Computer Science Project

on

Library Management System

Debayan Sutradhar

12 M

DPS Ruby Park, Kolkata

Certification

This is to certify that the project entitled:

“SnakeBrary - A Library Management System”

is the original work of Debayan Sutradhar, 12 M (2020/14275), DPS Ruby Park School, Kolkata. This system was developed as CBSE Class 12 Project.

_____ _____

Internal Examiner External Examiner

Acknowledgement

I would like to thank my parents for funding my education and providing computer to develop this project on and also internet services. I am also grateful to the developers of the Python programming language, and the respective owners/developers of all the 3rd party services and libraries I have used for this project.

Contents

Objective

The goal of the system is to be a simple, free, open-source, cross-platform, user-friendly, small, light-weight and scalable Library Management System. The source code for this program is available on GitHub, and this system is licensed under the GNU GPLv3 License, which allows others to improve and even make their own custom versions.

- A user login system that makes it easier to organise and administrate the system.
- A powerful search system to search for a book by its name, author name, ISBN or even genre.
- Automates and digitises the entire process of issuing/returning books and records all information digitially in a database.
- Rate already read books, that helps others to know general idea about the book.
- Lightweight system that can be run on cheap and low end hardware rather than a full computer system, hence reducing costs.
- Free and Opensource alternative to existing library management system.
- Cross platform alternative that can be run on a variety of hardware and Operating Systems.
- Easy to use and intuitive user system while maintaining the same amount of features as a regular library management system.

The system is based on the principal of users. Users are further divided into 3 different types:

1. Normal: This user is the most basic type of user and is consists of the average Library user. They can only issue, return, rate and search for books. They can also edit their own user information.
2. Administrator: This user can create, edit or delete users and books. They cannot delete the ‘Master’ or other Administrators. They can also do anything that a ‘Normal’ user can do.
3. Master: This user is created during the intial setup of the software. There can only be one master user. This type of user can create, edit or delete any user or book from the system. They can also reset the entire software, which will delete the System database from the SQL Server.

Features

1. User Account System: The system works on the main principal of a user-account system. This makes it easier to maintain the system, especially for bigger environments. Accounts can also be disabled and certain books can also be made available by the administrator or the master user.
2. Book rating System: Each book can be scored out of 5 points by any user who has read it. This helps other users know about how the book is generally perceived by the audience.
3. Powerful search interface: The user can search for a book by its name, author, ISBN or even its genre. An administrator/master user can also search for a user by their username or name.
4. Modern and clean UI: This system uses PySide2 for GUI, and adheres to Google's Material Design Language, which makes it far cleaner and modern than a standard Tkinter application.
5. Intuitive and User Friendly Interface: No manual or help is required to use the system. Everything is self explanatory and designed with simplicty in mind. Each book can have an 'About' section that the user can go through before issuing the book. Books also support addition of a cover photo and users support addition of profile picture.
6. Cheap, Portable and Lightweight: This system is cross-platform, and can be run on a variety of different systems, even on low-end, cheap platforms like Raspberry Pi. This could significantly bring costs down since a lower end single computer could be used to run this rather than a traditional computer.

Requirements

- OS: Microsoft Windows 7/8/8.1/10, Linux, Apple MacOS X
- RAM: 200 MB
- Processor: Any x86, x86_64 or ARM processor
- Storage: 50 MB (Not including MySQL Server)
- Python: 3.9+
- 3rd Party Python modules
    - **`PySide2`** - GUI Library
    - **`qtawesome`** - Icon Library
    - **`qt-material`** - Material Design Stylesheet for PySide2
    - **`mysql-connector-python`** - SQL Library to connect to MySQL Server

Database Structure

MySQL Server > snakebrary database

“users” table

Field	Type	Null	Key	Default	Extra
username	varchar(50)	NO	PRI	NULL	
password	text	NO		NULL	
password_hint	text	NO		NULL	
name	text	NO		NULL	
is_disabled	tinyint(1)	YES		NULL	
privilege	int(11)	NO		NULL	
photo	longblob	YES		NULL	
date_time_created	text	NO		NULL	

“account_settings” table

Field	Type	Null	Key	Default	Extra
username	varchar(50)	NO	PRI	NULL	
theme	text	NO		NULL	
accent_colour	text	NO		NULL	

“books” table

Field	Type	Null	Key	Default	Extra
ISBN	varchar(50)	NO	PRI	NULL	
name	text	NO		NULL	
authors	text	NO		NULL	
holders	text	NO		NULL	
genres	text	NO		NULL	
price	float	NO		NULL	
about	text	YES		NULL	
is_unavailable	tinyint(1)	YES		NULL	
photo	longblobt	YES		NULL	
Date_time_added	text	NO		NULL	

“books_ratings” table

Field	Type	Null	Key	Default	Extra
ISBN	varchar(50)	NO	PRI	NULL	
ratings	text	NO		NULL	

SQLite3 Local Database > snakebrary database

“local_settings” table

This table is used to store settings like the MySQL server username and password.

Field	Type	Null	Key	Default	Extra
key	varchar(50)	NO	PRI	NULL	
ratings	text	NO		NULL	

Code

The source code of this project has also been uploaded to a GitHub repository. Appropriate steps of executing it are also provided there.

GitHub Repository: https://github.com/rnayabed/SnakeBrary

The program can be started by executing the **`main.py`** file.

The project follows the particular file directory hierarchy :

- main.py
- ui
    - helpers
        - enhanced_controls.py
        - helpers.py
    - layouts_and_widgets
        - book_ratings_widget.py
        - user_info_vbox.py
        - user_wizard.py
    - window
        - dashboard
            - settings_tab
                - about.py
                - account_tab.py
                - general_tab.py
                - settings_tab.py
            - admin_users_table.py
            - books_tab_widget.py
            - dashboard.py
        - add_user.py
        - book_holders_window.py
        - book_info.py
        - book_reviewers_window.py
        - book_wizard_window.py
        - connection_details_widget.py
        - edit_user.py
        - license.py
        - login_prompt.py
        - user_info.py

        - welcome.py
- logic
    - book.py
    - database.py
    - user.py
- assets
    - app_icon.png
    - splash.png

main.py

```
from PySide2.QtCore import QCoreApplication, Qt
from PySide2.QtGui import QFontDatabase, QIcon, QPixmap
from PySide2.QtWidgets import QApplication, QSplashScreen
from logic.database import Database
from qt_material import apply_stylesheet
from mysql.connector import Error
from ui.helpers.helpers import center_screen
from ui.window.connection_details_widget import ConnectionDetailsWidget
from ui.window.login_prompt import LoginPrompt
from ui.window.welcome import Welcome

def start():
    is_fresh_run = True
    try:
        app = QApplication()
    except RuntimeError:
        is_fresh_run = False
        app = QCoreApplication.instance()

    app.setWindowIcon(QIcon('assets/app_icon.png'))
    app.setAttribute(Qt.AA_UseHighDpiPixmaps)

    if is_fresh_run:
        splash = QSplashScreen(QPixmap('assets/splash.png'))
        splash.show()

    app.processEvents()

    apply_stylesheet(app, theme='light_purple.xml')
    Database.create_local_connection()

    win = decide_window()

    if is_fresh_run:
        splash.finish(win)

    exit_code = app.exec_()
```

```
    Database.close_local_connection()
    Database.close_connection()

    if exit_code == 6504:
        start()

def start_connection_details_widget():
    connection_details = ConnectionDetailsWidget(decide_window)
    connection_details.show()
    center_screen(connection_details)
    return connection_details

def decide_window():
    if Database.is_new_local_setup():
        return start_connection_details_widget()
    else:
        if not Database.is_connected() :
            if Database.is_local_connection_settings_clear():
                return start_connection_details_widget()

            try:

Database.create_connection(Database.get_local_database_server_host(),

Database.get_local_database_server_user(),

Database.get_local_database_server_password(),

Database.get_local_database_server_port())
                return decide_window()
            except Error as e:
                print(e)
                connection_details_widget =
start_connection_details_widget()
                connection_details_widget.error_label.setText(e.msg)
                Database.clear_local_connection_settings()
                Database.save_local_database()
                return connection_details_widget

        if Database.is_new_server_setup():
            welcome = Welcome()
            welcome.show()
            center_screen(welcome)
            return welcome
        else:
            login_prompt = LoginPrompt()
            login_prompt.show()
            center_screen(login_prompt)
            return login_prompt
```

```
if __name__ == '__main__':
    start()
```

ui/helpers/enhanced_controls.py

```
import os
from PySide2 import QtCore
from PySide2.QtCore import QMargins, QSize, Qt
from PySide2.QtGui import QIcon, QImage, QPixmap
from PySide2.QtWidgets import QFileDialog, QLabel, QLineEdit,
QPlainTextEdit, QPushButton, QVBoxLayout, QHBoxLayout, \
    QComboBox, QWidget

class LineEdit(QWidget):

    def __init__(self, info=None, init_value=None, password_mode=False):
        super(LineEdit, self).__init__()

        self.info_label = QLabel(info)

        self.error_label = QLabel()
        self.error_label.setStyleSheet("color: red")

        self.error_label.setAlignment(Qt.AlignRight)

        self.upper = QHBoxLayout()
        self.upper.addWidget(self.info_label)
        self.upper.addWidget(self.error_label)

        self.line_edit = QLineEdit()

        self.line_edit.setText(init_value)

        vbox = QVBoxLayout()
        vbox.setContentsMargins(QMargins(0,0,0,0))
        vbox.addLayout(self.upper)

        lower = QHBoxLayout()
        lower.addWidget(self.line_edit)

        if password_mode:
            self.show_hide_button = QPushButton()

self.show_hide_button.clicked.connect(self.configure_show_hide_button)
            lower.addWidget(self.show_hide_button)
            self.password_mode_show(False)

        vbox.addLayout(lower)
        vbox.setSpacing(3)
```

```
        self.setLayout(vbox)

    def on_error(self, error):
        self.error_label.setText(error)

    def on_success(self):
        self.error_label.clear()

    def password_mode_show(self, show):
        self.current_password_mode = show
        if show:
            self.show_hide_button.setText('HIDE')
            self.line_edit.setEchoMode(QLineEdit.EchoMode.Normal)
        else:
            self.show_hide_button.setText('SHOW')
            self.line_edit.setEchoMode(QLineEdit.EchoMode.Password)

    def configure_show_hide_button(self):
        self.password_mode_show(not self.current_password_mode)

class PlainTextEdit(QWidget):

    def __init__(self, info, init_value=None):
        super(PlainTextEdit, self).__init__()

        self.info_label = QLabel(info)

        self.error_label = QLabel()
        self.error_label.setStyleSheet("color: red")

        self.error_label.setAlignment(Qt.AlignRight)

        upper = QHBoxLayout()
        upper.addWidget(self.info_label)
        upper.addWidget(self.error_label)

        self.plain_text_edit = QPlainTextEdit()

        self.plain_text_edit.setPlainText(init_value)

        vbox = QVBoxLayout()
        vbox.setContentsMargins(QMargins(0, 0, 0, 0))
        vbox.addLayout(upper)
        vbox.addWidget(self.plain_text_edit)
        vbox.setSpacing(3)

        self.setLayout(vbox)

    def on_error(self, error):
        self.error_label.setText(error)
```

```
    def on_success(self):
        self.error_label.clear()

class ComboBox(QWidget):

    def __init__(self, info, l):
        super(ComboBox, self).__init__()

        self.label = QLabel(info)

        self.combo_box = QComboBox()
        self.combo_box.addItems(l)

        hbox = QHBoxLayout()
        hbox.setContentsMargins(QMargins(0, 0, 0, 0))
        hbox.addWidget(self.label)
        hbox.addWidget(self.combo_box)
        hbox.setSpacing(3)

        self.setLayout(hbox)

class FilePicker(QWidget):

    def __init__(self, info, init_value=None, on_select=None,
on_clear=None):
        super(FilePicker, self).__init__()

        self.info = info
        self.on_select = on_select
        self.on_clear = on_clear

        self.info_label = QLabel(self.info)

        self.error_label = QLabel()
        self.error_label.setStyleSheet("color: red")

        self.error_label.setAlignment(Qt.AlignRight)

        upper = QHBoxLayout()
        upper.addWidget(self.info_label)
        upper.addWidget(self.error_label)

        self.line_edit = QLineEdit()
        self.line_edit.setEnabled(False)
        self.line_edit.setText(init_value)

        self.select_button = QPushButton('Select')
        self.select_button.clicked.connect(self.__select_file)
```

```
        self.clear_button = QPushButton('Clear')
        self.clear_button.clicked.connect(self.__clear_file)

        lower = QHBoxLayout()
        lower.addWidget(self.line_edit)
        lower.addWidget(self.select_button)
        lower.addWidget(self.clear_button)

        vbox = QVBoxLayout()
        vbox.setAlignment(QtCore.Qt.AlignCenter)
        vbox.setContentsMargins(QMargins(0, 0, 0, 0))
        vbox.addLayout(upper)
        vbox.addLayout(lower)
        vbox.setSpacing(3)

        self.setLayout(vbox)

    def on_error(self, error):
        self.error_label.setText(error)

    def on_success(self):
        self.error_label.clear()

    def __select_file(self):
        img_path = QFileDialog.getOpenFileName(None, 'Open File',
os.getcwd(), 'Image Files (*.jpg *.png)')[0]

        if img_path != '':
            self.line_edit.setText(img_path)
            if self.on_select != None:
                self.on_select(img_path)

    def __clear_file(self):
        self.line_edit.clear()
        if self.on_clear != None:
            self.on_clear()

class ImageView(QLabel):

    def __init__(self, info, width, height, style='border: 2px solid
black;'):
        super(ImageView, self).__init__()

        self.info = info
        self.style = style

        self.setText(self.info)
        self.setStyleSheet(self.style)
        self.setAlignment(QtCore.Qt.AlignCenter)
        self.setFixedSize(width, height)
```

```
        self.is_clear = True

    def set_image_from_blob(self, blob):
        self.setPixmap(QPixmap.fromImage(QImage.fromData(blob))
                       .scaled(self.width(), self.height(),
                               QtCore.Qt.KeepAspectRatio))
        self.is_clear = False

    def set_image_from_path(self, path):
        self.setPixmap(QPixmap(path).scaled(self.width(), self.height(),
QtCore.Qt.KeepAspectRatio,

QtCore.Qt.SmoothTransformation))
        self.is_clear = False

    def clear_image(self):
        self.clear()
        self.setText(self.info)
        self.is_clear = True
```

ui/helpers/helpers.py

```
from PySide2.QtGui import QIcon, QScreen, QFont
from PySide2.QtWidgets import QApplication, QLayout, QWidget

def center_screen(window):
    center =
QScreen.availableGeometry(QApplication.primaryScreen()).center()
    geo = window.frameGeometry()
    geo.moveCenter(center)
    window.move(geo.topLeft())

def get_font_size(size):
    font = QFont()
    font.setPixelSize(size)
    return font

def delete_layouts_in_layout(layout: QLayout):
    for i in range(layout.count()):
        layout.itemAt(i).layout().deleteLater()

def delete_widgets_in_layout(layout):
    if layout is not None:
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            if widget is not None:
```

```
                widget.setParent(None)
            else:
                deleteItemsOfLayout(item.layout())
```

ui/layouts_and_widgets/books_ratings_widget.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QHBoxLayout, QLabel, QProgressBar,
QPushButton, QSlider, QVBoxLayout, QWidget
import os
from logic.book import Book
from logic.database import Database
from logic.user import User
from ui.helpers.helpers import get_font_size, delete_widgets_in_layout

import os
import qtawesome as qta

class BookRatingsWidget(QWidget):

    def __init__(self, book: Book, current_user: User):
        super(BookRatingsWidget, self).__init__(None)

        self.book = book
        self.current_user = current_user

        header_label = QLabel('Ratings')
        header_label.setContentsMargins(QtCore.QMargins(0, 10, 0, 0))
        header_label.setFont(get_font_size(18))

        self.vbox = QVBoxLayout()

        self.vbox.addWidget(header_label)

        overview_hbox = QHBoxLayout()

        self.large_rating_label = QLabel()
        self.large_rating_label.setContentsMargins(QtCore.QMargins(0, 0,
50, 0))
        self.large_rating_label.setFont(get_font_size(35))

        self.rating_graph_hbox = QHBoxLayout()
```

```
        self.total_ratings_label = QLabel()
        self.total_ratings_label.setFont(get_font_size(14))

        left_rating_layout = QVBoxLayout()
        left_rating_layout.addWidget(self.large_rating_label)
        left_rating_layout.addLayout(self.rating_graph_hbox)
        left_rating_layout.addWidget(self.total_ratings_label)

        overview_hbox.addLayout(left_rating_layout)

        right_layout_vbox = QVBoxLayout()

        self.rating_progress_bar_5 = RatingProgressBar(5)
        self.rating_progress_bar_4 = RatingProgressBar(4)
        self.rating_progress_bar_3 = RatingProgressBar(3)
        self.rating_progress_bar_2 = RatingProgressBar(2)
        self.rating_progress_bar_1 = RatingProgressBar(1)

        right_layout_vbox.addLayout(self.rating_progress_bar_5)
        right_layout_vbox.addLayout(self.rating_progress_bar_4)
        right_layout_vbox.addLayout(self.rating_progress_bar_3)
        right_layout_vbox.addLayout(self.rating_progress_bar_2)
        right_layout_vbox.addLayout(self.rating_progress_bar_1)

        overview_hbox.addLayout(right_layout_vbox)

        self.vbox.addLayout(overview_hbox)

        self.rating_slider = QSlider(QtCore.Qt.Horizontal)
        self.rating_slider.setMinimum(1)
        self.rating_slider.setMaximum(5)
        self.rating_slider.setTickInterval(1)

        self.rating_slider_status_label = QLabel('1')


self.rating_slider.valueChanged.connect(self.rating_slider_value_changed)

        self.submit_rating_button = QPushButton('Submit Rating')

self.submit_rating_button.clicked.connect(self.submit_rating_button_clicke
d)

        self.delete_rating_button = QPushButton('Delete Rating')
        self.delete_rating_button.setProperty('class', 'danger')

self.delete_rating_button.clicked.connect(self.delete_rating_button_clicke
d)

        rating_layout = QVBoxLayout()
```

```
        rating_layout.setContentsMargins(QtCore.QMargins(0, 0, 0, 0))

        self.rating_current_status_label = QLabel()
        rating_layout.addWidget(self.rating_current_status_label)

        rating_layout_hbox = QHBoxLayout()
        rating_layout_hbox.setContentsMargins(QtCore.QMargins(0, 0, 0, 0))
        rating_layout_hbox.addWidget(self.rating_slider)
        rating_layout_hbox.addWidget(self.rating_slider_status_label)
        rating_layout_hbox.addWidget(self.submit_rating_button)
        rating_layout_hbox.addWidget(self.delete_rating_button)

        rating_layout.addLayout(rating_layout_hbox)

        self.rating_layout_widget = QWidget()
        self.rating_layout_widget.setLayout(rating_layout)

        self.vbox.addWidget(self.rating_layout_widget)

        self.setLayout(self.vbox)

        self.setContentsMargins(QtCore.QMargins(0, 0, 0, 0))

        self.configure_ui()

    def rating_slider_value_changed(self):

self.rating_slider_status_label.setText(str(self.rating_slider.value()))

    def delete_rating_button_clicked(self):

self.book_ratings.delete_rating_by_username(self.current_user.username)
        Database.update_book_ratings(self.book_ratings)
        self.configure_ui()

    def submit_rating_button_clicked(self):

self.book_ratings.set_rating_by_username(self.current_user.username,
self.rating_slider.value())
        Database.update_book_ratings(self.book_ratings)
        self.configure_ui()

    def reload(self, book):
        self.book = book
        self.configure_ui()

    def configure_ui(self):
        self.book_ratings = Database.get_book_ratings(self.book.ISBN)
        average_rating = self.book_ratings.get_average_rating()
```

```
        self.large_rating_label.setText(str(average_rating))

        self.set_rating_graphic(average_rating)

        if len(self.book_ratings.ratings) == 1:

self.total_ratings_label.setText(f'{len(self.book_ratings.ratings)}
rating')
        else:

self.total_ratings_label.setText(f'{len(self.book_ratings.ratings)}
ratings')

        self.rating_progress_bar_1.load(self.book_ratings)
        self.rating_progress_bar_2.load(self.book_ratings)
        self.rating_progress_bar_3.load(self.book_ratings)
        self.rating_progress_bar_4.load(self.book_ratings)
        self.rating_progress_bar_5.load(self.book_ratings)

        if not self.book.is_eligible_to_rate(self.current_user.username):
            self.rating_layout_widget.hide()
        else:
            self.rating_layout_widget.show()
            existing_rating =
self.book_ratings.get_rating_by_username(self.current_user.username)

            if existing_rating == None:
                self.rating_current_status_label.setText(
                    'You have read but not rated this book yet. Go ahead
and rate it!')
                self.delete_rating_button.hide()
            else:
                self.rating_current_status_label.setText(f'You have rated
this book {existing_rating} out of 5')
                self.rating_slider.setValue(existing_rating)
                self.delete_rating_button.show()

    def get_rating_progress_bar_for_rating(self, rating):
        rate_label = QLabel(str(rating))
        rating_bar = QProgressBar()

        if len(self.book_ratings.ratings) == 0:
            rating_bar.setValue(0)
        else:

rating_bar.setValue(self.book_ratings.get_ratings_by_proportion(rating))

        hbox = QHBoxLayout()
        hbox.addWidget(rate_label)
        hbox.addWidget(rating_bar)
```

