Report

By: Hassan Shahid (u2354758)

Contents

Abstract

This report details the design, development, and implementation of a full-stack, AI-integrated web application tailored to art education and engagement. Developed using Laravel for backend logic and Vue.js for frontend interactivity, the platform offers art students and enthusiasts a multifaceted experience: exploring curated galleries of artworks, receiving AI-generated critiques on their own creations, and participating in gamified learning activities. Central to the application are two artificial intelligence models: a custom-built Convolutional Neural Network (CNN) for art style classification, and BLIP (Bootstrapping Language-Image Pretraining), a vision-language model that generates descriptive feedback. Beyond its educational objectives, the platform also incorporates robust social features including real-time group chat and a user-driven community upload system. Originally scoped as a static educational tool, the project evolved to meet higher academic expectations through enhanced interactivity, user personalization, dynamic data handling, and multi-model AI integration — demonstrating the potential of AI in transforming traditional art critique and appreciation methods

1. Introduction

The introduction of new technology and creativity has led to a unique tool in the field of art education. This project introduces an intelligent, interactive web application designed to help the academic and creative environment for art students. The platform provides users with the ability to explore historical and present artworks, get AI-driven analysis on their own work, and engage in catalogs, some game experiences that enhance their learning in an enjoyable way. Additionally, as it promotes community interaction through social features such as group chat and community page, where users can upload their own piece of work.

This idea just began as a static gallery concept expanded into a dynamic, full-featured platform, combining personalized user experiences, real-time communication, and machine learning-powered art analysis. The development process was driven by the need to meet both functional and educational expectations of a university-level project, pushing the boundaries of a typical portfolio site into a collaborative and smart system.

1.1 Product Description

The application is a web-based platform that serves three primary functions:

- **Educational exploration** of artwork through a timeline-based catalog and artist database.
- **AI-powered art analysis**, where users receive style classification and descriptive feedback on uploaded art.
- **Community engagement**, enabling users to upload artwork, play learning games, chat in real time, and build a personal profile.

Built using Laravel (PHP) for backend logic and Vue.js for frontend interactivity, the platform combines modern design patterns and asynchronous data handling for a smooth user experience across devices.

1.2 Aim

To develop a accessible, AI-integrated web application that enhances the learning, critique, and appreciation of art through intelligent systems and user-centered design.

1.3 Objectives

- Design and build a visually engaging and intuitive user interface using Vue.js.
- Implement secure user authentication and personalized dashboards.
- Develop AI models to:
    1. Classify artworks by style.
    2. Generate intelligent, descriptive feedback using vision-language techniques.
- Enable interactive, educational gameplay for art history reinforcement.
- Allow real-time collaboration through an integrated group chat system.
- Foster community sharing via user-uploaded artwork and public galleries.
- Maintain a modular architecture with scalable components using Laravel's MVC framework.

2. Justification for Technologies and Design Decisions

The measure of success of a digital solution depends equally on its features as much as it depends on the technologies that power those features. At the core of this platform's development was a systematic evaluation of tools, languages, and frameworks focusing on scalability, maintainability, responsiveness, and seamless integration of AI services. Each technology considered was weighed according to the community support available, presence of documentation, compatibility with the ecosystem, and requirements specific to the responsive and dynamic nature of an art-based educational platform.

2.1 Why an Art-Based Web Platform?

The motivation behind developing an education-based site on art is a startling insecurity by making a "recognized gap" in such current e-learning platforms specifically targeted toward fine arts. Recognition of the scope of availability in e-learning technology is more limited when it is dedicated to the visual arts. Although they produce diverse tools for general education, they hardly ever leave room for interactivity, personalization, or adaptive feedback mechanisms. That is where a classic art critics' setting, such as classrooms and a workshop, seems to have been restricted: by geography, time, and instructor availability.

This project aims at going online with this model as also augmenting it with artificial intelligence sources, democratizing art education, and making it available around-the-globe, more customized to one's pace. It will be more exciting for learning when integrating features such as gamification and social collaboration - an innovation that promotes individual growth as well as discourse in the open discussion-for these are core tenets of art education.

This relates to theories such as constructivism, which focuses on active knowledge construction through interaction and reflection, and connectivism, which emphasizes that learning happens through social networks and shared experiences.

2.2 Why Laravel?

The backend was built with Laravel because of the sleekness and expressive syntax it offers, coupled with comprehensive development tools. PHP based, it is built on Model-View-Controller framework, again calling for organized, modular code and separation of concerns-an important aspect for building a scalable platform that can handle independently different features like chat, authentication, artwork uploads, and AI processing.

Key features that made Laravel ideal for this project:

- **Authentication & Security:** Laravel offers built-in solutions for user authentication, email verification, and password management, enabling a secure user system from the outset.
- **Routing & Middleware:** Its clear routing system and middleware support enabled API crafting and access control between guests and authenticated users.
- **Eloquent ORM:** Laravel's powerful object-relational mapping layer simplified database interactions, reducing boilerplate code and improving readability when managing complex data relationships like user-artwork mappings, likes, and comments.
- **Scalability:** The RESTful APIs and background queue support were enough to assure that Laravel can easily integrate external AI services without blocking user's interaction with the application.
- **Community Support & Documentation:** Laravel's mature ecosystem and active community reduced development roadblocks and provided reliable solutions for common challenges.

Moreover, Laravel's compatibility with Vue.js through Laravel Mix allowed for seamless frontend-backend interaction, which was vital smooth application experience.

2.3 Why Vue.js instead of Blade?

Laravel's native Blade templating engine is fine for static or mildly interactive applications but is incapable of delivering a real-time, interactive user interface. Vue.js, on the other hand, is a progressive JavaScript framework used for developing responsive and modular frontends, which is a core requirement for this project.

Reasons for choosing Vue.js:

- **Reactive Components:** Vue is perfect for dynamic DOM updates, preventing full-page updates that are key for a real-time chat, feedback updates with the AI, or a quiz with a fun gaming element.
- **Component-Based Architecture:** The application is maintainable and scalable because the UI is broken down into reusable components. Each feature (artwork card, comment thread, game quiz, etc.) can be iterated on and tested in isolation.
- **Vue Router & Vuex:** Favourable provisions in these libraries allowed for solid routing and state management features that are crucial when handling user sessions, saved favourites, scores, and other asynchronous data fetching inside a game.
- **Developer Experience:** With easy syntax and good documentation, Vue lent itself to fast prototyping making for rapid onboarding. Vue's learning curve is the lowest when compared to Angular or React, yet the framework is strong enough to build complex applications.

Thus, the project migrated from server-side rendering to a fully dynamic architecture, which, in turn, greatly increased speed, user engagement, and interactivity. Especially important for educational tools that rely on responsiveness, even while maintaining multiple pages within the Laravel application.

2.4 Why Laravel Breeze?

Laravel Breeze was chosen as the foundational scaffolding for user authentication and routing. Compared to other starter kits like Laravel Jetstream or Laravel Fortify, Breeze offers a minimalistic approach, making it ideal for developers who prefer customization and clean, unopinionated structures.

Why Breeze over alternatives:

- **Lightweight & Flexible:** Breeze provides just the essentials — login, registration, password reset, and email verification — without bundling excessive features, making it perfect for tailoring to project-specific needs.

- **Vue.js Integration:** It comes preconfigured with Vue, which simplified setup and ensured compatibility with the frontend framework of choice.
- **Security & Best Practices:** Breeze uses Laravel Sanctum for authentication and CSRF protection out of the box, meeting the project's security requirements without extensive additional setup.
- **Rapid Prototyping:** Its scaffolding allowed quick wireframe testing and frontend-backend integration early in development, saving time during critical sprint phases.

In essence, Laravel Breeze provided the perfect blend of simplicity and extensibility, laying the groundwork for a secure and personalized user experience while allowing full control over further development. Due to the time-sensitive nature of the project, the artificial intelligence models were trained to focus on a limited set of well-defined art styles. This narrowed scope ensured that the models could perform reliably within the constraints of available training data, compute resources, and development time. However, this constraint also presents an opportunity: with extended time and access to broader datasets, the AI could be refined to recognize a wider array of artistic styles and provide even more nuanced feedback, pushing the platform's capabilities toward near-professional critique quality.

3. Literature Review

The convergence of artificial intelligence and digital education has created a fertile ground for innovation in how creative subjects like visual art are taught, critiqued, and experienced online. This literature review explores the theoretical and technological foundations for this project by focusing on three core domains: (1) the use of AI for image classification and critique, (2) the application of natural language processing (NLP) in visual interpretation, and (3) the role of gamification in enhancing art education.

3.1 Machine Learning and Image Classification in Art

The intersection of machine learning and art classification has been widely explored through projects that attempt to replicate human-like recognition of artistic styles. One such study by Saleh and Elgammal (2015) examined how low-level image features such as brushstroke texture, color composition, and layout could be used to categorize artwork by style and artist. Their research laid the groundwork for using Convolutional Neural Networks (CNNs) to perform visual feature extraction — an approach widely adopted in subsequent studies.

More recently, researchers have used the WikiArt dataset to train models to identify over 25 distinct art movements with substantial accuracy. These findings reinforce the feasibility of applying deep learning to visually complex domains like art, where subjective style elements can be converted into quantifiable patterns. However, many of these models are trained on large datasets over extended periods, something this project could not fully replicate due to **academic time constraints**. Therefore, the CNN model developed here was limited to a subset of styles — carefully selected based on clarity, popularity, and visual distinctiveness.

Other studies, such as Tan et al. (2019), emphasize that while CNNs can effectively distinguish art styles, their ability to **understand** or interpret visual meaning remains limited. This realization has led to hybrid models that combine vision with language — forming the basis for the next part of this review.

3.1.1 Model Limitations: Style Classification

The AI model used for art style classification, Convolutional Neural Network (CNN) model was custom build from scratch that currently supports the detection of only three art styles: Cubism, Realism, Japanese_Art, Renaissance and Impressionism.

This limitation primarily came from the limited scope of the dataset used for training, which was selected to meet project deadlines. Additionally, the training was performed under time constraints, meaning that the model was only able to focus on a smaller set of art styles that were clearly distinguishable and widely recognized.

With extended training time and access to a broader and more diverse dataset, the model could be enhanced to recognize more art styles. However, this would require additional computational resources, as well as fine-tuning of the model to increase its ability to generalize across different artistic movements.

3.2 Vision-Language Models and Descriptive Feedback

Traditional image classifiers provide only a categorical label for an image (e.g., identifying a painting as "Impressionism") but fail to offer semantic richness or explanatory depth. To bridge this gap, the project integrated two Vision-Language models — BLIP (Bootstrapped Language-Image Pretraining) and GPT-3.5-turbo — to deliver more descriptive, human-like feedback on artworks.

BLIP was utilized as the initial model to generate a natural language description directly from visual input, capturing elements such as subject matter, color usage, and compositional style. This description, alongside the predicted art style from the CNN classifier, was then provided as input to GPT-3.5-turbo, a large-scale language model optimized for instruction-following tasks. GPT-3.5-turbo synthesized the information into a coherent, professional critique, offering not just a description but also evaluative feedback aligned with the visual and stylistic context.

