جامعة الإسكندرية
ALEXANDRIA UNIVERSITY
Faculty of Business
MIS Department

Elysium App

GRADUATION PROJECT

June 2024

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

(وَقُلِ اعْمَلُوا فَسَيَرَى اللَّهُ عَمَلَكُمْ وَرَسُولُهُ وَالْمُؤْمِنُونَ وَسَتُرَدُّونَ إِلَى عَالِمِ الْغَيْبِ وَالشَّهَادَةِ فَيُنَبِّئُكُمْ بِمَا كُنْتُمْ تَعْمَلُونَ)

(التوبة - 105)

جامعة الإسكندرية
ALEXANDRIA UNIVERSITY
Faculty of Business
Management Information Systems Department

Graduation Project

Supervised By:

Dr. Yasser Abdel-Ghaffar

Submitted By:

Mohamed Abdullah Ahmed Mohamed	Afnan Talaat Mohamed Farag
Rodina Mohamed Amin Mohamed Elfieky	Yasmin Abbas Ibrahim Abbas
Sara Elhoseny Ibrahim Mohamed Abdo	(آلاء) Alaa Awad Abdelmohsen Ali Zidan
Omar Wael Abdelkader Elsayed	Mohamed Ahmed Hamido
Mohamed Abdelhamid Elsayed Elbasal	Kareem Ehab Khamis Hassan
Lojain Ehab Abdelrahman Kamal Hamza	Meral Ahmed Ibrahim Alfawi

June 2024

Tabel of contents

List of Figures

Acknowledgement

We would like to express our sincere gratitude to all those who contributed to the success of this project, directly or indirectly.

- **Project Supervisor:** Dr. Yasser Abdel-Ghaffar, for his invaluable guidance and support throughout this endeavor.
- **Teaching Assistant:** Youssef Yassin, for his assistance and feedback.
- **Professors:**
    - Dr. Ghada Al-Khayyat
    - Dr. Abeer Amer
    - Dr. Mohamed Al-Abbasi

  Whose teachings and insights have greatly enriched our academic journey.
- **Family, and Friends:**

  For their encouragement and support.

Project Abstract

Welcome to our graduation project presentation, an application that gathers all artists in one place. The app is divided into three main sections: Community, Courses, and Projects. Each section offers special features to enhance the artist's experience.

1. Community

The Community section aims to promote interaction, creativity, and commerce among artists. It is divided into three parts:

a. Creativity

This is like a social media hub where artists can post their work, ask questions, and interact with others. They can like, comment, and share posts.

b. E-commerce Transactions

Artists can post details about their artwork for sale, communicate directly with Clients, and make transactions through PayPal. This makes buying and selling easier for artists.

c. Events

Artists can announce their events, whether physical or online, allowing others to attend and benefit. Events can include exhibitions, workshops, or live sessions on YouTube or Discord. Event posts include the location, date, and time.

2. Courses

The Courses section provides a platform for artists to share their knowledge and skills. Key features include:

- **Course Listings**: Artists can upload courses via YouTube links.
- **Payment and Access**: Customers can pay for courses through PayPal and gain access to the content.
- **Ratings and Reviews**: Customers can rate and review courses, helping others choose the right course. Artists must provide detailed descriptions, including the difficulty level (e.g., Advanced, Intermediate).

3. Projects

The Projects section works like a freelancing platform, connecting clients with artists for custom projects. The process involves:

- **Client Posts**: Clients create posts describing their project requirements, including a description and the desired timeline.
- **Artist Bids**: Artists view these posts and bid on projects, offering their services.
- **Selection and Agreement**: Clients choose from the bidding artists and start a contract outlining the agreed-upon terms.
- **Communication and Delivery**: Artists and clients communicate via chat, exchange images, and agree on details. Payments are made in stages, starting with a deposit and final payment upon project completion.

Finally

The Unified Artist Platform is designed to support the artistic community by providing tools for networking, learning, and commercial success. We believe this app will greatly enhance how artists interact, learn, and do business, making the art world more connected and accessible. Thank you for your attention, and we look forward to your feedback and questions.

Chapter 1

(Introduction)

The world of art is brimming with talent, but navigating its landscape can be a challenge. Artists often find themselves scattered across different platforms, making it difficult to connect, learn, and share their work effectively. This isolation can hinder both their creative growth and their ability to reach a wider audience. This presentation delves into a solution: The Unified Artist Platform. This application is driven by the desire to bridge these gaps within the artistic community.

Problem:

- **Disconnection**: Artists lack a centralized platform for interaction, collaboration, and promotion.
- **Limited reach**: Selling artwork and promoting events can be difficult, especially for new artists.
- **Uneven learning opportunities**: Finding quality art education and sharing knowledge can be challenging.
- **Freelancing difficulties**: Connecting with clients for custom projects is not streamlined.

App Idea:

The Unified Artist Platform addresses these problems by creating a one-stop shop for artists. Here's how:

- **Community**: Provides a social media hub for artists to connect, share work, and engage with potential clients.
- **E-commerce**: Enables artists to sell artwork directly through the platform, simplifying transactions.
- **Events**: Offers a platform for artists to promote and manage physical or online events, increasing visibility.
- **Courses**: Creates a space for artists to share their skills and knowledge through online courses.
- **Freelancing**: Connects artists with clients for custom projects through a bidding system and communication tools.

Overall, the application aims to be a comprehensive solution for artists, fostering a more connected and successful artistic community.

Motivation:

Our motivation is to create a unified platform that addresses the needs of artists by fostering interaction, learning, and commercial opportunities. We aim to provide a space where artists can thrive, share their work, and grow professionally.

Mission:

Our mission is to support the artistic community by providing tools for networking, education, and commerce. We want to create an environment that encourages creativity, collaboration, and financial success for artists.

Vision:

Our vision is to make the art world more connected and accessible. We envision a platform where artists from all backgrounds can easily find resources, connect with others, and pursue their artistic goals. Through this application, we hope to contribute to the growth and sustainability of the artistic community.

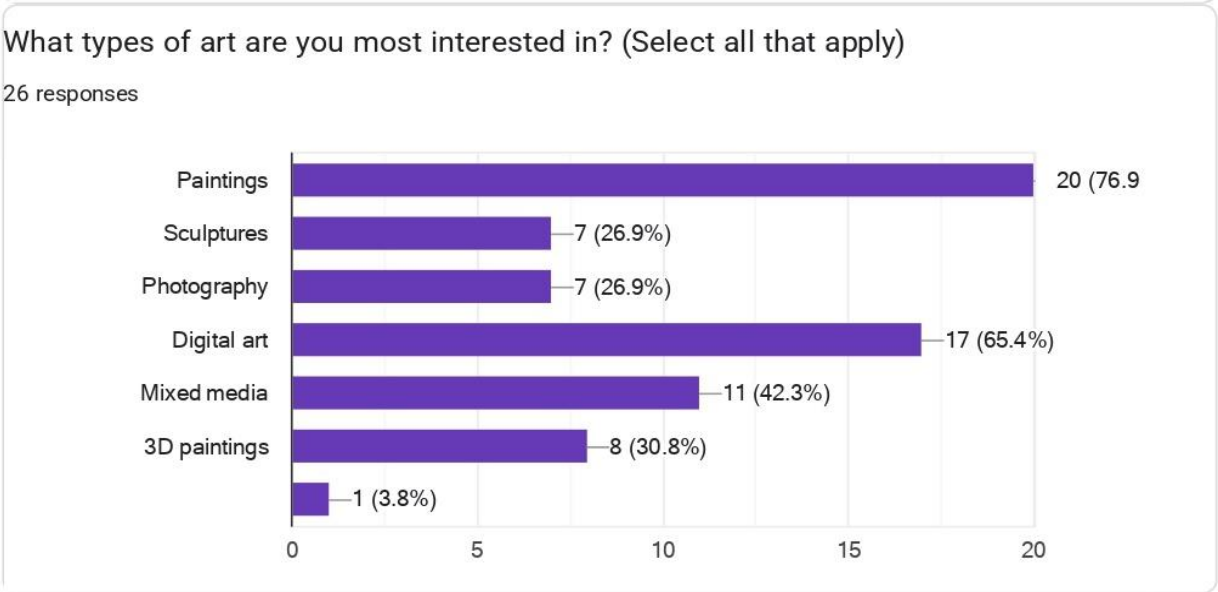
Chapter 2

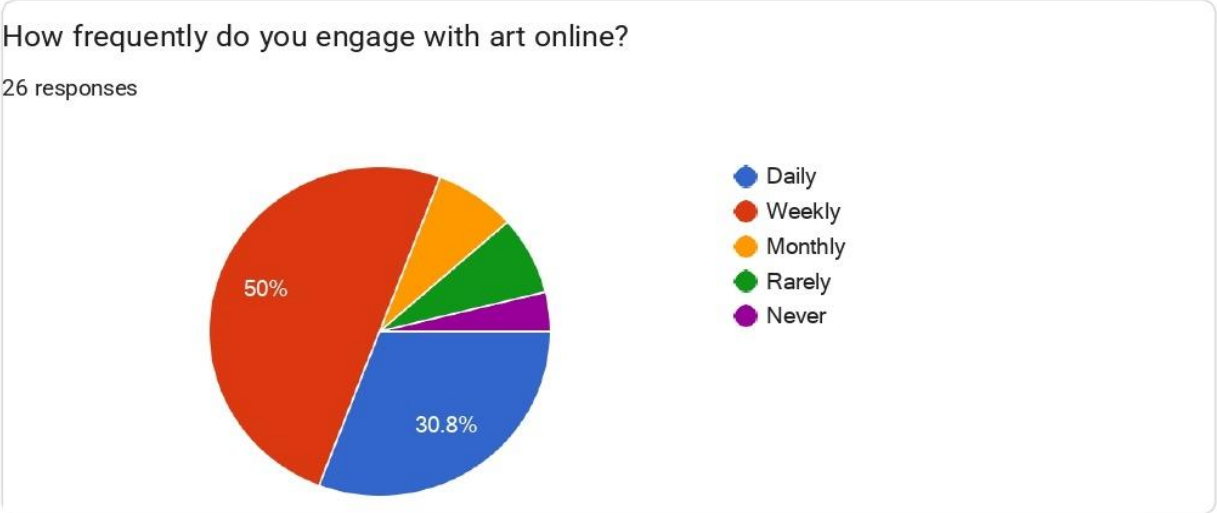
(Background and Literature Survey)

Elysium App Survey

This survey is designed to gather insights from artists about their experiences and preferences regarding online art platforms. The responses will help shape the development of an application that aims to bring artists together in a collaborative environment. The survey covers various aspects, including the challenges artists face on freelancing platforms, their engagement with different types of art, desired features in an online art platform, and preferences for community interaction and educational resources. The goal is to create a platform that effectively supports artists in showcasing their work, connecting with potential clients, and enhancing their skills through educational content.

(URL: https://bit.ly/45aLPhQ)





Can you highlight some common challenges artists face when freelancing on platforms like Upwork? Specifically, issues related to securing projects, managing client expectations, and sustaining a steady income?

17 responses

.

i unfortunately cannot, as i'm still a student that has no experience freelancing on upwork Yes

doesn't reach enough people

I can't find people with the same interest

Prompting their art to people, especially in Egypt

Protecting their work from being imitated

copyrightssssss (no one gonna assigns the work to himself)

I didn't use it before so I don't know but I heard you use connects for it and that seems a bit limiting but I really don't know since I didn't use it.

Finding clients in a platform full of talented artists

.

Competition: Upwork has a large pool of freelancers, including artists, which means there is fierce competition for projects. Standing out and securing projects can be challenging, especially for new or less established freelancers.

Pricing and Budget Constraints: Clients on platforms like Upwork often have limited budgets or specific pricing expectations. Artists may find it difficult to negotiate fair rates that align with their skills and expertise, while also meeting the client's budget constraints.

Building a Reputation: Building a strong reputation and positive feedback on platforms like Upwork takes time. Artists may initially struggle to attract clients and secure projects due to a lack of reviews or testimonials.

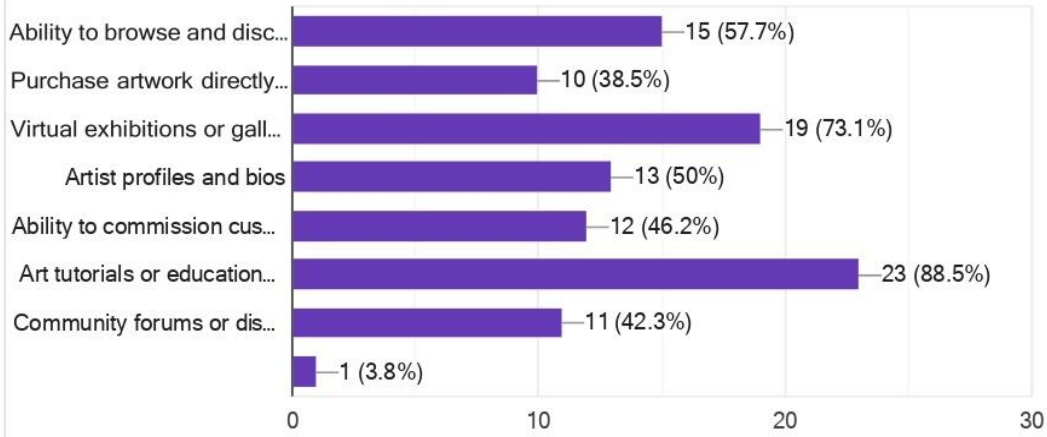
Communication and Language Barriers: Effective communication with clients is crucial for understanding project requirements and delivering satisfactory results. However, language barriers and different time zones can pose challenges, leading to miscommunications or delays in project completion.

all the clients have so many suggestions that sometimes they don't even know what the want

Competition can be fierce on freelancing platforms, making it hard for artists to stand out among other talented individuals.

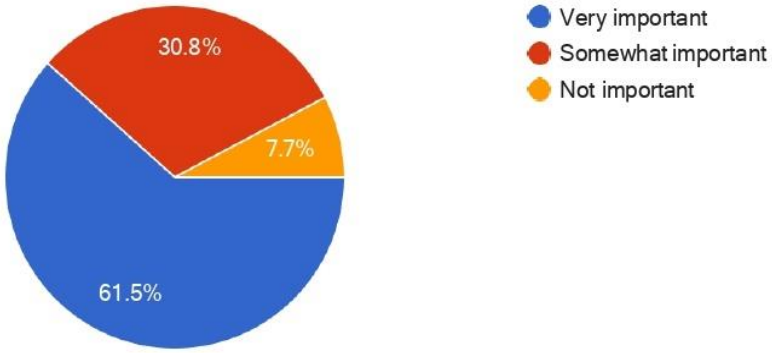
What features would you like to see in an online art platform? (Select all that apply)

26 responses



How important is it for the platform to provide information about the artist's background and inspiration for their work?

26 responses



What improvements or features would make you more likely to use an online art platform regularly?

26 responses

.

Digital art platforms

Fair algorithms
" الفن مجال واسع وكل category تحتها مصنفات وفئات كتير فلازم البرنامج يتعرف على ذوق ال client وتفضيلاتهم عشان يبعت painterest algorithm الاعمال الفنية للأشخاص اللى هيتفاعلوا معاها وينجذبو ليها "حاجه زى
وان المستخدم تطون عنده قابيلة يعمل قوائم مفضلة للفانين والاعمال الفنية اللى اعجبته بحيث يقدر يرجعلها بسهوله

كسوق لعرض الأعمال ...و التعرف علي فنانين وأساليبهم وفنهم

سرعه الوقت واختصار المجهود والتنوع

making mood boards

That it would be colorful and artistic to absorb my attention not like normal platforms Ability

to only share posts and applying credit and stuff so it won't be stolen

Easily access to all the platform subjects and always up to date also can provide exciting information about art and artists

و المعارض يوميا او المسابقات و صالونات الشباب و غيرها من التحديثات باستمرار و انها تكون مجمعة eventsانها تكون بتنزل ال
معلومات الفنانين الملهمين حديثا و فيها معرض للاعمال الفنية

smart algorithm that understands my interests and suggests more of them, allowing me to explore more artists

Is good

being updated regularly with the latest projects

Free educational sources

Useful and helpful tutorials

Reach more people and got to know me

if it's something like Pinterest. a huge collection of art and easy to use

Having a community of artists, the application connecting me with possible clients or projects to join other artists in. I'd like to see a place where artists could pitch an idea and post it for other artists to join or people who'd like to financially support the project.

Ease of use

Idk☺

Enhanced Discoverability: Improved search and discoverability features would make it easier for artists to showcase their work to a wider audience. This could include advanced tagging options, personalized recommendations based on user preferences, and curated collections or categories.

Secure Transactions and Payment Processing: A reliable and secure payment system is crucial for artists. An online art platform that offers secure transactions, escrow services, and smooth payment processing can instill confidence in both artists and buyers.

Portfolio Customization: Artists appreciate the ability to customize their portfolios to showcase their unique style and brand. The platform should provide flexible and visually appealing portfolio layouts, customizable themes, and options to add artist bios and descriptions.

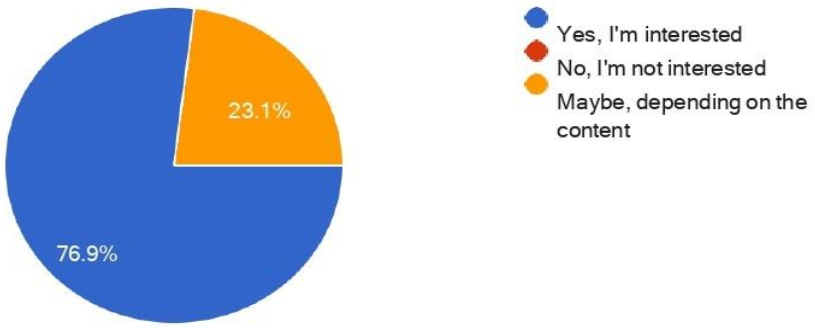
Reaching artwork to the largest possible number of people

have renewal news feed

Community Features: Integration of community forums, live art events, and interactive discussions would facilitate networking with other art enthusiasts, fostering a sense of belonging and shared passion for art.

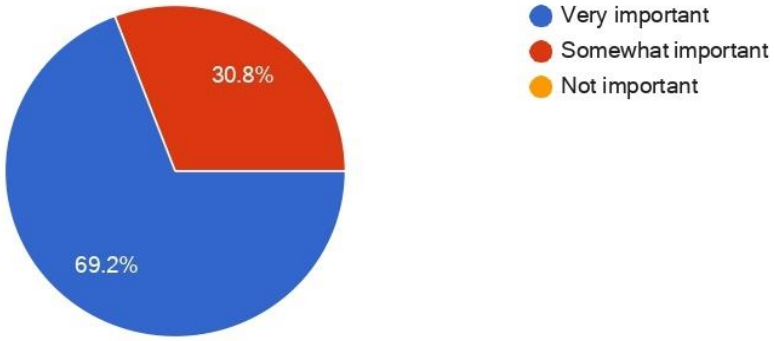
Are you interested in art-related educational content such as tutorials, workshops, or artist interviews?

26 responses



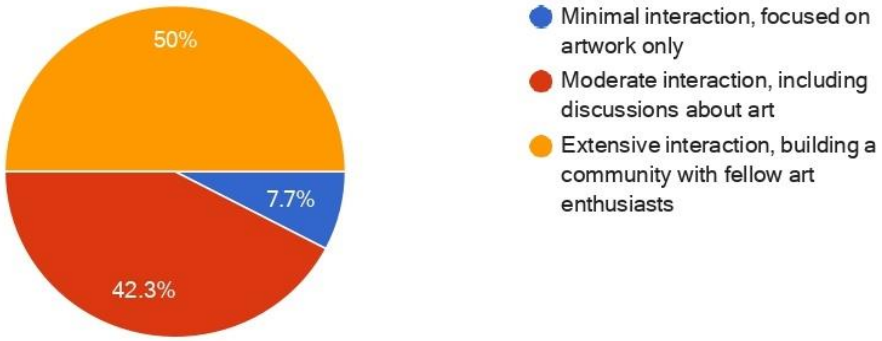
How important is it for the platform to have a mobile app?

26 responses



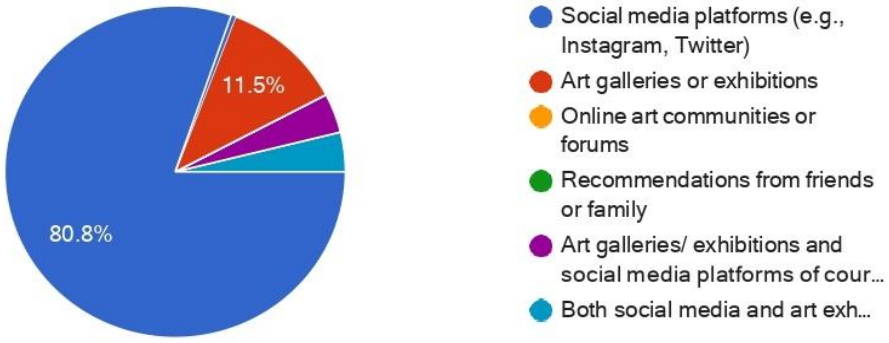
What level of interaction would you like to have with other users on an art platform?

