**Case Study**

**Virtual Art Galllery  
SaiPrabath Chowdary S**

**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.

CREATE DATABASE VirtualArtGallery

USE VirtualArtGallery

• **Entities and Attributes:**

• **Artwork**

ArtworkID (Primary Key)

Title

Description

CreationDate

Medium

ImageURL (or any reference to the digital representation)

CREATE TABLE Artwork (

ArtworkID INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255),

Description TEXT,

CreationDate DATE,

Medium VARCHAR(100),

ImageURL VARCHAR(255),

ArtistID INT

);

• **Artist**

ArtistID (Primary Key)

Name

Biography

BirthDate

Nationality

Website

Contact Information

CREATE TABLE Artist (

ArtistID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255),

Biography TEXT,

BirthDate DATE,

Nationality VARCHAR(100),

Website VARCHAR(255),

ContactInformation VARCHAR(255)

);

• **User**

UserID (Primary Key)

Username

Password

Email

First Name

Last Name

Date of Birth

Profile Picture

FavoriteArtworks (a list of references to ArtworkIDs)

CREATE TABLE User (

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Username VARCHAR(50),

Password VARCHAR(255),

Email VARCHAR(255),

FirstName VARCHAR(100),

LastName VARCHAR(100),

DateOfBirth DATE,

ProfilePicture VARCHAR(255)

);

• **Gallery**

GalleryID (Primary Key)

Name

Description

Location

Curator (Reference to ArtistID)

OpeningHours

CREATE TABLE Gallery (

GalleryID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255),

Description TEXT,

Location VARCHAR(255),

Curator INT,

OpeningHours VARCHAR(255),

FOREIGN KEY (Curator) REFERENCES Artist(ArtistID)

);

• **Relationships:**

• **Artwork - Artist (Many-to-One)**

An artwork is created by one artist.

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

ALTER TABLE Artwork ADD FOREIGN KEY (ArtistID) 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.

CREATE TABLE User\_Favorite\_Artwork (

UserID INT,

ArtworkID INT,

FOREIGN KEY (UserID) REFERENCES User(UserID),

FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID),

PRIMARY KEY (UserID, 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.

CREATE TABLE Artwork\_Gallery (

ArtworkID INT,

GalleryID INT,

FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID),

FOREIGN KEY (GalleryID) REFERENCES Gallery(GalleryID),

PRIMARY KEY (ArtworkID, GalleryID)

);

**Coding :**

**Create the model**/entity classes corresponding to the schema **within package** entity **with**

**variables declared private, constructors(default and parametrized) and getters,setters )**

**Service Provider Interface/Abstract class**

1. **Artist class**

|  |
| --- |
| **class** **Artist**:  **def** **\_\_init\_\_**(self, artistID, name, biography, birthDate, nationality, website, contactInformation):  self.\_\_artistID = artistID  self.\_\_name = name  self.\_\_biography = biography  self.\_\_birthDate = birthDate  self.\_\_nationality = nationality  self.\_\_website = website  self.\_\_contactInformation = contactInformation  # Getters  **def** **getArtistID**(self):  **return** self.\_\_artistID  **def** **getName**(self):  **return** self.\_\_name  **def** **getBiography**(self):  **return** self.\_\_biography  **def** **getBirthDate**(self):  **return** self.\_\_birthDate  **def** **getNationality**(self):  **return** self.\_\_nationality  **def** **getWebsite**(self):  **return** self.\_\_website  **def** **getContactInformation**(self):  **return** self.\_\_contactInformation  # Setters  **def** **setName**(self, name):  self.\_\_name = name  **def** **setBiography**(self, biography):  self.\_\_biography = biography  **def** **setBirthDate**(self, birthDate):  self.\_\_birthDate = birthDate  **def** **setNationality**(self, nationality):  self.\_\_nationality = nationality  **def** **setWebsite**(self, website):  self.\_\_website = website  **def** **setContactInformation**(self, contactInformation):  self.\_\_contactInformation = contactInformation  **def** **\_\_str\_\_**(self):  **return** f"Artist ID: {self.\_\_artistID}**\n**Artist Name: {self.\_\_name}, Contact: {self.\_\_contactInformation}**\n**Birth date: {self.\_\_birthDate}, Nationality: {self.\_\_nationality}, Website: {self.\_\_website}**\n**" |

1. **Artwork class**

|  |
| --- |
| **class** **Artwork**:  **def** **\_\_init\_\_**(self, artworkID, title, description, creationDate, medium, imageURL, artistID):  self.\_\_artworkID = artworkID  self.\_\_title = title  self.\_\_description = description  self.\_\_creationDate = creationDate  self.\_\_medium = medium  self.\_\_imageURL = imageURL  self.\_\_artistID = artistID  # Getters  **def** **getArtworkID**(self):  **return** self.\_\_artworkID  **def** **getTitle**(self):  **return** self.\_\_title  **def** **getDescription**(self):  **return** self.\_\_description  **def** **getCreationDate**(self):  **return** self.\_\_creationDate  **def** **getMedium**(self):  **return** self.\_\_medium  **def** **getImageURL**(self):  **return** self.\_\_imageURL  **def** **getArtistID**(self):  **return** self.\_\_ArtistID  # Setters  **def** **setTitle**(self, title):  self.\_\_title = title  **def** **setDescription**(self, description):  self.\_\_description = description  **def** **setCreationDate**(self, creationDate):  self.\_\_creationDate = creationDate  **def** **setMedium**(self, medium):  self.\_\_medium = medium  **def** **setImageURL**(self, imageURL):  self.\_\_imageURL = imageURL  **def** **setArtistID**(self, ArtistID):  self.\_\_ArtistID = ArtistID  **def** **\_\_str\_\_**(self):  **return** f"Artwork ID: {self.\_\_artworkID}**\n**Title: {self.\_\_title}, Description: {self.\_\_description}**\n**Date: {self.\_\_creationDate}, Medium: {self.\_\_medium}**\n**URL: {self.\_\_imageURL}, Artist ID: {self.\_\_artistID}**\n**" |

1. **Gallery class**

|  |
| --- |
| **class** **Gallery**:  **def** **\_\_init\_\_**(self, galleryID, name, description, location, curator, openingHours):  self.\_\_galleryID = galleryID  self.\_\_name = name  self.\_\_description = description  self.\_\_location = location  self.\_\_curator = curator  self.\_\_openingHours = openingHours  # Getters  **def** **getGalleryID**(self):  **return** self.\_\_galleryID  **def** **getName**(self):  **return** self.\_\_name  **def** **getDescription**(self):  **return** self.\_\_description  **def** **getLocation**(self):  **return** self.\_\_location  **def** **getCurator**(self):  **return** self.\_\_curator  **def** **getOpeningHours**(self):  **return** self.\_\_openingHours  # Setters  **def** **setName**(self, name):  self.\_\_name = name  **def** **setDescription**(self, description):  self.\_\_description = description  **def** **setLocation**(self, location):  self.\_\_location = location  **def** **setCurator**(self, curator):  self.\_\_curator = curator  **def** **setOpeningHours**(self, openingHours):  self.\_\_openingHours = openingHours  **def** **\_\_str\_\_**(self):  **return** f"Gallery ID: {self.\_\_galleryID}**\n**Name: {self.\_\_name}, Description: {self.\_\_description}**\n**Location: {self.\_\_location}, Curator: {self.\_\_curator}, Opening Hours: {self.\_\_openingHours}**\n**" |

1. **User class**

|  |
| --- |
| **class** **User**:  **def** **\_\_init\_\_**(self, userID, username, password, email, firstName, lastName, birthDate, profilePicture):  self.\_\_userID = userID  self.\_\_username = username  self.\_\_password = password  self.\_\_email = email  self.\_\_firstName = firstName  self.\_\_lastName = lastName  self.\_\_birthDate = birthDate  self.\_\_profilePicture = profilePicture  # Getters  **def** **getUserID**(self):  **return** self.\_\_userID  **def** **getUsername**(self):  **return** self.\_\_username  **def** **getPassword**(self):  **return** self.\_\_password  **def** **getEmail**(self):  **return** self.\_\_email  **def** **getFirstName**(self):  **return** self.\_\_firstName  **def** **getLastName**(self):  **return** self.\_\_lastName  **def** **getBirthDate**(self):  **return** self.\_\_birthDate  **def** **getProfilePicture**(self):  **return** self.\_\_profilePicture  # Setters  **def** **setUsername**(self, username):  self.\_\_username = username  **def** **setPassword**(self, password):  self.\_\_password = password  **def** **setEmail**(self, email):  self.\_\_email = email  **def** **setFirstName**(self, firstName):  self.\_\_firstName = firstName  **def** **setLastName**(self, lastName):  self.\_\_lastName = lastName  **def** **setBirthDate**(self, birthDate):  self.\_\_birthDate = birthDate  **def** **setProfilePicture**(self, profilePicture):  self.\_\_profilePicture = profilePicture  **def** **\_\_str\_\_**(self):  **return** f"User ID: {self.\_\_userID}**\n**UserName: {self.\_\_username}, email id: {self.\_\_email}**\n**First Name: {self.\_\_firstName}, Last Name: {self.\_\_lastName}**\n**Birth Date: {self.\_\_birthDate}, Profile Pic: {self.\_\_profilePicture}**\n**" |

