Computer Science Project

on

Library Management System

Name: Debayan Sutradhar

Class: 12

Section: M

CBSE Roll No.: 12675921

School: 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............................................................................................ | 5 |
| Features.............................................................................................. | 6 |
| Requirements..................................................................................... | 7 |
| Database Structure............................................................................. | 8 |
| Code................................................................................................... | 10 |
| Screenshots........................................................................................ | 85 |
| Bibliography...................................................................................... | 99 |

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
    - **Use Version 2.8.10 ONLY. Future versions will have issues with font size.**
  + 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](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
* LICENSE

**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\_clicked)

self.delete\_rating\_button = QPushButton('Delete Rating')

self.delete\_rating\_button.setProperty('class', 'danger')

self.delete\_rating\_button.clicked.connect(self.delete\_rating\_button\_clicked)

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.rating))

**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.password)

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.current\_user\_account\_settings.theme))

self.theme\_combo\_box.combo\_box.currentIndexChanged.connect(self.change\_theme)

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.change\_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\_local\_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.QHeaderView.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.AlignCenter)

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.QHeaderView.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\_account\_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.QHeaderView.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\_list\_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.ScrollBarAlwaysOff)

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\_clicked)

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\_password())

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\_button\_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

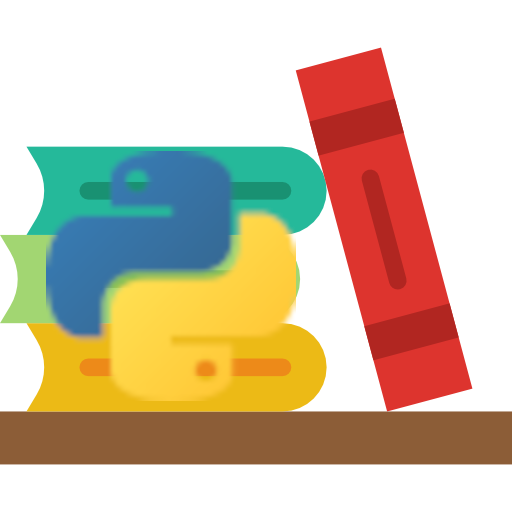
self.photo = photo

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**



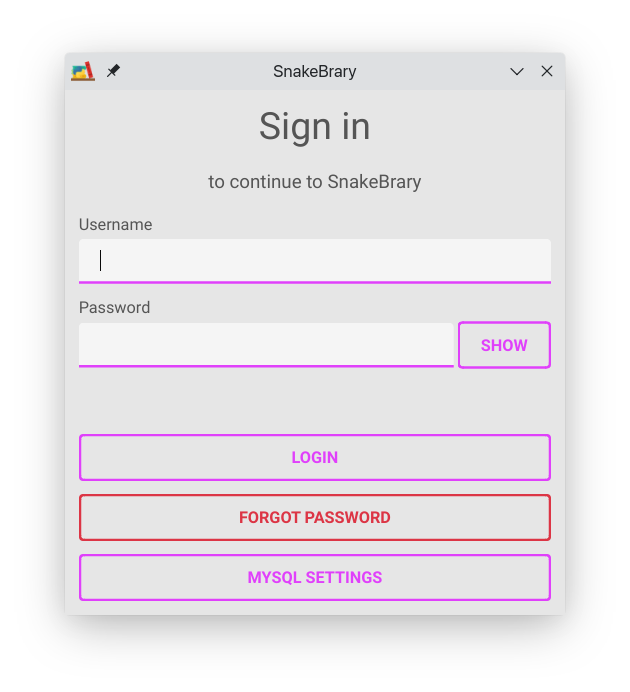
**LICENSE**

GNU General Public License v3.0

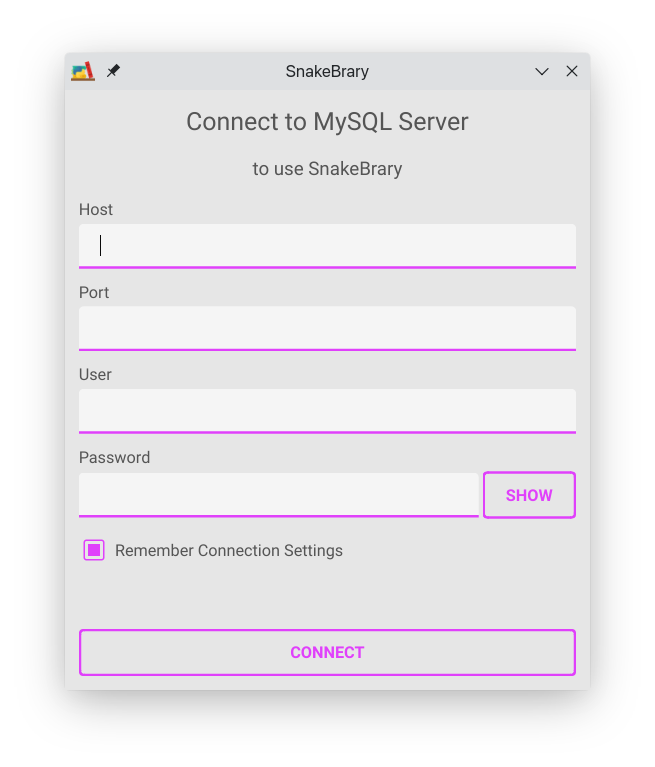
Link: <https://github.com/rnayabed/SnakeBrary/blob/master/LICENSE>

Screenshots

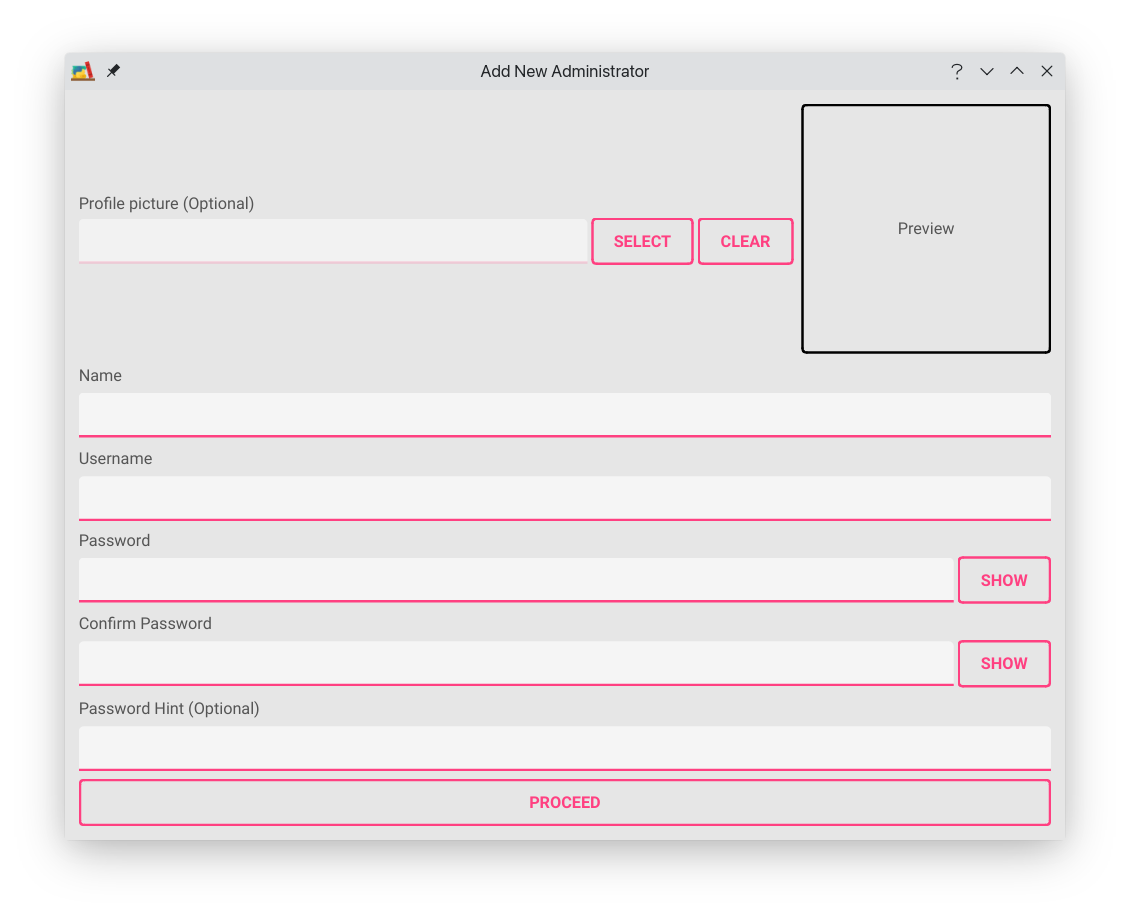
Login Prompt:



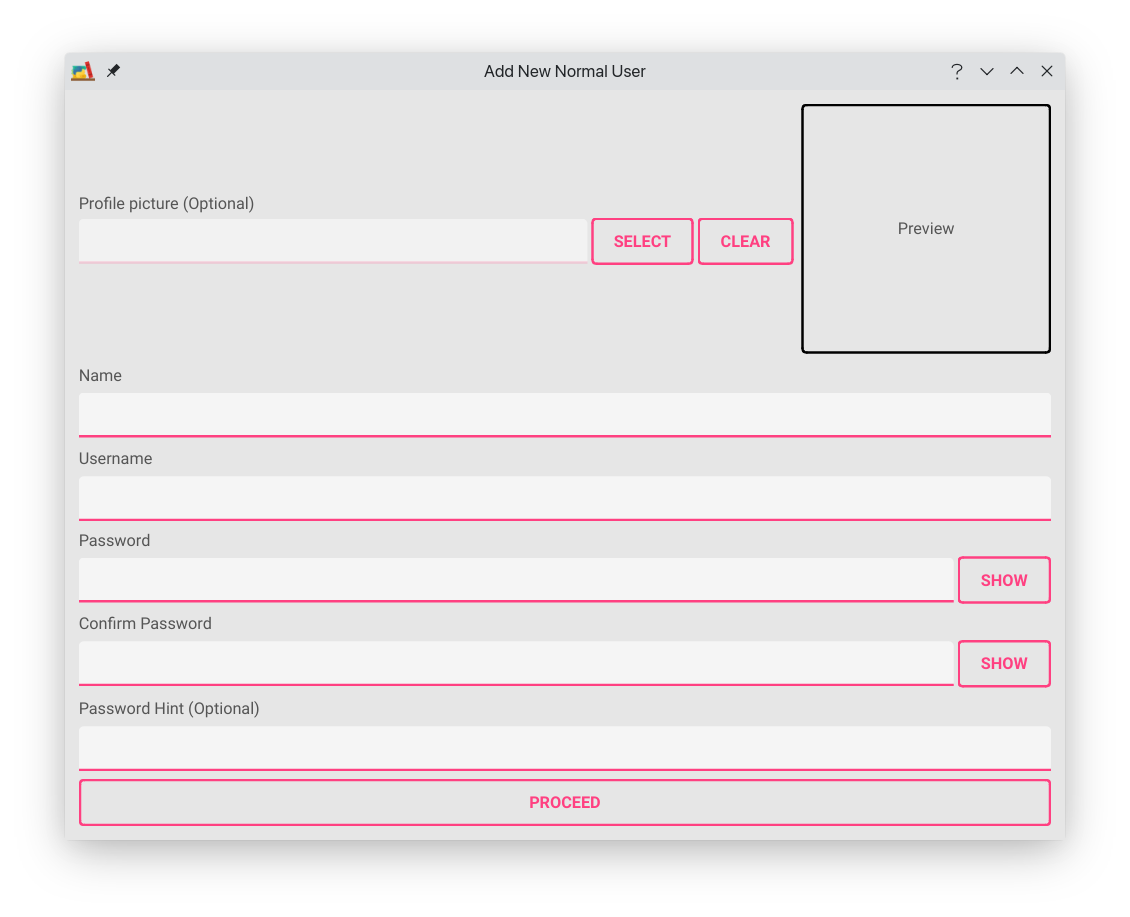
SQL Server Connection Prompt



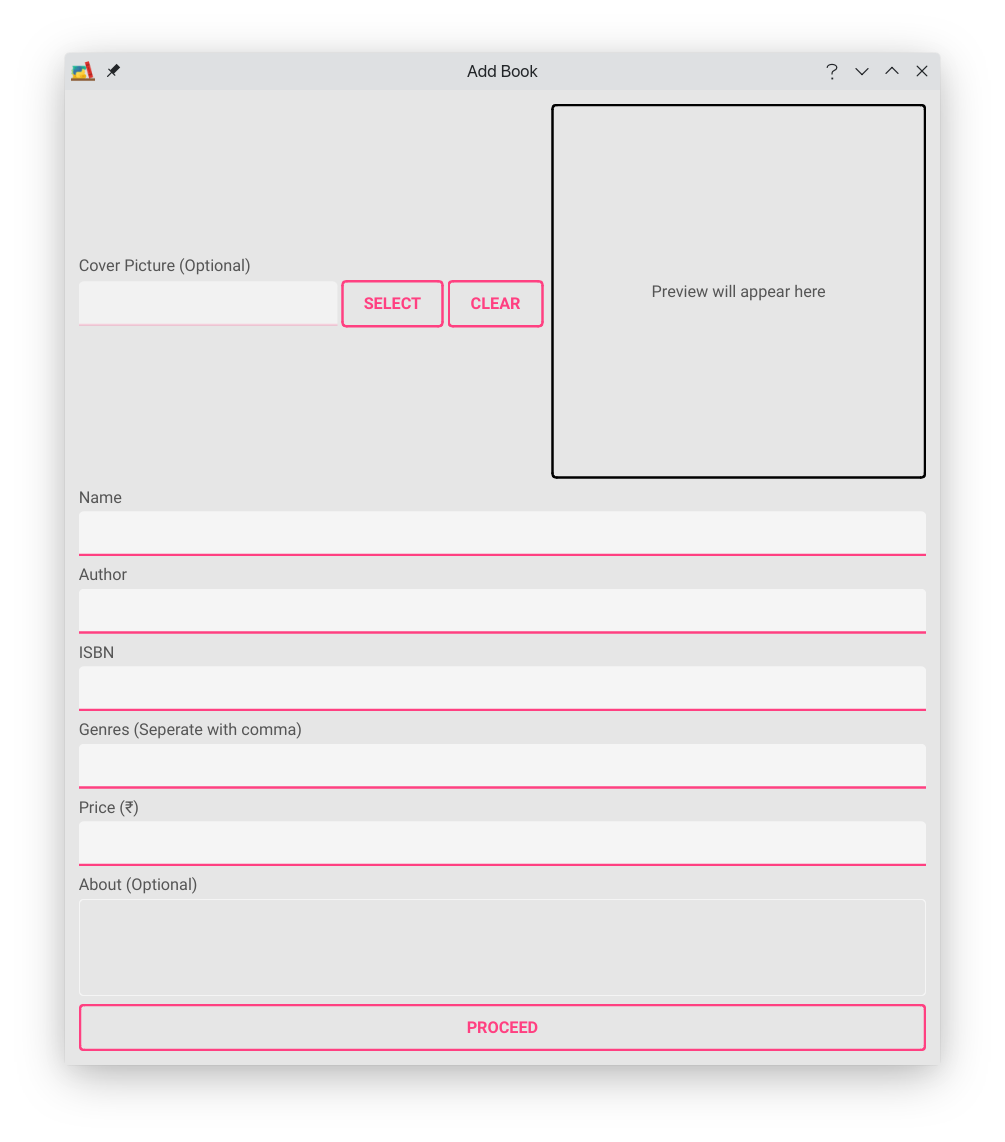
Add new Admin user (Accessible to master account only)