26 responses



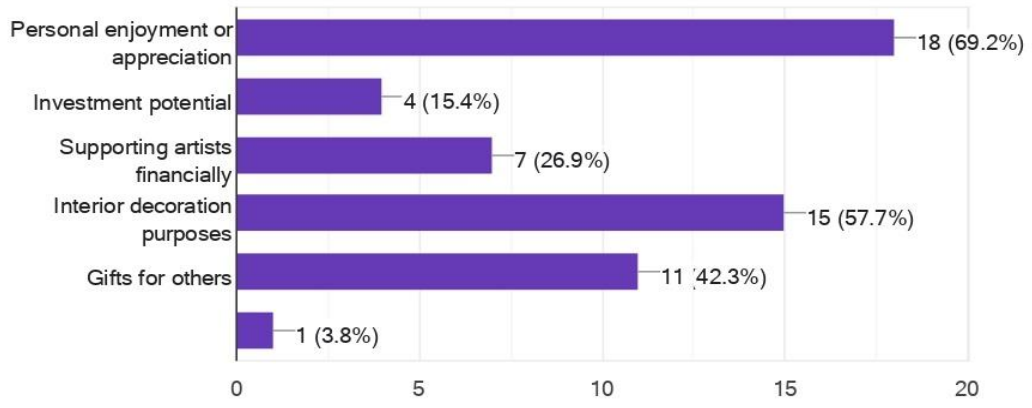
How do you typically discover new artists or artwork?

26 responses



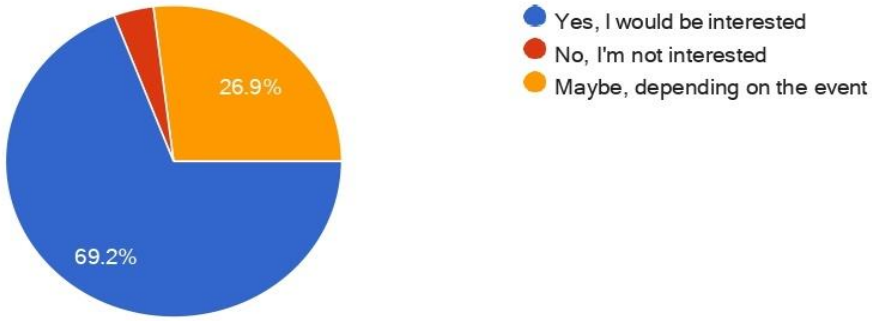
What motivates you to purchase artwork? (Select all that apply)

26 responses



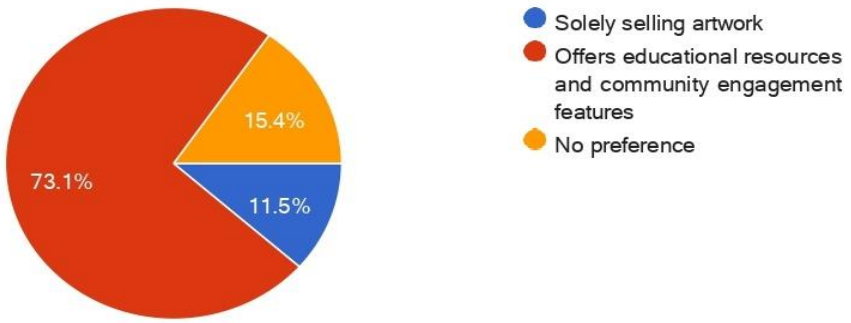
Would you be interested in participating in live art events or auctions hosted on the platform?

26 responses



Would you prefer an art platform that focuses solely on selling artwork, or one that also offers educational resources and community engagement features?

26 responses



What do you believe are the most significant barriers to purchasing art online, and how could a platform like ours address these barriers?

12 responses

.

قابلية عمل كوبونات اوتخفيضات على الاعمال الفنية يحددها الفنان ، حجب السعر للأعمال التى تم بيعها "بحيث ميكونش فى تقييد لسعر شغل الفنان مع الوقت لأن الأعمال الفنيه اسعارها بتزيد بزيادة خبرة الفنان وسنين الشغل وشهرته " ومهم جدا يكون فى option ان للتواصل ومعرفة السعر لأن احيانا بيحدث اتفاقات بين المستخدم والفنان على السعر وبيكون massagesو private السعر بيقا الفنان مش حابب يحط سعر محدد للعمل

وسط غير مؤهل او طبقه اجتماعيه غير مهتمه بالشراء .ف المنصه هتجمع الناس المهتمه بالفن وال يقدروا يقتنوه

المصداقية

Insecurity and trust - i don't know To

find a purchaser

most significant barriers when it comes to purchasing is making sure that the quality of the art piece high, you'll get a real painting not a print, the frame quality of the art piece.

Authenticity and Trust: Buyers may be concerned about the authenticity and quality of artwork when purchasing online. The platform can address this by implementing a strict vetting process for artists and artworks, ensuring that only genuine and high-quality pieces are listed. Additionally, providing detailed artist profiles, certifications of authenticity, and customer reviews can help build trust.

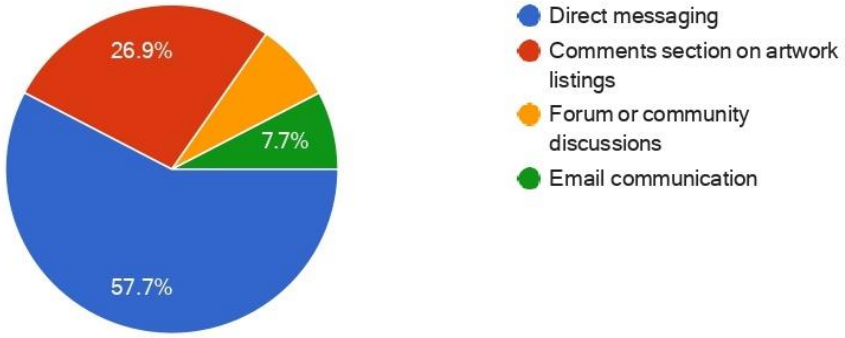
Artwork Representation: Online platforms often struggle to accurately represent the physicality and texture of artworks, which can be crucial for buyers' decision-making. Platforms can overcome this barrier by offering high-resolution images from multiple angles, zoom features, and even virtual reality or augmented reality experiences that allow buyers to visualize the artwork in their own space.

Pricing Transparency: Art buyers may find it challenging to understand the pricing structure, including factors like artist reputation, medium, size, and edition. A platform can address this by providing clear and transparent pricing information, offering price comparisons for similar artworks, and educating buyers about the factors that contribute to the artwork's value.

Returns and Refunds: Uncertainty about returns and refunds policies can make buyers hesitant to make a purchase. Implementing a clear and fair return policy, providing hassle-free returns, and offering refunds in case of dissatisfaction can build trust with buyers.

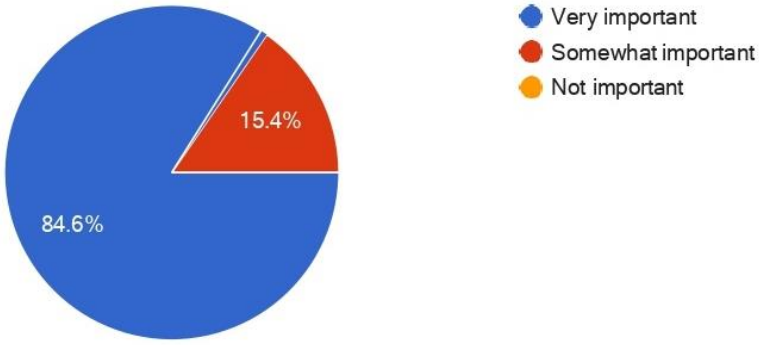
How do you prefer to communicate with artists or sellers on the platform?

26 responses



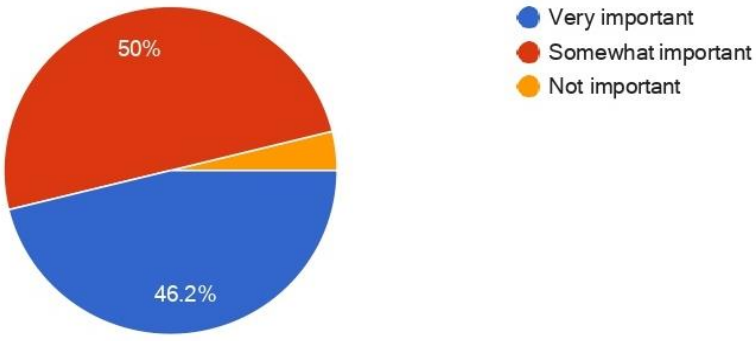
How important is it for the platform to have a diverse range of pricing options for artwork?

26 responses



How important is it for the platform to provide information about the artist's background and inspiration for their work?

26 responses



Can you share any specific features or functionalities from other online platforms that you think our art platform should emulate or improve upon?

15 responses

.

الاعتماد على اظهار اللوحات ف
فيكون فى اوبشن ان الكلاينت يشوف اللوحه على الحيطه او فى برواز painting يعنى لو اللوحه category الفنيه بشكل مناسب لل
فتكون فى امكانية عرضه من كذا جهة dبحيث يقدر يتخيلها بشكل افضل ،او لو العمل نحت 3

Pinterest

تنوع كافة مجالات الفن المعروضة

Reels or short form video like insta reels or tiktok, (buy still keep the photos)

Show the updated artwork to homepage with th

معظم المجالات ذات الدخل السلبي من مواقع تبيع عليها الديزاين بتاعك او شخصية راسمها او اي خدمه اونلاين مش بتدعم الدول
للمصرين و يتم تسويق المركت عالميا هيكون digital products العربية زي مصر فهيبقي حاجة كويسة اوي لو في خدمات لل
الموضوع انجح لان معظم الفنانين هنا مش بيعرفوا يبيعوا خدماتهم

platforms like pinterest allows exploring similar artworks like the one i clicked/showed interest in, it's an excellent feature

Yes i can

I am not aware so much of other platforms

sth like keywords people can u to search up what they're looking for

Robust Search and Filtering Options: Take inspiration from e-commerce platforms that offer advanced search and filtering capabilities. Allow users to refine their searches based on criteria such as art style, medium, size, price range, and artist location. This can help buyers quickly find the artworks that match their preferences.

Personalized Recommendations: Implement recommendation algorithms that suggest artworks to users based on their browsing history, saved favorites, or previous purchases. Personalized recommendations can enhance the user experience, increase engagement, and help buyers discover new artists and styles.

Social Sharing and Integration: Integrate social media sharing buttons and tools that allow users to easily share their favorite artworks with their networks. This can help generate organic exposure for artists and the platform, fostering a sense of community and expanding

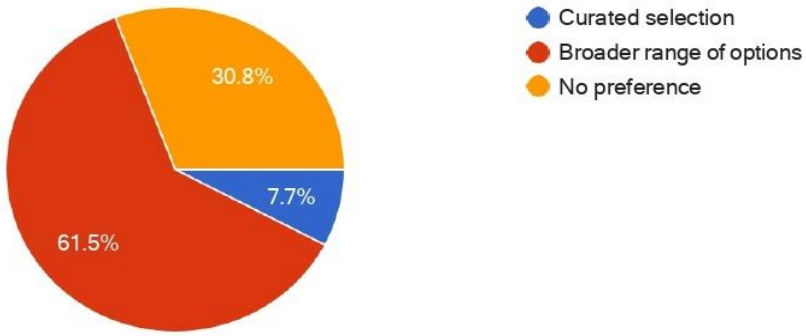
the reach of the artworks.

Virtual Reality (VR) and Augmented Reality (AR) Viewing: Explore the integration of VR and AR technologies to offer immersive viewing experiences. This can allow users to virtually place artworks on their walls using AR or provide virtual gallery tours using VR, enhancing the visualization and engagement for potential buyers.

User Profiles and Portfolios:
Behance: Allows artists to create detailed profiles showcasing their work, project descriptions, tools used, and even collaborative projects.
Instagram: Simple and visually appealing profiles that emphasize images, with options for bio links and story highlights.

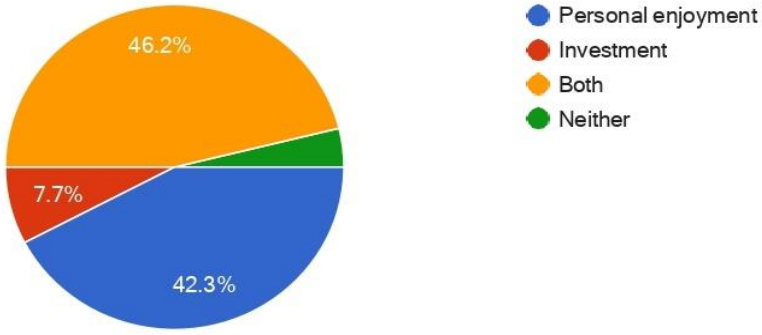
Would you prefer a curated selection of artwork or a broader range of options?

26 responses



Are you more interested in purchasing artwork for personal enjoyment or as an investment?

26 responses



What types of artwork do you feel are currently underrepresented in online art platforms that you would like to see more of?

15 responses

.

Traditional paintings and sculptures

فن التصوير سواء جداري او زيتي

المنحوتات مع التركيز علي اللوحات ايضا

Digital art, mixed art and realistic artwork

اللوحات و ال digital art

interior design

I don't know

Digital art

3D painting and Pen drawings

Caricature

Traditional and Indigenous Art: Online art platforms tend to have a strong focus on contemporary and digital art. Traditional art forms, such as folk art, indigenous art, or classical art styles, may be underrepresented. Increasing the visibility of traditional and indigenous artworks can help preserve cultural heritage and provide exposure for artists working in these mediums.

Experimental and Avant-Garde Art: Online art platforms often cater to more mainstream or commercially viable art styles. There is a need for platforms that embrace and promote experimental, boundary-pushing, and avant-garde artworks that challenge conventional norms and push the boundaries of artistic expression.

Art from Marginalized Communities: Online art platforms should strive to be inclusive and representative of diverse artists and communities. Artworks from marginalized communities, including artists of color, LGBTQ+ artists, artists with disabilities, and artists from underrepresented regions, deserve greater visibility and recognition.

Indigenous and Traditional Art:
Native Art: Art from indigenous cultures around the world, including Native American,

Tools and Technologies used in App

Front-End:

Figma:

Figma is a vector graphics editor and prototyping tool which is primarily web-based, with additional offline features enabled by desktop applications for mac OS and Windows. The feature set of Figma focuses on use in user interface and user experience design, with an emphasis on real-time collaboration.

(URL: https:// figma.com/)

Flutter:

Flutter is Google's open-source UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. It features a rich set of customizable widgets and hot reload functionality, enabling fast development and beautiful, high-performance apps. Flutter is ideal for creating visually appealing and efficient cross-platform applications.

(URL: https://flutter.dev/)

Dart:

Dart is a programming language developed by Google, optimized for building web, server, and mobile applications. It offers a fast development cycle with features like a just-in-time (JIT) compiler for development and an ahead-of-time (AOT) compiler for high-performance production code. Dart is the primary language used for building Flutter applications.

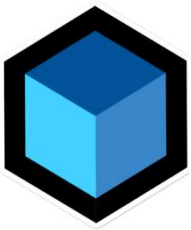
(URL: https://dart.dev/)

Firebase:

Firebase is a comprehensive app development platform by Google that provides a variety of tools and services to help developers build, improve, and grow their apps. It includes features like real-time databases, authentication, cloud storage, and analytics, all designed to streamline backend development and enhance user experiences. Firebase seamlessly integrates with other Google services and third-party tools, making it a popular choice for both web and mobile app developers.

(URL: https://firebase.google.com/)

Bloc:



BLOC (Business Logic Component) is a design pattern in Flutter used to manage state and separate business logic from the UI. It promotes a clear separation of concerns by utilizing streams to handle events and states, making the codebase more maintainable and testable.

(URL: https://pub.dev/packages/bloc)

Pub.dev:

Pub.dev is a central repository for Dart and Flutter packages, offering a comprehensive collection of libraries and tools to enhance development. It provides detailed package information, including descriptions, versions, and documentation, helping developers find and integrate packages into their projects efficiently.

(URL: https://pub.dev/packages)

Back-End:

Runtime Environment:

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project.

(URL: https://nodejs.org/en/)

Package Manager:

Node Package Manager(NPM) is the world's largest software registry. Open source developers from every continent use NPM to share and borrow packages, and many organizations use NPM to manage private development as well.

(URL: https://www.npmjs.com/)

Programming Languages:

JavaScript is a high-level, interpreted programming language primarily used to create interactive and dynamic content on websites. It runs in web browsers and enables client-side scripting to enhance user experiences on the web.

(URL: https://www.javascript.com/)

DataBase Tools:

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents, allowing for dynamic schema design. It supports high performance, scalability, and rich querying capabilities, making it suitable for modern applications. With features like indexing, replication, and sharding, MongoDB ensures efficient data management and high availability. Its powerful aggregation framework enables advanced data processing and analytics. MongoDB is highly adaptable and can be used across various platforms and languages, making it a popular choice for developers. Overall, MongoDB provides a robust solution for handling diverse and evolving data needs.

(URL: https://www.mongodb.com/)

MongoDB Atlas is a fully managed cloud database service that simplifies deploying, managing, and scaling MongoDB databases. It offers automated infrastructure provisioning, backup, and monitoring across multiple cloud providers such as AWS, Azure, and Google Cloud. With features like global distribution, built-in security, and performance optimization, Atlas ensures high availability and scalability for applications. It also includes a rich set of tools for data visualization, analysis, and operational insight. Developers can easily integrate MongoDB Atlas with their applications using various programming languages and frameworks. Overall, MongoDB Atlas provides a seamless and efficient way to manage modern applications' data needs in the cloud.

(URL: https://www.mongodb.com/products/platform/atlas-database/)

AI Model:

Visual Studio Code (Multi-purpose Text Editor):

VS code is used as our basic code editor using its extensions and other features and environments using Microsoft Venv for python as virtual runtime environment.

(URL: https://visualstudio.microsoft.com/)

TensorFlow libraries and Converters (VS Extensions and 3rd Party libraries):

TensorFlow is used as our basic neural network framework via python and jupyter notebooks.

(URL: https://tensorflow.org/)

Keras Library (3rd Party library):

Keras library is used in model saving, loading, converting, and Image pre-processing also, we used it in testing the converted model and original model also, we used it in inference also, we used it in data pipeline intialization, and the extraction of confusion matrix elements (entropy, accuracy, recall) and the core basis of our sequential feed forward neural network.

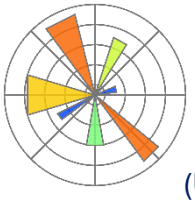
(URL: https://keras.io/)

NumPy Library (3rd Party library):

NumPy Arrays are used to Pre-process images' data and to break it down into arrays of pixel values and to resize and normalize the shape and data of tensors (multi-dimensional arrays), also we made training, validation and testing data batches using and the NumPy iteration module also, we used the Array manipulation in order to make predictions and batches (expand dims).

(URL: https://numpy.org/)

MatplotLib Library and PyPlot module (3rd party library):



MatplotLib is used to Track the performance of our convlutional neural network and evaluate it over time while training the model (data visualization tool).

(URL: https://matplotlib.org/)

Open CV Library (3rd Party library):

Open CV library is used for image scanning, filtering and dimensions manipulation and to handle the reading process in batch passing during training.

(URL: https://opencv.org/)

OS python library (Built-in Python module):

OS python library is used for Folder scanning and creating the data pipeline directory.

(URL: https://docs.python.org/3/library/os.html)

Stack Overflow (Devs Community Website):

Stack Overflow is used for researching missing code parts in our inference and deployment and it has been a great help for our model testing.

(URL: https://stackoverflow.com/)

Diagrams Tools:

Edraw max:

Edraw Max is like a digital toolbox for creating visual representations of your ideas. It lets you drag-and-drop shapes and elements to build all sorts of diagrams, from flowcharts and mind maps to floor plans and even electrical schematics. With hundreds of templates and the option for AI-powered assistance, Edraw Max can help you create professional-looking diagrams quickly and easily.

(URL: https://www.edrawmax.com/)

Draw.io:

Draw.io is a free online tool that lets you create all sorts of diagrams. Think of it as a digital whiteboard where you can drag-and-drop shapes and arrows to build things like flowcharts, mind maps, or even network diagrams. It works right in your web browser, so there's nothing to download or install. Plus, it lets you share your creations with others for real-time collaboration.

(URL: https://www.drawio.com/)

Chapter 3

(Analysis and Design)

This chapter discusses the analysis and design phases of the project. It shows the different diagrams built to show how the system works, what does it consists of, and whom are the users.

