Software Requirements Specification

for

Virtual Art Gallery

Version 1.0 approved

Prepared by Mehul Mathur
-21BCE2099

Vellore Institute of Technology

29th January, 2024

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to specify the software requirements for the Virtual Art Gallery project. This project involves creating a virtual art gallery application using Python, Pygame, and ModernGL. The software will simulate a gallery environment, allowing users to navigate through virtual rooms and view digital representations of artworks.

1.2 Document Conventions

This document follows standard documentation conventions. Priorities for higher-level requirements are assumed to be inherited by detailed requirements. Each requirement statement will have its own priority assigned.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, project managers, testers, and anyone involved in the development process. Developers should focus on the technical details, project managers on the overall scope, and testers on verification and validation. Readers are suggested to start with the overview sections and proceed to the sections most relevant to their roles.

1.4 Product Scope

The Virtual Art Gallery is a software application designed to create an immersive experience for art enthusiasts. Users will be able to explore virtual gallery spaces, view digital artworks, and interact with the environment. The primary goal is to provide an engaging platform for art appreciation, bridging the gap between physical and virtual art experiences.

1.5 References

- Pygame Documentation: https://www.pygame.org/docs/ref/pygame.html
- ModernGL Documentation: https://moderngl.readthedocs.io/en/5.8.2/
- Python Documentation: https://docs.python.org/3.11/

2. Overall Description

2.1 Product Perspective

The Virtual Art Gallery is a standalone application that provides a virtual environment for art exploration. It interacts with the Pygame and ModernGL libraries to create a visually appealing and interactive gallery experience.

2.2 Product Functions

- Room navigation
- Artwork display
- User interaction (e.g., zoom, rotate)
- Virtual gallery customization

2.3 User Classes and Characteristics

- Art Enthusiasts: Users who are interested in exploring and experiencing art in a virtual environment.
- Developers: Those involved in maintaining or extending the functionality of the Virtual Art Gallery.

2.4 Operating Environment

The application is designed to run on platforms supporting Python, Pygame, and ModernGL.

2.5 Design and Implementation Constraints

- Compatibility with Pygame and ModernGL
- Use of 3D models for artworks

2.6 Assumptions and Dependencies

- Users have a basic understanding of navigation controls.
- Pygame and ModernGL are installed on the user's system.

2.7 User Documentation

2.7.1 User Guides

Installation: Step-by-step guide for installing the Virtual Art Gallery.
Navigation: Instructions on navigating virtual rooms, zooming, rotating, and customizing the gallery.

2.7.2 Developer Documentation

- Code Structure: Overview of the codebase organization.
- API Documentation: Details on Pygame and ModernGL usage.
- Dependencies: Information on external tools and libraries.
- Contribution Guidelines: Coding standards and contribution best practices.

2.7.3 Help System

In-App Help: Context-sensitive guidance for users within the application.

2.7.4 Accessibility Documentation

Accessibility Guide: Making the Virtual Art Gallery user-friendly for individuals with disabilities.

3. External Interface Requirements

3.1 User Interfaces

The application will have an intuitive graphical user interface (GUI) implemented using Pygame.

3.2 Hardware Interfaces

The Virtual Art Gallery requires a standard computer system with compatible graphics hardware.

3.3 Software Interfaces

- Pygame version 2.5.2: Utilized for GUI implementation and user interaction.
- ModernGL version 5.8.2: Used for rendering 3D models of artworks.
- Python version 3.11: The programming language for Virtual Art Gallery development.

4. System Features

4.1 Room Navigation

Priority: High

Description: Users can navigate between virtual rooms using keyboard controls.

Functional Requirements:
- Provide forward, backward, left, and right navigation controls.
- Implement a smooth transition between rooms.

4.2 Artwork Display

Priority: High

Description: Display digital representations of artworks within the virtual gallery.

Functional Requirements:
- Load and render 3D models of artworks.
- Support various file formats for artwork representations.

4.3 User Interaction

Priority: Medium

Description: Allow users to interact with displayed artworks (e.g., zoom, rotate).

Functional Requirements:
- Implement zoom in/out controls.
- Enable rotation of artworks for detailed viewing.

4.4 Virtual Gallery Customization

Priority: Low

Description: Allow users to customize aspects of the virtual gallery.

Functional Requirements:
- Change gallery themes or layouts.
- Add or remove artworks from the gallery.

5. Non-functional Requirements

5.1 Performance Requirements

- The application should provide a smooth and responsive navigation experience.
- Artwork rendering should be efficient and visually appealing.

5.2 Security Requirements

No specific security requirements apply to this application.

5.3 Software Quality Attributes

- The user interface should be visually pleasing and easy to navigate.
- The application should handle errors gracefully and provide meaningful error messages.

5.4 Security Requirements

No specific security requirements apply to this application.

6. Other Requirements

6.1 Legal and Regulatory Requirements

The Virtual Art Gallery must comply with relevant copyright and intellectual property laws.

6.2 Documentation Requirements

Comprehensive documentation, including user guides and developer documentation, should be provided.

Conclusion

This Software Requirements Specification outlines the key features, functionalities, and constraints of the Virtual Art Gallery project. It serves as a guide for the development team, providing a clear understanding of the project's scope and requirements. Further details and refinements may be added as the project progresses.

Appendix A: Glossary

1. Virtual Art Gallery Terms

- Virtual Art Gallery: The software application designed to create an immersive art exploration experience in a virtual environment.
- Pygame: A cross-platform set of Python modules designed for writing video games, including graphical user interfaces.
- ModernGL: A Python binding to the OpenGL API, providing a modern approach to OpenGL programming.
- 3D Models: Digital representations of artworks in three-dimensional space, used for rendering in the virtual gallery.

- User Guides: Documentation components providing instructions for installation, navigation, and customization of the Virtual Art Gallery.

- Developer Documentation: Comprehensive documentation for developers, including code structure, API details, and contribution guidelines.

- In-App Help: Context-sensitive guidance system within the application to assist users during their virtual gallery experience.

2. Abbreviations and Acronyms

- SRS: Software Requirements Specification
- GUI: Graphical User Interface
- API: Application Programming Interface

Appendix B: Analysis Models

1. Data Flow Diagram

Figure 1: Represents the flow of data within the Virtual Art Gallery system.

2. Class Diagram

Figure 2: Illustrates the class structure and relationships within the Virtual Art Gallery application.

3. State-Transition Diagram

Figure 3: Depicts the states and transitions of the Virtual Art Gallery during user interactions.

Appendix C: To Be Determined List

1. TBD-1: Specific details of the file formats supported for artwork representations.
2. TBD-2: Clarification on the integration with external libraries for enhanced functionalities.
3. TBD-3: Additional security measures that may be required during user interactions.
4. TBD-4: Definition of potential business rules that could impact the application's design.

** *Note:* To Be Determined (TBD) list serves as a tracking mechanism for pending decisions or details within the Software Requirements Specification. Each TBD reference will be updated as decisions are made or details are finalized.

7. Development Model Selection: A Holistic Approach

For the Virtual Art Gallery project, an **Incremental Model** would be a suitable prescriptive process model. The Incremental Model is characterized by breaking down the software development process into smaller, manageable parts and delivering them incrementally. Each increment represents a portion of the final product, and new features or improvements are added with each iteration. This model aligns well with the nature of the project for the following reasons:

1. **Evolutionary Development:** The Incremental Model allows for the evolutionary development of the Virtual Art Gallery. Since the application involves creating a virtual environment with various features (e.g., navigation, artwork display, customization), developing and delivering the system incrementally will allow for continuous improvement and refinement.

2. **Feedback Incorporation:** As each increment is delivered, it can be tested and reviewed by stakeholders. This iterative approach allows for early feedback, and any necessary adjustments or enhancements can be incorporated into subsequent increments. This is particularly beneficial for a project like the Virtual Art Gallery, where user experience and visual appeal are crucial.

3. **Reduced Risk:** By dividing the project into manageable increments, the risk associated with the entire development process is reduced. It becomes easier to identify and address issues early in the development cycle. This is important for a project that involves multiple components like Pygame, ModernGL, and 3D models, as it allows for the early detection and resolution of integration or compatibility issues.

4. **Flexibility in Development:** The Incremental Model provides flexibility in accommodating changes and enhancements. This is advantageous for a project where requirements might evolve, and new features or modifications could be requested as the project progresses.

5. **Early Delivery of Core Functionality:** The Virtual Art Gallery can be delivered in functional increments, allowing users to start experiencing core functionalities sooner rather than waiting for the entire system to be completed. This is particularly beneficial for projects aiming for quick releases and user engagement.

6. **Parallel Development:** Different increments can be developed in parallel by different teams or individuals, making the development process more efficient. This is helpful for a project where various aspects like navigation, artwork rendering, and customization can be developed concurrently.

The Incremental Model strikes a balance between managing complexity and providing tangible deliverables at each stage, making it well-suited for the development of the Virtual Art Gallery.

