

Criterion C: Development

To develop this project, I used the following software:

- Python 3.6.5 programming language
- Python IDLE editor

This standalone application aims to create a solution that allows digital test taking and creation to help make the process more efficient through computer automation of simple tasks. This helps remove the human bottleneck which significantly slowed down the process before.

Below is a table of the complexities used in the building of this project:

Advanced Complexities	Complexities
1. Database management	1. Data Validation
2. Advanced Graphics Libraries	2. File operations (I/O)
3. Encapsulation	3. Object Oriented Programming
4. Polymorphism	4. Modular programming
5. Arrays of 2 dimensions	5. Error/Exception Handling
	6. Data searching

Database Management(backend):

Database management is the core tool of this project. One of the main requirements from my client for this project was that it should be lightweight in nature, i.e. not consume large amounts of RAM or CPU power. Initially, I was going to go with SQLite; however, it is implemented in C++. This could lead to error on the clients system depending on which compiler is installed on their various computers (Clang llvm, Turbo C++, G++, Cygwin). Therefore, after some consultation with my client, we settled on developing a proprietary database architecture based on CSV (comma separated values) using Python 3.X.X. This is an extremely lightweight solution as it is essentially a plaintext file with the various data entries separated by commas.

```
import csv;
import os;

univFilePath = "/Users/Urvish/Desktop/IB/Computers/IA/project_code/"# A universal Filepath to the location of program's FileSystem

def questionbank_add(Q,A,B,C,D,CAns,CurrBank):#Function to add questions to a QuestionBank DataBase
    with open("%sFileSystem/SuperV/Test_sets/tBank/%s"%(univFilePath,CurrBank), "a+") as file_handle:#FileHandling
        reader = csv.reader(file_handle, delimiter=',');#Initiation of reader handle
        writer = csv.writer(file_handle, delimiter=',');#Initiation of writer handle

        file_handle.seek(0) #Resets the cursor to the top of the file
        sl_no = 1          #Slot Number Counter to number the levels of data

        for x in reader:#Iteration loop through reader handle
            sl_no+=1      #Counts the number of rows present in the database
            writer.writerow([sl_no,Q,A,B,C,D,CAns]);

def questionbank_preview(bank):#returns the entire contents of a QuestionBank database
    with open("%sFileSystem/SuperV/Test_sets/tBank/%s"%(univFilePath,bank), "a+") as file_handle:
        reader = csv.reader(file_handle, delimiter=',');#Initiation of reader handle
        writer = csv.writer(file_handle, delimiter=',');#Initiation of writer handle
        preview = [] #Array that stores the entire database for live use in the program (eg: previewing database contents)

        file_handle.seek(0);#Resets the cursor to the top of the file

        for row in reader:      #Iteration loop through reader handle
            preview.append(row) #Appends 1 row of database at a time (iteratively) into elements (2D array of elements of rows)

        return preview          #Returns the entire database to the class that calls it
        #print(row) DEBUGGER
```

Image 1: Commented Code of Structural Csv Database

The above image is a picture from the generalFileFuntions.py file, which mainly deals with insertion/removal of questions from a question database and creation/removal of a question database. This image details:

- questionbank_add(Q,A,B,C,D,Cans,CurrBank) → A function that uses csv file reader operations to append a new question to bottom of the database. The arguments include the question, options(A,B,C,D), the correct answer (Cans) and the question bank (CurrBank) to which the question needs to be added. The reason for this function rather than dedicated operations in the main file is that it can be reused across all the code files and provides flexibility to incorporate other operations.
- questionbank_preview(bank) → A function that returns the entire database of a question bank as a 2-dimensional array to the function that calls it. It uses Csv File operations to handle file reads.
- Top down design for reusability and maintainability

```

def questionbank_remove(qNums,CurrBank):#Function to remove questions from a QuestionBank DataBase
    row_mem = [] #Holds entire questionbank database for live use by program
    try:
        with open("%sFileSystem/SuperV/Test_sets/tBank/%s"%(univFilePath,CurrBank), "r+") as file_handle:
            reader = csv.reader(file_handle, delimiter=',');#Initiation of reader handle
            writer = csv.writer(file_handle, delimiter=',');#Initiation of writer handle

            file_handle.seek(0) #Resets the cursor to the top of the file
            x_count = 0; #Counter that checks for serial number of question to be removed

            for row in reader: #Iteration loop through reader handle
                if(row[0]!=qNums): #Checks for serial number match
                    row_mem.append(row) #If doesn't match, then append to live array (temp database)
                    x_count += 1

            if x_count == 0:
                return 'QuestionBank Empty' #Empty database handling statement

            #print(row_mem) debugging purposes
            file_handle.close()
        with open("%sFileSystem/SuperV/Test_sets/tBank/%s"%(univFilePath,CurrBank), "w+") as file_handle:
            reader = csv.reader(file_handle, delimiter=',');#Initiation of reader handle
            writer = csv.writer(file_handle, delimiter=',');#Initiation of writer handle

            x_count = 1

            for row_append in row_mem: #Iteration loop through temporary database array
                row_append[0] = x_count #Serialisation
                writer.writerow(row_append) #Filewriter
                #print(row_append[0]) #debugging purposes
                x_count += 1

    except Exception:
        print("Unable to handle file")#Error handling erratic behaviour

```

Image 2: Commented Code of Structural Csv Database

The above image is the questionbank_remove() function from generalFileFunctions.py. It deals with the removal of a question from a question database. Complexity details:

- Csv File handling for lightweight operations. File is completely overwritten each time as there is no other way to delete a row in the middle of a csv database.
- row_mem [] is a 2D array used to read database into memory for overwriting.
- Modular programming helps method reusability across codebase and enables maintainability.
- Iterator loops (for x in reader) → Iterates through iterable python datastructures

```

def setBank(setName, confirmation=None):#Polymorphised function to create/remove test sets
#TestSet add new testBank polymorphised
if confirmation is None:#Functional Polymorphism in python
    setName = setName + ".csv"
    flag_duplicate = False #Duplicate error checking
    fileList = [] #holds directory content list
    fileList = os.listdir("%sFileSystem/SuperV/Test_sets/tBank"%(univFilePath)) #Searches directory contents
    for x in fileList:
        if x == setName:
            flag_duplicate = True #Flags duplicate
    if flag_duplicate == False: #Flag condition checking
        try:
            open("%sFileSystem/SuperV/Test_sets/tBank/%s" %(univFilePath,setName), "w+")
        except:
            print("unexpected error") #Unexpected error handling
    elif confirmation is not None:
        print("duplicate exists") #Duplicate error handling
else:
    #TestSet remove testBank polymorphised
    setName = setName + ".csv"
    try:
        os.remove("%sFileSystem/SuperV/Test_sets/tBank/%s" %(univFilePath,setName)) #Uses os library to remove test Database
    except:
        print("file does not exist")#Unexpected error handling

```

Image 3: Commented Code of Structural Csv Database

The above is setBank function of generalFileFunctions.py. It is a method that creates and removes question banks from FileSystem. Complexities :

- Polymorphism → polymorphism implemented as much of the code for both functions is largely the same. Python supports overloading in the form of functional polymorphism, therefore there are tests for null entry of data(parameter) for confirmation argument
- Modular programming → Code use flexibility
- Error handling to prevent unexpected behaviour
- Boolean flag conditions for flow control
- Iterator loops (for x in fileList) → Iterates through iterable python datastructures

```

def add_User(UID, passkey, passkey2,mailID):#Admin Function to add new user
    try:
        with open("%sFileSystem/SuperV/userBase.csv"%(univFilePath), "a+") as file_handle:#File handling
            reader = csv.reader(file_handle, delimiter=',');#File reader handling
            writer = csv.writer(file_handle, delimiter=',');#file writer handling
            file_handle.seek(0) #Reset cursor to database top

            for x in reader: #Iteration of reader handle
                if x[0] == UID: #Duplicate error handling
                    return "User already exists"
            if passkey != passkey2:
                return "passwords do not match" #Password match check

            file_handle.seek(2) #Cursor seek to database bottom

            m = hashlib.sha256()
            m.update(passkey.encode('utf8'))
            hash = m.hexdigest()

            writer.writerow([UID,hash,mailID]) #Write new user
            os.mkdir("%sFileSystem/SuperV/User/%s"%(univFilePath,UID)) #Make user file directory
            file_handle.close() #Memory Leak prevention
    except Exception:
        return traceback.print_exc()#Better error messages

def check_User(UID, passkey=None):#Functional Polymorphism
    if(passkey is not None):
        try:
            with open("%sFileSystem/SuperV/userBase.csv"%(univFilePath), "r") as file_handle:
                reader = csv.reader(file_handle, delimiter=',');#File writer handling
                file_handle.seek(0)#Reset cursor to database top
                #print(UID,passkey) DEBUGGER Code

                m = hashlib.sha256()
                m.update(passkey.encode('utf8'))
                hash = m.hexdigest()

                for x in reader:
                    #print(x[0],x[1]) DEBUGGER Code
                    if x[0] == UID and x[1] == hash: #Verifies user credentials from database
                        return True
                return False
        except:
            return "Error"

```

Image 4: Commented Code of Structural Csv Database

The above code is from adminFunctions.py. It contains various administrative functions that manipulate the userBase database in FileSystem such as add_User to add a new user and check_User to validate a user's login credentials or check if a user exists. Complexities:

- Polymorphism → check_User uses a polymorphised function to imitate function overloading. Overloading is used as much of the code for the 2 functions is similar. Polymorphism makes such code easier to read and maintain.
- Csv File I/O → Access database for verification
- Modular programming promotes reusability of code
- Exception handling → Prevents unexpected behavior
- Data searching → Scraping database to find credential matches
- Authentication → check_User authenticates login by data searching userBase database
- Security → Store passwords as their respective SHA256 hashes to prevent plaintext passwords from being leaked if database is accessed by unauthorized user.

```

def rm_User(UID, passkey):#Remove User credentials from database
try:
    with open("%sFileSystem/SuperV/userBase.csv"%(univFilePath), "r+") as file_handle:#File handling
        reader = csv.reader(file_handle, delimiter=',');#File reader handling
        writer = csv.writer(file_handle, delimiter=',');#File writer handling

        row_mem = []#Array to temp store entire userBase database
        x_count = 0
        y_count = 0

        for row in reader:
            #Iteration loop
            if row[0] != UID or row[1] != passkey: #User delete confirmation credential validation
                row_mem.append(row)
                x_count += 1
                y_count += 1

        if x_count == 0:
            return "User database empty"#Error handling for empty database
            file_handle.close() #Memory leak prevention
        with open("%sFileSystem/SuperV/userBase.csv"%(univFilePath), "w+") as file_handle:#File handling
            writer = csv.writer(file_handle, delimiter=',');#File writer handling
            write_count = 0 #Counter variable

            for row in row_mem: #Iteration loop to overwrite existing file without removed user
                writer.writerow(row)
                write_count += 1
            file_handle.close() #Memory leak prevention

        if write_count == y_count - 1:
            return "Task successful"#operation confirmation
        else:
            return "Task failed: Check UserID or Password" #Returns if UID password correct
except:
    return "Unable to remove user" #Exception handling unexpected behaviour

