SOFTWARE DESIGN DOCUMENT OF NOTE TAKING APP

Introduction:

Note taking desktop application is a useful application for everybody in our busy lives to keep track of everything that has to be done one time so that users don't miss out on simple, small and important tasks, events to be added etc.

This application is designed to provide users with a platform for creating, editing, searching, sorting,viewing,deleting and visualizing the count of pending and completed notes. It aims to enhance productivity by organizing information efficiently.

Objectives:

- Enable users to manage notes seamlessly.
- Facilitate easy access and modification of notes.
- Improve user productivity through effective note management

Significance:

- To make note of to-do list and save some useful information in a handy way.
- To be able to find the useful information quickly on time with a great search feature that uses status of note like ‘completed' or ‘pending’

Audience: The primary users of Notespro include students, professionals, and anyone who needs a digital platform for organizing notes effectively.

Requirements Analysis:

Need of supportive libraries and python programming language:

- Tkinter: GUI toolkit for building the user interface.
- Pymysql: Python MySQL database connector for managing database operations.
- OS module: in Python is used as it provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc. Firstly, needs to import the os module to interact with the underlying operating system.
- Matplotlib: visualize the pending to-do and completed list

System Architecture:

The application follows a client-server architecture where:

- **Client:** GUI built using tkinter that interacts with users.

- **Server:** MySQL database server manages storage and retrieval of notes.

The application opens with launch and quit with exit buttons on the user interface. On launching it, a user has access to the add, edit, delete, search, view notes and visualize the counts of pending and completed notes of the application.

Data Design:

Data Storage:

For this we need to download, install mysql software, set up the database and establish the mysql connection and insert data (here add notes)

- **Database:** MySQL database named Notespro.
- **Table:** test with columns id (note ID), date (the date on which note is added and the reminder date), text (note content) and status (completed/pending)

The text note added in the application uses mysql database to store it and uses 'pymysql' library to connect the databases.

When adding a new note, the application retrieves the note ID and content from the user interface (GUI) and adds this data into the table of the database.

Data Retrieval:

- Search for a Note (Retrieve Note by status(completed/pending) of it)

  When the user searches for a note by its *'status'*, the application queries the database and retrieves the notes.The retrieved data is then displayed or processed further as required by the application.

- Sorting Notes:

  Sorting is typically done at the database query level to retrieve all notes. It displays the list of notes that can be sorted ascendingly or descendingly on i*d, text and status of note* using the ascending and descending toggle icons provided on the headers of treeview/table..

Data Manipulation:

- Edit a Note

  When editing a note, the application updates the *note content* and *status* in the database using SQL.

The updated content is extracted from the GUI, and the database is updated accordingly.

- Delete a Note

  Deleting a note involves executing an SQL to remove the note from the database. The note is extracted and selected from the GUI input, and the deletion operation is performed.

User Interface Design:

The UI is designed to be intuitive and user-friendly:

- **Launcher Interface:** Allows to launch /exit the application. Launch leads to the functionality accessor interface where all note taking operations can be accessed to perform.
- **Functionality Accessor Interface:** It is a functionality accessor that launches a screen with options to add, edit, search by note status, delete,view notes and visualize the count of completed and pending notes.
- **Add Interface:** Text entry for note content with buttons for saving.
- **Edit Interface:** Allows for edit, delete selected note,delete all notes at once and refresh the notes.
- **View AllNotes Interface:** Allows to view notes in ascendingly/descendingly sorted manner based on the column headers(ID, note text and status) of the table displayed.
- **Search Interface:** Allows to search the notes based on its status such as completed or pending.

continued

Home page of Note Taking Desktop Application:

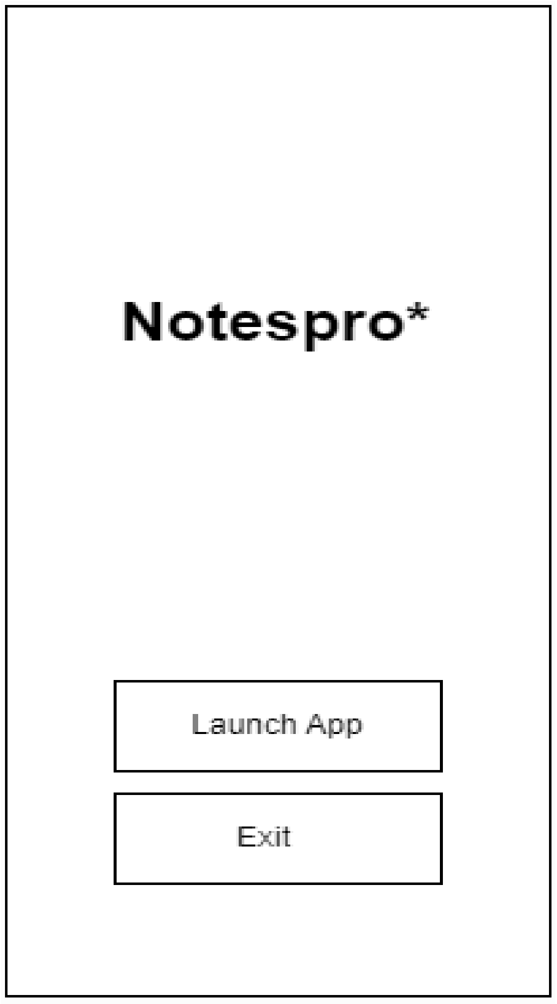


Fig: **Launcher Interface:** Allows to launch /exit the application. Launch leads to the functionality accessor interface where all note taking operations can be accessed to perform.

Functionality accessor (CRUD operationality) of Note making app:

Add Note:

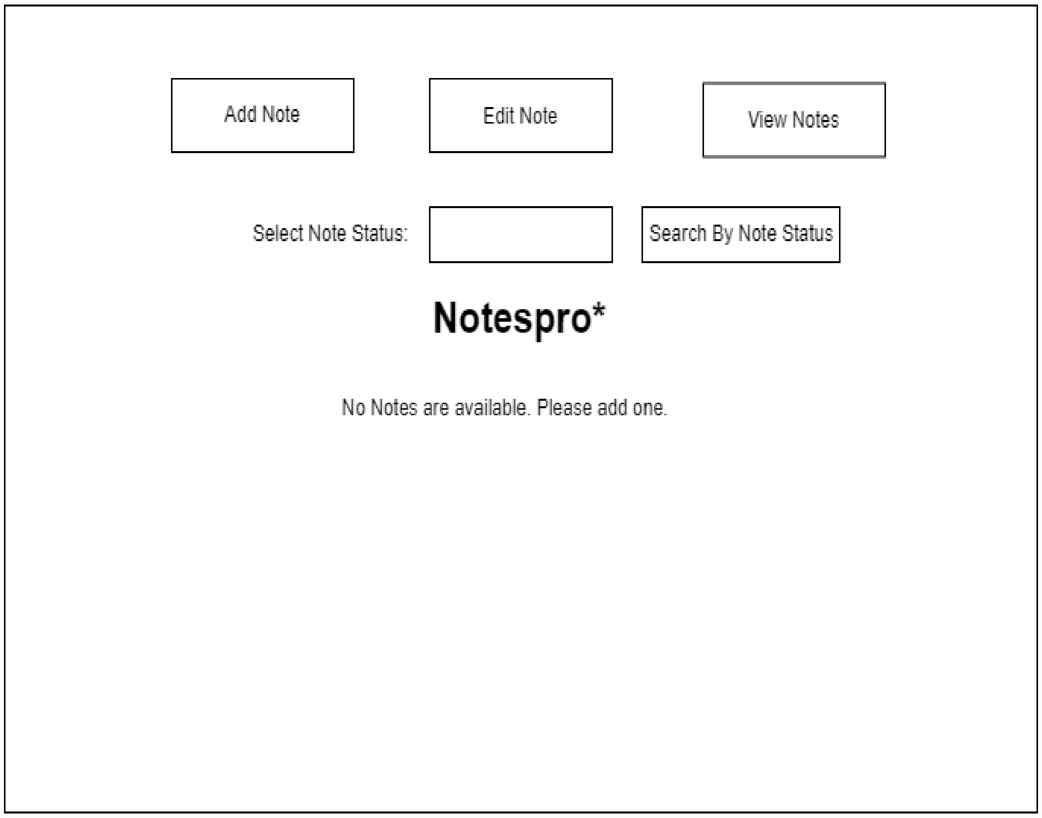


Fig: **Functionality Accessor Interface-** launches a screen with options to add, edit, search by note status, delete,view notes and visualize the count of completed and pending notes.

Add Note:

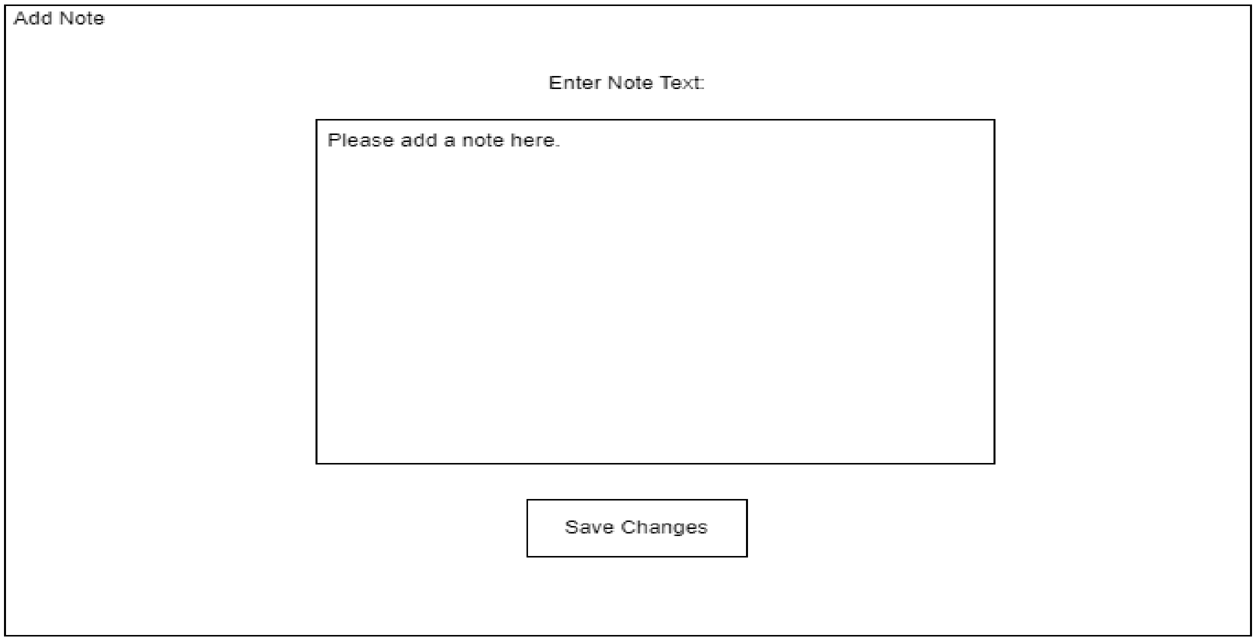


Fig: **Add Interface-**Text entry for note content with buttons for saving.

Edit Note Interface:

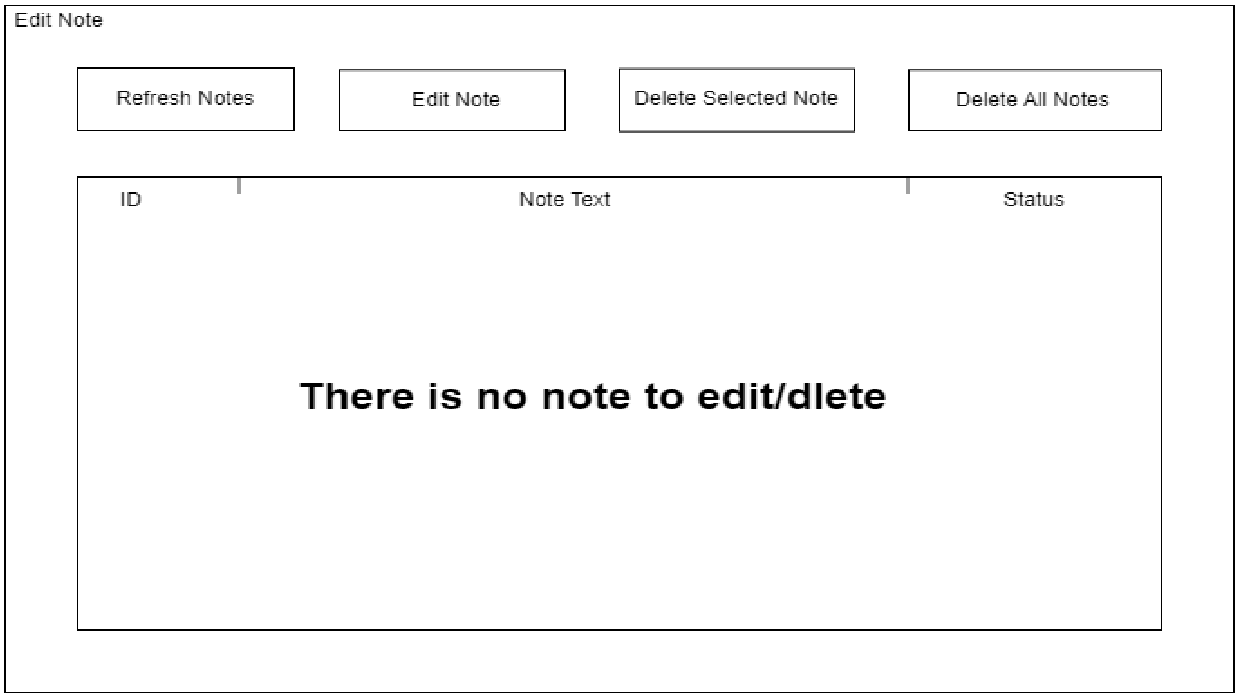
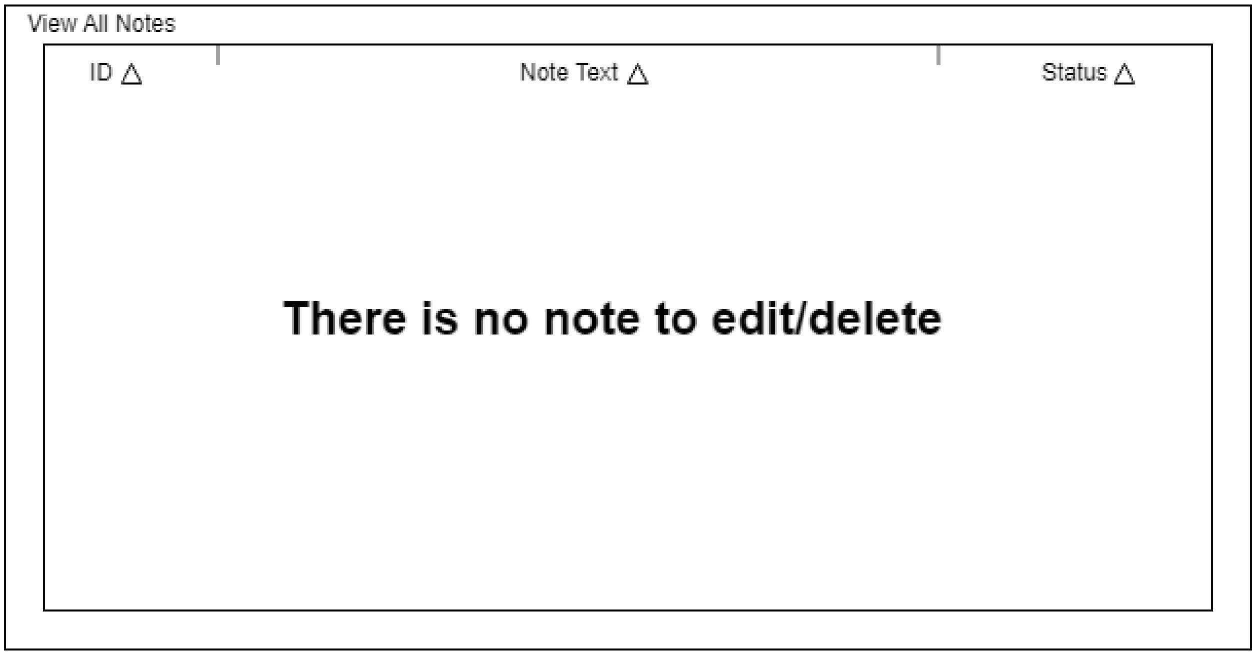


Fig: **Edit Interface -** allows for edit, delete selected note,delete all notes at once and refresh the notes

View All Notes Interface:



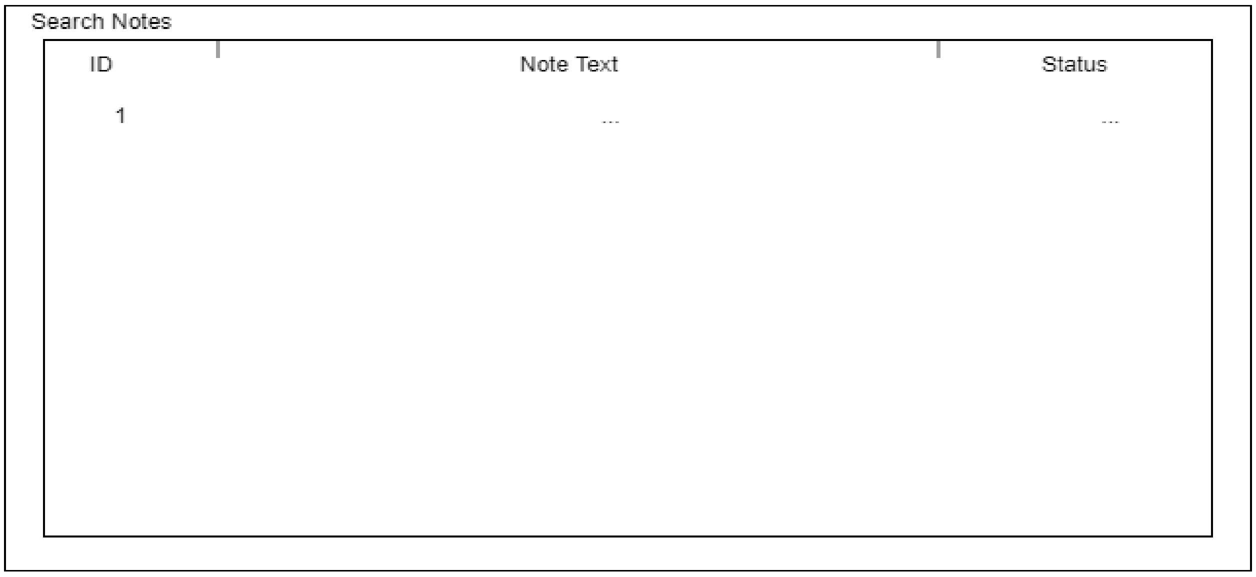
View All Notes

ID △	Note Text △	Status △

There is no note to edit/delete

Fig: **View All Notes Interface -** allows to view notes in ascendingly/descendingly sorted manner based on the column headers(ID, note text and status) of the table displayed.

Search Interface:



Search Notes

ID	Note Text	Status
1	...	...

Fig: **Search Interface - a**llows to search the notes based on its status such as completed or pending.

Module/Component Design:

1. **Main Module (start module):**
    - Launches the application with options to start Notespro or exit.
2. **Functionality Accessor Module (CRUD operations Accessor):**
    - Contains button functionality for adding, editing, searching by note status, sorting by note ID, text, status, viewing, deleting and visualizing completed/pending notes connected to other modules to perform these operations.
    - Uses ‘tkinter’ for GUI elements and ‘pymysql’ for database connectivity.

The components in this module are as follows:

<u>Add component:</u>

A user accesses the *‘Add Note’* component using the add button on the interface to add the note for his/her utilities like any to-do lists etc. to look up later handily. It launches GUI with note to be entered/added by the user and click the save button. This saves the note added in the text area with default status as ‘pending’ on the autoincrement ID.

<u>Edit component:</u>

Using this *‘Edit Note’* component, the earlier added note can be modified using the edit button on the interface. It launches another window with the GUI that prompts users to select the note to be edited and click on *’Edit Selected Note’* button to edit the note or/and change the status of note (optional) and click the *‘Save Changes’* button to save the edited/updated note with existent ID.

<u>Delete component:</u>

With this *‘Delete’* component, the user deletes the note added as per his/ her requirement like after the completion of the task etc. using the delete button on the graphical user interface of the application. Here this component works by the deletion of note by selecting the note and clicking on ‘*Delete Selected Note*’ so that delete operation occurs.

