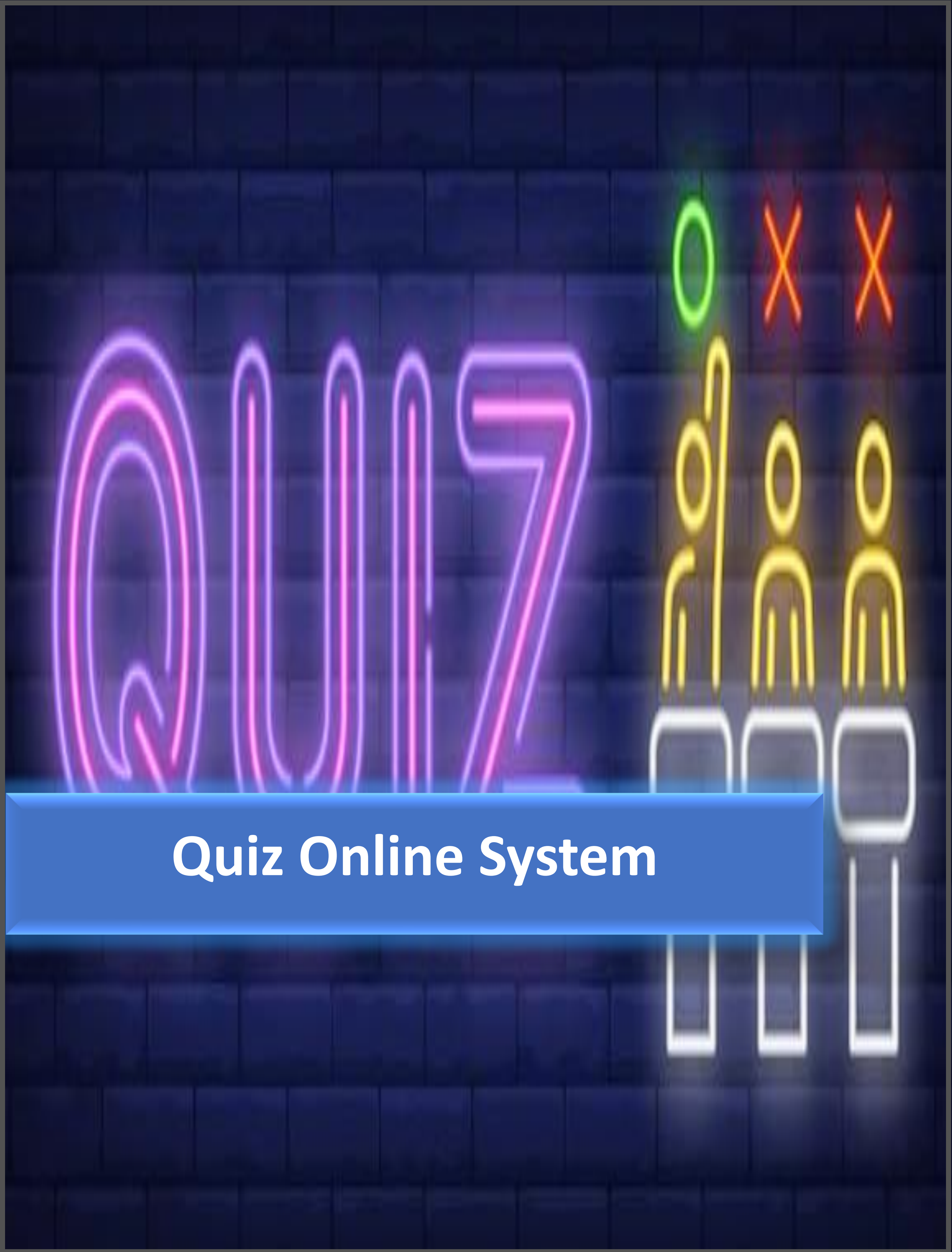


Quiz Online System

Information about me

★ Mohamed Reda Ramadan Khamis ★

college	★ Shubra Faculty of Engineering ★
University	★ Banha University ★
Department	★ Communications and Computers ★
Phone. No	★ 01554725661 ★
College Email	★ Muhammad20102@feng.bu.edu.eg ★
Gmail_1	★ redamedo841@gmail.com ★
Gmail_1	★ medokhamis29@gmail.com ★

