Fill in the Void:
Application development for artifact restoration while integrating Artificial Intelligence within Software Engineering practices

Corina Dimitriu, Diana Carcea, Răzvan Carcea, Antonio Iațu

Abstract

In the context of artifact restoration, advanced machine learning generative techniques and user-centered interfaces synergize in delivering a robust and comprehensive solution to society. The prototype described within this paper combines Generative Adversarial Networks (GANs) and Graph Neural Networks (GNNs) for the machine learning component, and mixes up-to-date technology with virtual reality concepts for the human-centered experience. Apart from the pipeline itself, the "meta-layer" of application development introduces some technical and creative lower level processes at the core of building a viable, society oriented product. We prove that, during certain development stages of the application, AI tools might positively interfere, yet there are still passionate debates about whether they help or hinder the transformative process. While artificial intelligence intervenes both as an algorithmic high-level method in the restoration component and as a catalyst for development in the lower level engineering layers, this paper focuses on the latter, being concerned with the blending between the (web) application development process and potential assistive AI tools. The emphasis is placed on intelligent simulations of what was previously known as typical human-in-the-loop engineering processes, such as testing and diagram design. The results are critically discussed from a comparative point of view if related to more traditional, non-AI solutions existing on the market. In terms of real-life use cases of the entire prototype we built, the culture preservation purpose comes first, but work refinement and art sharing could also constitute more subtle achievements.

Keywords: artifact restoration, artificial intelligence, engineering, human-centered.

Problem presentation / Introduction

Starting from the double meaning of the application's name ("Fill in the Void", also being pronounced similarly with "Feel in the Void"), until the functional current version of the application, we addressed artifact restoration as a matter of cross-disciplinary effort. Not only in terms of the technology stack, but also in terms of the practical domains it serves. The base need which gives meaning to the application idea is the preservation of damaged pieces of art. This fits perfectly in the actual context of generative technologies being widely discussed and evolved.

Damaged pieces of art or – more generally – pieces of personal heritage (such as old photos and letters belonging to family ancestors) may owe their weakened condition to historical events (e.g., wars, museum attacks), to a less appropriate storage environment or simply to their age causing inherent degradation. Restoring these artifacts to a more honorable condition might partially provide them with the aesthetics they used to promote and prevent them from further losing their value or message. Furthermore, restoring these artifacts digitally can simplify the access to matching resources and increase their affordability for a large pool of users. Before developing the application, we created personas and scenarios to address a variety of particular usability frameworks and prove the concept.

For instance, there may be worth it to consider Andrew Matthews, a 51-year-old mean who believes that technology is a productive tool to use to achieve all kinds of goals more efficiently. He is picky about the objects he buys or uses, as he wants the highest quality. He is a friendly art and history passionate from whom you can learn, but he also thinks about quality compared to price. Although a proponent of technology, he finds it difficult to understand some of its underlying foundations, so the interface should be intuitive enough. In addition, only having the restored paintings is not enough for him, he would like to be able to visualize the results in a simultaneous and entertaining manner, not to mention the digital backup in terms of storage, as he ended up possessing quite an impressive collection of valuable – both personal and historical –

objects. Therefore, creating his own digital museum would be a suitable option for this aspiration point. Moreover, it's a convenient way to further share his collected artifacts to other art or history addicts all at once. In all these regards, we prove that this paper proposes a relevant (web) application as context for the demonstration of employing artificial intelligence into software engineering processes. This being said, a brief user journey through the application includes the opportunity to submit a damaged photograph for restoration, relying on the premise of being shown a result in short time (currently, about one minute). Then, the inpainted result can contribute to creating an immersive simulation of a museum on the platform, with multiple rooms, such that each one may exhibit a specific niche. Using virtual reality for the touch of realism, the user is able to incrementally place more and more artifacts inside the museum.

The use of artificial intelligence within the software development of the application consisted in successively experimenting with different publicly available AI tools for consolidating or replacing human effort. Since the road map to generalized AI is still uncertain and the functioning of these tools under particular scenarios – software development being one of them – is still perfectible (but so are the more "human" approaches), objective comparisons with non-AI solutions, as well as critical pros and cons discussions, are issued at each main development stage: the research phase, followed by requirements formulation, the architectural design phase, the implementation phase, the verification (i.e., software testing) phase and the maintenance phase. Given the context of an artifact restoration application, we will show that the AI methods we employ for software engineering naturally fit the particular sequence of steps required for building the presented concept. Eventually, we will generalize to other use cases of applications, representing derived or different working fields from art / history / humanities.

On the challenges that arise at the same time with the evolving problem-solving capacity of artificial intelligence, we insist on highlighting that human intervention is still critical for modelling in detail the desired concept. Particularities have to be precisely requested, usually within follow-up (i.e., refinement) prompts. However, since there are roughly inevitable components of the interface (e.g., the CRUD operations) which fall back to general templates, already optimized and widely accepted, AI tools majorly enhance productivity. Repetitive practices are, after all, one of the salient reasons behind the invention and adoption of such tools.

Chapter 1

State-of-the-Art

1.1 Application component

Our application combines two essential functionalities: automatic restoration of artifacts from images and their visualization in an interactive 3D space. Unlike most existing applications that focus either on image restoration or 3D visualization, our application integrates both processes into a unified workflow. This allows users to restore and explore artifacts continuously and interactively in a 3D space.

In Chapter 5, Section "Similar Applications", we will analyze several applications that implement only a part of the functionalities offered by our application. Apps such as Pincel, DALL-E, and Inpaint focus on automatic image restoration, each offering features for enhancing and repairing images, but without including interactive 3D visualization. On the other hand, platforms like Google Arts & Culture, Artivive, and Artsteps offer immersive 3D experiences but do not integrate image restoration processes. Our project brings an important innovation by combining both capabilities, providing a complete tool that covers both restoration and visualization, significantly changing the way users can interact with restored artifacts.

1.2 Machine Learning component

As the artificial intelligence mechanisms behind the restoration component do not constitute the primary objective of this paper, we briefly synthesize only the most influential contributions with respect to the chosen solution. In scientific literature, the task of generating content to fill in

masked / missing picture regions is called semantic image inpainting.

The solution described in [6] has the advantage of accounting for the completion of large portions of the image. Various shapes of the missing portions are treated via embodying contextual losses, assigning higher weights to the regions surrounding missing pixels. Generative Adversarial Networks are currently designated as the State-of-the-Art method to constrain the generated images to lie on the same manifold as the authentic ones (i.e., be part of the same distribution), by competitively training a generator and a discriminator network with the scope of stimulating each other's resulting quality. This system is proved to improve the image quality in terms of FID. If the hardware system supports a more intensive training process, attention mechanisms and transformers can provide the generator with more in-detail context, since the integration of convolutions usually tends to capture local features. Using priors (pre-trained models) as an additional layer to the GAN network, has the effect of (again) enhancing the relevance of the context, but from a higher level (global) perspective.

On the premise of keeping the generative adversarial framework while also making use of graph neural networks for incremental learning, we adopted the coarser to finer approach described in [7]. Unlike more conventional methods, the GraphFill solution models the intricate relationships between image patches as nodes in a graph, capturing both local and global contextual information. The result is thoroughly refined through a series of image-to-graph and graph-to-image iterations.

Chapter 2

Our solution

2.1 Application component

Our application, "Fill in the Void," aims to bring deteriorated paintings and images back to life, whether they have been damaged by time or other external factors. The primary goal is to support users in restoring these artworks and filling the gaps within them using cutting-edge technologies. The restoration process leverages the GraphFill machine learning algorithm, which enables precise and realistic completion of missing areas. With this technology, users can revive the original brilliance and mastery of these creations, preserving their memory for future generations.

To enhance the user experience, we offer the option to create an account. This allows users to save their restored paintings and access all the application's features.

Once logged in, users can revisit their restored paintings at any time. As their collection grows, they have the option to organize their works into personalized albums. Albums enable users to classify their paintings based on their own criteria, while also allowing them to add titles and descriptions, making each album unique and easy to identify.

To add a special touch to the experience, we've introduced an innovative feature: transforming albums into virtual art galleries. Using TreeFiber technology, we provide users with an immersive environment where they can explore their restored works displayed on the walls of a virtual museum. This functionality, focused on the front-end, is designed to deliver a captivating experience, offering users the feeling of stepping into a real museum and connecting with their restored works like never before.

To provide a more detailed understanding of the application, we have utilized several types of

diagrams. These diagrams cover both the logical flows and user activities, as well as the technical architecture and data structure of the application. The presented diagrams illustrate various aspects of the application, helping to clarify functionalities, component relationships, and system behavior.

2.1.1 Application Flow and User Activity

The flow diagram outlines the steps the user follows in the image restoration process, starting from uploading the image and ending with saving it in a gallery.

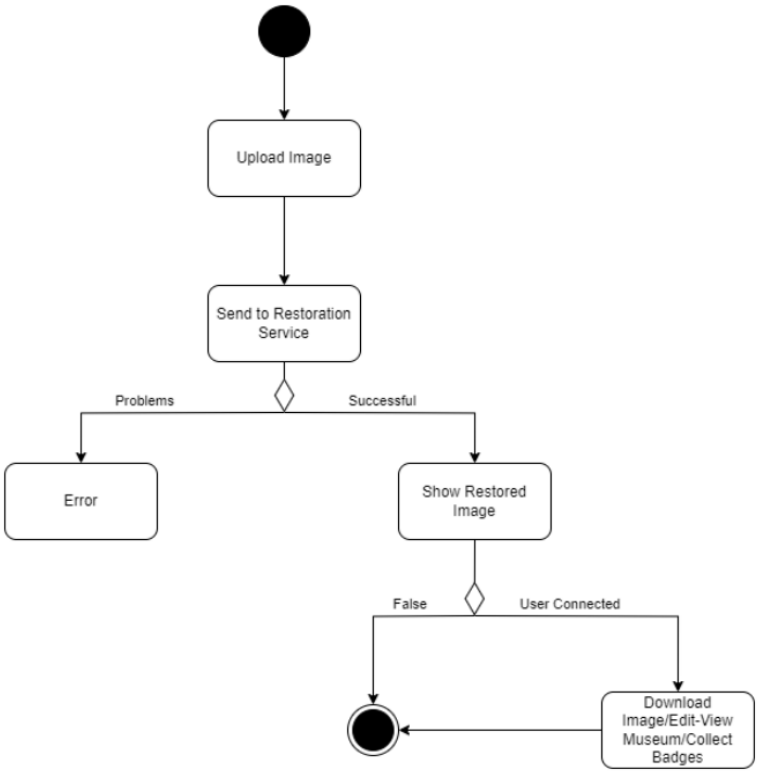


Figure 2.1: Flow Diagram

The use case diagram highlights the main interactions between the user and the application, as well as the available functionalities.

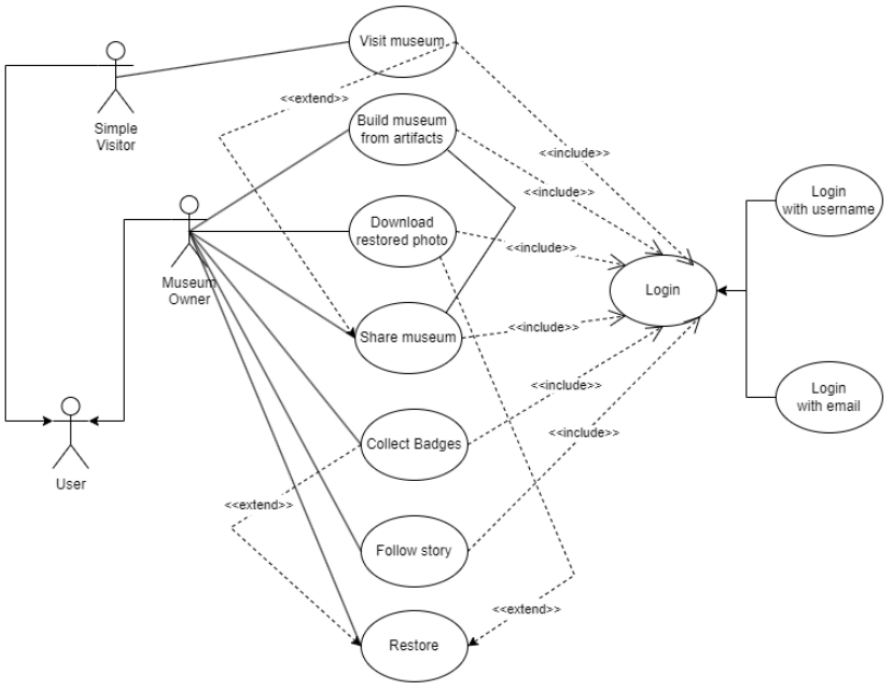


Figure 2.2: UseCase Diagram

2.1.2 Application Architecture

The C4 diagram provides a clear view of the application architecture at all levels – context, containers, components, and code structure. It helps to understand how the utilized technologies are integrated.

