# Приложения

## Приложение А

## Листинги кода

import sys  
from PyQt5.QtCore import Qt  
from PyQt5 import QtWidgets  
import db\_data as rf  
import Auth  
import Registration  
import EditProfile  
import Profile  
import Help  
import Content  
import pymysql  
import ItemClass  
import MakeComposition  
import datetime  
from pymysql.cursors import DictCursor  
  
  
class SearchDB:  
 @staticmethod  
 def get\_profile\_by\_secret\_code(  
 secret\_code: str, role: str  
 ) -> dict:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
  
 if len(secret\_code) > 7:  
 try:  
 cursor.execute(  
 "SELECT \* FROM " + role +  
 "s WHERE secret\_code=%s",  
 secret\_code  
 )  
 return cursor.fetchone()  
 except Exception as e:  
 HelpWindow().error(str(e))  
 else:  
 HelpWindow().error(  
 "Длина кода доступа должна быть больше 8 символов"  
 )  
  
 @staticmethod  
 def check\_user(login: str, password: str, role: str) -> dict:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = False  
 try:  
 success = cursor.execute(  
 "SELECT \* FROM " + role +  
 "s WHERE login\_" + role +  
 "=%s AND password\_" + role +  
 "=%s", (login, password))  
 except Exception as e:  
 AuthWindow().error(str(e))  
  
 if success:  
 return cursor.fetchone()  
 else:  
 AuthWindow().error(  
 "Пользователь не найден, "  
 "проверьте верность введённых данных"  
 )  
  
 @staticmethod  
 def get\_profile\_by\_id(id\_profile: int, role: str) -> dict:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = False  
 try:  
 success = cursor.execute(  
 "SELECT \* from " + role +  
 "s WHERE id=%s", id\_profile  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchone()  
  
 @staticmethod  
 def get\_list\_age\_limit():  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = False  
 list\_age = []  
  
 try:  
 success = cursor.execute(  
 "SELECT value FROM age\_limits"  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 for age in cursor.fetchall():  
 list\_age.append(age['value'])  
 return list\_age  
  
 @staticmethod  
 def get\_id\_age\_limit\_by\_value(value: str) -> int:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = False  
  
 try:  
 success = cursor.execute(  
 "SELECT id from age\_limits WHERE value=%s", value  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchone()['id']  
  
 @staticmethod  
 def check\_user\_by\_login(login: str, role: str) -> bool:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
  
 if len(login) >= 6:  
 flag\_find = cursor.execute(  
 "SELECT \* from " + role +  
 "s WHERE login\_" + role +  
 "=%s", login  
 )  
  
 if flag\_find:  
 return False  
 return True  
 return False  
  
 @staticmethod  
 def check\_user\_by\_secret\_code(secret\_code, role):  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = None  
  
 try:  
 success = cursor.execute(  
 "SELECT \* from " + role +  
 "s WHERE secret\_code=%s", secret\_code  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return True  
 return False  
  
 @staticmethod  
 def get\_writers() -> list:  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = None  
  
 try:  
 success = cursor.execute(  
 "SELECT \* FROM writers"  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchall()  
  
 @staticmethod  
 def get\_name\_by\_id(id\_writer):  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = None  
  
 try:  
 success = cursor.execute(  
 "SELECT name\_writer FROM writers"  
 " WHERE id=%s", id\_writer  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchone()['name\_writer']  
  
 @staticmethod  
 def get\_compositions():  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = None  
  
 try:  
 success = cursor.execute(  
 "SELECT \* FROM compositions"  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchall()  
  
 @staticmethod  
 def get\_age\_limit\_by\_id(id\_age\_limit):  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
 success = None  
  
 try:  
 success = cursor.execute(  
 "SELECT value FROM age\_limits"  
 " WHERE id=%s", id\_age\_limit  
 )  
 except Exception as e:  
 print(e)  
  
 if success:  
 return cursor.fetchone()['value']  
  
  
class DataBase:  
 @staticmethod  
 def connect():  
 connection = pymysql.connect(  
 host=rf.host(),  
 user=rf.user\_db(),  
 password=rf.password(),  
 db=rf.user\_db(),  
 charset='utf8mb4',  
 cursorclass=DictCursor  
 )  
 return connection  
  