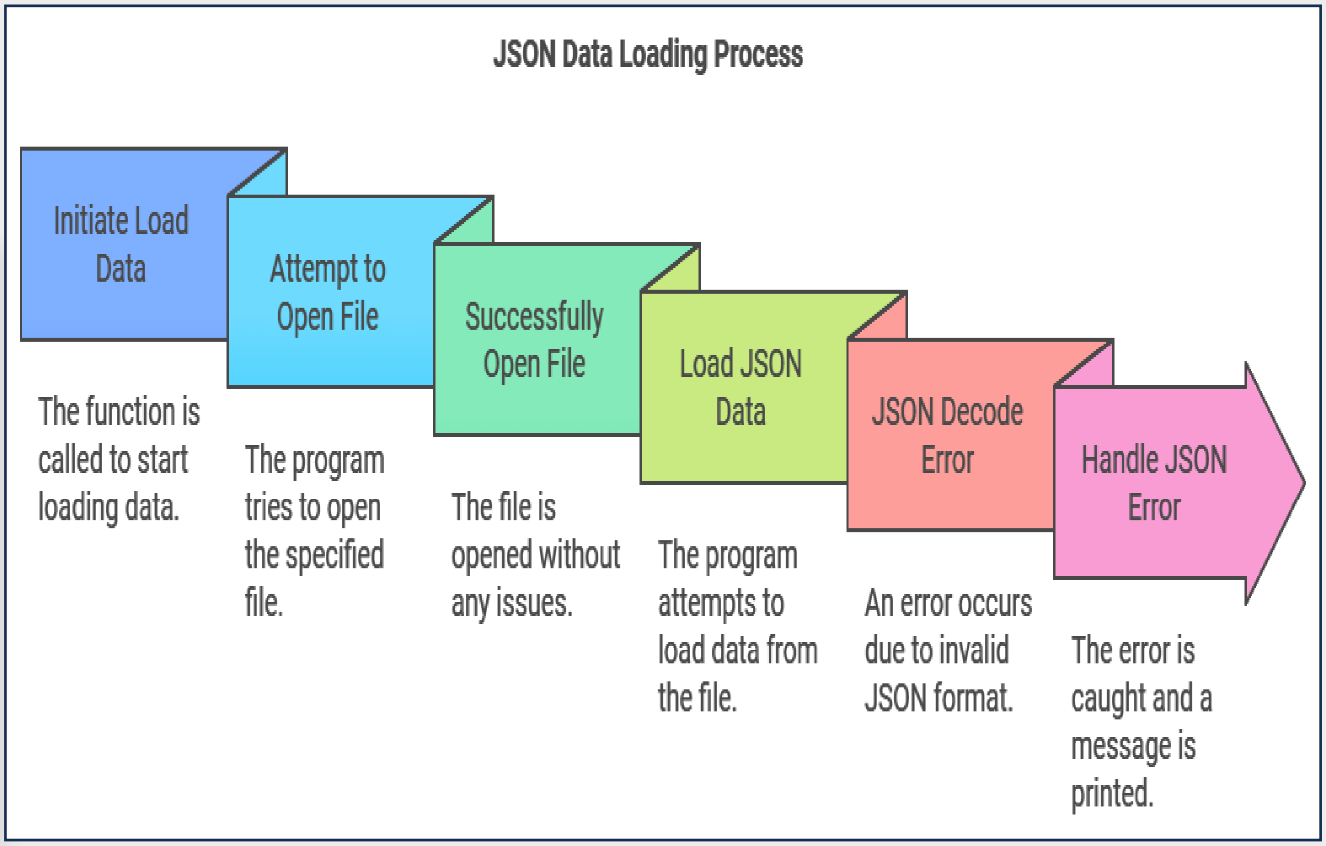
Abstract

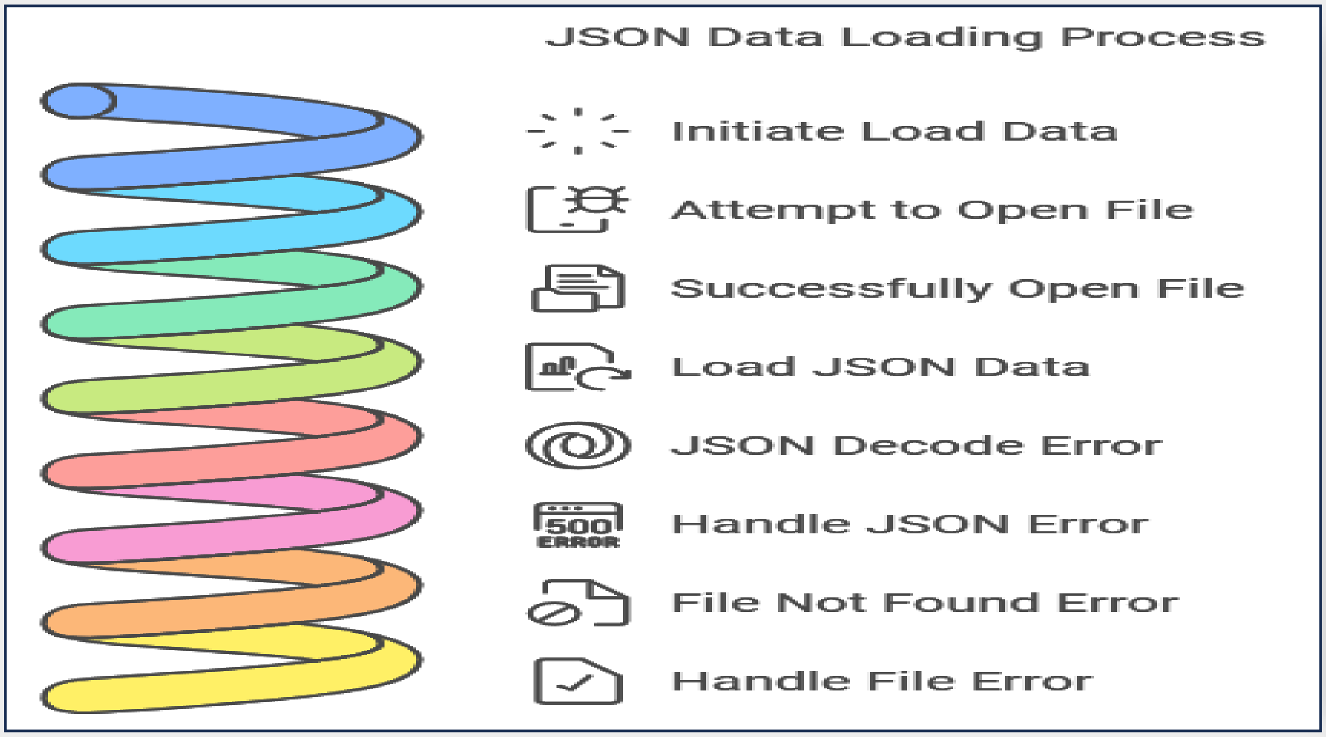
- The Online Quiz System is an advanced web-based platform designed to facilitate interactive quizzes for users, offering a streamlined experience for both participants and administrators. This system enables users to register and log in securely, utilizing hashed passwords and email verification for enhanced security. Administrators can create customized quizzes by selecting topics from a predefined question bank, allowing for tailored assessments that meet diverse educational needs. Participants can take quizzes at their convenience, with immediate feedback on their performance and automatic recording of results.
- The system stores user data, quiz questions, and results in JSON format, ensuring easy data management and retrieval. With features such as role-based access control, quiz password protection, and a results viewing function, the Online Quiz System promotes a structured approach to learning and assessment. This application can be further expanded to include analytics, leaderboard functionalities, and integrations with educational tools, making it a versatile choice for organizations aiming to enhance their training and evaluation processes.

Explanation of Python code

1 load data Function

- The load_data function is designed to read data from a specified JSON file and return it as a Python dictionary. Here's a breakdown of its components and functionality
- Function Definition: This line defines a function named load_data that takes one argument, file_path, which is expected to be a string representing the path to the JSON file.
- Error Handling: The try block is used to handle potential exceptions that may occur during the execution of the code within it. This allows the program to continue running even if an error occurs, rather than crashing.
- Opening the File: This line opens the specified file in read mode ('r'). The with statement ensures that the file is properly closed after its suite finishes, even if an error occurs.
- Loading JSON Data: The json.load(file) function reads the JSON data from the opened file and converts it into a Python dictionary, which is stored in the variable data.


JSON Data Loading Process
Initiate Load Data
The function is called to start loading data.
Attempt to Open File
The program tries to open the specified file.
Successfully Open File
The file is opened without any issues.
Load JSON Data
The program attempts to load data from the file.
JSON Decode Error
An error occurs due to invalid JSON format.
Handle JSON Error
The error is caught and a message is printed.


JSON Data Loading Process
Initiate Load Data
Attempt to Open File
Successfully Open File
Load JSON Data
JSON Decode Error
Handle JSON Error
File Not Found Error
Handle File Error

2 Save data Function

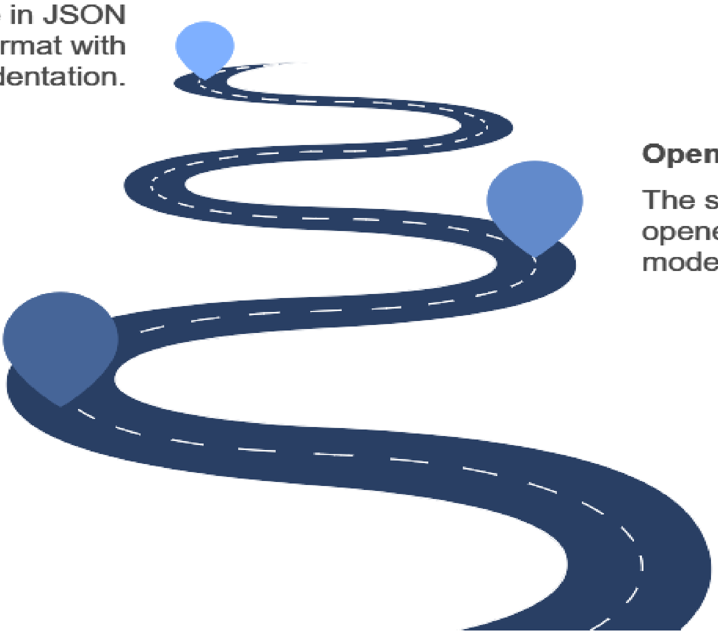
- The save_data function is designed to write a Python object (typically a dictionary) to a specified JSON file. Here's a breakdown of its components and functionality.
- Function Definition: This line defines a function named save_data that takes two arguments.

- Docstring: This is a brief comment that describes the purpose of the function. It indicates that the function is responsible for saving data to a JSON file.
- Opening the File: This line opens the specified file in write mode ('w').

- Context Manager: The with statement ensures that the file is properly closed after its suite finishes, even if an error occurs. If the file already exists, opening it in write mode will overwrite its contents.

- Writing JSON Data: The json.dump(data, file, indent=4) function takes the Python object (data) and converts it to JSON format, writing it directly to the opened file.

Saving Data to JSON File



Write Data

The data is written to the file in JSON format with indentation.

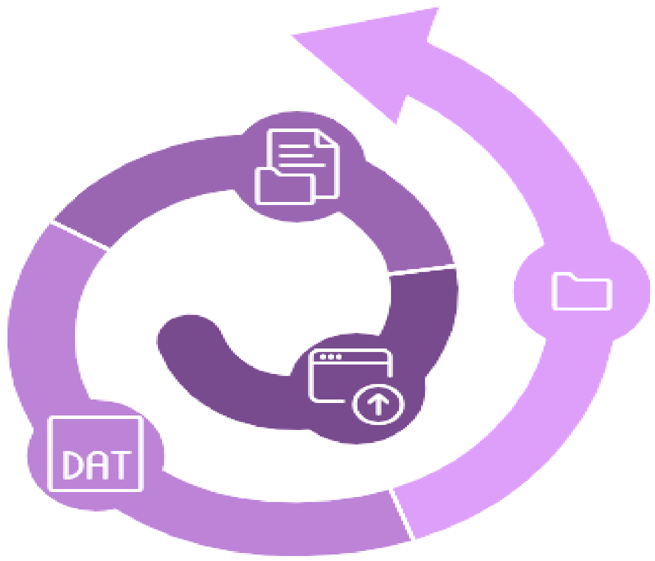
Open File

The specified file is opened in write mode.

Function Call

The function save_data is invoked with file_path and data as arguments.