```

Image 5: Commented Code of Structural Csv Database

The above code is from adminFunctions.py. It contains the rm_User function that removes a registered user from the userBase database. Complexities:

- Csv File I/O → Access Csv database for read/write file operations
- Exception Handling → To prevent unexpected behavior from causing crashes
- Modular programming → Facilitates code reusability across the codebase
- Authentication → Authenticate user removal request by checking credentials

```

def setBankPublish(setName):#TestBank Subset publish to ready to use
    flag_duplicate = False #Boolean flag variable
    fileList = [] #List of directory contents
    fileList = os.listdir("%sFileSystem/SuperV/Test_sets/tPublish"%(univFilePath))#Searched directory for content
    for x in fileList:
        if x == setName:
            #Duplicate error handling
            flag_duplicate = True #Flag handling
    if flag_duplicate == False:
        try:
            open("%sFileSystem/SuperV/Test_sets/tPublish/%s"%(univFilePath,setName), "w+")#Exception Handling file open
        except:
            print("unexpected error") #Unexpected behaviour handling
    else:
        print("duplicate exists") #Duplicate error handling

def questionbank_add_Publish(Q,A,B,C,D,CurrBank,Reference_No):#Add questions to final publishable subset TestBank
    with open("%sFileSystem/SuperV/Test_sets/tPublish/%s"%(univFilePath,CurrBank), "a+") as file_handle:#File handling
        reader = csv.reader(file_handle, delimiter=',');#File reader handle
        writer = csv.writer(file_handle, delimiter=',');#File writer handle

        file_handle.seek(0) #Returns cursor to file top
        sl_no = 1 #Slot number counter

        for x in reader: #Iteration through reader handle
            sl_no+=1
        writer.writerow([sl_no,Q,A,B,C,D,Reference_No]);#Csv writerow File I/O

def questionbank_preview_publish(bank):#Flexible bank preview that holds question for test taking
    with open("%sFileSystem/SuperV/Test_sets/tPublish/%s"%(univFilePath,bank), "a+") as file_handle:
        reader = csv.reader(file_handle, delimiter=',');#File reader handle
        writer = csv.writer(file_handle, delimiter=',');#File writer handle
        preview = [] #Array that temporarily holds final publishing test live
        file_handle.seek(0);#Returns cursor to file top

        for row in reader:
            #Iteration through reader handle
            preview.append(row) #Adding data to temp live array database

        return preview#Returns value to calling function

def questionbank_add_Answer(RealRef,Q,A,B,C,D,Response,CurrUser,CurrBank):#Records student response to publish qbase subset
    with open("%sFileSystem/SuperV/User/%s/%s"%(univFilePath,CurrUser,CurrBank), "a+") as file_handle:#File handling
        reader = csv.reader(file_handle, delimiter=',');#File reader handle
        writer = csv.writer(file_handle, delimiter=',');#File writer handle
        |
        file_handle.seek(0) #Returns cursor to file top
        sl_no = 1 #Slot number counter

        for x in reader:#Iteration through reader handle
            sl_no+=1 #Counts slot number
        writer.writerow([sl_no,RealRef,Q,A,B,C,D,Response]);#Writes row that records student response + actual question

```

Image 6: Commented Code of Structural Csv Database

The above code is from publishTest.py. It deals with test publishing from questionbank dataset as published tests are subsets of questionbank datasets. Complexities:

- Csv File I/O → Access csv database files for read/write operations for lightweight database management
- Exception handling → To prevent unexpected behaviors from disrupting program
- Modular programming to allow for wide reusability
- Data searching → Search directories for duplicate tests

Graphics (UI/UX):

The UI/UX of this test interface was designed for simplicity and minimalism to keep a clean and intuitive interface that is self-explanatory. This project makes use of Python's Tkinter graphics library. Tkinter is a native and lightweight graphics library, therefore it doesn't require additional library installations which helps maintain compatibility.

