Criterion C: Product development

Overview

My application, MovieFans, mainly includes the linkage among three databases, Baidu API, crawler, and GUI codes. In order to achieve certain functions, techniques like sorting, selection, and dictionaries are also used to build the product. Eventually, my product was converted into exe form, and the conversion of this process is shown in Appendix D.

Main techniques

1. GUI interface: Qt design
    1. Interface design
        1. Main interface
        2. Dialogue interface
        3. Widget interface
    2. Verifications
    3. Creating function connectors
    4. Adjusting previews
    5. Adding on-screen help
    6. Python & QT conversion
2. Database:
    1. User database
        1. Create a database
        2. Insert items into the database
        3. Check the appropriateness of user information
            1. Registration
            2. Login
    2. Local database
        1. Create a database
        2. Insert items into the database
        3. List items
    3. History database
        1. Create a database
        2. Insert items into the database
        3. List items
3. Existing modules:
    1. Crawler
    2. Baidu API[1]
    3. Hashlib
4. Basic techniques:
    1. Sorting
    2. Selection
    3. Dictionaries
    4. Security(Hash+Hide passwords)

[1] Appendix F

Relationships between Py files

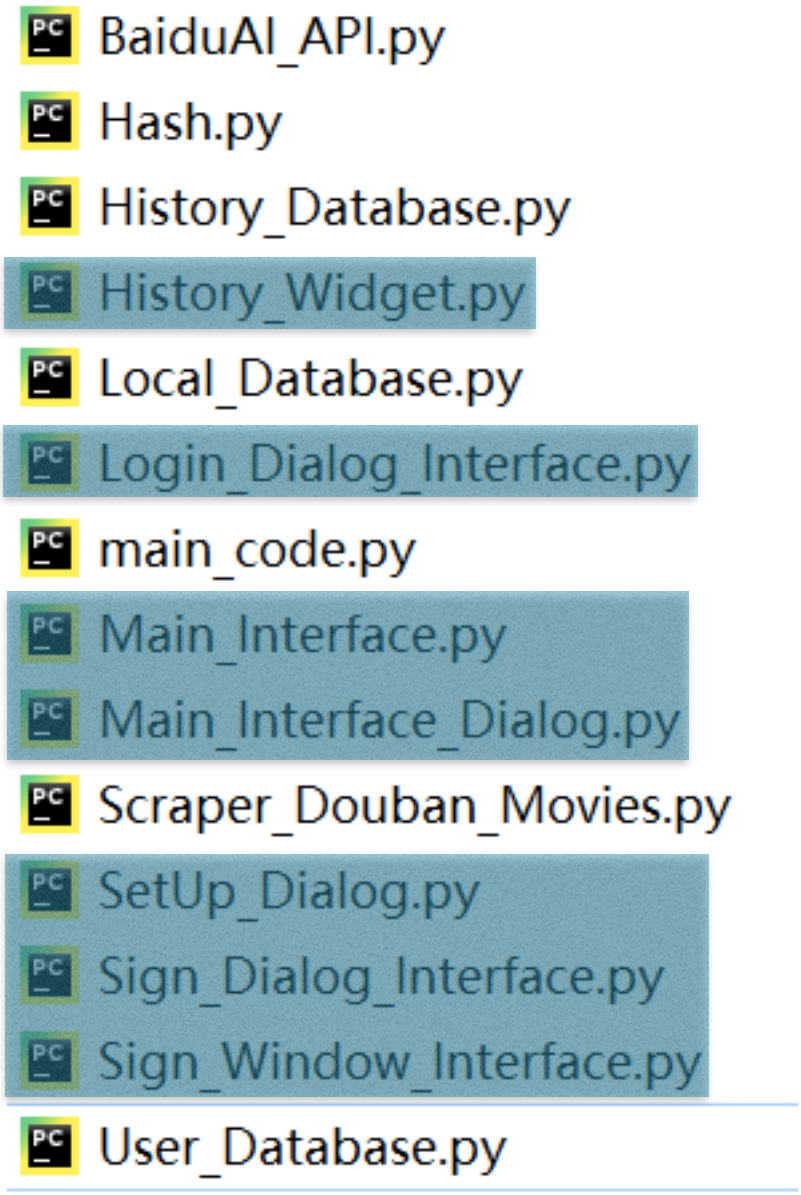

Figure 1: All Py files used to build up my product[2]

[2] The file names that are covered by blue rectangles are the Py files generated through the conversion between Python & QT

Table 1: File name and tagged name translation

File name shown in Figure 1	Illustration of the function of each file
	Module_Function_Py file name_Qtpyconversion; Module_Function_Py file name_Subfunction(The function that is referenced in main_code)
BaiduAI_API	Function_API_BaiduAI_API_Translatephoto
Hash	User system_Encryption_Hash_UserName_Hash
History_Database	Function_Database_History_Database_sqlite_insert Function_Database_History_Database_sqlite_list
History_Widget	Function_Display_History_Widget_Qtpyconversion
Local_Database	Function_Database_Local_Database_sqlite_insert Function_Database_Local_Database_sqlite_list
Login_Dialog_Interface	User system_Login_Login_Dialog_Interface_Qtpyconversion
Main_Interface	Interface_Main_Interface_Qtpyconversion
Main_Interface_Dialog	User system_Text input_Main_Interface_Dialog_Qtpyconversion
Scraper_Douban_Movies	Function_Crawler_Scraper_Douban_Movies_do_cralwer_book
SetUp_Dialog	User system_Configuration_SetUp_Dialog_Qtpyconversion
Sign_Dialog_Interface	User system_Registration_Sign_Dialog_Interface_Qtpyconversion
Sign_Window_Interface	User system_Sign & Login_Sign_Window_Interface_Qtpyconversion
User_Database	User system_Database_User_Database_sqlite_insert User system_Registration_User_Database_sqlite_find User system_Login_User_Database_sqlite_login

The following diagram represents the relationships between each Py file and referenced function.

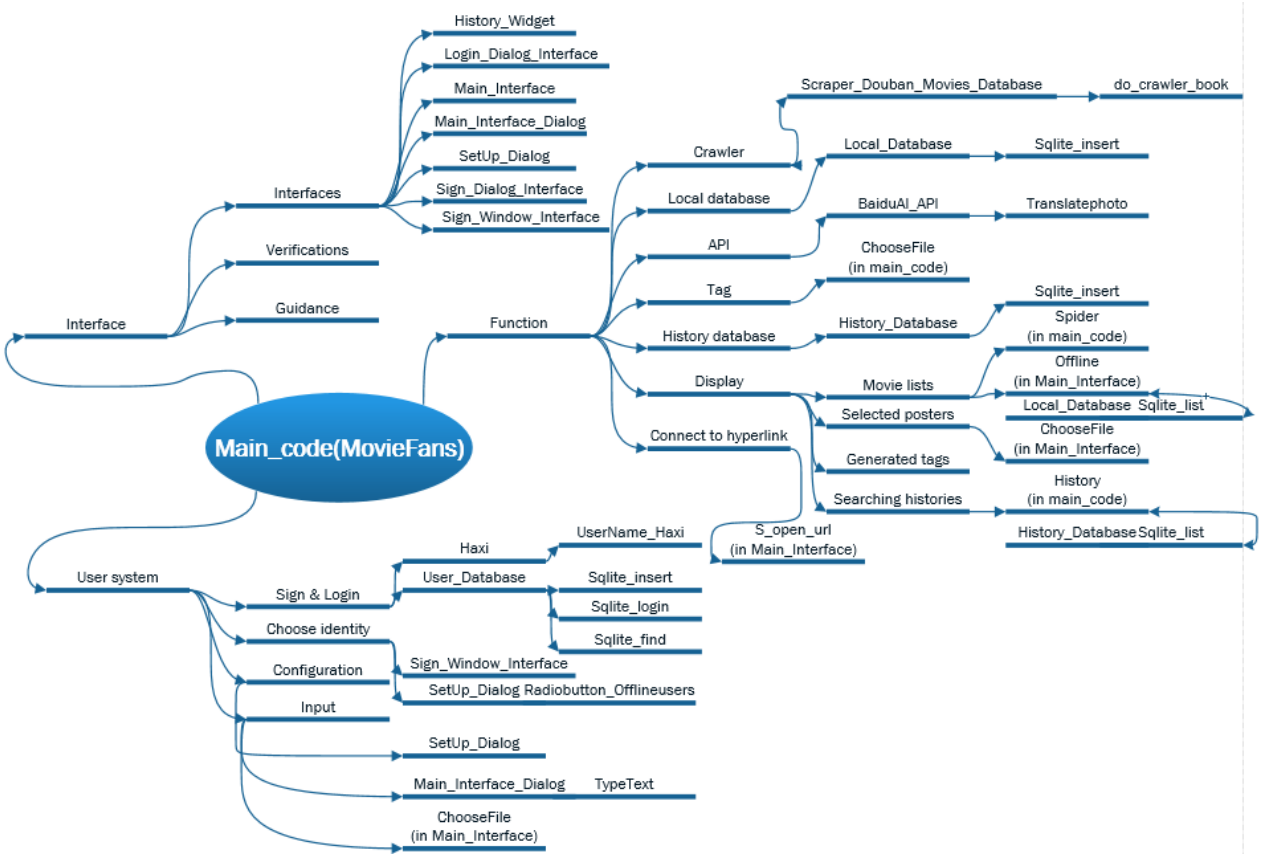

Figure 2: Linkage between Py files and functions

Progress of versions

Versions	Main functions
Version 1	- User authentication - Verifications - Get and movie lists by text input
Version 2	- Photo recognition - Verifications - Create tags - Get and movie lists by photo input
Version 3	After the second interview with my client[2], I made the following adjustments: - History function for registered users - Create a local database for offline users - Verifications - Configuration for users - Display posters
Version 4	- On-screen help for function buttons

[3] Appendix B

Main programs

Notification: Given that codes are selective, certain indexes or variables might not be shown or consistent.