```
        return hbox

    def set_rating_graphic(self, rating):
        rating_broken = str(rating).split('.')
        major = int(rating_broken[0])
        minor = int(rating_broken[1])

        delete_widgets_in_layout(self.rating_graph_hbox)

        for i in range(0, major):

self.rating_graph_hbox.addWidget(self.get_label_with_icon('mdi.star'))

        if minor >= 5:

self.rating_graph_hbox.addWidget(self.get_label_with_icon('mdi.star-half-
full'))
        else:

self.rating_graph_hbox.addWidget(self.get_label_with_icon('mdi.star-
outline'))

        for i in range(4 - major):

self.rating_graph_hbox.addWidget(self.get_label_with_icon('mdi.star-
outline'))

    def get_label_with_icon(self, icon_code):
        label = QLabel()
        label.setPixmap(qta.icon(icon_code,
color=os.environ.get('QTMATERIAL_PRIMARYCOLOR')).pixmap(32))
        return label

class RatingProgressBar(QHBoxLayout):

    def __init__(self, rating):
        super(RatingProgressBar, self).__init__(None)

        self.rating = rating
        self.rate_label = QLabel(str(self.rating))
        self.rating_bar = QProgressBar()

        self.addWidget(self.rate_label)
        self.addWidget(self.rating_bar)

    def load(self, book_ratings):
        if len(book_ratings.ratings) == 0:
            self.rating_bar.setValue(0)
        else:
```

```
self.rating_bar.setValue(book_ratings.get_ratings_by_proportion(self.ratin
g))
```

ui/layouts_and_widgets/user_info_vbox.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QHBoxLayout, QLabel, QMessageBox,
QPushButton, QVBoxLayout, QWidget

from logic.database import Database
from logic.user import User, UserPrivilege
from ui.helpers.enhanced_controls import ImageView
from ui.helpers.helpers import get_font_size, center_screen
from ui.window.edit_user import EditUser

class UserInfoVBox(QVBoxLayout):

    def __init__(self, user: User, current_user: User,
dashboard_on_user_edited, parent, is_account_tab=False,
                 disable_edit_options=False):
        super(UserInfoVBox, self).__init__(parent)

        self.dashboard_on_user_edited = dashboard_on_user_edited
        self.current_user = current_user
        self.parent = parent
        self.user = user
        self.is_account_tab = is_account_tab
        self.disable_edit_options = disable_edit_options

        self.setAlignment(QtCore.Qt.AlignTop)

        hbox_1 = QHBoxLayout()

        self.profile_photo = ImageView('Profile Photo', 300, 300)
        hbox_1.addWidget(self.profile_photo)

        self.name_label = QLabel()
        self.name_label.setFont(get_font_size(30))

        self.username_label = QLabel()

        self.password_widget = PasswordWidget(self.user)

        self.privilege_label = QLabel()

        self.date_time_created_label = QLabel()
```

```
        self.edit_user_button = QPushButton('Edit')

self.edit_user_button.clicked.connect(self.edit_user_button_onclick)

        self.delete_user_button = QPushButton('Delete')
        self.delete_user_button.setProperty('class', 'danger')

self.delete_user_button.clicked.connect(self.delete_user_button_onclick)

        self.edit_delete_button_hbox = QHBoxLayout()
        self.edit_delete_button_hbox.setContentsMargins(QtCore.QMargins(0,
0, 0, 0))
        self.edit_delete_button_hbox.addWidget(self.edit_user_button)
        self.edit_delete_button_hbox.addWidget(self.delete_user_button)

        self.disable_enable_button = QPushButton()

        self.edit_delete_button_widget = QWidget()

self.edit_delete_button_widget.setContentsMargins(QtCore.QMargins(0, 0, 0,
0))

self.edit_delete_button_widget.setLayout(self.edit_delete_button_hbox)

        vbox_labels_1 = QVBoxLayout()
        vbox_labels_1.setAlignment(QtCore.Qt.AlignTop)
        vbox_labels_1.addWidget(self.name_label)
        vbox_labels_1.addWidget(self.username_label)
        vbox_labels_1.addWidget(self.privilege_label)
        vbox_labels_1.addWidget(self.date_time_created_label)
        vbox_labels_1.addWidget(self.password_widget)
        vbox_labels_1.addWidget(self.disable_enable_button)
        vbox_labels_1.addWidget(self.edit_delete_button_widget)

        hbox_1.addLayout(vbox_labels_1)

        self.addLayout(hbox_1)

        self.configure_ui()

    def configure_ui(self):
        self.password_widget.reload_user(self.user)

        if self.user.photo == None:
            self.profile_photo.clear_image()
            self.profile_photo.hide()
        else:
            self.profile_photo.set_image_from_blob(self.user.photo)
            self.profile_photo.show()
```

```
        self.name_label.setText(self.user.name)
        self.username_label.setText(f'Username: {self.user.username}')
        self.privilege_label.setText(f'Privilege:
{UserPrivilege.get_ui_name(self.user.privilege)}')
        self.date_time_created_label.setText(f'Date/Time created:
{self.user.date_time_created}')

        is_enable_disable_button_visible = True

        if (self.current_user.privilege == UserPrivilege.ADMIN and
self.user.privilege == UserPrivilege.MASTER) or (
                self.current_user.privilege == self.user.privilege and
self.current_user.username != self.user.username and
self.current_user.privilege == UserPrivilege.ADMIN):
            self.password_widget.hide()
            self.edit_delete_button_widget.hide()
            self.disable_enable_button.hide()
            is_enable_disable_button_visible = False

        if self.current_user.privilege == UserPrivilege.NORMAL:
            self.password_widget.hide()
            self.disable_enable_button.hide()
            is_enable_disable_button_visible = False

        if self.is_account_tab:
            self.password_widget.hide()
            self.privilege_label.hide()

        if self.current_user.username == self.user.username:
            self.delete_user_button.hide()
            self.disable_enable_button.hide()
            is_enable_disable_button_visible = False

        if self.disable_edit_options:
            self.delete_user_button.hide()
            self.edit_delete_button_widget.hide()
            self.password_widget.hide()
            self.disable_enable_button.hide()
            is_enable_disable_button_visible = False

        if is_enable_disable_button_visible:
            self.configure_disable_enable_button()


    def configure_disable_enable_button(self):
        self.disconnect_slots_disable_enable_button()

        if self.user.is_disabled:
            self.disable_enable_button.show()
            self.disable_enable_button.setText('Enable')
```

```
            self.disable_enable_button.clicked.connect(lambda:
self.enable_disable_user(False))
        else:
            self.disable_enable_button.show()
            self.disable_enable_button.setText('Disable')
            self.disable_enable_button.clicked.connect(lambda:
self.enable_disable_user(True))

    def enable_disable_user(self, is_disabled):
        self.user.is_disabled = is_disabled
        Database.update_user(self.user)

        self.configure_disable_enable_button()


    def disconnect_slots_disable_enable_button(self):
        try:
            self.disable_enable_button.clicked.disconnect()
        except:
            pass

    def delete_user_button_onclick(self):
        warning_box = QMessageBox.warning(self.parent, 'Warning', f'''Are
you sure you want to delete the following user
Name: {self.user.name}
Username: {self.user.username}''', QMessageBox.Yes, QMessageBox.No)

        if warning_box == QMessageBox.Yes:
            Database.delete_user(self.user.username)
            if self.dashboard_on_user_edited != None:
                self.dashboard_on_user_edited()
            self.parent.close()

    def edit_user_button_onclick(self):
        self.edit_user_window = EditUser(self.user, self.on_user_edited,
self.parent)
        self.edit_user_window.exec()
        center_screen(self.edit_user_window)

    def on_user_edited(self):
        if self.dashboard_on_user_edited != None:
            self.dashboard_on_user_edited()
        self.user = Database.get_user_by_username(self.user.username)
        self.configure_ui()
        center_screen(self.parent)

class PasswordWidgetMode:
    HIDE = 0,
    SHOW = 1
```

```
class PasswordWidget(QWidget):

    def __init__(self, user):
        super(PasswordWidget, self).__init__(None)

        self.user = user

        self.password_label = QLabel()
        self.password_hint_label = QLabel()
        self.password_show_hide_button = QPushButton()
        self.password_show_hide_button.clicked.connect(self.toggle_mode)

        password_vbox = QVBoxLayout()
        password_vbox.setContentsMargins(QtCore.QMargins(0, 0, 0, 0))
        password_vbox.addWidget(self.password_label)
        password_vbox.addWidget(self.password_hint_label)
        password_vbox.addWidget(self.password_show_hide_button)

        self.setLayout(password_vbox)

        self.set_current_mode(PasswordWidgetMode.HIDE)

    def reload_user(self, user):
        self.user = user
        self.set_current_mode(self.mode)

    def set_current_mode(self, mode: PasswordWidgetMode):
        self.mode = mode
        if self.mode == PasswordWidgetMode.HIDE:
            self.password_label.setText('Password: ********')
            self.password_hint_label.setText('Password Hint: *******')
            self.password_show_hide_button.setText('Show Password and
Hint')
        elif self.mode == PasswordWidgetMode.SHOW:
            self.password_label.setText(f'Password :
{self.user.password}')

            if self.user.password_hint == '':
                self.password_hint_label.setText('Password Hint not
configured')
            else:
                self.password_hint_label.setText(f'Password Hint:
{self.user.password_hint}')

            self.password_show_hide_button.setText('Hide Password and
Hint')

    def toggle_mode(self):
        if self.mode == PasswordWidgetMode.HIDE:
            self.set_current_mode(PasswordWidgetMode.SHOW)
```

```
        else:
            self.set_current_mode(PasswordWidgetMode.HIDE)
```

ui/layouts_and_widgets/user_wizard.py

```
from PySide2.QtWidgets import QApplication, QHBoxLayout, QMessageBox,
QVBoxLayout, QPushButton

from logic.database import Database
from logic.user import UserPrivilege, User
from ui.helpers.enhanced_controls import FilePicker, ImageView, LineEdit

class UserWizardMode:
    ADD = 1,
    EDIT = 2

class UserWizard(QVBoxLayout):

    def __init__(self, on_success=None, on_error=None,
new_user_privilege=None, old_user=None):
        super(UserWizard, self).__init__()

        self.on_success = on_success
        self.on_error = on_error

        self.new_user_photo_path_field = FilePicker('Profile picture
(Optional)', on_select=self.on_user_photo_selected,

on_clear=self.on_user_photo_cleared)

        self.new_user_photo_preview = ImageView('Preview', 200, 200)

        self.photo_hbox = QHBoxLayout()
        self.photo_hbox.addWidget(self.new_user_photo_path_field)
        self.photo_hbox.addWidget(self.new_user_photo_preview)

        self.new_user_name_field = LineEdit()

        self.new_user_username_field = LineEdit()
        self.new_user_password_field = LineEdit(password_mode=True)
        self.new_user_password_confirm_field =
LineEdit(password_mode=True)
        self.new_user_password_field_hint = LineEdit('Password Hint
(Optional)')

        self.proceed_button = QPushButton('Proceed')

self.proceed_button.clicked.connect(self.on_proceed_button_clicked)
```

```
        # Create layout and add widgets
        self.addLayout(self.photo_hbox)
        self.addWidget(self.new_user_name_field)
        self.addWidget(self.new_user_username_field)
        self.addWidget(self.new_user_password_field)
        self.addWidget(self.new_user_password_confirm_field)
        self.addWidget(self.new_user_password_field_hint)
        self.addWidget(self.proceed_button)

        if old_user == None:
            self.mode = UserWizardMode.ADD
            self.user_privilege = new_user_privilege
        else:
            self.old_user = old_user
            self.user_privilege = self.old_user.privilege
            self.load_values_for_old_user()
            self.mode = UserWizardMode.EDIT
            self.new_user_username_field.line_edit.setReadOnly(True)

        self.new_user_name_field.info_label.setText('Name')
        self.new_user_username_field.info_label.setText('Username')
        self.new_user_password_field.info_label.setText('Password')
        self.new_user_password_confirm_field.info_label.setText('Confirm
Password')

    def load_values_for_old_user(self):
        if self.old_user.photo != None:

self.new_user_photo_preview.set_image_from_blob(self.old_user.photo)

        self.new_user_name_field.line_edit.setText(self.old_user.name)

self.new_user_username_field.line_edit.setText(self.old_user.username)

self.new_user_password_field.line_edit.setText(self.old_user.password)

self.new_user_password_confirm_field.line_edit.setText(self.old_user.passw
ord)

self.new_user_password_field_hint.line_edit.setText(self.old_user.password
_hint)

    def on_user_photo_selected(self, img_path):
        self.new_user_photo_preview.set_image_from_path(img_path)

    def on_user_photo_cleared(self):
        self.new_user_photo_path_field.line_edit.clear()
        self.new_user_photo_preview.clear_image()
```

```
    def on_proceed_button_clicked(self):
        proposed_new_user_photo_path =
self.new_user_photo_path_field.line_edit.text()
        proposed_new_user_name = self.new_user_name_field.line_edit.text()
        proposed_new_user_username =
self.new_user_username_field.line_edit.text()
        proposed_new_user_password =
self.new_user_password_field.line_edit.text()
        proposed_new_user_password_confirm =
self.new_user_password_confirm_field.line_edit.text()
        proposed_new_user_password_hint =
self.new_user_password_field_hint.line_edit.text()

        error = False

        if len(proposed_new_user_name) < 1:
            self.new_user_name_field.on_error('Required')
            error = True
        else:
            self.new_user_name_field.on_success()

        if len(proposed_new_user_username) < 1:
            self.new_user_username_field.on_error('Required')
            error = True
        elif len(proposed_new_user_username) > 50:
            self.new_user_username_field.on_error('Too long!')
            error = True
        else:
            self.new_user_username_field.on_success()

        if len(proposed_new_user_password) < 8:
            self.new_user_password_field.on_error('Too short - Must be at
least 8 characters')
            error = True
        else:
            self.new_user_password_field.on_success()

        if proposed_new_user_password_confirm !=
proposed_new_user_password:
            self.new_user_password_confirm_field.on_error('Passwords do
not match')
            error = True
        else:
            self.new_user_password_confirm_field.on_success()

        if error:
            if self.on_error is not None:
                self.on_error()
            return
```

```
        self.set_disable(True)

        if Database.is_new_server_setup():
            Database.create_new_tables()

        new_user = User(proposed_new_user_username,
proposed_new_user_password,
                        proposed_new_user_password_hint,
proposed_new_user_name,
                        privilege=self.user_privilege)

        if proposed_new_user_photo_path != '':
            file = open(proposed_new_user_photo_path, 'rb')
            new_user.photo = file.read()
            file.close()
        else:
            if self.mode == UserWizardMode.EDIT and
self.new_user_photo_preview.is_clear == False:
                new_user.photo = self.old_user.photo

        if self.mode == UserWizardMode.ADD:
            old_user =
Database.get_user_by_username(proposed_new_user_username)
            if old_user != None:
                QMessageBox.critical(None, 'Error', f'''User with same
username already exists.
Name: {old_user.name}
Privilege: {UserPrivilege.get_ui_name(old_user.privilege)}
Date Time Created: {old_user.date_time_created}''', QMessageBox.Ok)
                self.set_disable(False)
                return
            Database.create_new_user(new_user)

        else:
            Database.update_user(new_user)

        if self.on_success is not None:
            self.on_success()

    def set_disable(self, disable):
        self.proceed_button.setDisabled(disable)
        self.new_user_name_field.line_edit.setReadOnly(disable)
        self.new_user_username_field.line_edit.setReadOnly(disable)
        self.new_user_password_field.line_edit.setReadOnly(disable)

self.new_user_password_confirm_field.line_edit.setReadOnly(disable)
        self.new_user_password_field_hint.line_edit.setReadOnly(disable)
        QApplication.instance().processEvents()
```

ui/window/dashboard/settings_tab/about.py

```
import importlib.metadata
import platform
from sqlite3.dbapi2 import sqlite_version
from ui.window.license import License
from ui.helpers.enhanced_controls import ImageView

from PySide2 import QtCore
from PySide2.QtWidgets import QHBoxLayout, QPushButton, QWidget,
QVBoxLayout, QLabel

from ui.helpers.helpers import get_font_size

class About(QWidget):

    def __init__(self, parent=None):
        super(About, self).__init__(parent)

        github_url = 'https://github.com/rnayabed/SnakeBrary'
        synopsis_url =
'https://raw.githubusercontent.com/rnayabed/SnakeBrary/master/synopsis.pdf
'
        version = '1.0.0'

        layout = QVBoxLayout()
        layout.setAlignment(QtCore.Qt.AlignCenter)

        app_icon_hbox = QHBoxLayout()
        app_icon = ImageView('App Icon', 150, 150, style=None)
        app_icon.set_image_from_path('assets/app_icon.png')
        app_icon_hbox.addWidget(app_icon)
        layout.addLayout(app_icon_hbox)

        heading_label = QLabel('SnakeBrary')
        heading_label.setFont(get_font_size(25))
        heading_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(heading_label)

        sub_heading_label = QLabel('<i>A Sweet and Simple Library
Management System</i>')
        sub_heading_label.setFont(get_font_size(17))
        sub_heading_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(sub_heading_label)

        maker_label = QLabel('Made by Debayan Sutradhar, 12 M
(2020/14275)')
        maker_label.setFont(get_font_size(15))
        maker_label.setAlignment(QtCore.Qt.AlignCenter)
        maker_label.setContentsMargins(QtCore.QMargins(0, 0, 0, 30))
```

```
        layout.addWidget(maker_label)

        school_label = QLabel('DPS Ruby Park, Kolkata')
        school_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(school_label)

        small_info_label = QLabel('CBSE Class 12 Computer Science
Project')
        small_info_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(small_info_label)

        synopsis_label = QLabel(f'<a href="{synopsis_url}">Project
Synopsis</a>')
        synopsis_label.setOpenExternalLinks(True)
        synopsis_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(synopsis_label)

        source_code_hyperlink = QLabel(f'<a href="{github_url}">Source
Code</a>')
        source_code_hyperlink.setOpenExternalLinks(True)
        source_code_hyperlink.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(source_code_hyperlink)

        license_button = QPushButton('License')
        license_button.setMinimumWidth(5)
        license_button.clicked.connect(self.license_button_clicked)
        license_button.setContentsMargins(QtCore.QMargins(0, 0, 0, 30))
        layout.addWidget(license_button)

        version_info_hbox = QHBoxLayout()
        version_info_hbox.setAlignment(QtCore.Qt.AlignCenter)

        version_info_hbox.addWidget(QLabel(f'Version {version}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'Qt {QtCore.qVersion()}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'Python
{platform.python_version()}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'SQLite {sqlite_version}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'MySQL Connector
{importlib.metadata.version("mysql-connector-python")}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'qt-material
{importlib.metadata.version("qt-material")}'))
        version_info_hbox.addWidget(self.get_seperator())
        version_info_hbox.addWidget(QLabel(f'qtawesome
{importlib.metadata.version("qtawesome")}'))
        version_info_hbox.addWidget(self.get_seperator())
```

```
        version_info_hbox.addWidget(QLabel(f'{platform.system()}
{platform.release()}'))

        layout.addLayout(version_info_hbox)

        self.setLayout(layout)

    def get_seperator(self):
        seperator_label = QLabel('|')
        seperator_label.setStyleSheet('color: grey;')
        return seperator_label

    def license_button_clicked(self):
        license_window = License(self)
        license_window.exec()
```

ui/window/dashboard/settings_tab/account_tab.py

```
from PySide2.QtWidgets import QWidget

from ui.layouts_and_widgets.user_info_vbox import UserInfoVBox

class AccountTab(QWidget):

    def __init__(self, current_user, dashboard_on_user_edited):
        super(AccountTab, self).__init__()

        self.current_user = current_user
        self.dashboard_on_user_edited = dashboard_on_user_edited

        self.user_info_vbox = UserInfoVBox(self.current_user,
self.current_user, self.dashboard_on_user_edited, self, True)
        self.setLayout(self.user_info_vbox)
```

