Summer training report/ synopsis/ minor project

ON

Quiz Application using Python & MySQL

A project report/synopsis SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE AWARD OF

Bachelor OF engineering

In

Computer science and engineering

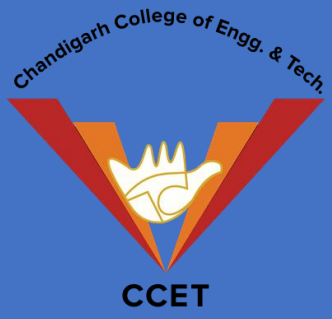
Submitted BY

Ravi Verma

roll NO: CO19352

Under THE SUPERVISION OF

Dr. Ankit Gupta Department of CSE CCET(Degree Wing) Chandigarh



Chandigarh college of engineering and technology

(degree wing)

Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University

, Chandigarh

Sector-26, Chandigarh. PIN-160019

July, 2020

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled " Quiz Application using Python and MySQL", in fulfillment of the requirement for the award of the degree Bachelor of Engineering in Computer Science & Engineering, submitted in CSE Department, Chandigarh College of Engineering & Technology(Degree Wing) affiliated to Punjab University, Chandigarh, is an authentic record of my work carried out during my degree under the guidance of Dr. Ankit Gupta. The work reported in this has not been submitted by me for the award of any other degree or diploma.

Date : 2/08/2020 Ravi Verma

Place : Chandigarh CO19352

CERTIFICATE

This is to certify that the Project work entitled "Quiz Application using Python and MySQL" submitted by Ravi Verma in fulfillment for the requirements of the award of Bachelor of Engineering Degree in Computer Science & Engineering at Chandigarh College of Engineering and Technology (Degree Wing), Chandigarh is an authentic work carried out by him/her under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree.

Date : 2/08/2020

Place : Chandigarh

Dr.Ankit Gupta

Deptt of CSE

CCET(Degree Wing)

Chandigarh

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Ankit Gupta and Chandigarh College Of Engineering and Technology for supporting me throughout my project Quiz application using MySQL and Python. First, I wish to express my sincere gratitude to my mentor Dr. Ankit Gupta, for his enthusiasm, patience, insightful comments, helpful information, practical advice and unceasing ideas that have helped me tremendously at all times in my project. His immense knowledge, profound experience and professional expertise has enabled me to complete this project successfully. Without his support and guidance, this project would not have been possible. I could not have imagined having a better mentor for my project.

I also wish to express my sincere thanks to the Chandigarh College of Engineering and Technology for accepting me into the under graduate program. In addition, I am deeply indebted to the Chandigarh College of Engineering and Technology for sponsoring my Coursera courses. This financial support has enabled me to complete project successfully.

Thanks for all your encouragement!

ABSTRACT

As technology is getting more accessible day by day. Digitization is happening in every sector. With the digitization there arises the demand for new software applications that cater to the needs of the user. One such field is education which has witnessed new approaches to teach students using the Internet and Technology. Online learning is one such approach. Online learning also requires new methods to evaluate student performance. Quiz format tests are easy to evaluate and are perfect to test student's knowledge. But it requires new software solutions to make it possible. One such solution is Quiz application using Python and MySQL. The application is simple yet powerful. It harnesses the power of data base management system and simplicity of Python language. It provides a graphical user interface to its user and provides data exchange capabilities. The quiz is stored in a widespread data format Json and Responses of quiz can be exported out of application in csv file to be analyzed using other applications too. The capabilities of this application can be expanded very easily. It makes it very maintainable. The application can be customized to the needs of organization easily. As the application uses a MySQL database it also has the potential of becoming a web application as well.

List of tables

List of figures

CONTENTS

Course Certificates

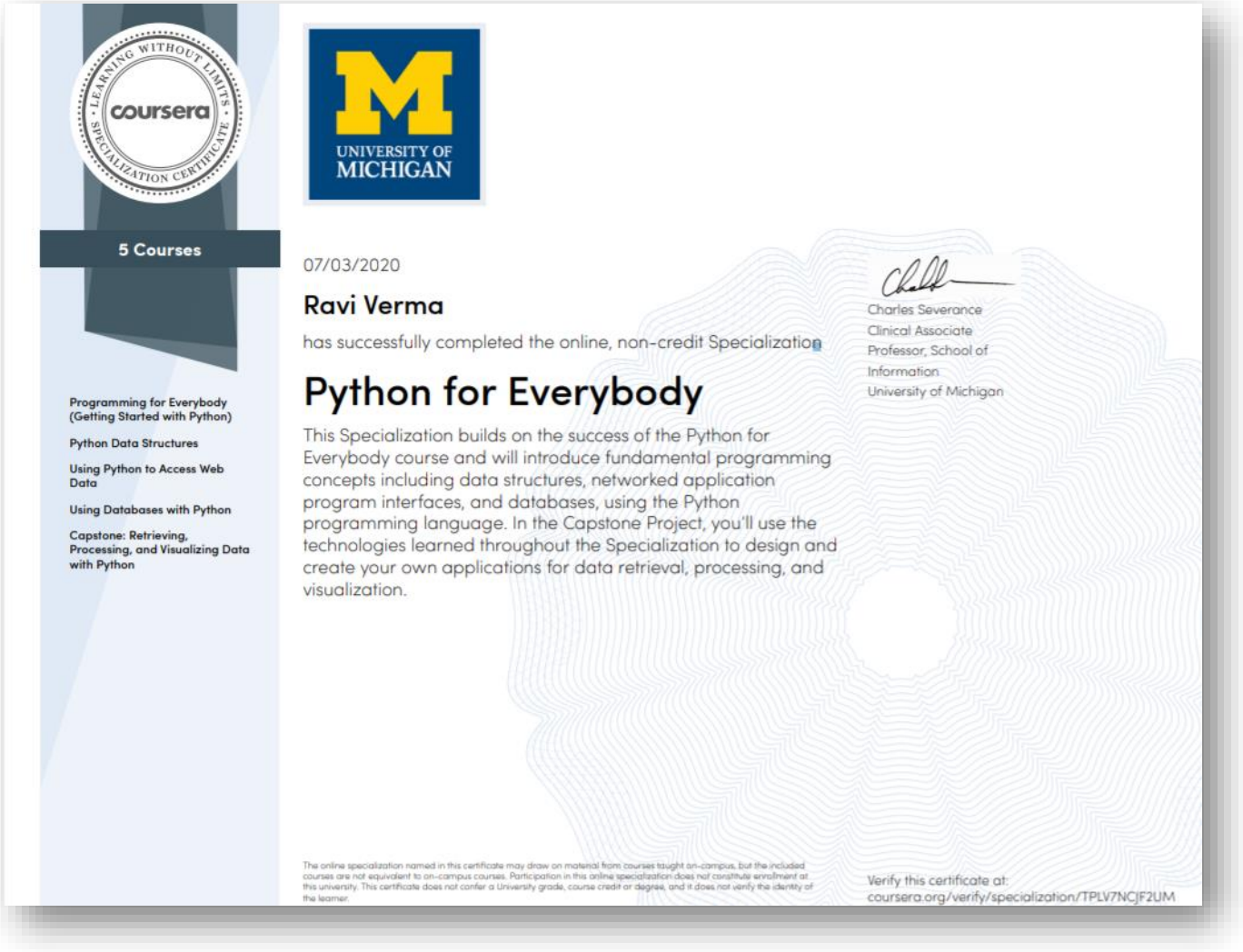


Figure 1 Course certificate

Verify at : https://www.coursera.org/verify/specialization/TPLV7NCJF2UM

Figure 2 course certificate

Verify at https://www.coursera.org/verify/XB5KH5CRT5AL

Introduction

Quiz application is a simple quiz application which takes quiz in digital format. The user has to create an account. Then the user can attempt quiz in the application. The user can also view their result in the quiz application through user dashboard. The result can also be saved into a pdf file. The admin can export all the responses of the quiz into a csv file which enables admin to perform analysis on the data and keep track of the user performances. The user can also reset their password in the user dashboard. It comes with a quiz editor also. Which enables the admin to create, edit and save quiz within the application. The quiz editor offers various key features like setting response limit, scheduling quiz for a particular date and time, setting time limit for the quiz.

The quiz can be divided into multiple sections and multiple questions can be added into any section. The questions are in multiple choice format. The admin can also add points to questions. The admin can set multiple options as answer key. The admin can save quiz in json file which enables the reusability of the data across other applications as well.

The application stores its data in a SQL database which can be hosted on a server and then the application can be turned into an online quiz application without much efforts.

Project Details

Development Environment

Language: Python 3.4

Operating System: Windows 10.

Processor: Intel Pentium.

Architecture: x86

Specifications

Operating System: Windows, Mac and Linux

Processor: Intel Pentium onwards.

Architecture: x86 and x64

RAM: 60 Mb

Project structure

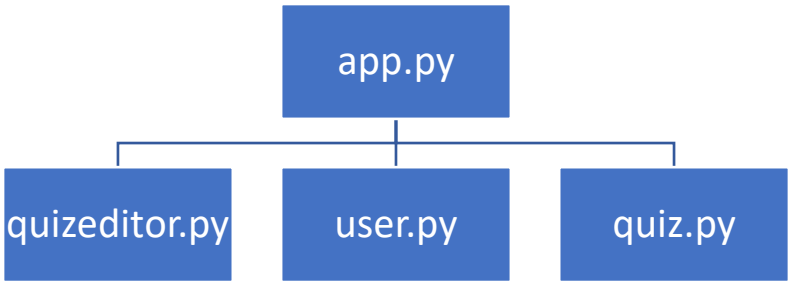


Figure 3 project file structure

1. app.py
2. user.py
3. quiz.py
4. quizeditor.py

app.py

app.py contains the Application class. This class provides the user login and signup window. User can create a user or admin account for him/her from signup window. The signup for admin type account requires permission from the admin.

The user/admin can login or exit. If the user/admin logs in successfully he/she is provided with an admin or user menu depending upon the type of account he/she used to login.

The admin menu has three options

- Quiz
- Quiz editor
- User dashboard

The user menu has three options

- Quiz
- User dashboard

quiz.py

quiz.py contains Quiz class which provides the user/admin to attempt the quizzes. It fetches all the available quizzes from the database and provides admin/user a menu of available quiz options. If there is no Quiz available at that moment it displays a message about it.

User/admin can select from the available options and attempt the quiz. It also checks following things before allowing the user to attempt the quiz.

- Response limit
- Date and time

If the above parameters are not satisfied the user is not allowed to attempt the quiz and the message about it is displayed on the right side of the option.

If the above parameters are satisfied it allows the user to attempt the quiz. It displays instructions about the quiz which are auto generated. The user can either accept or decline. If the user accepts response is counted and if the user declines response is not counted. If the quiz has time limit set it starts a timer and automatically submits the response when the time limit is exceeded.

User is provided with multiple choice questions to attempt.

User can either exit or submit from the quiz. In the both scenario response is counted.

quizeditor.py

quizeditor.py contains QuizEditor class which is used to create and edit quizzes. The admin can only edit and save quizzes. The quizzes are added to MySQL database when saved. The QuizEditor has a menu at the top it has two options

- File
- About

File option has

- New
- Open
- Save
- Close
- Exit

New option creates a new quiz.

Open option fetches an already saved quiz from MySQL Database after asking to save any opened or new quiz.

Save option saves the quiz opened or new quiz.

Close option closes any quiz if it's opened or new quiz after asking to save it.

Exit option exits from the Quiz Editor.

About option opens a window showing information about the Qui Editor

user.py

It contains the User class which stores user/admin details during run time and provide the user/admin dashboard. The user dashboard has two view admin mode and user mode.

User mode view has options to reset password and view previous quiz responses. If there are no responses available it shows a message about it. The user can view their individual response and export them into a file.

Admin mode view has all the capabilities of User mode and admin has the ability to export all the individual quiz responses into a file. Which he/she can then use to analyze the data.

Dependencies

- **sqlite3**

sqlite3 is a standard python module used for accessing, storing and modifying data in a database from a python application.

- **tkinter**

tkinter is a standard python module used for creating graphical user interface applications in python. It is available with a standard python installation and is cross-platform i.e compatible with Windows, Mac, and Linux operating systems.

- **json**

json is a standard python module used to parse and create JSON files.

- matplotlib

matplotlib is a python module used to plot graphs and charts.

- **os**

Os is a standard python module used to access operating system functions.

- **datetime**

datetime is a standard python module used to manipulate access date and date related data

- **time**
  time is a standard python module used to access time and manipulate time related data.
- **fpdf**

fpdf is a python module used to create, edit and save pdf files

Database structure

The Quiz.db database consists of three tables

1. quizzes
2. responses
3. users

quizzes table

Table 1 quiz table

id	name	path
1	Samplequiz1	C:/Samplequiz1.json
2	Samplequiz2	C:/Samplequiz2.json
3	Samplequiz3	C:/Samplequiz3.json
4	Samplequiz34	C:/Samplequiz4.json

responses table

Table 2 response table

user	quiz	response	response_count
1	23	response/name1_quiz.json	1
2	12	Response/name2_quiz.json	2

users table

privilege 1 for admin and privilege 0 for user

Table 3 user table

id	username	password	privilege
1	admin	password	1
2	user1	password1	0
3	user2	password2	0

Quiz File

File format JSON file.

Sample File structure.


```
{
   "id": 61,
   "test": "sample quiz name\n",
   "response_limit": 1,
   "date": {
      "day": 1,
      "month": 8,
      "year": 2020
   },
   "time": {
      "hour": 10,
      "minute": 30,
      "second": 0
   },
   "time_limit": {
      "hour": "1",
      "minute": "0",
      "second": "0"
   },
   "sections": [
      {
         "id": 0,
         "section": "sample section name",
         "questions": [
            {
```

```
                "id": 0,
                "question": "sample question",
                "options": [
                    {
                        "option": "sample option",
                        "id": 0,
                        "key": true
                    }
                ],
                "type": "single",
                "point": 1
            }
        ]
    }
  ],
  "instructions": [
    {
      "instruction": "You have 1 hours"
    },
    {
      "instruction": "There are 1 sections in total"
    },
    {
      "instruction": "In the section sample section name there are 1 questions in total"
    }
  ]
}
```

Quiz response file

File format JSON file.

Sample File structure.


```
{
    "test": 61,
    "response": [
 [ 0, 0, [0],1,[0], 1  ]
],
    "max_points": 1,
    "score": 1,
    "correct": 1,
    "incorrect": 0,
    "no_response": 0
}
```

