Wiki Formatting Design Document

Jean Rivera - jeanrivera@google.com
Rahman Gamble - thegamble@google.com
Darryle Mensah - darrylemensah@google.com

Background

The editing format of a wiki can be confusing and over complicated to some users. Our wiki implements a simplified HTML format that users can follow easily, not giving them trouble when using it.

Proposal

Editing a page is a main feature on our Wiki, one that is accessible to all users. In order to implement it, we have to take into account various parts of the formatting we will be using. The formatting will determine how the text will be organized. Based on different tags, it knows if the text is a header, a new paragraph or a link.

The Wiki Formatting will be implemented primarily through using flask methods and functions. We will utilize the HTML **<form>** element to create fields for the user to enter information. This information will be sent using HTML's **POST** method to a route dedicated to handling page edits. After the information is put through a filtering function that strips potentially harmful HTML tags, the information will be placed into a .txt file. This .txt file will then be plugged into a corresponding .html file, and its contents will be displayed onto the given page.

Implementation

Each page will have an .html and .txt file. The .html file will be the template for the information displayed on the page, while the .txt file will contain the page name and the raw HTML code.

We will use a few different HTML templates:

1. **<pagename>.html**
    a. This will be the html template for any page created on the wiki.
2. **edit_page_form.html**
    a. This will contain a template with inputs for the HTML text and a small description of what changes were made. The HTML box will be pre-filled with the current contents in the page's assigned .txt file.
    b. After the form is submitted, it will send back two variables called **contents** and **changes**.

The HTML tags that are allowed are:

1. <h1> </h1> - For main heading
2. <h2> </h2> - For subheading
3. <h3> </h3> - For sub-subheading
4. <p> </p> - For separated paragraphs
5. <a> </a> - For links can insert href for a hypertext reference on a target.
    a. Can insert href for a hypertext reference on a target. Ex: <a href=http://localhost:8080" >Home Page</a>

We will also create two functions to parse and add the new pages to the wiki. Each function will have a dedicated route assigned to it. The functions are the following:

1. **get_edit_form()**
    a. Route: @app.route("/view/page-edit-form")
    b. Runs when the user presses the Edit button on a wiki page. It brings up the form to edit the page.
2. **edit_page()**
    a. Route: @app.route("/view/handle-page-edits", methods=["GET", "POST"])
    b. This will read a from an HTML template called **edit_page_form.html**
    c. Using the request method from flask, we will extract the content passed through form submission (from edit_page_form.html) into two variables, **con** (contents) and **cha** (changes).
    d. This information will be passed into the page's text file, and will filter out any HTML tags that are not <h1>, <h2>, <h3>, <p>, or <a> utilizing the **readlines()** method.
    e. If a line contains an invalid tag, we will use flask's **escape()** method, which changes HTML characters in a string into safe text, and inform the user by opening an alert window.
    f. The function then reads the .txt file and places its contents through flask's **Markup()** method, which will safely convert the file contents into an HTML string.
    g. Finally, the filtered information is placed into the **<pagename>.html** template and displays the information on the page through the **render_template()** function.

Security Considerations

Our wiki design will not allow user script insertion. It will have a filtering system, where, when the user submits an input, it looks at the input to see if it has any dangerous keywords, such as <script></script> tags. If any keywords are found, they will be removed using flask's **escape()** method.

It is also noted that links are potentially dangerous, but we wanted to limit how much freedom of editing we were taking away.

Alternatives Considered

Content Security Policy

For Security, one alternative that was considered was having the HTTP Content Security Policy script-src implemented. This will only allow scripts coming from the specified <source> to be executed. We opted to go for filtering dangerous tags, since it seems like the safer option.

Library for filtering

Our filtering function is probably available in some other library, but we were unable to find one that did exactly what we desired.

User-inputted HTML

The intuition behind allowing the user to write HTML is essentially that we will allow only the use of header, paragraph, and hyperlink elements, which are fairly simple to implement. Other established wikis also seem to utilize HTML, so the hope is that if someone is editing, they will have a good chance of knowing what they are doing. However, we decided it would be wise to not underestimate an ignorant user. We will include a warning message stating that tags other than the ones allowed will be removed. We also decided to include a brief overview of the three tags required to format.

Resources

- HTML General Formatting
- HTML Forms
- Creating Hyperlinks
- XSS
- XSS Preventions
- Flask Escape() and Markup()