ui/window/dashboard/settings_tab/general_tab.py

```
from PySide2.QtCore import QCoreApplication, Qt
from PySide2.QtWidgets import QApplication, QMessageBox, QPushButton,
QWidget, QVBoxLayout
from qt_material import apply_stylesheet, QtStyleTools

from logic.database import Database
from logic.user import UserPrivilege
from ui.helpers.enhanced_controls import ComboBox

class GeneralTab(QWidget, QtStyleTools):
```

```
    def __init__(self, current_user, current_user_account_settings):
        super(GeneralTab, self).__init__()

        self.current_user_account_settings = current_user_account_settings

        layout = QVBoxLayout()

        layout.setAlignment(Qt.AlignTop)

        self.themes = [
            'light', 'dark'
        ]

        self.themes_ui = [
            'Light', 'Dark'
        ]

        self.accent_colours = [
            'amber', 'blue', 'cyan', 'lightgreen', 'pink', 'purple',
'red', 'teal', 'yellow'
        ]

        self.accent_colours_ui = [
            'Amber', 'Blue', 'Cyan', 'Light Green', 'Pink', 'Purple',
'Red', 'Teal', 'Yellow'
        ]

        self.theme_combo_box = ComboBox('Theme', self.themes_ui)

self.theme_combo_box.combo_box.setCurrentIndex(self.themes.index(self.curr
ent_user_account_settings.theme))

self.theme_combo_box.combo_box.currentIndexChanged.connect(self.change_the
me)
        self.accent_colour_combo_box = ComboBox('Accent Colour',
self.accent_colours_ui)
        self.accent_colour_combo_box.combo_box.setCurrentIndex(

self.accent_colours.index(self.current_user_account_settings.accent_colour
))

self.accent_colour_combo_box.combo_box.currentIndexChanged.connect(self.ch
ange_theme)

        layout.addWidget(self.theme_combo_box)
        layout.addWidget(self.accent_colour_combo_box)

        self.clear_local_connection_settings_button = QPushButton('Clear
Connection Settings')
```

```
self.clear_local_connection_settings_button.clicked.connect(self.clear_loc
al_connection_settings)

self.clear_local_connection_settings_button.setDisabled(Database.is_local_
connection_settings_clear())

        layout.addWidget(self.clear_local_connection_settings_button)

        self.logout_button = QPushButton('Logout')
        self.logout_button.setProperty('class', 'danger')
        self.logout_button.clicked.connect(self.restart)

        layout.addWidget(self.logout_button)

        self.reset_button = QPushButton('Reset')
        self.reset_button.setProperty('class', 'danger')
        self.reset_button.clicked.connect(self.reset)

        if current_user.privilege == UserPrivilege.MASTER:
            layout.addWidget(self.reset_button)

        self.setLayout(layout)

    def change_theme(self):
        chosen_theme =
self.themes[self.theme_combo_box.combo_box.currentIndex()]
        chosen_accent_colour =
self.accent_colours[self.accent_colour_combo_box.combo_box.currentIndex()]

        stylesheet_name = f'{chosen_theme}_{chosen_accent_colour}.xml'
        apply_stylesheet(QApplication.instance(), stylesheet_name)

        self.current_user_account_settings.theme = chosen_theme
        self.current_user_account_settings.accent_colour =
chosen_accent_colour

Database.update_user_account_settings(self.current_user_account_settings)

    def reset(self):
        confirm_delete_box = QMessageBox.warning(self, 'Warning', f'''This
will DELETE EVERYTHING - books, users, etc. No data can be recovered.
Continue?''', QMessageBox.Yes, QMessageBox.No)

        if confirm_delete_box == QMessageBox.Yes:
            Database.delete_database()
            Database.delete_local_database()

            self.restart()
```

```
    def restart(self):
        QApplication.closeAllWindows()
        QCoreApplication.exit(6504)

    def clear_local_connection_settings(self):
        Database.clear_local_connection_settings()
        Database.save_local_database()
        self.clear_local_connection_settings_button.setDisabled(True)
```

ui/window/dashboard/settings_tab/settings_tab.py

```
from PySide2.QtWidgets import QWidget, QVBoxLayout, QTabWidget

from ui.window.dashboard.settings_tab.about import About
from ui.window.dashboard.settings_tab.account_tab import AccountTab
from ui.window.dashboard.settings_tab.general_tab import GeneralTab

class SettingsTab(QWidget):

    def __init__(self, current_user, current_user_settings,
dashboard_on_user_edited):
        super(SettingsTab, self).__init__()

        layout = QVBoxLayout()

        tabs = QTabWidget()
        tabs.addTab(GeneralTab(current_user, current_user_settings),
'General')
        self.account_tab = AccountTab(current_user,
dashboard_on_user_edited)
        tabs.addTab(self.account_tab, 'Account')
        tabs.addTab(About(), 'About')
        layout.addWidget(tabs)

        self.setLayout(layout)
```

ui/window/dashboard/admin_users_tab.py

```
from PySide2 import QtWidgets
from PySide2.QtWidgets import QApplication, QLabel, QWidget, QVBoxLayout,
QTableWidget, QPushButton, QHBoxLayout

from logic.database import Database
from logic.user import UserPrivilege, User
from ui.helpers.enhanced_controls import LineEdit
from ui.helpers.helpers import center_screen
```

```
from ui.window.add_user import AddUser
from ui.window.user_info import UserInfo

class AdminUsersTab(QWidget):

    def __init__(self, current_user: User, dashboard_on_user_edited,
parent=None):
        super(AdminUsersTab, self).__init__(parent)

        self.current_user = current_user
        self.dashboard_on_user_edited = dashboard_on_user_edited

        layout = QVBoxLayout()

        button_bar = QHBoxLayout()

        self.add_admin_button = QPushButton('New Admin user')
        self.add_admin_button.clicked.connect(lambda:
self.add_new_user(UserPrivilege.ADMIN))

        self.add_normal_button = QPushButton('New Normal user')
        self.add_normal_button.clicked.connect(lambda:
self.add_new_user(UserPrivilege.NORMAL))

        self.reload_button = QPushButton('Reload')
        self.reload_button.clicked.connect(self.reload_button_clicked)

        button_bar.addWidget(self.add_admin_button)
        button_bar.addWidget(self.add_normal_button)
        button_bar.addWidget(self.reload_button)

        self.users_table = QTableWidget()
        self.users_table.clicked.connect(self.users_table_clicked)

        self.search_bar = LineEdit('Search for user')

self.search_bar.line_edit.textEdited.connect(self.search_bar_value_changed
)
        self.search_bar.line_edit.setPlaceholderText('Search by Name,
Username or Privilege')

        layout.addLayout(button_bar)
        layout.addWidget(self.search_bar)
        layout.addWidget(self.users_table)

        self.setLayout(layout)

        if self.current_user.privilege == UserPrivilege.ADMIN:
            self.add_admin_button.hide()
```

```
        self.configure_users_table()

    def reload_button_clicked(self):
        self.reload_button.setDisabled(True)
        QApplication.instance().processEvents()
        self.configure_users_table()
        self.reload_button.setDisabled(False)

    def search_bar_value_changed(self):
        search = self.search_bar.line_edit.text().lower()

        for i in range(self.users_table.rowCount()):
            user = self.users_table.cellWidget(i, 0).property('user_obj')

            if not search in (user.name + user.username +
UserPrivilege.get_ui_name(user.privilege).lower()):
                self.users_table.hideRow(i)
            else:
                self.users_table.showRow(i)

    def add_new_user(self, user_privilege):
        new_user_window = AddUser(user_privilege,
self.configure_users_table, self)
        new_user_window.exec()
        center_screen(new_user_window)

    def configure_users_table(self):
        l_users = Database.get_all_users()

        self.users_table.clear()
        self.users_table.setSortingEnabled(True)
        self.users_table.setRowCount(len(l_users))
        self.users_table.setColumnCount(3)
        self.users_table.setHorizontalHeaderLabels(["Name", " Username ",
" Privilege "])

        self.users_table.horizontalHeader().setSectionResizeMode(0,
QtWidgets.QHeaderView.Stretch)
        self.users_table.horizontalHeader().setSectionResizeMode(1,
QtWidgets.QHeaderView.ResizeToContents)
        self.users_table.horizontalHeader().setSectionResizeMode(2,
QtWidgets.QHeaderView.ResizeToContents)


self.users_table.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderVi
ew.Fixed)
        self.users_table.verticalHeader().setDefaultSectionSize(70)

        for i in range(len(l_users)):
            each_user = l_users[i]
```

```
            name_widget = QLabel(each_user.name)
            username_widget = QLabel(each_user.username)
            privilege_widget =
QLabel(UserPrivilege.get_ui_name(each_user.privilege))

            name_widget.setProperty('user_obj', each_user)
            username_widget.setProperty('user_obj', each_user)
            privilege_widget.setProperty('user_obj', each_user)

            self.users_table.setCellWidget(i, 0, name_widget)
            self.users_table.setCellWidget(i, 1, username_widget)
            self.users_table.setCellWidget(i, 2, privilege_widget)

    def users_table_clicked(self, index):
        user = self.users_table.cellWidget(index.row(),
index.column()).property('user_obj')
        self.users_info_window = UserInfo(user, self.current_user,
self.dashboard_on_user_edited, self)
        self.users_info_window.exec()
        center_screen(self.users_info_window)
```

ui/window/dashboard/books_tab_widget.py

```
from PySide2 import QtCore, QtWidgets
from PySide2.QtWidgets import QApplication, QHBoxLayout, QLabel, QWidget,
QVBoxLayout, QTableWidget, QPushButton

from logic.database import Database
from logic.user import UserPrivilege, User
from ui.helpers.enhanced_controls import LineEdit
from ui.helpers.helpers import center_screen, get_font_size
from ui.window.book_info import BookInfo
from ui.window.book_wizard_window import BookWizardWindow

class BooksTabWidget(QWidget):

    def __init__(self, current_user: User, parent=None):
        super(BooksTabWidget, self).__init__(parent)

        self.current_user = current_user

        layout = QVBoxLayout()

        button_bar = QHBoxLayout()

        self.add_book_button = QPushButton('New Book')
        self.add_book_button.clicked.connect(self.add_new_book)
```

```
        self.reload_button = QPushButton('Reload')
        self.reload_button.clicked.connect(self.reload_button_clicked)

        button_bar.addWidget(self.add_book_button)
        button_bar.addWidget(self.reload_button)

        self.books_table = QTableWidget()
        self.books_table.clicked.connect(self.books_table_clicked)

        self.search_bar = LineEdit('Search for book')

self.search_bar.line_edit.textEdited.connect(self.search_bar_value_changed
)
        self.search_bar.line_edit.setPlaceholderText('Search by Name,
Author, Genre or ISBN')

        self.get_random_book_button = QPushButton('I\'m feeling lucky!')
        self.get_random_book_button.clicked.connect(self.get_random_book)

        layout.addLayout(button_bar)

        self.books_widget = QWidget()
        books_widget_vbox = QVBoxLayout()
        books_widget_vbox.setContentsMargins(QtCore.QMargins(0,0,0,0))
        books_widget_vbox.addWidget(self.search_bar)
        books_widget_vbox.addWidget(self.get_random_book_button)
        books_widget_vbox.addWidget(self.books_table)
        self.books_widget.setLayout(books_widget_vbox)

        layout.addWidget(self.books_widget)

        self.no_books_widget = QWidget()
        no_books_vbox = QVBoxLayout()
        no_books_vbox.setContentsMargins(QtCore.QMargins(0,0,0,0))
        no_books_vbox.setAlignment(QtCore.Qt.AlignCenter)

        no_books_found_label = QLabel('No books found')
        no_books_found_label.setAlignment(QtCore.Qt.AlignCenter)
        no_books_found_label.setFont(get_font_size(18))

        self.no_books_non_admin_sub_heading_label = QLabel('Ask the
administrator to add some books!')

self.no_books_non_admin_sub_heading_label.setAlignment(QtCore.Qt.AlignCent
er)

        self.no_books_admin_sub_heading_label = QLabel('Click on "New
Book" to add one!')

self.no_books_admin_sub_heading_label.setAlignment(QtCore.Qt.AlignCenter)
```

```
        no_books_vbox.addWidget(no_books_found_label)
        no_books_vbox.addWidget(self.no_books_non_admin_sub_heading_label)
        no_books_vbox.addWidget(self.no_books_admin_sub_heading_label)
        self.no_books_widget.setLayout(no_books_vbox)

        layout.addWidget(self.no_books_widget)

        self.setLayout(layout)
        self.configure_books_table()

        if self.current_user.privilege == UserPrivilege.NORMAL:
            self.add_book_button.hide()


    def reload_button_clicked(self):
        self.reload_button.setDisabled(True)
        QApplication.instance().processEvents()
        self.configure_books_table()
        self.reload_button.setDisabled(False)

    def search_bar_value_changed(self):
        search = self.search_bar.line_edit.text().lower()

        for i in range(self.books_table.rowCount()):
            book = self.books_table.cellWidget(i, 0).property('book_obj')
            if not search in (book.name.lower() + book.author.lower() +
''.join(book.genres) + book.ISBN.lower()):
                self.books_table.hideRow(i)
            else:
                self.books_table.showRow(i)

    def add_new_book(self):
        self.new_book_window =
BookWizardWindow(self.configure_books_table)
        self.new_book_window.exec()
        center_screen(self.new_book_window)

    def configure_books_table(self):
        l_books = Database.get_all_books()

        self.books_table.clear()
        self.books_table.setSortingEnabled(True)
        self.books_table.setRowCount(len(l_books))
        self.books_table.setColumnCount(3)
        self.books_table.setHorizontalHeaderLabels(["Name", "Author",
"Genre"])

        self.books_table.horizontalHeader().setSectionResizeMode(0,
QtWidgets.QHeaderView.Stretch)
```

```
        self.books_table.horizontalHeader().setSectionResizeMode(1,
QtWidgets.QHeaderView.Stretch)
        self.books_table.horizontalHeader().setSectionResizeMode(2,
QtWidgets.QHeaderView.Stretch)


self.books_table.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderVi
ew.Fixed)
        self.books_table.verticalHeader().setDefaultSectionSize(70)

        if len(l_books) == 0:
            self.books_widget.hide()

            if self.current_user.privilege == UserPrivilege.NORMAL:
                self.no_books_non_admin_sub_heading_label.show()
                self.no_books_admin_sub_heading_label.hide()
            else:
                self.no_books_non_admin_sub_heading_label.hide()
                self.no_books_admin_sub_heading_label.show()

            self.no_books_widget.show()
        else:
            self.books_widget.show()
            self.no_books_widget.hide()

        for i in range(len(l_books)):
            each_book = l_books[i]
            name_widget = QLabel(each_book.name)
            author_widget = QLabel(each_book.author)
            genre_widget = QLabel(each_book.get_stylish_genres())

            name_widget.setProperty('book_obj', each_book)
            author_widget.setProperty('book_obj', each_book)
            genre_widget.setProperty('book_obj', each_book)

            self.books_table.setCellWidget(i, 0, name_widget)
            self.books_table.setCellWidget(i, 1, author_widget)
            self.books_table.setCellWidget(i, 2, genre_widget)

    def books_table_clicked(self, index):
        book = self.books_table.cellWidget(index.row(),
index.column()).property('book_obj')
        self.open_book_info(book)

    def get_random_book(self):
        book = Database.get_random_book()
        self.open_book_info(book)

    def open_book_info(self, book):
```

```
        self.book_info_window = BookInfo(book, self.configure_books_table,
self.current_user, self)
        self.book_info_window.exec()
        center_screen(self.book_info_window)
```

ui/window/dashboard/dashboard.py

```
from PySide2.QtWidgets import (QApplication, QVBoxLayout, QWidget,
QTabWidget)
from qt_material import apply_stylesheet, QtStyleTools

from logic.database import Database
from logic.user import User, UserPrivilege
from ui.window.dashboard.admin_users_tab import AdminUsersTab
from ui.window.dashboard.books_tab_widget import BooksTabWidget
from ui.window.dashboard.settings_tab.settings_tab import SettingsTab

class Dashboard(QWidget, QtStyleTools):

    def __init__(self, current_user: User, parent=None):
        super(Dashboard, self).__init__(parent)

        self.current_user = current_user
        self.current_user_account_settings =
Database.get_user_account_settings(self.current_user.username)

        self.configure_window_title()

        self.resize(1024, 768)

        layout = QVBoxLayout()

        self.tabs = QTabWidget()
        self.configure_tabs()
        layout.addWidget(self.tabs)

        self.configure_theme_and_accent_colour()

        self.setLayout(layout)

    def configure_window_title(self):
        self.setWindowTitle(f'Snakebrary - Logged in as
{self.current_user.username} ({self.current_user.name})')

    def configure_tabs(self):

        self.settings_tab = SettingsTab(self.current_user,
self.current_user_account_settings, self.dashboard_on_user_edited)
```

```
        if self.current_user.privilege != UserPrivilege.NORMAL:
            self.admin_users_table = AdminUsersTab(self.current_user,
self.dashboard_on_user_edited)
            self.tabs.addTab(self.admin_users_table, 'Users')

        self.tabs.addTab(BooksTabWidget(self.current_user), 'Books')

        self.tabs.addTab(self.settings_tab, "Settings")

    def configure_theme_and_accent_colour(self):
        stylesheet_name =
f'{self.current_user_account_settings.theme.lower()}_{self.current_user_ac
count_settings.accent_colour.lower().replace(" ", "")}.xml'
        apply_stylesheet(QApplication.instance(), stylesheet_name)

    def dashboard_on_user_edited(self):
        if self.current_user.privilege != UserPrivilege.NORMAL:
            self.admin_users_table.configure_users_table()

        self.current_user =
Database.get_user_by_username(self.current_user.username)
        self.admin_users_table.current_user = self.current_user
        self.settings_tab.account_tab.user_info_vbox.current_user =
self.current_user
        self.settings_tab.account_tab.user_info_vbox.configure_ui()
        self.configure_window_title()
```

ui/window/add_user.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QDialog, QMessageBox

from logic.user import UserPrivilege
from ui.layouts_and_widgets.user_wizard import UserWizard

class AddUser(QDialog):

    def __init__(self, new_user_privilege: UserPrivilege, on_successful,
parent=None):
        super(AddUser, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        if new_user_privilege == UserPrivilege.ADMIN:
            prefix_label = 'Administrator'
        else:
            prefix_label = 'Normal User'
```

```
        self.setWindowTitle(f'Add New {prefix_label}')
        self.resize(800, 600)

        self.setLayout(UserWizard(on_success=self.on_success1,
new_user_privilege=new_user_privilege))
        self.on_successful = on_successful

    def on_success1(self):
        self.on_successful()
        QMessageBox.information(self, 'Congratulations', 'Account was
successfully added!', QMessageBox.Ok)
        self.close()
```

ui/window/book_holders_window.py

```
from ui.window.user_info import UserInfo
from ui.helpers.helpers import center_screen
from PySide2 import QtCore
from PySide2 import QtWidgets
from PySide2.QtWidgets import QDialog, QLabel, QPushButton, QTableWidget,
QVBoxLayout, QWidget

from logic.database import Database

class BookHoldersWindow(QDialog):

    def __init__(self, book_holders, current_user, parent=None):
        super(BookHoldersWindow, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.book_holders = book_holders
        self.current_user = current_user

        self.setWindowTitle("Book Holders")
        self.resize(800, 600)

        self.book_holders_table = QTableWidget()

        vbox = QVBoxLayout()
        vbox.addWidget(self.book_holders_table)

        self.setLayout(vbox)

        self.configure_holders_table()

    def configure_holders_table(self):
```

```
        self.book_holders_table.setSortingEnabled(True)
        self.book_holders_table.setRowCount(len(self.book_holders))
        self.book_holders_table.setColumnCount(5)
        self.book_holders_table.setHorizontalHeaderLabels(["Username",
"Name", "   Issued On   ", "   Returned On   ", "          Action
"])

        self.book_holders_table.horizontalHeader().setSectionResizeMode(0,
QtWidgets.QHeaderView.Stretch)
        self.book_holders_table.horizontalHeader().setSectionResizeMode(1,
QtWidgets.QHeaderView.Stretch)
        self.book_holders_table.horizontalHeader().setSectionResizeMode(2,
QtWidgets.QHeaderView.ResizeToContents)
        self.book_holders_table.horizontalHeader().setSectionResizeMode(3,
QtWidgets.QHeaderView.ResizeToContents)
        self.book_holders_table.horizontalHeader().setSectionResizeMode(4,
QtWidgets.QHeaderView.ResizeToContents)


self.book_holders_table.verticalHeader().setSectionResizeMode(QtWidgets.QH
eaderView.Fixed)
        self.book_holders_table.verticalHeader().setDefaultSectionSize(70)

        for i in range(len(self.book_holders)):
            each_holder = self.book_holders[i]

            user_obj = Database.get_user_by_username(each_holder[0])
            username_widget = QLabel(each_holder[0])
            name_widget = QLabel(user_obj.name)
            issued_on_widget = QLabel(each_holder[1])
            returned_on_widget = QLabel(each_holder[2])

            view_profile_button = QPushButton('View Profile')
            view_profile_button.setProperty('user_obj', user_obj)
            view_profile_button.clicked.connect(self.view_holder_profile)

            vbox = QVBoxLayout()
            vbox.setContentsMargins(QtCore.QMargins(0,0,0,0))
            vbox.addWidget(view_profile_button)


            view_profile_button_widget = QWidget()

view_profile_button_widget.setContentsMargins(QtCore.QMargins(0,0,0,0))
            view_profile_button_widget.setLayout(vbox)

            self.book_holders_table.setCellWidget(i, 0, username_widget)
            self.book_holders_table.setCellWidget(i, 1, name_widget)
            self.book_holders_table.setCellWidget(i, 2, issued_on_widget)
```

