

**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)**

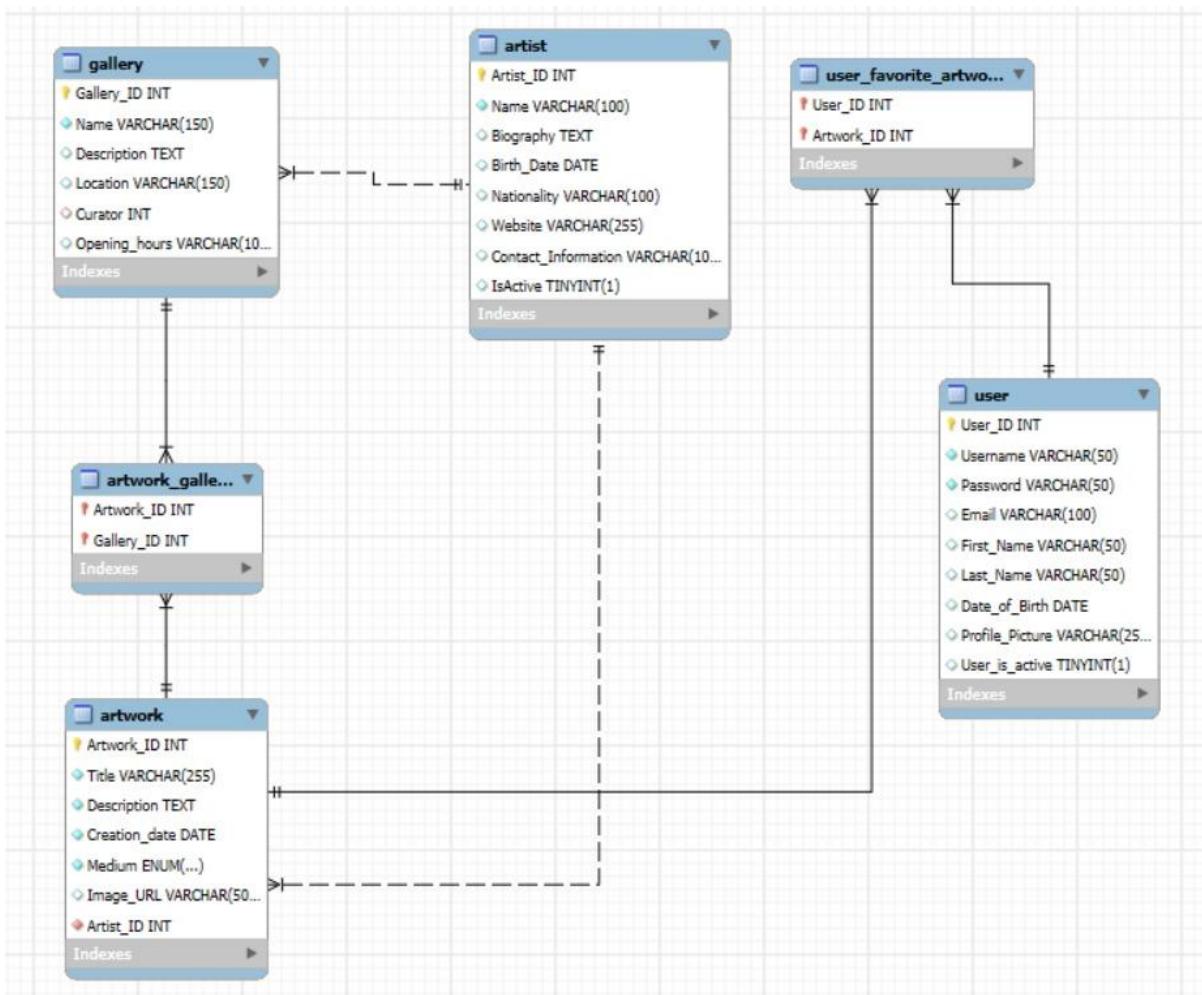
An artwork can be displayed in multiple galleries, and a gallery can have multiple artworks.

Artwork_Gallery (junction table):

ArtworkID (Foreign Key) references Artwork.ArtworkID.

GalleryID (Foreign Key) references Gallery.GalleryID.

ENTITY RELATIONSHIP DIAGRAM:



ENTITY CLASS

•CLASS ARTIST:

```
1  ass Artist:
2      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):
3          self._artist_id = artist_id
4          self._name = name
5          self._biography = biography
6          self._birth_date = birth_date
7          self._nationality = nationality
8          self._website = website
9          self._contact_information = contact_information
10         self._is_active = is_active
11         self._password = password
12
13     @property
14     def artist_id(self):
15         return self._artist_id
16     @artist_id.setter
17     def artist_id(self, artist_id):
18         self._artist_id = artist_id
19
20     @property
21     def name(self):
22         return self._name
23     @name.setter
24     def name(self, name):
25         self._name = name
26
27     @property
28     def biography(self):
29         return self._biography
30     @biography.setter
31     def biography(self, biography):
32         self._biography = biography
33
34     @property
35     def birth_date(self):
36         return self._birth_date
37     @birth_date.setter
38     def birth_date(self, birth_date):
39         self._birth_date = birth_date
40
41     @property
42     def nationality(self):
43         return self._nationality
44     @nationality.setter
45     def nationality(self, nationality):
46         self._nationality = nationality
47
```

```
48     @property
49     def website(self):
50         return self._website
51     @website.setter
52     def website(self, website):
53         self._website = website
54
55     @property
56     def contact_information(self):
57         return self._contact_information
58     @contact_information.setter
59     def contact_information(self, contact_information):
60         self._contact_information = contact_information
61
62     @property
63     def is_active(self):
64         return self._is_active
65     @is_active.setter
66     def is_active(self, is_active):
67         self._is_active = is_active
68
69     @property
70     def password(self):
71         return self._password
72     @password.setter
73     def password(self, password):
74         self._password = password
```

•CLASS ARTWORK:

```
1  class Artwork:
2      def __init__(self, artwork_id = None, title = None, description= None, creation_date = None,
3                   | medium=None,image_url = None, artist_id = None):
4          self.__artwork_id = artwork_id
5          self.__title = title
6          self.__description = description
7          self.__creation_date = creation_date
8          self.__medium = medium
9          self.__image_url = image_url
10         self.__artist_id = artist_id
11
12     @property
13     def artwork_id(self):
14         return self.__artwork_id
15     @artwork_id.setter
16     def artwork_id(self, artwork_id):
17         self.__artwork_id = artwork_id
18
19     @property
20     def title(self):
21         return self.__title
22     @title.setter
23     def title(self, title):
24         self.__title = title
25
26     @property
27     def description(self):
28         return self.__description
29     @description.setter
30     def description(self, description):
31         self.__description = description
32
33     @property
34     def creation_date(self):
35         return self.__creation_date
36     @creation_date.setter
37     def creation_date(self, creation_date):
38         self.__creation_date = creation_date
```

```
40     @property
41     def medium(self):
42         return self.__medium
43     @medium.setter
44     def medium(self, medium):
45         self.__medium = medium
46
47     @property
48     def image_url(self):
49         return self.__image_url
50     @image_url.setter
51     def image_url(self, image_url):
52         self.__image_url = image_url
53
54     @property
55     def artist_id(self):
56         return self.__artist_id
57     @artist_id.setter
58     def artist_id(self, artist_id):
59         self.__artist_id = artist_id
```

•CLASS USER:

```
1  class User:
2      def __init__(self, user_id=None, username=None, password=None, email=None, first_name=None, last_name=None,
3                  date_of_birth=None, profile_picture=None, user_is_active = None):
4          self._user_id = user_id
5          self._username = username
6          self._password = password
7          self._email = email
8          self._first_name = first_name
9          self._last_name = last_name
10         self._date_of_birth = date_of_birth
11         self._profile_picture = profile_picture
12         self._user_is_active = user_is_active
13
14     @property
15     def user_id(self):
16         return self._user_id
17
18     @user_id.setter
19     def user_id(self, user_id):
20         self._user_id = user_id
21
22     @property
23     def username(self):
24         return self._username
25
26     @username.setter
27     def username(self, username):
28         self._username = username
29
30     @property
31     def password(self):
32         return self._password
33
34     @password.setter
35     def password(self, password):
36         self._password = password
37
38     @property
39     def email(self):
40         return self._email
41
42     @email.setter
43     def email(self, email):
44         self._email = email
45
46     @property
47     def first_name(self):
48         return self._first_name
49
50     @first_name.setter
51     def first_name(self, first_name):
52         self._first_name = first_name
53
54     @property
55     def last_name(self):
56         return self._last_name
57
58     @last_name.setter
59     def last_name(self, last_name):
60         self._last_name = last_name
61
62     @property
63     def date_of_birth(self):
64         return self._date_of_birth
65
66     @date_of_birth.setter
67     def date_of_birth(self, date_of_birth):
68         self._date_of_birth = date_of_birth
69
70     @property
71     def profile_picture(self):
72         return self._profile_picture
73
74     @profile_picture.setter
75     def profile_picture(self, profile_picture):
76         self._profile_picture = profile_picture
77
78     @property
79     def user_is_active(self):
80         return self._user_is_active
81     @user_is_active.setter
82     def user_is_active(self, user_is_active):
83         self._user_is_active = user_is_active
```

•CLASS USER FAVOURITE:

```
1  class UserFavorite:
2      def __init__(self, user_id=None, artwork_id=None):
3          self.__user_id = user_id
4          self.__artwork_id = artwork_id
5
6      @property
7      def user_id(self):
8          return self.__user_id
9
10     @user_id.setter
11     def user_id(self, user_id):
12         self.__user_id = user_id
13
14     @property
15     def artwork_id(self):
16         return self.__artwork_id
17
18     @artwork_id.setter
19     def artwork_id(self, artwork_id):
20         self.__artwork_id = artwork_id
21
```

•CLASS ARTWORK GALLERY:

```
Virtual_Art_Gallery-main > entity > 🖼 artwork_gallery.py > ArtworkGallery > gallery_id
1  class ArtworkGallery:
2
3      def __init__(self, artwork_id: int = None, gallery_id: int = None):
4          self.__artwork_id = artwork_id
5          self.__gallery_id = gallery_id
6
7      @property
8      def artwork_id(self) -> int:
9          return self.__artwork_id
10
11     @artwork_id.setter
12     def artwork_id(self, artwork_id: int):
13         self.__artwork_id = artwork_id
14
15     @property
16     def gallery_id(self) -> int:
17         return self.__gallery_id
18
19     @gallery_id.setter
20     def gallery_id(self, gallery_id: int):
21         self.__gallery_id = gallery_id
```