Saving Data to JSON File



01 Initiate Save Process

02 Open File

03 Write Data

04 Close File

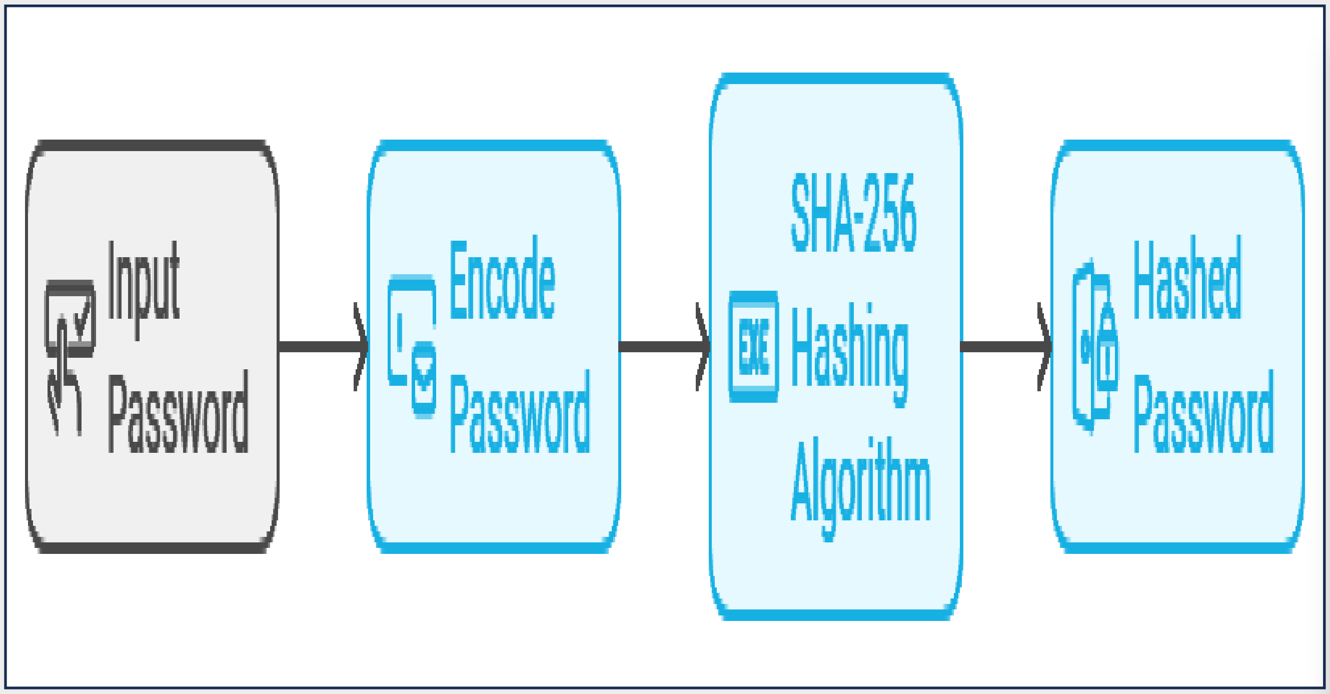
3 Hash Password Function

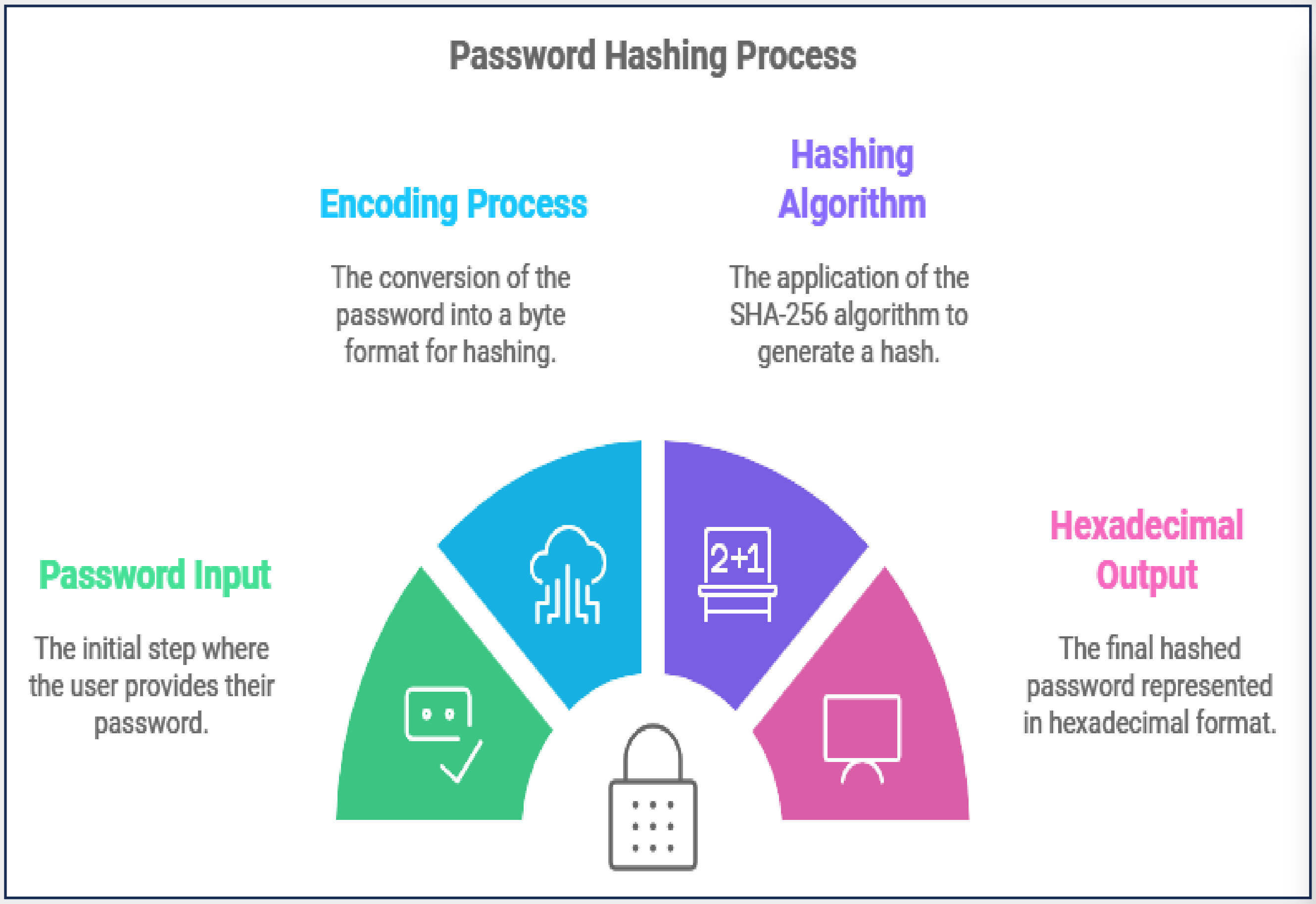
- The hash_password function is responsible for securely hashing a plaintext password using the SHA-256 hashing algorithm. This is a crucial practice in application security, as it helps protect user passwords from being easily accessed or compromised. Here’s a breakdown of its components and functionality.

- Function Definition: This line defines a function named hash_password that takes one argument:
  - password: A string representing the plaintext password that needs to be hashed.

- Encoding the Password: password.encode() converts the plaintext password from a string to bytes. This is necessary because the hashing function requires byte input.

- SHA-256 Hashing: hashlib.sha256(...) creates a SHA-256 hash object. The hashlib library is a built-in Python module that provides a way to create secure hash functions.


Input Password
Encode Password
SHA-256 Hashing Algorithm
Hashed Password


Password Hashing Process
Encoding Process
The conversion of the password into a byte format for hashing.
Hashing Algorithm
The application of the SHA-256 algorithm to generate a hash.
Password Input
The initial step where the user provides their password.
Hexadecimal Output
The final hashed password represented in hexadecimal format.
2+1

4

Send Verification Code Function

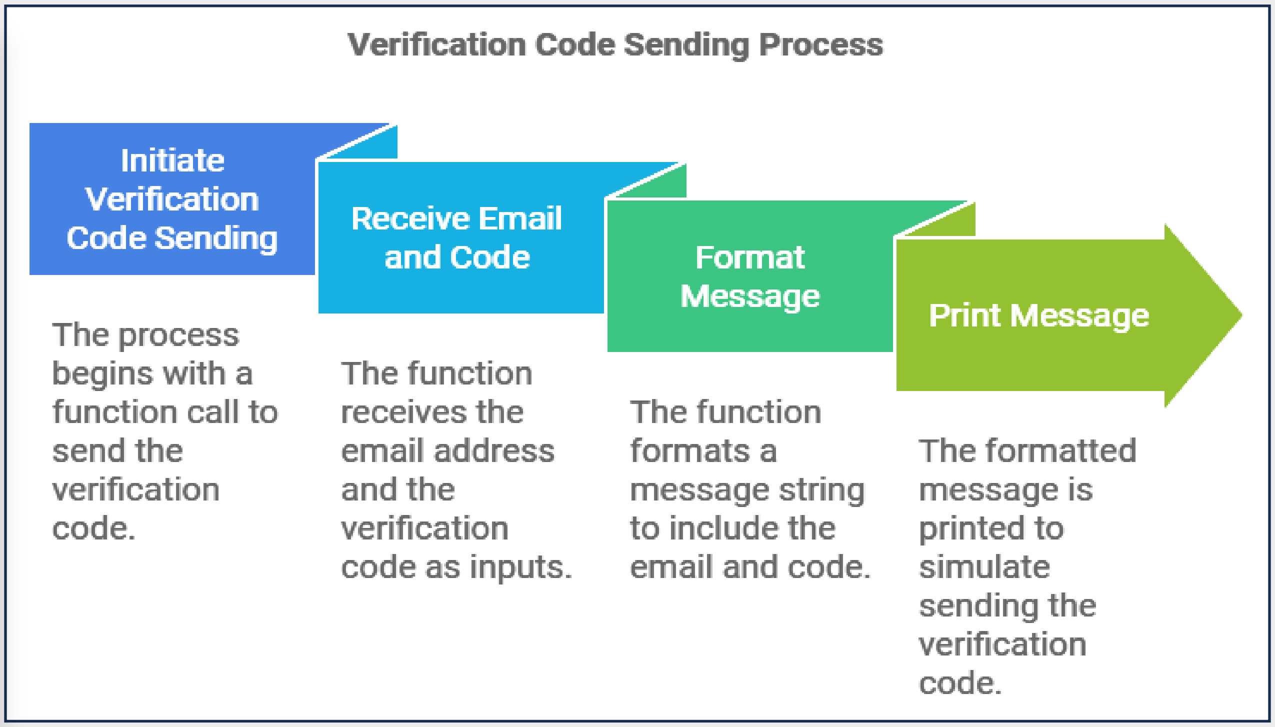
- The password.encode() converts the password to bytes (required by the hashing function).