Flow chart

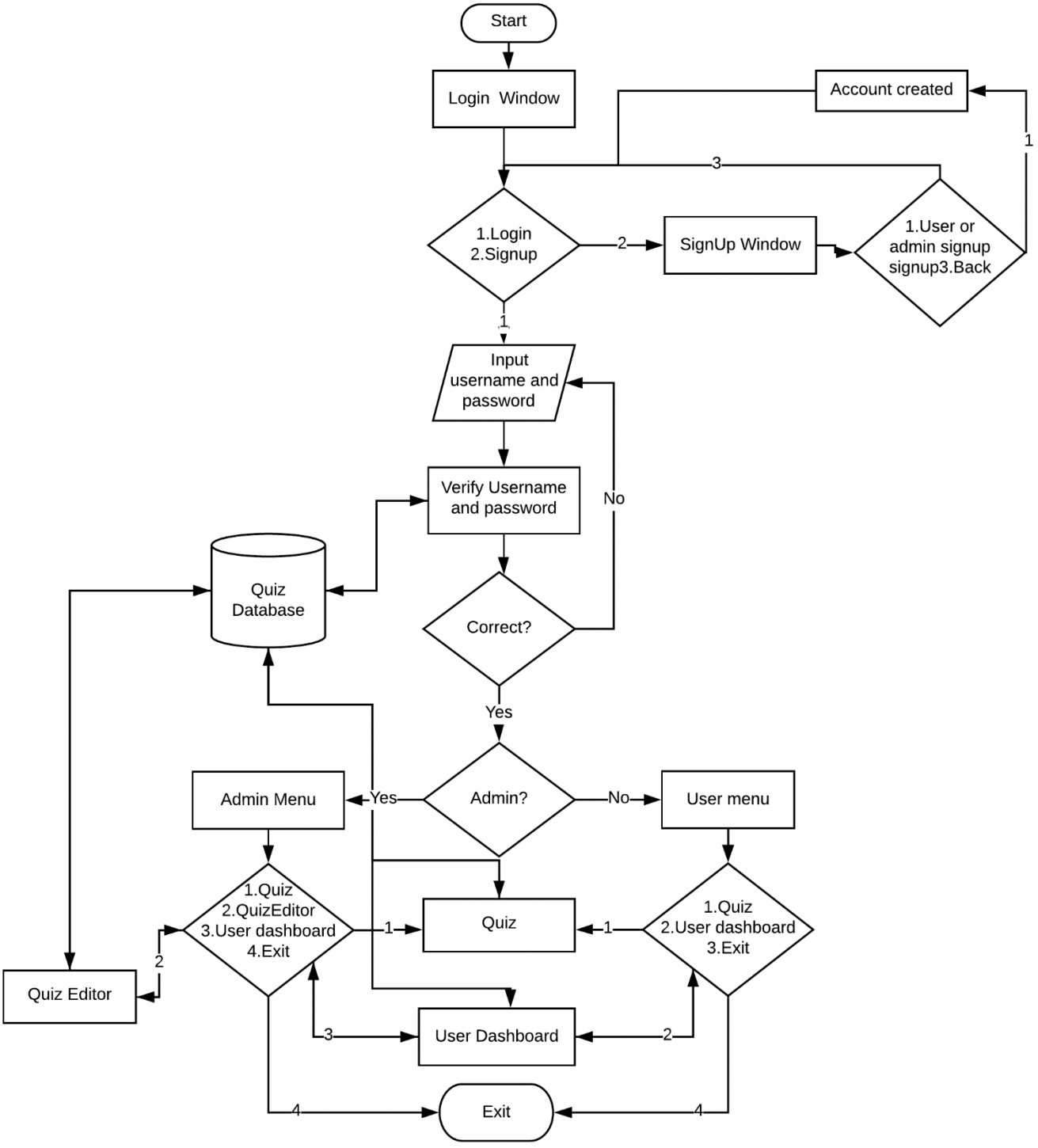


Figure 4 Flowchart

Usage

1. How to make a user account

1.1. Start the application

1.2. A login window will appear on the screen like below.

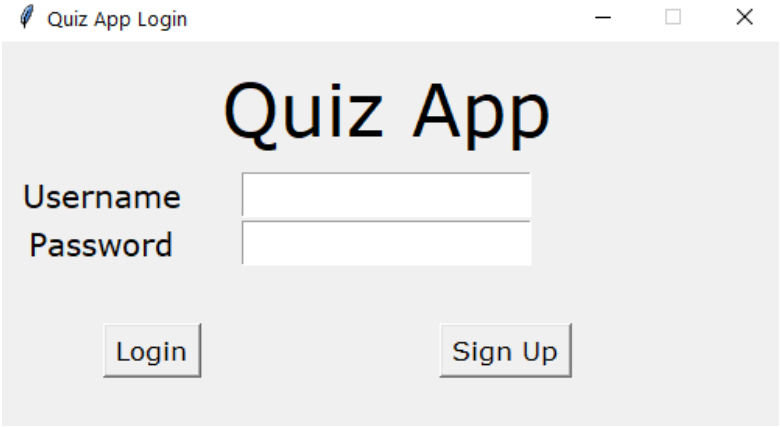


Figure 5 quiz app login

1.3. Click on the signup button

1.4. Signup window like below will appear.

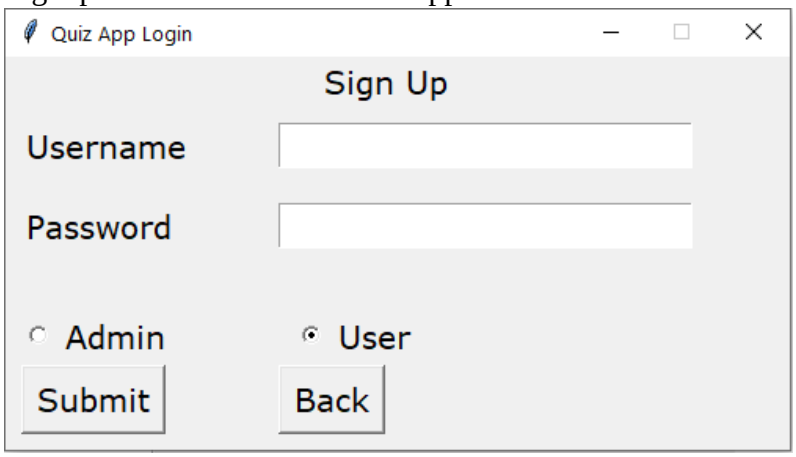


Figure 6 user signup

1.5. Enter username and password of your choice both should be at least 8 digit long.

1.6. Select the type of account. By default, it is set to User type.

1.7. Click Submit Button.

1.8. A message will appear after successful account creation.

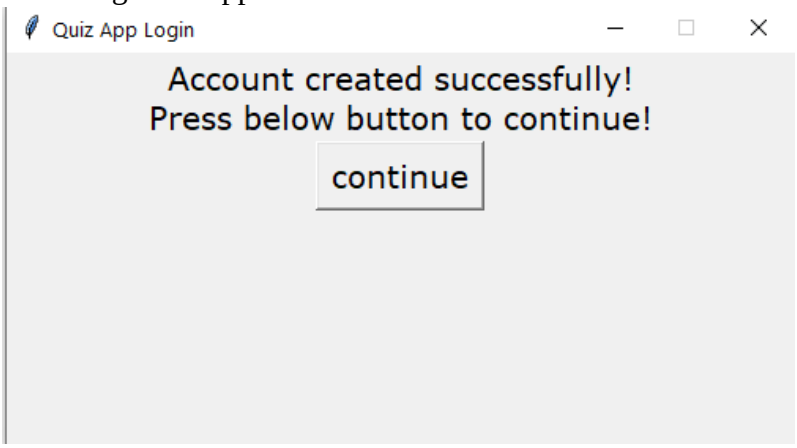


Figure 7 account creation

1.9. Press continue button to login.

2. How to make Admin Account

2.1. Follow 1.1.1 to 1.1.6.

2.2. For creating admin type account select admin option.

2.3. It will ask for permission from Administrator of the application like below

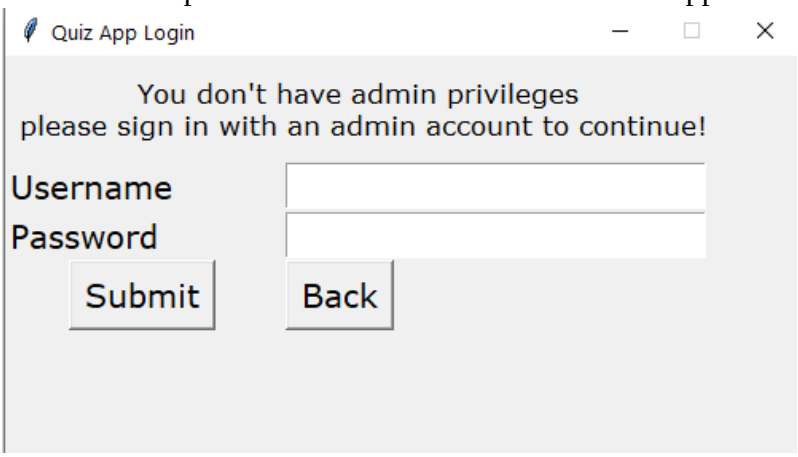


Figure 8 admin permission

2.4. Enter username and password of any administrator account to continue

2.5. Click submit button.

2.6. A message will appear after successful account creation.

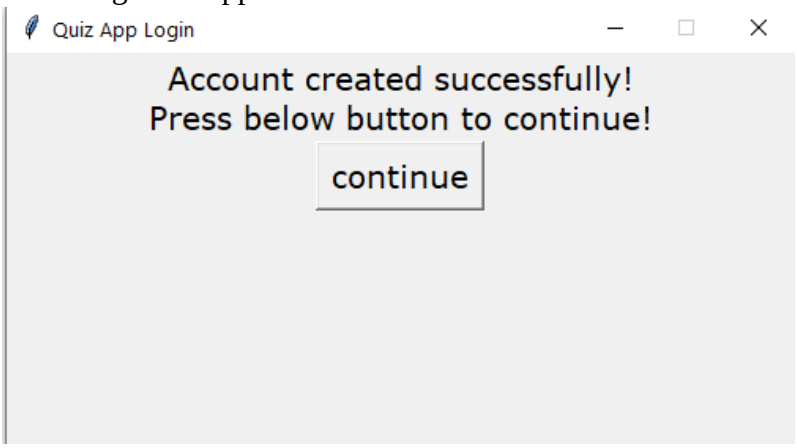


Figure 9 account creation

3. How to login

3.1. Start the application

3.2. A login window will appear on the screen like below.

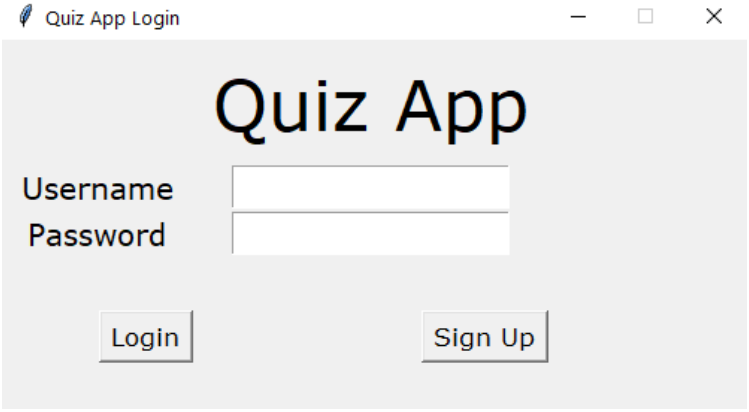


Figure 10 quiz login

3.3. Enter your username and password.

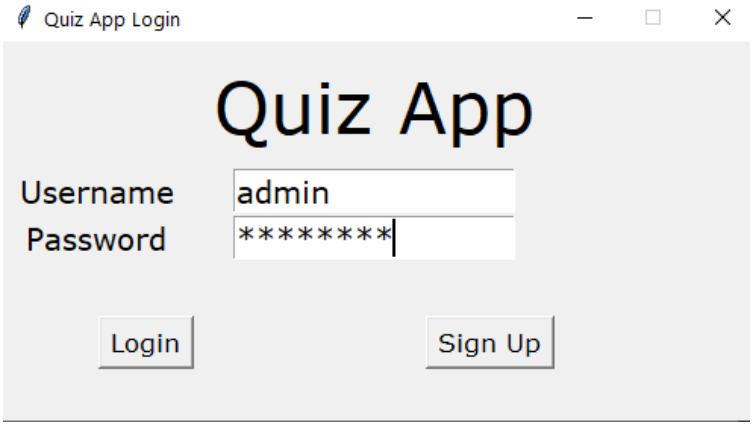


Figure 11 quiz login username and password

3.4. Click login.

3.5. After successful login user or admin menu will appear.

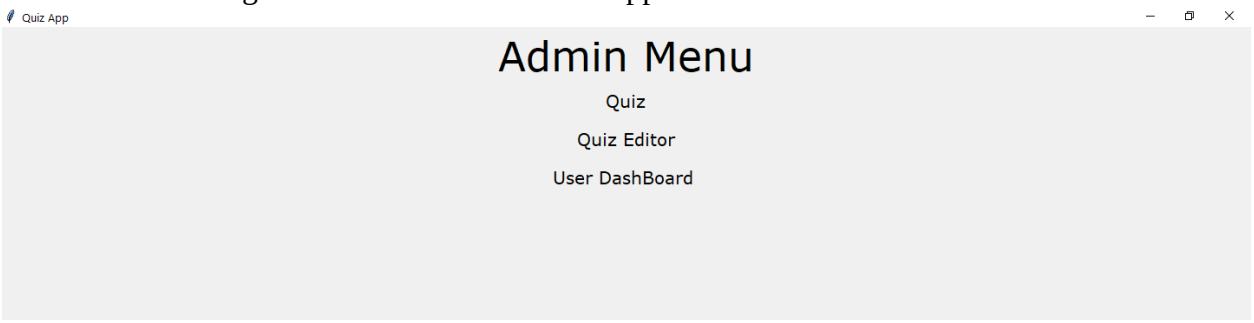


Figure 12 admin menu

3.6. or

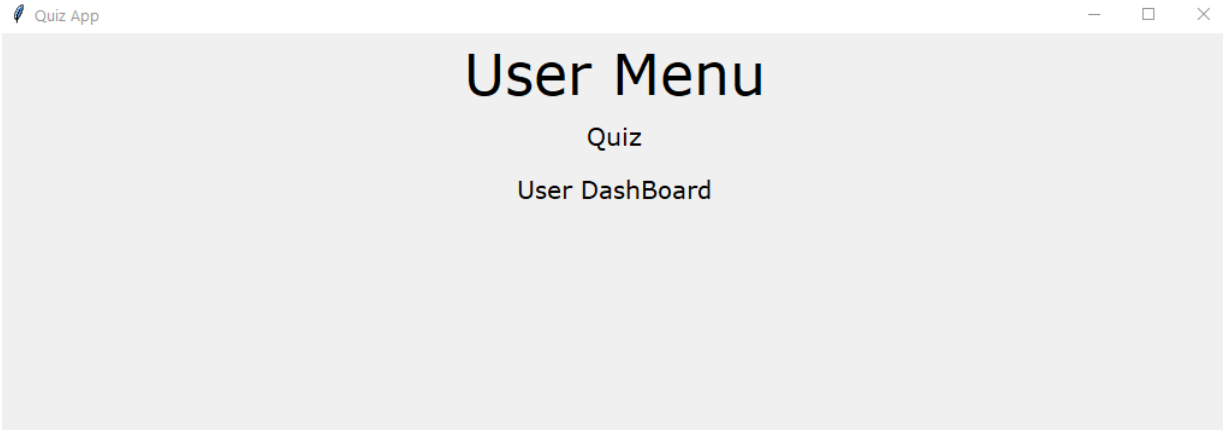


Figure 13 user admin

4. How to start a quiz

4.1. Select Quiz option from user or admin menu.

4.2. Quiz menu will appear like this is there are no quizzes available.

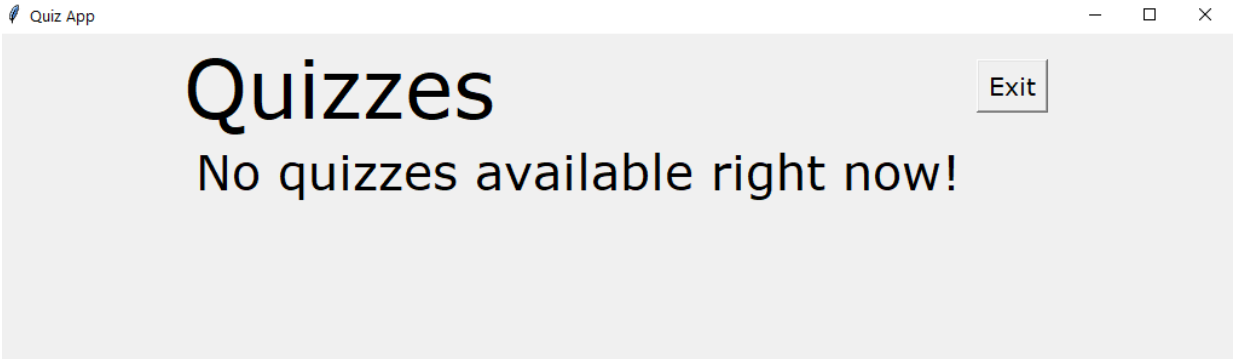


Figure 14 no quiz available

4.3. Quiz menu will appear like this if there are quizzes available.

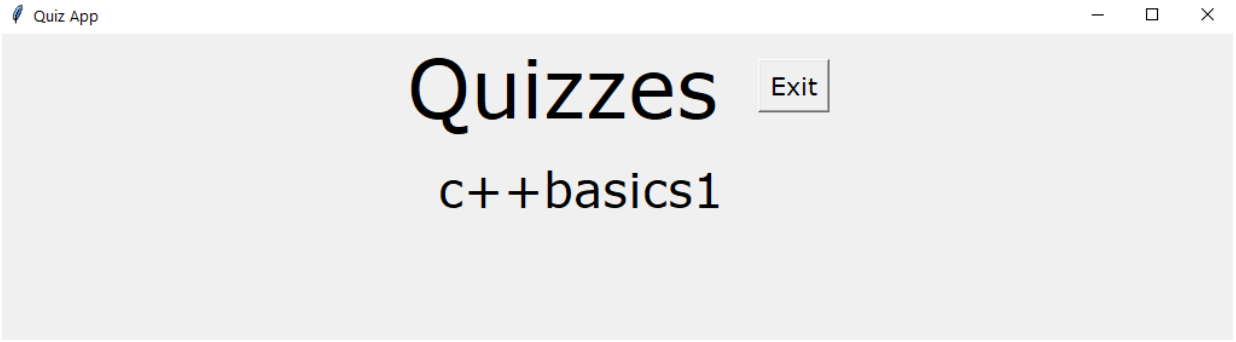


Figure 15 quiz available

4.4. Click on the quiz you want to attempt.

4.5. If you have exceeded response limit of quiz a message will appear

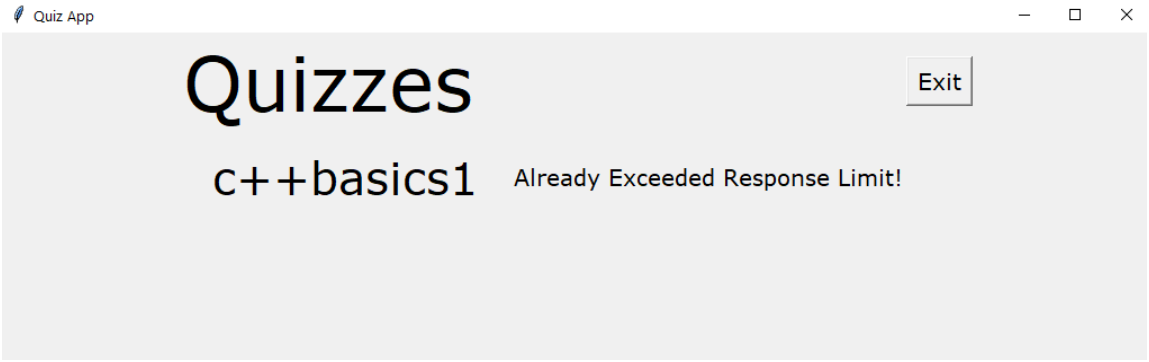


Figure 16 quiz response limit

4.6. If the scheduled date does not match the current date a message will appear

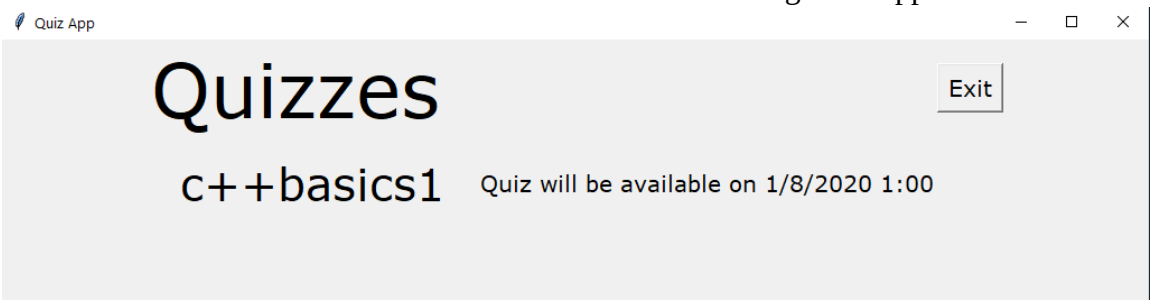


Figure 17 quiz schedule

4.7. If the scheduled time does not match the current time a message will appear

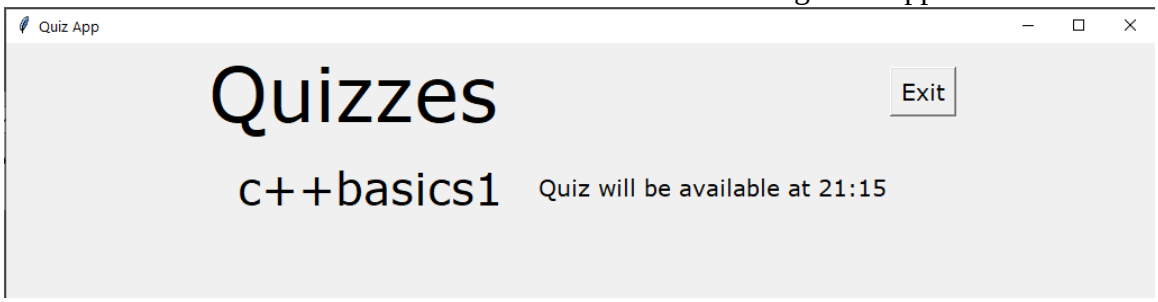


Figure 18 quiz schedule time

4.8. Instruction will appear if 3.5 to 3.7 is satisfied

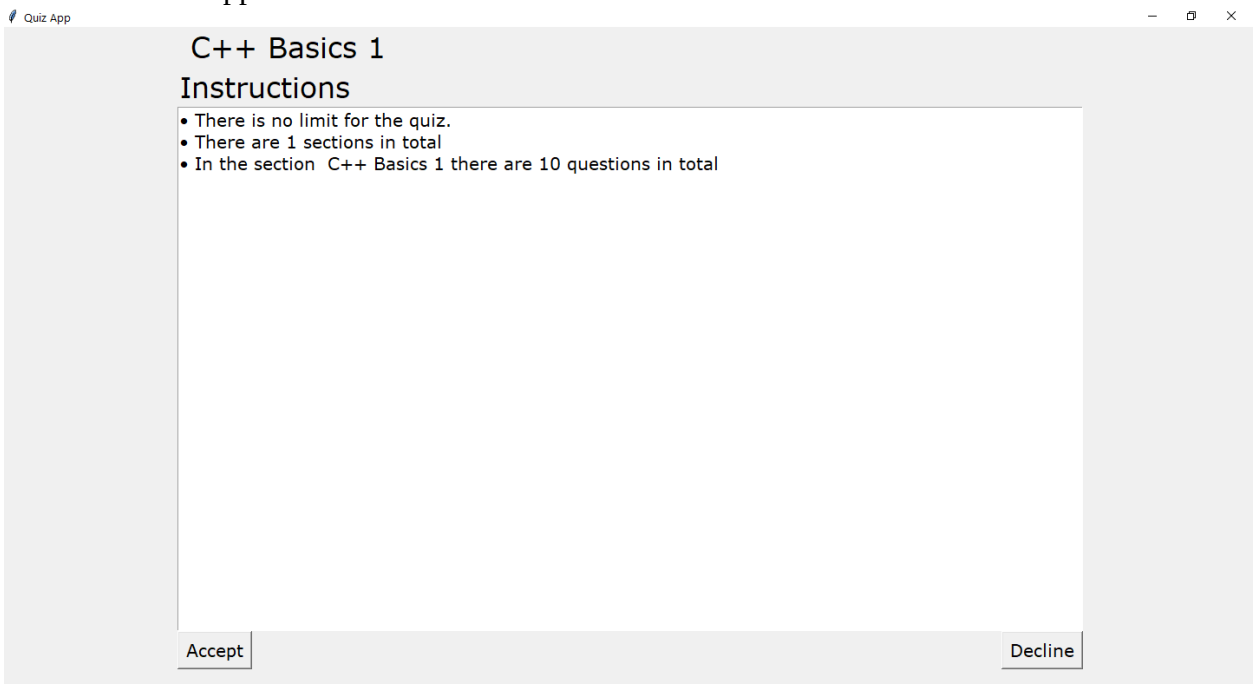


Figure 19 quiz instruction

4.9. Click Accept to start the quiz

4.10. Click Decline to quit the quiz but response won't be counted

5. **How to attempt a quiz**

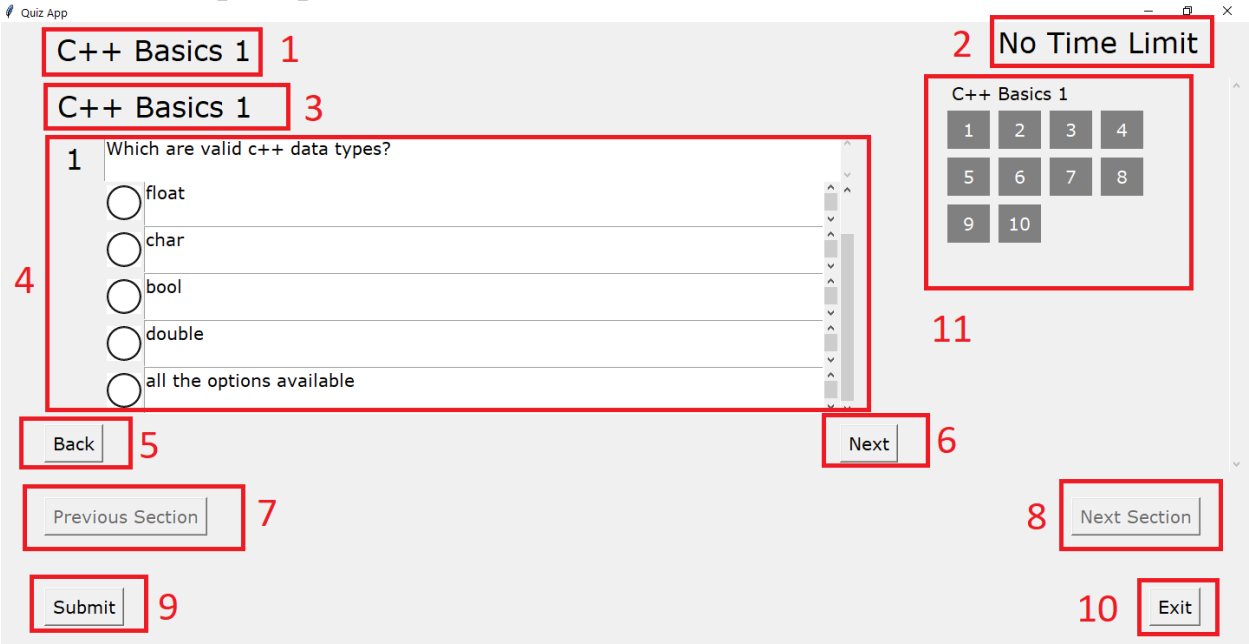


Figure 20 quiz attempt

5.1. Name of quiz
5.2. Time limit of quiz
5.3. Name of section in the quiz
5.4. Question and options
5.5. Back button to go to previous question in section in quiz
5.6. Next button to go to next question in section in quiz
5.7. Previous section button to go to previous section in quiz
5.8. Next section button to go to next section in quiz
5.9. Submit button to submit the response.
5.10. Exit button to exit the quiz.
5.11. Jump to any question of any section by clicking the question number.