8. Work Breakdown Structure (WBS)

8.1. Process-Based WBS:

8.1.1 Requirements Analysis
- 8.1.1.1 Elicitation of User Requirements
- 8.1.1.2 Analysis of Pygame and ModernGL Integration
- 8.1.1.3 Definition of Navigation and Artwork Display Requirements

8.1.2 Design
- 8.1.2.1 High-Level Design of Virtual Gallery Environment
- 8.1.2.2 Detailed Design of User Interface Components
- 8.1.2.3 3D Model Design for Artworks

8.1.3 Implementation
- 8.1.3.1 Setting Up Pygame and ModernGL Environment
- 8.1.3.2 Coding Navigation Controls
- 8.1.3.3 Implementing Artwork Rendering
- 8.1.3.4 User Interaction Implementation
- 8.1.3.5 Virtual Gallery Customization Logic

8.1.4 Testing
- 8.1.4.1 Unit Testing of Navigation and Artwork Display
- 8.1.4.2 Integration Testing of User Interaction Features
- 8.1.4.3 System Testing of the Complete Virtual Art Gallery

8.1.5 Deployment
- 8.1.5.1 Deployment of the Virtual Art Gallery Application
- 8.1.5.2 User Training and Documentation

8.2. Product-Based WBS:

8.2.1 Software
- 8.2.1.1 Pygame Library
- 8.2.1.2 ModernGL Library
- 8.2.1.3 Virtual Art Gallery Application

8.2.2 Documentation
- 8.2.2.1 User Guides
- 8.2.2.2 Developer Documentation
- 8.2.2.3 In-App Help Documentation

8.2.3 3D Models
- 8.2.3.1 Artwork 3D Models
- 8.2.3.2 Virtual Gallery Layout 3D Models

8.3. Geographic-Based WBS:

8.3.1 Development Team - Onsite
- 8.3.1.1 Requirements Analysis and Design
- 8.3.1.2 Implementation and Testing

8.3.2 Development Team - Offshore
- 8.3.2.1 Software Development (Pygame, ModernGL)
- 8.3.2.2 Documentation Development

8.3.3 User Training - Virtual Sessions
- 8.3.3.1 Virtual Training Sessions for End Users
- 8.3.3.2 Q&A Sessions for User Queries

8.4. Role-Based WBS:

8.4.1 Project Manager
- 8.4.1.1 Project Planning and Scheduling
- 8.4.1.2 Resource Allocation and Management

8.4.2 System Architect
- 8.4.2.1 High-Level System Design
- 8.4.2.2 Definition of System Architecture

8.4.3 Developers
- 8.4.3.1 Pygame and ModernGL Integration
- 8.4.3.2 Coding of Navigation and Artwork Display

8.4.4 Testers
- 8.4.4.1 Test Planning and Strategy
- 8.4.4.2 Unit, Integration, and System Testing

8.4.5 Technical Writers
- 8.4.5.1 User Guides
- 8.4.5.2 Developer Documentation

9. COCOMO Model Estimation

9.1 Function Points (FP) Estimation:

1. Unadjusted Function Points (UFP):
- External Inputs (EI): 6
- External Outputs (EO): 5
- External Inquiries (EQ): 4
- Internal Logical Files (ILF): 3
- External Interface Files (EIF): 2

2. Weighting Factors (WF):
Assigning weights (low, average, high):
- EI: 4
- EO: 5
- EQ: 4
- ILF: 7
- EIF: 5

3. Calculate UFP:
- UFP = (6 * 4) + (5 * 5) + (4 * 4) + (3 * 7) + (2 * 5) = 97

4. Adjustment Factor (AF):
- Assume a nominal adjustment factor of 1.0 for simplicity.

5. Adjusted Function Points (AFP):
- AFP = UFP * AF = 97 * 1.0 = 97

6. Effort Adjustment Factor (EAF):
- Assume a nominal effort adjustment factor of 1.0 for simplicity.

7. Person-Months (PM):
- PM = (AFP * EAF) / PROD, where PROD (Productivity) is assumed to be 2.4 (a typical value for simple/organic projects).
- PM = (97 * 1.0) / 2.4 ≈ 40.42 Person-Months

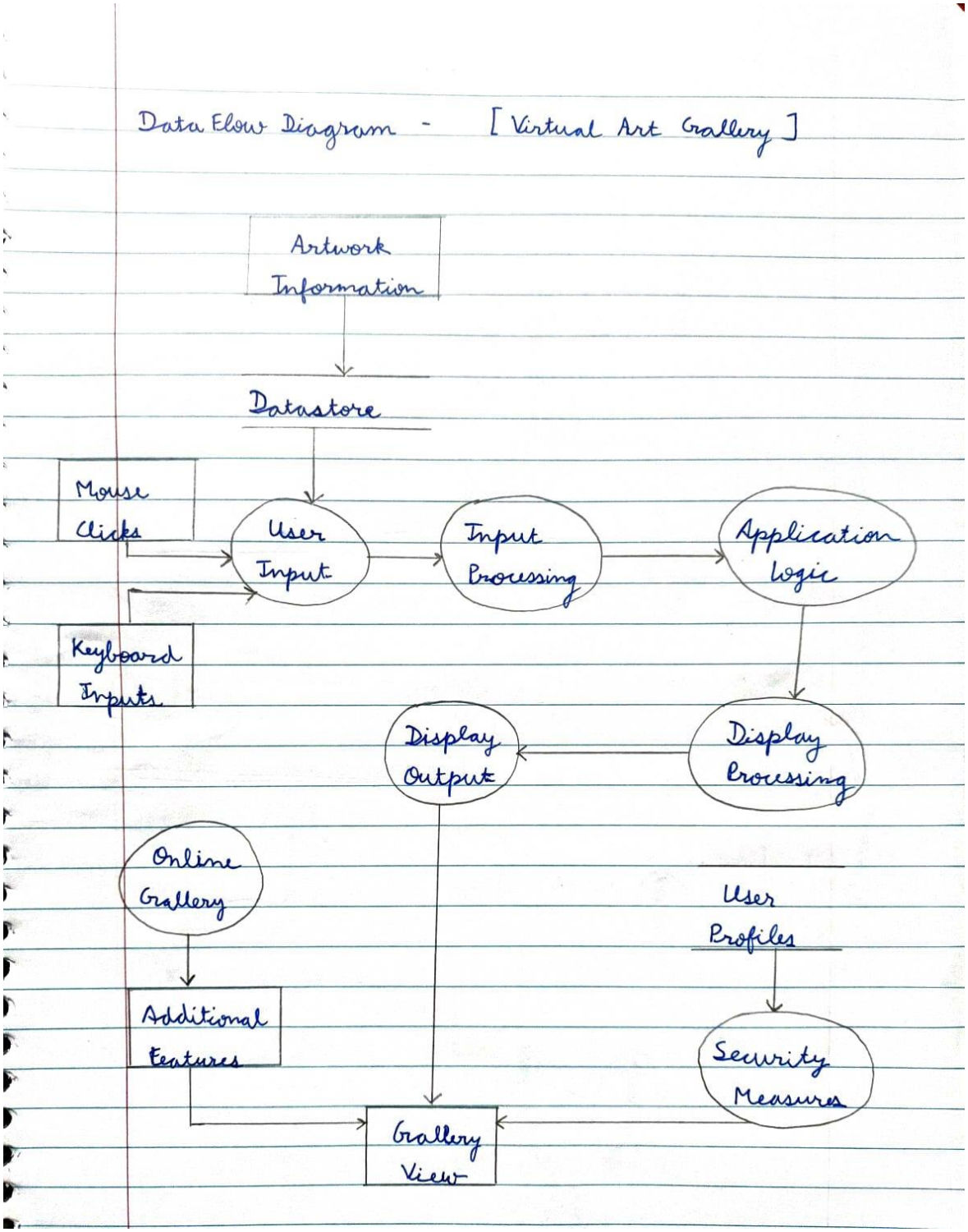
8. Duration (Months):
- Assuming a team size of 5, Duration = PM / Team Size = 40.42 / 5 ≈ 8.08 months