Use Case Diagram:

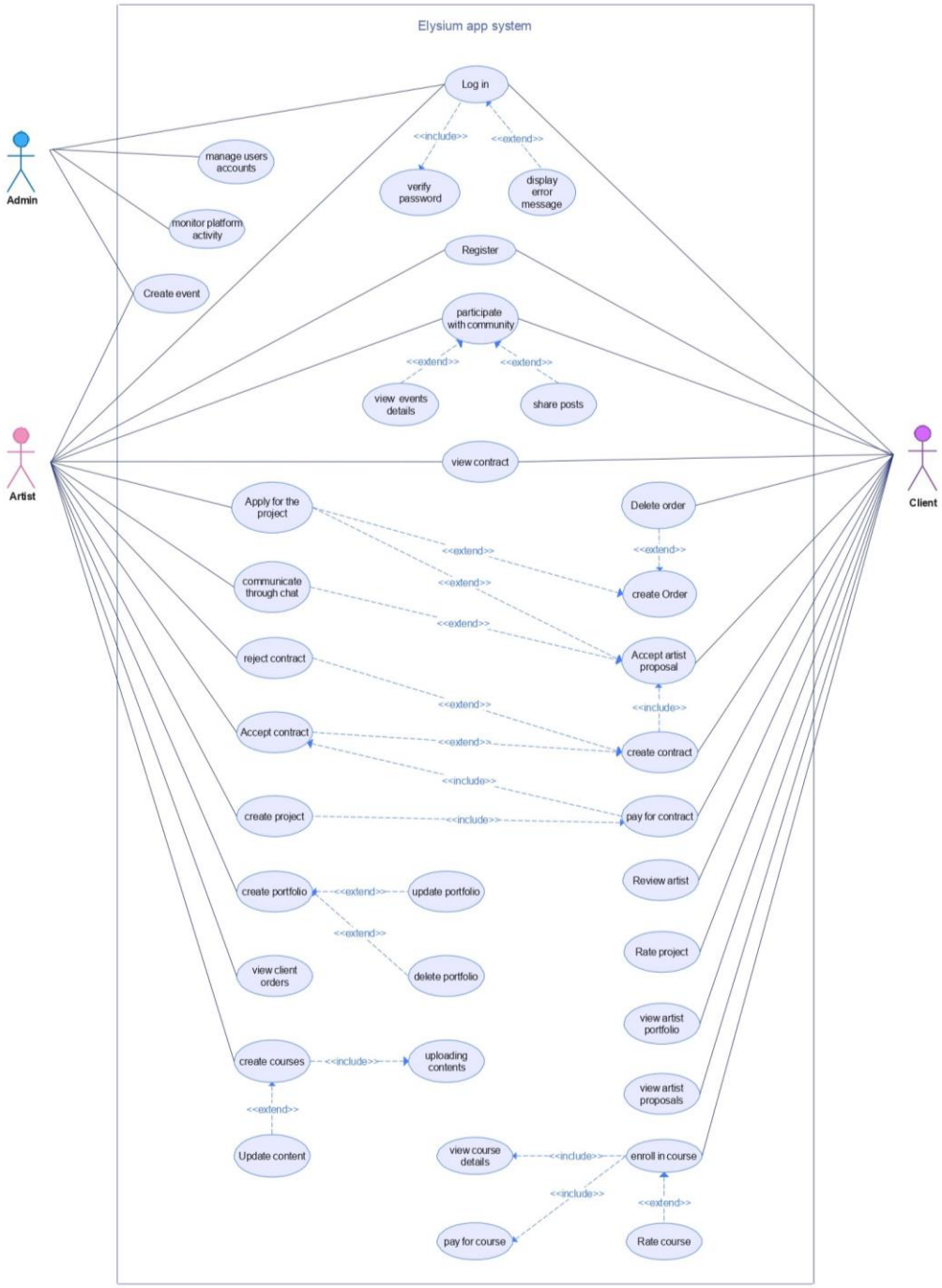


Figure (1) - Use Case Diagram

Actors:

1. **Admin:** Responsible for managing users, reports, and events. The admin receives artist and customer wishes.
2. **Artist:** Users who can register, log in, participate with the community, view event details, share posts, create event, apply for projects, communicate with client through chat, reject contracts, accept contracts, create projects, create portfolios, update portfolios, delete portfolios, view client orders, view contracts, create courses, upload course content, and update course content.
3. **Client:** Users who can register, log in, participate with the community, view event details, share posts, create orders, accept artist proposals, create contracts, pay for contracts, review artists, rate projects, view artist portfolios, view artist proposals, enroll in courses, view course details, pay for courses, and rate courses.

Use Cases:

1. **Register:** New users can register if it's their first time visiting mobile application by filling in some fields; their account is created successfully.
2. **Log In:** Users can log in to mobile application to access all activities by entering their email and password.
3. **Display error message:** An error message is displayed if the user enters incorrect data.
4. **Verify password:** Checking if the password user entered matches the one stored for his account.
5. **Manage users accounts:** Admin manages user accounts and all activities related to them.
6. **Monitor platform activity:** Admin monitors all activities and processes that occur in the application.
7. **Create event:** Admin and artist can create an event whether physical or online, allowing others to attend and benefit.
8. **Participate with Community:** Users can engage with the community by viewing event details and sharing posts.
9. **View Events Details:** Users can see detailed information about various events.
10. **Share Posts:** Users can share posts within the community to interact with others.
11. **Create Order:** Clients can place new orders for products.
12. **Delete Order:** Clients can delete orders they have created if they are no longer needed.

13. **Accept Artist Proposal:** Clients can accept service proposals from artists.
14. **Create Contract:** Clients can draft and create new contracts for products.
15. **Pay for Contract:** Clients can complete payments for services rendered.
16. **Review Artist:** Clients can provide reviews and feedback for artists they have worked with.
17. **Rate Project:** Clients can rate projects they have received based on quality and satisfaction.
18. **View Artist Portfolio:** Clients can browse through artists' portfolios to evaluate their work.
19. **View Artist Proposal:** Clients can review proposals submitted by artists.
20. **View Contract:** Clients can review the details of contracts they have created.
21. **Enroll in Course:** Clients can sign up for courses they are interested in.
22. **View Course Details:** Clients can see detailed information about available courses.
23. **Pay for the Course:** Clients can make payments to enroll in courses.
24. **Rate Course:** Clients can rate courses they have completed to provide feedback.
25. **Apply for the Project:** Artists can apply for available projects that match their skills.
26. **Communicate through Chat:** Artists and clients can send and receive messages.
27. **Reject Contract:** Artists can reject contracts that do not meet their terms.
28. **Accept Contract:** Artists can accept contracts offered to them.
29. **Create Project:** Artists can initiate new projects and manage them.
30. **Create Portfolio:** Artists can create portfolios to showcase their work.
31. **Update Portfolio:** Artists can update their portfolios with new achievements.
32. **Delete Portfolio:** Artists can remove their portfolios if they choose to.
33. **View Client Orders:** Artists can access and review orders placed by clients.
34. **View Contract:** Artists can review the details of contracts they are involved in.
35. **Create Courses:** Artists can design and create new educational courses.
36. **Upload Contents:** Artists can upload new content related to their work or courses.
37. **Update Content:** Artists can update existing content they have uploaded.

Activity Diagram:

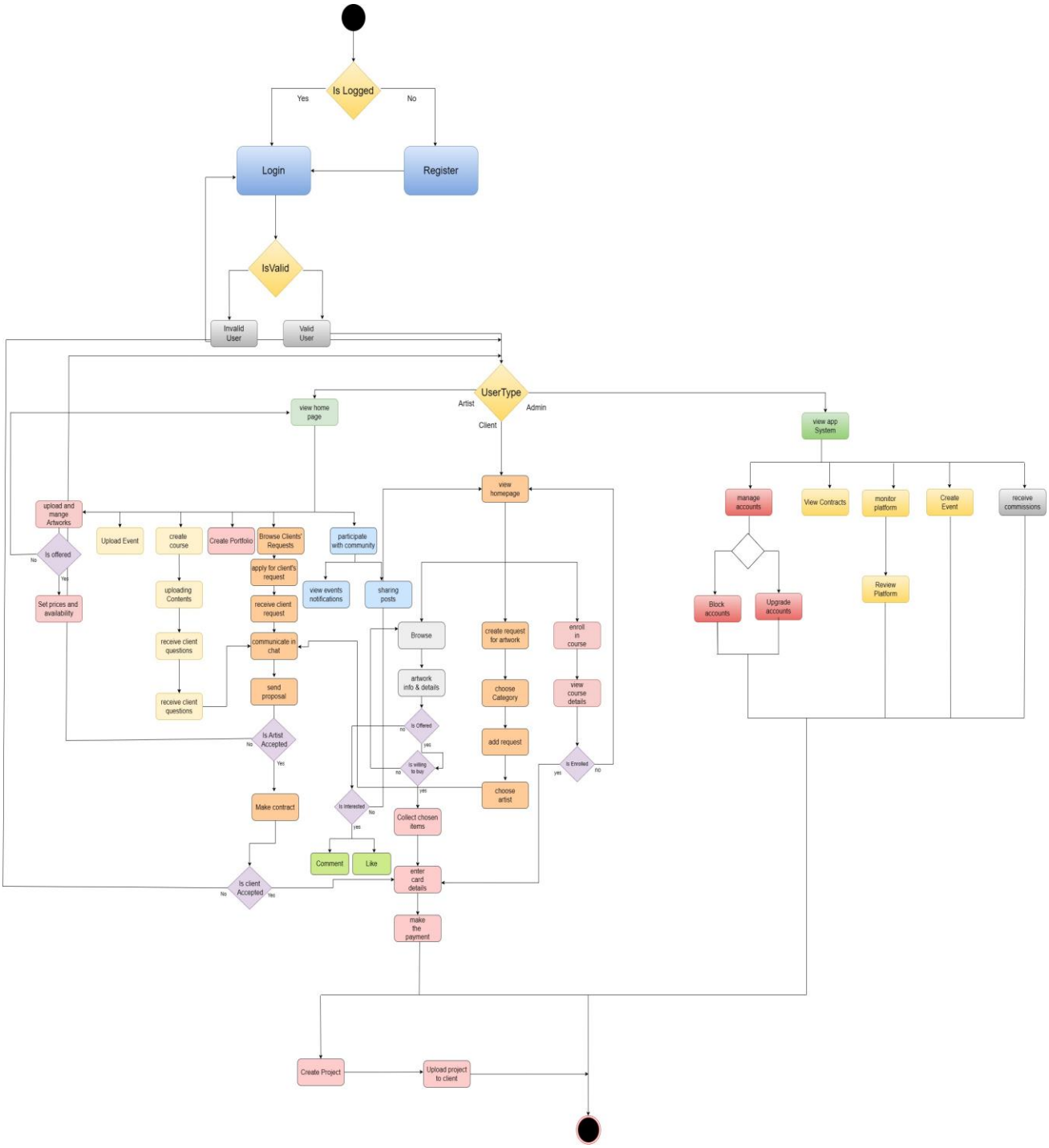


Figure (2)-Activity Diagram

This activity diagram represents a user interaction flow for a web application platform with different roles: Artist, Client, and Admin. Each user type has a specific set of actions they can perform based on their permissions. Below is a detailed explanation of each process, decision point, and interaction:

Initial Login/Register Process:

1. **Is Logged?** (Decision): Checks if the user is already logged in.
    - **Yes**: User is logged in.
        - **Login** (Action): User proceeds to log in.
    - **No**: User is not logged in.
        - **Register** (Action): New user proceeds to register.
2. **Is Valid** (Decision): Validates user credentials during login.
    - **Invalid User**: User credentials are not valid.
    - **Valid User**: User credentials are valid.

User Type Decision:

3. **User Type** (Decision): Determines the type of user after successful login.
    - **New User**: User is new and needs to fill out initial information.
    - **Artist**: User is an artist.
        - **view homepage** (Action): Artist views the homepage.
    - **Client**: User is a client.
        - **view homepage** (Action): Client views the homepage.
    - **Admin**: User is an admin.
        - **view app system** (Action): Admin views the app system.

Artist Flow:

5. **Artist Homepage Actions**:
    - **participate with community** (Action):
        - Artist can view events notified by other artists.
        - Artist can share posts related to his artwork.

- **Create Portfolio** (Action): Artist creates a portfolio.
- **Browse Clients' Projects** (Action): Artist browses clients' projects and offers services.
    - Artist applies for one of the clients' requests
    - He receives one of the client's requests that was applied before
    - The client can communicate with the artist to make a contract
    - The artist sends a proposal to get a full picture of what the client desires.
    - If the client approved the proposal, he creates the contract and then this contract get sent to the artist to check on it.
    - If the contract is accepted by the artist, the artist and the client insert the card details and then the payment process starts to activate by charging the client at the price mentioned in the contract.
    - Once the payment process is successful the artist begins to work on the project.
    - The moment the artist finishes his project, they upload it to the client and receive his payment from the application.
- **create event** (Action): Artist creates an event.
- **upload and showcase Artwork** (Action): Artist uploads and showcases artwork.
    - If the artist wants to upload his artwork for sale, he sets a suitable price for it.
    - If it's just for show he uploads it and then the app takes him back to the home page.
- **create course (Action):**
    - The artist begins to the upload the contents of the course (video link, description, images, etc....)
    - After Uploading the course, he receives some questions from the students just enrolled in the course.
    - Then he communicates to every student in the chat to answer their questions and manage their progress.

Client Flow:

6. **Client Homepage Actions**:
    - **create request for artwork** (Action): Client creates a request for artwork.
        - **choose Category** (Decision): Client chooses a category for the artwork to choose among the artists specified for this category.
        - **choose Artist** (Decision): Client chooses among the artists just applied for his project.
        - Once the client finds the suitable artist, he communicates with him through the chat.
        - Then he decides to accept the proposal or not, and if he did so he makes the contract and it get sent to artist to make a decision to accept it or not.
        - Once the artist accepts the contract, the app charges the client and holds the funds. The artist begins to work on the project once the payment process has been successful and uploads it to the client upon completion.
    - **Browse** (Action): Client browses available artworks and artists.
        - **Artwork info & details** (Action): Client views detailed information about the artwork.
        - The client checks whether the artwork is for sale and willing to buy it.
        - If he's willing to buy the artwork, he collects it in a collector as we can call it "cart" to view all the items he's just chosen before.
        - He inserts his card details and then get charged by the total price of all the items he just purchased.
        - If none of these conditions is achieved the client go back to browse for more artworks.
    - **Enroll in Course** (Action): Client finds an interesting course and is willing to enroll in it.
        - The client starts to view the details of this course
        - Client makes the payment after inserting his card details and becomes a student for the author of the course.

Admin Flow:

7. **Admin System Actions**:
    - **manage accounts** (Action): Admin manages user accounts.
        - **Block** (Action): Admin blocks an account.
        - **Upgrade accounts** (Action): Admin upgrades user accounts.
    - **view contracts** (Action): Admin views contracts.
    - **monitor platform** (Action): Admin monitors the platform activities.
        - **Review Platform** (Action): Admin reviews the overall platform performance.
    - **Create Event** (Action): Admin creates an event.
    - **Receive commissions** (Action): Admin receives commission details.

Additional Actions and Links:

- Multiple actions and decisions are interconnected, allowing users to navigate back and forth based on their choices.
- The diagram contains loops where users can return to previous actions or decisions if certain conditions are met or additional actions are required.

Entity Relationship Diagram (ERD):

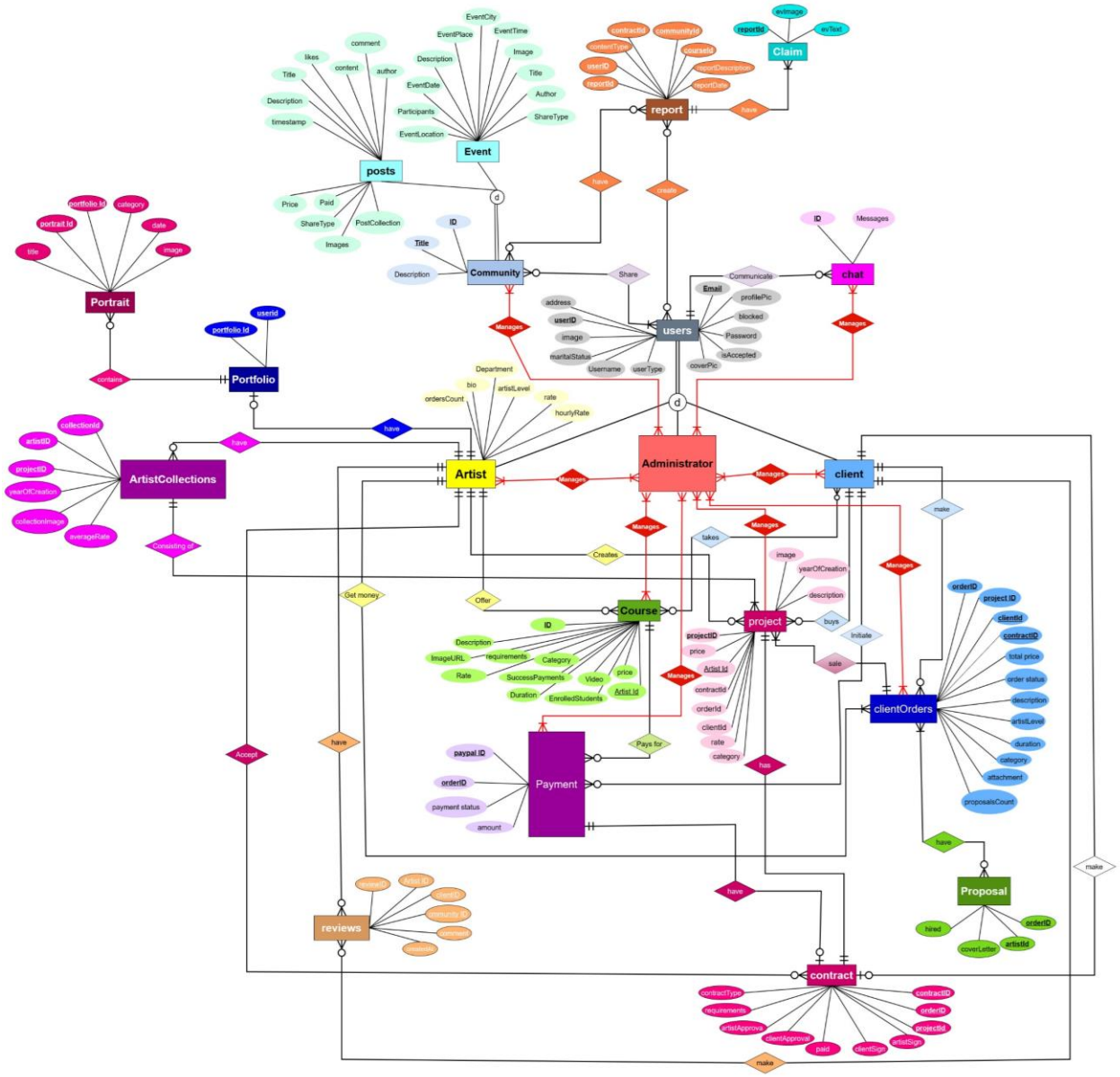


Figure (3) - ERD Diagram

ERD Business Roles:

User & Course:

- **User must join one or more Courses.**
    - Each user is encouraged to enroll in various courses to enhance their skills.
    - Users can choose courses based on their interests and skill levels.
- **Course may be joined by one or more Users.**
    - Courses are open for multiple users to join, promoting collaborative learning.

User & Chat:

- **User may participate in one or more Chats.**
    - Users engage in conversations through the chat feature, facilitating communication and collaboration.
- **Chat must involve one or more Users.**
    - Each chat session includes multiple users, promoting interactive discussions.

User & Review:

- **User may write one or more Reviews.**
    - Users provide feedback and ratings for courses and services, sharing their experiences.
- **Review must be written by one User.**
    - Each review is authored by a single user, reflecting their personal opinion and evaluation.

User & Report:

- **User may file one or more Reports.**
    - Users can report issues or inappropriate content to administrators.
- **Report must be filed by one User.**
    - Each report is submitted by a single user, detailing the specific issue.

Artist & Portfolio:

- **Artist must have one or more Portfolios.**
    - Artists create portfolios to showcase their work and highlight their skills and achievements.
- **Portfolio belongs to one Artist.**
    - Each portfolio is associated with a single artist, providing a personalized space for their work.

Artist & Artist Collection:

- **Artist may have one or more Artist Collections.**
    - Artists can organize their artworks into different collections based on themes or projects.
- **Artist Collection belongs to one Artist.**
    - Each collection is curated by a single artist, reflecting their unique style and vision.

Artist & Proposal:

- **Artist may submit one or more Proposals.**
    - Artists can submit proposals for projects or commissions to potential clients.
- **Proposal must be submitted by one Artist.**
    - Each proposal is created and owned by a single artist, outlining their approach and cost for the project.

Artist & Event:

- **Artist may participate in one or more Events.**
    - Artists can join events such as exhibitions, workshops, and competitions.
- **Event may have one or more participating Artists.**
    - Events are designed to include multiple artists, fostering community interaction and exposure.

Artist & Community Post:

- **Artist may create one or more Community Posts.**
    - Artists can share updates, artworks, and ideas with the community through posts.
- **Community Post must be created by one Artist.**
    - Each post is attributed to a single artist, encouraging individual expression and communication.

Artist & Course:

- **Artist may create or enroll in one or more Courses.**
    - Artists can both teach and learn by creating courses or enrolling in existing ones.
- **Course may be created or enrolled by one or more Artists.**
    - Courses are developed and attended by various artists, promoting knowledge sharing and skill development.