QT_GUI interfaces

I chose Qt design(QT) to build my interfaces because QT is:

- More intuitive and convenient than writing codes in Python or other programming languages.
- A good helpmate of Python and is compatible with various operating systems.

With the help of QT, I could achieve the goal of GUI interfaces and adding on-screen help for users.

1. Procedures

Every GUI files designed in QT would go over the following procedures. I will divide all my QT files into three categories—main interfaces, dialogue interfaces, and widget interfaces, and explain each of them using the following procedures. Yet, because adding functions, adjusting previews and converting ui files into py files are the same for every QT files, instead of repeating them in each category, I will illustrate them in:

- 5. creating function connectors
- 7. adjusting previews
- 9. Python_QT_conversion

Since I have created multiple files in each category, and because of limited spaces, I will pick one example as an explanation in each section.

Figure 3: Main procedures of using QT to achieve functions

2. Main interfaces

(a)Create a file

To design a main interface, I chose to use ‘Main Window’ in QT because it is in an appropriate size format of my design.

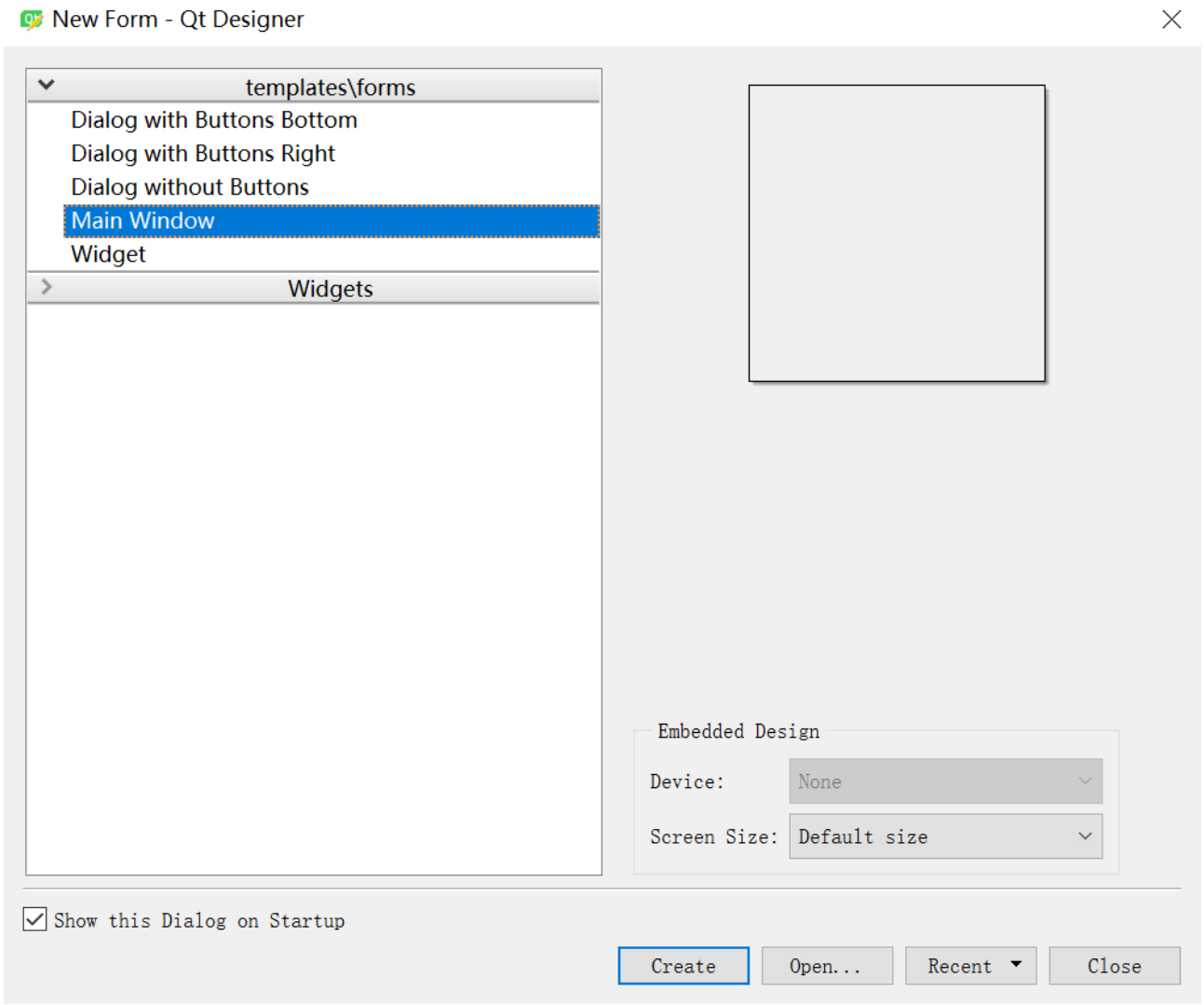

Figure 4: Create a main interface file in QT

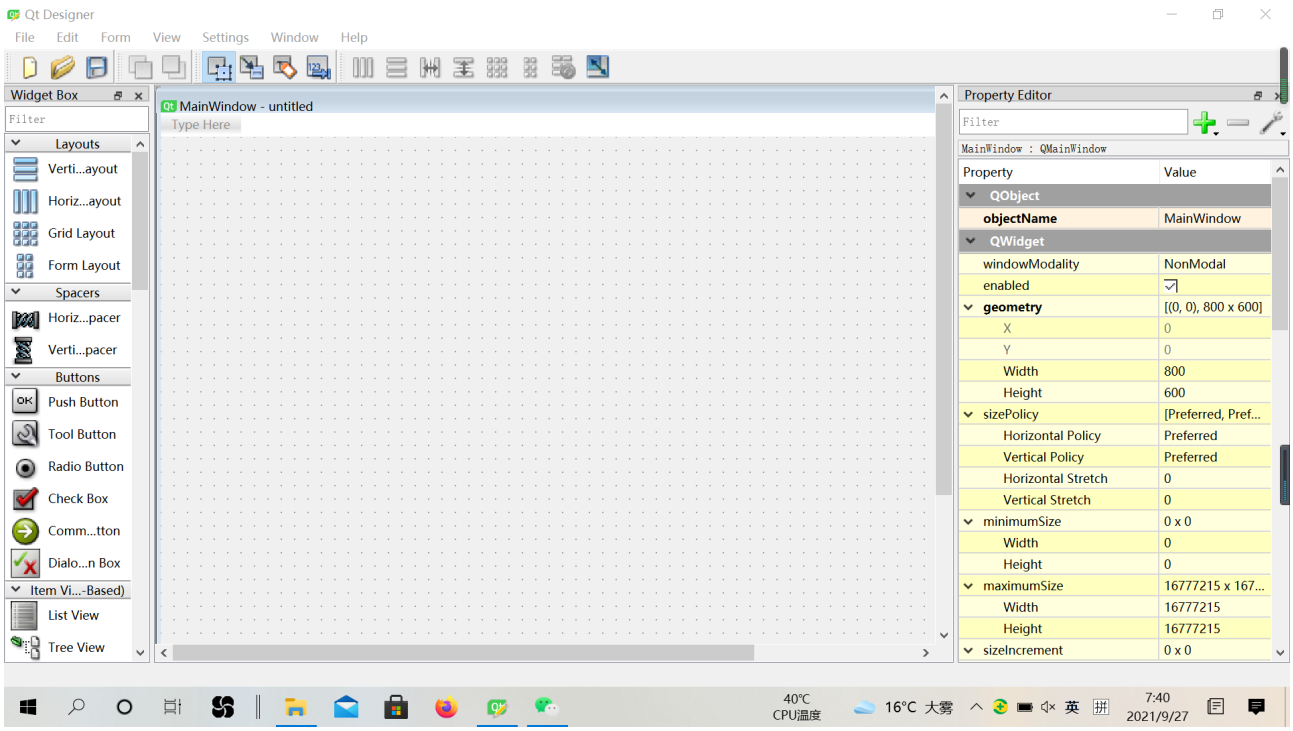

Figure 5: A main interface file in QT

(b)Relevant tools

To achieve the functions of ‘text input’, ‘file input’, ‘show history’, and ‘change configuration’, I added ‘Tool Button’ function for users to click their wanted choices. And because I would display several results on the ‘main interface’ as well, I added ‘Table Widget’ function to prepare for further creating forms as a means of showing results.

