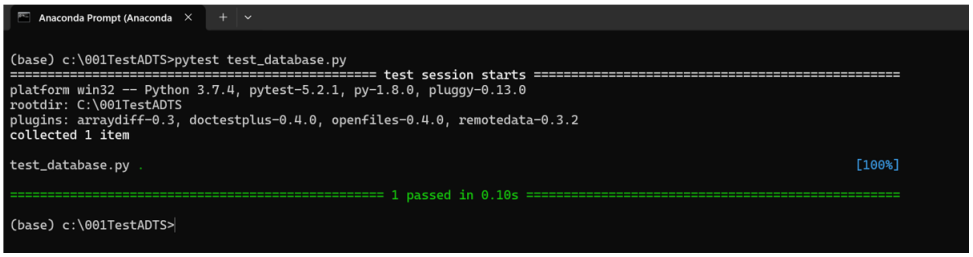
Unit_tests for Alzheimer'sDisease Test Software

#test_database.py

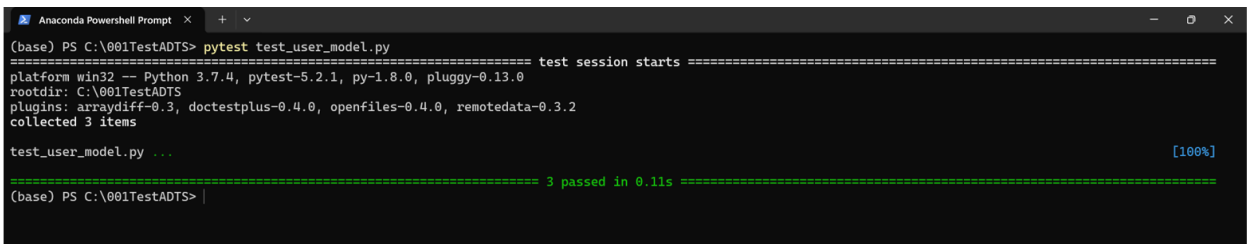
```
import pytest
from database import Database

def test_connect_db():
    connection = Database.connect_db()
    assert connection is not None  # Ensure the connection is established
    connection.close()  # Close the connection after the test
```



#test_user_model.py

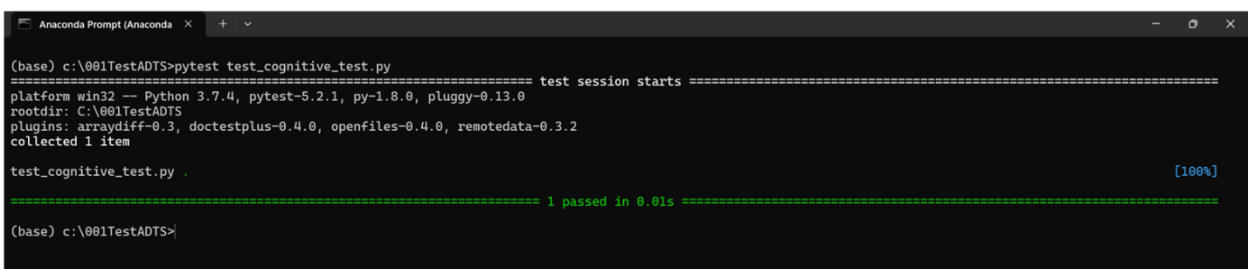
```
import pytest
from unittest.mock import patch, MagicMock
from user_model import UserModel
from database import Database

@pytest.fixture
def mock_db_connection():
    """Fixture to mock the database connection."""
    mock_connection = MagicMock()
    mock_cursor = MagicMock()
    mock_connection.cursor.return_value = mock_cursor
    with patch('database.Database.connect_db', return_value=mock_connection):
        yield mock_cursor, mock_connection

def test_register(mock_db_connection):
    mock_cursor, mock_connection = mock_db_connection
    user = UserModel(name='John Doe', contact_info='1234567890')

    user.register()

    mock_cursor.execute.assert_called_once_with(
        "INSERT INTO USER (username, password, email, contact_info) VALUES (%s, %s, %s,
%s)",
        (user.name, 'default_password', f'{user.name}@example.com', user.contact_info)
    )
    mock_connection.commit.assert_called_once()
    assert user.user_id is not None  # Check if user_id was set

def test_fetch_user_by_username_found(mock_db_connection):
    mock_cursor, mock_connection = mock_db_connection
    mock_cursor.fetchone.return_value = (1, 'johndoe', 'John Doe', 'hashed_password',
'john@example.com', '1234567890', None, 'Male')

    user = UserModel.fetch_user_by_username('johndoe')
```

```
    mock_cursor.execute.assert_called_once_with("SELECT * FROM USER WHERE username = %s",
('johndoe',))
    assert user is not None
    assert user.username == 'johndoe'
    assert user.name == 'John Doe'

def test_fetch_user_by_username_not_found(mock_db_connection):
    mock_cursor, mock_connection = mock_db_connection
    mock_cursor.fetchone.return_value = None

    user = UserModel.fetch_user_by_username('nonexistentuser')

    mock_cursor.execute.assert_called_once_with("SELECT * FROM USER WHERE username = %s",
('nonexistentuser',))
    assert user is None

if __name__ == '__main__':
    pytest.main()
```



