Project Milestone 5

Team - 200-2
Project - Webpage Analyzer
Team Members -

- Andrew Pickner
- AJ Jones
- Jake Henson
- Fengyuan Zhang
- Yuxi Liu

User Acceptance Testing (UAT) -

Data Used: In all cases, some baseline dummy data will be used to determine whether our pages are populating successfully. We will control and know exactly what is on each database, and so we can test to make sure all information is flowing properly from page to page, and from database to page, as well as that our main functionality of our app is working using values that we know (e.g., we know what data looks like from Apple.com because we can run it separately, and we will compare that data with the one populated from the URL)

Testing Procedures: Once we have successfully hosted everything and the project is finished, we will test it across multiple browsers (Firefox, Chrome, Edge) to ensure that it works.

3 Features + Test Cases:

Feature 1 - Ability for user to submit webpage, display GUI breakdown of metadata

Overview: The user should be able to use our Submit.html, type in a URL, and hit Submit. They will then be taken to the loading page, followed by the info.html page which will display the breakdown of their data.

Implementation: The user will type in a single URL in the box, and click “Submit” button. This will start the sequence of events that will display the data. The user will be passing in a URL to multiple API calls (one to append it to the database, one to actually run the python web crawler script)

Success Condition: The user sees the metadata and text analysis from the page they requested. They can toggle in between which type of analysis (metadata, text) but it will be on one dynamic info.html page.

Fail Conditions: The data does not display on the loading page (either blank)

Edge Cases:

- The loading page hangs indefinitely
    - ***SOLUTION*** - Build in a timeout function that will kick the user back to the login page with a message that their request timed out.
- The URL the user entered is invalid
    - ***SOLUTION*** - There is already a built-in Regex check to determine whether the user has entered an invalid URL when they hit submit. Before the actual API call is made, the regex will make sure the URL is actually valid.
- The URL entered is syntactically valid, but there's an issue with the page (Bad Gateway, Page not found)
    - ***SOLUTION -*** The Python script will return certain errors to determine what happened, and the API will let the page know what happened, which will in turn kick the user back to the login page that their request was bad.

Feature 2 - Ability to parse multiple requests

Overview: The user should be able to use our Submit.html, type in a URL, and hit Submit. They will then be taken to the loading page, followed by the info.html page which will display the breakdown of their data.

Implementation: The user will type in a single URL in the box, and click the "+" button to enter another. Another URL box will pop up to enter. The user is allowed to enter up to 3 URL's by hitting this plus, and they can remove the number of URL's by hitting the "-" button. Hitting "Submit" will again start the same sequence of events listed above that will display the data. The user will be passing in alls URL to multiple API calls (one to append it to the database, one to actually run the python web crawler script with all valid URLs)

Success Condition: The user sees the metadata and text analysis from the ALL of the pages they requested, all on the same page. They can toggle in between which URL's they would like to see using all the buttons on the top of the page,, as well as which type of analysis (metadata, text) but it will be on one dynamic info.html page.

Fail Conditions: The data does not display on the loading page (either blank) OR only data for one page will display

Edge Cases:

- The user tries to add more than three webpages:
    - ***SOLUTION*** - Javascript on the button will disable it and prevent the user from adding more
- The user tries to subtract/remove the only submission box:
    - ***SOLUTION*** - Javascript on the button will disable the - when there's only one URL box, but will enable when there are more than 1.
- The loading page hangs indefinitely
    - ***SOLUTION*** - See above
- Any/all URL's the user entered is invalid

    - ***SOLUTION*** - See above (URL regex works for any/all URL's entered)
- One of the URL submissions is blank
    - ***SOLUTION*** - Same as above
- Any/all URL's entered is syntactically valid, but there's an issue with the page (Bad Gateway, Page not found)
    - ***SOLUTION -*** See above

Feature 3 - Login, and View My Account

Overview: The user should be able to login to their existing account, and then click "My Account" to view the pages they've previously queried.

Implementation: Assuming the user has not logged-in already, they will click "Login" to sign into their account, using their username and password. This will use an API call to transfer the user to the Login page. Upon entering their information on the login page, they will hit "Submit", which will make an API call to the database and log them in. They will then be moved back to the start page. They should see their own username at the top-right corner of the page. Upon clicking "My Account", they will be transferred to the MyAccount page where information from another API call made to the "Users" database will populate this page with their username, and all of the previous URL's they've queried.

Success Conditions: The user can log in on the Login page. The user sees all of the previously URL's they've queried after clicking "My Account".

Fail Conditions: The user cannot login, and/or their data is not displayed

Edge Cases:

- The user doesn't have the correct password:
    - ***SOLUTION*** - The user will be told that their password is invalid. This is done via an API call to determine whether the username + password combo matches in the database!
- The user tries to login when they don't have an account
    - ***SOLUTION*** - The user will be informed that they don't have an account (done by using an API call to determine if their username exists in the database) and will be kicked back to the start page. They will be told to create an account before they can sign in!
- The user tries to click "My Account" before signing in:
    - ***SOLUTION*** - The button is disabled until the user signs in. This is verified via Javascript. If there's a valid username, then the button will become enabled.
- The user tries to see what URL's they've queried on the my account page, but they haven't queried any yet:
    - ***SOLUTION*** - Instead of a table with all of their URL's, the user will be told that they haven't queried any pages yet, and that they should go query some! (This will be done in the javascript of the page if their result from the username call for the API returns null or 0).