Login Screen: Class TestGUI_Interface_loginScr

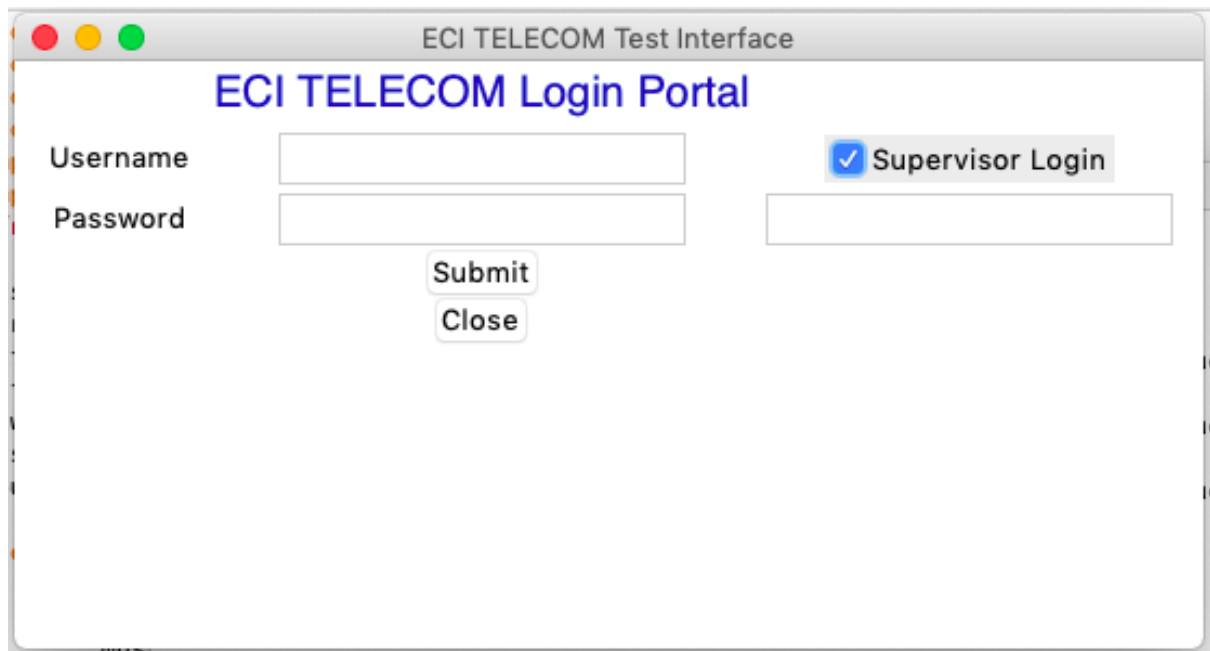


Image 7: Login Interface UI


```

def submit(self):#pack this in the controller class
#return to database
usernameArgs = self.UsernameBox.get()      #gets username data
passwordArgs = self.PasswrBox.get()        #gets password data
if (check_User(usernameArgs,passwordArgs)): #Authentication
    self.unlockmsg1.grid(row=5,column=2)    #Unlock success for employee
    #print(self.checkMaster.state()[0])DEBUGGER
    if(self.MasterBoxE.get() == masterKey and self.checkMaster.state()[0]=='selected'):#Supervisor unlock
        global CurrUser#Global current user store
        CurrUser = usernameArgs            #Alias
        root = Tk()                         #New root process spawn
        GUI1 = TestGUI_Interface_SuperInterface(root) #Object creation
        root.mainloop()                    #New window spin-off
    else:
        CurrUser = usernameArgs
        root = Tk()                         #New root process
        GUI1 = TestGUI_Interface_SubInterface_ViewTests(root)#object creation
        root.mainloop()                    #New window spin-off
else:
    self.unlockmsg2.grid(row=5,column=2)#Unlock error msg

```

Image 8: Login Interface Code(Interface.py)

Complexities:

- Encapsulation→Data and methods wrapped into a class unit to make spawning new processes easy
- Object oriented programming → Creation of objects to aid in design and planning
- Use of tkinter graphics library
- submit()→ gets username and password data from tkinter widgets and passes through check_User database function to authenticate login access