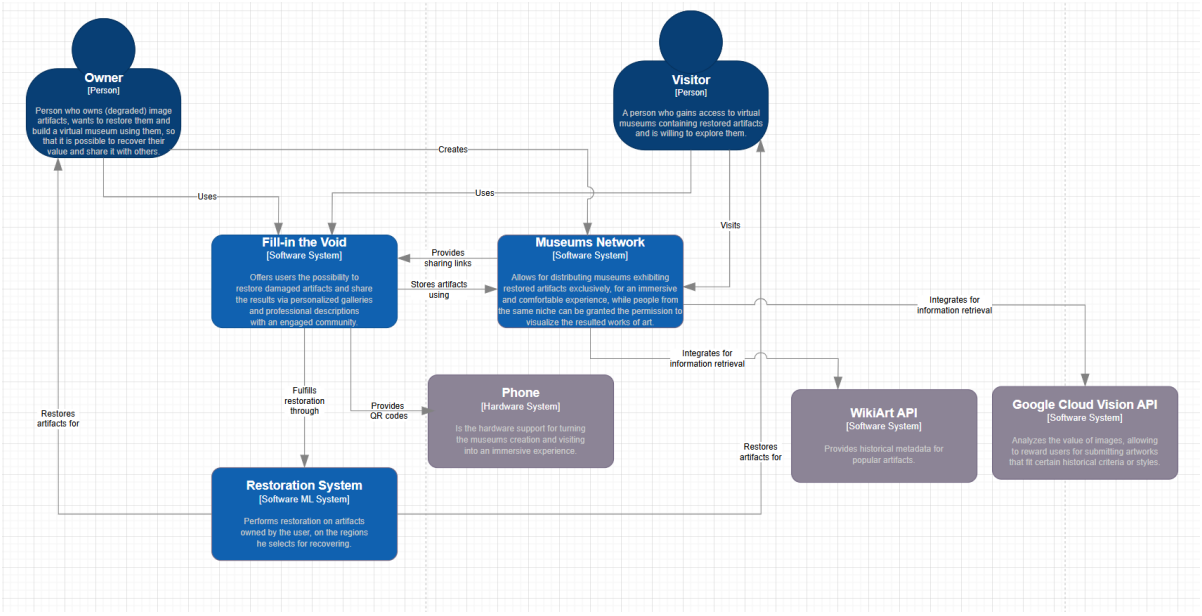


Figure 2.3: Flow Diagram

The package diagram illustrates the application's structure at the module level, showing how functionalities are distributed among different components.

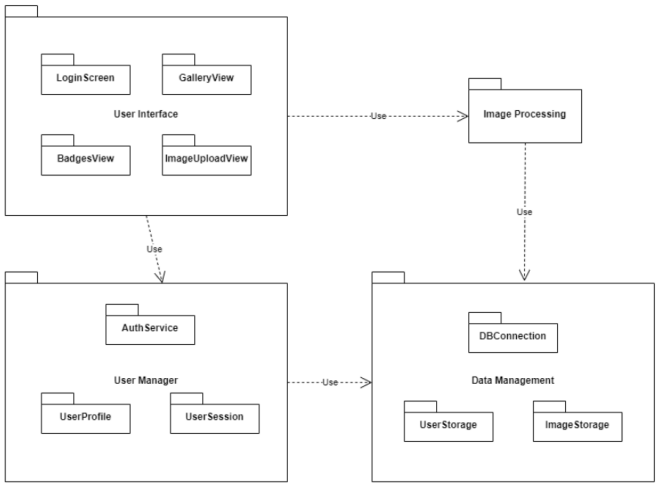


Figure 2.4: Package Diagram

2.1.3 Application Behavior

The sequence diagram demonstrates how the user interacts with the application and how their actions are managed in the backend, for example, during the image restoration process.

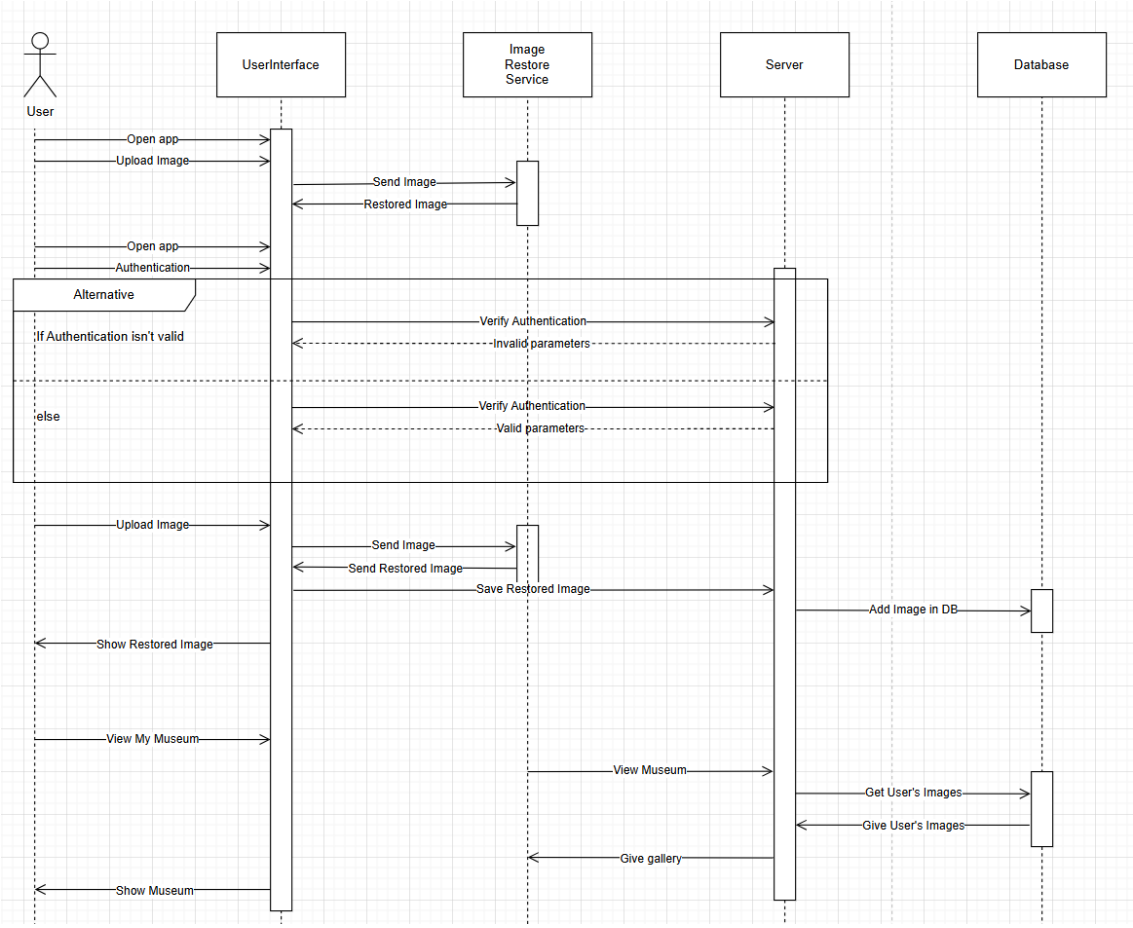


Figure 2.5: Sequence Diagram

The collaboration diagram highlights the relationships between the application objects involved in the restoration and image-saving processes.

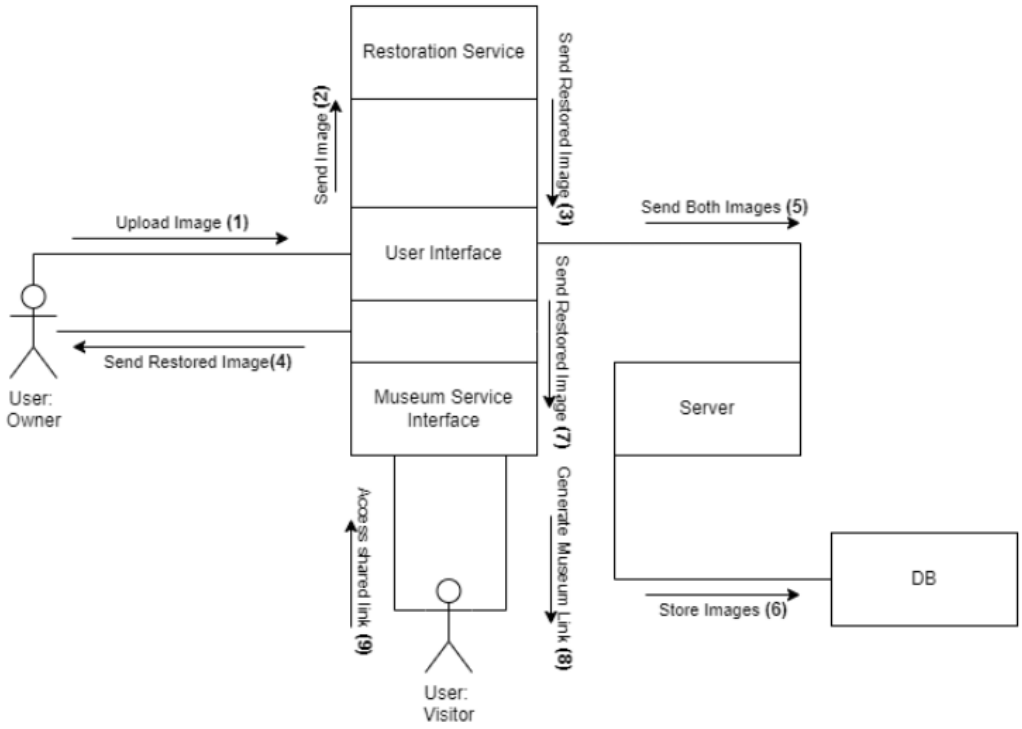


Figure 2.6: Collaboration Diagram

2.1.4 Application State and Data Structure

The state diagram illustrates the different states the application goes through during the image restoration and saving process.

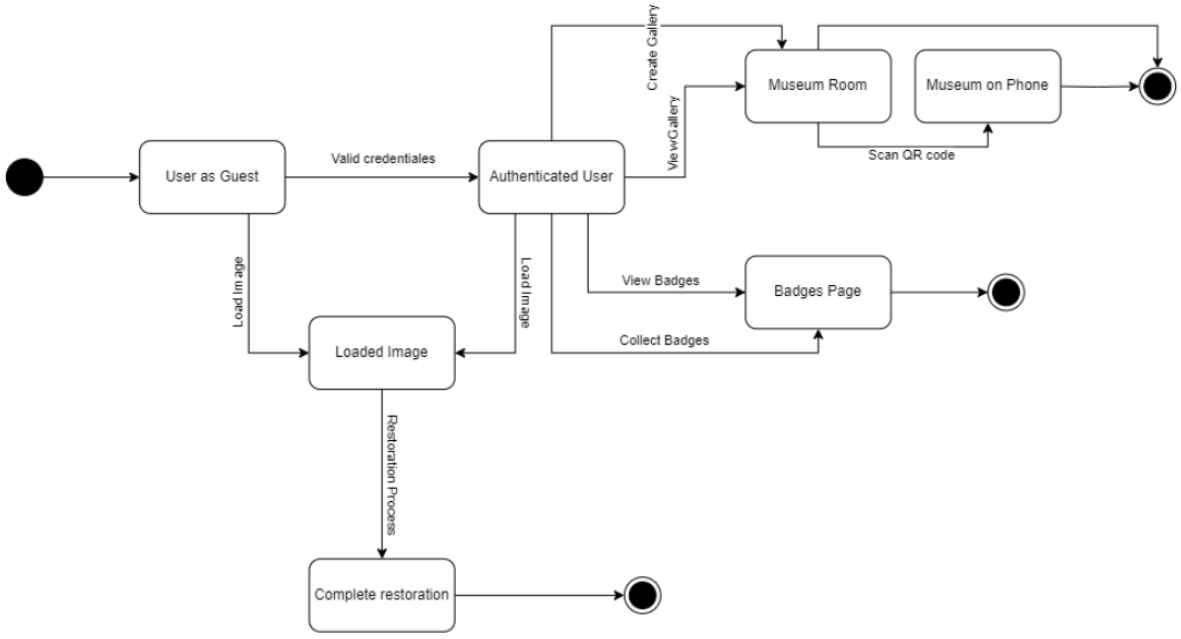


Figure 2.7: State Diagram

The class diagram shows the structure of the classes used in the application and the relationships between them, providing a clear perspective on the software design.