Keep the interfaces and implementation classes in package dao

Create **IVirtualArtGallery** Interface/abstract class with the following methods

**// Artwork Management**

**addArtwork();**

parameters- Artwork object

return type Boolean

**updateArtwork**();

parameters- Artwork object

return type Boolean

**removeArtwork()**

parameters-artworkID

return type Boolean

**getArtworkById**();

parameters-artworkID

return type Artwork

searchArtworks()

**searchArtworks();**

parameters- keyword

return type list of Artwork Object

**// User Favorites**

**addArtworkToFavorite**();

parameters- userId, artworkId

return type boolean

**removeArtworkFromFavorite**()

parameters- userId, artworkId

return type boolean

**getUserFavoriteArtworks()**

parameters- userId

return type boolean

}

|  |
| --- |
| **from** **abc** **import** ABC, abstractmethod  **class** **IVirtualArtGallery**(ABC):  **@abstractmethod**  **def** **createUser**(self,user):  **pass**  **@abstractmethod**  **def** **getAllArtworks**(self):  **pass**  **@abstractmethod**  **def** **addArtwork**(self, artwork):  **pass**  **@abstractmethod**  **def** **updateArtwork**(self, artwork):  **pass**  **@abstractmethod**  **def** **removeArtwork**(self, artworkID):  **pass**  **@abstractmethod**  **def** **getArtworkById**(self, artworkID):  **pass**  **@abstractmethod**  **def** **searchArtworks**(self, keyword):  **pass**  **@abstractmethod**  **def** **addArtworkToFavorite**(self, userID, artworkID):  **pass**  **@abstractmethod**  **def** **removeArtworkFromFavorite**(self, userID, artworkID):  **pass**  **@abstractmethod**  **def** **getUserFavoriteArtworks**(self, userID):  **pass**  **@abstractmethod**  **def** **displayGalleries**(self):  **pass**  **@abstractmethod**  **def** **addArtist**(self,artist):  **pass**  **@abstractmethod**  **def** **addGallery**(self, gallery):  **pass**  **@abstractmethod**  **def** **updateGallery**(self, gallery):  **pass**  **@abstractmethod**  **def** **removeGallery**(self, gallery\_id):  **pass**  **@abstractmethod**  **def** **searchGalleries**(self, keyword):  **pass** |

**7: Connect your application to the SQL database:**

* Write code to establish a connection to your SQL database.
* Create a utility class **DBConnection** in a package **util** with a static variable **connection** of Type
* **Connection** and a static method **getConnection()** which returns connection. Connection properties supplied in the connection string should be read from a property file.
* Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

|  |
| --- |
| **import** **mysql.connectorfrom** **mysql.connector** **import** Error  **class** **DBConnection**:  connection = **None**  **@staticmethod**  **def** **getConnection**():  **try**:  **if** DBConnection.connection **is** **None** **or** **not** DBConnection.connection.is\_connected():  connection\_string = PropertyUtil.getPropertyString("DBdata.txt")  DBConnection.connection = mysql.connector.connect(\*\*connection\_string)  **return** DBConnection.connection  **except** Error **as** e:  print("Error:", e)  **return** **None**  **class** **PropertyUtil**:  **@staticmethod**  **def** **getPropertyString**(property\_file):  properties = {}  **with** open(property\_file,'r') **as** file:  **for** line **in** file:  key, value = line.strip().split('=')  properties[key] = value  **return** properties |

|  |
| --- |
| **DBdata.txt**  host=localhost  user=root  password=chowdary@22  port=**3306**  database=VirtualArtGallery |

**8: Service implementation**

1. Create a Service class **IVirtualArtGalleryImpl** in **dao** with a static variable named connection of type **Connection** which can be assigned in the constructor by invoking the **getConnection()** method in **DBConnection** class

1. Provide implementation for all the methods in the interface.

|  |
| --- |
| **from** **VirtualArtGallery.dao.IVirtualArtGallery** **import** IVirtualArtGallery  **from** **VirtualArtGallery.util.dbutil** **import** DBConnection  **from** **VirtualArtGallery.exception.myexceptions** **import** ArtWorkNotFoundException  **from** **VirtualArtGallery.entity.artwork** **import** Artwork  **from** **VirtualArtGallery.entity.gallery** **import** Gallery  **import** **mysql.connector**  **class** **IVirtualArtGalleryImpl**(IVirtualArtGallery):  **def** **\_\_init\_\_**(self):  self.connection = DBConnection.getConnection()  **def** **createUser**(self,user):  **try**:  cursor = self.connection.cursor()  query = "INSERT INTO User (username, password, email, FirstName, LastName, DateOfBirth, ProfilePicture) VALUES (%s, %s, %s, %s, %s, %s, %s)"  values = (user.getUsername(), user.getPassword(), user.getEmail(), user.getFirstName(), user.getLastName(), user.getBirthDate(), user.getProfilePicture())  cursor.execute(query, values)  self.connection.commit()  query = "SELECT max(userID) FROM User"  cursor.execute(query)  uid = cursor.fetchone()  self.connection.commit()  cursor.close()  **return** [**True**, uid]  **except** mysql.connector.Error **as** err:  print("Error adding user:", err)  **return** **False**  **def** **getAllArtworks**(self):  **try**:  cursor = self.connection.cursor()  query = "SELECT \* FROM Artwork"  cursor.execute(query)  artwork\_data = cursor.fetchall()  cursor.close()  artworks = [Artwork(\*data) **for** data **in** artwork\_data]  **return** artworks  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **None**  **def** **addArtwork**(self, artwork):  **try**:  cursor = self.connection.cursor()  query = "INSERT INTO Artwork (title, description, creationDate, medium, imageURL, ArtistID) VALUES (%s, %s, %s, %s, %s, %s)"  values = (artwork.getTitle(), artwork.getDescription(), artwork.getCreationDate(), artwork.getMedium(), artwork.getImageURL(), artwork.getArtistID())  cursor.execute(query, values)  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **updateArtwork**(self, artwork):  **try**:  cursor = self.connection.cursor()  query = "UPDATE Artwork SET title=%s, description=%s, creationDate=%s, medium=%s, imageURL=%s, ArtistID=%s WHERE artworkID=%s"  values = (artwork.getTitle(), artwork.getDescription(), artwork.getCreationDate(), artwork.getMedium(), artwork.getImageURL(), artwork.getArtistID(), artwork.getArtworkID())  cursor.execute(query, values)  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **removeArtwork**(self, artworkID):  **try**:  cursor = self.connection.cursor()  query = "DELETE FROM Artwork WHERE artworkID=%s"  cursor.execute(query, (artworkID,))  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **getArtworkById**(self, artworkID):  **try**:  cursor = self.connection.cursor()  query = "SELECT \* FROM Artwork WHERE artworkID=%s"  cursor.execute(query, (artworkID,))  result = cursor.fetchone()  **if** result:  artwork = Artwork(\*result)  **return** artwork  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **None**  **def** **searchArtworks**(self, keyword):  **try**:  cursor = self.connection.cursor()  query = "SELECT \* FROM Artwork WHERE title LIKE %s OR description LIKE %s"  cursor.execute(query, (f"%{keyword}%", f"%{keyword}%"))  artwork\_data = cursor.fetchall()  cursor.close()  artworks = [Artwork(\*data) **for** data **in** artwork\_data]  **return** artworks  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** []  **def** **addArtworkToFavorite**(self, userId, artworkId):  **try**:  cursor = self.connection.cursor()  query = "INSERT INTO User\_Favorite\_Artwork (userID, artworkID) VALUES (%s, %s)"  cursor.execute(query, (userId, artworkId))  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **removeArtworkFromFavorite**(self, userId, artworkId):  **try**:  cursor = self.connection.cursor()  query = "DELETE FROM User\_Favorite\_Artwork WHERE userID=%s AND artworkID=%s"  cursor.execute(query, (userId, artworkId))  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **getUserFavoriteArtworks**(self, userId):  **try**:  cursor = self.connection.cursor()  query = "SELECT artworkID, title, description, creationDate, medium, imageURL, artistID FROM User\_Favorite\_Artwork uf join artwork aw on uf.artworkID=aw.artworkID WHERE userID=%s"  cursor.execute(query, (userId,))  artwork\_data = cursor.fetchall()  cursor.close()  favoriteArtworks = [Artwork(\*data) **for** data **in** artwork\_data]  **return** favoriteArtworks  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** []    **def** **displayGalleries**(self):  **try**:  cursor = self.connection.cursor()  query = "SELECT \* FROM gallery"  cursor.execute(query)  gallery\_data = cursor.fetchall()  cursor.close()  galleries = [Gallery(\*data) **for** data **in** gallery\_data]  **return** galleries  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** []  **def** **addArtist**(self,artist):  **try**:  cursor = self.connection.cursor()  query = "INSERT INTO Artist (name, biography, birthDate, nationality, website, ContactInformation) VALUES (%s, %s, %s, %s, %s, %s)"  values = (artist.getName(), artist.getBiography(), artist.getBirthDate(), artist.getNationality(),  artist.getWebsite(), artist.getContactInformation())  cursor.execute(query, values)  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **addGallery**(self, gallery):  **try**:  cursor = self.connection.cursor()  query = "INSERT INTO Gallery (name, description, location, openingHours, curator) VALUES (%s, %s, %s, %s, %s)"  values = (gallery.getName(), gallery.getDescription(), gallery.getLocation(), gallery.getOpeningHours(),  gallery.getCurator())  cursor.execute(query, values)  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **updateGallery**(self, gallery):  **try**:  cursor = self.connection.cursor()  query = "UPDATE Gallery SET name = %s, description = %s, location = %s, openingHours = %s, curator = %s WHERE galleryID = %s"  values = (gallery.getName(), gallery.getDescription(), gallery.getLocation(), gallery.getOpeningHours(),  gallery.getCurator(), gallery.getGalleryID())  cursor.execute(query, values)  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **removeGallery**(self, gallery\_id):  **try**:  cursor = self.connection.cursor()  query = "DELETE FROM Gallery WHERE galleryID = %s"  cursor.execute(query, (gallery\_id,))  self.connection.commit()  cursor.close()  **return** **True**  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** **False**  **def** **searchGalleries**(self, keyword):  **try**:  cursor = self.connection.cursor()  query = "SELECT \* FROM Gallery WHERE name LIKE %s OR description LIKE %s"  cursor.execute(query, ('%' + keyword + '%', '%' + keyword + '%'))  gallery\_data = cursor.fetchall()  cursor.close()  galleries=[Gallery(\*data) **for** data **in** gallery\_data]  **return** galleries  **except** mysql.connector.Error **as** err:  print("Error:", err)  **return** [] |