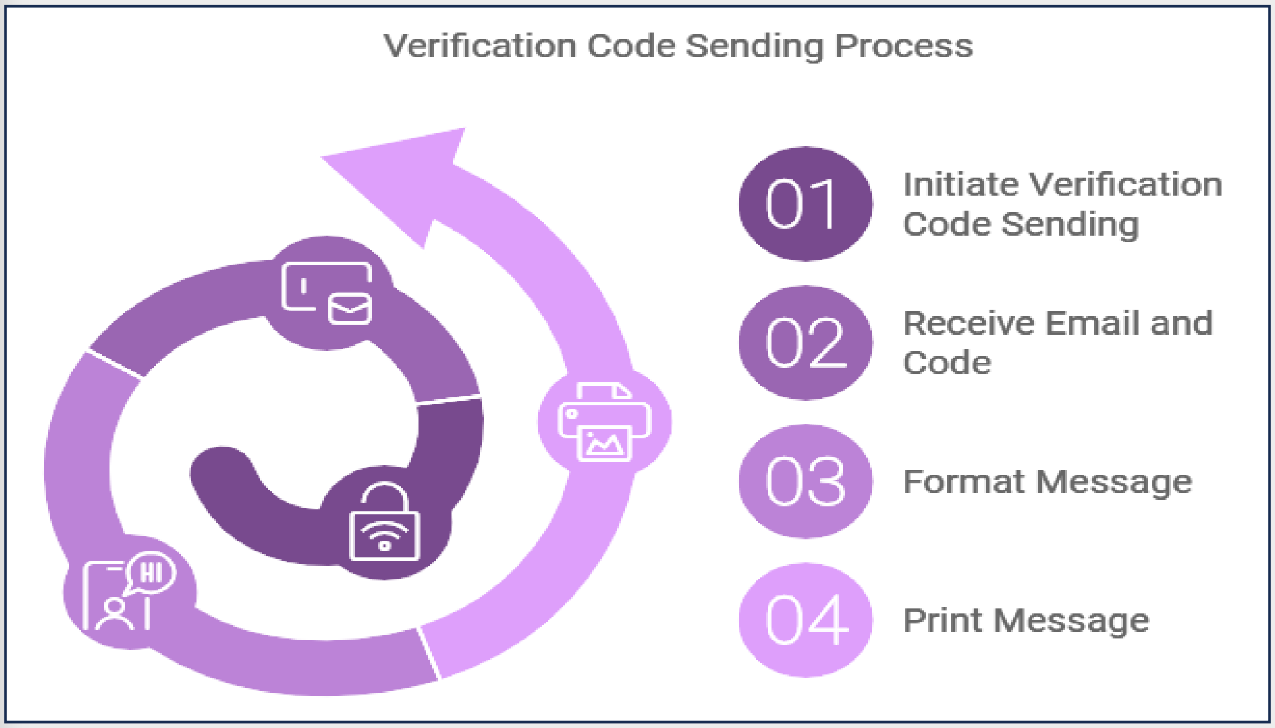
- .hexdigest() converts the result to a readable hexadecimal string, representing the hash value.

- This is commonly used to securely store passwords, as it’s challenging to reverse-engineer the original password from its hash.

- This function simulates sending a verification code to a user's email.
- email is expected to be a part of the email address (e.g., the username before "@gmail.com"), and code is the verification code.

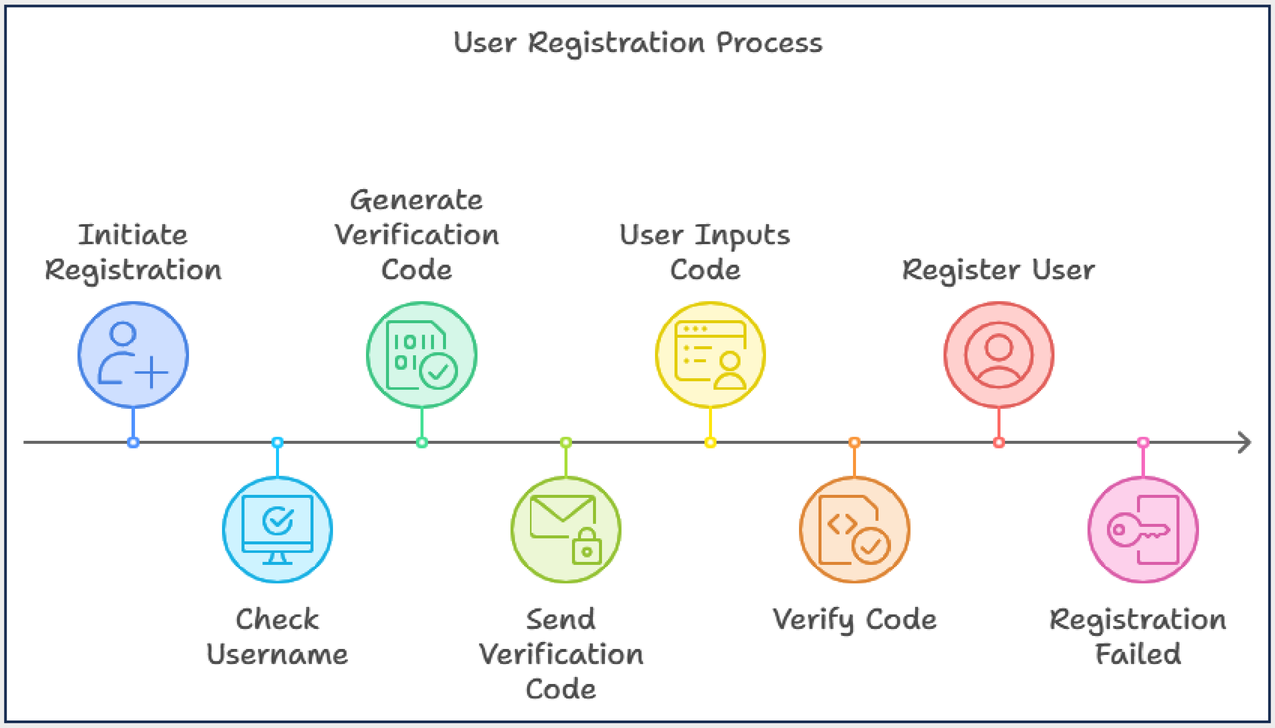
- The function then prints a message showing the email and the code, simulating a real code-sending mechanism for user verification.

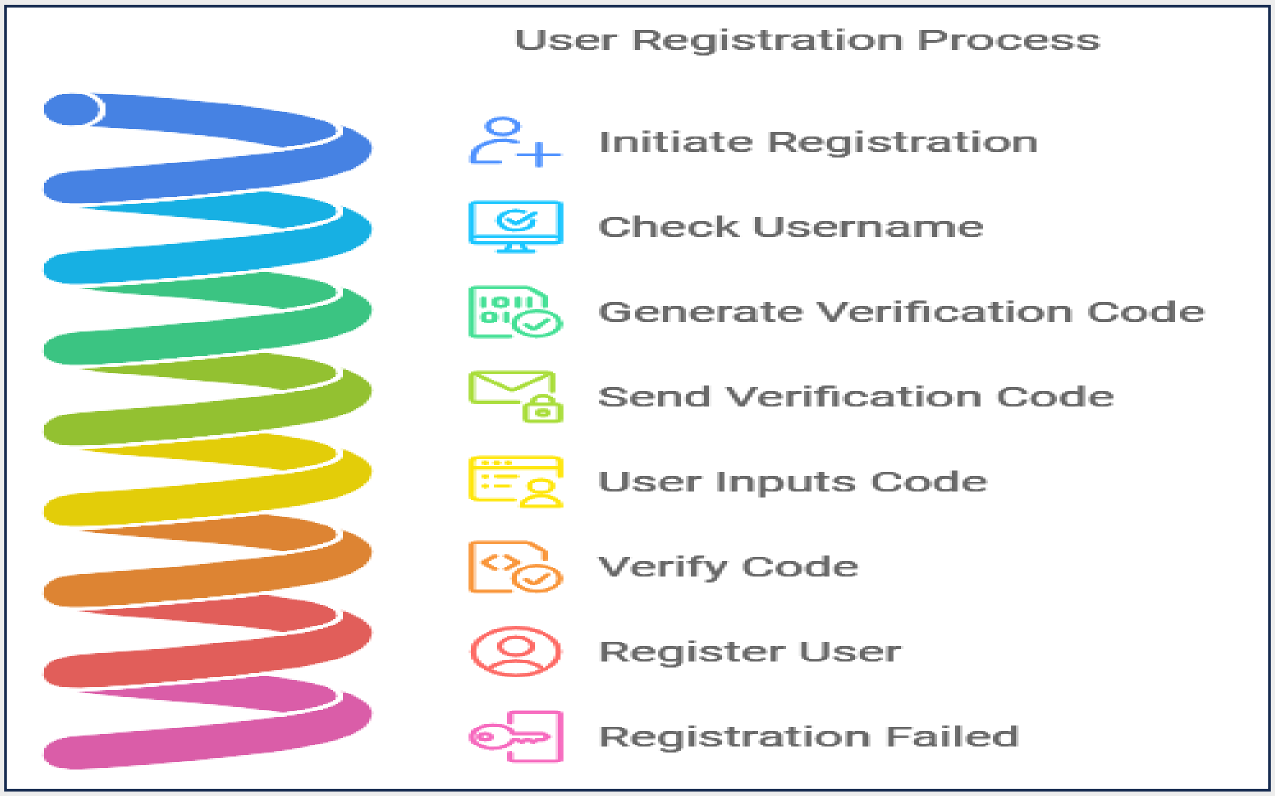

Verification Code Sending Process
Initiate Verification Code Sending
Receive Email and Code
Format Message
Print Message
The process begins with a function call to send the verification code.
The function receives the email address and the verification code as inputs.
The function formats a message string to include the email and code.
The formatted message is printed to simulate sending the verification code.


