**Report for project**

(Adaptive Quiz Generator)

**Artificial Intelligence**



**Submitted By:**

Muhammad Awais (23-CS-055)

Muhammad Huzaifa(23-CS-007)

Muhammad Ibraheem(23-CS-019)

Matti-ur-Rehman (23-CS-035)

[C]

**Submitted to:**

Ms Amna Shakeel

**Department of Computer Science,**

**Quiz Generator**

**Summary,Introduction 3**

**objective,scope 4**

**libraries,tools 5**

**flowchart 6**

**Outputs 7**

**Code 8-13**

**Executive Summary (Abstract)**

The Adaptive AI-Powered Quiz Generator is a Python-based application developed using Tkinter for GUI and pandas for data handling. It dynamically adjusts question difficulty based on user performance and provides an engaging, personalized quiz experience. The system implements machine learning-inspired adaptive logic to track user skill levels across easy, medium, and hard questions. The user's score and question history are saved, and the application offers a smooth login/signup interface, performance tracking, and future extensibility for AI features like predictive scoring.

**Introduction**

**Project Background:** Traditional quizzes offer static difficulty levels that fail to cater to a learner's individual pace and progress. This project aims to create a smart, AI-assisted quiz system that learns from user performance and adapts accordingly, enhancing learning outcomes.

**Objective:**

* Build a personalized quiz platform.
* Dynamically adapt difficulty based on accuracy.
* Provide intuitive GUI-based interactions.
* Store and track user performance history.

**Scope:**

* Supports adaptive difficulty adjustment.
* Extensible to include machine learning models.
* Tracks user login, score, and question history.
* Friendly GUI using Tkinter.

**System Requirements**

**Hardware Requirements:**

* Minimum: 2 GB RAM, Dual-Core Processor, 100 MB disk space
* Recommended: 4 GB RAM, Quad-Core Processor, 500 MB disk space

**Software Requirements:**

Python 3.8+

**Required Libraries:**

* pandas
* tkinter (standard)
* os, random

**Algorithms:**

* Adaptive Difficulty Logic (Custom rule-based logic)
* Extensible for Logistic Regression or other AI models

**Development Tools:**

* VS Code (Preferred)

**System Design**

Flowchart:

START

|

Login/Signup

|

Validate User

|

Load Unanswered Questions by Difficulty

|

Show Question + Options

|

Get Answer

|

Check Correct?

| |

Yes No

| |

+1 Score Adjust Difficulty

| |

Update History

|

More Questions?

| |

Yes No

| |

Loop END

**Technologies Used:**

* Python: Core logic and data handling.
* Tkinter: Interactive GUI.
* pandas: CSV data processing.

Why Chosen?

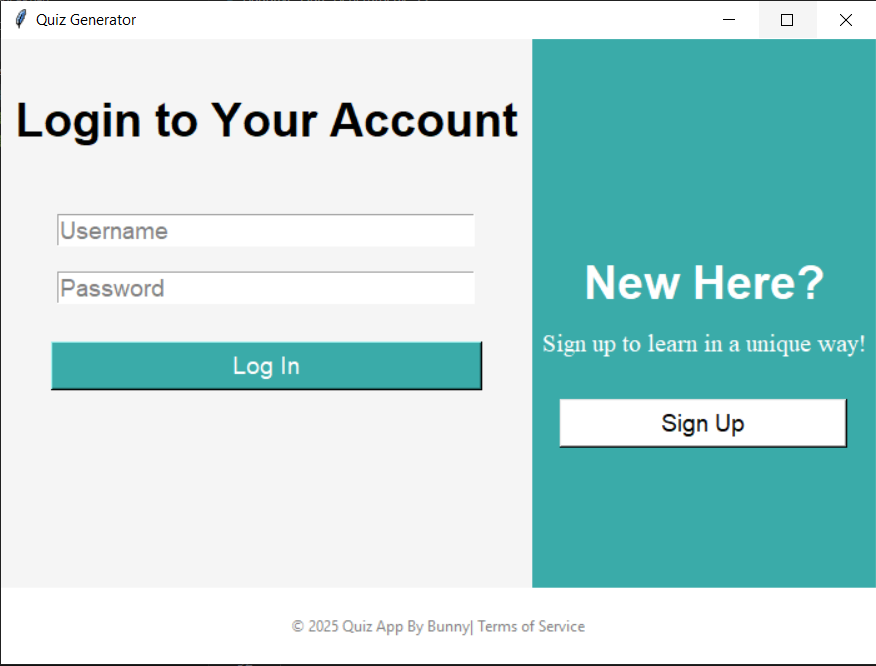
* Solves real life problem
* Good for deployment.
* Productive for student-level enhancement.