  
class PostDB:  
 @staticmethod  
 def create\_user(  
 login: str, password: str,  
 secret\_code: str, name: str,  
 role: str  
 ) -> None:  
  
 if User.check\_len\_str(  
 login, password, secret\_code, name  
 ):  
 connection = DataBase.connect()  
  
 cursor = connection.cursor()  
  
 sql\_profile = "INSERT INTO " + role + \  
 "s (login\_" + role + \  
 ", password\_" + role + \  
 ", secret\_code, " \  
 "name\_" + role + ") " \  
 "VALUES (%s, %s, %s, %s)"  
  
 val\_profile = (  
 login, password, secret\_code, name  
 )  
  
 try:  
 PostDB.post(  
 cursor, connection,  
 sql\_profile, val\_profile  
 )  
 except Exception as e:  
 RegistrationWindow().error(str(e))  
 else:  
 RegistrationWindow().error(  
 " Не соблюдены условия "  
 "запрашиваемых данных "  
 )  
  
 @staticmethod  
 def post\_composition(  
 name\_composition: str, age\_limit: str,  
 main\_genre: str, text\_composition: str,  
 id\_writer: int  
 ):  
 id\_age\_limit = SearchDB.get\_id\_age\_limit\_by\_value(age\_limit)  
  
 connection = DataBase.connect()  
 cursor = connection.cursor()  
  
 sql = "INSERT INTO compositions " \  
 "(name\_composition, main\_genre, " \  
 "id\_age\_limit, text\_composition, id\_writer) " \  
 "VALUES (%s, %s, %s, %s, %s)"  
  
 values = (  
 name\_composition, main\_genre,  
 id\_age\_limit, text\_composition,  
 id\_writer  
 )  
 try:  
 PostDB.post(  
 cursor, connection,  
 sql, values  
 )  
 except Exception as e:  
 print(e)  
  
 @staticmethod  
 def post(cursor, connection, sql, values):  
 success = cursor.execute(sql, values)  
 if success == 1:  
 connection.commit()  
 else:  
 RegistrationWindow().error(  
 'Ошибка!\n'  
 'Данные не были отправлены'  
 )  
  
  
class User:  
 def \_\_init\_\_(self,  
 id\_profile: int, login: str,  
 password: str, name: str,  
 role: str  
 ):  
 self.id\_profile = id\_profile  
 self.login = login  
 self.password = password  
 self.name = name  
 self.role = role  
 self.date\_of\_born = datetime.datetime.today()  
 self.additional\_field = None  
  
 @staticmethod  
 def convert(word: str) -> str:  
 if word == "Инвестор":  
 return "offer"  
 elif word == "Писатель":  
 return "writer"  
 elif word == "Читатель":  
 return "reader"  
  
 @staticmethod  
 def un\_convert(word: str) -> str:  
 if word == "offer":  
 return "offerИнвестор"  
 elif word == "writer":  
 return "Писатель"  
 elif word == "reader":  
 return "Читатель"  
  
 @staticmethod  
 def check\_len\_str(  
 login: str, password: str,  
 secret\_code: str, name: str  
 ) -> bool:  
 if len(login) < 6:  
 return False  
 elif len(password) < 8:  
 return False  
 elif len(secret\_code) < 8:  
 return False  
 elif len(name) < 3:  
 return False  
 else:  
 return True  
  
  
class MakeCompositionWindow(  
 QtWidgets.QMainWindow, MakeComposition.Ui\_MakeComposition  
):  
 def \_\_init\_\_(self, user\_connect):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.user\_connect = user\_connect  
 self.new\_window = None  
 self.send\_btn.clicked.connect(self.send\_composition)  
 self.back\_btn.clicked.connect(self.back\_window)  
  
 def send\_composition(self):  
 name\_composition = self.name\_value.text()  
 age\_limit = self.age\_limit\_box.currentText()  
 main\_genre = self.main\_genre\_value.text()  
 text\_composition = self.text\_composition\_value.toPlainText()  
  