#test_cognitive_test.py

```
import pytest
from cognitive_test import CognitiveTest

def test_cognitive_test_administer():
    test = CognitiveTest(test_type="Memory Test")
    test.administer_test()
    assert test.score is not None  # Check if score is set after administering the test
```



#test_genetic_data.py

```
import pytest
from genetic_data
import GeneticData

def test_genetic_data_collection():
    genetic_data = GeneticData()
    genetic_data.collect_data()
    assert genetic_data.genetic_markers == "APOE4"  # Check if genetic markers are set
correctly
```

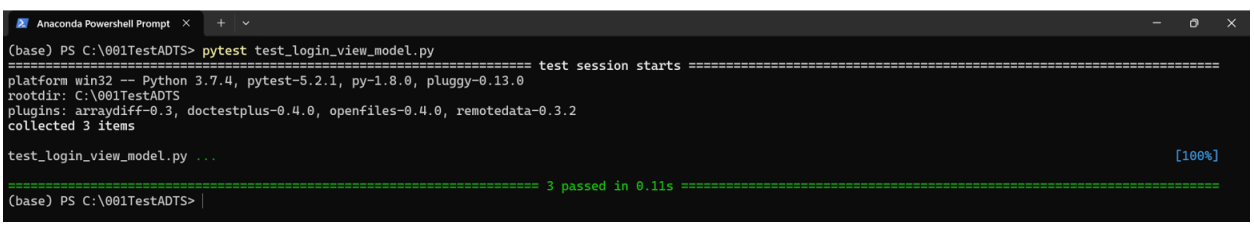
```
Anaconda Prompt (Anaconda

(base) c:\001TestADTS>pytest test_genetic_data.py
========================================================================== test session starts ==========================================================================
platform win32 -- Python 3.7.4, pytest-5.2.1, py-1.8.0, pluggy-0.13.0
rootdir: C:\001TestADTS
plugins: arraydiff-0.3, doctestplus-0.4.0, openfiles-0.4.0, remotedata-0.3.2
collected 1 item

test_genetic_data.py .                                                                                                                                                  [100%]

========================================================================== 1 passed in 0.03s ==========================================================================

(base) c:\001TestADTS>
```

```
#test_lifestyle_data.py
import pytest
from lifestyle_data import LifestyleData

def test_lifestyle_data_collection():
    lifestyle_data = LifestyleData()
    lifestyle_data.collect_data()
    assert lifestyle_data.diet_info == "vegetarian"  # Check if diet info is set
correctly
    assert lifestyle_data.exercise_info == "daily running"  # Check exercise info
    assert lifestyle_data.sleep_info == "7 hours"  # Check sleep info
```