Figure 6: Relevant tools used in designing main interfaces in QT

3. Dialogue interfaces

This section will use ‘sign in and login interface’ as an example.

(a)Create a file

To design sign in and login interfaces, I chose to use ‘Dialog with Buttons Bottom’ interface in QT because it is inserted certain functions to achieve user interactions. I will use ‘sign in and login interface’ as an example.

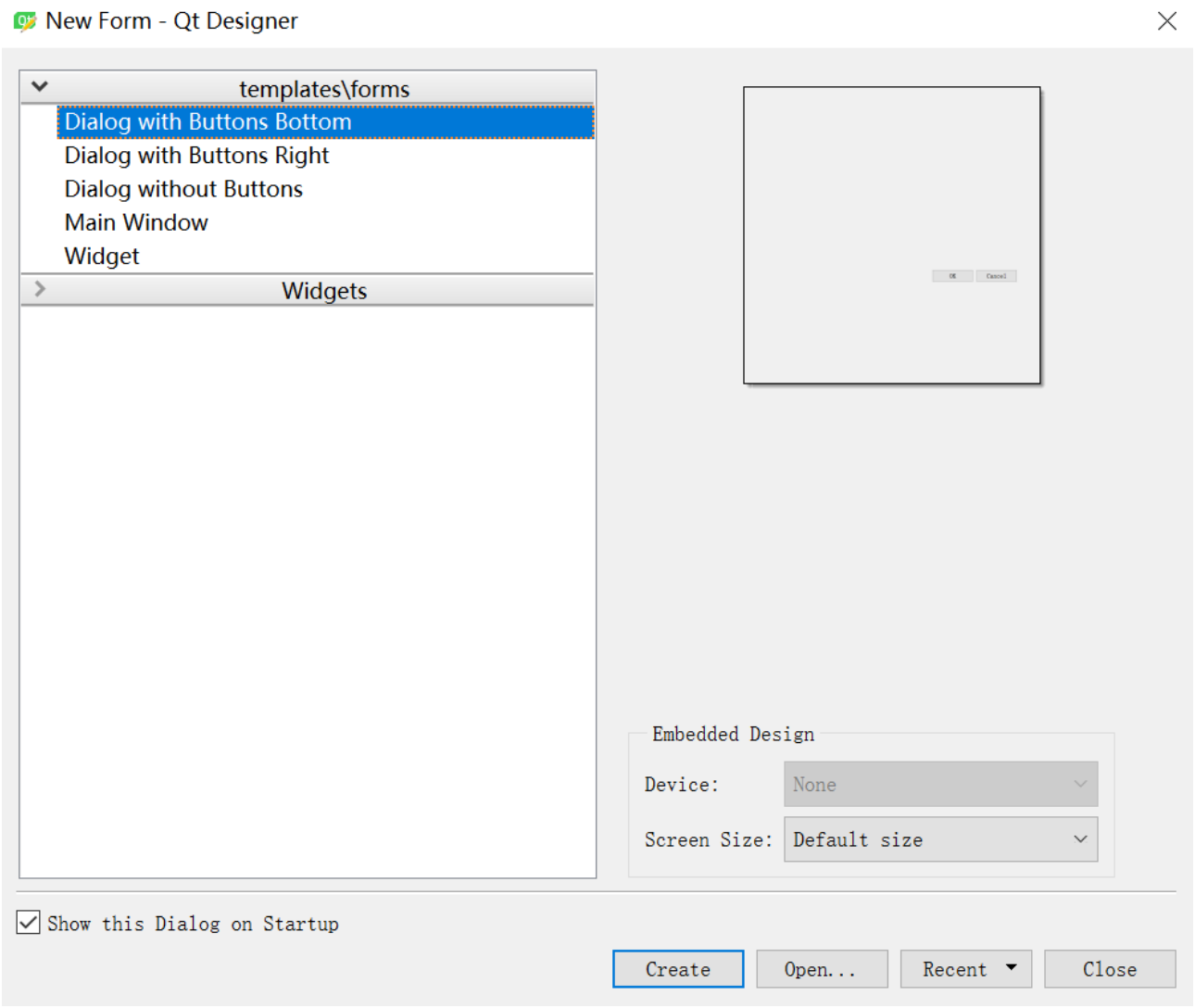

Figure 7: Create a dialogue interface file in QT

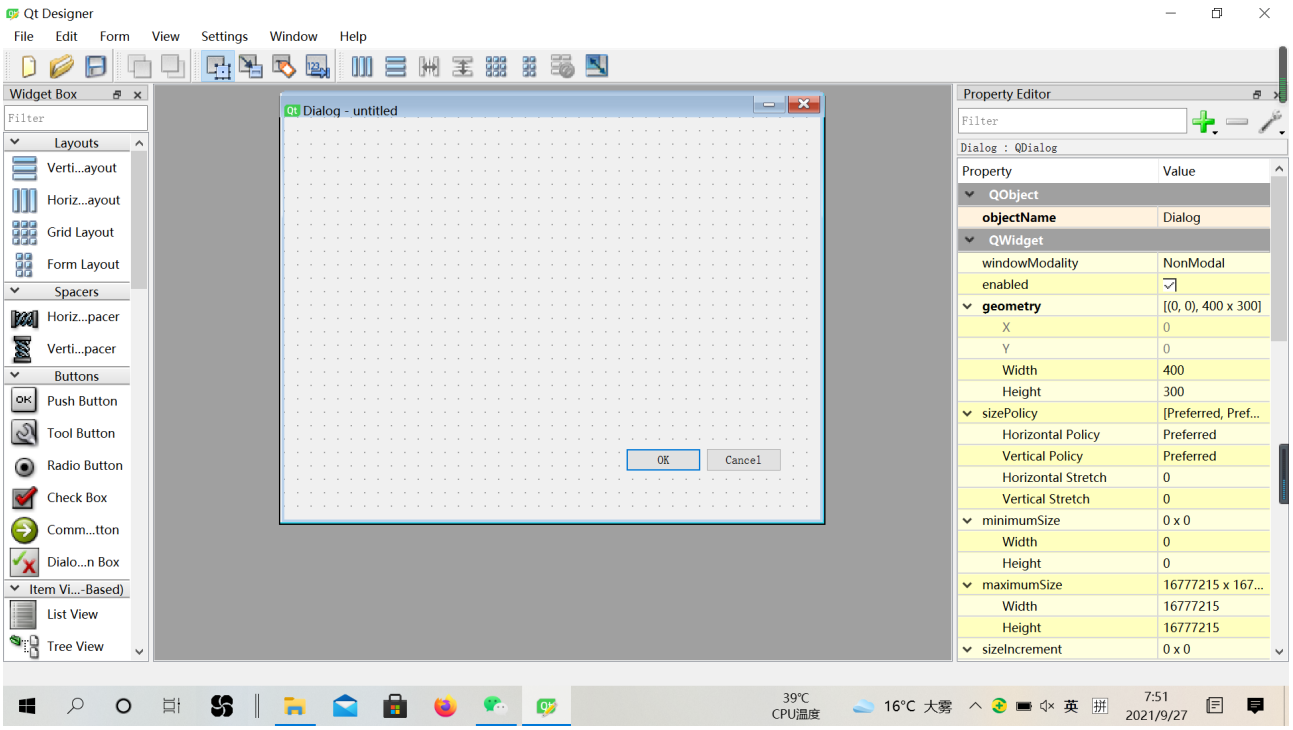

Figure 8: A dialog interface file in QT

(b)Relevant tools

I added ‘Label’ and ‘Line Edit’ functions to give information and get user inputs in dialogues.

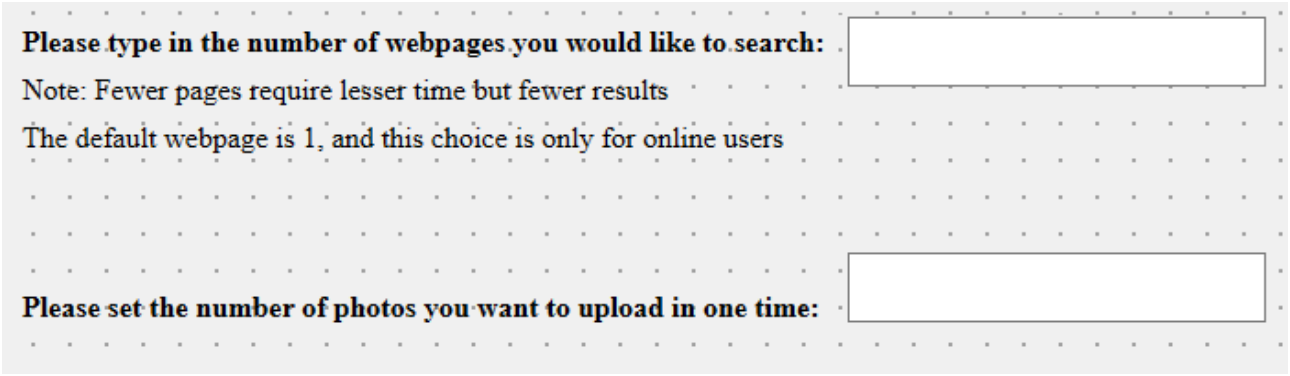

Figure 9: Relevant tools used in designing dialogue boxes in QT

4. Widget interfaces

This section will use ‘history widget’ as an example.