This two-stage architecture was selected for its ability to combine the strengths of both models: BLIP's capability for vision-to-language translation and GPT-3.5's ability for nuanced, contextually aware text generation. By separating vision understanding and language articulation, the system ensures that feedback is both visually grounded and linguistically sophisticated. Despite the advantages, limitations remain — such as dependency on the pretraining biases of both models and occasional generic phrasing — but the approach marks a significant step toward AI-assisted art critique systems capable of mimicking human aesthetic reasoning.

3.3 Gamification in Education and Art Learning

Gamification has become a powerful instrument in digital learning environments, which refers to the application of game-design elements in non-game contexts. The work of Deterding et al. (2011) and Gee (2007) shows that gamified learning experiences enhance user engagement, motivation, and knowledge retention, particularly in contexts that favour exploratory or associative learning, such as art history. Games that are educational in nature in the art of learning have been investigated in systems such as "Guess the Painting" or the "Art Timeline Challenge," wherein players match artists, styles, or dates to given artworks.

Educational games in the context of art education have been explored in systems like “Guess the Painting” or “Art Timeline Challenge,” where players must match artists, styles, or dates to given artworks. These systems provide cognitive reinforcement through **active recall** and **associative learning**, making them far more effective than traditional reading-based approaches.

This project incorporates similar methods through games like “Guess the Artist” and “Timeline Matcher,” which use visually rich content and time-based challenges to make learning engaging. Unlike static quiz systems, the Vue.js frontend allows for responsive transitions, animated scoring, and real-time feedback — maintaining user interest across multiple sessions.

3.4 Social Learning and Community-Driven Interaction

According to Bandura's Social Learning Theory (1977) and Vygotsky's Zone of Proximal Development (1978), students do learn more through peer interactions. Critique circles, exhibition spaces, and collaborative discussions have always been the undercurrents through which art has learned.

Bringing this model online, the project includes a **Community Upload Page** and **Group Chat Feature**, allowing users to share their work, receive informal feedback, and discuss art topics in real time. Studies in digital pedagogy (Siemens, 2005; Downes, 2012) also support the value of connectivism models, where knowledge is distributed across networks and enhanced through social engagement.

While many platforms offer gallery-style uploads (e.g., DeviantArt, Behance), few include AI-based feedback and integrated educational games — positioning this project as a hybrid between art portfolio, critique platform, and learning tool.

3.5 Limitations in Current Research and Tools

Most existing tools for digital art education are fragmented — offering either viewing experiences, tutorial-based instruction, or social networking — but rarely all three. The use of **AI as an art critique tool** is still in its infancy, often experimental and limited in scope. Moreover, most AI-based tools remain black-box systems, offering little transparency in their reasoning.

This project attempts to bridge that gap by offering both **qualitative feedback and educational interaction**, supported by explainable model output and user-facing critiques. The intent is not to replace human art instructors, but to augment their role, particularly in environments where professional critique access is limited.

4. Methodology

This site arose from a planned and reiterative development lifecycle process in which the Agile principles were followed. The great complexity of full-stack development, AI model integration, human-centered design, and gamification made it imperative to employ an adaptive methodology that could address changing scope as well as test incrementally.

4.1 Development Methodology: Agile with Scrum

Agile Scrum was selected as the primary development methodology to enable flexibility, continuous feedback, and incremental delivery. This framework allowed the project to evolve from its initial vision as a simple art catalogue to a sophisticated, interactive AI-powered learning environment.

Sprint Structure:

- The project was divided into 5 major sprints, each lasting approximately 2 weeks.
- Each sprint included the stages: Planning → Design → Implementation → Testing → Review.
- At the start of each stage, goals were identified using a product backlog created in Trello. Tasks were prioritized based on feature dependencies and integration challenges.
- Sprint reviews allowed reflection on what was achieved, what could be improved, and which new features could be added based on testing and usability feedback.

Evolving Scope: Originally, the project was intended to provide:

- A static gallery of artworks
- A simple AI model for art style classification

As development progressed and feedback was gathered from peers and supervisors, new features were added to enhance value and user engagement, including:

- User authentication and profile management
- Real-time group chat functionality
- Gamified learning tools
- AI-generated feedback using a vision-language model
- A community upload system
- Administrative upload control This adaptive scope was only made feasible by Agile's flexibility and focus on **delivering working software at the end of each stage**, regardless of overall scope shifts.

4.2 Tooling & Collaboration Platforms

To stay organized and ensure efficient version control, the following tools were used:

- **Trello** – for backlog management, sprint planning, and tracking feature status
- **Git** – for version control, with branches used for major features (e.g., ai-analysis, chat-feature)
- **Figma** – for UI mockups and wireframe designs
- **VS Code** – primary code editors for backend and frontend respectively
- **MySQL** – to manage and visualize relational database schemas
- **Python + Jupyter Notebooks** – used for training and validating the CNN model and integrating BLIP

4.3 Time Sensitivity and AI Constraints

The final-year academic timeline-imposed constraints on experimentation, iteration, and dataset preparation for AI training. As a result:

- The CNN model was trained to classify a limited number of distinctive art styles.
- The BLIP model was used with default pretraining rather than fine-tuned on a custom art-specific dataset.
- Image pre-processing, augmentation, and multi-class balancing techniques were used to improve reliability despite the time constraint.

Had there been more time available, the AI could have been trained on a broader style spectrum with greater dataset diversity and labeled nuance leading to enhanced classification accuracy and critique richness.

4.4 Deployment Strategy

The web application was developed and tested locally on a Laravel development server (localhost/theArt) and was intended for deployment via shared hosting or Docker on a cloud platform such as Heroku or AWS. While full deployment was not completed within the academic timeline, a migration-ready architecture (separation of environment config, .env, and config files) was maintained.

5. Requirements

The requirements for this project were established through a combination of self-directed research, peer feedback, supervisor consultation, and iterative refinement during Agile stage planning. They are divided into

functional, **non-functional**, and **potential stakeholder-centric requirements** to ensure a comprehensive understanding of system expectations.

5.1 Functional Requirements

Functional requirements define the specific behaviors and features the system must implement. These were prioritized based on user value, technical feasibility, and their role in supporting the educational and interactive goals of the platform.

User Authentication & Profile Management

- Users must be able to register, log in, and log out securely.
- Users can update personal information (username, email) and reset passwords.
- Users may delete their accounts permanently from the "My Account" page.

Artwork Exploration

- Users can browse artworks across different eras and art styles.
- Users can search artworks by name, artist, or keyword.
- Users can view artwork details including metadata, artist profile, and description.
- Users can like and save artworks to their personal collection.

Artist Database

- Users can explore a catalog of famous artists.
- Clicking on an artist displays a dedicated artist profile page with their biography and artworks.

AI Artwork Analysis

- Users can upload an image to receive:
  - Detected art style via a CNN classifier.
  - Generated textual feedback from the BLIP vision-language model.
- The frontend should dynamically display the result once the analysis is complete.

Community Page

- Authenticated users can upload their own artwork with title, description, and style.
- Users can like and comment on other user submissions.
- An admin panel supports moderated uploads.

Educational Games

- Users can access mini-games such as:
  - *Guess the Artist* – match artwork with the correct artist.
  - *Timeline Matcher* – place artworks or art movements in historical sequence.
- Games must provide immediate feedback and scoring.

Group Chat

- Logged-in users can engage in real-time text chat across the platform.
- Users can delete their own messages.
- The chat component must persist across all pages via a floating icon.

5.2 Non-Functional Requirements

Non-functional requirements address the system's operational characteristics and performance standards.

Responsiveness

- The platform must be fully responsive and optimized for both desktop and mobile usage.

Usability

- All navigation, feedback, and interactions should be intuitive and accessible to a non-technical audience, especially art students with limited digital background.

Performance

- AI feedback must be processed within 5–10 seconds per image on average.
- Pages should load within 2 seconds for standard operations.

Security

- Passwords must be encrypted and stored using Laravel's Hash facade.
- CSRF protection, input validation, and user access control must be implemented.
- Only authenticated users can upload or comment on content.

Scalability

- The system architecture must support future scaling (e.g., AI running on a GPU server, horizontal database scaling).
- Use of RESTful APIs and component-based frontend architecture to allow modular expansion.

Maintainability

- Clean, commented, and modular code using Laravel's MVC structure and Vue.js components.
- Use of Git for version control and branch isolation for major features.

The application is full response for a broader range of devices and can also be optimized for mobile and tablet devices. All elements being made responsive again reduce their volume once the screen width drops under 980 pixels so that it maintains all accessibility in mobile and desktop browsers.

5.3 Stakeholder & User-Centric Considerations

Primary Users: Art students, art educators, and hobbyists

Expectations:

- Immediate access to learning materials without long registration processes.
- Constructive feedback from the AI that mimics a real art critique.
- A safe and engaging environment to interact with peers and explore content at their own pace.

Future Stakeholders (e.g., Institutions, Galleries):

- Art institutions might integrate this platform to support digital critique in hybrid classrooms.
- Community admins could curate competitions, assign feedback tasks, or highlight themed artworks.

6. System Architecture

The system architecture of this art-based web application was designed with a modular, scalable structure in mind — balancing ease of development, maintainability, and the ability to integrate asynchronous AI services. It follows a **full-stack, multi-layered MVC (Model-View-Controller)** approach, with clearly defined interactions between frontend, backend, and external services.

The architecture can be divided into four core layers:

6.1 Presentation Layer (Frontend - Vue.js)

- **Technology**: Vue.js
- **Responsibilities**:
    - Render dynamic user interfaces and respond to user interactions in real-time.
    - Route users to pages like Gallery, AI Feedback, Community, Games, etc.
    - Handle form submissions (uploads, registration, feedback) and API calls using Axios.
    - Maintain state (e.g., logged-in user data, liked artworks) via Vuex.
- **Features**:
    - Real-time group chat widget
    - Artwork galleries and filters
    - AI feedback display with animation/loader
    - Game interface (e.g., Guess the Artist)
    - Personal profile and community upload tools

6.2 Application Layer (Backend - Laravel)

- **Technology**: Laravel (PHP), Laravel breeze for authentication
- **Responsibilities**:
    - Handle business logic, user authentication, session management
    - Serve RESTful API endpoints to the Vue.js frontend
    - Manage database operations using Eloquent ORM
    - Route artwork uploads and analysis requests to AI services
    - Apply security middleware (e.g., CSRF, route protection, validation)

Key Laravel Routes (Examples):

- /theArt/artworkscatalog – fetch artwork listings
- / theArt /aimodel– POST route for AI model invocation
- / theArt /chat – chat message endpoints
- / theArt /profile/update – user profile updates

6.3 AI Microservices Layer (Python + FastAPI)

- **Technologies**:
  - Python
  - FastAPI (microservice API)
  - PyTorch (CNN training + BLIP integration)
- **Responsibilities**:
  - Accept artwork image POST requests from Laravel
  - Run analysis using:
    - Custom-trained CNN model to predict art style
    - BLIP model to generate feedback text
  - Return JSON response to Laravel backend with:
    - Predicted style
    - AI-generated descriptive critique

Why FastAPI?

FastAPI was selected as the framework for handling the AI microservices — a decision based on its speed, simplicity, and strong support for modern Python tooling, including async I/O and deep learning libraries.