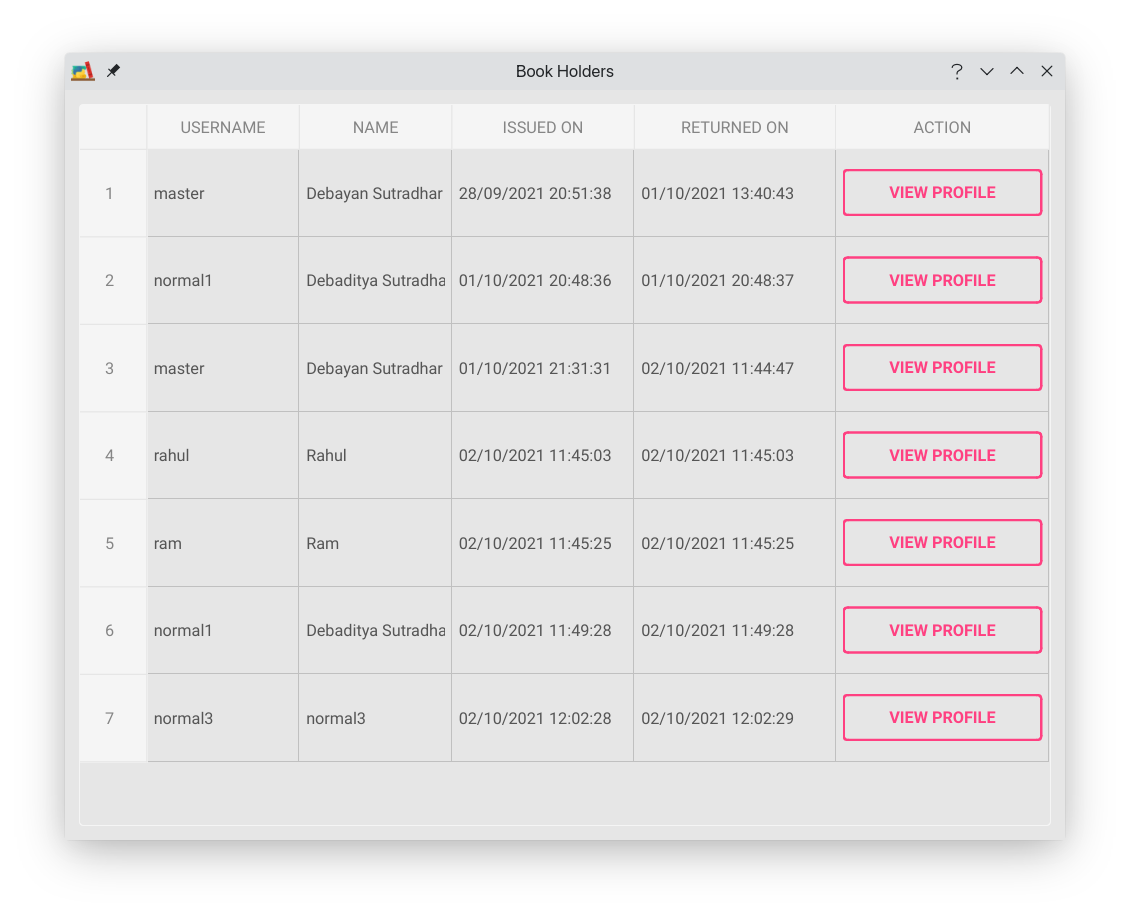
Add normal user (Accessible to master and admin only)



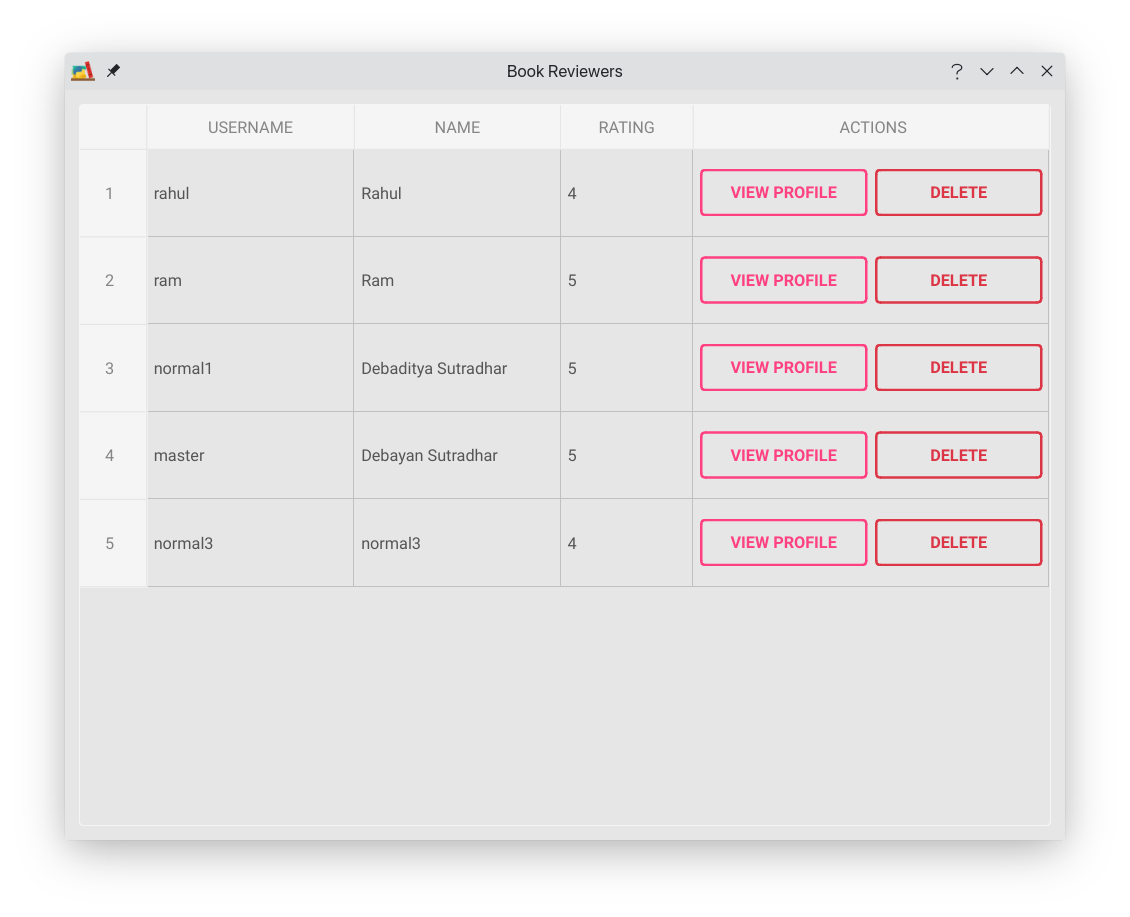
Add new book (Accessible to master and admin only)



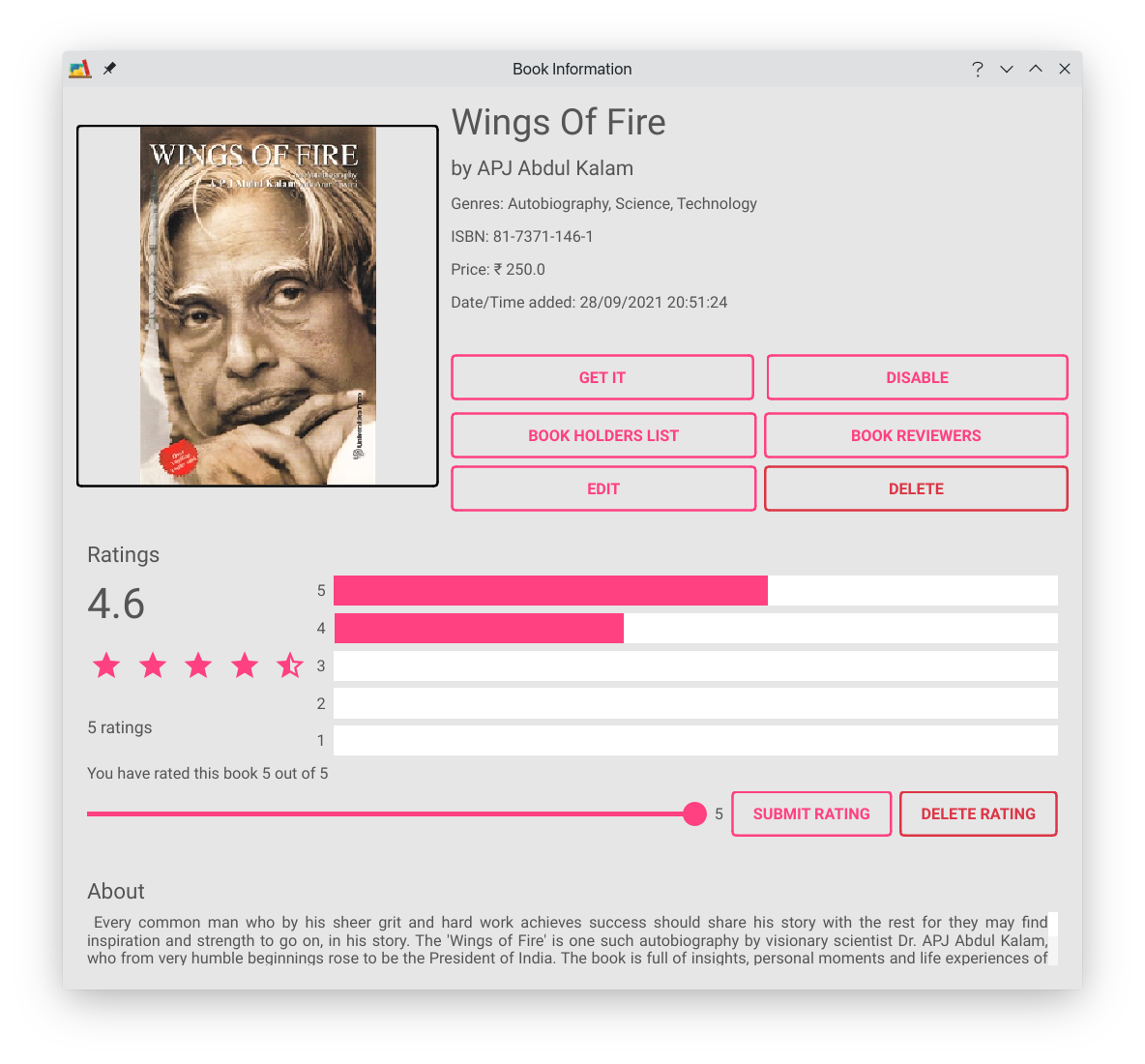
See book holders (Accessible to master and admin only)