```
            self.book_holders_table.setCellWidget(i, 3,
returned_on_widget)
            self.book_holders_table.setCellWidget(i, 4,
view_profile_button_widget)

    def view_holder_profile(self):
        self.users_info_window =
UserInfo(self.sender().property('user_obj'), self.current_user,
                                            self.configure_holders_table,
self, disable_edit_option=True)
        self.users_info_window.exec()
        center_screen(self.users_info_window)
```

ui/window/book_info.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QDialog, QHBoxLayout, QLabel, QMessageBox,
QPushButton, QScrollArea, QVBoxLayout, QWidget

from logic.book import BookHolder
from logic.database import Database
from logic.user import User, UserPrivilege
from ui.helpers.enhanced_controls import ImageView
from ui.helpers.helpers import get_font_size, center_screen
from ui.layouts_and_widgets.book_ratings_widget import BookRatingsWidget
from ui.window.book_holders_window import BookHoldersWindow
from ui.window.book_reviewers_window import BookReviewersWindow
from ui.window.book_wizard_window import BookWizardWindow

class BookInfo(QDialog):

    def __init__(self, book, dashboard_on_books_edited, current_user:
User, parent=None):
        super(BookInfo, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.book = book
        self.dashboard_on_books_edited = dashboard_on_books_edited
        self.current_user = current_user

        self.setWindowTitle("Book Information")
        self.resize(800, 700)

        main_vbox = QVBoxLayout()
        main_vbox.setAlignment(QtCore.Qt.AlignTop)

        hbox_1 = QHBoxLayout()
```

```
        hbox_1.setSpacing(10)

        self.cover_photo = ImageView('Cover Photo', 300, 300)
        hbox_1.addWidget(self.cover_photo)

        self.name_label = QLabel()
        self.name_label.setFont(get_font_size(30))

        self.author_label = QLabel()
        self.author_label.setFont(get_font_size(17))

        self.genres_label = QLabel()

        self.isbn_label = QLabel()
        self.price_label = QLabel()

        self.date_time_added_label = QLabel()
        self.current_holder_label = QLabel()
        size_policy = self.current_holder_label.sizePolicy()
        size_policy.setRetainSizeWhenHidden(True)
        self.current_holder_label.setSizePolicy(size_policy)

        self.get_return_button = QPushButton()

        self.disable_enable_button = QPushButton()

        get_return_disable_enable_button_hbox = QHBoxLayout()

get_return_disable_enable_button_hbox.addWidget(self.get_return_button)

get_return_disable_enable_button_hbox.addWidget(self.disable_enable_button
)

        self.get_book_holders_details = QPushButton('book holders list')

self.get_book_holders_details.clicked.connect(self.show_book_holders_list_
window)

        self.get_book_reviewers_details = QPushButton('book reviewers')

self.get_book_reviewers_details.clicked.connect(self.show_book_reviewers_l
ist_window)

        show_book_holders_reviewers_hbox = QHBoxLayout()

show_book_holders_reviewers_hbox.setContentsMargins(QtCore.QMargins(0, 0,
0, 0))

show_book_holders_reviewers_hbox.addWidget(self.get_book_holders_details)
```

```
show_book_holders_reviewers_hbox.addWidget(self.get_book_reviewers_details
)

        self.edit_book_button = QPushButton('Edit')
        self.edit_book_button.clicked.connect(self.on_edit_button_clicked)

        self.delete_book_button = QPushButton('Delete')
        self.delete_book_button.setProperty('class', 'danger')

self.delete_book_button.clicked.connect(self.on_delete_button_clicked)

        edit_delete_button_hbox = QHBoxLayout()
        edit_delete_button_hbox.setContentsMargins(QtCore.QMargins(0, 0,
0, 0))
        edit_delete_button_hbox.addWidget(self.edit_book_button)
        edit_delete_button_hbox.addWidget(self.delete_book_button)

        non_normal_buttons_vbox = QVBoxLayout()
        non_normal_buttons_vbox.setContentsMargins(QtCore.QMargins(0, 0,
0, 0))

non_normal_buttons_vbox.addLayout(show_book_holders_reviewers_hbox)
        non_normal_buttons_vbox.addLayout(edit_delete_button_hbox)

        self.non_normal_buttons_widget = QWidget()
        self.non_normal_buttons_widget.setLayout(non_normal_buttons_vbox)

        vbox_labels_1 = QVBoxLayout()
        vbox_labels_1.setAlignment(QtCore.Qt.AlignTop)
        vbox_labels_1.addWidget(self.name_label)
        vbox_labels_1.addWidget(self.author_label)
        vbox_labels_1.addWidget(self.genres_label)
        vbox_labels_1.addWidget(self.isbn_label)
        vbox_labels_1.addWidget(self.price_label)
        vbox_labels_1.addWidget(self.date_time_added_label)
        vbox_labels_1.addWidget(self.current_holder_label)
        vbox_labels_1.addLayout(get_return_disable_enable_button_hbox)
        vbox_labels_1.addWidget(self.non_normal_buttons_widget)

        hbox_1.addLayout(vbox_labels_1)

        main_vbox.addLayout(hbox_1)

        self.ratings_widget = BookRatingsWidget(self.book,
self.current_user)
        self.setContentsMargins(QtCore.QMargins(0, 0, 0, 0))

        main_vbox.addWidget(self.ratings_widget)
```

```
        self.about_label_header = QLabel('About')
        self.about_label_header.setContentsMargins(QtCore.QMargins(0, 10,
0, 0))
        self.about_label_header.setFont(get_font_size(18))

        self.about_label = QLabel()
        self.about_label.setAlignment(QtCore.Qt.AlignJustify)
        self.about_label.setWordWrap(True)

        self.about_label_scroll_area = QScrollArea()
        self.about_label_scroll_area.setWidgetResizable(True)

self.about_label_scroll_area.setHorizontalScrollBarPolicy(QtCore.Qt.Scroll
BarAlwaysOff)
        self.about_label_scroll_area.setWidget(self.about_label)

        about_layout = QVBoxLayout()
        about_layout.addWidget(self.about_label_header)
        about_layout.addWidget(self.about_label_scroll_area)

        self.about_widget = QWidget()
        self.about_widget.setLayout(about_layout)

        main_vbox.addWidget(self.about_widget)

        self.setLayout(main_vbox)
        self.configure_ui()

    def configure_ui(self):
        if self.book.photo == None:
            self.cover_photo.clear_image()
            self.cover_photo.hide()
        else:
            self.cover_photo.set_image_from_blob(self.book.photo)
            self.cover_photo.show()

        self.name_label.setText(self.book.name)
        self.author_label.setText(f'by {self.book.author}')

        if len(self.book.genres) == 1:
            self.genres_label.setText('Genre: ' +
self.book.get_stylish_genres())
        else:
            self.genres_label.setText('Genres: ' +
self.book.get_stylish_genres())

        self.isbn_label.setText(f'ISBN: {self.book.ISBN}')
        self.price_label.setText(f'Price: ₹ {self.book.price}')

        if self.book.about != '':
```

```
            self.about_label.setText(self.book.about)
            self.about_widget.show()
        else:
            self.about_widget.hide()

        self.configure_get_return_button()
        self.configure_disable_enable_button()

        if self.current_user.privilege == UserPrivilege.NORMAL:
            self.current_holder_label.hide()
            self.date_time_added_label.hide()
            self.non_normal_buttons_widget.hide()
            self.disable_enable_button.hide()
            self.current_holder_label.hide()
        else:
            self.date_time_added_label.setText(f'Date/Time added:
{self.book.date_time_added}')
            self.configure_current_holder_label()

        self.ratings_widget.reload(self.book)

    def on_delete_button_clicked(self):
        warning_box = QMessageBox.warning(self, 'Warning', f'''Are you
sure you want to delete the following book
Name: {self.book.name}
Author: {self.book.author}
ISBN: {self.book.ISBN}
Date Time Added: {self.book.date_time_added}''', QMessageBox.Yes,
QMessageBox.No)

        if warning_box == QMessageBox.Yes:
            Database.delete_book(self.book.ISBN)
            self.dashboard_on_books_edited()
            self.close()

    def get_book(self):
        self.get_return_button.setDisabled(True)

        new_holder = BookHolder(self.current_user.username)
        self.book.holders.append(new_holder.get_raw_list())
        Database.update_book_holders(self.book.holders, self.book.ISBN)

        self.configure_get_return_button()
        self.configure_disable_enable_button()
        self.configure_current_holder_label()

    def configure_current_holder_label(self):
        if self.current_user.privilege == UserPrivilege.NORMAL:
            return
```

```
        current_holder = self.book.get_current_holder()
        if current_holder == None:
            self.current_holder_label.hide()
        else:
            current_holder_user =
Database.get_user_by_username(current_holder)
            self.current_holder_label.setText(f'Current Holder:
{current_holder} ({current_holder_user.name})')
            self.current_holder_label.show()

    def return_book(self):

        self.get_return_button.setDisabled(True)
        self.book.return_now()
        Database.update_book_holders(self.book.holders, self.book.ISBN)

        self.configure_get_return_button()
        self.configure_disable_enable_button()
        self.ratings_widget.reload(self.book)
        self.configure_current_holder_label()

    def configure_get_return_button(self):
        self.disconnect_slots_get_return_button()


        if self.book.get_current_holder() == None:
            if self.book.is_unavailable:
                self.get_return_button.setDisabled(True)
                self.get_return_button.setText('Unavailable')
            else:
                self.get_return_button.setDisabled(False)
                self.get_return_button.setText('Get it')
                self.get_return_button.clicked.connect(self.get_book)
        else:
            current_holder_privilege =
Database.get_user_by_username(self.book.get_current_holder()).privilege
            if self.book.get_current_holder() ==
self.current_user.username or (self.current_user.privilege !=
UserPrivilege.NORMAL and current_holder_privilege !=
self.current_user.privilege and current_holder_privilege !=
UserPrivilege.MASTER):
                self.get_return_button.setDisabled(False)
                self.get_return_button.setText('Return')
                self.get_return_button.clicked.connect(self.return_book)
            else:
                self.get_return_button.setDisabled(True)
                self.get_return_button.setText('Unavailable')

    def disconnect_slots_get_return_button(self):
        try:
```

```
            self.get_return_button.clicked.disconnect()
        except:
            pass

    def configure_disable_enable_button(self):
        if self.current_user.privilege == UserPrivilege.NORMAL:
            return

        self.disconnect_slots_disable_enable_button()

        if self.book.get_current_holder() == None:
            if self.book.is_unavailable:
                self.disable_enable_button.show()
                self.disable_enable_button.setText('Enable')
                self.disable_enable_button.clicked.connect(lambda:
self.make_book_unavailable(False))
            else:
                self.disable_enable_button.show()
                self.disable_enable_button.setText('Disable')
                self.disable_enable_button.clicked.connect(lambda:
self.make_book_unavailable(True))
        else:
            self.disable_enable_button.hide()
            self.disable_enable_button.setText('Unavailable')

    def make_book_unavailable(self, is_unavailable):
        self.book.is_unavailable = is_unavailable
        Database.update_book(self.book)

        self.configure_get_return_button()
        self.configure_disable_enable_button()


    def disconnect_slots_disable_enable_button(self):
        try:
            self.disable_enable_button.clicked.disconnect()
        except:
            pass

    def show_book_holders_list_window(self):
        self.book_holders_list_window =
BookHoldersWindow(self.book.holders, self.current_user, self)
        self.book_holders_list_window.exec()
        center_screen(self.book_holders_list_window)

    def on_edit_button_clicked(self):
        self.book_wizard_window = BookWizardWindow(self.on_book_edited,
self.book, self)
        self.book_wizard_window.exec()
        center_screen(self.book_wizard_window)
```

```
    def on_book_edited(self):
        self.dashboard_on_books_edited()
        self.book = Database.get_book_by_ISBN(self.book.ISBN)
        self.configure_ui()

    def show_book_reviewers_list_window(self):
        self.book_reviewers_list_window = BookReviewersWindow(self.book,
self.current_user, self.configure_ui, self)
        self.book_reviewers_list_window.exec()
        center_screen(self.book_reviewers_list_window)
```

ui/window/book_reviewers_window.py

```
from ui.helpers.helpers import center_screen
from ui.window.user_info import UserInfo
from logic.user import User, UserPrivilege
from PySide2 import QtCore
from PySide2 import QtWidgets
from PySide2.QtWidgets import QApplication, QDialog, QHBoxLayout, QLabel,
QPushButton, QTableWidget, QVBoxLayout, \
    QWidget

from logic.database import Database

class BookReviewersWindow(QDialog):

    def __init__(self, book, current_user, on_review_deleted, parent):
        super(BookReviewersWindow, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.book = book
        self.current_user = current_user
        self.on_review_deleted = on_review_deleted

        self.setWindowTitle("Book Reviewers")
        self.resize(800, 600)

        self.book_reviewers_table = QTableWidget()

        vbox = QVBoxLayout()
        vbox.addWidget(self.book_reviewers_table)

        self.setLayout(vbox)

        self.configure_reviewers_table()
```

```
    def configure_reviewers_table(self):
        self.book_ratings = Database.get_book_ratings(self.book.ISBN)

        self.book_reviewers_table.clear()
        self.book_reviewers_table.setSortingEnabled(True)

self.book_reviewers_table.setRowCount(len(self.book_ratings.ratings))
        self.book_reviewers_table.setColumnCount(4)
        self.book_reviewers_table.setHorizontalHeaderLabels(["Username",
"Name", "Rating", "                              Actions
"])


self.book_reviewers_table.horizontalHeader().setSectionResizeMode(0,
QtWidgets.QHeaderView.Stretch)

self.book_reviewers_table.horizontalHeader().setSectionResizeMode(1,
QtWidgets.QHeaderView.Stretch)

self.book_reviewers_table.horizontalHeader().setSectionResizeMode(2,
QtWidgets.QHeaderView.ResizeToContents)

self.book_reviewers_table.horizontalHeader().setSectionResizeMode(3,
QtWidgets.QHeaderView.ResizeToContents)


self.book_reviewers_table.verticalHeader().setSectionResizeMode(QtWidgets.
QHeaderView.Fixed)

self.book_reviewers_table.verticalHeader().setDefaultSectionSize(70)

        i = 0
        for each_reviewer in self.book_ratings.ratings.keys():
            username_widget = QLabel(each_reviewer)
            each_reviewer_user_obj =
Database.get_user_by_username(each_reviewer)
            name_widget = QLabel(each_reviewer_user_obj.name)
            rating_widget =
QLabel(str(self.book_ratings.ratings[each_reviewer]))

            delete_button = QPushButton('Delete')
            delete_button.setProperty('class', 'danger')
            delete_button.setProperty('username', each_reviewer)
            delete_button.clicked.connect(self.delete_rating)

            view_profile_button = QPushButton('View Profile')
            view_profile_button.setProperty('user_obj',
each_reviewer_user_obj)

view_profile_button.clicked.connect(self.view_reviewer_profile)
```

```
            hbox = QHBoxLayout()
            hbox.setContentsMargins(QtCore.QMargins(0,0,0,0))
            hbox.addWidget(view_profile_button)
            if (self.current_user.privilege !=
each_reviewer_user_obj.privilege and each_reviewer_user_obj.privilege !=
UserPrivilege.MASTER) or self.current_user.username == each_reviewer:
                hbox.addWidget(delete_button)


            view_profile_delete_button_widget = QWidget()

view_profile_delete_button_widget.setContentsMargins(QtCore.QMargins(0,0,0
,0))
            view_profile_delete_button_widget.setLayout(hbox)

            self.book_reviewers_table.setCellWidget(i, 0, username_widget)
            self.book_reviewers_table.setCellWidget(i, 1, name_widget)
            self.book_reviewers_table.setCellWidget(i, 2, rating_widget)
            self.book_reviewers_table.setCellWidget(i, 3,
view_profile_delete_button_widget)

            i += 1

    def view_reviewer_profile(self):
        self.users_info_window =
UserInfo(self.sender().property('user_obj'), self.current_user,
                                         self.configure_reviewers_table,
self, True)
        self.users_info_window.exec()
        center_screen(self.users_info_window)

    def delete_rating(self):
        self.book_ratings.ratings.pop(self.sender().property('username'),
None)
        Database.update_book_ratings(self.book_ratings)
        self.configure_reviewers_table()
        self.on_review_deleted()
```

ui/window/book_wizard_window.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QApplication, QDialog, QHBoxLayout,
QMessageBox, QVBoxLayout, QPushButton

from logic.book import Book
```

```
from logic.database import Database
from ui.helpers.enhanced_controls import FilePicker, ImageView, LineEdit,
PlainTextEdit

class BookWizardWindowMode:
    ADD = 1,
    EDIT = 2

class BookWizardWindow(QDialog):

    def __init__(self, on_success, old_book=None, parent=None):
        super(BookWizardWindow, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.resize(700, 500)

        self.on_success = on_success

        self.new_book_cover_photo_path_field = FilePicker('Cover Picture
(Optional)',

on_select=self.on_cover_photo_selected,

on_clear=self.on_cover_photo_cleared)

        self.new_book_cover_photo_preview = ImageView('Preview will appear
here', 300, 300)

        self.photo_hbox = QHBoxLayout()
        self.photo_hbox.addWidget(self.new_book_cover_photo_path_field)
        self.photo_hbox.addWidget(self.new_book_cover_photo_preview)

        self.new_book_name_field = LineEdit('Name')
        self.new_book_author_field = LineEdit('Author')
        self.new_book_isbn_field = LineEdit('ISBN')
        self.new_book_genres_field = LineEdit('Genres (Seperate with
comma)')
        self.new_book_price_field = LineEdit('Price (₹)')
        self.new_book_about_field = PlainTextEdit('About (Optional)')

        self.proceed_button = QPushButton('Proceed')

self.proceed_button.clicked.connect(self.on_proceed_button_clicked)

        # Create layout and add widgets

        vbox = QVBoxLayout()

        vbox.addLayout(self.photo_hbox)
```

```
        vbox.addWidget(self.new_book_name_field)
        vbox.addWidget(self.new_book_author_field)
        vbox.addWidget(self.new_book_isbn_field)
        vbox.addWidget(self.new_book_genres_field)
        vbox.addWidget(self.new_book_price_field)
        vbox.addWidget(self.new_book_about_field)
        vbox.addWidget(self.proceed_button)

        self.setLayout(vbox)

        if old_book == None:
            self.setWindowTitle('Add Book')
            self.mode = BookWizardWindowMode.ADD
        else:
            self.setWindowTitle('Edit Book')
            self.old_book = old_book
            self.load_values_for_old_book()
            self.mode = BookWizardWindowMode.EDIT
            self.new_book_isbn_field.line_edit.setReadOnly(True)

    def load_values_for_old_book(self):
        if self.old_book.photo != None:

self.new_book_cover_photo_preview.set_image_from_blob(self.old_book.photo)

        self.new_book_name_field.line_edit.setText(self.old_book.name)
        self.new_book_author_field.line_edit.setText(self.old_book.author)
        self.new_book_isbn_field.line_edit.setText(self.old_book.ISBN)
        self.new_book_genres_field.line_edit.setText((',
'.join(self.old_book.genres)))

self.new_book_price_field.line_edit.setText(str(self.old_book.price))

self.new_book_about_field.plain_text_edit.setPlainText(self.old_book.about
)

    def on_cover_photo_selected(self, img_path):
        self.new_book_cover_photo_preview.set_image_from_path(img_path)

    def on_cover_photo_cleared(self):
        self.new_book_cover_photo_path_field.line_edit.clear()
        self.new_book_cover_photo_preview.clear_image()

    def on_proceed_button_clicked(self):
        proposed_new_book_cover_photo_path =
self.new_book_cover_photo_path_field.line_edit.text()
        proposed_new_book_name = self.new_book_name_field.line_edit.text()
        proposed_new_book_author =
self.new_book_author_field.line_edit.text()
        proposed_new_book_isbn = self.new_book_isbn_field.line_edit.text()
```

