Eric Jackman

Team Member Information

* + [jackmane1@mymail.nku.edu](mailto:jackmane1@mymail.nku.edu)

Stage B

Feature 1: Page version history and recovery

Rationale

This feature is important for a few reasons. Firstly, in the Wiki’s current form, it is far too easy to lose data. For example, a user could submit an edit in the editor without realizing that they had replaced important information. Without the ability to restore old versions of the page, this data is not recoverable. Additionally, logging user edits has advantages. Say for instance that the site has an issue with misinformation or tampering with other’s work. A moderator could use this information to ban problematic users.

Fortunately, this feature can be implemented without any new dependencies. Built-in Python file functions and existing wiki classes are all that is needed.

This feature will be implemented as a tab on the editor page (next to Editor and Preview). It will contain a complete list of every page update, represented as a username with a timestamp. As a result, this feature requires changes to the editor.html template. Additionally, there will need to be a new form: *VersionHistoryForm* to submit a selection to restore the page to.

A screenshot of a computer screen

Description automatically generated

All versions of the page will be stored within a directory inside of content. The directory will have an identical name to the page. Each version in the directory will be named *v1.md*, *v2.md*, *v3.md*, etc. When a page *Test*is created, *Test.md* is created in content as well as a directory called *Test*. Then, *v1.md* is created inside *Test*. When we edit *Test*, we update *Test.md* and create *v2.md.*When we restore *Test*, the version that we select will be copied to *Test.md.*This functionality will be implemented with a new class in core.py called *PageVersionManager*. The client will use this class to update pages and restore pages using *update\_page(page) and restore\_page(page, version)*.

A screenshot of a computer

Description automatically generated

Requirements

* As a user, I want to be able to see a list of all edits to a page. This includes the username of the user who made the change as well as the time it happened.
* As a user, I want to be able to restore a page to a previous version from the list.

Deliverables

* Design:

A diagram of a website

Description automatically generated

* code: https://github.com/lschwartz15/440project/tree/eric/
* test: https://github.com/lschwartz15/440project/tree/eric/
* documents: https://github.com/lschwartz15/440project/

Test Plan

Unit tests for this feature will need to cover three scenarios. Firstly, when a user creates a new page, we will need to assert that the correct files are created, including the page and the directory containing v1. Secondly, when a page is edited, we need to assert that the page now contains the new data, and the directory contains the new version. Thirdly, when a user restores a page, we need to assert that the page contains the correct restored data.

*test\_page\_created()*

*test\_page\_updated()*

*test\_page\_restored()*

Milestones and Deadline

* Milestone 1
  + Date: **11/6/2023**
  + Goal: **Design feature and create UML diagram and skeleton**
* Milestone 2
  + Date: **11/13/2023**
  + Goal: **Implement version history log feature**
* Milestone 3
  + Date: **11/20/2023**
  + Goal: **Implement version restore feature**
* Milestone 4
  + Date: **11/27/2023**
  + Goal: **Create unit tests for features**
* Milestone 5
  + Date: **12/4/2023**
  + Goal: **Clean up documentation and prepare for presentation**

Risk Analysis

Expected conflicts between Stage C and the following:

1. CSC425 group project, week 8-16
2. CSC425 final exam, week 15-16
3. ASE230 group project, week 8-16

Stage C: Progress

Feature Implementation

Week 1 (11/13)

Summary

This week I spent my available time furthering my understanding of the codebase and making adjustments to my design.

Milestones or risks in this week

Unfortunately, it did not meet milestone 2, as my history log feature was not implemented. This was due to a time-consuming project in another class, ASE 230. However, I did make some important progress in my design that should speed up the implementation process for next week.

Design

A diagram with text and words

Description automatically generated with medium confidence

Document

After my exploration, I discovered that the PageVersionManager would not need to interact with the Page class like I had thought. Instead, it operates independently to create and manage backups of previous page versions. Additionally, I have outlined some additional attributes and methods that the class will need in the UML diagram above.

Week 2 (11/20)

Summary

This week I was able to catch up and am now back on track. I implemented both the history log and version restore features. Additionally, during the implementation process, I improved the design even more.

Milestones or risks in this week

Thankfully, I was able to tackle my work in other classes, so I was able to make this project a priority. Because of this advantage, I picked up last week's slack as well as meeting this week's milestone. As of now, milestones 2 and 3 are finished.