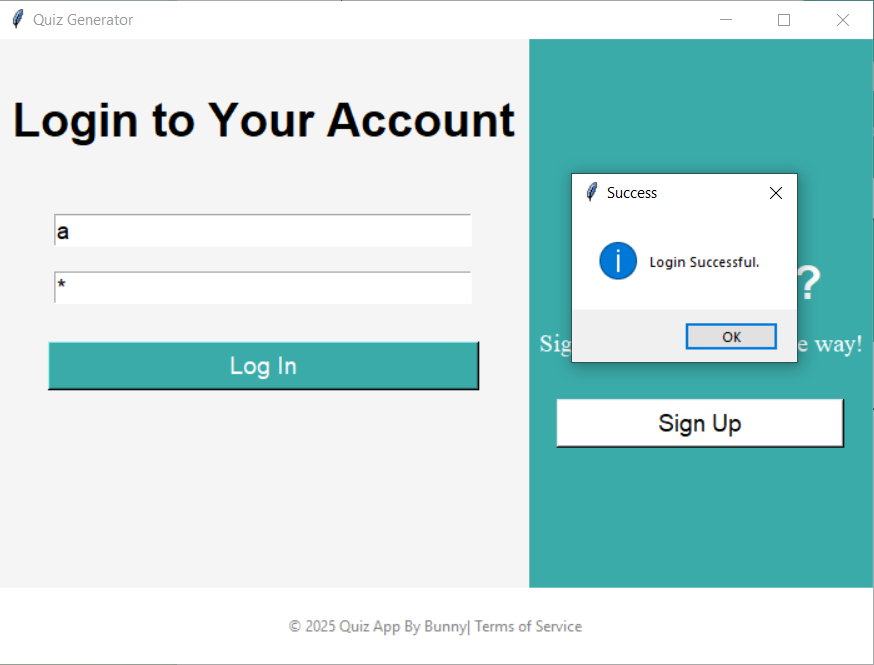
**Code Structure:**

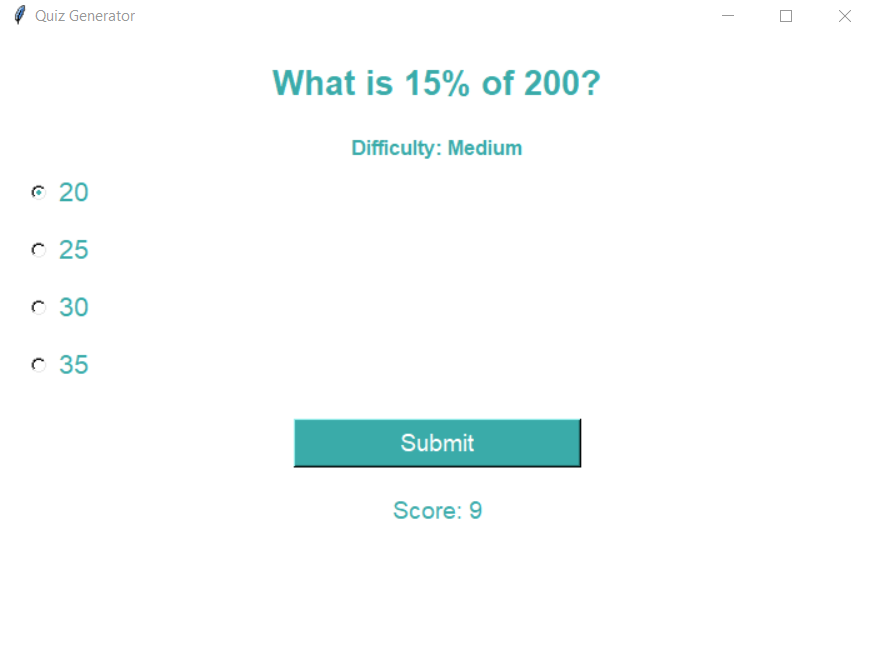
* Signup/Login`: Authenticates users, stores in `profiles.csv`
* questions.csv`: Stores questions, options, correct answers, and difficulty.
* AdaptiveDifficulty`: Adjusts current difficulty based on past performance.
* check\_answer()`: Evaluates response and updates performance.
* start\_quiz()`, `next\_question()`: Drives quiz flow and GUI transitions.

