Quail

[Questions You Answer In Lecture]

Design Document

Project Leader: Karen Feng karenfeng@princeton.edu
Katy Ho ktho@princeton.edu
Zachary Kendrick zacharyk@princeton.edu
Margaret Wang mw16@princeton.edu

Section 1 - Overview

Base implementation:
Have you been in a lecture setting where people are too shy or too afraid to speak up, even though they might have legitimate questions? Or perhaps, the questions that are being asked have been answered already, or interrupt the flow of the lecture? *Quail* makes in-class Q&A easy for both lecturers and students.

Quail provides a web-based platform for live Q&A, thereby facilitating better communication between professors and students in the lecture setting.
Quail allows enrolled students (and lecturers) to submit questions during the course of a lecture, which other students accessing the app may upvote or downvote in real-time to their liking. Moreover, through the use of tags, semantic parsing and statistical analysis, the app will recommend questions similar to a submission post prior to a posting, thereby discouraging question redundancy.
At the end of lecture, the professor can access the app to view the top questions, which can be ordered based on an algorithm sorting by time or popularity, and answer the ones most upvoted. The selected, answered questions may then be archived (e.g., through export to PDF) for future reference, if the professor desires to save and distribute them to the students for future reference. The professor may also post answers directly through the app.

Quail also allows students to provide feedback to the professor after each lecture, allowing for live communication to improve the course as it is developing.

Expanded implementation:
Quail can also be expanded to Q&A and postings outside of the classroom, to allow for a more generalized form of information exchange. The expanded app may be used for live Q&A regarding more casual, non-academic topics (and subtopics), organized as tags, possibly including, but not limited to, the following:

- Frist
    - Late Meal
    - Study Rooms
    - Tabling
    - Performances
- "The Street"
- Lost and Found items
- General Building queries;
    - Academic buildings (e.g., McCosh Hall)
    - Residential buildings (e.g., dorm rooms)
    - Service buildings (e.g., U-Store, Wawa)
- Room Draw
- Student Organization Events
- Public Speaker Events
- Transportation Services
- Public Safety queries
- Campus Life
- Miscellaneous

The expanded version would allow for a fast, convenient method of Q&A through crowdsourced information exchange. For instance, rather than having multiple people inefficiently asking whether or not Charter is "pass-only" this Friday night through multiple texts and relays of information, *Quail* would provide a definitive answer through a single posted question under "The Street" tag.
Like Piazza or Yik Yak, *Quail* will have an interface that allows for follow-up comments and answers to a posted question, in the case of an ambiguous or incorrect answer posting. Furthermore, *Quail* will have features such as "pinned" questions or posts, for particularly important or interesting information, which will be displayed at the top of a topical category no matter its popularity / time rank.
The expanded *Quail* would use the same design and business logic as the base implementation, but would allow for Q&A and postings in general topic categories (such as the ones listed above) in addition to the lecture courses.

Section 2 - Requirements and Target Audiences

Quail provides a pivotal service for our user base. Academic apps such as Piazza are built for an academic setting outside of the lecture hall and do not allow for popularity filtering, live Q&A apps such as Slido are expensive and targeted purely toward conferences, and social apps such as Yik Yak are not built for the formal academic environment. Furthermore, all of these lack key functionalities that *Quail* will provide: live feedback for lecturers and keyword-based moderation to avoid repetitive questions.

Base implementation:
Quail is targeted towards both Princeton students and faculty, authorized through the Central Authentication Service (CAS). As mentioned above, *Quail* would allow for a more efficient means of Q&A during lecture, solving the inconveniences of redundant, inappropriate questions and allowing the most interesting, popular topics to be the most visible to the professor.

Expanded implementation:
Quail is meant for all Princeton University users, authorized through CAS. The expanded implementation would allow for crowdsourced information exchange for other topics besides academic courses. It offers more formalized, organized, and moderated services than Yik Yak, and expands the features of Piazza to a more general audience and information base.

Section 3 - Functionality

Base implementation:

<u>Use case - Lecturer</u>
The lecturer logs into *Quail* using the CAS and enrolls as the lecturer of their class. Before the lecture, the lecturer can choose to submit pinned questions that they believe to be of interest to the students. In the final few minutes of lecture, they decide to answer the most popular questions, upvoted by the students, which can then be archived in PDF form, if they would like to distribute them for future reference.

<u>Use case - Student:</u>
The student logs into Quail using the Central Authentication System (CAS) and enrolls as a student of their classes.
During lecture, if the student has a question about a certain topic...

1. They browse existing questions or search for important keywords/tags in their question (either by time or popularity), and find that no one has asked it. They can then submit it using *Quail*.

2. They browse existing questions or search for important keywords/tags and find that someone has already asked the same question. They upvote the question.
3. They attempt to submit their question, and *Quail* comes up with recommendations for potentially similar questions. They upvote the most similar question.
4. They attempt to submit their question, and *Quail* comes up with recommendations for potentially similar questions. They believe that their question is unique (i.e., not in the recommendations) and submit it.
5. After lecture, the web app will ask the student "How was lecture for you?", where the student can provide feedback to the professor about the lecture. [tentative plan]

During lecture, if the student is just browsing the questions...

1. They see a question that they would like to be answered, and upvote it.
2. They see a question that is inappropriate, and downvote it.

Expanded implementation:

<u>Use case - general Princeton user</u>

The student or faculty member logs into *Quail* using the CAS and is presented with the homepage, designating two options: 1) class enrollment, and 2) campus life. Should the user choose option 1, he will then be presented with enrolling as a student or lecturer of the class. He will then be asked to enter in the class that he wishes to enroll in. The usage is then the same as the baseline implementation use case described above.