Design

A screenshot of a diagram

Description automatically generated

Code

In core.py:

class PageVersionManager(object):  
 def \_\_init\_\_(self, url, user):  
 self.url = url  
 self.user = user  
 self.page\_path = CONTENT\_DIR + '/' + url + '.md'  
 self.dir\_path = CONTENT\_DIR + '/\_' + url  
 self.edits\_path = CONTENT\_DIR + '/\_' + url + '/edits.json'  
  
 def get\_edits(self):  
 if not os.path.exists(self.edits\_path):  
 return {}  
 with open(self.edits\_path) as f:  
 data = json.loads(f.read())  
 return data  
  
 def create\_page(self):  
 os.mkdir(self.dir\_path)  
 timestamp = self.get\_timestamp()  
 edit = {timestamp: {'user': self.user}}  
 if not os.path.exists(self.edits\_path):  
 with open(self.edits\_path, "w") as f:  
 f.write(json.dumps(edit, indent=2))  
 shutil.copyfile(self.page\_path, self.dir\_path + '/' + timestamp.replace(':', ' ') + '.md')  
  
 def delete\_page(self):  
 if os.path.exists(self.edits\_path):  
 os.remove(self.edits\_path)  
 if os.path.exists(self.dir\_path):  
 shutil.rmtree(self.dir\_path)  
 return  
  
 def update\_page(self):  
 data = self.get\_edits()  
 timestamp = self.get\_timestamp()  
 data[timestamp] = {'user': self.user}  
 if os.path.exists(self.edits\_path):  
 with open(self.edits\_path, "w") as f:  
 f.write(json.dumps(data, indent=2))  
 shutil.copyfile(self.page\_path, self.dir\_path + '/' + timestamp.replace(':', ' ') + '.md')  
  
 def restore\_page(self, timestamp):  
 shutil.copyfile(self.dir\_path + '/' + timestamp.replace(':', ' ') + '.md', self.page\_path)  
  
 def get\_timestamp(self):  
 dt = datetime.datetime.now()  
 return dt.strftime('%b %d %Y %X')  
  
  
  
In routes.py:

@bp.route('/edit/<path:url>/', methods=['GET', 'POST'])  
@protect  
def edit(url):  
 page = current\_wiki.get(url)  
 pageVersionManager = PageVersionManager(url, current\_user.name)  
 form = EditorForm(obj=page)  
 if form.validate\_on\_submit():  
 if not page:  
 page = current\_wiki.get\_bare(url)  
 if os.path.exists(CONTENT\_DIR + '/' + url + '.md'):  
 form.populate\_obj(page)  
 page.save()  
 pageVersionManager.update\_page()  
 else:  
 form.populate\_obj(page)  
 page.save()  
 pageVersionManager.create\_page()  
 flash('"%s" was saved.' % page.title, 'success')  
 return redirect(url\_for('wiki.display', url=url))  
 return render\_template('editor.html', form=form, page=page)  
  
  
@bp.route('/edit/<path:url>/<path:version>/', methods=['GET', 'POST'])  
@protect  
def restore(url, version):  
 version = version.replace('\_', ' ', 3).replace('\_', ':')  
 pageVersionManager = PageVersionManager(url, current\_user.name)  
 pageVersionManager.restore\_page(version)  
 flash('Version from "%s" was restored.' % version, 'success')  
 return redirect(url\_for('wiki.display', url=url))

@bp.route('/history/<path:url>/', methods=['POST'])  
@protect  
def history(url):  
 pageVersionManager = PageVersionManager(url, current\_user.name)  
 edits = pageVersionManager.get\_edits()  
 edit\_list = '<ul>'  
 for edit in edits:  
 version = edit.replace(' ', '\_').replace(':', '\_')  
 edit\_list += '<li><a href="' + version + '" id="restorelink">' + edit + ' - ' + edits[edit]['user'] + '</a></li>'  
 edit\_list += '</ul>'  
 return edit\_list  
  
  
@bp.route('/delete/<path:url>/')  
@protect  
def delete(url):  
 page = current\_wiki.get\_or\_404(url)  
 pageVersionManager = PageVersionManager(url, current\_user.name)  
 pageVersionManager.delete\_page()  
 current\_wiki.delete(url)  
 flash('Page "%s" was deleted.' % page.title, 'success')  
 return redirect(url\_for('wiki.home'))

