COMW 4156 Iteration Two

Group WLTT
Junyue Wu (jw3674) Bowen Tan (bt2484)
Xuheng Li (xl2784) Shulan Tang (st3174)

November 28, 2018

1 User Story Reflection

In our revised proposal, there are six user stories that we proposed. We achieved five of them except for No. 4: editing and preview scrolling synchronization.

1.1 Use Cases

1.1.1 Markdown To HTML

User Story: As a note taker, I want to add common markdown formatting to my notes so that I can organize it to be more readable.

Use Case Name Write note in Markdown.

Actor Note Taker

Description Note Taker types markdown text into the note editing field.

Base Flow

1. The note taker clicks the note editing field.
2. A cursor exists in the filed.
3. The note taker starts to typed in notes.
4. The note editing field displays note taker's input in the note editing field.
5. The application parses the plain markdown text and generate the corresponding HTML code.

1.1.2 Preview Existed Note

User Story: As a note taker, I want to see my pdf preview and script side by side so that I can preview the note directly.

Use Case Name Preview Existed Note

Actor Note Taker

Description Note taker view the formatted note in the preview field.

Precondition The application is opened and the notes are loaded from database

Base Flow

1. Note taker click on the tile of the note that appears in the sidebar.
2. The note title and contents appear in the editing field.
3. The parser parsed the markdown to be displayed in the preview field.
4. The parsed html from the markdown is displaced in the preview field.

Postcondition When user clicks another note or creates a new note, the editing and preview fields update correspondingly.

1.1.3 Preview Edited Note in Real Time

User Story: As a note taker, I want to preview the note in real-time so that I can better understand whether the note is correctly taken.

Use Case Name View Updated Note in Real Time

Actor Note Taker

Description Note taker edits the note and view the updated note in the preview field.

Precondition One note is created and selected by the note taker.

Base Flow

1. The note taker click the editing field.
2. A cursor appears at the position the note taker clicks on.
3. The note taker types in some new contents or delete some existing contents.
4. The html parser detects the change and updated the corresponding html.
5. The preview field displays the updated html content.

1.1.4 Save Notes

User Story: As a note taker, I want to save my notes in the application so that I can read it.

Use Case Name Save note in the database

Actor Note Taker

Description Note taker saves the edited note in the database

Base Flow

1. The note taker click the title of a note in the note list on the left.
2. The note displayed in the editing field.
3. The note taker edits the note in the editing field.
4. The note taker clicks the Save button on the left corner of the editing field.
5. The application opens a connection to database
6. The application saves note in the editing field to the database

1.1.5 Delete Notes

User Story: As a note taker, I want to delete the note in my application so that I will not see it in the future.

Use Case Name Delete note in the application

Actor Note Taker

Description Note taker deletes an existed note

Base Flow

1. The note taker click the title of a note in the note list on the left.
2. The note displayed in the editing field.
3. The note taker clicks the Delete button on the right corner of the editing field.
4. The note title record is removed immediately from the sidebar.
5. The application gets the id of the note and send the delete command to database.
6. The note record is removed from the database

2 Testing Plan Explanation

Our test suite is located in our repo spec/*.js

2.1 Test 1: CRUD of Notes

2.1.1 Equivalence Partition 1

There is some text with valid visible UTF-8 characters in the editor, test whether the test can be save / update to and retrieve correctly from the database.
Boundary Conditions

- The note content editor can be empty
- The title can be empty
- The note content editor has at most 10,000 characters
- The title has at most 100 characters
- The max number of the notes is 1000

Test Cases Set 1 This set of test cases is corresponding to length of the note content editor. After input the following contents, test whether the input can be saved and retrieved correctly.

- The note content editor is empty
- The note content editor has 1 lower character(i.e character 'a')

- The note content editor has 1 upper character(i.e character 'A')
- The note content editor has 1 symbol(i.e comma)
- The note content editor has 9,999 characters
- The note content editor has 10,000 characters
- The note content editor has 10,001 characters

Test Cases Set 2 This set of test cases is corresponding to length of the title editor. After input the following contents, test whether the input can be saved and retrieved correctly.

- The title editor is empty
- The title editor has 1 lower character(i.e character 'a')
- The title editor has 1 upper character(i.e character 'A')
- The title editor has 1 symbol(i.e comma)
- The note content editor has 99 characters
- The note content editor has 100 characters
- The note content editor has 101 characters

Test Cases Set 3 This set of test cases is corresponding to number of notes.

