CASE STUDY

ON

VIRTUAL ART GALLERY SYSTEM

TEJASHREE G

R PRAVEEN RAJ

ABSTRACT

The *Virtual Art Gallery System* is a Python-based project that brings the experience of exploring and managing art into a digital environment. Designed with both functionality and simplicity in mind, this application allows users to browse artworks, mark their favorites, and view curated collections, while administrators can manage artists, artworks, galleries, and user data efficiently.

The system is built using object-oriented principles to reflect real-world entities like artists, artworks, galleries, and users. It connects to a MySQL database to store and manage data securely and accurately. Relationships like an artwork being featured in multiple galleries, or a user saving multiple favorites, are handled using well-designed relational tables.

Custom exceptions, clean data validation, and a structured menu-driven interface make the system user-friendly and reliable. Whether you're an art enthusiast looking to explore digital collections or an admin managing the backend, the application offers a smooth and intuitive experience.

This project not only demonstrates backend development skills using Python and SQL but also reflects how thoughtful design can bring real-world systems into the virtual space.

Key Functionalities

Artwork management: The Virtual Art Gallery System aims to provide an immersive and interactive experience for art enthusiasts to explore, view, and appreciate a diverse collection of artworks online.

Personal Galleries: Enable users to create their virtual galleries and curate their collections.

Architecture Overview: The architecture of the Virtual Art Gallery system follows a modular, layered design based on object-oriented programming principles. It separates concerns across different packages for better organization and scalability:

1. Entity Layer (entity/)

Contains Python classes that model real-world objects like Artist, Artwork, User, Gallery, and junction entities like UserFavorite and ArtworkGallery. Each class includes private attributes, constructors, getters, and setters.

2. DAO Layer (dao/)

Includes:

- IVirtualArtGallery – an interface defining all core functionalities (CRUD operations, favorites management, etc.)
- IVirtualArtGalleryImp – the implementation that handles all SQL interactions using mysql.connector.

3. Utility Layer (util/)

Handles database connection setup:

- DBPropertyUtil: Reads DB config from .properties file

- DBConnUtil: Provides a reusable connection object to the MySQL database

4. Exception Layer (exception/)

Defines custom exceptions to manage invalid input, missing records, duplicates, and other domain-specific validation issues.

5. Main Layer (main/)

Hosts the main.py script, which runs a **menu-driven interface** that allows users/admins to interact with the system from the console.

6. Database Layer (MySQL)

Stores all persistent data across normalized tables with foreign keys and many-to-many relationships handled through junction tables (like user_favorite_artwork, artwork_gallery).

SCHEMA DESIGN

Entities:

• Designing the schema for a Virtual Art Gallery involves creating a structured representation of the database that will store information about artworks, artists, users, galleries, and various relationships between them. Below is a schema design for a Virtual Art Gallery database:

• **Entities and Attributes:**

• **Artwork**

ArtworkID (Primary Key)

Title

Description

CreationDate

Medium

ImageURL (or any reference to the digital representation)

• **Artist**

ArtistID (Primary Key)

Name Biography

BirthDate

Nationality

Website

ContactInformation

• **User**

UserID (Primary Key)

Username

Password

Email

First Name

Last Name

Date of Birth

Profile Picture

FavoriteArtworks (a list of references to ArtworkIDs)

- **Gallery**

  GalleryID (Primary Key)

  Name

  Description

  Location

  Curator (Reference to ArtistID)

  OpeningHours

- **Relationships:**
- **Artwork - Artist (Many-to-One)**

  An artwork is created by one artist.

  Artwork.ArtistID (Foreign Key) references Artist.ArtistID.

- **User - Favorite Artwork (Many-to-Many):**

  A user can have many favorite artworks, and an artwork can be a favorite of multiple users.

  User_Favorite_Artwork (junction table):

UserID (Foreign Key) references User.UserID.

ArtworkID (Foreign Key) references Artwork.ArtworkID.

- **Artist - Gallery (One-to-Many):**

An artist can be associated with multiple galleries, but a gallery can have only one curator (artist).

Gallery.ArtistID (Foreign Key) references Artist.ArtistID.

- **Artwork - Gallery (Many-to-Many)**

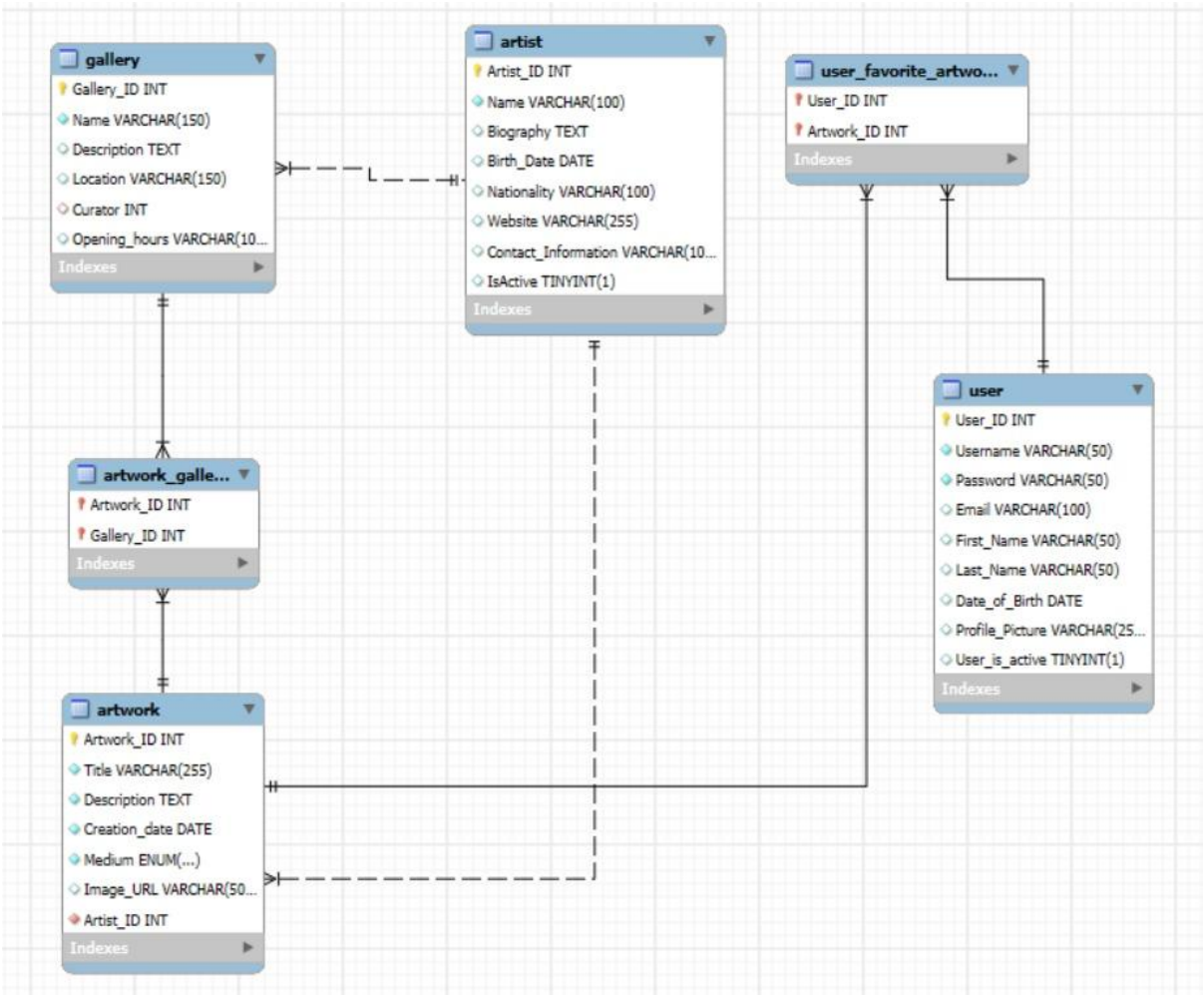
An artwork can be displayed in multiple galleries, and a gallery can have multiple artworks.

Artwork_Gallery (junction table):

ArtworkID (Foreign Key) references Artwork.ArtworkID.

GalleryID (Foreign Key) references Gallery.GalleryID.

ENTITY RELATIONSHIP DIAGRAM:



ENTITY CLASS

- CLASS ARTIST:

```
ass Artist:
  def __init__(self, artist_id = None, name=None, biography=None, birth_date = None, nationality = None, website = None, contact_information = None, is_active = None, password = None):
      self.__artist_id = artist_id
      self.__name = name
      self.__biography = biography
      self.__birth_date = birth_date
      self.__nationality = nationality
      self.__website = website
      self.__contact_information = contact_information
      self.__is_active = is_active
      self.__password = password

  @property
  def artist_id(self):
      return self.__artist_id
  @artist_id.setter
  def artist_id(self, artist_id):
      self.__artist_id = artist_id

  @property
  def name(self):
      return self.__name
  @name.setter
  def name(self, name):
      self.__name = name

  @property
  def biography(self):
      return self.__biography
  @biography.setter
  def biography(self, biography):
      self.__biography = biography

  @property
  def birth_date(self):
      return self.__birth_date
  @birth_date.setter
  def birth_date(self, birth_date):
      self.__birth_date = birth_date

  @property
  def nationality(self):
      return self.__nationality
  @nationality.setter
  def nationality(self, nationality):
      self.__nationality = nationality
```

```
    @property
    def website(self):
        return self.__website
    @website.setter
    def website(self, website):
        self.__website = website

    @property
    def contact_information(self):
        return self.__contact_information
    @contact_information.setter
    def contact_information(self, contact_information):
        self.__contact_information = contact_information

    @property
    def is_active(self):
        return self.__is_active
    @is_active.setter
    def is_active(self, is_active):
        self.__is_active = is_active

    @property
    def password(self):
        return self.__password
    @password.setter
    def password(self, password):
        self.__password = password
```

- CLASS ARTWORK:

```python
class Artwork:
    def __init__(self, artwork_id = None, title = None, description= None, creation_date = None,
                 medium=None,image_url = None, artist_id = None):
        self.__artwork_id = artwork_id
        self.__title = title
        self.__description = description
        self.__creation_date = creation_date
        self.__medium = medium
        self.__image_url = image_url
        self.__artist_id = artist_id

    @property
    def artwork_id(self):
        return self.__artwork_id
    @artwork_id.setter
    def artwork_id(self, artwork_id):
        self.__artwork_id = artwork_id

    @property
    def title(self):
        return self.__title
    @title.setter
    def title(self, title):
        self.__title = title

    @property
    def description(self):
        return self.__description
    @description.setter
    def description(self, description):
        self.__description = description

    @property
    def creation_date(self):
        return self.__creation_date
    @creation_date.setter
    def creation_date(self, creation_date):
        self.__creation_date = creation_date
```

```python
    @property
    def medium(self):
        return self.__medium
    @medium.setter
    def medium(self, medium):
        self.__medium = medium

    @property
    def image_url(self):
        return self.__image_url
    @image_url.setter
    def image_url(self, image_url):
        self.__image_url = image_url

    @property
    def artist_id(self):
        return self.__artist_id
    @artist_id.setter
    def artist_id(self, artist_id):
        self.__artist_id = artist_id
```

- CLASS USER:

```python
class User:
    def __init__(self, user_id=None, username=None, password=None, email=None, first_name=None, last_name=None,
                 date_of_birth=None, profile_picture=None, user_is_active = None):
        self.__user_id = user_id
        self.__username = username
        self.__password = password
        self.__email = email
        self.__first_name = first_name
        self.__last_name = last_name
        self.__date_of_birth = date_of_birth
        self.__profile_picture = profile_picture
        self.__user_is_active = user_is_active

    @property
    def user_id(self):
        return self.__user_id

    @user_id.setter
    def user_id(self, user_id):
        self.__user_id = user_id

    @property
    def username(self):
        return self.__username

    @username.setter
    def username(self, username):
        self.__username = username

    @property
    def password(self):
        return self.__password

    @password.setter
    def password(self, password):
        self.__password = password

    @property
    def email(self):
        return self.__email

    @email.setter
    def email(self, email):
        self.__email = email
```

```python
    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, first_name):
        self.__first_name = first_name

    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
    def last_name(self, last_name):
        self.__last_name = last_name

    @property
    def date_of_birth(self):
        return self.__date_of_birth

    @date_of_birth.setter
    def date_of_birth(self, date_of_birth):
        self.__date_of_birth = date_of_birth

    @property
    def profile_picture(self):
        return self.__profile_picture

    @profile_picture.setter
    def profile_picture(self, profile_picture):
        self.__profile_picture = profile_picture

    @property
    def user_is_active(self):
        return self.__user_is_active
    @user_is_active.setter
    def user_is_active(self, user_is_active):
        self.__user_is_active = user_is_active
```

•CLASS USER FAVOURITE:

```python
class UserFavorite:
    def __init__(self, user_id=None, artwork_id=None):
        self.__user_id = user_id
        self.__artwork_id = artwork_id

    @property
    def user_id(self):
        return self.__user_id

    @user_id.setter
    def user_id(self, user_id):
        self.__user_id = user_id

    @property
    def artwork_id(self):
        return self.__artwork_id

    @artwork_id.setter
    def artwork_id(self, artwork_id):
        self.__artwork_id = artwork_id
```

•CLASS ARTWORK GALLERY:

Virtual_Art_Gallery-main > entity > artwork_gallery.py > ArtworkGallery > gallery_id

```python
class ArtworkGallery:

    def __init__(self, artwork_id: int = None, gallery_id: int = None):
        self.__artwork_id = artwork_id
        self.__gallery_id = gallery_id

    @property
    def artwork_id(self) -> int:
        return self.__artwork_id

    @artwork_id.setter
    def artwork_id(self, artwork_id: int):
        self.__artwork_id = artwork_id

    @property
    def gallery_id(self) -> int:
        return self.__gallery_id

    @gallery_id.setter
    def gallery_id(self, gallery_id: int):
        self.__gallery_id = gallery_id
```

•CLASS GALLERY:

Virtual_Art_Gallery-main > entity > gallery.py > Gallery

```python
class Gallery:
    def __init__(self, gallery_id = None, name= None, description= None, location= None, curator= None, opening_hours= None):
        self.__gallery_id = gallery_id
        self.__name = name
        self.__description = description
        self.__location = location
        self.__curator = curator
        self.__opening_hours = opening_hours

    @property
    def gallery_id(self):
        return self.__gallery_id
    @gallery_id.setter
    def gallery_id(self, gallery_id):
        self.__gallery_id = gallery_id

    @property
    def name(self):
        return self.__name
    @name.setter
    def name(self, name):
        self.__name = name

    @property
    def description(self):
        return self.__description
    @description.setter
    def description(self, description):
        self.__description = description
```