9.2 Gantt Chart:

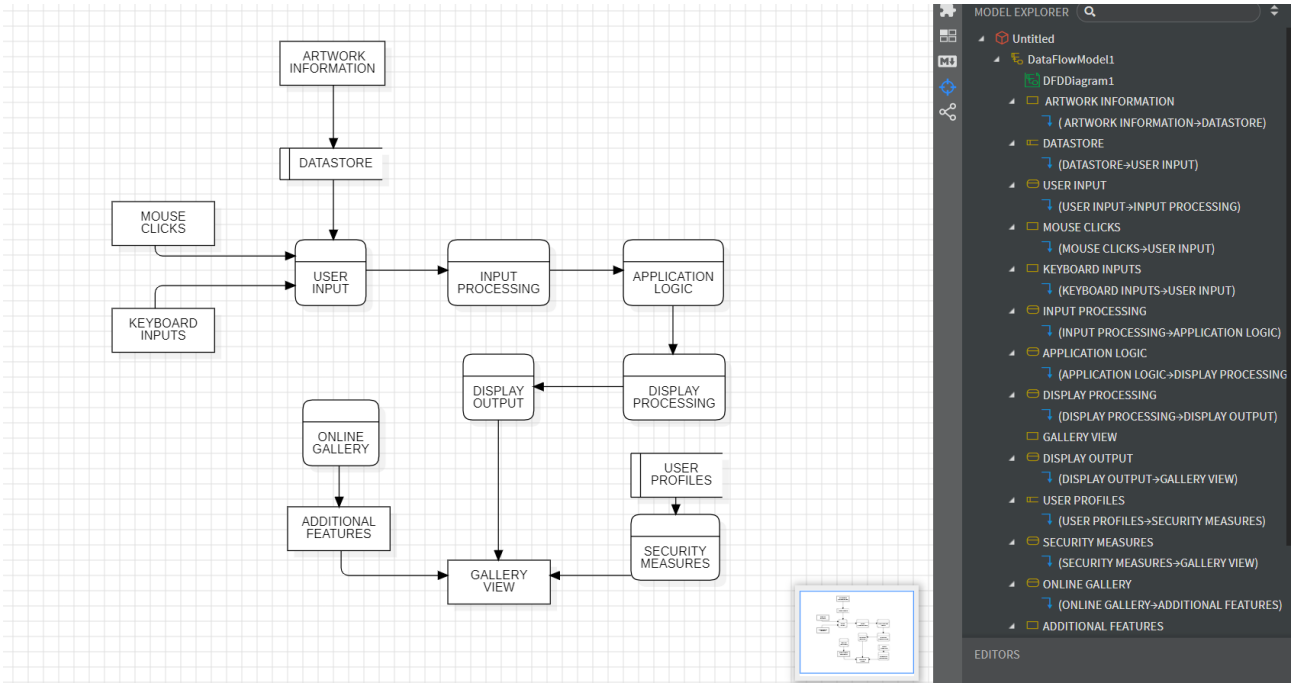
Task	Duration (Months)	Start Date	End Date
Requirements Analysis	1	20/01/2024	19/02/2024
Design	2	20/02/2024	19/4/2024
Implementation	3	20/4/2024	19/7/2024
Testing	2	20/7/2024	19/9/2024
Deployment	1	20/9/2024	19/10/2024
User Training and Documentation	1	20/10/2024	19/11/2024