6. How to reset password

6.1. From user menu or quiz menu select user dashboard option.

6.2. A user or admin dashboard will appear.

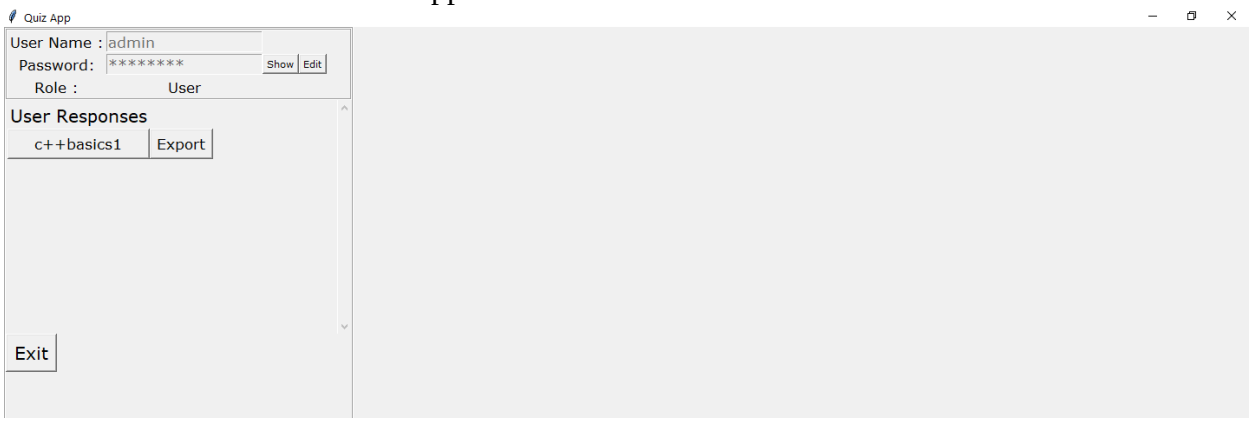


Figure 21 user or admin dashboard

6.3. Click show to view password

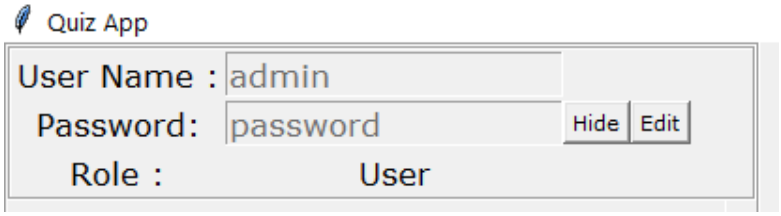


Figure 22 view password

6.4. Click edit and change the password

Figure 23 change password

6.5. Click save to save the reset password

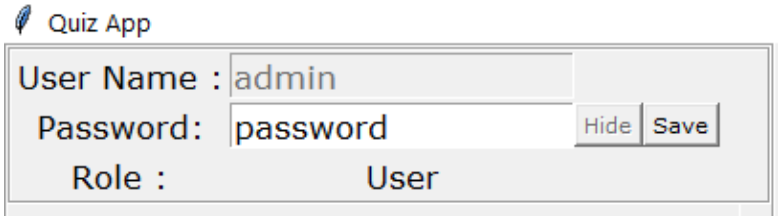


Figure 24 save password

7. How to View Quiz result

7.1. From user menu or quiz menu select user dashboard option.

7.2. A user or admin or user dashboard will appear.

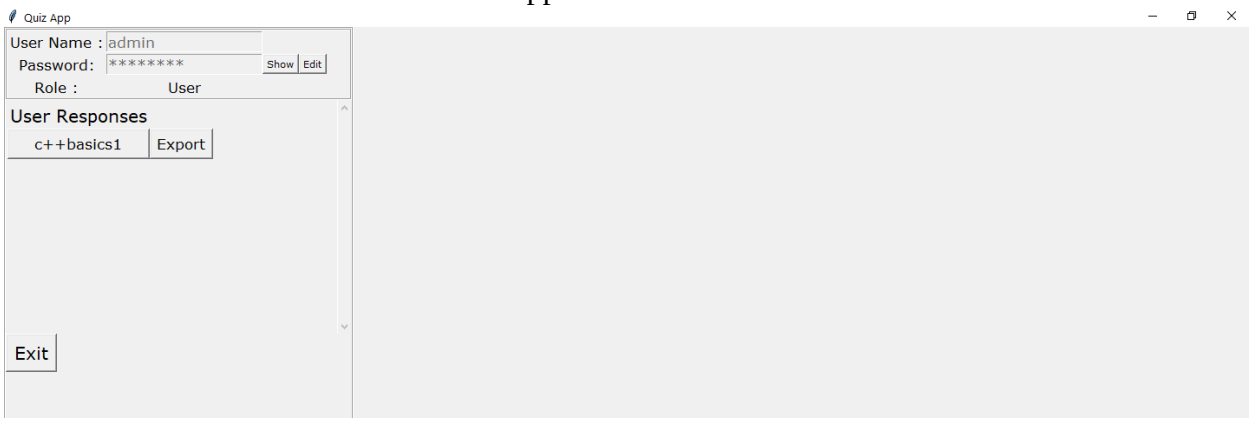


Figure 25 admin or user dashboard

7.3. If there are no previous attempts a message will appear.

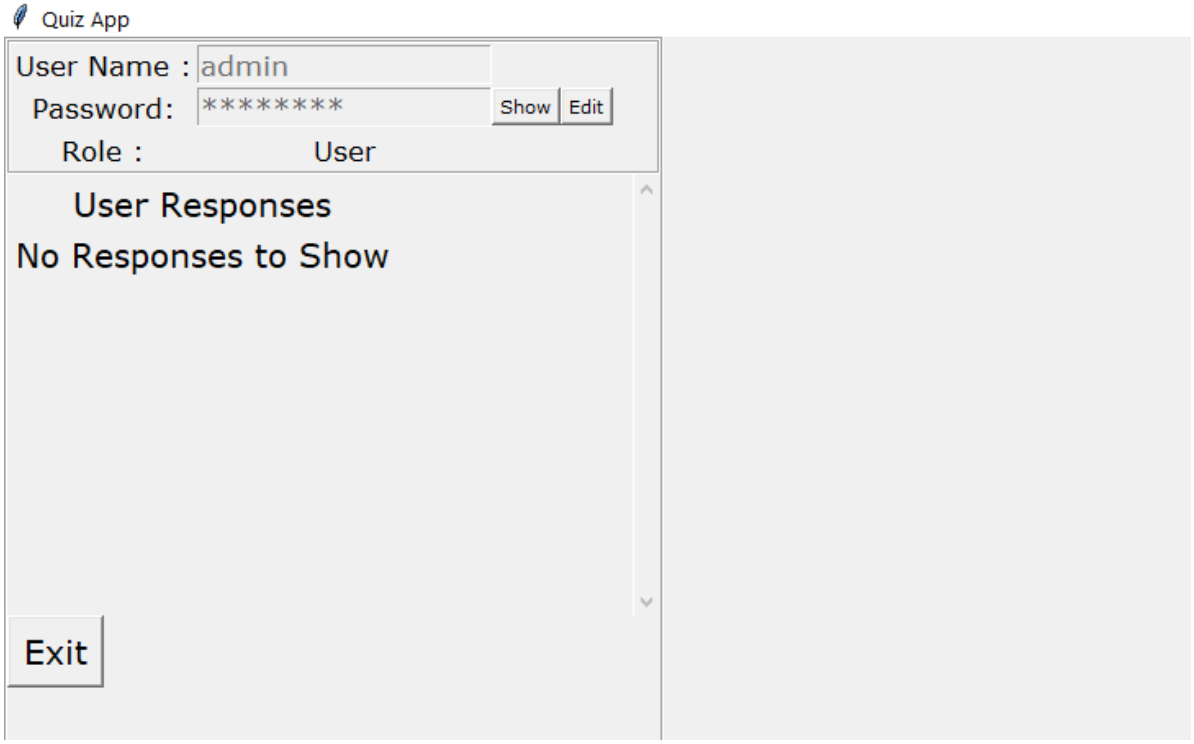


Figure 26 no response dashboard

7.4. If there are attempts available it will appear like this.

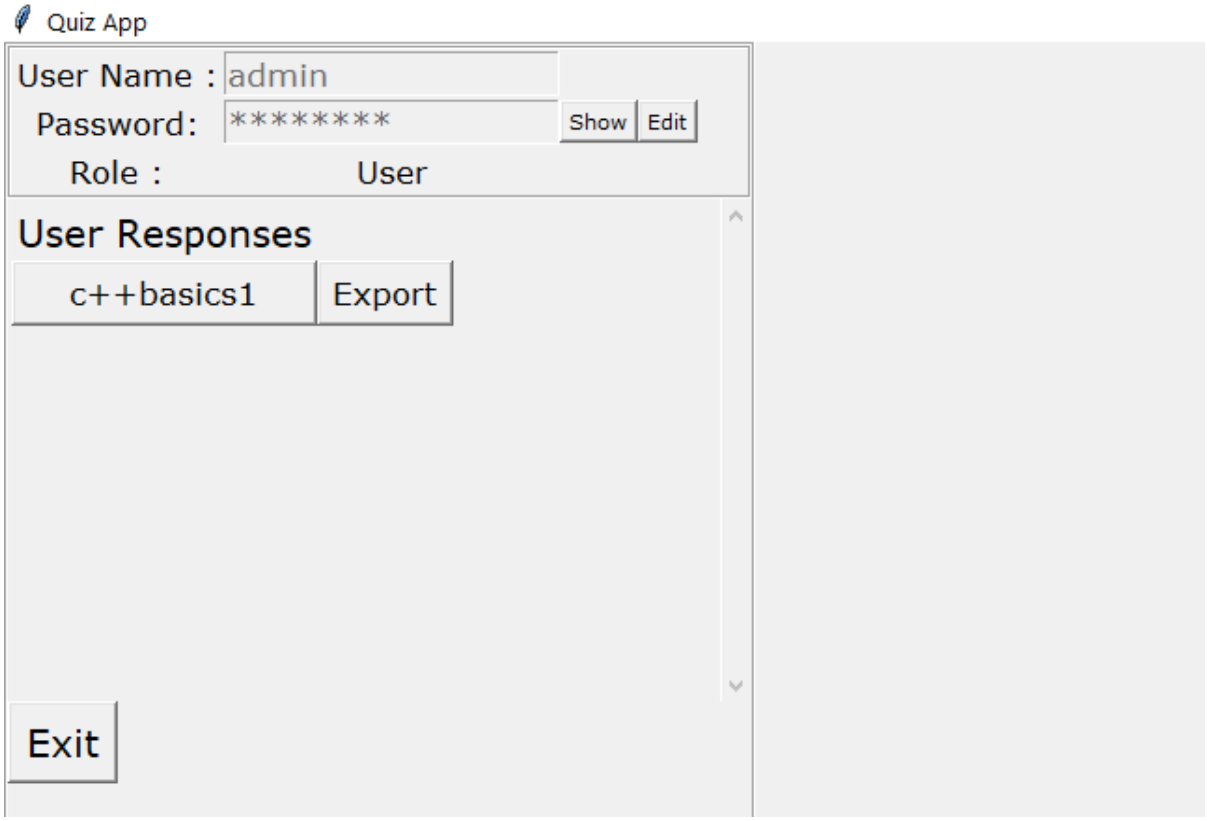


Figure 27 quiz response available

7.5. Click the quiz name you want to view the result.

7.6. The result will appear on right side.

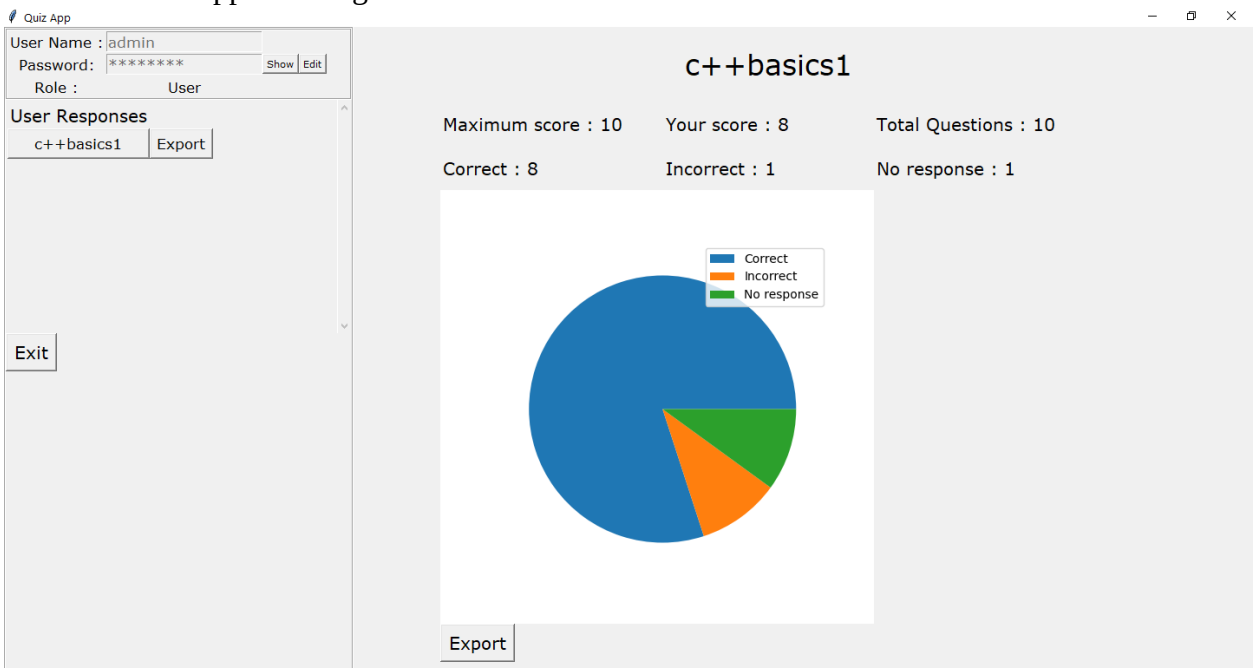


Figure 28 quiz result

8. How to save the quiz result

8.1. Follow 7.1 to 7.6

8.2. Click the export button on the bottom of the result.

8.3. It will ask the path and name to save.

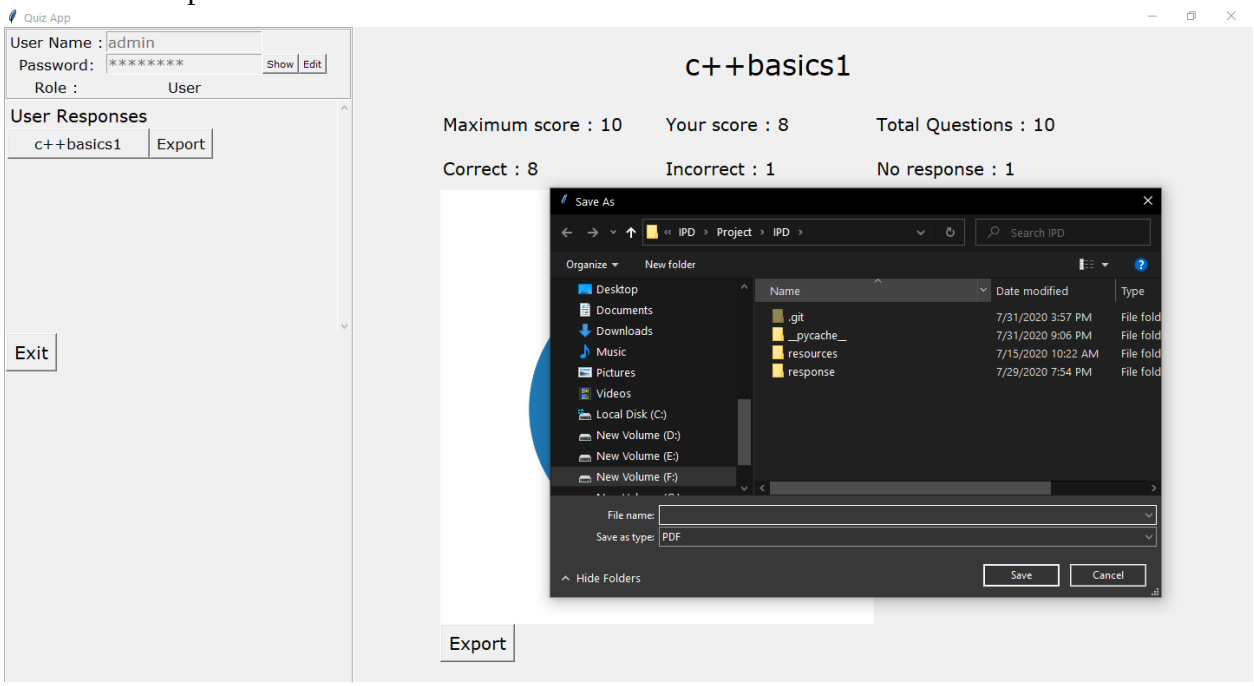


Figure 29 save quiz result dialog

8.4. If the quiz result was saved successfully a message will popup

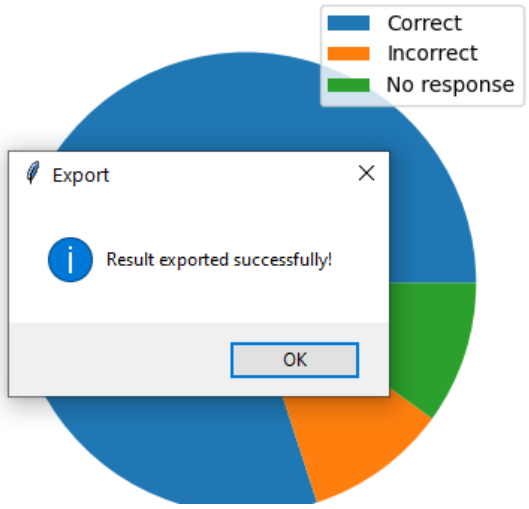


Figure 30 quiz result save message

8.5. If there are any errors occur whiling saving result a message will appear.

8.6. The Result will be saved in a PDF file.

9. How to export all the responses of a quiz

9.1. Follow 7.1 to 7.6
9.2. Requires admin account.
9.3. Click on export button on right side of the quiz you want to export all the responses.
9.4. It will ask for name and path of file to save