Should the user choose option 2, he will then be directed to the *Quail* campus life Q&A webpage. The student selects the tag that is most relevant to them, given suggestions that are based on the most popular or recent tags at the time. If a student has a question about a particular topic relevant to the current tag:

1. They browse existing questions or search for important keywords/tags in their question (either by time or popularity), and find that no one has asked it. They can then submit it using *Quail*.
2. They browse existing questions or search for important keywords/tags and find that someone has already asked the same question. They upvote the question.
3. They attempt to submit their question, and *Quail* comes up with recommendations for potentially similar questions. They upvote the most similar question.
4. They attempt to submit their question, and *Quail* comes up with recommendations for potentially similar questions. They believe that their question is unique (i.e., not in the recommendations) and submit it, with a relevant tag.

If the user sees a question that they wants to answer, then they can click on the question and take one of several options:

1. If the question is not answered already, the user can post his answer in the provided answer section.
2. If the question is already answered, but the answer is not satisfactory to the user, the user can post in the provided follow-up area to the question. The follow-up section will be a place where any user can post comments to discuss the question and the provided answer.

If the user is just browsing questions by searching through either a relevant tag or the suggested popular tags, then they have several options:

1. Upvote questions that are relevant to them and/or have satisfactory answers;
2. Downvote questions that are irrelevant, inappropriate, or invalid and/or have unsatisfactory answers;
3. Continue browsing as they see fit.

If the user believes a question to be particularly important or relevant to a topic, they can also pin the question to a tag, which will post the question to the top search result of the selected tag, regardless of time or popularity rating.

Section 4 - Design

Technical infrastructure

Frontend

- Heroku, designed with Bootstrap, as the main web interface
- Login: primary interface
    - Security validation through CAS
    - Separate login options for student and instructor/faculty
- Homepage: Two categories
    - 1) Enroll in a class (base implementation)
    - 2) Campus life (expanded implementation)
- Lecture Q&A page:
    - Questions pinned by professor at the top
    - Questions posted in lecture so far, with options to sort by time, number of upvotes, number of views
    - Search box for similar questions
- Campus Life Q&A page:
    - Tags for different topics and ability to switch between these tags
    - Questions posted for each tag so far, with options to sort by time, number of upvotes, number of views
    - Search box for similar questions
    - Search box, default results; home page; color scheme

Backend

- Business logic powered by Django web framework

- Database: mongoDB
    - Non-relational, document-based database; more convenient for storing user posts/comments data
- Decision not to use AJAX; but should we find that it would be better to do so, will use jQuery to connect infrastructure

Hosting and source control

- Hosted on Heroku
- We will use Github for version control, collaborative code editing, and issue tracking
- Using Asana for general communication, planning, scheduling

Section 5 - Timeline

Week 1: 03/13-03/19

- Create design document
- Create website
- During spring break, all group members will go through tutorials to understand the basics of working with the Django framework, and integration with a mongoDB database
- Solidify ideas of how the data should be represented in the database
- End of spring break: come up with pieces of preliminary code and algorithmic design needed for the base implementation

Week 2: 03/20-03/26

- Complete a basic front-end design
- Putting pieces together: integrate Django framework with Heroku
- Create prototype: the base implementation

Week 3: 03/27-04/02

- Be able to enroll into a class as a student/professor
- Be able to post, view, and upvote questions

Week 4: 04/03-04/09

- Implement suggested similar questions/search questions using keywords
- Begin testing in a classroom
- Prototype for expanded implementation, with the topics organized by tags

Week 5: 04/10-04/16

- Create alpha: working version of baseline implementation
- Thorough testing in a classroom

Week 6: 04/17-04/23

- Create beta: working version of both baseline and expanded implementation
- Thorough testing in a classroom and around campus

Week 7: 04/24-04/30

- More testing and bug fixes

Week 8: 05/01-05/07

- Demo!

Section 6 - Risks and Outcomes

- Possible expansion to a campus-wide setting with questions focused on specific parts of campus (eg. Frist Late Meal), with answers that may be upvoted.
- Integration with CAS: must-have for the base implementation, but necessary for the more generalized expanded implementation?
- Experience/knowledge:
    - Karen is the only one with experience using Heroku
    - Members have little experience with dealing with back-end services
        - Will have to go through a pretty steep learning curve before being able to start the base implementation of the project
- User problems:
    - Networking issues: how do we get students and professors to actually use this?
        - Will have to reach out to friends / well-acquainted professors to test-run the app during the production process
- Implementation problems:
    - Separating lecturer vs student users (in base implementation)
    - Categorizing information (in expanded implementation)
        - There are bound to be questions that cannot be appropriately categorized or that fit across multiple categories; we must find a way to allow these questions to be posted and searched for without getting lost inside the semantics of the app
    - Invalid answers (in expanded implementation)
        - How should we deal with wrong or inappropriate information?
            - In Piazza, the instructor can correct a student answer, or flag it as inappropriate / have admin privileges to delete it; with the more generalized Q&A in a casual setting, we have to be careful to not let it degenerate into Yik Yak, where potentially inaccurate/offensive/inappropriate information can be propagated
            - May have to implement some sort of moderation / oversight, which would require much more involvement with the data exchange
    - Too many features (pinned posts, separating lecturers vs students, follow-up answers/comments) may weigh down the lightweight concept of the app