DAO

•Service Provider interface (ARTIST INTERFACE)

```python
from abc import ABC, abstractmethod

class IVirtualArtGallery(ABC):
    """ Artist Interface """
    @abstractmethod
    def add_artist(self, artist):
        pass

    @abstractmethod
    def update_artist(self, artist):
        pass

    @abstractmethod
    def get_artist_by_id(self, artist_id):
        pass
    @abstractmethod
    def remove_artist(self, artist):
        pass

    @abstractmethod
    def reactivate_artist(self, artist):
        pass

    """ Artwork Interface """
    @abstractmethod
    def add_artwork(self, artwork):
        pass

    @abstractmethod
    def update_artwork(self, artwork):
        pass

    @abstractmethod
    def get_artwork_by_id(self, artwork_id):
        pass

    @abstractmethod
    def remove_artwork(self, artwork):
        pass

    @abstractmethod
    def search_artworks(self, artwork):
        pass
```

GALLERY INTERFACE & USER INTERFACE

```
""" Gallery Interface """
@abstractmethod
def add_gallery(self,gallery):
    pass

@abstractmethod
def update_gallery(self, gallery):
    pass

@abstractmethod
def get_gallery_by_id(self, gallery_id):
    pass

@abstractmethod
def remove_gallery(self, gallery):
    pass

@abstractmethod
def search_gallery(self, gallery):
    pass

""" User Interface """
@abstractmethod
def add_user(self, user):
    pass

@abstractmethod
def update_user(self, user):
    pass
@abstractmethod
def get_user_by_id(self, user_id):
    pass

@abstractmethod
def remove_user(self, user):
    pass

@abstractmethod
def reactivate_user(self, user):
    pass
```

USER FAVORITES INTERFACE:

```
""" User Favorites Interface"""
@abstractmethod
def add_artwork_to_favorite(self, user_id, artwork_id):
    pass
def remove_artwork_from_favorite(self, user_id, artwork_id):
    pass
def get_user_favorite_artworks(self, favorite_id):
    pass
```

ARTWORKGALLERY INTERFACE:

```
    """ Artwork Gallery """

    @abstractmethod
    def add_artwork_to_gallery(self, artwork_id, gallery_id):
        pass

    @abstractmethod
    def remove_artwork_from_gallery(self, artwork_id, gallery_id):
        pass

    @abstractmethod
    def get_artworks_by_gallery(self, gallery_id):
        pass
```

- IMPLEMENTATION CLASS

Artist Implementation

```
from datetime import date
import re
from dao.IVirtualArtGallery import IVirtualArtGallery
from entity.artwork import Artwork
from entity.artist import Artist
from entity.gallery import Gallery
from entity.user import User
from util.DBConnUtil import DBConnection
from exception.exceptions import (InvalidIDException, InvalidArtistNameException,  # Artist
                                  InvalidDateException, InvalidWebsiteException,
                                  InvalidArtworkException, InvalidMediumTypeException,  # Artwork
                                  DuplicateArtworkException, ArtworkDoesNotExistException,
                                  InvalidGalleryNameException, InvalidOpeningHoursException,  # Gallery
                                  DuplicateGalleryException, GalleryAssignmentException, GalleryNotFoundException,
                                  InvalidUsernameException, InvalidPasswordException, InvalidEmailException,  # User
                                  DuplicateUserException, UserNotFoundException, FavoriteAlreadyExistsException,
                                  FavoriteNotFoundException)  # FavoriteAlreadyExistsException


class VirtualArtGalleryImp(IVirtualArtGallery):
    def __init__(self, connection=None):
        self.connection = connection or DBConnection.get_connection()
```

```python
    """ Artist Implementation"""

    def validate_artist_fields(self, artist):
        if artist.name is not None and artist.name.strip() == "":
            raise InvalidArtistNameException("Artist Name cannot be empty")

        if artist.birth_date is not None and artist.birth_date > date.today():
            raise InvalidDateException("Birth date cannot be in the future")

        if artist.website is not None and not re.match(r"^https?://[^\s]+$", artist.website):
            raise InvalidWebsiteException("Website must be a valid URL")

    def add_artist(self, artist):
        if artist.artist_id is None:
            raise InvalidIDException("Artist ID is required")
        self.validate_artist_fields(artist)

        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artist where Artist_ID = %s", (artist.artist_id,))
        if cursor.fetchone():
                raise InvalidIDException("Artist ID is already in use")

        query = """
                INSERT INTO Artist
                (Artist_ID, Name, Biography, Birth_Date, Nationality, Website, Contact_Information, IsActive)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                    """
        cursor.execute(query, (artist.artist_id,artist.name,artist.biography or None,artist.birth_date or None,artist.nationality or None,
                               artist.website or None,artist.contact_information or None,artist.is_active))
        self.connection.commit()
        return cursor.rowcount > 0

    def update_artist(self, artist):
        cursor = None
        if artist.artist_id is None:
            raise InvalidIDException("Artist ID is required")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artist where Artist_ID = %s", (artist.artist_id,))
        existing = cursor.fetchone()
        if not existing:
            raise InvalidIDException("Artist with this ID does not exist")

        self.validate_artist_fields(artist)
        query = """
        UPDATE Artist SET
            Name = %s,
            Biography = %s,
            Birth_Date = %s,
            Nationality = %s,
            Website = %s,
            Contact_Information = %s
            WHERE Artist_ID = %s
        """
        cursor.execute(query, (artist.name or existing[1],artist.biography or existing[2],artist.birth_date or existing[3],artist.nationality or existing[4],
        artist.website or existing[5],artist.contact_information or existing[6],artist.artist_id))
        self.connection.commit()
        return cursor.rowcount > 0

    def get_artist_by_id(self, artist_id):
        cursor = None
        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artist where Artist_ID = %s", (artist_id,))
        row = cursor.fetchone()
```

```python
        row = cursor.fetchone()
        if row:
            return Artist(
                artist_id = row[0],
                name = row[1],
                biography = row[2],
                birth_date = row[3],
                nationality = row[4],
                website = row[5],
                contact_information = row[6],
                is_active = row[7]
            )
        else:
            return None

    def remove_artist(self, artist_id):
        if artist_id is None:
            raise InvalidIDException("Artist ID is required for deletion")
        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artist where Artist_ID = %s", (artist_id,))
        if not cursor.fetchone():
            raise InvalidIDException("Artist with this ID does not exist")
        query="UPDATE Artist SET IsActive = FALSE WHERE Artist_ID = %s"
        cursor.execute(query, (artist_id,))
        self.connection.commit()
        return cursor.rowcount > 0

    def reactivate_artist(self, artist_id):
        cursor = self.connection.cursor()
        cursor.execute("SELECT IsActive from artist where Artist_ID = %s", (artist_id,))
        row = cursor.fetchone()
        if not row:
            return "not_found"
        isActive = row[0]
        if isActive:
            return "already_active"
        cursor.execute("UPDATE Artist SET IsActive = TRUE WHERE Artist_ID = %s", (artist_id,))
        self.connection.commit()
        return 'reactivated'
```

Artwork Implementation

```python
    """Artwork Implementation"""
    def validate_artwork_fields(self, artwork):
        if artwork.title is not None and artwork.title.strip() == "":
            raise InvalidArtworkException("Title is required")
        if artwork.description is not None and artwork.description == "":
            raise InvalidArtworkException("Description is required")
        if artwork.creation_date is not None and artwork.creation_date > date.today():
            raise InvalidDateException("Creation date cannot be null or in the future")
        if artwork.medium is not None and artwork.medium not in self.valid_mediums:
            raise InvalidMediumTypeException(f"Medium type cannot be none and it should be one of {', '.join(self.valid_mediums)}")
        if artwork.image_url is not None and not re.match(r"^https?://[^\s]+$", artwork.image_url):
            raise InvalidWebsiteException("Website must be a valid URL")

    valid_mediums = ["Oil on Canvas","Watercolor", "Acrylic", "Digital Art", "Sculpture", "Mixed Media"]
    def add_artwork(self, artwork):
        self.validate_artwork_fields(artwork)
        cursor = self.connection.cursor()
        cursor.execute("SELECT * from Artwork where Artwork_ID = %s", (artwork.artwork_id,))
        if cursor.fetchone():
            raise InvalidIDException("Artwork ID is already in use")

        cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (artwork.artist_id,))
        if not cursor.fetchone():
            print("Artist ID does not exist.")
            return False

        cursor.execute("SELECT * FROM Artwork WHERE Artist_ID = %s AND title = %s", (artwork.artist_id,artwork.title))
        if cursor.fetchone():
            raise DuplicateArtworkException(f"Artwork with title '{artwork.title}' already exists for this artist.")

        cursor.execute("""INSERT INTO Artwork (Artwork_id, Title, Description, Creation_Date, Artist_ID, Medium, Image_url)
        VALUES (%s, %s, %s, %s, %s, %s,%s)""",
        (artwork.artwork_id, artwork.title, artwork.description,
        artwork.creation_date, artwork.artist_id, artwork.medium, artwork.image_url))
        self.connection.commit()
        return cursor.rowcount > 0

    def update_artwork(self, artwork):
        if artwork.artwork_id is None:
            raise InvalidIDException("Artwork ID is required")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork.artwork_id,))
        existing = cursor.fetchone()
        if not existing:
            raise InvalidIDException("Artwork with this ID does not exist")

        self.validate_artwork_fields(artwork)
        query = """
        UPDATE Artwork SET
            Title = %s,
            Description = %s,
            Creation_date = %s,
            Medium = %s,
            Image_URL = %s,
            Artist_ID = %s
        WHERE Artwork_ID = %s
        """
        cursor.execute(query, (artwork.title or existing[1],
            artwork.description or existing[2],
            artwork.creation_date or existing[3],
            artwork.medium or existing[4],
            artwork.image_url or existing[5],
            artwork.artist_id or existing[6],
```

```python
    def get_artwork_by_id(self, artwork_id):
        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork_id,))
        row = cursor.fetchone()
        if row:
            return Artwork(
                artwork_id = row[0],
                title = row[1],
                description = row[2],
                creation_date = row[3],
                medium = row[4],
                image_url = row[5],
                artist_id = row[6]
            )
        else:
            raise ArtworkDoesNotExistException(f"Artwork with ID '{artwork_id}' does not exist")

    def remove_artwork(self, artwork_id):
        cursor = None
        if artwork_id is None:
            raise InvalidIDException("Artwork ID is required")
        cursor = self.connection.cursor()
        cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork_id,))
        if not cursor.fetchone():
            raise InvalidIDException("Artwork with this ID does not exist")
        cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (artwork_id,))
        self.connection.commit()
        return cursor.rowcount > 0

    def search_artworks(self, keyword, search_by):
        cursor = self.connection.cursor()
        field_map = {
            'title': 'Title',
            'artist_id': 'Artist_ID',
            'medium': 'Medium'
        }
        if search_by not in field_map:
            raise ValueError("Invalid search criteria. Choose from: title, medium, artist_id")

        column = field_map[search_by]
        if search_by == 'artist':
            if not keyword.strip().isdigit():
                print("Artist ID must be a number.")
                return []
            artist_id = int(keyword.strip())
            query = "SELECT * FROM Artwork WHERE Artist_ID = %s"
            cursor.execute(query, (artist_id,))
        else:
            like_pattern = f"%{keyword}%"
            query = f"SELECT * FROM Artwork WHERE {column} LIKE %s"
            cursor.execute(query, (like_pattern,))
        rows = cursor.fetchall()
        artworks = []
        for row in rows:
            artworks.append(Artwork(
                artwork_id = row[0],
                title = row[1],
                description = row[2],
                creation_date = row[3],
                medium = row[4],
                image_url = row[5],
                artist_id = row[6]
            ))
        return artworks
```

Gallery Implementation

```python
    """ Gallery Implementation """

    def validate_gallery_fields(self, gallery):
        if gallery.gallery_id is None or str(gallery.gallery_id).strip() == "":
            raise InvalidIDException("Gallery ID is required")
        if gallery.name is None or gallery.name.strip() == "":
            raise InvalidGalleryNameException("Gallery Name is required")
        if gallery.curator is None or str(gallery.curator).strip() == "":
            raise GalleryAssignmentException("Curator ID cannot be empty")
        if gallery.opening_hours:
            hours = gallery.opening_hours.strip()
            if hours.lower() == "closed":
                return
            if "-" not in hours or not ("AM" in hours.upper() or "PM" in hours.upper()):
                raise InvalidOpeningHoursException("Invalid format for Opening Hours")
            parts = hours.split("-")
            if len(parts) != 2:
                raise InvalidOpeningHoursException(
                    "Opening Hours format should be 'hh:mm AM/PM - hh:mm AM/PM' or 'Closed'")

    def add_gallery(self,gallery):
        self.validate_gallery_fields(gallery)
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Gallery where Gallery_ID = %s", (gallery.gallery_id,))
        if cursor.fetchone():
            raise InvalidIDException("Gallery with this ID already exists")
        cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (gallery.curator,))
        if not cursor.fetchone():
            raise InvalidIDException(f"Curator with ID {gallery.curator} does not exist")
        cursor.execute("SELECT * FROM Gallery where Gallery_ID = %s AND Name = %s", (gallery.gallery_id,gallery.name,))
        if cursor.fetchone():
            raise DuplicateGalleryException(f"Gallery with this name '{gallery.name}' already exists")
        cursor.execute("""INSERT INTO Gallery
        (Gallery_ID, Name,Description, Location, Curator, Opening_Hours)
         VALUES (%s, %s, %s, %s, %s, %s)""",
         (gallery.gallery_id,gallery.name,gallery.description or None,gallery.location or None,gallery.curator or None,gallery.opening_hours or None))
        self.connection.commit()
        return cursor.rowcount > 0

    def get_gallery_by_id(self, gallery_id):
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
        row = cursor.fetchone()
        if row:
            return Gallery(
                gallery_id=row[0],
                name=row[1],
                description=row[2],
                location=row[3],
                curator=row[4],
                opening_hours=row[5]
            )
        else:
            return None

    def update_gallery(self, gallery):
        if gallery.gallery_id is None or str(gallery.gallery_id).strip() == "":
            raise InvalidIDException("Gallery ID is required")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery.gallery_id,))
        existing = cursor.fetchone()
        if not existing:
            raise GalleryNotFoundException("Gallery with this ID does not exist")