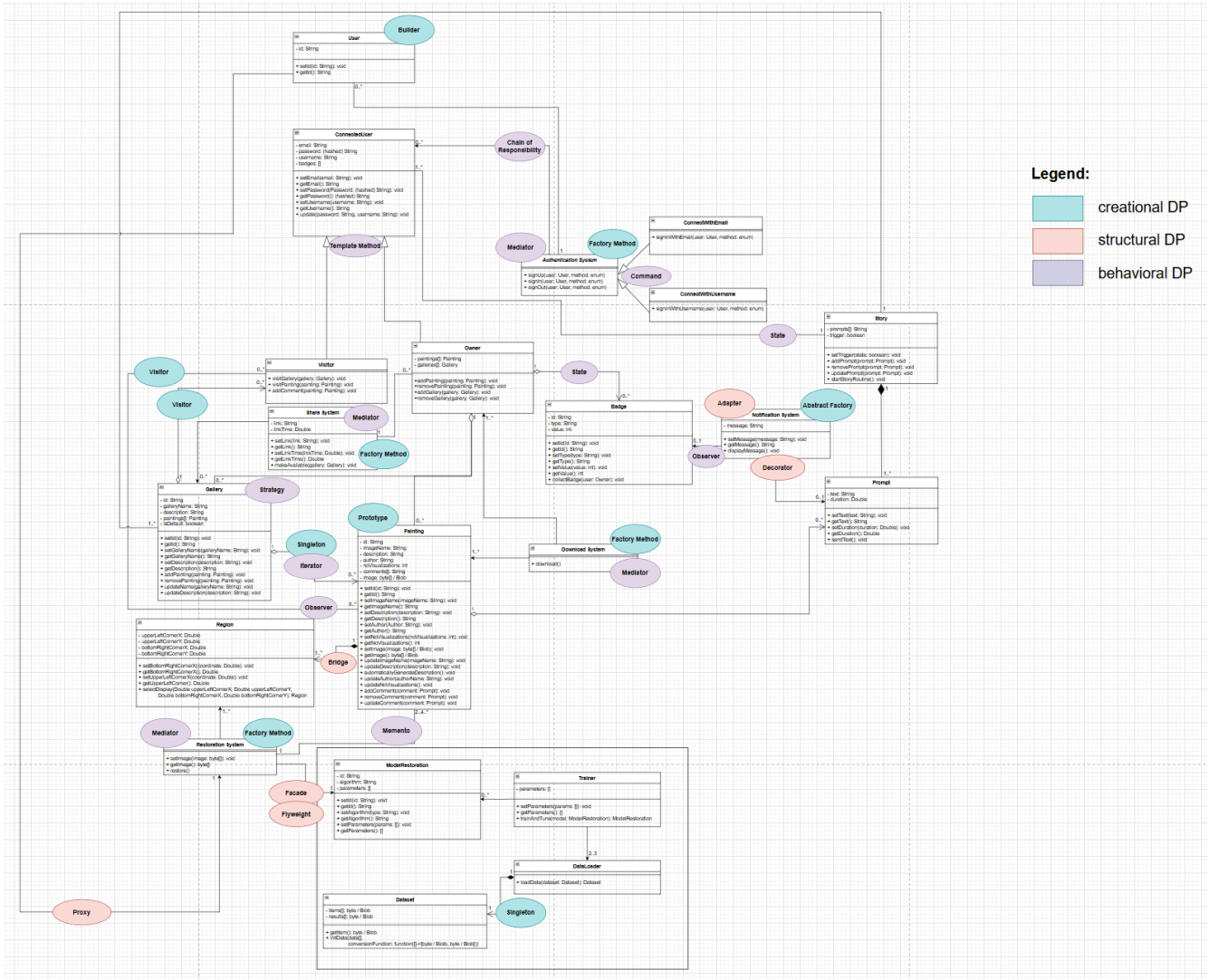


Figure 2.8: Class Diagram

The presented diagrams provide a detailed and integrated view of the application, highlighting both the user flows and the technical structure and architecture of the system. They emphasize how the application's functionalities are interconnected and implemented to deliver a complete user experience.

2.2 Machine Learning component

As proof-of-concept, our application focuses on the restoration of paintings. The simpler solution which preceeded the GraphFill architecture (1.2) consisted in a cycle generative adversarial network (GAN). The network is trained on a custom dataset pairing original paintings with batches

of counterparts varying the contained regions which are masked.

2.3 AI-driven software engineering component

In the project, we used AI tools to optimize various stages of development. Among these, we used:

LucidChart - for the automatic generation of class diagrams from text descriptions, enabling the rapid creation of structural diagrams based on the provided specifications.

DiagramGPT - for the automatic extraction of class diagrams from source code, ensuring a direct correlation between the actual code implementation and its visual representation.

Codeium - for automatic code generation, particularly for creating unit tests. This tool helped accelerate the test development process and reduced the time required for manual test writing.

StarUML - to automate the generation of code from UML diagrams. This tool also supported us in implementing Model-Driven Development, facilitating the automatic transformation of models into executable code and ensuring coherence between the initial design and the actual implementation.

Chapter 3

Results, Evaluation

3.1 Application component

The application starts with the main page, which marks the beginning of the image restoration process – the core purpose of the application. This page is accessible even to users who are not logged in and allows the image to be uploaded. The image can be added either by clicking on the area outlined by the dashed line or by dragging and dropping the file directly into that area. Once the image is uploaded, the user can proceed to the next step.

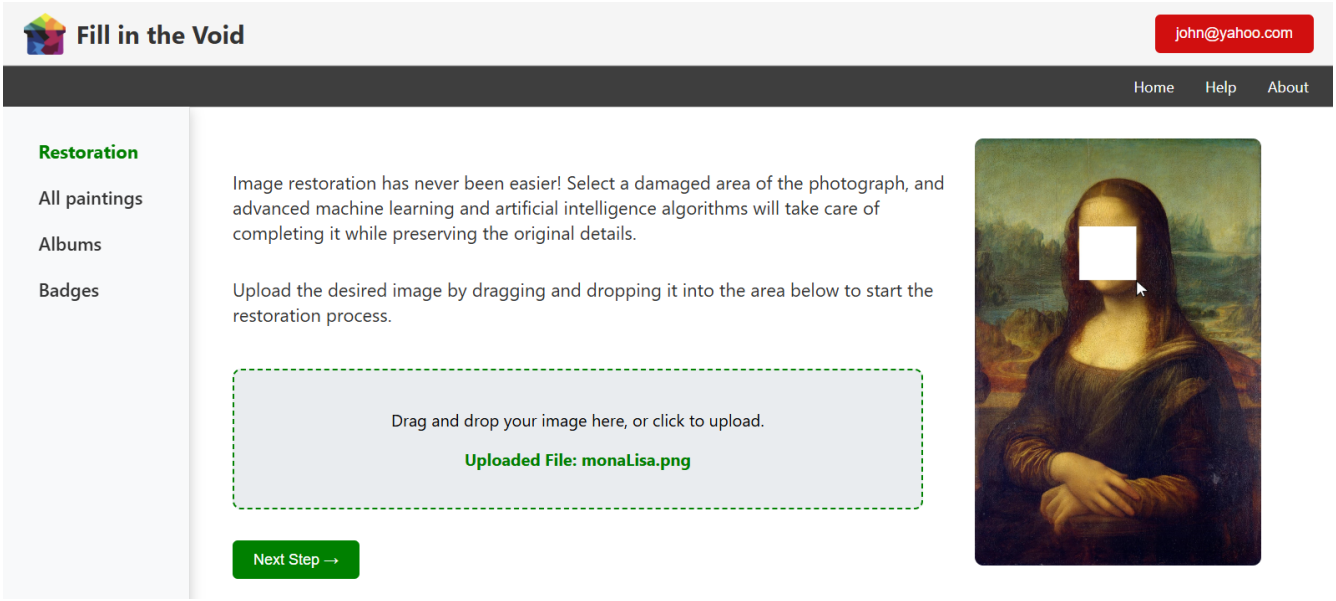


Figure 3.1: Main Page

After the user clicks the "Next Step" button, they will be redirected to a page where they need to select the area(s) of interest in the image they wish to restore. On this page, the user has access to a tool that allows them to adjust the brush size and another tool for zooming in on the image, enabling greater precision when selecting the desired area.

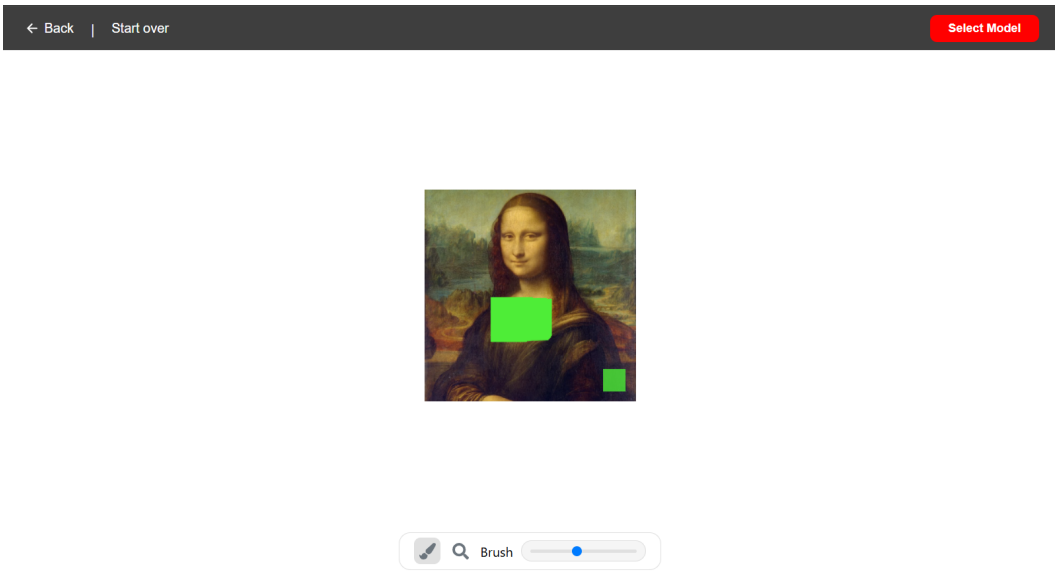


Figure 3.2: Restore page

To start the image restoration process, the user must select a portion of the image and also click the "Select Model" button to choose the image style: "Contemporary" or "Renaissance." This selection helps the machine learning algorithm determine the most suitable method for restoration.

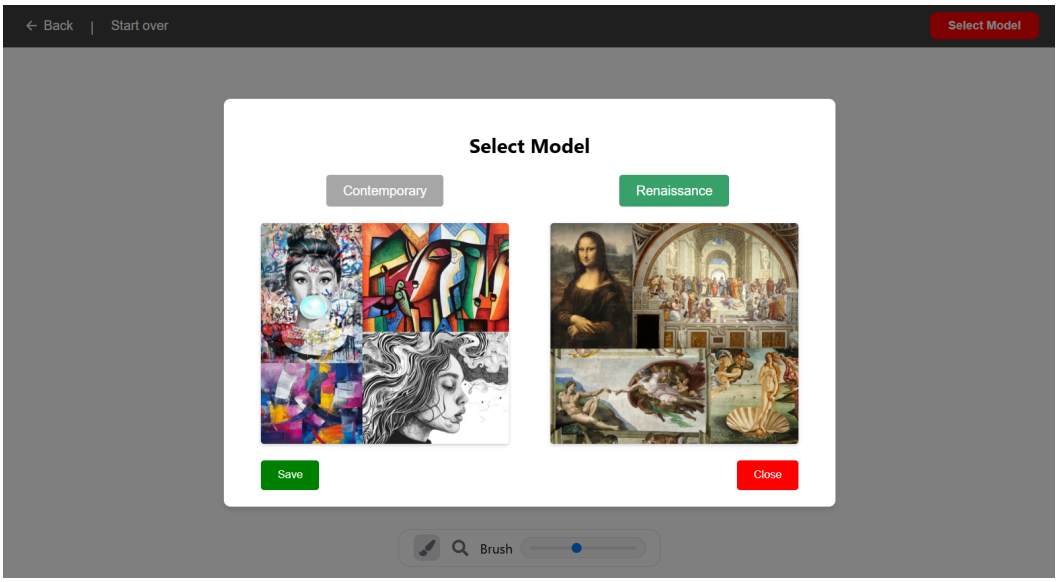


Figure 3.3: Select Model

Once all the requirements are met, the "Done" button will appear. When clicked, it will trigger the restoration process. Upon completion of the process, the user will be redirected to the results page. On this page, the original image, before restoration, will be displayed, with the restored image shown on the right side.