(a)Create a file

To display the searching results and history results, I chose to use ‘Widget’ interface in QT because it allows me to generate a form to better display my results.

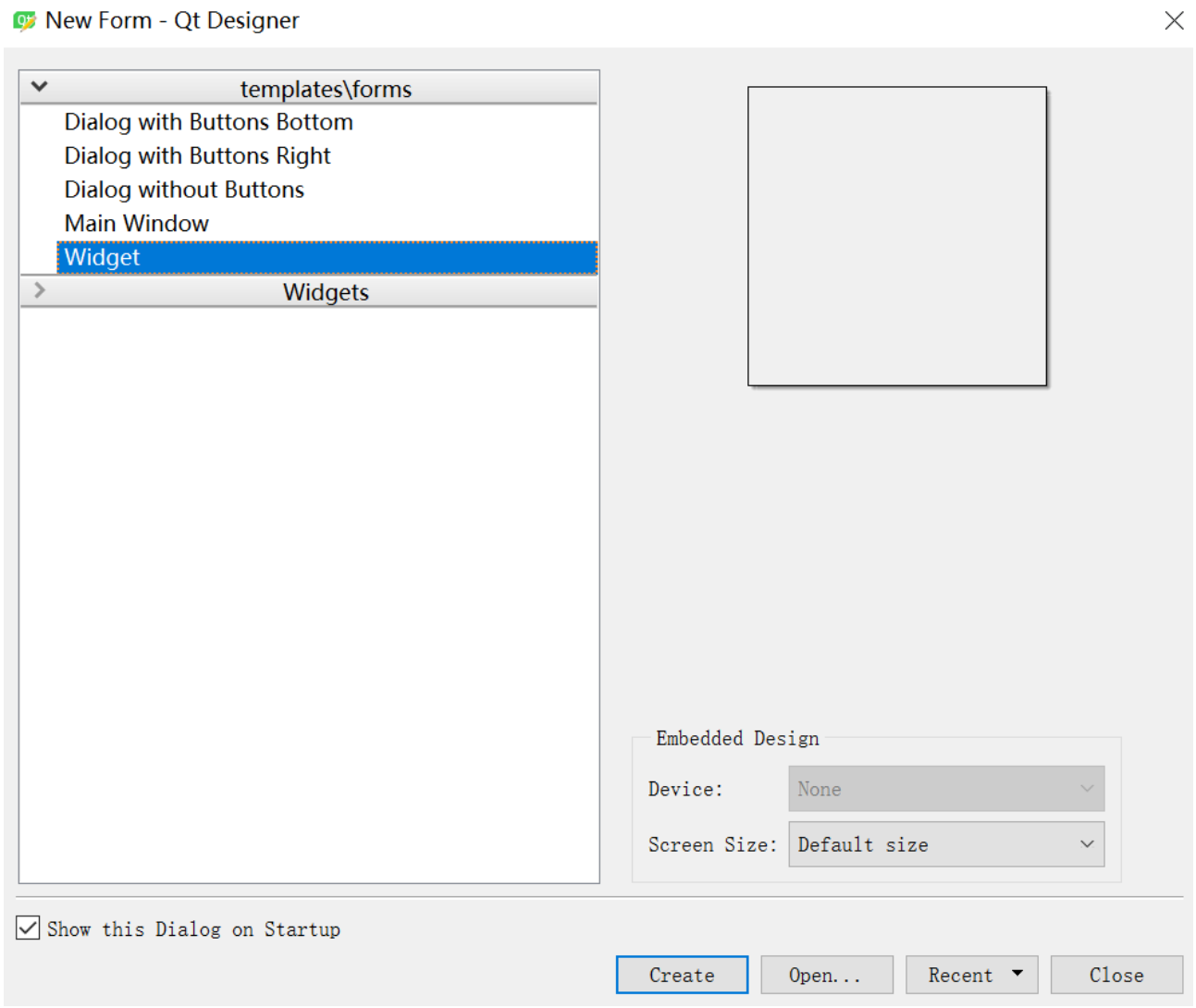

Figure 10: Create a widget interface file in QT

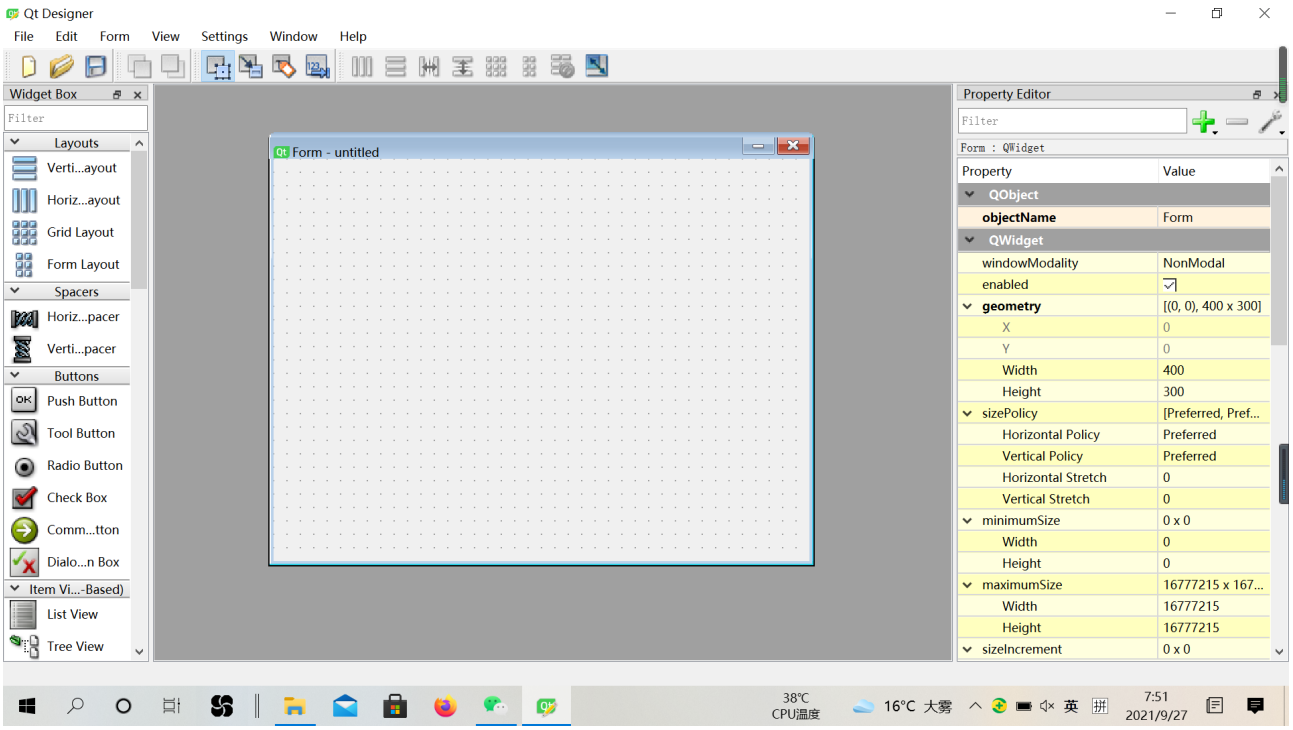

Figure 11: A widget interface file in QT

(b)Create a form in a widget interface

It is not enough to create a widget file. In order to insert items and display them in widget interfaces, I have inserted raws and columns and initialized the word fonts of headings.

```
# Function_History: For signed users, when they click on 'History' button, history results will be displayed
def History(self):
    user_history_db = []
    # Set tables in a defined arrangement to better display the results
    history_result.tableWidget_Historyresult.setColumnCount(6)
    history_result.tableWidget_Historyresult.setHorizontalHeaderLabels(['username', 'local_time', 'photo_url', 'tag', 'movie_name', 'movie_url'])
    history_result.tableWidget_Historyresult.horizontalHeader().setFont(QFont('song', 8))
    history_result.tableWidget_Historyresult.setFont(QFont('song', 8))
    history_result.showMaximized()
```