```

```
        if gallery.curator:
            cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (gallery.curator,))
            if not cursor.fetchone():
                raise InvalidIDException(f"Curator with ID {gallery.curator} does not exist")

        self.validate_gallery_fields(gallery)

        query = """
            UPDATE Gallery SET
                Name = %s,
                Description = %s,
                Location = %s,
                Curator = %s,
                Opening_Hours = %s
            WHERE Gallery_ID = %s
        """
        cursor.execute(query, (
            gallery.name or existing[1],
            gallery.description or existing[2],
            gallery.location or existing[3],
            gallery.curator or existing[4],
            gallery.opening_hours or existing[5],
            gallery.gallery_id
        ))
        self.connection.commit()
        return cursor.rowcount > 0

    def remove_gallery(self, gallery_id):
        if gallery_id is None or str(gallery_id).strip() == "":
            raise InvalidIDException("Gallery ID is required for deletion")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
        if not cursor.fetchone():
            raise GalleryNotFoundException("Gallery with this ID does not exist")

        query = "DELETE FROM Gallery WHERE Gallery_ID = %s"
        cursor.execute(query, (gallery_id,))
        self.connection.commit()
        return cursor.rowcount > 0

    def search_gallery(self, keyword, search_by):
        cursor = self.connection.cursor()
        field_map = {
            'name': 'Name',
            'location': 'Location',
            'curator': 'Curator'
        }
        if search_by not in field_map:
            raise ValueError("Invalid search criteria. Choose from: name, location, curator")

        column = field_map[search_by]

        if search_by == 'curator':
            if not keyword.strip().isdigit():
                print("Curator ID must be a number.")
                return []
            query = "SELECT * FROM Gallery WHERE Curator = %s"
            cursor.execute(query, (int(keyword.strip()),))
        else:
            like_pattern = f"%{keyword}%"
            query = f"SELECT * FROM Gallery WHERE {column} LIKE %s"
            cursor.execute(query, (like_pattern,))

        rows = cursor.fetchall()
        galleries = []
        for row in rows:
            galleries.append(Gallery(
                gallery_id=row[0],
                name=row[1],
                description=row[2],
                location=row[3],
                curator=row[4],
                opening_hours=row[5]
            ))
        return galleries
```

User Implementation

```python
    """ User Implementation """
    def validate_user_fields(self, user):
        if user.user_id is None or str(user.user_id).strip() == "":
            raise InvalidIDException("User ID is required")
        if user.username is None or user.username.strip() == "":
            raise InvalidUsernameException("Username is required")
        if user.password is None or user.password.strip() == "":
            raise InvalidPasswordException("Password is required")
        if user.email:
            email_pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
            if not re.match(email_pattern, user.email):
                raise InvalidEmailException("Invalid email format")
        if user.date_of_birth and user.date_of_birth > date.today():
            raise InvalidIDException("Date of birth cannot be in the future")
        if user.profile_picture is not None and not re.match(r"^https?://[^\s]+$", user.profile_picture):
            raise InvalidWebsiteException("Profile Picture must be a valid URL")

    def add_user(self, user):
        self.validate_user_fields(user)
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user.user_id,))
        if cursor.fetchone():
            raise InvalidIDException("User ID already exists")

        cursor.execute("SELECT * FROM User WHERE Username = %s OR Email = %s", (user.username, user.email))
        if cursor.fetchone():
            raise DuplicateUserException("Username or Email already in use")

        cursor.execute("""
            INSERT INTO User
            (User_ID, Username, Password, Email, First_Name, Last_Name, Date_of_Birth, Profile_Picture, User_is_active)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            user.user_id,
            user.username,
            user.password,
            user.email or None,
            user.first_name or None,
            user.last_name or None,
            user.date_of_birth or None,
            user.profile_picture or None,
            user.user_is_active
        ))

        self.connection.commit()
        return cursor.rowcount > 0

    def get_user_by_id(self, user_id):
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
        row = cursor.fetchone()

        if row:
            return User(
                user_id=row[0],
                username=row[1],
                password=row[2],
                email=row[3],
                first_name=row[4],
                last_name=row[5],
                date_of_birth=row[6],
                profile_picture=row[7],
                user_is_active=row[8]
            )
        else:
            return None

    def update_user(self, user):
        if user.user_id is None:
            raise InvalidIDException("User ID is required")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user.user_id,))
        existing = cursor.fetchone()

        if not existing:
            raise InvalidIDException("User with this ID does not exist")

```

```
        self.validate_user_fields(user)

        query = """
        UPDATE User SET
            Username = %s,
            Password = %s,
            Email = %s,
            First_Name = %s,
            Last_Name = %s,
            Date_of_Birth = %s,
            Profile_Picture = %s,
            User_is_active = %s
        WHERE User_ID = %s
        """

        cursor.execute(query, (
            user.username or existing[1],
            user.password or existing[2],
            user.email or existing[3],
            user.first_name or existing[4],
            user.last_name or existing[5],
            user.date_of_birth or existing[6],
            user.profile_picture or existing[7],
            user.user_is_active if user.user_is_active is not None else existing[8],
            user.user_id
        ))
        self.connection.commit()
        return cursor.rowcount > 0

    def remove_user(self, user_id):
        if user_id is None:
            raise InvalidIDException("User ID is required for deletion")

        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
        if not cursor.fetchone():
            raise InvalidIDException("User with this ID does not exist")

        query = "UPDATE User SET User_is_active = FALSE WHERE User_ID = %s"
        cursor.execute(query, (user_id,))
        self.connection.commit()
        return cursor.rowcount > 0

    def reactivate_user(self, user_id):
        cursor = self.connection.cursor()
        cursor.execute("SELECT User_is_active FROM User WHERE User_ID = %s", (user_id,))
        row = cursor.fetchone()
        if not row:
            return "not_found"
        is_active = row[0]
        if is_active:
            return "already_active"

        cursor.execute("UPDATE User SET User_is_active = TRUE WHERE User_ID = %s", (user_id,))
        self.connection.commit()
        return "reactivated"
```

User Favorite Implementation

```python
    """ User Favorite Implementation """

    def add_artwork_to_favorite(self, user_id, artwork_id):
        if user_id is None or artwork_id is None:
            raise InvalidIDException("User ID and Artwork ID are required")
        cursor = self.connection.cursor()

        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
        if not cursor.fetchone():
            raise UserNotFoundException("User with this ID does not exist")

        cursor.execute("SELECT * FROM Artwork WHERE Artwork_ID = %s", (artwork_id,))
        if not cursor.fetchone():
            raise ArtworkDoesNotExistException("Artwork with this ID does not exist")

        cursor.execute("SELECT * FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
                       (user_id, artwork_id))
        if cursor.fetchone():
            raise FavoriteAlreadyExistsException("Favorite Already Exists")

        cursor.execute("INSERT INTO User_Favorite_Artwork (User_ID, Artwork_ID) VALUES (%s, %s)", (user_id, artwork_id))
        self.connection.commit()
        return cursor.rowcount > 0

    def remove_artwork_from_favorite(self, user_id, artwork_id):
        if user_id is None or artwork_id is None:
            raise InvalidIDException("User ID and Artwork ID are required")

        cursor = self.connection.cursor()

        cursor.execute("SELECT * FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
                       (user_id, artwork_id))
        if not cursor.fetchone():
            raise FavoriteNotFoundException("Favorite Artwork with this ID does not exist")

        cursor.execute("DELETE FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
                       (user_id, artwork_id))
        self.connection.commit()
        return cursor.rowcount > 0

    def get_user_favorite_artworks(self, user_id):
        if user_id is None:
            raise InvalidIDException("User ID is required")
        cursor = self.connection.cursor()

        cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
        if not cursor.fetchone():
            raise InvalidIDException("User with this ID does not exist")

        query = """
        SELECT a.Artwork_ID, a.Title, a.Description, a.Creation_date, a.Medium, a.Image_URL, a.Artist_ID
        FROM Artwork a
        JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
        WHERE ufa.User_ID = %s
        """
        cursor.execute(query, (user_id,))
        rows = cursor.fetchall()
        if not rows:
            return False

        favorite_artworks = []
        for row in rows:
            favorite_artworks.append(Artwork(
                artwork_id=row[0],
                title=row[1],
                description=row[2],
                creation_date=row[3],
                medium=row[4],
                image_url=row[5],
                artist_id=row[6]
            ))
        return favorite_artworks
```

Artwork Gallery Implementation

```python
""" Artwork Gallery Implementation """

def add_artwork_to_gallery(self, artwork_id, gallery_id):
    cursor = self.connection.cursor()

    cursor.execute("SELECT * FROM Artwork WHERE Artwork_ID = %s", (artwork_id,))
    if not cursor.fetchone():
        raise InvalidIDException("Artwork with this ID does not exist")

    cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
    if not cursor.fetchone():
        raise GalleryNotFoundException("Gallery with this ID does not exist")

    cursor.execute("SELECT * FROM Artwork_Gallery WHERE Artwork_ID = %s AND Gallery_ID = %s",
                   (artwork_id, gallery_id))
    if cursor.fetchone():
        raise DuplicateArtworkException("This artwork is already assigned to the gallery")

    cursor.execute("INSERT INTO Artwork_Gallery (Artwork_ID, Gallery_ID) VALUES (%s, %s)", (artwork_id, gallery_id))
    self.connection.commit()
    return cursor.rowcount > 0

def remove_artwork_from_gallery(self, artwork_id, gallery_id):
    cursor = self.connection.cursor()
    cursor.execute("SELECT * FROM Artwork_Gallery WHERE Artwork_ID = %s AND Gallery_ID = %s",
                   (artwork_id, gallery_id))
    if not cursor.fetchone():
        raise ArtworkDoesNotExistException("Artwork not found in this gallery")

    cursor.execute("DELETE FROM Artwork_Gallery WHERE Artwork_ID = %s AND Gallery_ID = %s",
                   (artwork_id, gallery_id))
    self.connection.commit()
    return cursor.rowcount > 0

def get_artworks_by_gallery(self, gallery_id):
    cursor = self.connection.cursor()

    cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
    if not cursor.fetchone():
        raise GalleryNotFoundException("Gallery with this ID does not exist")

    cursor.execute("""
        SELECT a.Artwork_ID, a.Title, a.Description, a.Creation_Date, a.Medium, a.Image_URL, a.Artist_ID
        FROM Artwork a
        JOIN Artwork_Gallery ag ON a.Artwork_ID = ag.Artwork_ID
        WHERE ag.Gallery_ID = %s
    """, (gallery_id,))
    rows = cursor.fetchall()

    if not rows:
        return False

    artworks = []
    for row in rows:
        artworks.append(Artwork(
            artwork_id=row[0],
            title=row[1],
            description=row[2],
            creation_date=row[3],
            medium=row[4],
            image_url=row[5],
            artist_id=row[6]
        ))
    return artworks
```

User Login Implementation

```python
    """ User Login Implementation """

    def user_login(self, username, password):
        if not username or not password:
            raise InvalidUsernameException("Username and password are required")

        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT User_ID, Username, Password, User_is_active
            FROM User
            WHERE Username = %s
        """, (username,))

        user_data = cursor.fetchone()

        if not user_data:
            raise UserNotFoundException("User not found")

        user_id, db_username, db_password, is_active = user_data

        if not is_active:
            raise UserNotFoundException("User account is inactive")

        if password != db_password:
            raise InvalidPasswordException("Invalid password")

        # Get user's favorite artworks
        favorites = self.get_user_favorite_artworks(user_id)

        return {
            "user_id": user_id,
            "username": db_username,
            "favorites": favorites if favorites else "No favorites yet"
        }
```

Artist Login Implementation

```python
""" Artist Login Implementation """

def artist_login(self, name, password=None):
    if not name:
        raise InvalidArtistNameException("Artist name is required")

    cursor = self.connection.cursor()
    cursor.execute("""
        SELECT Artist_ID, Name, Contact_Information, IsActive
        FROM Artist
        WHERE Name = %s
    """, (name,))

    artist_data = cursor.fetchone()

    if not artist_data:
        raise InvalidArtistNameException("Artist not found")

    artist_id, artist_name, contact_info, is_active = artist_data

    if not is_active:
        raise InvalidArtistNameException("Artist account is inactive")

    # Simple password check using contact information (could be improved with proper auth)
    if password and password != contact_info:
        raise InvalidPasswordException("Invalid credentials")
```

```
        cursor.execute("""
            SELECT Artwork_ID, Title, Creation_Date, Medium
            FROM Artwork
            WHERE Artist_ID = %s
        """, (artist_id,))
        artist_artworks = cursor.fetchall()

        cursor.execute("""
            SELECT DISTINCT g.Gallery_ID, g.Name, g.Location
            FROM Gallery g
            JOIN Artwork_Gallery ag ON g.Gallery_ID = ag.Gallery_ID
            JOIN Artwork a ON ag.Artwork_ID = a.Artwork_ID
            WHERE a.Artist_ID = %s
        """, (artist_id,))
        galleries_with_artworks = cursor.fetchall()

        cursor.execute("""
            SELECT a.Artwork_ID, a.Title, COUNT(ufa.User_ID) as favorite_count
            FROM Artwork a
            LEFT JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
            WHERE a.Artist_ID = %s
            GROUP BY a.Artwork_ID, a.Title
            ORDER BY favorite_count DESC
        """, (artist_id,))
        artwork_favorites = cursor.fetchall()

        return {
            "artist_id": artist_id,
            "artist_name": artist_name,
            "artworks": [
                {
                    "artwork_id": artwork[0],
                    "title": artwork[1],
                    "creation_date": artwork[2],
                    "medium": artwork[3]
                } for artwork in artist_artworks
            ] if artist_artworks else "No artworks uploaded yet",
            "galleries_with_artworks": [
                {
                    "gallery_id": gallery[0],
                    "name": gallery[1],
                    "location": gallery[2]
                } for gallery in galleries_with_artworks
            ] if galleries_with_artworks else "No artworks in galleries yet",
            "artwork_favorites": [
                {
                    "artwork_id": fav[0],
                    "title": fav[1],
                    "favorite_count": fav[2]
                } for fav in artwork_favorites
            ]
        }

    def artist_signup(self, artist, password):
        if artist.artist_id is None:
            raise InvalidIDException("Artist ID is required")
        if not password:
            raise InvalidPasswordException("Password is required")
        self.validate_artist_fields(artist)

        cursor = self.connection.cursor()

        # Check if artist ID already exists
        cursor.execute("SELECT * FROM artist WHERE Artist_ID = %s", (artist.artist_id,))
```

```
        if cursor.fetchone():
            raise InvalidIDException("Artist ID is already in use")

        # Check if artist name already exists
        cursor.execute("SELECT * FROM artist WHERE Name = %s", (artist.name,))
        if cursor.fetchone():
            raise InvalidArtistNameException("Artist name is already in use")

        query = """
            INSERT INTO Artist
            (Artist_ID, Name, Biography, Birth_Date, Nationality, Website, Contact_Information, Password, IsActive)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(query, (
            artist.artist_id,
            artist.name,
            artist.biography or None,
            artist.birth_date or None,
            artist.nationality or None,
            artist.website or None,
            artist.contact_information or None,
            password,
            True
        ))
        self.connection.commit()
        return cursor.rowcount > 0

    def artist_login(self, name, password):
        if not name or not password:
            raise InvalidArtistNameException("Artist name and password are required")

        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT Artist_ID, Name, Contact_Information, IsActive, Password
            FROM Artist
            WHERE Name = %s
        """, (name,))

        artist_data = cursor.fetchone()

        if not artist_data:
            raise InvalidArtistNameException("Artist not found")

        artist_id, artist_name, contact_info, is_active, db_password = artist_data

        if not is_active:
            raise InvalidArtistNameException("Artist account is inactive")

        if password != db_password:
            raise InvalidPasswordException("Invalid password")

        # Get artworks by this artist
        cursor.execute("""
            SELECT Artwork_ID, Title, Creation_Date, Medium
            FROM Artwork
            WHERE Artist_ID = %s
        """, (artist_id,))
        artist_artworks = cursor.fetchall()

        # Get galleries containing this artist's works
        cursor.execute("""
            SELECT DISTINCT g.Gallery_ID, g.Name, g.Location
            FROM Gallery g
            JOIN Artwork_Gallery ag ON g.Gallery_ID = ag.Gallery_ID
            JOIN Artwork a ON ag.Artwork_ID = a.Artwork_ID
            WHERE a.Artist_ID = %s
        """, (artist_id,))
        galleries_with_artworks = cursor.fetchall()

        # Get user favorites for this artist's works
        cursor.execute("""
            SELECT a.Artwork_ID, a.Title, COUNT(ufa.User_ID) as favorite_count
            FROM Artwork a
            LEFT JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
            WHERE a.Artist_ID = %s
            GROUP BY a.Artwork_ID, a.Title
            ORDER BY favorite_count DESC
```

```
        """, (artist_id,))
        artwork_favorites = cursor.fetchall()

        return {
            "artist_id": artist_id,
            "artist_name": artist_name,
            "artworks": [
                {
                    "artwork_id": artwork[0],
                    "title": artwork[1],
                    "creation_date": artwork[2],
                    "medium": artwork[3]
                } for artwork in artist_artworks
            ] if artist_artworks else "No artworks uploaded yet",
            "galleries_with_artworks": [
                {
                    "gallery_id": gallery[0],
                    "name": gallery[1],
                    "location": gallery[2]
                } for gallery in galleries_with_artworks
            ] if galleries_with_artworks else "No artworks in galleries yet",
            "artwork_favorites": [
                {
                    "artwork_id": fav[0],
                    "title": fav[1],
                    "favorite_count": fav[2]
                } for fav in artwork_favorites
            ]
        }
```

