CS50's Web Programming with Python and JavaScript

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

Brian Yu (https://brianyu.me)
brian@cs.harvard.edu

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan)
(https://twitter.com/davidjmalan)

Wiki

> CS50W does not feature a one-to-one correspondence between lectures and projects. If you are attempting this project without having at least watched Lecture 3, you're trying this too soon!

Design a Wikipedia-like online encyclopedia.

Background

Wikipedia (https://www.wikipedia.org/) is a free online encyclopedia that consists of a number of encyclopedia entries on various topics.

Each encyclopedia entry can be viewed by visiting that entry's page. Visiting https://en.wikipedia.org/wiki/HTML (https://en.wikipedia.org/wiki/HTML), for example, shows the Wikipedia entry for HTML. Notice that the name of the requested page (HTML) is specified in the

route `/wiki/HTML`. Recognize too, that the page's content must just be HTML that your browser renders.

In practice, it would start to get tedious if every page on Wikipedia had to be written in HTML. Instead, it can be helpful to store encyclopedia entries using a lighter-weight human-friendly markup language. Wikipedia happens to use a markup language called Wikitext (https://en.wikipedia.org/wiki/Help:Wikitext), but for this project we'll store encyclopedia entries using a markup language called Markdown.

Read through GitHub's Markdown guide (https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax) to get an understanding for how Markdown's syntax works. Pay attention in particular to what Markdown syntax looks like for headings, bold text, links, and lists.

By having one Markdown file represent each encyclopedia entry, we can make our entries more human-friendly to write and edit. When a user views our encyclopedia entry, though, we'll need to convert that Markdown into HTML before displaying it to the user.

Getting Started

- Download the distribution code from https://cdn.cs50.net/web/2020/spring/projects/1/wiki.zip (https://cdn.cs50.net/web/2020/spring/projects/1/wiki.zip) and unzip it.

Understanding

In the distribution code is a Django project called `wiki` that contains a single app called `encyclopedia`.

First, open up `encyclopedia/urls.py`, where the URL configuration for this app is defined. Notice that we've started you with a single default route that is associated with the `views.index` function.

Next, look at `encyclopedia/util.py`. You won't need to change anything in this file, but notice that there are three functions that may prove useful for interacting with encyclopedia entries. `list_entries` returns a list of the names of all encyclopedia entries currently saved. `save_entry` will save a new encyclopedia entry, given its title and some Markdown content. `get_entry` will retrieve an encyclopedia entry by its title, returning its Markdown contents if the entry exists or `None` if the entry does not exist. Any of the views you write may use these functions to interact with encyclopedia entries.

Each encyclopedia entry will be saved as a Markdown file inside of the `entries/` directory. If you check there now, you'll see we've pre-created a few sample entries. You're welcome to add more!

Now, let's look at `encyclopedia/views.py`. There's just one view here now, the `index` view. This view returns a template `encyclopedia/index.html`, providing the template with a list of all of the entries in the encyclopedia (obtained by calling `util.list_entries`, which we saw defined in `util.py`).

You can find the template by looking at `encyclopedia/templates/encyclopedia/index.html`. This template inherits from a base `layout.html` file and specifies what the page's title should be, and what should be in the body of the page: in this case, an unordered list of all of the entries in the encyclopedia. `layout.html`, meanwhile, defines the broader structure of the page: each page has a sidebar with a search field (that for now does nothing), a link to go home, and links (that don't yet work) to create a new page or visit a random page.

Specification

Complete the implementation of your Wiki encyclopedia. You must fulfill the following requirements:

- **Entry Page**: Visiting `/wiki/TITLE`, where `TITLE` is the title of an encyclopedia entry, should render a page that displays the contents of that encyclopedia entry.
  - The view should get the content of the encyclopedia entry by calling the appropriate `util` function.
  - If an entry is requested that does not exist, the user should be presented with an error page indicating that their requested page was not found.
  - If the entry does exist, the user should be presented with a page that displays the content of the entry. The title of the page should include the name of the entry.
- **Index Page**: Update `index.html` such that, instead of merely listing the names of all pages in the encyclopedia, user can click on any entry name to be taken directly to that entry page.
- **Search**: Allow the user to type a query into the search box in the sidebar to search for an encyclopedia entry.
  - If the query matches the name of an encyclopedia entry, the user should be redirected to that entry's page.
  - If the query does not match the name of an encyclopedia entry, the user should instead be taken to a search results page that displays a list of all encyclopedia entries that have the query as a substring. For example, if the search query were `ytho`, then `Python` should appear in the search results.