**9: Exception Handling**

Create the exceptions in package **myexceptions** Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

1. **ArtWorkNotFoundException** :throw this exception when user enters an invalid id which doesn’t exist in db

1. **UserNotFoundException** :throw this exception when user enters an invalid id which doesn’t exist in db

|  |
| --- |
| **class** **ArtWorkNotFoundException**(**Exception**):  **def** **\_\_init\_\_**(self, message="Artwork not found."):  self.message = message  super().\_\_init\_\_(self.message)  **class** **UserNotFoundException**(**Exception**):  **def** **\_\_init\_\_**(self, message="User not found."):  self.message = message  super().\_\_init\_\_(self.message)  **class** **ArtistNotFoundException**(**Exception**):  **def** **\_\_init\_\_**(self, message="Artist not found."):  self.message = message  super().\_\_init\_\_(self.message)  **class** **GalleryNotFoundException**(**Exception**):  **def** **\_\_init\_\_**(self, message="Gallery not found."):  self.message = message  super().\_\_init\_\_(self.message) |

**9. Main Method**

Create class named MainModule with main method in main package.

Trigger all the methods in service implementation class.

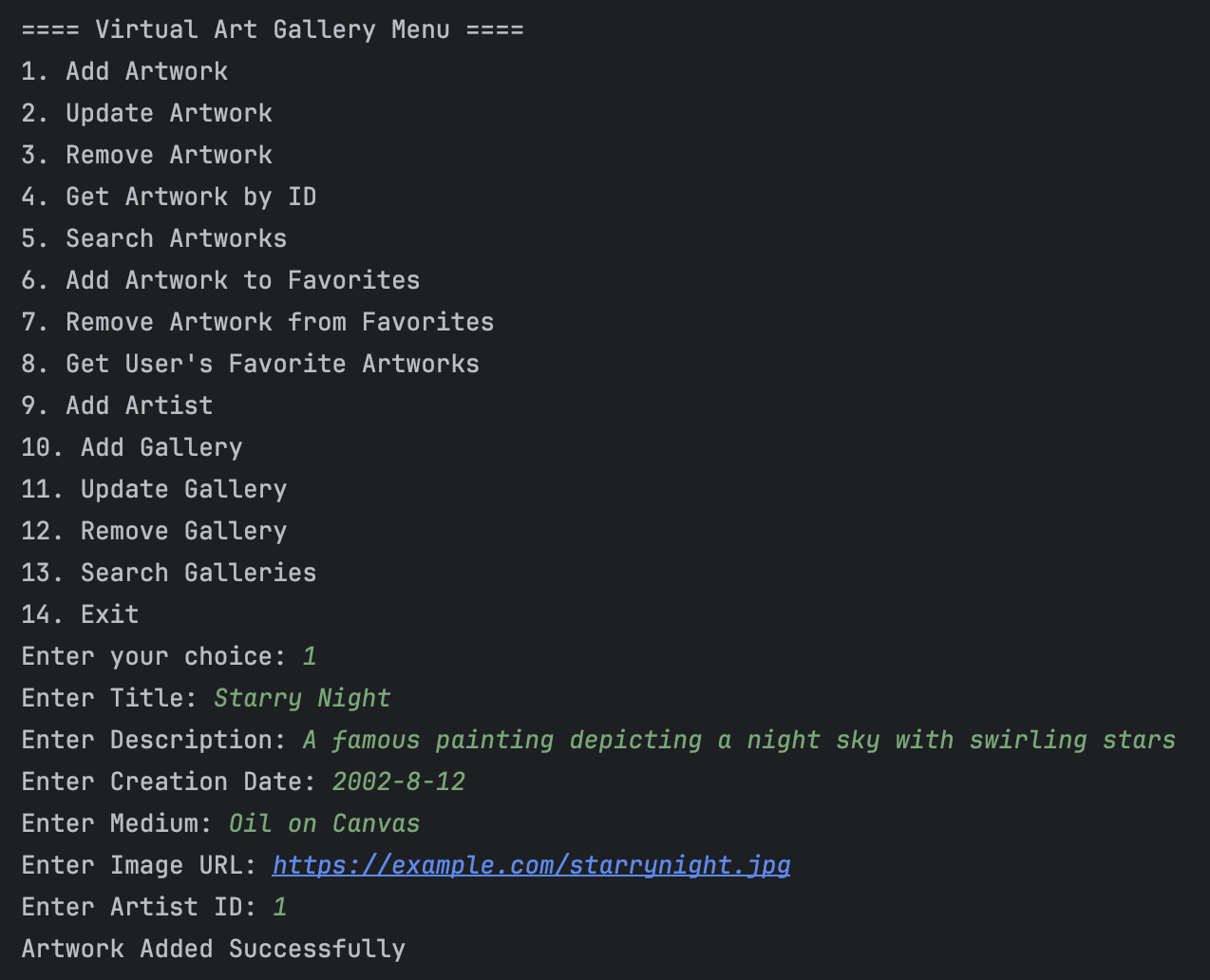
|  |
| --- |
| **from** **VirtualArtGallery.dao.IVirtualArtGalleryimpl** **import** IVirtualArtGalleryImpl  **from** **VirtualArtGallery.exception.myexceptions** **import** ArtWorkNotFoundException, UserNotFoundException  **from** **VirtualArtGallery.exception.myexceptions** **import** ArtistNotFoundException, GalleryNotFoundException  **from** **VirtualArtGallery.entity.artwork** **import** Artwork  **from** **VirtualArtGallery.entity.user** **import** User  **from** **VirtualArtGallery.entity.gallery** **import** Gallery  **from** **VirtualArtGallery.entity.artist** **import** Artist  **class** **MainModule**:  **def** **\_\_init\_\_**(self):  self.virtual\_gallery = IVirtualArtGalleryImpl()  # artwork management  **def** **create\_user**(self):  **try**:  username = input("Enter username: ")  email = input("Enter email: ")  password = input("Enter password: ")  firstName = input("Enter first Name: ")  lastName = input("Enter last Name: ")  birthDate = input("Enter : birthDate")  profilePicture = input("select profilePicture(enter url): ")  user = User(**None**, username, password, email, firstName, lastName, birthDate, profilePicture)  result = self.virtual\_gallery.createUser(user)  **if** result[**0**]:  print("User created successfully!")  **return** result[**1**]  **except** **Exception** **as** e:  print(f"Error creating user: {e}")  **return** **None**  **def** **display\_artworks**(self):  **try**:  artworks = self.virtual\_gallery.getAllArtworks()  **if** artworks:  print("Artworks:")  **for** artwork **in** artworks:  print(artwork)  **return** **True**  **else**:  print("No artworks found.")  **return** **False**  **except** **Exception** **as** e:  print(f"Error displaying Artworks: {e}")  **def** **add\_artwork**(self):  **try**:  title = input("Enter Title: ")  description = input("Enter Description: ")  creation\_date = input("Enter Creation Date: ")  medium = input("Enter Medium: ")  image\_url = input("Enter Image URL: ")  artist\_id = int(input("Enter Artist ID: "))  artwork = Artwork(**None**, title, description, creation\_date, medium, image\_url, artist\_id)  **if** self.virtual\_gallery.addArtwork(artwork):  print("Artwork Added Successfully")  **except** **Exception** **as** e:  print(f"Error adding artwork: {e}")  **def** **update\_artwork**(self):  arts = self.display\_artworks()  **if** **not** arts:  **return**  **try**:  artwork\_id = int(input("Enter Artwork ID to update: "))  title = input("Enter new Title: ")  description = input("Enter new Description: ")  creation\_date = input("Enter new Creation Date: ")  medium = input("Enter new Medium: ")  image\_url = input("Enter new Image URL: ")  artist\_id = int(input("Enter Artist ID: "))  artwork = Artwork(artwork\_id, title, description, creation\_date, medium, image\_url, artist\_id)  **if** self.virtual\_gallery.updateArtwork(artwork):  print("Artwork updated successfully!")  **except** ArtWorkNotFoundException **as** e:  print(f"Artwork not found: {e}")  **except** **Exception** **as** e:  print(f"Error updating artwork: {e}")  **def** **remove\_artwork**(self):  arts = self.display\_artworks()  **if** **not** arts:  **return**  **try**:  artwork\_id = int(input("**\n**Enter Artwork ID to remove: "))  **if** self.virtual\_gallery.removeArtwork(artwork\_id):  print("Artwork removed successfully!")  **except** ArtWorkNotFoundException **as** e:  print(f"Artwork not found: {e}")  **except** **Exception** **as** e:  print(f"Error removing artwork: {e}")  **def** **get\_artwork\_by\_id**(self):  **try**:  artwork\_id = input("Enter Artwork ID to retrieve: ")  artwork = self.virtual\_gallery.getArtworkById(artwork\_id)  **if** artwork:  print("Artwork details:")  print(artwork)  **else**:  print(f"Artwork {artwork\_id} not found")  **except** ArtWorkNotFoundException **as** e:  print(f"Artwork not found: {e}")  **except** **Exception** **as** e:  print(f"Error retrieving artwork: {e}")  **def** **search\_artworks**(self):  keyword = input("Enter keyword to search artworks: ")  **try**:  artworks = self.virtual\_gallery.searchArtworks(keyword)  **if** artworks:  print("Search results:")  **for** artwork **in** artworks:  print(artwork)  **else**:  print("No Artworks Found")  **except** **Exception** **as** e:  print(f"Error searching artworks: {e}")  **def** **add\_artwork\_to\_favorite**(self):  **try**:  existing\_user = input("Are you an existing user? (y/n): ").lower()  **if** existing\_user == 'y':  user\_id = input("Enter your user ID: ")  **else**:  user\_id = self.create\_user()  print(f"Your user id = {user\_id}")  **if** **not** user\_id:  print("Failed to add artwork to favorites. User ID not provided.")  **return**  arts = self.display\_artworks()  **if** **not** arts:  print("no artworks found to add")  **return**  artwork\_id = input("Enter Artwork ID to add to favorites: ")  **if** self.virtual\_gallery.addArtworkToFavorite(user\_id, artwork\_id):  print("Artwork added to favorites successfully!")  **except** (ArtWorkNotFoundException, UserNotFoundException) **as** e:  print(f"Error adding artwork to favorites: {e}")  **except** **Exception** **as** e:  print(f"Error adding artwork to favorites: {e}")  **def** **remove\_artwork\_from\_favorite**(self):  **try**:  user\_id = self.get\_user\_favorite\_artworks()  artwork\_id = input("Enter Artwork ID to remove from favorites: ")  **if** self.virtual\_gallery.removeArtworkFromFavorite(user\_id, artwork\_id):  print("Artwork removed from favorites successfully!")  **except** (ArtWorkNotFoundException, UserNotFoundException) **as** e:  print(f"Error removing artwork from favorites: {e}")  **except** **Exception** **as** e:  print(f"Error removing artwork from favorites: {e}")  **def** **get\_user\_favorite\_artworks**(self):  **try**:  user\_id = input("Enter User ID to retrieve favorite artworks: ")  favorite\_artworks = self.virtual\_gallery.getUserFavoriteArtworks(user\_id)  **if** favorite\_artworks:  print("User's favorite artworks:")  **for** artwork **in** favorite\_artworks:  print(artwork)  **else**:  print("No favourite artworks found")  **return** user\_id  **except** UserNotFoundException **as** e:  print(f"User not found: {e}")  **except** **Exception** **as** e:  print(f"Error retrieving user's favorite artworks: {e}")    # Gallery management  **def** **display\_galleries**(self):  **try**:  galleries = self.virtual\_gallery.displayGalleries()  **if** galleries:  print("Available Galleries:")  **for** gallery **in** galleries:  print(gallery)  **return** **True**  **else**:  print("No galleries found.")  **return** **False**  **except** **Exception** **as** e:  print(f"Error displaying galleries: {e}")  **def** **create\_artist**(self):  **try**:  name = input("Enter artist name: ")  biography = input("Enter artist biography: ")  birth\_date = input("Enter artist birth date: ")  nationality = input("Enter artist nationality: ")  website = input("Enter artist website: ")  contact\_info = input("Enter artist contact information: ")  artist = Artist(**None**, name, biography, birth\_date, nationality, website, contact\_info)  **if** self.virtual\_gallery.addArtist(artist):  print("Artist added successfully!")  **except** **Exception** **as** e:  print(f"Error adding artist: {e}")  **def** **create\_gallery**(self):  **try**:  name = input("Enter gallery name: ")  description = input("Enter gallery description: ")  location = input("Enter gallery location: ")  opening\_hours = input("Enter opening hours: ")  curator = input("Enter curator name : ")  gallery = Gallery(**None**, name, description, location, curator, opening\_hours)  **if** self.virtual\_gallery.addGallery(gallery):  print("Gallery created successfully!")  **except** **Exception** **as** e:  print(f"Error creating gallery: {e}")  **def** **update\_gallery**(self):  gall = self.display\_galleries()  **if** **not** gall:  **return**  **try**:  gallery\_id = int(input("Enter the ID of the gallery you want to update: "))  name = input("Enter new name : ")  description = input("Enter new description: ")  location = input("Enter new location : ")  opening\_hours = input("Enter new opening hours : ")  curator = input("Enter new curator name : ")  gallery = Gallery(gallery\_id, name, description, location, curator, opening\_hours)  **if** self.virtual\_gallery.updateGallery(gallery):  print("Gallery updated successfully!")  **else**:  print("Failed to update gallery.")  **except** GalleryNotFoundException **as** e:  print(f"Gallery not found: {e}")  **except** **Exception** **as** e:  print(f"Error updating gallery: {e}")  **def** **remove\_gallery**(self):  gall = self.display\_galleries()  **if** **not** gall:  **return**  **try**:  gallery\_id = int(input("Enter the ID of the gallery you want to remove: "))  self.virtual\_gallery.removeGallery(gallery\_id)  print("Gallery removed successfully!")  **except** GalleryNotFoundException **as** e:  print(f"Gallery not found: {e}")  **except** **Exception** **as** e:  print(f"Error removing gallery: {e}")  **def** **search\_galleries**(self):  keyword = input("Enter keyword to search galleries: ")  **try**:  galleries = self.virtual\_gallery.searchGalleries(keyword)  **if** galleries:  print("Search Results:")  **for** gallery **in** galleries:  print(gallery)  **else**:  print("No galleries found matching the keyword.")  **except** **Exception** **as** e:  print(f"Error searching galleries: {e}")  **def** **main**(self):  **while** **True**:  print("**\n**==== Virtual Art Gallery Menu ====")  print("1. Add Artwork")  print("2. Update Artwork")  print("3. Remove Artwork")  print("4. Get Artwork by ID")  print("5. Search Artworks")  print("6. Add Artwork to Favorites")  print("7. Remove Artwork from Favorites")  print("8. Get User's Favorite Artworks")  print("9. Add Artist")  print("10. Add Gallery")  print("11. Update Gallery")  print("12. Remove Gallery")  print("13. Search Galleries")  print("14. Exit")  choice = input("Enter your choice: ")  **if** choice == "1":  self.add\_artwork()  **elif** choice == "2":  self.update\_artwork()  **elif** choice == "3":  self.remove\_artwork()  **elif** choice == "4":  self.get\_artwork\_by\_id()  **elif** choice == "5":  self.search\_artworks()  **elif** choice == "6":  self.add\_artwork\_to\_favorite()  **elif** choice == "7":  self.remove\_artwork\_from\_favorite()  **elif** choice == "8":  self.get\_user\_favorite\_artworks()  **elif** choice == "9":  self.create\_artist()  **elif** choice == "10":  self.create\_gallery()  **elif** choice == "11":  self.update\_gallery()  **elif** choice == "12":  self.remove\_gallery()  **elif** choice == "13":  self.search\_galleries()  **elif** choice == "14":  print("Exiting...")  **break**  **else**:  print("Invalid choice. Please try again.") |

