET requests. It is up to the presenting application to choose the appropriate request verb.

Action parameter

The action parameter spells out the user's intent

Authentication

Every request, with the exception of the *help* actions, requires authentication. This can be in the form of an email and password or the token that is returned with each successful transaction that required authentication. The token expires after an hour of inactivity.

action	input	Success output	Failure Output
createUser	newEmail,newPassword	token	FALSE
login	email,password	token	FALSE
updatePassword	newPassword,[authentication]	TRUE,token	FALSE,token
createQuestion	fields,[authentication]	questionId,token	FALSE,token
updateQuestion	questionId,title,[authentication]	TRUE,token	FALSE,token
decideQuestion	questionId,[authentication]	TRUE,token	FALSE,token

deleteQuestion	questionId,[authentication]	TRUE,token	FALSE,token
createChoice	questionId,title,[authentication]	choiceId,token	FALSE,token
updateChoice	choiceId,fields,[authentication]	TRUE,token	FALSE,token
deleteChoice	choiceId,[authentication]	TRUE,token	FALSE,token
getQuestion	questionId,[authentication]	questionObj,token	FALSE,token
getQuestions	[authentication]	[questionObjs],token	FALSE,token
getBallot	questionId,[authentication]	ballotObj,token	FALSE,token
vote	ballotId,[votes],[authentication]	TRUE,token	FALSE,token
logout	[authentication]	TRUE	FALSE,token(?)

[authentication] means either a token or an email/password pair.

Sample requests

/api/?action=createUser&newEmail=joe@example.com&newPassword=123
You should receive a token in response. You can send this token in place of the email/password, though that pair will work as well. Note that it's "newEmail" and "newPassword." This deconflicts the email/password fields if you're automatically sending your own credentials at the same time.

/api/?action=createQuestion&title=What%20color%20should%20we%20paint%20the%20room?&token=1234567890
You should receive a questionId in response.

/api/?action=updateQuestion&questionId=0987654321&title=What%20color%20should%20we%20paint%20the%20kitchen?&token=1234567890
You should receive a 'true' in response.

For testing purposes, the API is available at danielalfred.com/agree/api/

Development History

Beginning in mid-November, the team began contributing to the two repositories listed at the top of this document: AgreeDesktop and AgreeAPI. As their names suggest, the first is the desktop application and the second is the API it communicates with.

Testing proved to be a challenge. While the test plan was reasonable, we had trouble setting up a working test environment to a point where the functionality, even if itself incomplete, could be tested.

Not every member was necessarily expected to contribute code, so the repositories' commit histories may not be an accurate reflection of everyone's contributions.

In the end, several of the desktop interfaces successfully interacted with the API, allowing users to register, login, and create a question. The API got as far as being able to do these three things, as well as create choices, update and delete the question, and invite users to vote the question. However, the central task -- receiving votes and tabulating a result -- remains incomplete.

Lessons Learned

Schedule, and insist on, a team meeting early in the week. A regularly-scheduled meeting early in the week to establish team goals and individual assignments would have mitigated several of the problems that appeared on Sunday evenings. One problem was that the week's deliverable was often poorly understood, and a large portion of the evening was spent sorting out *what* was to be delivered before real work began. We attempted a phone conference for the first team meeting, and another after that, but neither were successful.

Review accomplishments at beginning of each week. So long as assignments were made in the previous week's team meeting, each team member should be able to say definitively if they accomplished their tasks or not. A "no" is acceptable -- what matters is an accurate picture of the team's progress, especially on items that will affect other members' ability to do their work. For example, if one developer did not complete a testable method, the testing team needs to know that their tests for that method should be expected fail.

Lessons learned

Our group will send weekly scheduled program needs to be done on time on the first day of the week which is the most strength of the team. Each week individual knows what needs to be contributed accordingly. However, the group starts working on the project late Sunday every week and this makes some of us rush and reduce the quality of our project. overall , the project management is very nice. Some programing languages such as PHP we used will also affect some us who do not fully understand the PHP programming language. This project is a summary of all we have learned so far and it is a real toughtfull. I learned not only just coding, but how to create a meaningful application. Some of our group members are well experienced so I learned a lot on this project from the group and from individual of the team member as well.

Suggestions for Future Improvement

Develop a web interface. The desktop interface was a good place to start, but a likely scenario is one where users do not have several of their own desktops available. Instead, a simple interface reachable via a mobile device would be useful. A better but potentially more-involved solution would be developing iOS and Android versions.

Add options for inevitable ties. Even with ranked-choice voting, ties are possible when the voting population is just a few friends and the choice list is small. In that case, users might find an option to ask the application to randomly (and irrevocably) decide between the tied choices.