<u>Search By Note Status component:</u>

This component provides the user with a *Search by Note Status* button and a combo box to select the status of note taken so that a list of notes could be obtained for the viewer to look at either of all the completed notes or pending notes at once.

<u>View all Notes component:</u>

It displays the list of notes in a treeview/table with column headers having sort *toggles* that can be *sorted ascendingly/descendingly* on *ID*, *Note*(the text added), *Status* columns

Testing Strategy:

Testing Methods:

Functional Testing:

Objective: Validate the functional requirements of the application.

Expected Outcome:

Test the application's functionality such as note creation, editing, and deletion.

Test navigation between different screens or pages.

Testing Procedures:

The 4 phases of testing procedure are as follows:

- Test Plan Creation:
  - Develop a detailed test plan outlining testing objectives, scope, resources, and timelines.
- Test Case Development:
  - Create specific test cases for each functional requirement and scenario.
  - Include preconditions, test steps, expected results, and actual results.
- Execution of Tests:
  - Execute tests according to the test plan.
  - Record test results and any issues encountered during testing.
- Defect Reporting:
  - Document and report any defects found during testing.
  - Prioritize and track the resolution of defects.
- Regression Testing:
  - Perform regression tests after fixes or changes to ensure existing functionality is not affected.

Functional test cases are designed to validate the key functionalities of the note-taking application:

Launcher/Main Interface Functional Test Cases:

Set up: Connected to database:

Test Case 1: Application Launch

Objective: Verification of correct launch of application.

Steps:

Verify that the window background color is #145660.

Verify that the window is opened with the correct border and relief properties.

Click ‘launch app‘ button on the opened window

Expected Result: It should open a window where all functionalities like add, edit,delete, view, search and visualize the notes by status

Result Obtained:

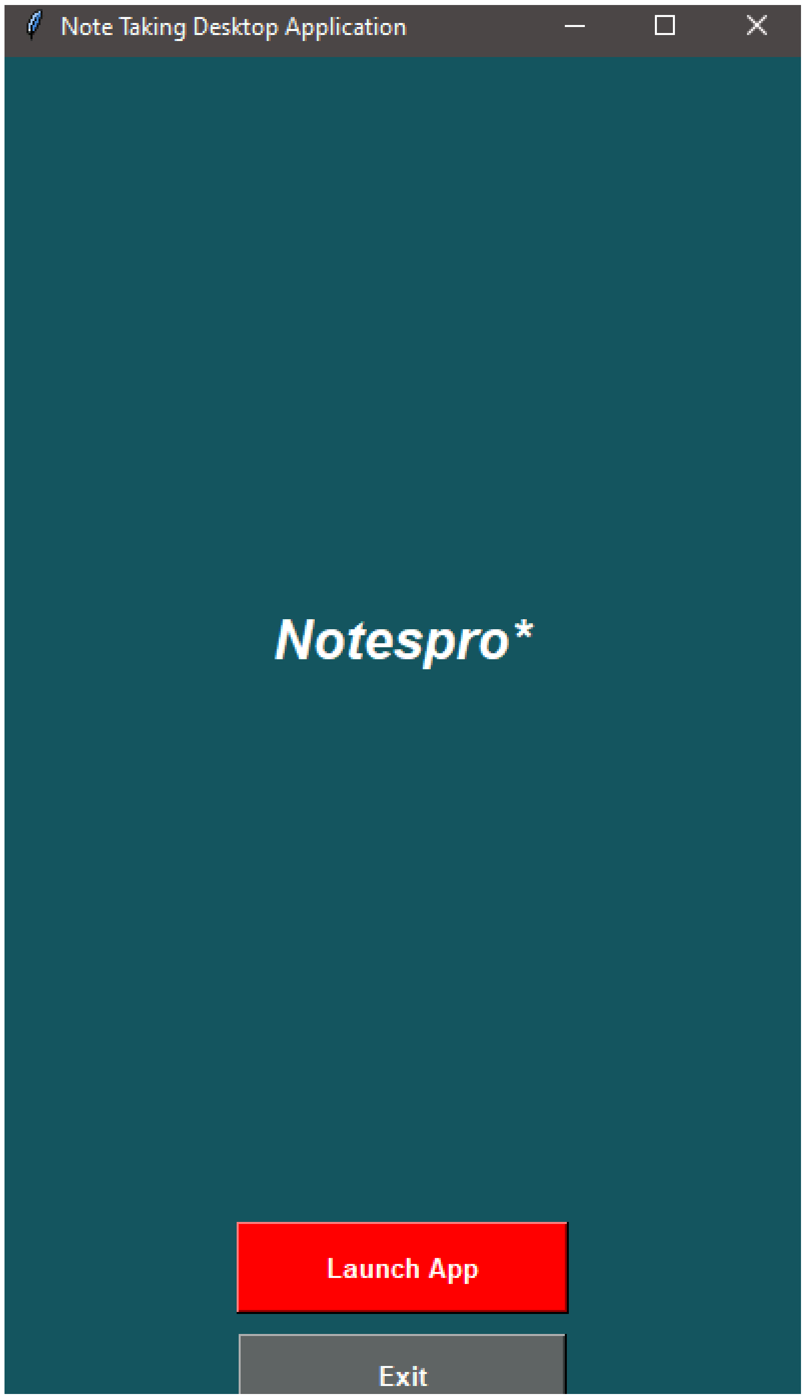
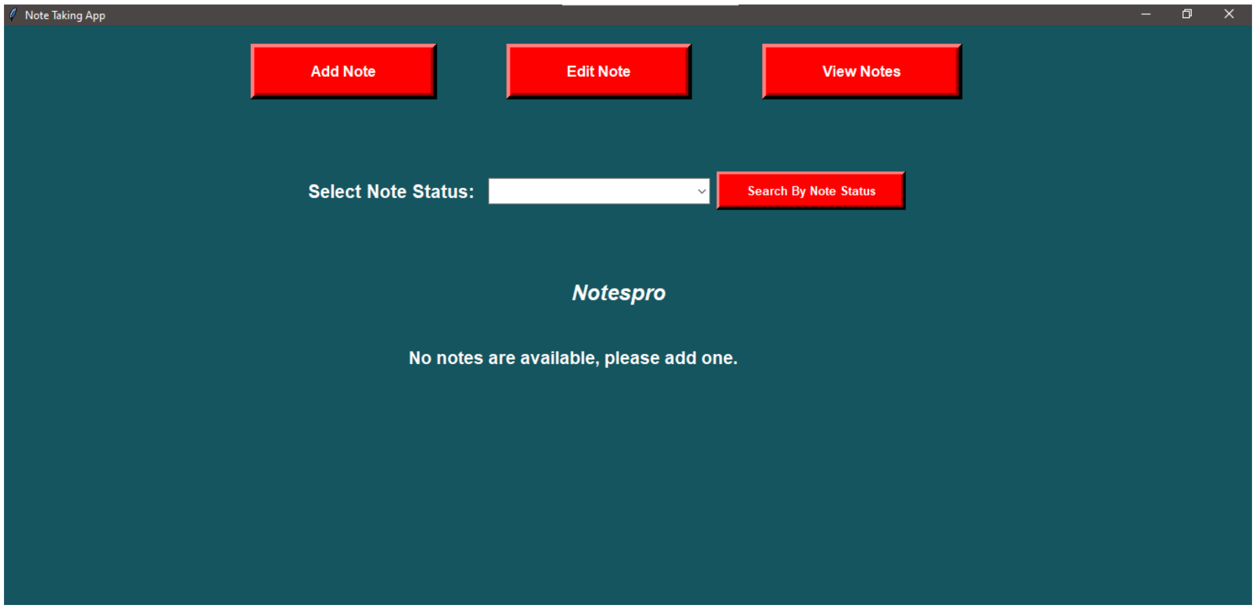


Fig: Launch page of Note taking desktop application



No notes are available, please add one.

Fig: *Launch App* button directed to *Functionality accessor* page

Test case 2: Exit Application

Objective: Verify that the application exits correctly.

Steps:

On launching the application,verify that the window background color is #145660.

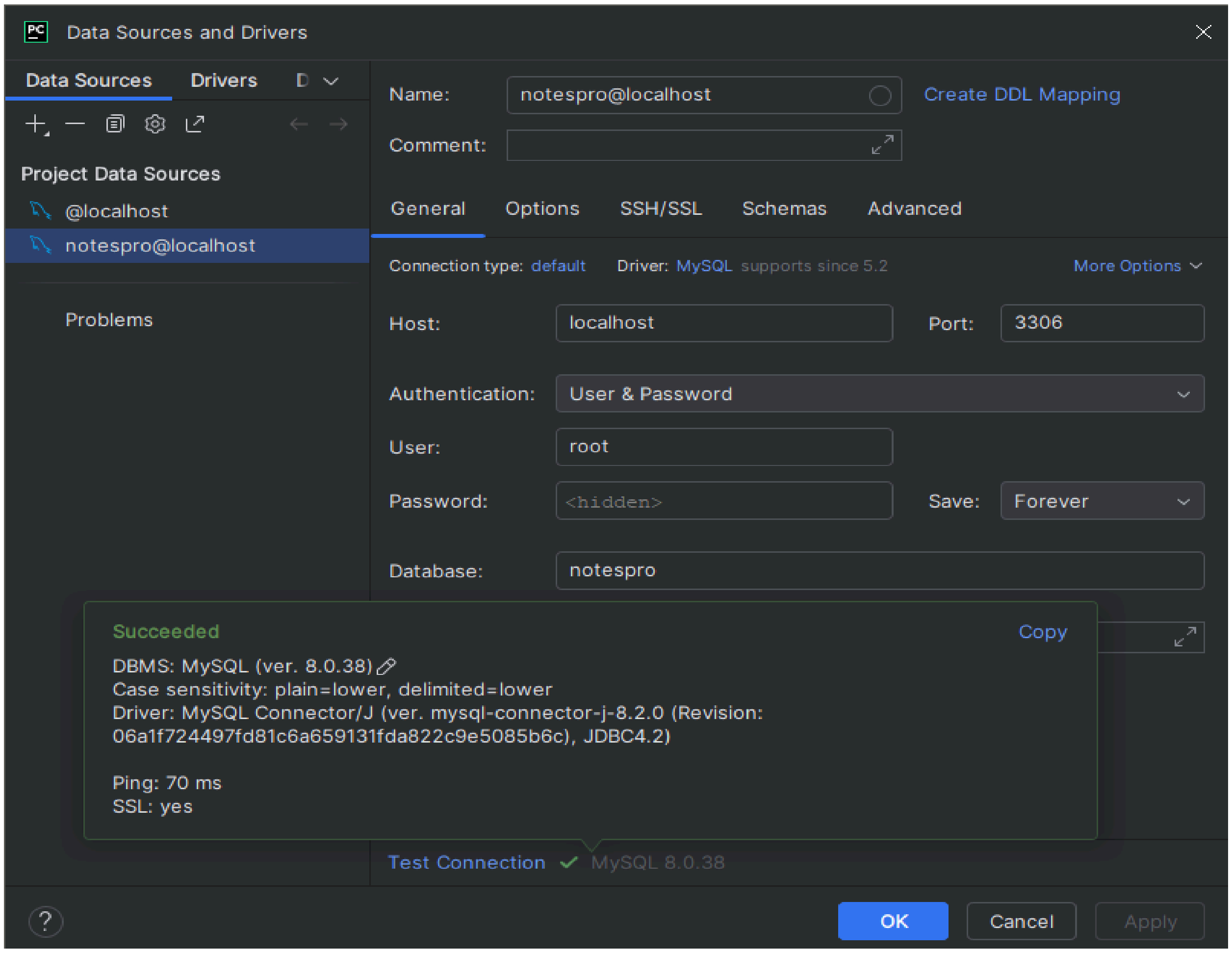
Verify that the window contains the correct border and relief properties.

Click the "Exit" button.

Expected Result: The application closes.

Set up: Connected to database

continued



Name: notespro@localhost
Comment:
Connection type: default Driver: MySQL supports since 5.2
Host: localhost Port: 3306
Authentication: User & Password
User: root
Password: <hidden> Save: Forever
Database: notespro

```
Succeeded
DBMS: MySQL (ver. 8.0.38)
Case sensitivity: plain=lower, delimited=lower
Driver: MySQL Connector/J (ver. mysql-connector-j-8.2.0 (Revision:
06a1f724497fd81c6a659131fda822c9e5085b6c), JDBC4.2)

Ping: 70 ms
SSL: yes
```

Fig: connection to *notespro* database

Functional Test Cases of *Add Note* component:

Test Case 1: Add Note using component *'Add Note'*

Description: Verify that a new note can be added successfully.

Steps:

Launch the application.

On The opened Functionality accessor page, click the "Add Note" button.

In the "Add Note" dialog box, enter "Test Note" in the note text area.

Click the "Save Note" button.

Expected Result:

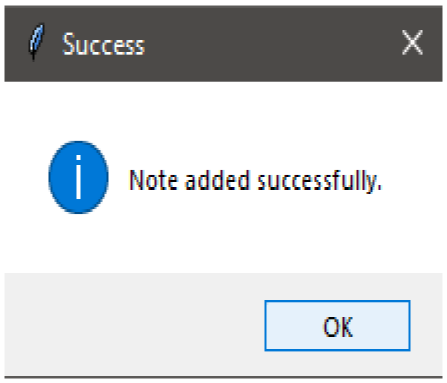
The note should be added to the database with the status *"Pending"*.

A success message *"Note added successfully."* should be displayed.

The text area should be cleared and reset to the placeholder text *"Please add a note here."*.

The visualization chart should be updated to reflect the new note.

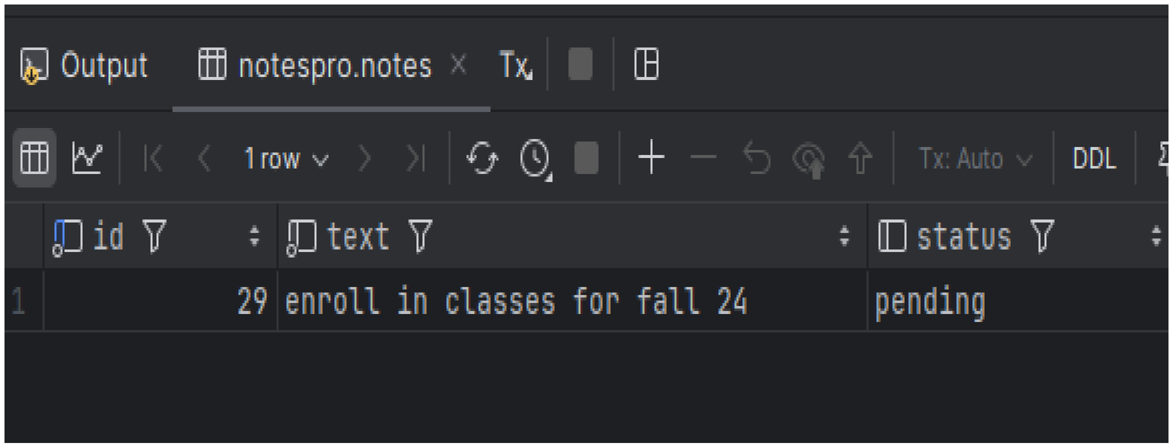
Result Obtained:


Success

Note added successfully.

Fig:A success message "Note added successfully." should be displayed

The note added is as follows and shown here by retrieving from the database:



	id	text	status
1	29	enroll in classes for fall 24	pending

Fig:The note should be added to the database with the status "Pending".

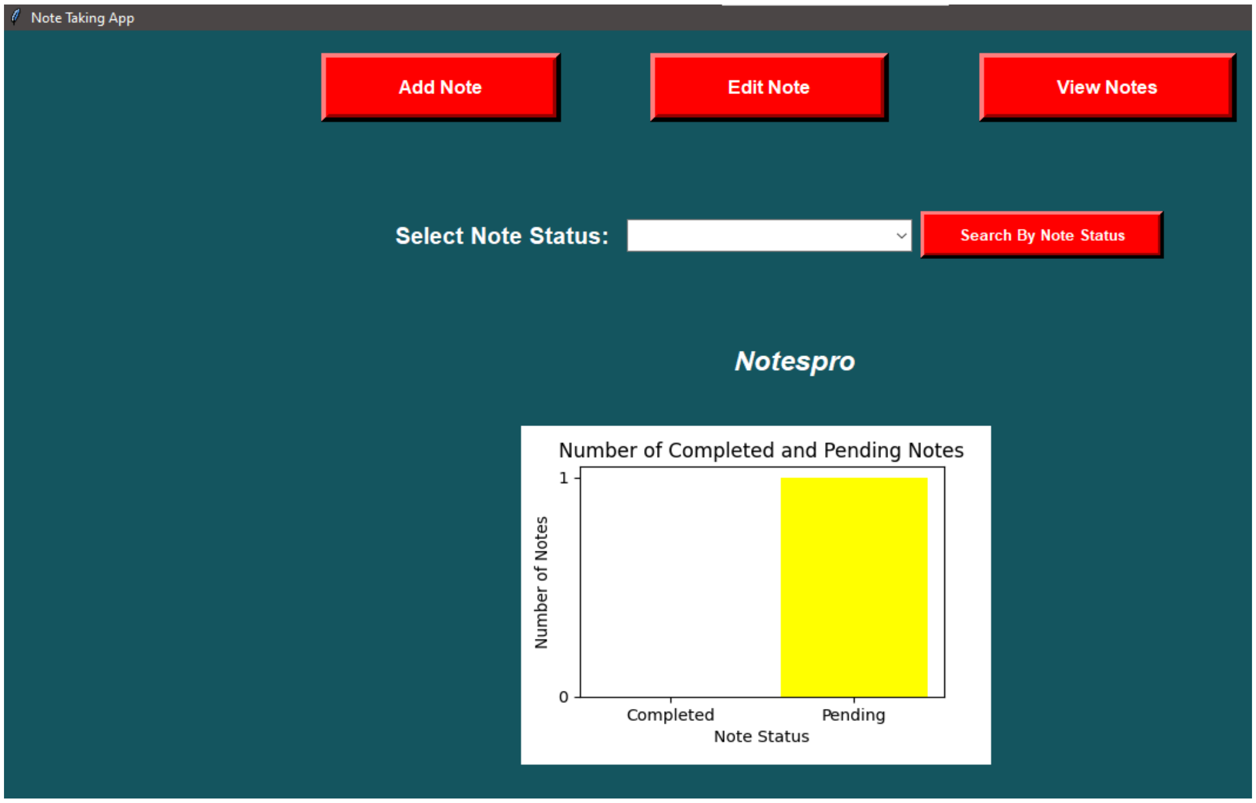


Fig:The visualization chart updated to reflect the new note.

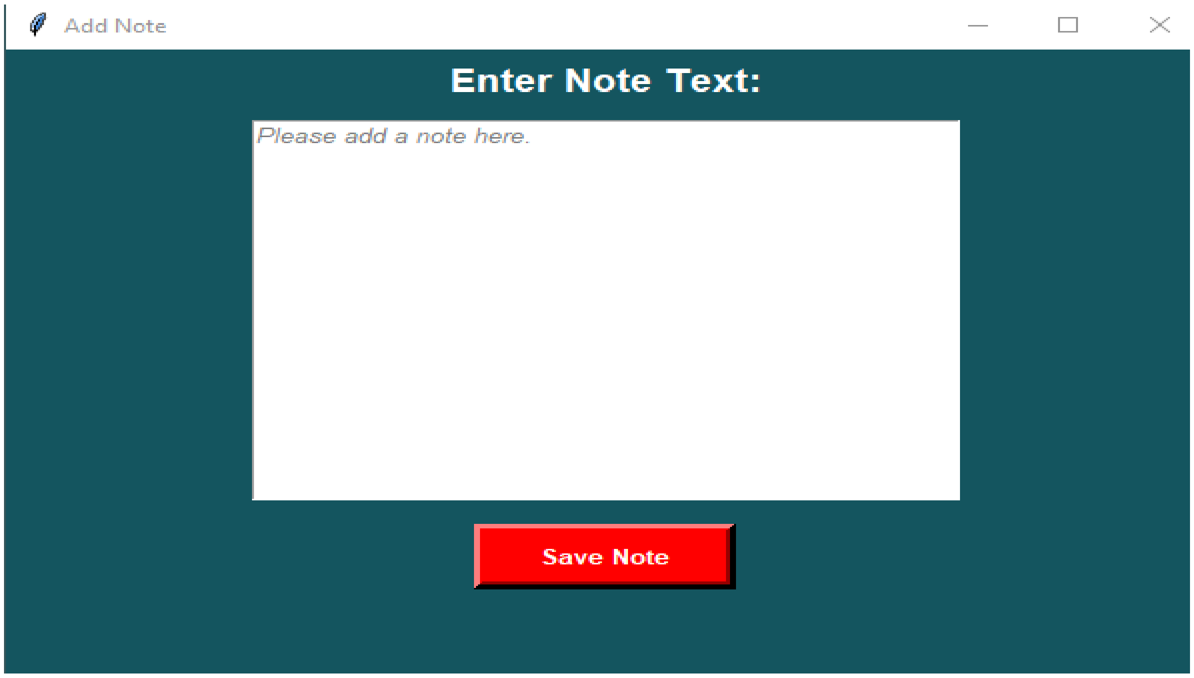


Fig: The text area should be cleared and reset to the placeholder text "Please add a note here.".

