InfoPedia

Implementer: Cameron Springer

API name: Wiki Version History

# Problems and Solutions

Problems

* We cannot keep track of changes over time
* We cannot navigate between different versions of the wiki, nor compare versions
* We want no page versions left behind when a page is deleted

Solutions

* We keep track of the changes over time
* We make a simple and easy to use version history record
* We make sure all versions of a page are deleted when a user deletes it.

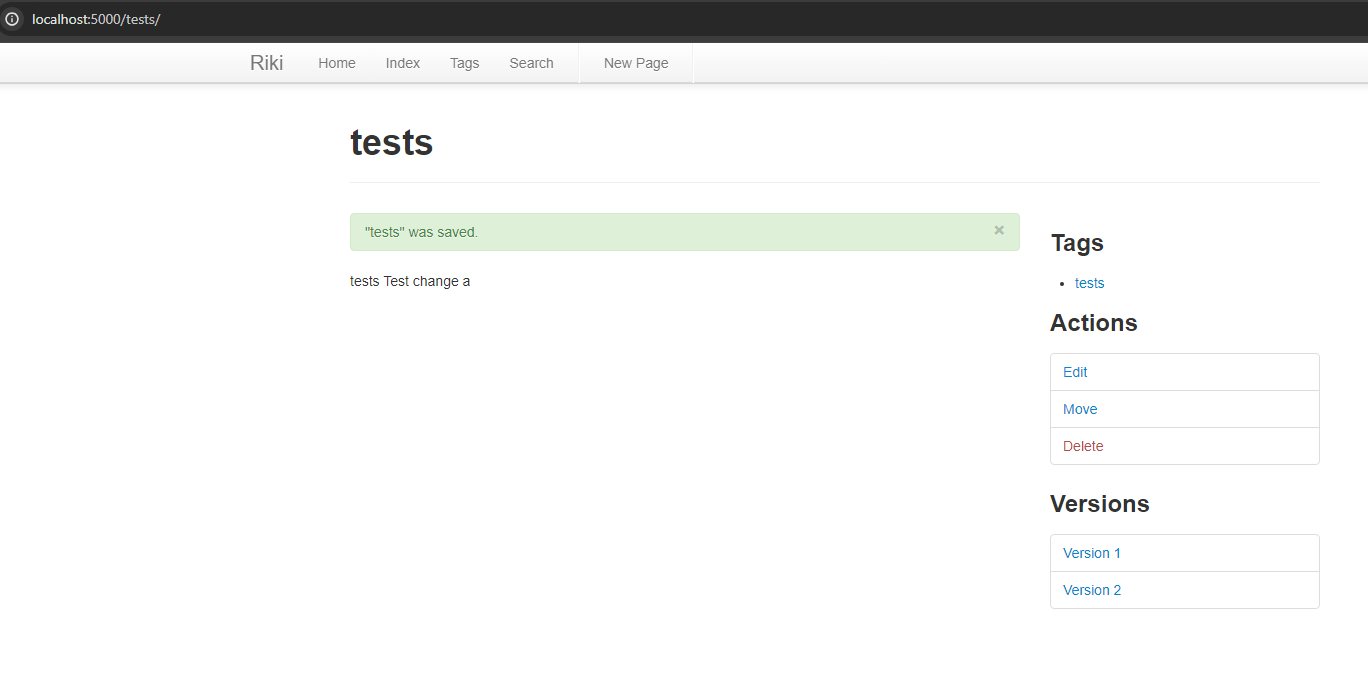
# Requirements (User Stories)

1. As a wiki author, I want to retain previous versions of a Wiki, so I can keep track of the changes made to it over time.
2. As a user, I want to be able to easily navigate between the versions of a Wiki, so I can compare versions and see what changes have been made.
3. As a wiki author, I want all versions of the wiki to be deleted, so that all traces of my wiki are removed when I choose to delete it.

# Usage Example

* <http://localhost:5000/display_version/testpage/1>
  + *Direction: Explain about this feature*
  + *Direction: Copy screen capture of the your feature output*

Whenever a page is created, edited, or deleted, one of our newly created methods is called. When a page is created, we have a new database entry that has version 0 attached to it. When a page is edited, we have a new database entry that has +1 of the previous version, and displays this to the user and allows them to change between them or compare them. Lastly, when a page is deleted, all database entries relating to that page are deleted, including all versions.