•CLASS GALLERY:

```
Virtual_Art_Gallery-main > entity > gallery.py > Gallery
 1  class Gallery:
 2      def __init__(self, gallery_id = None, name= None, description= None, location= None, curator= None, opening_hours= None):
 3          self.__gallery_id = gallery_id
 4          self.__name = name
 5          self.__description = description
 6          self.__location = location
 7          self.__curator = curator
 8          self.__opening_hours = opening_hours
 9
10      @property
11      def gallery_id(self):
12          return self.__gallery_id
13      @gallery_id.setter
14      def gallery_id(self, gallery_id):
15          self.__gallery_id = gallery_id
16
17      @property
18      def name(self):
19          return self.__name
20      @name.setter
21      def name(self, name):
22          self.__name = name
23
24      @property
25      def description(self):
26          return self.__description
27      @description.setter
28      def description(self, description):
29          self.__description = description
```

DAO

•Service Provider interface (ARTIST INTERFACE)

```
 1  from abc import ABC, abstractmethod
 2
 3  class IVirtualArtGallery(ABC):
 4      """ Artist Interface """
 5      @abstractmethod
 6      def add_artist(self, artist):
 7          pass
 8
 9      @abstractmethod
10      def update_artist(self, artist):
11          pass
12
13      @abstractmethod
14      def get_artist_by_id(self, artist_id):
15          pass
16      @abstractmethod
17      def remove_artist(self, artist):
18          pass
19
20      @abstractmethod
21      def reactivate_artist(self, artist):
22          pass
23
24      """ Artwork Interface """
25      @abstractmethod
26      def add_artwork(self, artwork):
27          pass
28
29      @abstractmethod
30      def update_artwork(self, artwork):
31          pass
32
33      @abstractmethod
34      def get_artwork_by_id(self, artwork_id):
35          pass
36
37      @abstractmethod
38      def remove_artwork(self, artwork):
39          pass
40
41      @abstractmethod
42      def search_artworks(self, artwork):
43          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

```
1  from datetime import date
2  import re
3  from dao.IVirtualArtGallery import IVirtualArtGallery
4  from entity.artwork import Artwork
5  from entity.artist import Artist
6  from entity.gallery import Gallery
7  from entity.user import User
8  from util.DBConnUtil import DBConnection
9  from exception.exceptions import (InvalidIDException, InvalidArtistNameException, # Artist
10                                         InvalidDateException, InvalidWebsiteException,
11                                         InvalidArtworkException, InvalidMediumTypeException, # Artwork
12                                         DuplicateArtworkException, ArtworkDoesNotExistException,
13                                         InvalidGalleryNameException, InvalidOpeningHoursException, # Gallery
14                                         DuplicateGalleryException, GalleryAssignmentException, GalleryNotFoundException,
15                                         InvalidUsernameException, InvalidPasswordException, InvalidEmailException, # User
16                                         DuplicateUserException, UserNotFoundException, FavoriteAlreadyExistsException,
17                                         FavoriteNotFoundException) # FavoriteAlreadyExistsException
18
19
20 class VirtualArtGalleryImp(IVirtualArtGallery):
21     def __init__(self, connection=None):
22         self.connection = connection or DBConnection.get_connection()
```

```

24     """ Artist Implementation"""
25
26     def validate_artist_fields(self, artist):
27         if artist.name is not None and artist.name.strip() == "":
28             raise InvalidArtistNameException("Artist Name cannot be empty")
29
30         if artist.birth_date is not None and artist.birth_date > date.today():
31             raise InvalidDateException("Birth date cannot be in the future")
32
33         if artist.website is not None and not re.match(r"^(https?://[^s]+)$", artist.website):
34             raise InvalidWebsiteException("Website must be a valid URL")
35
36     def add_artist(self, artist):
37         if artist.artist_id is None:
38             raise InvalidIDException("Artist ID is required")
39         self.validate_artist_fields(artist)
40
41         cursor = self.connection.cursor()
42         cursor.execute("SELECT * from artist where Artist_ID = %s", (artist.artist_id,))
43         if cursor.fetchone():
44             raise InvalidIDException("Artist ID is already in use")
45
46         query = """
47             INSERT INTO Artist
48                 (Artist_ID, Name, Biography, Birth_Date, Nationality, Website, Contact_Information, IsActive)
49             VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
50             """
51         cursor.execute(query, (artist.artist_id, artist.name, artist.biography or None, artist.birth_date or None, artist.nationality or None,
52                               artist.website or None, artist.contact_information or None, artist.is_active))
53         self.connection.commit()
54         return cursor.rowcount > 0
55
56     def update_artist(self, artist):
57         cursor = None
58         if artist.artist_id is None:
59             raise InvalidIDException("Artist ID is required")
60
61         cursor = self.connection.cursor()
62         cursor.execute("SELECT * from artist where Artist_ID = %s", (artist.artist_id,))
63         existing = cursor.fetchone()
64         if not existing:
65             raise InvalidIDException("Artist with this ID does not exist")
66
67         self.validate_artist_fields(artist)
68         query = """
69             UPDATE Artist SET
70                 Name = %s,
71                 Biography = %s,
72                 Birth_Date = %s,
73                 Nationality = %s,
74                 Website = %s,
75                 Contact_Information = %s
76                 WHERE Artist_ID = %s
77             """
78         cursor.execute(query, (artist.name or existing[1], artist.biography or existing[2], artist.birth_date or existing[3], artist.nationality or existing[4],
79                               artist.website or existing[5], artist.contact_information or existing[6], artist.artist_id))
80         self.connection.commit()
81         return cursor.rowcount > 0
82
83     def get_artist_by_id(self, artist_id):
84         cursor = None
85         cursor = self.connection.cursor()
86         cursor.execute("SELECT * from artist where Artist_ID = %s", (artist_id,))
87         row = cursor.fetchone()
88
89         if row:
90             return Artist(
91                 artist_id = row[0],
92                 name = row[1],
93                 biography = row[2],
94                 birth_date = row[3],
95                 nationality = row[4],
96                 website = row[5],
97                 contact_information = row[6],
98                 is_active = row[7]
99             )
100        else:
101            return None
102
103    def remove_artist(self, artist_id):
104        if artist_id is None:
105            raise InvalidIDException("Artist ID is required for deletion")
106        cursor = self.connection.cursor()
107        cursor.execute("SELECT * from artist where Artist_ID = %s", (artist_id,))
108        if not cursor.fetchone():
109            raise InvalidIDException("Artist with this ID does not exist")
110        query="UPDATE Artist SET IsActive = FALSE WHERE Artist_ID = %s"
111        cursor.execute(query, (artist_id,))
112        self.connection.commit()
113        return cursor.rowcount > 0
114
115    def reactivate_artist(self, artist_id):
116        cursor = self.connection.cursor()
117        cursor.execute("SELECT IsActive from artist where Artist_ID = %s", (artist_id,))
118        row = cursor.fetchone()
119        if not row:
120            return "not_found"
121        isActive = row[0]
122        if isActive:
123            return "already_active"
124        cursor.execute("UPDATE Artist SET IsActive = TRUE WHERE Artist_ID = %s", (artist_id,))
125        self.connection.commit()
126        return 'reactivated'

```

Artwork Implementation

```
127     """Artwork Implementation"""
128     def validate_artwork_fields(self, artwork):
129         if artwork.title is not None and artwork.title.strip() == "":
130             raise InvalidArtworkException("Title is required")
131         if artwork.description is not None and artwork.description == "":
132             raise InvalidArtworkException("Description is required")
133         if artwork.creation_date is not None and artwork.creation_date > date.today():
134             raise InvalidDateException("Creation date cannot be null or in the future")
135         if artwork.medium is not None and artwork.medium not in self.valid_mediums:
136             raise InvalidMediumTypeException(f"Medium type cannot be none and it should be one of {', '.join(self.valid_mediums)}")
137         if artwork.image_url is not None and not re.match(r"^(https?://[\^s]+)$", artwork.image_url):
138             raise InvalidWebsiteException("Website must be a valid URL")
139
140     valid_mediums = ["Oil on Canvas", "Watercolor", "Acrylic", "Digital Art", "Sculpture", "Mixed Media"]
141     def add_artwork(self, artwork):
142         self.validate_artwork_fields(artwork)
143         cursor = self.connection.cursor()
144         cursor.execute("SELECT * from Artwork where Artwork_ID = %s", (artwork.artwork_id,))
145         if cursor.fetchone():
146             raise InvalidIDException("Artwork ID is already in use")
147
148         cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (artwork.artist_id,))
149         if not cursor.fetchone():
150             print("Artist ID does not exist.")
151             return False
152
153         cursor.execute("SELECT * FROM Artwork WHERE Artist_ID = %s AND title = %s", (artwork.artist_id,artwork.title))
154         if cursor.fetchone():
155             raise DuplicateArtworkException(f"Artwork with title '{artwork.title}' already exists for this artist.")
156
157         cursor.execute("""INSERT INTO Artwork (Artwork_id, Title, Description, Creation_Date, Artist_ID, Medium, Image_url)
158 VALUES (%s, %s, %s, %s, %s, %s,%s)""",
159         (artwork.artwork_id, artwork.title, artwork.description,
160         artwork.creation_date, artwork.artist_id, artwork.medium, artwork.image_url))
161         self.connection.commit()
162         return cursor.rowcount > 0
163
164     def update_artwork(self, artwork):
165         if artwork.artwork_id is None:
166             raise InvalidIDException("Artwork ID is required")
167
168         cursor = self.connection.cursor()
169         cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork.artwork_id,))
170         existing = cursor.fetchone()
171         if not existing:
172             raise InvalidIDException("Artwork with this ID does not exist")
173
174         self.validate_artwork_fields(artwork)
175         query = """
176             UPDATE Artwork SET
177                 Title = %s,
178                 Description = %s,
179                 Creation_date = %s,
180                 Medium = %s,
181                 Image_URL = %s,
182                 Artist_ID = %s
183             WHERE Artwork_ID = %s
184             """
185         cursor.execute(query, (artwork.title or existing[1],
186             artwork.description or existing[2],
187             artwork.creation_date or existing[3],
188             artwork.medium or existing[4],
189             artwork.image_url or existing[5],
190             artwork.artist_id or existing[6],
```