- The note list is empty
- The note list has 1 note
- The note list has 999 notes
- The note list has 1,000 notes
- The note list has 1,001 notes

2.1.2 Equivalence Partition 2

There are some special characters and tokens in the editor, test whether the test can be save / update to and retrieve correctly.
Boundary Conditions

- The content of the note has quote signs(")
- The content of the note has back quote signs(‘)
- The content of the note has SQL keywords

Test Cases Set 1 This set of test cases is corresponding to the input which has quote / back quote signs. After input the following contents, test whether the input can be saved and retrieved correctly. There are two subsets of test cases, the combination of each of the 2 subsets is a complete test case.
Subset A

- The quote / back quote sign is at the beginning of the test
- The quote / back quote sign is at the middle of the test
- The quote / back quote sign is at the end of the test

Subset B

- All of the quote / back quote signs are in pairs.
- One of the quote / back quote sign is not in pair.

Test Cases Set 2 This set of test cases is corresponding to the input which has SQL keywords.

- There is a complete and valid SQL statement in the note content.
- There are 2 complete and valid SQL statements in the note content.

2.1.3 Equivalence Partition 3

Select a note and click the delete button, test whether it can be delete from database.
Boundary Conditions

- The max number of the notes is 1000
- The title of the note can be empty.
- The title of the note can be non-empty

Test Cases Set 1 This set of test cases are corresponding to the number of the notes in the database.

- When there is no note in the database, click the delete button.
- When there is 1 note in the database, click the delete button.
- When there is 999 note in the database, click the delete button.
- When there is 1,000 note in the database, click the delete button.

Test Cases Set 2 This set of test cases are corresponding to the existence of title.

- When the title is empty, delete the note.
- When the title is not empty, delete the note.

2.2 Test 2: Upload to Google Drive

2.2.1 Equivalence Partition 1

Test the result of uploaded pdf file in google drive with correct input credential and good internet connection.

Test Cases Set 1 This set of test cases are corresponding the correct input credential and good internet connection .

- Click the upload button, then log in the google drive to see whether the corresponding note is uploaded.

2.2.2 Equivalence Partition 2

Test the exception catch for invalid user typed credential or lack of internet connection.

Test Cases Set 1 This set of test cases are corresponding to invalid user typed credential or lack of internet connection..

- Click the upload button, log in with invalid/random credential.
- Click the upload button under no internet connection.

2.3 Test 3: Generate PDF

2.3.1 Equivalence Partition 1

Select a note and call the generate pdf function, test whether the corresponding pdf is generated in the file directory.
Boundary Conditions

- The content of the note can be empty.
- The content of the note can be non-empty

Test Cases Set 1 This set of test cases are corresponding to the notes with or without contents.

- When there is no content in the note, run the generate pdf function.
- When there is content in the note, run the generate pdf function.

2.3.2 Equivalence Partition 2

Select a note and click the generate pdf function, test whether the generate pdf API is triggered.
Boundary Conditions

- The content of the note can be empty.
- The content of the note can be non-empty

Test Cases Set 1 This set of test cases are corresponding to the notes with or without contents.

- When there is no content in the note, run the generate pdf function.
- When there is content in the note, run the generate pdf function.

2.3.3 Equivalence Partition 3

Select a note and click the generate pdf function, test whether the generate pdf API is triggered.

Test Cases Set 1 This set of test cases are corresponding to the notes with or without contents.

- When there is no content in the note, mock out the generate pdf function and run the generate pdf function.
- When there is content in the note, mock out the generate pdf function and run the generate pdf function.

3 Testing Coverage

3.1 Coverage Tool

To measure the the branch coverage achieved by our test suite, we use the coverage tool provided by Karma. In the .my.conf.js file, we add coverage tool as the preprocessor of our code and configure the format of the coverage report as HTML and saved at directory coverage/.
However, the coverage tool provided by Karma does not work with the other test preprocessor so the test result is not accurate. So we decide to temporarily measure our code coverage manually and continues seeking solution for the coverage tool.

3.2 Manually Coverage Measurement

We track every test suites manually and measure the coverage of our code. Currently, our core functions have 159 lines of code totally, our test covers over 150 lines. The all of the uncovered code and branches are the error handling call back functions of the WebSQL API.