Advanced Software Engineering
Final Iteration Report
Team Code Phoenix

Team Members
ag4015 (Aviral Gupta)
ag4020 (Arjun Gupta)
smk2256 (Sachin Kelkar)

Team Mentor
Fujunku Chen

Implemented user stories and use cases

User Story 1 -

As a user, I want a note-taking mechanism that allows me to take notes right in the shell which I am using for development/general use. This avoids the overhead of using a separate application and saves time. Along with notes, I want the app to handle my events, to do lists, reminders, etc. My conditions of satisfaction are -

A. Should have an interface to quickly launch the app.
B. Should be able to create new notes quickly and store to a local storage on the device running the application.
C. Should be able to show previously saved notes taken on the application.
D. Should be able to edit previously saved notes taken on the application.
~~E. Should support multiple types of notes (Note, Event etc).~~
~~F. Before starting any note, I should be asked to select the type.~~

A, B, C and D have already been implemented~~. E and F are part of the wishlist and will be implemented as a part of the final iteration.~~ E and F were not implemented.

Use Case -

Title - Launch the application from the command line and create a new note.

Actor - User of the application

Description - The user launches the application and chooses one of the available options shown to him on the main menu.

Basic Flow -
1. The user navigates to the directory with the application and types python notes.py in the terminal which opens the main menu with list of options.
2. The user can create a new note on the application by pressing 'a' followed by enter.
3. The user is then asked to input the title of the note and the contents of the note.
4. The user is asked whether he would like to save the note or not.
5. If the user chooses to save the note, he is asked to input a password to encrypt the content of the note before storing the note.
6. The user is also asked whether he wants to sync the note to Google Drive.
7. After this, the user is redirected to the home screen for the next action.

Alternate Flow -
Flow 1 -
1. The user can view previously stored notes by pressing 'v' followed by enter.
2. The user is guided to a screen with paginated entries showing a fixed number of entries.
3. The user can select a number, to open the entry corresponding to that entry.
4. The user can go to the next page by pressing 'n' or the previous page by pressing 'p'.
5. The user can go to the previous menu by pressing 'q'.

Flow 2 -
1. The user can quit the application by pressing 'q' followed by enter.

User Story 2-

Since the notes would be stored locally on my machine as well (for offline use), keeping privacy in mind, I want the app to store my notes in some kind of an encrypted format. My conditions of satisfaction are -

A. Notes should be encrypted with a key of my choice
B. Notes shouldn't be available to read in any case without a key
C. I should be informed in case the key I entered is incorrect.

A, B, C have been implemented

Use Case -

Title - Encrypt the content of the note that is stored in the database.

Actor - User of the application

Description - The notes stored in the database need to be encrypted because they are stored locally in a database on the machine on which the application runs. They should not be readable by anyone who gets access to the database. The notes are encrypted by a user input password which is also required to decrypt the note.

Basic Flow -

1. Once the user has added the content to the note, he is asked if he wants to save the note.
2. If the user inputs ‘y’, he is asked to input a password to encrypt the content.
3. On inputting a non-empty password, user is asked whether he wants to sync the note to google drive. For an empty password, user is prompted to input password again.
4. User inputs ‘y’ or ‘n’ after which he can go back to the previous page.

Alternate Flow -

1. If the user wants to view the contents of the note, he selects the entry corresponding to the relevant note on the menu.
2. User is prompted to input the password to retrieve the note.
3. If the user inputs the correct password, he is taken to a menu where he can choose to view the contents of the note, edit the note or view previous versions of the note.
4. If the user inputs an incorrect password, he is shown a prompt which tells him that the input password is incorrect and asks whether he wants to try again.

User Story 3 -

3. As a user who takes notes on multiple platforms, the notes which I explicitly select should be available to me on any device through syncing with a cloud service of my choice. My conditions of satisfaction are -

A. The app should support selective sync to cloud services of my choice(Google Drive, MS OneDrive, etc.)

B. I should have an option to select which notes to sync.

C. The notes that are already synced with some cloud service should sync automatically on updates.

D. Conflicts(for example, if I modify a note directly on the Google Drive and also on my local machine so that sync cannot be performed automatically), if any, should not be resolved automatically, but only after asking me.

We changed A to support only Google Drive for now. ~~Multiple cloud sync support will be supported if time permits.~~ Only Google Drive is supported.
B, C, D have been implemented.
Use Case

Title - Sync notes to Google Drive

Actor - User of the application

Description - The user should have an option to backup his note to a cloud provider(in this case Google Drive) for backup purposes. The user might also edit the notes on Google Drive and thus these changes should be reflected in the corresponding note in the note taking application. The conflicts, if any, should be resolved after asking the user.

Basic Flow -

1. When the user saves a new note, they are given a prompt whether they want to sync with Google Drive.
2. If the user selects "Yes", a *sync* flag is stored as True in DB against the note and the note is then uploaded to the Drive of the user and they exit to main menu.
3. Next time when the user opens this note, the note is checked against the copy on Google Drive. If there is a mismatch/updates, the user is given option to update local copy.
4. If the user selects "Yes", the local note updates and otherwise it doesn't.
5. Now when the user saves this note, the file is automatically synced with Google Drive.

Alternate Flow -

<u>When user decides to not sync the note</u>

1. Instead of choosing to sync the note if the user selects no sync, the file is saved normally.
2. Next time when the file is opened, the local copy loads as this file is not being synced.

<u>When there is no Internet connection and sync is enabled</u>

1. When the file is being synced and the user tries to save the file or open it, an error throws up and the Note Application continues as normal by pressing return.
2. The local copy of note is still created on save.