EXCEPTION

```python
""" Artist Exceptions """
class InvalidArtistNameException(Exception):
    def __init__(self, message = "Artist Name cannot be empty or NULL"):
        super().__init__(message)

""" Artwork Exceptions """
class InvalidArtworkException(Exception):
    def __init__(self, message = "Artwork Title/ Description / Creation Date cannot be Null"):
        super().__init__(message)

class InvalidMediumTypeException(Exception):
    def __init__(self, message    = "Medium Type should be from the medium types"):
        super().__init__(message)

class DuplicateArtworkException(Exception):
    def __init__(self, message = "Artwork Title already exists"):
        super().__init__(message)

class ArtworkDoesNotExistException(Exception):
    def __init__(self, message = "Artwork Does Not Exist"):
        super().__init__(message)

""" Gallery Exceptions """

class DuplicateGalleryException(Exception):
    def __init__(self, message = "Gallery Title already exists"):
        super().__init__(message)

class InvalidGalleryNameException(Exception):
    def __init__(self, message = "Gallery Name cannot be empty or NULL"):
        super().__init__(message)

class GalleryNotFoundException(Exception):
    def __init__(self, message = "Gallery with this ID Does Not Exist"):
        super().__init__(message)

class GalleryAssignmentException(Exception):
    def __init__(self, message="Failed to assign artwork to gallery."):
        super().__init__(message)

class InvalidOpeningHoursException(Exception):
    def __init__(self, message="Opening hours must be in format 'hh:mm AM/PM - hh:mm AM/PM' or 'Closed'."):
        super().__init__(message)

""" User Exceptions """
class DuplicateUserException(Exception):
    def __init__(self, message = "Username already exists"):
        super().__init__(message)

class InvalidUsernameException(Exception):
    def __init__(self, message = "Username cannot be empty or NULL"):
        super().__init__(message)

class InvalidPasswordException(Exception):
    def __init__(self, message = "Password cannot be empty or NULL"):
        super().__init__(message)

class InvalidEmailException(Exception):
    def __init__(self, message = "Email cannot be empty or NULL"):
        super().__init__(message)

class UserNotFoundException(Exception):
    def __init__(self, message="User with the given ID does not exist"):
        super().__init__(message)

""" Common Exceptions """
class InvalidWebsiteException(Exception):
    def __init__(self, message = "Website must be a valid URL (e.g., https://example.com) "):
        super().__init__(message)

class InvalidDateException(Exception):
    def __init__(self, message= "Date cannot be in future"):
        super().__init__(message)

class InvalidIDException(Exception):
    def __init__(self, message = "ID cannot be NULL or Already Exists"):
        super().__init__(message)

""" User Favorite Exceptions """
class FavoriteAlreadyExistsException(Exception):
    def __init__(self, message="This artwork is already in the user's favorites"):
        super().__init__(message)

class FavoriteNotFoundException(Exception):
    def __init__(self, message="This artwork is not in the user's favorites"):
        super().__init__(message)
```

UTIL- (DBPropertyUtil) (DBConnUtil)

```python
import mysql.connector
from util.DBPropertyUtil import PropertyUtil

class DBConnection:
    _connection = None

    @staticmethod
    def get_connection():
        if DBConnection._connection is None:
            props = PropertyUtil.get_property_string()
            if props:
                try:
                    DBConnection._connection = mysql.connector.connect(
                        host = props.get('host'),
                        port = props.get('port'),
                        database = props.get('database'),
                        user = props.get('user'),
                        password = props.get('password')
                    )
                except mysql.connector.Error as e:
                    print(f"Error connecting to MySql: {e}")
            else:
                print(f"Properties could not be loaded")
        return DBConnection._connection
```

```python
import os

class PropertyUtil:

    @staticmethod
    def get_property_string(file_path = 'config/db.properties'):
        properties = {}
        try:
            with open(file_path, 'r') as f:
                for line in f:
                    line = line.strip()
                    if line and not line.startswith('#') and '=' in line:
                        key,value = line.split('=', 1)
                        properties[key.strip()] = value.strip()
        except FileNotFoundError:
            print(f"Error: Property file '{file_path}' not found.")

        except Exception as e:
            print(f"Error reading property file: {e}")
        return properties
```