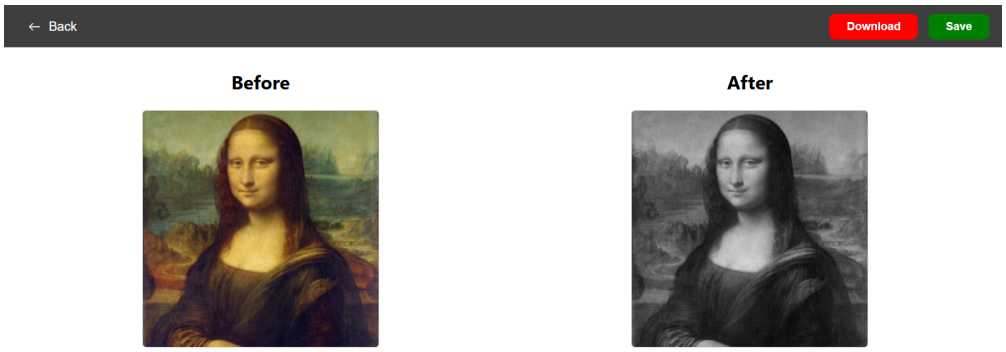


Figure 3.4: Result Page

On this page, there are two buttons: one "Download," which allows the user to download the restored image, and one "Save," which saves the image within the application for later use. The "Save" button functionality is available only to users who are logged into the application.

For registration, a first name, last name, email, and password are required, while for login, only the email and password are needed. The two forms are structured as follows:

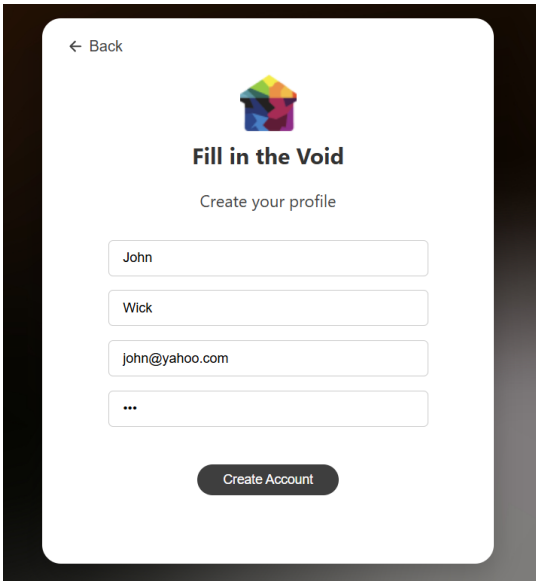


Figure 3.5: Account registration

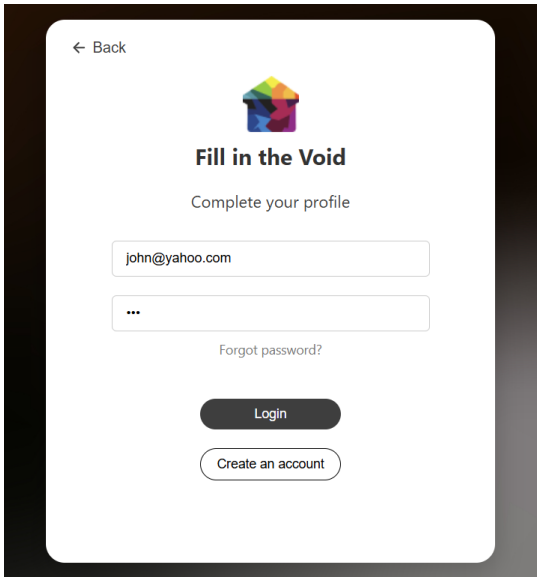


Figure 3.6: Account login

After clicking the "Save" button on the restoration results page shown in Figura 3.4 , the user will be prompted to enter a title for the image and, optionally, a description and an author. If the description and author are omitted, default values will be set automatically.

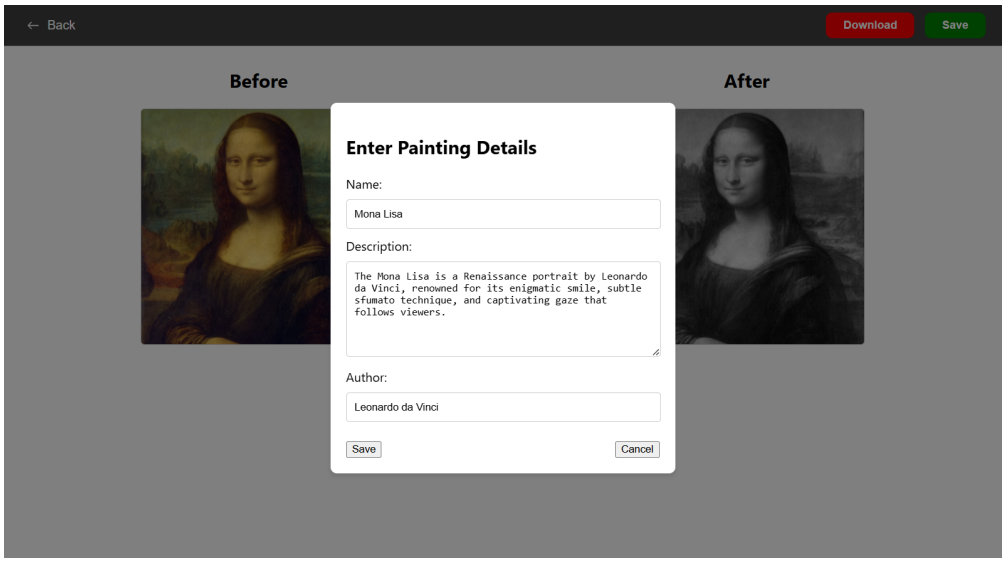


Figure 3.7: Save image in App

Once the image is saved, the user will be redirected to the paintings page, where they can view all the paintings saved in the application.

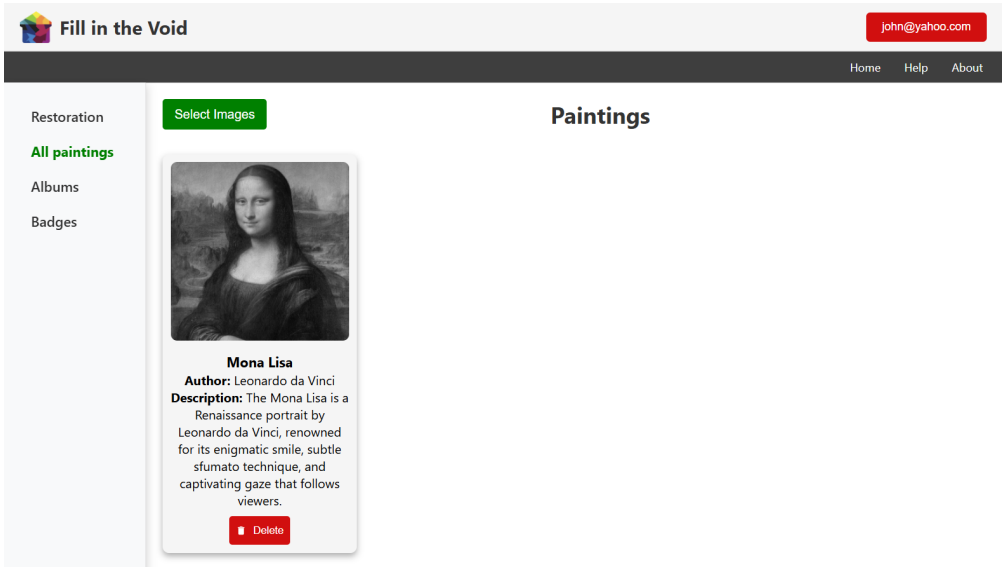


Figure 3.8: Page of paintings

The user can also edit an image by simply clicking on it.

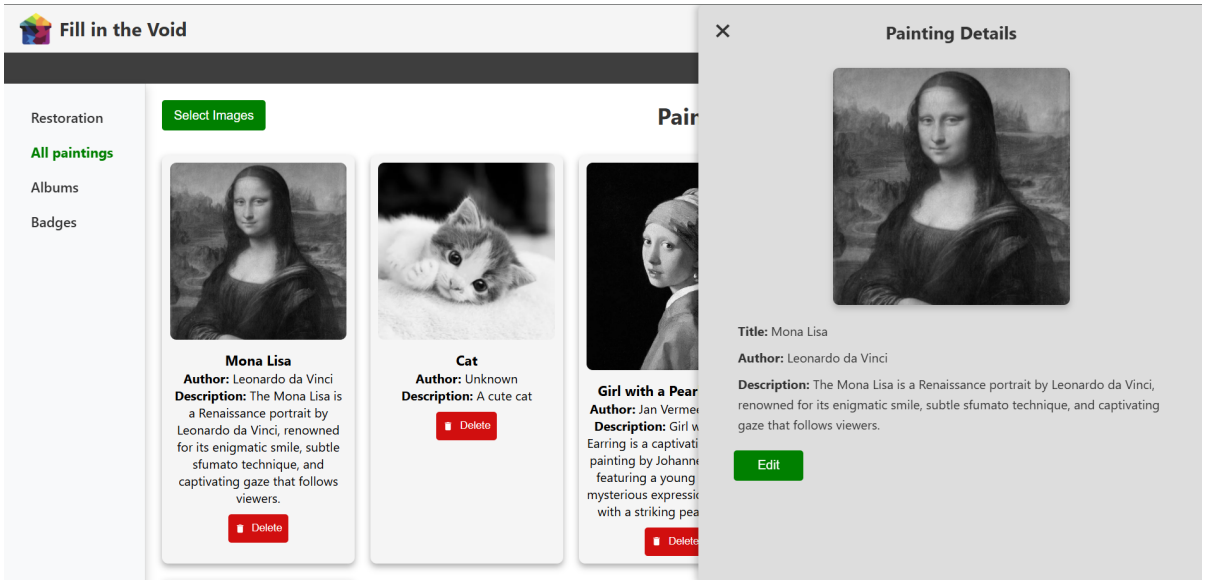


Figure 3.9: Edit paint

On this page, there is also a "Select Images" button, which allows the user to select multiple images to add to an album.

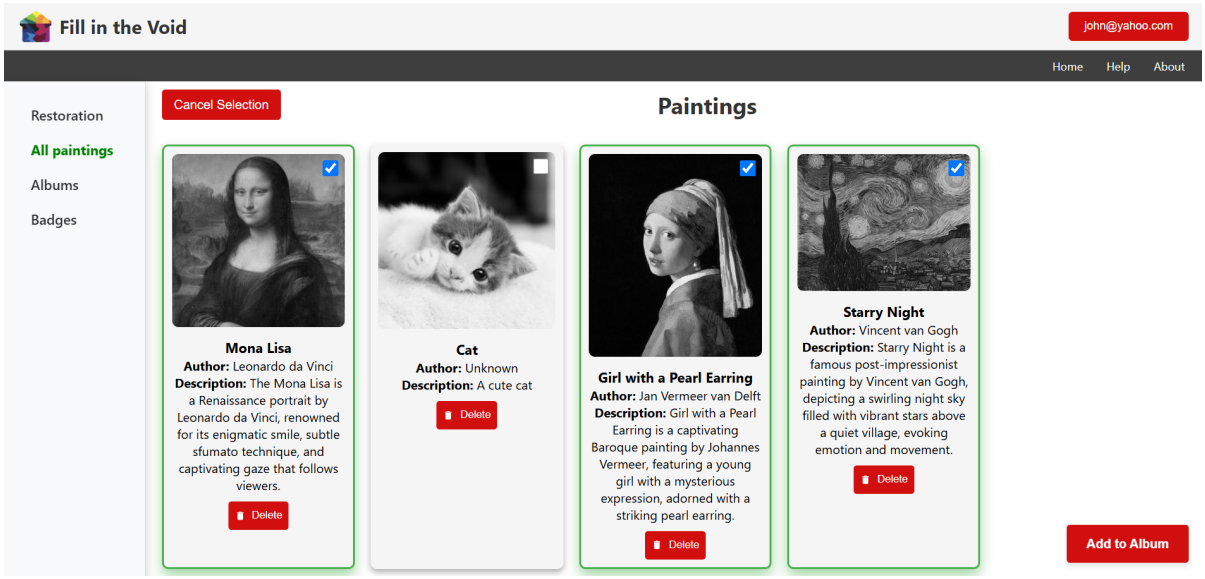


Figure 3.10: Select paintings

Once at least one image is selected, the "Add to Album" button appears at the bottom of the page. Clicking this button opens a side panel on the right side of the page, where all the albums created up to that point are displayed. Within the panel, the user also has the option to create a new album.

