Charles Drews
N11539474
csd305@nyu.edu
Open Source Tools Final Project
December 16, 2013

This document first describes how my project was designed and coded, then addresses how each specific requirement was met.

The project consists of a Python script (main.py) with separate handlers for each type of request a user can make, along with Jinja2 templates to render the output to the user. The following list explains the purpose of each handler and corresponding template:

- The MainPage handler checks whether the user is logged in, queries what blogs exist, and feeds that information to the template index.html. This provides the root page of the app at http://charlesdrews-blog.appspot.com/. If the user is logged in, this page lists all the user's blogs, provides a form to create a new blog, and provides a link to a page where images can be uploaded. If a user is not signed in, this page instead prompts the user with a login link. This is accomplished with an if statement in the template. Whether a user is logged in or not, the bottom portion of this page also lists all blogs hosted on the application with links to view each blog.
- The CreateBlog handler processes requests from the "create blog" form by creating a new datastore object for the new blog, and redirecting back to the root page of the app.
- The ViewBlog handler queries the 10 most recent posts from the selected blog, trims each post body to 500 characters, passes the post body through a parsing function which wraps image URLs in an <img> tag and other links in a <a href> tags, then feeds the data to the blog.html template.
    - If the user is the blog's author, the page provides a link to a page where a new post can be created, and each post displayed in the blog view includes an "edit post" link.
    - If the user is not the blog's author, the blog view simply shows the blog posts with no "edit post" links or link to create a new post.
- The CreatePost handler does two things:
    - The get() method provides the necessary data to the addpost.html template, which provides the user with blank forms to enter a new post's Title, Body, and Tags, and a submit button that calls the post method of CreatePost.
    - The post() method creates a new datastore object for the new post, then redirects the user back to the blog view.
- The ViewPost handler processes the action of a user clicking on a post's permalink. It gets the selected post object from the datastore and passes it to the post.html template for viewing.
- The EditPost handler is another two parter:
    - The get() method provides the necessary data to the editpost.html template, which is basically just like the addpost.html template, except the forms are pre-populated with the values from the selected post.
    - The post() method updates the selected posts with the newly provided input, then redirects the user back to the blog view.
- The ImagesHandler responds to logged-in users clicking on the link from the home page to upload images. This handler gets an upload URL from the blobstore and queries for existing uploaded images from the user, then provides both to the images.html template, which gives the user a form to select & upload a file and lists all previously uploaded images with their permalinks.
- The UploadHandler responds to users submitting a file for upload by storing the file in the blobstore, creating a datastore object associating the file with the user, and redirecting back to the image page, which now includes the newly uploaded image in the list of uploaded images.
- The ServeHandler responds to a user accessing the permalink for an uploaded image by retrieving that image from the blobstore and providing it to the browser.
- Finally, the FeedHandler responds to users clicking on the RSS links in each blog's blog-view by querying for all posts from that blog and sending the data to the rss.xml template.

There are a few additional pieces of code not described in the above list:

- main.py includes class definitions for the three datastore object types used by the app: Blog, BlogPost, and Image.
- main.py also includes a function named global_parent_key(). This returns a the same key each time, to a non-existent datastore object of kind “Global-Parent” with ID “charlesdrews.” The purpose of this key is to be used as a parent for all the blog objects. Having a common parent for all blog objects allows me to query for all blogs from all users in MainHandler and show them in alphabetical order. Without this common parent, I was not able to sort that query.
- There is a small CSS file (stylesheets/main.css) which styles the html templates.
- There are two yaml files (app.yaml and index.yaml) which configure the app and describe the datastore indexes.

Next, in order to demonstrate how my project meets the necessary requirements, I have addressed each requirement below:

1. *The system should handle multiple users and each user should be able to create one or more blogs.*

   The app uses the users library from google.appengine.api to create login/logout URLs which are displayed in the upper left of all pages. The app associates each Blog, BlogPost, and Image object with the user that created it via a UserProperty attribute, and allows a user to create an unlimited number of blogs. Also, certain functionality (creating posts, editing posts) is limited to a blog's author, via checking the current user vs. the blog's author attribute in both Python and in the html templates. Furthermore, the Python handlers that process such restricted activities always reconfirm the user before performing their task, and if the user is not the blog's author the user is redirected as necessary.

2. *A user can select a blog that they own and write a post to that blog. A post consists of a title and a body, which will be entered via a CGI form.*

   From the homepage, if a user clicks on a blog for which the user is the author, then the blog view page provides a link for creating a new blog. When that link is clicked, a page with the necessary CGI form is provided.