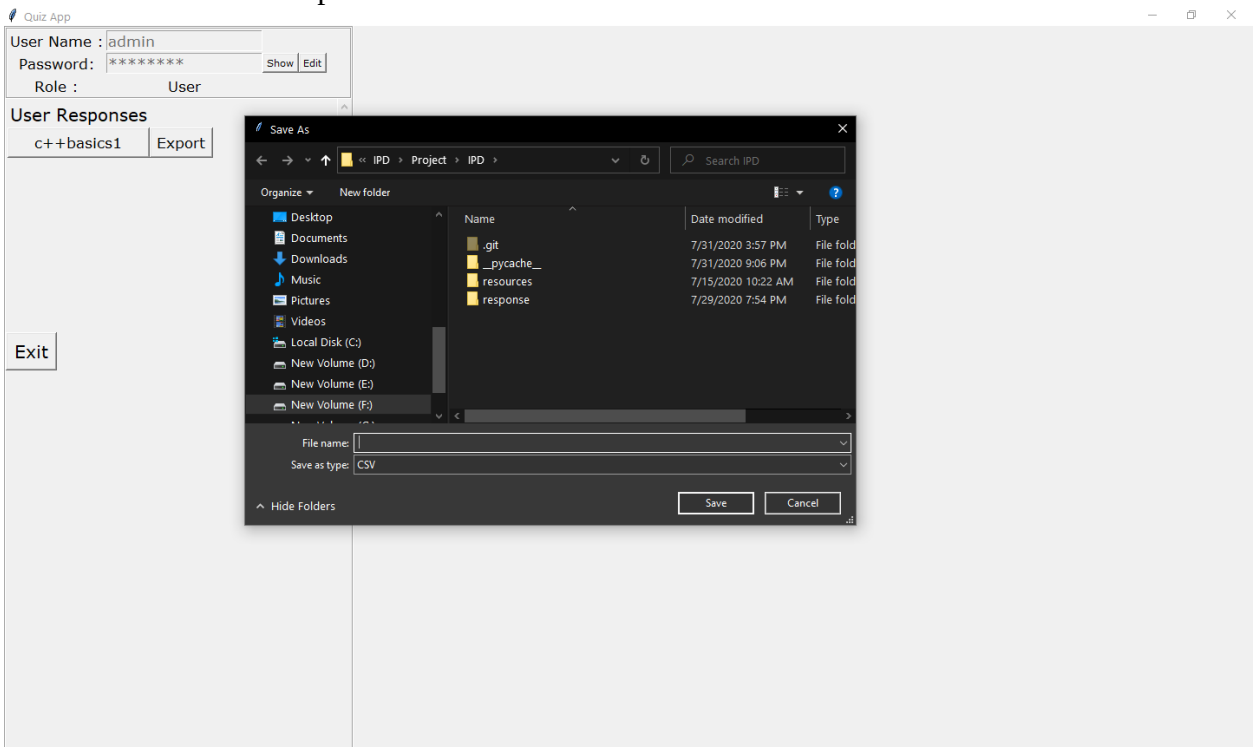


Figure 31 export all quiz response

9.5. If the file was exported successfully a message will popup

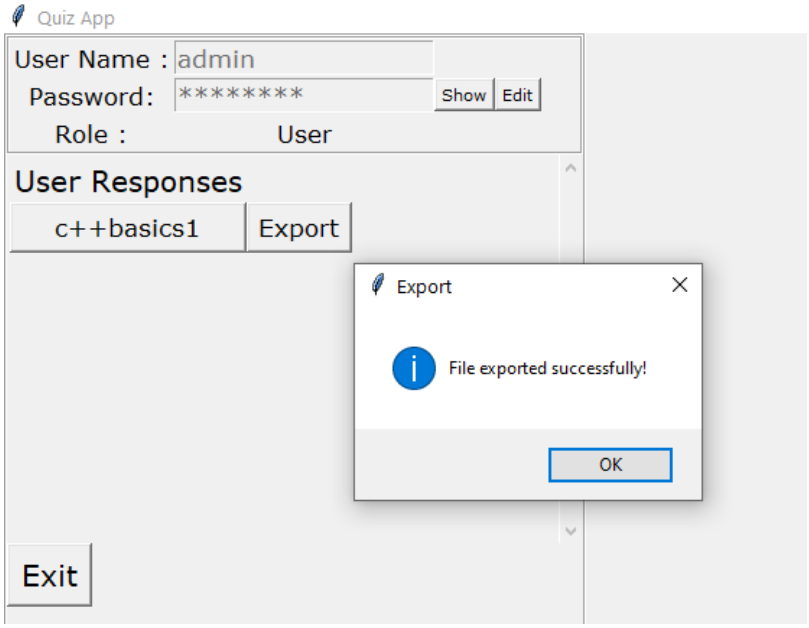


Figure 32 all quiz response exported successfully

10. How to create a quiz.

10.1. Requires admin account.

10.2. Click quiz editor option from admin menu.

10.3. Press Ctrl + N or

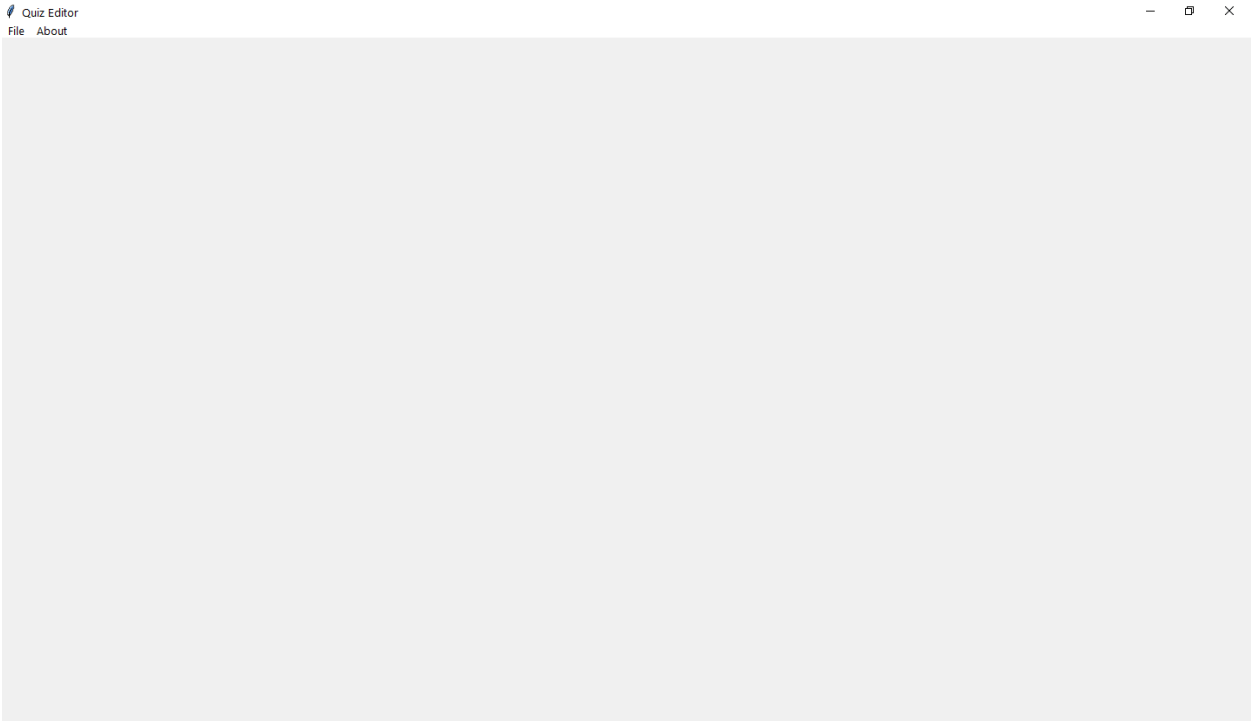


Figure 33 create quiz

10.4. Click File option from menu.

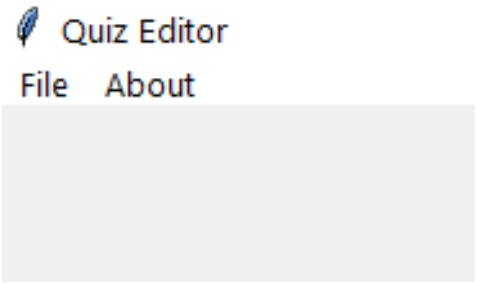


Figure 34 quiz editor menu

10.5. Click New option

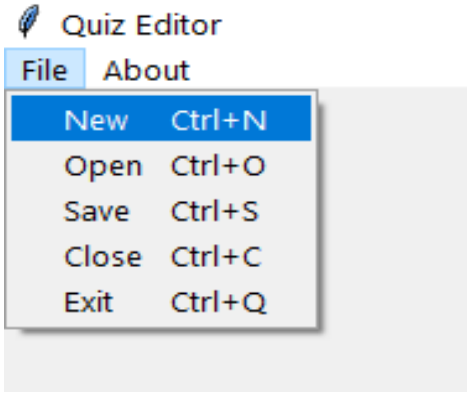


Figure 35 new quiz

10.6. If there is quiz opened already it will ask to save the already opened quiz

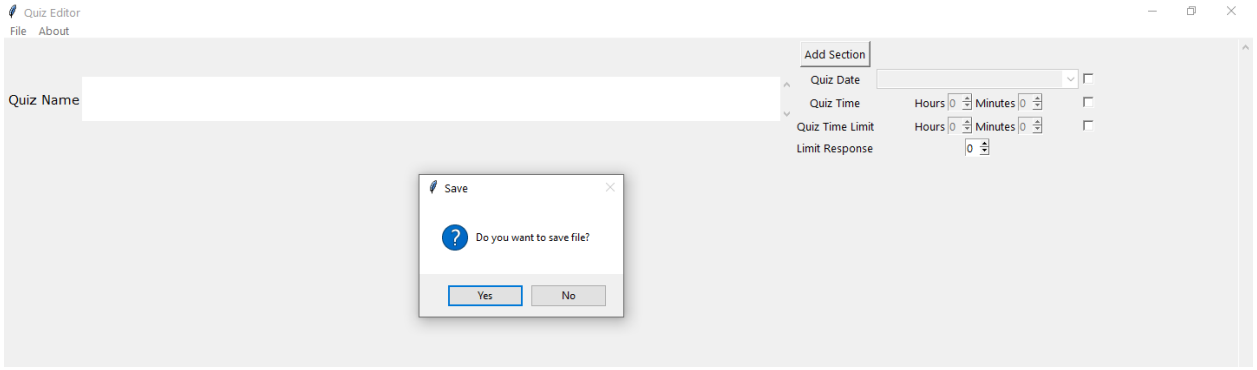


Figure 36 save quiz message

10.7. After that New quiz will be created to edit.

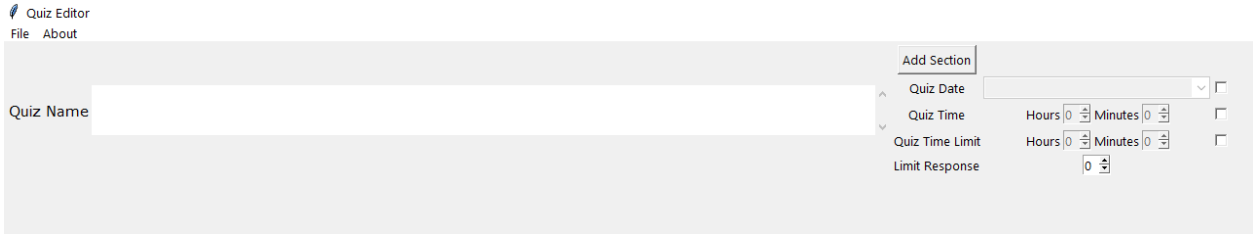


Figure 37 new quiz created

11.How to open saved quiz.

11.1. Requires admin account

11.2. Press Ctrl + O or

11.3. Click File option from menu in quiz editor

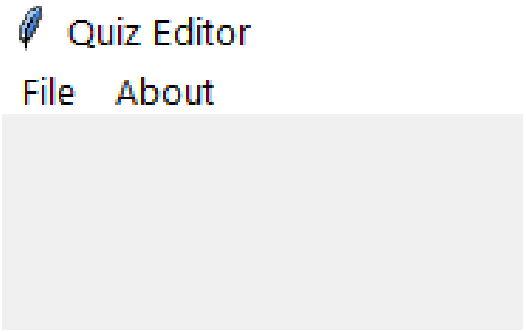


Figure 38 quiz editor file menu

11.4. Click open option

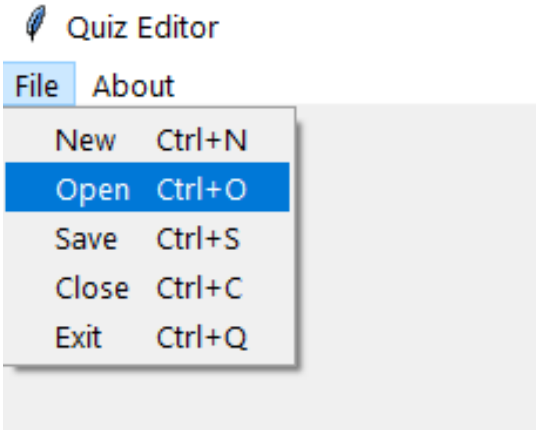


Figure 39 open quiz

11.5. Open window will appear like this if there are no quizzes saved already message will appear like this

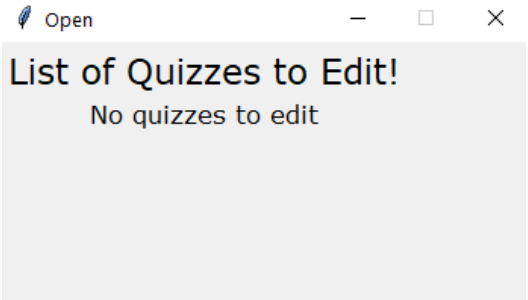


Figure 40 no quiz to edit

11.6. Open window will appear like this if there are quizzes available

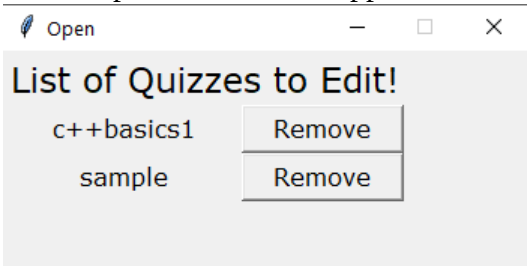


Figure 41 quiz to edit

11.7. Click the quiz you want to open.

12. How to save quiz

12.1. Requires admin account

12.2. Press Ctrl + S or

12.3. Click File option from menu in quiz editor.

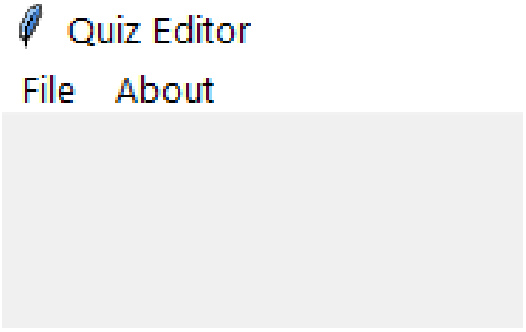


Figure 42 quiz editor menu

12.4. Click Save option.

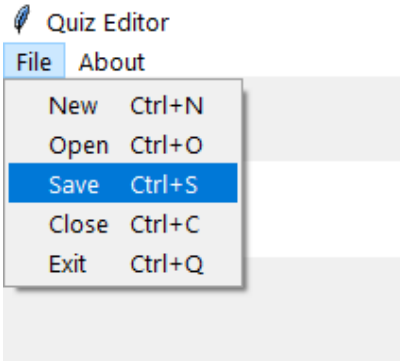


Figure 43 save quiz

12.5. A popup message will appear asking whether to save file or not.

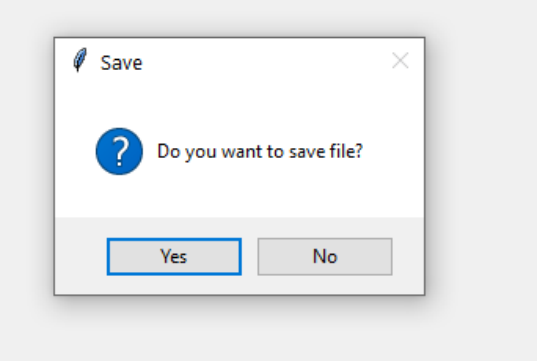


Figure 44 save quiz message

12.6. A popup message will appear saying file saved successfully.

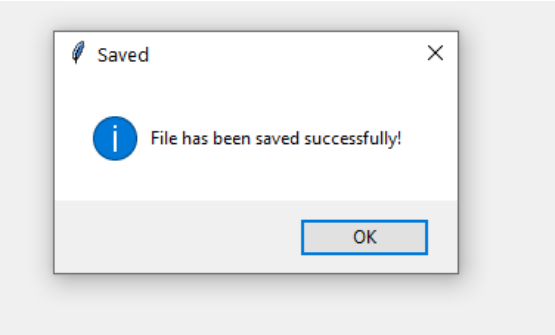


Figure 45 quiz saved message

13.How to remove quiz

13.1. Follow 9.1 to 9.5

13.2. Click Remove button on the right side of the quiz you want to remove.

13.3. A message will popup asking are you sure you want to remove the quiz

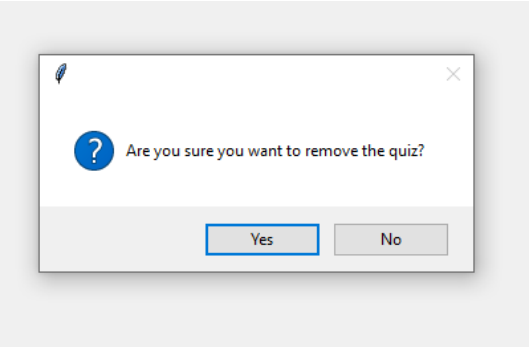


Figure 46 remove quiz

14.How to close quiz

14.1. Requires admin account

14.2. Press Ctrl + C or

14.3. Click File option from menu in quiz editor.

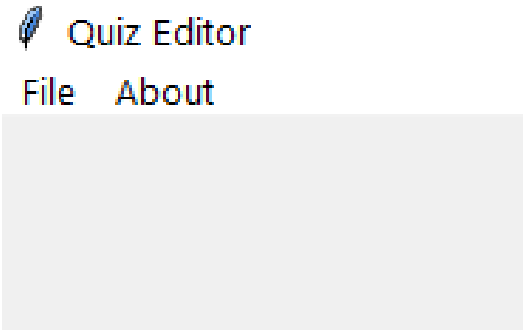


Figure 47 quiz editor menu

14.4. Click Close option.

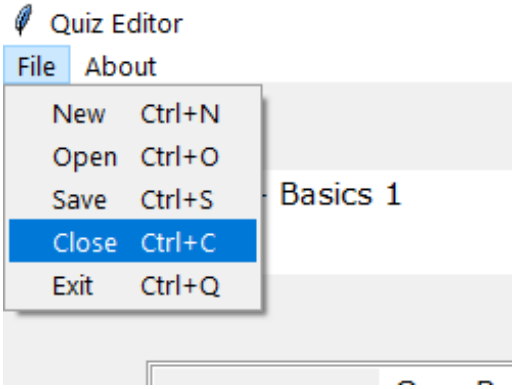


Figure 48 close quiz

14.5.

14.6. A message will popup asking to save the file or not.

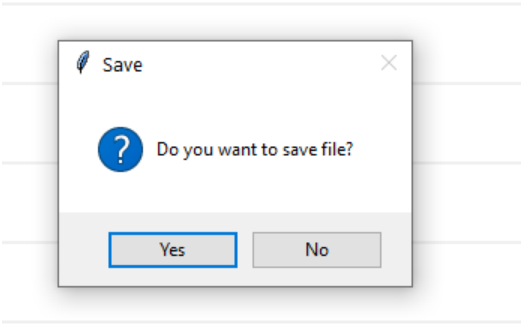


Figure 49 save quiz message

14.7.

14.8. A message will popup saying save file

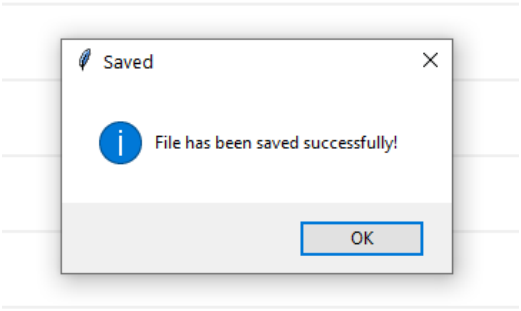


Figure 50 quiz saved message

15. How to edit quiz

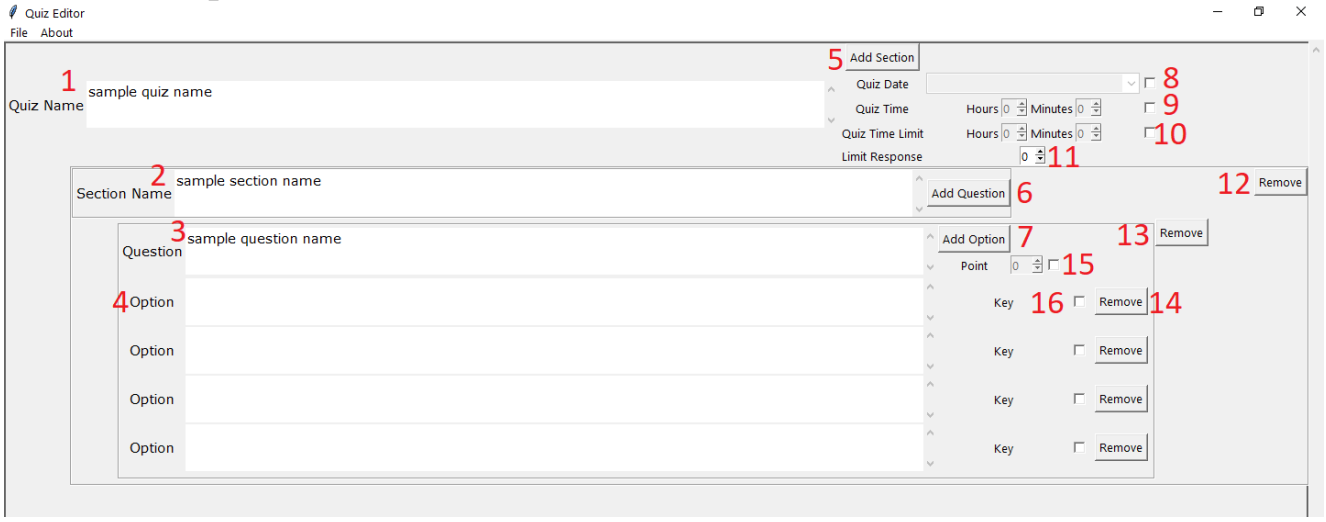


Figure 51 quiz edit

15.1. Enter quiz name here
15.2. Enter section name here.
15.3. Enter question here.
15.4. Enter option here
15.5. Click to Add Section
15.6. Click to Add Question
15.7. Click to Add Option
15.8. Schedule date of quiz by checking.
15.9. Scheduling time of quiz by checking.
15.10. Set time limit by checking.
15.11. Set response limit. NOTE 0 is default value meaning unlimited response.
15.12. Click to Remove section
15.13. Click to Remove question
15.14. Click to remove option
15.15. Set points of question by checking.

15.16. Set the option as answer key by checking.

16.How to exit quiz editor

16.1. Requires admin account

16.2. Press Ctrl + Q or

16.3. Click File option from menu in quiz editor.

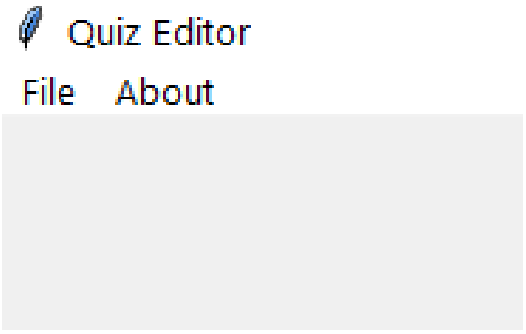


Figure 52 quiz editor menu

16.4.

16.5. Click Exit option

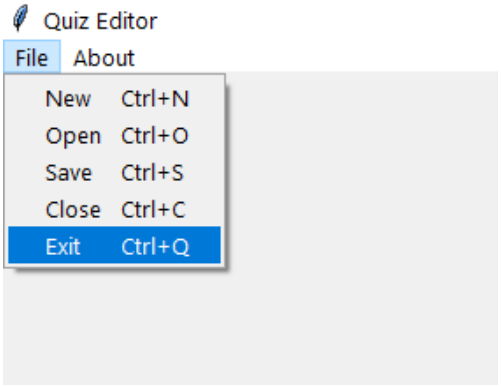


Figure 53 exit quiz

16.6.