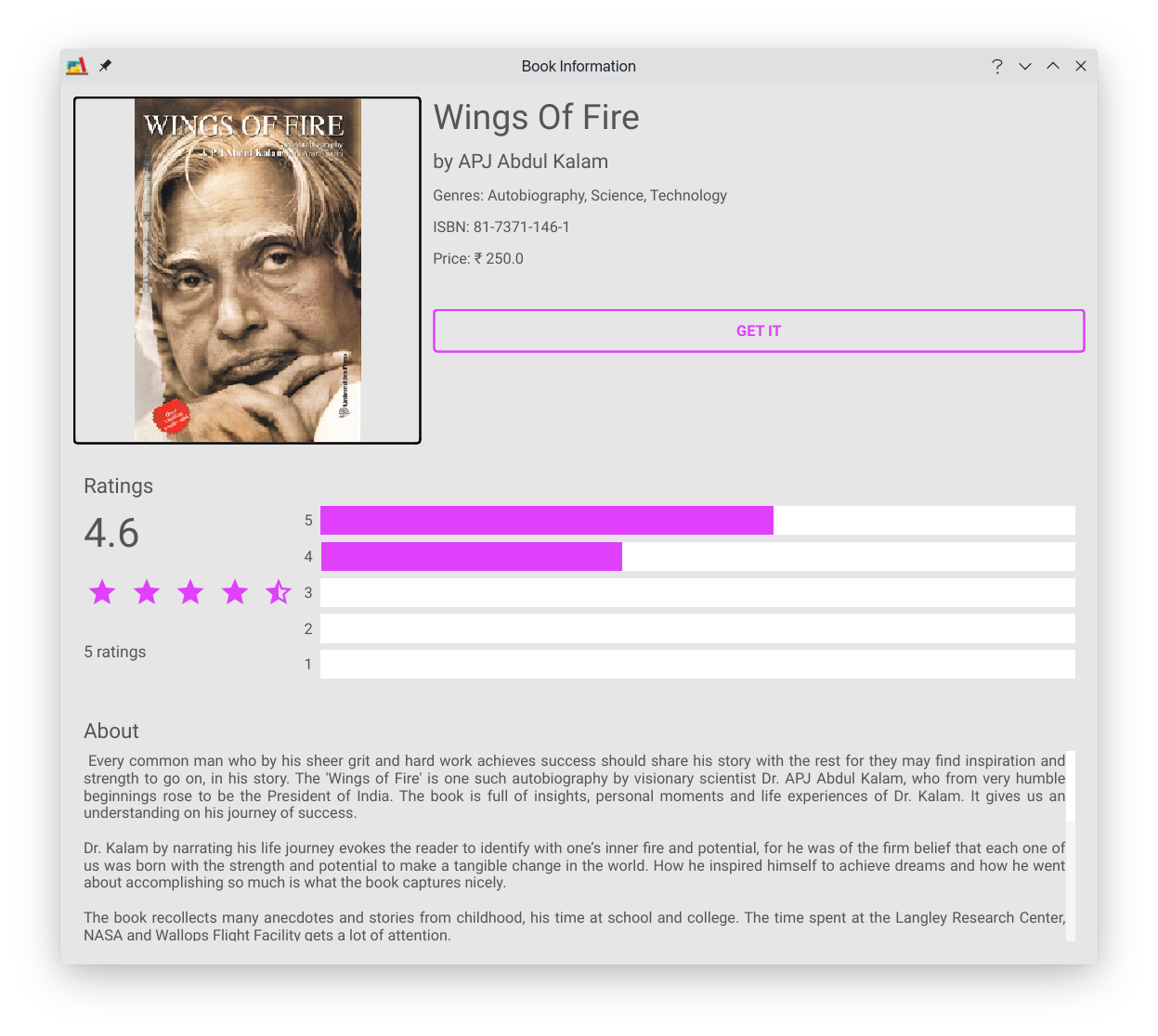
See book reviewers (Accessible to master and admin only)



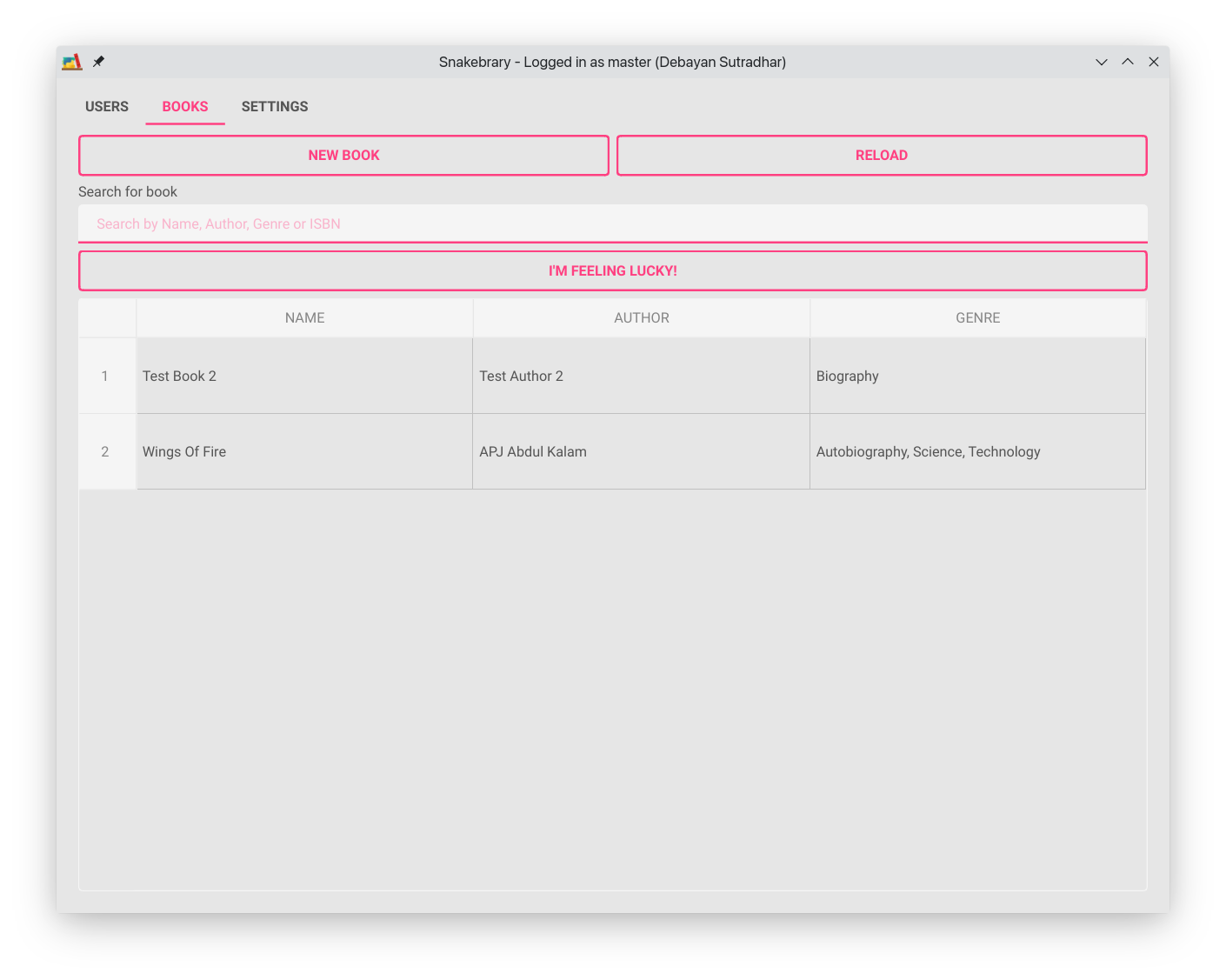
Book Info (Viewed from admin or master account)