    - Clicking on any of the entry names on the search results page should take the user to that entry's page.
- **New Page**: Clicking "Create New Page" in the sidebar should take the user to a page where they can create a new encyclopedia entry.
    - Users should be able to enter a title for the page and, in a `textarea` (https://www.w3schools.com/tags/tag_textarea.asp), should be able to enter the Markdown content for the page.
    - Users should be able to click a button to save their new page.
    - When the page is saved, if an encyclopedia entry already exists with the provided title, the user should be presented with an error message.
    - Otherwise, the encyclopedia entry should be saved to disk, and the user should be taken to the new entry's page.
- **Edit Page**: On each entry page, the user should be able to click a link to be taken to a page where the user can edit that entry's Markdown content in a `textarea`.
    - The `textarea` should be pre-populated with the existing Markdown content of the page. (i.e., the existing content should be the initial `value` of the `textarea`).
    - The user should be able to click a button to save the changes made to the entry.
    - Once the entry is saved, the user should be redirected back to that entry's page.
- **Random Page**: Clicking "Random Page" in the sidebar should take user to a random encyclopedia entry.
- **Markdown to HTML Conversion**: On each entry's page, any Markdown content in the entry file should be converted to HTML before being displayed to the user. You may use the `python-markdown2` (https://github.com/trentm/python-markdown2) package to perform this conversion, installable via `pip3 install markdown2`.
    - Challenge for those more comfortable: If you're feeling more comfortable, try implementing the Markdown to HTML conversion without using any external libraries, supporting headings, boldface text, unordered lists, links, and paragraphs. You may find using regular expressions in Python (https://docs.python.org/3/howto/regex.html) helpful.

Hints

- By default, when substituting a value in a Django template, Django HTML-escapes the value to avoid outputting unintended HTML. If you want to allow for an HTML string to be outputted, you can do so with the `safe` (https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#safe) filter (as by adding `|safe` after the variable name you're substituting).

▶ **Using the CS50 Codespace?**

How to Submit

1. Visit this link (https://submit.cs50.io/invites/89679428401548238ceb022f141b9947), log in with your GitHub account, and click **Authorize cs50**. Then, check the box indicating that you'd like to grant course staff access to your submissions, and click **Join course**.
2. Install Git (https://git-scm.com/downloads) and, optionally, install `submit50` (https://cs50.readthedocs.io/submit50/).

When you submit your project, the contents of your `web50/projects/2020/x/wiki` branch should match the file structure of the unzipped distribution code as originally received. That is to say, your files should not be nested inside of any other directories of your own creation. Your branch should also not contain any code from any other projects, only this one. Failure to adhere to this file structure will likely result in your submission being rejected.

By way of example, for this project that means that if the grading staff visits `https://github.com/me50/USERNAME/tree/web50/projects/2020/x/wiki` (where `USERNAME` is your own GitHub username as provided in the form, below) we should see the three subdirectories (`encyclopedia`, `entries`, `wiki`) and the `manage.py` file. If that's not how your code is organized when you check, reorganize your repository as needed to match this paradigm.

3. If you've installed `submit50`, execute

   ```
   submit50 web50/projects/2020/x/wiki
   ```

   Otherwise, using Git, push your work to `https://github.com/me50/USERNAME.git`, where `USERNAME` is your GitHub username, on a branch called `web50/projects/2020/x/wiki`.
4. Record a screencast (https://www.howtogeek.com/205742/how-to-record-your-windows-mac-linux-android-or-ios-screen/) not to exceed 5 minutes in length (and not uploaded more than one month prior to your submission of this project), in which you demonstrate your project's functionality. Be certain that every element of the specification, above, is demonstrated in your video. There's no need to show your code in this video, just your application in action; we'll review your code on GitHub. Upload that video to YouTube (https://www.youtube.com/upload) (as unlisted or public, but not private) or somewhere else. In your video's description, you must timestamp where your video demonstrates each of the seven (7) primary elements of the specification. **This is not optional**, videos without timestamps in their description will be automatically rejected.
5. Submit this form (https://forms.cs50.io/9ad31b82-b5b0-494c-ad19-2910c5dcd45e).

You can then go to https://cs50.me/cs50w (https://cs50.me/cs50w) to view your current progress!