Geon Kang
NetID: gk698
NYUID: N17120399
Open Source Tools
GA.2246-001: Fall 2014

# Final Project Documentation

## 1. Helpful Info when Testing Project

- A link to ask a new question and functionality to upload an image to the app will appear at the header once the user logs in. This header will be at the top of every page of the app (with the exception of the xml page).

- On image upload, the user will be provided with a permalink to the successfully uploaded image. The permalink is the serving url provided by the google appengine images api which does not contain the filename and filetype. (The project requirements only specified that a permalink to the image must be returned, so the format of the serving url was not modified.)

- A question can be viewed by clicking on the title. Due to this design choice, an empty title is given a default "(No Title)" title.

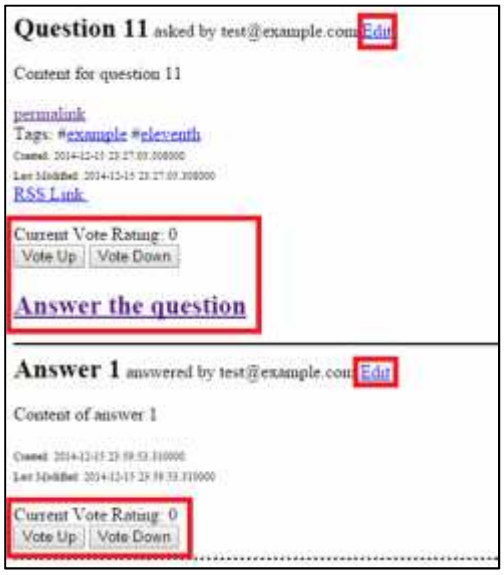
- Each question shows its content up to 500 characters. There is a permalink below the question content that redirects to a page which shows the question's content in its entirety.

- Tags (if any) are located beneath the permalink. A specific tag link can be clicked to show just the questions that are tagged with that specific tag.

- If there are more than 10 questions on a page, a next page link to view older questions will be at the bottom left corner of the page.

- When viewing a question, functionality to vote up and down, as well as a link to answer a question will only appear if a user is logged in. These functionalities will appear below the "Current Vote Rating" description.

- When viewing a question, questions and answers can only be edited by the user who created them. The link will appear next to the author description if the question or answer belongs to the currently logged on user.



# 2. Developer's Guide

## Overview

This web app is a simple question and answering app with vote functionality. The detailed specifications of the app can be found on the NYU OST Fall 2014 course website:
https://docs.google.com/document/d/1OqJjptD_kvq4QB8ANwublXD4hlVt628t0fC7VOeb9Dk/preview

This app was coded in python 2.7, deployed with Google App Engine and uses jinja2 to render HTML. Question, Answer, and Vote data are stored in Google App Engine's datastore using the ndb Datastore API. Uploaded images are stored with Google App Engine's Blobstore API.

## Datastore Entities

Question, Answer, and Vote are the Entities stored in the Datastore. They all are given ancestor keys in order to ensure that newly created Entities are quickly picked up by subsequent queries. Question and Answer Entities are given the same ancestor "QAlist" to make sure that Google App Engine doesn't give questions and answers the same random ID. This is necessary because the Vote Entity uses a single entityID field to represent which question or answer it belongs to rather than differentiating between question votes and answer votes. Question Entity has a last active time field that updates whenever an activity related to the question occurs. Questions are sorted in reverse order on this field. The last modified time for Question and Answer Entities are updated manually only when an edit occurs. Answer Entities stores its vote count in the field voteCount so that answers can be ordered on that field. The Vote entity has a voter field which holds a user google ID. This is to ensure that a user can only have one vote on a question or answer. Questions and Answers created with no title are given the title "(No Title)" by default.

## Helper Functions

**convertLinks():** Takes in a string and adds link html tags to any links that exist in the tag. If the link ends in .jpg, .png, or .gif, it adds an image inline html tag to the link instead. Question and Answer content are converted through this function before being stored.