Technical Benefits of FastAPI for this Project:

- **Asynchronous Processing**
  FastAPI supports asynchronous endpoints by default, enabling the system to handle large AI inference tasks (e.g., running CNN or BLIP models) without blocking requests. This was crucial for maintaining responsiveness in a web app that relies on real-time user interaction and smooth UX.
- **Speed & Performance**
  FastAPI is built on Starlette and Pydantic, offering one of the fastest Python-based API experiences — ideal for AI services that need to respond quickly with predictions or feedback.
- **Automatic Documentation & Type Validation**
  FastAPI auto-generates Swagger UI and Redoc documentation. During development, this helped validate requests/responses and visualize model parameters quickly. Pydantic-based data validation also ensured that image inputs and JSON structures were strictly enforced — reducing errors during integration with Laravel.
- **Seamless AI Integration**
  Since both CNN and BLIP models are built in Python (using PyTorch and Transformers libraries), FastAPI allowed direct model invocation with minimal setup. Laravel simply sends a POST request with the image payload, and FastAPI handles everything from preprocessing to prediction, then returns a structured response.
- **Scalability**
  FastAPI can be containerized (e.g., with Docker) and scaled independently from Laravel. This separation of concerns lays the groundwork for future deployment across microservice architectures or cloud inference servers (like AWS Lambda or GPU instances).

Why Not Integrate AI Directly into Laravel?

While Laravel is highly capable for web logic, PHP is not optimized for heavy computation or deep learning workflows. By decoupling AI logic into a FastAPI microservice, we can:

- Avoid overloading Laravel with tasks it's not built for.

- Gain flexibility to develop, test, and optimize AI pipelines independently of the frontend/backend UX logic.
- Align the architecture with modern deployment trends, where web and ML services are treated as modular and independently scalable components.

6.4 Data Layer (Database - MySQL)

- **Technology**: MySQL (via Laravel Eloquent ORM)
- **Entities**:
    - Users: stores login info, preferences, activity
    - Artworks: stores artwork metadata and image URLs
    - Comments, Likes, GameScores, Messages
    - AI_Feedback_Log: optional table to track submitted analyses and results

6.5 System Workflow Overview

Flow from user action to AI response:

1. **User uploads artwork** via Vue interface.
2. Image is sent to **Laravel**, which stores the image locally or on cloud and then routes it to the **FastAPI AI service**.
3. The AI service processes:
    - CNN model → style prediction
    - BLIP model → generates critique text
4. JSON response is returned to Laravel → passed to Vue.js.
5. Vue renders the response dynamically alongside a loading animation.

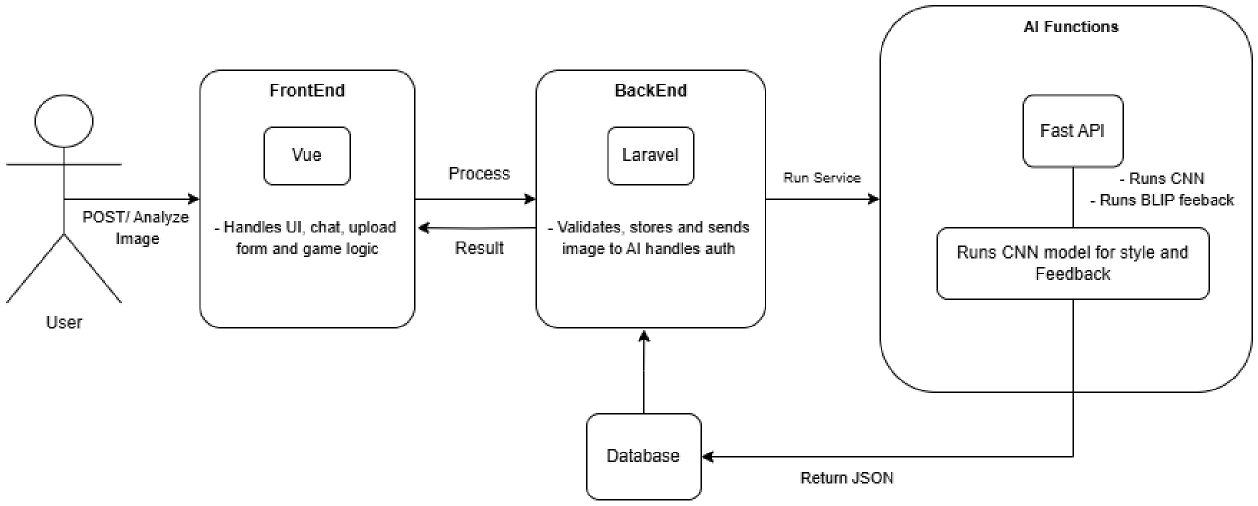


Fig: User to AI System Flow

7. Software Design

The software design for this platform was centered on a modular and maintainable architecture that supports a variety of user interactions, AI service calls, and dynamic user experiences. This section outlines the core architectural patterns, design decisions, UI/UX logic, and supporting diagrams that illustrate system functionality at both frontend and backend levels.

7.1 Wireframes and Mockups

To ensure the platform was user-centric and visually coherent, **medium-fidelity wireframes** were created using **Figma**. These wireframes served as blueprints for designing each of the major user journeys and pages:

- **Welcome Page**: A minimalist landing screen with clear calls to action (browse artworks, take a quiz, register/log in).
- **Home Page**: Displays trending artworks, community submissions, and most-liked or saved works.
- **Catalog Pages**: Both artworks and artists have dedicated filterable, searchable grid layouts.
- **AI Analysis Page**: Features a central file upload area, processing animation, and a results panel.
- **Game Pages**: Simple and colorful UI with quick feedback, scorekeeping, and educational reinforcement.
- **Group Chat Component**: Always accessible via a floating toggle; expands into a minimal messaging box.

Each page layout was designed with mobile usability in mind, ensuring full responsiveness at breakpoints below 980px. Key UI components (such as the chatbox, quiz modules, and image upload forms) were tested and styled to accommodate both portrait and landscape orientations on mobile devices.

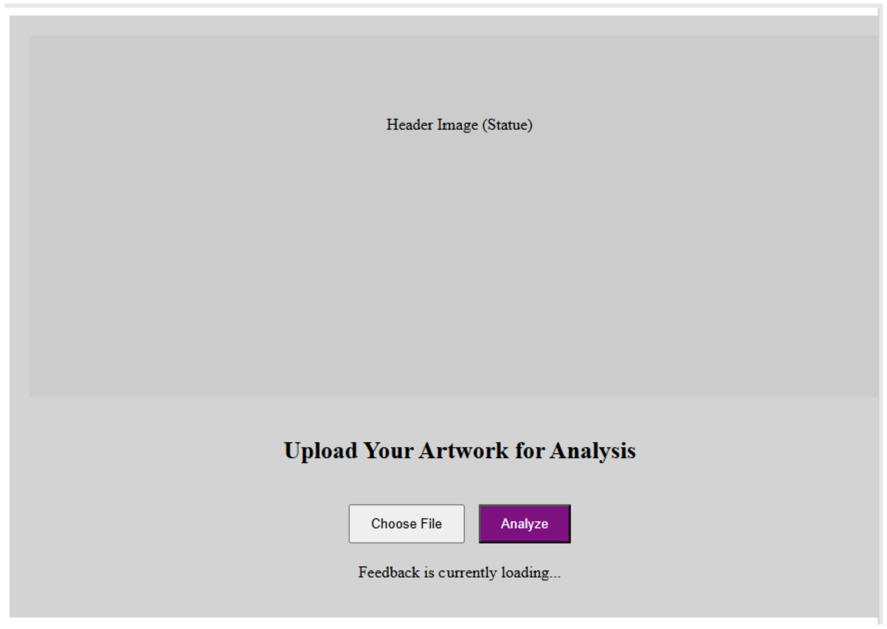

Header Image (Statue)
Upload Your Artwork for Analysis
Choose File
Analyze
Feedback is currently loading...

Fig: WireFrame of AI

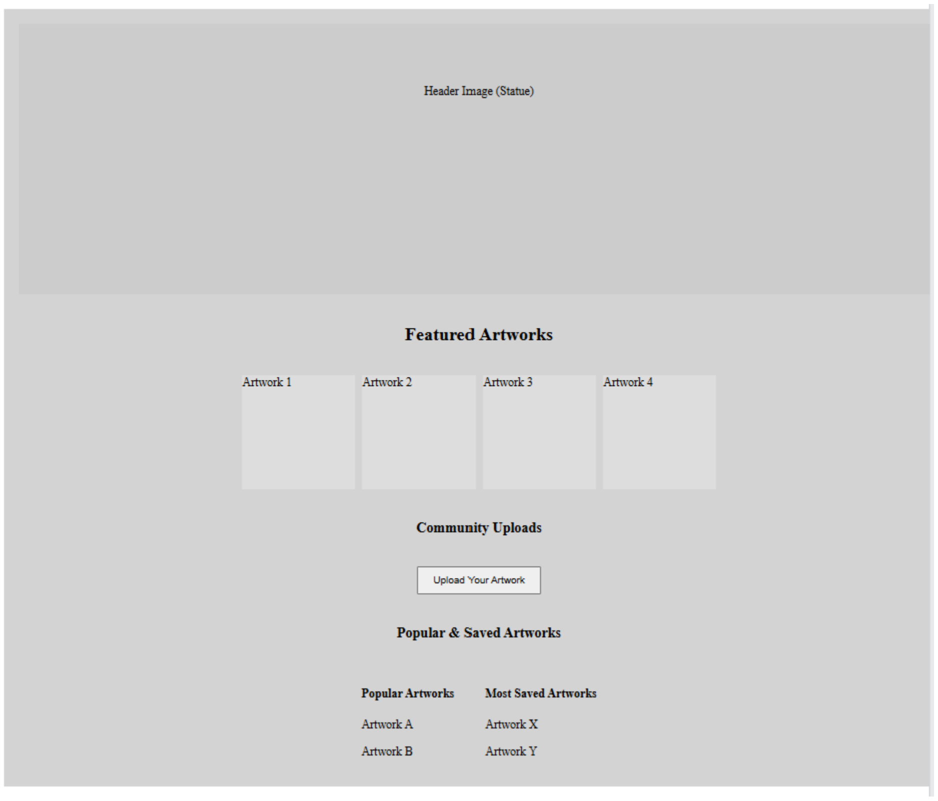


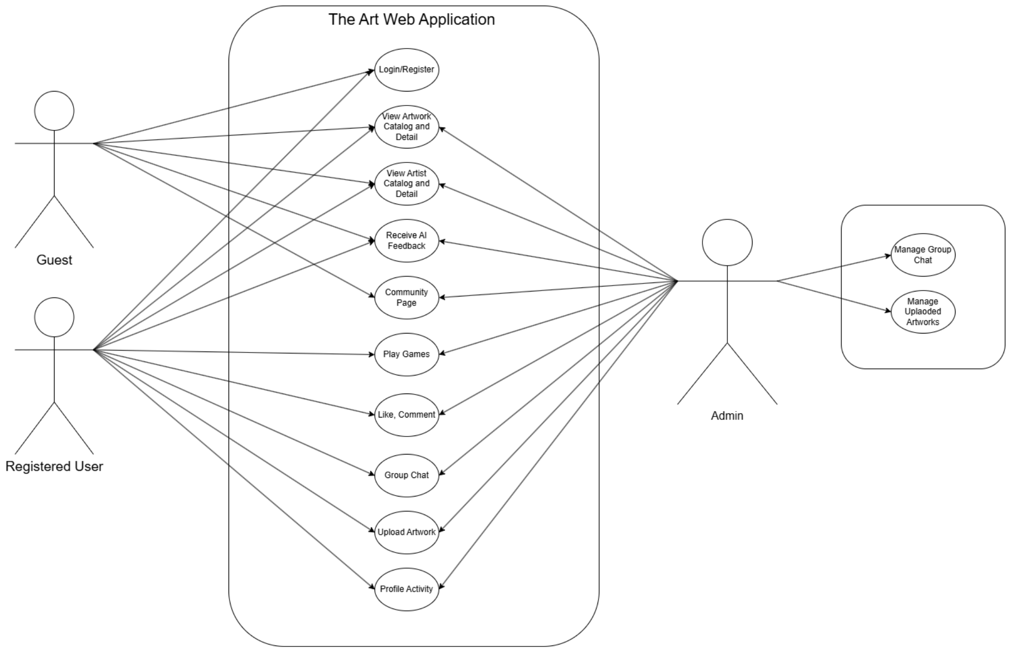
Fig: WireFrame of Home Page

7.2 Diagrams

To illustrate system logic and user flows, the following diagrams were developed:

7.2.1 Use Case Diagram (Access of Different Users)

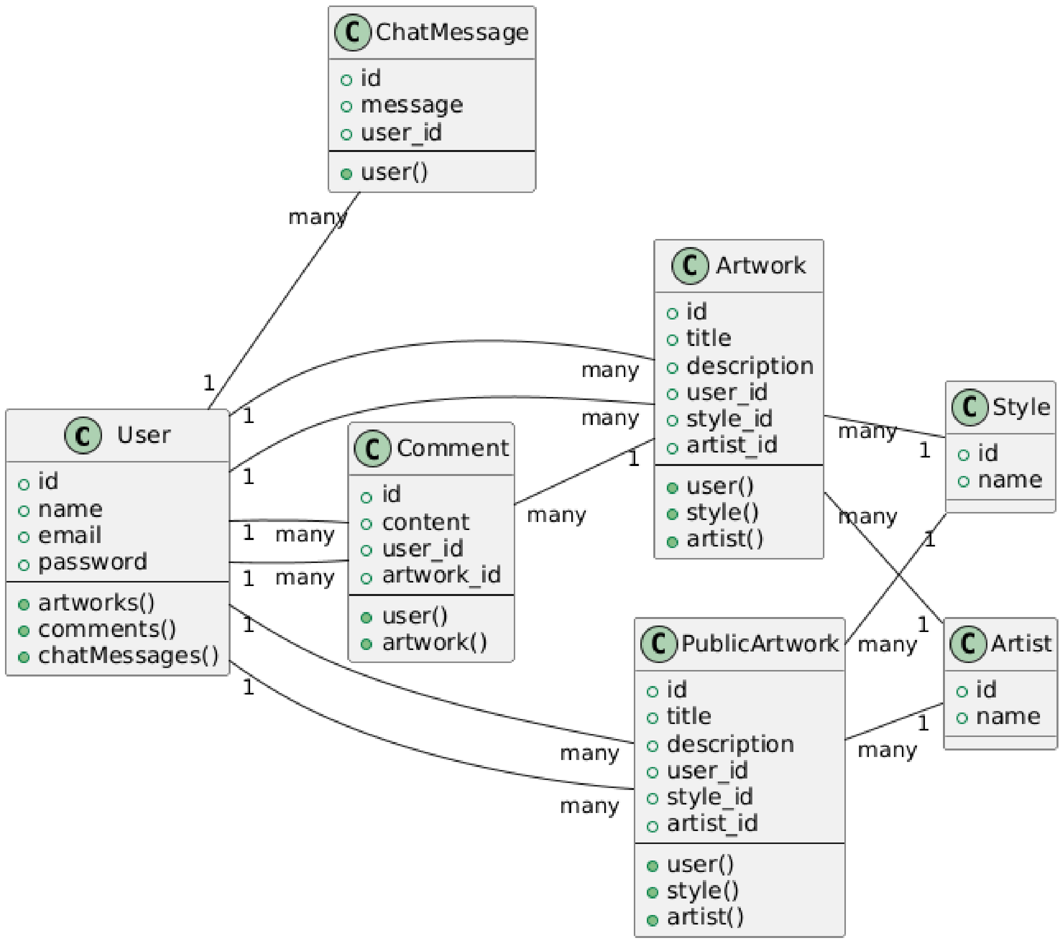
- Actors: Guest User, Authenticated User, Admin
- Use cases: Register/Login, Browse Artworks, Submit Artwork, Receive AI Feedback, Play Games, Send Chat Messages



7.2.2 Class Diagram (Laravel Models + Relationships)

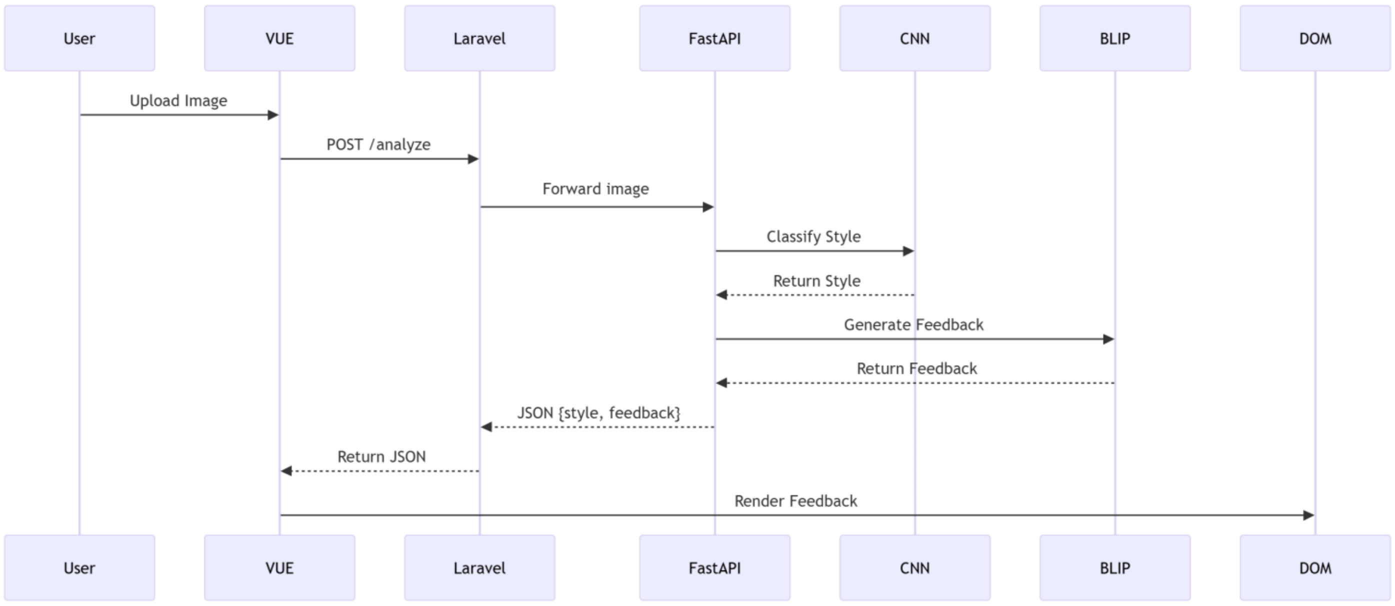
- User → hasMany → Artwork, Comment, ChatMessage
- Artwork → belongsTo → User, Style, Artist
- PublicArtwork → similar to Artwork, but community-submitted
- Comment → belongsTo → User, Artwork
- Style, Artist → reference data tables

This structure reflects adherence to Laravel's **Eloquent ORM** and ensures database normalization.