Artist & Project:

- **Artist may participate in one or more Projects.**
    - Artists collaborate on various projects, working together to create unique pieces or complete client requests.
- **Project must involve one or more Artists.**
    - Each project requires the involvement of at least one artist, ensuring that multiple talents can come together.

Artist & Contract:

- **Artist must fulfill one or more Contracts.**
    - Artists are responsible for delivering the work agreed upon in the contracts.
- **Contract must be fulfilled by one or more Artists.**
    - Each contract involves artists who ensure that the project's requirements are met.

Artist & Claim:

- **Artist may file one or more Claims.**
    - Artists can file claims for issues such as copyright infringement or disputes.
- **Claim must be filed by one Artist.**
    - Each claim is submitted by a single artist, detailing the specific issue.

Artist & Payment:

- **Artist must receive one or more Payments.**
    - Artists are compensated for their work upon the completion of projects or milestones.
- **Payment must be made to one Artist.**
    - Each payment is directed to a single artist, ensuring they are rewarded for their contributions.

Client & Order:

- **Client may place one or more Orders.**
    - Clients can request services or artworks from artists through orders.
- **Order must be placed by one Client.**
    - Each order is initiated by a single client, detailing their requirements and expectations.

Client & Proposal:

- **Client may receive one or more Proposals.**
    - Clients receive proposals from artists, outlining project details and costs.
- **Proposal must be addressed to one Client.**
    - Each proposal is directed to a specific client, facilitating targeted and personalized offers.

Client & Contract:

- **Client may have one or more Contracts.**
    - Clients enter into contracts with artists to formalize agreements for projects.

- **Contract must involve one Client.**
    - Each contract is associated with a single client, specifying the terms and conditions.

Client & Review:

- **Client may leave one or more Reviews.**
    - Clients can provide feedback on the artists' work and their overall experience.
- **Review must be left by one Client.**
    - Each review is authored by a single client, reflecting their personal assessment.

Client & Payment:

- **Client must make one or more Payments.**
    - Clients pay for the services or artworks received from the artists.
- **Payment must be made by one Client.**
    - Each payment is made by a single client, covering the agreed-upon costs.

Event & Administrator:

- **Event must be organized by one Administrator.**
    - Administrators are responsible for planning and managing events, ensuring smooth execution.
- **Administrator may organize one or more Events.**
    - Administrators handle multiple events, fostering a vibrant and active community.

Report & Administrator:

- **Report must be handled by one Administrator.**
    - Administrators review and address reports, maintaining community standards and resolving issues.
- **Administrator may handle one or more Reports.**

- Administrators manage multiple reports, ensuring thorough and timely responses.

Administrator & Community:

- **Administrator must manage one or more Community Posts.**
    - Administrators oversee community posts, ensuring they adhere to guidelines and promoting a positive environment.
- **Community Post must be managed by one Administrator.**
    - Each community post is monitored by an administrator to maintain quality and relevance.

Claim & Administrator:

- **Claim must be reviewed by one Administrator.**
    - Administrators are responsible for reviewing and resolving claims filed by artists.
- **Administrator may review one or more Claims.**
    - Administrators handle multiple claims, ensuring fair and timely resolutions.

Community Post & Comment:

- **Community Post may have one or more Comments.**
    - Posts can receive feedback and discussions through comments from other users.
- **Comment must belong to one Community Post.**
    - Each comment is tied to a specific post, ensuring relevant and contextual discussions.

Community Post & Report:

- **Community Post may have one or more Reports.**
    - Users can report inappropriate or problematic posts to administrators for review.
- **Report must be related to one Community Post.**
    - Each report targets a specific post, helping maintain community standards.

Course & Lesson:

- **Course must have one or more Lessons.**
    - Courses are structured into lessons, providing a clear and organized learning path.
- **Lesson must belong to one Course.**
    - Each lesson is a component of a course, ensuring cohesive content delivery.

Course & Review:

- **Course may have one or more Reviews.**
    - Users can leave reviews for courses, providing feedback and ratings.
- **Review must be related to one Course.**
    - Each review is specific to a single course, helping others gauge its quality and relevance.

Portfolio & Artwork:

- **Portfolio may contain one or more Artworks.**
    - Portfolios showcase a collection of artworks, representing an artist’s body of work.
- **Artwork must belong to one Portfolio.**
    - Each artwork is part of a portfolio, providing context and coherence to the artist's creations.

Project & Client:

- **Project may be requested by one or more Clients.**
    - Clients can commission projects based on their needs and preferences.
- **Client must request one or more Projects.**
    - Each project request is made by a client, detailing their specific requirements.

Project & Proposal:

- **Project may have one or more Proposals.**
    - Artists submit proposals to clients detailing their approach, timelines, and costs for the project.
- **Proposal must belong to one Project.**
    - Each proposal is tied to a specific project, providing clarity and structure for the agreement.

Project & Review:

- **Project may receive one or more Reviews.**
    - Completed projects can be reviewed by clients, highlighting the quality and success of the collaboration.
- **Review must be related to one Project.**
    - Each review pertains to a specific project, providing targeted feedback.

Proposal & Contract:

- **Proposal may lead to one or more Contracts.**
    - Once a proposal is accepted, it results in a formal contract outlining the terms and conditions.
- **Contract must be based on one Proposal.**
    - Each contract stems from a specific proposal, ensuring alignment between the artist's offerings and the client's expectations.

ERD Schema:

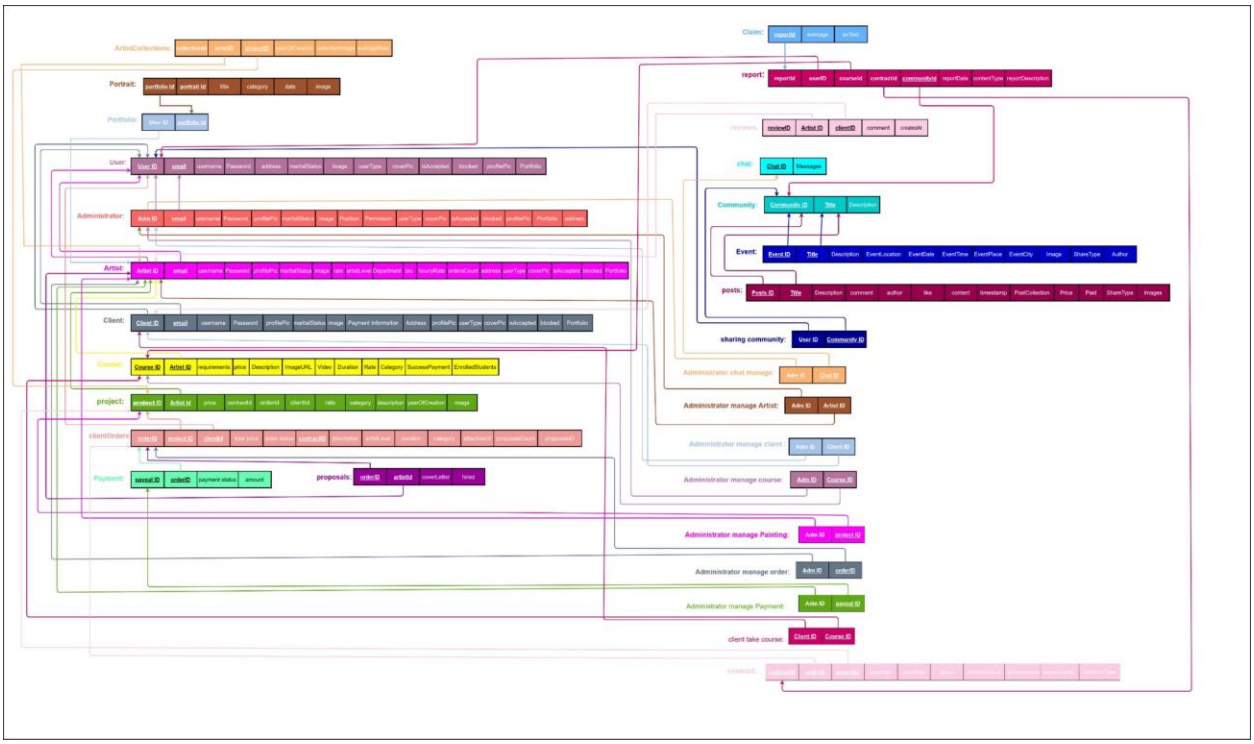


Figure (4) - ERD Schema

The database schema of our application (Elysium) represents a comprehensive system involving various entities and their relationships. Key entities include User, Administrator, Artist, and Client, each with attributes like usernames, passwords, emails, portfolios, rates, client orders, and payment statuses. The schema depicts relationships for tasks such as managing courses, projects, payments, and community events, highlighting interactions and data flow within the system. It supports functionalities for user account management, payment handling, project coordination, and communication, with administrators overseeing these aspects. Overall, it provides an efficient structure for storing and managing data related to users, clients, artists, and administrative tasks.

Relational Model:

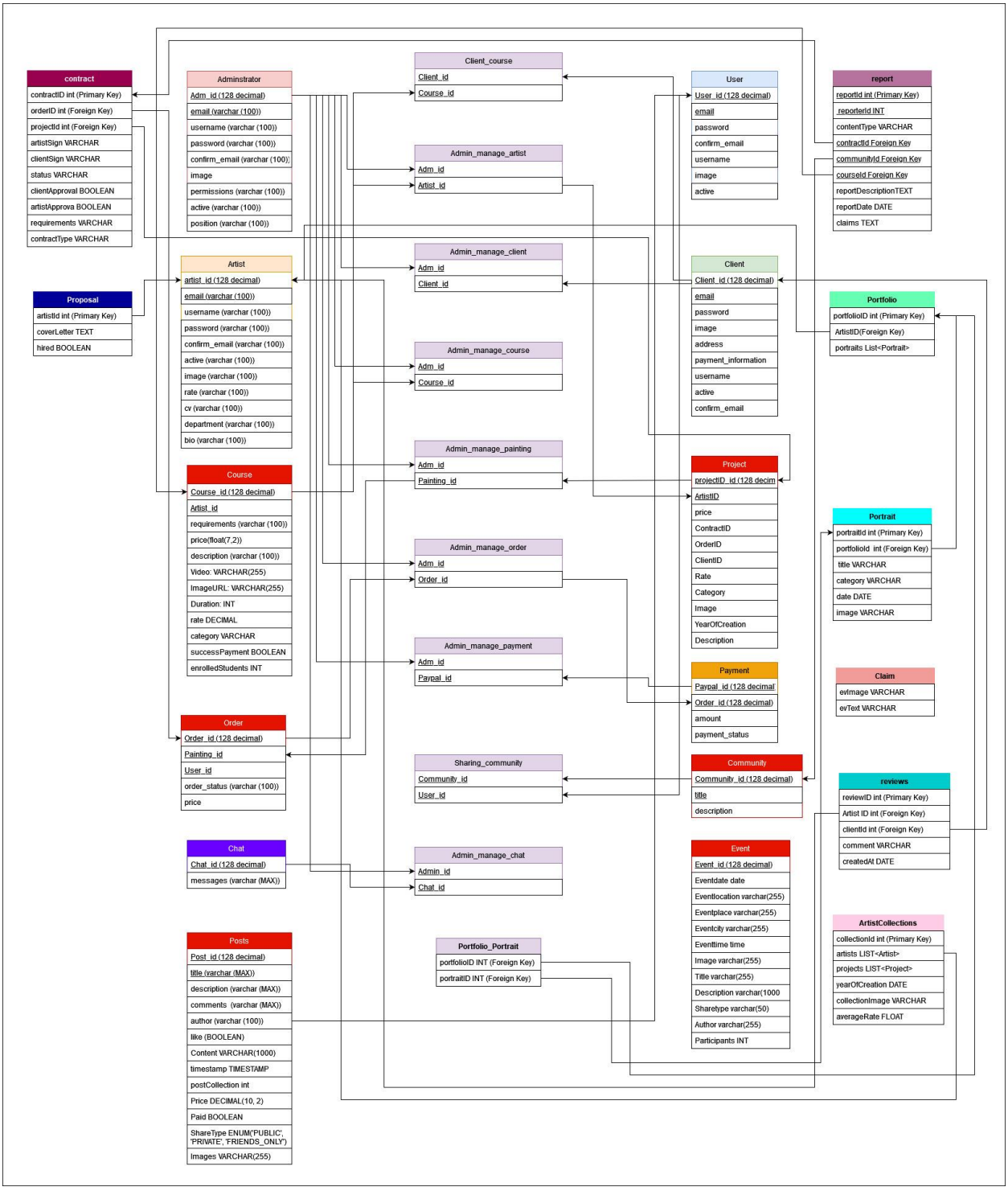


Figure (5) - Relational Model

The relational model diagram for our application details several interconnected tables storing various types of data.

User Management:

- **Administrator table**: Profiles and credentials for administrators.
- **Client table**: Profiles and credentials for clients.
- **Artist table**: Profiles and credentials for artists.
- **User table**: General user profiles.

Projects and Artworks:

- **Contract table**: Links contracts with clients, artists, and projects.
- **Proposal table**: Holds artist proposals.
- **Project table**: Stores projects and links them to artists and clients.
- **Portrait table**: Stores portraits created by artists.

Client Activity:

- **Order table**: Manages client orders, connecting clients and artists.

Community:

- **Community table**: Captures community interactions through posts.
- **Chat table**: Captures chat interactions among users.

Content and Learning:

- **Course table**: Represents courses and enrolled users.
- **Post table**: Stores user posts in community interactions.

Feedback and Management:

- **Report table**: Handles user reports regarding issues or inappropriate content.
- **Portfolio table**: Handles artist portfolios showcasing their work.
- **Claim table**: Contains user claims, such as copyright issues.
- **Review table**: Contains user reviews linked to courses, artists, and projects.

Relationships:

- **User & Course**: Users can enroll in multiple courses. Each course can have multiple users.
- **User & Chat**: Users can participate in multiple chats. Each chat involves multiple users.
- **User & Review**: Users can write multiple reviews. Each review is authored by a single user.
- **User & Report**: Users can file multiple reports. Each report is filed by a single user.
- **Artist & Portfolio**: Each artist can have multiple portfolios. Each portfolio belongs to a single artist.
- **Artist & Artist Collection**: Each artist can have multiple collections. Each collection belongs to a single artist.
- **Artist & Proposal**: Each artist can submit multiple proposals. Each proposal is submitted by a single artist.
- **Artist & Event**: Artists can participate in multiple events. Each event can have multiple participating artists.
- **Artist & Community Post**: Artists can create multiple community posts. Each post is created by a single artist.
- **Artist & Course**: Artists can create or enroll in multiple courses. Each course can have multiple artists.
- **Artist & Project**: Artists can participate in multiple projects. Each project involves at least one artist.
- **Artist & Contract**: Artists must fulfill multiple contracts. Each contract involves one or more artists.
- **Artist & Claim**: Artists can file multiple claims. Each claim is filed by a single artist.
- **Artist & Payment**: Artists receive multiple payments. Each payment is made to a single artist.
- **Client & Order**: Clients can place multiple orders. Each order is placed by a single client.
- **Client & Proposal**: Clients receive multiple proposals. Each proposal is directed to a single client.

- **Client & Contract**: Clients have multiple contracts. Each contract involves a single client.
- **Client & Review**: Clients can leave multiple reviews. Each review is left by a single client.
- **Client & Payment**: Clients make multiple payments. Each payment is made by a single client.
- **Event & Administrator**: Events are organized by an administrator. Each administrator can organize multiple events.
- **Report & Administrator**: Reports are handled by an administrator. Each administrator can handle multiple reports.
- **Administrator & Community**: Administrators manage community posts. Each post is managed by an administrator.
- **Claim & Administrator**: Claims are reviewed by an administrator. Each administrator can review multiple claims.
- **Community Post & Comment**: Community posts can have multiple comments. Each comment belongs to a single post.
- **Community Post & Report**: Community posts can have multiple reports. Each report is related to a single post.
- **Course & Review**: Courses can have multiple reviews. Each review is related to a single course.
- **Portfolio & Portrait**: Portfolios may contain multiple portraits. Each portrait belongs to a single portfolio.
- **Project & Client**: Projects may be requested by multiple clients. Each project request is made by a single client.
- **Project & Proposal**: Projects can have multiple proposals. Each proposal belongs to a single project.
- **Project & Review**: Projects can receive multiple reviews. Each review is related to a single project.
- **Proposal & Contract**: Proposals can lead to multiple contracts. Each contract is based on a single proposal.

Class Diagram:

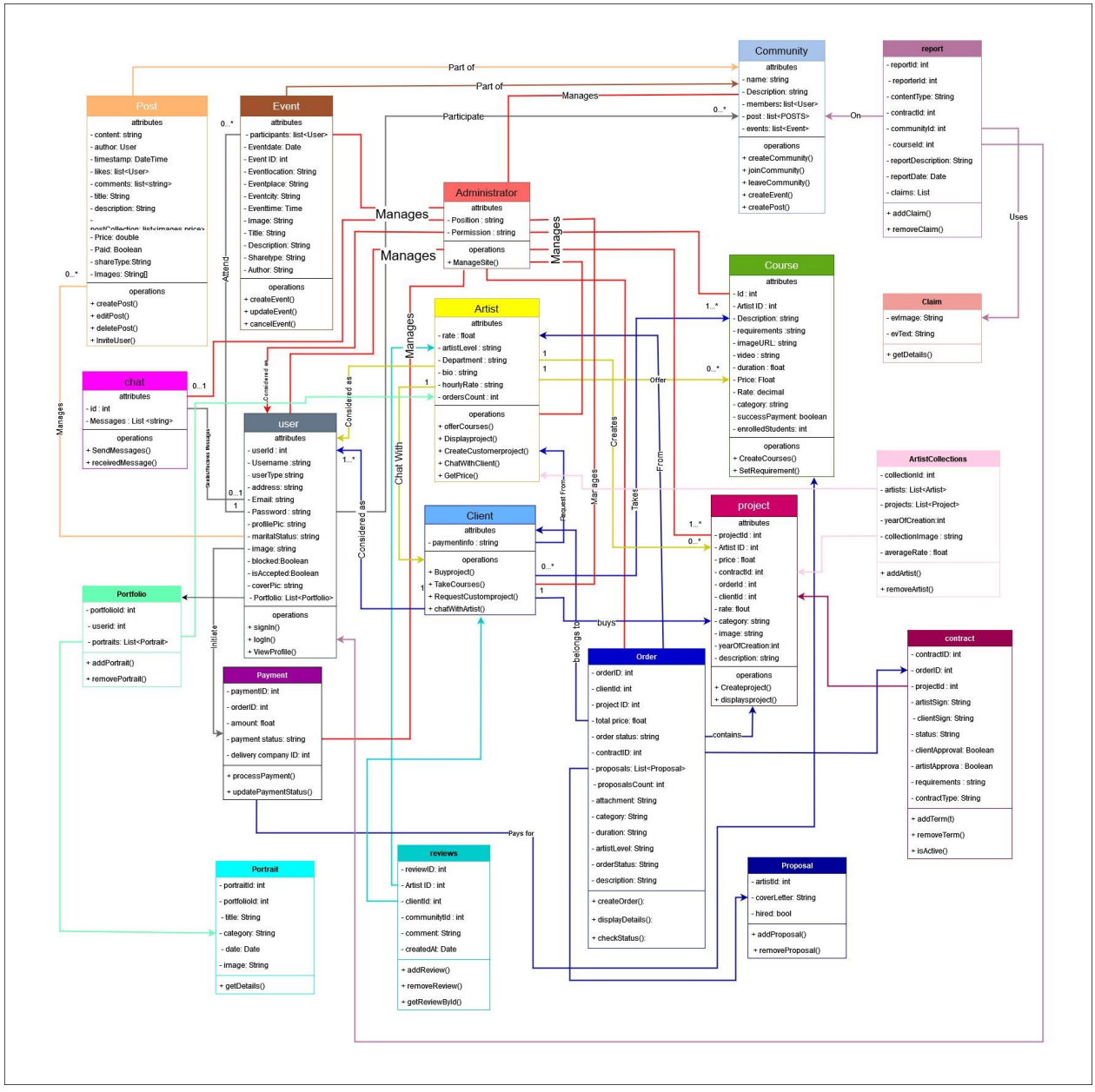


Figure (6) - Class Diagram

The class diagram provides a detailed and structured overview of our application's entities and their interactions.

Supports User Roles: Visualizes how users can be artists, clients, or administrators.

Community & Management: Facilitates artist-client engagements, community participation, and administrative management.

Clear Functionality: Illustrates relationships and dependencies among components for a clear understanding of the system's functionality and operational flow.

Planning & Maintenance: Helps plan and maintain the system by showcasing user roles and their management.

Key Processes: Highlights key processes like portfolio creation, project management, payment processing, and handling reviews and reports.

Interconnected System: Ensures all aspects of user interactions and administrative tasks are well-defined and interconnected.