**uncovertLinks():** Undoes the work of convertLinks(). The main purpose is to remove the added html tags from the Question and Answer content before using it to fill the edit form.

**votelist_key():** Returns the key for the Vote Entity's ancestor.

**qalist_key():** Returns the key for Question and Answer Entities' ancestor.

## Non-RequestHandler Class & Generated HTML

**header():** Checks whether the user is logged on and generates the appropriate login or logout url. Users are redirected to the main page on login or logout. header() also generates a upload url for the Blobstore for the purpose of uploading images.

**header.html:** Contains a link to the main page and a login or logout link depending on user login status. If the user is logged in, the header displays a link to ask a new question and forms to upload a picture. The header is at the top of every page of the app with the exception of the xml page as an administrative/navigational menu.

## RequestHandler Classes & Generated HTML

**MainPage():** This RequestHandler generates the main page which shows a list of all the questions. With a get request, this handler queries for all questions in reverse order of last active time with a cursor to get 10

questions. If there are more than 10 questions and the next page link is clicked in main.html, the next 10 queries starting at the cursor will be returned.

A post request on the main page only occurs when a new question is created. Thus, the handler checks if the question was actually submitted and not canceled by checking for the variable submitq. If submitq is in the request, the handler creates a Question entity, populates it with all the submitted information, and puts it in the Datastore. Then the handler queries for all questions in reverse order of last active time using a cursor and returns 10 questions. If there are more than 10 questions and the next page link is clicked in main.html, the subsequent get request on the main page will pick up from where the cursor was and return the next 10 questions.

**main.html:** Loops through the questions passed by MainPage() and shows displays question title, author, content, tags, creation date, and last modified date for each question. Each question title is given a link which links to the view page ('view' handled by the View() RequestHandler) with the question ID passed with the get method. Tags are each made into a link which links to the tag view page ('/taglist' handled by the ViewTaggedQuestions RequestHandler) with the tag passed with the get method. Question content on main.html is cut at 500 characters, and a permalink to the view permalink page ('/permalink' which is handled by the ViewPermalink RequestHandler) with the question ID passed with the get method, is provided which will allow users to view the entire content on a separate page. If there are more than 10 questions, a next page link is generated. When clicked, this link requests the main page with the cursor's last position passed with the get method. The MainPage() RequestHandler handles this request and renders main.html with the next 10 questions if necessary.

**Create():** This RequestHandler generates a page with the form to create a new question. Since the '/create' request link is only viewable in the header by logged in users, logged out users should not be able to reach this RequestHandler. However, just to be safe, Create() checks to see if the user is logged on to prevent non-logged-in users who accidentally stumble upon 'create.html' from creating new questions.

**create.html:** If the user is logged in, the forms for creating a new question are generated. The values are sent with the post method to the main page along with either the variable cancelq or submitq depending on the submit button pressed. As mentioned above, MainPage() will only create a new question when it finds submitq. If the user is not logged in, the page simply displays a message that asks the user to log in to ask a question.

**CreateAnswer():** Similar to Create(), this RequestHandler generates a page with the form to create a new answer. CreateAnswer() also checks if the user is logged on.

**createanswer.html:** If the user is logged in, the forms for creating a new answer are generated. The values are sent with the post method to the view page ('/view' handled by the View() RequestHandler) along with either the variable cancela or submita depending on the submit button pressed. View() will only create a new answer when it finds submita. If the user is not logged in, the page simply displays a message that asks for the user to log in to answer a question.

**View():** This RequestHandler generates the question view page which shows the question and all the answers to the question along with the votes for the question and each of its answers. A get request indicates that nothing was changed. (The question wasn't edited, no votes were made or changed, no answers were added or edited.) So with a get request, the handler checks if the user is logged in, gets the question ID that was passed in and gets the question with the question ID from the Datastore. It then queries for votes with entityID = question ID and sums them up to get the vote count. Finally, the handler fetches all the answers

associated with the question using the Answer.questionID field . Answer votes do not have to be counted because they are stored in each answer entity.