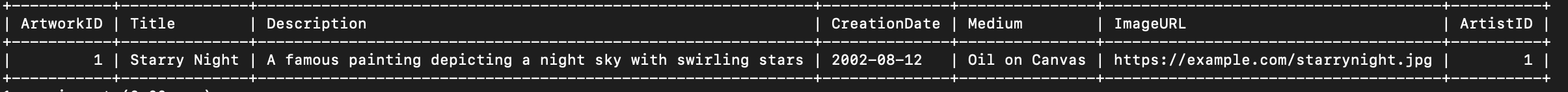
**10. Unit Testing**

Creating Unit test cases for a Virtual Art Gallery system is essential to ensure that the system

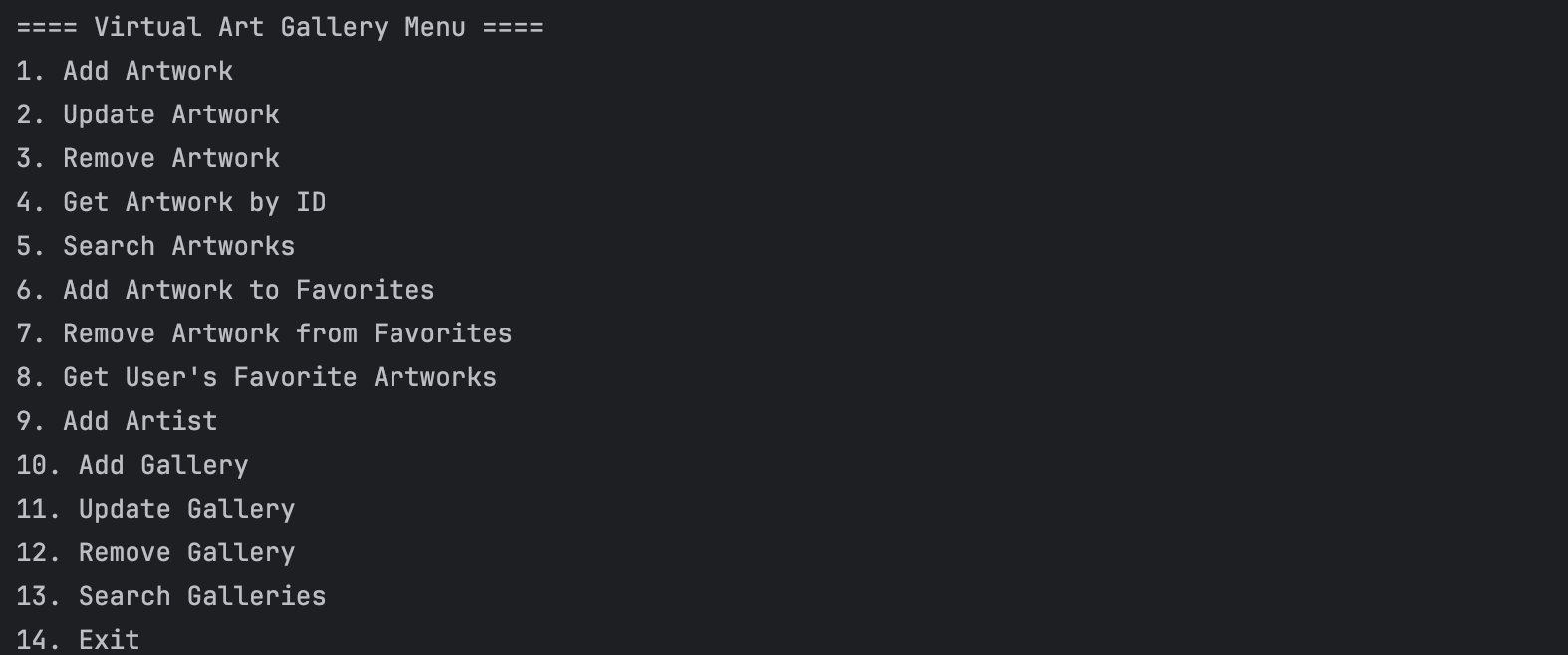
functions correctly. Below are sample test case questions that can serve as a starting point for your JUnit test suite:

1. **Artwork Management:**
2. Test the ability to upload a new artwork to the gallery.



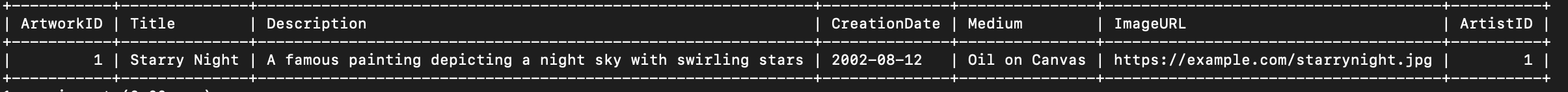


1. Verify that updating artwork details works correctly.

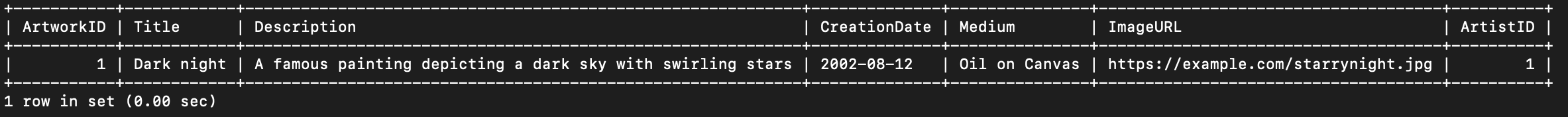




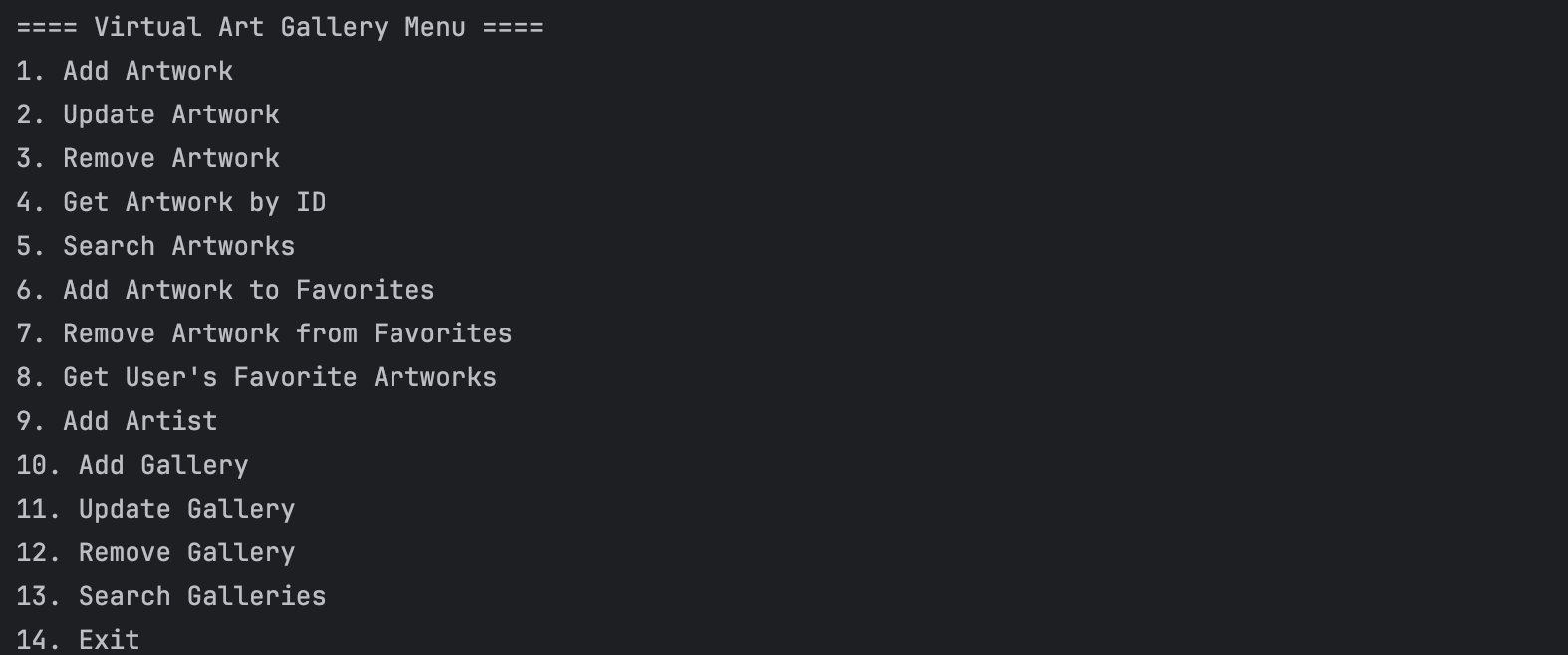
old :



updated:



1. Test removing an artwork from the gallery.

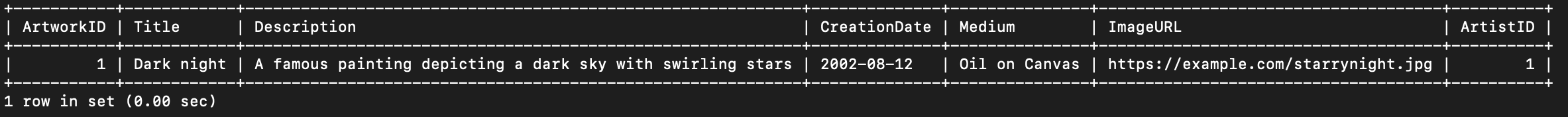




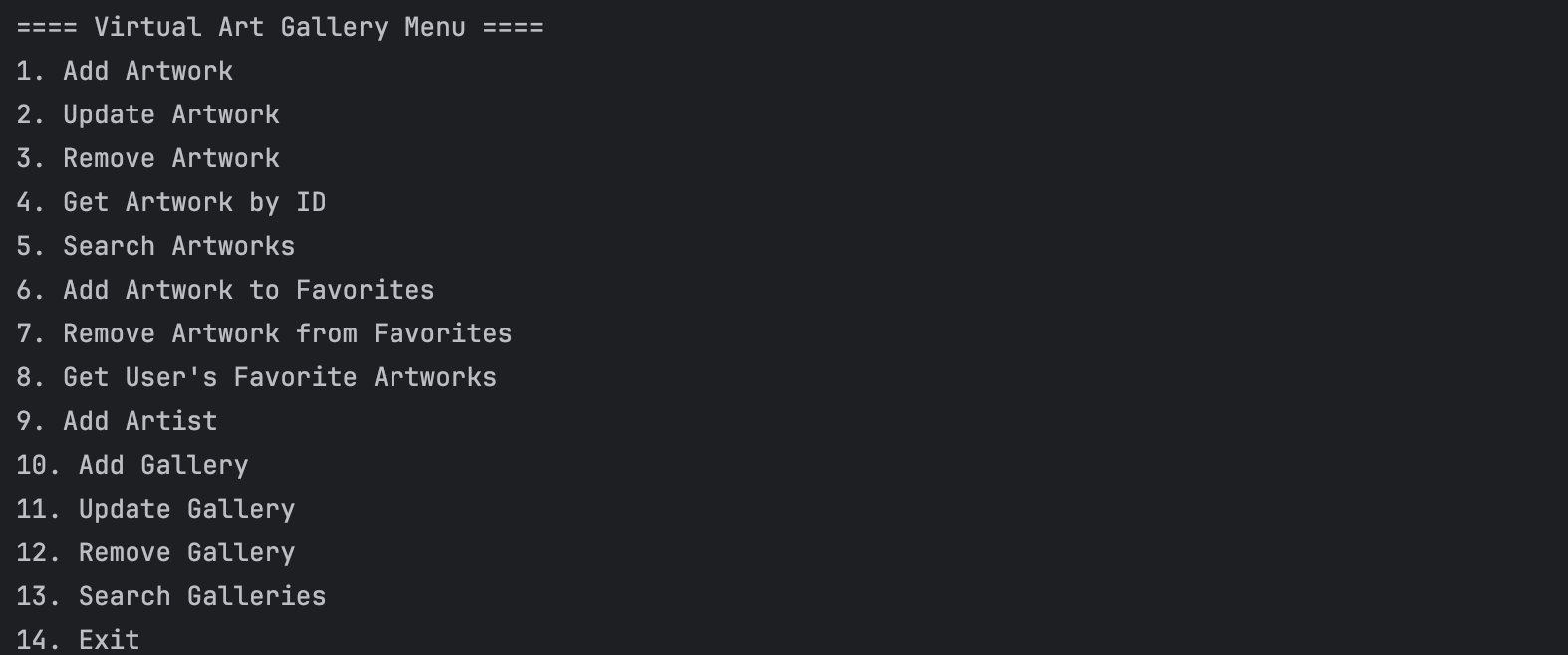
old:

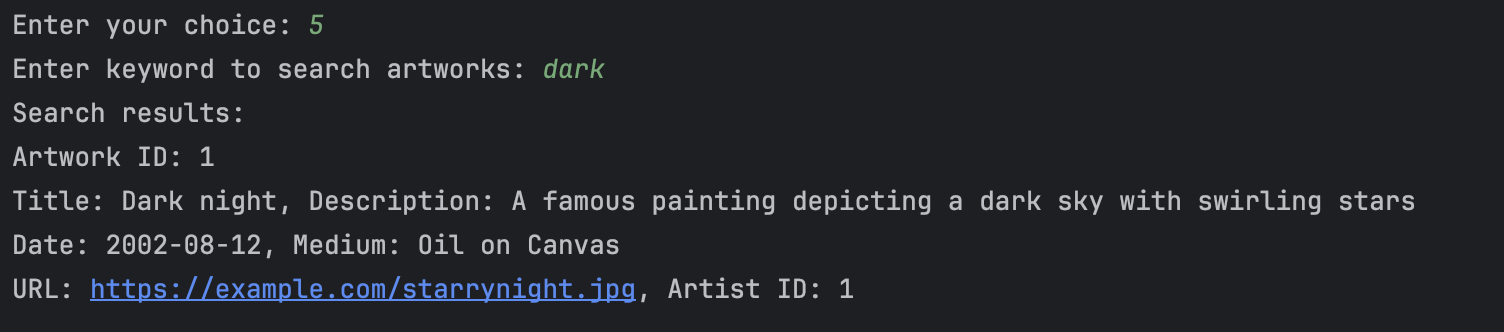


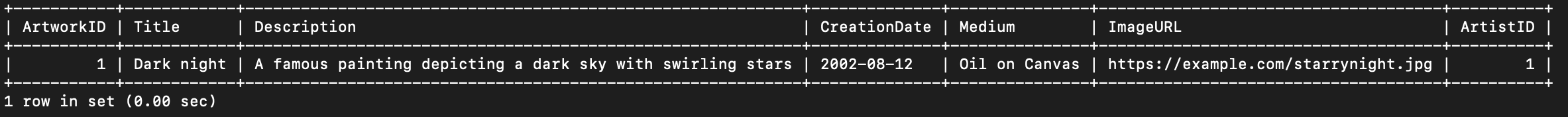
Removed:



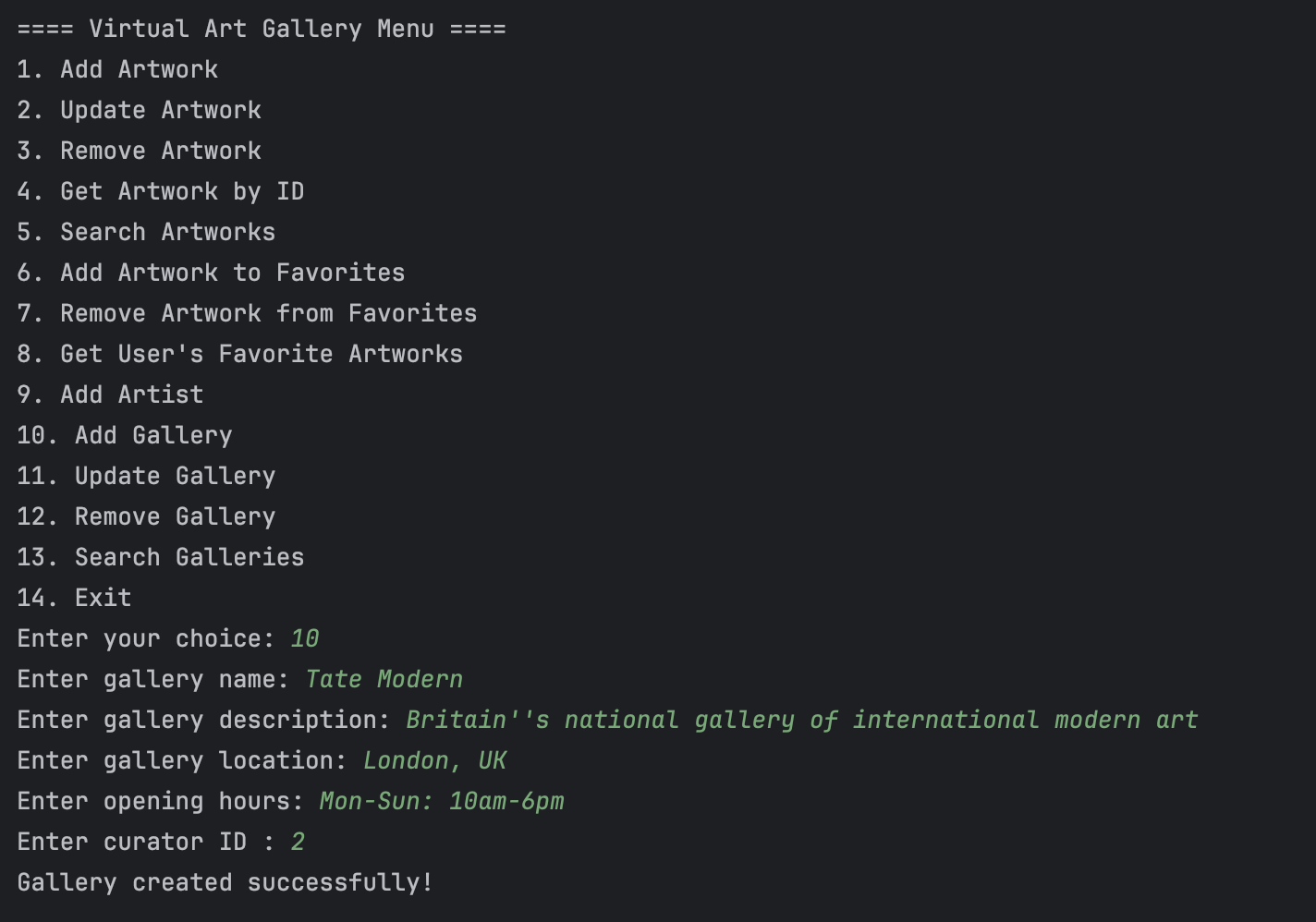
d. Check if searching for artworks returns the expected results.