16.7. If any quiz is opened it ask to save it.

16.8. Then it will come back to user or admin menu

17.How to view Quiz editor version

17.1. Requires admin account

17.2. Click About option from menu in quiz editor.

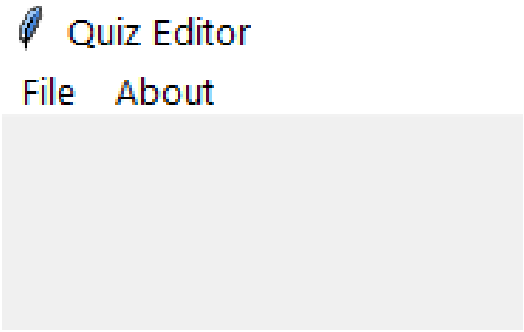


Figure 54 quiz editor menu

17.3. A window will appear showing version of quiz editor

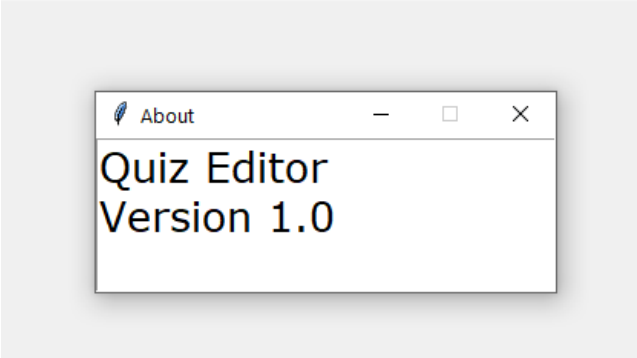


Figure 55 quiz editor version

Source Code

app.py

```
1. import tkinter as tk
2. from tkinter import messagebox as mb
3. import sqlite3
4. import json
5. import quizeditor as QE
6. import quiz as Q
7. import user as U
8.
9. VERY_LARGE_FONT = ("Verdana",35)
10. LARGE_FONT = ("Verdana",25)
11. MEDIUM_FONT = ("Verdana",15)
12. SMALL_FONT =("Verdana",13)
13. VERY_SMALL_FONT = ("Verdana",8)
14.
15. class Application:
16.   def __init__(self):
17.           self.window = None
18.           self.loginwindow = None
19.           self.loginframe = None
20.           self.signupframe = None
21.           self.adminpermissionfarme = None
22.           self.successsignupframe = None
23.           self.user = None
24.           self.app = None
25.           self.frame = None
26.           self.uentry  = None
27.           self.pentry = None
28.           self.LoginWindow()
29.
30.   def LoginWindow(self):
31.           self.loginwindow = tk.Tk()
32.           self.loginwindow.title("Quiz App Login")
33.           self.loginwindow.geometry("500x250+200+200")
34.           self.loginwindow.resizable(height = False , width = False)
35.           self.loginframe = tk.Frame(self.loginwindow,)
36.
37.           self.loginstatus = tk.Label(self.loginframe,font=SMALL_FONT)
38.
39.           tlabel = tk.Label(self.loginframe,text="Quiz App",font
    =VERY_LARGE_FONT)
40.           ulabel = tk.Label(self.loginframe,text="Username",font = MEDIUM_FONT)
41.           self.uentry = tk.Entry(self.loginframe,font =MEDIUM_FONT,width=14)
42.           plabel = tk.Label(self.loginframe,text="Password",font = MEDIUM_FONT)
43.           self.pentry = tk.Entry(self.loginframe,font =
    MEDIUM_FONT,show="*",width=14)
44.           lbutton = tk.Button(self.loginframe,font =
    SMALL_FONT,text="Login",command=self.Login)
45.           sbutton = tk.Button(self.loginframe,font = SMALL_FONT,text="Sign
    Up",command=self.SignUp)
46.
47.           tlabel.grid(row = 0 , column = 1,padx=10,pady=10)
48.           ulabel.grid(row = 1, column = 0,padx=10)
49.           self.uentry.grid(row = 1, column = 1)
50.           plabel.grid(row = 2, column = 0,padx=10)
51.           self.pentry.grid(row = 2, column = 1)
52.           self.loginstatus.grid(row=3,column=0,columnspan=2,sticky="e")
53.           lbutton.grid(row = 4, column = 0,sticky="E",pady=10)
```

```
54.           sbutton.grid(row = 4, column = 1,sticky="E",pady=10)
55.           self.loginframe.pack(fill="both")
56.           try:
57.                   conn  = sqlite3.connect('file:Quiz.db?mode=rw', uri=True)
58.           except:
59.                   mb.showerror("Database Error","Couldn't connect to Database")
60.                   exit()
61.
62.           self.loginwindow.mainloop()
63.
64.   def BackToLogin(self):
65.           self.signupframe.pack_forget()
66.           self.loginframe.pack(fill="both")
67.
68.   def SignUp(self):
69.           if self.adminpermissionfarme is not None:
70.                   self.adminpermissionfarme.pack_forget()
71.           self.signupframe = tk.Frame(self.loginwindow)
72.           self.loginframe.pack_forget()
73.           signuplbl = tk.Label(self.signupframe,text="Sign
    Up",font=MEDIUM_FONT)
74.           usernamelbl =
    tk.Label(self.signupframe,text="Username",font=MEDIUM_FONT)
75.           self.usernameentry = tk.Entry(self.signupframe,font=MEDIUM_FONT)
76.           self.status = tk.Label(self.signupframe,font=SMALL_FONT)
77.           passwordlbl =
    tk.Label(self.signupframe,text="Password",font=MEDIUM_FONT)
78.           self.passwordentry =
    tk.Entry(self.signupframe,show="*",font=MEDIUM_FONT)
79.
80.           self.rolevar = tk.IntVar()
81.           adminradio =
    tk.Radiobutton(self.signupframe,text="Admin",font=MEDIUM_FONT,value=1,variable=
    self.rolevar,command=self.Permission)
82.           userradio =
    tk.Radiobutton(self.signupframe,text="User",font=MEDIUM_FONT,value=0,variable=s
    elf.rolevar,)
83.           backbtn =
    tk.Button(self.signupframe,text="Back",font=MEDIUM_FONT,command=self.BackToLogi
    n)
84.           sbmtbtn =
    tk.Button(self.signupframe,text="Submit",font=MEDIUM_FONT,command=self.CreateAc
    count)
85.           signuplbl.grid(row=0,column=0,padx=200,columnspan=2,sticky="w")
86.           usernamelbl.grid(row=1,column=0,padx=10,pady=10,sticky="w")
87.           self.usernameentry.grid(row=1,column=1,sticky="w")
88.           passwordlbl.grid(row=2,column=0,padx=10,pady=10,sticky="w")
89.           self.passwordentry.grid(row=2,column=1,sticky="w")
90.
91.           adminradio.grid(row=4,column=0,padx=10,sticky="w")
92.           userradio.grid(row=4,column=1,padx=10,sticky="w")
93.           sbmtbtn.grid(row=5,column=0,padx=10,sticky="w")
94.           backbtn.grid(row=5,column=1,sticky="w")
95.           self.status.grid(row=3,column=0,columnspan=2,sticky="w")
96.           self.signupframe.pack(fill="both")
97.
98.   def Permission(self):
99.           self.adminpermissionfarme = tk.Frame(self.loginwindow,)
100.                  ydhplbl = tk.Label(self.adminpermissionfarme,text="You don't
    have admin privileges\n please sign in with an admin account to
    continue!",font=SMALL_FONT)
101.                  aplbl =
    tk.Label(self.adminpermissionfarme,text="Username",font=MEDIUM_FONT)
```

```
102.                    self.apentry =
    tk.Entry(self.adminpermissionfarme,font=MEDIUM_FONT)
103.                    pplbl =
    tk.Label(self.adminpermissionfarme,text="Password",font=MEDIUM_FONT)
104.                    self.ppentry =
    tk.Entry(self.adminpermissionfarme,show="*",font=MEDIUM_FONT)
105.                    sbmtbtn =
    tk.Button(self.adminpermissionfarme,text="Submit",font=MEDIUM_FONT,command=self
    .CheckPermission)
106.                    backbtn =
    tk.Button(self.adminpermissionfarme,text="Back",font=MEDIUM_FONT,command=self.S
    ignUp)
107.                    self.signupframe.pack_forget()
108.                    ydhplbl.grid(row=0,column=0,columnspan=2,pady=10,sticky="w")
109.                    aplbl.grid(row=1,column=0,sticky="w")
110.                    self.apentry.grid(row=1,column=1,sticky="w")
111.                    pplbl.grid(row=2,column=0,sticky="w")
112.                    self.ppentry.grid(row=2,column=1,sticky="w")
113.                    sbmtbtn.grid(row=3,column=0,sticky="w",padx=40)
114.                    backbtn.grid(row=3,column=1,sticky="w",)
115.                    self.adminpermissionfarme.pack(fill="both")
116.
117.            def CreateAccount(self):
118.                    conn = sqlite3.connect("Quiz.db")
119.                    cursor = conn.cursor()
120.                    cursor.execute("SELECT * FROM users WHERE
    username=?",(self.usernameentry.get(),))
121.                    row = cursor.fetchone()
122.                    if row is None:
123.                            if self.usernameentry.get() == "" or
    self.passwordentry.get() == "":
124.                                    self.status.configure(text="Username or
    password cannot be empty!")
125.                            elif len(self.usernameentry.get()) < 8 and
    len(self.passwordentry.get()) < 8:
126.                                    self.status.configure(text="Username or
    password should be atleast 8 digit long!")
127.                            else:
128.                                    cursor.execute("INSERT INTO users
    (username,password,privilege) VALUES
    (?,?,?)",(self.usernameentry.get(),self.passwordentry.get(),self.rolevar.get(),
    ))
129.                                    self.signupframe.pack_forget()
130.                                    self.successsignupframe =
    tk.Frame(self.window)
131.                                    self.successsignupframe.pack()
132.                                    lbl =
    tk.Label(self.successsignupframe,text="Account created successfully!\nPress
    below button to continue!",font=MEDIUM_FONT)
133.                                    lbl.pack(expand=True,fill="both")
134.                                    btn =
    tk.Button(self.successsignupframe,text="continue",font=MEDIUM_FONT,command=self
    .SuccessSignUpToLogin)
135.                                    btn.pack()
136.                    else:
137.                            self.status.configure(text="Username already
    Taken!")
138.
139.                    conn.commit()
140.
141.            def SuccessSignUpToLogin(self):
142.                    self.successsignupframe.pack_forget()
143.                    self.loginframe.pack(fill="both")
```

```
144.
145.
146.          def CheckPermission(self):
147.                  conn = sqlite3.connect("Quiz.db")
148.                  cursor = conn.cursor()
149.                  cursor.execute("SELECT * FROM users WHERE username=? AND
   password=? ",(self.apentry.get(),self.ppentry.get(),))
150.                  row = cursor.fetchone()
151.                  if row is not None:
152.                          if row[3] == 1:
153.                                  self.adminpermissionfarme.pack_forget()
154.                                  self.signupframe.pack(fill="both")
155.
156.          def Login(self):
157.                  self.username = self.uentry.get()
158.                  self.password = self.pentry.get()
159.                  if  self.username == "" or self.password == "":
160.                          self.loginstatus.configure(text="Username or
   password cannot be empty!")
161.                  else:
162.                          conn = sqlite3.connect("Quiz.db")
163.                          cursor = conn.execute("SELECT * FROM users WHERE
   username=? AND password=?",(self.username,self.password))
164.                          row = cursor.fetchone()
165.                          if row is not None:
166.                                  self.loginwindow.destroy()
167.                                  self.window = tk.Tk()
168.                                  self.user =
   U.User(self.window,self.CallBack,row[0],row[1],row[2],row[3])
169.                                  self.Run()
170.                          else:
171.                                  self.loginstatus.configure(text="Username or
   password is invalid!")
172.
173.          def Run(self):
174.                  self.window.title("Quiz App")
175.                  self.window.geometry("1024x768")
176.                  self.frame =  tk.Frame(self.window)
177.                  if self.user.GetPrivilege() == 1:
178.                          lbl = tk.Label(self.frame,text="Admin
   Menu",font=VERY_LARGE_FONT)
179.                          quizbtn =
   tk.Button(self.frame,text="Quiz",relief=tk.FLAT,font=MEDIUM_FONT,command=self.R
   unQuiz)
180.                          quizeditorbtn = tk.Button(self.frame,text="Quiz
   Editor",relief=tk.FLAT,font=MEDIUM_FONT,command=self.RunQuizEditor)
181.                          userdashboardbtn = tk.Button(self.frame,text="User
   DashBoard ",relief=tk.FLAT,font=MEDIUM_FONT,command=self.RunDashBoard)
182.
183.                          lbl.pack()
184.                          quizbtn.pack()
185.                          quizeditorbtn.pack()
186.                          userdashboardbtn.pack()
187.                  else:
188.                          lbl = tk.Label(self.frame,text="User
   Menu",font=VERY_LARGE_FONT)
189.                          quizbtn =
   tk.Button(self.frame,text="Quiz",relief=tk.FLAT,font=MEDIUM_FONT,command=self.R
   unQuiz)
190.                          userdashboardbtn = tk.Button(self.frame,text="User
   DashBoard",relief=tk.FLAT,font=MEDIUM_FONT,command=self.RunDashBoard)
191.
192.                          lbl.pack()
```

```
193.                            quizbtn.pack()
194.                            userdashboardbtn.pack()
195.                    self.frame.pack()
196.                    self.window.mainloop()
197.
198.            def CallBack(self):
199.                    del self.app
200.                    self.app = None
201.                    self.frame.pack()
202.
203.            def RunQuiz(self):
204.                    if self.app is None:
205.                            self.Hide()
206.                            self.app =
    Q.Quiz(self.window,self.user.GetUserId(),self.CallBack)
207.
208.            def RunQuizEditor(self):
209.                    if self.app is None:
210.                            self.Hide()
211.                            self.app = QE.QuizEditor(self.window,self.CallBack)
212.
213.            def RunDashBoard(self):
214.                    if self.app is None:
215.                            self.Hide()
216.                            self.user.DashBoard()
217.
218.
219.            def Hide(self):
220.                    self.frame.pack_forget()
221.
222. if __name__ == "__main__":
223.            A = Application()
224.
225.
```

quiz.py

```
1. import tkinter as tk
2. from tkinter import messagebox as mb
3. import tkinter.scrolledtext as st
4. import json
5. import sqlite3
6. import time
7. from datetime import datetime
8. from datetime import date
9.
10.  VERY_LARGE_FONT = ("Verdana",30)
11.  LARGE_FONT = ("Verdana",25)
12.  MEDIUM_FONT = ("Verdana",15)
13.  SMALL_FONT =("Verdana",10)
14.
15.  class ScrollableFrame(tk.Frame):
16.      def __init__(self, container, *args, **kwargs):
17.          super().__init__(container, *args, **kwargs)
18.          canvas = tk.Canvas(self)
19.          scrollbar = tk.Scrollbar(self, orient="vertical",
    command=canvas.yview)
20.          self.scrollable_frame = tk.Frame(canvas)
21.
22.          self.scrollable_frame.bind(
23.              "<Configure>",
24.              lambda e: canvas.configure(
25.                  scrollregion=canvas.bbox("all")
26.              )
27.          )
28.
29.          canvas.create_window((0, 0), window=self.scrollable_frame,
    anchor="nw")
30.
31.          canvas.configure(yscrollcommand=scrollbar.set)
32.
33.          canvas.pack(side="left", fill="both", expand=True)
34.          scrollbar.pack(side="right", fill="y",)
35.
36.  class RadioButton:
37.          def __init__(self,root,_id=None,callback=None):
38.                  self.root = root
39.                  self.checked =
    tk.PhotoImage(file="resources/radiochecked.png")
40.                  self.unchecked =
    tk.PhotoImage(file="resources/radiounchecked.png")
41.                  self.state = False
42.                  self._id = _id
43.                  self.button =
    tk.Button(self.root,image=self.unchecked,command=self.Toggle,relief=tk.
    FLAT)
44.                  self.callback = callback
45.
46.          def Toggle(self):
47.                  if self.state is True:
```

```
48.                         self.state = False
49.                         if self.callback is not None:
50.                                 self.callback(-1)
51.                         self.button.configure(image=self.unchecked)
52.                 else:
53.                         self.state = True
54.                         self.button.configure(image=self.checked)
55.                         if self.callback is not None:
56.                                 self.callback(self._id)
57.
58.         def GetId(self):
59.                 return self._id
60.
61.         def GetState(self):
62.                 return self.state
63.
64.         def SetState(self,state=None):
65.                 if state is not None:
66.                         self.state = state
67.                         if state is False:
68.
    self.button.configure(image=self.unchecked)
69.                         else:
70.
    self.button.configure(image=self.checked)
71.
72.         def Grid(self,row=None,column=None):
73.                 self.button.grid(row=row,column=column)
74.
75.
76.         def Pack(self):
77.                 self.button.pack()
78.
79.  class CheckButton:
80.         def __init__(self,root,_id=None,callback=None):
81.                 self.root = root
82.                 self.checked =
   tk.PhotoImage(file="resources/checked.png")
83.                 self.unchecked =
   tk.PhotoImage(file="resources/unchecked.png")
84.                 self.state = False
85.                 self._id = _id
86.                 self.button =
   tk.Button(self.root,image=self.unchecked,relief=tk.FLAT,command=self.To
   ggle)
87.                 self.callback = callback
88.
89.         def Toggle(self):
90.                 if self.state is True:
91.                         self.state = False
92.                         if self.callback is not None:
93.                                 self.callback(self._id)
94.                         self.button.configure(image=self.unchecked)
95.                 else:
96.                         self.state = True
97.                         self.button.configure(image=self.checked)
98.                         if self.callback is not None:
```

```
                                self.callback(self._id)

        def GetId(self):
                return self._id

        def GetState(self):
                return self.state

        def SetState(self,state=None):
                if state is not None:
                        self.state = state
                        if state is False:

    self.button.configure(image=self.unchecked)
                        else:

    self.button.configure(image=self.checked)

        def Grid(self,row=0,column=0):
                self.button.grid(row=row,column=column)


class Question:
        def __init__(self,root,data=None,num=0):
                self.root = root
                self._id = None
                self.frame = tk.Frame(self.root)
                self.optionstxt = []
                self.optionsbtn = []
                self.var =  None
                self.keys = []
                self._type = None
                if data is not None:
                        self._id=data["id"]
                        self.questionlbl =
   tk.Label(self.frame,width=3,text=str(num),font=LARGE_FONT)
                        self.questionlbl.grid(row=0,column=0)
                        self.questiontxt =
   st.ScrolledText(self.frame,font=MEDIUM_FONT,height=2,width=65,wrap=tk.W
   ORD)

    self.questiontxt.insert(tk.INSERT,data["question"])
                        self.questiontxt.configure(state="disabled")
                        self.questiontxt.grid(row=0,column=1,)
                        self.point = data["point"]
                        self.optionframe = ScrollableFrame(self.frame)
                        if data["type"] == "single":
                                self._type = "single"
                                for i in range(0,len(data["options"])):
                                        if data["options"][i]["key"] ==
   True:

    self.keys.append(data["options"][i]["id"])

    self.optionsbtn.append(RadioButton(self.optionframe.scrollable_frame,d
   ata["options"][i]["id"],self.CallBackRadio))
                                        self.optionsbtn[i].Grid(i+1,0)
```

```
146.
147.
     self.optionstxt.append(st.ScrolledText(self.optionframe.scrollable_fra
   me,font=MEDIUM_FONT,height=2,width=60))
148.
     self.optionstxt[i].insert(tk.INSERT,data["options"][i]["option"])
149.
     self.optionstxt[i].configure(state="disabled")
150.
     self.optionstxt[i].grid(row=i+1,column=1,)
151.                         else:
152.                                 self._type = "multiple"
153.                                 self.var = []
154.                                 for i in range(0,len(data["options"])):
155.                                         if data["options"][i]["key"] ==
   True:
156.
     self.keys.append(data["options"][i]["id"])
157.
     self.optionsbtn.append(CheckButton(self.optionframe.scrollable_frame,d
   ata["options"][i]["id"],self.CallBackCheck))
158.                                         self.optionsbtn[i].Grid(i+1,0)
159.
160.
     self.optionstxt.append(st.ScrolledText(self.optionframe.scrollable_fra
   me,font=MEDIUM_FONT,height=2,width=50))
161.
     self.optionstxt[i].insert(tk.INSERT,data["options"][i]["option"])
162.
     self.optionstxt[i].configure(state="disabled")
163.
     self.optionstxt[i].grid(row=i+1,column=1,)
164.
     self.optionframe.grid(row=1,column=1,sticky="nwse")
165.
166.         def CallBackRadio(self,_id=None):
167.                 if _id is not None:
168.                         if _id == -1:
169.                                 self.var = None
170.                                 for i in self.optionsbtn:
171.                                         i.SetState(False)
172.                         else:
173.                                 self.var = _id
174.                                 for i in self.optionsbtn:
175.                                     if i.GetId() != _id and
   i.GetState() == True:
176.                                         i.SetState(False)
177.
178.         def CallBackCheck(self,_id=None):
179.                 if _id is not None:
180.                         for i in range(0,len(self.var)):
181.                                 if _id == self.var[i]:
182.                                         del self.var[i]
183.                                         break
184.                         else:
185.                                 self.var.append(_id)
186.                         self.var.sort()
```

```
187.
188.         def Grid(self,row=0,column=0):
189.
     self.frame.grid(row=row,column=column,columnspan=2,sticky="nwse",padx=
   50)
190.
191.         def Hide(self):
192.                 self.frame.grid_forget()
193.
194.         def Response(self):
195.                 #data = ["section id","question
   id","keys","point","options id chosen",]
196.                 if self._type == "single":
197.                         if self.var is None:
198.                                 self.var = []
199.                         else:
200.                                 self.var = [self.var,]
201.                 data = [0,self._id,self.keys,self.point,self.var,]
202.                 return data
203.
204. class Section:
205.         def __init__(self,root,data=None):
206.                 self.root = root
207.                 self._id = None
208.                 self.frame = tk.Frame(self.root)
209.                 self.questions = []
210.                 self.sectionlbl =
   tk.Label(self.frame,text=data["section"],font=LARGE_FONT)
211.                 self.currquestion = 0
212.                 if data is not None:
213.                         self._id = data["id"]
214.                         for i in range(0,len(data["questions"])):
215.
     self.questions.append(Question(self.frame,data["questions"][i],i+1))
216.                 self.prevbtn =
   tk.Button(self.frame,text="Back",font=MEDIUM_FONT,command=self.Back,)
217.                 self.nextbtn =
   tk.Button(self.frame,text="Next",font=MEDIUM_FONT,command=self.Next,)
218.                 if len(self.questions) == 1 or 0:
219.                         self.prevbtn.configure(state="disabled")
220.                         self.nextbtn.configure(state="disabled")
221.
222.         def Jump(self,i=0):
223.                 self.questions[self.currquestion].Hide()
224.                 self.currquestion = i
225.                 self.questions[self.currquestion].Grid(1,0)
226.
227.         def Grid(self,row=0,column=0):
228.
     self.sectionlbl.grid(row=0,column=0,sticky="nw",pady=10,padx=50)
229.                 if len(self.questions) > 0:
230.                         self.questions[0].Grid(1,0)
231.
     self.prevbtn.grid(row=2,column=0,sticky="nw",pady=10,padx=50)
232.                 self.nextbtn.grid(row=2,column=1,sticky="ne",pady=10,)
233.                 self.frame.grid(row=row,column=column,sticky="nwse")
234.
```

```
235.         def Next(self):
236.                 if self.currquestion < len(self.questions)-1:
237.                         self.questions[self.currquestion].Hide()
238.                         self.currquestion += 1
239.                         self.questions[self.currquestion].Grid(1,0)
240.
241.         def Back(self):
242.                 if self.currquestion > 0 :
243.                         self.questions[self.currquestion].Hide()
244.                         self.currquestion -= 1
245.                         self.questions[self.currquestion].Grid(1,0)
246.
247.         def Hide(self):
248.                 self.frame.grid_forget()
249.
250.         def Response(self):
251.                 data = []
252.                 for i in range(0,len(self.questions)):
253.                         response = self.questions[i].Response()
254.                         response[0] = self._id
255.                         data.append(response)
256.                 return data
257.
258. class Test:
259.         def __init__(self,root,data=None,callback=None):
260.                 self.root = root
261.                 self.callback = callback
262.                 self._id =None
263.                 self.frame =tk.Frame(self.root)
264.                 self.btnframe =
   ScrollableFrame(self.frame,height=100,width=50)
265.                 self.testlbl =
   tk.Label(self.frame,text=data["test"],font=LARGE_FONT,pady=10,padx=50)
266.                 self.timelimit = []
267.                 self.timelimitlbl = None
268.                 self.sections = []
269.                 self.currsection = 0
270.                 self.instructionframe = None
271.                 self.data = data
272.                 if data is not None:
273.                         self._id = data["id"]
274.                         for i in range(0,len(data["sections"])):
275.
     self.sections.append(Section(self.frame,data["sections"][i]))
276.                         if data["time_limit"]["hour"] == "0" and
   data["time_limit"]["minute"] == "0":
277.                                 timestr = "No Time Limit"
278.                         else:
279.
     self.timelimit.append(int(data["time_limit"]["hour"]))
280.
     self.timelimit.append(int(data["time_limit"]["minute"]))
281.
     self.timelimit.append(int(data["time_limit"]["second"]))
282.                                 if self.timelimit[0]>9:
283.                                         timestr = str(self.timelimit[0])
284.                                 else:
```

```
285.                                        timestr = "0" +
   str(self.timelimit[0])
286.                                if self.timelimit[1]>9:
287.                                        timestr =
   timestr+":"+str(self.timelimit[1])
288.                                else:
289.                                        timestr = timestr+":0" +
   str(self.timelimit[1])
290.                                if self.timelimit[2]>9:
291.                                        timestr =
   timestr+":"+str(self.timelimit[2])
292.                                else:
293.                                        timestr = timestr+":0" +
   str(self.timelimit[2])
294.                            self.timelimitlbl =
   tk.Label(self.frame,text=timestr,font=LARGE_FONT)
295.                    self.prevbtn = tk.Button(self.frame,text="Previous
   Section",font=MEDIUM_FONT,command=self.Back,)
296.                    self.nextbtn =tk.Button(self.frame,text="Next
   Section",font=MEDIUM_FONT,command=self.Next,)
297.                    if len(self.sections) == 1 or 0:
298.                            self.prevbtn.configure(state="disabled")
299.                            self.nextbtn.configure(state="disabled")
300.                    self.submitbtn =
   tk.Button(self.frame,text="Submit",font=MEDIUM_FONT,command=self.Submit
   ,)
301.                    self.exitbtn =
   tk.Button(self.frame,text="Exit",font=MEDIUM_FONT,command=self.Exit)
302.
303.                    for i in range(0,len(data["sections"])):
304.                            lbl
   =tk.Label(self.btnframe.scrollable_frame,text=data["sections"][i]["sect
   ion"],font=MEDIUM_FONT)
305.                            lbl.grid(row=i*2,column=0,sticky=tk.NW,padx=50)
306.                            frame =
   tk.Frame(self.btnframe.scrollable_frame)
307.                            for j in
   range(0,len(data["sections"][i]["questions"])):
308.                                    btn =
   tk.Button(frame,text=str(j+1),width=3,font=MEDIUM_FONT,bg="gray",fg="wh
   ite",relief="flat",command=lambda ii=i,jj=j:self.Jump(ii,jj))
309.
     btn.grid(row=int(j/4),column=j%4,padx=5,pady=5,sticky="nw")
310.
     frame.grid(row=i*2+1,column=0,sticky="nw",padx=50)
311.
312.            def Grid(self,):
313.                    self.HideInstruction()
314.                    self.testlbl.grid(row=0,column=0,sticky="nw",)
315.
     self.timelimitlbl.grid(row=0,column=1,sticky="ne",padx=50)
316.                    if self.data["time_limit"]["hour"] == "0" and
   self.data["time_limit"]["minute"] == "0":
317.                            pass
318.                    else:
319.
     self.timelimitlbl.after(1000,self.UpdateCountDownTimer)
```

```
320.                if len(self.sections) > 0 :
321.                        self.sections[0].Grid(row=1,column=0,)
322.
    self.prevbtn.grid(row=2,column=0,sticky="nw",pady=30,padx=50)
323.
    self.nextbtn.grid(row=2,column=1,sticky="ne",pady=30,padx=50)
324.
    self.submitbtn.grid(row=3,column=0,sticky="nw",pady=30,padx=50)
325.
    self.exitbtn.grid(row=3,column=1,sticky="ne",pady=30,padx=50)
326.                self.btnframe.grid(row=1,column=1,sticky="nwse",)
327.                #self.frame.pack(expand=True,fill="both",)
328.                self.frame.grid(row=0,column=0,sticky="nwse")
329.                self.frame.rowconfigure(0,weight=1)
330.                self.frame.columnconfigure(0,weight=1)
331.
332.
333.
334.        def Jump(self,i=0,j=0):
335.                self.sections[self.currsection].Hide()
336.                self.currsection = i
337.                self.sections[self.currsection].Grid(row=1,column=0,)
338.                self.sections[self.currsection].Jump(j)
339.
340.        def Next(self):
341.                if self.currsection < len(self.sections)-1:
342.                        self.sections[self.currsection].Hide()
343.                        self.currsection += 1
344.
    self.sections[self.currsection].Grid(row=1,column=0,)
345.
346.        def Back(self):
347.                if self.currsection > 0 :
348.                        self.sections[self.currsection].Hide()
349.                        self.currsection -= 1
350.
    self.sections[self.currsection].Grid(row=1,column=0,)
351.
352.        def UpdateCountDownTimer(self):
353.                if self.timelimit[2] == 0:
354.                        self.timelimit[2] = 59
355.                        if self.timelimit[1] == 0:
356.                                self.timelimit[1] = 59
357.                                if self.timelimit[0] == 0:
358.                                        self.timelimit[1] = 0
359.                                        self.timelimit[2] = 0
360.                                        self.Response()
361.                                else:
362.                                        self.timelimit[0] -= 1
363.
    self.timelimitlbl.after(1000,self.UpdateCountDownTimer)
364.                        else:
365.                                self.timelimit[1] -= 1
366.
    self.timelimitlbl.after(1000,self.UpdateCountDownTimer)
367.                else:
368.                        self.timelimit[2] -= 1
```

```
369.
    self.timelimitlbl.after(1000,self.UpdateCountDownTimer)
370.                 if self.timelimit[0]>9:
371.                         timestr = str(self.timelimit[0])
372.                 else:
373.                         timestr = "0" + str(self.timelimit[0])
374.                 if self.timelimit[1]>9:
375.                         timestr = timestr+":"+str(self.timelimit[1])
376.                 else:
377.                         timestr = timestr+":0" + str(self.timelimit[1])
378.                 if self.timelimit[2]>9:
379.                         timestr = timestr+":"+str(self.timelimit[2])
380.                 else:
381.                         timestr = timestr+":0" + str(self.timelimit[2])
382.                 if self.timelimit[0] == 0 and self.timelimit[1] == 0
   and self.timelimit[2] == 0:
383.                         pass
384.                 else:
385.                    self.timelimitlbl.configure(text=timestr)
386.
387.         def Instruction(self,data=None,callback1=None,callback2=None):
388.                 self.instructionframe = tk.Frame(self.root)
389.                 testlbl =
   tk.Label(self.instructionframe,text=data["test"],font=LARGE_FONT)
390.                 instructionlbl =
   tk.Label(self.instructionframe,text="Instructions",font=LARGE_FONT)
391.                 instructiontxt =
   tk.Text(self.instructionframe,font=MEDIUM_FONT)
392.                 if "instructions" in data:
393.                         for i in range(0,len(data["instructions"])):
394.                                 instructiontxt.insert(tk.INSERT,"• " +
   data["instructions"][i]["instruction"]+ "\n")
395.                 instructiontxt.configure(state="disabled")
396.                 instructionacceptbtn =
   tk.Button(self.instructionframe,text="Accept",font=MEDIUM_FONT,command=
   callback1)
397.                 instructiondeclinebtn =
   tk.Button(self.instructionframe,text="Decline",font=MEDIUM_FONT,command
   =callback2)
398.                 testlbl.grid(row=0,column=0,sticky="W",padx=200)
399.                 instructionlbl.grid(row=1,column=0,sticky="W",padx=200)
400.
    instructiontxt.grid(row=2,column=0,columnspan=2,padx=200,sticky="W")
401.
    instructionacceptbtn.grid(row=3,column=0,padx=200,sticky="W")
402.
    instructiondeclinebtn.grid(row=3,column=1,padx=200,sticky="E")
403.                 self.instructionframe.pack(expand=True,fill="both")
404.
405.         def HideInstruction(self):
406.                 self.instructionframe.pack_forget()
407.
408.         def Exit(self):
409.                 exitmb = mb.askquestion("Exit","Are you sure you want
   to exit?",icon="warning")
410.                 if exitmb == "yes":
411.                         self.Response()
```

```
412.
413.        def Submit(self):
414.                submitmb = mb.askquestion("Exit","Are you sure you want
   to submit now?",icon="question")
415.                if submitmb == "yes":
416.                        self.Response()
417.
418.        def Response(self):
419.                if self.callback is not None:
420.                        data = {}
421.                        data["test"] = self._id
422.                        data["response"] = []
423.                        for i in range(0,len(self.sections)):
424.
    data["response"].extend(self.sections[i].Response())
425.                        self.callback(data)
426.
427.        def Destroy(self):
428.                self.frame.destroy()
429.
430. class Quiz:
431.        def __init__(self,root,user_id=None,callback=None):
432.                self.root = root
433.                self.user_id =user_id
434.                self.quiz_id = None
435.                self.menuframe = tk.Frame(self.root)
436.                self.test = None
437.                self.conn = sqlite3.connect("Quiz.db")
438.                self.menubuttons = None
439.                self.Menu()
440.                self.menuframe.pack()
441.                self.data = None
442.                self.callback  = callback
443.
444.        def Menu(self):
445.                if self.menubuttons is None:
446.                        cursor =  self.conn.execute("SELECT * FROM
   quizzes")
447.                        row = cursor.fetchall()
448.                        label =
   tk.Label(self.menuframe,text="Quizzes",font=("Verdana",50))
449.                        label.grid(row=0,column=0,sticky="W")
450.                        exitbtn =
   tk.Button(self.menuframe,text="Exit",font=MEDIUM_FONT,command=self.Exit
   )
451.                        exitbtn.grid(row=0,column=2,sticky="E")
452.                        self.menubuttons = []
453.                        if len(row) == 0:
454.                                noquizavailablelbl =
   tk.Label(self.menuframe,text="No quizzes available right
   now!",font=VERY_LARGE_FONT)
455.
    noquizavailablelbl.grid(row=1,column=0,padx=10,sticky="W")
456.                        else:
457.                                for i in range(0,len(row)):
```

```
458.
    self.menubuttons.append(tk.Button(self.menuframe,text=row[i][1],font=(
   "Verdana",30),relief=tk.FLAT,command=lambda j=i:self.Start(row[j],j)))
459.
    self.menubuttons[i].grid(row=i+1,column=0,padx=10,sticky="W")
460.                         self.menuframe.pack()
461.                 else:
462.                         self.test.HideInstruction()
463.                         self.menuframe.pack()
464.
465.         def Start(self,row,button_row):
466.                 file = open(row[2],"r")
467.                 try:
468.                         data = json.loads(file.read())
469.                 except:
470.                         return
471.                 if  data["date"]["year"] != 0 and data["date"]["month"]
   != 0 and  data["date"]["day"] != 0 :
472.                         if date.today().year == data["date"]["year"]
   and date.today().month == data["date"]["month"] and date.today().day ==
   data["date"]["day"]:
473.                                 pass
474.                         else:
475.                                 if data["time"]["hour"] == 0  and
   data["time"]["minute"] == 0:
476.                                         lbl =
   tk.Label(self.menuframe,text="Quiz will be available on " +
   str(data["date"]["day"]) + "/" + str(data["date"]["month"]) + "/" +
   str(data["date"]["year"]),font=MEDIUM_FONT)
477.                                 else:
478.                                         timestr =
   str(data["time"]["hour"]) + ":"
479.                                         if data["time"]["minute"] < 10:
480.                                                 timestr += "0" +
   str(data["time"]["minute"])
481.                                         else:
482.                                                 timestr +=
   str(data["time"]["minute"])
483.                                         lbl =
   tk.Label(self.menuframe,text="Quiz will be available on " +
   str(data["date"]["day"]) + "/" + str(data["date"]["month"]) + "/" +
   str(data["date"]["year"]) + " " + timestr  ,font=MEDIUM_FONT)
484.                                 lbl.grid(row=button_row+1,column=1)
485.                                 return
486.                 if data["time"]["hour"] != 0 or data["time"]["minute"]
   != 0:
487.                         now = datetime.now()
488.                         current_time = current_time =
   now.strftime("%H:%M:%S")
489.                         current_time = current_time.split(":")
490.                         current_time[0] = int(current_time[0])
491.                         current_time[1] = int(current_time[1])
492.                         if current_time[0] >= data["time"]["hour"] and
   current_time[1] >= data["time"]["minute"]:
493.                                 pass
494.                         else:
495.                                 quiz_time = ""
```

```
496.                                    if data["time"]["hour"] < 10:
497.                                            quiz_time = "0"+
   str(data["time"]["hour"])
498.                                    else:
499.                                            quiz_time =
   str(data["time"]["hour"])
500.                                    quiz_time += ":"
501.                                    if data["time"]["minute"] < 10:
502.                                            quiz_time += "0"+
   str(data["time"]["minute"])
503.                                    else:
504.                                            quiz_time +=
   str(data["time"]["minute"])
505.                                    lbl =
   tk.Label(self.menuframe,text="Quiz will be available at " +
   quiz_time,font=MEDIUM_FONT)
506.                                    lbl.grid(row=button_row+1,column=1)
507.                                    return
508.                    self.test = Test(self.root,data,self.CallBack)
509.                    conn = sqlite3.connect("Quiz.db")
510.                    cur = conn.cursor()
511.                    cur.execute("SELECT response_count FROM responses WHERE
   user=? AND quiz=?",(self.user_id,data["id"]))
512.                    rrow = cur.fetchall()
513.                    if len(rrow) > 0:
514.                            if rrow[0][0] < data["response_limit"] or
   data["response_limit"] == 0:
515.                                    self.test =
   Test(self.root,data,self.CallBack)
516.                                    self.menuframe.pack_forget()
517.
     self.test.Instruction(data,self.test.Grid,self.Menu)
518.                                    self.data = data
519.                            else:
520.                                    lbl =
   tk.Label(self.menuframe,text="Already Exceeded Response
   Limit!",font=MEDIUM_FONT)
521.                                    lbl.grid(row=button_row+1,column=1)
522.                    else:
523.                            self.test = Test(self.root,data,self.CallBack)
524.                            self.menuframe.pack_forget()
525.
     self.test.Instruction(data,self.test.Grid,self.Menu)
526.                            self.data = data
527.
528.
529.            def CallBack(self,data=None):
530.                    conn = sqlite3.connect("Quiz.db")
531.                    curr = conn.cursor()
532.                    curr.execute("SELECT * FROM responses WHERE user=? AND
   quiz=?",(self.user_id,data["test"],))
533.                    row = curr.fetchall()
534.                    if len(row) == 0:
535.                            curr.execute("SELECT username FROM users WHERE
   id=?",(self.user_id,))
536.                            row = curr.fetchall()
537.                            uname = row[0][0]
```

```
538.                         curr.execute("SELECT name,id FROM quizzes WHERE
   id=?",(data["test"],))
539.                         row = curr.fetchall()
540.                         qname = row[0][0]
541.                         path = "response/"+uname+"_"+qname+".json"
542.                         file = open(path,"w")
543.                         self.Evaluate(data)
544.                         json.dump(data,file)
545.                         curr.execute("INSERT INTO
   responses(user,quiz,response)
   VALUES(?,?,?)",(self.user_id,row[0][1],path,))
546.                 else:
547.                         curr.execute("UPDATE responses SET
   response_count=response_count+1 WHERE user=? AND quiz=?
   ",(self.user_id,row[0][1],))
548.                         file = open(row[0][2],"w")
549.                         self.Evaluate(data)
550.                         json.dump(data,file)
551.                 conn.commit()
552.                 self.test.Destroy()
553.                 del self.test
554.                 self.test = None
555.                 self.menuframe.pack()
556.
557.
558.         def Evaluate(self,response=None,):
559.                 if self.data and response is not None:
560.                         response["max_points"] = 0
561.                         response["score"] = 0
562.                         response["correct"] = 0
563.                         response["incorrect"] = 0
564.                         response["no_response"] = 0
565.                         for i in range(0,len(response["response"])):
566.                                 response["max_points"] +=
   response["response"][i][3]
567.                                 if response["response"][i][2] ==
   response["response"][i][4]:
568.                                         #correct
569.                                         response["score"] +=
   response["response"][i][3]
570.
     response["response"][i].append(1)
571.                                         response["correct"] += 1
572.                                 elif  len(response["response"][i][4])
   == 0:
573.                                         #no response
574.
     response["response"][i].append(0)
575.                                         response["no_response"] += 1
576.                                 else:
577.                                         #wrong
578.                                         response["response"][i].append(-
   1)
579.                                         response["incorrect"] += 1
580.
581.         def Exit(self):
582.                 if self.menuframe is not None:
```

```
583.                         self.menuframe.destroy()
584.                 if __name__ == "__main__":
585.                         self.root.destroy()
586.                 else:
587.                         if self.callback is not None:
588.                                 self.callback()
589.
590.
591. if __name__ == "__main__" :
592.         window = tk.Tk()
593.         window.geometry("1024x768")
594.         window.title("Quiz App")
595.         #file = open("quiz.json","r")
596.         #data = json.loads(file.read())
597.
598.         Q =Quiz(window,1)
599.         window.mainloop()
600.
```