 if self.check\_len(name\_composition, 2) \  
 and self.check\_len(main\_genre, 2) \  
 and self.check\_len(text\_composition, 100):  
 PostDB.post\_composition(  
 name\_composition,  
 age\_limit,  
 main\_genre,  
 text\_composition,  
 self.user\_connect.id\_profile  
 )  
 self.information\_label.setText("Пост успешно отправлен!")  
 else:  
 self.error(  
 "Не оставляйте поля пустыми!"  
 " Минимальный порог пропуска: 2 символа"  
 )  
  
 def back\_window(self):  
 self.new\_window = ProfileWindow(  
 self.user\_connect  
 )  
 self.new\_window.on\_load\_profile()  
 self.new\_window.show()  
 self.close()  
  
 def on\_load\_age\_limit(self):  
 try:  
 self.age\_limit\_box.addItems(  
 SearchDB.get\_list\_age\_limit()  
 )  
 except Exception as e:  
 print(e)  
  
 def error(self, e):  
 QtWidgets.QMessageBox.information(self, 'Error', e)  
  
 @staticmethod  
 def check\_len(text: str, count: int) -> bool:  
 if len(text) >= count:  
 return True  
 return False  
  
  
class AuthWindow(QtWidgets.QMainWindow, Auth.Ui\_AuthWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.new\_window = None  
 self.user\_connect = None  
 self.login\_btn.clicked.connect(self.check\_existence)  
 self.registration\_btn.clicked.connect(self.registration\_window)  
 self.help\_btn.clicked.connect(self.help\_window)  
  
 def check\_existence(self):  
 login = self.login\_value.text()  
 password = self.password\_value.text()  
 role = User.convert(  
 self.role\_value.currentText()  
 )  
 profile = None  
  
 if len(login) > 5 and len(password) > 7\  
 and role:  
 profile = SearchDB.check\_user(login, password, role)  
 # Если пользователь существует, то вернет его профиль  
 else:  
 self.error("Форма заполнена неверно")  
  
 if isinstance(profile, dict):  
 self.fill\_user\_class(profile, role)  
  
 def fill\_user\_class(self, profile: dict, role: str):  
 try:  
 self.user\_connect = User(  
 profile['id'],  
 profile['login\_' + role],  
 profile['password\_' + role],  
 profile['name\_' + role],  
 role  
 )  
 self.profile\_window()  
 except Exception as e:  
 self.error(str(e))  
  
 def registration\_window(self):  
 self.new\_window = RegistrationWindow()  
 self.new\_window.show()  
 self.close()  
  
 def help\_window(self):  
 self.new\_window = HelpWindow()  
 self.new\_window.show()  
 self.close()  
  
 def profile\_window(self):  
 if self.user\_connect:  
 self.new\_window = ProfileWindow(  
 self.user\_connect  
 )  
 self.new\_window.on\_load\_profile()  
 self.new\_window.show()  
 self.close()  
  
 def error(self, error: str):  
 QtWidgets.QMessageBox.information(  
 self, 'Error', error  
 )  
  
  
class RegistrationWindow(  
 QtWidgets.QMainWindow, Registration.Ui\_RegistrationWindow  
):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.flag = False  
 self.new\_window = None  
 self.agree\_label.setVisible(False)  
 self.agree\_checkbox.setVisible(False)  
 self.role\_value.currentTextChanged.connect(self.change\_visible\_checkbox)  
 self.agree\_checkbox.stateChanged.connect(self.change\_enable\_btn)  
 self.registration\_btn.setEnabled(False)  
 self.registration\_btn.clicked.connect(self.registration\_user)  
 self.back\_btn.clicked.connect(self.back\_window)  
  
 def registration\_user(self):  
 login = self.login\_value.text()  
 password = self.password\_value.text()  
 secret\_code = self.secret\_code\_value.text()  
 name = self.name\_value.text()  
 role = User.convert(  
 self.role\_value.currentText()  
 )  
  
 if role:  
 PostDB.create\_user(  
 login, password,  
 secret\_code, name,  
 role  
 )  
 self.new\_window = AuthWindow()  
 self.new\_window.show()  
 self.close()  
 else:  
 self.error(  
 " Вы не выбрали роль "  
 )  
  