```
195     def get_artwork_by_id(self, artwork_id):
196         cursor = self.connection.cursor()
197         cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork_id,))
198         row = cursor.fetchone()
199         if row:
200             return Artwork(
201                 artwork_id = row[0],
202                 title = row[1],
203                 description = row[2],
204                 creation_date = row[3],
205                 medium = row[4],
206                 image_url = row[5],
207                 artist_id = row[6]
208             )
209         else:
210             raise ArtworkDoesNotExistException(f"Artwork with ID '{artwork_id}' does not exist")
211
212     def remove_artwork(self, artwork_id):
213         cursor = None
214         if artwork_id is None:
215             raise InvalidIDException("Artwork ID is required")
216         cursor = self.connection.cursor()
217         cursor.execute("SELECT * from artwork where Artwork_ID = %s", (artwork_id,))
218         if not cursor.fetchone():
219             raise InvalidIDException("Artwork with this ID does not exist")
220         cursor.execute("DELETE FROM Artwork WHERE Artwork_ID = %s", (artwork_id,))
221         self.connection.commit()
222         return cursor.rowcount > 0
223
224     def search_artworks(self, keyword, search_by):
225         cursor = self.connection.cursor()
226         field_map = {
227             'title': 'Title',
228             'artist_id': 'Artist_ID',
229             'medium': 'Medium'
230         }
231         if search_by not in field_map:
232             raise ValueError("Invalid search criteria. Choose from: title, medium, artist_id")
233
234         column = field_map[search_by]
235         if search_by == 'artist':
236             if not keyword.strip().isdigit():
237                 print("Artist ID must be a number.")
238                 return []
239             artist_id = int(keyword.strip())
240             query = "SELECT * FROM Artwork WHERE Artist_ID = %s"
241             cursor.execute(query, (artist_id,))
242         else:
243             like_pattern = f"%{keyword}%""
244             query = f"SELECT * FROM Artwork WHERE {column} LIKE %s"
245             cursor.execute(query, (like_pattern,))
246             rows = cursor.fetchall()
247             artworks = []
248             for row in rows:
249                 artworks.append(Artwork(
250                     artwork_id = row[0],
251                     title = row[1],
252                     description = row[2],
253                     creation_date = row[3],
254                     medium = row[4],
255                     image_url = row[5],
256                     artist_id = row[6]
257                 ))
258             return artworks
```

Gallery Implementation

```
260     """ Gallery Implementation """
261
262     def validate_gallery_fields(self, gallery):
263         if gallery.gallery_id is None or str(gallery.gallery_id).strip() == "":
264             raise InvalidIDException("Gallery ID is required")
265         if gallery.name is None or gallery.name.strip() == "":
266             raise InvalidGalleryNameException("Gallery Name is required")
267         if gallery.curator is None or str(gallery.curator).strip() == "":
268             raise GalleryAssignmentException("Curator ID cannot be empty")
269         if gallery.opening_hours:
270             hours = gallery.opening_hours.strip()
271             if hours.lower() == "closed":
272                 return
273             if "-" not in hours or not ("AM" in hours.upper() or "PM" in hours.upper()):
274                 raise InvalidOpeningHoursException("Invalid format for Opening Hours")
275             parts = hours.split("-")
276             if len(parts) != 2:
277                 raise InvalidOpeningHoursException(
278                     "Opening Hours format should be 'hh:mm AM/PM - hh:mm AM/PM' or 'Closed'")
279
280     def add_gallery(self, gallery):
281         self.validate_gallery_fields(gallery)
282         cursor = self.connection.cursor()
283         cursor.execute("SELECT * FROM Gallery where Gallery_ID = %s", (gallery.gallery_id,))
284         if cursor.fetchone():
285             raise InvalidIDException("Gallery with this ID already exists")
286         cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (gallery.curator,))
287         if not cursor.fetchone():
288             raise InvalidIDException(f"Curator with ID {gallery.curator} does not exist")
289         cursor.execute("SELECT * FROM Gallery where Gallery_ID = %s AND Name = %s", (gallery.gallery_id, gallery.name))
290         if cursor.fetchone():
291             raise DuplicateGalleryException(f"Gallery with this name '{gallery.name}' already exists")
292         cursor.execute("""INSERT INTO Gallery
293             (Gallery_ID, Name, Description, Location, Curator, Opening_Hours)
294             VALUES (%s, %s, %s, %s, %s, %s)""",
295             (gallery.gallery_id, gallery.name, gallery.description or None, gallery.location or None, gallery.curator or None, gallery.opening_hours or None))
296         self.connection.commit()
297         return cursor.rowcount > 0
298
299     def get_gallery_by_id(self, gallery_id):
300         cursor = self.connection.cursor()
301         cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
302         row = cursor.fetchone()
303         if row:
304             return Gallery(
305                 gallery_id=row[0],
306                 name=row[1],
307                 description=row[2],
308                 location=row[3],
309                 curator=row[4],
310                 opening_hours=row[5]
311             )
312         else:
313             return None
314
315     def update_gallery(self, gallery):
316         if gallery.gallery_id is None or str(gallery.gallery_id).strip() == "":
317             raise InvalidIDException("Gallery ID is required")
318
319         cursor = self.connection.cursor()
320         cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery.gallery_id,))
321         existing = cursor.fetchone()
322         if not existing:
323             raise GalleryNotFoundException("Gallery with this ID does not exist")
```

```

325     if gallery.curator:
326         cursor.execute("SELECT * FROM Artist WHERE Artist_ID = %s", (gallery.curator,))
327         if not cursor.fetchone():
328             raise InvalidIDException(f"Curator with ID {gallery.curator} does not exist")
329
330     self.validate_gallery_fields(gallery)
331
332     query = """
333         UPDATE Gallery SET
334             Name = %s,
335             Description = %s,
336             Location = %s,
337             Curator = %s,
338             Opening_Hours = %s
339         WHERE Gallery_ID = %s
340     """
341     cursor.execute(query, (
342         gallery.name or existing[1],
343         gallery.description or existing[2],
344         gallery.location or existing[3],
345         gallery.curator or existing[4],
346         gallery.opening_hours or existing[5],
347         gallery.gallery_id
348     ))
349     self.connection.commit()
350     return cursor.rowcount > 0
351
352     def remove_gallery(self, gallery_id):
353         if gallery_id is None or str(gallery_id).strip() == "":
354             raise InvalidIDException("Gallery ID is required for deletion")
355
356         cursor = self.connection.cursor()
357         cursor.execute("SELECT * FROM Gallery WHERE Gallery_ID = %s", (gallery_id,))
358         if not cursor.fetchone():
359             raise GalleryNotFoundException("Gallery with this ID does not exist")
360
361         query = "DELETE FROM Gallery WHERE Gallery_ID = %s"
362         cursor.execute(query, (gallery_id,))
363         self.connection.commit()
364         return cursor.rowcount > 0
365
366     def search_gallery(self, keyword, search_by):
367         cursor = self.connection.cursor()
368         field_map = {
369             'name': 'Name',
370             'location': 'Location',
371             'curator': 'Curator'
372         }
373         if search_by not in field_map:
374             raise ValueError("Invalid search criteria. Choose from: name, location, curator")
375
376         column = field_map[search_by]
377
378         if search_by == 'curator':
379             if not keyword.strip().isdigit():
380                 print("Curator ID must be a number.")
381             return []
382             query = "SELECT * FROM Gallery WHERE Curator = %s"
383             cursor.execute(query, (int(keyword.strip()),))
384         else:
385             like_pattern = f"%{keyword}%""
386             query = f"SELECT * FROM Gallery WHERE {column} LIKE %s"
387             cursor.execute(query, (like_pattern,))
388
389         rows = cursor.fetchall()
390         galleries = []
391         for row in rows:
392             galleries.append(Gallery(
393                 gallery_id=row[0],
394                 name=row[1],
395                 description=row[2],
396                 location=row[3],
397                 curator=row[4],
398                 opening_hours=row[5]
399             ))
400     return galleries

```

User Implementation

```
482     """ User Implementation """
483     def validate_user_fields(self, user):
484         if user.user_id is None or str(user.user_id).strip() == "":
485             raise InvalidIDException("User ID is required")
486         if user.username is None or user.username.strip() == "":
487             raise InvalidUsernameException("Username is required")
488         if user.password is None or user.password.strip() == "":
489             raise InvalidPasswordException("Password is required")
490         if user.email:
491             email_pattern = r'^[\w\.-]+@[^\w\.-]+\.\w+$'
492             if not re.match(email_pattern, user.email):
493                 raise InvalidEmailException("Invalid email format")
494         if user.date_of_birth and user.date_of_birth > date.today():
495             raise InvalidDEException("Date of birth cannot be in the future")
496         if user.profile_picture is not None and not re.match(r"^(https?://[^$]+)", user.profile_picture):
497             raise InvalidWebsiteException("Profile Picture must be a valid URL")
498
499     def add_user(self, user):
500         self.validate_user_fields(user)
501         cursor = self.connection.cursor()
502         cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user.user_id,))
503         if cursor.fetchone():
504             raise InvalidIDException("User ID already exists")
505
506         cursor.execute("SELECT * FROM User WHERE Username = %s OR Email = %s", (user.username, user.email))
507         if cursor.fetchone():
508             raise DuplicateUserException("Username or Email already in use")
509
510         cursor.execute("""
511             INSERT INTO User
512                 (User_ID, Username, Password, Email, First_Name, Last_Name, Date_of_Birth, Profile_Picture, User_is_active)
513             VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
514         """, (
515             user.user_id,
516             user.username,
517             user.password,
518             user.email or None,
519             user.first_name or None,
520             user.last_name or None,
521             user.date_of_birth or None,
522             user.profile_picture or None,
523             user.user_is_active
524         ))
525
526         self.connection.commit()
527         return cursor.rowcount > 0
528
529     def get_user_by_id(self, user_id):
530         cursor = self.connection.cursor()
531         cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
532         row = cursor.fetchone()
533
534         if row:
535             return User(
536                 user_id=row[0],
537                 username=row[1],
538                 password=row[2],
539                 email=row[3],
540                 first_name=row[4],
541                 last_name=row[5],
542                 date_of_birth=row[6],
543                 profile_picture=row[7],
544                 user_is_active=row[8]
545             )
546         else:
547             return None
548
549     def update_user(self, user):
550         if user.user_id is None:
551             raise InvalidIDException("User ID is required")
552
553         cursor = self.connection.cursor()
554         cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user.user_id,))
555         existing = cursor.fetchone()
556
557         if not existing:
558             raise InvalidIDException("User with this ID does not exist")
```