With a post request, there are many events to check for:

- If the request does not contain the value ‘cancelq’: An event has occurred, so get the question with the passed in question ID and update the last active time to now.
- If the request contains the value ‘submitq’: a question has been edited. Get the question with the passed in question ID and update the field values with the passed in values.
- If the request contains qupvote or qdownvote: a question has been voted on. If the vote on the question for the current user exists, get it and modify the vote value accordingly. If the vote on the question for the current user does not exist, create the respective vote.
- If the request contains aupvote or adownvote: an answer has been voted on. If the vote on the answer for the current user exists, get it and modify the vote value accordingly. If the vote on the answer for the current user does not exist, create the respective vote.
- If the request contains the value ‘submita’: an answer has been created or edited. If the answer with the passed in answer ID exists, get the answer and update the field values with the passed in values. If the answer with the passed in answer ID does not exists, create an answer and populate it with the passed in values.

The votes for the question are fetched and summed to get the total vote count for the question. Also, if an answer was voted on, the vote counts for the answer are fetched and summed to update the total count for the answer as well.

**view.html:** The question information is generated in the same manner as the main page with the addition of an RSS link (‘/rss’ handled by RSSHandler with the question ID passed with the get method) and a vote rating display below the last modified time. If the user is logged on, buttons to vote up and down are rendered below the question and each answer. The vote buttons request the view page again, passing the various values mentioned in the View() description with the post method. If the question or answer author ID matches the current user ID, then an edit link (‘/editq’ or ‘/edita’ handled by EditQuestion or EditAnswer) is rendered next to the author description.

*Note: The reason for View() being so long is due to the design choice. It makes the most sense for users to see the view page again after a question/answer edit or a vote. Thus ‘/view’ gets the most requests giving rise to the various condition checks.

**ViewPermalink():** This RequestHandler generates a page that shows the entire question content. It gets the question ID that was passed in and gets the question associated with the question ID from the Datastore.

**permalink.html:** Displays the question with entire content on a separate page. Exactly the same as how an individual question is rendered on the main page with the exception of the question content not being capped at 500 characters.

**ViewTaggedQuestions():** This RequestHandler generates the page that lists the questions with a specific tag. It is exactly the same as MainPage() get request handler except that it queries for questions with the tag that was passed in. Only clicks on tags create this request so there is no post request handler.

**taglist.html:** Exactly the same as main.html except that the next page link (if necessary) requests ‘/taglist’ again and passes in the tag along with the cursor position.

**EditQuestion():** This RequestHandler generates the page with the forms to edit a question. It gets the relevant question from the Datastore using the passed in question ID, and also double checks if the current user has permission to edit the question.

**editq.html():** Exactly the same as create.html except the forms have the default values filled with the values of the question that was retrieved by EditQuestion(). If the user does not have permission to edit the question, a no permission message is displayed instead of the forms.

**EditAnswer():** This RequestHandler generates the page with the forms to edit an answer. It gets the relevatn answer from the Datastore using the passed in answer ID, and also double checks if the current user has permission to edit the answer.

**edita.html:** Exactly the same as createanswer.html except the forms have the default values filled with the values of the answer that was retrieved by EditAnswer(). If the user does not have permission to edit the answer, a no permission message is displayed instead of the forms.

**ImageUploadHandler():** This BlobstoreUploadHandler stores the uploaded image as a blob and uses the Google App Engine images API to get the serving url.

**upload.html:** Displays a message saying that the image has been uploaded and provides a link to use the image.

**RSSHandler():** This RequestHandler generates an xml page for a question and its answers. It gets the question from the Datastore using the passed in question ID, and fetches all the answers to the question using the by querying Answer.questionID = question ID. It also stores the site url for the purpose of providing links in the xml.

**rss.xml:** Question and its Answers in RSS format. Only the <title><author><description> and <link> tags are used for Question and Answers. Custom tags were not used for information such as vote count, created date, modified date as custom tags do not offer much without adding namespaces. The additional information could have been added to the description tag, but I chose to leave it out.