Main

```python
from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
from entity.artist import Artist
from datetime import datetime
from entity.artwork import Artwork
from entity.gallery import Gallery
from entity.user import User
from exception.exceptions import (InvalidIDException, InvalidArtistNameException, InvalidDateException,
                                  InvalidWebsiteException, InvalidArtworkException, InvalidMediumTypeException,
                                  DuplicateArtworkException, DuplicateGalleryException, InvalidGalleryNameException,
                                  GalleryNotFoundException, GalleryAssignmentException, InvalidOpeningHoursException,
                                  DuplicateUserException, InvalidUsernameException, InvalidPasswordException,
                                  InvalidEmailException, UserNotFoundException, FavoriteAlreadyExistsException,
                                  ArtworkDoesNotExistException, FavoriteNotFoundException)
import sys
import os


def clean_input(value):
    return value.strip() if value.strip() != "" else None


class MainModule:
    @staticmethod
    def artist_menu(service):
        while True:
            print("\n ------- Artist Menu -------")
            print("\n 1. Add Artist")
            print("\n 2. Update Artist")
            print("\n 3. Delete Artist")
            print("\n 4. Get Artist By ID")
            print("\n 5. Reactivate Artist")
            print("\n 0. Exit")
            choice = input("Enter you choice:")

            try:
                if choice == "1":
                    artist_id = int(input("Enter Artist ID: "))
                    name = input("Enter Artist Name: ")
                    biography = clean_input(input("Enter Biography: "))
                    birth_date_input = clean_input(input("Enter Birth Date (YYYY-MM-DD): "))
                    birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date() if birth_date_input else None
                    nationality = clean_input(input("Enter Nationality: "))
                    website = clean_input(input("Enter Website: "))
                    contact_information = clean_input(input("Enter Contact Information: "))

                    artist = Artist(artist_id=artist_id,name=name,biography=biography or None,birth_date=birth_date or None,nationality=nationality or None,
                                    website=website or None, contact_information=contact_information or None,is_active=True)

                    if service.add_artist(artist):
                        print("Artist Added Successfully!")
                    else:
                        print("Failed to add artist")

                elif choice == "2":
                    artist_id = int(input("Enter Artist ID to update: "))
                    existing_artist = service.get_artist_by_id(artist_id)
                    if not existing_artist:
                        raise InvalidIDException("Artist with this ID does not exist")

                    print("Please leave the field empty to retain existing value")
                    name = input(f"Enter New Artist Name [{existing_artist.name}]: ") or existing_artist.name
                    biography = clean_input(input(f"Enter New Biography [{existing_artist.biography}]: ")) or existing_artist.biography
                    birth_date_input = clean_input(input(f"Enter New Birth Date (YYYY-MM-DD) [{existing_artist.birth_date}]: "))

                    if birth_date_input:
                        birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date()
                    else:
                        birth_date = existing_artist.birth_date

                    nationality = clean_input(input(f"Enter New Nationality: [{existing_artist.nationality}]: ")) or existing_artist.nationality
                    website = clean_input(input(f"Enter Website: [{existing_artist.website}]: ")) or existing_artist.website
                    contact_information = clean_input(input(f"Enter Contact Information: [{existing_artist.contact_information}]: ")) or existing_artist.contact_information

                    updated_artist = Artist(artist_id=artist_id,name=name,biography=biography,birth_date=birth_date,nationality=nationality,
                        website=website,contact_information=contact_information,is_active=True)
                    if service.update_artist(updated_artist):
                        print("Artist Updated Successfully")

                elif choice == "3":
                    artist_id = int(input("Enter Artist ID to delete: "))
```

```
                    confirm = input(f"Are you sure you want to delete Artist {artist_id}? [y/N] ")
                    if confirm.lower() == "y":
                        if service.remove_artist(artist_id):
                            print("Artist deleted successfully")
                        else:
                            print("Failed to delete artist")
                    else:
                        print("Attempt to Delete the Artist has been cancelled")

                elif choice == "4":
                    artist_id = int(input("Enter Artist ID to fetch: "))
                    artist = service.get_artist_by_id(artist_id)
                    if artist:
                        print("\n--------- Artist Details ---------")
                        print(f"ID: {artist.artist_id}")
                        print(f"Name: {artist.name}")
                        print(f"Biography: {artist.biography or 'N/A'}")
                        print(f"Nationality: {artist.nationality or 'N/A'}")
                        print(f"Website: {artist.website or 'N/A'}")
                        print(f"Contact Information: {artist.contact_information or 'N/A'}")
                        print(f"Active Status: {'Active' if artist.is_active else 'Inactive'}")
                    else:
                        raise InvalidIDException("Artist with this ID does not exist")

                elif choice == "5":
                    artist_id = int(input("Enter Artist ID to reactivate: "))
                    result = service.reactivate_artist(artist_id)
                    if result == 'reactivated':
                        print("Artist account Reactivated Successfully")
                    elif result == 'already_active':
                        print("Artist account Already Active, No Action Needed")
                    elif result == 'not_found':
                        raise InvalidIDException("Artist with this ID does not exist")

                elif choice == "0":
                    print("Exiting... Thank You!")
                    break
                else:
                    print("Invalid Choice. Try again.")

            except (InvalidIDException, InvalidArtistNameException, InvalidDateException, InvalidWebsiteException) as e:
                print(f"Validation Error: {e}")
            except ValueError:
                print("Invalid input type. Please enter numeric value.")
            except Exception as e:
                print(f"Unexpected Error: {e}")

    @staticmethod
    def artwork_menu(service):
        while True:
            print("\n ------- Artwork Menu -------")
            print("\n 1. Add Artwork")
            print("\n 2. Update Artwork")
            print("\n 3. Delete Artwork")
            print("\n 4. Get Artwork By ID")
            print("\n 5. Search Artwork")
            print("\n 0. Exit")
            choice = input("Enter you choice:")
            try:
                if choice == "1":
                    artwork_id = clean_input(input("Enter Artwork ID: "))
                    artist_id = clean_input(input("Enter Artist ID: "))
                    title = clean_input(input("Enter Artwork Title: "))
                    description = clean_input(input("Enter Artwork Description: "))
                    creation_date_input = clean_input(input("Enter Creation Date (YYYY-MM-DD): "))
                    creation_date = datetime.strptime(creation_date_input, "%Y-%m-%d").date() if creation_date_input else None
                    medium = clean_input(input("Enter Medium Type: "))
                    image_url=clean_input(input("Enter Image URL: "))

                    artwork = Artwork(artwork_id = artwork_id, title = title, description = description, creation_date = creation_date, medium = medium, image_url = image_url or None, artist_id = artist_id)
                    success = service.add_artwork(artwork)
                    if success:
                        print("Artwork Added Successfully")
                    else:
                        print("Failed to add Artwork")

                elif choice == "2":
```

```
                elif choice == "2":
                    artwork_id = int(input("Enter Artwork ID to update: "))
                    existing_artwork = service.get_artwork_by_id(artwork_id)
                    if not existing_artwork:
                        raise InvalidIDException("Artwork with this ID does not exist")

                    print("Please leave the field empty to retain existing value")
                    title = input(f"Enter New Title [{existing_artwork.title}]: ") or existing_artwork.title
                    description = clean_input(
                        input(f"Enter New Description [{existing_artwork.description}]: ")) or existing_artwork.description
                    creation_date_input = clean_input(
                        input(f"Enter New Creation Date (YYYY-MM-DD) [{existing_artwork.creation_date}]: "))

                    if creation_date_input:
                        creation_date = datetime.strptime(creation_date_input, "%Y-%m-%d").date()
                    else:
                        creation_date = existing_artwork.creation_date

                    medium = clean_input(input(
                        f"Enter New Medium: [{existing_artwork.medium}]: ")) or existing_artwork.medium
                    image_url = clean_input(
                        input(f"Enter Image URL: [{existing_artwork.image_url}]: ")) or existing_artwork.image_url
                    artist_id = clean_input(input(
                        f"Enter Artist ID: [{existing_artwork.artist_id}]: ")) or existing_artwork.artist_id

                    updated_artwork = Artwork(artwork_id=artwork_id, title = title, description = description, creation_date=creation_date,
                                              medium = medium, image_url = image_url or None, artist_id = artist_id)

                    if service.update_artwork(updated_artwork):
                        print("Artwork Updated Successfully")
```

```
                elif choice == "3":
                    artwork_id = int(input("Enter Artwork ID to delete: "))
                    confirm = input(f"Are you sure you want to delete Artwork {artwork_id}? [y/N] ")
                    if confirm.lower() == "y":
                        if service.remove_artwork(artwork_id):
                            print("Artwork deleted successfully")
                        else:
                            print("Failed to delete artwork")
                    else:
                        print("Attempt to Delete the Artwork has been cancelled")

                elif choice == "4":
                    artwork_id = int(input("Enter Artwork ID to fetch: "))
                    artwork = service.get_artwork_by_id(artwork_id)
                    if artwork:
                        print("\n--------- Artist Details ---------")
                        print(f"ID: {artwork.artwork_id}")
                        print(f"Title: {artwork.title}")
                        print(f"Description: {artwork.description or 'N/A'}")
                        print(f"Creation Date: {artwork.creation_date or 'N/A'}")
                        print(f"Medium {artwork.medium or 'N/A'}")
                        print(f"Image URL: {artwork.image_url or 'N/A'}")
                        print(f"Artist ID: {artwork.artist_id}")
                    else:
                        raise InvalidIDException("Artwork with this ID does not exist")
                elif choice == "5":
                    print("\n Search Artworks By:")
                    print("1. Title")
                    print("2. Artist ID")
                    print("3. Medium")
                    search_choice = input("Enter your choice (1/2/3): ")

                    if search_choice == "1":
                        search_by = "title"
                    elif search_choice == "2":
                        search_by = "artist_id"
                    elif search_choice == "3":
                        search_by = "medium"
                    else:
                        print("Invalid choice. Please select valid option")
                        continue

                    keyword = input(f"Enter keyword to search in {search_by.capitalize()}: ")
                    artworks = service.search_artworks(keyword, search_by)
                    if artworks:
                        print("\n--------- Matching Artworks ---------")
                        for art in artworks:
                            print(f"\nID: {art.artwork_id}")
                            print(f"Title: {art.title}")
                            print(f"Description: {art.description or 'N/A'}")
                            print(f"Creation Date: {art.creation_date or 'N/A'}")
                            print(f"Medium: {art.medium or 'N/A'}")
                            print(f"Image URL: {art.image_url or 'N/A'}")
                            print(f"Artist ID: {art.artist_id}")
                    else:
                        print("No artworks found for the given keyword")

                elif choice == "0":
                    print("Exiting... Thank You!")
                    break
                else:
                    print("Invalid Choice. Try again.")
            except (InvalidIDException,InvalidArtworkException, InvalidDateException, InvalidWebsiteException, InvalidMediumTypeException, DuplicateArtworkException) as e:
                print(f"Validation Error: {e}")
            except ValueError:
                print("Invalid input type. Please enter numeric value..")
            except Exception as e:
                print(f"Unexpected Error: {e}")

    @staticmethod
    def gallery_menu(service):
        while True:
            print("\n ------- Gallery Menu -------")
            print("\n 1. Add Gallery")
            print("\n 2. Update Gallery")
            print("\n 3. Get Gallery By ID")
            print("\n 4. Delete Gallery")
```

```
            print("\n 0. Exit")
            choice = input("Enter you choice:")
            try:
                if choice == "1":
                    gallery_id = clean_input(input("Enter Gallery ID: "))
                    name = clean_input(input("Enter Gallery Name: "))
                    description = clean_input(input("Enter Gallery Description: "))
                    location = clean_input(input("Enter Gallery Location: "))
                    curator = clean_input(input("Enter Gallery Curator: "))
                    opening_hours = clean_input(input("Enter Gallery Opening Hours: "))
                    gallery = Gallery(gallery_id = gallery_id, name = name, description = description or None, location = location or None, curator = curator or None, opening_hours = opening_hours or None)
                    success = service.add_gallery(gallery)
                    if success:
                        print("Gallery Added Successfully")
                    else:
                        print("Failed to add Gallery")
```

```
            elif choice == "2":
                gallery_id = clean_input(input("Enter Gallery ID to update: "))
                existing = service.get_gallery_by_id(gallery_id)
                if not existing:
                    print("Gallery not found.")
                    continue

                name = clean_input(input(f"Enter new name [{existing.name}]: ")) or existing.name
                description = clean_input(
                    input(f"Enter new description [{existing.description}]: ")) or existing.description
                location = clean_input(input(f"Enter new location [{existing.location}]: ")) or existing.location
                curator = clean_input(input(f"Enter new curator ID [{existing.curator}]: ")) or existing.curator
                opening_hours = clean_input(
                    input(f"Enter new opening hours [{existing.opening_hours}]: ")) or existing.opening_hours

                updated_gallery = Gallery(
                    gallery_id=gallery_id,
                    name=name,
                    description=description,
                    location=location,
                    curator=curator,
                    opening_hours=opening_hours
                )
                if service.update_gallery(updated_gallery):
                    print("Gallery updated successfully")
                else:
                    print("Update failed")

            elif choice == "3":
                gallery_id = clean_input(input("Enter Gallery ID to fetch: "))
                gallery = service.get_gallery_by_id(gallery_id)
                if gallery:
                    print(f"\nGallery ID: {gallery.gallery_id}")
                    print(f"Name: {gallery.name}")
                    print(f"Description: {gallery.description}")
                    print(f"Location: {gallery.location}")
                    print(f"Curator: {gallery.curator}")
                    print(f"Opening Hours: {gallery.opening_hours}")
                else:
                    print("Gallery not found.")

            elif choice == "4":
                gallery_id = int(input("Enter Gallery ID to delete: "))
                confirm = input(f"Are you sure you want to delete Gallery {gallery_id}? [y/N] ")
                if confirm.lower() == "y":
                    if service.remove_gallery(gallery_id):
                        print("Gallery deleted successfully")
                    else:
                        print("Failed to delete gallery")
                else:
                    print("Gallery deletion cancelled.")

            elif choice == "5":
                print("\n Search Galleries By:")
                print("1. Name")
                print("2. Location")
                print("3. Curator ID")
                search_choice = input("Enter your choice (1/2/3): ")

                if search_choice == "1":
                    search_by = "name"
                elif search_choice == "2":
                    search_by = "location"
                elif search_choice == "3":
                    search_by = "curator"
                else:
                    print("Invalid choice. Please select a valid option")
                    return

                keyword = input(f"Enter keyword to search in {search_by.capitalize()}: ")
                galleries = service.search_gallery(keyword, search_by)
                if galleries:
                    print("\n--------- Matching Galleries ---------")
                    for g in galleries:
                        print(f"\nID: {g.gallery_id}")
                        print(f"Name: {g.name}")
                        print(f"Description: {g.description or 'N/A'}")
```

```
                print("Invalid Choice. Try again.")

        except (InvalidIDException, DuplicateGalleryException, InvalidGalleryNameException, GalleryNotFoundException, GalleryAssignmentException,InvalidOpeningHoursException) as e:
            print(f"Validation Error: {e}")
        except ValueError:
            print("Invalid input type. Please enter numeric value..")
        except Exception as e:
            print(f"Unexpected Error: {e}")

""" User Menu """
@staticmethod
def user_menu(service):
    while True:
        print("\n ------- User Menu -------")
        print("\n 1. Add User")
        print("\n 2. Update User")
        print("\n 3. Get User By ID")
        print("\n 4. Delete User")
        print("\n 5. Reactivate User")
        print("\n 0. Exit")
        choice = input("Enter your choice: ")

        try:
            if choice == "1":
                user_id = int(input("Enter User ID: "))
                username = input("Enter Username: ")
                password = input("Enter Password: ")
                email = clean_input(input("Enter Email: "))
                first_name = clean_input(input("Enter First Name: "))
                last_name = clean_input(input("Enter Last Name: "))
                dob_input = clean_input(input("Enter Date of Birth (YYYY-MM-DD): "))
                date_of_birth = datetime.strptime(dob_input, "%Y-%m-%d").date() if dob_input else None
                profile_picture = clean_input(input("Enter Profile Picture URL: "))
                user = User(user_id = user_id,username=username, password=password, email=email or None, first_name=first_name or None,last_name=last_name or None, date_of_birth=date_of_birth or None, profile_picture=profile_picture or None, user_is_active = True )
                if service.add_user(user):
                    print("User Added Successfully!")
                else:
                    print("Failed to add user")

            elif choice == "2":
                user_id = int(input("Enter User ID to update: "))
                existing_user = service.get_user_by_id(user_id)
                if not existing_user:
                    raise InvalidIDException("User with this ID does not exist")

                print("Leave fields empty to keep existing values.")

                username = input(f"Enter New Username [{existing_user.username}]: ") or existing_user.username
                password = input(f"Enter New Password [{existing_user.password}]: ") or existing_user.password
                email = clean_input(input(f"Enter New Email [{existing_user.email}]: ")) or existing_user.email
                first_name = clean_input(
                    input(f"Enter New First Name [{existing_user.first_name}]: ")) or existing_user.first_name
                last_name = clean_input(
                    input(f"Enter New Last Name [{existing_user.last_name}]: ")) or existing_user.last_name
                dob_input = clean_input(
                    input(f"Enter New Date of Birth (YYYY-MM-DD) [{existing_user.date_of_birth}]: "))

                if dob_input:
                    date_of_birth = datetime.strptime(dob_input, "%Y-%m-%d").date()
                else:
                    date_of_birth = existing_user.date_of_birth

                profile_picture = clean_input(input(
                    f"Enter New Profile Picture URL [{existing_user.profile_picture}]: ")) or existing_user.profile_picture

                updated_user = User(user_id=user_id,username=username,password=password,email=email,first_name=first_name,last_name=last_name,date_of_birth=date_of_birth,profile_picture=profile_picture,user_is_active=True)
                if service.update_user(updated_user):
                    print("User updated successfully!")
                else:
                    print("Failed to update user.")

            elif choice == "3":
                user_id = int(input("Enter User ID to fetch: "))
                user = service.get_user_by_id(user_id)
                if user:
                    print("\n--------- User Details ---------")
                    print(f"ID: {user.user_id}")
```

```
                    print(f"Username: {user.username}")
                    print(f"Email: {user.email or 'N/A'}")
                    print(f"First Name: {user.first_name or 'N/A'}")
                    print(f"Last Name: {user.last_name or 'N/A'}")
                    print(f"Date of Birth: {user.date_of_birth or 'N/A'}")
                    print(f"Profile Picture: {user.profile_picture or 'N/A'}")
                    print(f"Active Status: {'Active' if user.user_is_active else 'Inactive'}")
                else:
                    raise InvalidIDException("User with this ID does not exist")

            elif choice == "4":
                user_id = int(input("Enter User ID to delete: "))
                confirm = input(f"Are you sure you want to delete User {user_id}? [y/N] ")
                if confirm.lower() == "y":
                    if service.remove_user(user_id):
                        print("User deleted successfully")
                    else:
                        print("Failed to delete user")
                else:
                    print("Attempt to Delete the User has been cancelled")

            elif choice == "5":
                user_id = int(input("Enter User ID to reactivate: "))
                password = input("Enter the user's password for verification: ").strip()
                result = service.reactivate_user(user_id, password)
                if result == 'reactivated':
                    print("User account Reactivated Successfully")
                elif result == 'already_active':
                    print("User account Already Active, No Action Needed")
                elif result == 'invalid_credentials':
                    print("\n Security Alert: Incorrect password - Reactivation denied")
                    print("Please verify credentials with the user.")
                elif result == 'not_found':
                    raise InvalidIDException("User with this ID does not exist")


            elif choice == "0":
                print("Exiting... Thank You!")
                break
            else:
                print("Invalid Choice. Try again.")

        except (InvalidIDException, DuplicateUserException, InvalidUsernameException, InvalidPasswordException, InvalidEmailException) as e:
            print(f"Validation Error: {e}")
        except ValueError:
            print("Invalid input type. Please enter numeric value.")
        except Exception as e:
            print(f"Unexpected Error: {e}")
```

```
    """ User Favorite """

    @staticmethod
    def user_favorite_menu(service):
        while True:
            print("\n ------- User Favorite Menu -------")
            print("\n 1. Add Artwork to Favorite")
            print("\n 2. Remove Artwork from Favorite")
            print("\n 3. Get User favorite Artworks by ID")
            print("\n 0. Exit")
            choice = input("Enter your choice: ")

            try:
                if choice == "1":
                    user_id = int(input("Enter User ID: "))
                    artwork_id = int(input("Enter Artwork ID to favorite: "))
                    if service.add_artwork_to_favorite(user_id, artwork_id):
                        print("Artwork added to favorites successfully.")
                    else:
                        print("Failed to add artwork to favorites.")
                elif choice == "2":
                    user_id = int(input("Enter User ID: "))
                    artwork_id = int(input("Enter Artwork ID to remove from favorites: "))
                    if service.remove_artwork_from_favorite(user_id, artwork_id):
                        print("Artwork removed from favorites successfully.")
                    else:
                        print("Failed to remove artwork from favorites.")
                elif choice == "3":
                    user_id = int(input("Enter User ID to fetch favorite artworks: "))
                    favorites = service.get_user_favorite_artworks(user_id)
                    if not favorites:
                        print("This user has no favorite artworks.")
                    else:
                        print("\n------ Favorite Artworks ------")
                        for art in favorites:
                            print(f"\nID: {art.artwork_id}")
                            print(f"Title: {art.title}")
                            print(f"Description: {art.description}")
                            print(f"Medium: {art.medium}")
                            print(f"Creation Date: {art.creation_date}")
                            print(f"Artist ID: {art.artist_id}")
                elif choice == "0":
                    print("Exiting... Thank You!")
                    break

            except (UserNotFoundException, ArtworkDoesNotExistException, FavoriteAlreadyExistsException,
                    InvalidIDException, FavoriteNotFoundException) as e:
                print(f"Validation Error: {e}")
            except Exception as e:
                print(f"Unexpected Error: {e}")

    @staticmethod
    def artwork_gallery_menu(service):
        while True:
            print("\n ------- Artwork Gallery Menu -------")
            print("\n 1. Add Artwork to Gallery")
            print("\n 2. Remove Artwork from Gallery")
            print("\n 3. Get Artworks by Gallery ID")
            print("\n 0. Exit")
            choice = input("Enter your choice: ")

            try:
                if choice == "1":
                    artwork_id = int(input("Enter Artwork ID: "))
                    gallery_id = int(input("Enter Gallery ID: "))
                    if service.add_artwork_to_gallery(artwork_id, gallery_id):
                        print("Artwork successfully added to the gallery")
                    else:
                        print("Failed to add artwork")
                elif choice == "2":
                    artwork_id = int(input("Enter Artwork ID: "))
                    gallery_id = int(input("Enter Gallery ID: "))
                    if service.remove_artwork_from_gallery(artwork_id, gallery_id):
                        print("Artwork removed from gallery successfully")
                    else:
                        print("Failed to remove artwork")
                elif choice == "3":
                    gallery_id = int(input("Enter Gallery ID to fetch artworks: "))
```

```
                        print("No artworks found for this gallery")
                elif choice == "0":
                    print("Exiting... Thank You!")
                    break

            except (GalleryNotFoundException, DuplicateArtworkException,ArtworkDoesNotExistException,
                    InvalidIDException, GalleryNotFoundException) as e:
                print(f"Validation Error: {e}")
            except Exception as e:
                print(f"Unexpected Error: {e}")

    """ Artist Panel Implementation """

    @staticmethod
    def artist_panel(service):
        print("\n-------- Artist Portal --------")
        while True:
            print("\n1. Login")
            print("2. Sign Up")
            print("0. Back to Main Menu")
            choice = input("Enter your choice: ").strip()

            if choice == "1":
                MainModule._artist_login_flow(service)
                break
            elif choice == "2":
                MainModule._artist_signup_flow(service)
            elif choice == "0":
                return
            else:
                print("Invalid choice. Please try again.")

    @staticmethod
    def _artist_login_flow(service):
        print("\n-------- Artist Login --------")
        while True:
            artist_name = input("Enter your artist name: ").strip()
            password = input("Enter your password: ").strip()

            if not artist_name or not password:
                print("Both artist name and password are required.")
                continue

            try:
                # Authenticate the artist
                artist_info = service.artist_login(artist_name, password)
                print(f"\nWelcome, {artist_info['artist_name']}!")
                MainModule._artist_dashboard(service, artist_info)
                break
            except Exception as e:
                print(f"Login failed: {str(e)}")
                retry = input("Would you like to try again? (y/n): ").lower()
                if retry != 'y':
                    return

    @staticmethod
    def _artist_signup_flow(service):
        print("\n-------- Artist Sign Up --------")
        try:
            artist_id = int(input("Enter your artist ID : ").strip())
            name = input("Enter your artist username: ").strip()
            password = input("Create a password: ").strip()
            confirm_password = input("Confirm password: ").strip()

            if password != confirm_password:
                print("Passwords do not match!")
                return

            if len(password) < 6:
                print("Password must be at least 6 characters long!")
                return

            biography = clean_input(input("Enter your biography (optional): "))

            birth_date_input = clean_input(input("Enter your birth date (YYYY-MM-DD, optional): "))
            birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date() if birth_date_input else None