Verification Code Sending Process
01 Initiate Verification Code Sending
02 Receive Email and Code
03 Format Message
04 Print Message
HI

5 Register Function

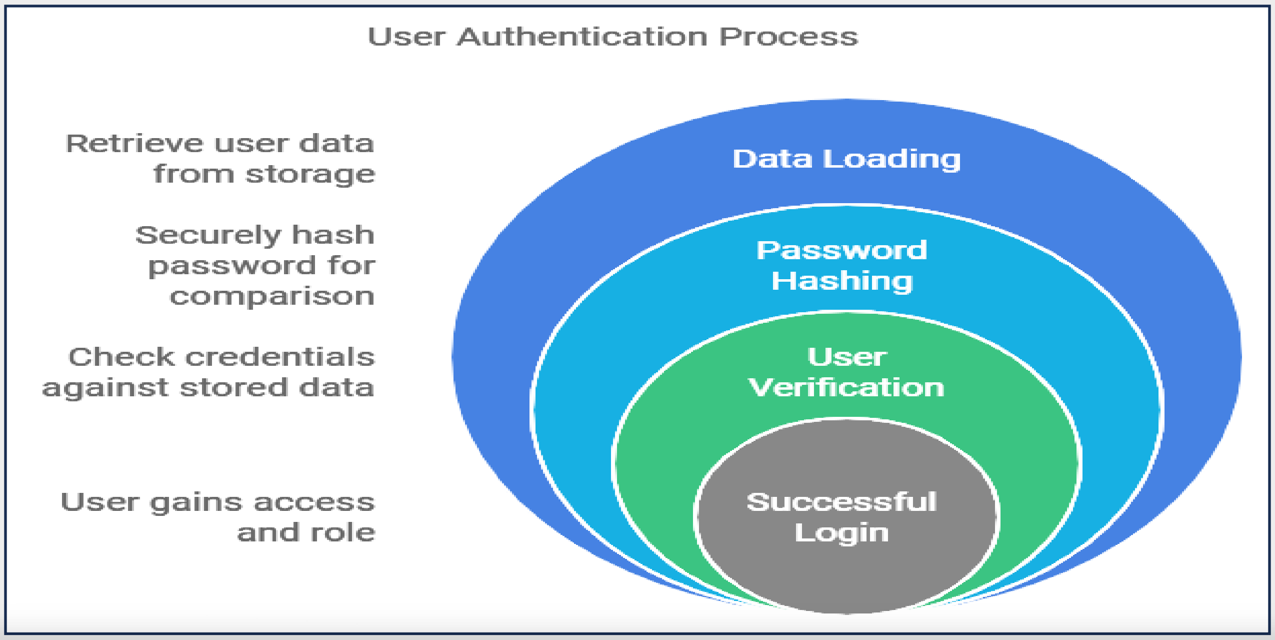
- loads existing user data from a specified file (USERS_FILE) to check if the username already exists.
- If the username is already in the loaded user data, the function prints a message and returns False, indicating registration failure due to a duplicate username.
- A six-digit verification code is generated using random.randint to create a number between 100000 and 999999.
- The function prompts the user for an email address and simulates sending the code to the specified email using send_verification_code.
- The function then stores the user's information (username, hashed password, and role) in the users dictionary and saves this updated dictionary back to the data file using save_data.
- If the codes do not match, an error message is displayed, and False is returned, indicating registration failure.
- This register_user function ensures that new user registration includes both unique usernames and verification, adding an extra layer of security and integrity to the registration process.


User Registration Process
Initiate Registration
Check Username
Generate Verification Code
Send Verification Code
User Inputs Code
Verify Code
Register User
Registration Failed


User Registration Process
Initiate Registration
Check Username
Generate Verification Code
Send Verification Code
User Inputs Code
Verify Code
Register User
Registration Failed

6 Login Function

- The login_user function authenticates a user by verifying their username and password.
- The function begins by loading the existing user data from the USERS_FILE to retrieve stored usernames and passwords.
- First, it checks if the username exists in the users dictionary.
- If the username exists, the function compares the hashed version of the provided password with the stored hashed password for that username :
    - (using hash_password(password)).
- If the hashes match, a welcome message is printed, and the user's role (e.g., "admin", "user") is returned. This role can be used later for role-based access control.
- If the username is not found or the password doesn't match, it displays an "Invalid credentials" message.
- None is returned, indicating that the login attempt was unsuccessful.
- login_user securely verifies credentials using hashed passwords and provides role-based access for successful logins.


User Authentication Process
Retrieve user data from storage
Securely hash password for comparison
Check credentials against stored data
User gains access and role
Data Loading
Password Hashing
User Verification
Successful Login

7 Create Quiz Function

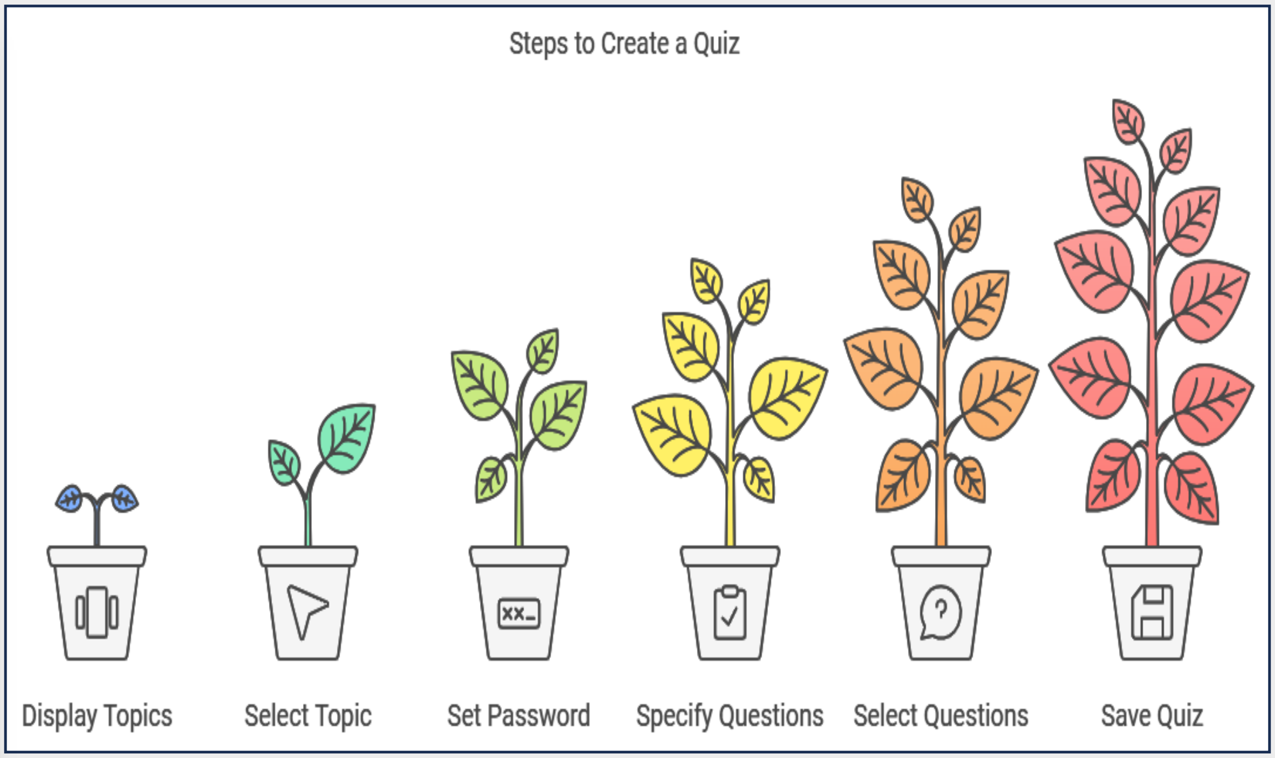
- **The create_quiz function allows an admin to create a new quiz, selecting questions based on a specified topic.**