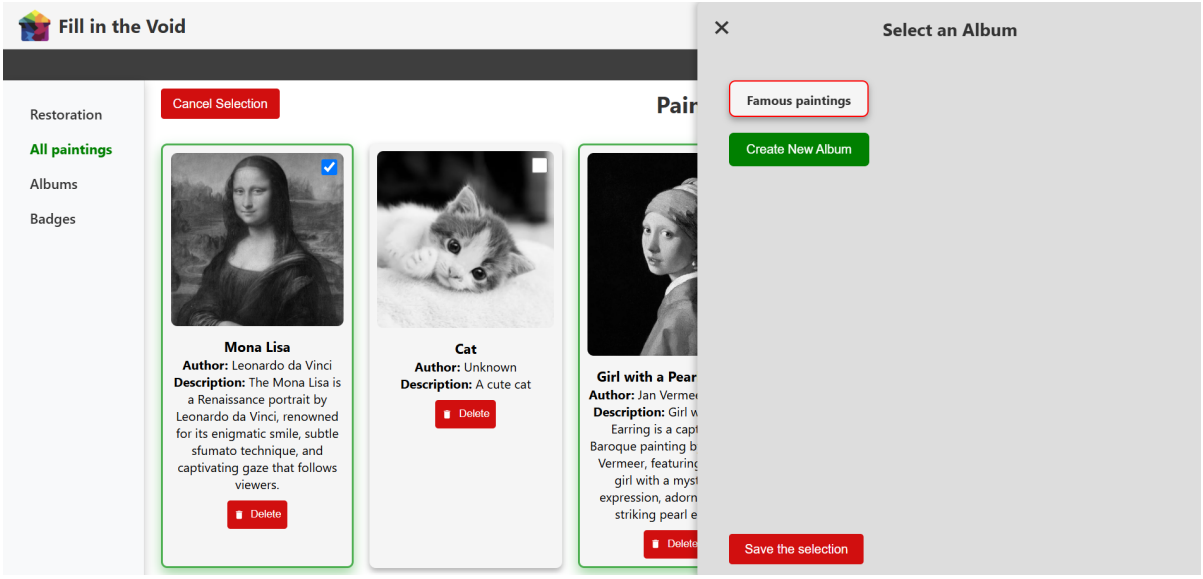


Figure 3.11: Select album

Once an album is selected, the user will be redirected to the album page, which will include the selected images along with any other paintings previously added.

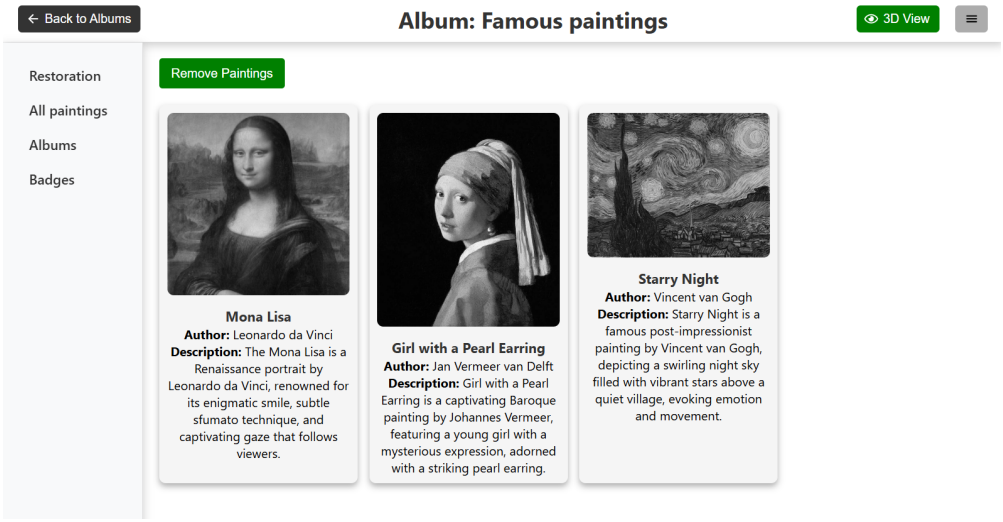


Figure 3.12: Album content

Paintings can also be viewed in a more immersive way. By clicking the "3D View" button, the user is placed in a virtual room where they can admire the paintings from the album displayed on the walls, similar to an art gallery. The user can move around using the W, A, S, D keys and adjust their perspective by rotating the mouse. Additionally, we can move to other rooms of the museum by clicking on one of the doors, thus exploring different areas and thematic spaces.

In the top-right corner, there is an "Actions" option, which, when accessed, provides the user

with a set of features: adding paintings, rearranging them, removing paintings, and sharing them.

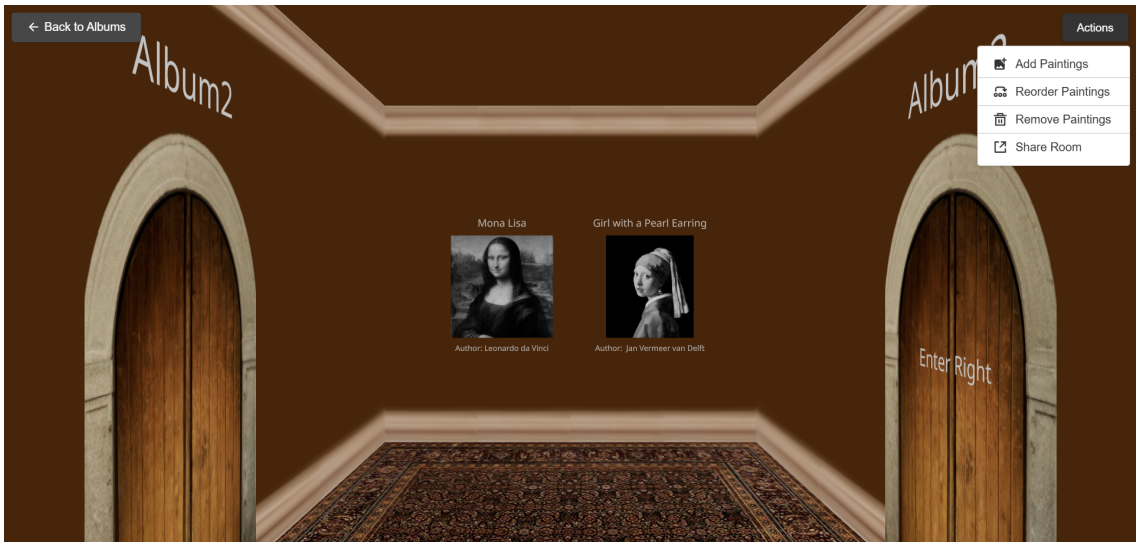


Figure 3.13: Room with paintings

Adding paintings can be done either by selecting the desired ones and clicking the "Add Selected Paintings" button or by simply dragging the paintings directly into the room. An interesting feature is that when the mouse cursor moves out of the painting addition panel, the user can still navigate through the room, allowing them to see in real-time how the paintings are arranged on the walls.

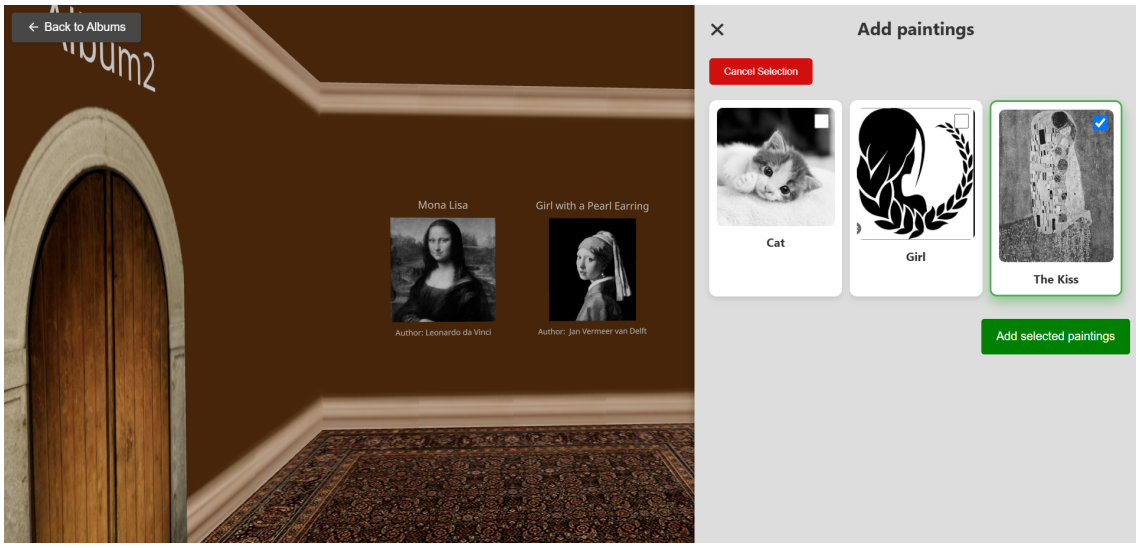


Figure 3.14: Adding paintings to the room

Removing paintings is done by accessing the "Remove Paintings" option. In this mode, each painting will display a trash icon in the top-right corner, and clicking it will instantly remove the

respective painting from the room. To exit the painting removal mode, the user can click the "Close painting removal mode" button.

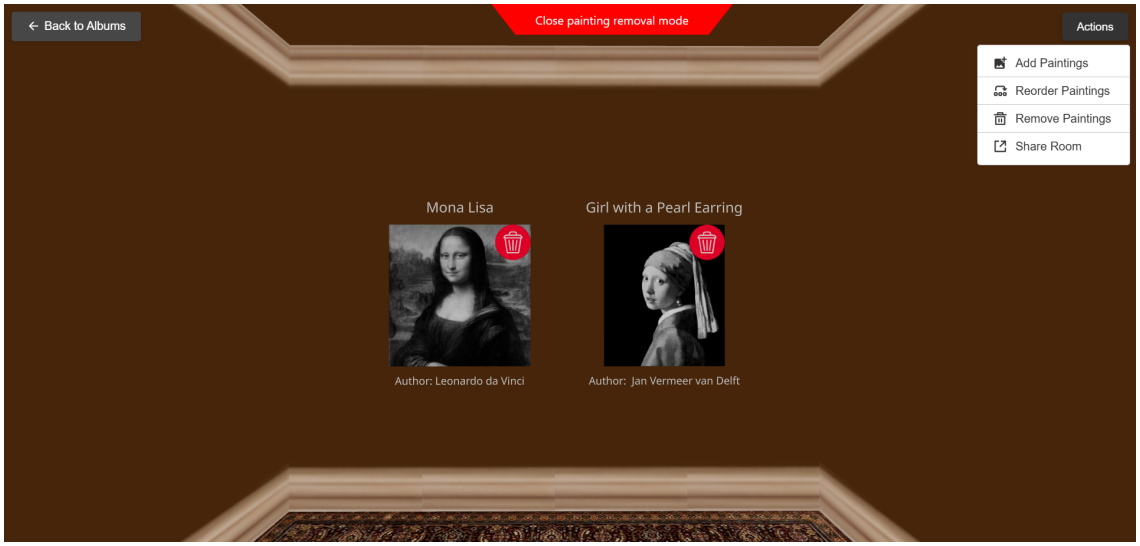


Figure 3.15: Removal of paintings in the room

If we want to view the description of a painting, we can click on "Read More," which will appear when hovering over the respective painting, as seen with the painting on the right. Once clicked, the description will appear on a transparent background, as shown with the painting on the left.