Improve API usability. Right now, an empty request returns a 401 HTTP response. Ideally, an empty response would return a 200 and direct users to some hints or an abbreviated help guide, reserving 401s for requests where authentication was intended and failed.

Appendix A: Contribution Reports

Week 3 (November 5-11, 2018)

Evans Ampomah	Drove much of the initial conversation and facilitated the group decision making. Reviewed all documents for readability and formatting.
Shafro Batyrov	Made large contributions to the both the project plan and user guide and introduced several questions that informed the user workflow. Did proofreading as well.
Mercymoy Gurmesa	Researched a possible Java framework and offered some edits to both documents.
Seth Kendrick	Got the group organized and made major text and formatting contributions to the project plan. Produced the entire test plan format and most of the starter content.
Daniel Oliver	Wrote scenario and concept explanation sections.

Week 4 (November 12-18, 2018)

Evans Ampomah	Developed test protocols for next phase.
Shafro Batyrov	Provided preliminary GUI, edited documentation, compiled submission archive.
Mercymoy Gurmesa	Wrote a basic command-line implementation of the application in Java
Seth Kendrick	Built GUI and compiled documentation
Daniel Oliver	Built basics of web API, built decision flowchart and provided database structure

Week 5 (November 19-25, 2018)

Evans Ampomah	--
Shafro Batyrov	Corrected formatting
Mercymoy Gurmesa	Provided Java secure-coding guidelines
Seth Kendrick	Combined three documents into this single document
Daniel Oliver	Published web API to shared repository, created and assigned tasks in with the associated project management tools.

Week 6 (November 26 - December 2, 2018)

Evans Ampomah	--
Shafro Batyrov	--
Mercymoy Gurmesa	Tested a local implementation of the application.
Seth Kendrick	--
Daniel Oliver	Built more API endpoints and documented them.

Week 7 (December 3-9, 2018)

Evans Ampomah	--
Shafro Batyrov	--
Mercymoy Gurmesa	Tested live API.
Seth Kendrick	--
Daniel Oliver	Added “Running the Application” appendix. Rearranged some items in the documentation and cleaned up some formatting. Assisted Mercymoy in testing efforts.

Week 8 (December 10-16, 2018)

Evans Ampomah	
Shafro Batyrov	
Mercymoy Gurmesa	
Seth Kendrick	
Daniel Oliver	Some bug fixes, added logging to the API, and contributed to the lessons learned and recommended improvements sections.

Appendix B: Running the Application

The application is broken into two parts: the desktop application, and the remote API it connects to.

AgreeAPI

https://dev.azure.com/DOLIVER28/group6/_git/AgreeAPI

1. In order to support multiple users across multiple devices, we store question data on a remote server to be accessed through our API. To run this service locally, you will need to be running **PHP 7+** and **MySQL 5.5+**. The application may work on older versions, but it has not been tested on them.
2. Create an empty database. Name it however you'd like.
3. Add your environment's information to an agreeConfig.ini file one directory up from your root. It should look something like this:
   ```
   hostname = localhost
   database = agreedb
   username = root
   password = “lengthypasswordhere”
   port = 82
   ```
   Note that you can leave out port altogether if your database instance is accessed through the default port.
4. Make an HTTP GET request to your localhost to confirm the API is running. Because you have not submitted any credentials, you should receive an error: *Could not authenticate!* That's the expected behavior.
5. Use the API guide sample requests (using your own test credentials) to interact with the API.

AgreeDesktop

https://dev.azure.com/DOLIVER28/group6/_git/AgreeDesktop

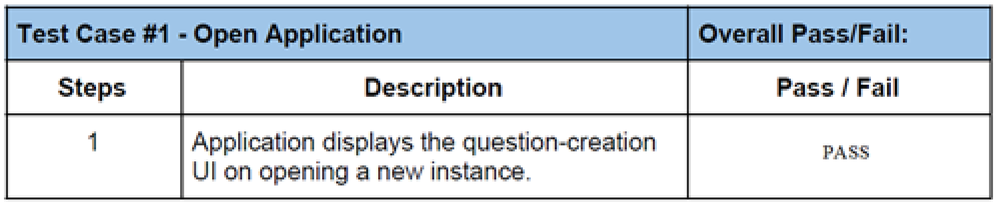
1. With JDK 8+ installed, open the AgreeDesktop repository in your preferred IDE. This has been tested in NetBeans and Visual Studio Code.
   a. For Visual Studio Code users, ensure you set configuration->console to “internalConsole” in your launch.json file in the .vscode folder. This ensures any debugging messages will be readable from Visual Studio Code.
2. Ensure the API is reachable.
   a. If you are attempting to reach your localhost, update the “apiUrl” string in GUI_Class.java. It appears three times (approximately lines 80, 193, and 471).

    b. If you are attempting to reach the live API, try reaching it through your browser to ensure both your internet connection and the API are working.