Test Availability Screen: Class TestGUI_Interface_SubInterface_ViewTests

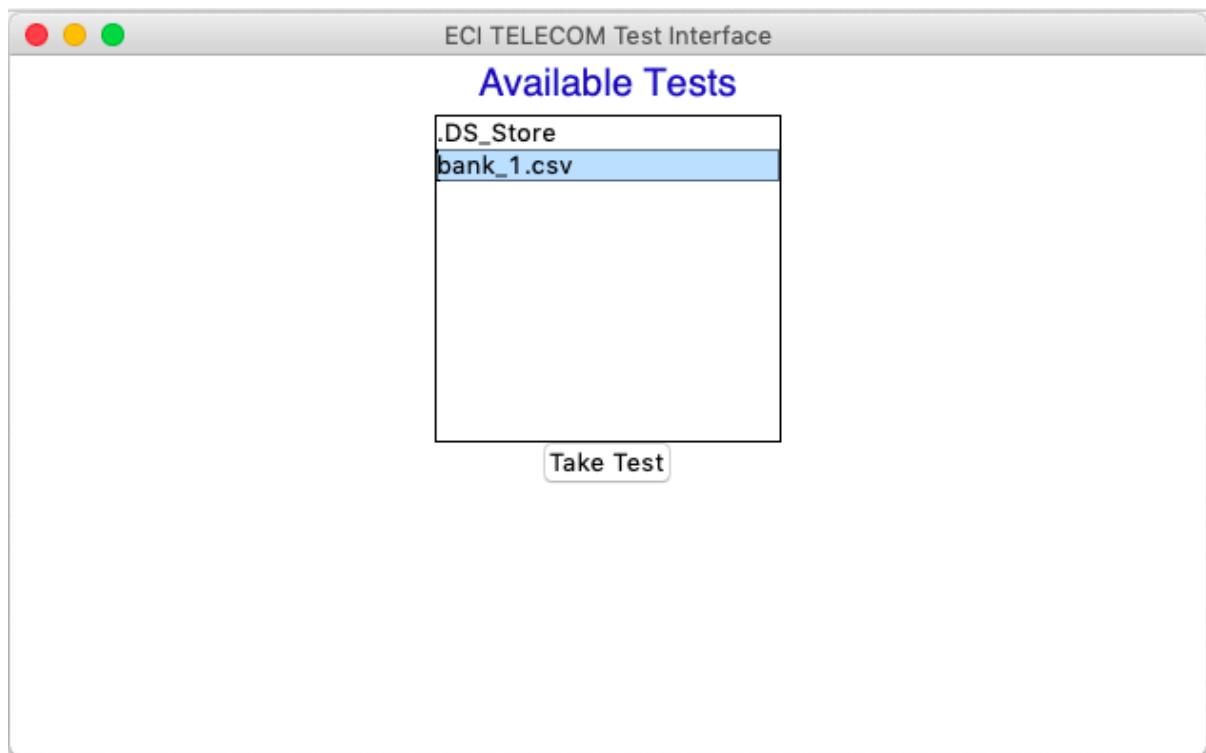


Image 9: Test Availability UI

```
def TestForward(self):
    SessionBankNameVariable = self.listbox1.get(ACTIVE)#Which bank
    #print(SessionBankNameVariable)
    global qList
    qList = questionbank_preview_publish(SessionBankNameVariable)#stores session preview
    #print(qList)
    global count_terminate
    for f in qList:
        count_terminate+=1

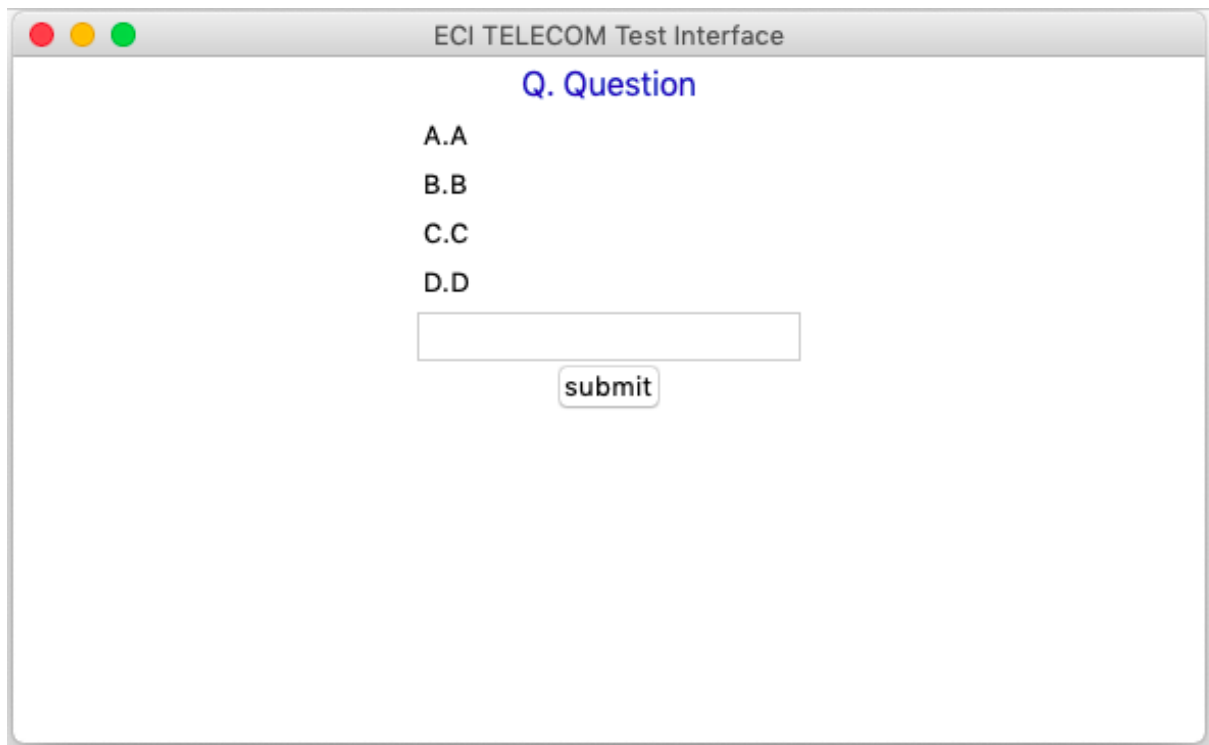
    root = Tk()      #New root
    GUI1 = TestGUI_Interface_SubInterface_TestTake(root, SessionBankNameVariable)#Object creation
    root.mainloop() #New window
```

Image 10: Test Availability Code

Complexities:

- Encapsulation → Data and methods wrapped into a class unit to make spawning new processes easy
- Object oriented programming → Creation of objects to aid in design and planning
- Use of tkinter graphics library
- Referencing global and static class variables

Testing Interface Screen: Class TestGUI_Interface_SubInterface_TestTake



ECI TELECOM Test Interface

Q. Question

A.A

B.B

C.C

D.D

submit

Image 11: Test Interface UI

```

def Submit(self):
    answerSub = self.Ans.get()
    answer = ""
    global qlist,mainCount#Global reference
    print(qlist)
    if answerSub == 'A' or answerSub == 'a':
        answer = qlist[mainCount][2]
    if answerSub == 'B' or answerSub == 'b':
        answer = qlist[mainCount][3]
    if answerSub == 'C' or answerSub == 'c':
        answer = qlist[mainCount][4]
    if answerSub == 'D' or answerSub == 'd':
        answer = qlist[mainCount][5]

    questionbank_add_Answer(qlist[mainCount][6],qlist[mainCount][1],qlist[mainCount][2],qlist[mainCount][3],qlist[mainCount][4],qlist[mainCount][5],answer)
    mainCount+=1
    print(mainCount,count_terminate)
    if mainCount < count_terminate:
        root = Tk()#New root
        GUI1 = TestGUI_Interface_SubInterface_TestTake(root,testNameGlobe)#Object creation
        root.mainloop()#New window
    else:
        root = Tk()#New root
        GUI1 = TestGUI_Interface_SubInterface_TestComplete(root)#Object creation
        root.mainloop()#New window

```

Image 12: Test Interface Code

- submit() → gets user input in tkinter entry widget and converts it to uniform standard parsable by database. Then, questionbank_add_Answer() called to add response to test take log.
- Object oriented programming → used to aid design and planning. Allows multiple instances of question screen to be open at the same time.