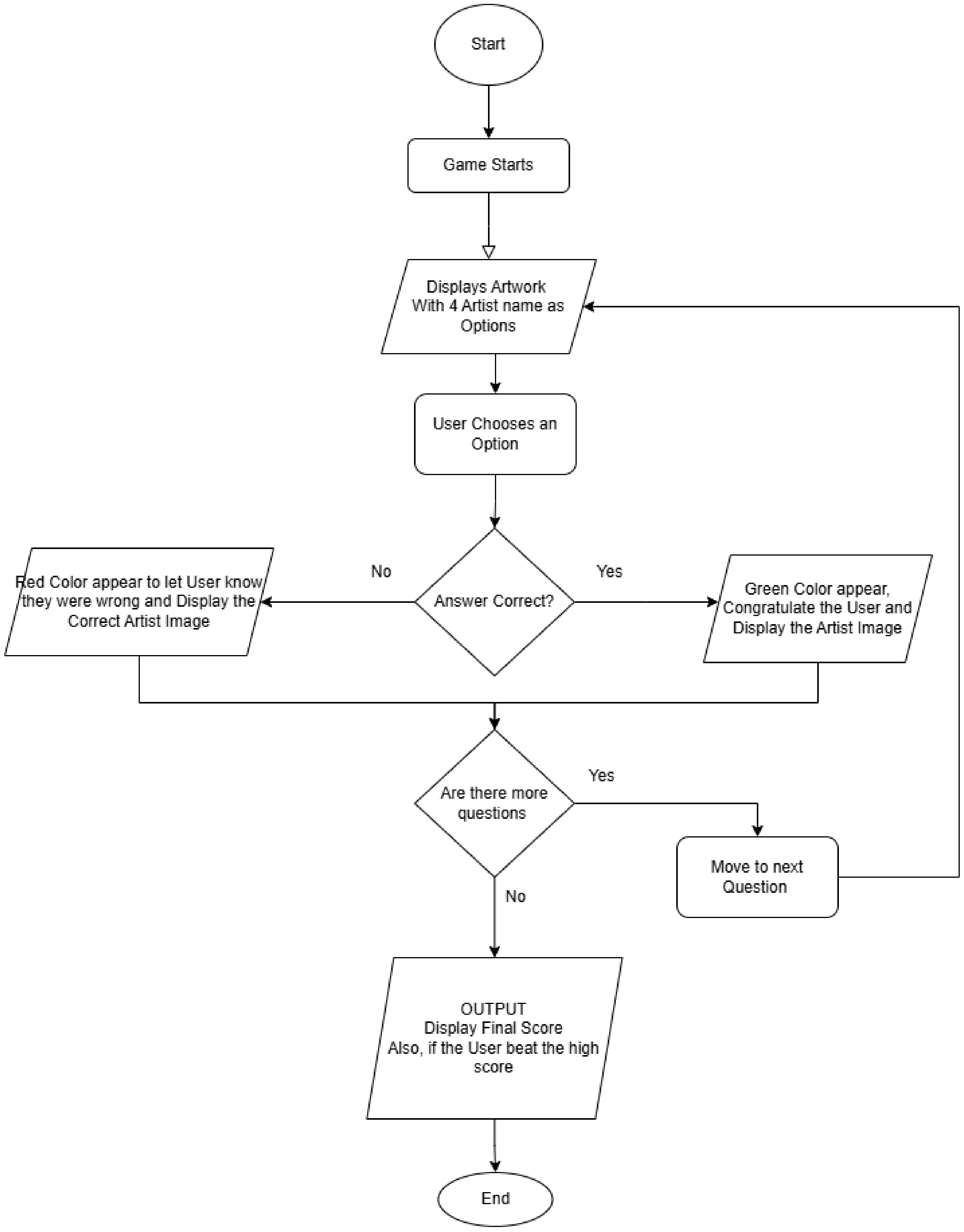


7.2.3 Sequence Diagram (AI Feedback Flow)

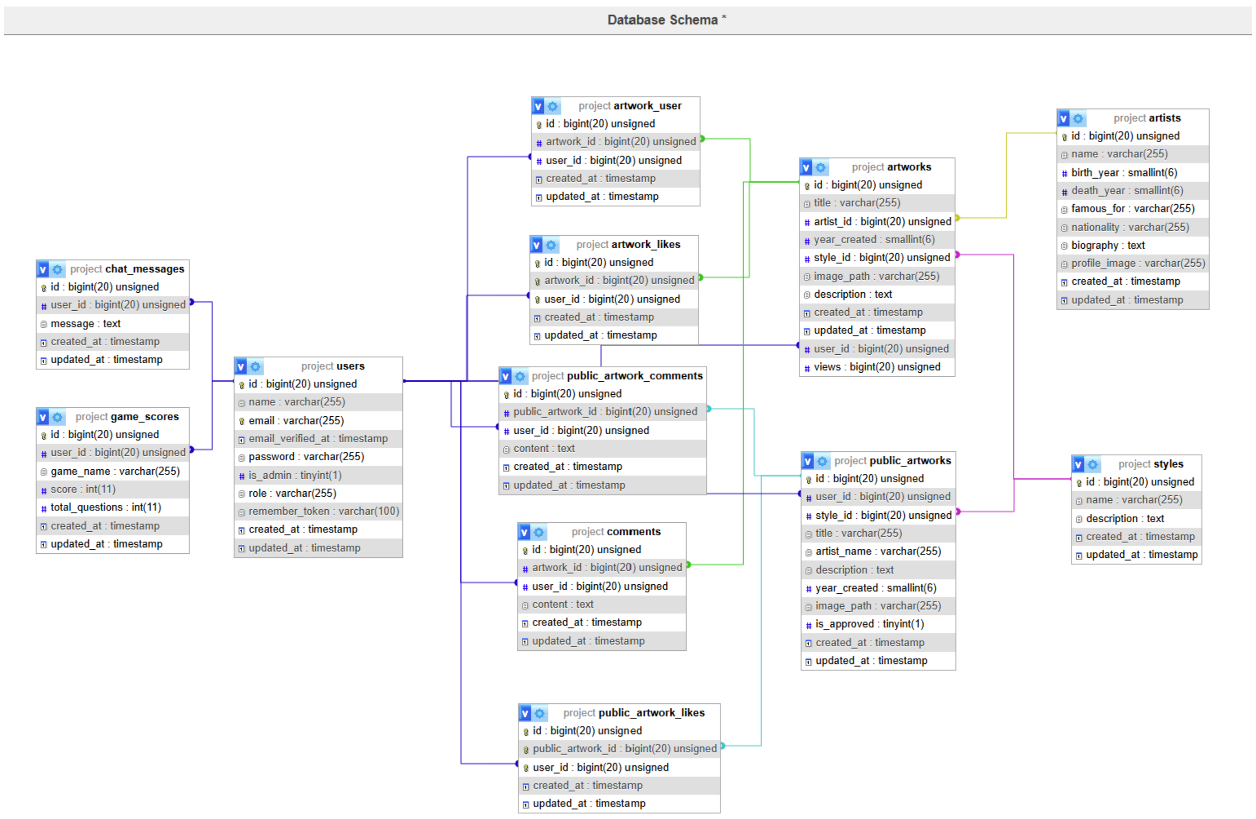
1. User uploads image via Vue form
2. Axios sends request to Laravel /api/analyze
3. Laravel POSTs image to FastAPI endpoint
4. FastAPI runs:
   - CNN → returns style
   - BLIP → returns text feedback
5. Laravel relays results to Vue
6. Vue renders results live



7.2.4 Flow Chart (Game Flow Example: "Guess the Artist")


Start
Game Starts
Displays Artwork With 4 Artist name as Options
User Chooses an Option
Answer Correct?
No
Yes
Red Color appear to let User know they were wrong and Display the Correct Artist Image
Green Color appear, Congratulate the User and Display the Artist Image
Are there more questions
Yes
Move to next Question
No
OUTPUT Display Final Score Also, if the User beat the high score
End

7.2.5 Database Schema (ERD)



7.3 Frontend-Backend Integration

The frontend is built as a Single Page Application (SPA) using Vue.js. It communicates with Laravel via Axios-powered API calls. Each page is structured as a Vue component, and Vuex is used for state management (e.g., user login state, AI response cache, likes).

On the backend, Laravel routes process incoming requests, validate data, interact with the MySQL database through Eloquent models, and forward AI-related requests to FastAPI. The architecture separates concerns between presentation, logic, and services, ensuring maintainability and scalabiliy

7.4 Frontend Component Architecture (Vue.js)

The Vue component structure used in this project reflects a clean, modular, and scalable design approach, adhering to best practices in modern frontend engineering. The architecture promotes reusability, maintainability, and clear separation of concerns, which is essential in an evolving educational platform like this one.

- **Reusable UI Elements**
  Common interface components such as PrimaryButton.vue, TextInput.vue, Modal.vue, and DropdownLink.vue are isolated into standalone files. This allows them to be reused across multiple views, reducing redundancy and enforcing a consistent design language.
- **Game Components**
  Educational game logic is modularized into guessArtist.vue and timelineGame.vue. This separation ensures that future games can be added without modifying the core layout logic.

- **Context-Aware Layouts and Navigation**
  The Pages/ directory maps directly to the application's route structure, with files like ArtCatalog.vue, Community.vue, and Upload.vue acting as page-level views. These pages are wrapped in global layout components and are fully reactive to authentication state, navigation, and user actions.
- **Vuex & API-Connected Components**
  Each page-level component is connected to state management logic via Vuex and communicates with Laravel's API layer using Axios. This model-view separation mirrors the backend's MVC pattern and supports consistent frontend state control.

This organization facilitates modular testing, makes onboarding easier for future developers, and supports the platform's potential growth into a larger, multi-user ecosystem.

7.5 Ethical Handling

In designing and deploying the AI-driven artwork analysis system, ethical considerations were carefully addressed. Sensitive data such as API keys were securely managed using environment variables (.env files) to prevent accidental exposure. To ensure user privacy, no uploaded images are stored permanently; all processing occurs in memory during the API request. Additionally, GPT-3.5-turbo prompts were specifically designed to avoid personal, political, or sensitive content generation, ensuring that feedback remains respectful and focused solely on artistic elements. If deployed publicly, further safeguards such as authentication tokens, request rate limiting, and billing usage monitoring would be implemented to prevent unauthorized access and misuse.

Also, to ensure security and prevent credential leaks, the OpenAI API key was managed through a separate .env file loaded via environment variables. An example .env.example file is provided with the project source code to illustrate the required format without exposing sensitive information.

8. Implementation Overview

The implementation phase involved bringing together the system's UI, backend functionality, and AI services into a cohesive platform. It followed a **feature-driven development approach**, where each sprint focused on a deliverable: login system, gallery rendering, AI feedback integration, games, etc.

Below is a breakdown of major features, alongside screenshots (provided), and a description of their implementation and behavior.

8.1 Welcome Page

Purpose:
First point of contact. Introduces users to the application and highlights its three core features — Explore Artworks, Meet the Artists, and Test Your Knowledge.

Tech Notes:

- Built as a Vue component with responsive layout
- Navigation bar conditionally renders based on auth state
- Hero image uses layered CSS with dark overlay to improve contrast

8.2 Home Dashboard

Purpose:
Personalized landing after login, blending discovery with activity.

Features:

- Featured Artworks (curated)
- Community Uploads
- Popular vs Most Saved

Logic:
Vue renders FeaturedSection.vue, CommunityGrid.vue, and ArtworkList.vue, each pulling data via /api/home.

8.3 Artworks Catalog

Purpose:
Display of all artworks, searchable and filterable.

Backend Logic:

- ArtworkController.php@index() serves search and filter parameters
- Uses Eloquent to join Style, Artist, and User models
- Search bar is debounced for performance

8.4 Artists Catalog

Purpose:
Explore notable artists, presented in a grid.

Implementation:

- Each artist is a Card.vue linking to a route like /artists/:id
- Data fetched from ArtistController.php@show

8.5 Artist Profile Page

Features:

- Profile photo, bio, lifespan, nationality
- Related artworks shown below

Logic:

- Backend joins artworks based on foreign key
- Displays top 3–6 artworks dynamically

8.6 Artwork Detail Page

Features:

- Artwork image, metadata, description
- “Like” button updates via AJAX
- Related artworks logic: similar style or same artist

Implementation:

- Route: /artworks/:id
- Backend: ArtworkController@show() + Eloquent eager loading

8.7 Community Upload Page

Features:

- Authenticated users can upload
- Artworks are stored with user ID and optional style tag

Validation & Storage:

- Laravel validates title, file type, and size
- Image stored via Laravel’s Storage::disk('public')

8.8 Admin Upload Page

Purpose:
Admins can bypass moderation queue and push curated works.

Differences:

- Admin view includes metadata override
- Accessible only via middleware check (role=admin)

Admin Moderation Capabilities

Administrators are granted elevated privileges that support the overall integrity, safety, and academic quality of the platform. This aligns with best practices in content moderation and responsible platform governance.

Admin Functions:

- **Delete Any User Artwork:** Admins can moderate public and private artworks via backend routes secured with role-based middleware.
- **Moderate Group Chat:** Admins can remove inappropriate or irrelevant messages from the global chat system. This ensures the discussion space remains constructive and respectful.
- **Platform Oversight:** Admins monitor uploads, flagged content, and can enforce community guidelines by removing or disabling user content as necessary.

- **Visibility into All Activity:** Backend logic and views are implemented to give admins access to content metadata, user activity logs, and moderation flags (if extended in future versions).

Technical Implementation:

- A role field exists in the users table to differentiate admins from regular users.
- Laravel middleware (IsAdmin) is used to protect admin-only routes.
- Admin tools are rendered via conditional logic in Vue.js (v-if="user.role === 'admin'").
- Backend controllers such as PublicArtworkController.php and ChatMessageController.php include methods like destroy() restricted to admins.

8.9 AI Analysis Page

Pipeline:

1. Upload triggers POST to /api/analyze
2. Laravel passes file to FastAPI using CURL or Guzzle
3. FastAPI:
    - CNN model → style classification
    - BLIP model → textual feedback
4. JSON result passed back and displayed with loader animation

Frontend Vue Logic:

- Shows “Analyzing...” loader
- Renders FeedbackCard.vue with response

8.10 Community Uploads Feed

Features:

- View other users’ uploads
- Comment on their upload using backend CommentController.php
- Real-time like count
- Content uplaod by users can be seen in a detailed page by other users

8.11 Test Your Knowledge (Games)

Game 1: Guess the Artist

- Shown an artwork, guess from 4 options
- Timed, scores saved locally or via Vuex

Game 2: Timeline Matcher

- Drag artworks into historical order
- Immediate validation + educational reinforcement

Game 3: Style Matcher

- Users are shown an artwork and a set of style options. They must select the correct style that matches the artwork. This game helps users better understand and recognize different art styles.