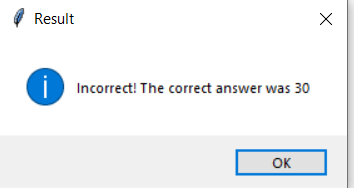
**Results and Discussion**

**Screenshots:**









**Challenges Faced:**

* Designing difficulty transitions without actual ML model.
* Solved using rule-based logic and performance thresholds.
* Preventing repeated questions.
* Solved using a user history and filtering logic.
* GUI consistency and responsiveness.
* Addressed by structuring Tkinter elements into containers and panels.

**Appendices**

Source Code: Full source code is included in this document below.

**Code**

|  |
| --- |
| from tkinter import \*  from tkinter import messagebox  import pandas as pd  import os  import random  # Data handling functions (same as your original)  profiles\_file = 'profiles.csv'  question\_file = 'questions.csv'  questions\_df = pd.read\_csv(question\_file)  class AdaptiveDifficulty:  def \_\_init\_\_(self, initial\_difficulty='medium'):  self.user\_stats = {  'easy': {'correct': 0, 'total': 0},  'medium': {'correct': 0, 'total': 0},  'hard': {'correct': 0, 'total': 0}  }  self.current\_difficulty = initial\_difficulty    def update\_user\_response(self, is\_correct):  level\_stats = self.user\_stats[self.current\_difficulty]  level\_stats['total'] += 1  if is\_correct:  level\_stats['correct'] += 1    def calculate\_success\_rate(self, difficulty):  stats = self.user\_stats[difficulty]  if stats['total'] == 0:  return 0  return stats['correct'] / stats['total']    def adjust\_difficulty(self):  success\_rate = self.calculate\_success\_rate(self.current\_difficulty)    if self.current\_difficulty == 'easy':  if success\_rate > 0.7:  self.current\_difficulty = 'medium'    elif self.current\_difficulty == 'medium':  if success\_rate > 0.7:  self.current\_difficulty = 'hard'  elif success\_rate < 0.3:  self.current\_difficulty = 'easy'    elif self.current\_difficulty == 'hard':  if success\_rate < 0.3:  self.current\_difficulty = 'medium'    return self.current\_difficulty  def Signup(username, password):  if os.path.exists(profiles\_file):  df = pd.read\_csv(profiles\_file)  else:  df = pd.DataFrame(columns=['username', 'password', 'score', 'question\_history'])  if username in df['username'].tolist():  return False    new = pd.DataFrame([[username, password, 0, '']],  columns=['username', 'password', 'score', 'question\_history'])  df = pd.concat([df, new], ignore\_index=True)  df.to\_csv(profiles\_file, index=False)  return True  def Login(username, password):  if not os.path.exists(profiles\_file):  return False    df = pd.read\_csv(profiles\_file)  if (username in df['username'].tolist()) and (df.loc[df['username']==username,'password'].values[0] == password):  return True  return False  def get\_history(username):  df = pd.read\_csv(profiles\_file)  idx = df.index[df['username']==username][0]  history = df.at[idx,'question\_history']  if pd.isnull(history) or history == '' or history is None:  return set()  return set(history.split('|'))  def update\_history(username, question):  df = pd.read\_csv(profiles\_file)  idx = df.index[df['username']==username][0]  history = df.at[idx,'question\_history']  if pd.isnull(history) or history == '' or history is None:  questions = []  else:  questions = history.split('|')  questions.append(question)  df.at[idx,'question\_history'] = "|".join(questions)  df.to\_csv(profiles\_file, index=False)  def update\_score(username, points):  