System Functions Redirect Screen: Class TestGUI_Interface_SuperInterface



Image 13: System Functions Interface UI

```
def nextWindow(self):
    root = Tk()
    GUI1 = TestGUI_Interface_SuperInterface_QuestionManager(root)
    root.mainloop()

    widgetF.pack()
def addRemove_Submit(self):
    root = Tk()
    GUI1 = TestGUI_Interface_SuperInterface_AdF(root)
    root.mainloop()

def Publish(self):
    root = Tk()
    GUI1 = TestGUI_Interface_SuperInterface_PublishTest(root)
    root.mainloop()

def Employee(self):
    root = Tk()
    GUI1 = TestGUI_Interface_SuperInterface_Overview(root)
    root.mainloop()
```

Image 14: System Function Interface Code

- Encapsulation → Data and methods wrapped into a class unit to make spawning new processes easy
- Object oriented programming → Creation of objects to aid in design and planning

Admin Functions Screen: Class TestGUI_Interface_SuperInterface_AdF

The screenshot shows a window titled "ECI TELECOM Test Interface". Inside the window, the text "Administrative Functions" is displayed in blue. Below this, there are two sections for user management. The "Add User:" section has four input fields labeled "Username", "Password", "Password", and "E-mail", with a "Submit" button below them. The "Remove User:" section has two input fields and a "Submit" button below them.

	Username	Password	Password	E-mail
Add User:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="button" value="Submit"/>			
Remove User:	<input type="text"/>	<input type="text"/>		
	<input type="button" value="Submit"/>			

Image 15: Admin Functions Interface UI

```

def submit(self):
    #print("lol")
    usernameArgs = self.UsernameBox.get()#checks username to prevent duplicates
    passwordArgs = self.PasswrBox.get()
    passwordArgs1 = self.PasswrBox01.get()
    EmailArgs = self.EmailBox.get()
    #print(usernameArgs)

    if (check_User(usernameArgs) != True and passwordArgs != "" and passwordArgs == passwordArgs1):
        self.unlockmsg2.grid_forget()
        self.unlockmsg1.grid(row=6,column=2)
        add_User(usernameArgs, passwordArgs, passwordArgs1, EmailArgs)
        return True

    else:
        self.unlockmsg2.grid(row=6,column=2)
def submit2(self):
    usernameArgs = self.UsernameBox1.get()#checks username to prevent duplicates
    passwordArgs = self.PasswrBox1.get()
    #print(usernameArgs, passwordArgs)
    if (check_User(usernameArgs,passwordArgs)):
        #print(check_User(usernameArgs,passwordArgs))
        self.unlockmsg3.grid_forget()
        self.unlockmsg1.grid(row=6,column=2)
        rm_User(usernameArgs, passwordArgs)
        return True
    else:
        self.unlockmsg3.grid(row=6,column=2)

```

Image 16: Admin Functions Interface Code

- submit() → pulls data from tkinter graphics widgets and places it as arguments in add_User() database function
- submit2() → pulls data from tkinter graphics widgets and places it as arguments in rm_User() database function to remove users
- Use of logical blocks to handle abnormal data input and display pertinent errors

QuestionBank options: Class TestGUI_Interface_SuperInterface_QuestionManager

The screenshot displays a software window titled "ECI TELECOM Test Interface". The interface is divided into two main panels: "TestBank" on the left and "Test Questions" on the right. The "TestBank" panel contains a text area with the text ".DS_Store" and "bank_1.csv", and a "submit" button below it. The "Test Questions" panel is currently empty. To the right of the "Test Questions" panel is a "remove question" button. Below these panels is an "Add question" section with five input fields labeled "Question:", "Option A:", "Option B:", "Option C:", and "Option D:", followed by an "Answer:" field. An "add question" button is positioned to the right of these fields. At the bottom, there is a section for "TestBank" and "Add or Remove TestBank?" with two input fields and a "submit" button.

ECI TELECOM Test Interface

TestBank

.DS_Store
bank_1.csv

Test Questions

remove question

submit

Add question

Question:

Option A:

Option B:

Option C:

Option D:

Answer:

add question

TestBank **Add or Remove TestBank?**

submit

Image 17: QuestionBank Functions Interface UI


```

def submitTBankSelection_Function(self):#Modular function allows reuse of button
    TBank_choice = self.listbox1.get(ACTIVE)
    TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVariable = TBank_choice#Because variable is not acces
    #print(TBank_choice)
    #print(TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVariable)
    count = 1;
    for f in questionbank_preview(TBank_choice):
        self.listbox2.insert(count, f)
        count +=1

def removeQuestion_Function(self):#Modular function allows reuse of button
    #print(TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVariable)
    removeQuestion_selection = self.listbox2.get(ACTIVE)
    questionbank_remove(removeQuestion_selection[0], TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVari
    #print(removeQuestion_selection)

def submitAddQuestion(self):#Modular function allows reuse of button
    questionbank_add(self.Box4e.get(),self.Box5e.get(),self.Box6e.get(),self.Box7e.get(),self.Box8e.get(),self.Box9e.get(

def polyTestSet(self):
    BankName = self.Box10e.get()
    BankAction = self.Box11e.get()
    #print(BankName, BankAction)DEBUGGER
    if BankAction == 'add' or BankAction == 'Add':
        setBank(BankName)
        self.PolySuccess.grid(row=3,column=3)
    elif BankAction == 'remove' or BankAction == 'Remove':
        setBank(BankName, BankAction)
        self.PolySuccess2.grid(row=4,column=3)
    else:
        self.PolySuccess3.grid(row=5,column=3)