Game 4: Art Puzzle

- An artwork is split into 9 pieces that are randomly shuffled. The user is challenged to reassemble the puzzle within a set time limit to win the game.

8.11.1 Game Controller:

- A **GameController** in the Laravel backend was introduced to manage the functionalities related to these games. This controller tracks user progress, handles game data, and provides a seamless integration between the game interface and the backend system.

8.11.2 User Progress:

- The progress of each game player is stored and can be accessed through the user's **profile page**. The profile page displays the score the user achieved in each game and shows how many questions or challenges they have completed.

8.12 Group Chat Component

Implementation:

- Floating widget via ChatWidget.vue
- Backend via ChatMessageController.php
- Messages stored in DB; optionally broadcasted if Laravel Echo used
- Only accessible to logged-in users

8.13 Profile Management

Features:

- Users can update profile info, password, email
- Delete account permanently

Security:

- Laravel uses built-in Auth and Breeze scaffolding
- All update routes protected via auth middleware

8.14 Search and Filtering Optimization

To ensure a seamless user experience when browsing the artworks catalog, a robust search and filtering system was developed using Vue.js and Inertia.js. To minimize latency and server load, search input handling was debounced by 300 milliseconds using lodash.debounce, allowing users to type freely without triggering excessive API requests. Filters (such as style, time period, artist, and nationality) were integrated dynamically with the search system, enabling real-time refinement of search results without page reloads.

Performance optimizations were a key focus: skeleton loading animations were implemented to reduce perceived waiting times during data fetching, and image assets were lazy-loaded to improve page responsiveness and reduce initial bandwidth usage. These strategies significantly enhanced the fluidity and responsiveness of the catalog browsing experience, addressing potential bottlenecks caused by large result sets and frequent re-rendering.

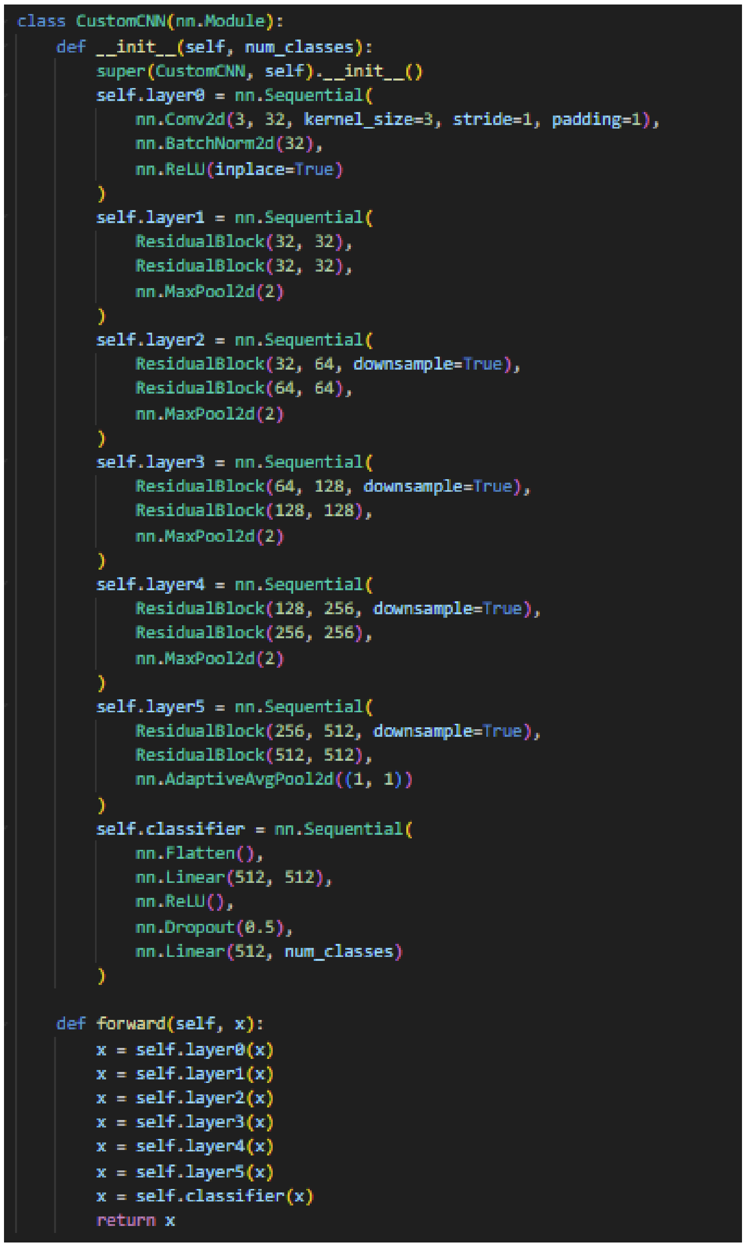
The search and filter logic was abstracted into a reusable composable (useDebouncedSearch.js), maintaining modularity and reusability across components. Overall, this optimization elevated the application's usability, ensuring that users could quickly and intuitively discover artworks of interest without disruptive lag or excessive server strain.

8.15 Custom AI Model Training and Testing

The training of the art classifier model was handled through a custom-built Convolutional Neural Network (CNN) designed in PyTorch. It utilized residual connections to address deep feature extraction challenges and incorporated modern deep learning best practices to enhance model performance.

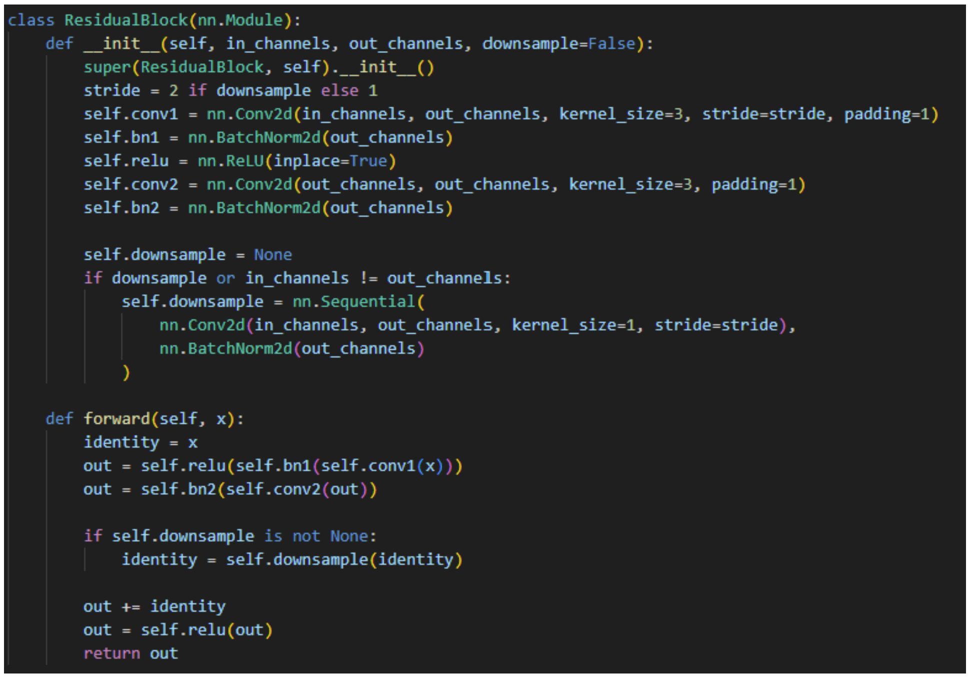
8.15.1 Model Architecture

The CustomCNN model begins with an initial *convolutional layer (layer0)* that extracts low level features from the images that has been feeded to it. This is followed by five *sequential blocks (layer 1 - 5)*, each consisting of two *ResidualBlocks* and a *pooling* operation. As the model progresses, the number of *filters increases (32 -> 64 -> 128-> 256 -> 512)* while spatial resolution is reduced, allowing the network to learn increasingly complex and abstract represntations. The final *AdaptiveAvgPool2d* layer compresses the feature maps to a single value per channel, which is then flattened and passed through a fully connected classifier with *ReLU* activation and dropout regularisation, finally outputting predictions across the specified number of classes.



```
class CustomCNN(nn.Module):
    def __init__(self, num_classes):
        super(CustomCNN, self).__init__()
        self.layer0 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True)
        )
        self.layer1 = nn.Sequential(
            ResidualBlock(32, 32),
            ResidualBlock(32, 32),
            nn.MaxPool2d(2)
        )
        self.layer2 = nn.Sequential(
            ResidualBlock(32, 64, downsample=True),
            ResidualBlock(64, 64),
            nn.MaxPool2d(2)
        )
        self.layer3 = nn.Sequential(
            ResidualBlock(64, 128, downsample=True),
            ResidualBlock(128, 128),
            nn.MaxPool2d(2)
        )
        self.layer4 = nn.Sequential(
            ResidualBlock(128, 256, downsample=True),
            ResidualBlock(256, 256),
            nn.MaxPool2d(2)
        )
        self.layer5 = nn.Sequential(
            ResidualBlock(256, 512, downsample=True),
            ResidualBlock(512, 512),
            nn.AdaptiveAvgPool2d((1, 1))
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.classifier(x)
        return x
```

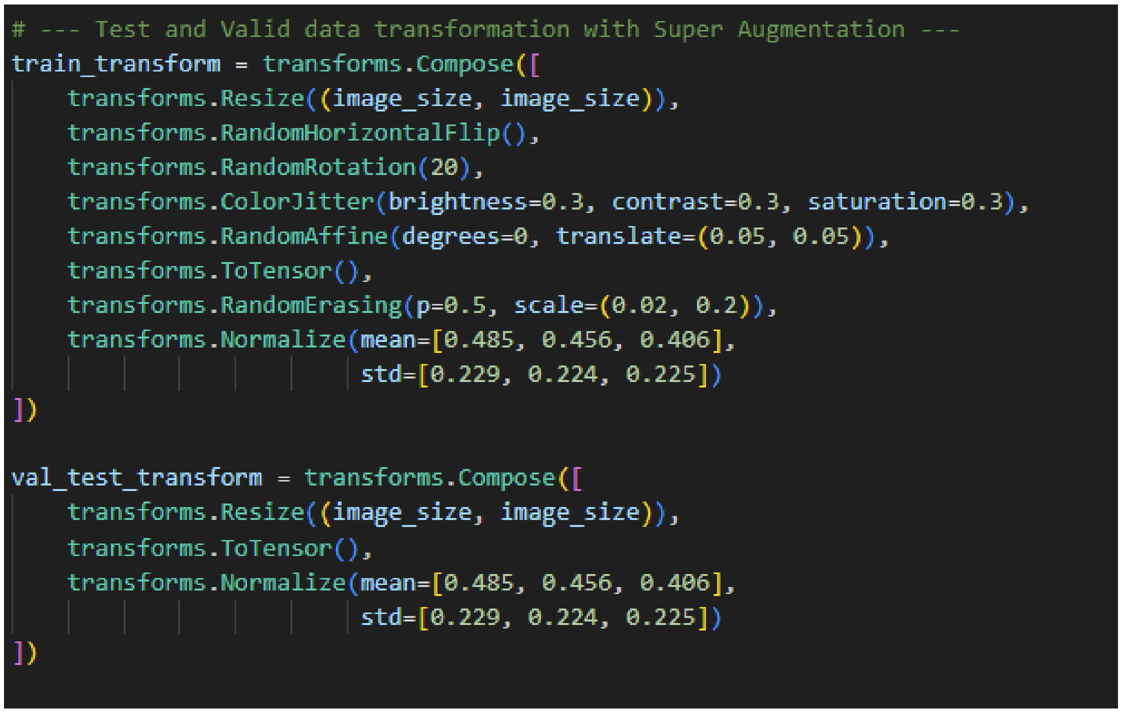
CustomCNN consists of ResidualBlock layers with progressively increasing filters (32, 64, 128, 256, 512). Residual blocks allow learning deep features without degradation by carrying forward gradients directly.



```python
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        stride = 2 if downsample else 1
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.downsample = None
        if downsample or in_channels != out_channels:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        identity = x
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))

        if self.downsample is not None:
            identity = self.downsample(identity)

        out += identity
        out = self.relu(out)
        return out
```