Figure 12: An illustration code for creating a form in a widget interface

After initializing the forms, I inserted items into them.

```
history_db_list = History_Database.sqlite_list()
for i in range(0,len(history_db_list)):
    # Select history movie results of a particular user from history database
    if history_db_list[i][0] == username_Hash_database:
        user_history_db.append(history_db_list[i])
    else: # Interface_Error verification: Remind users that they haven't had history results yet
        history_result.tableWidget_Historyresult.setItem(0, 0, QTableWidgetItem('No history found.'))

history_result.tableWidget_Historyresult.setRowCount(0)
history_result.tableWidget_Historyresult.clearContents() # Redundant history results will be removed each time

# Display results
for j in range(0, len(user_history_db)):
    history_result.tableWidget_Historyresult.insertRow(j)
    global username
    history_result.tableWidget_Historyresult.setItem(j, 0, QTableWidgetItem(username))
    history_result.tableWidget_Historyresult.setItem(j, 1, QTableWidgetItem(user_history_db[len(user_history_db)-j-1][1]))
    history_result.tableWidget_Historyresult.setItem(j, 2, QTableWidgetItem(user_history_db[len(user_history_db)-j-1][2]))
    history_result.tableWidget_Historyresult.setItem(j, 3, QTableWidgetItem(user_history_db[len(user_history_db)-j-1][3]))
    history_result.tableWidget_Historyresult.setItem(j, 4, QTableWidgetItem(user_history_db[len(user_history_db)-j-1][4]))
    history_result.tableWidget_Historyresult.setItem(j, 5, QTableWidgetItem(user_history_db[len(user_history_db)-j-1][5]))
```

Figure 13: An illustration code for inserting items into an initialized widget form

5. Creating function connectors

I used slots on buttons in QT to prepare for further programming the functions in PyCharm to enable the functionalities. The programming codes will be shown in 9. Python_QT_conversion.

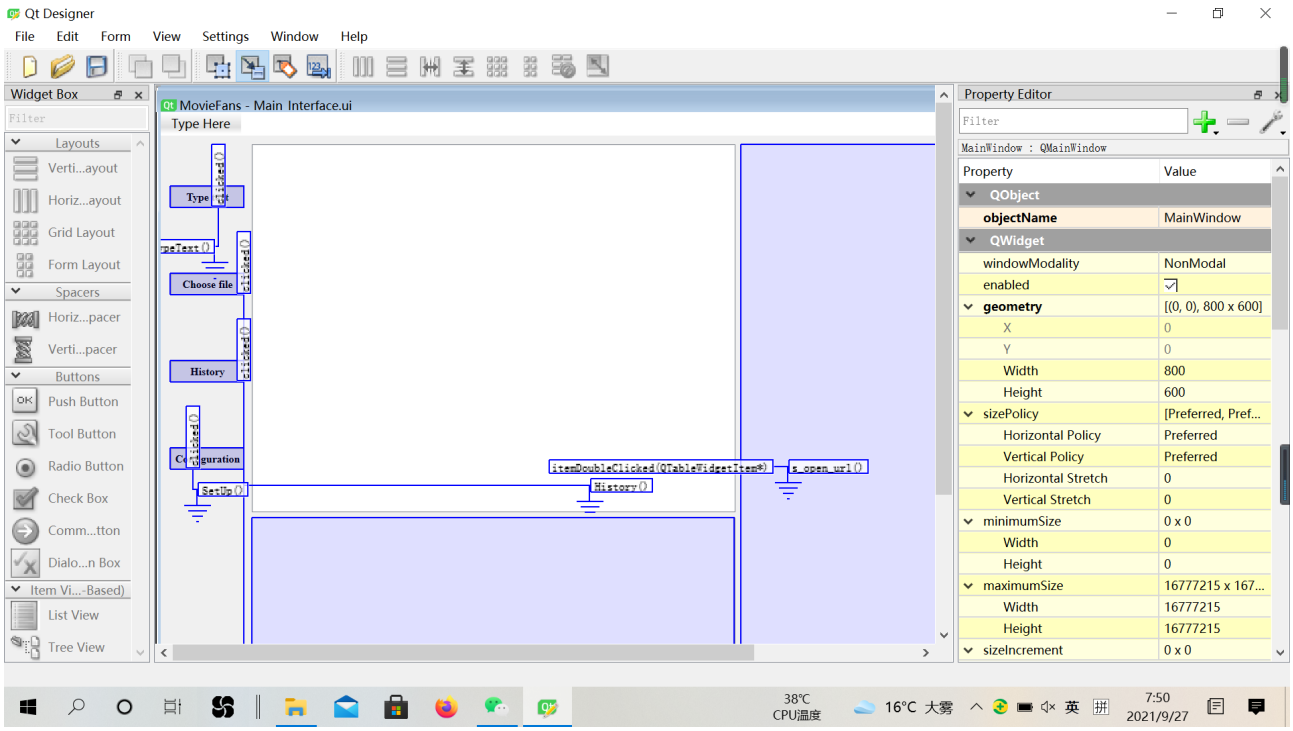

Figure 14: Using slot function in QT to create connectors

6. Verifications

(a)Successful verification

(i) Successful verification

When users successfully create an account, a successful verification will be shown. I used QTMessage to achieve this because it could pop up with an icon showing warning, and it has buttons to fulfill user interaction.

```
QMessageBox.warning(self, 'Congratulation', 'Registered successfully!', QMessageBox.Yes | QMessageBox.No)
```

Figure 15: Codes of successful verifications

(b)Error verifications & reminders

When users mis-operate certain functions, an error verification or a reminder will pop up. I also used QTMessage to achieve this by adjusting certain words from the successful verification.

```
QMessageBox.warning(self, 'Error', 'Username is unavailable. Please design a new one.',
                    QMessageBox.Yes | QMessageBox.No)
```

Figure 16: Codes of error verifications

7. Adjusting previews

Because the interface layouts created manually might not be appropriate and adaptive after magnifying when users actually use the system, I chose to use the 'Horizontal Spacer', 'Vertical Spacer' and 'Preview' function in QT to achieve a clear and aesthetic interface. Spacers could fulfill this because it is a built-in tool to enhance the layouts of QT windows. The way to adjust previews will be shown by the example of 'main interface' file.

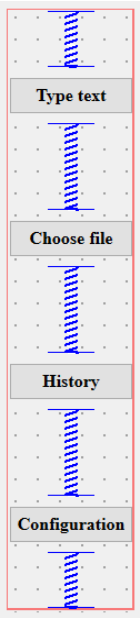

Figure 17: An illustration of spacers in QT

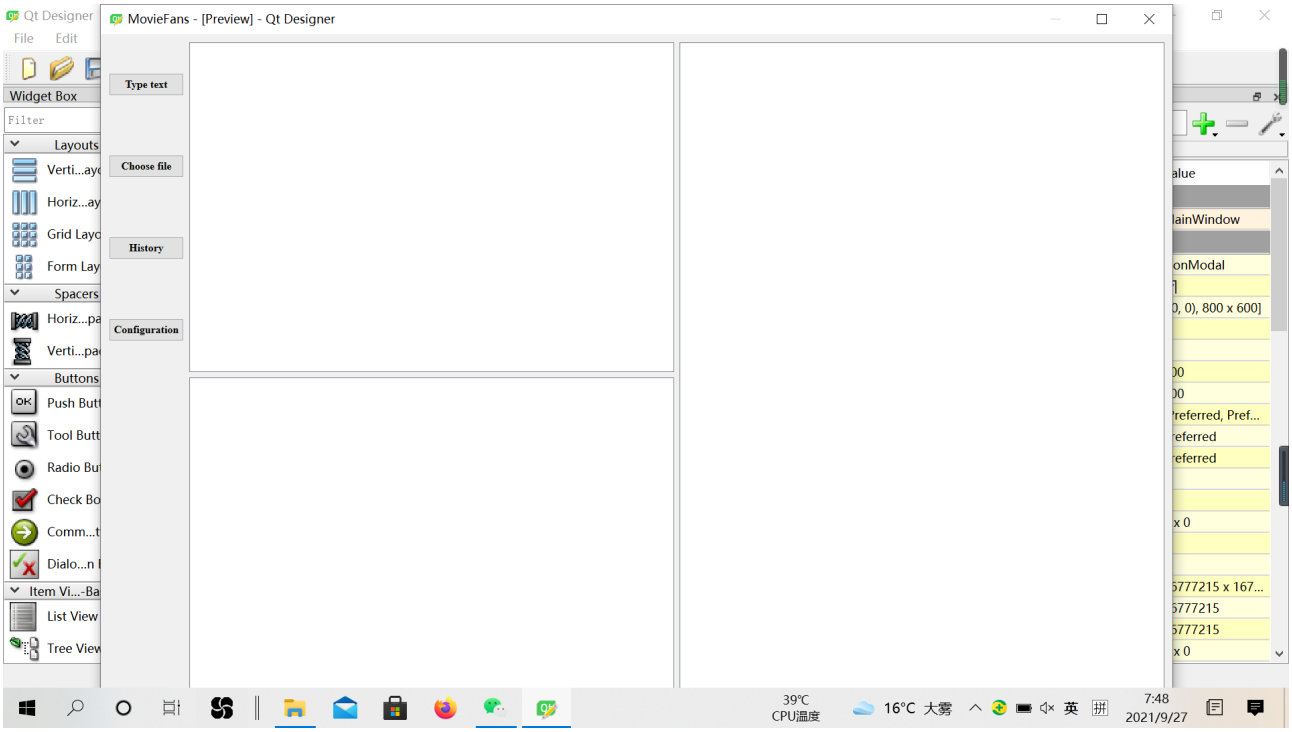

Figure 18: An illustration of the preview function in QT

8. Adding on-screen help

To achieve the function of on-screen help, I used ‘Edit ToolTip’ function on the ‘main interface’ in QT because it is a convenient and easy way to fulfill the goal compared to additional programing.