 def change\_enable\_btn(self):  
 self.flag = not self.flag  
 self.registration\_btn.setEnabled(self.flag)  
  
 def change\_visible\_checkbox(self):  
 value\_combo = self.role\_value.currentText()  
  
 if value\_combo == "Инвестор":  
 self.agree\_label.setVisible(True)  
 self.agree\_checkbox.setVisible(True)  
 else:  
 self.agree\_label.setVisible(False)  
 self.agree\_checkbox.setVisible(False)  
  
 def back\_window(self):  
 self.new\_window = AuthWindow()  
 self.new\_window.show()  
 self.close()  
  
 def error(self, e):  
 QtWidgets.QMessageBox.information(self, 'Error', e)  
  
  
class HelpWindow(QtWidgets.QMainWindow, Help.Ui\_HelpWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.new\_window = None  
 self.send\_btn.clicked.connect(self.check\_data)  
 self.back\_btn.clicked.connect(self.auth\_window)  
  
 def auth\_window(self):  
 self.new\_window = AuthWindow()  
 self.new\_window.show()  
 self.close()  
  
 def check\_data(self):  
 role = User.convert(  
 self.role\_value.currentText()  
 )  
 secret\_code = self.secret\_code\_value.text()  
  
 if role:  
 profile = SearchDB.get\_profile\_by\_secret\_code(  
 secret\_code, role  
 )  
 self.fill\_labels(profile, role)  
 else:  
 self.error("Нужно выбрать роль")  
  
 def error(self, message):  
 QtWidgets.QMessageBox.information(  
 self, 'Error', message  
 )  
  
 def fill\_labels(self, profile: dict, role: str) -> None:  
 if isinstance(profile, dict):  
 login = self.login\_label.text() + profile['login\_' + role]  
 password = self.password\_label.text() + profile['password\_' + role]  
  
 self.login\_label.setText(login)  
 self.password\_label.setText(password)  
 else:  
 self.error(" Повторите попытку ")  
  
  
class ProfileWindow(QtWidgets.QMainWindow, Profile.Ui\_ProfileWindow):  
 def \_\_init\_\_(self, user\_connect):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.new\_window = None  
 self.user\_connect = user\_connect  
 self.back\_btn.clicked.connect(self.back\_window)  
 self.content\_btn.clicked.connect(self.content\_window)  
 self.edit\_data\_btn.clicked.connect(self.edit\_window)  
 self.history\_auth\_btn.clicked.connect(self.history\_window)  
  
 def on\_load\_profile(self):  
 re\_role = User.un\_convert(  
 self.user\_connect.role  
 )  
 self.login\_value\_label.setText(self.user\_connect.login)  
 self.name\_value\_label.setText(self.user\_connect.name)  
 self.role\_value\_label.setText(re\_role)  
  
 self.profile\_filling\_processing\_by\_role(self.user\_connect.role)  
 # Заполнение оставшейся части профиля под роль  
  
 def profile\_filling\_processing\_by\_role(  
 self, role: str  
 ):  
 profile = SearchDB.get\_profile\_by\_id(  
 self.user\_connect.id\_profile, role  
 )  
 if role == "writer":  
 date = profile['date\_of\_born']  
 self.user\_connect.date\_of\_born = date  
 self.date\_value\_label.setText(str(date))  
 self.additional\_btn.setText("Написать новый пост")  
 self.additional\_btn.clicked.connect(self.open\_make\_composition\_w)  
 elif role == "reader":  
 pass  
 elif role == "offer":  
 pass  
  
 def back\_window(self):  
 self.new\_window = AuthWindow()  
 self.new\_window.show()  
 self.close()  
  
 def content\_window(self):  
 self.new\_window = ContentWindow(  
 self.user\_connect  
 )  
 self.new\_window.load\_content()  
 self.new\_window.show()  
 self.close()  
  
 def edit\_window(self):  
 self.new\_window = EditProfileWindow(self.user\_connect)  
 self.new\_window.show()  
 self.close()  
  
 def history\_window(self):  
 pass  
  
 def open\_make\_composition\_w(self):  
 self.new\_window = MakeCompositionWindow(  
 self.user\_connect  
 )  
 self.new\_window.on\_load\_age\_limit()  
 self.new\_window.show()  
 self.close()  
  