8.15.2 Data Augmentation Strategy

Extensive augmentations like rotation, flip, color jitter, affine transformations, and random erasing were applied to simulate real-world variations and prevent overfitting.



```python
# --- Test and Valid data transformation with Super Augmentation ---
train_transform = transforms.Compose([
    transforms.Resize((image_size, image_size)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3),
    transforms.RandomAffine(degrees=0, translate=(0.05, 0.05)),
    transforms.ToTensor(),
    transforms.RandomErasing(p=0.5, scale=(0.02, 0.2)),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_test_transform = transforms.Compose([
    transforms.Resize((image_size, image_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

8.15.3 Loss Function and Optimization

Cross-entropy loss with class weights was used to handle class imbalance. Optimizer: Adam with CosineAnnealingLR scheduler to gradually reduce the learning rate for better convergence.

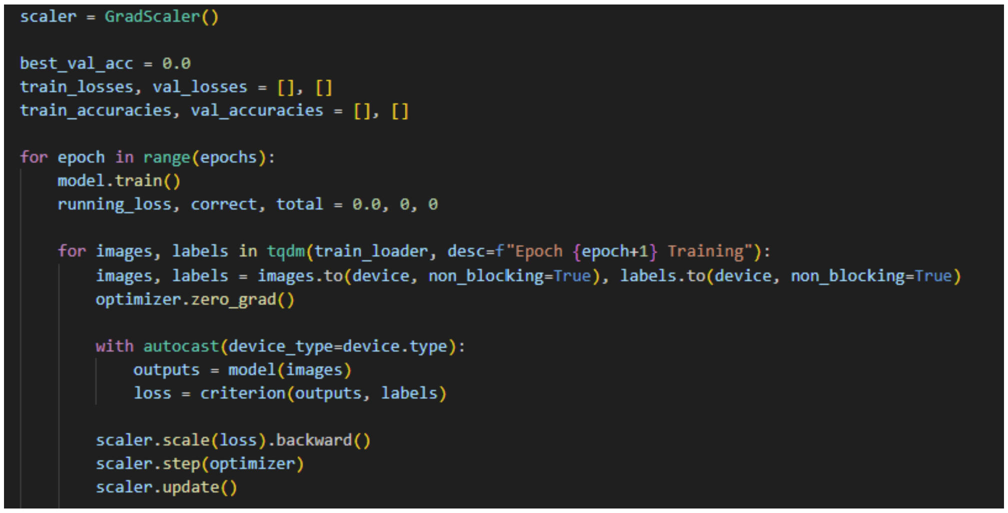
criterion = nn.CrossEntropyLoss(weight=torch.tensor(class_weights, dtype=torch.float).to(device))

optimizer = optim.Adam(model.parameters(), lr=0.0003, weight_decay=1e-4)

scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

8.15.4 Mixed Precision Training

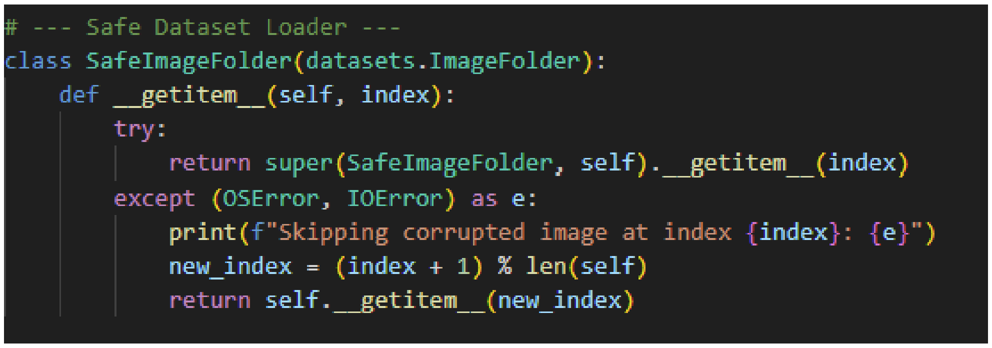
Using torch.amp's autocast and GradScaler improved training speed and reduced memory usage, allowing larger batch sizes without numerical instability.



```python
scaler = GradScaler()

best_val_acc = 0.0
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(epochs):
    model.train()
    running_loss, correct, total = 0.0, 0, 0

    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1} Training"):
        images, labels = images.to(device, non_blocking=True), labels.to(device, non_blocking=True)
        optimizer.zero_grad()

        with autocast(device_type=device.type):
            outputs = model(images)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
```

8.15.5 Safe Data Loading

SafeImageFolder was used to automatically skip corrupted or truncated images during training.



