

**To-Do-Er**

### A Project Report Submitted

in Partial Fulfillment of the Requirements

**AISSCE - 2022**

### In

**COMPUTER SCIENCE (083)**

By:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No** | **Student Name** | **Class** | **ADMIT CARD NO.** |
| 1 | Hiten Dalmia | 12 |  |

**CERTIFICATE**

### Certified that the work contained in the project titled “**To-Do-Er**” by “**Hiten Dalmia”** has been carried out under my supervision as prescribed by CBSE **AISSCE – 2022**.

**Internal Examiner External Examiner**

**Date: Institution Stamp:**

**ACKNOWLEDGEMENT**

### I would like to express my gratitude to my Computer Teacher **Mr. Mallikarjun** for his guidance, support and encouragement through the project and the school lab assistant.

### I would further like to thank my parents and friends for helping me with the research required for this project.

**CONTENTS**

### Introduction of the Project

* System Requirements of the Project
* Python Coding
* Output Screens of the Project
* References

### **INTRODUCTION OF THE PROJECT**

The project is attempt to create a useful program all the while fulfilling the requirements of the activity. It is a simple to-do app with a GUI called To-Do-Er. It uses a local instance of MySQL/MariaDB to store that data. Multiple users can use the same app by creating different accounts. It uses two SQL tables; one stores that user data and the other stores the tasks in the to-do list corresponding to a user’s email.

### **SYSTEM REQUIREMENTS OF THE PROJECT**

**Recommended System Requirements**

Processors: Intel® Core™ i3 processor 4300M at 2.60 GHz. Disk space: 2 to 4 GB.

Operating systems: Windows® 10, MACOS, UBUNTU. Python Versions: 3.X.X or Higher.

**Minimum System Requirements**

Processors: Intel Atom® processor or Intel® Core™ i3 processor. Disk space: 1 GB.

Operating systems: Windows 7 or later, MACOS, and UBUNTU. Python Versions: 2.7.X, 3.6.X.

**Prerequisites before installing MySQL Connector Python**

You need root or administrator privileges to perform the installation process.

Python must be installed on your machine.

**Note:** MySQL Connector Python requires python to be in the system’s PATH. Installation fails if it doesn’t find Python.

On Windows, If Python doesn’t exist in the system’s PATH, please manually add the directory containing python.exe yourself.