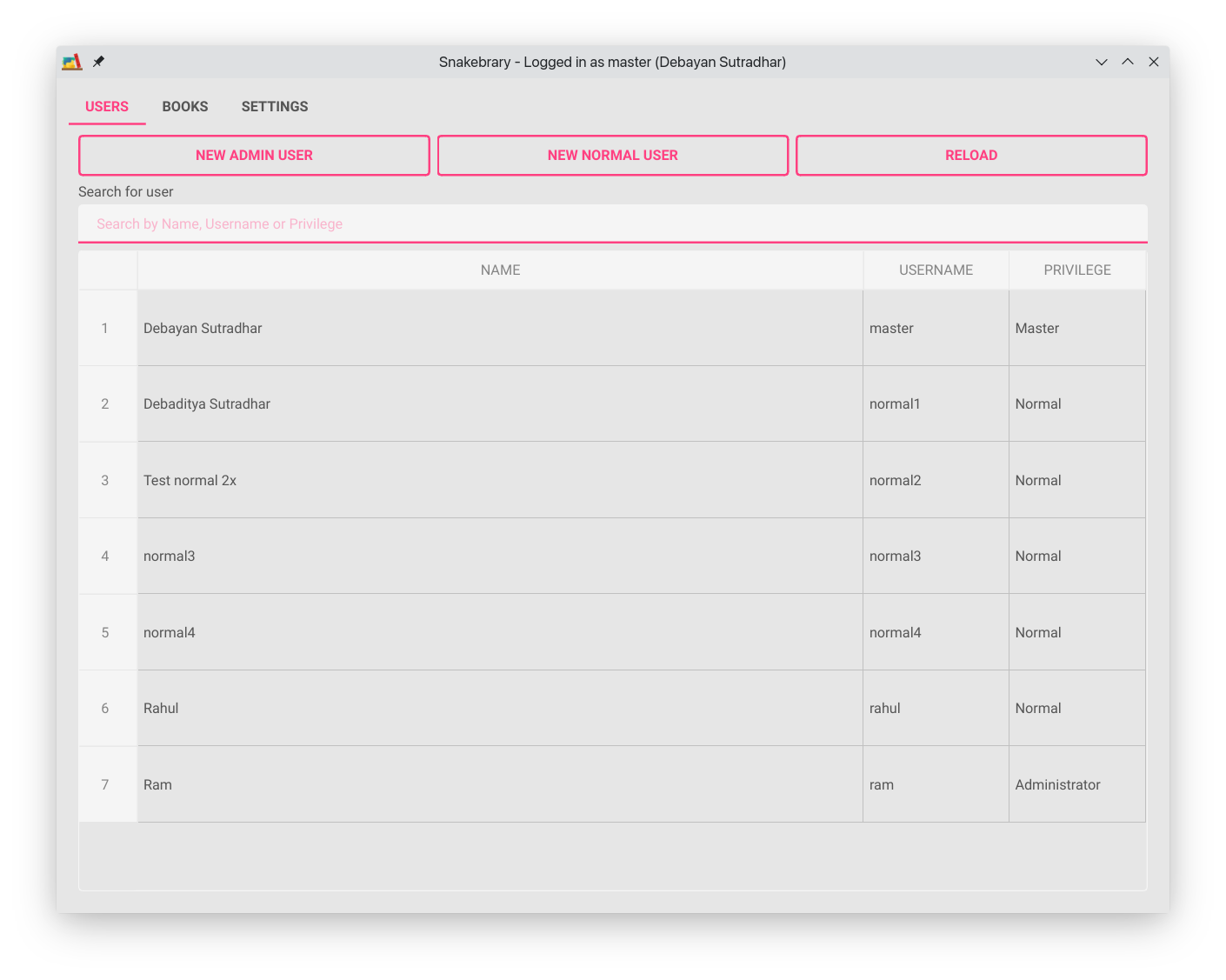
Book Info (Viewed from normal account)



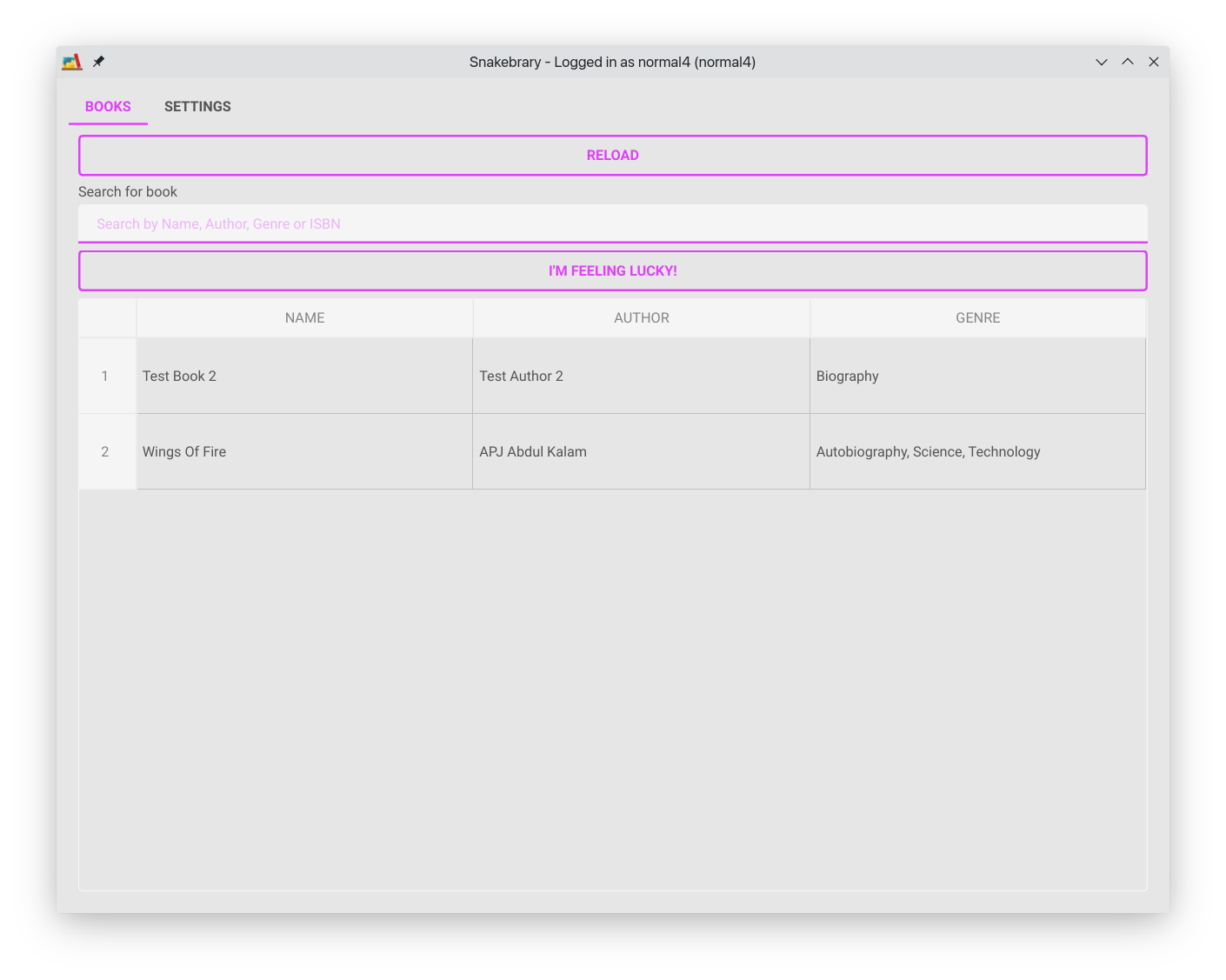
Books Tab (Viewed from admin or master account)