****

****

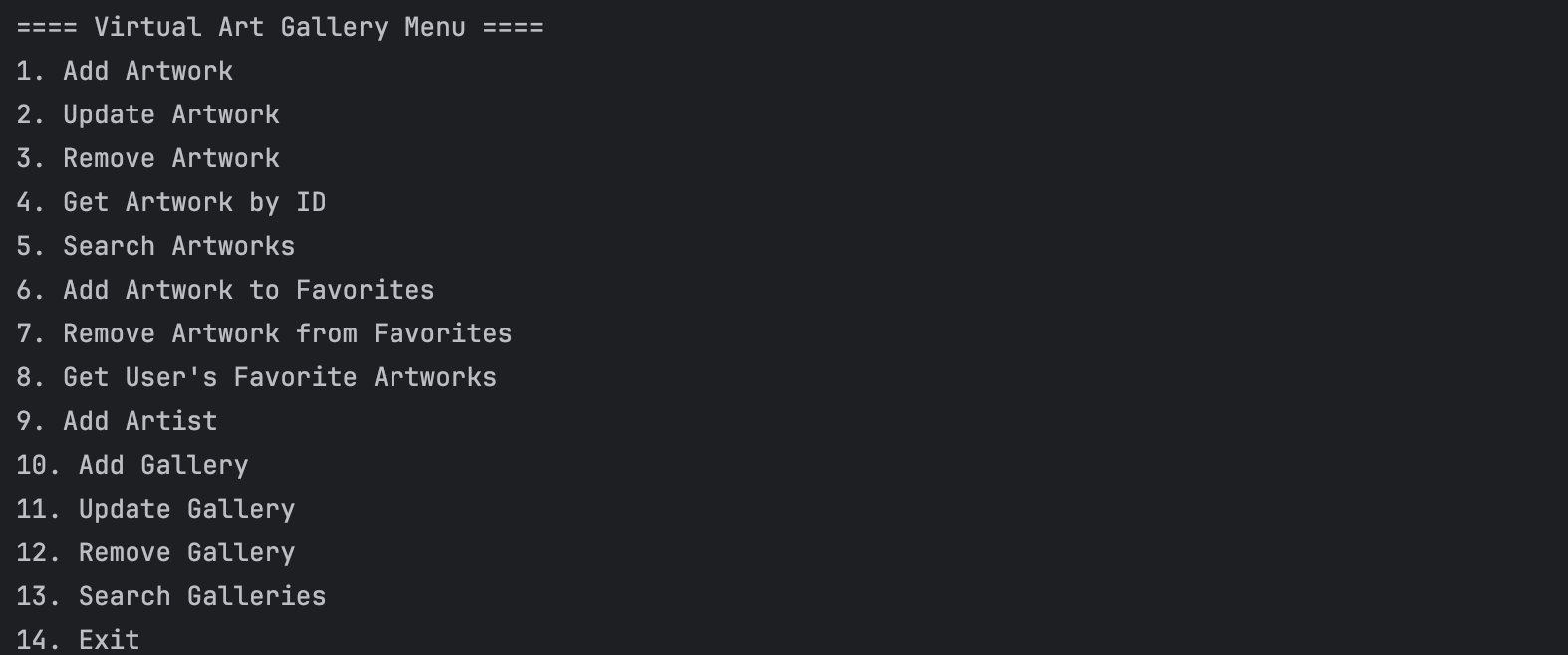
****

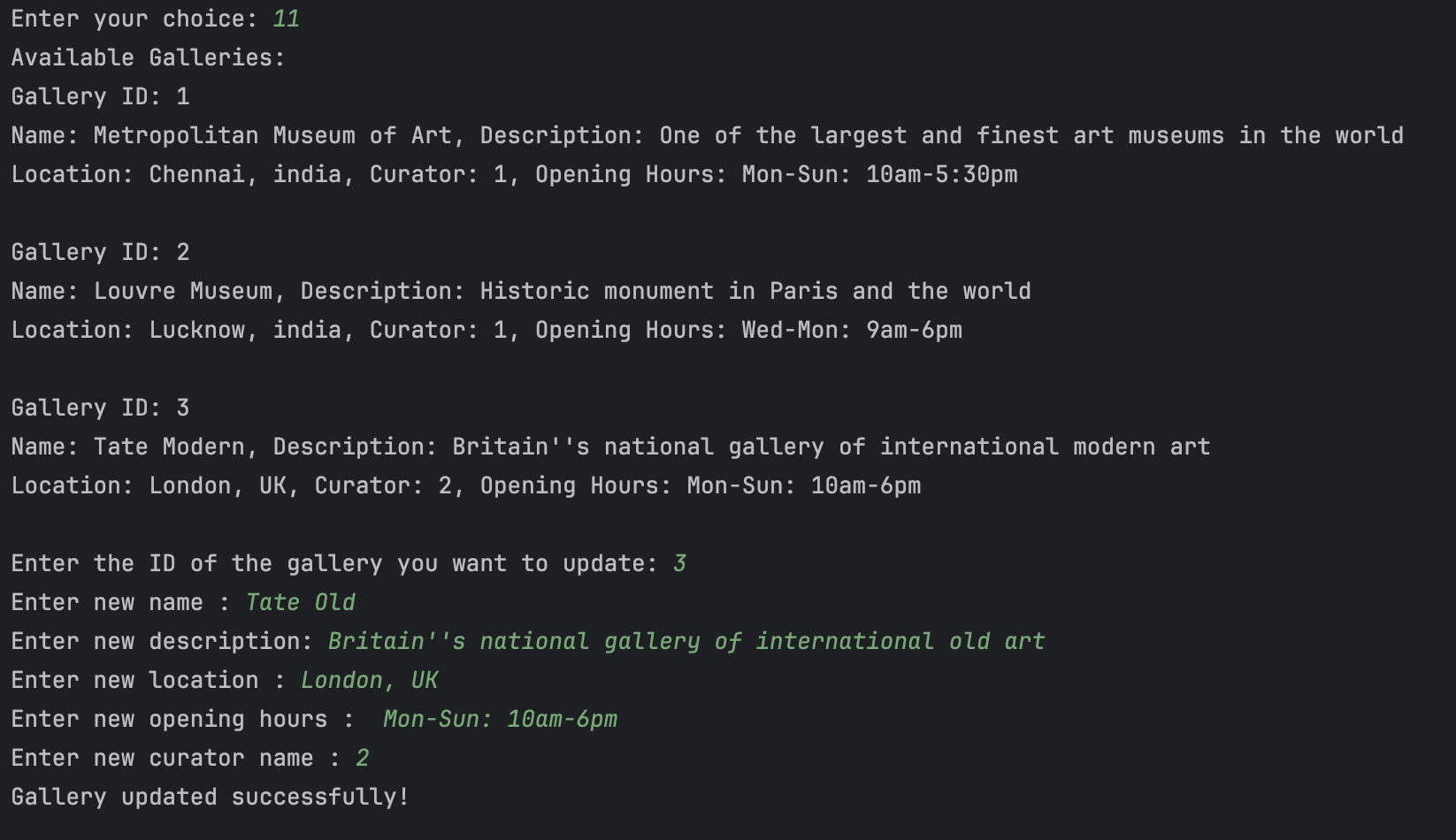
1. **Gallery Management:**
2. Test creating a new gallery.





