Team: McBain Fire Alarm

Team Members: Arsalaan Ansari (aaa2325) Trevor Jones (tcj2110)

Raphael Owino (ro2295) Kevin Mejia (km3204)

Bolded conditions were satisfied in this iteration. Green conditions were satisfied in the second iteration.

User Stories and an update on their completion:

1.<Front end>: As a <software developer>, I want <to query all of my open issues on a GitHub repo> so that <I may view them in the Sublime Text editor>

My conditions of satisfaction are <

1. The plugin can login to a GitHub repo,
2. T**he plugin can query GitHub Issues,**
3. **The plugin can query GitHub Issues assigned to me,**
4. **The plugin can query GitHub Issues which are open,**
5. The plugin can query the Github repo pull requests and their status,
6. The plugin can query the Github repo pull requests commits and comments.

>

USE CASES:

- Developer has an internet connection. They open up Sublime Text 3, instead of github.com on the web browser. If the user is previously logged in, they should expect to stay logged in. Some visual confirmation should be visually available to confirm the github username and ‘logged on’ status. If the user hovers their mouse over the codebase that has comments on a pull request tied to them, they see some visual indicator that they have something to check out. Upon clicking that indicator, they decide on next steps: these include

- add a summary of the proposed changes,
- review the changes made by commits,
- add labels
- add assignees
- @mention individual contributors or teams

2.<Front end>: As a <software developer>, I want <to be able to open and close GitHub Issues, Pull review commit comments in my Sublime Text editor> so that <I don't have to open GitHub.com in a browser to close or view the issue there>

My conditions of satisfaction are <

1. **The plugin can query open GitHub Issues** and **pull review comments**,
2. The plugin can close an open GitHub Issue,
3. **The plugin can submit responses to comments on open pull requests**,
4. **The plugin can read GitHub Issues in the Sublime Text Editor,**
5. The plugin can close a GitHub Issue in the Sublime Text Editor

>

USE CASES

- Developer has an internet connection. They open up Sublime Text 3, instead of github.com on the web browser. If the user is previously logged in, they should expect to stay logged in. Some visual confirmation should be visually available to confirm the github username and 'logged on' status. When the user accesses some git-backed codebase, they have the option to
    - Click sections of a commented region to view pull review comments
    - Read a recent github issue

- Close, or dismiss the issue without changing the code or closing the Sublime Text 3 window

3. <Front end>: As a <software developer>, I want <to be able to add comments, open new GitHub Issues in my Sublime Text editor> so that <I don't have to open GitHub.com in a browser to add new issues, comments there>

My conditions of satisfaction are <

1. **The plugin can send data to a GitHub repo,**
2. **The plugin can submit comments to open pull requests,**
3. **The plugin can add new GitHub Issues to a GitHub repo**

>

USE CASES:

- Developer has an internet connection. They open up Sublime Text 3, instead of github.com on the web browser. If the user is previously logged in, they should expect to stay logged in. Some visual confirmation should be visually available to confirm the github username and 'logged on' status. The expectation is the codebase being accessed is shared among two or more github accounts. When the user accesses some git-backed codebase, they have the option to
    - Highlight a code region
    - Right click to open a tab of options
    - From those options, click "submit comments"
        - From those comments, there is a submit pull request button as a final confirmation

4.<Front end>: As a <software developer>, I want <to be able to save my user credentials> so that < the plugin can automatically login to my GitHub.com account>

My conditions of satisfaction are <

1. The plugin can query a database (SQL or NoSQL tbd),
2. The plugin can insert data into a database,
3. The plugin can prompt the user for credentials,
4. **The plugin can verify user credentials**
5. The database stores the data persistently (in-memory, commitlog, etc.)
6. **The plugin can make an oath authentication request to GitHub**

>

USE CASES:

- User opens Sublime window:
  - User has used the plugin before, but logged out during the last usage before closing Sublime Text 3. A familiar tab should open, and in that the user provides either (user +password) or (an access token).
  - User is a recurring user of the plugin. A tab opens saying that the current Github username is displayed.
    - If the user is the user shown, all that us needed is the password
    - If the user is different than the one shown. They click a ‘log out’ button and then provides either (user +password) or (an access token) to access our plugin as the correct user
  - User has never used the plugin before, and this has not logged on to github via Sublime Text 3. A tab opens, and in that the user provides either (user +password) or (an access token).

Equivalence classes:

-First time user, the user has to login using credentials. There seems to be no way around it

-The user “logs out” and a new user steps in. Sublime plugin can first check if this new user, after providing credentials, has access to the same files that the previous user has used.

-Recurring user. Sublime plugin can probably verify that its the user with IP address, and less credentials than the initial log-in

Testing the user story:

Prefix Values: - Open sublime text, begin the request to use the plugin to connect to the github account. This is the values needed to set up the initial connection to github.com

Postfix Values: -these are what(if any) other information is needed to be provided if the Sublime plugin has enough to help verify that it’s the same user

Verification Values: Login can always be verified with email and password, these correspond to exactly one github account

Exit Values: Some boolean value thay confirms that a user is logged in should be returned.