Test Case 2: Add Note with Text length of above 255 characters

Description: Verification of an error message is shown when trying to add a note with text length of above 255 characters

Steps:

Launch the application.

Click the "Add Note" button.

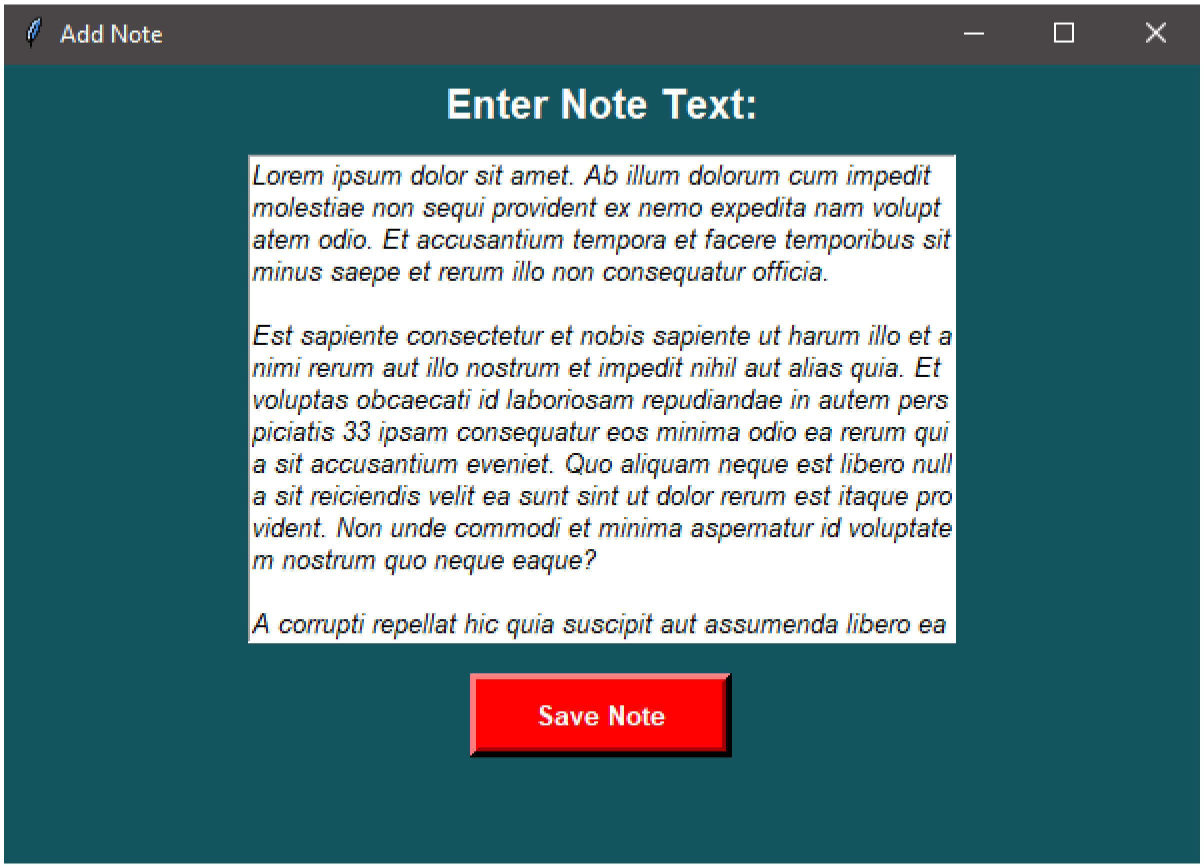
In the "Add Note" dialog box, leave the note text area with Text length of above 255 characters.

Click the "Save Note" button.

Expected Result:

An error message "data too long for column text at row1" should be displayed.

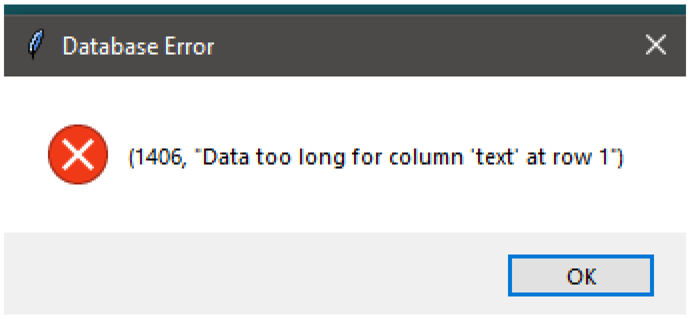
The note should not be added to the database.



Lorem ipsum dolor sit amet. Ab illum dolorum cum impedit molestiae non sequi provident ex nemo expedita nam volupt atem odio. Et accusantium tempora et facere temporibus sit minus saepe et rerum illo non consequatur officia.

Est sapiente consectetur et nobis sapiente ut harum illo et a nimi rerum aut illo nostrum et impedit nihil aut alias quia. Et voluptas obcaecati id laboriosam repudiandae in autem pers piciatis 33 ipsam consequatur eos minima odio ea rerum qui a sit accusantium eveniet. Quo aliquam neque est libero null a sit reiciendis velit ea sunt sint ut dolor rerum est itaque pro vident. Non unde commodi et minima aspernatur id voluptate m nostrum quo neque eaque?

A corrupti repellat hic quia suscipit aut assumenda libero ea


Database Error

(1406, "Data too long for column 'text' at row 1")

Fig:When a user enters a note with character count of above 255 characters,the above database error appears.

Test Case 3: Add Note with Empty Text

Description: Verify that an error message is shown when trying to add a note without entering text.

Steps:

Launch the application.

Click the "Add Note" button.

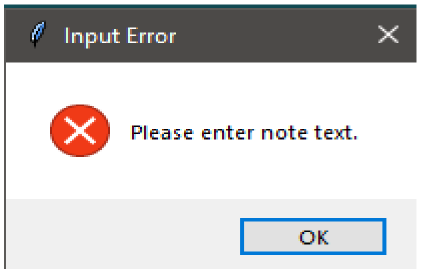
In the "Add Note" dialog box, leave the note text area empty.

Click the "Save Note" button.

Expected Result:

An error message "Please enter note text." should be displayed.

The note should not be added to the database.


Input Error

Please enter note text.

Fig:When user tries to click ‘Save Note’ without adding note, the following error message box appears.

Functional Test Cases of Edit Note:

Test Case 1: *Edit Note* added using *Add Note* component

Description: To verify that an existing note can be edited successfully.

Steps:

Launch the application.

Click the "Edit Note" button.

Select an existing note from the Table.

Click the "Edit Selected Note" button.

In the "Edit Note" dialog box, change the note text to "Updated Test Note" and status to "Completed".

Click the "Save Changes" button.

Expected Result:

The note should be updated in the database with the new text and status.

A success message "Note updated successfully." should be displayed.

The Table should be updated to reflect the changes.

The visualization chart should be updated to reflect the changes.

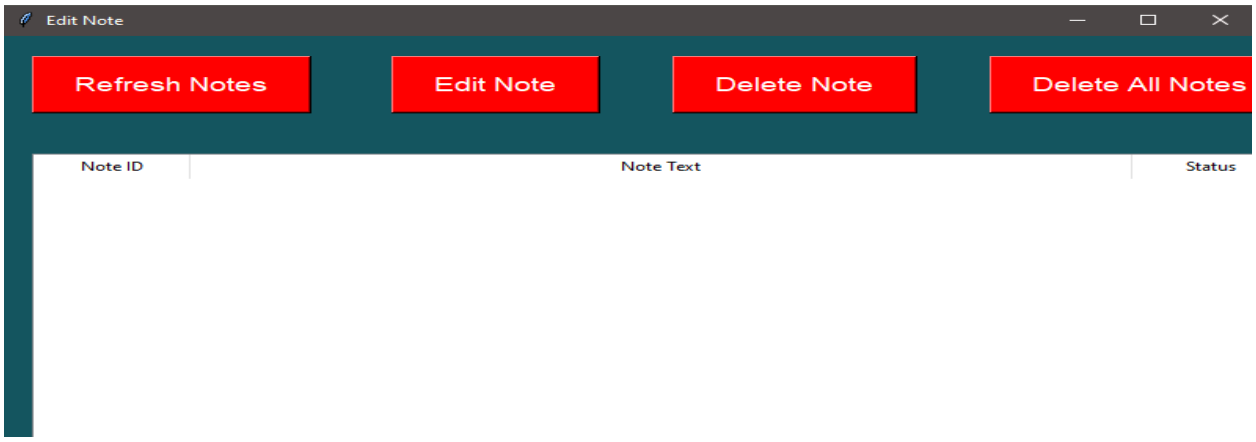
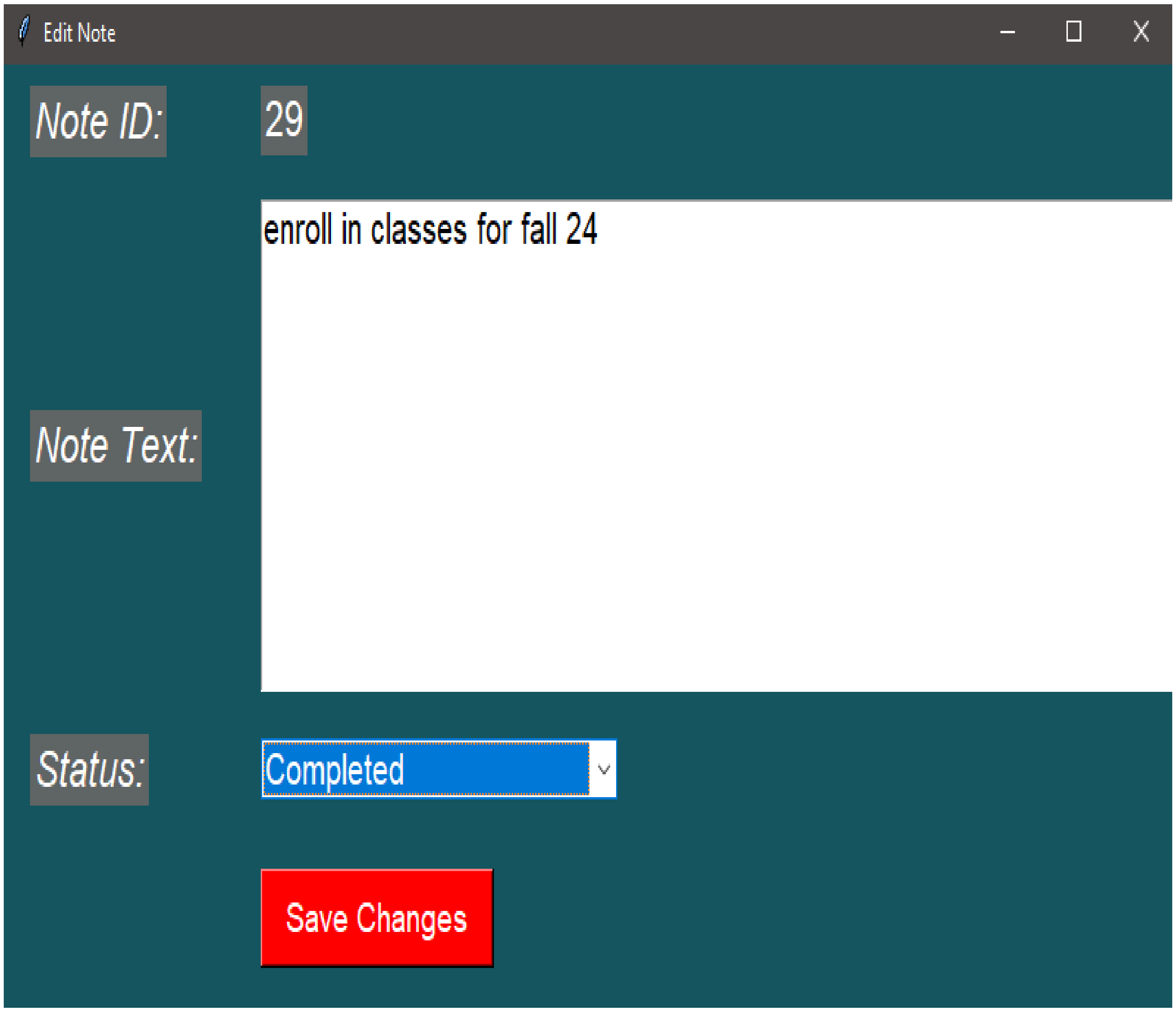


Fig: On clicking '*Edit Note*', the above window launches to *'Edit a Selected Note'*


Edit Note

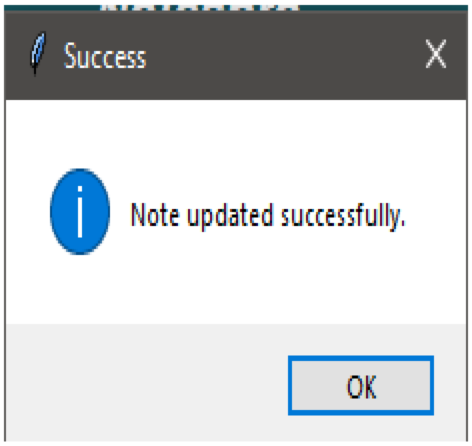
Note ID: 29

Note Text: enroll in classes for fall 24

Status: Completed

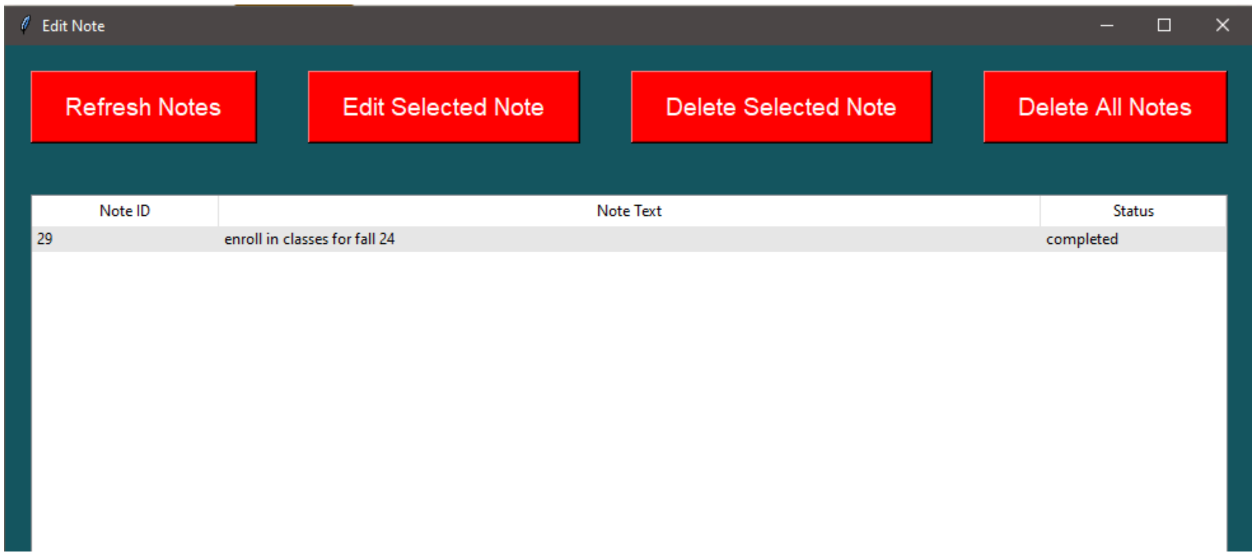
Save Changes

Fig:A success message "Note updated successfully." should be displayed.


Success

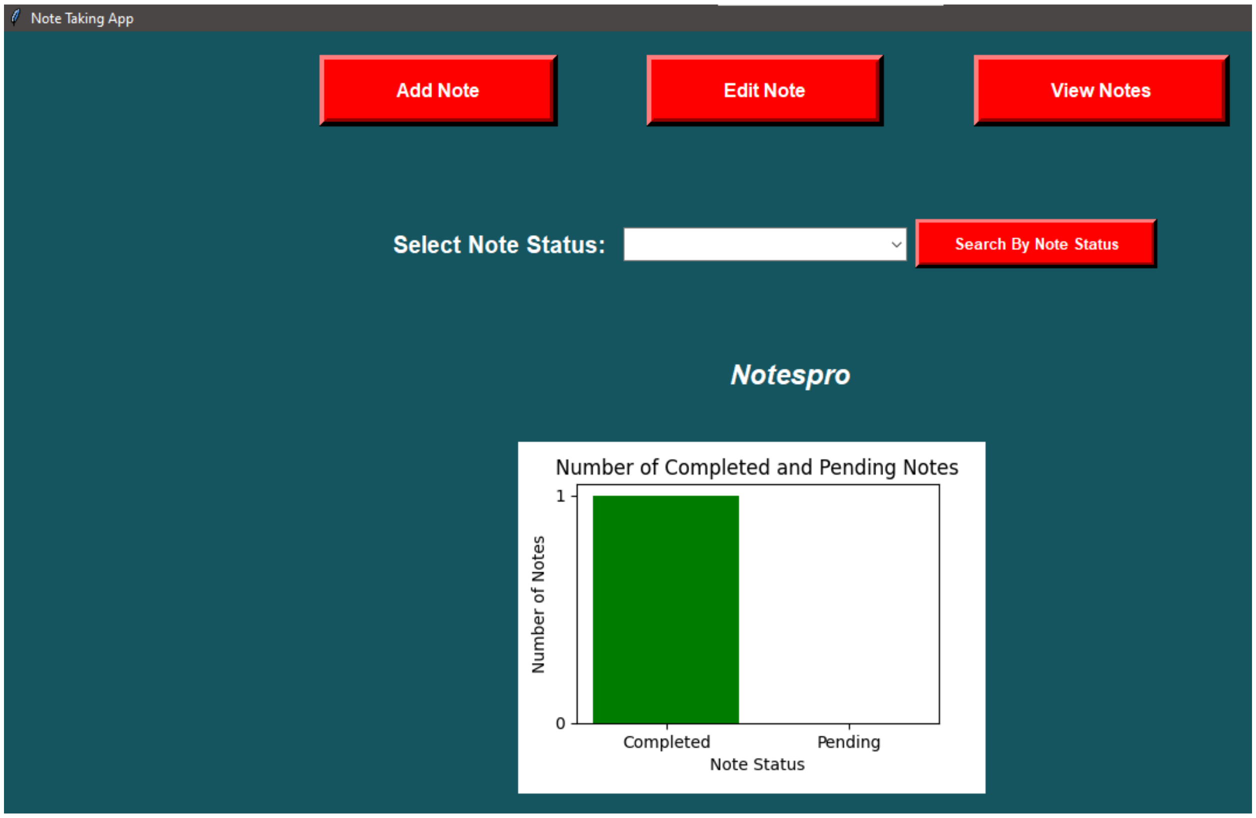
Note updated successfully.

OK



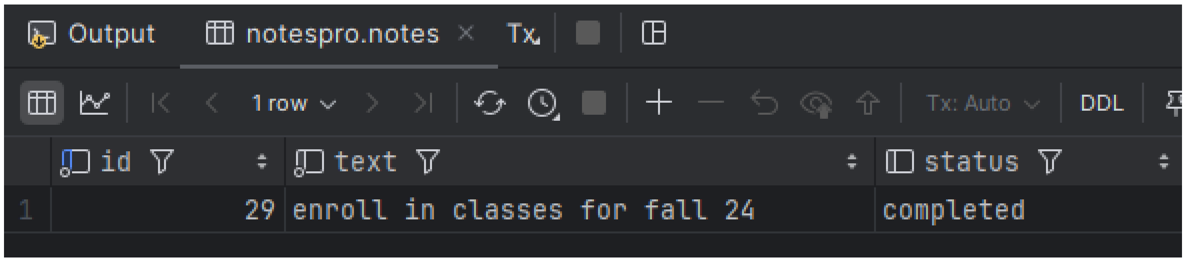
Note ID	Note Text	Status
29	enroll in classes for fall 24	completed

Fig:The Table is updated to reflect the changes.