 def error(self, error: str):  
 QtWidgets.QMessageBox.information(  
 self, 'Error', error  
 )  
  
  
class EditProfileWindow(  
 QtWidgets.QMainWindow, EditProfile.Ui\_EditProfileWindow  
):  
 def \_\_init\_\_(self, user\_connect):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.new\_window = None  
 self.user\_connect = user\_connect  
 self.edit\_btn.clicked.connect(self.send\_edit\_data)  
 self.back\_btn.clicked.connect(self.profile\_window)  
  
 def send\_edit\_data(self):  
 login = self.login\_value.text()  
 password = self.password\_value.text()  
 name = self.name\_value.text()  
 date = self.date\_value.date().toPyDate()  
  
 response = self.validate\_data\_for\_relevance(  
 login, password, name, date  
 )  
 success = None  
  
 if response == "Невозможный логин":  
 self.error(response)  
 else:  
 secret\_code = self.secret\_code\_value.text()  
 success = SearchDB.check\_user\_by\_secret\_code(  
 secret\_code, self.user\_connect.role  
 )  
  
 if success:  
 print("Замена успешна")  
 else:  
 self.error("Неверный секретный ключ")  
  
 def profile\_window(self):  
 self.new\_window = ProfileWindow(  
 self.user\_connect  
 )  
 self.new\_window.on\_load\_profile()  
 self.new\_window.show()  
 self.close()  
  
 def validate\_data\_for\_relevance(  
 self, login: str,  
 password: str, name: str,  
 date  
 ) -> str:  
  
 try:  
 date\_of\_born = self.user\_connect.date\_of\_born  
 list\_values = []  
  
 if SearchDB.check\_user\_by\_login(  
 login, self.user\_connect.role  
 ):  
 if len(login) >= 6:  
 list\_values.append("login")  
 if len(password) >= 8:  
 list\_values.append("password")  
 if len(name) >= 3:  
 list\_values.append("name")  
 if date != date\_of\_born:  
 list\_values.append("date")  
  
 return ", ".join(list\_values)  
 else:  
 return "Невозможный логин"  
 except Exception as e:  
 print(e)  
  
 def error(self, error: str):  
 QtWidgets.QMessageBox.information(  
 self, 'Error', error  
 )  
  
  
class ContentWindow(QtWidgets.QMainWindow, Content.Ui\_ContentWindow):  
 def \_\_init\_\_(self, user\_connect):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.new\_window = None  
 self.label\_composition = None  
 self.user\_connect = user\_connect  
 self.content\_btn.clicked.connect(self.profile\_window)  
  
 def profile\_window(self):  
 self.new\_window = ProfileWindow(  
 self.user\_connect  
 )  
 self.new\_window.on\_load\_profile()  
 self.new\_window.show()  
 self.close()  
  
 def load\_content(self):  
 try:  
 items = SearchDB.get\_compositions()  
 print(items)  
 for item in items:  
 print(item)  
 name\_writer = SearchDB.get\_name\_by\_id(  
 item['id\_writer']  
 )  
 print(name\_writer)  
 name\_writer = "Автор: " + name\_writer  
 print(name\_writer)  
 genre = "Основной жанр: " + item["main\_genre"]  
 print(genre)  
 left\_label = "\n".join([  
 name\_writer, genre  
 ])  
 print(left\_label)  
  
 name = "Название произведения: " + item['name\_composition']  
 print(name)  
 age = "Ограничение возраста: " + SearchDB.get\_age\_limit\_by\_id(  
 item['id\_age\_limit']  
 )  
 print(age)  
 composition = item['text\_composition']  
 print(composition)  
 text\_area = "\n".join([  
 name, age, composition  
 ])  
 print(text\_area)  
  
 item\_widget = ItemClass.Item(  
 left\_label, text\_area,  
 self.user\_connect, item['id\_writer']  
 )  
 item\_layout = QtWidgets.QVBoxLayout()  
 item\_layout.addWidget(item\_widget)  
 self.item.addLayout(item\_layout)  
 self.item.setAlignment(Qt.AlignTop)  
 self.scroll\_area\_widget\_contents.setLayout(self.item)  
 except Exception as e:  
 print(e)  
  