```
        proposed_new_book_genres =
self.new_book_genres_field.line_edit.text()
        proposed_new_book_price =
self.new_book_price_field.line_edit.text()
        proposed_new_book_about =
self.new_book_about_field.plain_text_edit.toPlainText()

        error = False

        if len(proposed_new_book_name) < 1:
            self.new_book_name_field.on_error('Required')
            error = True
        else:
            self.new_book_name_field.on_success()

        if len(proposed_new_book_author) < 1:
            self.new_book_author_field.on_error('Required')
            error = True
        else:
            self.new_book_author_field.on_success()

        if len(proposed_new_book_isbn) > 13 or len(proposed_new_book_isbn)
< 1:
            self.new_book_isbn_field.on_error('Invalid ISBN!')
            error = True
        else:
            self.new_book_isbn_field.on_success()

        if len(proposed_new_book_genres) < 1:
            self.new_book_genres_field.on_error('Required')
            error = True
        else:
            self.new_book_genres_field.on_success()

        try:
            float(proposed_new_book_price)
            self.new_book_price_field.on_success()
        except ValueError:
            self.new_book_price_field.on_error('Invalid price!')
            error = True

        if error:
            return

        self.set_disable(True)

        genres = proposed_new_book_genres.split(',')
        for i in range(len(genres)):
            genres[i] = genres[i].strip().lower()
```

```
        new_book = Book(proposed_new_book_isbn, proposed_new_book_name,
                        proposed_new_book_author, [], genres,
proposed_new_book_price,
                        proposed_new_book_about)

        if proposed_new_book_cover_photo_path != '':
            file = open(proposed_new_book_cover_photo_path, 'rb')
            new_book.photo = file.read()
            file.close()
        else:
            if self.mode == BookWizardWindowMode.EDIT and
self.new_book_cover_photo_preview.is_clear == False:
                new_book.photo = self.old_book.photo

        if self.mode == BookWizardWindowMode.ADD:
            old_book = Database.get_book_by_ISBN(proposed_new_book_isbn)
            if old_book != None:
                QMessageBox.critical(None, 'Error', f'''Book with same
ISBN already exists.
Name: {old_book.name}
Author: {old_book.author}
Price: {old_book.price}''', QMessageBox.Ok)
                self.set_disable(False)
                return
            Database.create_new_book(new_book)

            close_message = 'Book was successfully added!'
        else:
            Database.update_book(new_book)
            close_message = 'Book was successfully edited!'

        self.on_success()
        QMessageBox.information(self, 'Congratulations', close_message,
QMessageBox.Ok)
        self.close()

    def set_disable(self, disable):
        self.proceed_button.setDisabled(disable)
        self.new_book_isbn_field.line_edit.setReadOnly(disable)
        self.new_book_name_field.line_edit.setReadOnly(disable)
        self.new_book_author_field.line_edit.setReadOnly(disable)
        self.new_book_genres_field.line_edit.setReadOnly(disable)
        self.new_book_price_field.line_edit.setReadOnly(disable)
        self.new_book_about_field.plain_text_edit.setReadOnly(disable)
        QApplication.instance().processEvents()
```

ui/window/connection_details_widget.py

```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QApplication, QPushButton, QVBoxLayout,
QWidget, QLabel, QCheckBox

from logic.database import Database
from ui.helpers.enhanced_controls import LineEdit
from ui.helpers.helpers import get_font_size

class ConnectionDetailsWidget(QWidget):

    def __init__(self, on_success=None, parent=None):
        super(ConnectionDetailsWidget, self).__init__(parent)

        self.setWindowTitle('SnakeBrary')
        self.on_success = on_success
        self.setFixedSize(420, 480)

        layout = QVBoxLayout()

        heading = QLabel('Connect to MySQL Server')
        heading.setAlignment(Qt.AlignCenter)
        heading.setFont(get_font_size(20))
        layout.addWidget(heading)

        sub_heading = QLabel('to use SnakeBrary')
        sub_heading.setAlignment(Qt.AlignCenter)
        sub_heading.setFont(get_font_size(15))
        layout.addWidget(sub_heading)

        self.host_field = LineEdit('Host')
        self.port_field = LineEdit('Port')
        self.user_field = LineEdit('User')
        self.password_field = LineEdit('Password', password_mode=True)

        self.remember_me_checkbox = QCheckBox('Remember Connection
Settings')
        self.remember_me_checkbox.setChecked(True)

        self.connect_server_button = QPushButton('Connect')

self.connect_server_button.clicked.connect(self.connect_server_button_clic
ked)

        self.error_label = QLabel()
        self.error_label.setWordWrap(True)
        self.error_label.setStyleSheet("color: red;")
        self.error_label.setAlignment(Qt.AlignCenter)

        # Create layout and add widgets
        layout.addWidget(self.host_field)
```

```
        layout.addWidget(self.port_field)
        layout.addWidget(self.user_field)
        layout.addWidget(self.password_field)
        layout.addWidget(self.remember_me_checkbox)
        layout.addWidget(self.error_label)
        layout.addWidget(self.connect_server_button)

        layout.setSpacing(10)

        # Set dialog layout
        self.setLayout(layout)

        self.get_local_saved_settings()

    def get_local_saved_settings(self):

self.host_field.line_edit.setText(Database.get_local_database_server_host(
))

self.user_field.line_edit.setText(Database.get_local_database_server_user(
))

self.port_field.line_edit.setText(Database.get_local_database_server_port(
))

self.password_field.line_edit.setText(Database.get_local_database_server_p
assword())

    def connect_server_button_clicked(self):

        self.disable_prompt(True)

        host = self.host_field.line_edit.text()
        port = self.port_field.line_edit.text()
        user = self.user_field.line_edit.text()
        password = self.password_field.line_edit.text()

        error = False

        if len(host) < 1:
            self.host_field.on_error('Invalid host')
            error = True
        else:
            self.host_field.on_success()

        if not port.isnumeric():
            self.port_field.on_error('Invalid port')
            error = True
        else:
            self.port_field.on_success()
```

```
        if error:
            self.error_label.setText('')
            self.disable_prompt(False)
            return

        QApplication.instance().processEvents()

        try:
            Database.create_connection(host, user, password, port)

            if Database.is_new_local_setup():
                Database.create_local_database_settings_table()

            if self.remember_me_checkbox.isChecked():
                Database.set_local_connection_settings(host, port, user,
password)
            else:
                Database.clear_local_connection_settings()

            Database.save_local_database()

            self.error_label.setText('')

            if self.on_success != None:
                self.x = self.on_success()

            self.close()
        except Exception as e:
            self.error_label.setText(str(e))

        self.disable_prompt(False)

    def disable_prompt(self, status):
        self.host_field.line_edit.setReadOnly(status)
        self.port_field.line_edit.setReadOnly(status)
        self.user_field.line_edit.setReadOnly(status)
        self.password_field.line_edit.setReadOnly(status)
        self.connect_server_button.setDisabled(status)
        QApplication.instance().processEvents()
```

ui/window/edit_user.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QDialog, QMessageBox

from ui.layouts_and_widgets.user_wizard import UserWizard
```

```
class EditUser(QDialog):

    def __init__(self, user, on_successful, parent=None):
        super(EditUser, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.setWindowTitle('Edit User')
        self.resize(800, 600)

        self.setLayout(UserWizard(on_success=self.on_success1,
old_user=user))
        self.on_successful = on_successful

    def on_success1(self):
        self.on_successful()
        QMessageBox.information(self, 'Congratulations', 'Account was
successfully edited!', QMessageBox.Ok)
        self.close()
```

ui/window/license.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QDialog, QPlainTextEdit, QPushButton,
QVBoxLayout

class License(QDialog):

    def __init__(self, parent):
        super(License, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.setWindowTitle('SnakeBrary - License')
        self.resize(450,500)

        layout = QVBoxLayout()

        license_plain_text_edit = QPlainTextEdit()
        license_plain_text_edit.setReadOnly(True)
        license_plain_text_edit.setPlainText(open('LICENSE').read())

        layout.addWidget(license_plain_text_edit)

        close_button = QPushButton('Close')
        close_button.clicked.connect(self.close)

        layout.addWidget(close_button)
```

```
        self.setLayout(layout)
```

ui/window/login_prompt.py

```
from PySide2 import QtCore
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QApplication, QGraphicsColorizeEffect,
QWidget, QVBoxLayout, QLabel, QPushButton, \
    QMessageBox

from logic.database import Database
from logic.user import UserPrivilege
from ui.helpers.enhanced_controls import LineEdit
from ui.helpers.helpers import get_font_size, center_screen
from ui.window.connection_details_widget import ConnectionDetailsWidget
from ui.window.dashboard.dashboard import Dashboard


class LoginPrompt(QWidget):

    def __init__(self, parent=None):
        super(LoginPrompt, self).__init__(parent)

        self.setWindowTitle('SnakeBrary')
        self.setFixedSize(400, 420)

        heading = QLabel('Sign in')
        heading.setAlignment(Qt.AlignCenter)
        heading.setFont(get_font_size(30))

        sub_heading = QLabel('to continue to SnakeBrary')
        sub_heading.setAlignment(Qt.AlignCenter)
        sub_heading.setFont(get_font_size(15))

        # Create layout and add widgets
        layout = QVBoxLayout()
        layout.addWidget(heading)
        layout.addWidget(sub_heading)

        self.username_field = LineEdit('Username')
        self.password_field = LineEdit('Password', password_mode=True)

        self.login_button = QPushButton('Login')
        self.login_button.clicked.connect(self.on_login_button_click)

        self.forgot_password_button = QPushButton('Forgot Password')
```

```
self.forgot_password_button.clicked.connect(self.on_forgot_password_button
_click)
        self.forgot_password_button.setProperty('class', 'danger')

        self.sql_server_settings_button = QPushButton('MySQL Settings')

self.sql_server_settings_button.clicked.connect(self.sql_server_settings_b
utton_clicked)

        self.error_label = QLabel()
        self.error_label.setAlignment(Qt.AlignCenter)

        # Create layout and add widgets
        layout.addWidget(self.username_field)
        layout.addWidget(self.password_field)
        layout.addWidget(self.error_label)
        layout.addWidget(self.login_button)
        layout.addWidget(self.forgot_password_button)
        layout.addWidget(self.sql_server_settings_button)

        layout.setSpacing(10)

        # Set dialog layout
        self.setLayout(layout)

    def sql_server_settings_button_clicked(self):
        Database.close_connection()
        self.connection_details =
ConnectionDetailsWidget(self.on_connection_configure_success)
        self.connection_details.show()
        center_screen(self.connection_details)
        self.close()

    def on_connection_configure_success(self):
        self.login_prompt = LoginPrompt()
        self.login_prompt.show()
        center_screen(self.login_prompt)

    def on_login_button_click(self):
        self.disable_prompt(True)

        self.username_field.on_success()

        try_username = self.username_field.line_edit.text()
        try_password = self.password_field.line_edit.text()

        user = Database.get_user_by_username(try_username)

        if user == None:
```

```
            self.set_error("Invalid username/password")
            self.disable_prompt(False)
            return

        if user.password != try_password:
            self.set_error("Invalid username/password")
            self.disable_prompt(False)
            return

        if user.is_disabled:
            self.set_error('Account disabled. Contact administrator.')
            self.disable_prompt(False)
            return


        self.set_success('Successfully Logged in!')
        self.dash = Dashboard(user)
        self.dash.show()
        center_screen(self.dash)
        self.close()

    def on_forgot_password_button_click(self):
        try_username = self.username_field.line_edit.text()

        if len(try_username) < 1:
            self.username_field.on_error('Empty username!')
            return

        self.username_field.on_success()

        user = Database.get_user_by_username(try_username)

        if user == None:
            msg_text = 'No user with the provided username was found.
Contact administrator.'
        else:
            hint = user.password_hint

            if hint == '':
                msg_text = 'Your account has no password hint.'
            else:
                msg_text = f'Your password hint is:\n{hint}\n'

            if user.privilege == UserPrivilege.NORMAL:
                msg_text += '\nContact administrator for further help.'
            elif user.privilege == UserPrivilege.ADMIN:
                msg_text += '\nContact master administrator for further
help.'
            else:
```

```
                msg_text += '\nThis account cannot be recovered if
password is forgotten.'

        QMessageBox.warning(self, 'Warning', msg_text, QMessageBox.Ok)

    def set_error(self, error):
        self.error_label.setText(error)
        self.error_label.setStyleSheet("color: red;")
        QApplication.instance().processEvents()

    def set_success(self, error):
        self.error_label.setText(error)
        self.error_label.setStyleSheet("color: green;")
        QApplication.instance().processEvents()

    def disable_prompt(self, disable):
        self.username_field.line_edit.setReadOnly(disable)
        self.password_field.line_edit.setReadOnly(disable)
        self.login_button.setDisabled(disable)
        self.forgot_password_button.setDisabled(disable)
        QApplication.instance().processEvents()
```

ui/window/user_info.py

```
from PySide2 import QtCore
from PySide2.QtWidgets import QDialog

from logic.user import User
from ui.layouts_and_widgets.user_info_vbox import UserInfoVBox

class UserInfo(QDialog):

    def __init__(self, user: User, current_user: User,
dashboard_on_user_edited=None, parent=None,
                 disable_edit_option=False):
        super(UserInfo, self).__init__(parent)

        self.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint)

        self.setWindowTitle("User Information")
        self.setFixedHeight(320)

        self.user_info_vbox = UserInfoVBox(user, current_user,
dashboard_on_user_edited, self,

disable_edit_options=disable_edit_option)
        self.setLayout(self.user_info_vbox)
```

ui/window/welcome.py

```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QVBoxLayout, QWidget, QLabel, QMessageBox

from logic.user import UserPrivilege
from ui.helpers.helpers import get_font_size, center_screen
from ui.layouts_and_widgets.user_wizard import UserWizard
from ui.window.login_prompt import LoginPrompt

class Welcome(QWidget):

    def __init__(self):
        super(Welcome, self).__init__(None)

        self.setWindowTitle('Welcome')
        self.resize(800, 600)

        heading = QLabel('Welcome to SnakeBrary!')
        heading.setAlignment(Qt.AlignCenter)
        heading.setFont(get_font_size(30))

        sub_heading_1 = QLabel('<i>A Sweet and Simple Library Management
System</i>')
        sub_heading_1.setAlignment(Qt.AlignCenter)
        sub_heading_1.setFont(get_font_size(13))
        sub_heading_1.setStyleSheet('padding-bottom: 20')

        sub_heading_2 = QLabel('Fill the form below to create Master
account and get started!')
        sub_heading_2.setAlignment(Qt.AlignCenter)
        sub_heading_2.setFont(get_font_size(15))

        # Create layout and add widgets
        layout = QVBoxLayout()
        layout.addWidget(heading)
        layout.addWidget(sub_heading_1)
        layout.addWidget(sub_heading_2)

        master_user_layout = UserWizard(on_success=self.on_success,
new_user_privilege=UserPrivilege.MASTER)

        layout.addLayout(master_user_layout)

        layout.setSpacing(10)

        self.setLayout(layout)
```

```
        self.login_prompt = LoginPrompt()

    def on_success(self):
        QMessageBox.information(self, 'Congratulations', 'SnakeBrary is
now all set. You may now login as your master '
                                                         'account and
start adding books, create new administrators, '
                                                         'users, etc.',
                                QMessageBox.Ok)

        self.login_prompt.show()
        center_screen(self.login_prompt)
        self.close()
```

logic/book.py

```
from datetime import datetime

class BookRatings:
    def __init__(self, ISBN, ratings):
        self.ISBN = ISBN
        self.ratings = ratings

    def get_average_rating(self):
        if len(self.ratings) == 0:
            return 0.0

        s = 0.0
        for each_rating in self.ratings.values():
            s += each_rating

        return round(s / len(self.ratings), 1)

    def get_rating_by_username(self, username):
        if username in self.ratings:
            return self.ratings[username]
        return None

    def set_rating_by_username(self, username, rating):
        self.ratings[username] = rating

    def delete_rating_by_username(self, username):
        del self.ratings[username]

    def get_ratings_by_proportion(self, rating):
```

```
        return (self.get_total_ratings_for_particular_rating(rating) /
len(self.ratings)) * 100

    def get_total_ratings_for_particular_rating(self, rating):
        c = 0
        for each_rating in self.ratings.values():
            if each_rating == rating:
                c += 1

        return c

class BookHolder:
    def __init__(self, username, issued_on=None, returned_on=None):
        self.username = username
        self.returned_on = returned_on

        if issued_on == None:
            issued_on = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

        self.issued_on = issued_on

    def get_raw_list(self):
        return [self.username, self.issued_on, self.returned_on]

    @staticmethod
    def from_list(raw_list):
        return BookHolder(raw_list[0], raw_list[1], raw_list[2])

class Book:
    def __init__(self, ISBN, name, author, holders, genres, price, about,
is_unavailable=False, photo=None,
                 date_time_added=None):
        self.ISBN = ISBN
        self.name = name
        self.author = author
        self.holders = holders
        self.genres = genres
        self.price = price
        self.about = about
        self.is_unavailable = is_unavailable
        self.photo = photo

        if date_time_added == None:
            date_time_added = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

        self.date_time_added = date_time_added

    def is_eligible_to_rate(self, username):
        for each_holder in self.holders:
            if each_holder[0] == username and each_holder[2] != None:
```

```
                return True

        return False

    def get_stylish_genres(self):
        l = len(self.genres)
        g = ''
        for i in range(l):
            g += self.genres[i].capitalize()
            if i < (l - 1):
                g += ', '
        return g

    def return_now(self):
        self.holders[-1][2] = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

    def get_current_holder(self):
        if len(self.holders) > 0:
            last_holder = self.holders[-1]
            if last_holder[2] == None:
                return last_holder[0]

        return None
```

logic/database.py

```
import sqlite3
from ast import literal_eval
from pathlib import Path

import mysql.connector
from mysql.connector.connection import MySQLConnection
from mysql.connector.cursor import MySQLCursorBuffered

from logic.book import Book, BookRatings
from logic.user import User, UserSettings

class Database:
    __db_con: MySQLConnection
    __db_con_cursor: MySQLCursorBuffered

    __local_db_con: sqlite3.dbapi2
    __local_db_con_cursor: sqlite3.Cursor

    @staticmethod
    def is_connected():
        global __db_con_cursor
```

```
        try:
            return __db_con_cursor != None
        except:
            return False

    @staticmethod
    def get_local_database_location():
        return str(Path.home()) + "/snakebrary.db"

    @staticmethod
    def create_connection(host, user, password, port):
        global __db_con
        global __db_con_cursor

        __db_con = mysql.connector.connect(host=host, user=user,
password=password, port=int(port))
        __db_con.cursor().execute('create database if not exists
snakebrary')
        __db_con.cmd_init_db('snakebrary')
        __db_con_cursor = __db_con.cursor(buffered=True)

    @staticmethod
    def create_local_connection():
        global __local_db_con
        global __local_db_con_cursor

        __local_db_con =
sqlite3.connect(Database.get_local_database_location())
        __local_db_con_cursor = __local_db_con.cursor()

    @staticmethod
    def close_connection():
        global __db_con
        global __db_con_cursor

        if Database.is_connected():
            __db_con_cursor.close()
            __db_con.close()
            __db_con_cursor = None
            __db_con = None

    @staticmethod
    def close_local_connection():
        global __local_db_con
        global __local_db_con_cursor
        __local_db_con_cursor.close()
        __local_db_con.close()

    @staticmethod
    def create_local_database_settings_table():
```

```
        global __local_db_con_cursor
        __local_db_con_cursor.execute('''CREATE TABLE local_settings
        (key   TEXT  PRIMARY KEY  NOT NULL,
        value    TEXT    NOT NULL)''')

    @staticmethod
    def set_local_setting(key, value):
        global __local_db_con_cursor
        __local_db_con_cursor.execute(f'''INSERT OR REPLACE INTO
local_settings(key, value)
            VALUES ("{key}", "{value}");''')

    @staticmethod
    def get_local_setting(key):
        global __local_db_con_cursor
        try:
            return list(__local_db_con_cursor.execute(f'SELECT * FROM
local_settings WHERE key="{key}"'))[0][1]
        except:
            return None

    @staticmethod
    def get_local_database_server_host():
        return Database.get_local_setting('server_host')

    @staticmethod
    def get_local_database_server_user():
        return Database.get_local_setting('server_user')

    @staticmethod
    def get_local_database_server_password():
        return Database.get_local_setting('server_password')

    @staticmethod
    def get_local_database_server_port():
        return Database.get_local_setting('server_port')

    @staticmethod
    def set_local_database_server_host(host):
        Database.set_local_setting('server_host', host)