Fig;The visualization chart is updated to reflect the changes.

In the database too it is updated as follows:


Output | notespro.notes

id	text	status
29	enroll in classes for fall 24	completed

Fig:The note is updated in the database with the new text and status.

Test Case 2: Edit Note with status from 'completed ' to 'pending'

Description: Now also changed the status back to 'pending' and verified that an existing note could be edited successfully.

Steps:

Launch the application.

Click the "Edit Note" button.

Select an existing note from the Table.

Click the "Edit Selected Note" button.

In the "Edit Note" dialog box, change the note text to "Updated Test Note" and status to "Pending".

Click the "Save Changes" button.

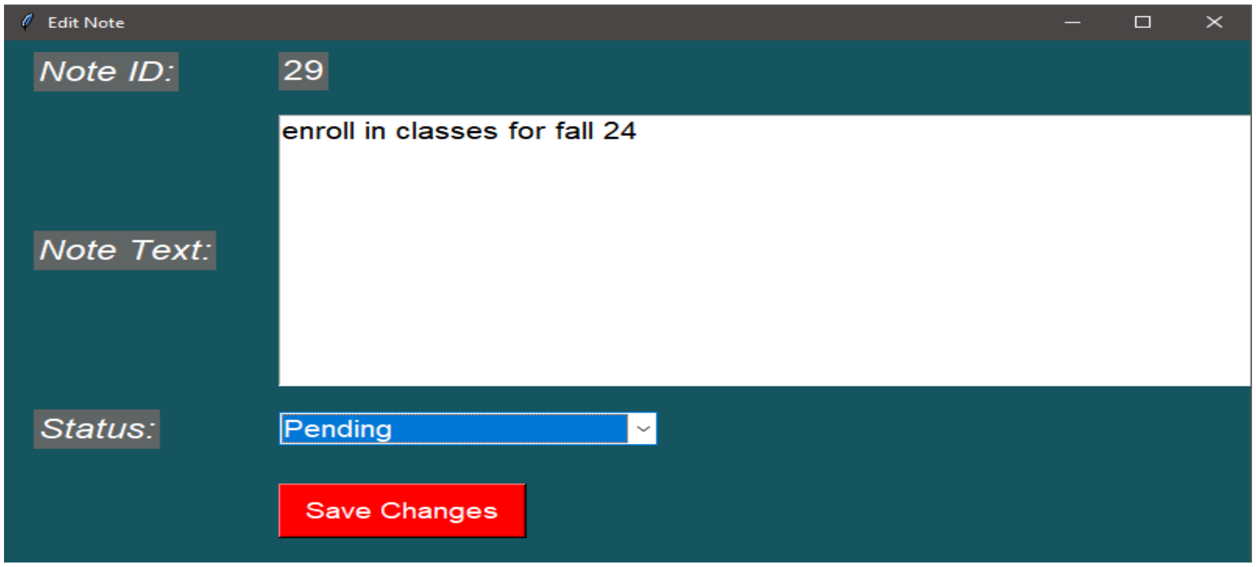
Expected Result:

The note should be updated in the database with the new text

A success message "Note updated successfully." should be displayed.

The Table should be updated with recent text note and status to pending.

The visualization chart should be updated to reflect the changes to 'pending'.



Note ID: 29

Note Text: enroll in classes for fall 24

Status: Pending

Fig: Initial text *'enroll in classes for fall 24'*

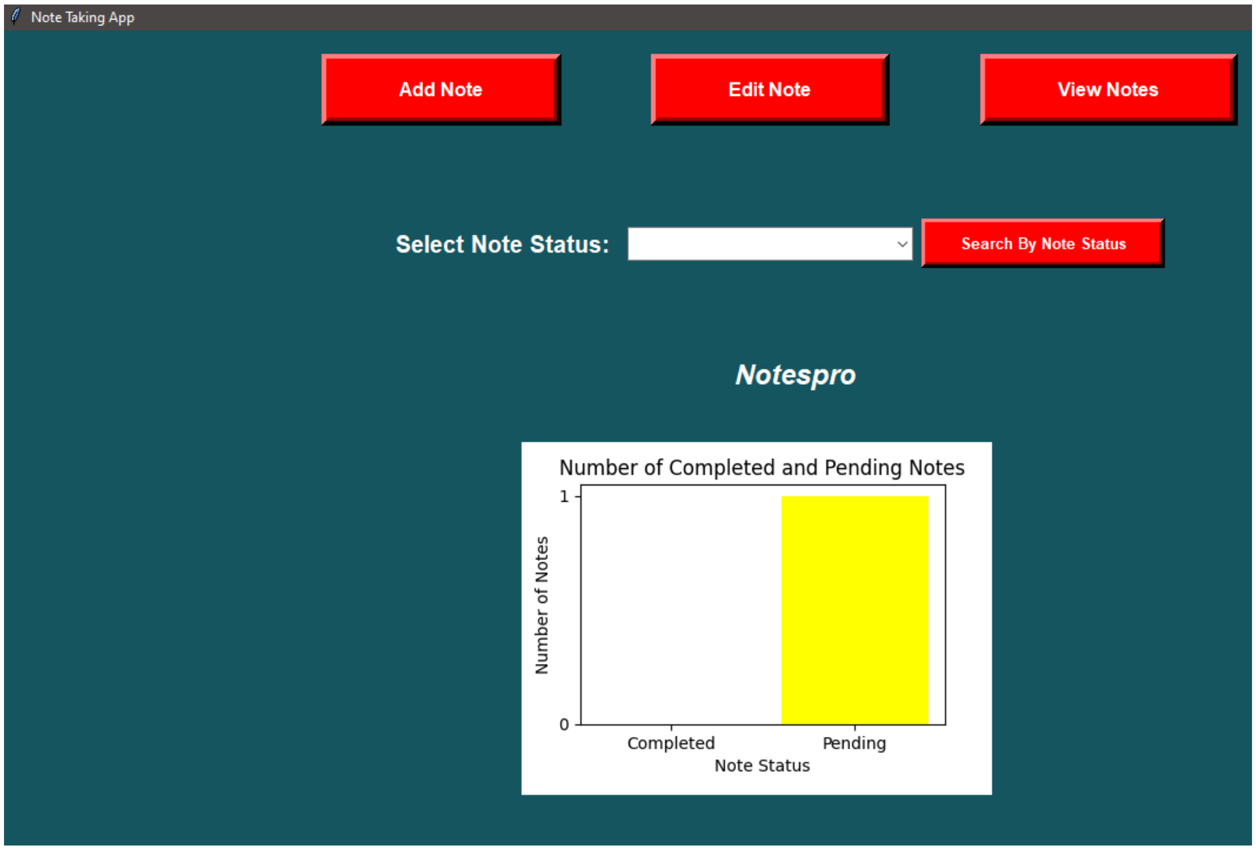
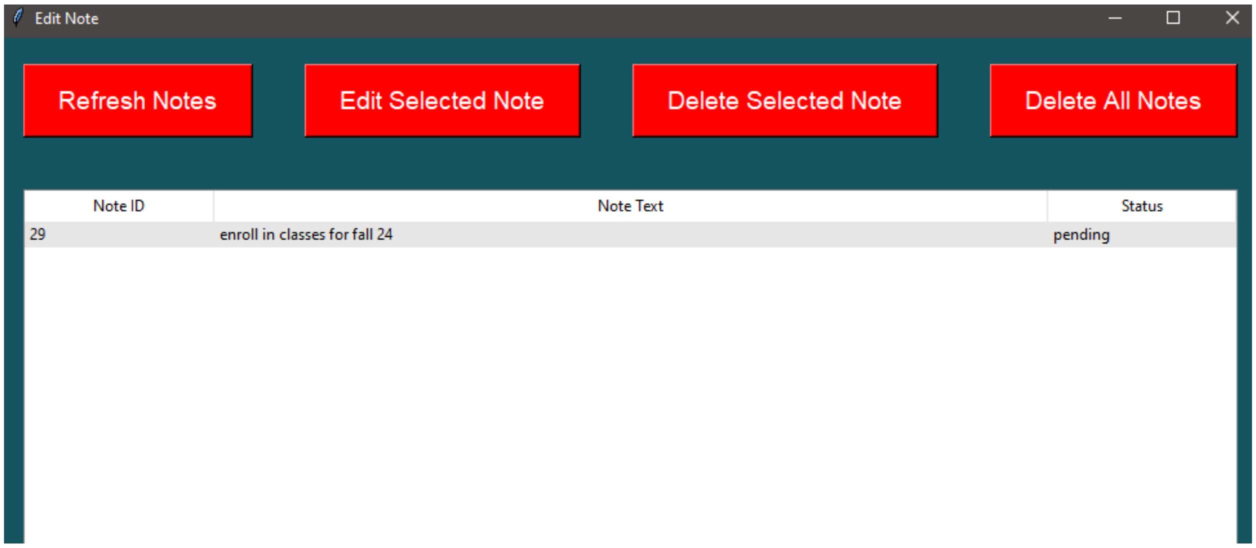
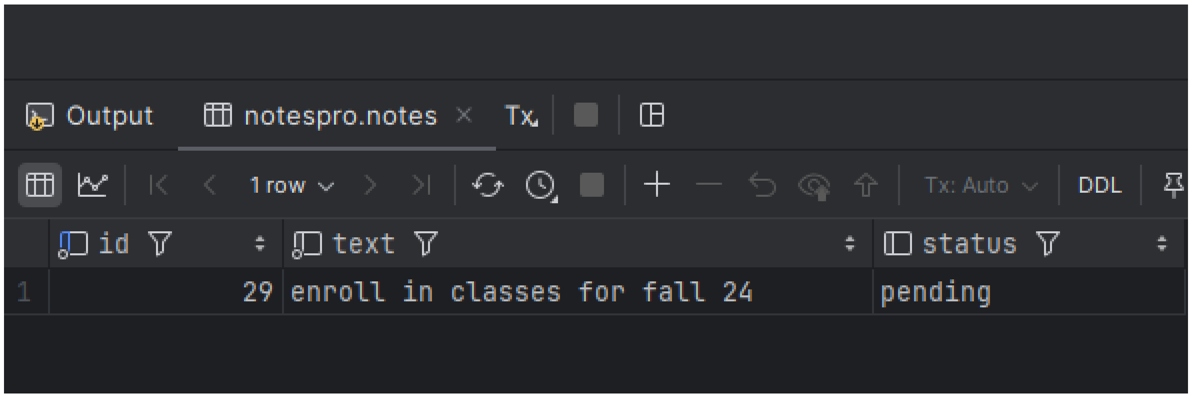


Fig: The visualization chart should be updated to reflect the changes to 'pending'.



Note ID	Note Text	Status
29	enroll in classes for fall 24	pending



	id	text	status
1	29	enroll in classes for fall 24	pending

Fig: The note updated in the database with the new text and status.

Test Case 3: Edit Note text without changing the status of it in the combobox

Description: Verify that an existing note can be edited successfully by just changing the text without changing the status. Changed the text from ‘enroll in classes for fall 2024’ to

‘enroll in classes for summer 2024’.

Steps:

Launch the application.

Click the "Edit Note" button.

Select an existing note *‘enroll in classes for fall 2024’* from the Table.

Click the "Edit Selected Note" button.

In the "Edit Note" dialog box, update/edit it to *'enroll in classes for summer 2024'*..

Click the "Save Changes" button.

Expected Result:

The note should be updated in the database with the new text *'enroll in classes for summer 2024'*.

A success message "Note updated successfully." should be displayed.

The Table should be updated to reflect the changes.

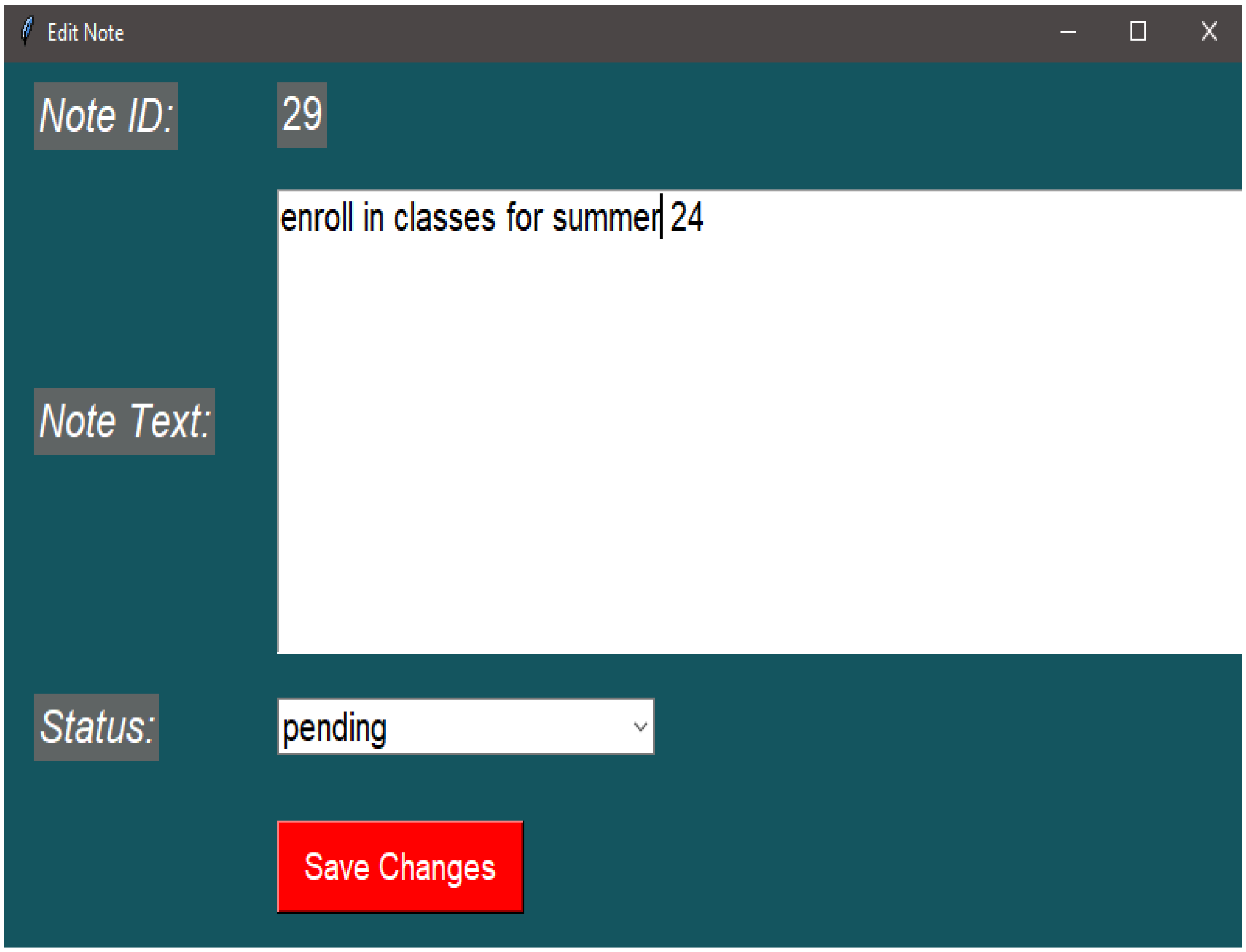
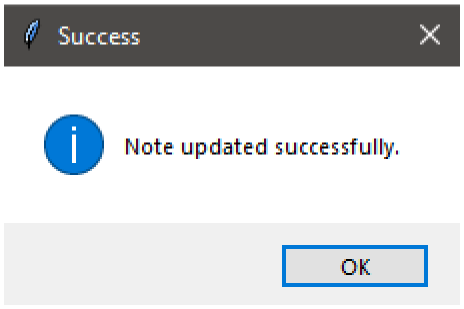
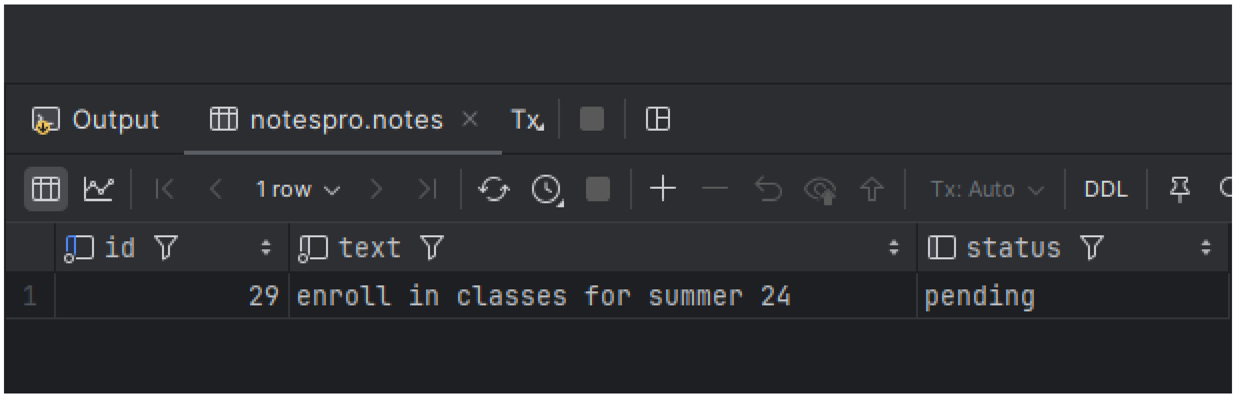


Fig: The note should be updated in the database with the new text *'enroll in classes for summer 2024'*.


Success

Note updated successfully.

Fig: The note updated in the database with the new text *'enroll in classes for summer 2024'* with success message "Note updated successfully." displayed.



	id	text	status
1	29	enroll in classes for summer 24	pending

Fig: The Table in the database should be updated to reflect the changes from 'fall' to 'summer' in the note text

Test Case 4: Click *'Edit Selected Note'* without selecting note (error case)

Description: Edit note clicking the "Edit Selected Note" button without selection of an existing note from the Treeview/table.

Steps:

Launch the application.

Click the "Edit Note" button.

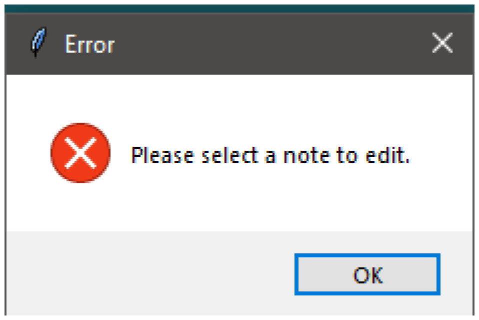
Click the "Edit Selected Note" button without selection of an existing note from the Treeview/table.

Expected Result:

An error message "*Please select a note to edit*" should be displayed.

Obtained Result:

When no note is available to edit,if we click ‘Edit Note’ button it displayed as follows:


Error

Please select a note to edit.

Fig:An error message "*Please select a note to edit*" displayed.

Functional Test Cases of ‘Delete Note’ :

Test Case 1: Delete Selected Note

Description: To verify if an existing note can be deleted successfully.

Steps:

Launch the application.

Click the "Edit Note" button.

Select an existing note from the Treeview/table.

Click the "Delete Selected Note" button.

Confirm the deletion in the confirmation dialog.

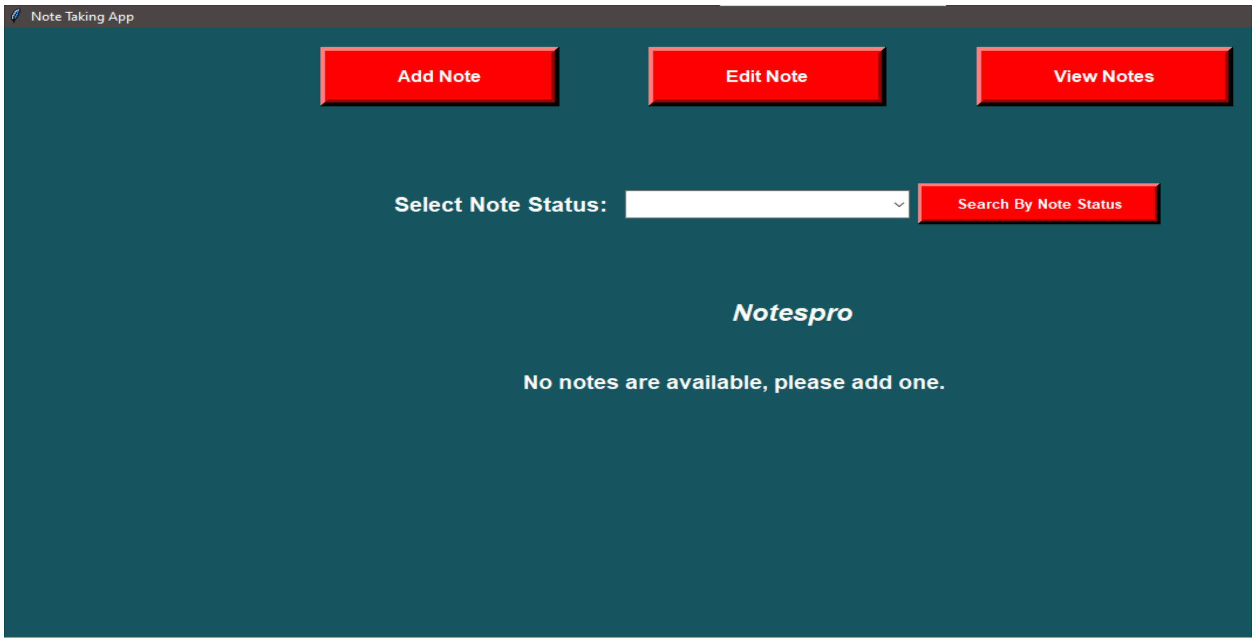
Expected Result:

The note should be deleted from the database.