3. *Blogs can be viewed without an account or login. When viewing a blog, it will show at most 10 posts on a page, with a link to go to the next page of older posts (you don't need a "newer posts" link).*

   Even if the user is not logged in, all existing blogs are still listed on the homepage and can be viewed when clicked on. For all users, logged in and not, when the blog view is shown, only the 10 most recent posts are displayed, according to the creation date of the post (as opposed to the edit date of the post), along with a link to more older posts if necessary.

4. *When multiple posts are shown on the same page (the standard blog view), each post will display the content capped at 500 characters. Each post will have a "permalink" that, when followed, shows the complete content of the post on its own page.*

   Each post body is clipped to 500 characters in the ViewBlog handler before being sent to the blog.html template. Each post is accompanied by a permalink of the format “http://charlesdrews-blog.appspot.com/post/{blog id}/{post id}” where both {blog id} and {post id} are numerical values assigned by the datastore that uniquely identify each object within its “kind” (kinds are Blog, BlogPost, and Image). The permalinks are handled by the ViewPost handler.

5. *Posts should be stored along with a timestamp when the post was created. Posts can be edited, in which case the modification time is stored (and the creation time is unchanged); these timestamps will be shown anywhere a post is shown. The form presented to a user to edit a post should have the original contents (title and body) filled in by default.*

   If the user is the blog's author, then each post is accompanied by an “edit post” link. That link brings the user to a page with CGI forms pre-populated with the selected post's values. When the user clicks the “Edit Post” submit button, the post object's edit_date attribute is automatically overwritten with the current date/time, while the create_date attribute is not updated. The create_date attribute is automatically populated when the post is first created, then never changed.

6. *The author of a post can give the post zero or more tags, like "tech" or "new york".*

   The CGI forms on both the create and edit post pages include a Tags field. The CreatePost and EditPost handlers parse the input to the Tags field using a comma as a delimiter. The tags attribute of the BlogPost object is a repeated attribute consisting of a list of strings.

7. *Posts can be searched for by clicking on a tag, which means only posts with the given tag are displayed on the page (again, at most 10 with a link for older posts). The list of tags is generated from the set of posts that have been stored, and will be displayed on the main page of the blog.*

   In the blog view, below each post is a list of tags, each of which is a clickable link that will reload the blog view showing only posts with that particular tag. The blog view also contains a list, displayed to the right of the blog post snippets, of all tags that exist for that bog. Once again each is a clickable link that will reload the blog view showing only posts with that particular tag. When a tag has been selected, it is specified at the top of the list of blog post snippets so the user is reminded what tag they clicked on. Also the same 10 most recent posts with a link for more if needed approach applies.

8. *When posts contain links (text that begins with http:// or https://), they will be displayed as HTML links when viewed. If a link ends with .jpg, .png, or .gif, it will be displayed inline rather than as a link.*

   Blog post bodies are passed through a parsing function that wraps image URLs in an <img> tag and other links in <a href> tags before they are sent to the html template for display. This same parser also replaces "\n" characters with <br> tags so the post displays as the user intended. It is important to note that these modifications are not stored in the datastore; they are made after the posts are retrieved via query and are not saved. This way, if the user later edits a post, when the existing body value is pre-populated in the edit post form, the links show up as regular text, just like how the user originally entered them.

9. *Images can be uploaded. These will be available via a permalink after uploaded, and can be referenced using links in the posts.*

   From the homepage a logged-in user will see a link to an images page. This page provides a CGI form for uploading new images, and displays a list of previously uploaded images with their permalinks. The user is provided with instructions to copy & paste those permalinks into a blog post body in order to use the images in a post.

10. *Each blog will have an RSS link, that dumps an entire blog in XML format (see wiki page for example).*

    The blog view provides the RSS link for each blog in two places: at the very bottom, below the blog post snippets, and on the right side underneath the list of tags. Clicking the RSS link brings the user to an xml permalink that can be copied and pasted into any RSS reader.

Additionally, the Git repository for my project contains the following features:

- In the "master" branch there is a tag named "v1.0" which marks the commit of the production version of my code that is currently deployed at http://charlesdrews-blog.appspot.com/.
- There is a separate branch named "experiment". In that branch there is a tag also named "experiment" which marks a commit where the RSS links are turned off (via commenting them out in the blog.html template).
- I used a .gitignore file to ignore files that were not needed (.pyc and .swp in my case).