```
480     self.validate_user_fields(user)
481
482     query = """
483     UPDATE User SET
484         Username = %s,
485         Password = %s,
486         Email = %s,
487         First_Name = %s,
488         Last_Name = %s,
489         Date_of_Birth = %s,
490         Profile_Picture = %s,
491         User_is_active = %s
492     WHERE User_ID = %s
493     """
494
495     cursor.execute(query, (
496         user.username or existing[1],
497         user.password or existing[2],
498         user.email or existing[3],
499         user.first_name or existing[4],
500         user.last_name or existing[5],
501         user.date_of_birth or existing[6],
502         user.profile_picture or existing[7],
503         user.user_is_active if user.user_is_active is not None else existing[8],
504         user.user_id
505     ))
506     self.connection.commit()
507     return cursor.rowcount > 0
508
509 def remove_user(self, user_id):
510     if user_id is None:
511         raise InvalidIDException("User ID is required for deletion")
512
513     cursor = self.connection.cursor()
514     cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
515     if not cursor.fetchone():
516         raise InvalidIDException("User with this ID does not exist")
517
518     query = "UPDATE User SET User_is_active = FALSE WHERE User_ID = %s"
519     cursor.execute(query, (user_id,))
520     self.connection.commit()
521     return cursor.rowcount > 0
522
523 def reactivate_user(self, user_id):
524     cursor = self.connection.cursor()
525     cursor.execute("SELECT User_is_active FROM User WHERE User_ID = %s", (user_id,))
526     row = cursor.fetchone()
527     if not row:
528         return "not_found"
529     is_active = row[0]
530     if is_active:
531         return "already_active"
532
533     cursor.execute("UPDATE User SET User_is_active = TRUE WHERE User_ID = %s", (user_id,))
534     self.connection.commit()
535     return "reactivated"
```

User Favorite Implementation

```
137     """ User Favorite Implementation """
138
139     def add_artwork_to_favorite(self, user_id, artwork_id):
140         if user_id is None or artwork_id is None:
141             raise InvalidIDException("User ID and Artwork ID are required")
142         cursor = self.connection.cursor()
143
144         cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
145         if not cursor.fetchone():
146             raise UserNotFoundException("User with this ID does not exist")
147
148         cursor.execute("SELECT * FROM Artwork WHERE Artwork_ID = %s", (artwork_id,))
149         if not cursor.fetchone():
150             raise ArtworkDoesNotExistException("Artwork with this ID does not exist")
151
152         cursor.execute("SELECT * FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
153                         (user_id, artwork_id))
154         if cursor.fetchone():
155             raise FavoriteAlreadyExistsException("Favorite Already Exists")
156
157         cursor.execute("INSERT INTO User_Favorite_Artwork (User_ID, Artwork_ID) VALUES (%s, %s)", (user_id, artwork_id))
158         self.connection.commit()
159         return cursor.rowcount > 0
160
161     def remove_artwork_from_favorite(self, user_id, artwork_id):
162         if user_id is None or artwork_id is None:
163             raise InvalidIDException("User ID and Artwork ID are required")
164
165         cursor = self.connection.cursor()
166
167         cursor.execute("SELECT * FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
168                         (user_id, artwork_id))
169         if not cursor.fetchone():
170             raise FavoriteNotFoundException("Favorite Artwork with this ID does not exist")
171
172         cursor.execute("DELETE FROM User_Favorite_Artwork WHERE User_ID = %s AND Artwork_ID = %s",
173                         (user_id, artwork_id))
174         self.connection.commit()
175         return cursor.rowcount > 0
176
177     def get_user_favorite_artworks(self, user_id):
178         if user_id is None:
179             raise InvalidIDException("User ID is required")
180         cursor = self.connection.cursor()
181
182         cursor.execute("SELECT * FROM User WHERE User_ID = %s", (user_id,))
183         if not cursor.fetchone():
184             raise InvalidIDException("User with this ID does not exist")
185
186         query = """
187             SELECT a.Artwork_ID, a.Title, a.Description, a.Creation_date, a.Medium, a.Image_URL, a.Artist_ID
188             FROM Artwork a
189             JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
190             WHERE ufa.User_ID = %s
191             """
192
193         cursor.execute(query, (user_id,))
194         rows = cursor.fetchall()
195         if not rows:
196             return False
197
198         favorite_artworks = []
199         for row in rows:
200             favorite_artworks.append(Artwork(
201                 artwork_id=row[0],
202                 title=row[1],
203                 description=row[2],
204                 creation_date=row[3],
205                 medium=row[4],
206                 image_url=row[5],
207                 artist_id=row[6]
208             ))
209
210         return favorite_artworks
```

Artwork Gallery Implementation

```
""" 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

```
""" 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

```
""" 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")
```

```
742     cursor.execute("""
743         SELECT Artwork_ID, Title, Creation_Date, Medium
744         FROM Artwork
745         WHERE Artist_ID = %s
746     """ , (artist_id,))
747     artist_artworks = cursor.fetchall()
748
749     cursor.execute("""
750         SELECT DISTINCT g.Gallery_ID, g.Name, g.Location
751         FROM Gallery g
752         JOIN Artwork_Gallery ag ON g.Gallery_ID = ag.Gallery_ID
753         JOIN Artwork a ON ag.Artwork_ID = a.Artwork_ID
754         WHERE a.Artist_ID = %s
755     """ , (artist_id,))
756     galleries_with_artworks = cursor.fetchall()
757
758     cursor.execute("""
759         SELECT a.Artwork_ID, a.Title, COUNT(ufa.User_ID) as favorite_count
760         FROM Artwork a
761         LEFT JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
762         WHERE a.Artist_ID = %s
763         GROUP BY a.Artwork_ID, a.Title
764         ORDER BY favorite_count DESC
765     """ , (artist_id,))
766     artworkFavorites = cursor.fetchall()
767
768     return {
769         "artist_id": artist_id,
770         "artist_name": artist_name,
771         "artworks": [
772             {
773                 "artwork_id": artwork[0],
774                 "title": artwork[1],
775                 "creation_date": artwork[2],
776                 "medium": artwork[3]
777             } for artwork in artist_artworks
778         ] if artist_artworks else "No artworks uploaded yet",
779         "galleries_with_artworks": [
780             {
781                 "gallery_id": gallery[0],
782                 "name": gallery[1],
783                 "location": gallery[2]
784             } for gallery in galleries_with_artworks
785         ] if galleries_with_artworks else "No artworks in galleries yet",
786         "artworkFavorites": [
787             {
788                 "artwork_id": fav[0],
789                 "title": fav[1],
790                 "favorite_count": fav[2]
791             } for fav in artworkFavorites
792         ]
793     }
794
795     def artist_signup(self, artist, password):
796         if artist.artist_id is None:
797             raise InvalidIDException("Artist ID is required")
798         if not password:
799             raise InvalidPasswordException("Password is required")
800         self.validate_artist_fields(artist)
801
802         cursor = self.connection.cursor()
803
804         # Check if artist ID already exists
805         cursor.execute("SELECT * FROM artist WHERE Artist_ID = %s", (artist.artist_id,))
```

```
806     if cursor.fetchone():
807         raise InvalidIDException("Artist ID is already in use")
808
809     # Check if artist name already exists
810     cursor.execute("SELECT * FROM artist WHERE Name = %s", (artist.name,))
811     if cursor.fetchone():
812         raise InvalidArtistNameException("Artist name is already in use")
813
814     query = """
815         INSERT INTO Artist
816             (Artist_ID, Name, Biography, Birth_Date, Nationality, Website, Contact_Information, Password, IsActive)
817             VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
818     """
819
820     cursor.execute(query, (
821         artist.artist_id,
822         artist.name,
823         artist.biography or None,
824         artist.birth_date or None,
825         artist.nationality or None,
826         artist.website or None,
827         artist.contact_information or None,
828         password,
829         True
830     ))
831     self.connection.commit()
832     return cursor.rowcount > 0
833
834 def artist_login(self, name, password):
835     if not name or not password:
836         raise InvalidArtistNameException("Artist name and password are required")
837
838     cursor = self.connection.cursor()
839     cursor.execute("""
840         SELECT Artist_ID, Name, Contact_Information, IsActive, Password
841         FROM Artist
842         WHERE Name = %s
843     """, (name,))
844
845     artist_data = cursor.fetchone()
846
847     if not artist_data:
848         raise InvalidArtistNameException("Artist not found")
849
850     artist_id, artist_name, contact_info, is_active, db_password = artist_data
851
852     if not is_active:
853         raise InvalidArtistNameException("Artist account is inactive")
854
855     if password != db_password:
856         raise InvalidPasswordException("Invalid password")
857
858     # Get artworks by this artist
859     cursor.execute("""
860         SELECT Artwork_ID, Title, Creation_Date, Medium
861         FROM Artwork
862         WHERE Artist_ID = %s
863     """, (artist_id,))
864     artist_artworks = cursor.fetchall()
865
866     # Get galleries containing this artist's works
867     cursor.execute("""
868         SELECT DISTINCT g.Gallery_ID, g.Name, g.Location
869         FROM Gallery g
870         JOIN Artwork_Gallery ag ON g.Gallery_ID = ag.Gallery_ID
871         JOIN Artwork a ON ag.Artwork_ID = a.Artwork_ID
872         WHERE a.Artist_ID = %s
873     """, (artist_id,))
874     galleries_with_artworks = cursor.fetchall()
875
876     # Get user favorites for this artist's works
877     cursor.execute("""
878         SELECT a.Artwork_ID, a.Title, COUNT(ufa.User_ID) as favorite_count
879         FROM Artwork a
880         LEFT JOIN User_Favorite_Artwork ufa ON a.Artwork_ID = ufa.Artwork_ID
881         WHERE a.Artist_ID = %s
882         GROUP BY a.Artwork_ID, a.Title
883         ORDER BY favorite_count DESC
884     """, (artist_id,))
885     user_favorites = cursor.fetchall()
886
887     # Get user reviews for this artist's works
888     cursor.execute("""
889         SELECT a.Artwork_ID, a.Title, COUNT(uar.User_ID) as review_count
890         FROM Artwork a
891         LEFT JOIN User_Review_Artwork uar ON a.Artwork_ID = uar.Artwork_ID
892         WHERE a.Artist_ID = %s
893         GROUP BY a.Artwork_ID, a.Title
894         ORDER BY review_count DESC
895     """, (artist_id,))
896     user_reviews = cursor.fetchall()
```