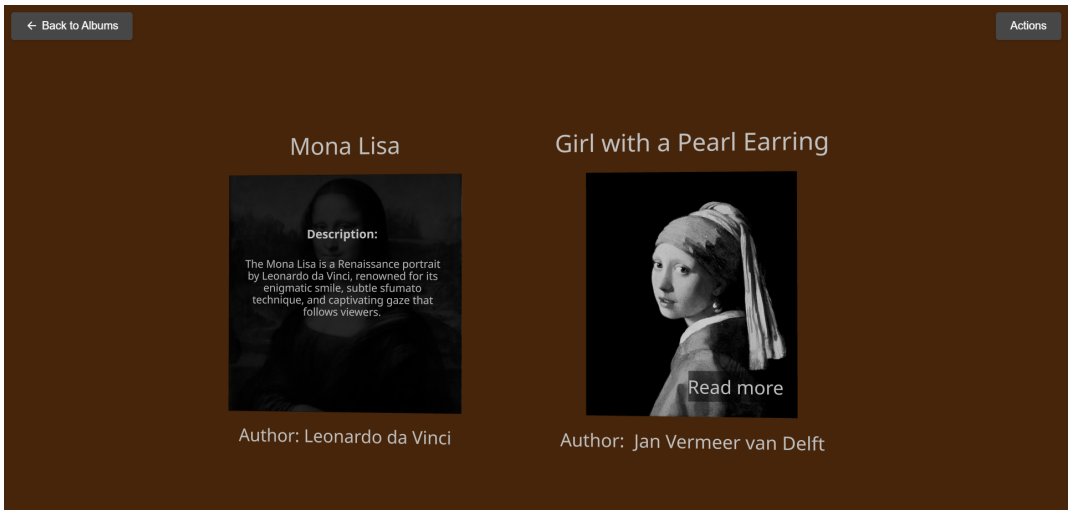


Figure 3.16: Description of the painting

If we want to exit this view mode and return to the page with all albums, we can click the "Back to Albums" button in the top-left corner. On this page, the user can create as many albums as desired, giving them a name and an appropriate description. The albums can later be edited if needed.

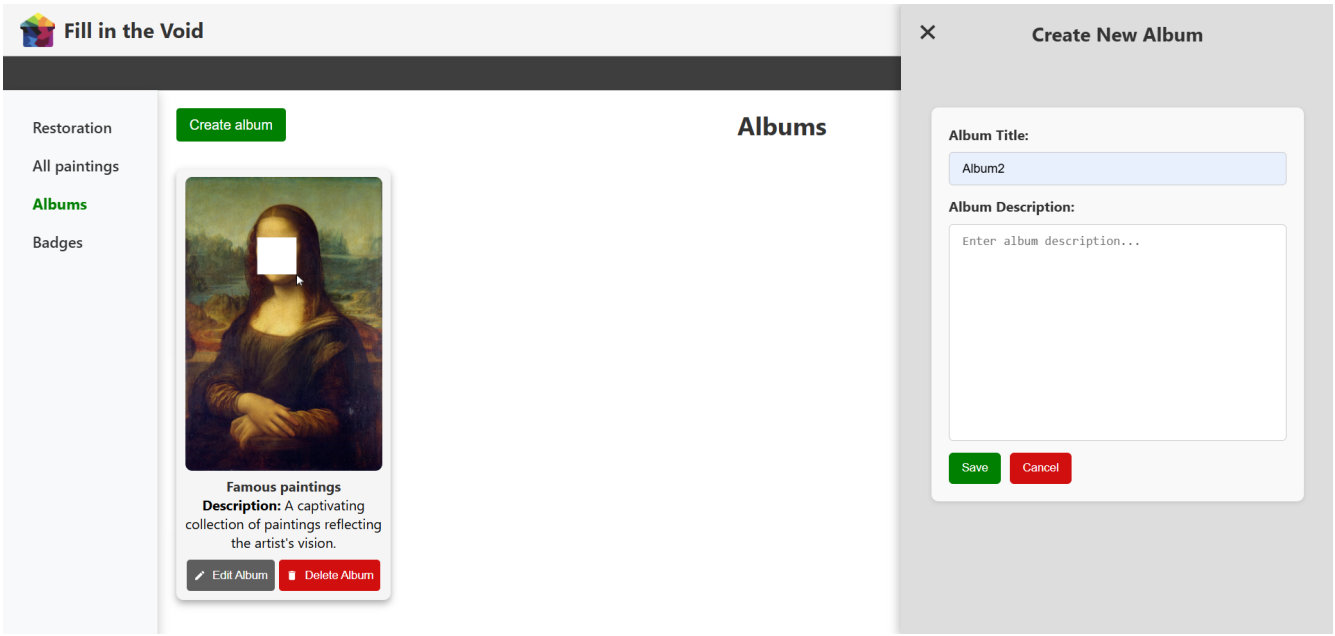


Figure 3.17: Create New Album

3.2 Machine Learning component

Figure 3.18 depicts the result of applying the first ML solution (GAN-based). As observed from previous figures, incorporated within the final application's flow description, there is a significant improvement when it comes to the second solution (Graph-based) employed for this component.

3.3 AI-driven software engineering component

In the project, we used AI-based tools to streamline the process of creating class diagrams, a crucial element in software architecture design. Two distinct approaches were utilized:

1. **Generating Class Diagrams from Text Using *LucidChart*** This tool allowed us to input specifications in natural or semi-formal language, and it automatically converted the

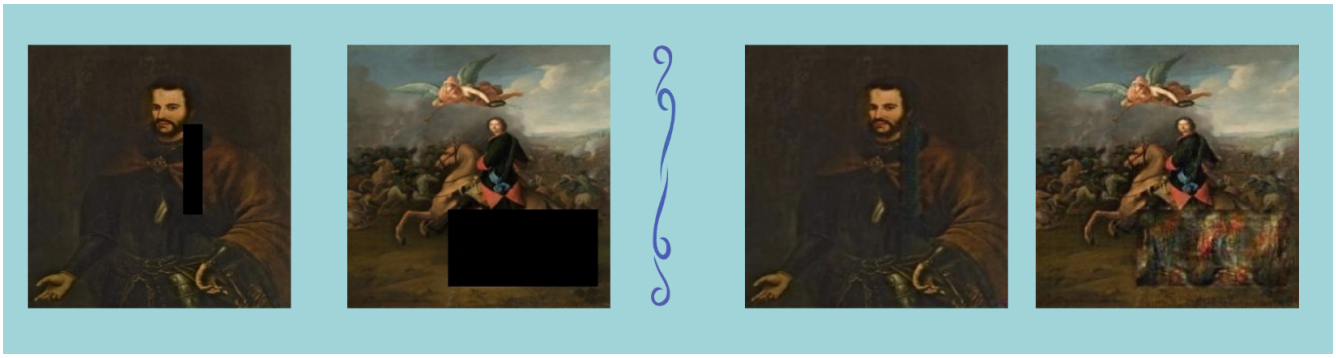


Figure 3.18: Results of applying the GAN-based pipeline

descriptions into well-defined structural diagrams. The application of this tool revealed several advantages and disadvantages.

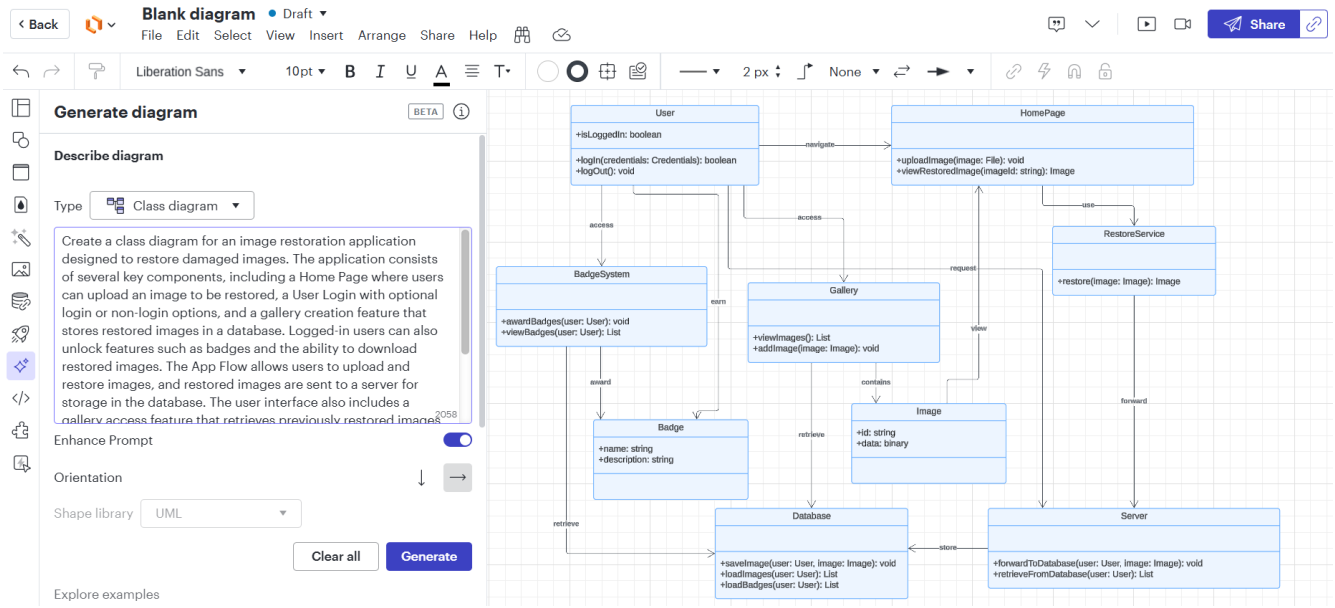


Figure 3.19: LucidChart - Class Diagram from Text

Advantages:

- **Fast Diagram Generation:** LucidChart enables rapid creation of diagrams based on simple textual descriptions. This process significantly saves time and provides users with a starting point for organizing complex structures.
- **Starting Point for Complex Diagrams:** The tool is particularly useful in the early stages of a project when the team does not yet have a clear vision of the final structure. It offers a logical base that can later be adjusted and enriched with additional details.
- **Practical Examples:** Automatic diagram generation can offer new perspectives and help users visualize and analyze various problems through a clear graphical representation.
- **Theoretical Refresher:** The tool is ideal for scenarios where users may no longer be familiar with theoretical details related to class diagrams, such as component relationships, the correct use of associations, or the essential elements of class boxes.

Disadvantages:

- **Misinterpretation of Descriptions:** There are cases where LucidChart does not accurately interpret the user's intentions, especially if the textual description is not sufficiently precise or complex. Adding clear details is crucial for obtaining relevant results.
- **Lack of Relationship Diversity:** Automatically generated diagrams tend to predominantly use association relationships, with other types, such as aggregation or composition, rarely included automatically.
- **Inconsistency in Diagram Generation:** The same text can yield different results, which may confuse users. This lack of consistency can reduce confidence in the tool.
- **Complexity in Understanding the Diagram:** In some cases, the automatically generated diagrams can become difficult to understand, especially if they include elements or relationships that do not align with what the user intended to represent.

2. **Generating Class Diagrams from Code Using *DiagramGPT***

Another method we employed was the use of DiagramGPT, a tool that automatically extracted the structure of class diagrams from source code. Through this instrument, we were able to generate diagrams based on actual implementation, ensuring a direct correlation between the written code and its visual representation. This was particularly useful for

documenting and validating the software architecture without requiring additional effort to manually update diagrams.

