4. Modularity and Reusability

The Alzheimer's Disease Testing Software is designed with modularity and reusability in mind. Below are the eight modules, each with inline comments, docstrings, and examples showcasing coupling and cohesion principles.

1. Database Module

python
```
import mysql.connector
from mysql.connector import Error

class Database:
    """Handles database connections and operations."""

    @staticmethod
    def connect_db():
        """
        Establishes a connection to the MySQL database.
        Returns:
            connection: A MySQL connection object or None if connection fails.
        """

        try:
            connection = mysql.connector.connect(
                host='localhost',
                database='alzheimer_testing',
                user='root',
                password='your_password'
            )
            if connection.is_connected():
                return connection
        except Error as e:
            print("Error while connecting to MySQL", e)

        return None
```

2. User Model Module

python
```
from database import Database  # Ensure this line is present


# user_model.py
class UserModel:
    """Represents a user in the system."""
    def __init__(self, user_id=None, username=None, name=None, password=None, email=None,
contact_info=None, date_of_birth=None, gender=None):
        """
        Initializes a user_model instance.

        Args:
            user_id: Unique identifier for the user.
            username: Username of the user.
            name: Full name of the user.
            password: User's password (should be hashed).
            email: User's email address.
            contact_info: User's contact information.
        """
        self.user_id = user_id
        self.username = username  # Added username field
        self.name = name
        self.password = password  # Ensure this is stored securely
        self.email = email
        self.contact_info = contact_info
        self.date_of_birth = date_of_birth
        self.gender = gender
    def register(self):
        """Registers a new user in the database."""
        connection = Database.connect_db()
        if connection:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO USER (username, password, email, contact_info)
VALUES (%s, %s, %s, %s)",
                           (self.name, 'default_password', f'{self.name}@example.com',
self.contact_info))
            connection.commit()
            self.user_id = cursor.lastrowid
            cursor.close()
            connection.close()


    @staticmethod
    def fetch_user_by_username(username):
        """
    Fetches a user from the database by their username
    Args:
        username (str): The username of the user to be fetched.
    Returns:
        UserModel or None: Returns a UserModel instance if found, otherwise None.
    """

        connection = Database.connect_db()
        user = None
        if connection:
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM USER WHERE username = %s", (username,))
            user_data = cursor.fetchone()
            if user_data:
                user = UserModel(
                    user_id=user_data[0],
                    username=user_data[1],
                    name=user_data[2],
                    password=user_data[3],  # Assuming password is stored in the database
                    email=user_data[4],
                    contact_info=user_data[5],
                    date_of_birth=user_data[6],
                    gender=user_data[7]
                )
            cursor.close()
            connection.close()
        return user
```

3. Login ViewModel Module

python
```
from user_model import UserModel

# login_view_model.py
class LoginViewModel:
    """Handles user login logic."""

    def __init__(self):
        self.user = None

    def login_user(self, username, password):
        """
        Authenticates a user based on username and password.

        Args:
            username: The username of the user.
            password: The password of the user.

        Returns:
            bool: True if login is successful, False otherwise.
        """

        self.user = UserModel.fetch_user_by_username(username)  # Implement this method
in UserModel
        if self.user and self.user.password == password:  # Assuming password is stored
securely
            return True
        return False
```

4. Cognitive Test Module

python
```
class CognitiveTest:
     """Represents a user in the system."""
     def __init__(self, test_id=None, user_id=None, test_type=None, score=None,
date_taken=None):
         """
        Initializes a cognitive_test instance.

        Args:
            test_id: Unique identifier for the test.
            user_id: ID of the user taking the test.
            test_type: Type of cognitive test.
            score: Score obtained in the test.
        """

         self.test_id = test_id
         self.user_id = user_id
         self.test_type = test_type
         self.score = score
         self.date_taken = date_taken
     def administer_test(self):
        print("Administering cognitive test...")
        self.score = 95.0  # Placeholder score

        print("Cognitive test administered")
```

5. Genetic Data Module

python
```
class GeneticData:
    """Represents genetic data collected from a user."""

    def __init__(self, genetic_data_id=None, user_id=None, collection_date=None,
genetic_markers=None):
        """
        Initializes a genetic_data instance.

        Args:
            genetic_data_id: Unique identifier for the genetic data.
            user_id: ID of the user.
            genetic_markers: Genetic markers collected.
        """
        self.genetic_data_id = genetic_data_id
        self.user_id = user_id
        self.collection_date = collection_date
        self.genetic_markers = genetic_markers

    def collect_data(self):
        """Simulates collecting genetic data."""
        print("Collecting genetic data...")
        self.genetic_markers = "APOE4"
        print("Genetic data collected")
```