```
883     """ , (artist_id,))
884     artworkFavorites = cursor.fetchall()
885
886     return {
887         "artist_id": artist_id,
888         "artist_name": artist_name,
889         "artworks": [
890             {
891                 "artwork_id": artwork[0],
892                 "title": artwork[1],
893                 "creation_date": artwork[2],
894                 "medium": artwork[3]
895             } for artwork in artist_artworks
896         ] if artist_artworks else "No artworks uploaded yet",
897         "galleries_with_artworks": [
898             {
899                 "gallery_id": gallery[0],
900                 "name": gallery[1],
901                 "location": gallery[2]
902             } for gallery in galleries_with_artworks
903         ] if galleries_with_artworks else "No artworks in galleries yet",
904         "artworkFavorites": [
905             {
906                 "artwork_id": fav[0],
907                 "title": fav[1],
908                 "favorite_count": fav[2]
909             } for fav in artworkFavorites
910         ]
911     }
```

EXCEPTION

```
""" 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)

```
1  ✓ import mysql.connector
2   from util.DBPropertyUtil import PropertyUtil
3
4  ✓ class DBConnection:
5      _connection = None
6
7      @staticmethod
8      def get_connection():
9          if DBConnection._connection is None:
10              props = PropertyUtil.get_property_string()
11              if props:
12                  try:
13                      DBConnection._connection = mysql.connector.connect(
14                          host = props.get('host'),
15                          port = props.get('port'),
16                          database = props.get('database'),
17                          user = props.get('user'),
18                          password = props.get('password')
19                      )
20                  except mysql.connector.Error as e:
21                      print(f"Error connecting to MySql: {e}")
22              else:
23                  print(f"Properties could not be loaded")
24
25      return DBConnection._connection
```

```
1   import os
2
3  ✓ class PropertyUtil:
4
5      @staticmethod
6      def get_property_string(file_path = 'config/db.properties'):
7          properties = {}
8          try:
9              with open(file_path, 'r') as f:
10                  for line in f:
11                      line = line.strip()
12                      if line and not line.startswith('#') and '=' in line:
13                          key,value = line.split('=', 1)
14                          properties[key.strip()] = value.strip()
15          except FileNotFoundError:
16              print(f"Error: Property file '{file_path}' not found.")
17
18          except Exception as e:
19              print(f"Error reading property file: {e}")
20
21          return properties
```

Main

```
1  from dao.IVirtualArtGalleryImp import VirtualArtGalleryImp
2  from entity.artist import Artist
3  from datetime import datetime
4  from entity.artwork import Artwork
5  from entity.gallery import Gallery
6  from entity.user import User
7  from exception.exceptions import (InvalidIDException, InvalidArtistNameException, InvalidDateException,
8  |                                     InvalidWebsiteException, InvalidArtworkException, InvalidMediumTypeException,
9  |                                     DuplicateArtworkException, DuplicateGalleryException, InvalidGalleryNameException,
10 |                                    GalleryNotFoundException, GalleryAssignmentException, InvalidOpeningHoursException,
11 |                                    DuplicateUserException, InvalidUsernameException, InvalidPasswordException,
12 |                                    InvalidEmailException, UserNotFoundException, FavoriteAlreadyExistsException,
13 |                                    ArtworkDoesNotExistException, FavoriteNotFoundException)
14 import sys
15 import os
16
17
18 def clean_input(value):
19     return value.strip() if value.strip() != "" else None
20
21
22 class MainModule:
23     @staticmethod
24     def artist_menu(service):
25         while True:
26             print("\n ----- Artist Menu -----")
27             print("\n 1. Add Artist")
28             print("\n 2. Update Artist")
29             print("\n 3. Delete Artist")
30             print("\n 4. Get Artist By ID")
31             print("\n 5. Reactivate Artist")
32             print("\n 0. Exit")
33             choice = input("Enter your choice:")
34
35         try:
36             if choice == "1":
37                 artist_id = int(input("Enter Artist ID: "))
38                 name = input("Enter Artist Name: ")
39                 biography = clean_input(input("Enter Biography: "))
40                 birth_date_input = clean_input(input("Enter Birth Date (YYYY-MM-DD): "))
41                 birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date() if birth_date_input else None
42                 nationality = clean_input(input("Enter Nationality: "))
43                 website = clean_input(input("Enter Website: "))
44                 contact_information = clean_input(input("Enter Contact Information: "))
45
46                 artist = Artist(artist_id=artist_id, name=name, biography=biography or None, birth_date=birth_date or None, nationality=nationality or None,
47 |                               website=website or None, contact_information=contact_information or None, is_active=True)
48
49                 if service.add_artist(artist):
50                     print("Artist Added Successfully!")
51                 else:
52                     print("Failed to add artist")
53
54             elif choice == "2":
55                 artist_id = int(input("Enter Artist ID to update: "))
56                 existing_artist = service.get_artist_by_id(artist_id)
57                 if not existing_artist:
58                     raise InvalidIDException("Artist with this ID does not exist")
59
60                 print("Please leave the field empty to retain existing value")
61                 name = input("Enter New Artist Name [{existing_artist.name}]: ") or existing_artist.name
62                 biography = clean_input(input("Enter New Biography [{existing_artist.biography}]: ") or existing_artist.biography
63                 birth_date_input = clean_input(input("Enter New Birth Date (YYYY-MM-DD) [{existing_artist.birth_date}]: "))
64
65                 if birth_date_input:
66                     birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date()
67                 else:
68                     birth_date = existing_artist.birth_date
69
70                 nationality = clean_input(input("Enter New Nationality: [{existing_artist.nationality}]: ")) or existing_artist.nationality
71                 website = clean_input(input("Enter Website: [{existing_artist.website}]: ")) or existing_artist.website
72                 contact_information = clean_input(input("Enter Contact Information: [{existing_artist.contact_information}]: ")) or existing_artist.contact_information
73
74                 updated_artist = Artist(artist_id=artist_id, name=name, biography=biography, birth_date=birth_date, nationality=nationality,
75 |                               website=website, contact_information=contact_information, is_active=True)
76                 if service.update_artist(updated_artist):
77                     print("Artist Updated Successfully")
78
79             elif choice == "3":
80                 artist_id = int(input("Enter Artist ID to delete: "))
```

```

81         confirm = input("Are you sure you want to delete Artist {artist_id}? [y/N] ")
82         if confirm.lower() == "y":
83             if service.remove_artist(artist_id):
84                 print("Artist deleted successfully")
85             else:
86                 print("Failed to delete artist")
87         else:
88             print("Attempt to Delete the Artist has been cancelled")
89
90     elif choice == "4":
91         artist_id = int(input("Enter Artist ID to fetch: "))
92         artist = service.get_artist_by_id(artist_id)
93         if artist:
94             print("\n----- Artist Details -----")
95             print(f"ID: {artist.artist_id}")
96             print(f"Name: {artist.name}")
97             print(f"Biography: {artist.biography or 'N/A'}")
98             print(f"Nationality: {artist.nationality or 'N/A'}")
99             print(f"Website: {artist.website or 'N/A'}")
100            print(f"Contact Information: {artist.contact_information or 'N/A'}")
101            print(f"Active Status: { 'Active' if artist.is_active else 'Inactive' }")
102        else:
103            raise InvalidIDException("Artist with this ID does not exist")
104
105    elif choice == "5":
106        artist_id = int(input("Enter Artist ID to reactivate: "))
107        result = service.reactivate_artist(artist_id)
108        if result == 'reactivated':
109            print("Artist account Reactivated Successfully")
110        elif result == 'already_active':
111            print("Artist account Already Active, No Action Needed")
112        elif result == 'not_found':
113            raise InvalidIDException("Artist with this ID does not exist")
114
115    elif choice == "0":
116        print("Exiting... Thank You!")
117        break
118    else:
119        print("Invalid Choice. Try again.")
120
121 except (InvalidIDException, InvalidArtistNameException, InvalidDateException, InvalidURLException) as e:
122     print(f"Validation Error: {e}")
123 except ValueError:
124     print("Invalid input type. Please enter numeric value.")
125 except Exception as e:
126     print(f"Unexpected Error: {e}")
127
128 @staticmethod
129 def artwork_menu(service):
130     while True:
131         print("\n----- Artwork Menu -----")
132         print("\n 1. Add Artwork")
133         print("\n 2. Update Artwork")
134         print("\n 3. Delete Artwork")
135         print("\n 4. Get Artwork By ID")
136         print("\n 5. Search Artwork")
137         print("\n 0. Exit")
138         choice = input("Enter your choice:")
139     try:
140         if choice == "1":
141             artwork_id = clean_input(input("Enter Artwork ID: "))
142             artist_id = clean_input(input("Enter Artist ID: "))
143             title = clean_input(input("Enter Artwork Title: "))
144             description = clean_input(input("Enter Artwork Description: "))
145             creation_date_input = clean_input(input("Enter Creation Date (YYYY-MM-DD): "))
146             creation_date = datetime.strptime(creation_date_input, "%Y-%m-%d").date() if creation_date_input else None
147             medium = clean_input(input("Enter Medium Type: "))
148             image_url=clean_input(input("Enter Image URL: "))
149
150             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)
151             success = service.add_artwork(artwork)
152             if success:
153                 print("Artwork Added Successfully")
154             else:
155                 print("Failed to add Artwork")
156
157         elif choice == "2":
158             artwork_id = int(input("Enter Artwork ID to update: "))
159             existing_artwork = service.get_artwork_by_id(artwork_id)
160             if not existing_artwork:
161                 raise InvalidIDException("Artwork with this ID does not exist")
162
163             print("Please leave the field empty to retain existing value")
164             title = input(f"Enter New Title [{existing_artwork.title}]: ") or existing_artwork.title
165             description = clean_input(
166                 input(f"Enter New Description [{existing_artwork.description}]: ")) or existing_artwork.description
167             creation_date_input = clean_input(
168                 input(f"Enter New Creation Date (YYYY-MM-DD) [{existing_artwork.creation_date}]: "))
169
170             if creation_date_input:
171                 creation_date = datetime.strptime(creation_date_input, "%Y-%m-%d").date()
172             else:
173                 creation_date = existing_artwork.creation_date
174
175             medium = clean_input(input(
176                 f"Enter New Medium: [{existing_artwork.medium}]: ")) or existing_artwork.medium
177             image_url = clean_input(
178                 input(f"Enter Image URL: [{existing_artwork.image_url}]: ")) or existing_artwork.image_url
179             artist_id = clean_input(input(
180                 f"Enter Artist ID: [{existing_artwork.artist_id}]: ")) or existing_artwork.artist_id
181
182             updated_artwork = Artwork(artwork_id=artwork_id, title = title, description = description, creation_date=creation_date,
183                                     medium = medium, image_url = image_url or None, artist_id = artist_id)
184
185             if service.update_artwork(updated_artwork):
186                 print("Artwork Updated Successfully")