10. Data Flow Diagram

10.1 Hand-Rendered Illustration

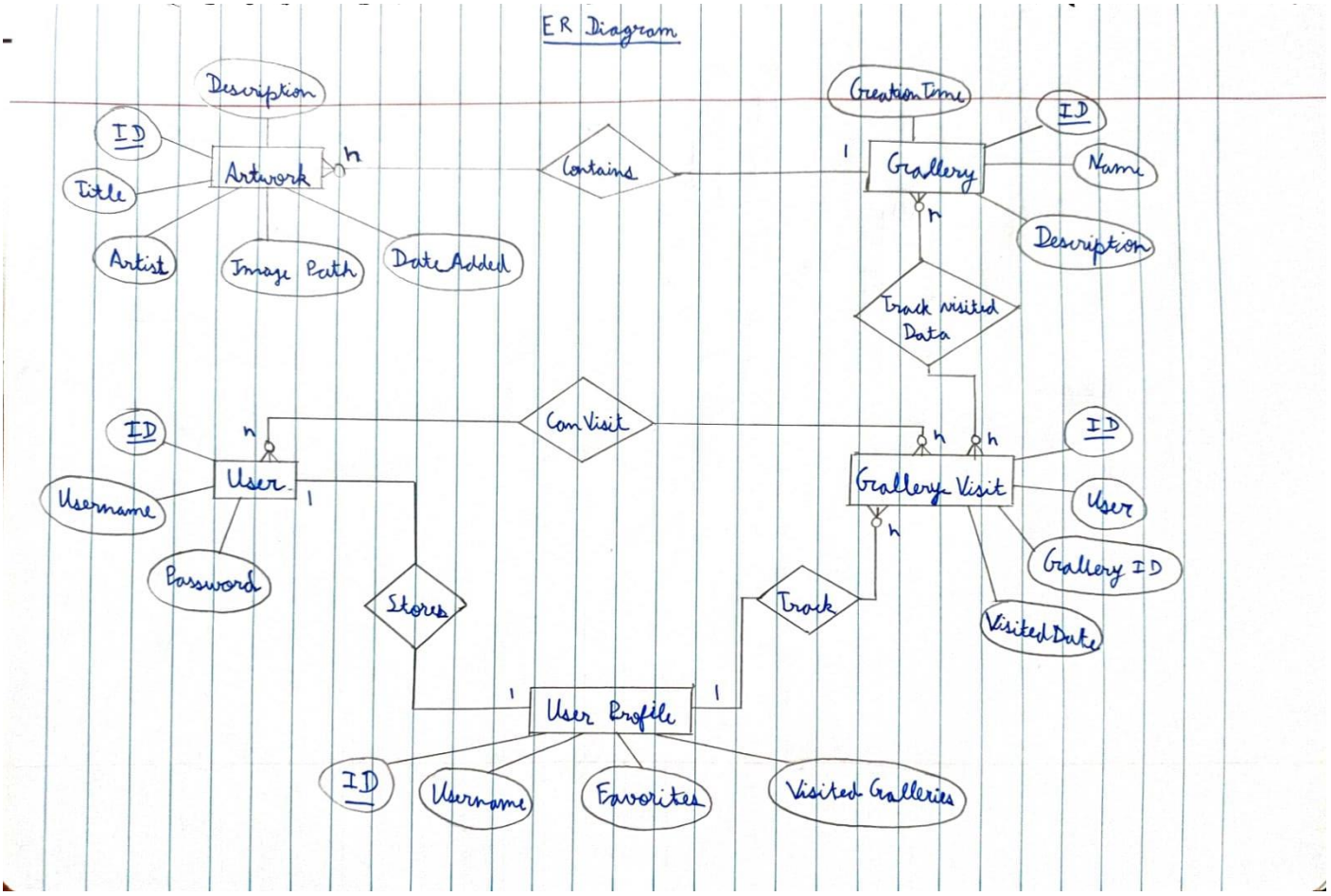


10.2 UML Diagram

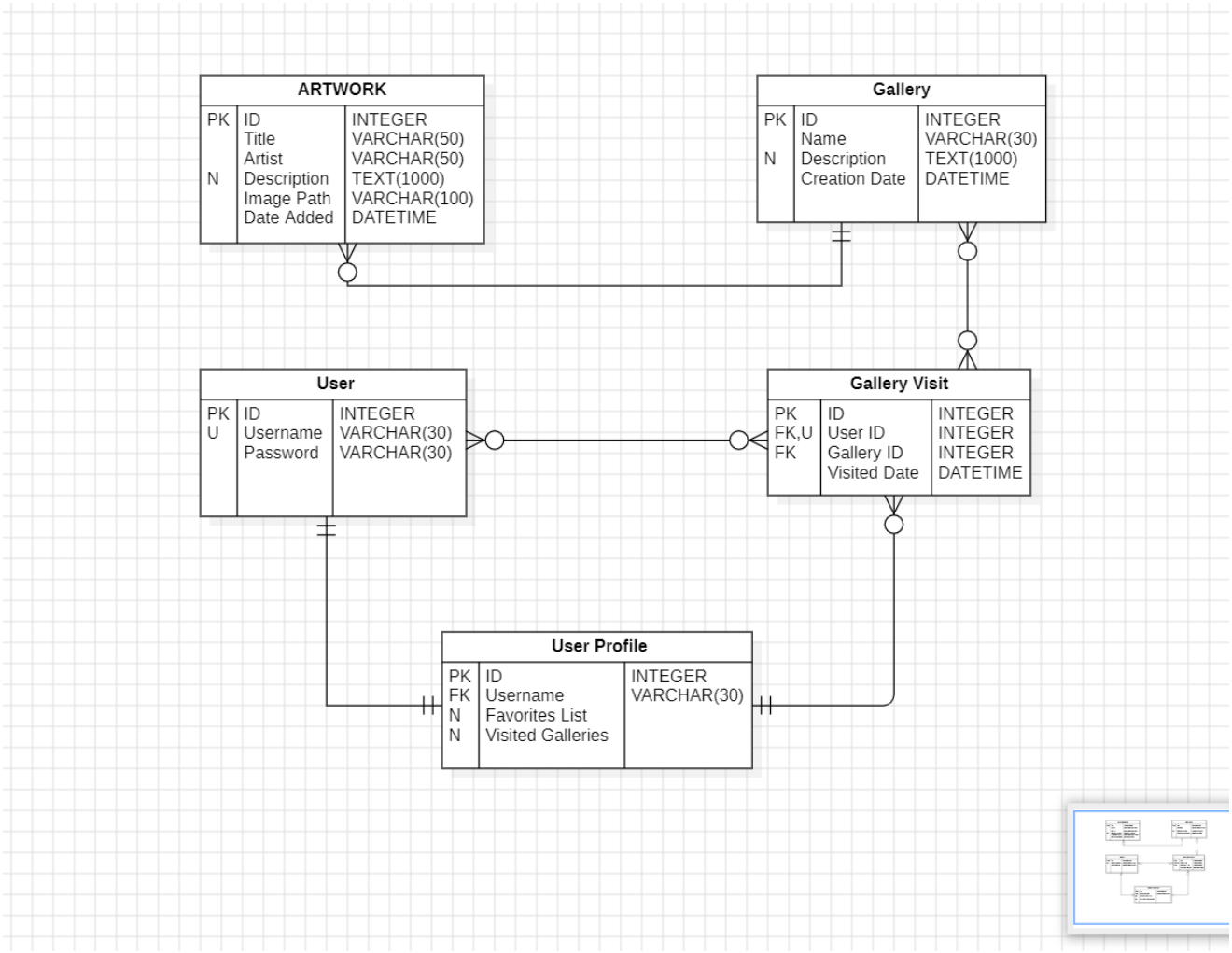


11. ER Diagram

11.1 Hand-Rendered Illustration

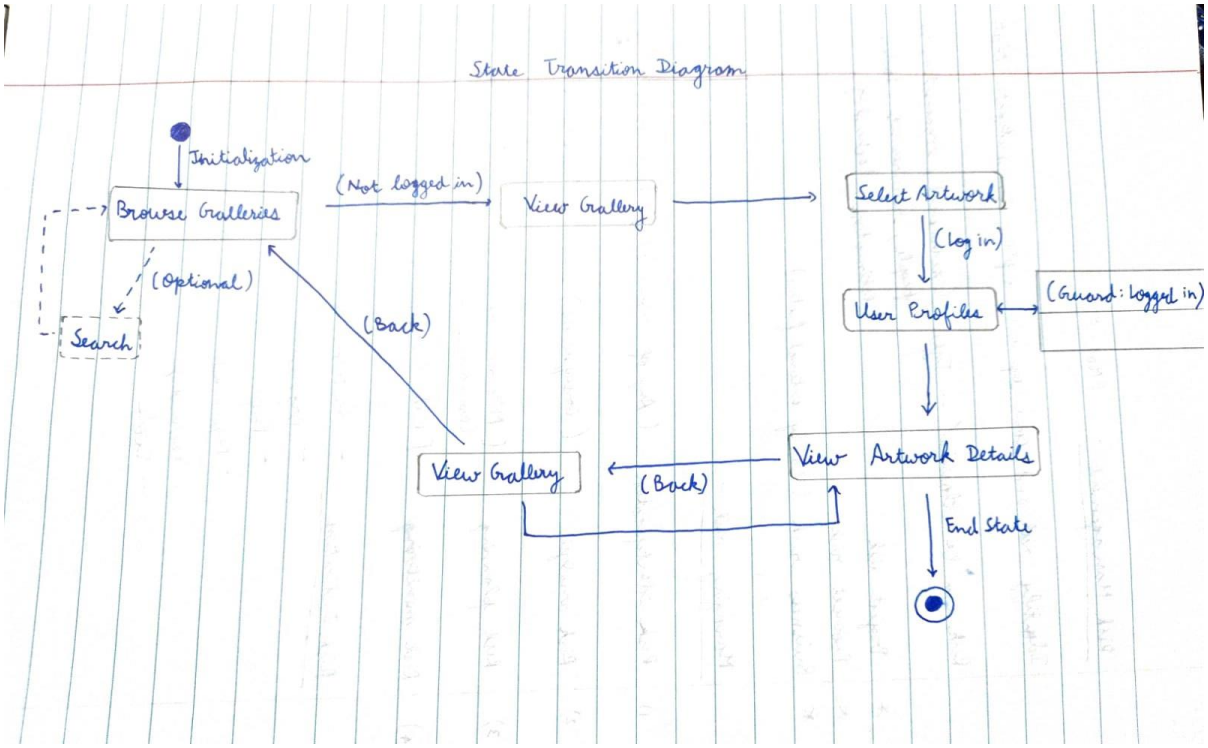


11.2 UML Diagram

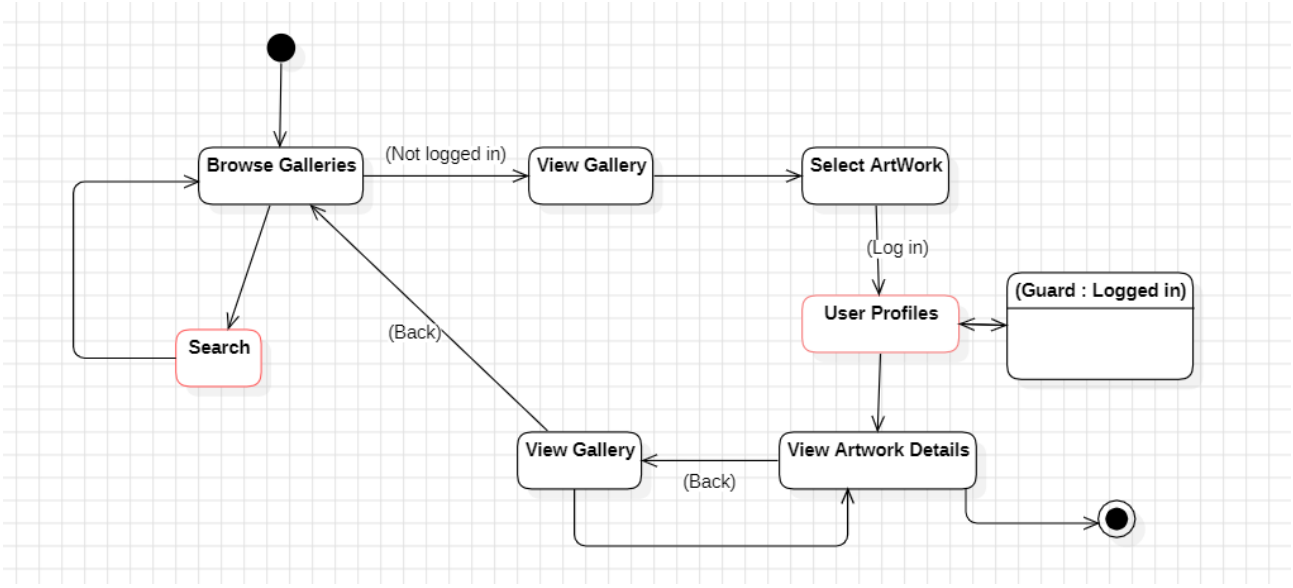