Expected Results: Same as exit value

<Front end>: As a <software developer>, I want <to be able to change the layout of how GitHub Issues are presented to me in the Sublime Text editor > so that < I can interact with my GitHub Issues according to my preferences.>

My conditions of satisfaction are <

The plugin can query data pertaining to the layout of text in Sublime Text Editor,

The plugin can modify data pertaining to the layout of text in Sublime Text Editor

>

USE CASE

- User logs in according to the use cases described above.
    - If no preferences were previously provided, data is laid out using a default setting{font, font size, window location and size, font color etc}
    - Custom exist , so it's a straightforward way to present the github comments in the manner they want.

Equivalence classes:

-No initial layout preferences on Sublime Text, and data collection of some sort sees that nothing is saved for this user. A set of default settings have to be used

- Settings exist on Sublime, so it's a straightforward way to present the github comments in the manner they want.

-No preferences are made for the user, but for previous users there exists preferences on record(stored on something like MongoDB) that are not the default settings. For this case, just prompt the user if they want the previous settings. After they see some sort of preview, they decide on the spot, and if so, it becomes recorded as their preferences.

<Front end>: As a <software developer>, I want <to be able to save the layout preferences of how GitHub Issues are presented to me in the Sublime Text editor> so that <I do not have to set up the configuration on each startup>

My conditions of satisfaction are <

The plugin can query a database,

The plugin can insert data into the database,

The database stores the data persistently

>

NB: GitHub Issues may include but not limited to code review comments to pull requests and subsequent commits to that particular pull request.

USE CASE:

- User logs in according to the use cases described above.
  - If no preferences were previously provided, data is laid out using a default setting{font, font size, window location and size, font color etc}
  - Custom settings exist , they are retrievable from an existing database., so it's a straightforward way to present the github comments in the manner they want.

Testing plan

1.1 The plugin can login to a GitHub repo

Equivalence classes

1. The user is not connected to the internet
2. The user enters incorrect credentials
3. The user enters correct credentials
4. The user's credentials are not recognized.

1.2 The plugin can query the Github repo pull requests and their status,

Equivalence classes

1. Github repo has only active pull requests
2. Github repo has only closed pull requests
3. Github repo has no pull requests
4. Github repo has both active and closed pull requests

1.3 The plugin can query the Github repo pull requests commits and comments.

Equivalence classes

1. Github repo pull request has no comments
2. Github repo pull request has comments

2.1 **The plugin can query open GitHub Issues**

Equivalence classes

1. Github repo pull request had no open issues
2. Github repo pull request has open issues

2.4 **The plugin can read GitHub Issues in the Sublime Text Editor,**

Equivalence classes

1. There are no github issues for the opened file
2. There are github issues for the opened file

3.3 **The plugin can add new GitHub Issues to a GitHub repo**

Equivalence classes

1. The user can submit github issues to a repo
2. The user can submit github issues and comments to a pull request

4.3

Details on our CI: https://travis-ci.com/Raphaeljunior/resolve-comments/builds/92942006

We used Travis CI as our continuous integration system in a .travis,yml file which ran a suite of tests on our package named “git_comments”. Our travis system targeted Unix as its operating system because that was the platform our developers programmed on. For the front end travis downloads the Sublime Text Plugin Unit Testing platform to allow travis to run a headless version of Sublime Text that can run the tests written for it. For the backend we used nosetests for unit tests, and pylint for static analysis, coveralls for coverage, and pip env for virtualization of the python environment.

Unit Testing and end to end testing report

```
$ sh travis.sh run_tests
Running command: run_tests
Schedule:
  package: git_comments
  syntax_test: None
  syntax_compatibility: None
  color_scheme_test: None
  coverage: None
  output: /home/travis/.config/sublime-text-3/Packages/User/UnitTesting/git_comments/result
Xlib:  extension "RANDR" missing on display ":99.0".
Wait for tests output......
Start to read output...
test_token_empty (test_credentials.TestTokenEmpty) ... ok
test_user_correct_token_correct (test_credentials.TestUserCorrectTokenCorrect) ... ok
test_user_correct_token_incorrect (test_credentials.TestUserCorrectTokenIncorrect) ... ok
test_user_empty (test_credentials.TestUserEmpty) ... ok
test_user_incorrect_token_correct (test_credentials.TestUserIncorrectTokenCorrect) ... ok
test_quick_panel_st3 (test_quick_panel.TestQuickPanel) ... ok
----------------------------------------------------------------------
Ran 6 tests in 0.608s
OK
UnitTesting: Done.
The command "sh travis.sh run_tests" exited with 0.
0.00s$ cd utils
The command "cd utils" exited with 0.
0.44s$ nosetests
....
----------------------------------------------------------------------
Ran 4 tests in 0.003s
OK
The command "nosetests" exited with 0.
```

Static analysis report

```
Flake8..........................................................................P
assed
```

```
Trim Trailing
Whitespace.................................................Passed
Fix requirements.txt.................................(no files to
check)Skipped
Pretty format JSON...................................(no files to
check)Skipped
Don't commit to
branch...................................................Passed
Detect Private
Key......................................................Passed
Check JSON...........................................(no files to
check)Skipped
```