Feature - Archiving of files.

- Programmer: Ryan Guard
- Date: 04/9/2018

Problem Definition

Necessary portion for the archive Database system. Edits of the markdown files aren't currently documented in any fashion and there is no way of reverting or tracking file changes. Users have requested the feature for having the data of a markdown file change to be tracked in a SQL database. This allows for many future feature additions such as reverts, displaying to the user the changes and so on.

Requirement

On edit or creation of a page the archive system must be fired off with necessary information for storage (username, file name, contents).

Archive system takes data and formats it into insert query to insert into the database.

User stories

1. As a user, I want to have my markdown files archived into a database so that I can have record of the current change set.
2. As a user, I want the wiki system to archive the changes I make to a markdown file on creation or edit of a page so that I don't have to keep track of when I make changes myself.
3. As a user, I want the person who submits a change to be tracked so that I have full record of who is editing documents in the wiki system.

Design

User story 1, 2, 3

Interface

- API name: ArchivePage.py
- Input: the parameter is the name of the wiki page, and its url path.
- Action: markdown file contents, and file name are inserted into database as a commit to be later referenced and used
- Output: Database now has given data so it must be able to be found in a select statement.

Detailed design

- To trigger Archive of the page, users simply hit the save button through the edit panel so that creation or edit of a page is stored in database. (User story 2)
- Reference initialization of an ArchivePage Object should be added to the `core.py` file in the `save` method to call the `store method`. (User story 2)
- The markdown file is then selected from the given url location (in our case the content directory as supplied in the save method). The name of the markdown file is used to select the file from the content folder to pull the contents of the file for storage as a string using basic python file I/O reading.
- Using the session portion of the flask library, we retrieve the current `user_id` to store along with the records for tracking purposes (User story 3)
- In the `store` method, `cursor`, `execute`, and `fetchall` methods will be used from the "psycopg2" library to insert into and select from the postgresql database. (User Story 1)
- We first select to see if the database has any record of the file we're trying to store: (User Story 1)
  - If it does we increment the max `commit_id` by one and store it with the same: `page_id`, `page_name`, and store our new: `date_update`, `page_file` (stores file contents), and `user_last_update` in accordance to our database structure.
  - If it doesn't we increment the max `page_id` by one and store the `commit_id` with the value 1 and store our new: `page_name`, `date_update`, `page_file` (stores file contents), and `user_last_update` in accordance to our database structure.

- Once the data is stored in the database it is ready to be used in record keeping of any manner. (Our next steps of the full feature is in the UI and retrieval features)

Interface

- API name: ArchiveDatabaseConnection.py
- Input: N/A
- Action: Makes a connection with our Amazon cloud based Postgresql database
- Output: Connection for reuse in other files.

Detailed design

- Using `psycopg2` we use the connect function to connect to our database
- We use the parameters of `dbname`, `user`, `host`, `port`, and `password` to connect to ours which is hard coded in. If different database is desired it will have to be edited.

Implementation

The dependencies used in this feature are:

- `import psycopg2` - to interact with our postgresql database.
- `from io import open` - to pull contents of mark down files for storage.
- `from flask import session` - to pull the user_id that is currently being stored in the sessions

User story 1 and 2 and 3

The Python code for implementation can be found at Team Wiki Wiki GitHub.

Tests

- This test uses `psycopg2` to interface with the database.
- The tests for the added features are included in the test feature folder. This test is can be found from Team Wiki Wiki GitHub
- In this test case, a test file is saved to the database using the `store` method of `ArchivePage.py` to store a test markdown file. Then the system checks queries the database to see if the contents match the page and if it is there. If successful the test will pass.

Checking the requirements

- User story1 - Test results (entries of test file in database) shows that the information is properly being stored.
- User story2 - UI related so it is extremely hard to test.
- User story3 - Test results (matching info about user, and other document information exists).

Limitations of tests

1. As mentioned above the tests fail to cover the UI portion of hitting the save button and it being triggered. We must test this manually
2. We kept to purely unit tests for this feature as it is mostly back end..

User documentation

Storage of markdown files is handled on save during an edit or creation of a page. So users don't have to worry about handling when a doucment is stored.