12. State Transition Diagram

12.1 Hand-Rendered Illustration

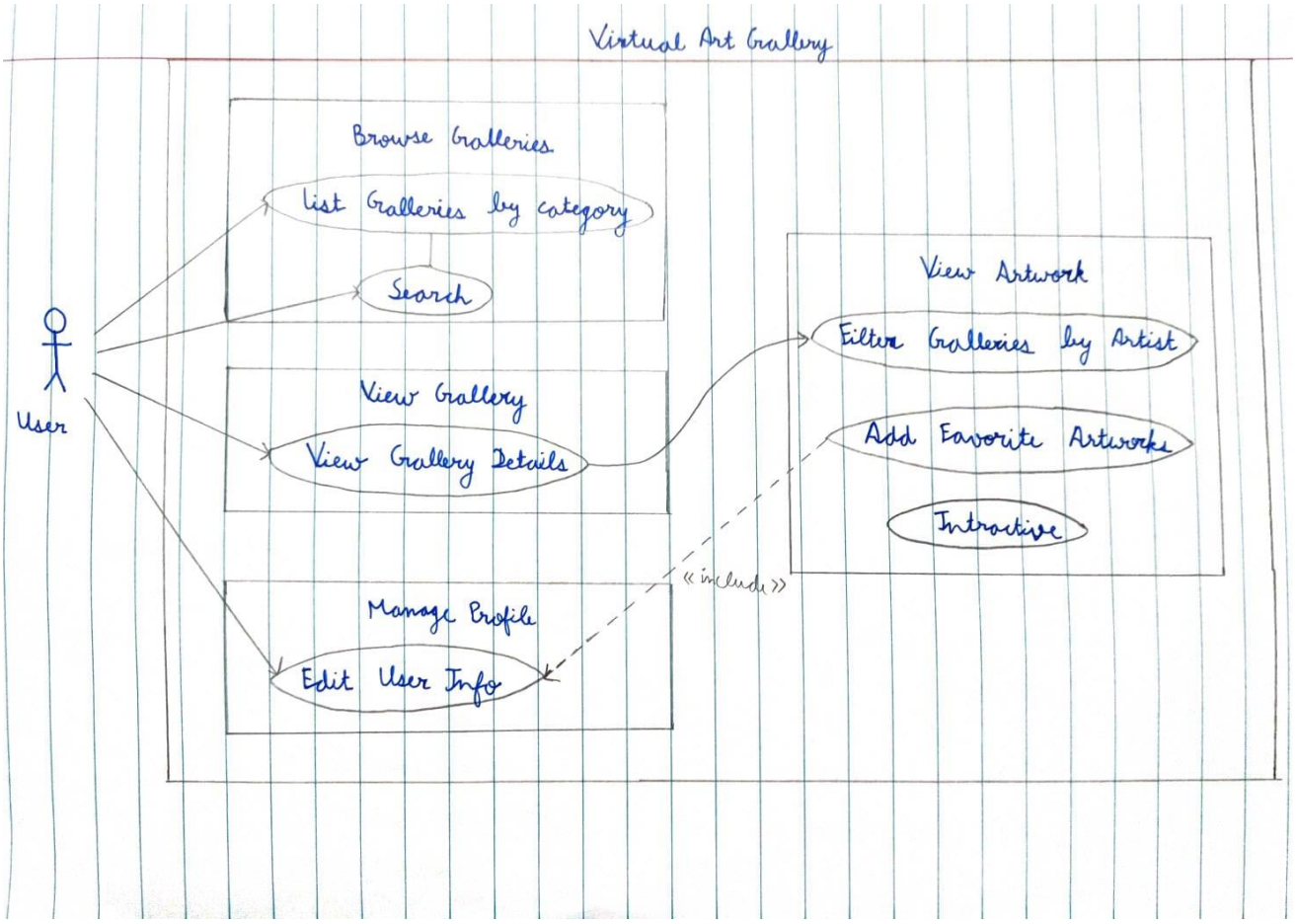


12.2 UML Diagram

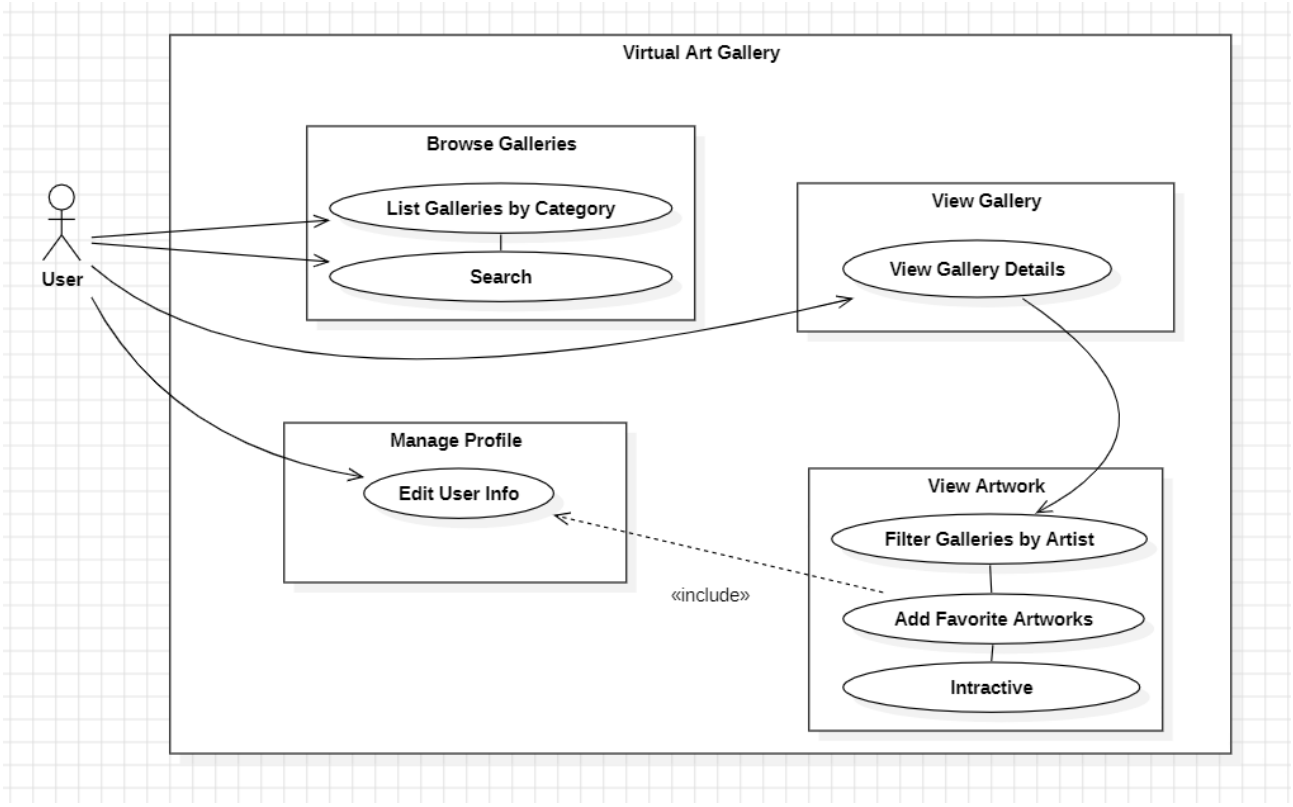


13. Use Case Diagram

13.1 Hand-Rendered Illustration

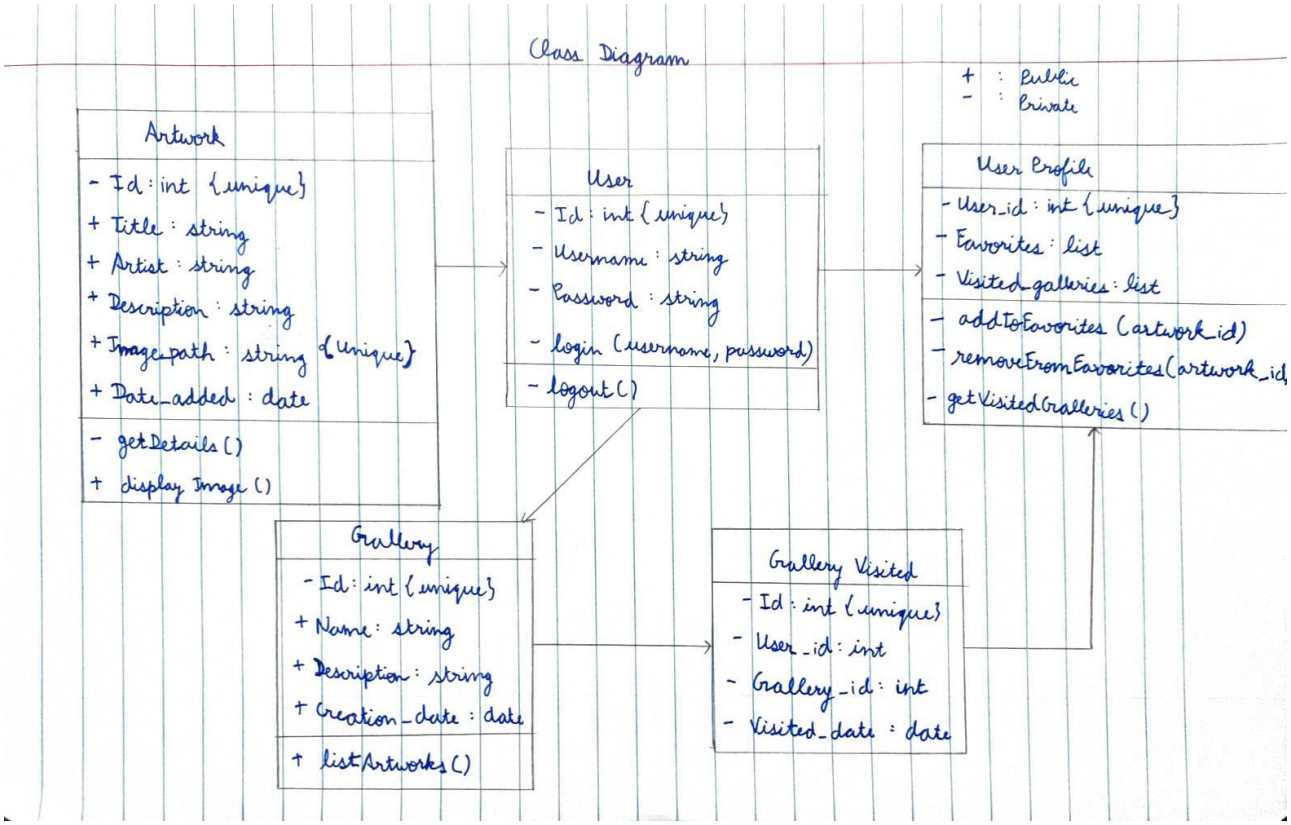


13.2 UML Diagram