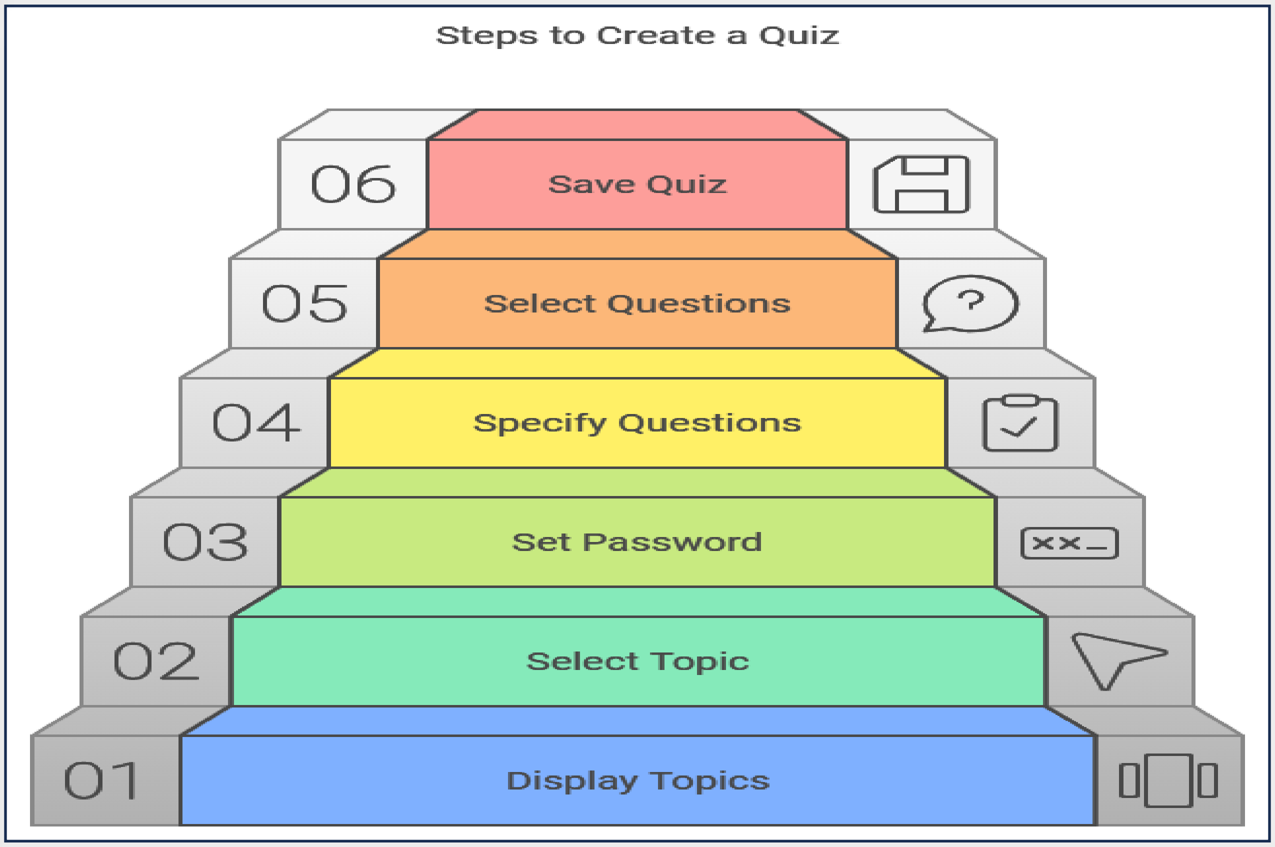
- **QUIZZES_FILE is loaded to access and update the existing quizzes.**

- **question_bank.json is loaded to access a collection of available quiz questions, organized by topic.**

- **quiz_id is generated using the current timestamp to ensure uniqueness.**

- The available topics in question_bank are displayed, allowing the admin to choose a valid topic.
- If the entered topic is not in question_bank, an error message is shown, and the function exits.
- The admin sets a password for the quiz to restrict access.
- The admin specifies the number of questions they want in the quiz.
- If the requested number of questions exceeds the available questions for the chosen topic, an error is shown.
- Otherwise, random.sample selects the specified number of random questions from the topic.
- The quiz details, including the admin, topic, password, selected questions, and creation timestamp, are saved in the quizzes dictionary.
- The updated quizzes are saved back to QUIZZES_FILE, and a success message is printed.
- This function lets an admin create a quiz by selecting random questions from a chosen topic, adding flexibility and variety to quiz creation. It also allows for quiz protection through a password.


Steps to Create a Quiz
Display Topics
Select Topic
Set Password
Specify Questions
Select Questions
Save Quiz


Steps to Create a Quiz
06
Save Quiz
05
Select Questions
04
Specify Questions
03
Set Password
02
Select Topic
01
Display Topics

8 Take Quiz Function

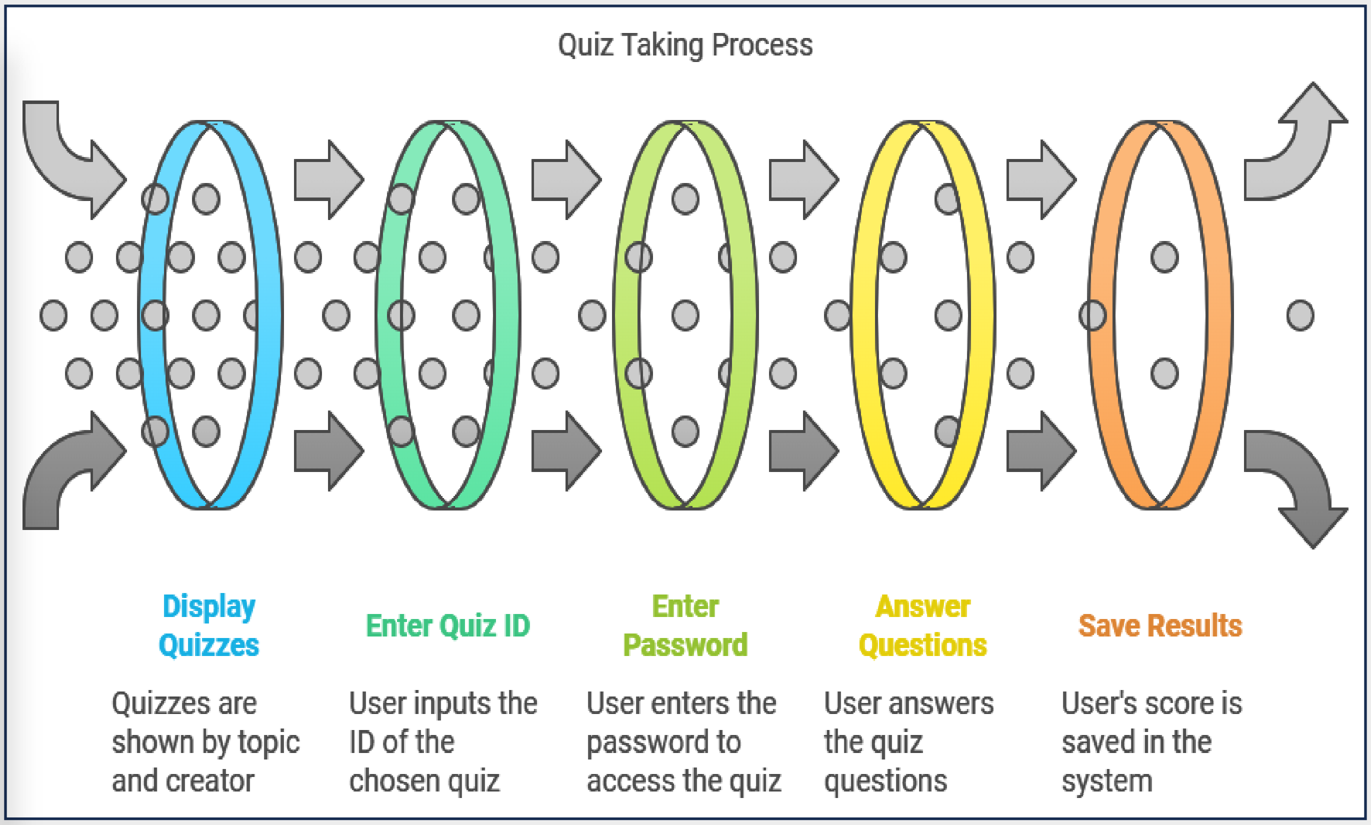
- The take_quiz function enables a user to take a selected quiz, score their answers, and save the result.