user.py

```
1. import tkinter as tk
2. import matplotlib.pyplot as plt
3. import matplotlib.figure
4. import matplotlib.patches
5. from tkinter import filedialog as fd
6. from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
7. from tkinter import messagebox as mb
8. import sqlite3
9. import json
10.  from fpdf import FPDF
11.  from datetime import datetime
12.  from datetime import date
13.  import os
14.
15.  VERY_LARGE_FONT = ("Verdana",35)
16.  LARGE_FONT = ("Verdana",25)
17.  MEDIUM_FONT = ("Verdana",15)
18.  SMALL_FONT =("Verdana",13)
19.  VERY_SMALL_FONT = ("Verdana",8)
20.
21.  class ScrollableFrame(tk.Frame):
22.      def __init__(self, container, *args, **kwargs):
23.          super().__init__(container, *args, **kwargs)
24.          canvas = tk.Canvas(self)
25.          scrollbar = tk.Scrollbar(self, orient="vertical",
    command=canvas.yview)
26.          self.scrollable_frame = tk.Frame(canvas)
27.
28.          self.scrollable_frame.bind(
29.              "<Configure>",
30.              lambda e: canvas.configure(
31.                  scrollregion=canvas.bbox("all")
32.              )
33.          )
34.
35.          canvas.create_window((0, 0), window=self.scrollable_frame,
    anchor="nw")
36.
37.          canvas.configure(yscrollcommand=scrollbar.set)
38.
39.          canvas.pack(side="left", fill="both", expand=True)
40.          scrollbar.pack(side="right", fill="y",)
41.
42.  class User:
43.          def
    __init__(self,root,callback=None,_id=None,username=None,password=None,p
    rivilege=None):
44.                  self.root =root
45.                  self.callback = callback
46.                  self._id = _id
47.                  self.username =  username
48.                  self.password = password
49.                  self.privilege = privilege
50.                  self.viewresponseframe = None
```

```
51.                 self.frame =  None
52.                 self.userleftframe =  None
53.                 self.userdetailsframe=  None
54.                 self.userresponseframe=  None
55.                 self.userrightframe=  None
56.                 self.usernameentry=  None
57.                 self.passwordentry=  None
58.                 self.passwordviewbtn=  None
59.                 self.userdetailsframe=  None
60.                 self.userleftbottomframe = None
61.
62.                 self.passwordviewstatus = False
63.                 self.passwordstatus = False
64.
65.         def DashBoard(self):
66.                 if self.frame is not None:
67.                         self.frame.pack(fill="both",expand="true")
68.                         return
69.                 self.frame = tk.Frame(self.root)
70.                 self.userleftframe =
   tk.LabelFrame(self.frame,height=self.root.winfo_screenheight()/10*9,wid
   th=self.root.winfo_screenwidth()/20*6)
71.
72.                 self.userdetailsframe =
   tk.LabelFrame(self.userleftframe,)
73.
74.                 self.userresponseframe =
   ScrollableFrame(self.userleftframe,)
75.
76.                 self.userrightframe =
   tk.Frame(self.frame,height=self.root.winfo_screenheight()/10*9,width=se
   lf.root.winfo_screenwidth()/20*14)
77.
78.                 usernamelbl = tk.Label(self.userdetailsframe,text="User
   Name :",font=SMALL_FONT)
79.                 usernamelbl.grid(row=0,column=0)
80.                 self.usernameentry =
   tk.Entry(self.userdetailsframe,width=16,font=SMALL_FONT,)
81.                 self.usernameentry.insert(tk.INSERT,self.username)
82.                 self.usernameentry.configure(state="disabled")
83.                 self.usernameentry.grid(row=0,column=1,)
84.
85.                 passwordlbl
   =tk.Label(self.userdetailsframe,text="Password:",font=SMALL_FONT)
86.                 passwordlbl.grid(row=1,column=0)
87.                 self.passwordentry =
   tk.Entry(self.userdetailsframe,show="*",width=16,font=SMALL_FONT)
88.                 self.passwordentry.insert(tk.INSERT,self.password)
89.                 self.passwordentry.configure(state="disabled")
90.                 self.passwordentry.grid(row=1,column=1)
91.                 self.passwordviewbtn =
   tk.Button(self.userdetailsframe,text="Show",font=VERY_SMALL_FONT,comman
   d=self.PasswordViewStatusChange)
92.                 self.passwordviewbtn.grid(row=1,column=2)
93.                 self.passwordeditbtn =
   tk.Button(self.userdetailsframe,text="Edit",font=VERY_SMALL_FONT,comman
   d=self.PasswordEditStatusChange)
```

```
94.                 self.passwordeditbtn.grid(row=1,column=3)
95.
96.                 privilegelbl =tk.Label(self.userdetailsframe,text="Role
   :",font=SMALL_FONT)
97.                 privilegelbl.grid(row=2,column=0)
98.                 privilege = None
99.                 if self.privilege == 0:
100.                        privilege =
   tk.Label(self.userdetailsframe,text="User",font=SMALL_FONT)
101.                else:
102.                       privilege =
   tk.Label(self.userdetailsframe,text="User",font=SMALL_FONT)
103.                privilege.grid(row=2,column=1)
104.
105.                responseslbl =
   tk.Label(self.userresponseframe.scrollable_frame,text="User
   Responses",font=MEDIUM_FONT)
106.                responseslbl .grid(row=0,column=0)
107.
108.                conn = sqlite3.connect("Quiz.db")
109.                cur = conn.cursor()
110.
111.                if self.privilege == 1:
112.                       cur.execute("SELECT DISTINCT quiz FROM
   responses")
113.                       row = cur.fetchall()
114.                       if len(row) > 0:
115.                              for i in range(0,len(row)):
116.                                     cur.execute("SELECT * FROM
   responses WHERE user=? AND quiz =?",(self._id,row[i][0]))
117.                                     rrow = cur.fetchone()
118.                                     cur.execute("SELECT * FROM
   quizzes WHERE id=? ",(row[i][0],))
119.                                     name = cur.fetchone()
120.                                     if name is not None:
121.                                            rbutton =
   tk.Button(self.userresponseframe.scrollable_frame,text=name[1],font=SMA
   LL_FONT,command=lambda r=rrow ,n = name[1] ,: self.ViewResponse(r,n))
122.
     rbutton.grid(row=i+1,column=0,sticky=tk.W+tk.E)
123.                                    if rrow is None:
124.
     rbutton.configure(state="disabled")
125.                                    exportbtn =
   tk.Button(self.userresponseframe.scrollable_frame,text="Export",font=SM
   ALL_FONT,command=lambda n=name[1],qid =
   name[0]:self.ExportResponse(n,qid))
126.
     exportbtn.grid(row=i+1,column=1,sticky=tk.W+tk.E)
127.                       else:
128.                              noresponsemsglbl =
   tk.Label(self.userresponseframe.scrollable_frame,text="No Responses to
   Show",font=MEDIUM_FONT)
129.
     noresponsemsglbl.grid(row=1,column=0,sticky="E")
130.                else:
```

```
131.                         cur.execute("SELECT * FROM responses WHERE
    user=?",(self._id,))
132.                         row = cur.fetchall()
133.                         if len(row) > 0:
134.                                 for i in range(0,len(row)):
135.                                         cur.execute("SELECT * FROM
    quizzes WHERE id=? ",(row[i][1],))
136.                                         name = cur.fetchone()
137.                                         if name is not None:
138.                                                 rbutton =
    tk.Button(self.userresponseframe.scrollable_frame,text=name[1],font=SMA
    LL_FONT,command=lambda r=row[i] ,n = name[1] ,: self.ViewResponse(r,n))
139.
     rbutton.grid(row=i+1,column=0,sticky=tk.W+tk.E)
140.                         else:
141.                                 noresponsemsglbl =
    tk.Label(self.userresponseframe.scrollable_frame,text="No Responses to
    Show",font=MEDIUM_FONT,)
142.
     noresponsemsglbl.grid(row=1,column=0,sticky="E")
143.
144.                 if self.callback is not None:
145.                         exitbtn =
    tk.Button(self.userleftframe,text="Exit",font=MEDIUM_FONT,command=self.
    Exit)
146.                         exitbtn.grid(row=3,column=0,sticky="w")
147.
148.                 self.userdetailsframe.grid(row=0,column=0,sticky="nwe")
149.
     self.userresponseframe.grid(row=2,column=0,sticky="ns",)
150.                 self.userleftframe.grid(row=0,column=0,sticky="nwse")
151.                 self.userrightframe.grid(row=0,column=1,sticky="nwse")
152.                 self.frame.pack(fill="both",expand="true")
153.                 self.frame.rowconfigure(0,weight=1)
154.
155.         def PasswordViewStatusChange(self):
156.                 if self.passwordviewstatus == False:
157.                         self.passwordviewstatus = True
158.                         self.passwordviewbtn.configure(text="Hide")
159.                         self.passwordentry.configure(show="")
160.                 else:
161.                         self.passwordviewstatus = False
162.                         self.passwordviewbtn.configure(text="Show")
163.                         self.passwordentry.configure(show="*")
164.
165.         def PasswordEditStatusChange(self):
166.                 if self.passwordstatus is False:
167.                         if self.passwordviewstatus is False:
168.                                 self.PasswordViewStatusChange()
169.                         self.passwordentry.configure(state="normal")
170.                         self.passwordeditbtn.configure(text="Save")
171.                         self.passwordstatus = True
172.
     self.passwordviewbtn.configure(state="disabled")
173.                 else:
174.                         self.PasswordSave()
175.                         self.passwordstatus = False
```

```
176.                         self.passwordeditbtn.configure(text="Edit")
177.                         self.passwordentry.configure(state="disabled")
178.                         self.passwordviewbtn.configure(state="normal")
179.
180.
181.
182.         def PasswordSave(self):
183.                 if len(self.passwordentry.get()) > 7:
184.                         conn = sqlite3.connect("Quiz.db")
185.                         cursor = conn.cursor()
186.                         cursor.execute("UPDATE users SET password=?
    WHERE id=?",(self.passwordentry.get(),self._id,))
187.                         conn.commit()
188.                         self.PasswordViewStatusChange()
189.
190.         def ViewResponse(self,response=None,name=None):
191.                 if response and name is not None:
192.                         if self.viewresponseframe is not None:
193.                                 self.viewresponseframe.destroy()
194.                                 self.viewresponseframe = None
195.                         self.viewresponseframe =
    tk.Frame(self.userrightframe,width=self.root.winfo_screenwidth()/20*14)
196.
197.                         file = open(response[2],"r")
198.                         data = json.loads(file.read())
199.                         testnamelbl =
    tk.Label(self.viewresponseframe,text=name,font=LARGE_FONT)
200.
      testnamelbl.grid(row=0,column=1,sticky="ew",pady=20)
201.                         if data["max_points"] == 0:
202.                                 scorelabel =
    tk.Label(self.viewresponseframe,text="Maximum score :
    None",font=MEDIUM_FONT)
203.                                 youscorelbl =
    tk.Label(self.viewresponseframe,text="Your score :
    None",font=MEDIUM_FONT)
204.                         else:
205.                                 scorelabel =
    tk.Label(self.viewresponseframe,text="Maximum score : " +
    str(data["max_points"]),font=MEDIUM_FONT)
206.                                 youscorelbl =
    tk.Label(self.viewresponseframe,text="Your score : "  +
    str(data["score"]) ,font=MEDIUM_FONT)
207.
      scorelabel.grid(row=1,column=0,pady=10,sticky="w")
208.
      youscorelbl.grid(row=1,column=1,pady=10,sticky="w")
209.                         totalscorelbl =
    tk.Label(self.viewresponseframe,text="Total Questions : " +
    str(len(data["response"])),font=MEDIUM_FONT)
210.
      totalscorelbl.grid(row=1,column=2,pady=10,sticky="w")
211.                         fig = matplotlib.figure.Figure(figsize=(5,5))
212.                         ax = fig.add_subplot(111)
213.
214.                         correct = str(data["correct"])
215.                         incorrect = str(data["incorrect"])
```

```
216.                         noresponse = str(data["no_response"])
217.
218.                         correctlbl =
    tk.Label(self.viewresponseframe,text="Correct : " +
    str(correct),font=MEDIUM_FONT)
219.                         incorrectlbl =
    tk.Label(self.viewresponseframe,text="Incorrect : " +
    str(incorrect),font=MEDIUM_FONT)
220.                         noresponselbl =
    tk.Label(self.viewresponseframe,text="No response : " +
    str(noresponse),font=MEDIUM_FONT)
221.
222.
     correctlbl.grid(row=2,column=0,pady=10,sticky="w")
223.
     incorrectlbl.grid(row=2,column=1,pady=10,sticky="w")
224.
     noresponselbl.grid(row=2,column=2,pady=10,sticky="w")
225.
226.                         ax.pie([correct,incorrect,noresponse])
227.                         ax.legend(["Correct","Incorrect","No
    response"])
228.                         #ax.color(["green","red","gray"])
229.                         circle=matplotlib.patches.Circle( (0,0), 0.0,
    color='white')
230.                         ax.add_artist(circle)
231.                         canvas = FigureCanvasTkAgg(fig,
    master=self.viewresponseframe)
232.
     canvas.get_tk_widget().grid(row=3,column=0,columnspan=2)
233.                         canvas.draw()
234.                         fig.savefig("temp.png",format="png",
    bbox_inches='tight')
235.                         exportdata =
    (name,str(data["max_points"]),str(data["score"]),str(len(data["response
    "])),str(correct),str(incorrect),str(noresponse))
236.                         exportbtn =
    tk.Button(self.viewresponseframe,text="Export",font=MEDIUM_FONT,command
    =lambda : self.Export(exportdata))
237.                         exportbtn.grid(row=4,column=0,sticky="sw")
238.
     self.viewresponseframe.pack(expand=True,fill="both",padx=100)
239.
240.         def GetPrivilege(self):
241.                 return self.privilege
242.
243.         def GetUsername(self):
244.                 return self.username
245.
246.         def GetUserId(self):
247.                 return self._id
248.
249.         def Exit(self):
250.                 self.frame.pack_forget()
251.                 if self.viewresponseframe is not None:
252.                         self.viewresponseframe.destroy()
253.                 self.callback()
```

```
254.
255.        def ExportResponse(self,name=None,quiz_id=None):
256.                if quiz_id is not None:
257.                        try:
258.                                conn = sqlite3.connect("Quiz.db")
259.                                cursor = conn.cursor()
260.                                cursor.execute("SELECT * FROM responses
   WHERE quiz=?",(quiz_id,))
261.                                data = cursor.fetchall()
262.                                filedialog = fd.asksaveasfile(filetypes
   = [("CSV","*.csv")], defaultextension = [("CSV","*.csv")])
263.                                if filedialog is not None:
264.                                        filedialog.write("\"serial
   No.\",\"username\",\"quiz name\",\"maximum points\",\"score\",\"total
   questions\",\"correct\",\"incorrect\",\"no response\",\n")
265.                                        for i in range(0,len(data)):
266.                                                temp_file =
   open(data[i][2])
267.                                                rdata =
   json.loads(temp_file.read())
268.                                                cursor.execute("SELECT
   username FROM users WHERE id=?",(data[i][0],))
269.                                                username =
   cursor.fetchone()[0]
270.                                                rrow =
   (str(i+1),username,name,str(rdata["max_points"]),str(rdata["score"]),st
   r(len(rdata["response"])),str(rdata["correct"]),str(rdata["incorrect"])
   ,str(rdata["no_response"]),)
271.                                                temp_file.close()
272.                                                strrow = ",".join(rrow)
   + "\n"
273.                                                filedialog.write(strrow)
274.                                        filedialog.close()
275.                                        mb.showinfo("Export","File
   exported successfully!")
276.                        except:
277.                                mb.showerror("Export","File export
   error!")
278.
279.        def Export(self,data=None):
280.                pdf = FPDF()
281.                pdf.add_page()
282.                pdf.set_font("Arial", size = 8)
283.                now = datetime.now()
284.                current_time = current_time = now.strftime("%H:%M:%S")
285.                pdf.cell(200, 10, txt = "Date : "+str(date.today())+"
   Time: "+current_time ,  ln = 1, align = 'L')
286.                pdf.set_font("Arial", size = 20)
287.                pdf.cell(200, 10, txt = "Quiz : "+data[0] ,  ln = 1,
   align = 'C')
288.                pdf.set_font("Arial", size = 15)
289.                pdf.cell(0, 10, txt = "Username           :
   "+self.username ,  ln = 1, align = 'L')
290.                pdf.cell(0, 10, txt = "Maximum Score      : "+data[1] ,
   ln = 1, align = 'L')
291.                pdf.cell(0, 10, txt = "Scored             : "+data[2] ,
   ln = 1, align = 'L')
```

```
292.                 pdf.cell(0, 10, txt = "Total Questions    : "+data[3] ,
   ln = 1, align = 'L')
293.                 pdf.cell(0, 10, txt = "Correct            : "+data[4] ,
   ln = 1, align = 'L')
294.                 pdf.cell(0, 10, txt = "Incorrect          : "+data[5] ,
   ln = 1, align = 'L')
295.                 pdf.cell(0, 10, txt = "No Response        : "+data[6] ,
   ln = 1, align = 'L')
296.                 pdf.image(name="temp.png",)
297.                 if os.path.exists("temp.png"):
298.                         os.remove("temp.png")
299.                 exportdialog = fd.asksaveasfile(filetypes =
   [("PDF","*.pdf")], defaultextension = [("PDF","*.pdf")])
300.                 if exportdialog is not None:
301.                         pdf.output(exportdialog.name)
302.                         mb.showinfo("Export","Result exported
   successfully!")
303.
304.
305. if __name__ == "__main__":
306.         window = tk.Tk()
307.         U = User(window,None,1,"admin","password",1)
308.         U.DashBoard()
309.         window.mainloop()
```