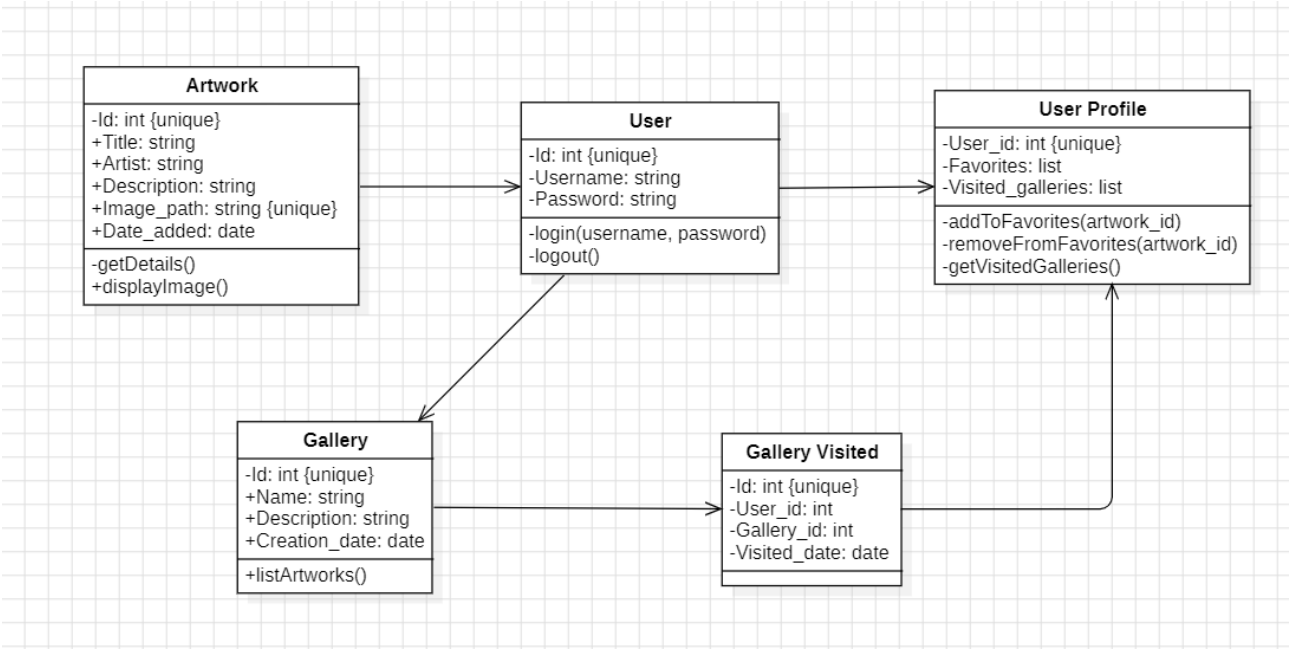


14. Class Diagram

14.1 Hand-Rendered Illustration

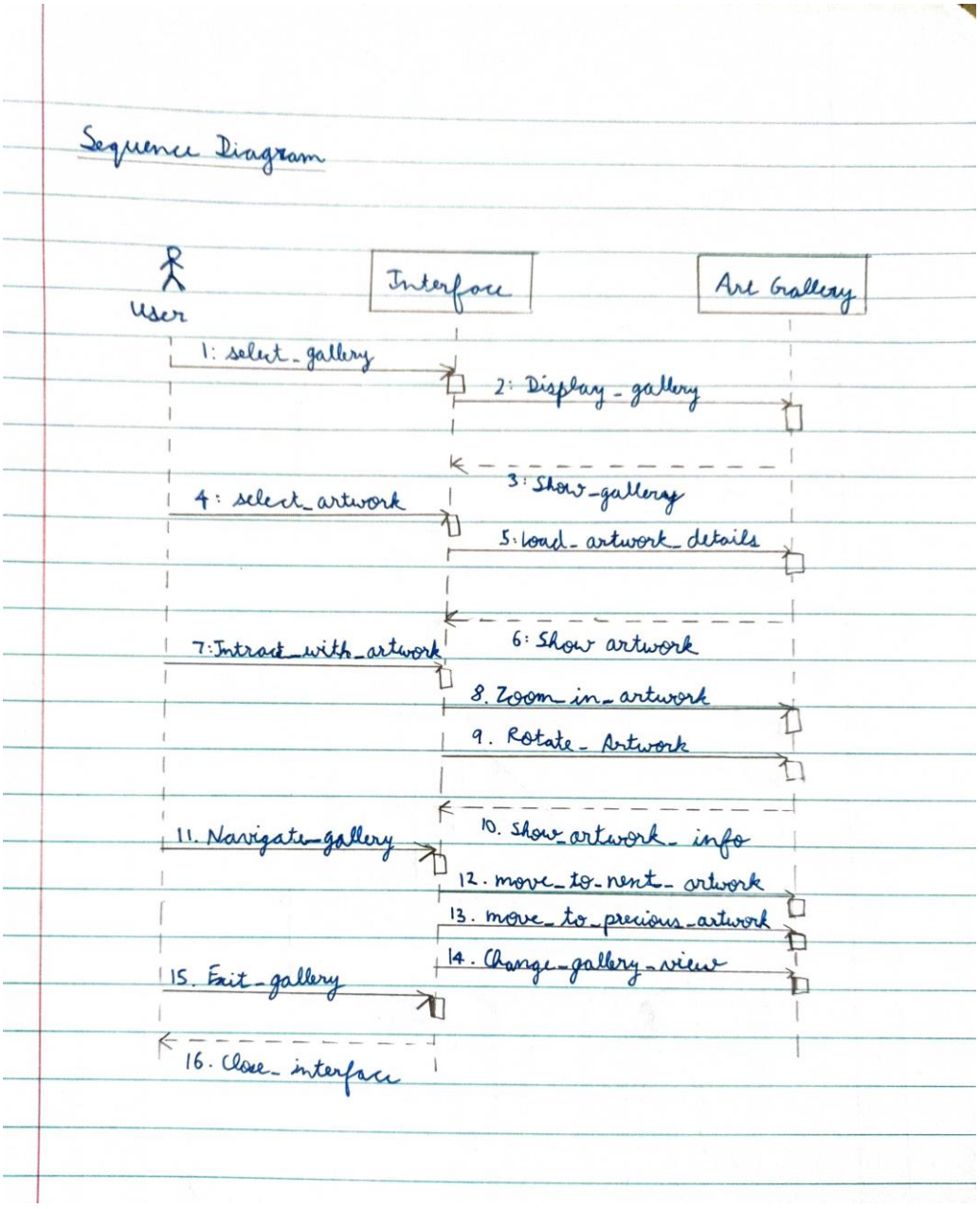


14.2 UML Diagram

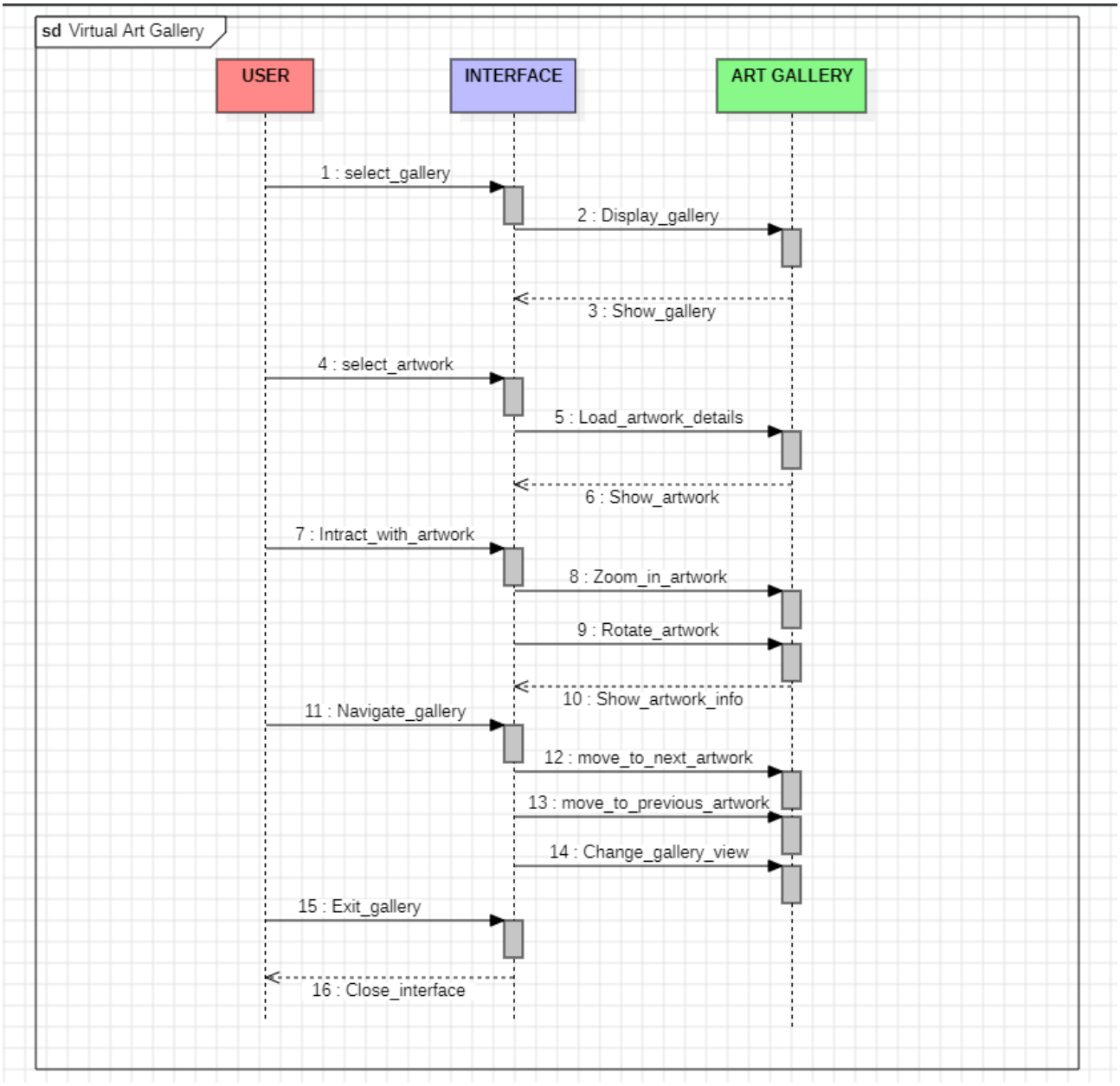


15. Sequence Diagram

15.1 Hand-Rendered Illustration

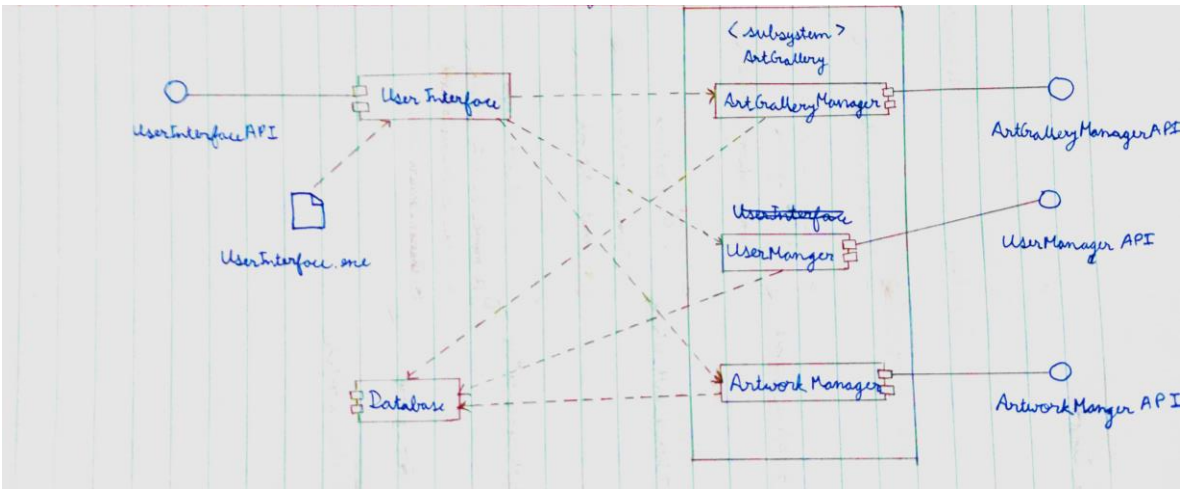