```

```
            nationality = clean_input(input("Enter your nationality (optional): "))
            website = clean_input(input("Enter your website URL (optional): "))
            contact_info = input("Enter your contact information: ").strip()

            artist = Artist(
                artist_id=artist_id,
                name=name,
                biography=biography,
                birth_date=birth_date,
                nationality=nationality,
                website=website,
                contact_information=contact_info,
                is_active=True
            )

            if service.artist_signup(artist, password):
                print("\nArtist account created successfully!")
                print("You can now login with your artist name and password.")
            else:
                print("Failed to create artist account.")

        except ValueError as e:
            print(f"Invalid input: {e}")
        except Exception as e:
            print(f"Error: {e}")

    @staticmethod
    def _artist_dashboard(service, artist_info):
        while True:
            print("\n-------- Artist Dashboard --------")
            print(f"Welcome, {artist_info['artist_name']} (ID: {artist_info['artist_id']})")
            print("\n1. View My Artworks")
            print("2. View My Artworks in Galleries")
            print("3. View Favorite Counts of My Artworks")
            print("4. Add New Artwork")
            print("5. Update Existing Artwork")
            print("6. Remove Artwork")
            print("0. Logout")

            choice = input("Enter your choice: ").strip()

            try:
                if choice == "1":
                    artworks = artist_info['artworks']
                    if isinstance(artworks, str):
                        print(artworks)
                    else:
                        print("\n--- My Artworks ---")
                        for artwork in artworks:
                            print(f"\nID: {artwork['artwork_id']}")
                            print(f"Title: {artwork['title']}")
                            print(f"Medium: {artwork['medium']}")
                            print(f"Created: {artwork['creation_date']}")

                elif choice == "2":
                    galleries = artist_info['galleries_with_artworks']
                    if isinstance(galleries, str):
                        print(galleries)
                    else:
                        print("\n--- Galleries Featuring My Artworks ---")
                        for gallery in galleries:
                            print(f"\nGallery ID: {gallery['gallery_id']}")
                            print(f"Name: {gallery['name']}")
                            print(f"Location: {gallery['location']}")

                elif choice == "3":
                    favorites = artist_info['artwork_favorites']
                    print("\n--- Favorite Counts for My Artworks ---")
                    for fav in favorites:
                        print(f"\nArtwork ID: {fav['artwork_id']}")
                        print(f"Title: {fav['title']}")
                        print(f"Number of favorites: {fav['favorite_count']}")

                elif choice == "4":
                    print("\n--- Add New Artwork ---")
                    try:
                        artwork_id = int(input("Enter artwork ID: ").strip())
```

```
                    description = input("Enter artwork description: ").strip()
                    creation_date = input("Enter creation date (YYYY-MM-DD): ").strip()
                    medium = input("Enter medium type: ").strip()
                    image_url = input("Enter image URL (optional): ").strip() or None

                    creation_date = datetime.strptime(creation_date, "%Y-%m-%d").date()
                    artwork = Artwork(
                        artwork_id=artwork_id,
                        title=title,
                        description=description,
                        creation_date=creation_date,
                        medium=medium,
                        image_url=image_url,
                        artist_id=artist_info['artist_id']
                    )
                    if service.add_artwork(artwork):
                        print("Artwork added successfully!")

                        artist_info = service.artist_login(artist_info['artist_name'],
                                                           input("Enter your password to refresh: ").strip())
                    else:
                        print("Failed to add artwork.")
                except ValueError as e:
                    print(f"Invalid input: {e}")
                except Exception as e:
                    print(f"Error: {e}")

            elif choice == "5":
                print("\n--- Update Artwork ---")
                try:
                    artwork_id = int(input("Enter artwork ID to update: ").strip())
                    artwork = service.get_artwork_by_id(artwork_id)

                    if not artwork:
                        print("Artwork not found.")
                        continue

                    if str(artwork.artist_id) != str(artist_info['artist_id']):
                        print("You can only update your own artworks.")
                        continue

                    print("\nCurrent Artwork Details:")
                    print(f"Title: {artwork.title}")
                    print(f"Description: {artwork.description}")
                    print(f"Creation Date: {artwork.creation_date}")
                    print(f"Medium: {artwork.medium}")
                    print(f"Image URL: {artwork.image_url}")

                    print("\nEnter new values (leave blank to keep current):")
                    title = input(f"Title [{artwork.title}]: ").strip() or artwork.title
                    description = input(f"Description [{artwork.description}]: ").strip() or artwork.description
                    creation_date_input = input(f"Creation Date [{artwork.creation_date}] (YYYY-MM-DD): ").strip()
                    creation_date = datetime.strptime(creation_date_input,
                                                      "%Y-%m-%d").date() if creation_date_input else artwork.creation_date
                    medium = input(f"Medium [{artwork.medium}]: ").strip() or artwork.medium
                    image_url = input(f"Image URL [{artwork.image_url}]: ").strip() or artwork.image_url

                    updated_artwork = Artwork(
                        artwork_id=artwork_id,
                        title=title,
                        description=description,
                        creation_date=creation_date,
                        medium=medium,
                        image_url=image_url,
                        artist_id=artist_info['artist_id']
                    )

                    if service.update_artwork(updated_artwork):
                        print("Artwork updated successfully!")
                        # Refresh artist info
                        artist_info = service.artist_login(artist_info['artist_name'],
                                                           input("Enter your password to refresh: ").strip())
                    else:
                        print("Failed to update artwork.")
                except ValueError as e:
                    print(f"Invalid input: {e}")
                except Exception as e:
```

```
                elif choice == "6":
                    print("\n--- Remove Artwork ---")
                    try:
                        artwork_id = int(input("Enter artwork ID to remove: ").strip())
                        artwork = service.get_artwork_by_id(artwork_id)

                        if not artwork:
                            print("Artwork not found.")
                            continue

                        if str(artwork.artist_id) != str(artist_info['artist_id']):
                            print("You can only remove your own artworks.")
                            continue

                        confirm = input(
                            f"Are you sure you want to permanently delete artwork '{artwork.title}'? (y/n): ").lower()
                        if confirm == 'y':
                            if service.remove_artwork(artwork_id):
                                print("Artwork removed successfully!")
                                # Refresh artist info
                                artist_info = service.artist_login(artist_info['artist_name'],
                                                                   input("Enter your password to refresh: ").strip())
                            else:
                                print("Failed to remove artwork.")
                    except Exception as e:
                        print(f"Error: {e}")

                elif choice == "0":
                    print("Logging out...")
                    break

                else:
                    print("Invalid choice. Please try again.")

            except Exception as e:
                print(f"An error occurred: {e}")

    @staticmethod
    def user_panel(service):
        print("\n-------- User Portal --------")
        while True:
            print("\n1. Login")
            print("2. Sign Up")
            print("0. Back to Main Menu")
            choice = input("Enter your choice: ").strip()

            if choice == "1":
                MainModule._user_login_flow(service)
                break
            elif choice == "2":
                MainModule._user_signup_flow(service)
            elif choice == "0":
                return
            else:
                print("Invalid choice. Please try again.")

    @staticmethod
    def _user_login_flow(service):
        print("\n-------- User Login --------")
        while True:
            username = input("Enter your username: ").strip()
            password = input("Enter your password: ").strip()

            if not username or not password:
                print("Both username and password are required.")
                continue

            try:
                # Authenticate the user
                user_info = service.user_login(username, password)
                print(f"\nWelcome, {user_info['username']}!")
                MainModule._user_dashboard(service, user_info)
                break
            except Exception as e:
                print(f"Login failed: {str(e)}")
                retry = input("Would you like to try again? (y/n): ").lower()
                if retry != 'y':
```

```
                return

    @staticmethod
    def _user_signup_flow(service):
        print("\n-------- User Sign Up --------")
        try:
            user_id = int(input("Enter your user ID (number): ").strip())
            username = input("Choose a username: ").strip()
            password = input("Create a password: ").strip()
            confirm_password = input("Confirm password: ").strip()

            if password != confirm_password:
                print("Passwords do not match!")
                return

            if len(password) < 6:
                print("Password must be at least 6 characters long!")
                return

            email = clean_input(input("Enter your email: "))
            first_name = clean_input(input("Enter your first name: "))
            last_name = clean_input(input("Enter your last name: "))

            dob_input = clean_input(input("Enter your date of birth (YYYY-MM-DD): "))
            date_of_birth = datetime.strptime(dob_input, "%Y-%m-%d").date() if dob_input else None

            profile_picture = clean_input(input("Enter profile picture URL: "))

            user = User(
                user_id=user_id,
                username=username,
                password=password,
                email=email,
                first_name=first_name,
                last_name=last_name,
                date_of_birth=date_of_birth,
                profile_picture=profile_picture,
                user_is_active=True
            )

            if service.add_user(user):
                print("\nUser account created successfully!")
                print("You can now login with your username and password.")
            else:
                print("Failed to create user account.")

        except ValueError as e:
            print(f"Invalid input: {e}")
        except Exception as e:
            print(f"Error: {e}")

    @staticmethod
    def _user_dashboard(service, user_info):
        while True:
            print("\n-------- User Dashboard --------")
            print(f"Welcome, {user_info['username']} (ID: {user_info['user_id']})")
            print("\n1. View My Favorites")
            print("2. Add Artwork to Favorites")
            print("3. Remove Artwork from Favorites")
            print("0. Logout")

            choice = input("Enter your choice: ").strip()

            try:
                if choice == "1":
                    favorites = user_info['favorites']
                    if isinstance(favorites, str):
                        print(favorites)
                    else:
                        print("\n--- My Favorite Artworks ---")
                        for artwork in favorites:
                            print(f"\nID: {artwork.artwork_id}")
                            print(f"Title: {artwork.title}")
                            print(f"Description: {artwork.description}")
                            print(f"Medium: {artwork.medium}")
                            print(f"Created: {artwork.creation_date}")
```

```
                    print(f"Description: {artwork.description}")
                    print(f"Medium: {artwork.medium}")
                    print(f"Created: {artwork.creation_date}")

                elif choice == "2":
                    print("\n--- Add Artwork to Favorites ---")
                    artwork_id = int(input("Enter artwork ID to add to favorites: ").strip())
                    if service.add_artwork_to_favorite(user_info['user_id'], artwork_id):
                        print("Artwork added to favorites successfully!")
                        user_info = service.user_login(user_info['username'],
                                                       input("Enter your password to refresh: ").strip())
                    else:
                        print("Failed to add artwork to favorites.")

                elif choice == "3":
                    print("\n--- Remove Artwork from Favorites ---")
                    artwork_id = int(input("Enter artwork ID to remove from favorites: ").strip())
                    if service.remove_artwork_from_favorite(user_info['user_id'], artwork_id):
                        print("Artwork removed from favorites successfully!")
                        # Refresh user info
                        user_info = service.user_login(user_info['username'],
                                                       input("Enter your password to refresh: ").strip())
                    else:
                        print("Failed to remove artwork from favorites.")

                elif choice == "0":
                    print("Logging out...")
                    break

                else:
                    print("Invalid choice. Please try again.")

            except ValueError:
                print("Invalid input. Please enter a valid number.")
            except Exception as e:
                print(f"An error occurred: {e}")
    @staticmethod
    def main():
        service = VirtualArtGalleryImp()
        print("------------------- Welcome Virtual Art Gallery ----------------")
        while True:
            print("\n Main Menu")
            print("1. Artist")
            print("2. Artwork")
            print("3. Gallery")
            print("4. User")
            print("5. User Favorite")
            print("6. Artwork Gallery")
            print("7. Artist Panel")
            print("8. User Panel")
            print("0. Exit")

            choice = input("Enter your choice: ")
            if choice == "1":
                MainModule.artist_menu(service)
            elif choice == "2":
                MainModule.artwork_menu(service)
            elif choice == "3":
                MainModule.gallery_menu(service)
            elif choice == "4":
                MainModule.user_menu(service)
            elif choice == "5":
                MainModule.user_favorite_menu(service)
            elif choice == "6":
                MainModule.artwork_gallery_menu(service)
            elif choice == "7":
                MainModule.artist_panel(service)
            elif choice == "8":
                MainModule.user_panel(service)
            elif choice == '0':
                print("GoodBye!")
                break
            else:
                print("Invalid Choice. Try again.")

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

TESTING

ARTIST MANAGEMENT:

```python
import pytest
from datetime import datetime
from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
from entity.artist import Artist

@pytest.fixture
def artists():
   artists= VirtualArtGalleryImp()
   return artists

def test_add_artist(artists):
    cursor = artists.connection.cursor()
    test_artist_id = 16
    test_artist_name = 'test name'