# User Manuals

*Direction: make a user manual – focus on easy usage of your feature*

One important thing to note at the start is when you download our Riki system, you will need to make sure the database is accessible. The way I did this in PyCharm was to click on View -> Tool Windows -> Database. On the database window that pops up, under wiki.db where it says “No schemas selected ...”, if you click on the “...” and check the box next to All Schemas, your database should now work. This may be an issue with just our groups systems, but we thought it would be important to note.

This feature is quite simple, whenever you create, edit, or delete a wiki page, a version is either created or deleted from the GUI under the “Versions” section. When you first create a page, there is nothing under the versions section because this is version 0. If you edit the page you will create multiple versions that you would then see under the versions section. Lastly if you delete a page, all versions of that page will be deleted.

# Implementation Goal

*Direction: make sure what is your goal. Students can copy wiki440 contents. Also, describe the changes in goal from instructor’s feedback (if you have it).*

The goal of this implementation is to save different versions of wiki pages so that the user can decide between versions/changes that were made.

# Implementation Details

*Direction:*

1. *Explain what modules in the Riki system you modified/added to implement what part of your feature.*
2. *make this document as a record of what you did for implementing this feature. Think what information should be recorded for your boss, team members, and future yourself.*

We added a small SQL-Lite database that keeps records of id’s, URL’s, versions, contents, and dates created. We basically store different versions in this database depending on if the user makes a new version or not and choose which version to display based on what the version the user chooses. We modified the wiki.db file and core.py where most of the methods for accessing the database to save or select versions.

# Test

*Direction:*

1. *Reuse the test plan you made in B stage.*
2. *Explain how you could test this feature. Referencing the file name and line number will help.*

A mock database connection was created, and the wiki pages table was initialized to simulate the creation of the app database. Since core modules were integrated using the SQLite database, the unit tests replicate the SQL queries used to simulate manipulation of the wiki pages table. The test file for this API is version\_hist\_tests.py in our tests directory.

# Measurement

*Direction:*

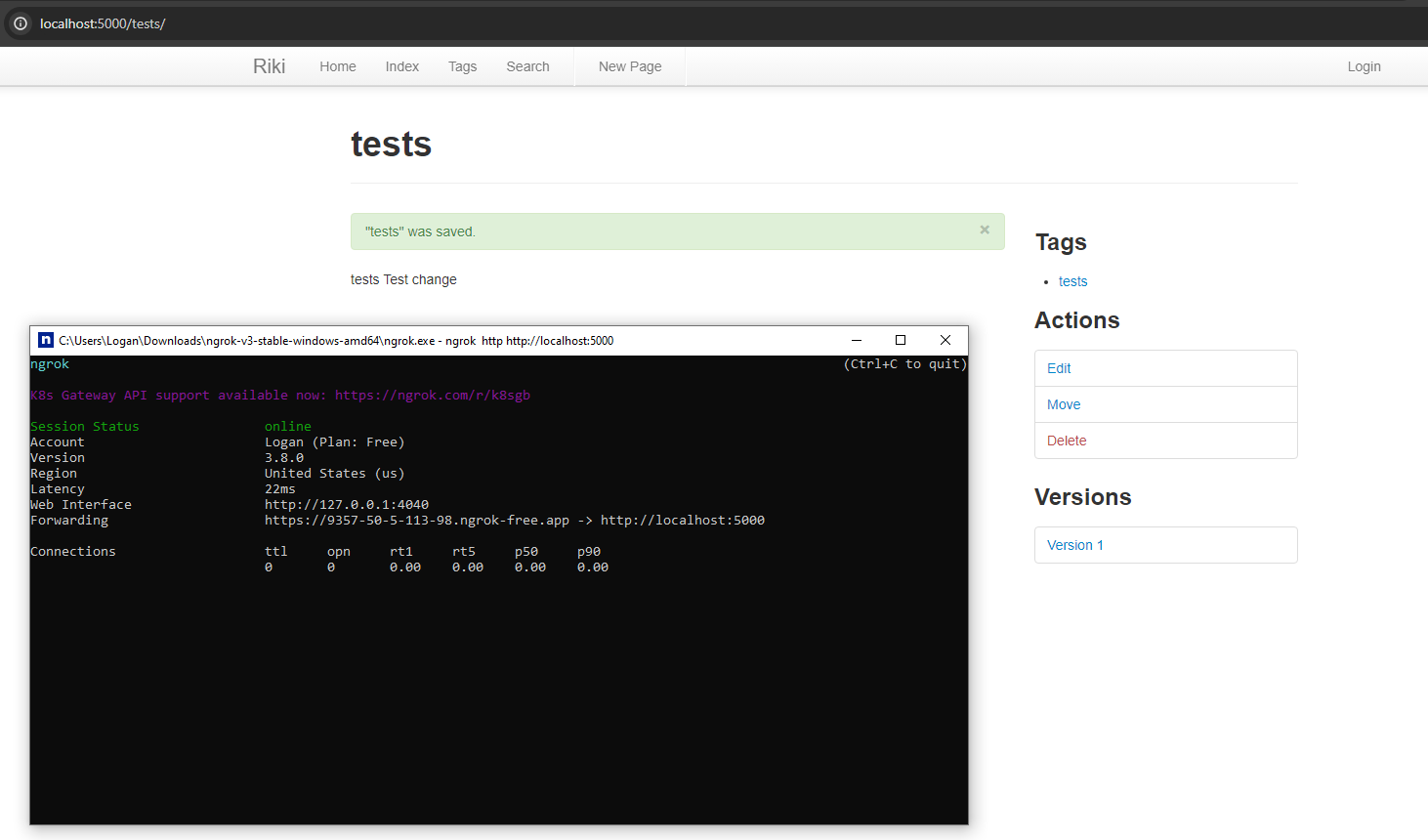
1. *Use tools such as cloc (*[*http://cloc.sourceforge.net*](http://cloc.sourceforge.net)*) to calculate the LoC you added (updated) for your feature.*