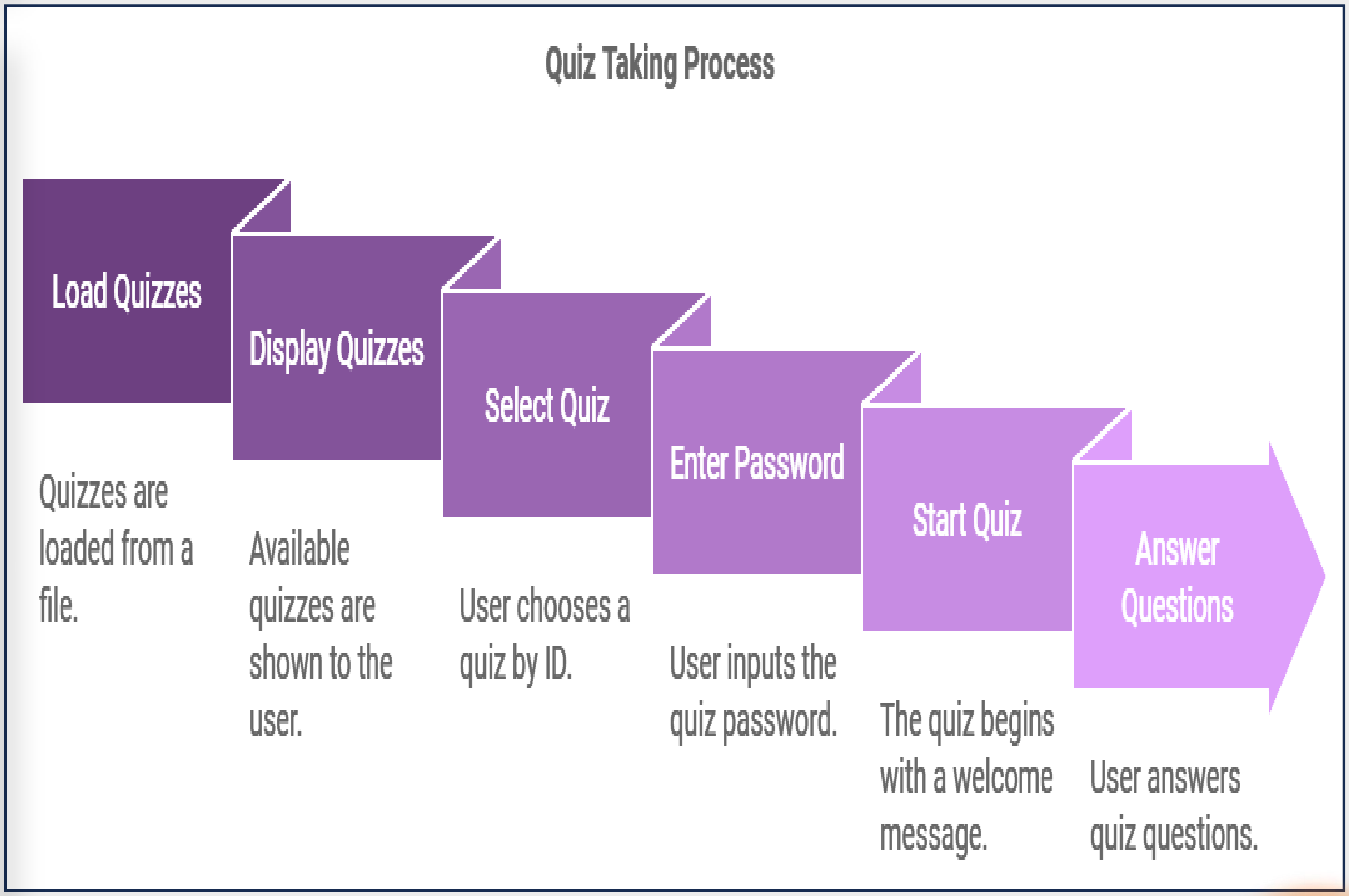
- The function first loads the quizzes from QUIZZES_FILE.
- If no quizzes are available, it displays a message and exits.
- Each available quiz is displayed along with its ID, topic, and creator's username (admin).
- This information helps the user choose a quiz by ID.

- Each question is displayed with its multiple-choice options.
- The user selects an answer for each question, and if the answer is correct, the score is incremented.

- At the end of the quiz, the user's score is displayed.
- The function loads existing quiz results and updates the result for the current username.

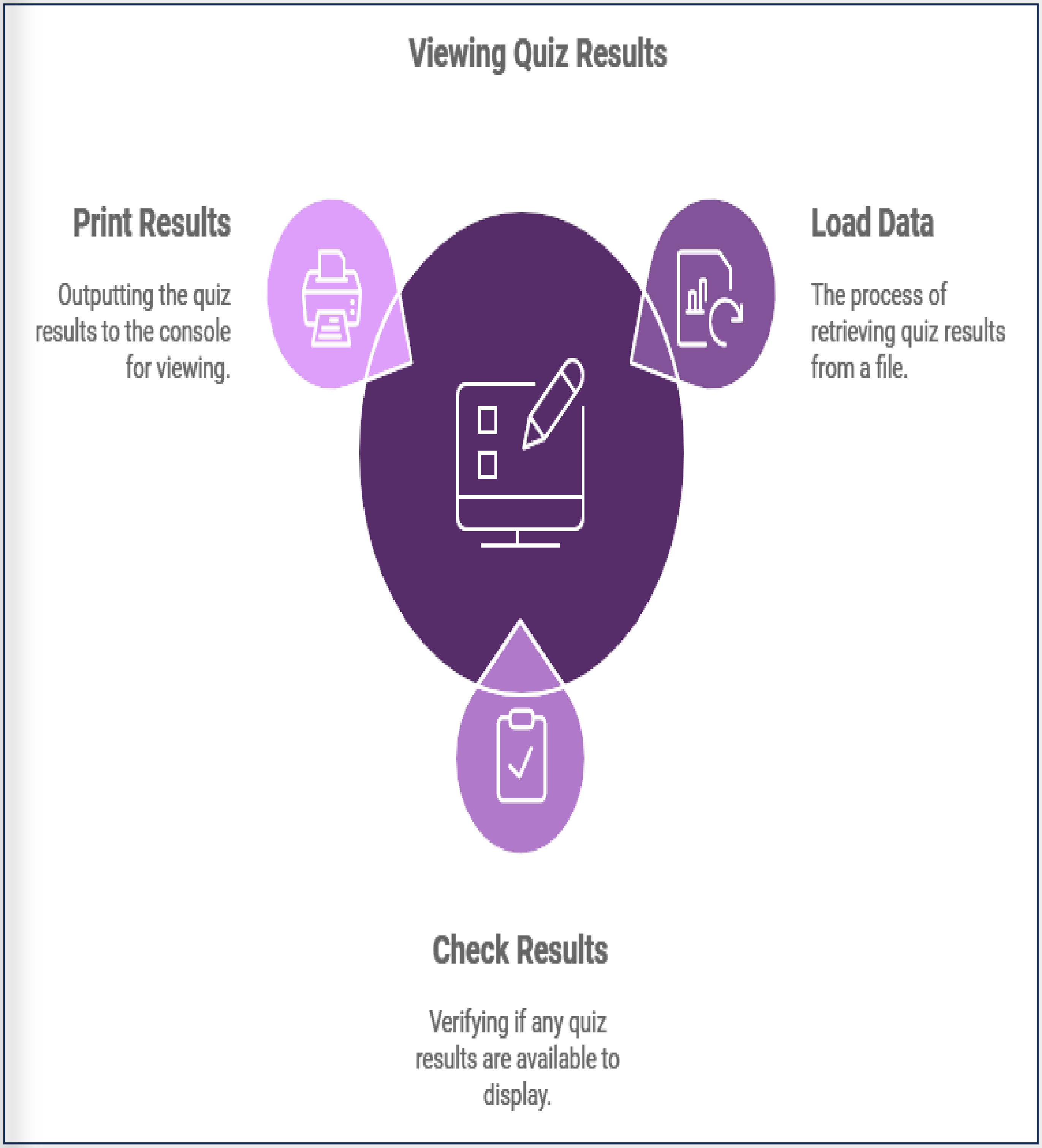
- The results include the score, total number of questions, and timestamp of quiz completion.


Quiz Taking Process
Display Quizzes
Quizzes are shown by topic and creator
Enter Quiz ID
User inputs the ID of the chosen quiz
Enter Password
User enters the password to access the quiz
Answer Questions
User answers the quiz questions
Save Results
User's score is saved in the system


Quiz Taking Process
Load Quizzes
Quizzes are loaded from a file.
Display Quizzes
Available quizzes are shown to the user.
Select Quiz
User chooses a quiz by ID.
Enter Password
User inputs the quiz password.
Start Quiz
The quiz begins with a welcome message.
Answer Questions
User answers quiz questions.

9 View Results Function

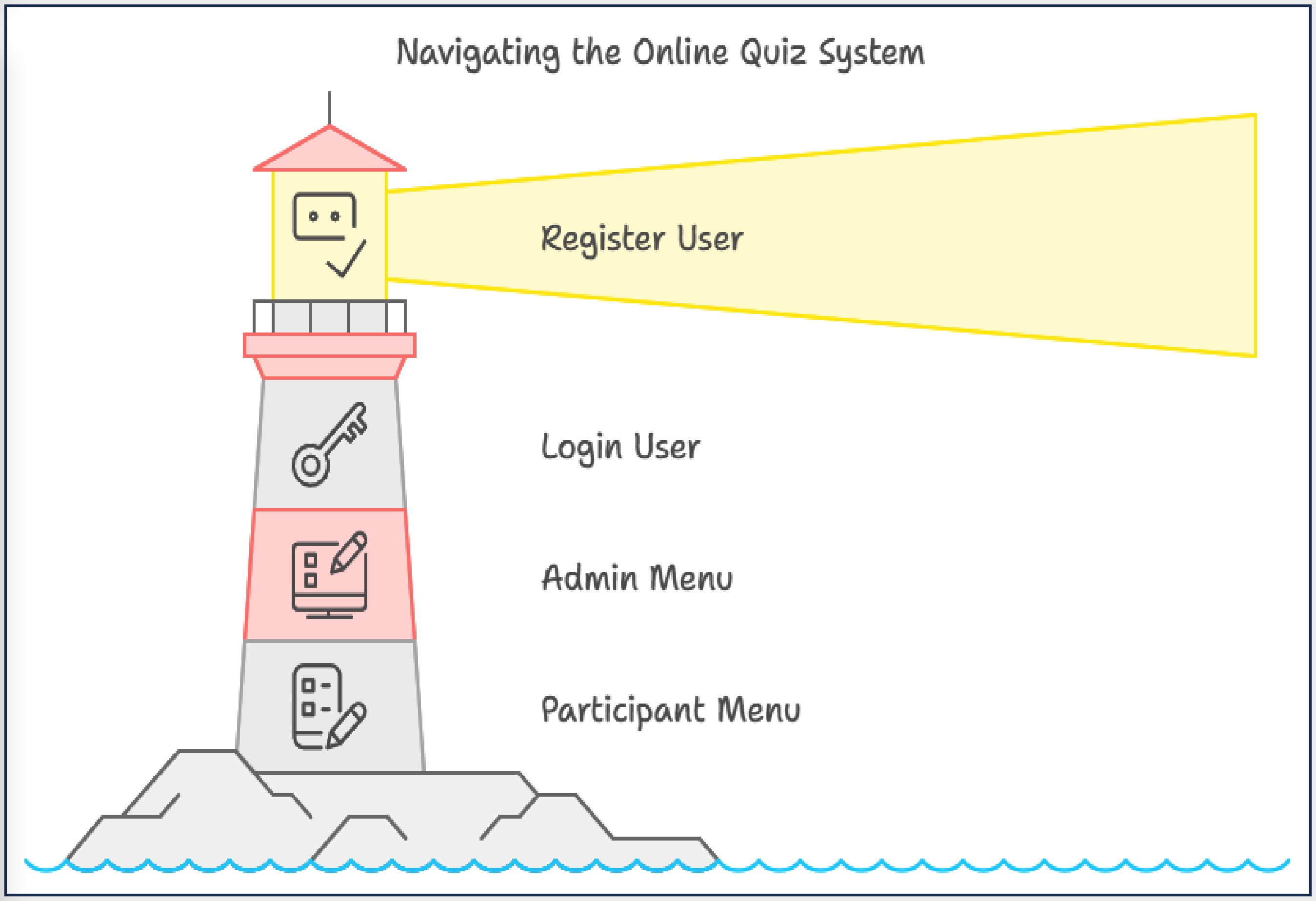
- The view_results function displays quiz results for each participant.
- The function loads quiz results from RESULTS_FILE to access previously saved scores and related information.
- For each participant (username), their results are displayed under "Participant: <username>."
- Each quiz result includes:
  - ➢ quiz_id: the unique identifier of the quiz.
  - ➢ score and total_questions: the participant's score and the total number of questions.
  - ➢ taken_at: the timestamp of when the quiz was taken.
- This structured display allows easy viewing of individual quiz performance for each user.
- The function provides an organized way to review participants' performance across multiple quizzes, making it useful for tracking and evaluating quiz outcomes.