A success message "Note deleted successfully." should be displayed.

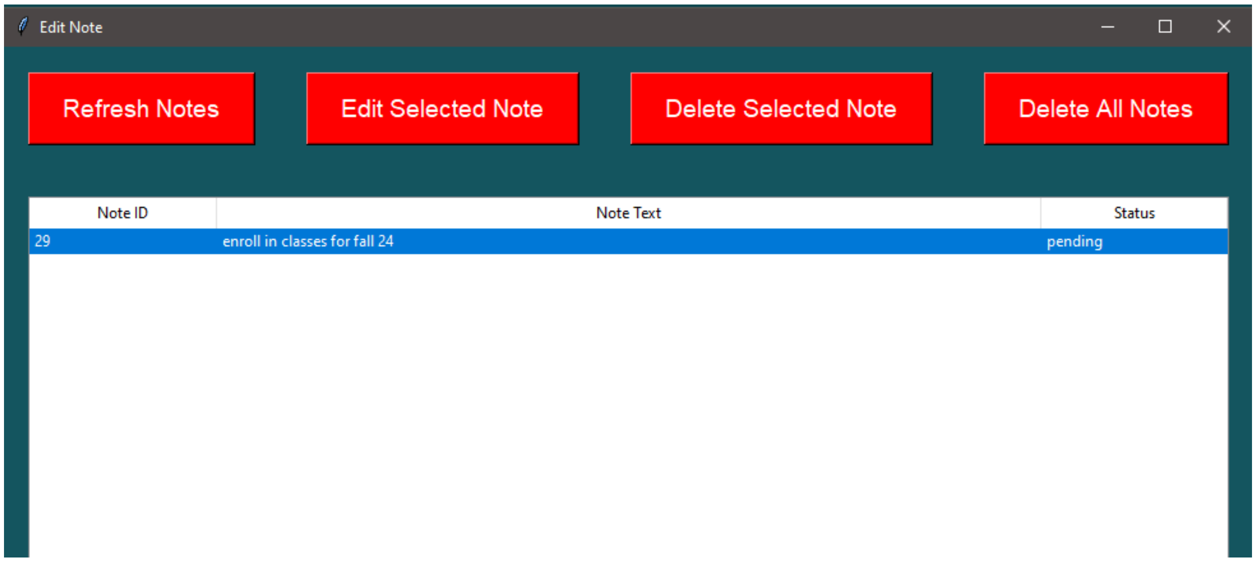
The Treeview should be updated to reflect the deletion.

The visualization chart should be updated to reflect the deletion.



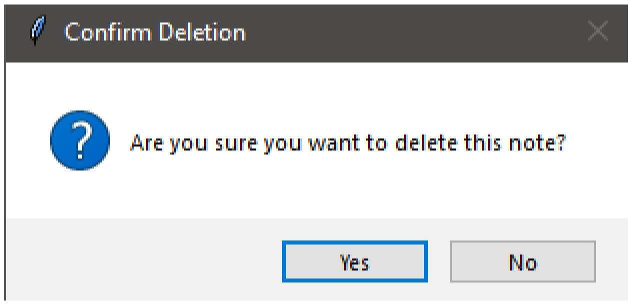
No notes are available, please add one.

Fig:On clicking Edit Note launches the below window where a selected can be edited.



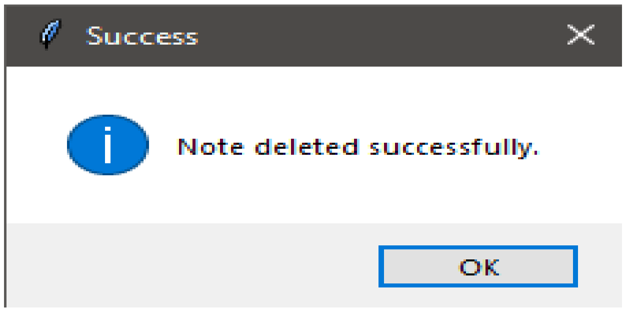
Note ID	Note Text	Status
29	enroll in classes for fall 24	pending

Fig:Note to be deleted is selected and clicked on *‘Edit selected Note’* button


Confirm Deletion

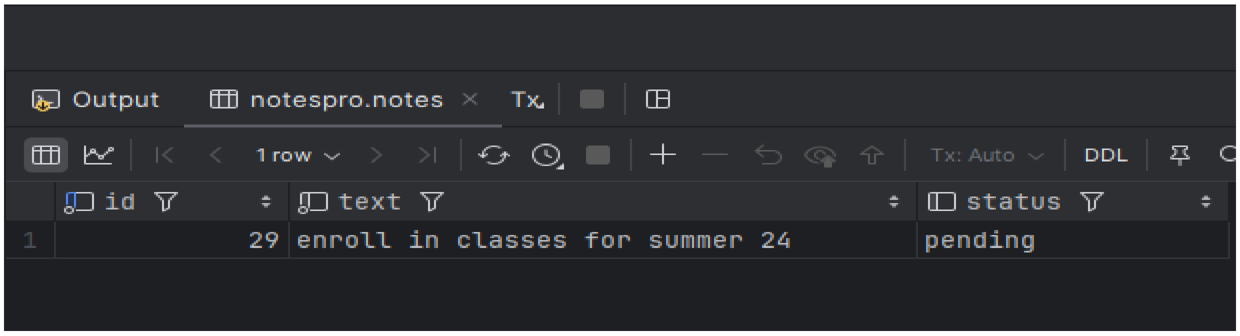
Are you sure you want to delete this note?

On clicking ‘yes’ the following displayed:


Success

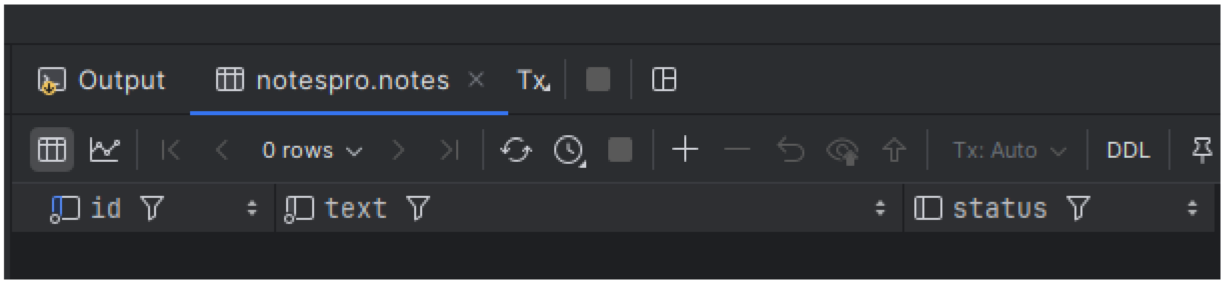
Note deleted successfully.

Fig: A success message *"Note deleted successfully."* displayed *after deleting selected note*



	id	text	status
1	29	enroll in classes for summer 24	pending

Fig: *Before deletion of selected note*, the notes table in the database is as above



id	text	status

Fig: The note deleted from the *notes* table in the database ‘notespro’.

Test Case 2: Click on *Delete Selected Note* without selecting a note (error case)

Description: Verify that an existing note can be deleted successfully.

Steps:

Launch the application.

Click the "Edit Note" button.

Select an existing note from the Treeview.

Click the "Delete Selected Note" button.

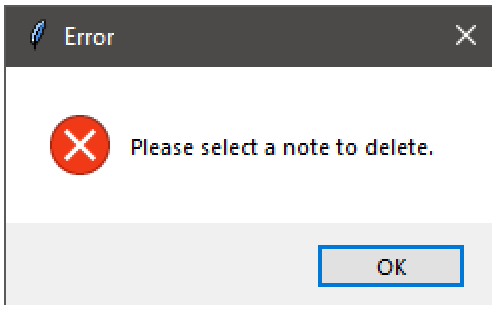
Confirm the deletion in the confirmation dialog.

Expected Result:

An error message "Please select a note to delete." should be displayed.

Result Obtained:

When a user clicks on 'Delete Note' button without selecting a note to be deleted,the following message will be displayed to the user:


Error

Please select a note to delete.

Fig: An error message "Please select a note to delete." should be displayed.

Functional Test case of '*Delete All Notes*' component:

Test Case 1: Delete All Notes(when data is available)

Description: Make sure the table is not empty and see that all notes can be deleted successfully when clicked on *'Delete All Notes'* button.

Steps:

Launch the application.

Click the *"Edit Note"* button.

Click the *"Delete All Notes"* button.

Confirm the deletion in the confirmation dialog.

Expected Result:

All notes should be deleted from the database.

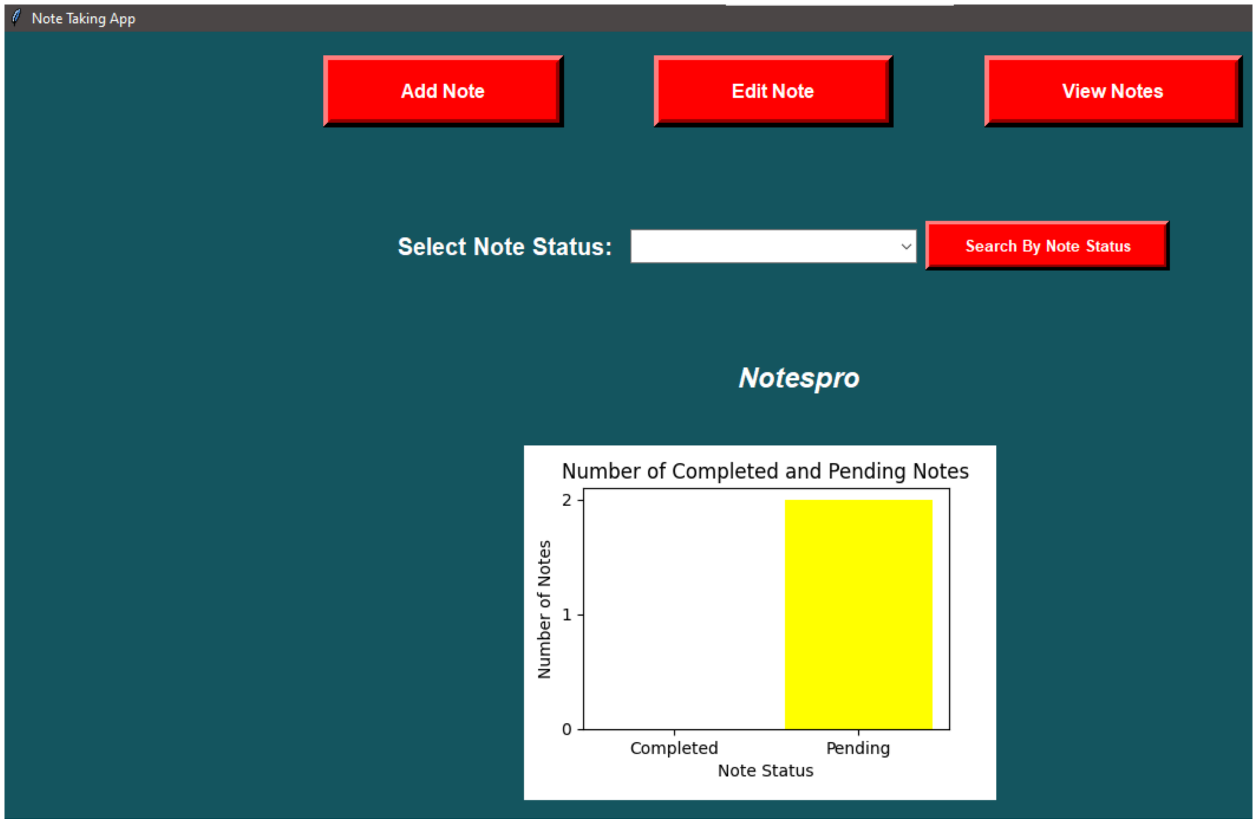
A success message *"All notes deleted successfully."* should be displayed.

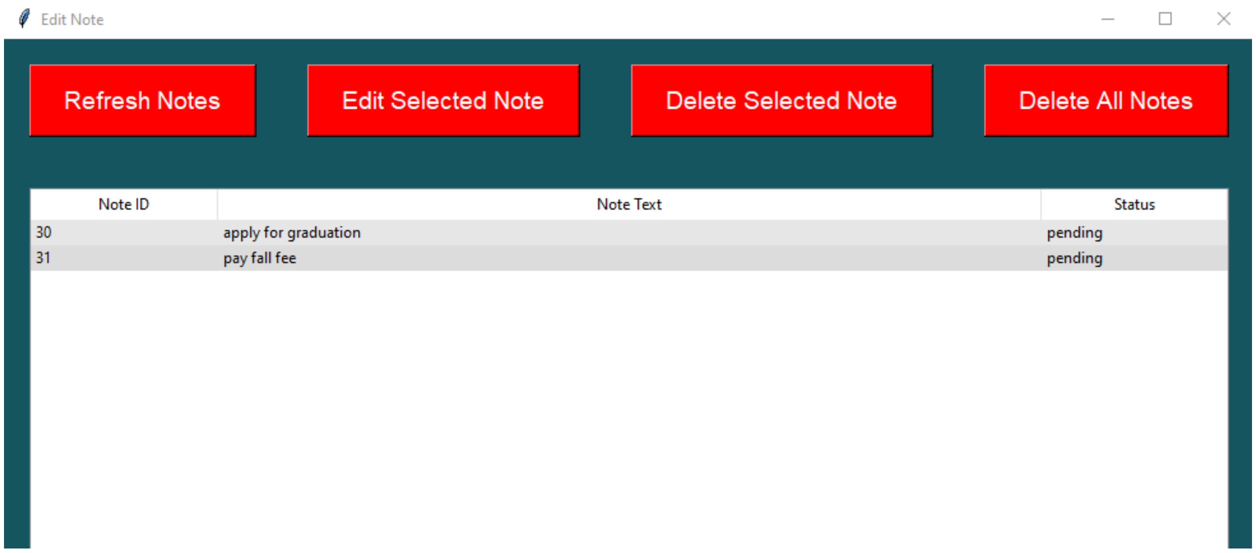
The Treeview should be updated to reflect the deletion.

The visualization chart should be updated to reflect the deletion.

Result Obtained:

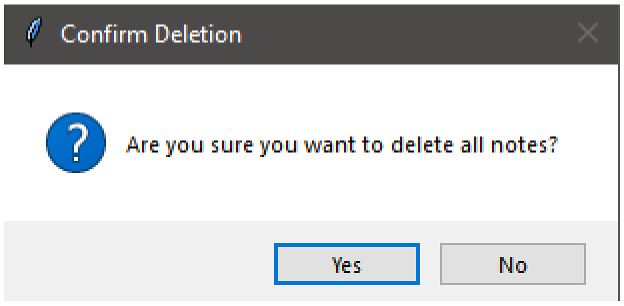
The table is as shown as below after clicking on *'Edit Note'* component:




Edit Note

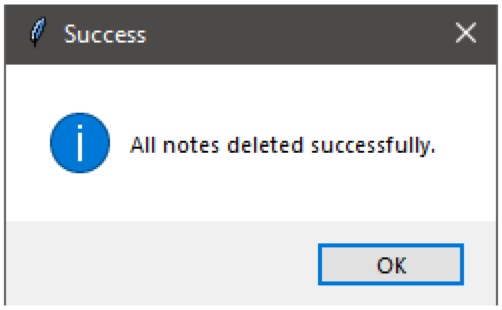
Note ID	Note Text	Status
30	apply for graduation	pending
31	pay fall fee	pending

When a user clicks on ‘Delete All’ button, confirmation query message will be given to user as follows:


Confirm Deletion

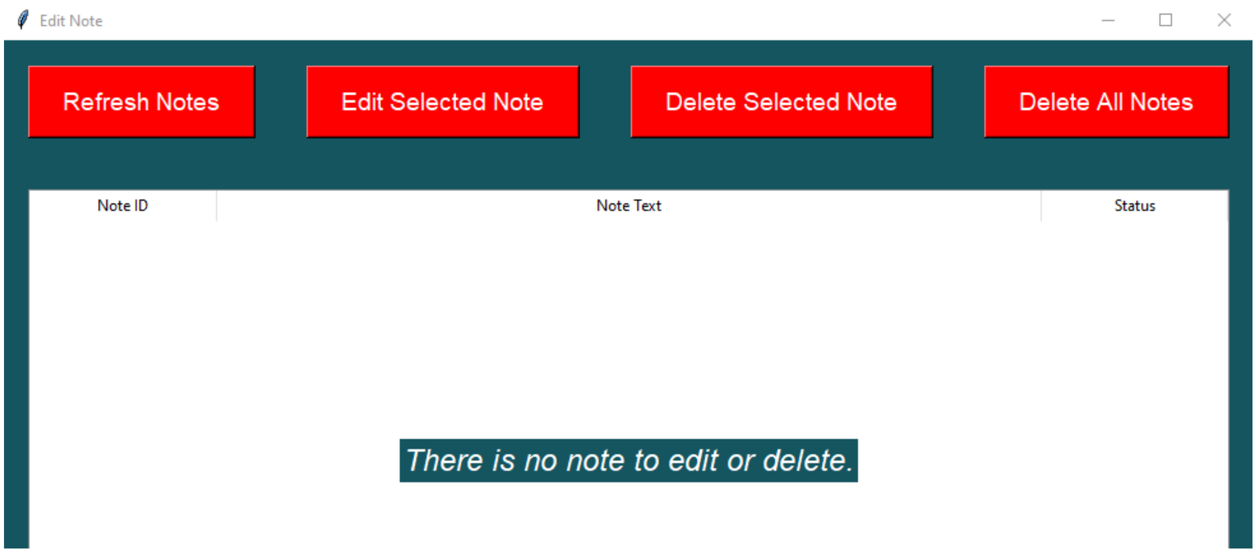
Are you sure you want to delete all notes?

After clicking ‘Yes’, the following message will be displayed:


Success

All notes deleted successfully.

On verification found that all notes are deleted and the table is empty as shown below:



There is no note to edit or delete.

When 'No' is chosen by the user in the above confirmation query, no message like above will be displayed and the confirmation query message box exits.

Functional Test Cases of *Search By Note Status* :

Test Case 1: Search Notes by Status (when data is available)

Description: Verify that notes can be searched by status.

Steps:

Launch the application.

Select *"Pending"* from the status dropdown menu.

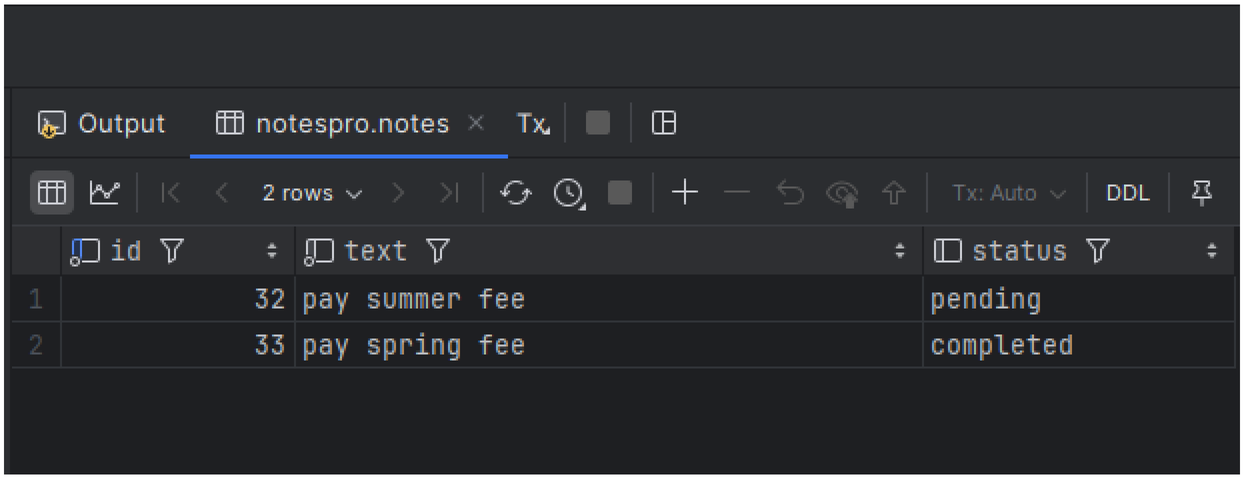
Click the *"Search By Note Status"* button.