We added 181 lines of code for this API.

# Deployment

*Direction:*

1. *Use ngrok (*[*https://ngrok.com*](https://ngrok.com)*) to deploy your feature so that your feature can be accessed from clients. Read “project-tools-installation.pdf” for the usage of ngrok.*
2. *Copy a screen capture that shows ngrok is working.*



Implementer: Cameron Springer

API name: Author/Editor Permissions

# Problems and Solutions

Problems

* Anyone can edit pages on the wiki
* Anyone can delete any page on the wiki

Solutions

* We make the ability to approve/disapprove edits
* We allow the author to submit changes so that more people can contribute and the author can select which changes to submit
* We only allow authors to delete their own pages to maintain integrity and control.

# Requirements (User Stories)

1. As a wiki author, I want to have the ability to approve/disapprove edits made to a wiki, so that I can preserve the integrity of my wiki.

2. As a wiki user, I want to have the ability to submit changes to the author of a wiki for approval, so that I can contribute to any wiki in the system.

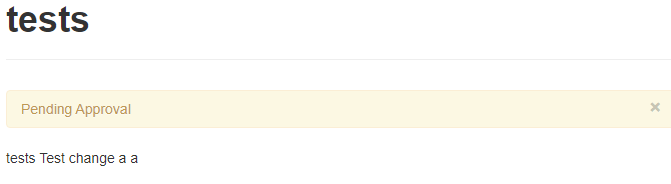
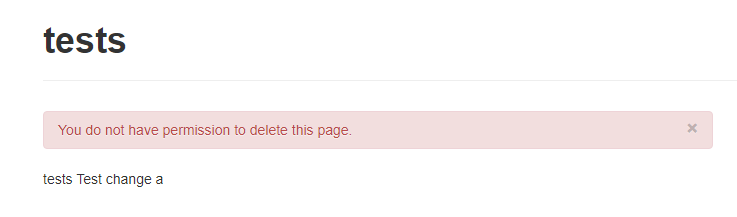
3. As a wiki author, I want to be the only user with delete privileges for my wikis, so that I have full control of my wikis.