df = pd.read\_csv(profiles\_file)  idx = df.index[df['username']==username][0]  df.at[idx,'score'] += points  df.to\_csv(profiles\_file, index=False)  def get\_score(username):  df = pd.read\_csv(profiles\_file)  idx = df.index[df['username']==username][0]  return int(df.at[idx,'score'])  #--------------------------------------------------------------------------#  current\_user = None  question\_pool = []  question = None  options = []  difficulty\_adapter = None  def start\_quiz(username):  global current\_user, question\_pool, difficulty\_adapter  current\_user = username  difficulty\_adapter = AdaptiveDifficulty() # Initialize adapter  answered = get\_history(username)    # Get questions filtered by current difficulty  update\_question\_pool()    if not question\_pool:  messagebox.showinfo("Done", "All questions answered at this difficulty level.")  show\_login()  return    next\_question()  def update\_question\_pool():  global question\_pool, difficulty\_adapter  answered = get\_history(current\_user)  current\_diff = difficulty\_adapter.current\_difficulty    # Filter questions by current difficulty and unanswered  question\_pool = questions\_df[  (questions\_df['difficulty'] == current\_diff) &  (~questions\_df['question'].isin(answered))  ]['question'].tolist()    # If none left in current difficulty, try other difficulties  if not question\_pool:  question\_pool = questions\_df[  ~questions\_df['question'].isin(answered)  ]['question'].tolist()  def next\_question():  for widget in root.winfo\_children():  widget.destroy()  if not question\_pool:  messagebox.showinfo("Done", "You are now Einstein :)")  show\_login()  return  global question, options, var    question = random.choice(question\_pool)  row = questions\_df.loc[questions\_df['question']==question].iloc[0]  options = [row['option1'], row['option2'], row['option3'], row['option4']]  correct = row['correct\_answer']  question\_lbl = Label(root, text=question, font=('Helvetica', 20, 'bold'), wraplength=400, fg='#3aaba9', bg='white')  question\_lbl.pack(pady=20)  # Add difficulty display  difficulty\_lbl = Label(root,  text=f"Difficulty: {difficulty\_adapter.current\_difficulty.capitalize()}",  font=('Helvetica', 12, 'bold'),  fg='#3aaba9',  bg='white')  difficulty\_lbl.pack()  var = IntVar()  for i, opt in enumerate(options, start=1):  r = Radiobutton(root, text=opt, variable=var, value=i, font=('Helvetica', 16),  fg='#3aaba9', bg='white', selectcolor='white')  r.pack(anchor='w', padx=20, pady=5)  submit = Button(root, text='Submit', font=('Helvetica', 14), command=check\_answer,  bg='#3aaba9', fg='white', width=20)  submit.pack(pady=20)  score = get\_score(current\_user)  score\_lbl = Label(root, text=f"Score: {score}", font=('Helvetica', 14), fg='#3aaba9', bg='white')  score\_lbl.pack()  def check\_answer():  choice = var.get()  row = questions\_df.loc[questions\_df['question']==question].iloc[0]  correct = row['correct\_answer']  is\_correct = (options[choice-1] == correct)  # Update difficulty system  difficulty\_adapter.update\_user\_response(is\_correct)  difficulty\_adapter.adjust\_difficulty()    if is\_correct:  update\_score(current\_user, 1)  feedback = "Correct!"  