```

```

188     elif choice == "3":
189         artwork_id = int(input("Enter Artwork ID to delete: "))
190         confirm = input("Are you sure you want to delete Artwork {artwork_id}? [y/N] ")
191         if confirm.lower() == "y":
192             if service.remove_artwork(artwork_id):
193                 print("Artwork deleted successfully")
194             else:
195                 print("Failed to delete artwork")
196         else:
197             print("Attempt to Delete the Artwork has been cancelled")
198
199     elif choice == "4":
200         artwork_id = int(input("Enter Artwork ID to fetch: "))
201         artwork = service.get_artwork_by_id(artwork_id)
202         if artwork:
203             print("\n----- Artist Details -----")
204             print(f"ID: {artwork.artwork_id}")
205             print(f"Title: {artwork.title}")
206             print(f"Description: {artwork.description or 'N/A'}")
207             print(f"Creation Date: {artwork.creation_date or 'N/A'}")
208             print(f"Medium {artwork.medium or 'N/A'}")
209             print(f"Image URL: {artwork.image_url or 'N/A'}")
210             print(f"Artist ID: {artwork.artist_id}")
211         else:
212             raise InvalidIDException("Artwork with this ID does not exist")
213     elif choice == "5":
214         print("\n Search Artworks By:")
215         print("1. Title")
216         print("2. Artist ID")
217         print("3. Medium")
218         search_choice = input("Enter your choice (1/2/3): ")
219
220         if search_choice == "1":
221             search_by = "title"
222         elif search_choice == "2":
223             search_by = "artist_id"
224         elif search_choice == "3":
225             search_by = "medium"
226         else:
227             print("Invalid choice. Please select valid option")
228             continue
229
230         keyword = input("Enter keyword to search in {search_by.capitalize()}: ")
231         artworks = service.search_artworks(keyword, search_by)
232         if artworks:
233             print("\n----- Matching Artworks -----")
234             for art in artworks:
235                 print(f"\nID: {art.artwork_id}")
236                 print(f"Title: {art.title}")
237                 print(f"Description: {art.description or 'N/A'}")
238                 print(f"Creation Date: {art.creation_date or 'N/A'}")
239                 print(f"Medium: {art.medium or 'N/A'}")
240                 print(f"Image URL: {art.image_url or 'N/A'}")
241                 print(f"Artist ID: {art.artist_id}")
242             else:
243                 print("No artworks found for the given keyword")
244
245         elif choice == "0":
246             print("Exiting... Thank You!")
247             break
248         else:
249             print("Invalid Choice. Try again.")
250     except (InvalidIDException, InvalidArtworkException, InvalidDateException, InvalidWebsiteException, InvalidMediumTypeException, DuplicateArtworkException) as e:
251         print(f"Validation Error: {e}")
252     except ValueError:
253         print("Invalid input type. Please enter numeric value..")
254     except Exception as e:
255         print(f"Unexpected Error: {e}")
256
257     @staticmethod
258     def gallery_menu(service):
259         while True:
260             print("\n ----- Gallery Menu -----")
261             print("\n 1. Add Gallery")
262             print("\n 2. Update Gallery")
263             print("\n 3. Get Gallery By ID")
264             print("\n 4. Delete Gallery")
265
266             print("\n 0. Exit")
267             choice = input("Enter your choice:")
268             try:
269                 if choice == "1":
270                     gallery_id = clean_input(input("Enter Gallery ID: "))
271                     name = clean_input(input("Enter Gallery Name: "))
272                     description = clean_input(input("Enter Gallery Description: "))
273                     location = clean_input(input("Enter Gallery Location: "))
274                     curator = clean_input(input("Enter Gallery Curator: "))
275                     opening_hours = clean_input(input("Enter Gallery Opening Hours: "))
276                     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)
277                     success = service.add_gallery(gallery)
278                     if success:
279                         print("Gallery Added Successfully")
280                     else:
281                         print("Failed to add Gallery")
282
283

```

```

283     elif choice == "2":
284         gallery_id = clean_input(input("Enter Gallery ID to update: "))
285         existing = service.get_gallery_by_id(gallery_id)
286         if not existing:
287             print("Gallery not found.")
288             continue
289
290         name = clean_input(input(f"Enter new name [{existing.name}]: ")) or existing.name
291         description = clean_input(
292             input(f"Enter new description [{existing.description}]: ")) or existing.description
293         location = clean_input(input(f"Enter new location [{existing.location}]: ")) or existing.location
294         curator = clean_input(input(f"Enter new curator ID [{existing.curator}]: ")) or existing.curator
295         opening_hours = clean_input(
296             input(f"Enter new opening hours [{existing.opening_hours}]: ")) or existing.opening_hours
297
298         updated_gallery = Gallery(
299             gallery_id=gallery_id,
300             name=name,
301             description=description,
302             location=location,
303             curator=curator,
304             opening_hours=opening_hours
305         )
306         if service.update_gallery(updated_gallery):
307             print("Gallery updated successfully")
308         else:
309             print("Update failed")
310
311     elif choice == "3":
312         gallery_id = clean_input(input("Enter Gallery ID to fetch: "))
313         gallery = service.get_gallery_by_id(gallery_id)
314         if gallery:
315             print(f"\nGallery ID: {gallery.gallery_id}")
316             print(f"Name: {gallery.name}")
317             print(f"Description: {gallery.description}")
318             print(f"Location: {gallery.location}")
319             print(f"Curator: {gallery.curator}")
320             print(f"Opening Hours: {gallery.opening_hours}")
321         else:
322             print("Gallery not found.")
323
324     elif choice == "4":
325         gallery_id = int(input("Enter Gallery ID to delete: "))
326         confirm = input(f"Are you sure you want to delete Gallery {gallery_id}? [y/N] ")
327         if confirm.lower() == "y":
328             if service.remove_gallery(gallery_id):
329                 print("Gallery deleted successfully")
330             else:
331                 print("Failed to delete gallery")
332         else:
333             print("Gallery deletion cancelled.")
334
335     elif choice == "5":
336         print("\n Search Galleries By:")
337         print("1. Name")
338         print("2. Location")
339         print("3. Curator ID")
340         search_choice = input("Enter your choice (1/2/3): ")
341
342         if search_choice == "1":
343             search_by = "name"
344         elif search_choice == "2":
345             search_by = "location"
346         elif search_choice == "3":
347             search_by = "curator"
348         else:
349             print("Invalid choice. Please select a valid option")
350             return
351
352         keyword = input(f"Enter keyword to search in {search_by.capitalize()}: ")
353         galleries = service.search_gallery(keyword, search_by)
354         if galleries:
355             print("\n----- Matching Galleries -----")
356             for g in galleries:
357                 print(f"\nID: {g.gallery_id}")
358                 print(f"Name: {g.name}")
359                 print(f"Description: {g.description or 'N/A'}")

```

```

170     |         print("Invalid Choice. Try again.")
171     |     except (InvalidIDException, DuplicateGalleryException, InvalidGalleryNameException, GalleryNotFoundException, GalleryAssignmentException, InvalidOpeningHoursException) as e:
172     |         print(f"Validation Error: {e}")
173     |     except ValueError:
174     |         print("Invalid input type. Please enter numeric value..")
175     |     except Exception as e:
176     |         print(f"Unexpected Error: {e}")
177
178     """ User Menu """
179     @statmethod
180     def user_menu(service):
181         while True:
182             print("\n----- User Menu -----")
183             print("1. Add user")
184             print("2. Update user")
185             print("3. Delete user by ID")
186             print("4. Delete user")
187             print("5. Reactivate user")
188             print("6. Exit")
189             choice = input("Enter your choice: ")
190
191             try:
192                 if choice == "1":
193                     user_id = int(input("Enter User ID: "))
194                     user_name = input("Enter Username: ")
195                     password = input("Enter Password: ")
196                     email = clean_input(input("Enter Email: "))
197                     first_name = clean_input(input("Enter First Name: "))
198                     last_name = clean_input(input("Enter Last Name: "))
199                     dob_input = clean_input(input("Enter Date of Birth (YYYY-MM-DD): "))
200                     date_of_birth = datetime.datetime.strptime(dob_input, "%Y-%m-%d").date() if dob_input else None
201                     profile_picture = clean_input(input("Enter Profile Picture URL: "))
202                     user = User(user_id=user_id,username=username, password=password, email=email, 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 )
203                     if service.add_user(user):
204                         print("User Added Successfully!")
205                     else:
206                         print("Failed to add user")
207
208                 elif choice == "2":
209                     user_id = int(input("Enter User ID to update: "))
210                     existing_user = service.get_user_by_id(user_id)
211                     if not existing_user:
212                         raise InvalidIDException("User with this ID does not exist")
213
214                     print("Leave fields empty to keep existing values..")
215
216                     username = input("Enter New Username [{existing_user.username}]: ") or existing_user.username
217                     password = input("Enter New Password [{existing_user.password}]: ") or existing_user.password
218                     email = clean_input(input("Enter New Email [{existing_user.email}]: ")) or existing_user.email
219                     first_name = clean_input(
220                         input("Enter New First Name [{existing_user.first_name}]: ")) or existing_user.first_name
221                     last_name = clean_input(
222                         input("Enter New Last Name [{existing_user.last_name}]: ")) or existing_user.last_name
223                     dob_input = clean_input(
224                         input("Enter New Date of Birth (YYYY-MM-DD) [{existing_user.date_of_birth}]: "))
225                     date_of_birth = datetime.datetime.strptime(dob_input, "%Y-%m-%d").date()
226
227                     if dob_input:
228                         date_of_birth = datetime.datetime.strptime(dob_input, "%Y-%m-%d").date()
229                     else:
230                         date_of_birth = existing_user.date_of_birth
231
232                     profile_picture = clean_input(
233                         input("Enter New Profile Picture URL [{existing_user.profile_picture}]: ")) or existing_user.profile_picture
234
235                     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)
236                     if service.update_user(updated_user):
237                         print("User updated successfully!")
238                     else:
239                         print("Failed to update user.")
240
241                 elif choice == "3":
242                     user_id = int(input("Enter User ID to fetch: "))
243                     user = service.get_user_by_id(user_id)
244                     if user:
245                         print("----- User Details -----")
246                         print(f"ID: {user.user_id}")
247
248                         print(f"Username: {user.username}")
249                         print(f"Email: {user.email or 'N/A'}")
250                         print(f"First Name: {user.first_name or 'N/A'}")
251                         print(f"Last Name: {user.last_name or 'N/A'}")
252                         print(f"Date of Birth: {user.date_of_birth or 'N/A'}")
253                         print(f"Profile Picture: {user.profile_picture or 'N/A'}")
254                         print(f"Active Status: {'Active' if user.user_is_active else 'Inactive'}")
255
256                     else:
257                         raise InvalidIDException("User with this ID does not exist")
258
259                 elif choice == "4":
260                     user_id = int(input("Enter User ID to delete: "))
261                     confirm = input("Are you sure you want to delete User {user_id}? [y/N] ")
262                     if confirm.lower() == "y":
263                         if service.remove_user(user_id):
264                             print("User deleted successfully")
265                         else:
266                             print("Failed to delete user")
267                     else:
268                         print("Attempt to Delete the User has been cancelled")
269
270                 elif choice == "5":
271                     user_id = int(input("Enter User ID to reactivate: "))
272                     password = input("Enter the user's password for verification: ").strip()
273                     result = service.reactivate_user(user_id, password)
274                     if result == 'reactivated':
275                         print("User account Reactivated Successfully")
276                     elif result == 'already_active':
277                         print("User account Already Active, No Action Needed")
278                     elif result == 'invalid_credentials':
279                         print("\n Security Alert: Incorrect password - Reactivation denied")
280                         print("Please verify credentials with the user.")
281                     elif result == 'not_found':
282                         raise InvalidIDException("User with this ID does not exist")
283
284
285                 elif choice == "0":
286                     print("Exiting... Thank You!")
287                     break
288
289                 else:
290                     print("Invalid Choice. Try again.")
291
292             except (InvalidIDException, DuplicateUserException, InvalidUsernameException, InvalidPasswordException, InvalidEmailException) as e:
293                 print(f"Validation Error: {e}")
294             except ValueError:
295                 print("Invalid input type. Please enter numeric value..")
296             except Exception as e:
297                 print(f"Unexpected Error: {e}")