6. Lifestyle Data Module

python
```
class LifestyleData:
    """Represents lifestyle data collected from a user."""
    def __init__(self, lifestyle_data_id=None, user_id=None, collection_date=None,
diet_info=None, exercise_info=None, sleep_info=None):
        """
        Initializes a lifestyle_data instance.

        Args:
            lifestyle_data_id: Unique identifier for the lifestyle data.
            user_id: ID of the user.
            diet_info: Information about the user's diet.
            exercise_info: Information about the user's exercise habits.
        """

        self.lifestyle_data_id = lifestyle_data_id
        self.user_id = user_id
        self.collection_date = collection_date
        self.diet_info = diet_info
        self.exercise_info = exercise_info
        self.sleep_info = sleep_info

    def collect_data(self):
        """Simulates collecting lifestyle data."""
        print("Collecting lifestyle data...")
        self.diet_info = "vegetarian"
        self.exercise_info = "daily running"
        self.sleep_info = "7 hours"
        print("Lifestyle data collected")
```

7. User View Module

python

```
import tkinter as tk
from tkinter import messagebox
from user_view_model import UserViewModel

# user_view.py
class UserView:
     """Handles the user interface for user interactions."""
     def __init__(self, view_model):
         """
        Initializes the user_view instance.
        Args:
            view_model: The view model that handles user logic.
        """

         self.view_model = view_model
         self.root = tk.Tk()
         self.root.title("Alzheimer's Disease Testing Software")

         self.frame = tk.Frame(self.root)
         self.frame.pack(pady=20)

        # User Registration
         tk.Label(self.frame, text="Name").grid(row=0, column=0, padx=10, pady=5)
         self.name_entry = tk.Entry(self.frame)
         self.name_entry.grid(row=0, column=1, padx=10, pady=5)

         tk.Label(self.frame, text="Date of Birth").grid(row=1, column=0, padx=10,
pady=5)
         self.dob_entry = tk.Entry(self.frame)
         self.dob_entry.grid(row=1, column=1, padx=10, pady=5)

         tk.Label(self.frame, text="Gender").grid(row=2, column=0, padx=10, pady=5)
         self.gender_entry = tk.Entry(self.frame)
         self.gender_entry.grid(row=2, column=1, padx=10, pady=5)

         tk.Label(self.frame, text="Contact Info").grid(row=3, column=0, padx=10, pady=5)
         self.contact_entry = tk.Entry(self.frame)
         self.contact_entry.grid(row=3, column=1, padx=10, pady=5)

         tk.Button(self.frame, text="Register", command=self.register_user).grid(row=4,
columnspan=2, pady=10)

        # Cognitive Test
```

```
         tk.Label(self.frame, text="Cognitive Test Type").grid(row=5, column=0, padx=10,
pady=5)
         self.test_type_entry = tk.Entry(self.frame)
         self.test_type_entry.grid(row=5, column=1, padx=10, pady=5)

         tk.Label(self.frame, text="Date Taken").grid(row=6, column=0, padx=10, pady=5)
         self.date_taken_entry = tk.Entry(self.frame)
         self.date_taken_entry.grid(row=6, column=1, padx=10, pady=5)

         tk.Button(self.frame, text="Submit Cognitive Test",
command=self.submit_cognitive_test).grid(row=7, columnspan=2, pady=10)

        # Additional UI elements for genetic and lifestyle data...

     def register_user(self):
         """Handles user registration."""
         name = self.name_entry.get()
         dob = self.dob_entry.get()
         gender = self.gender_entry.get()
         contact = self.contact_entry.get()
         self.view_model.register_user(name, dob, gender, contact)
         messagebox.showinfo("Registration", "User registered successfully!")

     def submit_cognitive_test(self):
        """Handles the submission of cognitive test data."""
        test_type = self.test_type_entry.get()
        date_taken = self.date_taken_entry.get()
        self.view_model.submit_cognitive_test(test_type, date_taken)
        messagebox.showinfo("Cognitive Test", "Cognitive test submitted successfully!")

     def start(self):
         """Starts the Tkinter main loop."""

         self.root.mainloop()
```