    @staticmethod
    def set_local_database_server_user(user):
        Database.set_local_setting('server_user', user)

    @staticmethod
    def set_local_database_server_password(password):
        Database.set_local_setting('server_password', password)

    @staticmethod
```

```
def set_local_database_server_port(port):
    Database.set_local_setting('server_port', port)

@staticmethod
def create_new_tables():
    Database.create_new_users_table()
    Database.create_new_account_settings_table()
    Database.create_new_books_table()
    Database.create_new_books_ratings_table()

@staticmethod
def create_new_users_table():
    global __db_con_cursor
    __db_con_cursor.execute('''CREATE TABLE users
    (username   VARCHAR(50)  PRIMARY KEY  NOT NULL,
    password    TEXT    NOT NULL,
    password_hint   TEXT    NOT NULL,
    name    TEXT    NOT NULL,
    is_disabled BOOLEAN,
    privilege   INT NOT NULL,
    photo   LONGBLOB,
    date_time_created    TEXT    NOT NULL);''')

@staticmethod
def create_new_account_settings_table():
    global __db_con_cursor
    __db_con_cursor.execute('''CREATE TABLE account_settings
    (username   VARCHAR(50)  PRIMARY KEY  NOT NULL,
    theme    TEXT    NOT NULL,
    accent_colour   TEXT    NOT NULL);''')

@staticmethod
def create_new_books_table():
    global __db_con_cursor
    __db_con_cursor.execute('''CREATE TABLE books
    (ISBN   VARCHAR(50)  PRIMARY KEY  NOT NULL,
    name    TEXT    NOT NULL,
    author   TEXT   NOT NULL,
    holders    TEXT    NOT NULL,
    genres    TEXT    NOT NULL,
    price   FLOAT NOT NULL,
    about   TEXT,
    is_unavailable BOOLEAN,
    photo   LONGBLOB,
    date_time_added    TEXT    NOT NULL);''')

@staticmethod
def create_new_books_ratings_table():
    global __db_con_cursor
    __db_con_cursor.execute('''CREATE TABLE books_ratings
```

```
        (ISBN   VARCHAR(50)  PRIMARY KEY  NOT NULL,
        ratings   TEXT    NOT NULL);''')

    @staticmethod
    def create_new_user(new_user: User):
        global __db_con_cursor

        if new_user.photo == None:
            __db_con_cursor.execute(f'''INSERT INTO users(username,
password, password_hint, name, is_disabled, privilege, photo,
date_time_created)
            VALUES ("{new_user.username}", "{new_user.password}",
            "{new_user.password_hint}", "{new_user.name}",
{new_user.is_disabled}, "{new_user.privilege}", NULL,
            "{new_user.date_time_created}");''')
        else:
            __db_con_cursor.execute(f'''INSERT INTO users(username,
password, password_hint, name, is_disabled, privilege, photo,
date_time_created)
            VALUES ("{new_user.username}", "{new_user.password}",
            "{new_user.password_hint}", "{new_user.name}",
{new_user.is_disabled}, "{new_user.privilege}", %s,
            "{new_user.date_time_created}");''', (new_user.photo,))

        __db_con_cursor.execute(f'''INSERT INTO account_settings(username,
theme, accent_colour)
        VALUES ("{new_user.username}", "light", "purple")''')

        Database.save_database()

    @staticmethod
    def update_user(user: User):
        global __db_con_cursor

        if user.photo == None:
            __db_con_cursor.execute(f'''UPDATE users
            SET password="{user.password}",
password_hint="{user.password_hint}", name="{user.name}",
            is_disabled={user.is_disabled}, privilege="{user.privilege}",
photo=NULL
            WHERE username="{user.username}"''')
        else:
            __db_con_cursor.execute(f'''UPDATE users
            SET password="{user.password}",
password_hint="{user.password_hint}", name="{user.name}",
            is_disabled={user.is_disabled}, privilege="{user.privilege}",
photo=%s
            WHERE username="{user.username}"''', (user.photo,))

        Database.save_database()
```

```
    @staticmethod
    def create_new_book(new_book: Book):
        global __db_con_cursor

        if new_book.photo == None:
            __db_con_cursor.execute(f'''INSERT INTO books(ISBN, name,
author, holders, genres, price, about, is_unavailable, photo,
date_time_added)
            VALUES ("{new_book.ISBN}", "{new_book.name}",
            "{new_book.author}", "{new_book.holders}",
"{new_book.genres}", "{new_book.price}", "{new_book.about}",
{new_book.is_unavailable}, NULL, "{new_book.date_time_added}");''')
        else:
            __db_con_cursor.execute(f'''INSERT INTO books(ISBN, name,
author, holders, genres, price, about, is_unavailable, photo,
date_time_added)
            VALUES ("{new_book.ISBN}", "{new_book.name}",
            "{new_book.author}", "{new_book.holders}",
"{new_book.genres}", "{new_book.price}", "{new_book.about}",
{new_book.is_unavailable}, %s, "{new_book.date_time_added}");''',
                                    (new_book.photo,))

        __db_con_cursor.execute(f'''INSERT INTO books_ratings(ISBN,
ratings)
        VALUES ("{new_book.ISBN}", "{{}}")''')

        Database.save_database()

    @staticmethod
    def update_book(book: Book):
        global __db_con_cursor
        if book.photo == None:
            __db_con_cursor.execute(f'''UPDATE books
            SET name="{book.name}", author="{book.author}",
genres="{book.genres}",
            price="{book.price}", is_unavailable={book.is_unavailable},
about="{book.about}", photo=NULL
            WHERE ISBN="{book.ISBN}"''')
        else:
            __db_con_cursor.execute(f'''UPDATE books
            SET name="{book.name}", author="{book.author}",
genres="{book.genres}",
            price="{book.price}", is_unavailable={book.is_unavailable},
about="{book.about}", photo=%s
            WHERE ISBN="{book.ISBN}"''', (book.photo,))

        Database.save_database()

    @staticmethod
```

```
    def update_book_holders(holders, ISBN):
        global __db_con_cursor
        __db_con_cursor.execute(f'UPDATE books SET holders="{holders}"
WHERE ISBN="{ISBN}"')

        Database.save_database()

    @staticmethod
    def update_book_ratings(book_ratings: BookRatings):
        global __db_con_cursor

        __db_con_cursor.execute(f'''UPDATE books_ratings
        SET ratings="{book_ratings.ratings}"
        WHERE ISBN="{book_ratings.ISBN}"''')

        Database.save_database()

    @staticmethod
    def get_user_by_username(username):
        tbr = Database.__filter_users(f'SELECT * FROM users WHERE
username="{username}"')
        if tbr == []:
            return None
        else:
            return tbr[0]

    @staticmethod
    def get_book_by_ISBN(ISBN):
        tbr = Database.__filter_books(f'SELECT * FROM books WHERE
ISBN="{ISBN}"')
        if tbr == []:
            return None
        else:
            return tbr[0]

    @staticmethod
    def get_all_users():
        return Database.__filter_users(f'SELECT * FROM users')

    @staticmethod
    def __filter_users(sql):
        global __db_con_cursor
        tbr = []
        __db_con_cursor.execute(sql)
        users = list(__db_con_cursor.fetchall())
        for i in users:
            tba = User(i[0], i[1], i[2], i[3], i[4], i[5], i[6], i[7])
            tbr.append(tba)

        return tbr
```

```
    @staticmethod
    def get_all_books():
        return Database.__filter_books(f'SELECT * FROM books')

    @staticmethod
    def __filter_books(sql):
        global __db_con_cursor
        tbr = []

        __db_con_cursor.execute(sql)
        books = list(__db_con_cursor.fetchall())
        for i in books:
            tba = Book(i[0], i[1], i[2], literal_eval(i[3]),
literal_eval(i[4]), i[5], i[6], i[7], i[8], i[9])
            tbr.append(tba)

        return tbr

    @staticmethod
    def get_user_account_settings(username):
        global __db_con_cursor
        __db_con_cursor.execute(f'SELECT * FROM account_settings WHERE
username="{username}"')
        s = list(__db_con_cursor.fetchall())[0]
        return UserSettings(s[0], s[1], s[2])

    @staticmethod
    def update_user_account_settings(user_settings: UserSettings):
        global __db_con_cursor
        __db_con_cursor.execute(f'''UPDATE account_settings
                                SET theme="{user_settings.theme}",
accent_colour="{user_settings.accent_colour}"
                                WHERE username="{user_settings.username}"
''')

        Database.save_database()

    @staticmethod
    def get_book_ratings(ISBN):
        global __db_con_cursor
        __db_con_cursor.execute(f'SELECT * FROM books_ratings WHERE
ISBN="{ISBN}"')
        s = list(__db_con_cursor.fetchall())[0]
        return BookRatings(s[0], literal_eval(s[1]))

    @staticmethod
    def set_book_ratings(book_ratings: BookRatings):
        global __db_con_cursor
        __db_con_cursor.execute(f'''UPDATE books_ratings
```

```
                                SET ratings="{book_ratings.ratings}"
                                WHERE ISBN="{book_ratings.ISBN}" ''')

            Database.save_database()

        @staticmethod
        def save_database():
            global __db_con
            __db_con.commit()

        @staticmethod
        def save_local_database():
            global __local_db_con
            __local_db_con.commit()

        @staticmethod
        def is_new_local_setup():
            global __local_db_con_cursor
            return len(list(__local_db_con_cursor.execute(
                'SELECT name FROM sqlite_master WHERE type="table" AND
name="local_settings";'))) == 0

        @staticmethod
        def is_new_server_setup():
            global __db_con_cursor
            __db_con_cursor.execute('SHOW TABLES LIKE "users"')
            return (not __db_con_cursor.fetchone())

        @staticmethod
        def delete_user(username):
            global __db_con_cursor
            __db_con_cursor.execute(f'DELETE FROM users WHERE
username="{username}"')
            __db_con_cursor.execute(f'DELETE FROM account_settings WHERE
username="{username}"')

            Database.save_database()

            for each_book in Database.get_all_books():
                each_book.holders[:] = [x for x in each_book.holders if not
x[0] == username]
                Database.update_book_holders(each_book.holders,
each_book.ISBN)

                each_book_ratings = Database.get_book_ratings(each_book.ISBN)
                each_book_ratings.ratings.pop(username, None)
                Database.update_book_ratings(each_book_ratings)

        @staticmethod
        def delete_book(ISBN):
```

```
        global __db_con_cursor
        __db_con_cursor.execute(f'DELETE FROM books WHERE ISBN="{ISBN}"')
        __db_con_cursor.execute(f'DELETE FROM books_ratings WHERE
ISBN="{ISBN}"')

        Database.save_database()

    @staticmethod
    def delete_database():
        global __db_con_cursor
        __db_con_cursor.execute('DROP DATABASE snakebrary')

        Database.save_database()

    @staticmethod
    def delete_local_database():
        global __local_db_con_cursor
        __local_db_con_cursor.execute('DROP TABLE local_settings')

        Database.save_local_database()

    @staticmethod
    def get_random_book():
        tbr = Database.__filter_books(f'SELECT * FROM books ORDER BY
RAND() LIMIT 1')
        if tbr == []:
            return None
        else:
            return tbr[0]

    @staticmethod
    def clear_local_connection_settings():
        Database.set_local_connection_settings('', '', '', '')

    @staticmethod
    def set_local_connection_settings(host, port, user, password):
        Database.set_local_database_server_host(host)
        Database.set_local_database_server_port(port)
        Database.set_local_database_server_user(user)
        Database.set_local_database_server_password(password)

    @staticmethod
    def is_local_connection_settings_clear():
        return Database.get_local_database_server_host()==''
```

logic/user.py

```
from datetime import datetime
```

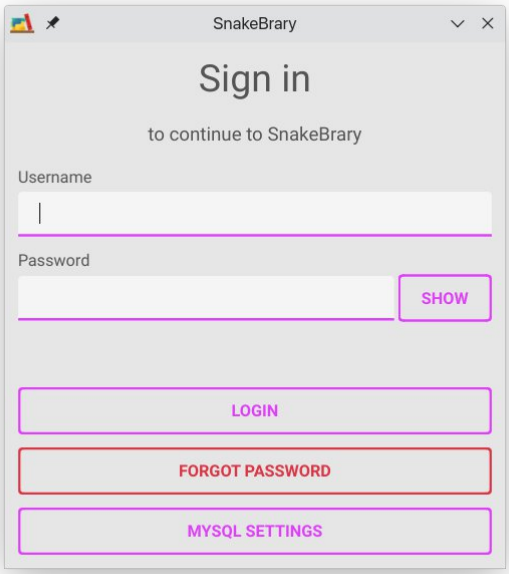
```
class UserPrivilege:
    MASTER = 0
    ADMIN = 1
    NORMAL = 2

    @staticmethod
    def get_ui_name(user_privilege):
        if user_privilege == UserPrivilege.MASTER:
            return 'Master'
        elif user_privilege == UserPrivilege.ADMIN:
            return 'Administrator'
        elif user_privilege == UserPrivilege.NORMAL:
            return 'Normal'

class UserSettings:
    def __init__(self, username, theme, accent_colour):
        self.username = username
        self.theme = theme
        self.accent_colour = accent_colour

class User:
    def __init__(self, username, password, password_hint, name,
is_disabled=False,
                 privilege=UserPrivilege.NORMAL, photo=None,
date_time_created=None):
        self.username = username
        self.password = password
        self.password_hint = password_hint
        self.name = name
        self.is_disabled = is_disabled
        self.privilege = privilege
        self.photo = photo

        if date_time_created == None:
            date_time_created =
datetime.now().strftime("%d/%m/%Y %H:%M:%S")