15.2 UML Diagram

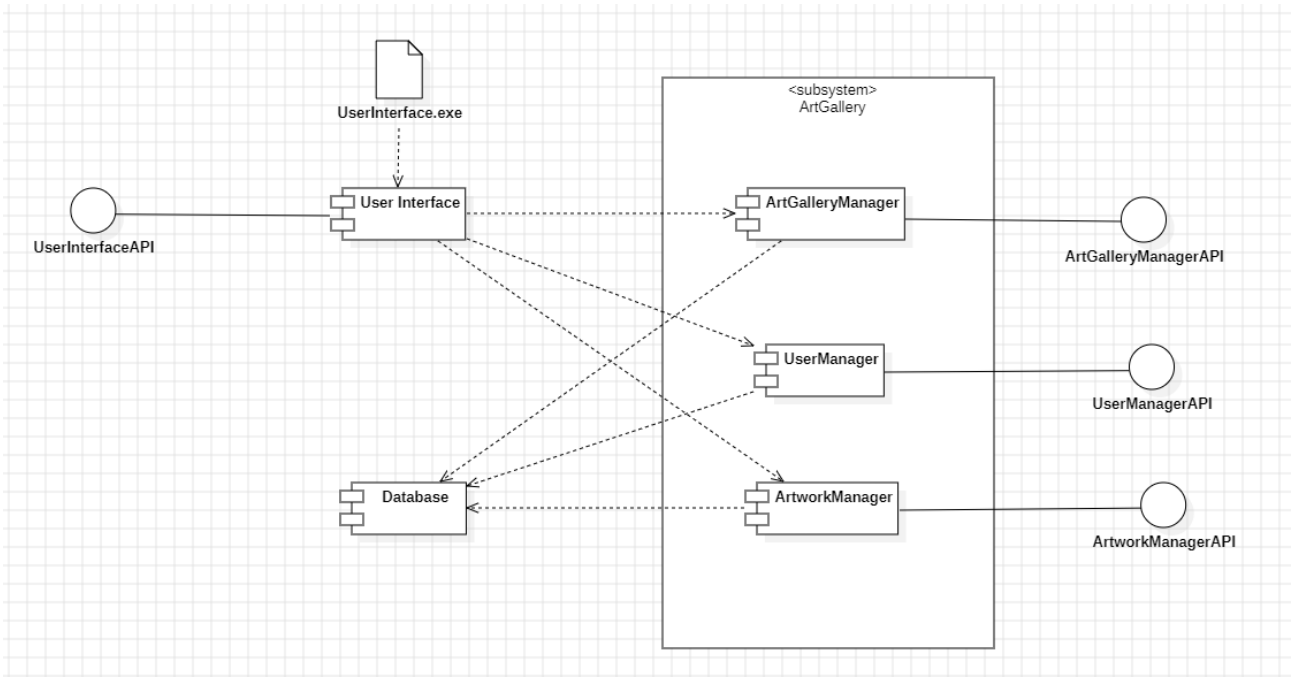


16. Component Diagram

16.1 Hand-Rendered Illustration

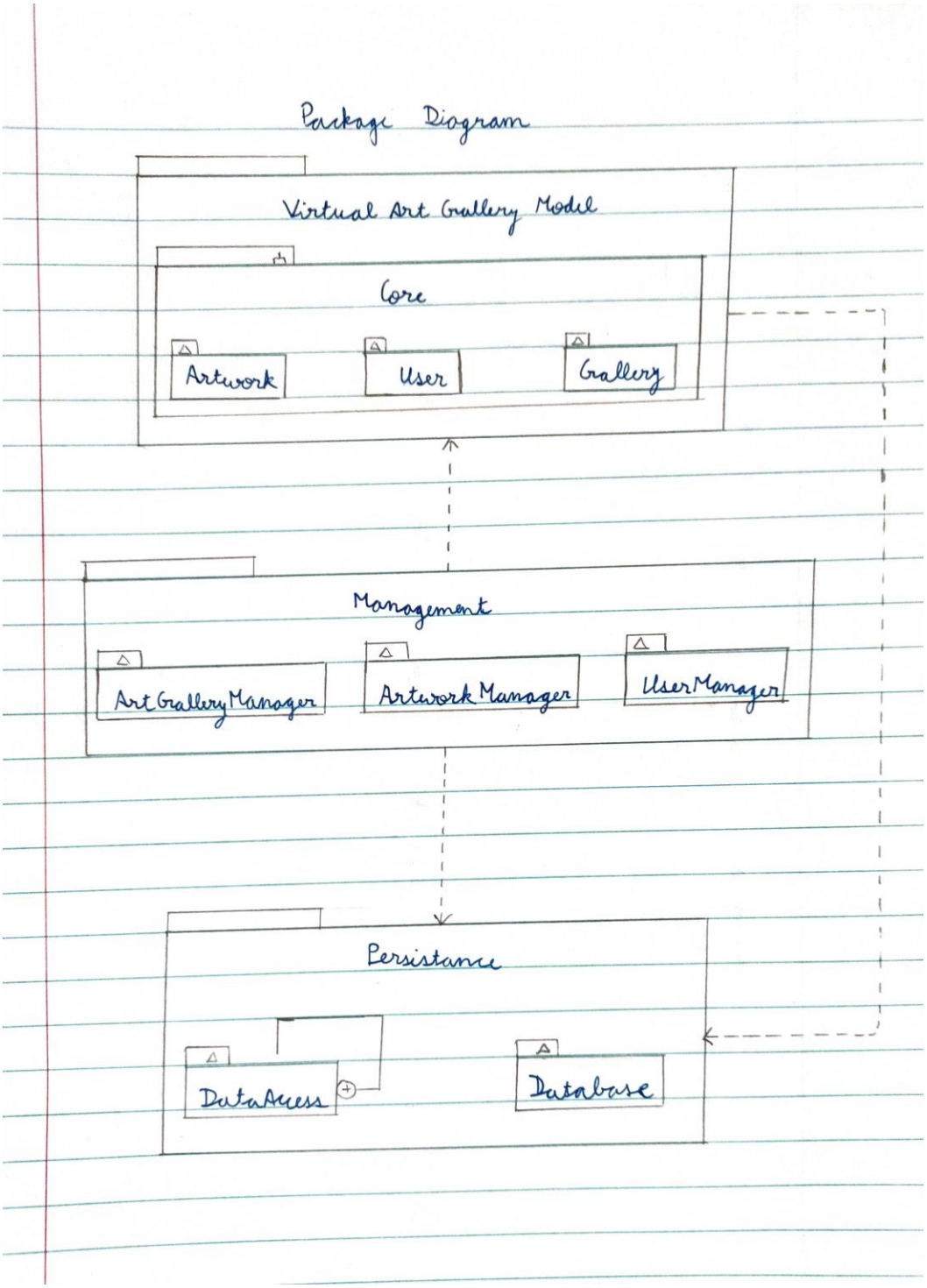


16.2 UML Diagram

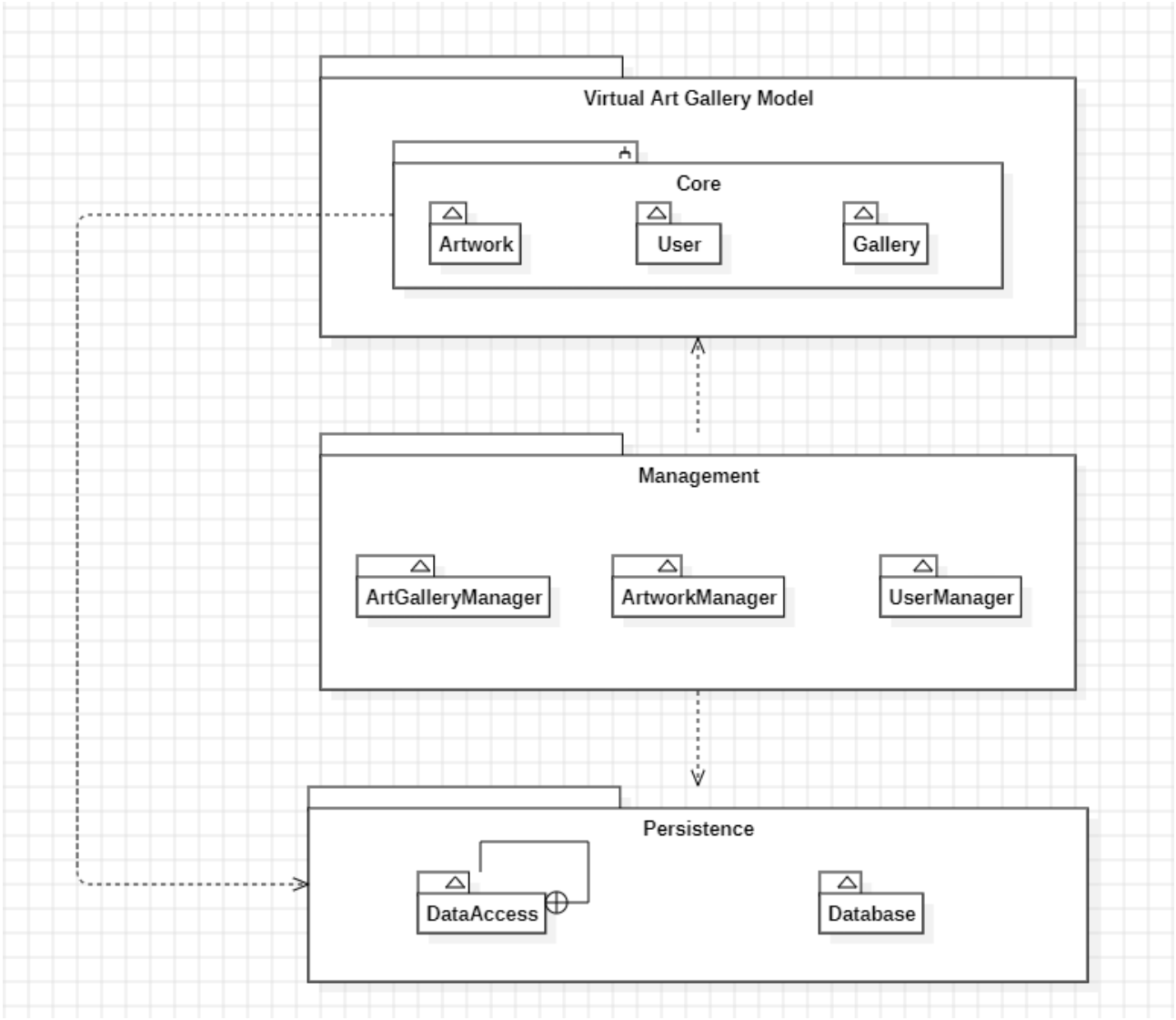


17. Package Diagram

17.1 Hand-Rendered Illustration