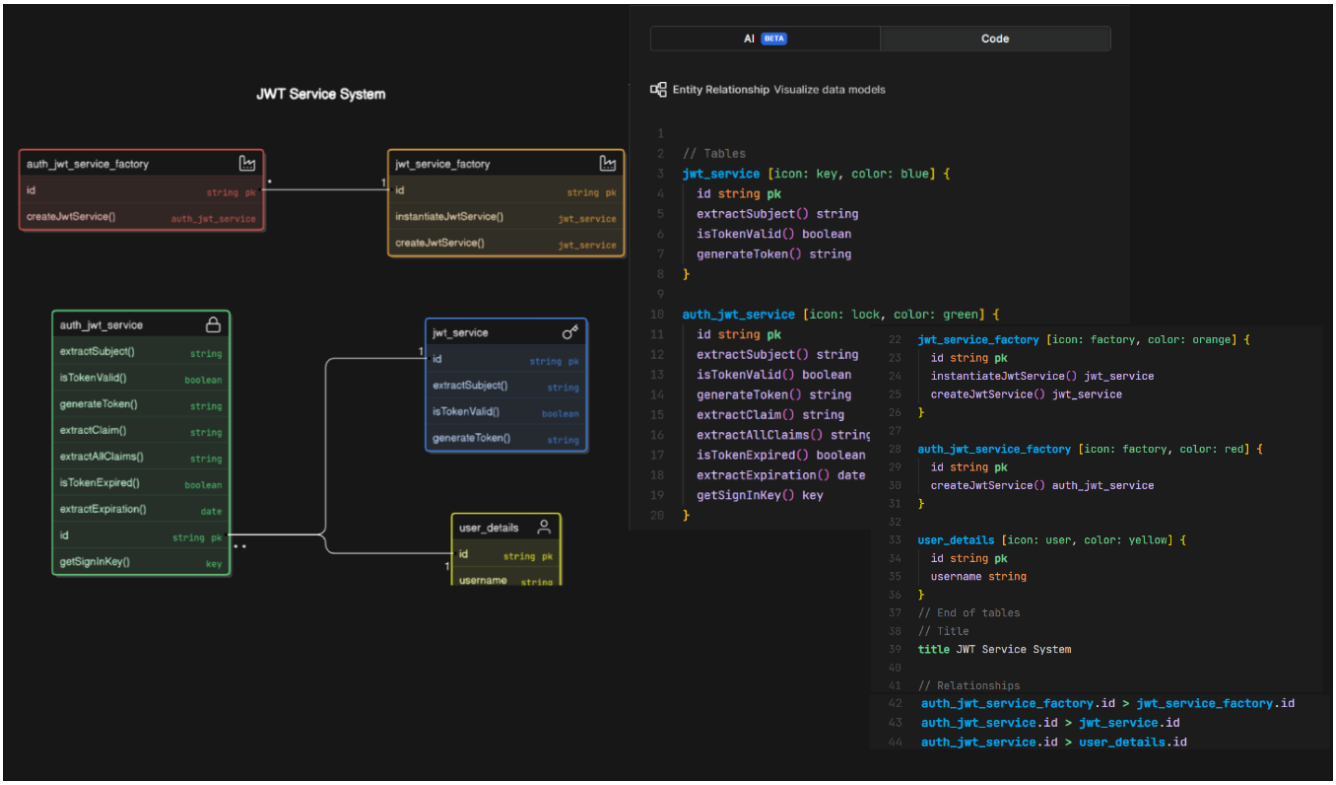


Figure 3.20: DiagramGPT - Class Diagram from Code

DiagramGPT operates on a similar principle to **IntelliJ IDEA**, as both are valuable tools for generating class diagrams. However, each offers distinct functionalities and caters to different needs.

DiagramGPT is better suited for rapidly generating diagrams when working with small code segments or smaller projects. The tool identifies basic relationships such as inheritance, interface implementation, and class associations but omits critical details like access modifiers (public, private, protected). As a result, some attributes or methods may be missing from the generated diagrams, limiting its utility for more complex projects. DiagramGPT is ideal in scenarios where speed and simplicity are the priority, especially in early design phases or for minimal documentation.

On the other hand, **IntelliJ IDEA** excels in its deep integration with complex software projects. Diagrams generated by IntelliJ are dynamic and automatically update as the source code changes, providing an accurate, real-time representation of the project architecture.

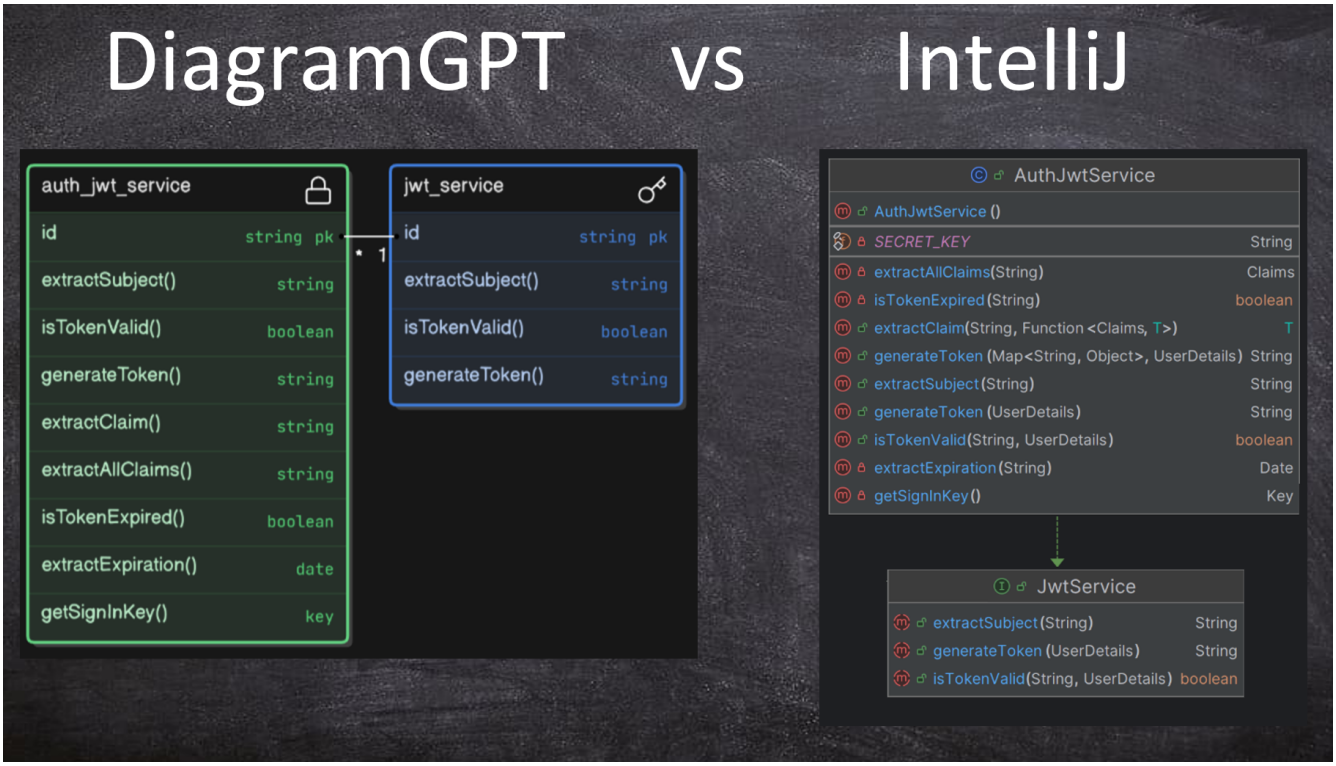


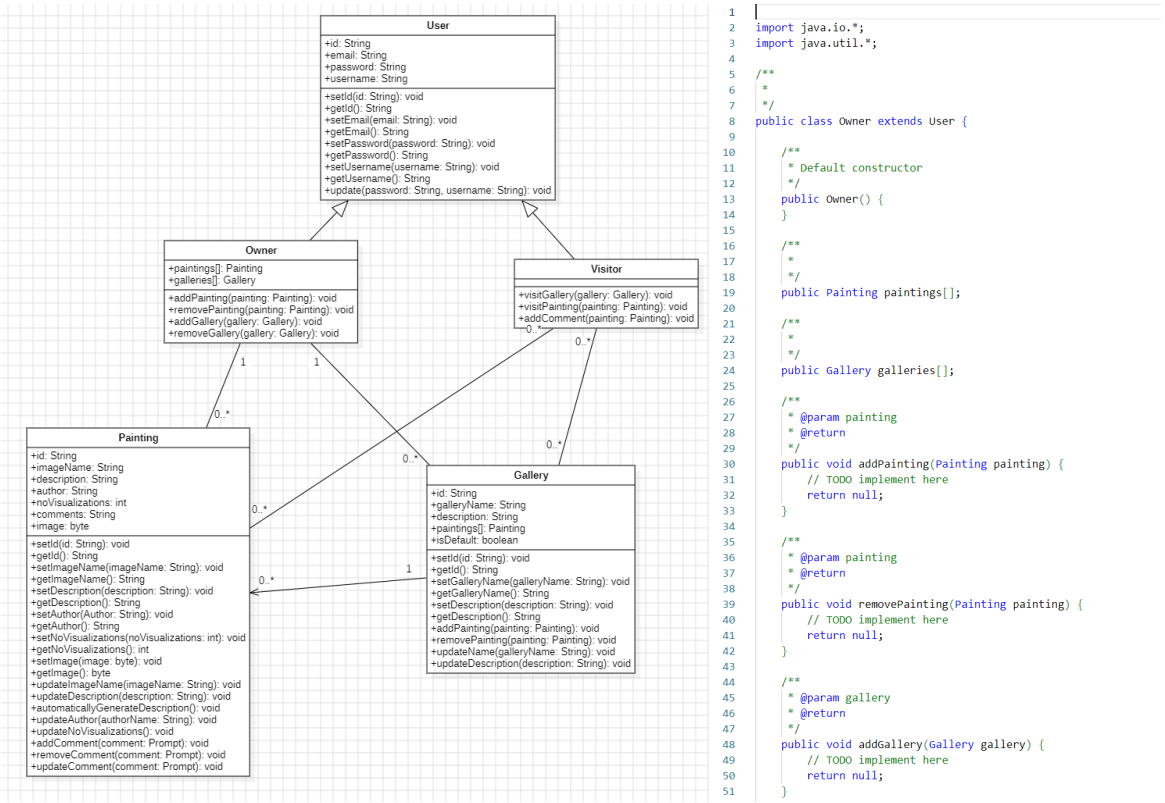
Figure 3.21: DiagramGPT vs. IntelliJ IDEA

Furthermore, IntelliJ includes complete details about access modifiers, methods, attributes, and different types of relationships (such as inheritance, composition, and aggregation). This level of granularity makes IntelliJ the preferred choice for large projects where accuracy and continuous updates are essential.

3. **Generating Code from Diagram using *StarUML*** First of all, this tool provides us with everything we need to draw UML diagrams as we see fit. A very interesting feature that helped us quite a lot was the ability to generate code from the diagram we created.

   Advantages:

   - **Automation in the development process:** StarUML helps automate the code generation process from diagrams, allowing for a swift transition from visual modeling to the actual implementation of the application.
   - **Support for Model-Driven Development:** With StarUML, you can implement Model-Driven Development, ensuring consistency between the initial design of the application and its actual implementation, thus reducing the risks of errors during the coding process.




```java
import java.io.*;
import java.util.*;

/**
 *
 */
public class Owner extends User {

    /**
     * Default constructor
     */
    public Owner() {
    }

    /**
     *
     */
    public Painting paintings[];

    /**
     *
     */
    public Gallery galleries[];

    /**
     * @param painting
     * @return
     */
    public void addPainting(Painting painting) {
        // TODO implement here
        return null;
    }

    /**
     * @param painting
     * @return
     */
    public void removePainting(Painting painting) {
        // TODO implement here
        return null;
    }

    /**
     * @param gallery
     * @return
     */
    public void addGallery(Gallery gallery) {
        // TODO implement here
        return null;
    }
```

Figure 3.22: StarUML Diagram and Code

- **Clear visualization of the application's structure:** StarUML provides a clear visual representation of the application's architecture, facilitating the understanding and analysis of complex structures, which aids in better planning and implementation of features.
- **Interactive environment for design and development:** StarUML allows for an interactive approach in creating UML diagrams, and subsequent adjustments to them are easy to make, helping maintain a consistent and efficient workflow.

Disadvantages:

- **Limited Code Generation:** StarUML generates only the basic structure of the class, without including the actual method implementations or other internal details. This can be a disadvantage, as you will need to manually add business logic and the full implementation of functionalities.
- **Learning Curve:** Although StarUML is relatively intuitive for users familiar with UML, for those new to the domain, the interface and available options may require a significant learning period. Additionally, some features may be difficult to discover for

beginner users.

- **Limited Support for Other Programming Languages:** StarUML offers more limited support for code generation in various programming languages, focusing primarily on generating code in Java or C#. This can be an inconvenience if you're working with a less standardized language.
- **Costs for the Full Version:** StarUML has a limited free version, and advanced features are only available in the paid version. As a result, teams or individuals who need access to all of StarUML's features will have to bear the cost of the license.

In the project, we integrated the use of the AI tool **Codeium** for automatic code generation, focusing particularly on creating unit tests. This tool simplified and accelerated the test development process, reducing the time required for manual test writing and ensuring a high level of accuracy.

Figure 3.23: Codeium - Code generation

Advantages:

- **Free Plan Available:** Codeium offers a free plan, making it accessible to developers without requiring an initial financial commitment. This is especially beneficial for smaller projects or for those who want to explore the tool's capabilities before deciding to invest in a paid version. The free plan provides essential features, helping users improve their productivity without additional costs.
- **Seamless Integration:** Codeium integrates easily with popular development environments and IDEs like VS Code and JetBrains. This allows developers to use the tool directly within their familiar workspace, without the need for complex setup or configuration.
- **Rapid Code Autocomplete:** One of the main advantages of Codeium is its ability to provide fast and accurate code autocomplete. The AI-driven autocomplete functionality significantly accelerates the programming process, reducing manual effort and helping developers write code more efficiently. This feature minimizes syntax errors and improves code quality by suggesting relevant completions based on context.