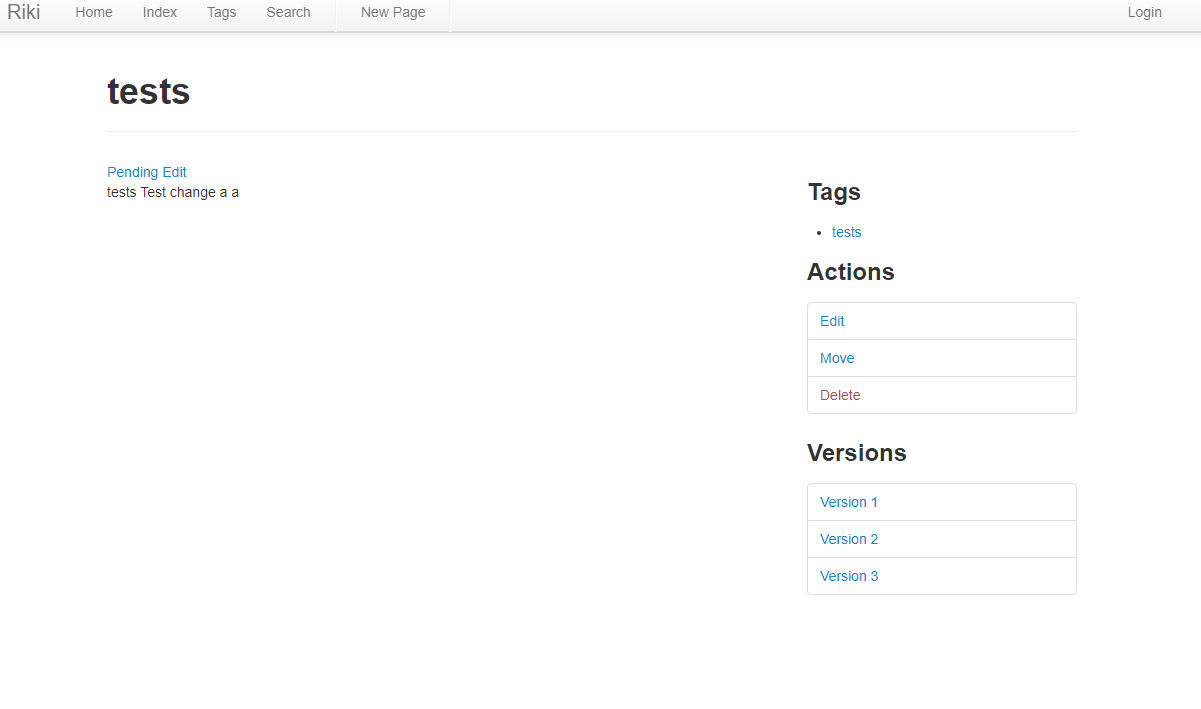
# Usage Example

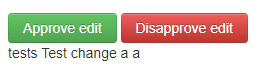
* Sign in as Username: sam and Password: 1234
* <http://localhost:5000/testpendingedit/>
  + *Direction: Explain about this feature*
  + *Direction: Copy screen capture of the your feature output*

This feature considers users that create a page to be the “author” of that page. If you are not the author of the page, then you cannot delete them at all. You also cannot make edits on them without the author’s permission.

As Sam:

As name, also author:





# User Manuals

*Direction: make a user manual – focus on easy usage of your feature*

This feature is also quite simple, when you create a wiki page, you are marked down as that wiki pages author. If anyone tries to edit or delete a wiki page that is not the author, it will request the authors approval or not allow it respectively.

# Implementation Goal

*Direction: make sure what is your goal. Students can copy wiki440 contents. Also, describe the changes in goal from instructor’s feedback (if you have it).*

The goal of this implementation is to secure the data in a sense, we want the author of the page to feel some sort of control over their page and make it secured to them. With edit approvals and deletion denials, we easily accomplish this.

# Implementation Details

*Direction:*

1. *Explain what modules in the Riki system you modified/added to implement what part of your feature.*
2. *make this document as a record of what you did for implementing this feature. Think what information should be recorded for your boss, team members, and future yourself.*

We added the ability for an author to maintain integrity over their wiki page. We made it so that when we add an entry to the database for wikipages, we now have some new tags with the database, one being author. Whenever we are trying to edit or delete a page now, it will check and see if we are the author of the page before allowing the edit or delete. If it is an edit, and we are not the author, it will put a edit request approval on the authors wiki page. The author can then compare the edits and decide to approve or disapprove the edit. If we are not the author and we try to delete it, it will tell us that we do not have permission to do that.