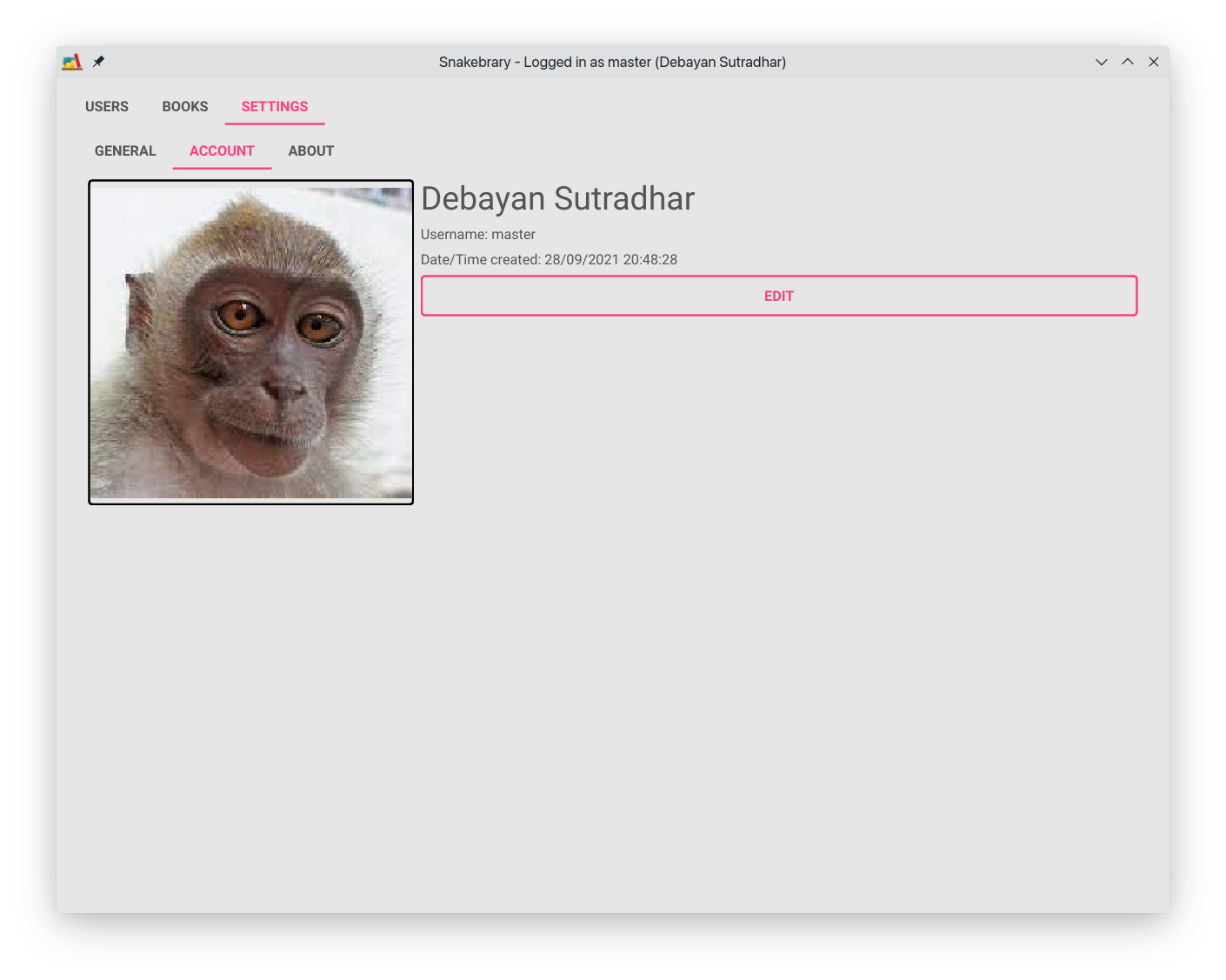
Users Tab (Viewed from admin or master account)



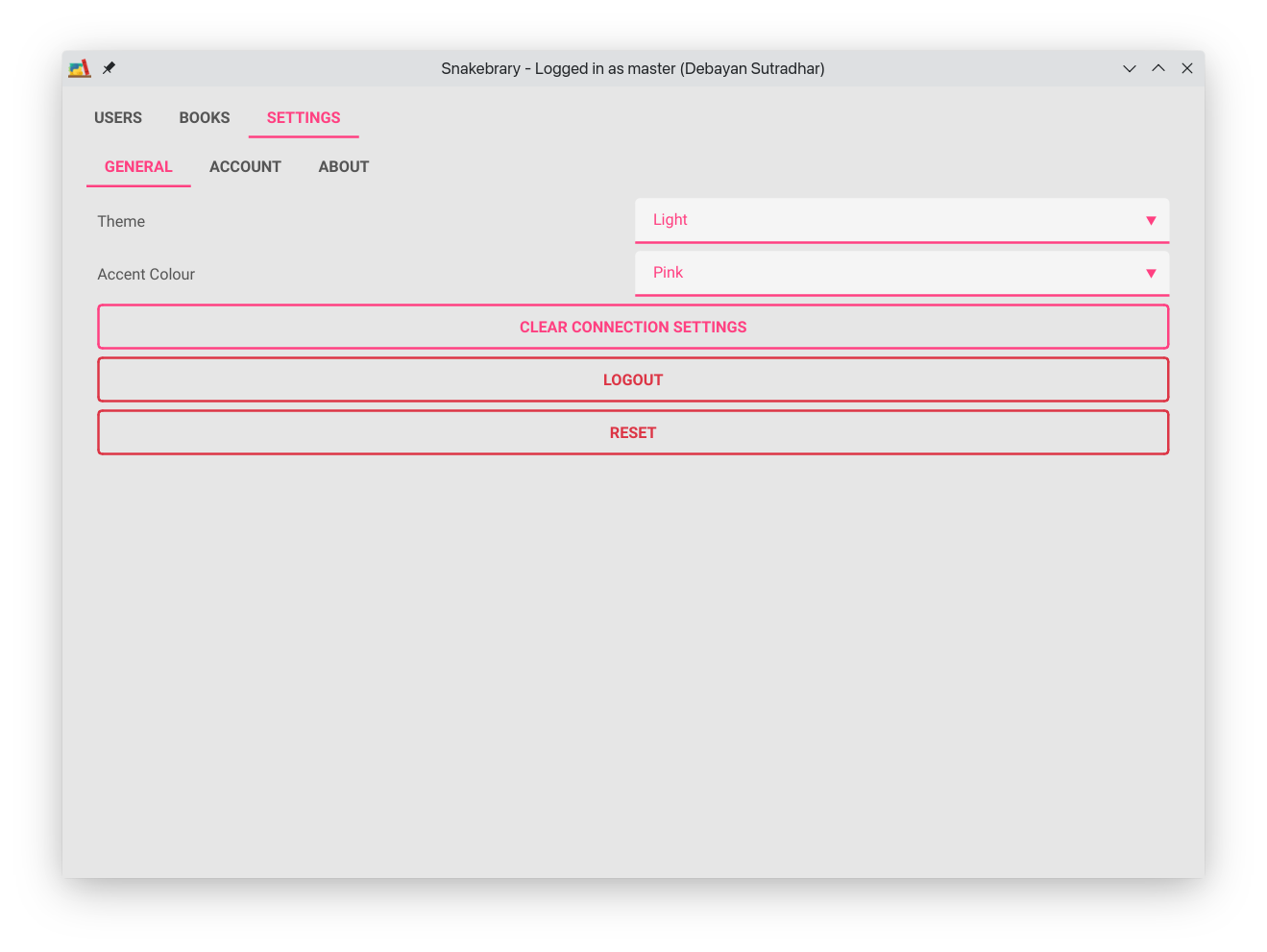
Books Tab (Viewed from normal account)