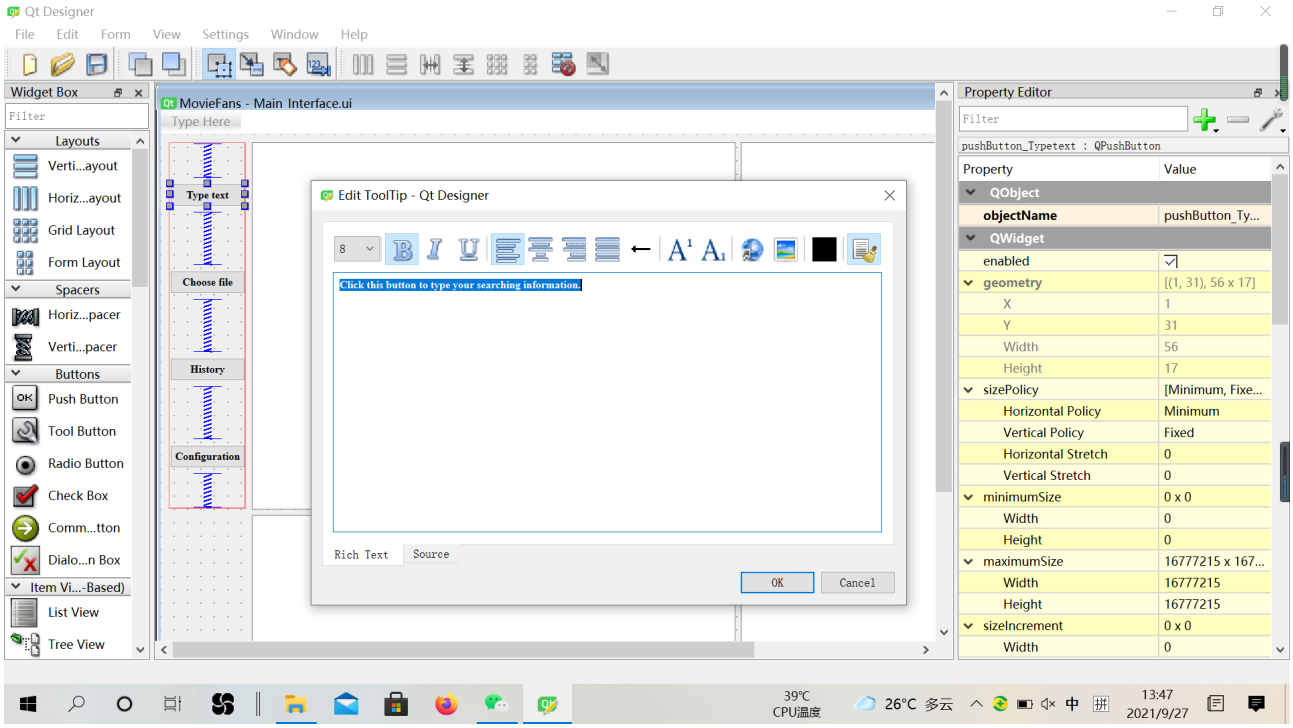

Figure 19: Adding on-screen help in QT

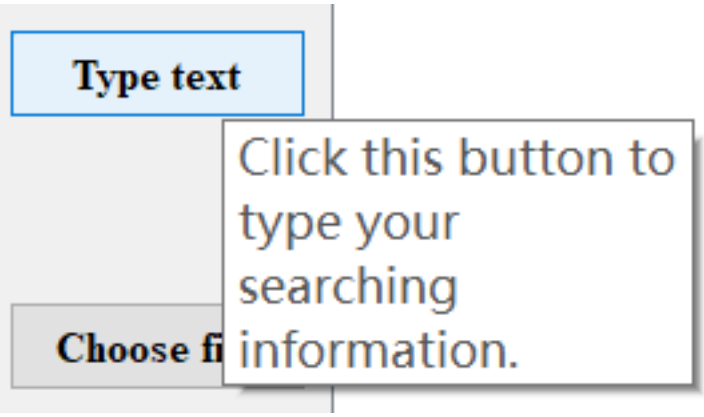

Figure 20: An example of the on-screen help

9. Python_QT_conversion

After designing my interfaces and organizing all the above functions in QT, I needed to convert them in Python with other techniques to build the system. I will use the 'main interface' file to illustrate the conversion.

(1) Select the 'Main interface.ui' file, do a right-click, choose 'External tool', and select 'Main_Interface' to convert the QT interface into Python code.

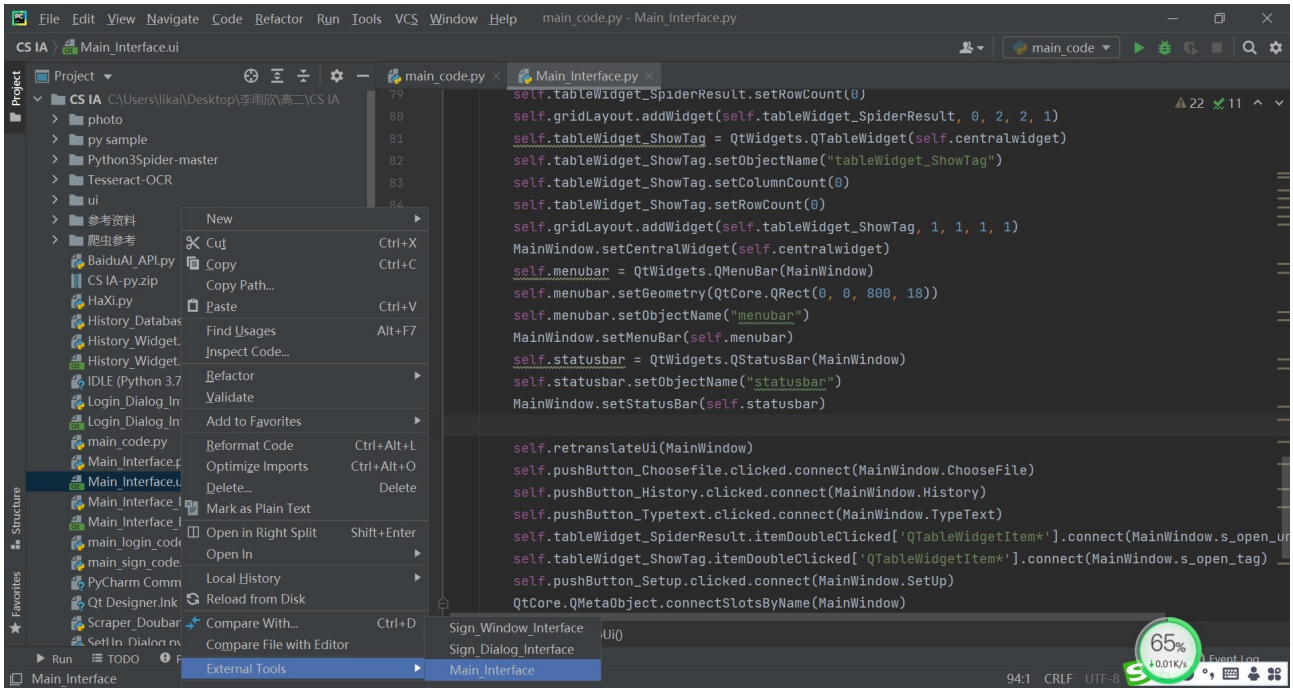

Figure 21: Conversion between Ui files and Py files

(2) Define a class in PyCharm and write the initial state of my PyQt files.

```
# User system_Registered users: Main_Interface Ui: Once the user passes the login process, they will go to main interface
class manage_main_window(Main_Interface.Ui_MainWindow, QtWidgets.QMainWindow):
    def __init__(self):
        super(manage_main_window, self).__init__()
        self.setupUi(self)
        # Function_Result display: Insert tables with a defined arrangement
        self.tableWidget_SpiderResult.setColumnCount(6)
        self.tableWidget_SpiderResult.setHorizontalHeaderLabels(['name','scores','character','date','description','detail_url'])
        self.tableWidget_ShowTag.setColumnCount(1)
        self.tableWidget_ShowTag.horizontalHeader().setVisible(False)

        # Set word fonts
        self.tableWidget_SpiderResult.setFont(QFont('song', 8))
        self.tableWidget_ShowTag.setFont(QFont('song', 8))
        self.tableWidget_SpiderResult.horizontalHeader().setFont(QFont('song', 8))
```

Figure 21: Codes to initialize the state of PyQt files

(3) Define a variable in main line to operate the class.

```
main_window = manage_main_window()
```

Figure 22: Code to operate a class in PyCharm

(4) Define functions and program them in ‘main code’ after the conversion to initiate the functions of buttons with connectors.

```
# Function_Photo input: Users can upload their movie posters after clicking on 'Choose File' button
def ChooseFile(self):
```

```
# Do crawler based on selected tag
def s_open_tag(self, QTableWidgetItem_P):
```

Figure 23: Selective codes to call functions in PyCharm

Database

I created three databases to achieve the functions:

- User authentication
- Movie research under offline condition
- History result illustration

All databases used in my system are Sqlite databases because Sqlite is:

- Convenient to program for they are built-in python
- Capable to fulfill my needs given that the system does not require a heavy database to store loaded items

In this section, the usefulness of using database will be illustrated.