# Test

*Direction:*

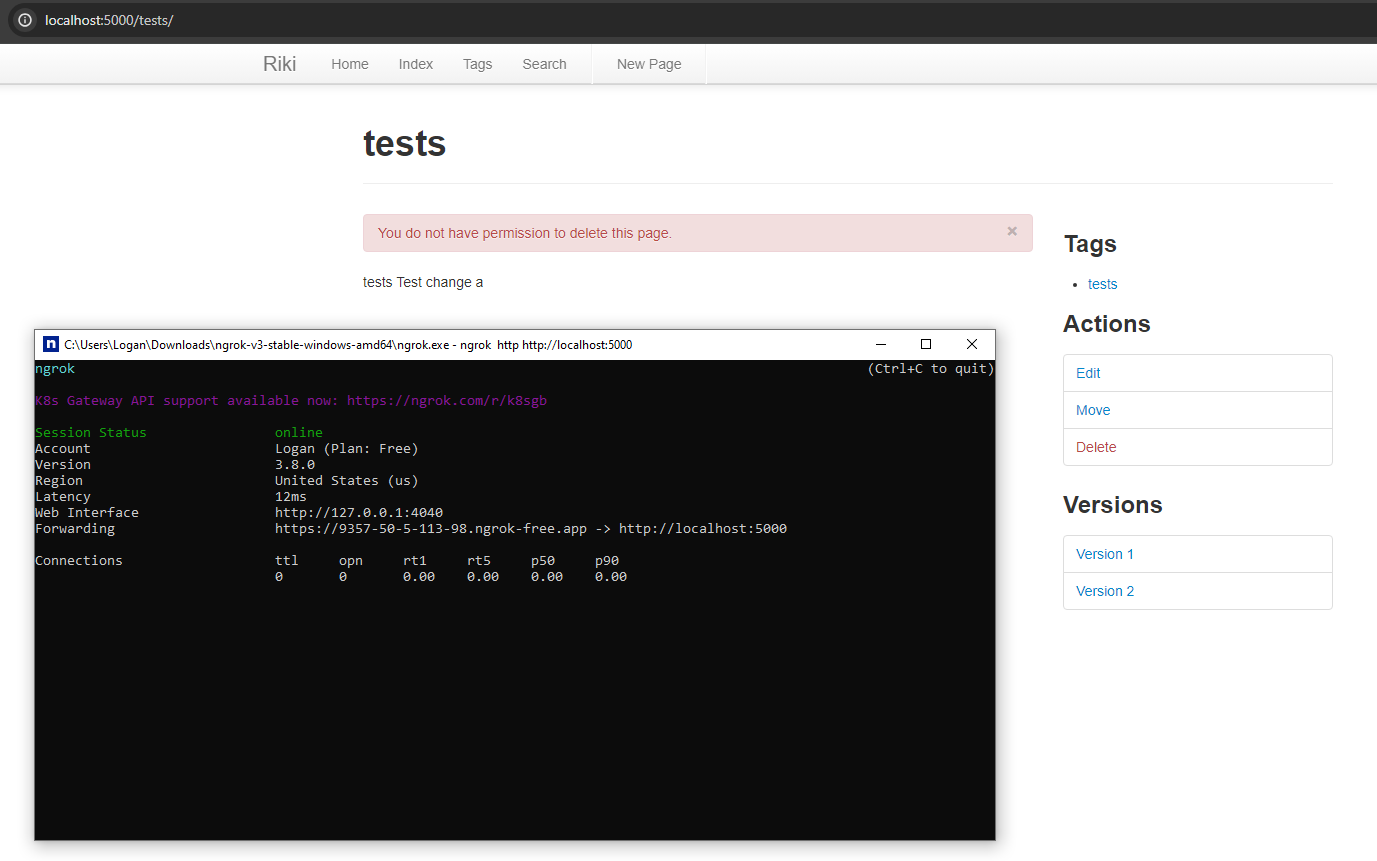
1. *Reuse the test plan you made in B stage.*
2. *Explain how you could test this feature. Referencing the file name and line number will help.*

A mock database connection was created, and the wiki pages table was initialized to simulate the creation of the app database. Since core modules were integrated using the SQLite database, the unit tests replicate the SQL queries used to simulate manipulation of the wiki pages table. The test file for this API is permissions\_tests.py in our tests directory.

# Measurement

We added 364 lines of code for this API.