Document

Most of the design changes and implementation are shown from the UML diagram and from the code above. The PageVersionManager (PVM) is responsible for managing a directory of page backups and edit data. When a page is created, the PVM creates a directory in the content folder with the same name as the page, but preceded by an underscore. For example, a page called *test*would result in a directory called *\_test* to be created. Next, the PVM creates a json file called edits.json in said directory to store all edits to that page. Each object in the json file is identified by the timestamp created when the edit happens. It has one field, *user*, which stores the username of the user who made the edit. After three edits, the json file would look like this:

A screenshot of a computer code

Description automatically generated

Each time an edit happens, the PVM creates a new md file, named with the timestamp, and the contents from the new page are copied into the backup. This way, each json object is easily related to the backup file that it is referencing. (Note: the colons are replaced with spaces as colons are not a valid character in a filename)

A screenshot of a computer program

Description automatically generated

When the *history*tab is clicked on the edit page, the PVM reads the contents from the edits.json file and displays them as a list of links. Clicking one of these links will call the PVM to copy the contents from the corresponding backup file into the current file.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

When deleting a page, the PVM deletes the directory containing both the edits.json and all of the backup md files.

Finally, I made one small tweak to the index page. We don't want to see all of the backup pages, so I added a conditional to filter out all pages that have a URL starting with an underscore.

Week 3 (11/27)

Summary

This week I created 4 unit tests to test the PageVersionManager class, completing milestone 4.

Milestones or risks in this week

This week, Thanksgiving was a clear conflict with this course and others. Thankfully, I was able to complete the work and meet the milestone regardless.

Code

New file, pageVersionManagerTests:

import os  
import unittest  
import time  
  
from Riki import create\_app  
from wiki.core import PageVersionManager  
  
  
class TestPageVersionManager(unittest.TestCase):  
 def setUp(self):  
 self.app = create\_app(os.getcwd())  
 self.client = self.app.test\_client()  
 self.app.testing = True  
 self.app.config['WTF\_CSRF\_ENABLED'] = False  
 self.user = 'TEST'  
 self.url = 'TESTPAGE'  
 self.pvm = PageVersionManager(self.url, self.user)  
 self.client.post('/user/login', data={'name': self.user, 'password': 'TEST'}, follow\_redirects=True)  
  
 def tearDown(self):  
 self.client.get('/delete/' + self.url, follow\_redirects=True)  
  
 def get\_page\_data(self):  
 with open(self.pvm.page\_path, 'r', encoding='utf-8') as f:  
 content = f.read().splitlines()  
 title = content[0][7:]  
 tags = content[1][6:]  
 body = content[3]  
 return title, tags, body  
  
 def test\_create\_page(self):  
 title = 'TEST\_TITLE'  
 tags = 'TEST'  
 body = 'TEST\_BODY'  
 response = self.client.post('/edit/' + self.url,  
 data={'title': title, 'body': body, 'tags': tags},  
 follow\_redirects=True)  
 self.assertEqual(response.status\_code, 200)  
 self.assertTrue(os.path.exists(self.pvm.page\_path))  
 self.assertTrue(os.path.exists(self.pvm.dir\_path))  
 self.assertTrue(os.path.exists(self.pvm.edits\_path))  
 self.assertTrue(os.path.exists(self.pvm.dir\_path + '/' + self.pvm.get\_timestamp().replace(':', ' ') + '.md'))  
 edits = self.pvm.get\_edits()  
 self.assertEqual(len(edits), 1)  
 page\_title, page\_tags, page\_body = self.get\_page\_data()  
 self.assertEqual(page\_title, title)  
 self.assertEqual(page\_tags, tags)  
 self.assertEqual(page\_body, body)  
  