        self.date_time_created = date_time_created
```

assets/app_icon.png

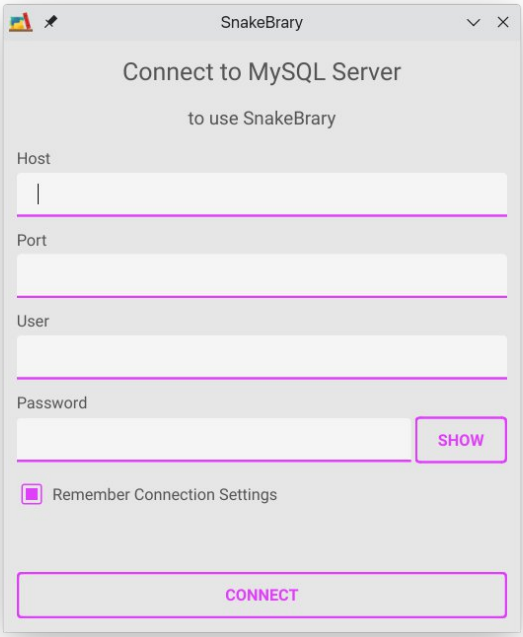
assets/splash.png

Screenshots

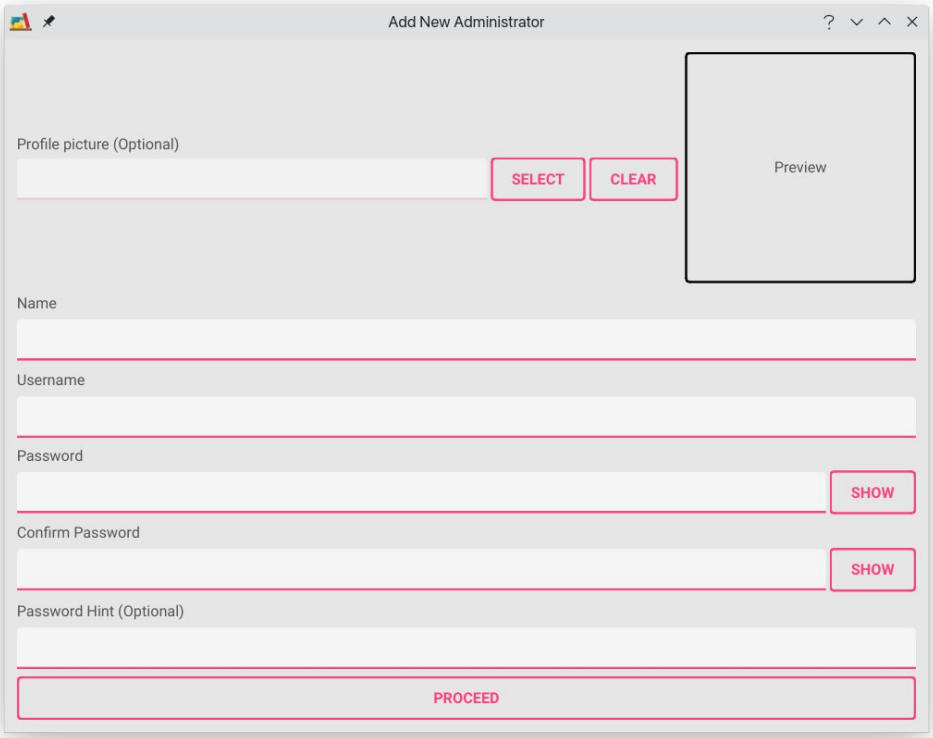
Login Prompt:



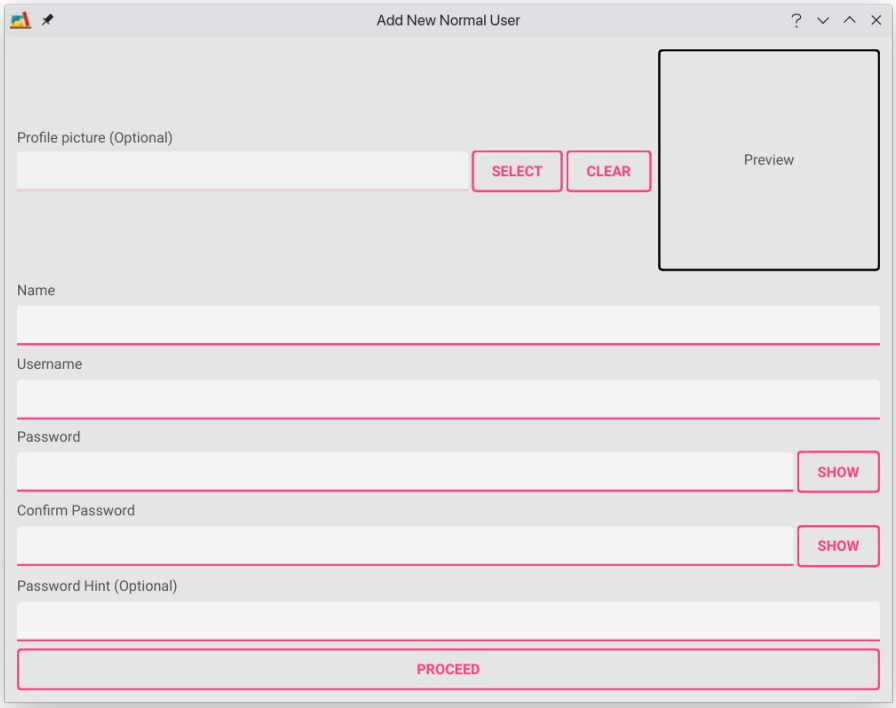
SQL Server Connection Prompt



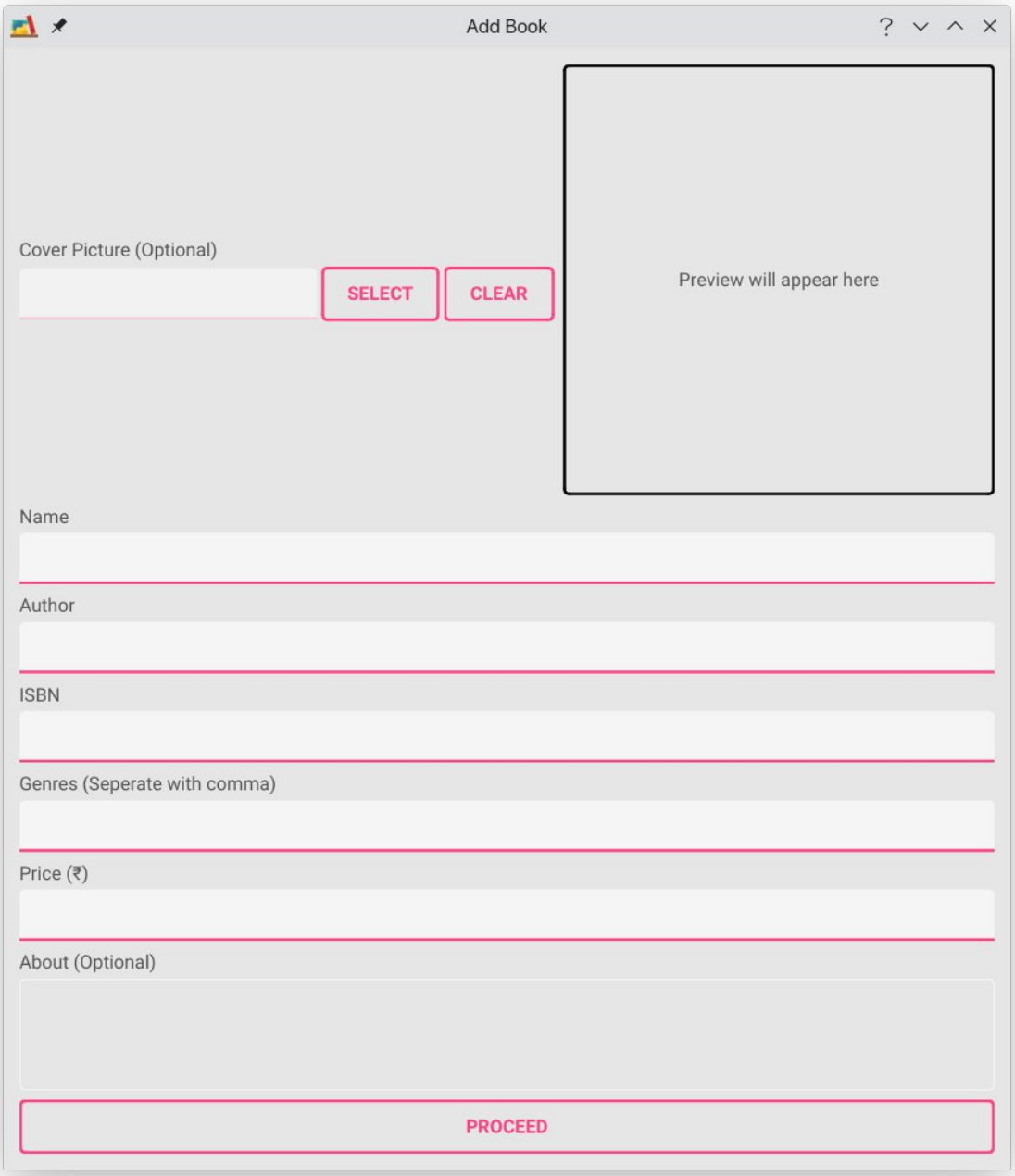
Add new Admin user (Accessible to master account only)



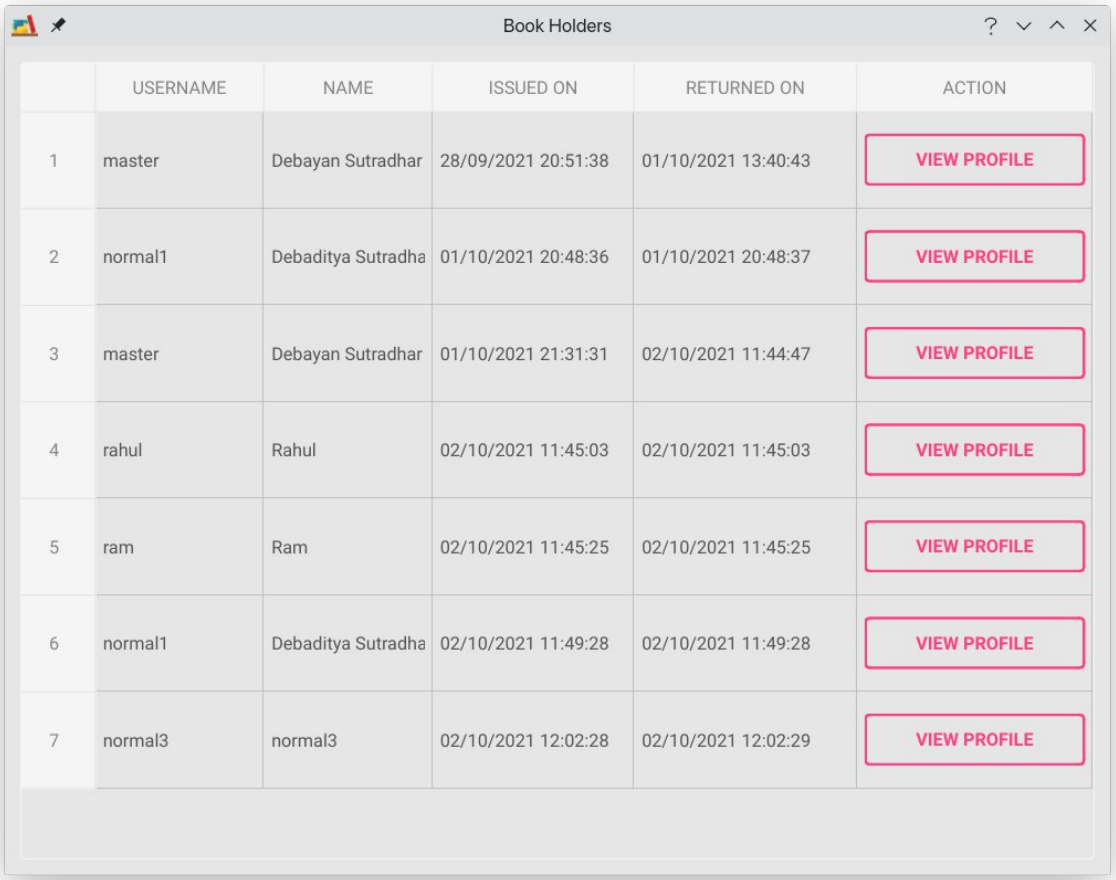
Add normal user (Accessible to master and admin only)



Add new book (Accessible to master and admin only)

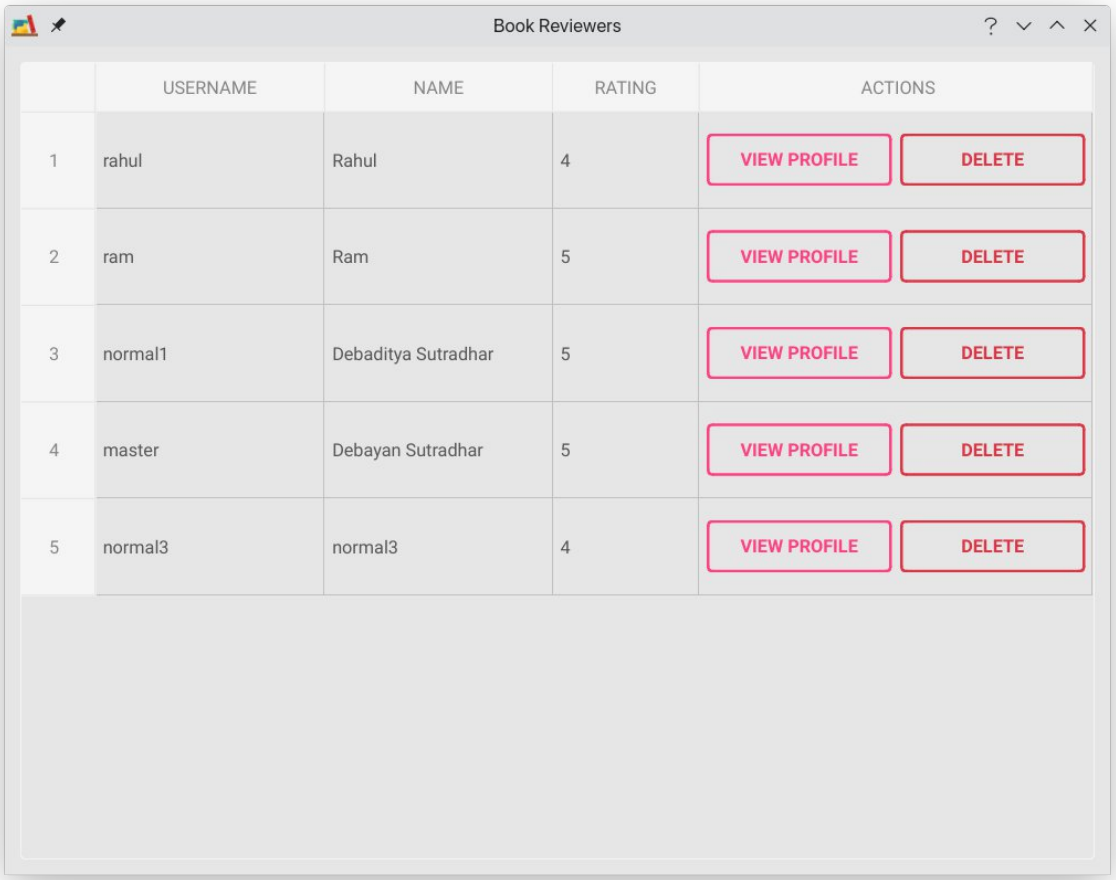


See book holders (Accessible to master and admin only)

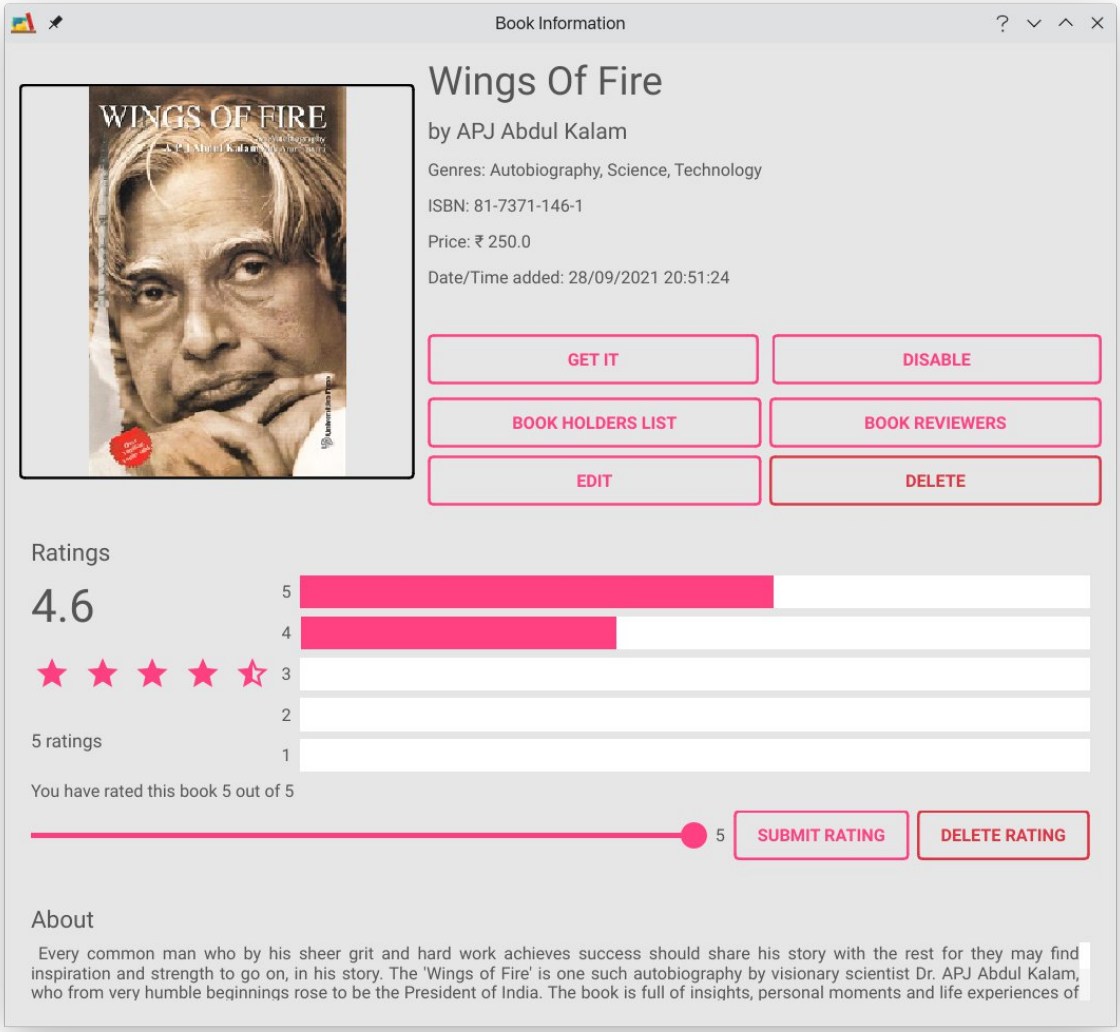


	USERNAME	NAME	ISSUED ON	RETURNED ON	ACTION
1	master	Debayan Sutradhar	28/09/2021 20:51:38	01/10/2021 13:40:43	VIEW PROFILE
2	normal1	Debaditya Sutradha	01/10/2021 20:48:36	01/10/2021 20:48:37	VIEW PROFILE
3	master	Debayan Sutradhar	01/10/2021 21:31:31	02/10/2021 11:44:47	VIEW PROFILE
4	rahul	Rahul	02/10/2021 11:45:03	02/10/2021 11:45:03	VIEW PROFILE
5	ram	Ram	02/10/2021 11:45:25	02/10/2021 11:45:25	VIEW PROFILE
6	normal1	Debaditya Sutradha	02/10/2021 11:49:28	02/10/2021 11:49:28	VIEW PROFILE
7	normal3	normal3	02/10/2021 12:02:28	02/10/2021 12:02:29	VIEW PROFILE

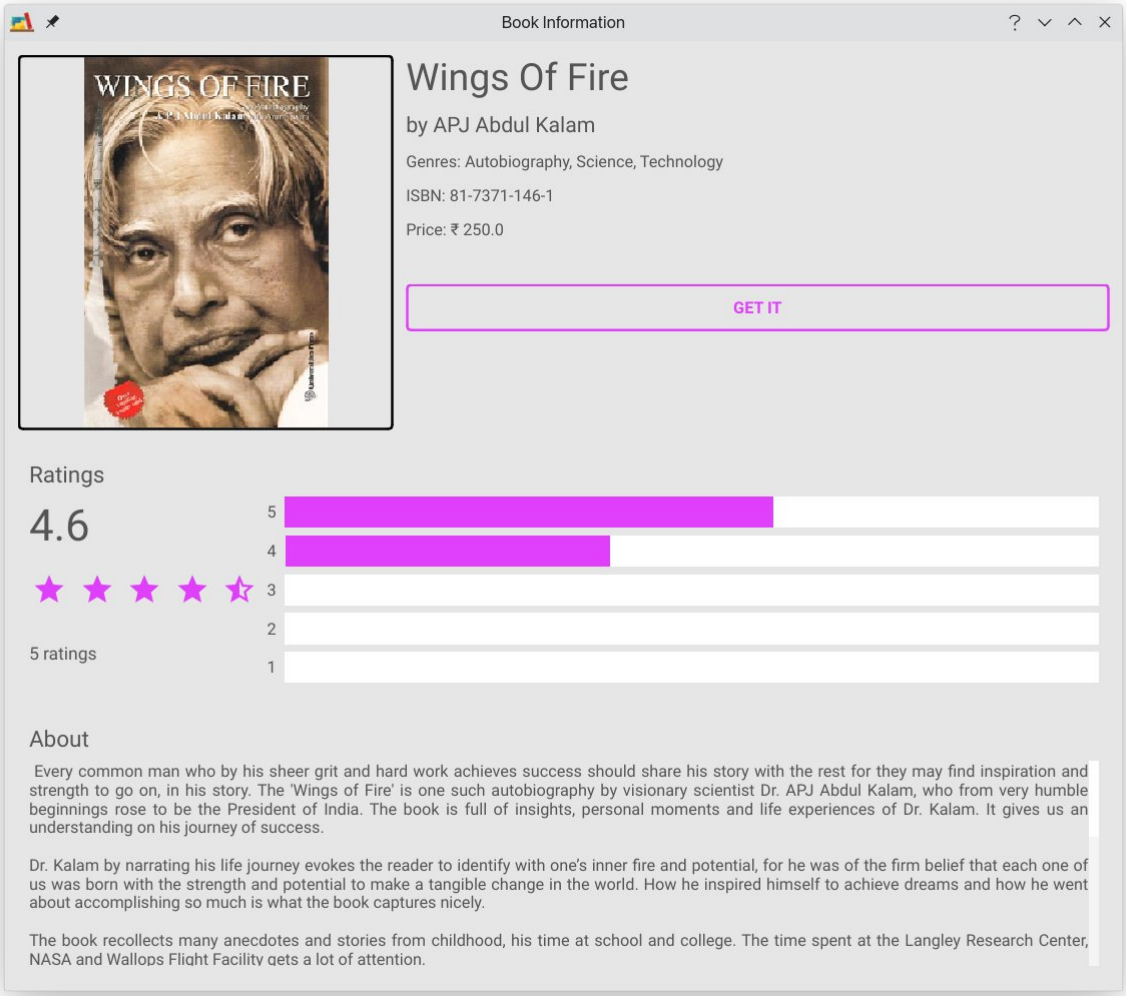
See book reviewers (Accessible to master and admin only)



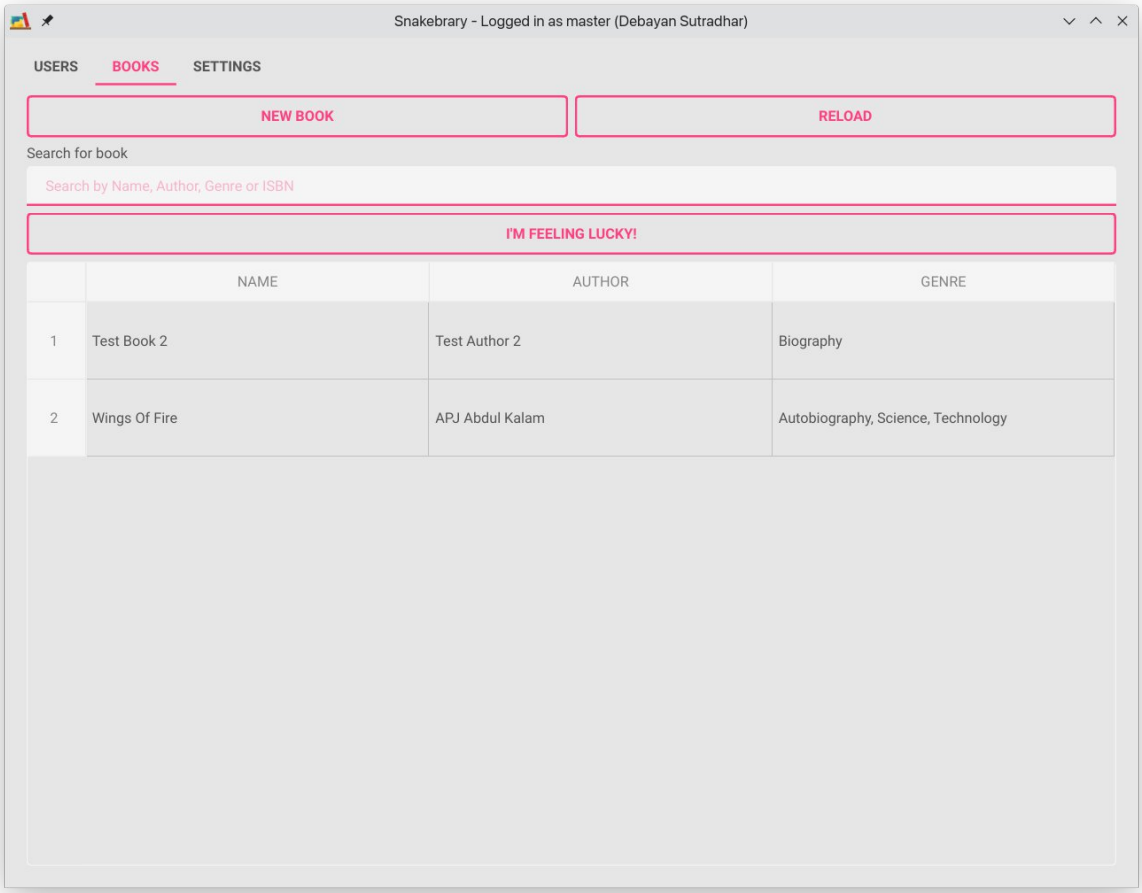
Book Info (Viewed from admin or master account)



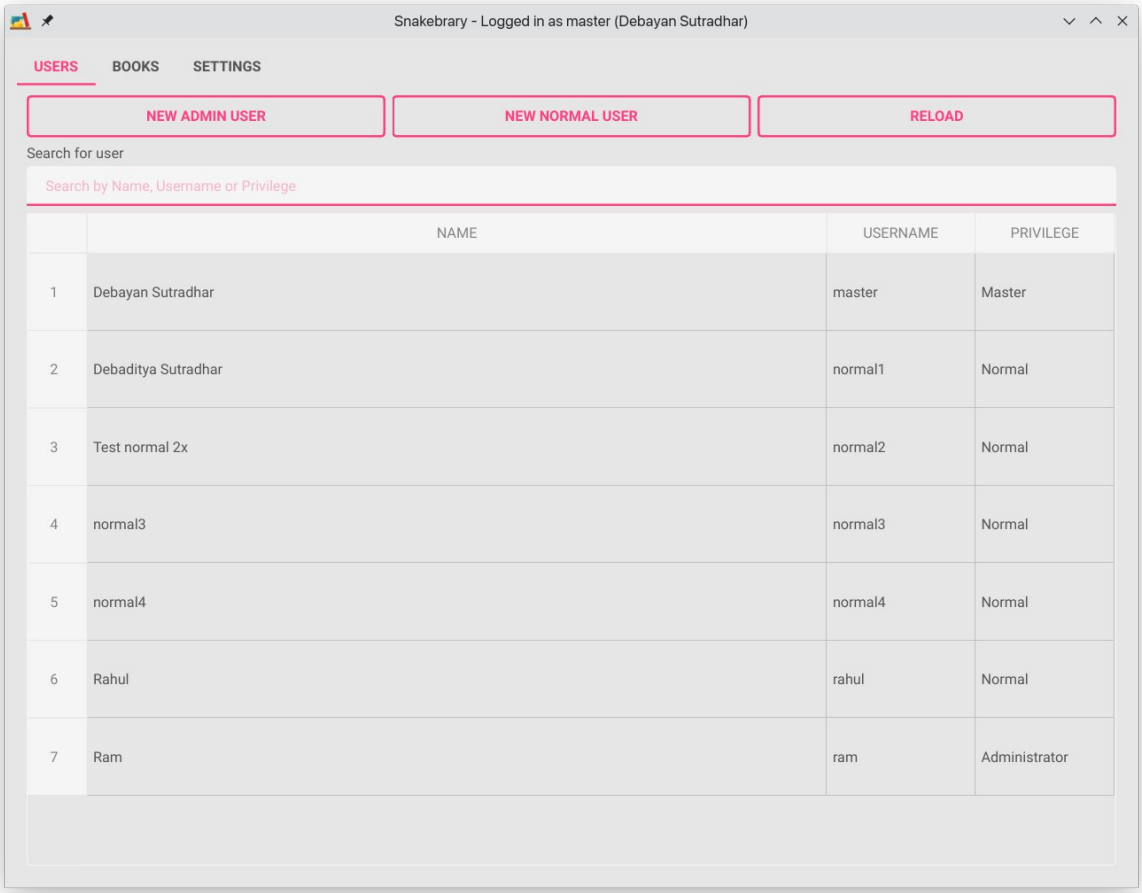
Book Info (Viewed from normal account)



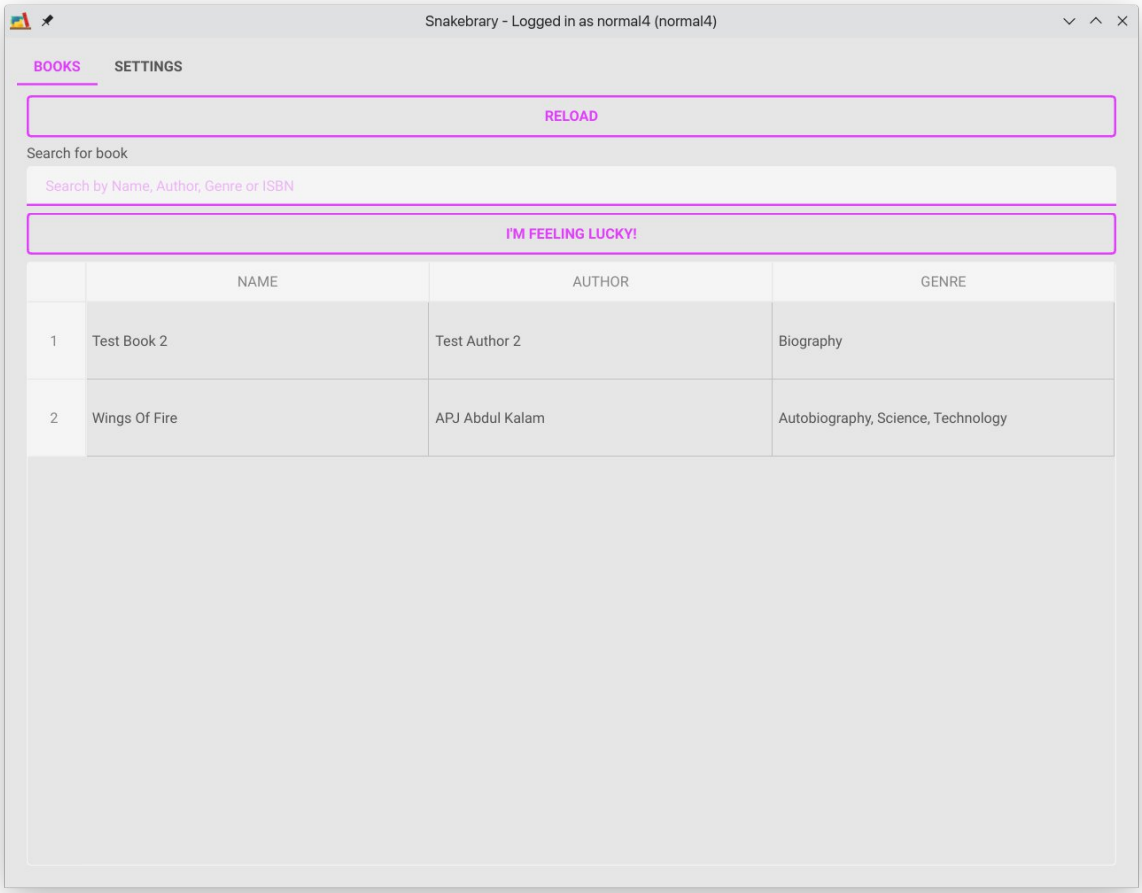
Books Tab (Viewed from admin or master account)



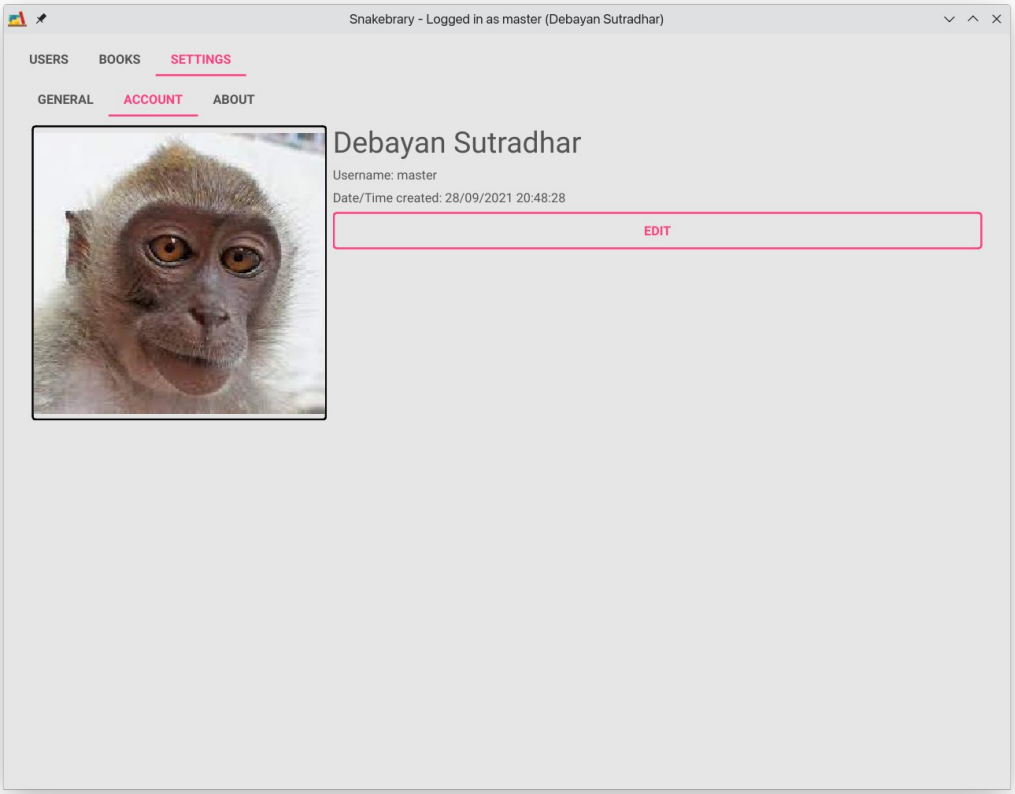
Users Tab (Viewed from admin or master account)



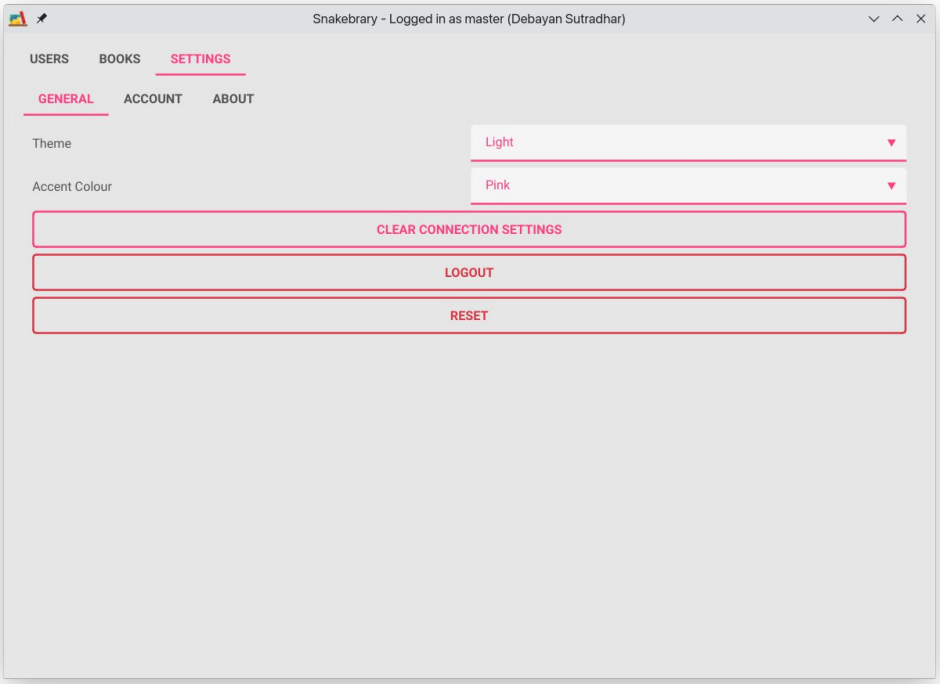