# Deployment



Implementer: Gavin Jett

API name: Search Bar Suggestions

# Problems and Solutions

Problems

* We cannot find what we are looking for very easily
* We do not know what all the wiki has to offer

Solutions

* Add suggestive solutions to the search bar
* Allow the user to navigate using suggestive solutions

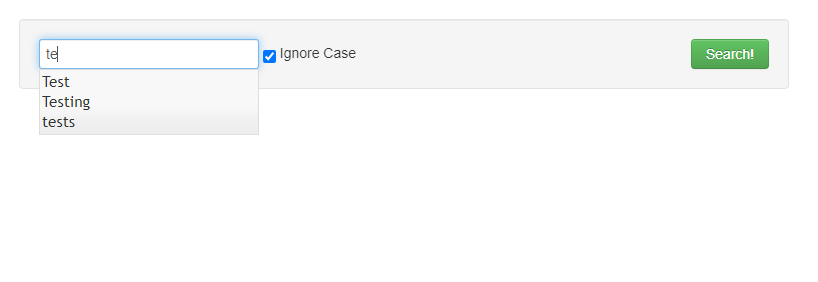
# Requirements (User Stories)

1. As a user, I want to see suggestions based on what I have typed in for pages to visit so that I can easily find what I'm looking for.
2. As a user, I want to be able to click a suggestion item, so that I can be redirected to the page found.

# Usage Example

* <http://localhost:5000/search/>
* <http://localhost:5000/search_autocomplete>
  + *Direction: Explain about this feature*
  + *Direction: Copy screen capture of the your feature output*

This feature allows for users to find wikipages easily based on which wiki pages are already available. It also allows for users to click on the suggestions to navigate to the suggested page.



# User Manuals

*Direction: make a user manual – focus on easy usage of your feature*

This feature is also very simple, it allows the user to find any of the available wiki pages. Before, on the search page, there was no way to tell what all pages were in the database unless you knew what you were searching for and searched it. Now we have suggestions that only suggest available wiki pages, so we can know what pages are available and search quicker.

# Implementation Goal

*Direction: make sure what is your goal. Students can copy wiki440 contents. Also, describe the changes in goal from instructor’s feedback (if you have it).*

The goal of this implementation is to make searches easier, faster, and more knowledgeable of what is within the site.

# Implementation Details

*Direction:*

1. *Explain what modules in the Riki system you modified/added to implement what part of your feature.*
2. *make this document as a record of what you did for implementing this feature. Think what information should be recorded for your boss, team members, and future yourself.*

We added a dropdown bar to the search bar that provides available wiki pages as suggestions. We made it so that for each wiki page available, you can navigate using the search bar to these pages much easier. We added the search directory to the wiki which includes Dropdown.py, DropdownItem.py, and DropdownSearch.py, as well as in the templates folder we added a suggestions directory with the search.js file.

# Test

*Direction:*

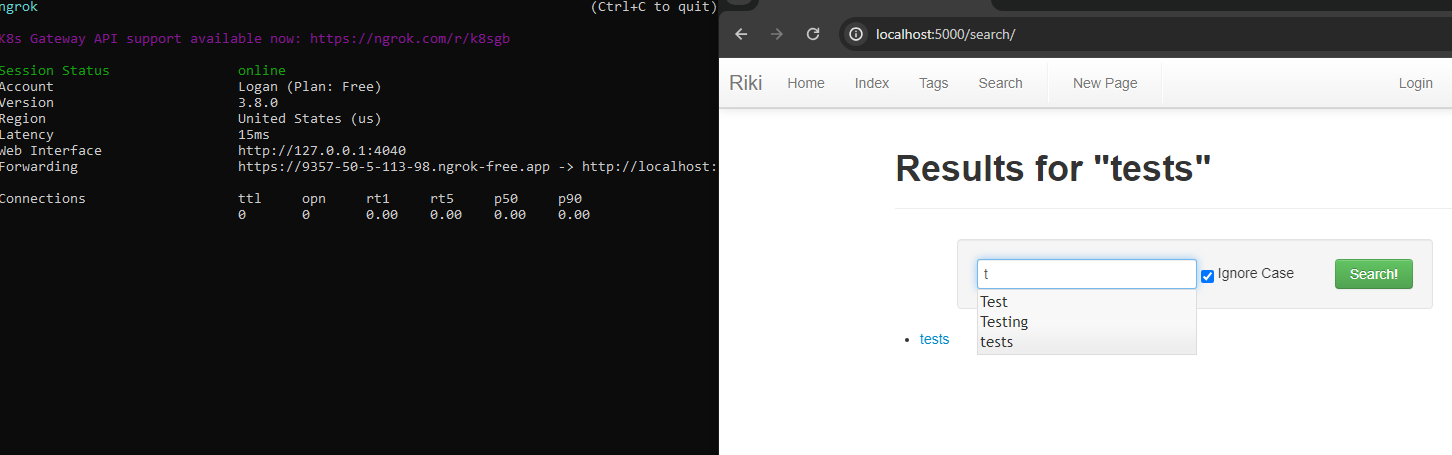
1. *Reuse the test plan you made in B stage.*
2. *Explain how you could test this feature. Referencing the file name and line number will help.*