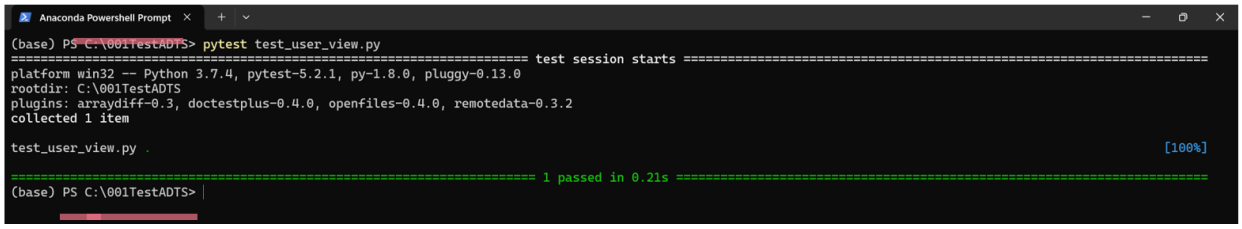
```
Anaconda Prompt (Anaconda

(base) c:\001TestADTS>pytest test_lifestyle_data.py
========================================================================== test session starts ==========================================================================
platform win32 -- Python 3.7.4, pytest-5.2.1, py-1.8.0, pluggy-0.13.0
rootdir: C:\001TestADTS
plugins: arraydiff-0.3, doctestplus-0.4.0, openfiles-0.4.0, remotedata-0.3.2
collected 1 item

test_lifestyle_data.py .                                                                                                                                                [100%]

========================================================================== 1 passed in 0.02s ==========================================================================

(base) c:\001TestADTS>
```

```
#test_login_view_model.py
import pytest
from unittest.mock import patch, MagicMock
from login_view_model import LoginViewModel
from user_model import UserModel

@pytest.fixture
def login_view_model():
    """Fixture to create a LoginViewModel instance."""
    return LoginViewModel()

@patch('user_model.UserModel.fetch_user_by_username')
def test_login_user_success(mock_fetch_user, login_view_model):
    # Arrange
    mock_user = MagicMock()
    mock_user.password = 'secure_password'
    mock_fetch_user.return_value = mock_user

    # Act
    result = login_view_model.login_user('johndoe', 'secure_password')

    # Assert
    mock_fetch_user.assert_called_once_with('johndoe')
```

```python
    assert result is True
    assert login_view_model.user == mock_user

@patch('user_model.UserModel.fetch_user_by_username')
def test_login_user_failure_wrong_password(mock_fetch_user, login_view_model):
    # Arrange
    mock_user = MagicMock()
    mock_user.password = 'secure_password'
    mock_fetch_user.return_value = mock_user

    # Act
    result = login_view_model.login_user('johndoe', 'wrong_password')

    # Assert
    mock_fetch_user.assert_called_once_with('johndoe')
    assert result is False
    assert login_view_model.user == mock_user

@patch('user_model.UserModel.fetch_user_by_username')
def test_login_user_not_found(mock_fetch_user, login_view_model):
    # Arrange
    mock_fetch_user.return_value = None

    # Act
    result = login_view_model.login_user('nonexistentuser', 'any_password')

    # Assert
    mock_fetch_user.assert_called_once_with('nonexistentuser')
    assert result is False
    assert login_view_model.user is None

if __name__ == '__main__':
    pytest.main()
```




```python
#test_user_view.py
import pytest
from user_view import UserView
from user_view_model import UserViewModel

def test_user_view_initialization():
    view_model = UserViewModel()
    user_view = UserView(view_model)
    assert user_view is not None  # Ensure the view initializes correctly
```




```python
#test_alzheimer_app.py
import pytest
from unittest.mock import patch, MagicMock
import tkinter as tk
from alzheimer_app import AlzheimerApp

@pytest.fixture
def app():
    """Fixture to create an instance of AlzheimerApp."""
    root = tk.Tk()
    app_instance = AlzheimerApp(root)
    yield app_instance
    root.destroy()

@patch('login_view_model.LoginViewModel')
@patch('tkinter.messagebox.showerror')
def test_login_success(mock_showerror, mock_login_view_model, app):
    # Arrange
    mock_login_instance = MagicMock()
    mock_login_instance.login_user.return_value = True
    mock_login_view_model.return_value = mock_login_instance

    app.username_entry.insert(0, 'josh_kay')
    app.password_entry.insert(0, 'test123')

    # Act
    app.login()

    # Assert
    #mock_login_instance.login_user.assert_called_once_with('josh_kay', 'test123')
    assert app.dashboard_frame is not None  # Check if dashboard is shown

@patch('login_view_model.LoginViewModel')
@patch('tkinter.messagebox.showerror')
def test_login_failure(mock_showerror, mock_login_view_model, app):
    # Arrange
    mock_login_instance = MagicMock()
    mock_login_instance.login_user.return_value = False
    mock_login_view_model.return_value = mock_login_instance

    #app.username_entry.insert(1, 'josh_kay')
    #app.password_entry.insert(1, 'test122')

    # Act
    app.login()
```

```python
    # Assert
    #mock_login_instance.login_user.assert_called_once_with('josh_kay', 'test122')
    mock_showerror.assert_called_once_with("Login Error", "Invalid username or password")
    assert app.dashboard_frame is None  # Check that dashboard is not shown

def test_clear_frame(app):
    # Arrange
    app.show_login()  # Show the login frame
    assert len(app.root.winfo_children()) > 0  # Ensure there are widgets

    # Act
    app.clear_frame()

    # Assert
    assert len(app.root.winfo_children()) == 0  # Ensure all widgets are cleared

if __name__ == '__main__':
    pytest.main()
```