Expected Result:

A new window should open displaying the notes with the status *"Pending"*.

The notes should be displayed in a Treeview widget.

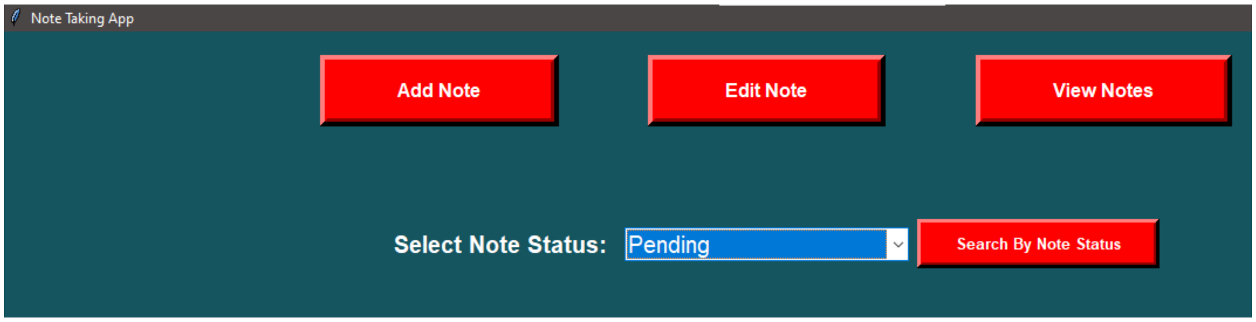
Current status:



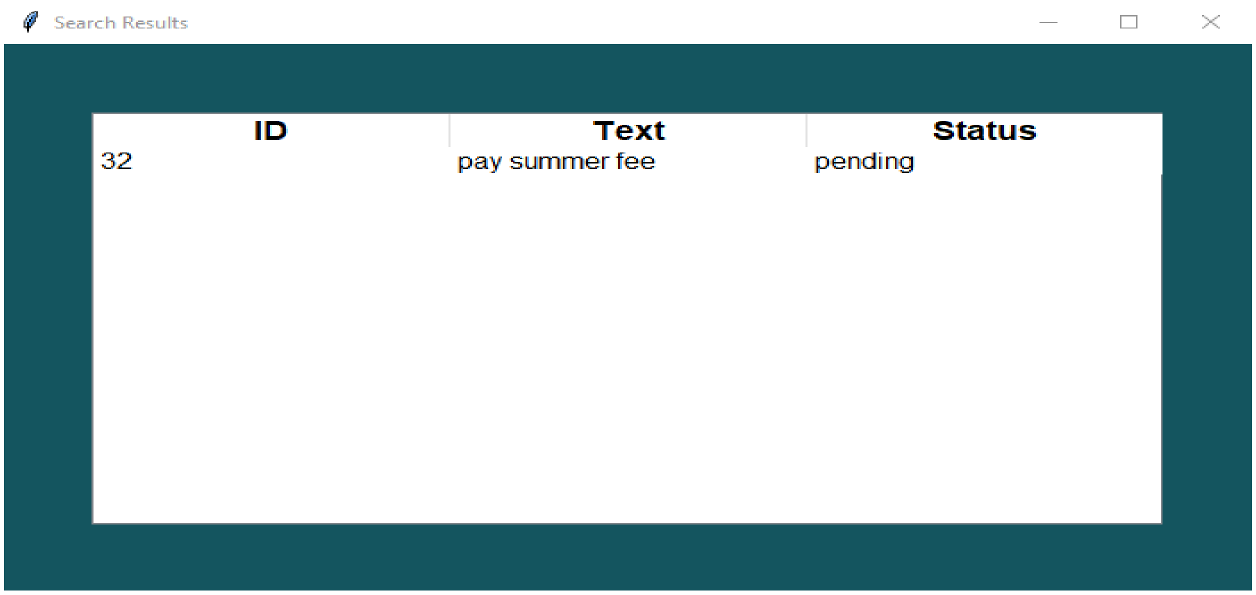
id	text	status
32	pay summer fee	pending
33	pay spring fee	completed

We have one pending and one completed note as shown above.

Result obtained:



Select Note Status: Pending



ID	Text	Status
32	pay summer fee	pending

Fig: After selecting ‘pending’ note status, list of pending notes are displayed.

Test Case 2: Search Notes by Status as *'completed'*

Description: Verification of search by selecting note status as 'completed'

Steps:

Launch the application.

Select "*Completed*" from the status dropdown menu.

Click the *"Search By Note Status"* button.

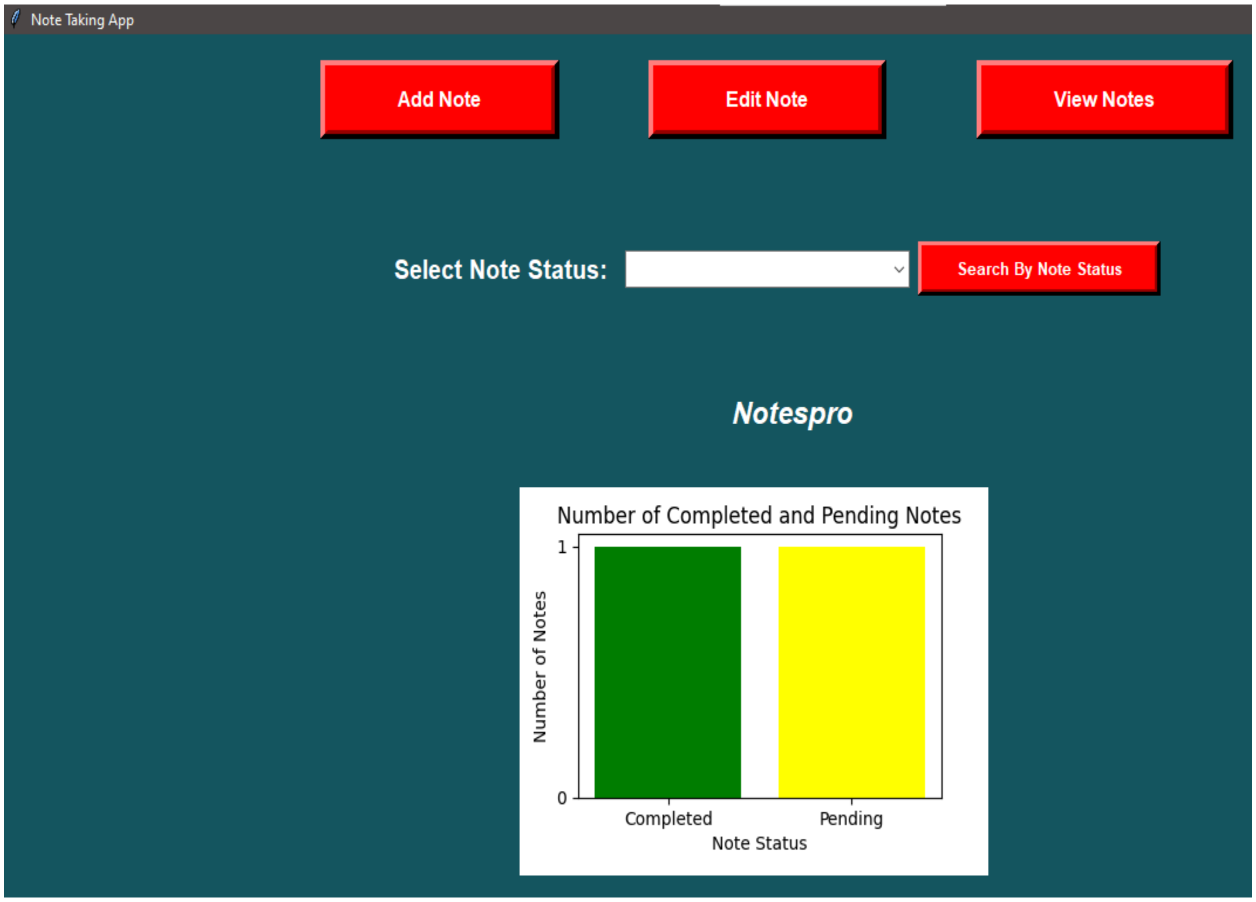
Expected Result:

A new window should open displaying the notes with the status *"Pending"*.

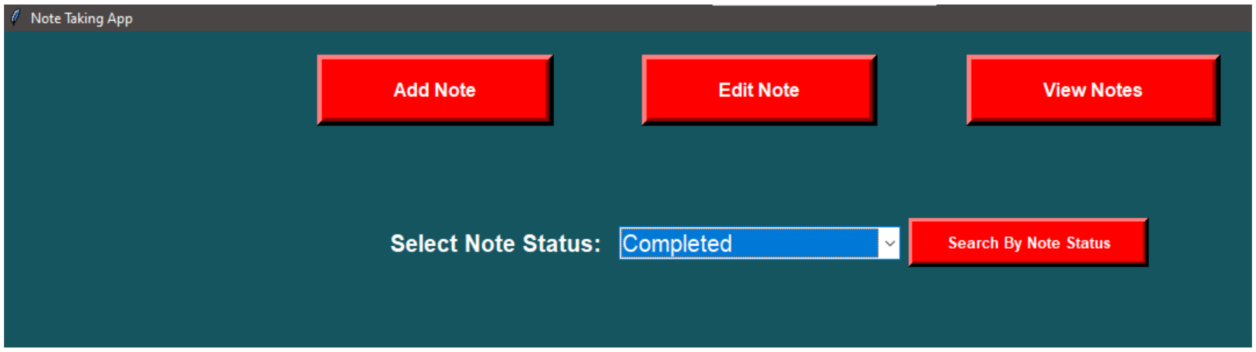
The notes should be displayed in a Treeview widget.

Current status:

We have one pending and completed note as shown in the chart below:

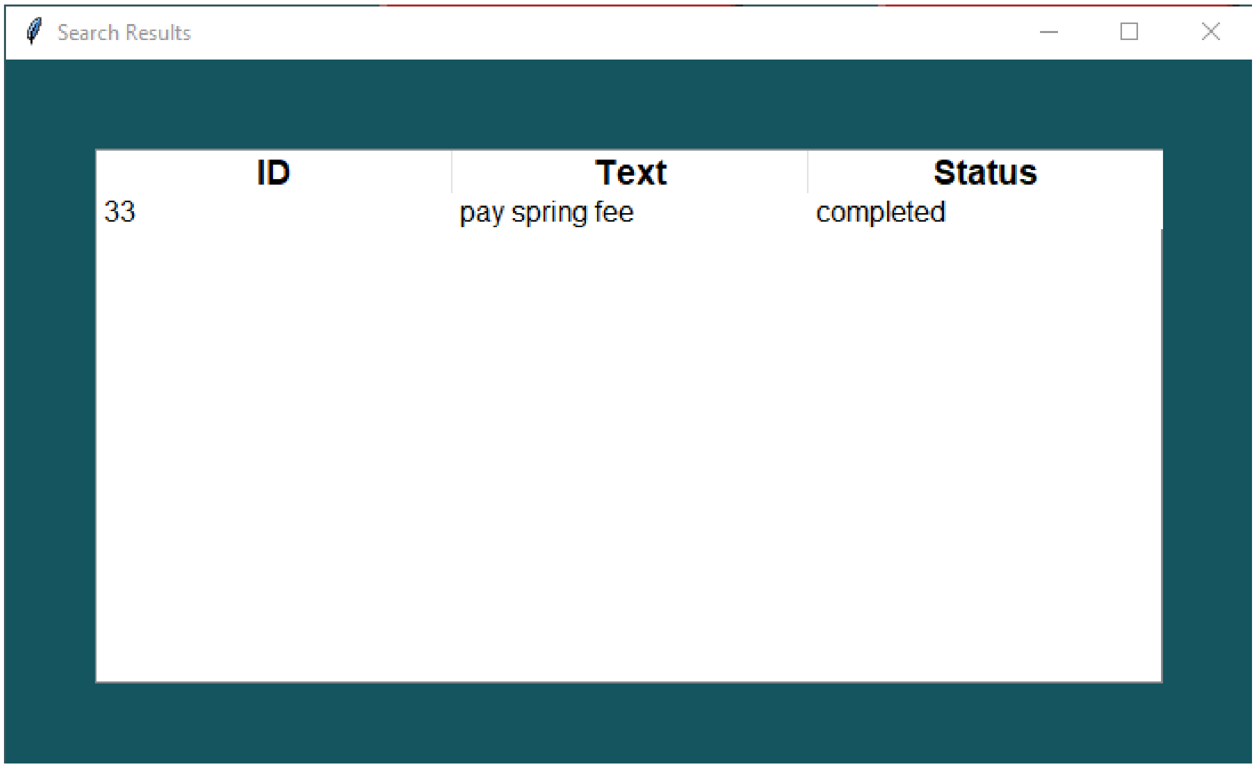


Result obtained:



Select Note Status: Completed

Fig: Selected note status *‘completed’*



ID	Text	Status
33	pay spring fee	completed

Fig: search resulted in a table with notes that are yet to be completed.

Test Case 3: Search Notes by Status without selecting a status (*error case*)

Description: Verify that notes can be searched by status.

Steps:

Launch the application.

The status is not selected in the dropdown menu.

Click the "Search By Note Status" button.

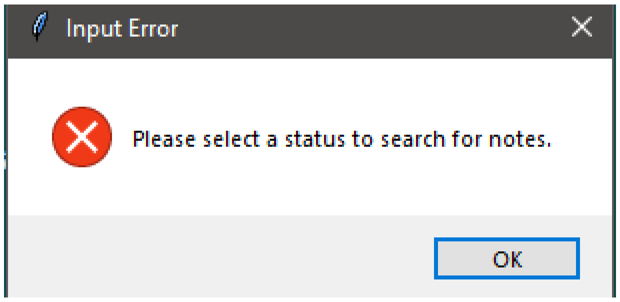
Expected Result:

A new window should open displaying the notes with the status "Pending".

The notes should be displayed in a Treeview widget.

Result obtained:

When a user clicks on ‘Search Status’ button without selecting status (completed/pending), the following error message box will appear:


Input Error

Please select a status to search for notes.

Test Case 4: When note is not available for the selected status and chosen to *Search By Note Status.*

Description: Verify that notes can be searched by status by selecting status.

Steps:

Launch the application.

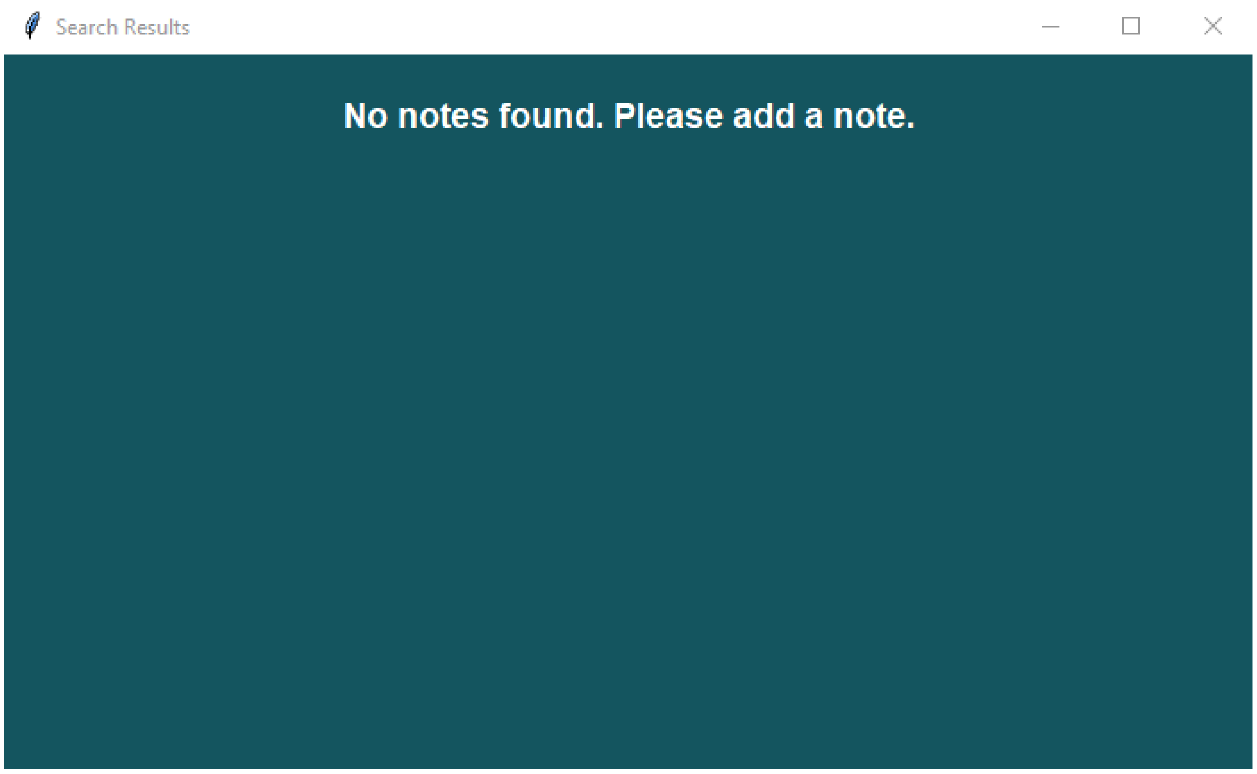
Select *"Pending"* from the status dropdown menu.

Click the *"Search By Note Status"* button.

Expected Result:

A new window should open displaying the Treeview widget with a message at its center *‘No notes found. Please add one’*.

If user selects a ‘status’ from the combobox/drop down box and then clicks on ‘Search By Note Status’ button,then *the following window will appear* ***when no note is available for the status selected.***



No notes found. Please add a note.

Fig:The message in the window displays only when ‘Notes are not available to show’.

Functional Test cases of ‘*View All Notes*’ component:

Test Case 1: View All Notes(when data is available)

Description: To verify that the "View All Notes" window is created with the correct title, size, background color, and contains a frame for the table with the correct columns and headings and the notes are fetched and displayed correctly in the Treeview when data is available.

.**Setup**: Ensure the MySQL database is running and accessible.

Steps:

Launch the application.

Click the "View Notes" button.

Verify that a new window titled "View All Notes" is created.

Verify that the window size is 700x400.

Verify that the window background color is #145660.

Verify that the window contains a frame with the correct border and relief properties.

Verify that the Treeview has columns "ID", "Note", and "Status".

Verify that the column headings are correctly set to "ID", "Note", and "Status".

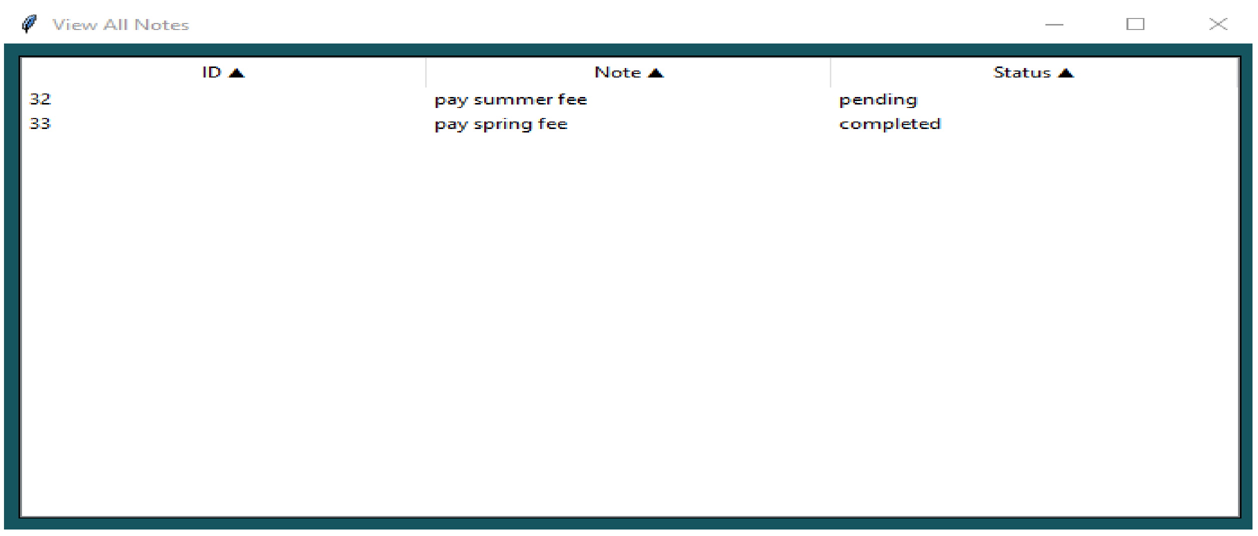
Verify that the notes are fetched and displayed correctly in the Treeview with the correct values.