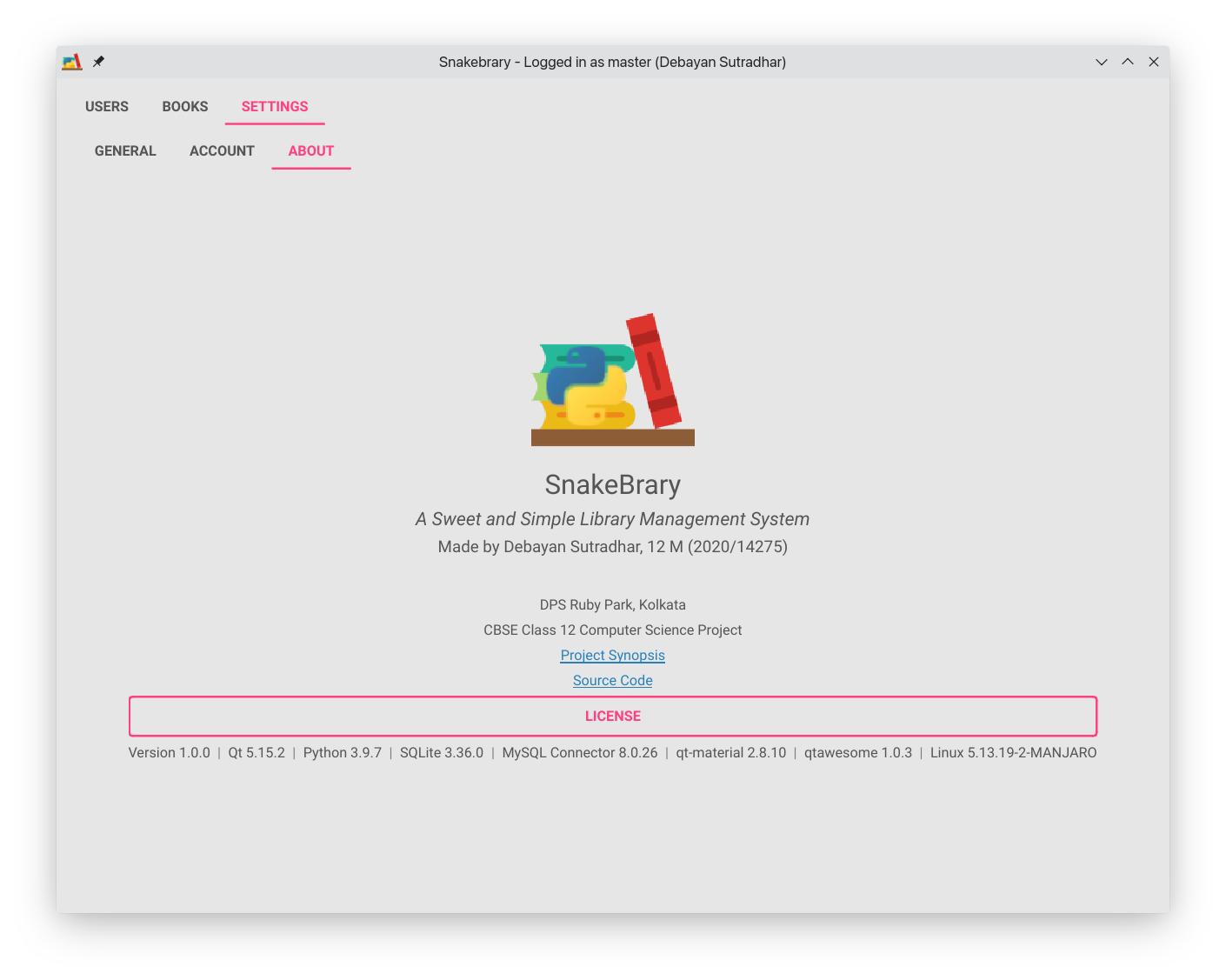
Account Settings



General Settings



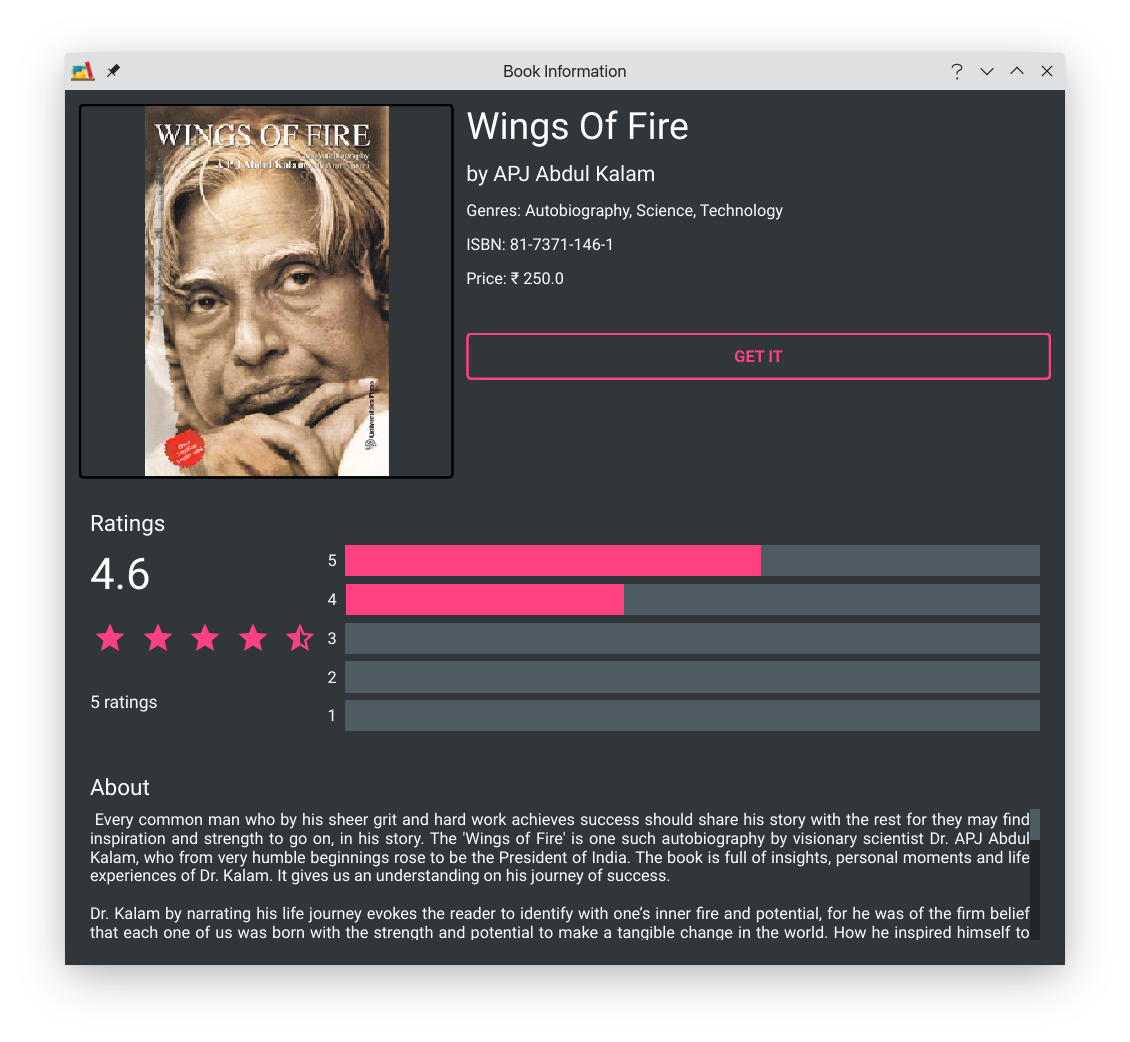
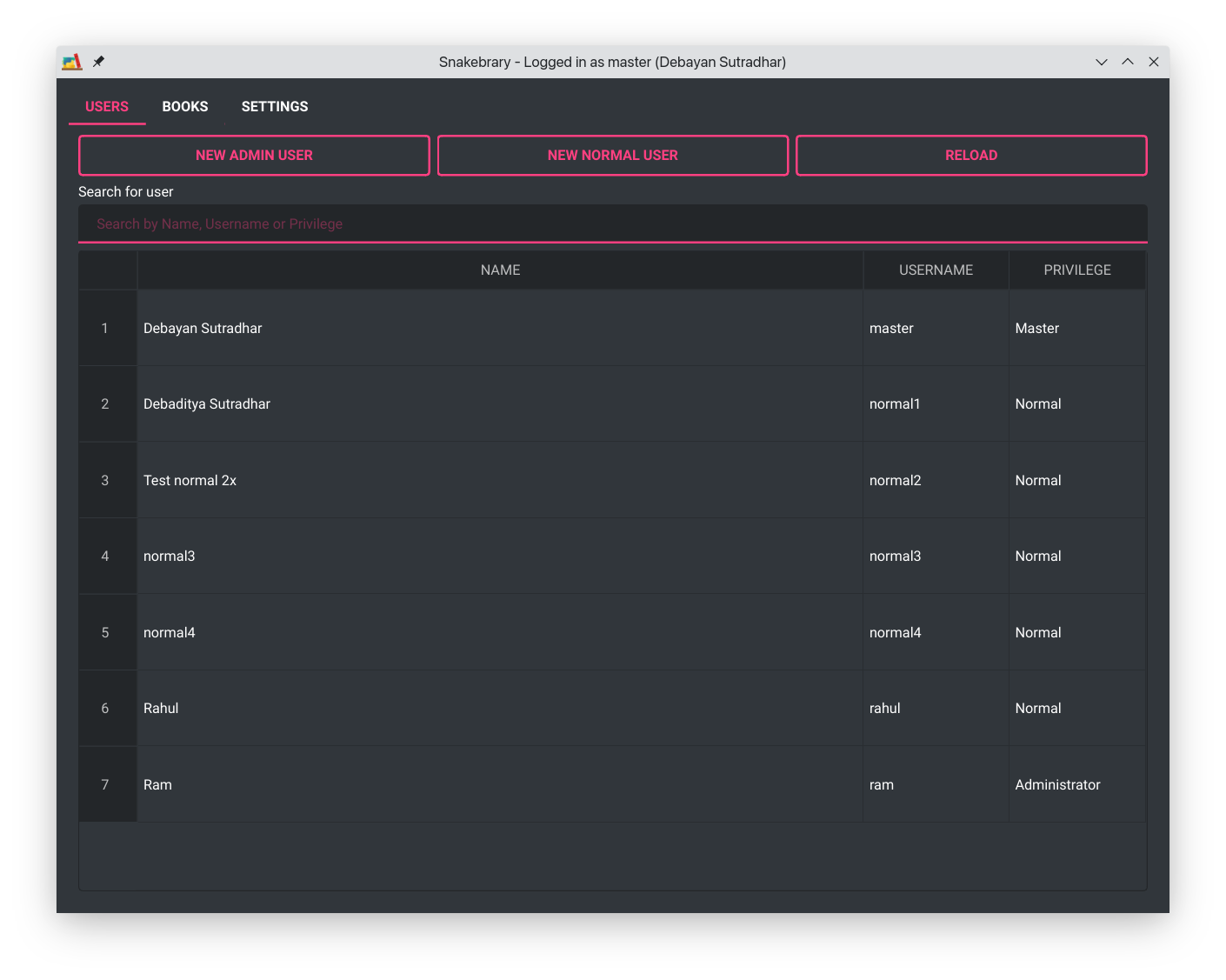
About