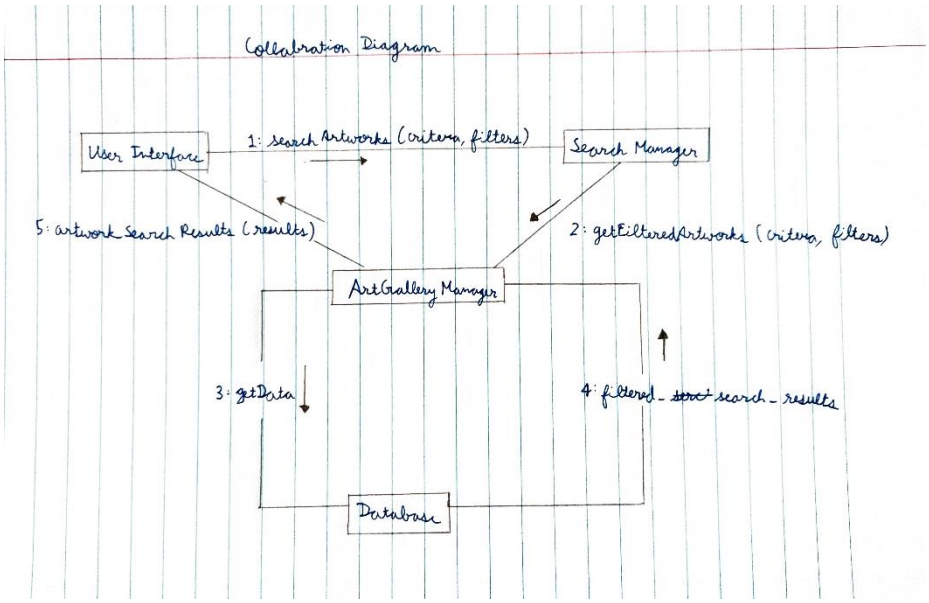


17.2 UML Diagram

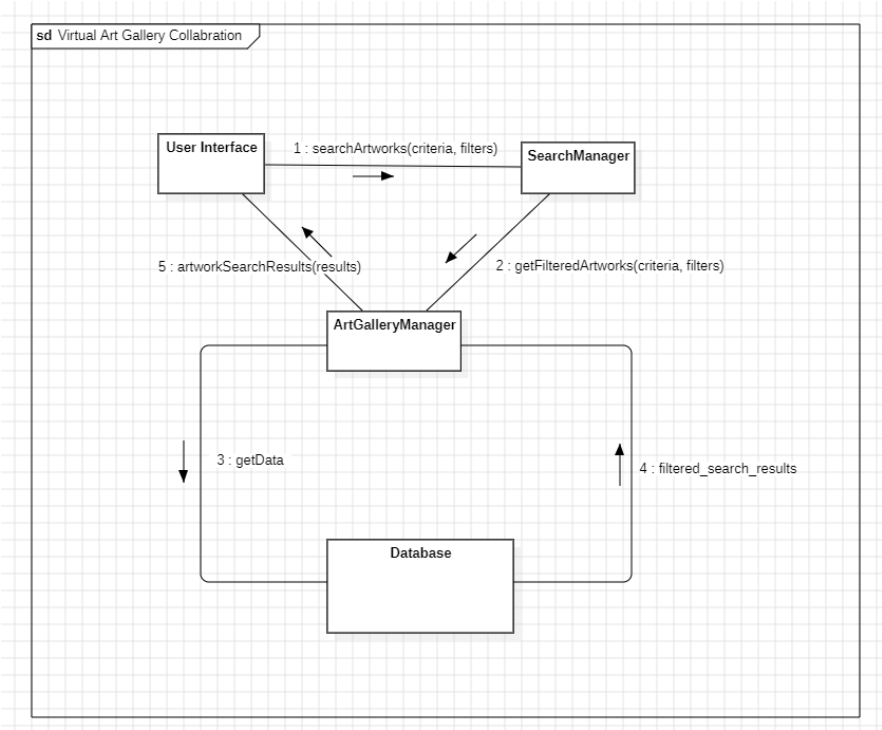


18. Collaboration Diagram

18.1 Hand-Rendered Illustration

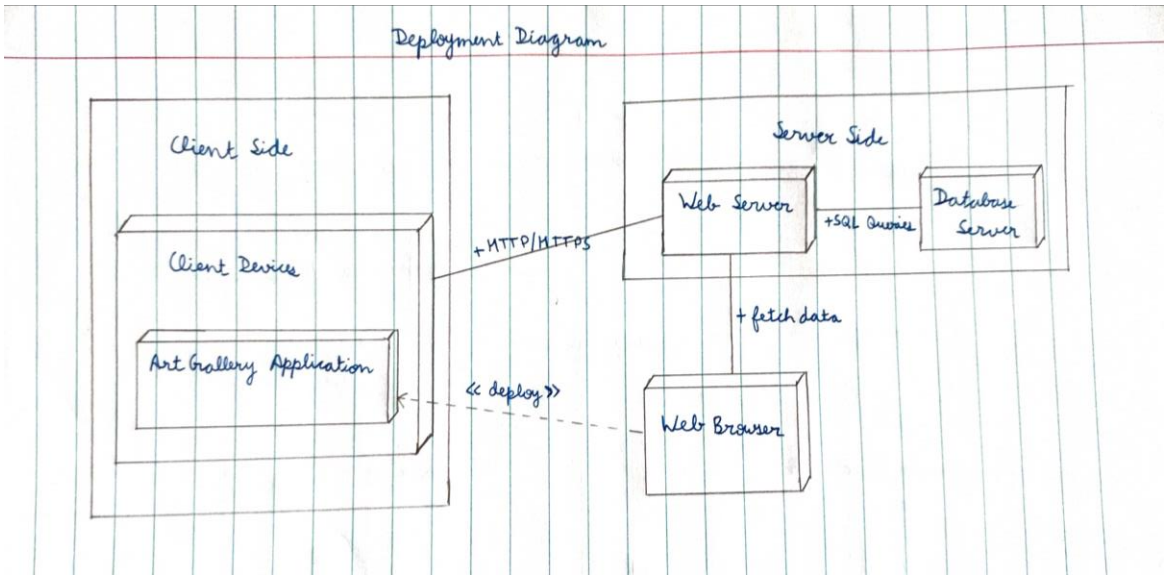


18.2 UML Diagram



19. Deployment Diagram

19.1 Hand-Rendered Illustration



19.2 UML Diagram