  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 window = AuthWindow()  
 window.show()  
 app.exec\_()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

from PyQt5 import QtWidgets  
from PyQt5.QtCore import pyqtSignal  
import UserAccount  
import main  
import FullComposition  
  
  
class FullCompositionWindow(  
 QtWidgets.QMainWindow, FullComposition.Ui\_FullComposition  
):  
 def \_\_init\_\_(self, text\_composition):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.text\_composition = text\_composition  
 self.text.setStyleSheet(  
 "font: 10pt \"Lucida Sans\";"  
 "color: #2121F2;"  
 "background-color: #F5E4FA;"  
 "border-bottom: 2px solid #4169E1;"  
 )  
  
 def load\_text(self):  
 self.text.setText(self.text\_composition)  
 self.text.setReadOnly(True)  
  
  
class UserAccountWindow(QtWidgets.QMainWindow, UserAccount.Ui\_UserAccount):  
 def \_\_init\_\_(self, id\_writer):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.id\_writer = id\_writer  
  
 def load\_user\_account(self):  
 writers = main.SearchDB.get\_writers()  
 for writer in writers:  
 if writer['id'] == self.id\_writer:  
 self.login\_input\_label.setText(writer['name\_writer'])  
 self.date\_input\_label.setText(str(writer['date\_of\_born']))  
 self.status\_input\_label.setText(str(writer['work\_experience']))  
  
  
class QLabel\_click(QtWidgets.QLabel):  
 clicked = pyqtSignal()  
  
 def \_\_init\_\_(self, parent=None):  
 super(QLabel\_click, self).\_\_init\_\_(parent)  
  
 def mouseReleaseEvent(self, event):  
 self.clicked.emit()  
  
  
class QTextEdit\_click(QtWidgets.QTextEdit):  
 clicked = pyqtSignal()  
  
 def \_\_init\_\_(self, parent=None):  
 super(QTextEdit\_click, self).\_\_init\_\_(parent)  
  
 def mouseReleaseEvent(self, event):  
 self.clicked.emit()  
  
  
class Item(QtWidgets.QFrame):  
  
 def \_\_init\_\_(self, name, text,  
 user\_connect, id\_writer):  
 super().\_\_init\_\_()  
 self.user\_connect = user\_connect  
 self.id\_writer = id\_writer  
 self.mainLayout = QtWidgets.QHBoxLayout()  
 self.left\_label = QLabel\_click(self)  
 self.left\_label.setMinimumWidth(200)  
 self.left\_label.setText(name)  
 self.left\_label.setWordWrap(True)  
 self.left\_label.setStyleSheet("QLabel {"  
 "border: 1px solid blue;"  
 "border-radius: 20px;"  
 "font: 10pt \"Lucida Sans\";"  
 "color: #2121F2;"  
 "background-color: #ECCEF5;"  
 "}"  
 "\n"  
 "QLabel:hover {"  
 "border: 3px solid blue;"  
 "}"  
 )  
  
 self.left\_label.clicked.connect(self.click\_label)  
 self.right\_label = QTextEdit\_click(self)  
 self.right\_label.setStyleSheet(  
 "font: 10pt \"Lucida Sans\";"  
 "color: #2121F2;"  
 "background-color: #F5E4FA;"  
 "border-bottom: 2px solid #4169E1;"  
 )  
 self.right\_label.setText(text)  
 self.right\_label.setReadOnly(True)  
 self.right\_label.clicked.connect(self.click\_text\_edit)  
 self.mainLayout.addWidget(self.left\_label)  
 self.mainLayout.addWidget(self.right\_label)  
 self.setLayout(self.mainLayout)  
 self.new\_window = -1  
  
 def click\_label(self):  
 try:  
 self.new\_window = UserAccountWindow(  
 self.id\_writer  
 )  
 self.new\_window.load\_user\_account()  
 self.new\_window.show()  
 except Exception as e:  
 print(e)  
  
 def click\_text\_edit(self):  
 self.new\_window = FullCompositionWindow(  
 self.right\_label.toPlainText()  
 )  
 self.new\_window.load\_text()  
 self.new\_window.show()