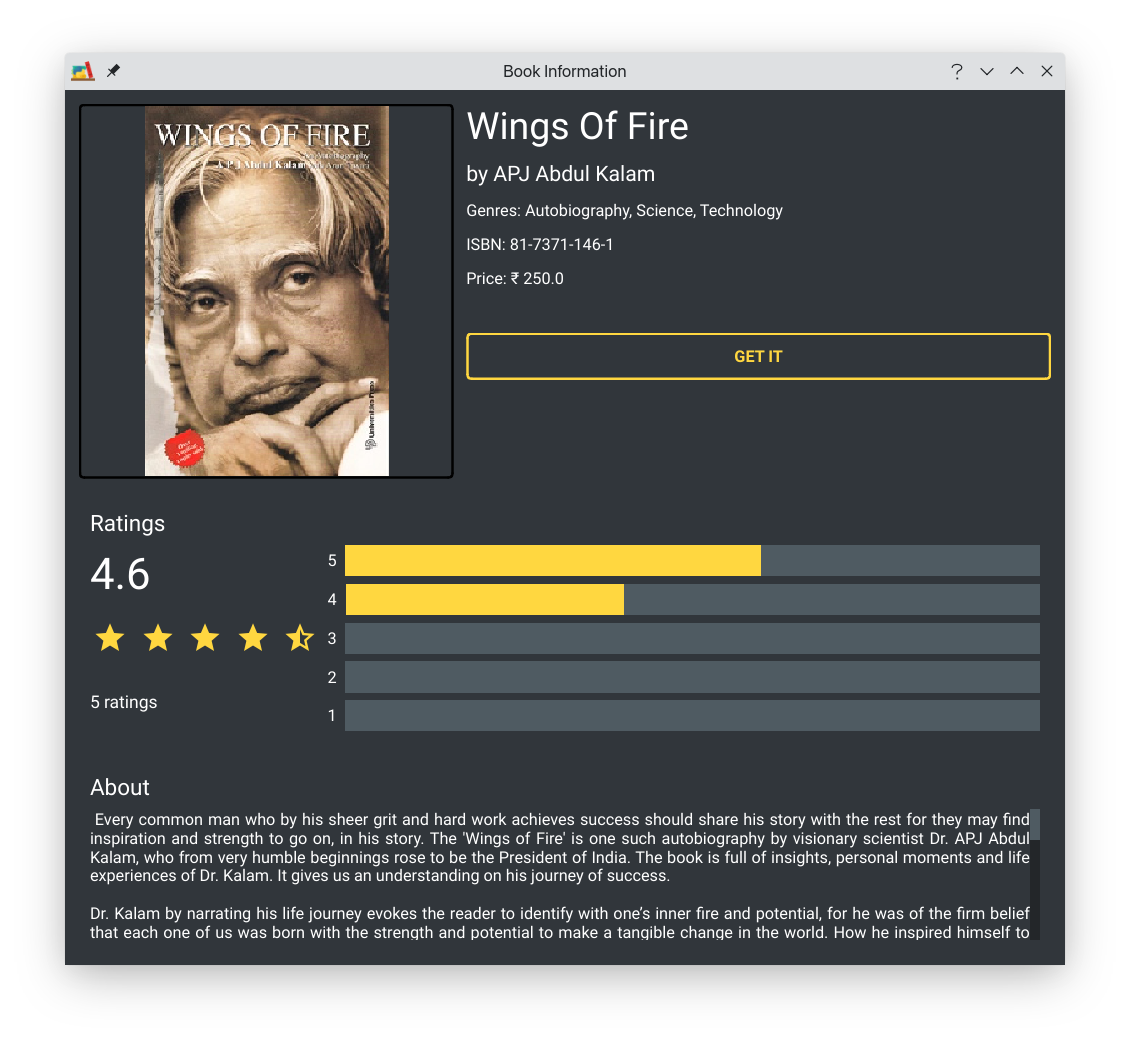
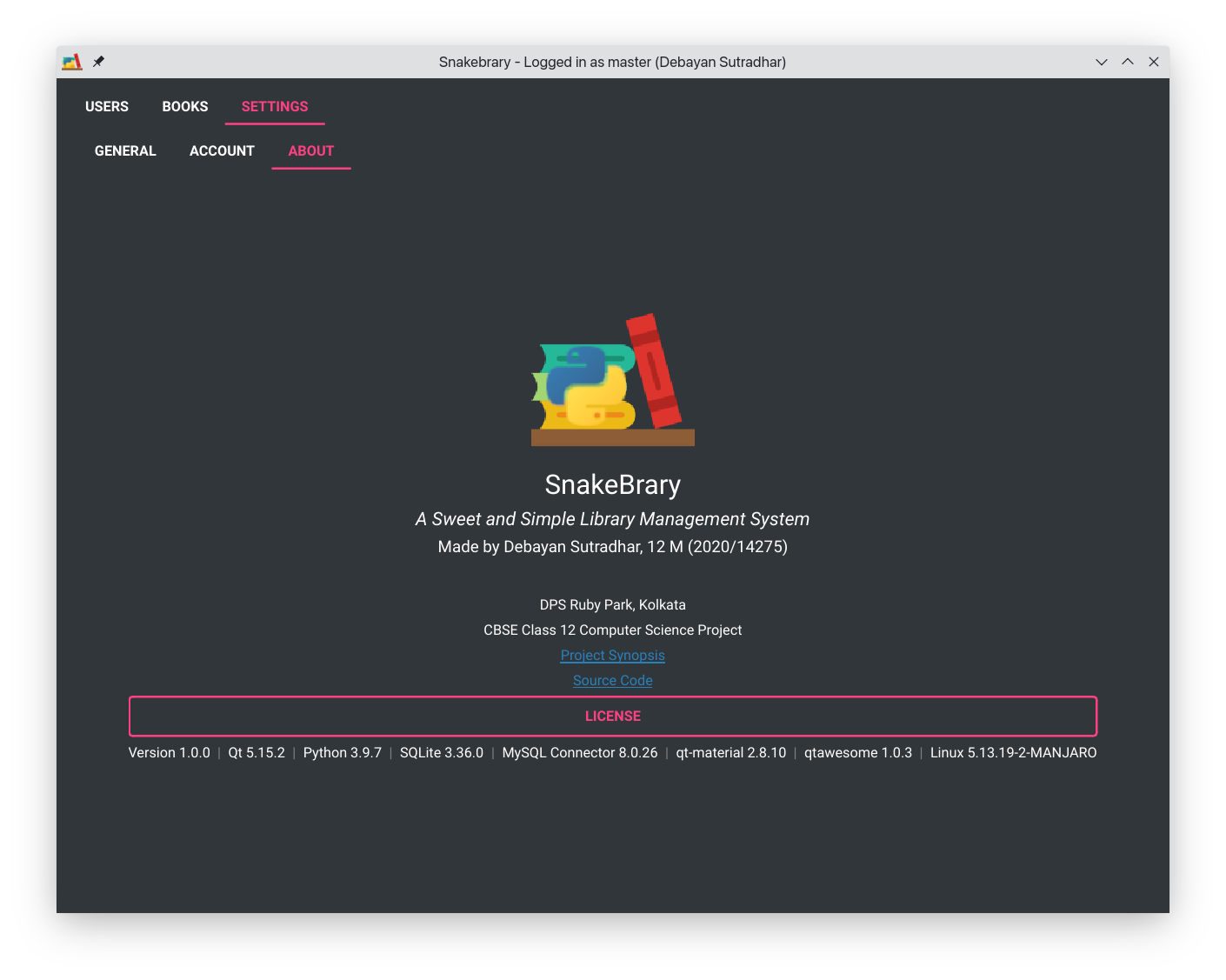
Splash screen



Some screenshots in dark theme and other colour combinations



Bibliography



* PySide2 Documentation
* [Python MySQL Tutorial](https://www.w3schools.com/python/python_mysql_getstarted.asp)