Disadvantages:

- **AI Limitations:** While Codeium offers impressive AI-powered code generation, there are still limitations. The algorithm cannot fully replace the creative thinking and experience of a human developer, meaning it may sometimes suggest solutions that are not the most efficient or suitable for a particular context.
- **Limited Context Awareness:** Codeium has limited awareness of the context in which it is used. While it can quickly complete lines of code or functions, the tool may not always fully understand the logic behind a larger block of code or the developer's intentions, leading to suggestions that may not be correct at all.
- **Lack of Specific Cases:** It struggles to handle more complex or very specific use cases. If there are special requirements or the code logic is more unusual, the AI may not be able to provide relevant completions or solutions, requiring manual intervention from the developer.
- **Redundant Code:** In some cases, Codeium may generate redundant or repetitive code, which can lead to less efficient project structure. While this can help speed up development, it is important for the developer to review and optimize the generated code to avoid unnecessary redundancy.
- **Dependence on AI Assistance:** Constantly using AI for code generation can create a

dependency on external tools, which can reduce the developer's ability to solve problems without its assistance. Over time, this could impact programming skills and understanding of the code in detail, making the developer less autonomous.

Chapter 4

Comparison with other solutions

4.1 Similar Applications

Our application offers two essential main functionalities: the first focuses on restoring artifacts from images, while the second allows users to view and interact with the restoration results in an interactive 3D space. This immersive experience enables users to explore and manipulate the restored objects directly on their device screen in an intuitive way.

Currently, there are no applications that simultaneously integrate both of the components offered by our application: the automatic restoration of artifacts from images and their interactive visualization in a 3D space. Most existing applications focus on one of these functionalities, either image restoration or 3D visualization, but do not combine these processes into a single workflow. Applications that allow image restoration are usually aimed at improving old or damaged photographs, while those offering interactive 3D visualization focus on objects or scenes that are already digitally restored, but do not provide automatic restoration features. Thus, our application brings an important innovation by combining these two functionalities into one platform.

Regarding automatic image restoration, applications such as Pincel, DALL-E, and Inpaint use artificial intelligence technology to enhance and restore old or damaged photographs.

Pincel is a versatile app that stands out by combining creative image generation with practical tools for editing and design. It not only focuses on creating art but also offers a range of useful features for tasks like face swaps, background changes, and even architectural designs. This makes it more comprehensive than typical image generators. Key features include Text to Image, which generates images based on user-provided descriptions, Object Remover for effortlessly

removing unwanted elements, and AI Image Editor, offering advanced editing tools powered by AI. The app also includes AI Image Replicator for duplicating images with consistent quality, AI Portrait for creating stylized portraits, and an AI Background Generator to automatically generate or modify backgrounds. Additional functionalities like Clothes Swap, AI Logo Artwork, AI Architect, and Face Swap allow for even more creative freedom, while tools such as Text to Speech, AI Upscaler, and Photo Extender further enhance the image manipulation process by improving quality, generating speech, or extending image boundaries with new details, similar to inpainting.

DALL-E (OpenAI) stands out for its ability to generate highly complex and detailed images from textual descriptions, offering users advanced options for pinpoint editing and automatic visual corrections. The platform's main features include advanced image generation, where users can create personalized images based on detailed textual prompts, and pinpoint editing, which allows users to select specific parts of an image and add extra descriptions for local modifications. Additionally, DALL-E provides image generation based on existing images, enabling users to upload images and have the platform generate additional content based on them. The app also offers automatic color adjustments and visual corrections to enhance image quality, along with automatic saving of images to the user's account for quick access. Users can organize their creations into a gallery, complete with management options for easy browsing and editing.

Inpaint is a tool that focuses primarily on editing and restoring existing photos, with a strong emphasis on removing unwanted elements and providing advanced retouching features. Its main functionalities include removing unwanted objects from photos, offering a simple and efficient process, and restoring old photographs, including the recovery of lost or damaged details. Additionally, INPAINT provides secondary features such as removing watermarks from images, filling empty or incomplete areas of an image, and portrait retouching to improve facial features or other important details. It also helps with reconstructing the background in missing or damaged areas and removing unwanted shadows and reflections to clean up the image, ensuring a polished and refined result.

On the other hand, for 3D interactive visualization, platforms such as and Google Arts & Culture, Artivive, Artsteps allow users to explore and interact with 3D models of artifacts and artworks, providing an immersive experience, but without integrating the image restoration process.

Google Arts & Culture is an extensive platform that provides access to art collections from museums around the world, allowing users to explore virtual galleries and view artworks in high resolution. The app offers an interactive experience, enabling users to navigate through exhibitions, zoom in on details, and view artworks in 3D. Users can interact with each object, gaining access to relevant historical and contextual information about the works on display, making the exploration not only visual but also educational. Features include 3D visualization of artworks, virtual museum tours, interaction with artwork details, and providing additional information for each exhibit.

Artivive is an augmented reality application that allows users to view artworks interactively. Typically used on mobile devices, the app can also be used to create interactive experiences for art exhibitions, adding an AR (augmented reality) component that provides additional information or animations to paintings and artworks. This app functions like a digital museum, where displayed objects "come to life" through AR technology, offering an immersive and innovative experience for users. Features include viewing artworks via AR, creating interactive experiences for exhibits, and the ability to add animations or additional information for each piece.

Artsteps allows users to create and explore 3D virtual exhibitions, offering an immersive experience similar to visiting a gallery or museum. Users can upload images, videos, or 3D models and build a virtual space where they can interact with these objects. The app provides the ability to display artwork on walls, and users can explore and interact with them intuitively. Its features include creating personalized virtual galleries and exhibitions, adding images and 3D models, an interactive exploration experience, and the option to add descriptions and audio for each exhibit, further enriching the visual experience.

The main goal of these platforms, whether for restoration or visualization, is to make art and culture accessible in an innovative way, bringing technology closer to the general public while offering an educational and engaging experience. Ultimately, these applications aim to transform the way people access, explore, and preserve cultural heritage, presenting it in an interactive, accessible, and personalized digital format.

4.2 Machine Learning component

The improvement in the Machine Learning solution is represented by making transition from the GAN-based solution to the Graph-based one, which follows the following pattern: represents

the image as a graph, where image patches or superpixels are nodes, and the edges encode contextual relationships between these nodes. This representation ensures the model captures both local and global dependencies, essential for achieving structural coherence and visual realism.

The process begins with graph construction, where the image is divided into overlapping patches or superpixels treated as nodes. Edges between nodes are weighted based on spatial proximity and feature similarity, allowing structurally related patches to influence one another. Features for each node are extracted using a convolutional neural network (CNN), which captures local texture and pattern information, enriching the graph representation for accurate restoration. The core of the GraphFill pipeline involves a Graph Neural Network (GNN), which propagates information across the graph, synthesizing realistic textures and ensuring restored regions align with the overall image structure. This iterative update of node features balances local detail with global coherence.

Once the GNN processes the graph, the updated node features reconstruct the missing regions. A refinement step, often involving adversarial loss from a GAN, ensures seamless blending of restored regions with the original image. This combination of GNN and GAN methodologies results in robust, context-aware restoration. The advantages of GraphFill include its ability to handle irregularly shaped missing regions, its computational efficiency, and its enhanced structural fidelity. These attributes make it particularly suitable for artifact restoration, where preserving the authenticity of visual and structural elements is paramount.

4.3 AI-driven software engineering component

A notable rival of **StarUML** is the **Eclipse Modeling Framework (EMF)**, which also offers the functionality of transforming diagrams into code.

StarUML stands out for its more user-friendly and intuitive interface, particularly geared towards users who want to quickly create diagrams without dealing with complex configurations. The drawing process is straightforward and efficient, making it accessible even to beginners. Additionally, local installation is simple, and the tool does not require integration into a complex environment, making it ideal for users seeking quick and direct solutions.

Eclipse Modeling Framework (EMF), on the other hand, provides more advanced functionalities, generating not only the basic structure of classes but also getters, setters, and basic im-

plementations for certain functionalities. Furthermore, it includes mechanisms for validation and persistence, making it suitable for managing complex models, such as detailed relationships between classes or custom data types. EMF also allows for the customization of code templates to meet specific project requirements. Another significant advantage is that EMF is free and open-source.

However, EMF also has several disadvantages compared to StarUML. It is a more complex tool, challenging to use for beginners, as it requires advanced knowledge of modeling and UML. Its dependency on the Eclipse environment can be a limitation for users who prefer other IDEs. Additionally, the learning curve is considerably steeper, and the initial configuration and understanding of the workflow can take a significant amount of time.

In conclusion, StarUML is more suitable for smaller projects or users looking for a simple and quick solution, while EMF is tailored for complex projects and advanced developers who need more sophisticated functionalities.

Chapter 5

Future work

To enhance the functionality and user experience, we are considering the following developments for our application:

- **Badge System for User Engagement:** We plan to implement a badge system that rewards users based on their restoration activity. For example, users could earn badges for restoring a certain number of paintings or based on the type or historical period of the restored artworks. To achieve this, we intend to integrate external APIs such as the Wikiart API to retrieve information about the paintings and their historical context, as well as an API like Google Vision API to identify and classify artworks (e.g., Renaissance, Classicism). Additionally, badges could be awarded based on the age of the paintings or the complexity of the restoration process.
- **Sharing Virtual Art Galleries:** Currently, the application allows users to create immersive virtual galleries to showcase their restored works. In the future, we aim to add a functionality that enables sharing these galleries through a unique link. This feature would allow others to directly explore users' galleries, either from a PC or other devices. We also plan to extend this functionality to mobile devices, ensuring a more complete and accessible experience. This way, users could share their virtual galleries with friends, family, or collaborators in an innovative and straightforward manner.
- **Interactive Notification System:** We intend to add a notification system to keep users informed about their progress in the application. For instance, users could receive notifications when they earn a new badge or when they advance to the next level of a badge (e.g., a badge requiring the restoration of 5 paintings could notify the user when they reach 3 paintings).

Furthermore, we aim to use notifications to create a more interactive experience by introducing a "Story" system. Users could receive messages and personalized stories that immerse them in a captivating narrative, increasing engagement and interest in the application.

Chapter 6

Conclusions

Eventually, we highlighted the favorable and disruptive effects of applying artificial intelligence within software engineering processes. Given the way these applications have been presented, there is evidence that not only so-called intelligent tools assist humans for a more productive outline of development, but humans should also substantially assist and guide intelligent systems to produce reliable outcomes. Among the outcomes, higher-paced programming and testing, more insightful documentation and realistic diagram design are all subject to a collaboration between human and technology, where both intelligences – the human one and the programmed one – bring concurrent contributions.

Additionally, we framed the above mentioned applications within a real-life setting, that of cultural heritage preservation and continuation through the restoration of artifacts. From a high level perspective, we integrated AI tools to indirectly create an intuitive and engaging user experience. As far as usability is concerned, the proposed solution makes sense of virtual reality concepts while following a dual objective: the enjoyable access to facilities and the future prospect of broader access by sharing the assembled museums. In terms of results satisfaction, the generative graph-based pipeline provides the presented solution with state-of-the-art level accuracy. This resulted confluence, of high level and low level AI applicability, framed within concludent end-user scenarios, aims to increase participation of both humans and technology in culture awareness initiatives.

Bibliography

[1] Google Vision API, Software, Available: https://cloud.google.com/vision?hl=en#demo

[2] Wikiart API, Software, Available: https://www.wikiart.org/en/App/GetApi

[3] StarUML, Software, Available: https://staruml.io/

[4] Eclipse Modeling Framework, Software, Available: https://projects.eclipse.org/projects/modeling.emf.emf/downloads

[5] Draw a diagram, Software, Available: https://app.diagrams.net/

[6] Raymond A. Yeh, Chen Chen, Teck Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, Minh N. Do, "Semantic Image Inpainting with Deep Generative Models", arXiv, 2017, Available: https://openaccess.thecvf.com/content_cvpr_2017/papers/Yeh_Semantic_Image_Inpainting_CVPR_2017_paper.pdf

[7] S. Verma, A. Sharma, R. Sheshadri, S. Raman, "GraphFill: Deep Image Inpainting using Graphs", Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, Available: https://shash29-dev.github.io/GraphFill/

[8] S. Verma, A. Sharma, R. Sheshadri, S. Raman, "GraphFill: Deep Image Inpainting using Graphs", GitHub, Available: https://github.com/shash29-dev/GraphFill?tab=readme-ov-file