1. User database

By creating a user database, I could store user information to check whether they create an appropriate account in registration process(has the username they designed already been used by someone else) and to check users’ login process(whether they input correct user information).

(a)Create a database

```
import sqlite3

DATEBASE_NAME = 'user.db'

def sqlite_creat_table():
    conn = sqlite3.connect(DATEBASE_NAME)
    print("Opened database successfully")
    # set cursor
    cursor = conn.cursor()

    # create a table
    try:
        sql = 'CREATE TABLE user(username  varchar(50), password varchar(50))'
        cursor.execute(sql)
    except:
        print("table is already available")

    # close cursor
    cursor.close()
    # commit items
    conn.commit()
    # close connection
    conn.close()
```

Figure 23: The creation of user database

(b)Insert items into the database

```
UserName_Sign = self.lineEdit_Username_Sign.text()
Password_Sign = self.lineEdit_Password_Sign.text()
ConfirmPassword_Sign = self.lineEdit_ConfirmPassword_Sign.text()
```

Figure 24: Codes for inserting items into user database

(c)Check the appropriateness of user information

(i)Registration

```
# Check whether the registering username can be already found in database
def sqlite_find(UserName_Sign_p):
    conn = sqlite3.connect(DATEBASE_NAME)
    cursor = conn.cursor()
    cursor.execute("select * from user where username='%s'"%(UserName_Sign_p))
    value = cursor.fetchone()
    cursor.close()
    conn.commit()
    conn.close()

    return value
```

Figure 25: Codes for username check

(ii)Login

```
# Check whether the username and password user login can be found in database
def sqlite_login(UserName_Sign_p,password_p):
    conn = sqlite3.connect(DATEBASE_NAME)
    cursor = conn.cursor()
    cursor.execute("select * from user where username='%s' AND password='%s'"%(UserName_Sign_p,password_p))
    value = cursor.fetchone()
    cursor.close()
    conn.commit()
    conn.close()

    return value
```

Figure 26: Codes to check login information

2. Local database

To fulfill the function of offline movie search, I created a local database. The database will stored all searchings by all users under online condition, so that if users are offline or under low signal condition, the system could ensure the timeliness of their search from local database.

(a)Create a database

```
import sqlite3

DATEBASE1_NAME = 'local.db'

def sqlite_creat_table():
    conn = sqlite3.connect(DATEBASE1_NAME)
    print("Opened database successfully")
    # set cursor
    cursor = conn.cursor()

    # create a table
    try:
        sql = 'CREATE TABLE user(username  varchar(100), time varchar(100), photo_url varchar(100), tag varchar(100), spider_result_name varchar(100), spider_result_scores varchar(100), spider_result_ch
        cursor.execute(sql)
    except:
        print("table is already available")

    # close cursor
    cursor.close()
    # commit items
    conn.commit()
    # stop connection
    conn.close()
```

Figure 27: The creation of local database

(b)Insert items into the database

```
local_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
```

```
spider_result_name = []
spider_result_scores = []
spider_result_character = []
spider_result_date = []
spider_result_description = []
spider_result_url = []

if spider_result != '': # Check whether there are any history results
    for i in range(0, page_size):
        if spider_result[i] == '':
            break
        for j in range(0, len(spider_result[i])):
            if spider_result[i][j]['name'] == '': # Double check: delete invalid crawler results
                continue
            else: # Insert every displayed movie elements into corresponding lists
                spider_result_name.append(spider_result[i][j]['name'])
                spider_result_scores.append(spider_result[i][j]['scores'])
                spider_result_character.append(spider_result[i][j]['character'])
                spider_result_date.append(spider_result[i][j]['date'])
                spider_result_description.append(spider_result[i][j]['description'])
                spider_result_url.append(spider_result[i][j]['detail_url'])
```

Figure 28: Codes for inserting items into local database

(c)List items

To further operate the data stored in local database, I need to list the items in the form of dictionaries.

```
def sqlite_list():
    conn = sqlite3.connect(DATEBASE1_NAME)
    # create a cursor
    cursor = conn.cursor()

    cursor.execute("select * from user")
    value = cursor.fetchall()

    # close the cursor
    cursor.close()
    # commit items
    conn.commit()
    # close connection
    conn.close()

    return value
```

Figure 29: Codes to list items inside local database

3. History database

For online registered users, they could get their history searching results. In order to achieve this function, I created a history database. Unlike the local database, history database only stores selective searching information for an individual and allows repetitions.

(a)Create a database

```
import sqlite3

DATEBASE_NAME = 'history.db'

def sqlite_creat_table():
    conn = sqlite3.connect(DATEBASE_NAME)
    print("Opened database successfully")
    # create a cursor
    cursor = conn.cursor()

    # create a table
    try:
        sql = 'CREATE TABLE user(username  varchar(100), time varchar(100), photo_url varchar(100), tag varchar(100), spider_result_name varchar(100), spider_result_url varchar(100))'
        cursor.execute(sql)
    except:
        print("table is already available")

    # close the cursor
    cursor.close()
    # commit items
    conn.commit()
    # close connection
    conn.close()
```

Figure 30: The creation of history database

(b)Insert items into the database

```
# Function_History: Insert defined elements into history database
for i in range(0, len(spider_result_name)):
    History_Database.sqlite_insert(username_Haxi_database, local_time, photo_url, name, spider_result_name[i], spider_result_url[i])
```

Figure 31: Codes for inserting items into history database

(c)List items

```
def sqlite_list():
    conn = sqlite3.connect(DATEBASE_NAME)
    # create a cursor
    cursor = conn.cursor()

    cursor.execute("select * from user")
    value = cursor.fetchall()

    # close the cursor
    cursor.close()
    # commit items
    conn.commit()
    # close connection
    conn.close()

    return value
```

Figure 32: Codes to list items inside history database

Existing modules

1. Crawler

My system requires a collection of different movie results. To achieve this, I used Crawler module to get movie lists from Douban Movie because after negotiation with my client, this is:

- The most convenient and time efficient way to fulfill our functions
- An efficient way to get movie results

```
# Actually crawl the data
def do_crawl_movie(search_txt: str, page_size: int): # parameter is input
    base_url = f'https://search.douban.com/movie/subject_search?search_text={search_txt}&start=%s'
    spider_movive = []
    for i in range(page_size):
        # response = requests.get(base_url % (i*15),headers=headers)
        # print(response.text)
        html_str = get_response_by_selenium(base_url % (i * 15))
        # print(html_str)
        tree = etree.HTML(html_str)
        div_list = tree.xpath('//div[@id="root"]/div/div[2]/div/div/div')
        # print(div_list)
        # 边界，如果div_list为空，则循环结束
        if not div_list:
            break
        temp = parse_page(div_list)
        spider_movive.append(temp)
    return spider_movive # output
```

Figure 33: Selective codes for doing crawler

2. Baidu API

Because of the need of uploading posters and getting word contents of the posters from my client, I chose the use Baidu API to fulfill the goal. This is also a result of our consultation because as an existing tool, Baidu API:

- Does not require me to do trainings while doing photo recognition
- Has higher accuracy since it is used by multiple users

```
# Function_API: Extract text contents from uploaded photos
def Translatephoto(filename):
    client = AipOcr(APP_ID, API_KEY, SECRET_KEY)
    options = {}
    options["language_type"] = "CHN_ENG"
    options["detect_direction"] = "true"
    options["detect_language"] = "true"
    options["probability"] = "false"
    image = get_file_content(filename)
    result = client.basicGeneral(image, options)
    words_result = result['words_result']

    return words_result
```

Figure 34: Using Baidu API to do photo recognition and word extraction

3. Hashlib

To ensure the data security of user information, I used Hashlib as a tool to encrypt usernames and passwords stored in database files.

```
import hashlib

# Encrypt username
def UserName_Hash(UserName_Sign_p):
    md5 = hashlib.md5()
    md5.update(UserName_Sign_p.encode('utf-8'))
    return md5.hexdigest()

# Encrypt password
def Password_Hash(Password_Sign_p):
    md5 = hashlib.md5()
    md5.update(Password_Sign_p.encode('utf-8'))
    return md5.hexdigest()
```