User Story 4 -

1. As a user who will modify the same notes multiple times, I want the app to keep a version history of my notes. My conditions of satisfaction are -
    A. Keep versioned copies of the notes
    B. It should store upto last 10 versions
    C. Versioning should be automatic
    D. Should be able to get the difference between two versions of a note

Use Case -

Title - Notes Versioning

Actor - User of the application

Description - Maintain and View previous versions of a note and get difference between any 2 versions of the note.

Basic Flow -

1. When the user saves a new note, a versioned copy is created and stored in local storage.
2. When the user chooses to edit a note, after inputting the correct password, he is taken to a menu where he can decide to view the note, edit the note, view previous versions or return to previous menu.
3. On pressing 'v', the user is taken to a menu with a list of previously stored versions for the selected note.
4. He can input the index corresponding to a version entry to view the contents of that version.

Alternate Flow -

1. Instead of inputting the index of an entry, the user can press 'd' to get a diff between two versions.
2. The user is prompted to enter the index of the first version for diff.
3. The user is prompted to enter the index of the second version for diff.
4. If both versions inputted are valid, the user is shown the diff, otherwise he gets a prompt which tells him the input is invalid and asks him to press enter key to continue.

New Features implemented in the Final Iteration

1. As a user, I want a way to sort my notes in an organized way with the help of tags. My conditions of satisfaction are -
    A. I should be able to tag my notes with labels while creating a new note.
    B. Different notes must be properly grouped together in their respective tags.

2. As a user, I do not want to type the password repeatedly for every note. Thus, I want a way to store my password for the session. My conditions of satisfactions are -
    A. I should be able to set a session password
    B. I should be able to reset the session password

We also added colors to the app.

~~2. As a general purpose user who will be using this instead of the normal note taking apps , I want the app to have features of the other note taking apps. My conditions of satisfaction are –~~
~~a. A spell check command to report spelling errors for my notes~~
~~b. Even though it is a command line app, I still want it to have some color(for example – git, vim, etc.)~~
~~c. Word count/Character count for my notes.~~

~~3. As someone who would like to share my notes with others, I want there to be a way to be able to do this easily. My conditions of satisfaction are –~~
~~a. I would want a feature for sharing my notes with a link, to be supported in this app, which I should get in my console.~~
~~b. Sharing should work only by my explicit choice and shouldn't be automatic.~~

Test files in our repository

There are two files starting with test_* in our repository(`test_notes.py` and `test_crypto.py`).

Test plan -

We have note implemented every test as we planned to do it in the final iteration. For now, we have implemented a basic testing functionality.

Equivalence partitions are given in the tables below. Any parameter value not in the equivalence partitions is considered to be invalid.

```
add_entry(data, title, password, sync)
```

data	A string of any length and can contain any characters
title	A non-empty single line string of any length and can contain any characters
password	A non-empty single line string which can contain any characters
sync	Boolean value - [True, False]

```
upload_drive(title, data)
```

title	A non-empty single line string of any length and can contain any characters
data	A string of any length and can contain any characters

`download_drive(entry, title, data, password)`

entry	An object of type Note defined in models.py
title	A non-empty single line string of any length and can contain any characters
data	A string of any length and can contain any characters
password	A non-empty single line string which can contain any characters

`edit_entry(entry, title, data, password)`

entry	An object of type Note defined in models.py
title	A non-empty single line string of any length and can contain any characters
data	A string of any length and can contain any characters
password	A non-empty single line string which can contain any characters

`delete_entry(entry)`

entry	An object of type Note defined in models.py

`view_previous_versions(entry, password)`

entry	An object of type Note defined in models.py
password	A non-empty single line string which can contain any characters

`view_entry(entry, password)`

entry	An object of type Note defined in models.py
password	A non-empty single line string which can contain any characters

`view_entries()`
This function is a helper function which has no parameters and is called by other functions mentioned above. Thus this has no equivalence partitions.

`search_entries()`
This function is a helper function which has no parameters and is called by other functions mentioned above. Thus this has no equivalence partitions.

`process_tags()`

tags	A string of any length and can contain any character. ',' character is used as a delimiter.

`diffcheck()`
This function is a helper function which has no parameters and is called by other functions mentioned above. Thus this has no equivalence partitions.

first	A string value, can take any character. Valid characters are digits less than i
second	A string value, can take any character. Valid characters are digits less than i.
password	A non-empty single line string which can contain any characters
i	A integer value signifying the number of entries in versions list
versions	A list of versions which can have a non-negative quantity of versions which is an object type defined in the file models.py

Coverage

We measure line and branch coverage using the pytest-cov[1] plugin for pytest which internally uses the coverage[2] package in Python. It is configured by a .coveragerc file which has the following contents -

```
[report]
omit =
   download_from_drive.py
   upload_to_drive.py
[coverage:run]
branch = True
```

The branch:True parameter measures branch coverage in addition to line coverage.
TravisCI has been configured to push the coverage reports to the gh-pages branch(https://github.com/s4chin/coms_4156/tree/gh-pages).
Our coverage is currently at ~~**61%**~~ **71%**.

Project repo and related files

- GitHub repo - https://github.com/s4chin/coms_4156
- Coverage configuration file - https://github.com/s4chin/coms_4156/blob/master/.coveragerc

- TravisCI configuration file - https://github.com/s4chin/coms_4156/blob/master/.travis.yml
- pre-commit file - https://github.com/s4chin/coms_4156/blob/master/pre-commit
- Pylint configuration file - https://github.com/s4chin/coms_4156/blob/master/.pylintrc
- TravisCI script to push coverage report to GitHub repo - https://github.com/s4chin/coms_4156/blob/master/.travis/push.sh

References:

1. https://pytest-cov.readthedocs.io/en/latest/
2. https://coverage.readthedocs.io/en/v4.5.x/