Viewing Quiz Results
Print Results
Outputting the quiz results to the console for viewing.
Load Data
The process of retrieving quiz results from a file.
Check Results
Verifying if any quiz results are available to display.

10 Main Function

- The main function is the primary control loop of the Online Quiz System, handling registration, login, and navigation for different user roles.

- This program loop provides a smooth experience for both admin and participant users, with distinct menu paths tailored to each role's functionality.



Python code

```
import hashlib
import json
import os
import time
import random
from datetime import datetime
```

```
# Load data from JSON files
def load_data(file_path):
    try:
        with open(file_path, 'r') as file:
            data = json.load(file)
            print(f"Successfully loaded data from {file_path}")
            return data
    except json.JSONDecodeError:
        print(f"Error decoding JSON from {file_path}. Check
JSON format.")
        return {}
    except FileNotFoundError:
        print(f"File {file_path} not found.")
        return {}
```

```
# File paths
USERS_FILE = 'users.json'
QUIZZES_FILE = 'quiz_questions.json'
RESULTS_FILE = 'results.json'
```

```
def save_data(file_path, data):
    with open(file_path, 'w') as file:
        json.dump(data, file, indent=4)
```

```
# Hash password
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

```
# Send verification code (simulate sending)
def send_verification_code(email, code):
    print(f"Verification code sent to {email}@gmail.com:
{code}")
```

```
def register_user(username, password, role):
    users = load_data(USERS_FILE)
    if username in users:
        print("Username already exists.")
        return False
    verification_code = str(random.randint(100000,
999999))
    email = input("Enter your email address for verification:
")
    send_verification_code(email, verification_code)
    user_input_code = input("Please enter the verification
code sent to your email: ")
    if user_input_code == verification_code:
        print("Verification successful!")
        users[username] = {
            'password': hash_password(password),
            'role': role}
        save_data(USERS_FILE, users)
        print(f"User '{username}' registered successfully as
{role}.")
        return True
    else:
        print("Invalid verification code. Registration failed.")
```

```
# Login user
def login_user(username, password):
    users = load_data(USERS_FILE)
    if username in users and users[username]['password']
== hash_password(password):
        print(f"Welcome, {username}!")
        return users[username]['role']
    else:
        print("Invalid credentials.")
        return None
```

```
# Create a new quiz (admin only)
def create_quiz(admin_username):
    quizzes = load_data(QUIZZES_FILE)
    question_bank = load_data('question_bank.json')  #
Assuming you have a separate question bank JSON
    quiz_id = f"quiz_{int(time.time())}"  # Unique quiz ID
based on current timestamp
    # Prompt for quiz details
    topic = input("Enter the quiz topic: ")
    password = input("Set a password for this quiz: ")
    num_questions = int(input("How many questions do you
want in this quiz? "))
```

```
# Validate number of questions
    if num_questions > len(question_bank):
        print(f"You can only select up to {len(question_bank)}
questions from the question bank.")
        return

    # Randomly select questions from the question bank
    selected_questions = random.sample(question_bank,
num_questions)

    # Structure the quiz data
    quizzes[quiz_id] = {
        "admin": admin_username,
        "topic": topic,
        "password": password,
        "questions": selected_questions,
        "created_at": datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    }

    # Save the quiz data to quiz_questions.json
    save_data(QUIZZES_FILE, quizzes)
    print(f"Quiz '{quiz_id}' created successfully.")
```

```
# Take a quiz
def take_quiz(username):
    quizzes = load_data(QUIZZES_FILE)
    if not quizzes:
        print("No quizzes available.")
        return

    # Display all available quizzes grouped by topic
    print("\nAvailable Quizzes by Topic:")
    for quiz_id, quiz in quizzes.items():
        # Check if 'topic' and 'admin' keys exist before
accessing them
        topic = quiz.get('topic', 'N/A')  # Use 'N/A' if 'topic' is
missing
        admin = quiz.get('admin', 'N/A')  # Use 'N/A' if 'admin'
is missing
        print(f"- ID: {quiz_id}, Topic: {topic}, Created by:
{admin}")

    # Ask user to choose a quiz by ID
    quiz_id = input("\nEnter the quiz ID you want to take:
").strip()
```

```
if quiz_id not in quizzes:
        print("Quiz not found.")
        return
    quiz = quizzes[quiz_id]
    quiz_password = quiz["password"]
    password_input = input("Enter the password for this
quiz: ")
    if password_input != quiz_password:
        print("Incorrect password. Access denied.")
        return
    score = 0
    print(f"\nStarting quiz '{quiz_id}'...\n")
    for idx, q in enumerate(quiz["questions"], 1):
        print(f"Q{idx}: {q['question']}")
        for i, option in enumerate(q['options'], 1):
            print(f"  {i}. {option}")
        answer = input("Your answer (1-4): ")
        if answer == q['answer']:
            score += 1
    print(f"\nYou completed the quiz with a score:
{score}/{len(quiz['questions'])}.")

    # Save the result
    results = load_data(RESULTS_FILE)
```

```
results[username] = results.get(username, {})
    results[username][quiz_id] = {
        "score": score,
        "total_questions": len(quiz['questions']),
        "taken_at": datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    }
    save_data(RESULTS_FILE, results)
    print("Your result has been saved.")
```

```
def view_results():
    results = load_data(RESULTS_FILE)
    if not results:
        print("No results available.")
        return
    print("\nQuiz Results:")
    for username, user_results in results.items():
        print(f"\nParticipant: {username}")
        for quiz_id, quiz_result in user_results.items():
            score = quiz_result["score"]
            total_questions = quiz_result["total_questions"]
            taken_at = quiz_result["taken_at"]
```

```
print(f"  - Quiz ID: {quiz_id}")
            print(f"    Score: {score}/{total_questions}")
            print(f"    Taken at: {taken_at}")
```

```
# Main program loop
def main():
    print("Welcome to the Online Quiz System")
    while True:
        print("1. Register")
        print("2. Login")
        print("3. Quit")
        choice = input("Choose an option: ")
        if choice == '1':
            username = input("Enter username: ")
            print(f"you will enter username : {username} ")
            password = input("Enter password: ")
            print("Do you want to see password, please enter
'yes' or 'No'")
            choice=input().lower()
            if choice == 'yes':
                print(f"you enter the password : {password}")
            elif choice == 'no':
```

```
elif choice == 'no':
            print("I will display password hashed to secure")
            print(f"You entered password: {'*' *
len(password)}")

         role = input("Enter role (admin/participant):
").lower()
         register_user(username, password, role)

      elif choice == '2':
         username = input("Enter username: ")
         password = input("Enter password: ")
         role = login_user(username, password)

         if role == 'admin':
            print("Admin Menu")
            while True:
               print("1. Create Quiz")
               print("2. View Results")
               print("3. Logout")

               admin_choice = input("Choose an option: ")
```

```
if admin_choice == '1':
                create_quiz(username)
            elif admin_choice == '2':
                view_results()
            elif admin_choice == '3':
                break
            else:
                print("Invalid choice.")
    elif role == 'participant':
        print("Participant Menu")
        while True:
            print("1. Take Quiz")
            print("2. Logout")
            participant_choice = input("Choose an option:
            if participant_choice == '1':
                take_quiz(username)
            elif participant_choice == '2':
                break
            else:
                print("Invalid choice.")
    else:
        print("Login failed.")
```

```
elif choice == '3':
            print("Goodbye!")
            break
        else:
            print("Invalid choice.")

# Run the main program
main()
```

Thank You