### **PYTHON CODING**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351 | **import** **tkinter** **as** **tk**  **import** **mysql.connector**  **import** **tkinter.ttk** **as** **ttk**  mydb = mysql.connector.connect(host="localhost", user="root", password="")  cursor = mydb.cursor(buffered=True)  cursor.execute("CREATE DATABASE IF NOT EXISTS tododb")  cursor.execute("USE tododb")  *# Create tables.*  command = """CREATE TABLE IF NOT EXISTS todo\_users(  email VARCHAR(255) NOT NULL,  name VARCHAR(255) NOT NULL,  password VARCHAR(255) NOT NULL,  PRIMARY KEY (email)  )"""  cursor.execute(command)  command = """CREATE TABLE IF NOT EXISTS user\_items(  itemName VARCHAR(255) NOT NULL,  email VARCHAR(255) NOT NULL,  FOREIGN KEY (email) REFERENCES todo\_users(email)  )"""  cursor.execute(command)  mydb.commit()  **print**("Tables created.")  *# Initialize root window object.*  window = tk.Tk()  window.title("Todo List")  window.iconbitmap("logo.ico")  window.resizable(False, False)  *# Define Text Input class.*  **class** **TextInput**:  **def** \_\_init\_\_(self, frame, text, height=50, focus=False, padx=40, pady=10):  self.bg\_color = "#221f1f"  self.fg\_color = "#a19a9a"  input\_frame = tk.Frame(frame, width=400, height=height, bg=self.bg\_color)  self.input\_label = tk.Label(input\_frame, text=text, bg=self.bg\_color, fg=self.fg\_color)  self.input\_field = tk.Entry(input\_frame, width=40, bg=self.fg\_color, fg=self.bg\_color)  **if** focus:  self.input\_field.focus()  self.input\_label.pack(side="left")  self.input\_field.pack(side="right")  input\_frame.pack(pady=pady, padx=padx)  input\_frame.pack\_propagate(0)  **def** get\_input\_ref(self):  **return** self.input\_field  *# Define screens, buttons and event handlers.*  **class** **GuiWindow**:  **def** \_\_init\_\_(self, bg\_color, fg\_color, root):  self.bg\_color = bg\_color  self.fg\_color = fg\_color  self.root = root  self.items = []  self.list\_vars = {}  self.list\_buttons = {}  self.list\_buttons\_edit = {}  *# Methods involving MySQL.*  *# Fetch all records from user\_items table.*  **def** update\_items(self):  query = f"SELECT itemName FROM user\_items WHERE email=**\"**{self.email}**\"**"  cursor.execute(query)  self.items = [i[0] **for** i **in** cursor]  **print**("Logged in user's item list:", self.items)  *# Search for itemName by keyword in user\_items table.*  **def** search\_items(self):  search\_string = "%" + self.name\_field.get\_input\_ref().get() + "%"  query = f"SELECT \* FROM user\_items WHERE email=**\"**{self.email}**\"** AND itemName LIKE**\"**{search\_string}**\"**"  cursor.execute(query)  self.items = [i[0] **for** i **in** cursor]  **print**("Search string:", search\_string)  self.list\_frame.destroy()  self.item\_list\_window()  *# Delete item from user\_items table.*  **def** delete\_list\_item(self, label):  query = f"DELETE FROM user\_items WHERE email=**\"**{self.email}**\"** AND itemName=**\"**{label}**\"**"  cursor.execute(query)  mydb.commit()  **print**("Item name of the record to be deleted:", label)  self.update\_items()  self.list\_frame.destroy()  self.item\_list\_window()  *# Delete itemName in user\_items table.*  **def** edit\_item\_handler(self):  *# Get value from name input field.*  name = self.name\_field.get\_input\_ref().get()  query = f"UPDATE user\_items SET itemName=**\"**{name}**\"** WHERE email=**\"**{self.email}**\"** AND itemName=**\"**{self.edit\_name}**\"**"  cursor.execute(query)  mydb.commit()  **print**(f"Updated itemName from **\"**{self.edit\_name}**\"** to **\"**{name}**\"**.")  self.update\_items()  self.open\_window(self.change\_item\_frame, self.item\_list\_window)  *# Insert item in user\_items table.*  **def** add\_item\_handler(self):  *# Get value from name input field.*  item\_name = self.name\_field.get\_input\_ref().get()  query = f"INSERT INTO user\_items VALUES(**\"**{item\_name}**\"**, **\"**{self.email}**\"**)"  cursor.execute(query)  mydb.commit()  **print**(f"Inserted new record into user\_items: (**\"**{item\_name}**\"**, **\"**{self.email}**\"**)")  self.update\_items()  self.open\_window(self.change\_item\_frame, self.item\_list\_window)  *# Check if user record exists in todo\_users table.*  **def** log\_in\_handler(self):  *# Get value from user input fields.*  self.email = self.email\_field.get\_input\_ref().get()  password = self.password\_field.get\_input\_ref().get()  query = f"SELECT \* FROM todo\_users WHERE email=**\"**{self.email}**\"** AND password=**\"**{password}**\"**"  cursor.execute(query)  *# Return error if fields are empty.*  **if** **not** self.email **or** **not** password:  self.warning\_label["text"] = "Please fill all the fields."  **return**  **print**(f"Checking if user exists:**\n**Email: {self.email}**\n**Password: {password}")  *# Return error if user does not exist in todo\_users table.*  **if** **not** cursor.rowcount:  self.warning\_label["text"] = "Username or password invalid."  **print**("No record corresponding to the given email and password found.")  **return**  **print**("Record found.")  self.update\_items()  self.open\_window(self.form\_frame, self.item\_list\_window)  *# Insert new user record into todo\_users table.*  **def** register\_handler(self):  *# Get value from user input fields.*  self.email = self.email\_field.get\_input\_ref().get()  name = self.name\_field.get\_input\_ref().get()  password = self.password\_field.get\_input\_ref().get()  **if** **not** self.email **or** **not** password **or** **not** name:  self.warning\_label["text"] = "Please fill all the fields."  **return**  **try**:  query = f"INSERT INTO todo\_users VALUES(**\"**{self.email}**\"**, **\"**{name}**\"**, **\"**{password}**\"**)"  cursor.execute(query)  mydb.commit()  **print**(f"Inserted new record into todo\_users: (**\"**{self.email}**\"**, **\"**{name}**\"**, **\"**{password}**\"**)")  self.open\_window(self.form\_frame, self.item\_list\_window)  **except**:  self.warning\_label["text"] = "Email already registered."  *# Warning text for form handling.*  **def** warning\_text(self, frame):  self.warning\_label = tk.Label(frame, text="", fg="#f10000", bg=self.bg\_color)  self.warning\_label.pack()  *# Define button widget method.*  **def** primary\_button(self, frame, text, handler, pady=10, small=False):  button = tk.Button(  frame,  text=text,  font=("Tahoma", 8),  command=handler,  bg=self.fg\_color,  fg=self.bg\_color,  activebackground=self.bg\_color,  activeforeground=self.fg\_color  )  **if** **not** small:  pady = 20  button.configure(width=10, font=("Tahoma", 12))  button.pack(pady=pady)  **def** open\_window(self, initial\_frame, target\_frame, extra\_frame=False):  initial\_frame.destroy()  **if** extra\_frame:  extra\_frame.destroy()  target\_frame()  **def** open\_edit\_window(self, edit\_name):  self.edit\_name = edit\_name  self.open\_window(self.list\_frame, **lambda**: self.change\_item\_window(2))  *# Login/Register Window*  **def** form\_window(self, category):  **if** category == 1:  primary\_text = "Log In"  secondary\_text = "Don't have an account? Register"  handler = self.log\_in\_handler  goto\_category = 2  **elif** category == 2:  primary\_text = "Register"  secondary\_text = "Already have an account? Log in"  handler = self.register\_handler  goto\_category = 1  self.form\_frame = tk.Frame(self.root, width=400, height=400, bg=self.bg\_color)  header = tk.Label(  self.form\_frame,  text=primary\_text,  font=("Tahoma", 24),  width=30, fg=self.fg\_color, bg=self.bg\_color  )  header.pack(pady=30)  self.warning\_text(self.form\_frame)  self.email\_field = TextInput(self.form\_frame, "Email: ", 30, focus=True)  **if** category == 2:  self.name\_field = TextInput(self.form\_frame, "Name: ", 30)  self.password\_field = TextInput(self.form\_frame, "Password: ", 30)  self.primary\_button(self.form\_frame, secondary\_text, **lambda**: self.open\_window(  self.form\_frame, **lambda**: self.form\_window(goto\_category), self.warning\_label), small=True)  self.primary\_button(self.form\_frame, primary\_text, handler)  self.form\_frame.pack()  self.form\_frame.pack\_propagate(0)  *# Edit/Add Item Window*  **def** change\_item\_window(self, category):  **if** category == 1:  primary\_text = "Add Item"  handler = self.add\_item\_handler  **elif** category == 2:  primary\_text = "Edit Item"  handler = self.edit\_item\_handler  self.change\_item\_frame = tk.Frame(self.root, width=400, height=400, bg=self.bg\_color)  header = tk.Label(  self.change\_item\_frame,  text=primary\_text,  font=("Tahoma", 24),  width=30, fg=self.fg\_color, bg=self.bg\_color  )  header.pack(pady=30)  self.name\_field = TextInput(self.change\_item\_frame, "Item Name: ", focus=True, padx=50, pady=40)  self.primary\_button(self.change\_item\_frame, "Save", handler)  self.primary\_button(self.change\_item\_frame, "Go back to the To-do List.", **lambda**: self.open\_window(self.change\_item\_frame, self.item\_list\_window), small=True)  self.change\_item\_frame.pack()  self.change\_item\_frame.pack\_propagate(0)  *# To-do List Window*  **def** item\_list\_window(self):  self.list\_frame = tk.Frame(self.root, width=400, height=400, bg=self.bg\_color)  header = tk.Label(  self.list\_frame,  text="To-do List",  font=("Tahoma", 24),  width=30, fg=self.fg\_color, bg=self.bg\_color  )  header.pack(pady=10)  self.primary\_button(self.list\_frame, "Add Item", **lambda**: self.open\_window(self.list\_frame, **lambda**: self.change\_item\_window(1)), pady=2, small=True)  self.name\_field = TextInput(self.list\_frame, "Search: ", 30, focus=True, padx=50, pady=5)  self.primary\_button(self.list\_frame, "Search", self.search\_items, pady=2, small=True)  *# Create the todo list dynamically.*  self.items\_frame = tk.Frame(self.list\_frame, width=400, height=300, bg=self.bg\_color)  **for** i **in** self.items:  self.list\_vars[i] = tk.Frame(  self.items\_frame,  width=400,  height=50,  bg=self.fg\_color,  relief=tk.GROOVE,  borderwidth=5,  )  list\_item\_text = tk.Label(self.list\_vars[i], text=i, bg=self.fg\_color, fg=self.bg\_color)  self.list\_buttons[i+"\_button"] = tk.Button(  self.list\_vars[i],  text="X",  bg="#f10000",  fg="#ffffff",  width=2,  command=**lambda** i=i: self.delete\_list\_item(i)  )  self.list\_buttons\_edit[i+"\_button\_edit"] = tk.Button(  self.list\_vars[i],  text="E",  bg="#00b86b",  fg="#ffffff",  width=2,  command=**lambda** i=i: self.open\_edit\_window(i)  )  list\_item\_text.pack(side="left", padx=10)  self.list\_buttons\_edit[i + "\_button\_edit"].pack(side="right", padx=10)  self.list\_buttons[i+"\_button"].pack(side="right", padx=10)  self.list\_vars[i].pack(padx=40)  self.list\_vars[i].pack\_propagate(0)  self.items\_frame.pack(pady=10)  self.list\_frame.pack()  self.list\_frame.pack\_propagate(0)  *# Initialize an object of the class GuiWindow.*  windows = GuiWindow("#221f1f", "#a19a9a", window)  *# Display login window.*  windows.form\_window(1)  window.mainloop() |

### **OUTPUT OF THE PROJECT**

### **Startup**

### 

### **Creating a New User**

### 

### **Logging In**

### 

### **Adding an Item**

### 

### **Searching**

### 

### **Editing an Item**

### 

### **Deleting an Item**

### 

### **REFERENCES**

www.python.org

www.codecademy.com

www.tutorialspoint.com

developers.google.com/edu/python

www.learnpython.org