Expected Result:

A new window should open displaying all the notes in the database. The window should be created with the specified title, size, background color, and contain the frame with the correct properties and the Treeview table is created with the correct columns and headings.

The notes should be displayed in a Treeview widget with sorting options by column headers.

Result Obtained:


View All Notes

ID ▲	Note ▲	Status ▲
32	pay summer fee	pending
33	pay spring fee	completed

Fig: View all notes window with a treeview widget displaying notes

Test Case 2: Verification of sorting functionality in *View Notes* Component

Objective: Verify that the sorting functionality works correctly for each column in the Treeview.

Setup: Populate the notes table in the MySQL database with sample data.

Test Steps:

Click on *View Notes* button

Click on the "Note" column heading to sort the notes by the "Note" column.

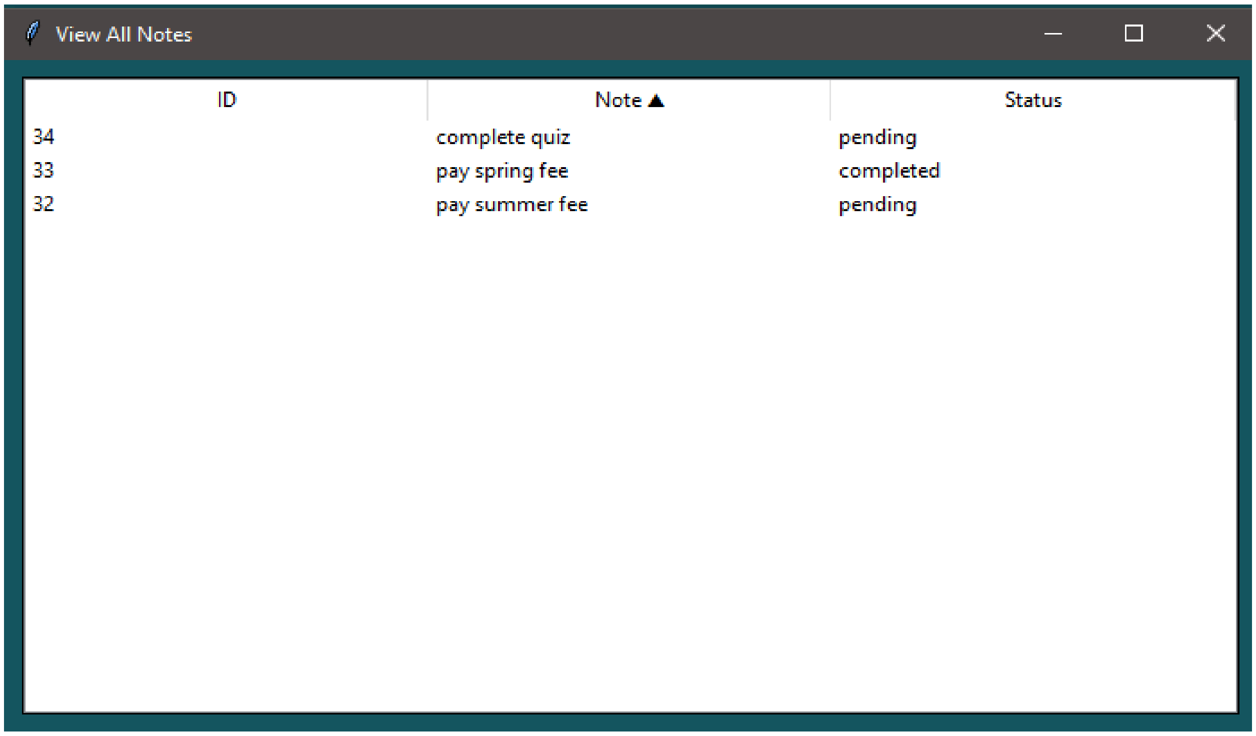
Verify that the notes are sorted in ascending order by the "Note" column.

Click on the "Note" column heading again to sort the notes in descending order.

Verify that the notes are sorted in descending order by the "Note" column.

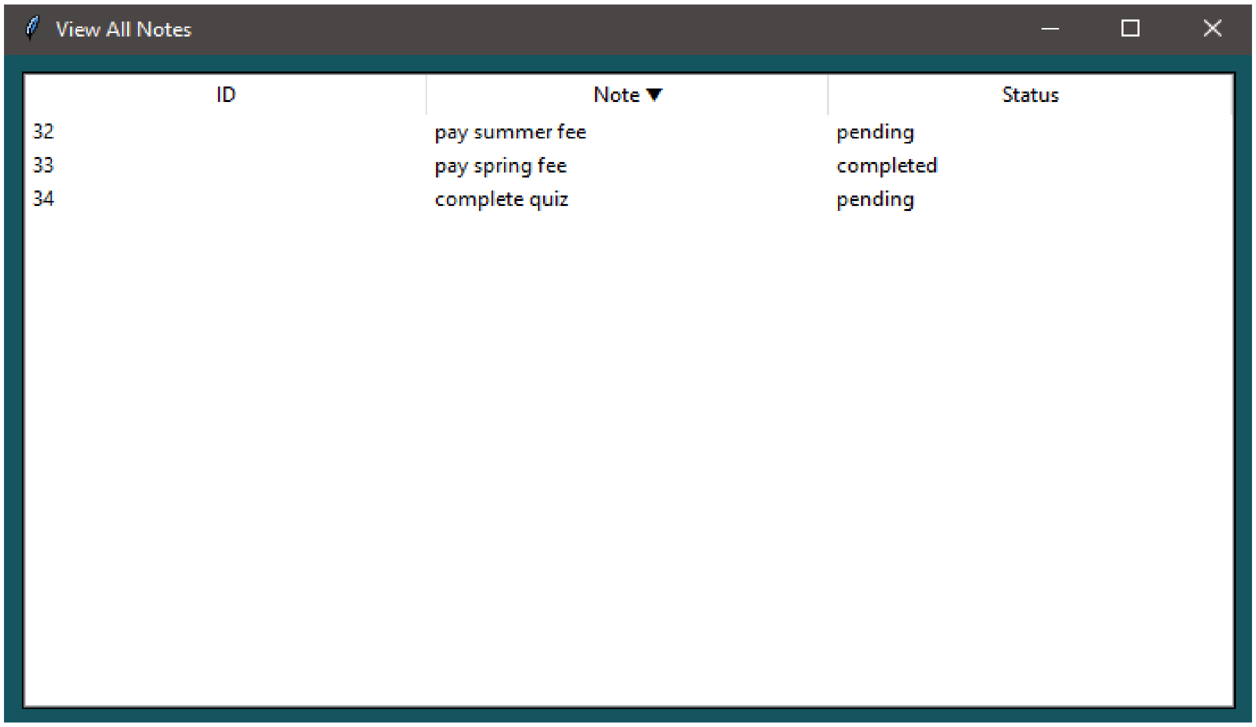
Repeat steps 2-5 for the "Status" and "ID" column.

Expected Result: The notes should be sorted correctly in ascending and descending order for each column when the column heading is clicked.



ID	Note ▲	Status
34	complete quiz	pending
33	pay spring fee	completed
32	pay summer fee	pending

Fig:Notes column sorted in ascending order of text in 'Note' column

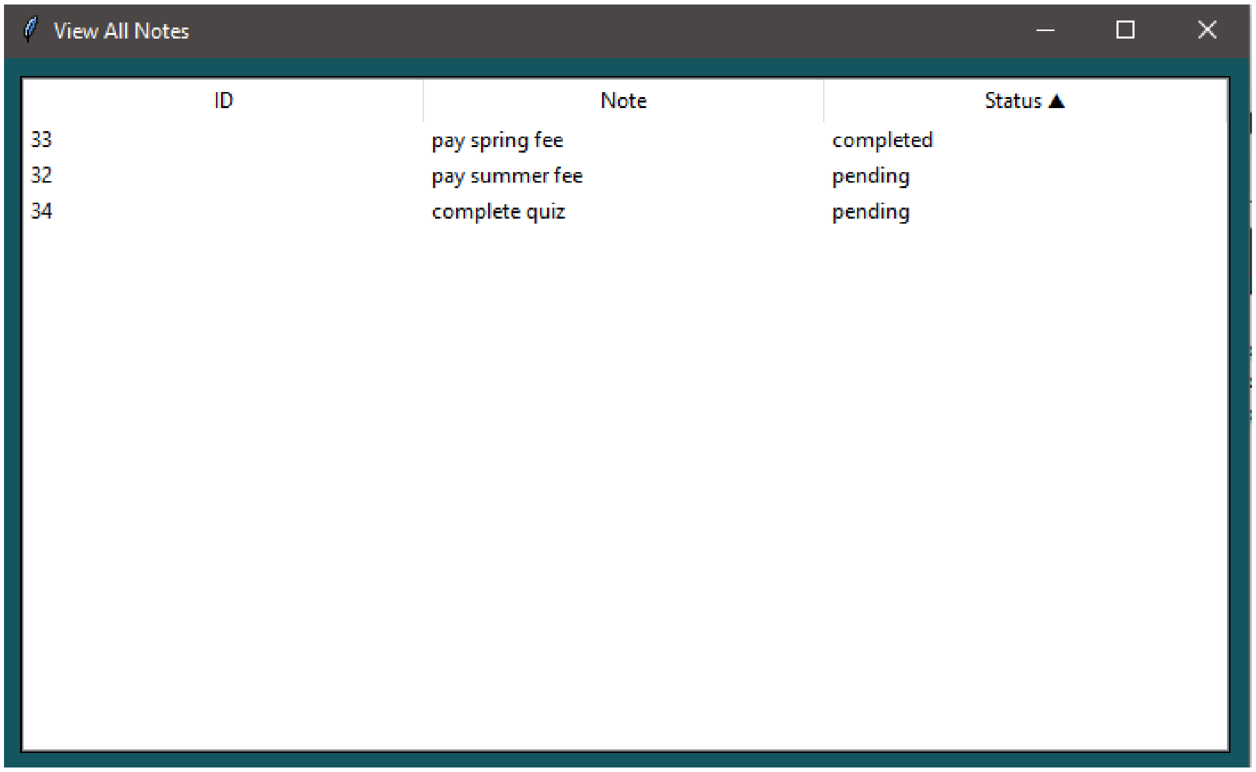


View All Notes

ID	Note ▼	Status
32	pay summer fee	pending
33	pay spring fee	completed
34	complete quiz	pending

Fig: Text in the Note column sorted descendingly as shown below:

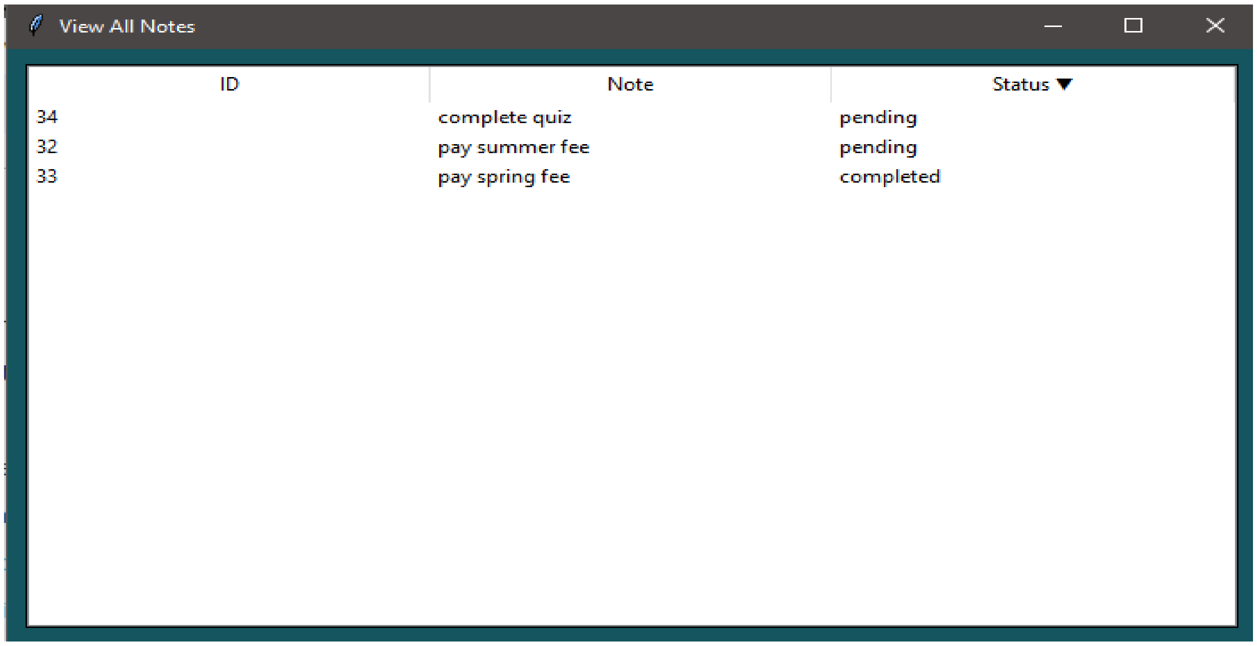
Status column sorted ascendingly:



View All Notes

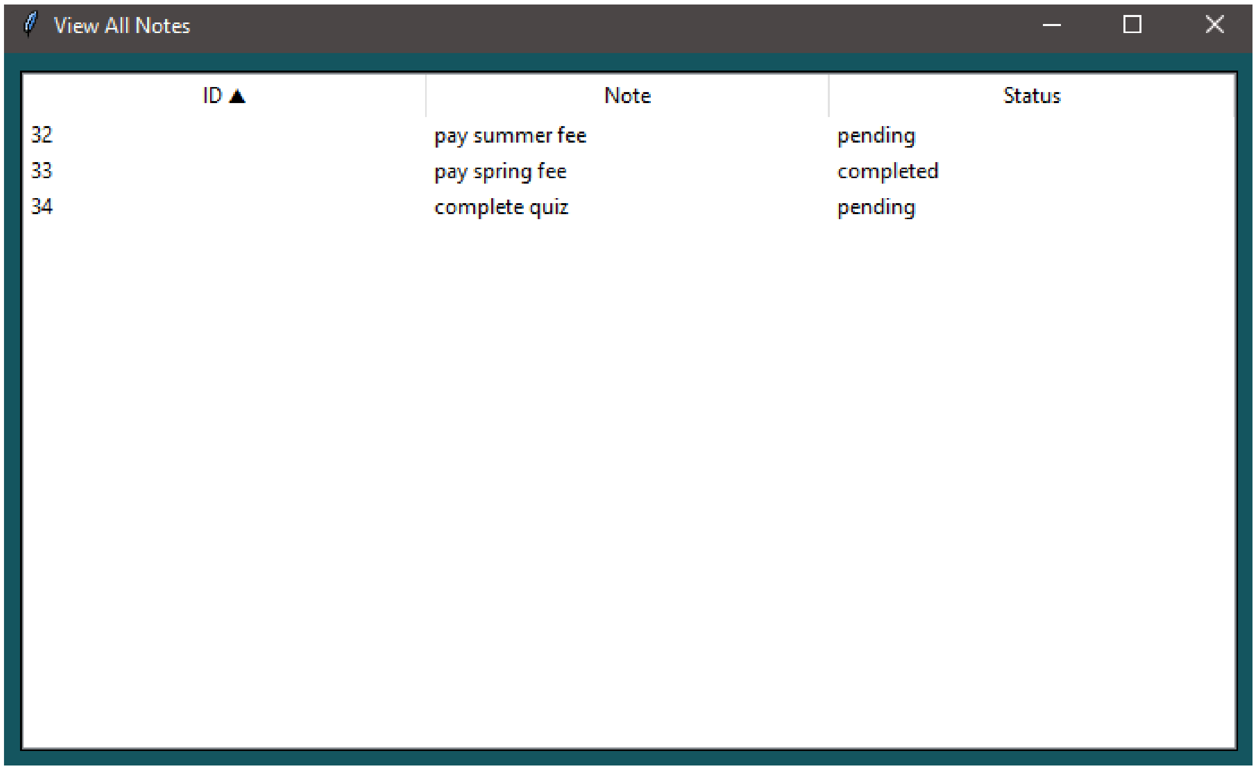
ID	Note	Status ▲
33	pay spring fee	completed
32	pay summer fee	pending
34	complete quiz	pending

Status column sorted descendingly:



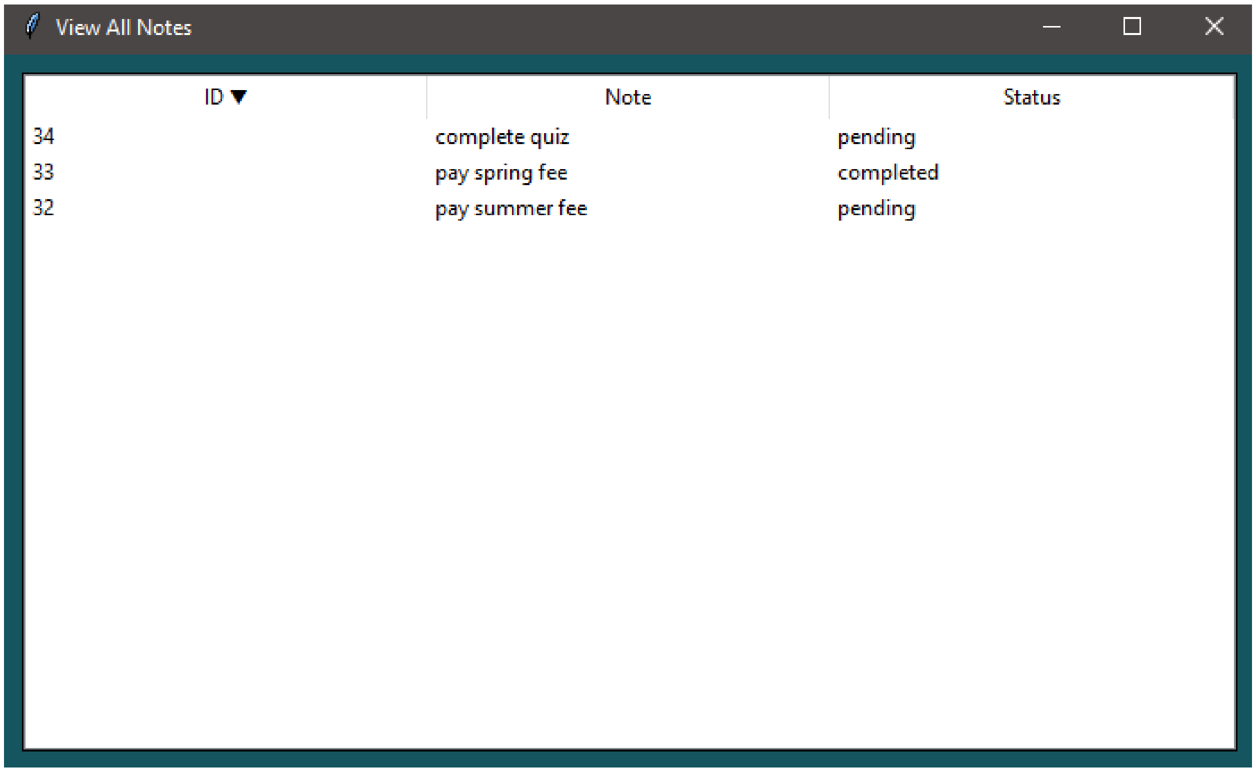
ID	Note	Status ▼
34	complete quiz	pending
32	pay summer fee	pending
33	pay spring fee	completed

ID column sorted ascendingly:



ID ▲	Note	Status
32	pay summer fee	pending
33	pay spring fee	completed
34	complete quiz	pending