```python
# --- Safe Dataset Loader ---
class SafeImageFolder(datasets.ImageFolder):
    def __getitem__(self, index):
        try:
            return super(SafeImageFolder, self).__getitem__(index)
        except (OSError, IOError) as e:
            print(f"Skipping corrupted image at index {index}: {e}")
            new_index = (index + 1) % len(self)
            return self.__getitem__(new_index)
```

8.15.6 Model Saving and Monitoring

Best validation accuracy checkpoints were saved, and Matplotlib was used to plot loss and accuracy curves over epochs.

if val_acc > best_val_acc:

    best_val_acc = val_acc

torch.save(model.state_dict(), model_save_path)

8.15.7 Model Testing and Evaluation

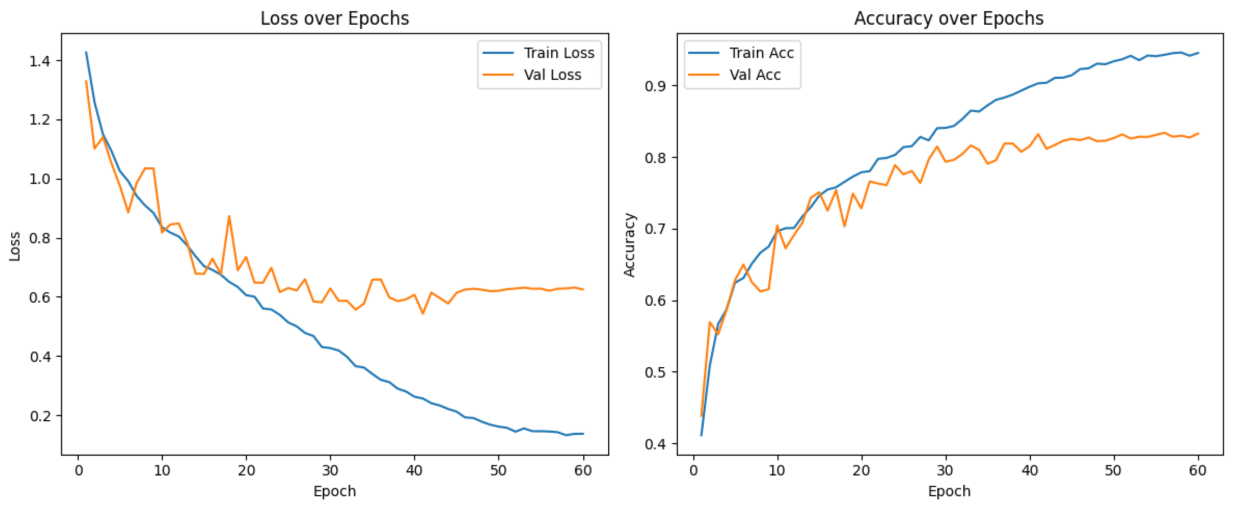
The saved model was evaluated using minimal test-time augmentation. Overall test accuracy was calculated and visual predictions were displayed to qualitatively assess results.

model = CustomCNN(num_classes=num_classes).to(device)

model.load_state_dict(torch.load(model_path))

model.eval()

8.15.8 Training Curve Graph



This is shows the model has been learning well during it's training as the Validation and Train accuracy is almost the same. The model is not overfitting as it used to in my previous testing results.

Overall it has been able to learn well given the time period and short dataset I was able to find. Testing Accuracy end up hitting 85%+, which means model was able to classify the images to therir respective labels pretty well.

9. Testing Strategy

The testing process was iterative and followed an Agile test-as-you-build model. Both manual and automated testing approaches were applied depending on the component type. The goal was to ensure functional correctness, data integrity, AI output validity, and a seamless user experience.

9.1 Types of Testing Performed

Unit Testing

- **Scope**: Laravel backend routes, model relationships, and core controller logic
- **Tools**: Laravel's built-in PHPUnit framework

- **Examples**:
    - Validation logic in ArtworkController ; store method
    - Admin deletion permissions
    - AI response formatting from FastAPI JSON to Laravel view

Functional Testing

- **Scope**: Complete user flows such as login → upload → AI analysis → feedback
- **Method**: Manual test scripts
- **Key Scenarios**:
    - Uploading a valid image returns style + feedback
    - Games register correct/incorrect answers and score
    - Saving and unliking artworks updates UI and DB
    - User deletion removes all related data

UI/UX Testing

- **Scope**: Vue.js components, page responsiveness, visual hierarchy
- **Method**: Device simulation (Chrome DevTools), peer feedback
- **Devices Tested**:

    - Responsive behavior specifically tested for screen widths under 980px
    - Components like navbars, modals, and image cards adapt using CSS media queries and Vue dynamic class bindings

Integration Testing

- **Scope**: Communication between Laravel and FastAPI microservice
- **Focus**:
    - Image POST format (multi-part/form-data)
    - Response consistency (JSON with style, feedback)
    - Error handling (non-image input, missing fields)

AI Output Validation

- **Scope**: CNN style predictions and BLIP textual responses
- **Method**: Compare AI output to known image labels
- **Validation Logic**:
    - CNN must return style from the trained set
    - BLIP descriptions must match primary subject & tone
- **Limitation**: AI is not perfect; some vague or overly general responses were observed

Security Testing

- **Scope**: Authentication, CSRF, form validation
- **Focus**:
    - Guest users blocked from restricted pages
    - Cross-site scripting (XSS) protection on all inputs
    - Secure password storage (bcrypt)

9.2 Bug Tracking & Issue Resolution

Bugs and issues were tracked using Trello cards within each sprint. Common issues included:

- Vue state not updating after API calls (fixed via nextTick() or reactivity watchers)
- Axios timeout on large image uploads (resolved by increasing backend timeout)
- Comment field accepting empty submissions (fixed via frontend + backend validation)
- Chat messages persisting when user logs out (cleared via logout lifecycle hook)

10. Evaluation

The project, as implemented, successfully achieved its core objective: to deliver a fully functional, AI-assisted web application tailored for art education and interactive learning. The platform integrates a range of features — from image-based feedback and community uploads to gamified learning and real-time group chat — each of which was developed with scalability, usability, and engagement in mind.

10.1 Project Strengths

Multi-Modal Learning Environment

By combining visual exploration, textual feedback, social interaction, and gamification, the platform addresses multiple learning styles. This holistic design improves the user's ability to engage with and retain art knowledge.

AI Integration for Feedback

The dual-model AI system (CNN + BLIP) provides users with unique, automated insights into their uploaded artwork. This feature transforms a traditionally subjective, instructor-led experience into a scalable, 24/7 self-learning opportunity.

Modern Tech Stack

Utilizing Laravel and Vue.js allowed for a highly responsive, single-page application that feels modern and intuitive. Integration of FastAPI further allowed AI components to run independently, improving modularity and performance.

Clean MVC + Component-Based Architecture

Both backend (Laravel Controllers & Models) and frontend (Vue Components) follow clear architectural conventions, making the codebase readable, maintainable, and extensible for future contributors.

Community-Driven Design

From the upload system to real-time group chat, the platform promotes interaction, feedback, and learning among peers — mimicking a digital version of the studio critique environment found in art schools.

Role-Based Moderation Tools

Admin controls allow for deletion of user uploads and chat messages, ensuring content guidelines are enforced and the platform remains safe and focused on educational outcomes.

10.2 Limitations & Challenges

AI Model Scope (Time-Constrained Training)

Due to the academic deadline, the CNN model was trained on a limited set of art styles, and BLIP was used with pretrained weights rather than fine-tuned on an art-specific dataset. This limits the accuracy and nuance of the AI feedback.

No Real-Time Multiplayer for Games
Games are currently single-user only. While effective for self-assessment, they lack multiplayer or leaderboard functionality which could have further boosted engagement.

Cloud Deployment Deferred
Although the app is fully functional in a local development environment, time constraints prevented deployment to a live server. Docker-based or Heroku/AWS deployment would be the next logical step.

Limited Accessibility Features
Basic accessibility was considered (e.g., contrast, labels), but deeper WCAG compliance (screen reader support, ARIA roles) could be improved in future versions.

Chat Limitations
Group chat currently stores messages in a database and does not yet support WebSocket-based real-time updates. Laravel Echo or Pusher could be integrated to improve this in future versions.

10.3 Reflection

This project pushed the boundaries of what can be accomplished in a final-year timeframe. It required a blend of design thinking, deep learning model integration, asynchronous system architecture, and frontend interactivity — all developed by a single contributor.

Throughout development, Agile methodologies helped maintain momentum, adapt scope, and stay focused on user value. The iterative testing of AI features, constant frontend refinements, and real-time user feedback simulations contributed to a system that is not only academically sound but also practically usable.

The decision to integrate FastAPI as a separate AI inference engine was critical to balancing performance and separation of concerns — and proved highly effective both technically and educationally.

10.4 Future Enhancements

If additional time and resources were available, the following enhancements are proposed:

AI Model Expansion & Fine-Tuning
Train CNN on a broader style set, with more granular distinctions (e.g., Neo-Impressionism vs Post-Impressionism). Fine-tune BLIP or replace it with newer models trained specifically on art critiques.

Add More Games & Challenges
New interactive features like “Art Bingo”, “Guess the Medium”, or “Match the Movement” could expand engagement. Multiplayer options and timed challenges could drive peer competition.

Live Cloud Deployment
Deploy to a full-stack cloud environment (e.g., AWS, Render, Railway, or Heroku with CI/CD), potentially integrating GPU-accelerated inference for faster AI feedback.

User Notification System
Add email or in-app notifications for new comments, likes, and AI feedback completion.

User Learning Dashboard
Track progress in games, AI feedback history, and saved artworks to generate user-centric learning analytics over time.

Moderation Dashboard for Admins
Expand admin controls with dashboards that track flagged content, user reports, and usage metrics..

11. Conclusion

This project set out to bridge the gap between traditional art education and modern, AI-driven interactivity — and in doing so, it demonstrates how technology can enhance both learning and creativity in meaningful ways. By developing a full-stack web application powered by Laravel, Vue.js, and FastAPI-integrated machine learning models, the platform offers a multi-dimensional experience for students and art enthusiasts alike.

Users are not only able to explore historical and contemporary artworks, but also to receive personalized AI feedback on their own creations, engage in community-based critique, and test their knowledge through interactive games. These features collectively transform the application from a static resource into a dynamic educational environment.

In a technical sense, the modular MVC architecture, component-based frontend, and microservice-structured integration of AI make the system capable of scaling, securing, and maintaining itself. Chat functionality ensures real-time interactions and prepares a path forward for expansion in multiplayer learning games and AI-based real-time feedback systems.

Despite the limitations imposed by a final year academic schedule, the result was an unusually rich and functional feature set, largely due to an agile development approach, careful planning, and a willingness to iterate on design and technical implementations.

The project underlines the growing importance of AI in the area of creativity: neither as substitute for human judgment, nor as a mere tool for augmentation, self-assessment, and democratization of feedback. This defines the application not only as a student project, but as a prototype of what art education could actually look like in the near future having smart technologies, gamification, and user-centric design.

12: Future Enhancements

While the application has satisfied several critical objectives set out at the beginning of this project, there is still a lot of room left for further development and improvements. Below are some of the areas of possible enhancement that could be undertaken with all or any combination of an extended time frame, extra resources, cooperation with a larger development team, or with one of the institutional partners.

12.1 AI Model Improvements

- **Expanded Style Classification:**
  The current CNN model is trained on a select few artistic styles. With more time and dataset diversity, the model could be expanded to cover a broader range — including lesser-known movements and non-Western art forms — for deeper critique capabilities.
- **Fine-Tuning BLIP or Transitioning to Advanced Models:**
  BLIP provides strong generic feedback but lacks specificity. Fine-tuning it on an art critique corpus or replacing it with more recent models like GPT-Vision or Flamingo (if open-source) could result in more personalized, structured, and helpful feedback.
- **Real-Time Inference:**
  Implement GPU-accelerated model hosting (using services like AWS SageMaker or Hugging Face Inference Endpoints) to enable near-instant AI analysis results.

12.2 Gamification Expansion

- **New Games & Art Challenges:**
  Introduce additional learning tools like:
  - *Guess the Medium*
  - *Art Technique Quiz*
  - *Match Artist to Quote*
  - *Mini tournaments with leaderboards*
- **Multiplayer Game Modes:**
  Real-time competition between users, either timed or turn-based, could be added using WebSockets and game state management in Vuex.

12.3 Community & Social Features

- **Comment Threads with Threads/Voting:**
  Expand comment systems into full discussion threads with upvotes/downvotes, fostering deeper engagement and content quality.
- **User Profiles & Portfolios:**
  Let users create public profiles that showcase their uploads, game scores, achievements, and learning progress.
- **Follow & Notification System:**
  Allow users to follow favorite artists or peers and receive notifications when they upload, comment, or receive feedback.

12.4 Admin & Moderation Tools

- **Admin Dashboard:**
  Build a visual dashboard where moderators can review reported content, manage user roles, and analyze platform usage metrics.
- **Content Reporting System:**
  Allow users to flag inappropriate artwork or chat messages, making moderation scalable and community driven.

12.5 Deployment and Accessibility

- **Live Cloud Deployment:**
  Move the application to a public domain using services like Heroku, Render, or DigitalOcean. Include CI/CD pipelines for updates.
- **Accessibility Enhancements:**
  Improve screen reader support, ARIA roles, and keyboard navigation to meet WCAG 2.1 AA standards.

12.6 Educational Analytics Dashboard

- **User Progress Tracking:**
  Introduce analytics that track learning behavior — such as quiz performance over time, feedback trends, or most-viewed artists.
- **Teacher Dashboard (Institutional Version):**
  Allow art instructors to monitor student activity, assign AI critique tasks, and organize class-based community discussions.

13: How to run the Web Application

To set up and run the website on your local machine, follow the steps below:

13.1. Install XAMPP:

Download and install XAMPP from https://www.apachefriends.org/, which includes Apache, MySQL, and PHP for running the backend.

13.2. Configure Directory of XAMPP:

By clicking on config (right next to apache or on the same line after openening xampp)and then click http.config file, change the directory path to point to the folder containing the web application, as shown below.

```
DocumentRoot "C:/xampp/htdocs/project-app/public"

<Directory "C:/xampp/htdocs/project-app/public">
```

13.3. Migrate Database:

Open your terminal and navigate to the project folder using "cd folder_name" command. Run the following command to migrate the database and seed it with initial data:

```
php artisan migrate:fresh --seed
```

13.4. Run Frontend:

Since the frontend is built using Vue.js, open the terminal as last time and run:

```
npm run dev
```

This will start the development server and enable the frontend to run smoothly.

13.5. AI Model Setup:

Open a new terminal window and navigate to the directory containing the AI model files called "ai-model" inside project-app folder.

Create a virtual environment, using

```
python -m venv venv
```

Run the following command to start the FastAPI server that handles AI model requests:

```
uvicorn main:app --reload --port 9000
```

This command will start the FastAPI server in development mode, enabling it to reload automatically when any changes are made. Double check the link in the AiController, analyze method as it should be posting to "*http://127.0.0.1:9000/analyze-art*"

The feedback model might still not work as it contains pretrained GPT-3.5 which needs an api key to run which can be provieded to the examiner upon request.

13.6 Fully Functional Website:

After completing these steps, the website should be fully functional. Both the frontend and backend servers will be running, and the AI model will be accessible for generating feedback.

14: References

1. Bandura, A., 1977. *Social Learning Theory*. Englewood Cliffs, NJ: Prentice Hall.
2. Deterding, S., Dixon, D., Khaled, R. and Nacke, L., 2011. From game design elements to gamefulness: defining "gamification". *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pp.9-15.
3. Downes, S., 2012. *Connectivism and Connective Knowledge: Essays on meaning and learning networks*. National Research Council Canada.
4. Gee, J.P., 2007. *What Video Games Have to Teach Us About Learning and Literacy*. 2nd ed. New York: Palgrave Macmillan.
5. Li, J., Selvaraju, R.R., Gotmare, A., Joty, S., Xiong, C., and Hoi, S.C., 2022. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. *arXiv preprint arXiv:2201.12086*.
6. Radford, A., Kim, J.W., Hallacy, C., et al., 2021. Learning Transferable Visual Models From Natural Language Supervision. *International Conference on Machine Learning (ICML)*. PMLR.
7. Saleh, B. and Elgammal, A., 2015. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*.
8. Siemens, G., 2005. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, 2(1), pp.3-10.
9. Tan, M., Leong, H.W. and Lim, E.P., 2019. ArtGAN: Artwork synthesis with conditional categorical GANs. *Proceedings of the 27th ACM International Conference on Multimedia*.
10. Vygotsky, L.S., 1978. *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press.
11. WikiArt Dataset, 2020. *The WikiArt dataset: Large-scale artwork classification*. [online] Available at: https://www.wikiart.org/ [Accessed 20 Apr. 2025].