```

Image 18: Questionbank Functions Interface Code

- submitTBankSelection_function() → records active questionbank selection in listbox and iterates through it to display questions in the next listbox.
- removeQuestion_Function → removes question from choice of questionbank using questionbank_remove() database method
- submitAddQuestion() → adds question to questionbank with questionbank_add() database method.
- polyTestSet → add/remove question database with polymorphic setBank() function

Publish Test Functions: Class TestGUI_Interface_SuperInterface_PublishTest

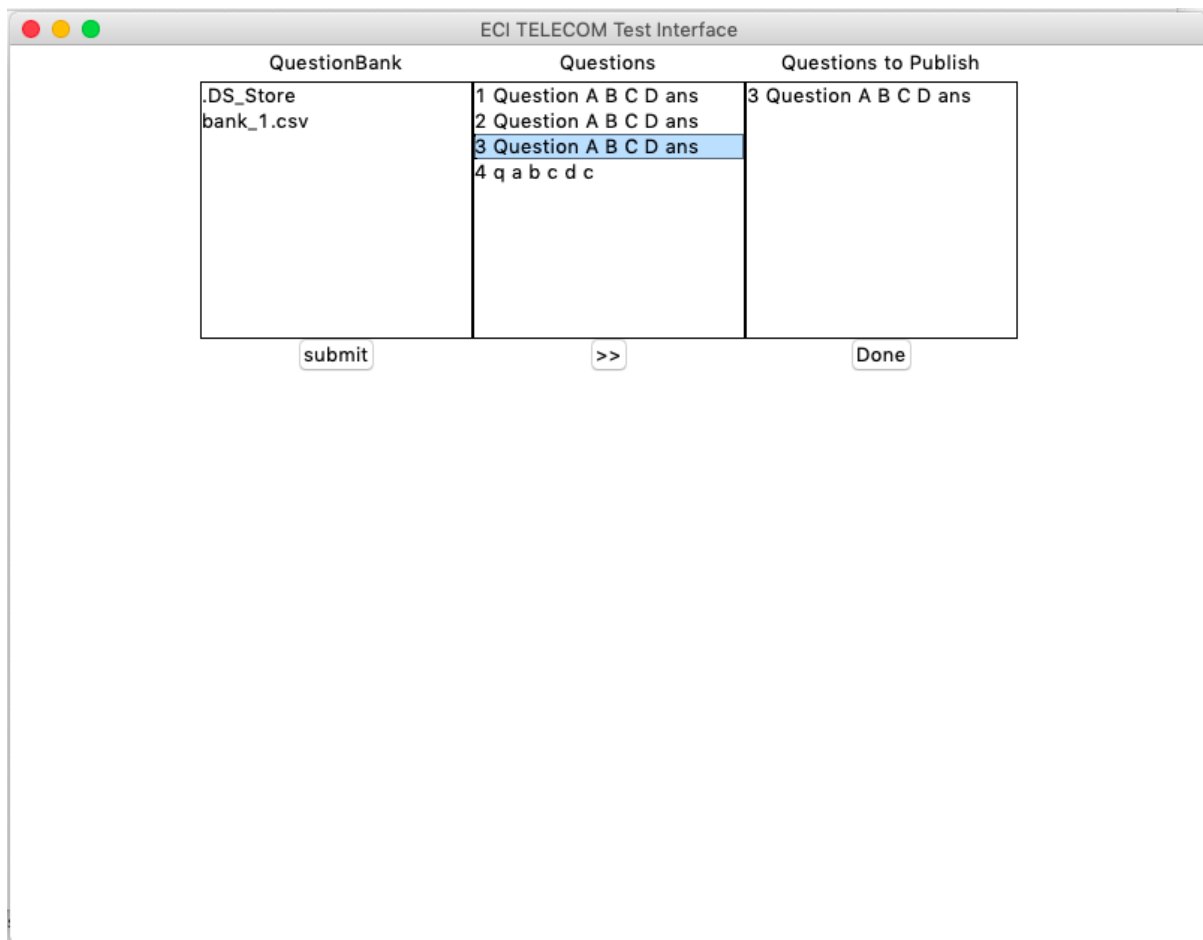


Image 19: PublishTest Functions Interface UI

```
def TBankSelect(self):#Select questionbank
    count2 =1
    TBank_choice = self.listbox1.get(ACTIVE)#Get active listbox selection
    TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVariable = TBank_choice#Because variable is not acces
    for f in questionbank_preview(TBank_choice):#Iteration loop through live preview array
        self.listbox2.insert(count2, f)#insert new listbox entries
        count2+=1
def TBankQuestionSelect(self):#Select questions
    TBank_choice = self.listbox2.get(ACTIVE)#Get active listbox selection
    self.listbox3.insert(TestGUI_Interface_SuperInterface_PublishTest.countGlobal, TBank_choice)#insert new listbox entry
    TestGUI_Interface_SuperInterface_PublishTest.countGlobal+=1#Global count variable increment
    TestGUI_Interface_SuperInterface_PublishTest.listGlobal.append(TBank_choice)#Global list append new entry
    #print(TestGUI_Interface_SuperInterface_PublishTest.listGlobal) DEBUGGER
def Done(self):#Publish Exevute
    setBankPublish(TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNameVariable)
    for f in TestGUI_Interface_SuperInterface_PublishTest.listGlobal:
        #print(f) DEBUGGER
        questionbank_add_Publish(f[1],f[2],f[3],f[4],f[5],TestGUI_Interface_SuperInterface_QuestionManager.SessionBankNam
```