```

```

496     """ User Favorite """
497
498     @staticmethod
499     def user_favorite_menu(service):
500         while True:
501             print("\n ----- User Favorite Menu -----")
502             print("\n 1. Add Artwork to Favorite")
503             print("\n 2. Remove Artwork from Favorite")
504             print("\n 3. Get User favorite Artworks by ID")
505             print("\n 0. Exit")
506             choice = input("Enter your choice: ")
507
508         try:
509             if choice == "1":
510                 user_id = int(input("Enter User ID: "))
511                 artwork_id = int(input("Enter Artwork ID to favorite: "))
512                 if service.add_artwork_to_favorite(user_id, artwork_id):
513                     print("Artwork added to favorites successfully.")
514                 else:
515                     print("Failed to add artwork to favorites.")
516             elif choice == "2":
517                 user_id = int(input("Enter User ID: "))
518                 artwork_id = int(input("Enter Artwork ID to remove from favorites: "))
519                 if service.remove_artwork_from_favorite(user_id, artwork_id):
520                     print("Artwork removed from favorites successfully.")
521                 else:
522                     print("Failed to remove artwork from favorites.")
523             elif choice == "3":
524                 user_id = int(input("Enter User ID to fetch favorite artworks: "))
525                 favorites = service.get_user_favorite_artworks(user_id)
526                 if not favorites:
527                     print("This user has no favorite artworks.")
528                 else:
529                     print("\n----- Favorite Artworks -----")
530                     for art in favorites:
531                         print(f"\nID: {art.artwork_id}")
532                         print(f"Title: {art.title}")
533                         print(f"Description: {art.description}")
534                         print(f"Medium: {art.medium}")
535                         print(f"Creation Date: {art.creation_date}")
536                         print(f"Artist ID: {art.artist_id}")
537             elif choice == "0":
538                 print("Exiting... Thank You!")
539                 break
540
541         except (UserNotFoundException, ArtworkDoesNotExistException, FavoriteAlreadyExistsException,
542                 InvalidIDException, FavoriteNotFoundException) as e:
543             print(f"Validation Error: {e}")
544         except Exception as e:
545             print(f"Unexpected Error: {e}")
546
547     @staticmethod
548     def artwork_gallery_menu(service):
549         while True:
550             print("\n ----- Artwork Gallery Menu -----")
551             print("\n 1. Add Artwork to Gallery")
552             print("\n 2. Remove Artwork from Gallery")
553             print("\n 3. Get Artworks by Gallery ID")
554             print("\n 0. Exit")
555             choice = input("Enter your choice: ")
556
557         try:
558             if choice == "1":
559                 artwork_id = int(input("Enter Artwork ID: "))
560                 gallery_id = int(input("Enter Gallery ID: "))
561                 if service.add_artwork_to_gallery(artwork_id, gallery_id):
562                     print("Artwork successfully added to the gallery")
563                 else:
564                     print("Failed to add artwork")
565             elif choice == "2":
566                 artwork_id = int(input("Enter Artwork ID: "))
567                 gallery_id = int(input("Enter Gallery ID: "))
568                 if service.remove_artwork_from_gallery(artwork_id, gallery_id):
569                     print("Artwork removed from gallery successfully")
570                 else:
571                     print("Failed to remove artwork")
572             elif choice == "3":
573                 gallery_id = int(input("Enter Gallery ID to fetch artworks: "))

```

```

581             print("No artworks found for this gallery")
582         elif choice == "0":
583             print("Exiting... Thank You!")
584             break
585
586     except (GalleryNotFoundException, DuplicateArtworkException, ArtworkDoesNotExistException,
587             InvalidIDException, GalleryNotFoundException) as e:
588         print(f"Validation Error: {e}")
589     except Exception as e:
590         print(f"Unexpected Error: {e}")
591
592     """ Artist Panel Implementation """
593
594     @staticmethod
595     def artist_panel(service):
596         print("\n----- Artist Portal -----")
597         while True:
598             print("\n1. Login")
599             print("2. Sign Up")
600             print("0. Back to Main Menu")
601             choice = input("Enter your choice: ").strip()
602
603             if choice == "1":
604                 MainModule._artist_login_flow(service)
605                 break
606             elif choice == "2":
607                 MainModule._artist_signup_flow(service)
608             elif choice == "0":
609                 return
610             else:
611                 print("Invalid choice. Please try again.")
612
613     @staticmethod
614     def _artist_login_flow(service):
615         print("\n----- Artist Login -----")
616         while True:
617             artist_name = input("Enter your artist name: ").strip()
618             password = input("Enter your password: ").strip()
619
620             if not artist_name or not password:
621                 print("Both artist name and password are required.")
622                 continue
623
624             try:
625                 # Authenticate the artist
626                 artist_info = service.artist_login(artist_name, password)
627                 print(f"\nWelcome, {artist_info['artist_name']}!")
628                 MainModule._artist_dashboard(service, artist_info)
629                 break
630             except Exception as e:
631                 print(f"Login failed: {str(e)}")
632                 retry = input("Would you like to try again? (y/n): ").lower()
633                 if retry != 'y':
634                     return
635
636     @staticmethod
637     def _artist_signup_flow(service):
638         print("\n----- Artist Sign Up -----")
639         try:
640             artist_id = int(input("Enter your artist ID : ").strip())
641             name = input("Enter your artist username: ").strip()
642             password = input("Create a password: ").strip()
643             confirm_password = input("Confirm password: ").strip()
644
645             if password != confirm_password:
646                 print("Passwords do not match!")
647                 return
648
649             if len(password) < 6:
650                 print("Password must be at least 6 characters long!")
651                 return
652
653             biography = clean_input(input("Enter your biography (optional): "))
654
655             birth_date_input = clean_input(input("Enter your birth date (YYYY-MM-DD, optional): "))
656             birth_date = datetime.strptime(birth_date_input, "%Y-%m-%d").date() if birth_date_input else None
657

```

```

658     nationality = clean_input(input("Enter your nationality (optional): "))
659     website = clean_input(input("Enter your website URL (optional): "))
660     contact_info = input("Enter your contact information: ").strip()
661
662     artist = Artist(
663         artist_id=artist_id,
664         name=name,
665         biography=biography,
666         birth_date=birth_date,
667         nationality=nationality,
668         website=website,
669         contact_information=contact_info,
670         is_active=True
671     )
672
673     if service.artist_signup(artist, password):
674         print("\nArtist account created successfully!")
675         print("You can now login with your artist name and password.")
676     else:
677         print("Failed to create artist account.")
678
679     except ValueError as e:
680         print(f"Invalid input: {e}")
681     except Exception as e:
682         print(f"Error: {e}")
683
684     @staticmethod
685     def _artist_dashboard(service, artist_info):
686         while True:
687             print("\n----- Artist Dashboard -----")
688             print(f"\nWelcome, {artist_info['artist_name']} (ID: {artist_info['artist_id']})")
689             print("\n1. View My Artworks")
690             print("2. View My Artworks in Galleries")
691             print("3. View Favorite Counts of My Artworks")
692             print("4. Add New Artwork")
693             print("5. Update Existing Artwork")
694             print("6. Remove Artwork")
695             print("0. Logout")
696
697             choice = input("Enter your choice: ").strip()
698
699             try:
700                 if choice == "1":
701                     artworks = artist_info['artworks']
702                     if isinstance(artworks, str):
703                         print(artworks)
704                     else:
705                         print("\n--- My Artworks ---")
706                         for artwork in artworks:
707                             print(f"\nID: {artwork['artwork_id']}")
708                             print(f"Title: {artwork['title']}")
709                             print(f"Medium: {artwork['medium']}")
710                             print(f"Created: {artwork['creation_date']}")

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

721                 elif choice == "3":
722                     favorites = artist_info['artworkFavorites']
723                     print("\n--- Favorite Counts for My Artworks ---")
724                     for fav in favorites:
725                         print(f"\nArtwork ID: {fav['artwork_id']}")
726                         print(f"Title: {fav['title']}")
727                         print(f"Number of favorites: {fav['favorite_count']}")

728                 elif choice == "4":
729                     print("\n--- Add New Artwork ---")
730                     try:
731                         artwork_id = int(input("Enter artwork ID: ").strip())
732
733
734