Books Tab (Viewed from normal account)



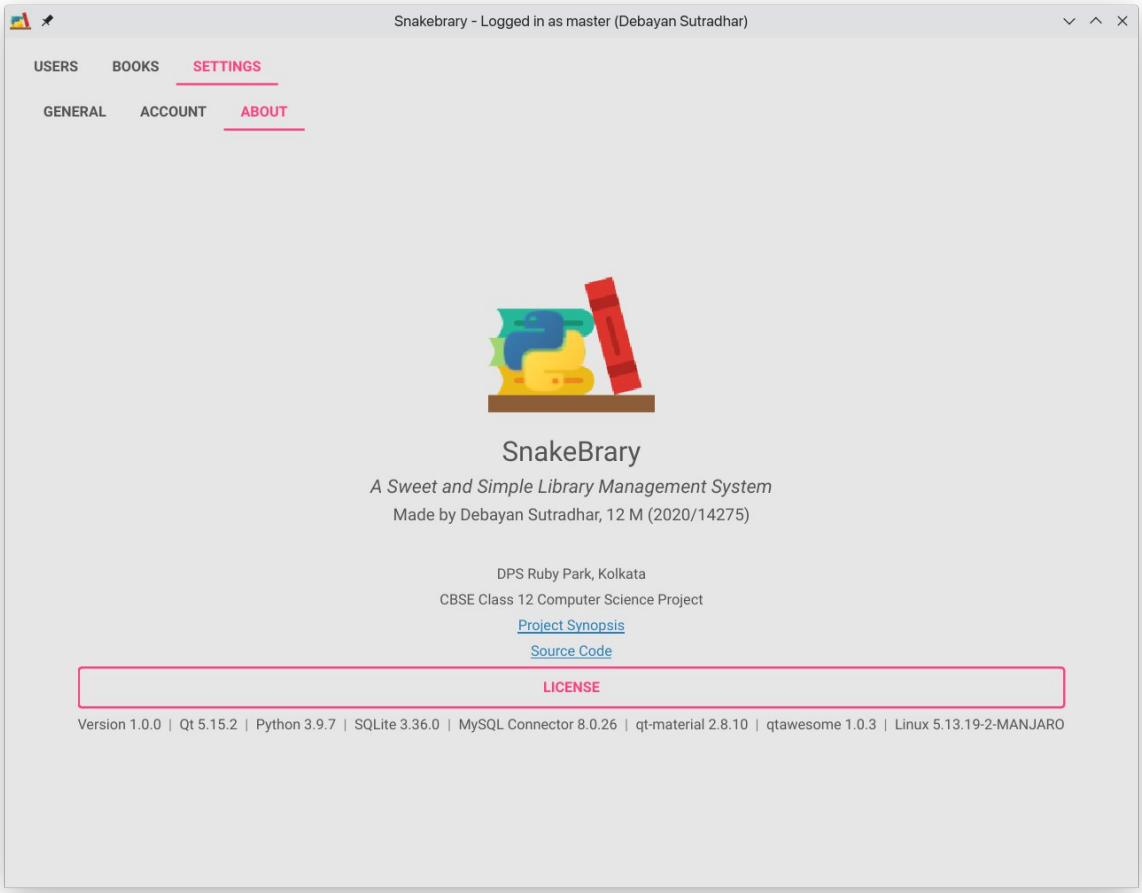
Account Settings



General Settings

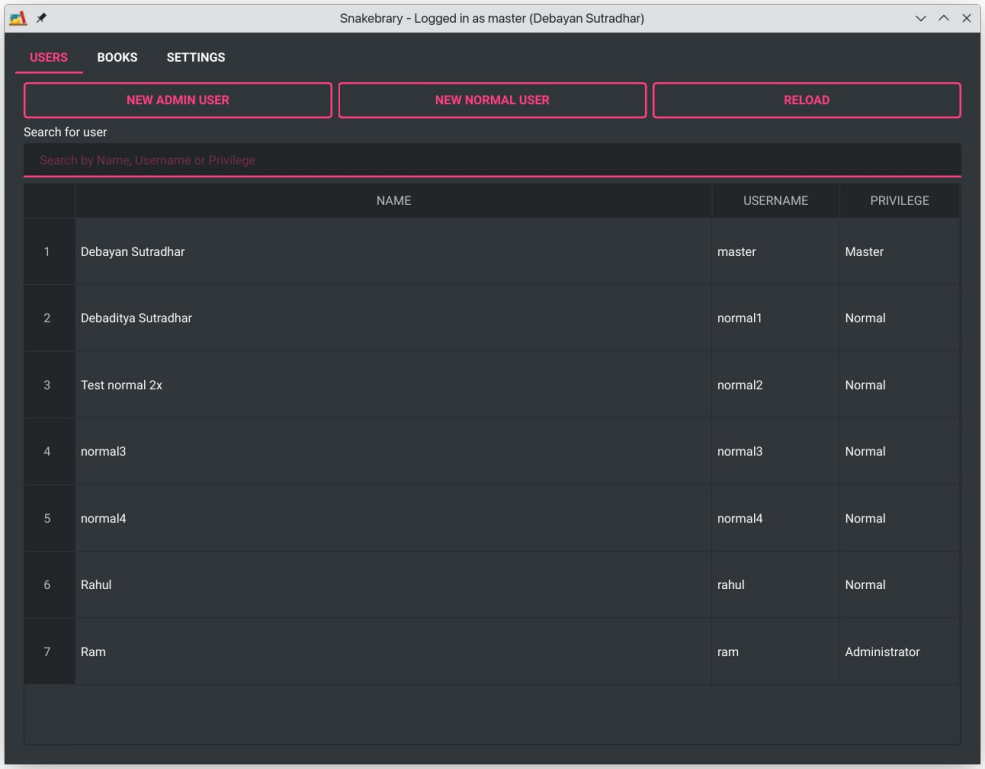


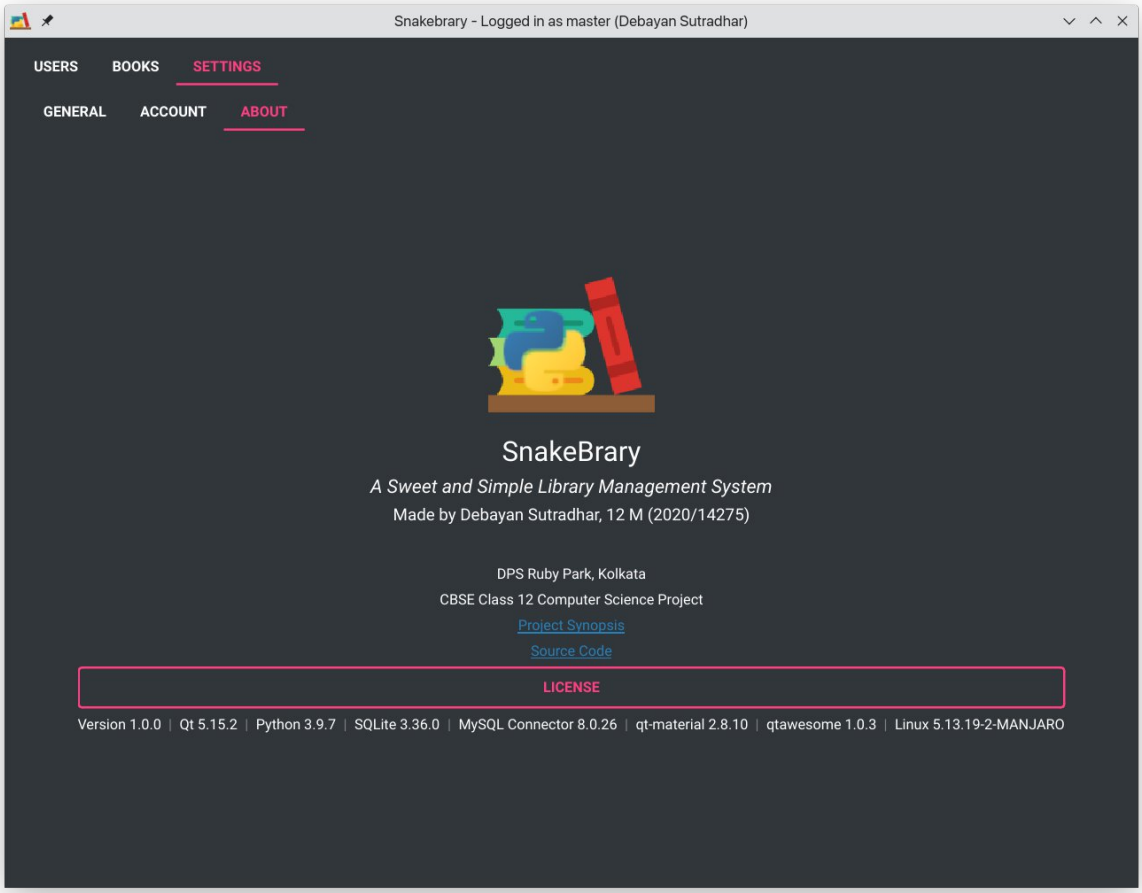
About



Splash screen

Some screenshots in dark theme and other colour combinations




Snakebrary - Logged in as master (Debayan Sutradhar)
USERS
BOOKS
SETTINGS
GENERAL
ACCOUNT
ABOUT
SnakeBrary
A Sweet and Simple Library Management System
Made by Debayan Sutradhar, 12 M (2020/14275)
DPS Ruby Park, Kolkata
CBSE Class 12 Computer Science Project
Project Synopsis
Source Code
LICENSE
Version 1.0.0 | Qt 5.15.2 | Python 3.9.7 | SQLite 3.36.0 | MySQL Connector 8.0.26 | qt-material 2.8.10 | qtawesome 1.0.3 | Linux 5.13.19-2-MANJARO


Book Information
WINGS OF FIRE
Wings Of Fire
by APJ Abdul Kalam
Genres: Autobiography, Science, Technology
ISBN: 81-7371-146-1
Price: ₹ 250.0
GET IT
Ratings
4.6
5 ratings
5
4
3
2
1
About
Every common man who by his sheer grit and hard work achieves success should share his story with the rest for they may find inspiration and strength to go on, in his story. The 'Wings of Fire' is one such autobiography by visionary scientist Dr. APJ Abdul Kalam, who from very humble beginnings rose to be the President of India. The book is full of insights, personal moments and life experiences of Dr. Kalam. It gives us an understanding on his journey of success.
Dr. Kalam by narrating his life journey evokes the reader to identify with one's inner fire and potential, for he was of the firm belief that each one of us was born with the strength and potential to make a tangible change in the world. How he inspired himself to

Bibliography

- PySide2 Documentation
- Python MySQL Tutorial