quizeditor.py

```
1. import tkinter as tk
2. from tkinter import filedialog as fd
3. import tkinter.scrolledtext as st
4. from tkcalendar import Calendar,DateEntry
5. from tkinter import messagebox as mb
6. import datetime as dt
7. import json
8. import sqlite3
9. import os
10.
11.
12.  MEDIUM_FONT = ("Verdana",25)
13.  SMALL_MEDIUM_FONT = ("Verdana",17)
14.  SMALL_FONT = ("Verdana",12)
15.  VERY_SMALL_FONT = ("Verdana",11)
16.
17.  class ScrollableFrame(tk.Frame):
18.      def __init__(self, container, *args, **kwargs):
19.          super().__init__(container, *args, **kwargs)
20.          canvas = tk.Canvas(self)
21.          scrollbar = tk.Scrollbar(self, orient="vertical",
    command=canvas.yview)
22.          self.scrollable_frame = tk.Frame(canvas)
23.
24.          self.scrollable_frame.bind(
25.              "<Configure>",
26.              lambda e: canvas.configure(
27.                  scrollregion=canvas.bbox("all")
28.              )
29.          )
30.
31.          canvas.create_window((0, 0), window=self.scrollable_frame,
    anchor="nw")
32.
33.          canvas.configure(yscrollcommand=scrollbar.set)
34.
35.          canvas.pack(side="left", fill="both", expand=True)
36.          scrollbar.pack(side="right", fill="y",)
37.
38.  class Option:
39.          def __init__(self,root,data=None,_id=None):
40.                  self.root = root
41.                  self.text = tk.StringVar()
42.                  self._id = _id
43.                  self.keyvar = tk.IntVar()
44.                  self.optiontxtlbl =
    tk.Label(self.root,text="Option",font=VERY_SMALL_FONT)
45.                  self.optiontxt =
    st.ScrolledText(self.root,height=1,relief=tk.FLAT,font=VERY_SMALL_FONT)
46.                  self.optionkeylbl = tk.Label(self.root,text="Key")
47.                  self.optionkeychkbtn =
    tk.Checkbutton(self.root,variable=self.keyvar)
48.
49.                  if data is not None:
```

```
50.                         if "option" in data:
51.
    self.optiontxt.insert(tk.INSERT,data["option"])
52.                         if "id" in data:
53.                                 self._id = data["id"]
54.                         if "key"in data:
55.                                 self.keyvar.set(data["key"])
56.
57.         def Grid(self,row=0,column=0):
58.                 self.optiontxtlbl.grid(row=row,column=column,)
59.                 self.optiontxt.grid(row=row,column=column+1,pady=1)
60.                 self.optionkeylbl.grid(row=row,column=column+2)
61.                 self.optionkeychkbtn.grid(row=row,column=column+3)
62.
63.         def Destroy(self):
64.                 self.optiontxtlbl.destroy()
65.                 self.optiontxt.frame.destroy()
66.                 self.optionkeylbl.destroy()
67.                 self.optionkeychkbtn.destroy()
68.
69.         def SetId(self,_id):
70.                 self._id = _id
71.
72.         def GetId(self):
73.                 return self._id
74.
75.         def Export(self):
76.                 data = {}
77.                 data["option"] =
   self.optiontxt.get("1.0","end").rstrip("\n")
78.                 data["id"]  = self._id
79.                 if self.keyvar.get() == 0:
80.                         data["key"] = False
81.                 else:
82.                         data["key"] = True
83.                 return data
84.
85.         def IsKey(self):
86.                 return self.keyvar.get()
87.
88.  class Question:
89.         def __init__(self,root,data=None,_id=None):
90.                 self.root = root
91.                 self._id = _id
92.                 self.options = None
93.                 self.removeoptionsbtn = None
94.                 self.time_limit = None
95.                 self.pointvar = tk.IntVar()
96.
97.                 self.frame = tk.LabelFrame(self.root,)
98.                 self.questiontxtlbl =
   tk.Label(self.frame,text="Question",font=VERY_SMALL_FONT)
99.                 self.questiontxt =
   st.ScrolledText(self.frame,relief=tk.FLAT,height=2,font=VERY_SMALL_FONT
   )
100.                self.quesbtnframe = tk.Frame(self.frame,)
```

```
101.                self.addoptionsbtn =
    tk.Button(self.quesbtnframe,text="Add Option",command=self.AddOption)
102.                self.pointlbl =
    tk.Label(self.quesbtnframe,text="Point")
103.                self.pointspnbox =
    tk.Spinbox(self.quesbtnframe,from_=0,to=999,wrap=True,width=3,state="di
    sabled")
104.                self.pointchkbtn =
    tk.Checkbutton(self.quesbtnframe,variable=self.pointvar,command=self.To
    gglePoint)
105.
106.                if data is not None:
107.                        if "id" in data:
108.                                self._id = data["id"]
109.                        if "question" in data:
110.
      self.questiontxt.insert(tk.INSERT,data["question"])
111.                        if "options" in data:
112.                                for i in range(0,len(data["options"])):
113.
      self.AddOption(data["options"][i])
114.                        if "point" in data:
115.                                if data["point"] != "0":
116.                                        self.pointvar.set(1)
117.
      self.pointspnbox.configure(state="normal")
118.                                        self.pointspnbox.delete(0,"end")
119.
      self.pointspnbox.insert(tk.INSERT,str(data["point"]))
120.
121.        def AddOption(self,data=None,):
122.                if self.options is None:
123.                        self.options = list()
124.                        if data is None:
125.
      self.options.append(Option(self.frame,None,len(self.options)))
126.                        else:
127.
      self.options.append(Option(self.frame,data))
128.                        self.options[len(self.options)-
    1].Grid(len(self.options),0)
129.
130.                        self.removeoptionsbtn = list()
131.
      self.removeoptionsbtn.append(tk.Button(self.frame,text="Remove",comman
    d=lambda i=len(self.options) : self.RemoveOption(i-1)))
132.
      self.removeoptionsbtn[len(self.removeoptionsbtn)-
    1].grid(row=len(self.removeoptionsbtn),column=4,)
133.                else:
134.                        if data is None:
135.
      self.options.append(Option(self.frame,None,len(self.options)))
136.                        else:
137.
      self.options.append(Option(self.frame,data,))
```

```
138.                         self.options[len(self.options)-
   1].Grid(len(self.options),0)
139.
140.
     self.removeoptionsbtn.append(tk.Button(self.frame,text="Remove",comman
   d=lambda i=len(self.options) : self.RemoveOption(i-1)))
141.
     self.removeoptionsbtn[len(self.removeoptionsbtn)-
   1].grid(row=len(self.removeoptionsbtn),column=4,)
142.
143.         def RemoveOption(self,index=None):
144.                 if index is not None:
145.                         self.options[index].Destroy()
146.                         self.removeoptionsbtn[index].destroy()
147.                         del self.options[index]
148.                         del     self.removeoptionsbtn[index]
149.                         for i in range(0,len(self.options)):
150.                                 self.options[i].SetId(i)
151.
     self.removeoptionsbtn[i].config(command=lambda: self.RemoveOption(i))
152.
153.         def Grid(self,row=0,column=0):
154.                 self.questiontxtlbl.grid(row=0,column=0)
155.                 self.questiontxt.grid(row=0,column=1)
156.                 self.addoptionsbtn.grid(row=1,column=0)
157.                 self.pointlbl.grid(row=2,column=0)
158.                 self.pointspnbox.grid(row=2,column=1)
159.                 self.pointchkbtn.grid(row=2,column=2)
160.                 self.quesbtnframe.grid(row=0,column=2)
161.
     self.frame.grid(row=row,column=column,pady=5,ipadx=3,ipady=3,sticky=tk
   .NW)
162.
163.         def Destroy(self):
164.                 self.frame.destroy()
165.
166.         def TogglePoint(self):
167.                 if self.pointvar.get() == 0:
168.                         self.pointspnbox.configure(state="disabled")
169.                 else:
170.                         self.pointspnbox.configure(state="normal")
171.
172.         def Export(self):
173.                 data = {}
174.                 data["id"] = self._id
175.                 data["question"] =
   self.questiontxt.get("1.0","end").rstrip("\n")
176.                 data["options"] = []
177.                 data["type"] = "single"
178.                 if self.pointvar.get() == 0:
179.                         data["point"] = 0
180.                 else:
181.                         data["point"] = int(self.pointspnbox.get())
182.                 if self.options is not None:
183.                         count = 0
184.                         for i in range(0,len(self.options)):
```

```
185.
    data["options"].append(self.options[i].Export())
186.                                 if self.options[i].IsKey() == True:
187.                                         count += 1
188.                         if count>1:
189.                                 data["type"] = "multiple"
190.                 return data
191.
192.         def SetId(self,_id):
193.                 self._id = _id
194.
195.
196. class Section:
197.         def __init__(self,root,data=None,_id=None):
198.                 self.root = root
199.                 self._id = _id
200.                 self.name = None
201.                 self.time_limit = None
202.                 self.questions = None
203.                 self.jumble = None
204.                 self.removequestionsbtn = None
205.
206.                 self.frame = tk.Frame(self.root,)
207.                 self.sectionframe = tk.LabelFrame(self.frame,)
208.                 self.questionframe = tk.Frame(self.frame,)
209.
210.                 self.sectionnamelbl =
    tk.Label(self.sectionframe,text="Section Name",font=VERY_SMALL_FONT)
211.                 self.sectionametext=
    st.ScrolledText(self.sectionframe,relief=tk.FLAT,height="2",font=VERY_S
    MALL_FONT)
212.                 self.sectionbtnframe = tk.Frame(self.sectionframe,)
213.                 #self.sectionremovebtn =
    tk.Button(self.sectionbtnframe,text="Remove",command=lambda:self.frame.
    destroy())
214.                 self.addquestionbtn =
    tk.Button(self.sectionbtnframe,text="Add
    Question",command=self.AddQuestion)
215.
216.                 if data is not None:
217.                         if "id" in data:
218.                                 self._id =data["id"]
219.                         if "section" in data:
220.
    self.sectionametext.insert(tk.INSERT,data["section"])
221.                                 for i in
    range(0,len(data["questions"])):
222.
    self.AddQuestion(data["questions"][i])
223.
224.
225.         def SetName(self,name=None):
226.                 if name is not None:
227.                         self.name = name
228.
229.         def SetTimeLimit(self,time_limit=None):
230.                 if time_limit is not None:
```

```
231.                         pass
232.
233.         def AddQuestion(self,data=None):
234.                 if self.questions is None:
235.                         self.questions = list()
236.                         if data is None:
237.
    self.questions.append(Question(self.questionframe,None,len(self.questi
   ons)))
238.                         else:
239.
    self.questions.append(Question(self.questionframe,data,))
240.                         self.questions[len(self.questions)-
   1].Grid(len(self.questions)-1,0)
241.
242.                         self.removequestionsbtn = list()
243.
    self.removequestionsbtn.append(tk.Button(self.questionframe,text="Remo
   ve",command=lambda i=len(self.questions):self.RemoveQuestion(i-1)))
244.
    self.removequestionsbtn[len(self.removequestionsbtn)-
   1].grid(row=len(self.questions)-1,column=1,sticky= tk.NW)
245.                 else:
246.                         if data is None:
247.
    self.questions.append(Question(self.questionframe,None,len(self.questi
   ons)))
248.                         else:
249.
    self.questions.append(Question(self.questionframe,data,))
250.                         self.questions[len(self.questions)-
   1].Grid(len(self.questions)-1,0)
251.
252.
    self.removequestionsbtn.append(tk.Button(self.questionframe,text="Remo
   ve",command=lambda i=len(self.questions):self.RemoveQuestion(i-1)))
253.
    self.removequestionsbtn[len(self.removequestionsbtn)-
   1].grid(row=len(self.questions)-1,column=1,sticky= tk.NW)
254.
255.         def RemoveQuestion(self,index=None):
256.                 if index is not None:
257.                         self.questions[index].Destroy()
258.                         self.removequestionsbtn[index].destroy()
259.                         del self.questions[index]
260.                         del self.removequestionsbtn[index]
261.                         for i in range(0,len(self.questions)):
262.                                 self.questions[i].SetId(i)
263.
    self.removequestionsbtn[i].config(command=lambda:self.RemoveQuestion(i
   ))
264.
265.         def Destroy(self):
266.                 self.frame.destroy()
267.
268.         def Grid(self,row=0,column=0):
269.                 self.sectionnamelbl.grid(row=0,column=0)
```

```
270.                 self.sectionametext.grid(row=0,column=1)
271.                 self.sectionbtnframe.grid(row=0,column=2)
272.                 #self.sectionremovebtn.grid(row=0,column=0)
273.                 self.addquestionbtn.grid(row=1,column=0)
274.                 self.sectionframe.grid(row=0,column=0,sticky=tk.NW)
275.
    self.questionframe.grid(row=1,column=0,sticky=tk.NW,padx=50)
276.                 self.frame.grid(row=row,column=column,sticky=tk.NW)
277.
278.         def Export(self):
279.                 data = {}
280.                 data["id"] = self._id
281.                 data["section"] =
   self.sectionametext.get("1.0","end").rstrip("\n")
282.                 data["questions"] = []
283.                 if self.questions is not None:
284.                         for i in self.questions:
285.                                 data["questions"].append(i.Export())
286.                 return data
287.
288.         def SetId(self,_id):
289.                 self._id = _id
290.
291. class Instruction:
292.         def __init__(self,data=None):
293.                 self.data = data
294.
295.         def Export(self):
296.                 data = {}
297.                 data["instructions"] = []
298.                 if "time_limit" in self.data:
299.                         if self.data["time_limit"]["hour"] == "0" and
   self.data["time_limit"]["minute"] == "0":
300.
    data["instructions"].append({"instruction":"There is no limit for the
   quiz.",})
301.                         else:
302.                                 if self.data["time_limit"]["hour"] ==
   "0" and self.data["time_limit"]["minute"] != "0" :
303.
    data["instructions"].append({"instruction":"You have " +
   self.data["time_limit"]["minute"] + " minutes", })
304.                                 elif self.data["time_limit"]["hour"] !=
   "0" and self.data["time_limit"]["minute"] == "0":
305.
    data["instructions"].append({"instruction":"You have " +
   self.data["time_limit"]["hour"] + " hours", })
306.                                 else:
307.
    data["instructions"].append({"instruction":"You have " +
   self.data["time_limit"]["hour"] + " hours and " +
   self.data["time_limit"]["minute"] + " minutes", })
308.
309.                 if "sections" in self.data:
310.                         if len(self.data["sections"]) > 0:
```

```
311.
     data["instructions"].append({"instruction": "There are " +
   str(len(self.data["sections"])) + " sections in total",})
312.                                 for i in
   range(0,len(self.data["sections"])):
313.                                         if "questions" in
   self.data["sections"][i]:
314.                                                 if
   len(self.data["sections"][i]["questions"]) > 0 :
315.
     data["instructions"].append({"instruction":"In the section " +
   self.data["sections"][i]["section"] + " there are " +
   str(len(self.data["sections"][i]["questions"])) + " questions in
   total",})
316.
317.                 return data
318.
319. class Test:
320.         def
   __init__(self,root,data=None,_id=None,name=None,path=None):
321.                 self.root = root
322.                 self._id = _id
323.                 self.name = name
324.                 self.path = path
325.                 self.sections = None
326.                 self.removesectionsbtn = None
327.                 self.time_limit = None
328.                 self.instruction = None
329.
330.                 self.testdatevar = tk.IntVar()
331.                 self.testtimevar = tk.IntVar()
332.                 self.testtimelimitvar = tk.IntVar()
333.
334.                 self.frame  = ScrollableFrame(self.root,)
335.                 self.testframe  =
   tk.Frame(self.frame.scrollable_frame,)
336.                 self.sectionframe =
   tk.LabelFrame(self.frame.scrollable_frame,)
337.
338.                 self.testnamelabel = tk.Label(self.testframe,text="Quiz
   Name",font=VERY_SMALL_FONT,)
339.                 self.testnametext =
   st.ScrolledText(self.testframe,relief=tk.FLAT,height=1,font=VERY_SMALL_
   FONT)
340.
341.                 self.testbtnframe = tk.Frame(self.testframe,)
342.                 self.testtimelimitlbl =
   tk.Label(self.testbtnframe,text="Quiz Time Limit")
343.                 self.testtimelimitframe = tk.Frame(self.testbtnframe)
344.                 self.testtimelimithrlbl =
   tk.Label(self.testtimelimitframe,text="Hours")
345.                 self.testtimelimithrspnbox = tk.
   Spinbox(self.testtimelimitframe,from_=0,to=24,wrap=True,width=2,state="
   disabled")
346.                 self.testtimelimitminlbl =
   tk.Label(self.testtimelimitframe,text="Minutes")
```

```
347.                 self.testtimelimitminspnbox = tk.
    Spinbox(self.testtimelimitframe,from_=0,to=59,wrap=True,width=2,state="
    disabled")
348.                 #self.testtimelimitseclbl =
    tk.Label(self.testtimelimitframe,text="Seconds")
349.                 #self.testtimelimitsecspnbox = tk.
    Spinbox(self.testtimelimitframe,from_=0,to=60,wrap=True,width=2,state="
    disabled")
350.                 self.testtimelimitchkbtn =
    tk.Checkbutton(self.testbtnframe,variable=self.testtimelimitvar
    ,command=self.ToggleTimeLimit)
351.                 self.addsectionbtn =
    tk.Button(self.testbtnframe,text="Add Section",command=self.AddSection)
352.                 self.testtimelbl =tk.Label(self.testbtnframe,text="Quiz
    Time")
353.                 self.testdatelbl =
    tk.Label(self.testbtnframe,text="Quiz Date")
354.                 self.testtimeframe = tk.Frame(self.testbtnframe)
355.                 self.testtimehrlbl =
    tk.Label(self.testtimeframe,text="Hours")
356.                 self.testtimehrspnbox = tk.
    Spinbox(self.testtimeframe,from_=0,to=24,wrap=True,width=2,state="disab
    led")
357.                 self.testtimeminlbl =
    tk.Label(self.testtimeframe,text="Minutes")
358.                 self.testtimeminspnbox = tk.
    Spinbox(self.testtimeframe,from_=0,to=59,wrap=True,width=2,state="disab
    led")
359.                 #self.testtimeseclbl =
    tk.Label(self.testtimeframe,text="Seconds")
360.                 #self.testtimesecspnbox = tk.
    Spinbox(self.testtimeframe,from_=0,to=60,wrap=True,width=2,state="disab
    led")
361.                 self.testtimechkbtn =
    tk.Checkbutton(self.testbtnframe,variable=self.testtimevar,command=self
    .ToggleTime)
362.                 self.testdate =
    DateEntry(self.testbtnframe,width=30,bg="darkblue",fg="white",year=2020
    ,mindate=dt.date.today(),state="disabled")
363.                 self.testdatechkbtn =
    tk.Checkbutton(self.testbtnframe,variable=self.testdatevar,command=self
    .ToggleDate)
364.                 self.testlimitresponselbl =
    tk.Label(self.testbtnframe,text="Limit Response")
365.                 self.testlimitresponsespnbox =
    tk.Spinbox(self.testbtnframe,from_=0,to=99,wrap=True,width=2)
366.
367.                 if data is not None:
368.                         if "id" in data:
369.                                 self._id = _id
370.                         if "test" in data:
371.
        self.testnametext.insert(tk.INSERT,data["test"])
372.                                 for i in
    range(0,len(data["sections"])):
373.
        self.AddSection(data["sections"][i])
```

```
374.                         if "date" in data:
375.                                 if data["date"]["day"] == 0 and
   data["date"]["month"] == 0 and data["date"]["year"] == 0:
376.                                         self.testdatevar.set(0)
377.                                 else:
378.                                         self.testdatevar.set(1)
379.
     self.testdate.configure(state="enabled")
380.
     self.testdate.set_date(dt.datetime(data["date"]["year"],data["date"]["
   month"],data["date"]["day"]))
381.                         if "time" in data:
382.                                 if data["time"]["hour"] == 0 and
   data["time"]["minute"] == 0:
383.                                         self.testtimevar.set(0)
384.                                 else:
385.                                         self.testtimevar.set(1)
386.
     self.testtimehrspnbox.configure(state="normal")
387.
     self.testtimehrspnbox.delete(0,"end")
388.
     self.testtimehrspnbox.insert(tk.INSERT,str(data["time"]["hour"]))
389.
     self.testtimeminspnbox.configure(state="normal")
390.
     self.testtimeminspnbox.delete(0,"end")
391.
     self.testtimeminspnbox.insert(tk.INSERT,str(data["time"]["minute"]))
392.                         if "time_limit" in data:
393.                                 if data["time_limit"]["hour"] == "0"
   and data["time_limit"]["minute"] == "0":
394.                                         self.testtimelimitvar.set(0)
395.                                 else:
396.                                         self.testtimelimitvar.set(1)
397.
     self.testtimelimithrspnbox.configure(state="normal")
398.
     self.testtimelimithrspnbox.delete(0,"end")
399.
     self.testtimelimithrspnbox.insert(tk.INSERT,data["time_limit"]["hour"]
   )
400.
     self.testtimelimitminspnbox.configure(state="normal")
401.
     self.testtimelimitminspnbox.delete(0,"end")
402.
     self.testtimelimitminspnbox.insert(tk.INSERT,data["time_limit"]["minut
   e"])
403.                         if "response_limit" in data:
404.
     self.testlimitresponsespnbox.delete(0,"end")
405.
     self.testlimitresponsespnbox.insert(tk.INSERT,str(data["response_limit
   "]))
406.
407.         def AddSection(self,data=None):
```

```
408.                if self.sections is None:
409.                        self.sections = list()
410.                        if data is not None:
411.
    self.sections.append(Section(self.sectionframe,data,))
412.                        else:
413.
    self.sections.append(Section(self.sectionframe,None,len(self.sections)
  ))
414.                        self.sections[len(self.sections)-
  1].Grid(len(self.sections)-1,0)
415.
416.                        self.removesectionsbtn = list()
417.
    self.removesectionsbtn.append(tk.Button(self.sectionframe,text="Remove
  ",command=lambda i=len(self.sections): self.RemoveSection(i-1)))
418.
    self.removesectionsbtn[len(self.removesectionsbtn)-
  1].grid(row=len(self.sections)-1,column=1,sticky=tk.NW)
419.                else:
420.                        if data is not None:
421.
    self.sections.append(Section(self.sectionframe,data,))
422.                        else:
423.
    self.sections.append(Section(self.sectionframe,None,len(self.sections)
  ))
424.                        self.sections[len(self.sections)-
  1].Grid(len(self.sections)-1,0)
425.
426.
    self.removesectionsbtn.append(tk.Button(self.sectionframe,text="Remove
  ",command=lambda i=len(self.sections): self.RemoveSection(i-1)))
427.                        self.removesectionsbtn[len(self.sections)-
  1].grid(row=len(self.sections)-1,column=1,sticky=tk.NW)
428.
429.        def Grid(self,row=0,column=0):
430.                self.testnamelabel.grid(row=0,column=0)
431.                self.testnametext.grid(row=0,column=1)
432.                self.addsectionbtn.grid(row=0,column=0)
433.                self.testtimelimitlbl.grid(row=3,column=0)
434.                self.testtimelimithrlbl.grid(row=0,column=0)
435.                self.testtimelimithrspnbox.grid(row=0,column=1)
436.                self.testtimelimitminlbl.grid(row=0,column=2)
437.                self.testtimelimitminspnbox.grid(row=0,column=3)
438.                #self.testtimelimitseclbl.grid(row=0,column=4)
439.                #self.testtimelimitsecspnbox.grid(row=0,column=5)
440.                self.testtimelimitframe.grid(row=3,column=1)
441.                self.testtimelimitchkbtn.grid(row=3,column=2)
442.                self.testdatelbl.grid(row=1,column=0)
443.                self.testdate.grid(row=1,column=1)
444.                self.testdatechkbtn.grid(row=1,column=2)
445.                self.testtimelbl.grid(row=2,column=0)
446.                self.testtimehrlbl.grid(row=0,column=0)
447.                self.testtimehrspnbox.grid(row=0,column=1)
448.                self.testtimeminlbl.grid(row=0,column=2)
449.                self.testtimeminspnbox.grid(row=0,column=3)
```

```
450.                #self.testtimeseclbl.grid(row=0,column=4)
451.                #self.testtimesecspnbox.grid(row=0,column=5)
452.                self.testtimechkbtn.grid(row=2,column=2)
453.                self.testlimitresponselbl.grid(row=4,column=0)
454.                self.testlimitresponsespnbox.grid(row=4,column=1)
455.                self.testtimeframe.grid(row=2,column=1)
456.                self.testbtnframe.grid(row=0,column=2)
457.                self.testframe.grid(row = 0,column = 0,sticky=tk.NW)
458.                self.sectionframe.grid(row = 1,column = 0,padx = 70)
459.                self.frame.pack(fill="both",expand=True)
460.
461.        def RemoveSection(self,index=None):
462.                if index is not None:
463.                        self.sections[index].Destroy()
464.                        self.removesectionsbtn[index].destroy()
465.                        del self.sections[index]
466.                        del self.removesectionsbtn[index]
467.                        for i in range(0,len(self.sections)):
468.                                self.sections[i].SetId(i)
469.
    self.removesectionsbtn[i].config(command=lambda:self.RemoveSection(i))
470.
471.        def Destroy(self):
472.                #self.wrapperframe.destroy()
473.                self.frame.destroy()
474.
475.        def GetID(self):
476.                return self._id
477.
478.        def ToggleTimeLimit(self):
479.                if self.testtimelimitvar.get() == 0:
480.
    self.testtimelimithrspnbox.configure(state="disabled")
481.
    self.testtimelimitminspnbox.configure(state="disabled")
482.
    #self.testtimelimitsecspnbox.configure(state="disabled")
483.                else:
484.
    self.testtimelimithrspnbox.configure(state="normal")
485.
    self.testtimelimitminspnbox.configure(state="normal")
486.
    #self.testtimelimitsecspnbox.configure(state="normal")
487.
488.        def ToggleTime(self):
489.                if self.testtimevar.get() == 0:
490.
    self.testtimehrspnbox.configure(state="disabled")
491.
    self.testtimeminspnbox.configure(state="disabled")
492.
    #self.testtimesecspnbox.configure(state="disabled")
493.                else:
494.                        self.testtimehrspnbox.configure(state="normal")
495.
    self.testtimeminspnbox.configure(state="normal")
```

```
496.
    #self.testtimesecspnbox.configure(state="normal")
497.
498.        def ToggleDate(self):
499.                if self.testdatevar.get() == 0:
500.                        self.testdate.configure(state="disabled")
501.                else:
502.                        self.testdate.configure(state="enabled")
503.
504.        def Export(self,savedialog=None):
505.                if self.path and  self._id and self.name is not None:
506.                        data = {}
507.                        data["id"] =self._id
508.                        data["test"] =
   self.testnametext.get("1.0","end").rstrip("\n")
509.                        data["response_limit"] =
   int(self.testlimitresponsespnbox.get())
510.                        if self.testdatevar.get() == 0:
511.                                data["date"] =
   {"day":0,"month":0,"year":0,}
512.                        else:
513.                                date = str(self.testdate.get_date())
514.                                date = date.split("-")
515.                                data["date"] =
   {"day":int(date[2]),"month":int(date[1]),"year":int(date[0]),}
516.                        if self.testtimevar.get() == 0:
517.                                data["time"] = {"hour":0 ,"minute":
   0,"second":0,}
518.                        else:
519.                                data["time"] =
   {"hour":int(self.testtimehrspnbox.get()) ,"minute":
   int(self.testtimeminspnbox.get()),"second":0,}
520.                        if self.testtimelimitvar.get() == 0:
521.                                data["time_limit"] = {"hour": "0",
   "minute": "0","second":"0",}
522.                        else:
523.                                data["time_limit"] = {"hour":
   self.testtimelimithrspnbox.get() , "minute":
   self.testtimelimitminspnbox.get(),"second":"0",}
524.                        data["sections"] = []
525.                        if self.sections is not None:
526.                                for i in self.sections:
527.
    data["sections"].append(i.Export())
528.                        I = Instruction(data)
529.                        data.update(I.Export())
530.                        with open(self.path,"w") as file:
531.                                json.dump(data,file)
532.
533.                else:
534.                        self.name = savedialog.name.split("/")
535.                        self.name = self.name[len(self.name)-1]
536.                        self.name = self.name.split(".")
537.                        self.name = self.name[0]
538.                        self.path = savedialog.name
539.                        conn = sqlite3.connect("Quiz.db")
540.                        cursor = conn.cursor()
```

```
541.                         cursor.execute("INSERT INTO quizzes (name,path)
   VALUES (?,?)",(self.name,self.path))
542.                         conn.commit()
543.                         cursor.execute("SELECT id FROM quizzes WHERE
   path=?",(self.path,))
544.                         self._id = cursor.fetchone()[0]
545.                         data = {}
546.                         data["id"] = self._id
547.                         data["test"] =
   self.testnametext.get("1.0","end")
548.                         data["response_limit"] =
   int(self.testlimitresponsespnbox.get())
549.                         if self.testdatevar.get() == 0:
550.                                 data["date"] =
   {"day":0,"month":0,"year":0,}
551.                         else:
552.                                 date = str(self.testdate.get_date())
553.                                 date = date.split("-")
554.                                 data["date"] =
   {"day":int(date[2]),"month":int(date[1]),"year":int(date[0]),}
555.                         if self.testtimevar.get() == 0:
556.                                 data["time"] = {"hour":0 ,"minute":
   0,"second":0,}
557.                         else:
558.                                 data["time"] =
   {"hour":int(self.testtimehrspnbox.get()) ,"minute":
   int(self.testtimeminspnbox.get()),"second":0,}
559.                         if self.testtimelimitvar.get() == 0:
560.                                 data["time_limit"] = {"hour": "0",
   "minute": "0","second":"0",}
561.                         else:
562.                                 data["time_limit"] = {"hour":
   self.testtimelimithrspnbox.get() , "minute":
   self.testtimelimitminspnbox.get(),"second":"0",}
563.                         data["sections"] = []
564.                         if self.sections is not None:
565.                                 for i in self.sections:
566.
     data["sections"].append(i.Export())
567.                         I = Instruction(data)
568.                         data.update(I.Export())
569.                         savedialog.write(json.dumps(data))
570.                 mb.showinfo('Saved', "File has been saved
   successfully!")
571.
572. class QuizEditor:
573.         def __init__(self,root,callback=None):
574.                 self.root = root
575.                 self.root.title("Quiz Editor")
576.                 self.test = None
577.                 self.openwindow = None
578.                 self.openwindowbtns = None
579.                 self.menubar = tk.Menu(root)
580.                 self.callback = callback
581.
582.                 self.filemenubar = tk.Menu(self.menubar,tearoff=0)
```

```
583.
    self.filemenubar.add_command(label="New",command=self.New,accelerator=
  "Ctrl+N")
584.                self.root.bind('<Control-n>',lambda e:self.New())
585.
    self.filemenubar.add_command(label="Open",command=self.Open,accelerato
  r="Ctrl+O")
586.                self.root.bind('<Control-o>',lambda e:self.Open())
587.
    self.filemenubar.add_command(label="Save",command=self.Save,accelerato
  r="Ctrl+S")
588.                self.root.bind('<Control-s>',lambda e:self.Save())
589.
    self.filemenubar.add_command(label="Close",command=self.Close,accelera
  tor="Ctrl+C")
590.                self.root.bind('<Control-q>',lambda e:self.Close())
591.
    self.filemenubar.add_command(label="Exit",command=self.Exit,accelerato
  r="Ctrl+Q")
592.                self.root.bind('<Control-q>',lambda e:self.Exit())
593.
594.
    self.menubar.add_cascade(label="File",menu=self.filemenubar)
595.
    self.menubar.add_command(label="About",command=self.About)
596.                self.root.configure(menu=self.menubar)
597.
598.                self.savedialog = None
599.                #self.root.bind("<Destroy>",self.Save)
600.
601.        def About(self):
602.                aboutwindow = tk.Toplevel(self.root)
603.                aboutwindow.title("About")
604.                aboutwindow.geometry("300x100+100+100")
605.                aboutwindow.resizable(height = False , width = False)
606.                abouttext = tk.Text(aboutwindow,font=("Verdana", 20))
607.                abouttext.insert(tk.INSERT,"Quiz Editor \nVersion 1.0")
608.                abouttext.configure(state='disabled')
609.                abouttext.pack()
610.                aboutwindow.grab_set()
611.
612.        def New(self,_id=None,name=None,path=None):
613.                if self.test is None:
614.                        if _id and name and path is not None:
615.                                file = open(path,"r")
616.                                try :
617.                                        data = json.loads(file.read())
618.                                except:
619.                                        return
620.                                self.test =
  Test(self.root,data,_id,name,path)
621.                                self.openwindow.destroy()
622.                                self.openwindow = None
623.                                file.close()
624.                        else:
625.                                self.test = Test(self.root,)
626.                        self.test.Grid()
```

```
627.                 else:
628.                         self.Save()
629.                         self.DestroyTest()
630.                         self.New()
631.
632.         def Open(self):
633.                 if self.test is None:
634.                         self.openwindow = tk.Toplevel(self.root)
635.                         self.openwindow.geometry("350x400")
636.                         self.openwindow.title("Open")
637.                         self.openwindow.grab_set()
638.
    self.openwindow.resizable(height=False,width=False)
639.                         self.scrollframe =
   ScrollableFrame(self.openwindow)
640.                         conn =sqlite3.connect("Quiz.db")
641.                         cursor = conn.cursor().execute("SELECT * FROM
   quizzes")
642.                         quizzes = cursor.fetchall()
643.                         tlabel =
   tk.Label(self.scrollframe.scrollable_frame,text="List of Quizzes to
   Edit!",font=SMALL_MEDIUM_FONT)
644.
    tlabel.grid(row=0,column=0,columnspan=2,sticky="ew")
645.                         if len(quizzes) == 0:
646.                                 lbl =
   tk.Label(self.scrollframe.scrollable_frame,text="No quizzes to
   edit",font=SMALL_FONT)
647.
    lbl.grid(row=1,column=0,columnspan=2,sticky="ew")
648.                         else:
649.                                 self.openwindowbtns = []
650.                                 for i in range(0,len(quizzes)):
651.
    self.openwindowbtns.append(tk.Button(self.scrollframe.scrollable_frame
   ,text=quizzes[i][1],relief=tk.FLAT,font=SMALL_FONT,command=lambda
   j=i:self.New(quizzes[j][0],quizzes[j][1],quizzes[j][2])))
652.
    self.openwindowbtns[i].grid(row=i+1,column=0,padx=10,sticky="ew")
653.                                         rmovequizbtn  =
   tk.Button(self.scrollframe.scrollable_frame,text="Remove",font=SMALL_FO
   NT,command=lambda j=i:self.RemoveQuiz(quizzes[j]))
654.
    rmovequizbtn.grid(row=i+1,column=1,sticky="ew")
655.                         self.scrollframe.pack(expand=True,fill="both")
656.                 else:
657.                         self.Save()
658.                         self.DestroyTest()
659.                         self.Open()
660.
661.         def RemoveQuiz(self,row=None):
662.                 if row is not None:
663.                         quizrmqmb = mb.askquestion("","Are you sure you
   want to remove the quiz?")
664.                         if quizrmqmb == "yes":
665.                                 if os.path.exists(row[2]):
```

```
666.                                            conn =
   sqlite3.connect("Quiz.db")
667.                                            cursor = conn.cursor()
668.                                            cursor.execute("DELETE FROM
   quizzes WHERE id=?",(row[0],))
669.                                            conn.commit()
670.                                            os.remove(row[2])
671.                                            self.openwindow.destroy()
672.                                            self.Open()
673.
674.         def Save(self):
675.                 if self.test is not None:
676.                         svfmb = mb.askquestion("Save","Do you want to
   save file?")
677.                         if svfmb == "yes":
678.                                 if self.test.GetID() is None:
679.                                         self.SaveDialog()
680.                                 else:
681.                                         self.test.Export()
682.
683.         def Close(self):
684.                 if self.test is not None:
685.                         self.Save()
686.                         self.DestroyTest()
687.
688.         def Exit(self):
689.                 if self.test is not None:
690.                         self.Save()
691.                         self.menubar.destroy()
692.                         self.DestroyTest()
693.                 else:
694.                         self.menubar.destroy()
695.                 if __name__ == "__main__":
696.                         self.root.destroy()
697.                 else:
698.                         self.root.title("Quiz App")
699.                         if self.callback is not None:
700.                                 self.callback()
701.
702.         def SaveDialog(self):
703.                 self.savedialog = fd.asksaveasfile(filetypes =
   [("JSON","*.json")], defaultextension = [("JSON","*.json")])
704.                 if self.savedialog is not None:
705.                         self.test.Export(self.savedialog)
706.                         self.savedialog.close()
707.
708.         def DestroyTest(self):
709.                 self.test.Destroy()
710.                 del self.test
711.                 self.test = None
712.
713. if __name__ == "__main__" :
714.         window = tk.Tk()
715.         window.geometry("1024x768")
716.         window.title("Quiz Editor App")
717.         Q =  QuizEditor(window)
```

```
718.         window.mainloop()
```

Quiz database

Quiz.db

users table

```
1. CREATE TABLE "users" (
2.   "id"    INTEGER NOT NULL UNIQUE,
3.   "username"      TEXT NOT NULL,
4.   "password"      TEXT NOT NULL,
5.   "privilege"     INTEGER NOT NULL,
6.   PRIMARY KEY("id" AUTOINCREMENT)
7. )
```

responses table

```
1. CREATE TABLE "responses" (
2.   "user"  INTEGER NOT NULL,
3.   "quiz"  INTEGER NOT NULL,
4.   "response"      TEXT NOT NULL UNIQUE,
5.   "response_count"        INTEGER DEFAULT 1,
6.   PRIMARY KEY("user","quiz")
7. )
```

quizzes table

```
1. CREATE TABLE "quizzes" (
2.   "id"    INTEGER NOT NULL UNIQUE,
3.   "name"  TEXT NOT NULL,
4.   "path"  TEXT NOT NULL UNIQUE,
5.   PRIMARY KEY("id" AUTOINCREMENT)
6. )
```