```

```

736     description = input("Enter artwork description: ").strip()
737     creation_date = input("Enter creation date (YYYY-MM-DD): ").strip()
738     medium = input("Enter medium type: ").strip()
739     image_url = input("Enter image URL (optional): ").strip() or None
740
741     creation_date = datetime.strptime(creation_date, "%Y-%m-%d").date()
742     artwork = Artwork(
743         artwork_id=artwork_id,
744         title=title,
745         description=description,
746         creation_date=creation_date,
747         medium=medium,
748         image_url=image_url,
749         artist_id=artist_info['artist_id']
750     )
751     if service.add_artwork(artwork):
752         print("Artwork added successfully!")
753
754         artist_info = service.artist_login(artist_info['artist_name'],
755                                         input("Enter your password to refresh: ").strip())
756     else:
757         print("Failed to add artwork.")
758 except ValueError as e:
759     print(f"Invalid input: {e}")
760 except Exception as e:
761     print(f"Error: {e}")
762
763 elif choice == "5":
764     print("\n--- Update Artwork ---")
765     try:
766         artwork_id = int(input("Enter artwork ID to update: ").strip())
767         artwork = service.get_artwork_by_id(artwork_id)
768
769         if not artwork:
770             print("Artwork not found.")
771             continue
772
773         if str(artwork.artist_id) != str(artist_info['artist_id']):
774             print("You can only update your own artworks.")
775             continue
776
777         print("\nCurrent Artwork Details:")
778         print(f"Title: {artwork.title}")
779         print(f"Description: {artwork.description}")
780         print(f"Creation Date: {artwork.creation_date}")
781         print(f"Medium: {artwork.medium}")
782         print(f"Image URL: {artwork.image_url}")
783
784         print("\nEnter new values (leave blank to keep current):")
785         title = input(f"Title [{artwork.title}]: ").strip() or artwork.title
786         description = input(f"Description [{artwork.description}]: ").strip() or artwork.description
787         creation_date_input = input(f"Creation Date [{artwork.creation_date}] (YYYY-MM-DD): ").strip()
788         creation_date = datetime.strptime(creation_date_input,
789                                         "%Y-%m-%d").date() if creation_date_input else artwork.creation_date
790         medium = input(f"Medium [{artwork.medium}]: ").strip() or artwork.medium
791         image_url = input(f"Image URL [{artwork.image_url}]: ").strip() or artwork.image_url
792
793         updated_artwork = Artwork(
794             artwork_id=artwork_id,
795             title=title,
796             description=description,
797             creation_date=creation_date,
798             medium=medium,
799             image_url=image_url,
800             artist_id=artist_info['artist_id']
801         )
802
803         if service.update_artwork(updated_artwork):
804             print("Artwork updated successfully!")
805             # Refresh artist info
806             artist_info = service.artist_login(artist_info['artist_name'],
807                                         input("Enter your password to refresh: ").strip())
808         else:
809             print("Failed to update artwork.")
810     except ValueError as e:
811         print(f"Invalid input: {e}")
812     except Exception as e:
813         print(f"Error: {e}")

```

```

815     elif choice == "6":
816         print("\n--- Remove Artwork ---")
817         try:
818             artwork_id = int(input("Enter artwork ID to remove: ").strip())
819             artwork = service.get_artwork_by_id(artwork_id)
820
821             if not artwork:
822                 print("Artwork not found.")
823                 continue
824
825             if str(artwork.artist_id) != str(artist_info['artist_id']):
826                 print("You can only remove your own artworks.")
827                 continue
828
829             confirm = input(
830                 f"Are you sure you want to permanently delete artwork '{artwork.title}'? (y/n): ").lower()
831             if confirm == 'y':
832                 if service.remove_artwork(artwork_id):
833                     print("Artwork removed successfully!")
834                     # Refresh artist info
835                     artist_info = service.artist_login(artist_info['artist_name'],
836                     input("Enter your password to refresh: ").strip())
837                 else:
838                     print("Failed to remove artwork.")
839             except Exception as e:
840                 print(f"Error: {e}")
841
842             elif choice == "0":
843                 print("Logging out...")
844                 break
845
846             else:
847                 print("Invalid choice. Please try again.")
848
849         except Exception as e:
850             print(f"An error occurred: {e}")
851
852     @staticmethod
853     def user_panel(service):
854         print("\n----- User Portal -----")
855         while True:
856             print("1. Login")
857             print("2. Sign Up")
858             print("0. Back to Main Menu")
859             choice = input("Enter your choice: ").strip()
860
861             if choice == "1":
862                 MainModule._user_login_flow(service)
863                 break
864             elif choice == "2":
865                 MainModule._user_signup_flow(service)
866             elif choice == "0":
867                 return
868             else:
869                 print("Invalid choice. Please try again.")
870
871     @staticmethod
872     def _user_login_flow(service):
873         print("\n----- User Login -----")
874         while True:
875             username = input("Enter your username: ").strip()
876             password = input("Enter your password: ").strip()
877
878             if not username or not password:
879                 print("Both username and password are required.")
880                 continue
881
882             try:
883                 # Authenticate the user
884                 user_info = service.user_login(username, password)
885                 print(f"\nWelcome, {user_info['username']}!")
886                 MainModule._user_dashboard(service, user_info)
887                 break
888             except Exception as e:
889                 print(f"Login failed: {str(e)}")
890                 retry = input("Would you like to try again? (y/n): ").lower()
891                 if retry != 'y':

```

```

92         return
93
94     @staticmethod
95     def _user_signup_flow(service):
96         print("\n----- User Sign Up -----")
97         try:
98             user_id = int(input("Enter your user ID (number): ").strip())
99             username = input("Choose a username: ").strip()
100            password = input("Create a password: ").strip()
101            confirm_password = input("Confirm password: ").strip()
102
103            if password != confirm_password:
104                print("Passwords do not match!")
105                return
106
107            if len(password) < 6:
108                print("Password must be at least 6 characters long!")
109                return
110
111            email = clean_input(input("Enter your email: "))
112            first_name = clean_input(input("Enter your first name: "))
113            last_name = clean_input(input("Enter your last name: "))
114
115            dob_input = clean_input(input("Enter your date of birth (YYYY-MM-DD): "))
116            date_of_birth = datetime.strptime(dob_input, "%Y-%m-%d").date() if dob_input else None
117
118            profile_picture = clean_input(input("Enter profile picture URL: "))
119
120            user = User(
121                user_id=user_id,
122                username=username,
123                password=password,
124                email=email,
125                first_name=first_name,
126                last_name=last_name,
127                date_of_birth=date_of_birth,
128                profile_picture=profile_picture,
129                user_is_active=True
130            )
131
132            if service.add_user(user):
133                print("\nUser account created successfully!")
134                print("You can now login with your username and password.")
135            else:
136                print("Failed to create user account.")
137
138        except ValueError as e:
139            print(f"Invalid input: {e}")
140        except Exception as e:
141            print(f"Error: {e}")
142
143    @staticmethod
144    def _user_dashboard(service, user_info):
145        while True:
146            print("\n----- User Dashboard -----")
147            print(f"Welcome, {user_info['username']} (ID: {user_info['user_id']})")
148            print("\n1. View My Favorites")
149            print("2. Add Artwork to Favorites")
150            print("3. Remove Artwork from Favorites")
151            print("0. Logout")
152
153            choice = input("Enter your choice: ").strip()
154
155            try:
156                if choice == "1":
157                    favorites = user_info['favorites']
158                    if isinstance(favorites, str):
159                        print(favorites)
160                    else:
161                        print("\n--- My Favorite Artworks ---")
162                        for artwork in favorites:
163                            print(f"\nID: {artwork.artwork_id}")
164                            print(f"Title: {artwork.title}")
165                            print(f"Description: {artwork.description}")
166                            print(f"Medium: {artwork.medium}")
167                            print(f"Created: {artwork.creation_date}")

```

```

15         print(f"Description: {artwork.description}")
16         print(f"Medium: {artwork.medium}")
17         print(f"Created: {artwork.creation_date}")
18
19     elif choice == "2":
20         print("\n--- Add Artwork to Favorites ---")
21         artwork_id = int(input("Enter artwork ID to add to favorites: ").strip())
22         if service.add_artwork_to_favorite(user_info['user_id'], artwork_id):
23             print("Artwork added to favorites successfully!")
24             user_info = service.user_login(user_info['username'],
25                                             input("Enter your password to refresh: ").strip())
26         else:
27             print("Failed to add artwork to favorites.")
28
29     elif choice == "3":
30         print("\n--- Remove Artwork from Favorites ---")
31         artwork_id = int(input("Enter artwork ID to remove from favorites: ").strip())
32         if service.remove_artwork_from_favorite(user_info['user_id'], artwork_id):
33             print("Artwork removed from favorites successfully!")
34             # Refresh user info
35             user_info = service.user_login(user_info['username'],
36                                             input("Enter your password to refresh: ").strip())
37         else:
38             print("Failed to remove artwork from favorites.")
39
40     elif choice == "0":
41         print("Logging out...")
42         break
43
44     else:
45         print("Invalid choice. Please try again.")
46
47 except ValueError:
48     print("Invalid input. Please enter a valid number.")
49 except Exception as e:
50     print(f"An error occurred: {e}")
51
52 @staticmethod
53 def main():
54     service = VirtualArtGalleryImp()
55     print("----- Welcome Virtual Art Gallery -----")
56     while True:
57         print("\n Main Menu")
58         print("1. Artist")
59         print("2. Artwork")
60         print("3. Gallery")
61         print("4. User")
62         print("5. User Favorite")
63         print("6. Artwork Gallery")
64         print("7. Artist Panel")
65         print("8. User Panel")
66         print("0. Exit")
67
68         choice = input("Enter your choice: ")
69         if choice == "1":
70             MainModule.artist_menu(service)
71         elif choice == "2":
72             MainModule.artwork_menu(service)
73         elif choice == "3":
74             MainModule.gallery_menu(service)
75         elif choice == "4":
76             MainModule.user_menu(service)
77         elif choice == "5":
78             MainModule.user_favorite_menu(service)
79         elif choice == "6":
80             MainModule.artwork_gallery_menu(service)
81         elif choice == "7":
82             MainModule.artist_panel(service)
83         elif choice == "8":
84             MainModule.user_panel(service)
85         elif choice == '0':
86             print("GoodBye!")
87             break
88         else:
89             print("Invalid Choice. Try again.")
90
91 if __name__ == '__main__':
92     MainModule.main()

```

TESTING

ARTIST MANAGEMENT:

```
        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:

```
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(),

```

```

#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:

```
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

```

```

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
```

```
... EXIT  
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 birds  
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  
===== 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>
```