 def test\_update\_page(self):  
 title = 'TEST\_TITLE'  
 tags = 'TEST'  
 body = 'TEST\_BODY'  
 title2 = 'TEST\_TITLE2'  
 tags2 = 'TEST2'  
 body2 = 'TEST\_BODY2'  
 self.client.post('/edit/' + self.url,  
 data={'title': title, 'body': body, 'tags': tags},  
 follow\_redirects=True)  
 self.client.post('/user/login', data={'name': 'eric\_jackman', 'password': 'pass'}, follow\_redirects=True)  
 time.sleep(1)  
 response = self.client.post('/edit/' + self.url,  
 data={'title': title2, 'body': body2, 'tags': tags2},  
 follow\_redirects=True)  
 self.assertTrue(os.path.exists(self.pvm.dir\_path + '/' + self.pvm.get\_timestamp().replace(':', ' ') + '.md'))  
 self.assertEqual(response.status\_code, 200)  
 edits = self.pvm.get\_edits()  
 self.assertEqual(len(edits), 2)  
 page\_title, page\_tags, page\_body = self.get\_page\_data()  
 self.assertEqual(page\_title, title2)  
 self.assertEqual(page\_tags, tags2)  
 self.assertEqual(page\_body, body2)  
  
 def test\_restore\_page(self):  
 title = 'TEST\_TITLE'  
 tags = 'TEST'  
 body = 'TEST\_BODY'  
 title2 = 'TEST\_TITLE2'  
 tags2 = 'TEST2'  
 body2 = 'TEST\_BODY2'  
 self.client.post('/edit/' + self.url,  
 data={'title': title, 'body': body, 'tags': tags},  
 follow\_redirects=True)  
 self.client.post('/user/login', data={'name': 'eric\_jackman', 'password': 'pass'}, follow\_redirects=True)  
 time.sleep(1)  
 response = self.client.post('/edit/' + self.url,  
 data={'title': title2, 'body': body2, 'tags': tags2},  
 follow\_redirects=True)  
 page\_title, page\_tags, page\_body = self.get\_page\_data()  
 self.assertEqual(page\_title, title2)  
 self.assertEqual(page\_tags, tags2)  
 self.assertEqual(page\_body, body2)  
 self.client.get('/edit/' + self.url + '/0', follow\_redirects=True)  
 self.assertEqual(response.status\_code, 200)  
 page\_title, page\_tags, page\_body = self.get\_page\_data()  
 self.assertEqual(page\_title, title)  
 self.assertEqual(page\_tags, tags)  
 self.assertEqual(page\_body, body)  
  
 def test\_delete\_page(self):  
 title = 'TEST\_TITLE'  
 tags = 'TEST'  
 body = 'TEST\_BODY'  
 self.client.post('/edit/' + self.url,  
 data={'title': title, 'body': body, 'tags': tags},  
 follow\_redirects=True)  
 response = self.client.get('/delete/' + self.url, follow\_redirects=True)  
 self.assertEqual(response.status\_code, 200)  
 self.assertTrue(not os.path.exists(self.pvm.page\_path))  
 self.assertTrue(not os.path.exists(self.pvm.edits\_path))  
 self.assertTrue(not os.path.exists(self.pvm.dir\_path))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

In PageVersionManager:

def restore\_page\_by\_index(self, index):  
 try:  
 index = int(index)  
 except:  
 return False  
 edits = self.get\_edits()  
 i = 0  
 for timestamp in edits:  
 if i == index:  
 self.restore\_page(timestamp)  
 return True  
 i += 1

In routes:

@bp.route('/edit/<path:url>/<path:version>/', methods=['GET', 'POST'])  
@protect  
def restore(url, version):  
 pageVersionManager = PageVersionManager(url, current\_user.name)  
 if len(version) == 1:  
 pageVersionManager.restore\_page\_by\_index(version)  
 else:  
 version = version.replace('\_', ' ', 3).replace('\_', ':')  
 pageVersionManager.restore\_page(version)  
 flash('Version from "%s" was restored.' % version, 'success')  
 return redirect(url\_for('wiki.display', url=url))

Document

This week, only minor changes were made to my existing code. I updated the restore method to additionally accept an integer as an index to restore the page. As a result, PageVersionManager has a new helper method to account for receiving an integer. The purpose of making these changes is to allow for easier testing, without changing anything from the user's perspective. On top of this, I have written 4 unit tests to test PageVersionManager. (1) test\_create\_page asserts that the necessary files are created and that their contents are correct. (2) test\_update\_page asserts that the main page is correctly changed, the edit is correctly logged, and that the new backup file is created. (3) test\_restore\_page asserts that restoring a page correctly changes the main file's contents. (4) test\_delete\_page asserts that all relevant files are correctly cleaned up after the page has been deleted.