Figure 35: Codes to encrypt user information

Basic techniques

1. Sorting

Since my client and I agreed to sort the movie results by their scores, I used the sorting technique to reorganize the movie lists gotten from the Internet and the data stored in the local database.

```
            if spider_result[i][j]['scores'] == '':
                sort_score.append(0)
            else:
                sort_score.append(float(spider_result[i][j]['scores']))
    # Get sorted score indexes: The scores of the movies will be sorted from the lowest to the highest, and this variable outputs the indexes of movies of corresponding scores
    sort_score_index = np.argsort(sort_score)

# Function_Result display
Index = 0
for i in range(0, len(spider_result_name)):
    score_index = sort_score_index[len(spider_result_name) - 1 - i]
    main_window.tableWidget_SpiderResult.insertRow(Index)
    main_window.tableWidget_SpiderResult.setItem(Index, 0, QTableWidgetItem(spider_result_name[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 1, QTableWidgetItem(spider_result_scores[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 2, QTableWidgetItem(spider_result_character[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 3, QTableWidgetItem(spider_result_date[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 4, QTableWidgetItem(spider_result_description[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 5, QTableWidgetItem(spider_result_url[score_index]))
    Index += 1
```

Figure 36: Codes for sorting movies

```
# Function_Result display
Index = 0
for i in range(0, len(spider_result_name)):
    score_index = sort_score_index[len(spider_result_name) - 1 - i]
    main_window.tableWidget_SpiderResult.insertRow(Index)
    main_window.tableWidget_SpiderResult.setItem(Index, 0, QTableWidgetItem(spider_result_name[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 1, QTableWidgetItem(spider_result_scores[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 2, QTableWidgetItem(spider_result_character[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 3, QTableWidgetItem(spider_result_date[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 4, QTableWidgetItem(spider_result_description[score_index]))
    main_window.tableWidget_SpiderResult.setItem(Index, 5, QTableWidgetItem(spider_result_url[score_index]))
    Index += 1
```

Figure 37: Codes for result display of online users

2. Selection

The selection technique mainly helps me to:

- Reduce the redundancy
- Enhance the effectiveness of my system

I used the technique to:

- Reduce the redundancy when the system inserts items into the local database to eliminate invalid data extracted from Douban Movie
- Select individual history results from the local database and insert them into the history database.

```
    for j in range(0, len(spider_result[i])):
        if spider_result[i][j]['name'] == '': # Double check: delete invalid crawler results
            continue
        else: # Insert every displayed movie elements into corresponding lists
            spider_result_name.append(spider_result[i][j]['name'])
            spider_result_scores.append(spider_result[i][j]['scores'])
            spider_result_character.append(spider_result[i][j]['character'])
            spider_result_date.append(spider_result[i][j]['date'])
            spider_result_description.append(spider_result[i][j]['description'])
            spider_result_url.append(spider_result[i][j]['detail_url'])
            if spider_result[i][j]['scores'] == '':
                sort_score.append(0)
            else:
                sort_score.append(float(spider_result[i][j]['scores']))
```

Figure 38: Codes for reduction in redundancy and invalid results when inserting into local database

```
history_db_list = History_Database.sqlite_list()
for i in range(0,len(history_db_list)):
    # Select history movie results of a particular user from history database
    if history_db_list[i][0] == username_Hash_database:
        user_history_db.append(history_db_list[i])
    else: # Interface_Error verification: Remind users that they haven't had history results yet
        history_result.tableWidget_Historyresult.setItem(0, 0, QTableWidgetItem('No history found.'))
```

Figure 38: Selection technique used in inserting data into history database

3. Dictionaries

I used dictionaries to organize and store the results extracted from the Douban Movie website for the convenience when doing further operations like selection or insertion to database.

```
# Find the location of each target elements
def parse_page(div_list):
    movies = []
    count = 0
    for div in div_list:
        # The name of the movies
        name_list = get_text(div.xpath('.//div[@class="title"]/a/text()'))
        name_date = name_list.split('\u200e')
        date = ""
        if len(name_date) > 1:
            name = name_date[0]
            date = name_date[1]
        else:
            name = name_date[0]
        # score
        scores = get_text(div.xpath('.//span[@class="rating_nums"]/text()'))
        # description
        info = get_text(div.xpath('.//div[@class="meta abstract"]/text()'))
        # character
        character = get_text(div.xpath('.//div[@class="meta abstract_2"]/text()'))
        detail_url = get_text(div.xpath('.//div[@class="title"]/a/@href'))

        item = {'name': name, 'scores': scores, 'character': character, 'date': date, 'description': info, 'detail_url': detail_url}
        movies.append(item)
        count += 1
    return movies
```

Figure 39: Dictionaries used in Crawler

4. Security_Hide passwords

Besides Hash encryption, I programmed a code to hide passwords when users type in their information during the login or registration process. This could further enhance the security of user information.

```
self.lineEdit_password.setEchoMode(QtWidgets.QLineEdit.Password)
```

Figure 40: Codes to hide user passwords

Word Count: 1052 (Excluding subheadings, bullet points, figures, and footnotes)

References:

Baidu AI cloud OCR Document. (n.d.). Retrieved Jan 15, 2021, from https://bce-cdn.bj.bcebos.com/p3m/pdf/ai-cloud-share/online/OCR/OCR.pdf

Baidu AI cloud OCR API Document. (n.d.). Retrieved Aug 02, 2021, from https://cloud.baidu.com/doc/OCR/s/1k3h7y3db

Beautiful Soup Documentation. (n.d.). Retrieved Oct 03, 2020, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Lawson, R. (2015). Web Scraping with Python. Birmingham: Packt publishing

Liu, C., et al. (2020). A Deep Practice of OCR: Character Recognition Based on Deep Learning. Beijing: China Machine Press

Long, S., He, X., & Yao, C. (2021). Scene Text Detection and Recognition: The Deep Learning Era. International Journal of Computer Vision, 129(1), 161–184. https://doi.org/10.1007/s11263-020-01369-0

Patel, C., Patel, A., & Patel, D. (2012). Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study. International Journal of Computer Applications, 55(10), 50–56. https://doi.org/10.5120/8794-2784

Pycharm生成可执行文件.exe_Joke-CSDN博客_pycharm生成exe. (2019, January 20). Blog.csdn.net. https://blog.csdn.net/qq_32939413/article/details/86564611

Python Standard Library Internet Protocols and Support. (n.d.). Retrieved Sep 24, 2021, from https://docs.python.org/3/library/urllib.html

PyQt（Python+Qt）学习随笔：QTableWidgetItem项数据的data和setData访问方法_老猿Python-CSDN博客. (2020, February 28). Blog.csdn.net. https://blog.csdn.net/LaoYuanPython/article/details/104565545/

PyQt5 QTableWidget 删除所有行_shiyue41的博客-CSDN博客. (2021, January 21). Blog.csdn.net. https://blog.csdn.net/qq_27694835/article/details/112965336

pyqt5 TableWidget 隐藏表头方法_qiya_pk的专栏-CSDN博客_tablewidget隐藏表头. (2020, May 23). Blog.csdn.net. https://blog.csdn.net/qiya_pk/article/details/106303224

Python GUI编程入门(37)-通用的ListView类_面向对象思考-CSDN博客. (2019, November 4). Blog.csdn.net. https://blog.csdn.net/craftsman1970/article/details/102903106

python pyqt5 QTableWidget 显示图片. (2019, July 20). 简书. https://www.jianshu.com/p/dae2a017ac70

python Table Widget的详细用法，在python中画表格_lives_xiaoma的博客-CSDN博客. (2021, June 30). Blog.csdn.net. https://blog.csdn.net/lives_xiaoma/article/details/118360637

Python3 如何检查字符串是否是以指定子字符串开头或结尾_极客点儿-CSDN博客. (2019, March 2). Blog.csdn.net. https://blog.csdn.net/yilovexing/article/details/88071492

python笔记： 用columns修改列名_gene博客-CSDN博客. (2018, September 12). Blog.csdn.net. https://blog.csdn.net/qq_39348113/article/details/82649361

pythongui登录界面密码显示_Python GUI编程：设计一个简单的登录界面_weixin_39934869的博客-CSDN博客. (2021, January 29). Blog.csdn.net. https://blog.csdn.net/weixin_39934869/article/details/113498874

Python通过浏览器打开网页的方法-百度经验. (2019, August 5). Jingyan.baidu.com. https://jingyan.baidu.com/article/15622f24bc6fc4bdfdbea50b.html

python文件选择对话框_韦海涛的博客-CSDN博客_python选择文件. (2017, September 11). Blog.csdn.net. https://blog.csdn.net/Abit_Go/article/details/77938938

Qt使用教程_不当初-CSDN博客_qt使用. (2018, July 19). Blog.csdn.net. https://blog.csdn.net/lk142500/article/details/81109083

QT实战3：QTableWidget表头、内容字体大小、颜色、背景颜色等设置_ydyuse的专栏-CSDN博客_qtablewidget设置表头. (2020, April 2). Blog.csdn.net. https://blog.csdn.net/ydyuse/article/details/105155286

如何获取QTableView列宽 - 问答 - Python中文网. (n.d.). Www.cnpython.com. Retrieved September 6, 2021, from https://www.cnpython.com/qa/422676