8. AlzheimerApp Module

python
```
import tkinter as tk
from tkinter import messagebox
from PIL import ImageTk, Image
from login_view_model import LoginViewModel
from genetic_data import GeneticData
from lifestyle_data import LifestyleData
from cognitive_test import CognitiveTest

# alzheimer_app.py
class AlzheimerApp:
    def __init__(self, root):
        """
        Initializes the alzheimer_app instance.
        Args:
            root: The main Tkinter window.
        """

        self.root = root
        self.root.title("Alzheimer's Disease Testing Software")
        self.login_view_model = LoginViewModel()  # Initialize the LoginViewModel
        self.show_login()

        # Initialize data objects
        self.genetic_data = GeneticData()
        self.lifestyle_data = LifestyleData()
        self.cognitive_test = CognitiveTest()

    def show_login(self):
        """Displays the login interface for the user."""
        self.clear_frame()
        self.login_frame = tk.Frame(self.root)
        self.login_frame.pack(pady=60)

        tk.Label(self.login_frame, text="Username").grid(row=0, column=0, padx=20,
pady=5)
        self.username_entry = tk.Entry(self.login_frame)
        self.username_entry.grid(row=0, column=1, padx=20, pady=5)

        tk.Label(self.login_frame, text="Password").grid(row=1, column=0, padx=20,
pady=5)
        self.password_entry = tk.Entry(self.login_frame, show="*")
        self.password_entry.grid(row=1, column=1, padx=20, pady=5)

        tk.Button(self.login_frame, text="Login", command=self.login).grid(row=2,
columnspan=2, pady=10)

    def login(self):
        """Handles user login."""
        username = self.username_entry.get()
        password = self.password_entry.get()

        if self.login_view_model.login_user(username, password):
            self.show_dashboard()
        else:
            messagebox.showerror("Login Error", "Invalid username or password")

    def show_dashboard(self):
        """Displays the dashboard after successful login."""
        self.clear_frame()
        self.dashboard_frame = tk.Frame(self.root)
        self.dashboard_frame.pack(pady=20)

        profile_img = Image.open("profile_pic.png")
        profile_img = profile_img.resize((100, 100), Image.ANTIALIAS)
        profile_img = ImageTk.PhotoImage(profile_img)
        tk.Label(self.dashboard_frame, image=profile_img).grid(row=0, columnspan=2)
        tk.Label(self.dashboard_frame, text="Welcome, Josh!").grid(row=1, columnspan=2,
pady=10)

        self.profile_img = profile_img  # Keep reference to avoid garbage collection

        self.nav_frame = tk.Frame(self.dashboard_frame)
        self.nav_frame.grid(row=2, columnspan=2, pady=10)

        tk.Button(self.nav_frame, text="Cognitive Tests",
command=self.show_cognitive_tests).grid(row=0, column=0, padx=10)
```

```
        tk.Button(self.nav_frame, text="Lifestyle Data",
command=self.show_lifestyle_data).grid(row=0, column=1, padx=10)
        tk.Button(self.nav_frame, text="Genetic Data",
command=self.show_genetic_data).grid(row=0, column=2, padx=10)
        tk.Button(self.nav_frame, text="View Results",
command=self.show_view_results).grid(row=0, column=3, padx=10)
        tk.Button(self.nav_frame, text="Logout", command=self.logout).grid(row=0,
column=4, padx=10)  # Logout button

    def logout(self):
        """Logs out the user and returns to the login screen."""
        self.clear_frame()
        self.show_login()  # Show the login screen again

    def show_cognitive_tests(self):
        """Displays the cognitive tests interface and shows the score."""
        self.clear_frame()
        self.cognitive_test.administer_test()  # Simulate administering the test
        tk.Label(self.root, text="Cognitive Test Score:
{}".format(self.cognitive_test.score)).pack(pady=20)
        tk.Button(self.root, text="Back to Dashboard",
command=self.show_dashboard).pack(pady=10)

    def show_lifestyle_data(self):
        """Displays the lifestyle data collected from the user."""
        self.clear_frame()
        self.lifestyle_data.collect_data()  # Simulate data collection
        tk.Label(self.root, text="Lifestyle Data:").pack(pady=20)
        tk.Label(self.root, text=f"Diet: {self.lifestyle_data.diet_info}, Exercise:
{self.lifestyle_data.exercise_info}, Sleep:
{self.lifestyle_data.sleep_info}").pack(pady=10)
        tk.Button(self.root, text="Back to Dashboard",
command=self.show_dashboard).pack(pady=10)

    def show_genetic_data(self):
        """Displays the genetic data collected from the user."""
        self.clear_frame()
        self.genetic_data.collect_data()  # Simulate data collection
        tk.Label(self.root, text="Genetic Markers:
{}".format(self.genetic_data.genetic_markers)).pack(pady=20)
        tk.Button(self.root, text="Back to Dashboard",
command=self.show_dashboard).pack(pady=10)

    def show_view_results(self):
        """Displays the results page with options to view submitted data."""
        self.clear_frame()
        tk.Label(self.root, text="Results Page").pack(pady=20)
        tk.Button(self.root, text="Lifestyle Data Submitted", command=lambda:
self.show_result_message("Lifestyle data is submitted")).pack(pady=5)
        tk.Button(self.root, text="Cognitive Data Submitted", command=lambda:
self.show_result_message("Cognitive data is submitted")).pack(pady=5)
        tk.Button(self.root, text="Genetic Data Submitted", command=lambda:
self.show_result_message("Genetic data is submitted")).pack(pady=5)
        tk.Button(self.root, text="Back to Dashboard",
command=self.show_dashboard).pack(pady=10)

    def show_result_message(self, message):
        """Displays a message box with the result message."""
```

```python
        messagebox.showinfo("Result", message)

    def clear_frame(self):
        """Clears the current frame."""
        for widget in self.root.winfo_children():
            widget.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = AlzheimerApp(root)
    root.mainloop()
```