1. Verify that updating gallery information works correctly.

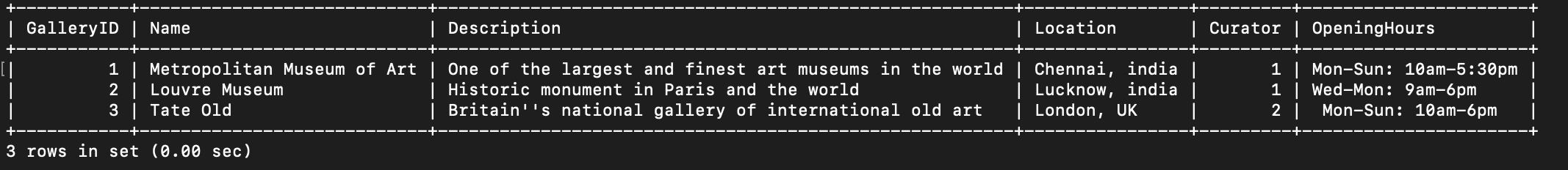




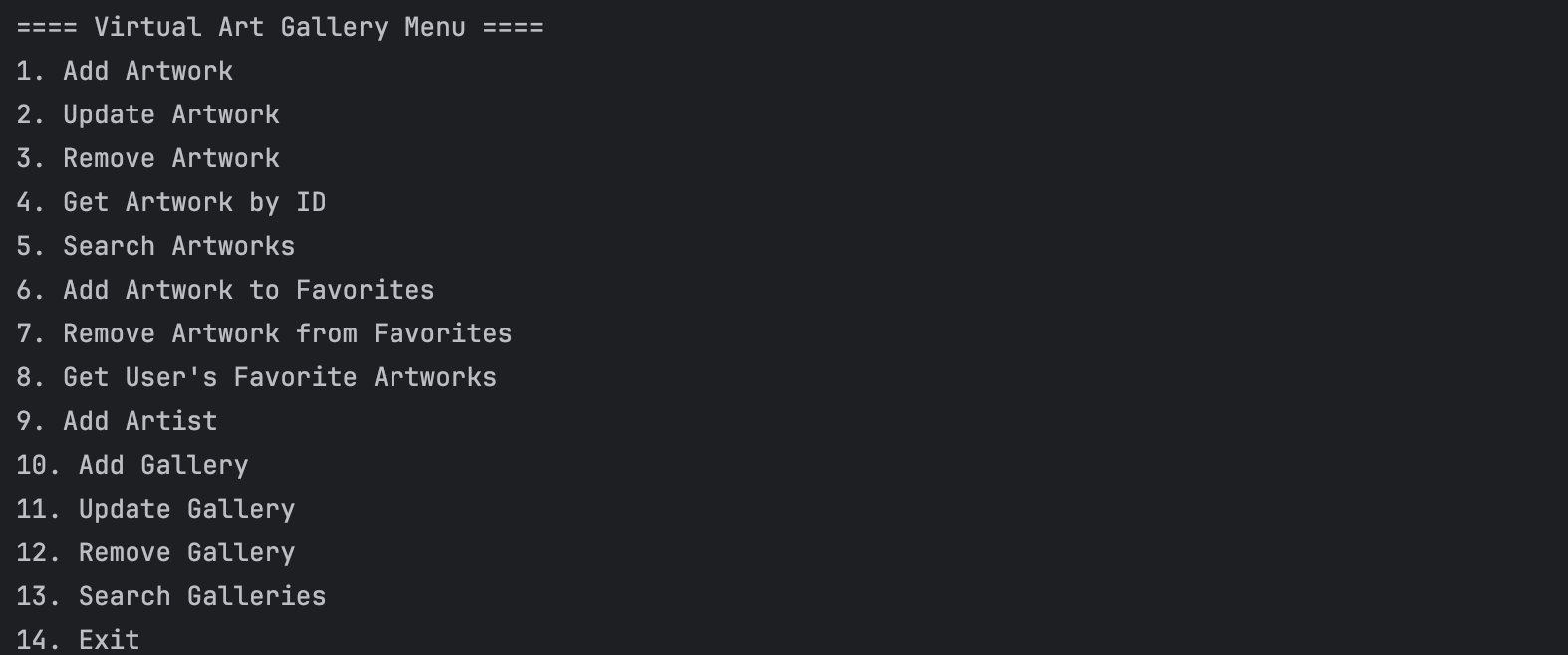
Old:

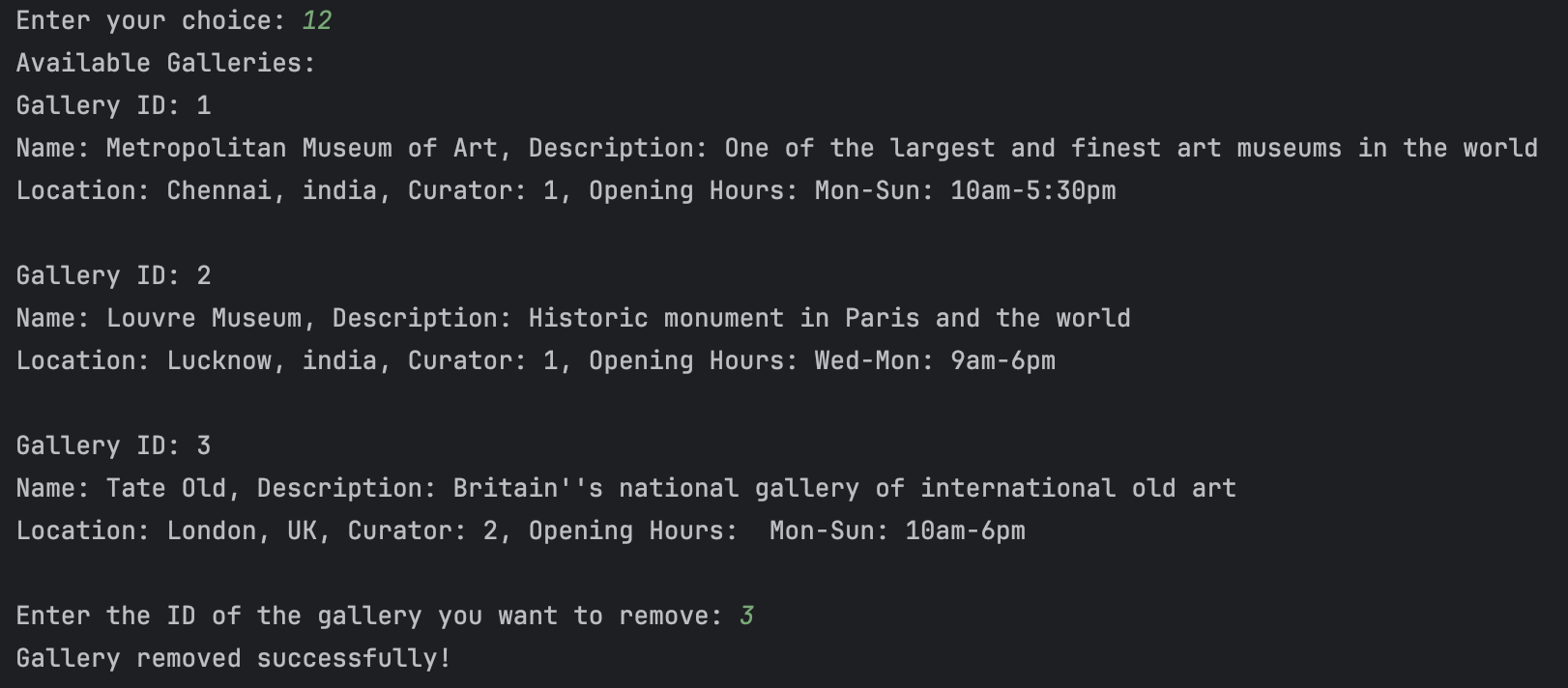


Updated:

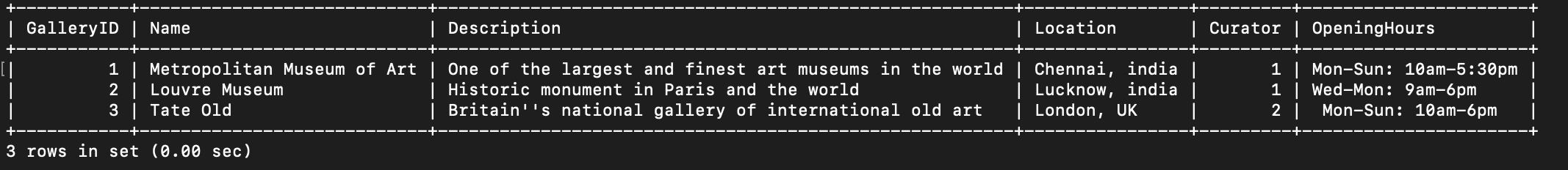


1. Test removing a gallery from the system.

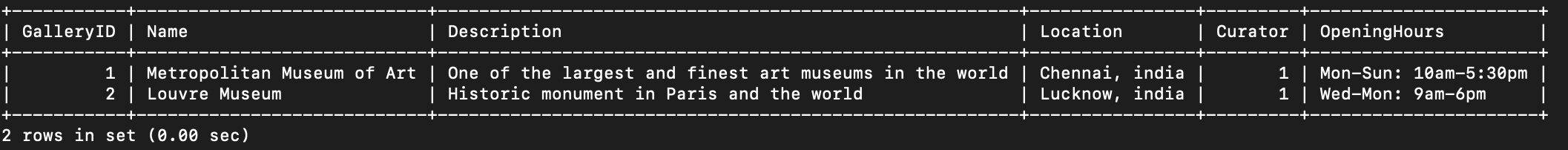




Old:



Removed:



1. Check if searching for galleries returns the expected results.