    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

    artist_obj = Artist(artist_id = test_artist_id,
                        name = test_artist_name,
                        biography = "Test Bio for artist",
                        birth_date=datetime.strptime("2002-03-03", "%Y-%m-%d").date(),
                        nationality="Test Nationality",
                        website="https://tests.com",
                        contact_information="Test Contact Information",
                        is_active=True)
    results = artists.add_artist(artist_obj)
    assert results is True
    cursor.execute("SELECT * FROM Artist WHERE artist_id = %s", (test_artist_id,))
    assert cursor.fetchone() is not None
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

def test_update_artist(artists):
    cursor = artists.connection.cursor()
    test_artist_id = 16
    test_artist_name = 'test name'
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

    artist_obj = Artist(artist_id=test_artist_id,
                        name=test_artist_name,
                        biography="Test Bio for artist",
                        birth_date=datetime.strptime("2002-03-03", "%Y-%m-%d").date(),
                        nationality="Test Nationality",
                        website="https://tests.com",
                        contact_information="Test Contact Information",
                        is_active=True)
    results = artists.add_artist(artist_obj)
    assert results is True

    new_birth_date = datetime.strptime("2003-03-16", "%Y-%m-%d").date()
    cursor.execute("UPDATE Artist SET birth_date = %s WHERE artist_id = %s", (new_birth_date,test_artist_id,))
    artists.connection.commit()
    cursor.execute("SELECT Birth_Date FROM Artist WHERE artist_id = %s", (test_artist_id,))
    updated_birth_date = cursor.fetchone()[0]
    assert updated_birth_date == new_birth_date

def test_delete_artist(artists):
    cursor = artists.connection.cursor()
    test_artist_id = 16
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()
    artist_obj = Artist(artist_id=test_artist_id,
                        name="Test Artist",
                        biography="Test Bio for artist",
```

```
                        birth_date=datetime.strptime("2002-03-03", "%Y-%m-%d").date(),
                        nationality="Test Nationality",
                        website="https://tests.com",
                        contact_information="Test Contact Information",
                        is_active=True)
    results = artists.add_artist(artist_obj)
    assert results is True
    cursor.execute("SELECT * FROM Artist WHERE artist_id = %s", (test_artist_id,))
    assert cursor.fetchone() is not None
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

def test_get_artist_by_id(artists):
    cursor = artists.connection.cursor()
    test_artist_id = 16
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()
    artist_obj = Artist(artist_id=test_artist_id,
                        name="Test Artist",
                        biography="Test Bio for artist",
                        birth_date=datetime.strptime("2002-03-03", "%Y-%m-%d").date(),
                        nationality="Test Nationality",
                        website="https://tests.com",
                        contact_information="Test Contact Information",
                        is_active=True)
    results = artists.add_artist(artist_obj)
    assert results is True
    fetched_artist = artists.get_artist_by_id(test_artist_id)
    assert fetched_artist is not None
    assert fetched_artist.artist_id == test_artist_id
    assert fetched_artist.name == "Test Artist"
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

def test_reactivate_artist(artists):
    cursor = artists.connection.cursor()
    test_artist_id = 16
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()

    #case 1: artist not found
    result = artists.reactivate_artist(test_artist_id)
    assert result == "not_found"

    #adding inactive artist
    artist_obj = Artist(artist_id=test_artist_id,
                        name="Inactive artist",
                        biography="Test Bio for artist",
                        birth_date=datetime.strptime("2002-03-03", "%Y-%m-%d").date(),
                        nationality="Test Nationality",
                        website="https://tests.com",
                        contact_information="Test Contact Information",
                        is_active=False)
    assert artists.add_artist(artist_obj) is True

    #case 2: reactivate artist
    result = artists.reactivate_artist(test_artist_id)
    assert result == "reactivated"

    #case 3: artist already active
    result = artists.reactivate_artist(test_artist_id)
    assert result == "already_active"
    cursor.execute("DELETE FROM Artist WHERE artist_id = %s", (test_artist_id,))
    artists.connection.commit()
```

ARTWORK MANAGEMENT:

```python
import pytest
from datetime import datetime
from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
from entity.artwork import Artwork

@pytest.fixture
def gallery():
    gallery = VirtualArtGalleryImp()
    return gallery

def test_add_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_artwork_id = 13
    test_title = "Test Art to be added"

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()

    artwork = Artwork(
        artwork_id=test_artwork_id,
        title=test_title,
        description="Uploaded for testing",
        creation_date=datetime(2022, 5, 10).date(),
        medium="Oil on Canvas",
        image_url="https://test_art.jpg",
        artist_id=1
    )

    result = gallery.add_artwork(artwork)
    assert result is True

    cursor.execute("SELECT * FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    assert cursor.fetchone() is not None

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()


def test_update_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_artwork_id = 15
    test_title = "Artwork to Update"

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()

    artwork = Artwork(
        artwork_id=test_artwork_id,
        title=test_title,
        description="Original description",
        creation_date=datetime(2022, 5, 10).date(),
        medium="Watercolor",
        image_url="https://test_art.jpg",
        artist_id=1
    )

    result = gallery.add_artwork(artwork)
    assert result is True

    new_medium = 'Acrylic'
    cursor.execute("UPDATE Artwork SET Medium = %s WHERE Artwork_ID = %s", (new_medium, test_artwork_id))
    gallery.connection.commit()

    cursor.execute("SELECT Medium FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    updated_medium = cursor.fetchone()[0]
    assert updated_medium == new_medium

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()

#Delete the artwork from the gallery
def test_delete_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_artwork_id = 17

    artwork = Artwork(
        artwork_id=test_artwork_id,
        title="Test Title",
        description="Original description",
        creation_date=datetime(2022, 5, 10).date(),
```

```python
#Delete the artwork from the gallery
def test_delete_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_artwork_id = 17

    artwork = Artwork(
        artwork_id=test_artwork_id,
        title="Test Title",
        description="Original description",
        creation_date=datetime(2022, 5, 10).date(),
        medium="Watercolor",
        image_url="https://test_art.jpg",
        artist_id=1
    )
    result = gallery.add_artwork(artwork)
    assert result is True

    cursor.execute("SELECT * FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    assert cursor.fetchone() is not None

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()


#searching for artworks
def test_search_for_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_artwork_id = 18
    test_title = "Artwork to Search"
    test_artist_id = 1
    test_medium = "Watercolor"
    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()
    artwork = Artwork(
        artwork_id=test_artwork_id,
        title=test_title,
        description="test description",
        creation_date=datetime(2023, 5, 11).date(),
        medium= test_medium,
        image_url="https://test_art.jpg",
        artist_id=test_artist_id
    )
    gallery.add_artwork(artwork)
    search_by = 'medium'
    keyword = "Watercolor"
    field = {
        'title': 'Title',
        'artist_id': 'Artist_ID',
        'medium': 'Medium'
    }
    column = field[search_by]

    if search_by == 'artist':
        if not keyword.strip().isdigit():
            print("Artist ID must be a number.")
            assert False
            return
        artist_id = int(keyword.strip())
        query = "SELECT * FROM Artwork WHERE Artist_ID = %s"
        cursor.execute(query, (artist_id,))
    else:
        like_pattern = f"%{keyword}%"
        query = f"SELECT * FROM Artwork WHERE {column} LIKE %s"
        cursor.execute(query, (like_pattern,))
    results = cursor.fetchall()

    assert any(row[0] == test_artwork_id for row in results)

    cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (test_artwork_id,))
    gallery.connection.commit()
```

GALLERY MANAGEMENT:

```python
import pytest
from datetime import datetime
from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
from entity.gallery import Gallery

@pytest.fixture
def gallery():
    gallery = VirtualArtGalleryImp()
    return gallery

def test_add_gallery(gallery):
    cursor = gallery.connection.cursor()
    test_gallery_id = 12
    cursor.execute("DELETE FROM Gallery where Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

    gallery_obj = Gallery(
        gallery_id=test_gallery_id,
        name = "Test Gallery Name added",
        description = "Test Gallery Description added",
        location = "Test Gallery Location",
        curator = 1,
        opening_hours = "10 AM - 10 PM"
    )
    result = gallery.add_gallery(gallery_obj)
    assert result is True
    cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    assert cursor.fetchone() is not None

    cursor.execute("DELETE FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

def test_update_gallery(gallery):
    cursor = gallery.connection.cursor()
    test_gallery_id = 15
    cursor.execute("DELETE FROM Gallery where Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

    gallery_obj = Gallery(
        gallery_id=test_gallery_id,
        name="Test Original Title",
        description="Test Original Gallery Description",
        location="Test Original Gallery Location",
        curator=1,
        opening_hours="10 AM - 10 PM"
    )
    result = gallery.add_gallery(gallery_obj)
    assert result is True

    new_opening_hours = "11 AM - 6 PM"
    new_curator = 2
    cursor.execute("UPDATE Gallery SET Opening_hours = %s WHERE Gallery_ID = %s", (new_opening_hours, test_gallery_id,))
    gallery.connection.commit()
    cursor.execute("SELECT Opening_hours FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    updated_opening_hours = cursor.fetchone()[0]
    assert updated_opening_hours == new_opening_hours
    cursor.execute("UPDATE Gallery SET Curator = %s WHERE Gallery_ID = %s", (new_curator, test_gallery_id,))
    gallery.connection.commit()
    cursor.execute("SELECT Curator FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    updated_curator = cursor.fetchone()[0]
    assert updated_curator == new_curator

    cursor.execute("DELETE FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

def test_delete_gallery(gallery):
    cursor = gallery.connection.cursor()
    test_gallery_id = 15
    gallery_obj = Gallery(
        gallery_id=test_gallery_id,
        name="Test Original Title",
        description="Test Original Gallery Description",
        location="Test Original Gallery Location",
        curator=1,
        opening_hours="10 AM - 10 PM"
    )
    result = gallery.add_gallery(gallery_obj)
    assert result is True

    cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
```

```
    cursor.execute("DELETE FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

def test_search_for_artwork(gallery):
    cursor = gallery.connection.cursor()
    test_gallery_id = 15
    test_name = "Gallery to be searched"
    test_location = "Gallery Location"
    test_curator = 1
    cursor.execute("DELETE FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()

    gallery_obj = Gallery(
        gallery_id=test_gallery_id,
        name= test_name,
        description="Test Original Gallery Description",
        location=test_location,
        curator=test_curator,
        opening_hours="10 AM - 10 PM"
    )
    gallery.add_gallery(gallery_obj)
    search_by = 'curator'
    keyword = "1"
    field_map = {
        'name': 'Name',
        'location': 'Location',
        'curator': 'Curator'
    }

    column = field_map[search_by]

    if search_by == 'curator':
        if not keyword.strip().isdigit():
            print("Curator ID must be a number.")
            assert False
            return
        curator = int(keyword.strip())
        query = "SELECT * FROM Gallery WHERE Curator = %s"
        cursor.execute(query, (curator,))
    else:
        like_pattern = f"%{keyword}%"
        query = f"SELECT * FROM Gallery WHERE {column} LIKE %s"
        cursor.execute(query, (like_pattern,))
    results = cursor.fetchall()
    assert any(row[0] == test_gallery_id for row in results)

    cursor.execute("DELETE FROM Gallery WHERE Gallery_ID = %s", (test_gallery_id,))
    gallery.connection.commit()
```

USER MANAGEMENT:

```
import pytest
from datetime import datetime
from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
from entity.user import User

@pytest.fixture
def users():
  users= VirtualArtGalleryImp()
  return users

def test_add_user(users):
    cursor = users.connection.cursor()
    test_user_id = 22
    test_user_name = "Test User Name"
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()

    user_obj = User(user_id = test_user_id,
                    username=test_user_name,
                    password="Test password",
                    email="testemail@gmail.com",
                    first_name="Test First Name",
                    last_name="Test Last Name",
                    date_of_birth=datetime.strptime("1970-01-01", "%Y-%m-%d").date(),
                    profile_picture="https://testimg.png")
    results = users.add_user(user_obj)
    assert results is True
    cursor.execute("SELECT * FROM User WHERE user_id = %s", (test_user_id,))
    assert cursor.fetchone() is not None
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()

def test_update_user(users):
    cursor = users.connection.cursor()
    test_user_id = 22
    test_user_name = "Test User Name"
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()
    user_obj = User(user_id=test_user_id,
                    username=test_user_name,
                    password="Test password",
                    email="testemail@gmail.com",
                    first_name="Test First Name",
                    last_name="Test Last Name",
                    date_of_birth=datetime.strptime("1970-01-01", "%Y-%m-%d").date(),
                    profile_picture="https://testimg.png")
    results = users.add_user(user_obj)
    assert results is True
```

```python
def test_delete_user(users):
    cursor = users.connection.cursor()
    test_user_id = 22
    test_user_name = "Test User Name"
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()
    user_obj = User(user_id=test_user_id,
                    username=test_user_name,
                    password="Test password",
                    email="testemail@gmail.com",
                    first_name="Test First Name",
                    last_name="Test Last Name",
                    date_of_birth=datetime.strptime("1970-01-01", "%Y-%m-%d").date(),
                    profile_picture="https://testimg.png")
    results = users.add_user(user_obj)
    assert results is True
    cursor.execute("SELECT * FROM User WHERE user_id = %s", (test_user_id,))
    assert cursor.fetchone() is not None
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()

def test_get_user_by_id(users):
    cursor = users.connection.cursor()
    test_user_id = 22
    test_user_name = "Test User Name"
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()
    user_obj = User(user_id=test_user_id,
                    username=test_user_name,
                    password="Test password",
                    email="testemail@gmail.com",
                    first_name="Test First Name",
                    last_name="Test Last Name",
                    date_of_birth=datetime.strptime("1970-01-01", "%Y-%m-%d").date(),
                    profile_picture="https://testimg.png")
    results = users.add_user(user_obj)
    assert results is True

    fetched_user = users.get_user_by_id(test_user_id)
    assert fetched_user is not None
    assert fetched_user.user_id == test_user_id
    assert fetched_user.username == test_user_name
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()

def test_reactivate_user(users):
    cursor = users.connection.cursor()
    test_user_id = 22
    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()

    #case 1: user not found
    result = users.reactivate_user(test_user_id)
    assert result == "not_found"

    #adding inactive user
    user_obj = User(user_id=test_user_id,
                    username="Active User name",
                    password="Test password",
                    email="testemail@gmail.com",
                    first_name="Test First Name",
                    last_name="Test Last Name",
                    date_of_birth=datetime.strptime("1970-01-01", "%Y-%m-%d").date(),
                    profile_picture="https://testimg.png")
    users.add_user(user_obj)

    #case 2: reactivate user
    results = users.reactivate_user(test_user_id)
    assert results == "reactivated"

    #case 3: user already active
    result = users.reactivate_user(test_user_id)
    assert result == "already_active"

    cursor.execute("DELETE FROM User WHERE user_id = %s", (test_user_id,))
    users.connection.commit()
```

OUT PUT SCREENSHOT

ARTISTS:

```
 Main Menu
1. Artist
2. Artwork
3. Gallery
4. User
5. User Favorite
6. Artwork Gallery
7. Artist Panel
8. User Panel
0. Exit
Enter your choice: 7

-------- Artist Portal --------

1. Login
2. Sign Up
0. Back to Main Menu
Enter your choice: 1

-------- Artist Login --------
Enter your artist name: NewArtist
Enter your password: NewArtist1

Welcome, NewArtist!