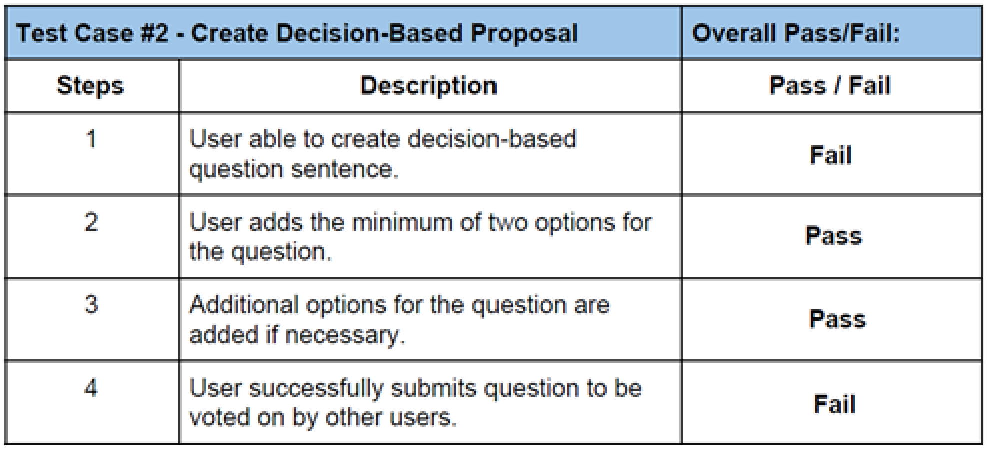
3. Compile and run the application.

Appendix C: Tests

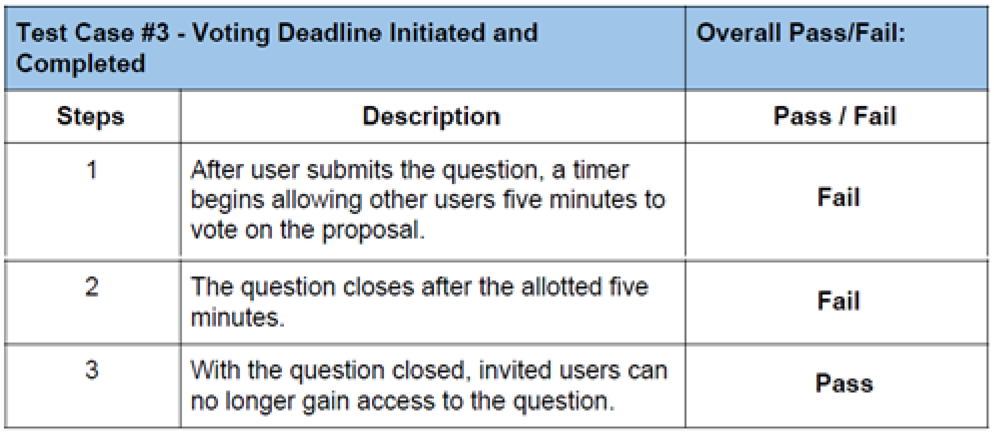
Mercymoy tested the application during Phase 2 of this project. His results are below.



Test Case #1 - Open Application		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	Application displays the question-creation UI on opening a new instance.	PASS



Test Case #2 - Create Decision-Based Proposal		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	User able to create decision-based question sentence.	**Fail**
2	User adds the minimum of two options for the question.	**Pass**
3	Additional options for the question are added if necessary.	**Pass**
4	User successfully submits question to be voted on by other users.	**Fail**



Test Case #3 - Voting Deadline Initiated and Completed		**Overall Pass/Fail:**
Steps	**Description**	**Pass / Fail**
1	After user submits the question, a timer begins allowing other users five minutes to vote on the proposal.	**Fail**
2	The question closes after the allotted five minutes.	**Fail**
3	With the question closed, invited users can no longer gain access to the question.	**Pass**

Test Case #4 - Invited Users Vote on the Proposal		Overall Pass/Fail:
Steps	**Description**	**Pass / Fail**
1	Invited users can access the question.	**Pass**
2	Invited users select acceptable options and rank them in order of preference.	**Fail**
3	Invited users submit their voted option.	**Pass**
4	Uninvited users cannot gain access to the question.	**Pass**

Test Case #5 - Display Results		Overall Pass/Fail:
Steps	**Description**	**Pass / Fail**
1	When the decision conditions are met, be it through a hard deadline or the max-time-after-last-vote limit, the application's decision is calculated and displayed.	**Fail**
2	If too few users voted, and the results would single out a user and make their choice obvious, the results *do not* display.	**Pass**

Summary

Test Case	Pass	Fail
Test Case #1	✓	
Test Case #2		✓
Test Case #3		✓
Test Case #4	✓	
Test Case #5		✓