Image 20: PublishTest Functions Interface Code

- TBankSelect()→records active questionbank selection in listbox and iterates through it to display questions in the next listbox.
- TBankQuestionSelect()→choice of question moved to static class array for temporary storage
- Done()→Publishes temporary array of questions for test taking use
- Encapsulation→wraps data and functions into a single unit for easy deployment

Employee Overview: Class TestGUI_Interface_SuperInterface_Overview

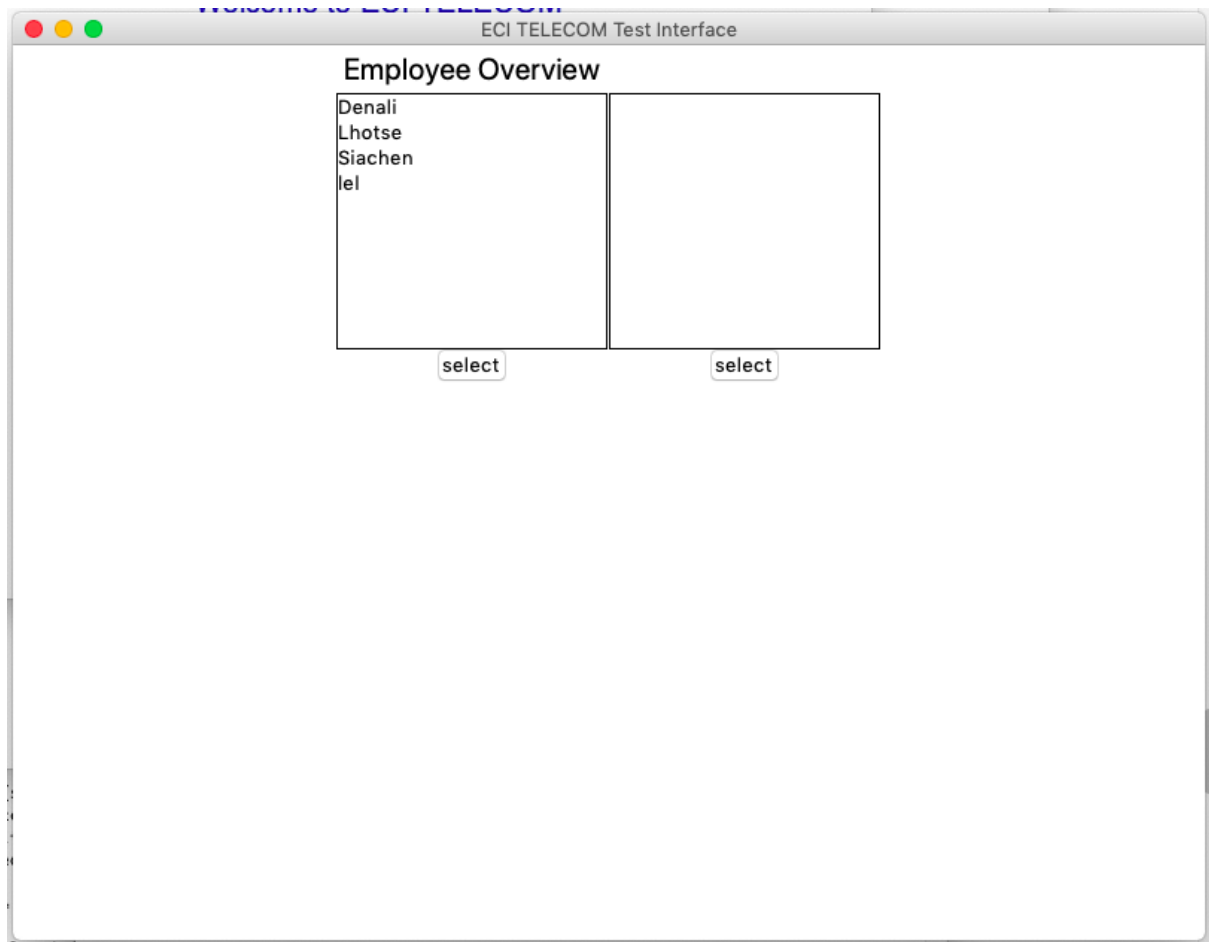


Image 21: Employee Overview Functions Interface UI

```

def select(self):
    TestGUI_Interface_SuperInterface_Overview.person = self.EmployeeList.get(ACTIVE)
    dirlist = os.listdir("%s/FileSystem/SuperV/User/%s"%(univFilePath,TestGUI_Interface_SuperInterface_Overview.person))
    count = 1
    for f in dirlist:
        self.TestList.insert(count,f)
        count+=1

def submit(self):
    #widgetG = Frame(master)
    choice = self.TestList.get(ACTIVE)
    TestGUI_Interface_SuperInterface_Overview.bank = choice
    data = open("%sFileSystem/SuperV/User/%s/%s"%(univFilePath,TestGUI_Interface_SuperInterface_Overview.person,choice))
    #print(data)
    testOut = []

    count = 1
    for i in data:
        newOut = []
        thing = [word.replace("\n", "") for word in i.split(",")]
        for j in range(1,8):
            newOut.append(thing[j])
        testOut.append(newOut)
    print(testOut)
    count+=1

root = Tk()
GUI1 = Out(root,testOut,TestGUI_Interface_SuperInterface_Overview.bank)
root.mainloop()

```

Image 22: PublishTest Functions Interface Code

- select() → uses static class variables to record choice of employee. Used to scan directory for taken test logs using system.os library for display in next listbox widget
- submit() → submit choice of taken test for overview of said test. Iterates through test output logs and publish test log to give a comparative view to check answers.
- Encapsulation → wraps data and functions into single unit for easy deployment
- Object-oriented programming → used to aid modular design of code for deployment