else:  feedback = f"Incorrect! The correct answer was {correct}"  messagebox.showinfo("Result", feedback)  update\_history(current\_user, question)  update\_question\_pool() # Refresh pool with new difficulty  next\_question()  #login/signup GUI functions  def on\_entry\_click(event):  entry = event.widget  if entry == username\_entry and entry.get() == "Username":  entry.delete(0, END)  entry.config(fg="black", show="")  elif entry == password\_entry and entry.get() == "Password":  entry.delete(0, END)  entry.config(fg="black", show="\*")  def on\_focus\_out(event):  entry = event.widget  if entry == username\_entry and entry.get() == "":  entry.insert(0, "Username")  entry.config(fg="grey", show="")  elif entry == password\_entry and entry.get() == "":  entry.insert(0, "Password")  entry.config(fg="grey", show="")  def handle\_login():  username = username\_entry.get()  password = password\_entry.get()    if Login(username, password):  messagebox.showinfo("Success", "Login Successful.")  start\_quiz(username)  else:  messagebox.showerror("Error", "Invalid credentials.")  def handle\_signup():  username = username\_entry.get()  password = password\_entry.get()    if Signup(username, password):  messagebox.showinfo("Success", "Signup Successful.")  start\_quiz(username)  else:  messagebox.showerror("Error", "Username already exists.")  def show\_login():  for widget in root.winfo\_children():  widget.destroy()  root.config(bg="white")  main\_container = Frame(root, bg="white")  main\_container.pack(fill=BOTH, expand=True)  # Left panel (login form)  left\_panel = Frame(main\_container, bg="#f5f5f5", width=350)  left\_panel.pack(side=LEFT, fill=BOTH, expand=True)  left\_panel.pack\_propagate(False)  head = Label(left\_panel, text="Login to Your Account", font=("Arial", 28, "bold"),  fg="black", bg="#f5f5f5")  head.pack(pady=40)  global username\_entry, password\_entry  username\_entry = Entry(left\_panel, width=30, font=('Helvetica', 14), fg="grey")  username\_entry.insert(0, "Username")  username\_entry.bind("<FocusIn>", on\_entry\_click)  username\_entry.bind("<FocusOut>", on\_focus\_out)  username\_entry.pack(pady=10)  password\_entry = Entry(left\_panel, width=30, font=('Helvetica', 14), fg="grey")  password\_entry.insert(0, "Password")  password\_entry.bind("<FocusIn>", on\_entry\_click)  password\_entry.bind("<FocusOut>", on\_focus\_out)  password\_entry.pack(pady=10)  sign\_in = Button(left\_panel, text="Log In", bg="#3aaba9", fg="white",  font=('Helvetica', 14), command=handle\_login)  sign\_in.pack(pady=20, padx=40, fill=X)  # Right panel (signup portion)  right\_panel = Frame(main\_container, bg="#3aaba9", width=200)  right\_panel.pack(side=RIGHT, fill=BOTH, expand=True)  right\_panel.pack\_propagate(False)  right\_head = Label(right\_panel, text="New Here?", font=("Arial", 28, "bold"),  bg="#3aaba9", fg="white")  right\_head.pack(pady=(170,0))  right\_subhead = Label(right\_panel, text="Sign up to learn in a unique way!",  font=("Times new roman", 15), bg="#3aaba9", fg="white")  right\_subhead.pack(pady=10)    sign\_up = Button(right\_panel, text="Sign Up", width=20, bg="white",  fg="black", font=('Helvetica', 14), command=handle\_signup)  sign\_up.pack(pady=20)  # Footer  footer = Label(root, text="© 2025 Quiz App By Bunny| Terms of Service",  bg="white", fg="gray")  footer.pack(side=BOTTOM, pady=20)  # Main window setup  root = Tk()  root.title("Quiz Generator")  root.geometry("700x500")  root.config(bg="#ffffff")  show\_login()  root.mainloop() |