ID column sorted descendingly:



ID ▼	Note	Status
34	complete quiz	pending
33	pay spring fee	completed
32	pay summer fee	pending

Test Case 3: Verification of fetching and displaying Notes in *View All Notes*(No Data Available)

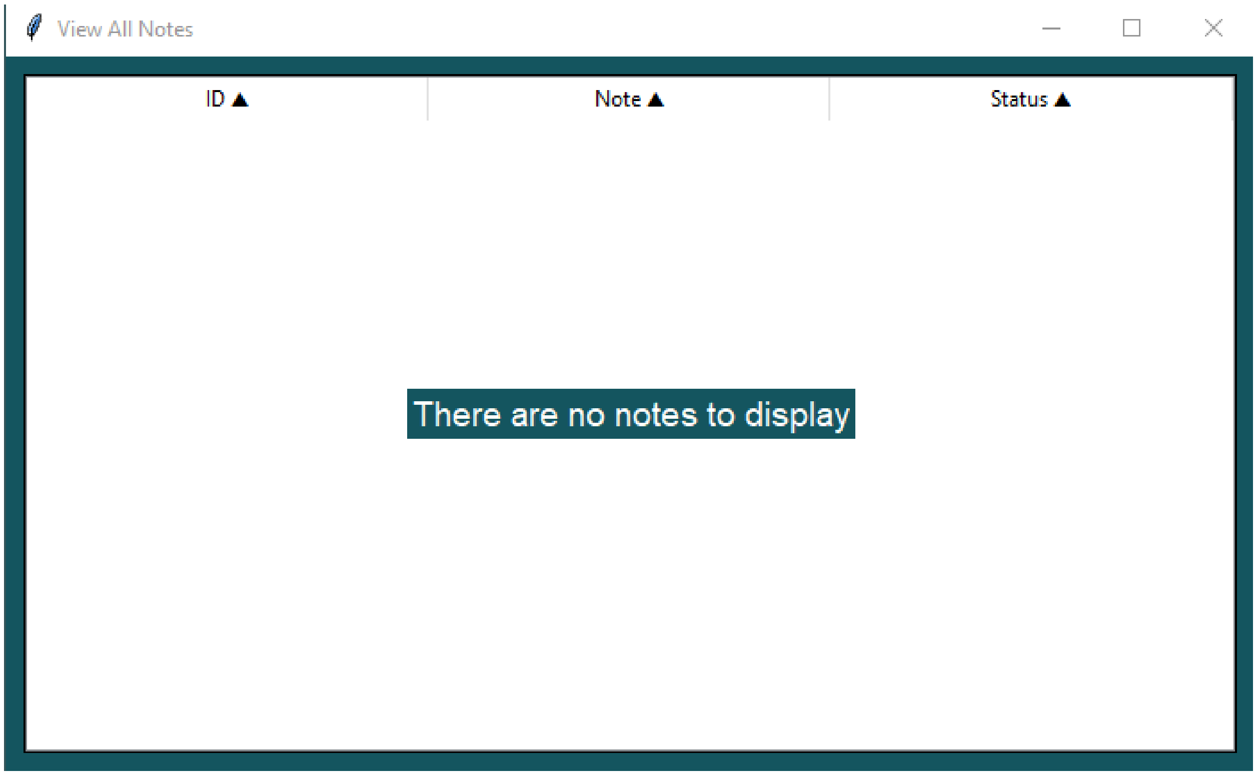
Objective: Verify that a message is displayed in the center of the window when no notes are available in the database.

Setup: Ensure the notes table in the MySQL database is empty.

Steps:
Click on *View All Notes* function.
Verify that the Treeview is cleared of any existing data.
Verify that a centered message "There are no notes to display" is displayed in the frame.

Expected Result: The message "There are no notes to display" should be centered in the frame when no notes are available in the database.

Result Obtained:



There are no notes to display

Fig: View All Notes displays list of notes if available otherwise a message a shown above

Functional Test Cases of Visualization feature:

Test Case 1: Visualization Update

Description: Verify that the visualization chart is updated correctly after

a)adding,

b)editing, or

c)deleting selected note and

d)deleting all notes at one go..

Steps:

Launch the application.

Perform actions to

a) add,

b)edit, and

c)delete selected note and

d) deleting all notes at one go.

Observed the visualization chart after each action

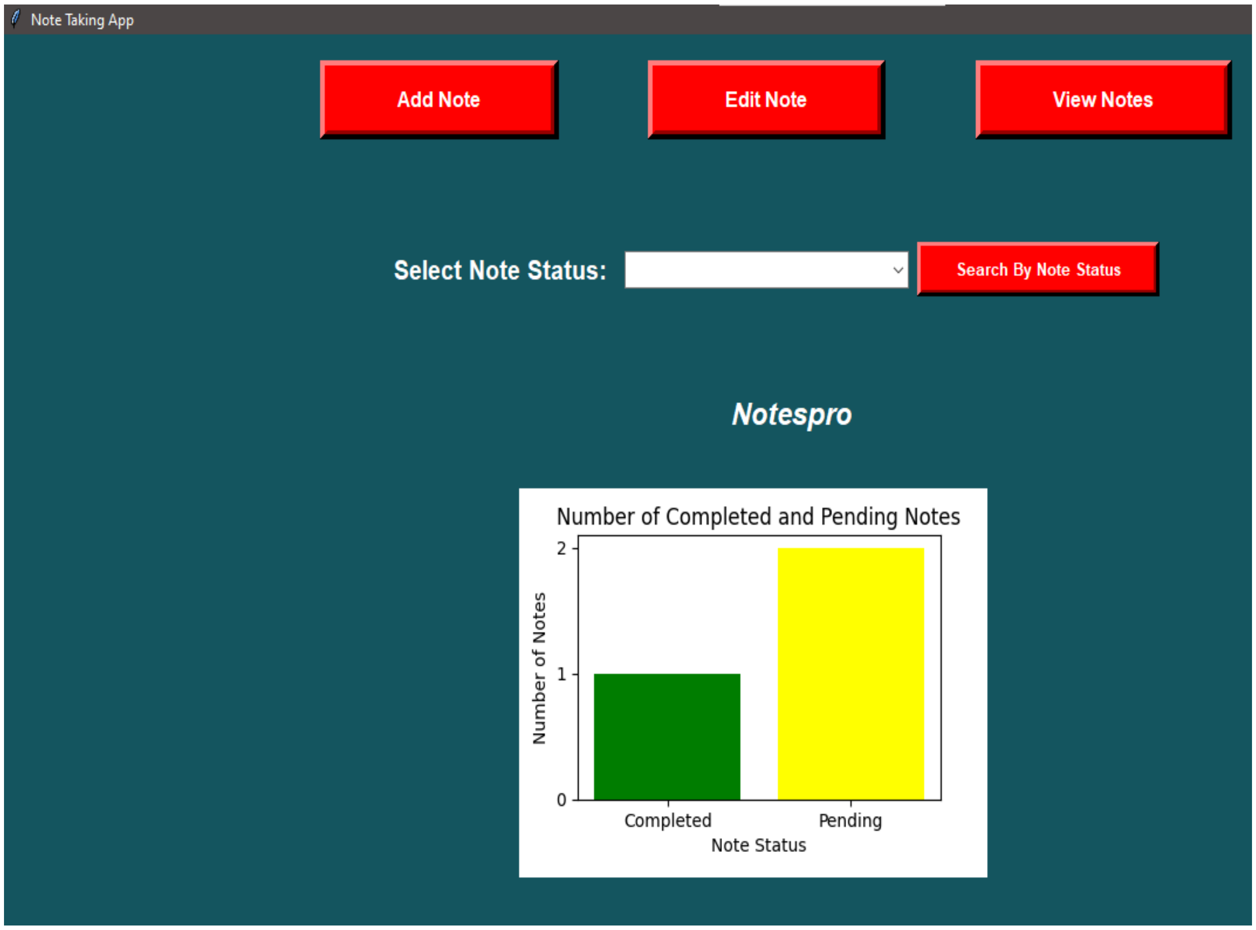
Expected Result:

The visualization chart should reflect the current state of the notes in the database.

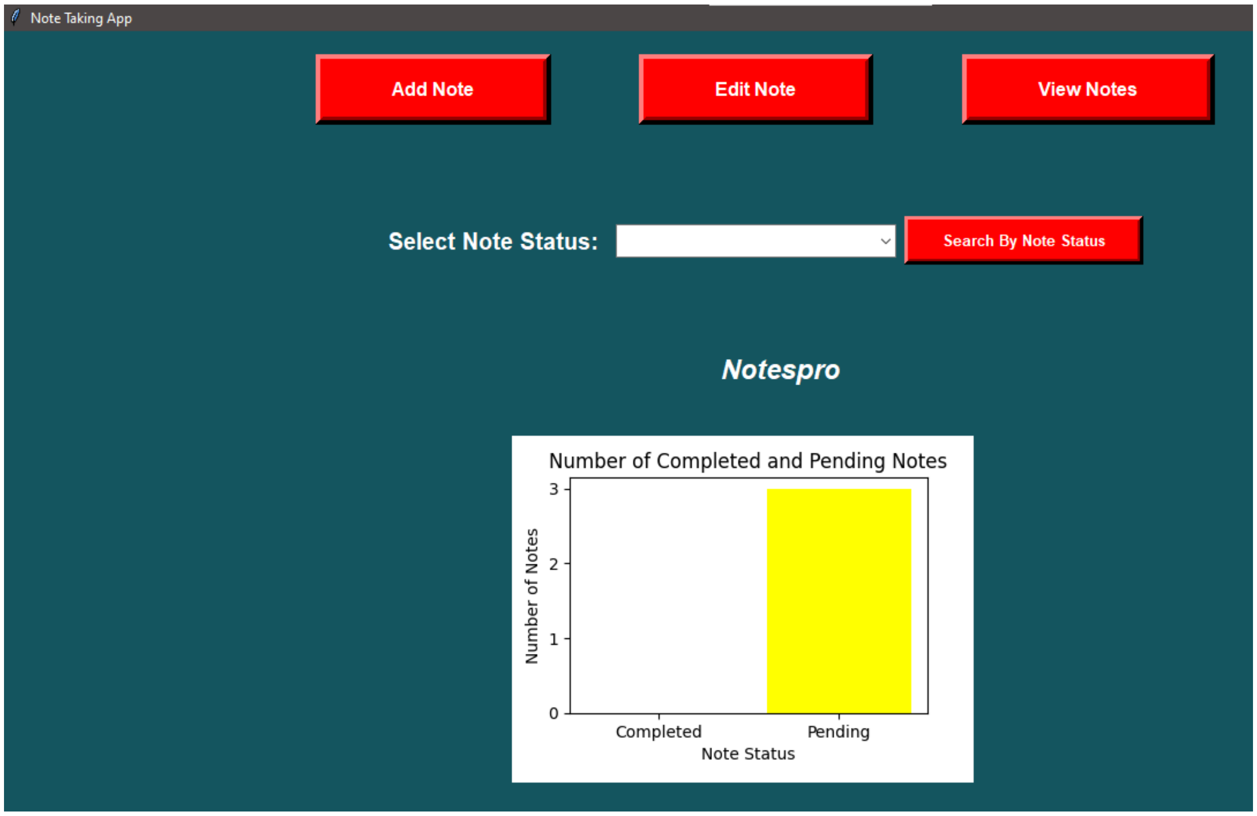
Current status:

Counts for "Completed" and "Pending" notes should be accurate.

a)***After adding* 3 notes** in which 2 are pending and one is a completed note:



b)After editing a note, from completed to pending,the chart gets updated as follows:

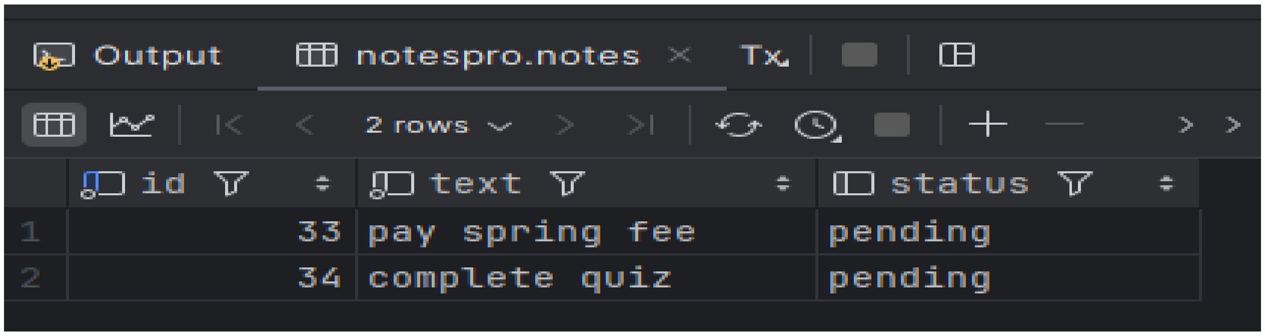


Current state:

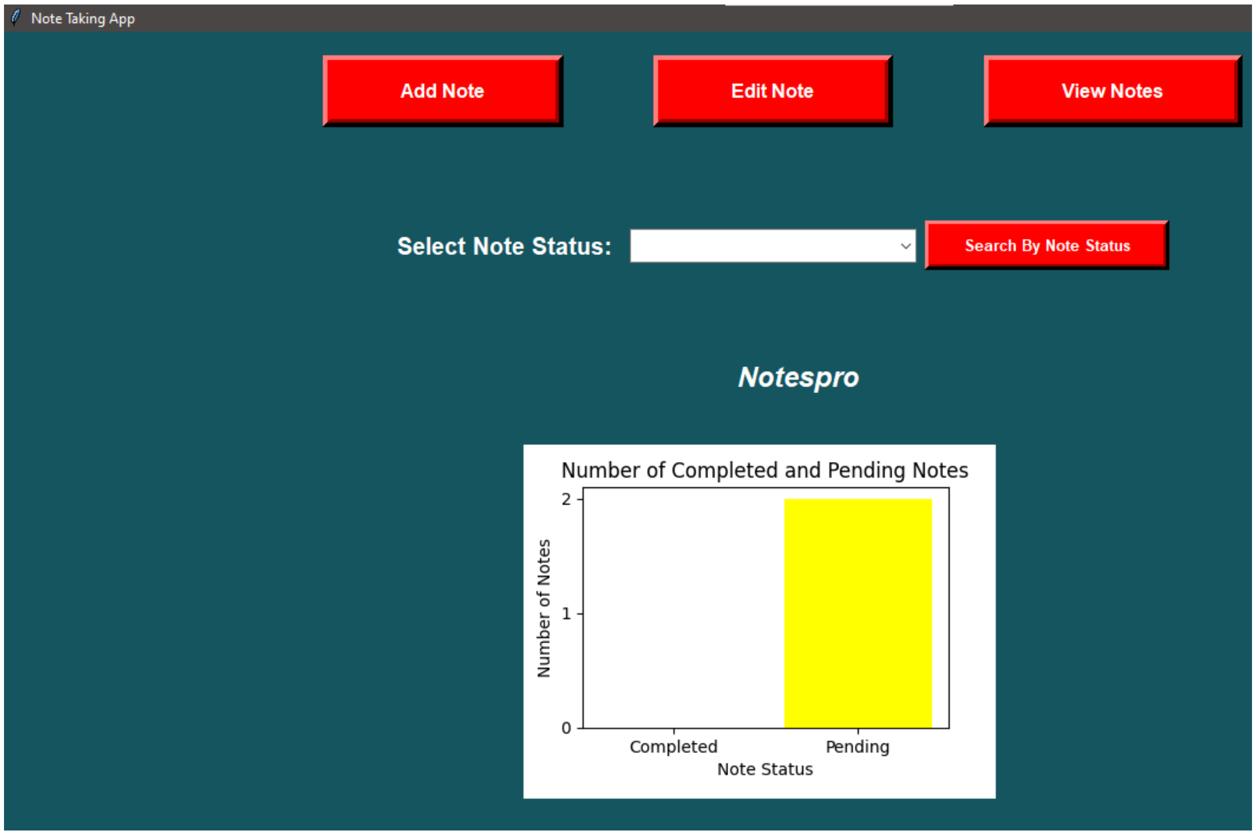
Now all 3 are pending notes.

c)***after deleting a note*** clicking on 'DeleteSelected Note' only 2 pending notes should be available and the chart should be updated accordingly as shown below:

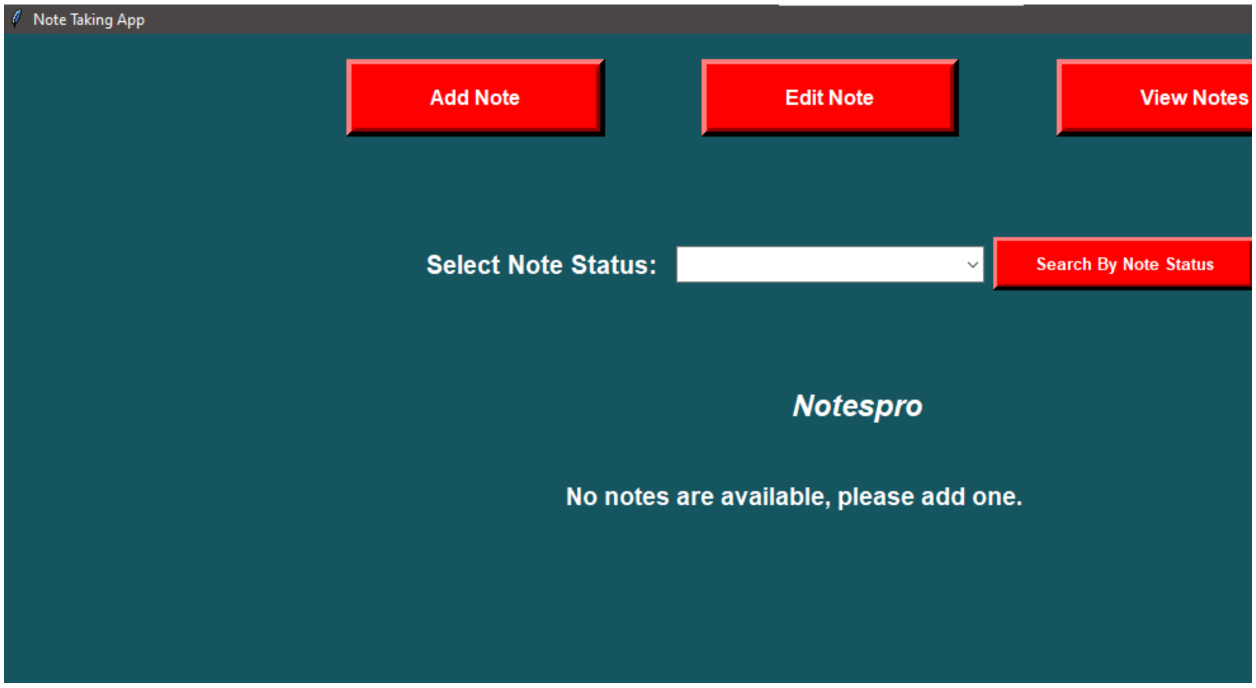
The note 'pay summer fee is deleted' is a pending note and is deleted as shown below from table 'notes' in database:



	id	text	status
1	33	pay spring fee	pending
2	34	complete quiz	pending



d)After deleting all notes, visualization gets updated as follows:



No notes are available, please add one.

Therefore, the above test cases covered the core functionalities of the note-taking application, ensuring that notes can be added, edited, deleted, searched, and viewed correctly, and that the visualization is updated accordingly.

Implementation Plan:

Milestones:

- **UI Design and Basic Functionality**
  - **Deliverables:** Initial UI mockups, basic note addition functionality.
  - **Timeline:** June 5, 2024 - June 30, 2024
- **Database Integration and CRUD Operations**
  - **Deliverables:** Database schema, CRUD operations for notes.
  - **Timeline:** July 1, 2024 - July 7, 2024
- **Testing and Bug Fixing**
  - **Deliverables:** Test cases, bug fixes.
  - **Timeline:** July 8, 2024 - July 14, 2024
- **Deployment and User Acceptance**
  - **Deliverables:** Final application deployment, Functional Testing results.
  - **Timeline:** July 15, 2024 - July 21, 2024

Responsibilities:

- Overall coordination, timeline adherence.
- UI mockups, user flow.
- Database design, schema implementation.
- Function implementation, integration.
- Test case creation, execution.

Conclusion:

The Note taking application aims to streamline note management for users, offering a robust interface with essential functionalities. This design document outlines the architecture, functionalities, testing strategy, and implementation plan necessary for successful development and deployment of Notespro.

—---THE END—---