-------- Artist Dashboard --------
Welcome, NewArtist (ID: 88)

1. View My Artworks
2. View My Artworks in Galleries
3. View Favorite Counts of My Artworks
4. Add New Artwork
5. Update Existing Artwork
6. Remove Artwork
0. Logout
Enter your choice:
```

```
------------------- Welcome Virtual Art Gallery -----------------

 Main Menu
1. Artist
2. Artwork
3. Gallery
4. User
5. User Favorite
6. Artwork Gallery
0. Exit
Enter your choice: 1

 ------- Artist Menu -------

 1. Add Artist

 2. Update Artist

 3. Delete Artist

 4. Get Artist By ID

 5. Reactivate Artist

 0. Exit
Enter you choice:1
Enter Artist ID: 4
Enter Artist Name: Raj
Enter Biography: Indian artist known for vibrant colors
Enter Birth Date (YYYY-MM-DD): 1990-03-22
Enter Nationality: Indian
Enter Website: http://rajmarts.com
Enter Contact Information: raj@gmail.com
Artist Added Successfully!
```

```
Enter you choice:2
Enter Artist ID to update: 4
Please leave the field empty to retain existing value
Enter New Artist Name [Raj]:
Enter New Biography [Indian artist known for vibrant colors]:
Enter New Birth Date (YYYY-MM-DD) [1990-03-22]: 1992-03-02
Enter New Nationality: [Indian]:
Enter Website: [http://rajmarts.com]:
Enter Contact Information: [raj@gmail.com]:
Artist Updated Successfully
```

```
Enter you choice:3
Enter Artist ID to delete: 3
Are you sure you want to delete Artist 3? [y/N] y
Artist deleted successfully
```

```
Enter you choice:4
Enter Artist ID to fetch: 4

--------- Artist Details ---------
ID: 4
Name: Raj
Biography: Indian artist known for vibrant colors
Nationality: Indian
Website: http://rajmarts.com
Contact Information: raj@gmail.com
Active Status: Active
```

```
 ------- Artist Menu -------

 1. Add Artist

 2. Update Artist

 3. Delete Artist

 4. Get Artist By ID

 5. Reactivate Artist

 0. Exit
Enter you choice:5
Enter Artist ID to reactivate: 3
Artist account Reactivated Successfully
```

ARTWORK:

```
------- Artwork Menu -------

 1. Add Artwork

 2. Update Artwork

 3. Delete Artwork

 4. Get Artwork By ID

 5. Search Artwork

 0. Exit
Enter you choice:1
Enter Artwork ID: 5
Enter Artist ID: 4
Enter Artwork Title: The Thinker Reimagined
Enter Artwork Description: Modern take on classic sculpture.
Enter Creation Date (YYYY-MM-DD): 2018-06-30
Enter Medium Type: Sculpture
Enter Image URL: http://images.com/thinker.jpg
Artwork Added Successfully
```

```
Enter you choice:2
Enter Artwork ID to update: 5
Please leave the field empty to retain existing value
Enter New Title [The Thinker Reimagined]: Thinker Reimagined
Enter New Description [Modern take on classic sculpture.]:
Enter New Creation Date (YYYY-MM-DD) [2018-06-30]:
Enter New Medium: [Sculpture]:
Enter Image URL: [http://images.com/thinker.jpg]:
Enter Artist ID: [4]:
Artwork Updated Successfully
```

```
Enter you choice:3
Enter Artwork ID to delete: 5
Are you sure you want to delete Artwork 5? [y/N] y
Artwork deleted successfully
```

```
Enter you choice:4
Enter Artwork ID to fetch: 2

--------- Artist Details ---------
ID: 2
Title: Ray
Description: Hope of Ray
Creation Date: 2022-03-04
Medium Oil on Canvas
Image URL: https://ray.com
Artist ID: 1
```

```
 Search Artworks By:
1. Title
2. Artist ID
3. Medium
Enter your choice (1/2/3): 3
Enter keyword to search in Medium: Acrylic

--------- Matching Artworks ---------

ID: 6
Title: City Reflections
Description: An abstract interpretation of city life
Creation Date: 2020-05-10
Medium: Acrylic
Image URL: http://images.com/city_reflections.jpg
Artist ID: 3

ID: 7
Title: Festival of Lights
Description: Celebration captured with vibrant colors
Creation Date: 2023-10-24
Medium: Acrylic
Image URL: http://images.com/festival_lights.jpg
Artist ID: 3
```

GALLERY:

```
------- Gallery Menu -------

1. Add Gallery

2. Update Artwork

3. Get Gallery By ID

4. Delete Gallery

5. Search Gallery

0. Exit
Enter you choice:1
Enter Gallery ID: 3
Enter Gallery Name: Urban Art Gallery
Enter Gallery Description: Focuses on contemporary city-themed artwork
Enter Gallery Location: Venice
Enter Gallery Curator: 3
Enter Gallery Opening Hours: 10:00 AM - 6:00 PM
Gallery Added Successfully
```

```
------- Gallery Menu -------

1. Add Gallery

2. Update Artwork

3. Get Gallery By ID

4. Delete Gallery

5. Search Gallery

0. Exit
Enter you choice:2
Enter Gallery ID to update: 4
Enter new name [Sculpture Space]:
Enter new description [Dedicated to modern sculptures]:
Enter new location [India]:
Enter new curator ID [2]:
Enter new opening hours [11:00 AM - 5:00 PM]: 10 AM - 4 PM
Gallery updated successfully
```

```
Enter you choice:3
Enter Gallery ID to fetch: 4

Gallery ID: 4
Name: Sculpture Space
Description: Dedicated to modern sculptures
Location: India
Curator: 2
Opening Hours: 10 AM - 4 PM
```

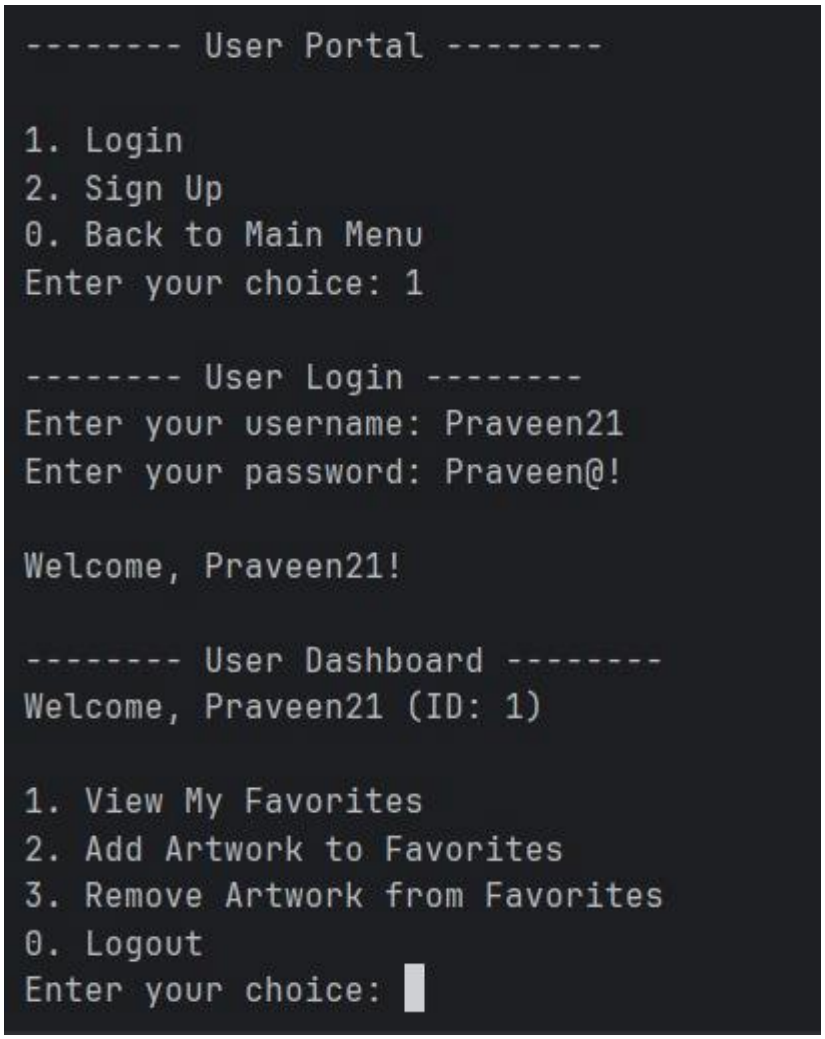
```
Enter you choice:4
Enter Gallery ID to delete: 4
Are you sure you want to delete Gallery 4? [y/N] y
Gallery deleted successfully
```

```
 Search Galleries By:
1. Name
2. Location
3. Curator ID
Enter your choice (1/2/3): 1
Enter keyword to search in Name: Urban Art Gallery

--------- Matching Galleries ---------

ID: 3
Name: Urban Art Gallery
Description: Focuses on contemporary city-themed artwork
Location: Venice
Curator ID: 3
Opening Hours: 10:00 AM - 6:00 PM
```

USER:


-------- User Portal --------
1. Login
2. Sign Up
0. Back to Main Menu
Enter your choice: 1
-------- User Login --------
Enter your username: Praveen21
Enter your password: Praveen@!
Welcome, Praveen21!
-------- User Dashboard --------
Welcome, Praveen21 (ID: 1)
1. View My Favorites
2. Add Artwork to Favorites
3. Remove Artwork from Favorites
0. Logout
Enter your choice:

```
------- User Menu -------

1. Add User

2. Update User

3. Get User By ID

4. Delete User

5. Reactivate User

0. Exit
Enter your choice: 1
Enter User ID: 5
Enter Username: artlover@example.com
Enter Password: Alice
Enter Email: artlover@example.com
Enter First Name: Alice
Enter Last Name: Green
Enter Date of Birth (YYYY-MM-DD): 2000-01-15
Enter Profile Picture URL: https://alice_img.jpeg
User Added Successfully!
```

```
Enter your choice: 2
Enter User ID to update: 6
Leave fields empty to keep existing values.
Enter New Username [urbanexplorer]:
Enter New Password [urban2024]:
Enter New Email [urban@gmail.com]:
Enter New First Name [Will]:
Enter New Last Name [None]: James
Enter New Date of Birth (YYYY-MM-DD) [1995-06-20]:
Enter New Profile Picture URL [None]: https://image.png
User updated successfully!
```

```
Enter your choice: 3
Enter User ID to fetch: 2

--------- User Details ---------
ID: 2
Username: Tejashree
Email: teja@gmail.com
First Name: Tejashree
Last Name: Ganesan
Date of Birth: 2003-01-04
Profile Picture: N/A
Active Status: Active
```

```
Enter your choice: 4
Enter User ID to delete: 4
Are you sure you want to delete User 4? [y/N] y
User deleted successfully
```

```
Enter your choice: 5
Enter User ID to reactivate: 4
User account Reactivated Successfully
```

USER FAVOURITE:

```
------- User Favorite Menu -------

1. Add Artwork to Favorite

2. Remove Artwork from Favorite

3. Get User favorite Artworks by ID

0. Exit
Enter your choice: 1
Enter User ID: 5
Enter Artwork ID to favorite: 2
Artwork added to favorites successfully.
```

```
 0. Exit
 Enter your choice: 2
 Enter User ID: 5
 Enter Artwork ID to remove from favorites: 2
 Artwork removed from favorites successfully.
```

```
Enter your choice: 3
Enter User ID to fetch favorite artworks: 1

------ Favorite Artworks ------

ID: 1
Title: Sky Whisperer
Description: A fantasy-themed digital illustration of a girl riding clouds with bird
s
Medium: Digital Art
Creation Date: 2022-03-03
Artist ID: 1
```

ARTWORK GALLERY:

```
 ------- Artwork Gallery Menu -------

 1. Add Artwork to Gallery

 2. Remove Artwork from Gallery

 3. Get Artworks by Gallery ID

 0. Exit
Enter your choice: 1
Enter Artwork ID: 2
Enter Gallery ID: 3
Artwork successfully added to the gallery
```

```
  ------- Artwork Gallery Menu -------

  1. Add Artwork to Gallery

  2. Remove Artwork from Gallery

  3. Get Artworks by Gallery ID

  0. Exit
 Enter your choice: 2
 Enter Artwork ID: 2
 Enter Gallery ID: 3
 Artwork removed from gallery successfully
```

```
------- Artwork Gallery Menu -------

1. Add Artwork to Gallery

2. Remove Artwork from Gallery

3. Get Artworks by Gallery ID

0. Exit
Enter your choice: 3
Enter Gallery ID to fetch artworks: 1

ID: 1
Title: Sky Whisperer
Medium: Digital Art
```

PYTEST:

```
PS E:\Virtual_Art_Gallery> pytest -v
============================= test session starts ==============================
platform win32 -- Python 3.11.5, pytest-8.4.1, pluggy-1.6.0 -- C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: E:\Virtual_Art_Gallery
plugins: anyio-4.4.0
collected 18 items

tests/test_artist_management.py::test_add_artist PASSED                  [  5%]
tests/test_artist_management.py::test_update_artist PASSED               [ 11%]
tests/test_artist_management.py::test_delete_artist PASSED               [ 16%]
tests/test_artist_management.py::test_get_artist_by_id PASSED            [ 22%]
tests/test_artist_management.py::test_reactivate_artist PASSED           [ 27%]
tests/test_artwork_management.py::test_add_artwork PASSED                [ 33%]
tests/test_artwork_management.py::test_update_artwork PASSED             [ 38%]
tests/test_artwork_management.py::test_delete_artwork PASSED             [ 44%]
tests/test_artwork_management.py::test_search_for_artwork PASSED         [ 50%]
tests/test_gallery_management.py::test_add_gallery PASSED                [ 55%]
tests/test_gallery_management.py::test_update_gallery PASSED             [ 61%]
tests/test_gallery_management.py::test_delete_gallery PASSED             [ 66%]
tests/test_gallery_management.py::test_search_for_artwork PASSED         [ 72%]
tests/test_user_management.py::test_add_user PASSED                      [ 77%]
tests/test_user_management.py::test_update_user PASSED                   [ 83%]
tests/test_user_management.py::test_delete_user PASSED                   [ 88%]
tests/test_user_management.py::test_get_user_by_id PASSED                [ 94%]
tests/test_user_management.py::test_reactivate_user PASSED               [100%]

============================== 18 passed in 0.37s ==============================
PS E:\Virtual_Art_Gallery>
```