We created a mock index of all the wiki pages and passed it into the dropdown object for it to search through using SuggestionSearch. We then tested if it found the correct items when queried and if the objects were formatted correctly. The test file for this API is test\_search\_suggestions.py in our tests directory.

# Measurement

We added 191 lines of code for this API.

# Deployment



Implementer: Gavin Jett

API name: Search Bar History

# Problems and Solutions

Problems

* We have found what we wanted before, but we are struggling now
* We want to be able to navigate to sites we have been to before

Solutions

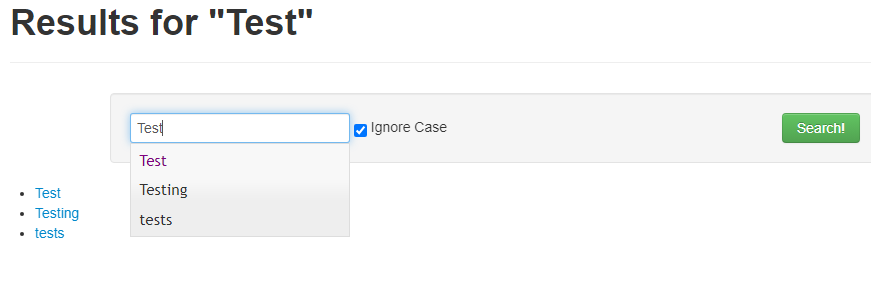
* We will implement an edit to the previous dropdown that will allow us to see previously searched pages
* We will allow the user to navigate to these pages by clicking on them in the dropdown

# Requirements (User Stories)

1. As a user, I want to see previously searched pages in my dropdown, so that I can easily return to previously searched pages.
2. As a user, I want to be able to click a suggested history item, so that I can be redirected to the page I previously have been to.

# Usage Example

* <http://localhost:5000/search/>
* <http://localhost:5000/search_autocomplete>
  + *Direction: Explain about this feature*
  + *Direction: Copy screen capture of the your feature output*



This feature allows for users to find wikipages easily based on which wiki pages they have already been to. The text shows as purple when the user has already visited that wiki page. It also allows for users to click on the suggestions to navigate to the suggested page.

# User Manuals

*Direction: make a user manual – focus on easy usage of your feature*

In Progress

# Implementation Goal

*Direction: make sure what is your goal. Students can copy wiki440 contents. Also, describe the changes in goal from instructor’s feedback (if you have it).*

The goal of this implementation is to provide quality of life. Users may want to revisit a page they have already been to, or something they have looked up before may be relevant to what they want. Humans are creatures of habit after all.

# Implementation Details

*Direction:*

1. *Explain what modules in the Riki system you modified/added to implement what part of your feature.*
2. *make this document as a record of what you did for implementing this feature. Think what information should be recorded for your boss, team members, and future yourself.*

We added a small database table that stores information based on what pages users visit. We added some code in core.py for connecting our new database table, as well as in routes.py, and we added all of our methods that find what pages we have already been to in search directory “Dropdown” files. These methods simply find what pages we have already been to before implementing the previously implemented search suggestions API.

# Test

*Direction:*

1. *Reuse the test plan you made in B stage.*
2. *Explain how you could test this feature. Referencing the file name and line number will help.*

We created another mock index of the pages for this test, but a mock database as well, and passed it to the dropdown object for the HistorySearch to use. After initializing the database, we searched through it and found out whether the correct items were output from our sample input query. The test file for this API is test\_search\_suggestions\_history.py in our test's directory.

# Measurement

We added 315 lines of code for this API.

# Deployment