AI Model:

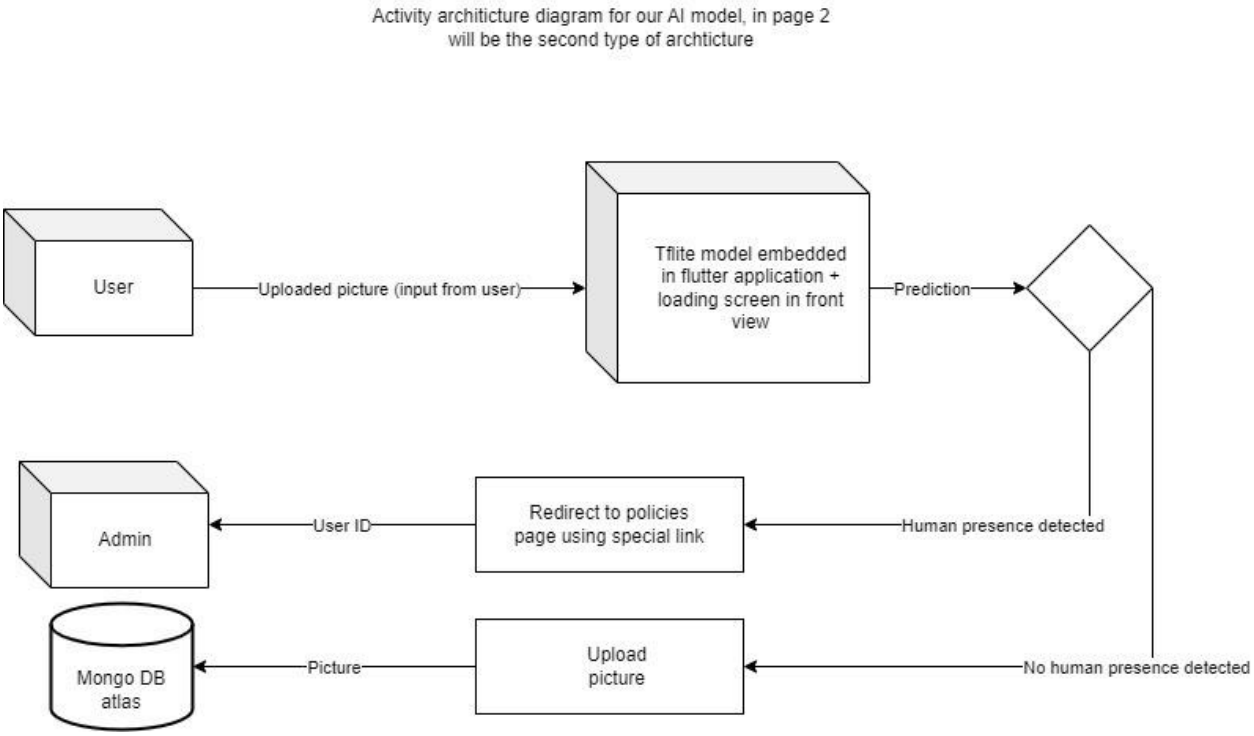


Figure (7) - AI Model

We coded our AI model using TensorFlow framework from google, and we chose in-in Deployment model, briefly we made a convlutional neural network that recognizes human presence in photos since we are impressionists platform, and we have policies that forbid human presence in artwork also you must upload the artwork strictly in one of these formats (jpg, jpeg, png , bmp) so the neural network can function properly and recognize human presence in the uploaded artworks, basically it's binary classification model which recognizes by predictions based on average inference of RGB color pixel values and convlutional filters of conv2D layer, and the in-in model we used was a conversion from h5 keras model to tflite model in order to use it in edge machine deployment (smartphones) so, we use the used the tflite model because it's way easier to integrate than other models, secure, light-weighted, and can be integrated directly into our flutter application, without jerking user data around between servers which may have risks of infringement and IP theft, so it is the easiest and safest model and we made inference that makes the model do its job when user taps the upload button automatically, and shows the user loading screen, and if there was human presence it will direct user to policies page and send the ID to the administrator to take proper actions otherwise, It will let it pass gloriously to our MongoDB atlas (our database) and this is the main point of deploying in this way.

Chapter 4

(Feasibility and (lean) Business Model Canvas)

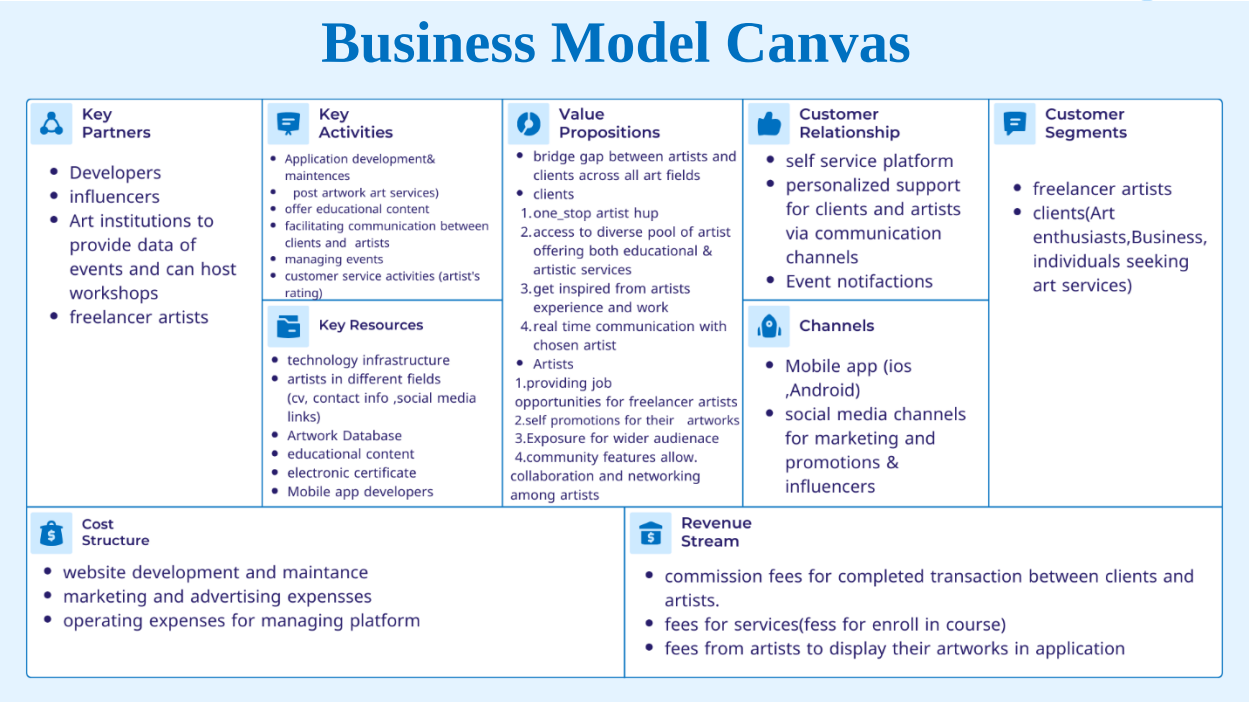


Figure (8) - Business Model Canvas

The Business Model Canvas outlines the essential components of our artist application, designed to bridge the gap between artists and clients, and provide a platform for showcasing and promoting artwork. The key elements of our business model are as follows:

Key Partners:

Developers: Responsible for application development and maintenance.
Influencers: Help in promoting the platform.
Art Institutions: Provide data on events and host workshops.
Freelancer Artists: Contribute content and engage with the community.

Key Activities:

Application development and maintenance.
Posting artwork and offering educational content.
Facilitating communication between clients and artists.
Managing events and customer service activities.

Key Resources:

Technology infrastructure and mobile app developers.
Database of artworks and educational content.
Network of artists and their contact information.

Value Propositions:

For Clients:
A comprehensive hub to find and hire artists.
Access to a diverse pool of artist profiles and their work.
Community features for inspiration and collaboration.
For Artists:
Job opportunities and exposure.
Platforms for promotions and showcasing their work.
Networking opportunities within the artist community.

Customer Relationships:

Self-service platform with personalized support.
Event notifications to keep users informed.

Channels:

Mobile application (iOS and Android).
Social media channels for marketing and promotions.

Customer Segments:

Freelancer artists seeking job opportunities and exposure.
Clients include art enthusiasts, businesses, and individuals seeking art services.

Cost Structure:

Expenses for website development and maintenance.
Operating expenses for managing the platform.
Marketing and advertising costs.

Revenue Streams:

Commission fees from transactions between clients and artists.
Fees for displaying artwork on the platform.
Service fees for additional features and course enrollments.
This structured approach ensures a robust platform that not only supports artists in showcasing their work but also facilitates seamless interaction and transactions with clients, thereby fostering a thriving art community.

(URL: https://next.canvanizer.com/canvas/w59tH3KWAxR5L)

Chapter 5

(Implementation and Testing)

Elysium App UI

Artist-Client Pages:

Sign Up:

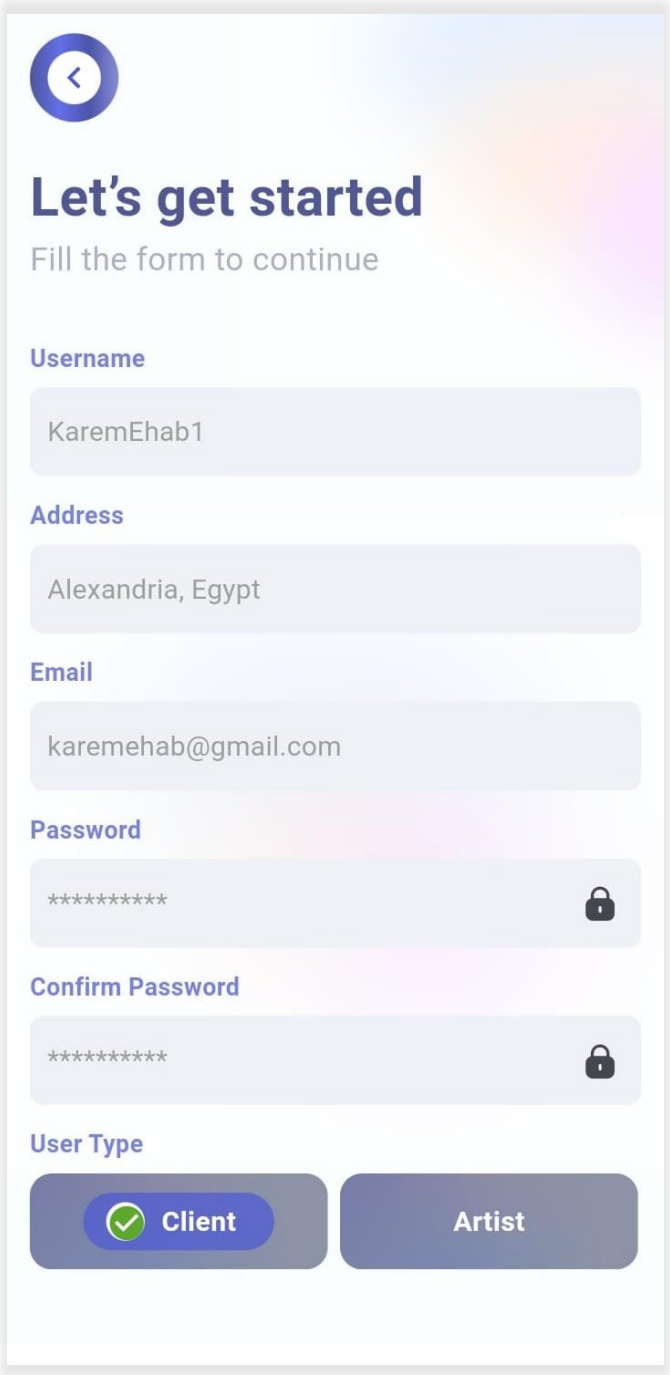


Figure (10) - Sign Up

Sign In:

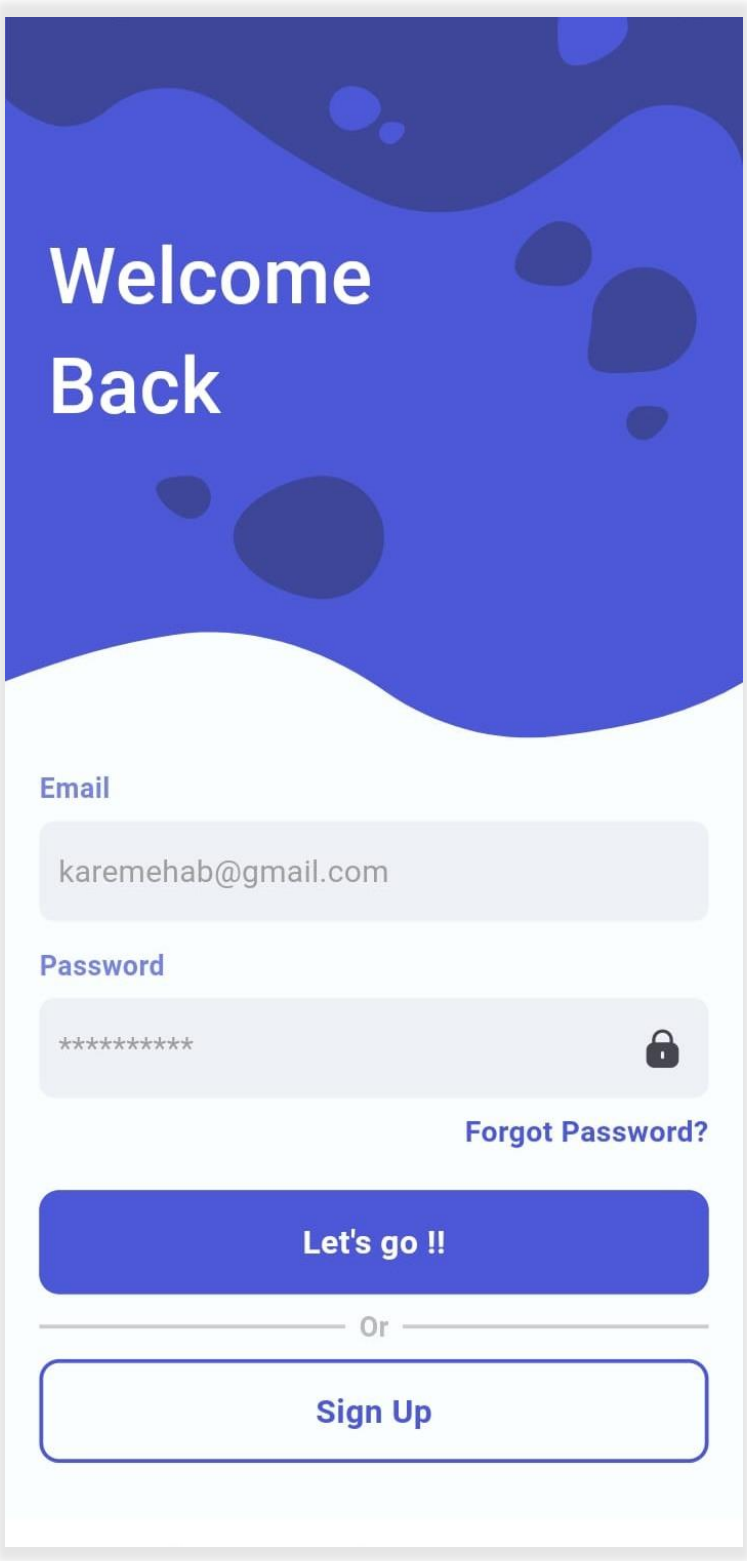


Figure (9) - Sign In

Home Page:

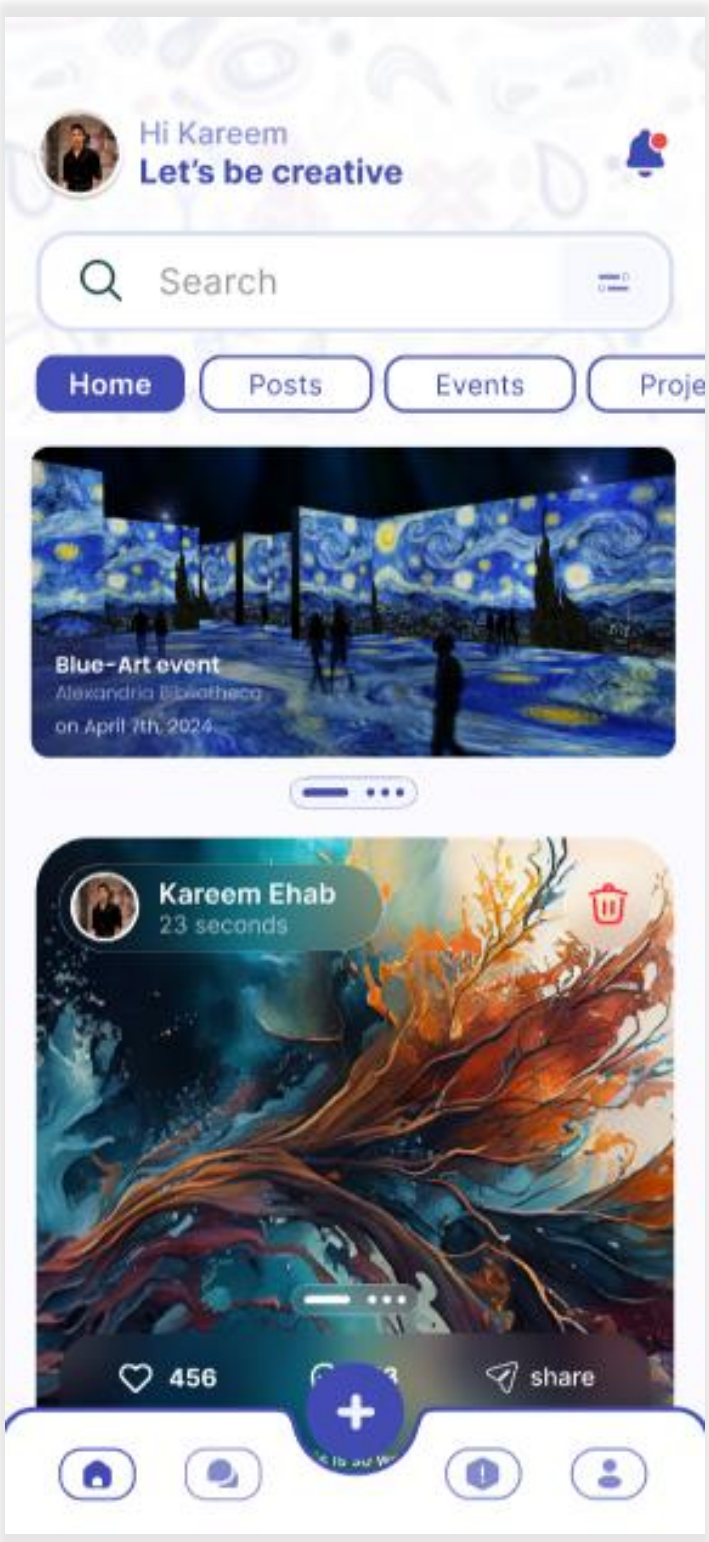


Figure (10) - Home Page

Artist-Client Inbox:

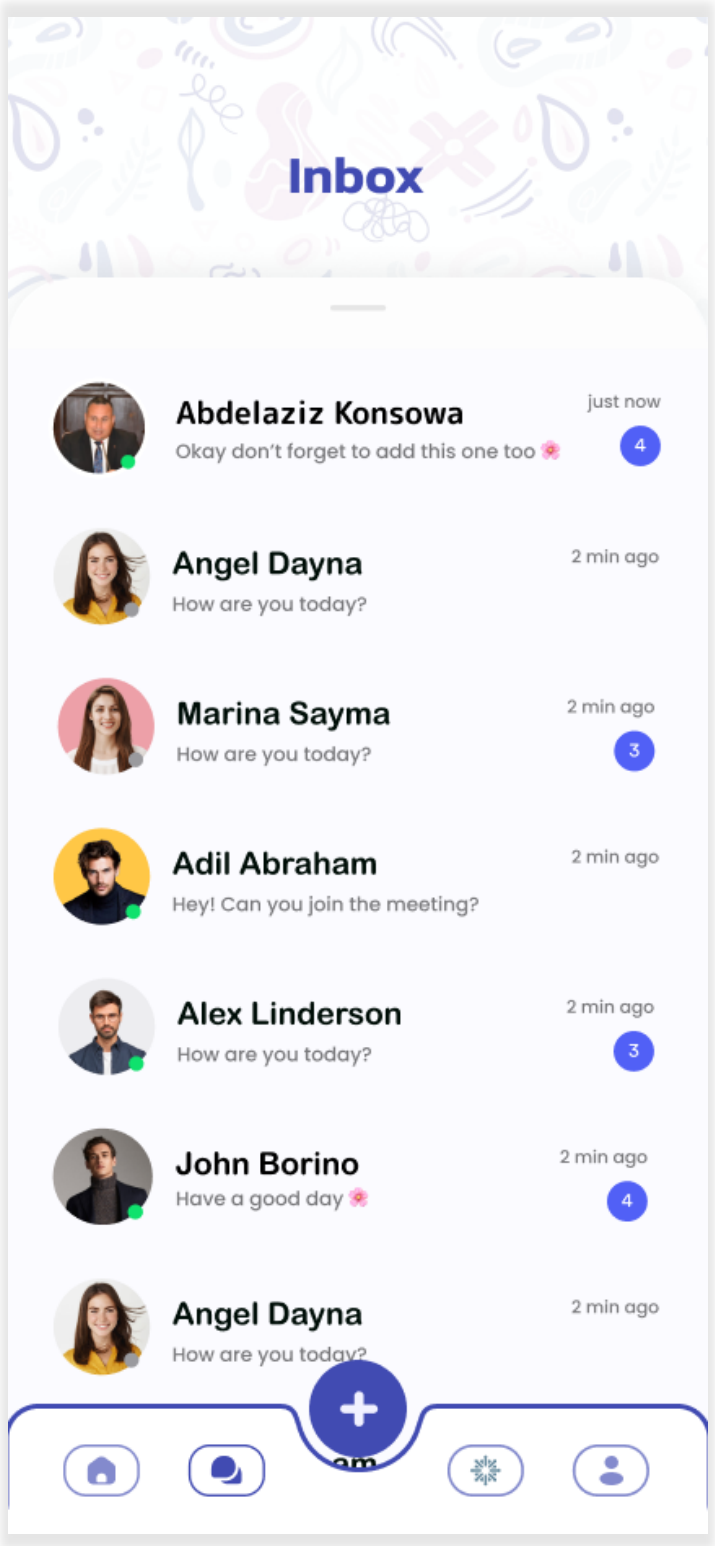


Figure (11) - Artist-Client Inbox

Artist-Client Chat:

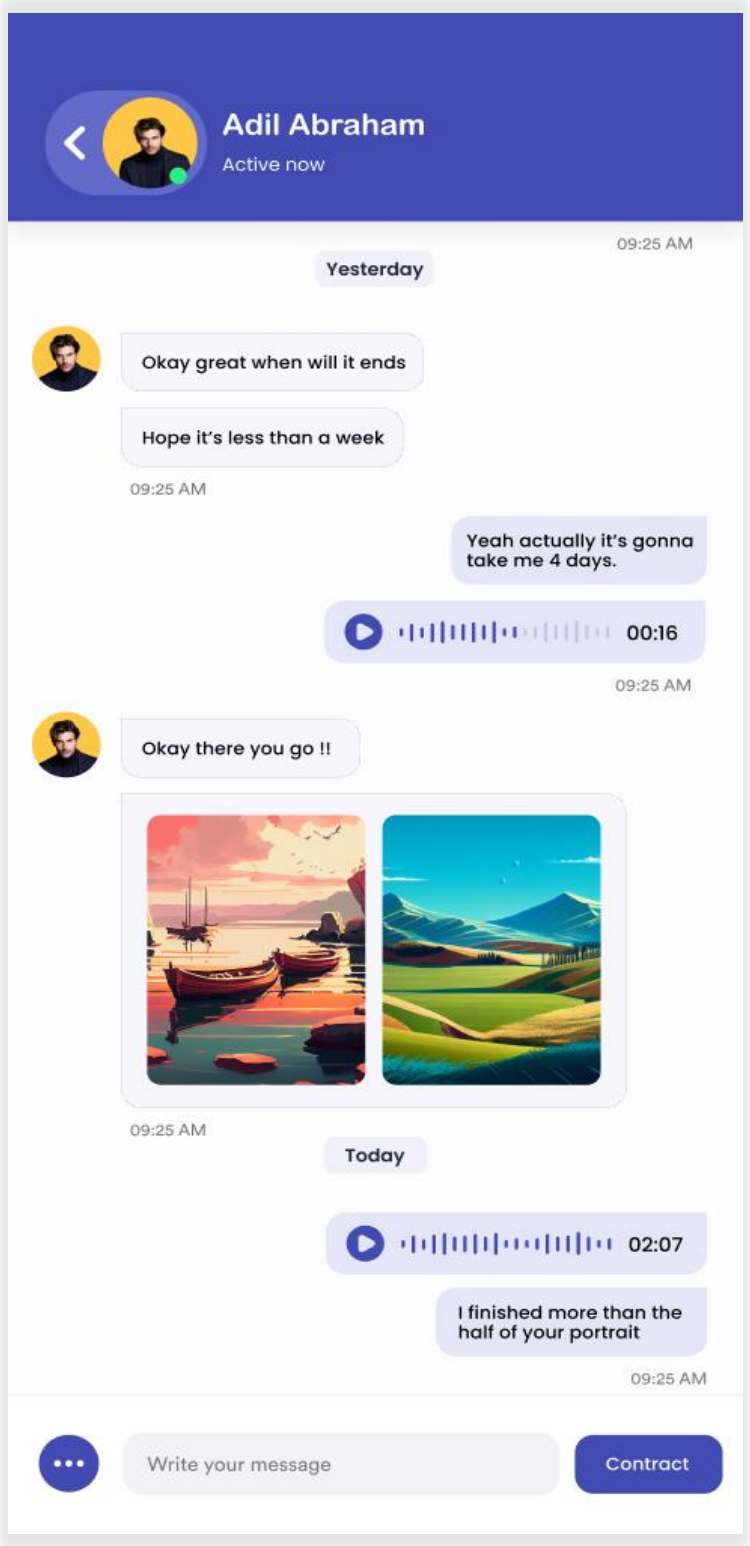


Figure (14) - Artist-Client Chat

Milestones:

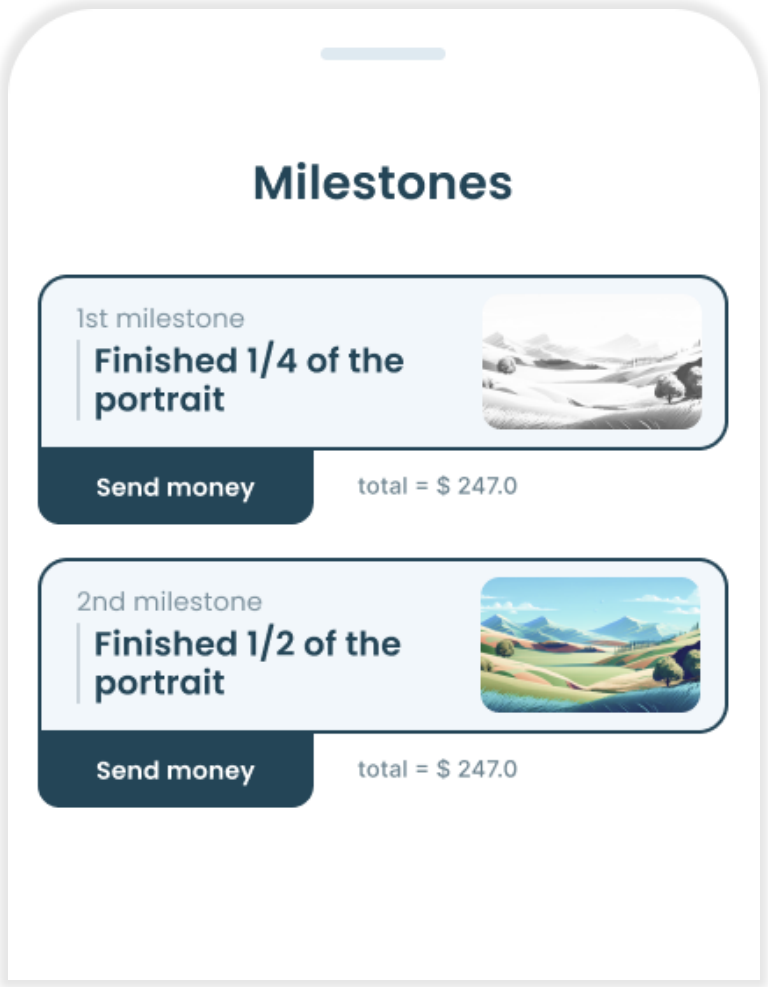


Figure (12) - Milestones

Community Posts:

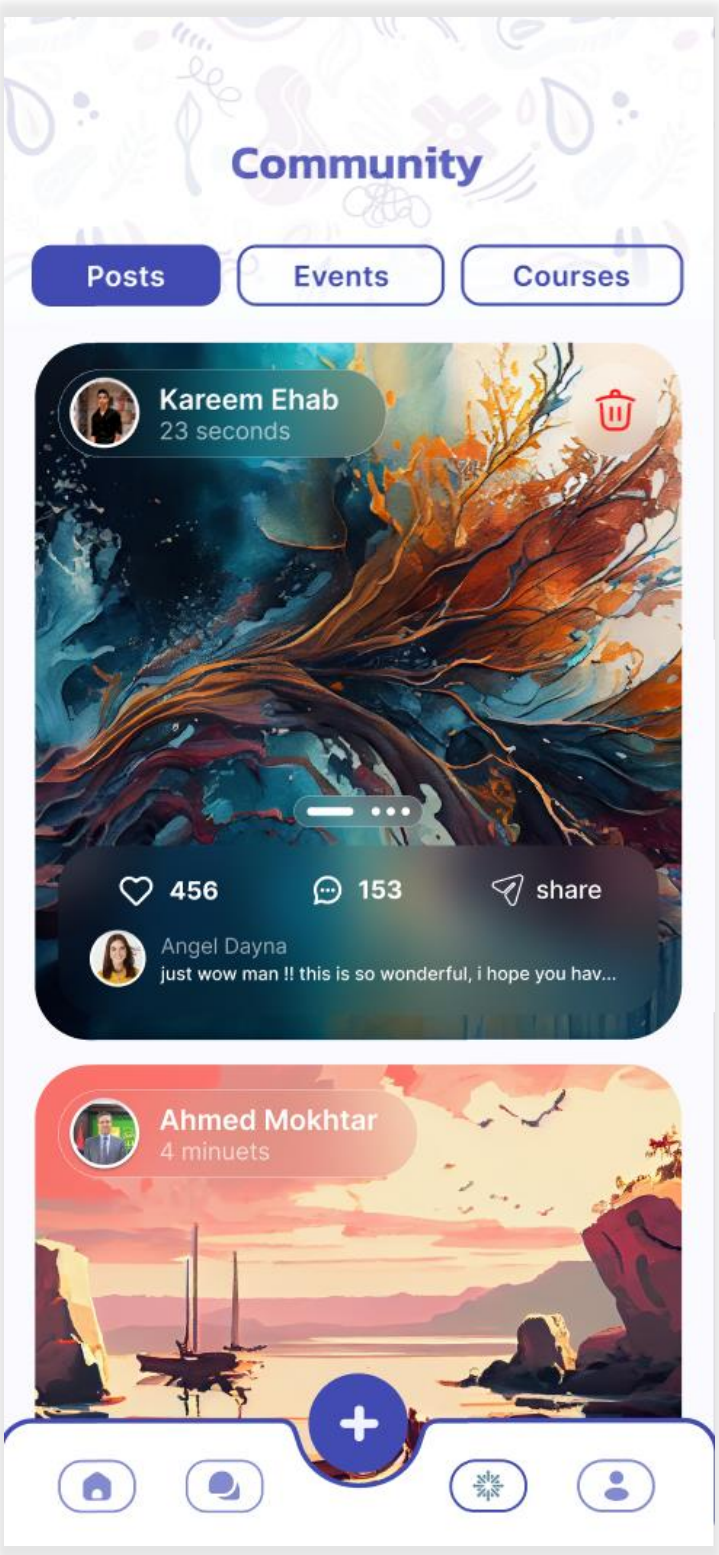


Figure (13) - Community Posts

Artist Collection:

Figure (14) - Artist Collection

Community Events:

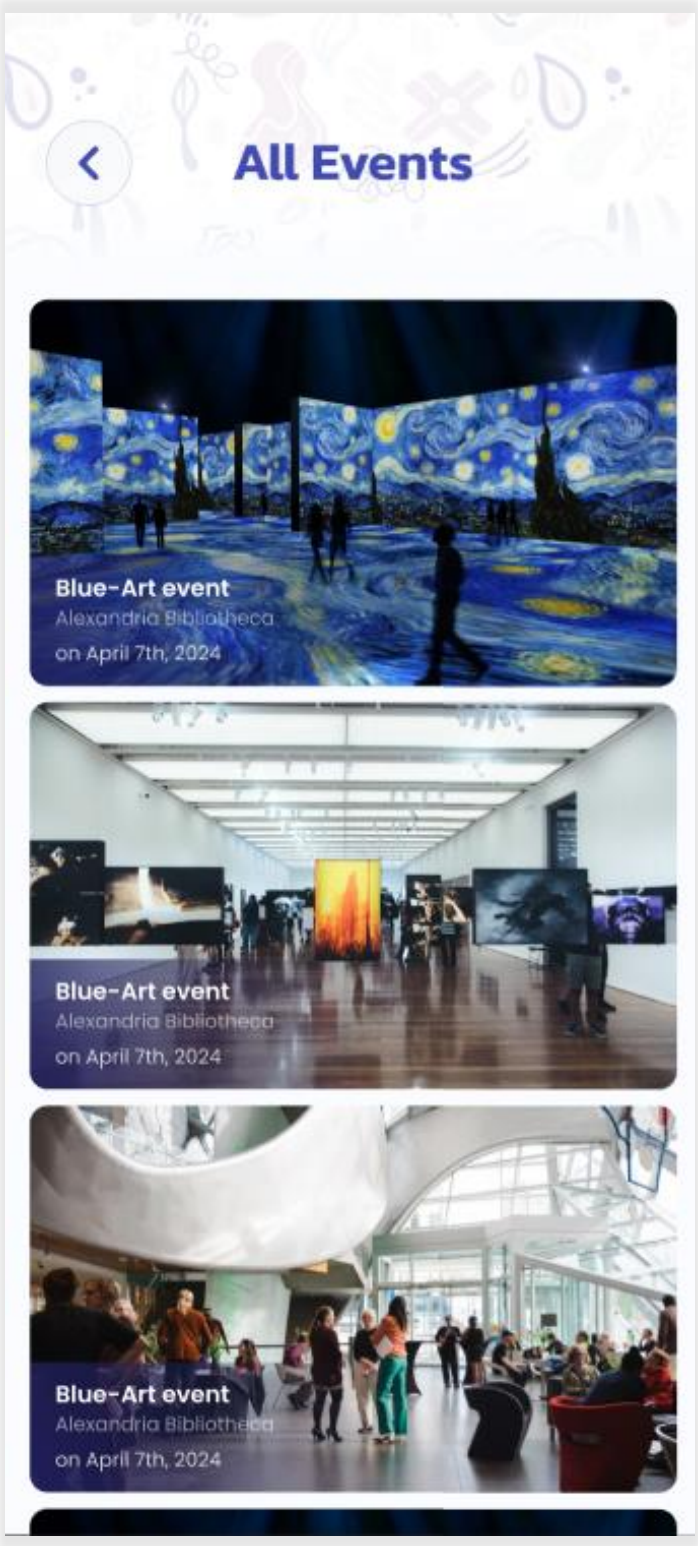


Figure (15) - Community Events

Event Details:

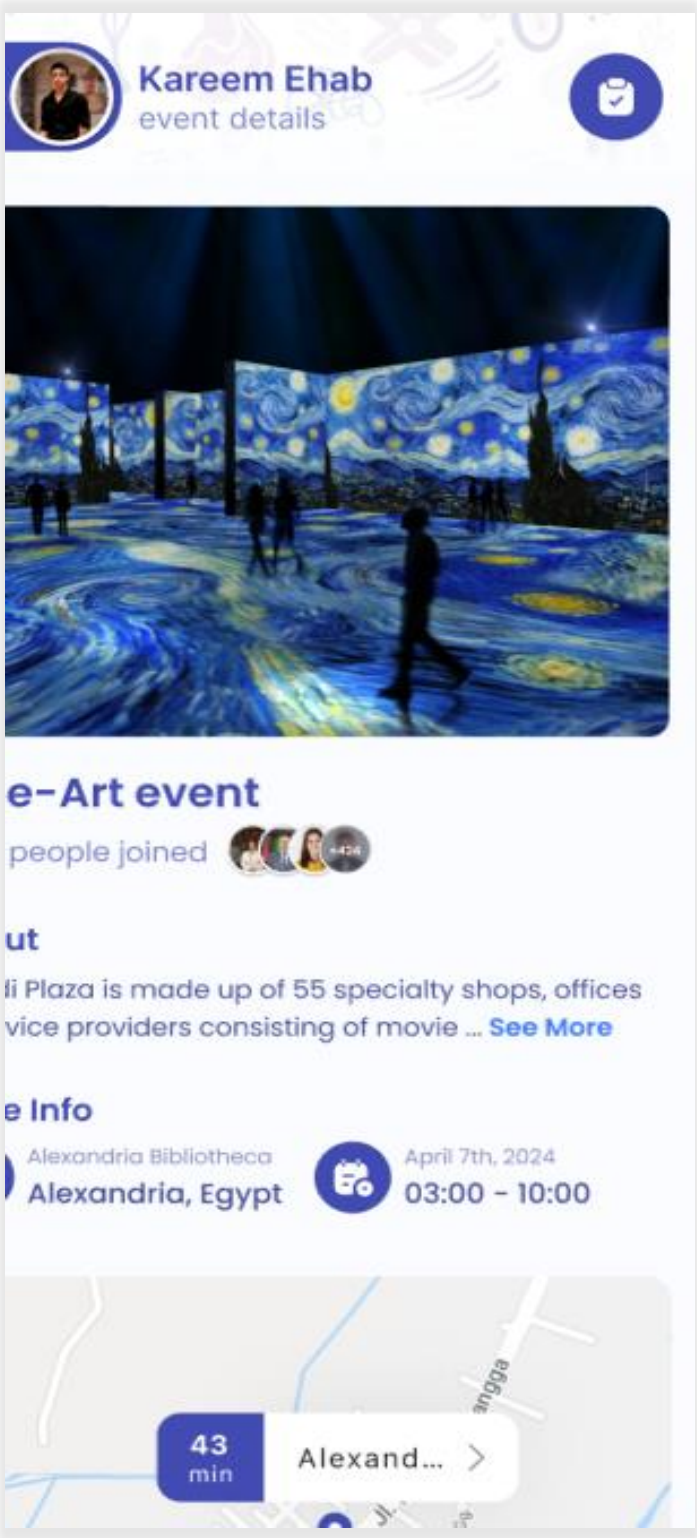


Figure (16) - Event Details

Course Details:

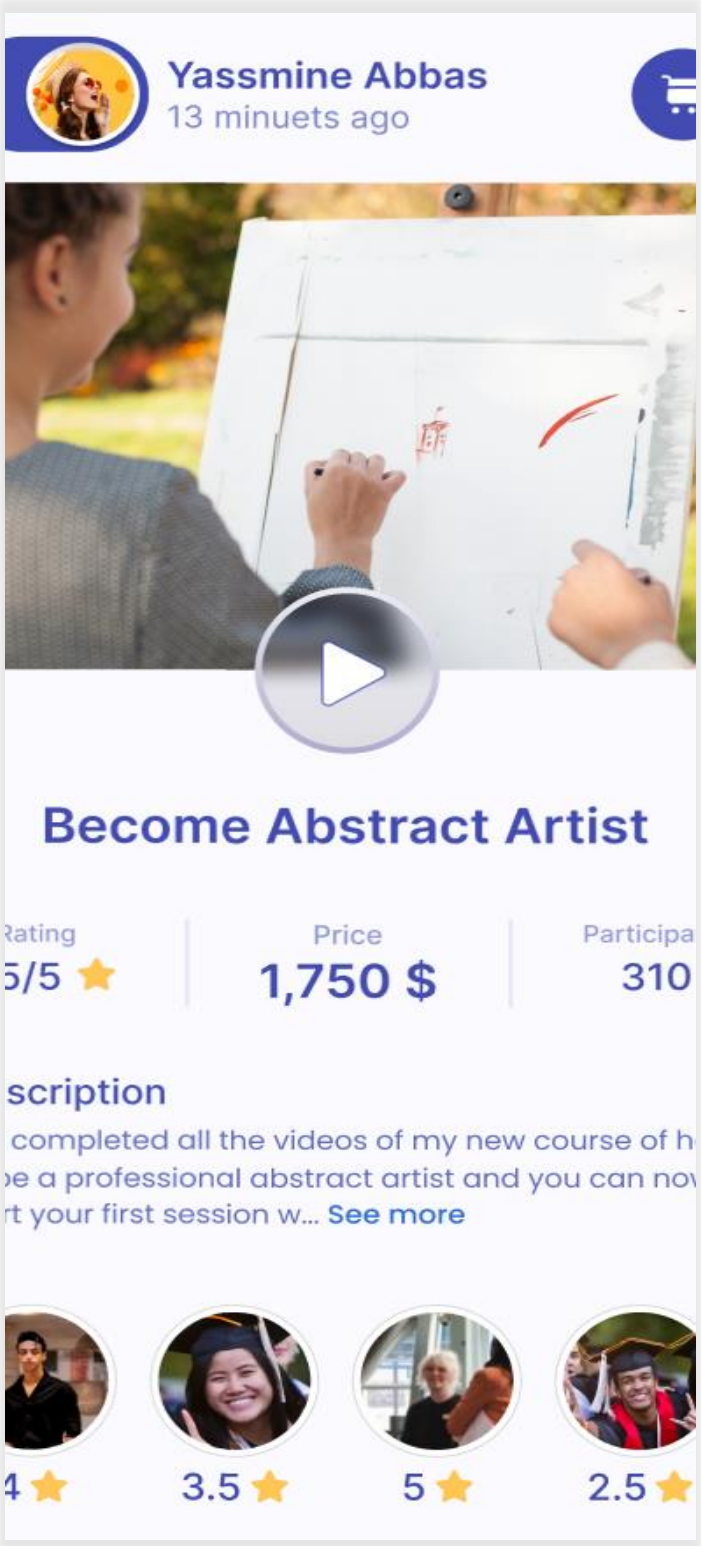


Figure (17) - Course Details

Create Post:

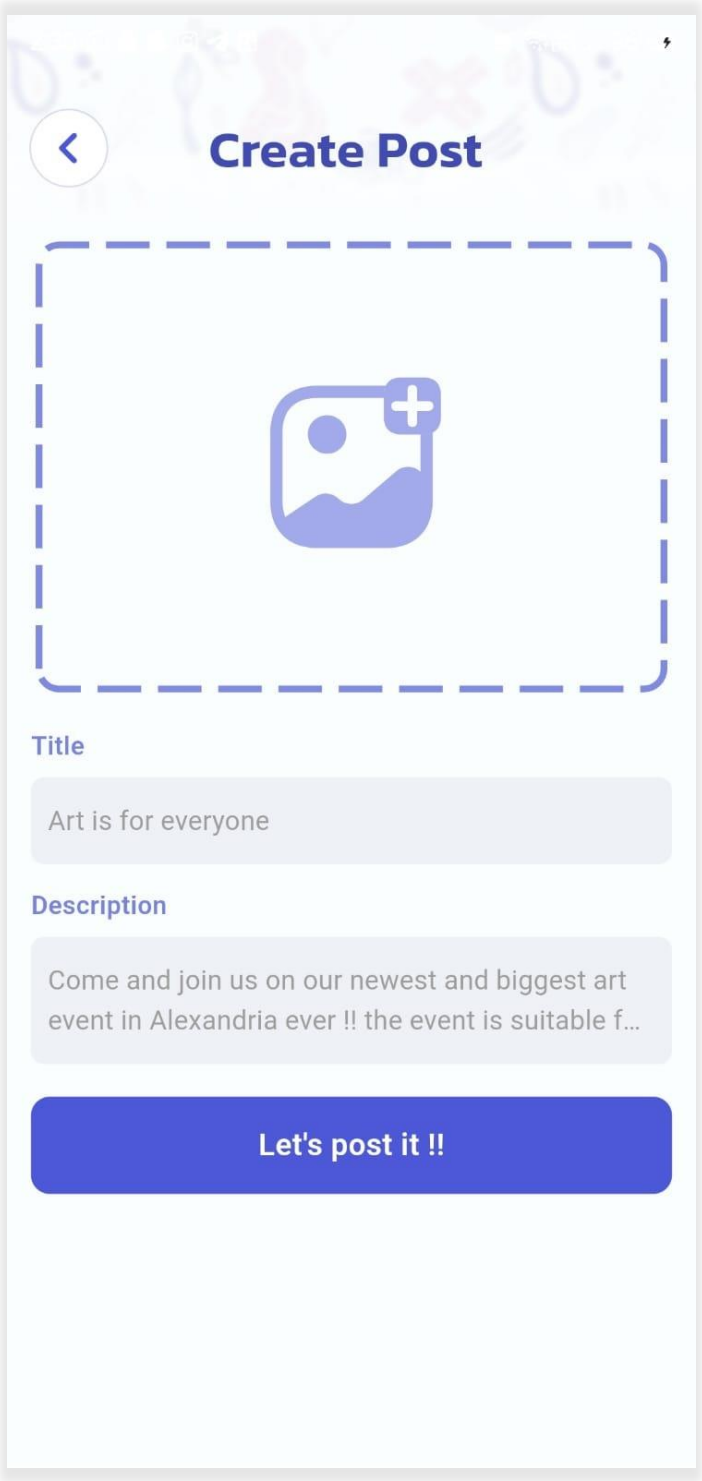


Figure (18) - Create Post

Artist Pages:

Community Courses:

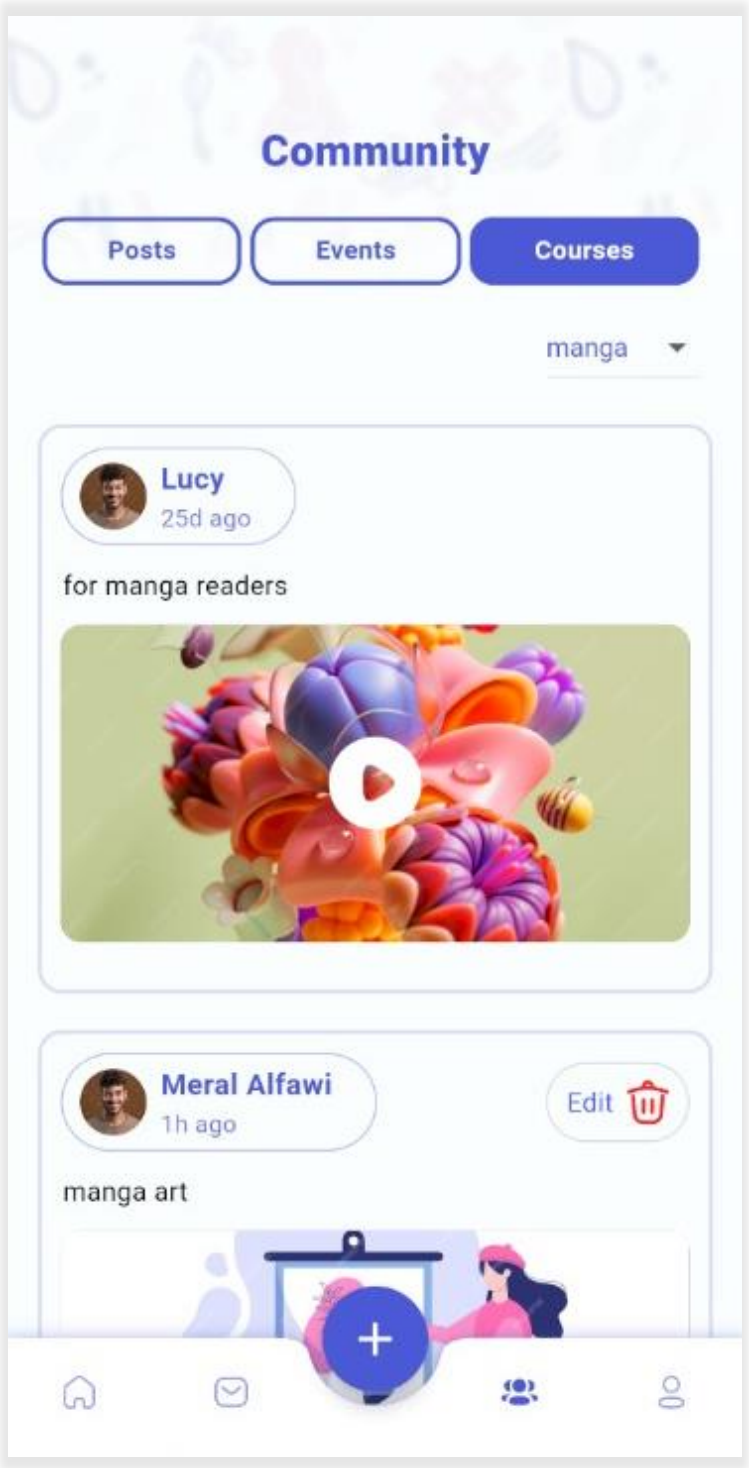


Figure (19) - Community Courses

Artist Profile:

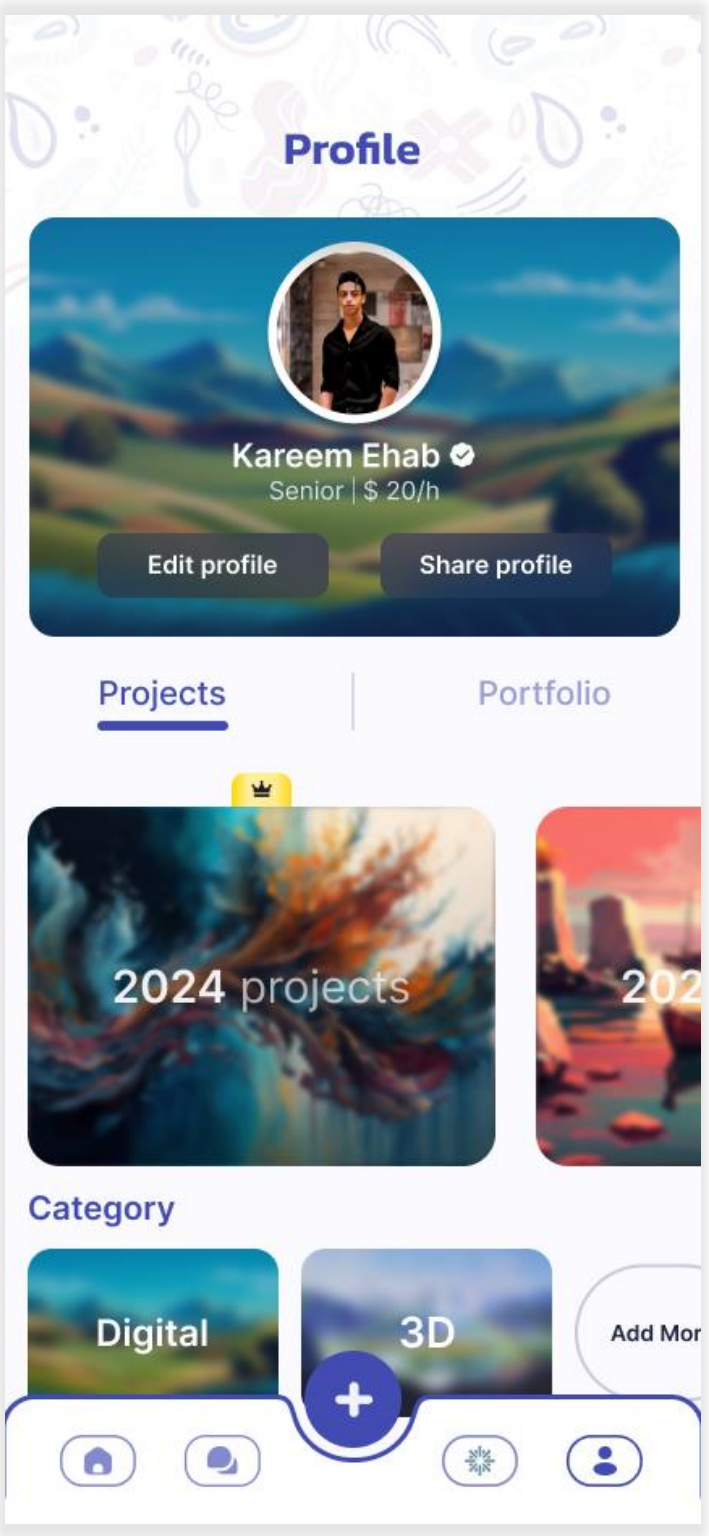


Figure (20) - Artist Profile

Create Event:

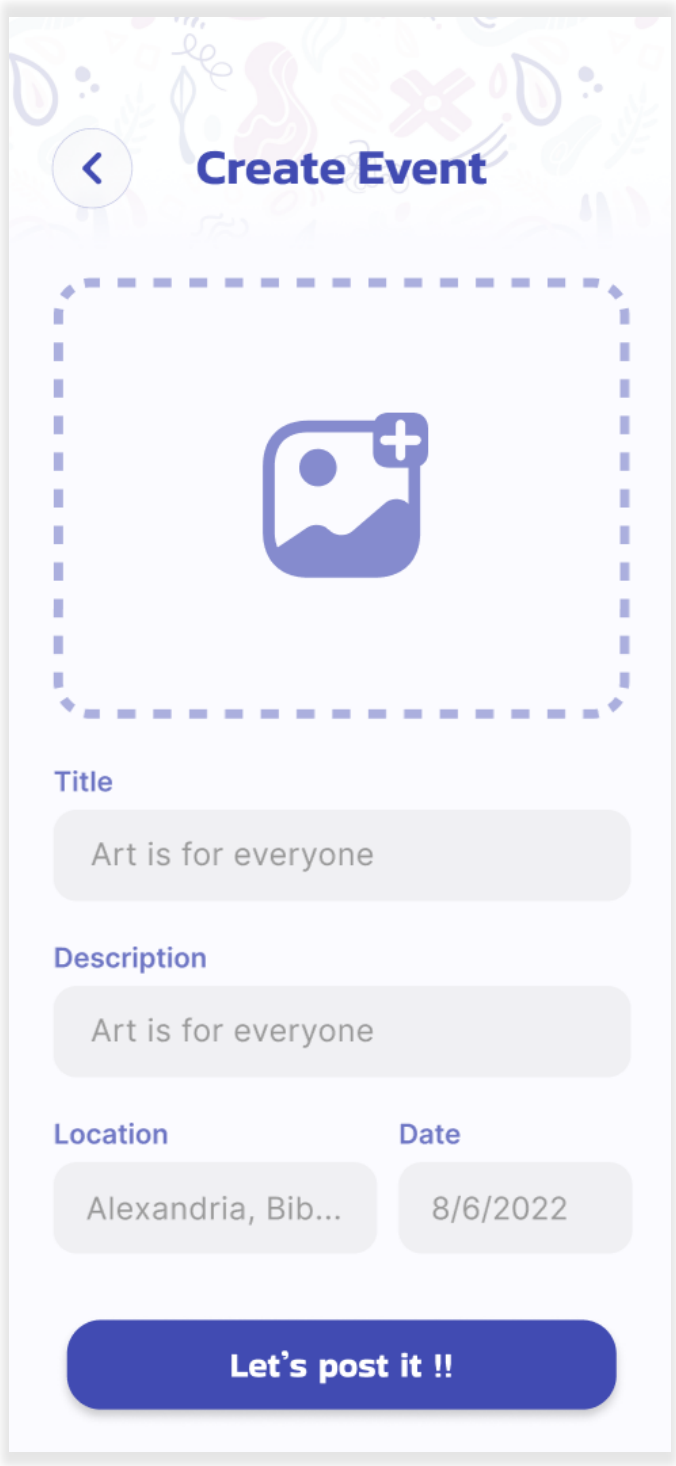


Figure (21) - Create Event

Project Details:

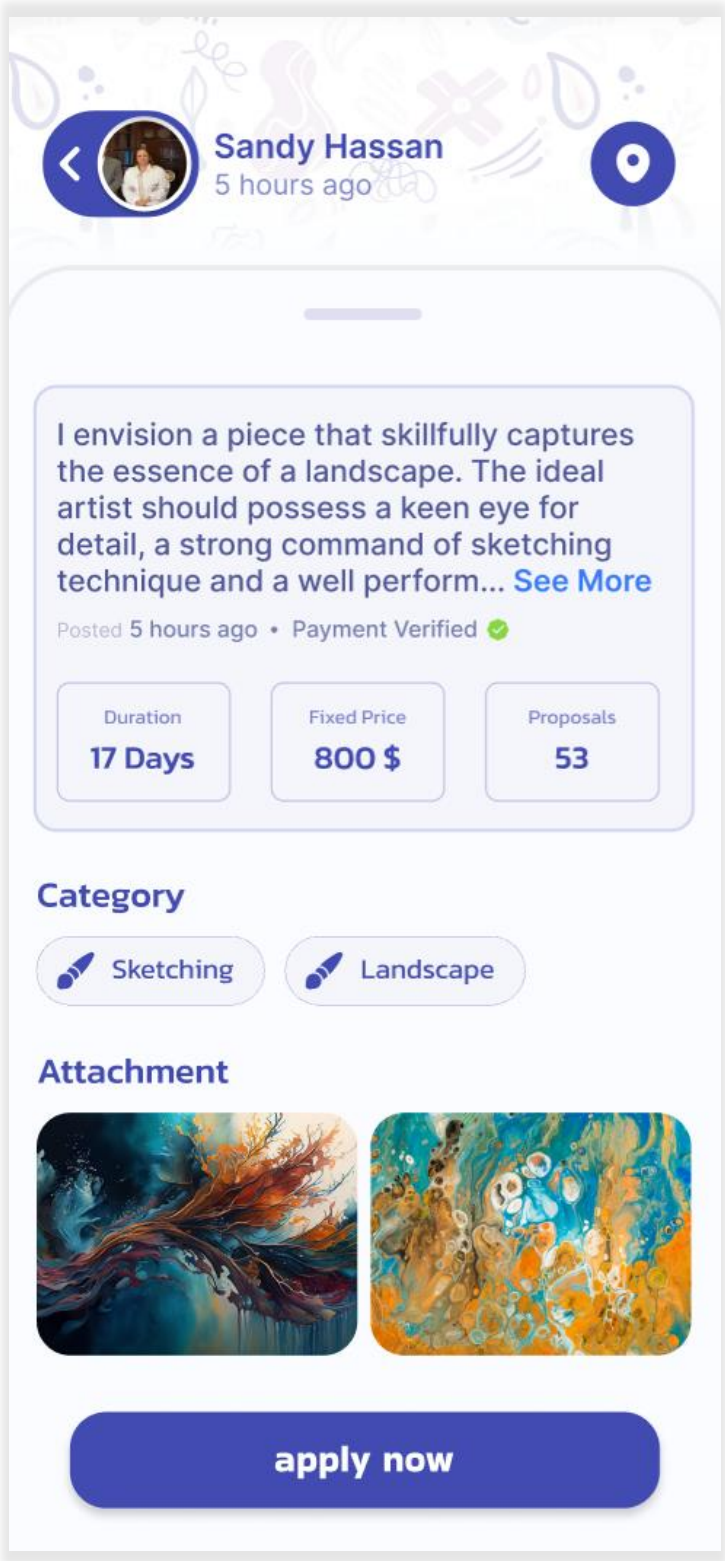


Figure (22) - Project Details

Proposal:

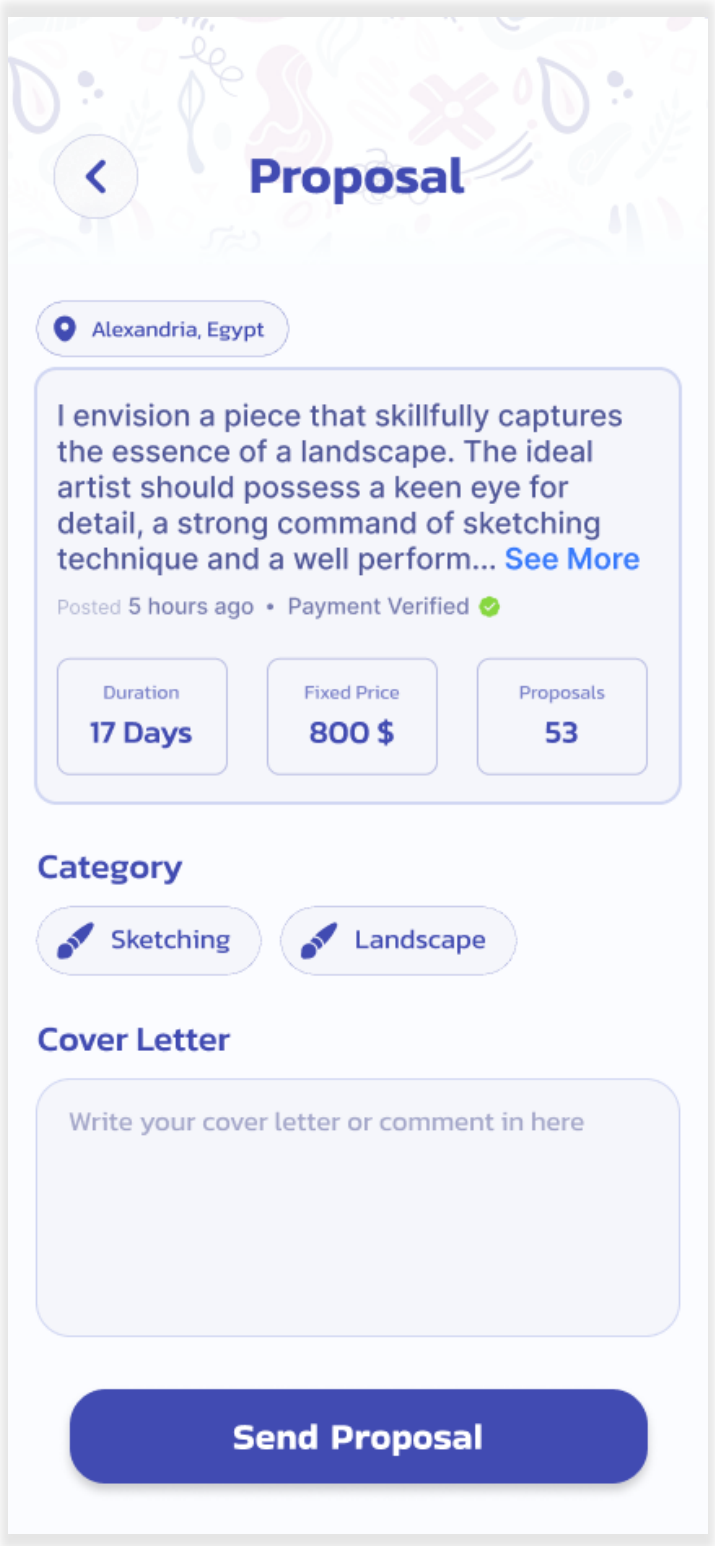


Figure (23) - Proposal

Contracts:

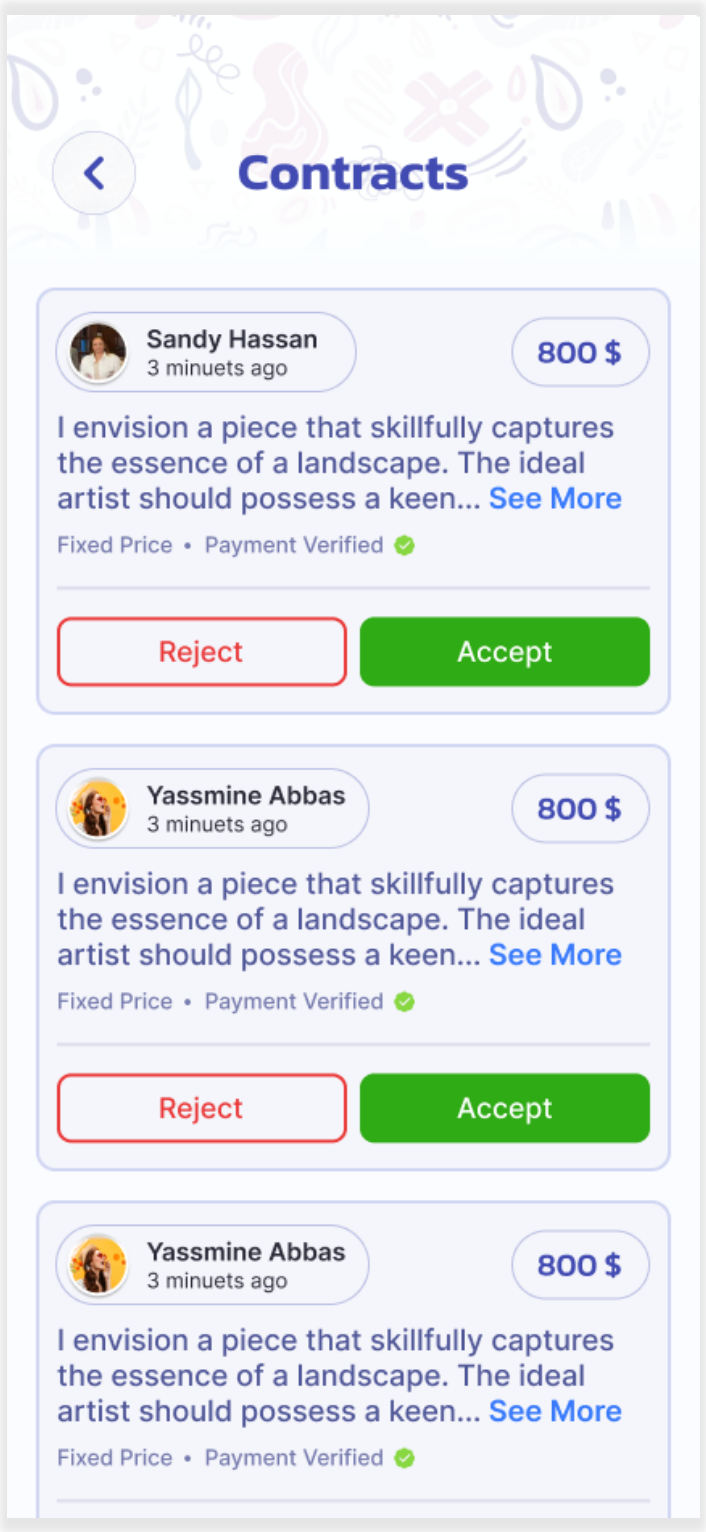


Figure (24) - Contracts

Contract Details:

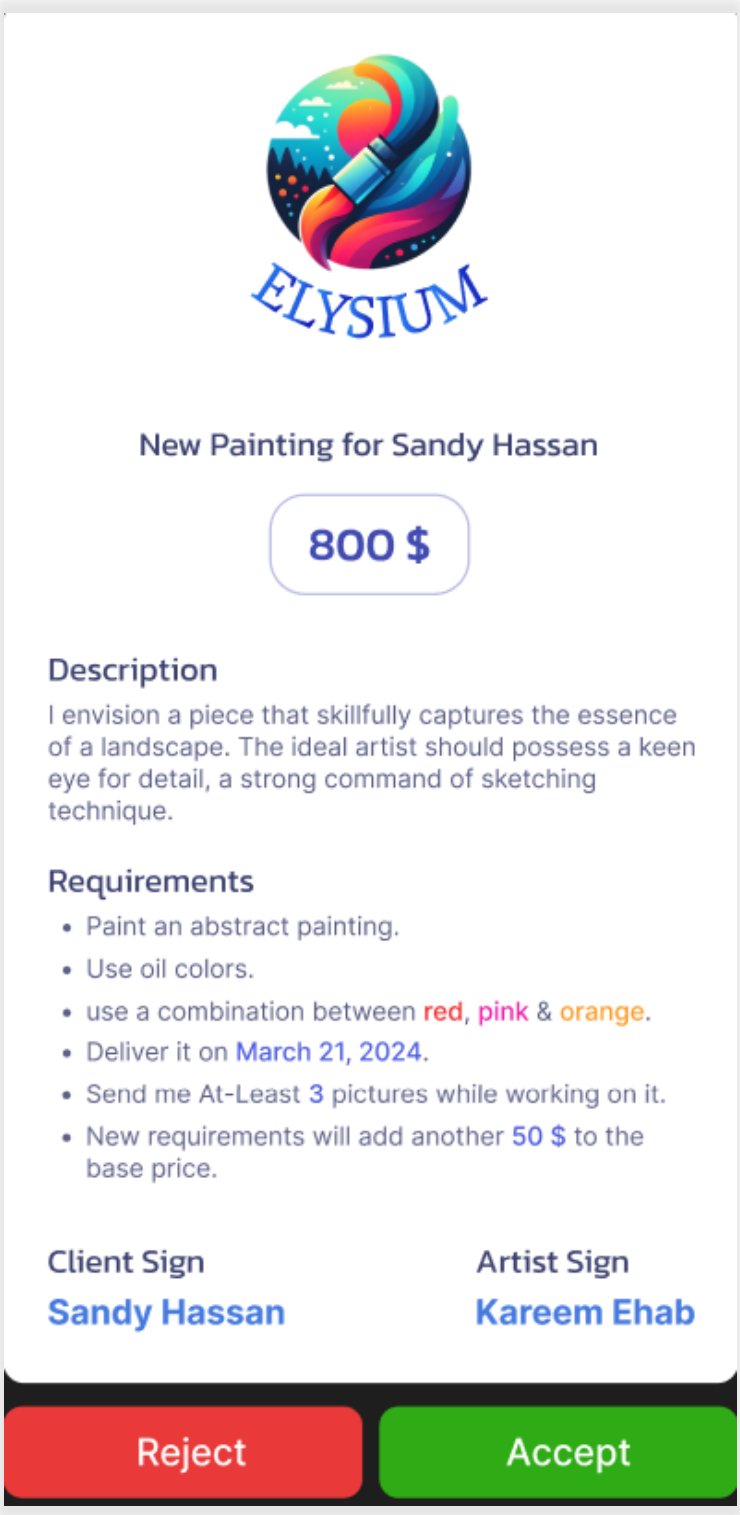


Figure (25) - Contract Details

Client pages:

Community Courses:

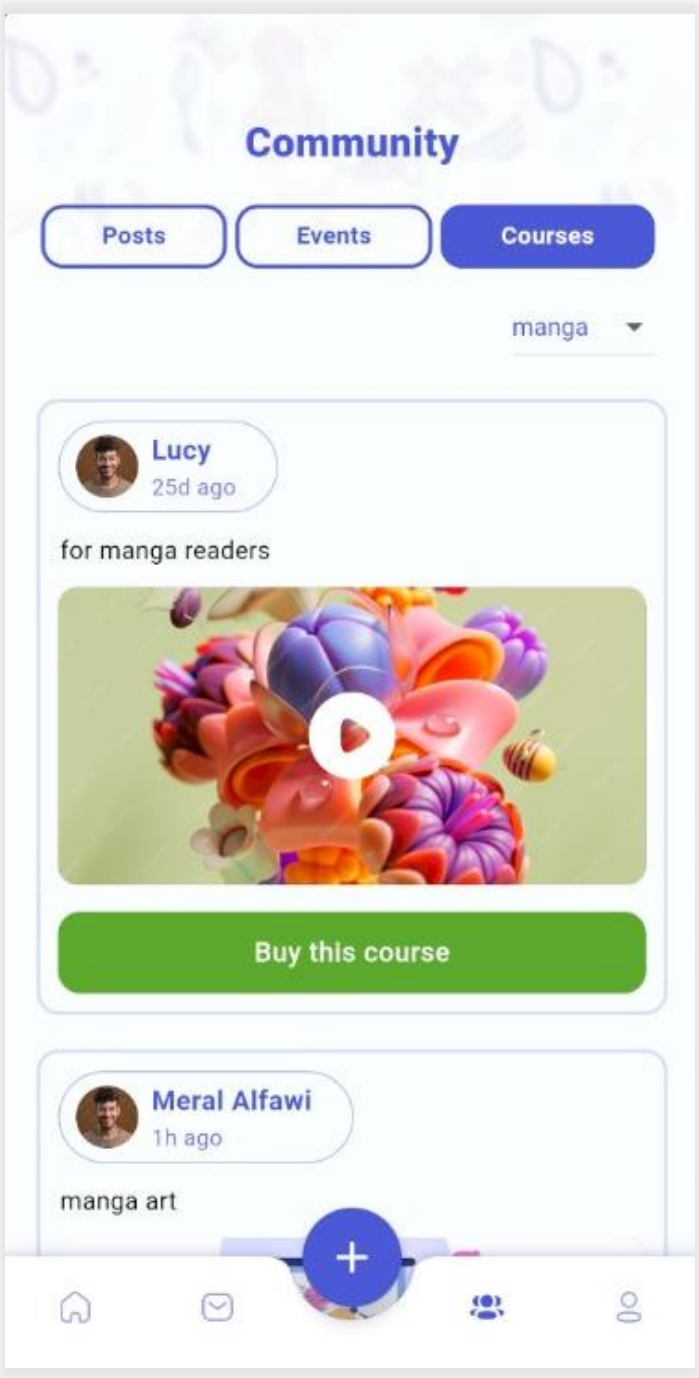


Figure (26) - Community Courses

Client Profile:

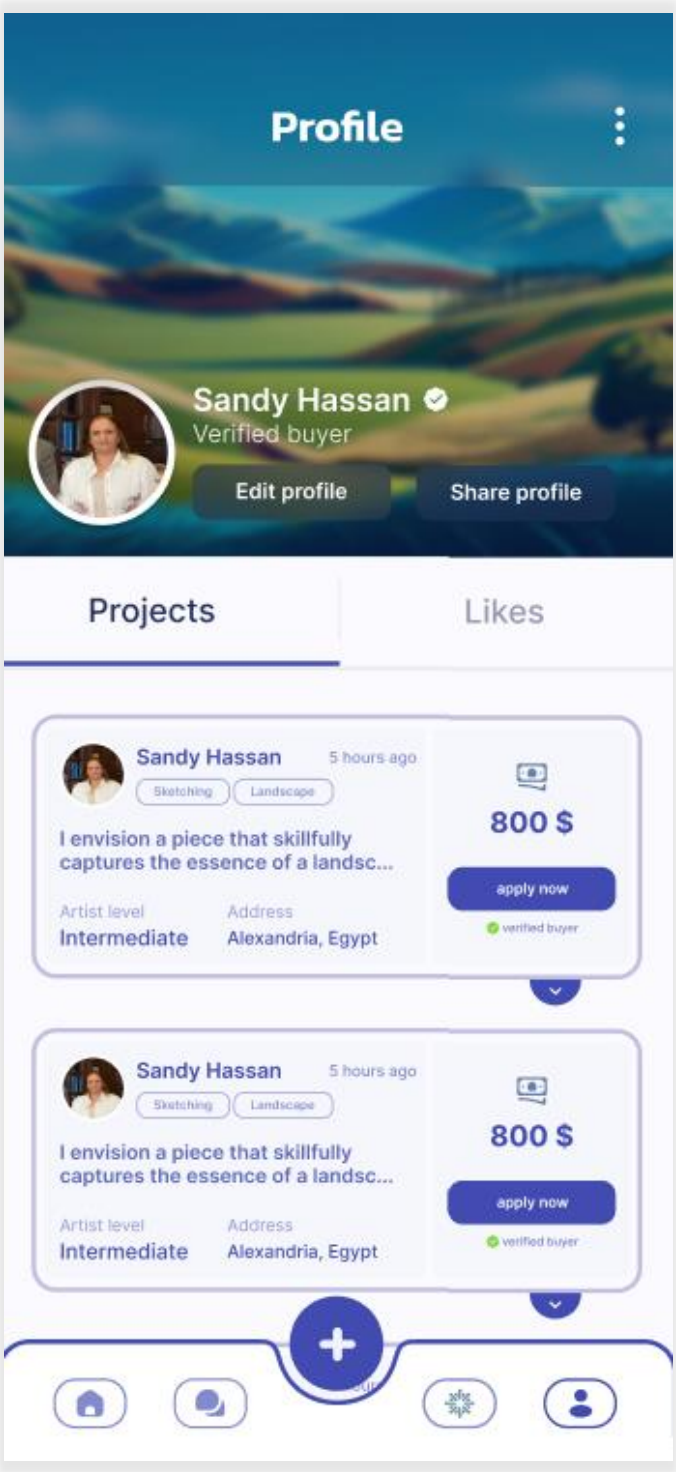


Figure (27) - Client Profile

Create Project:

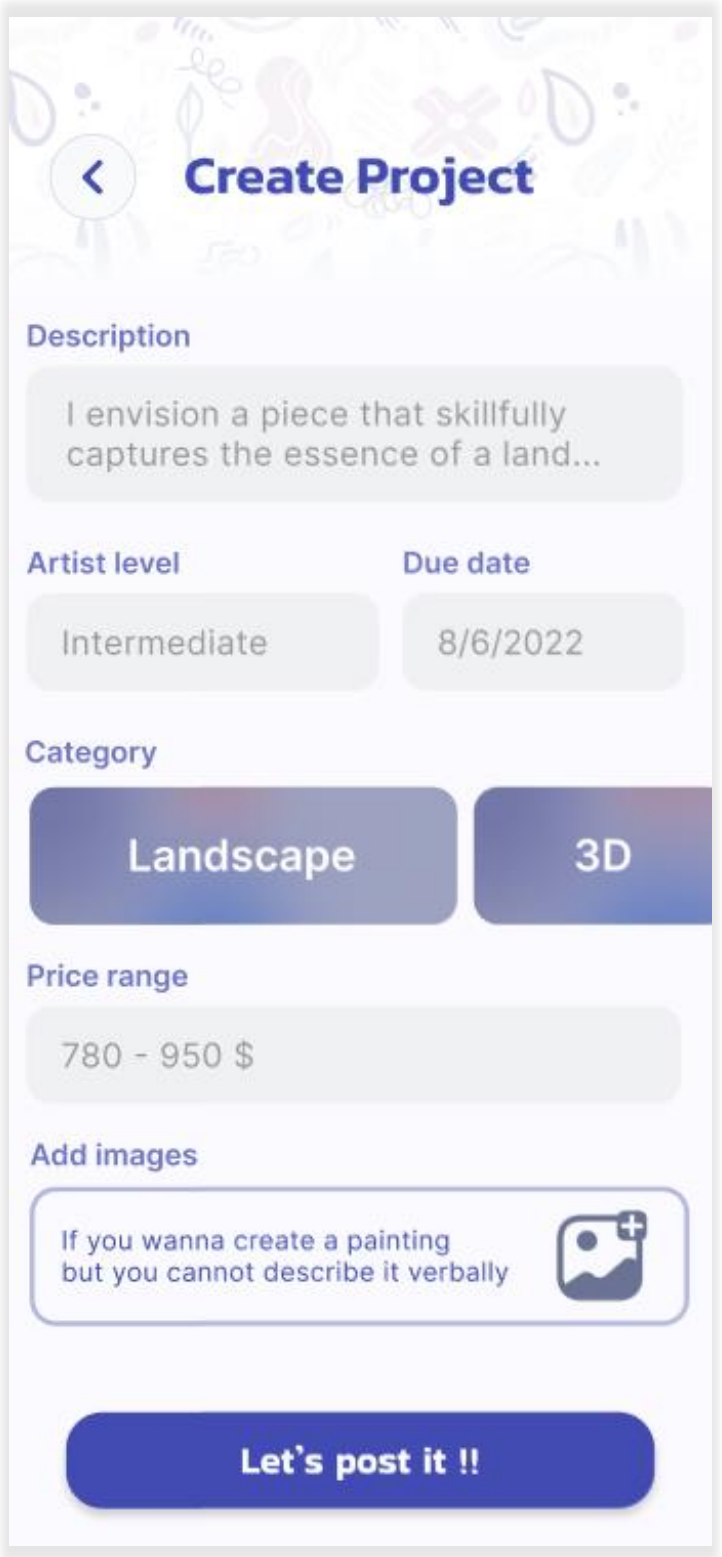


Figure (28) - Create Project

Client project:

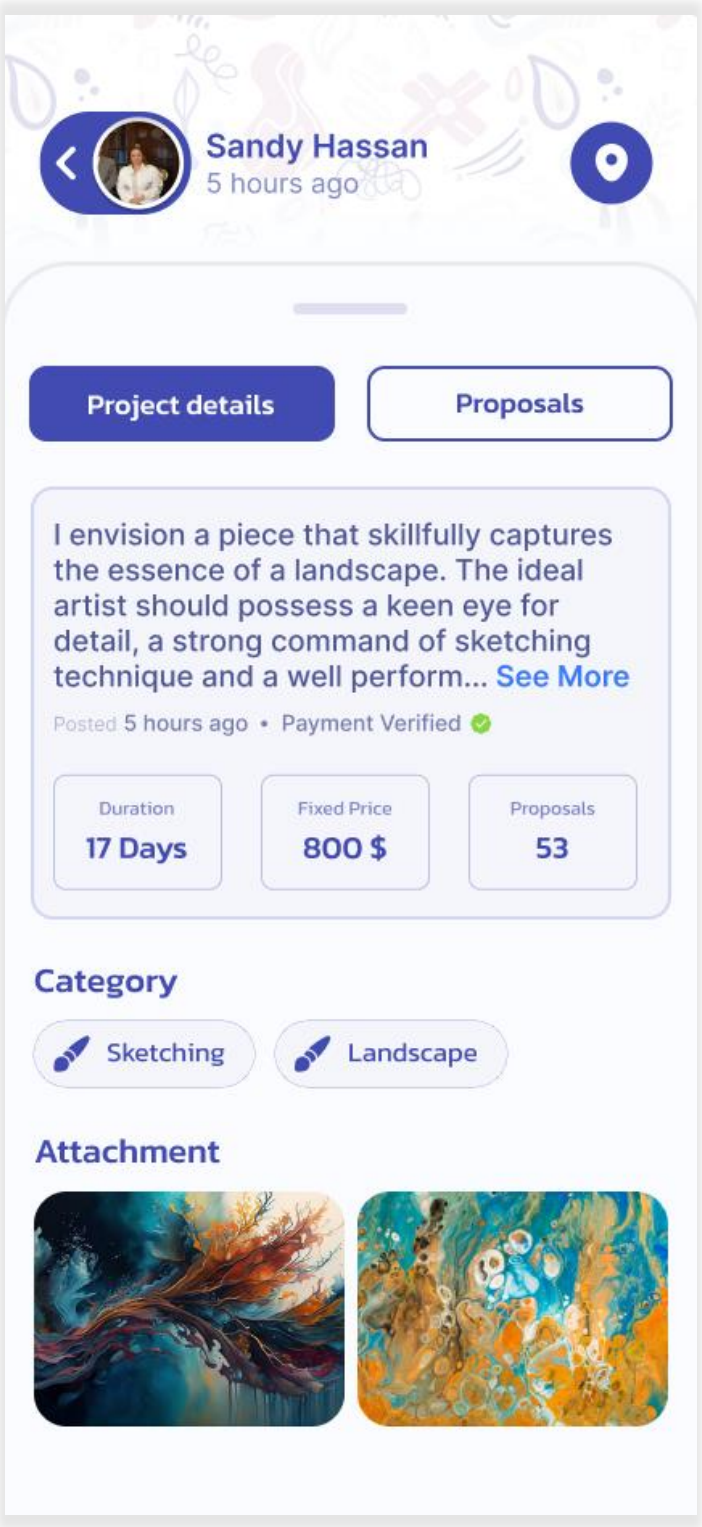


Figure (29) - Client project

Client Cart:

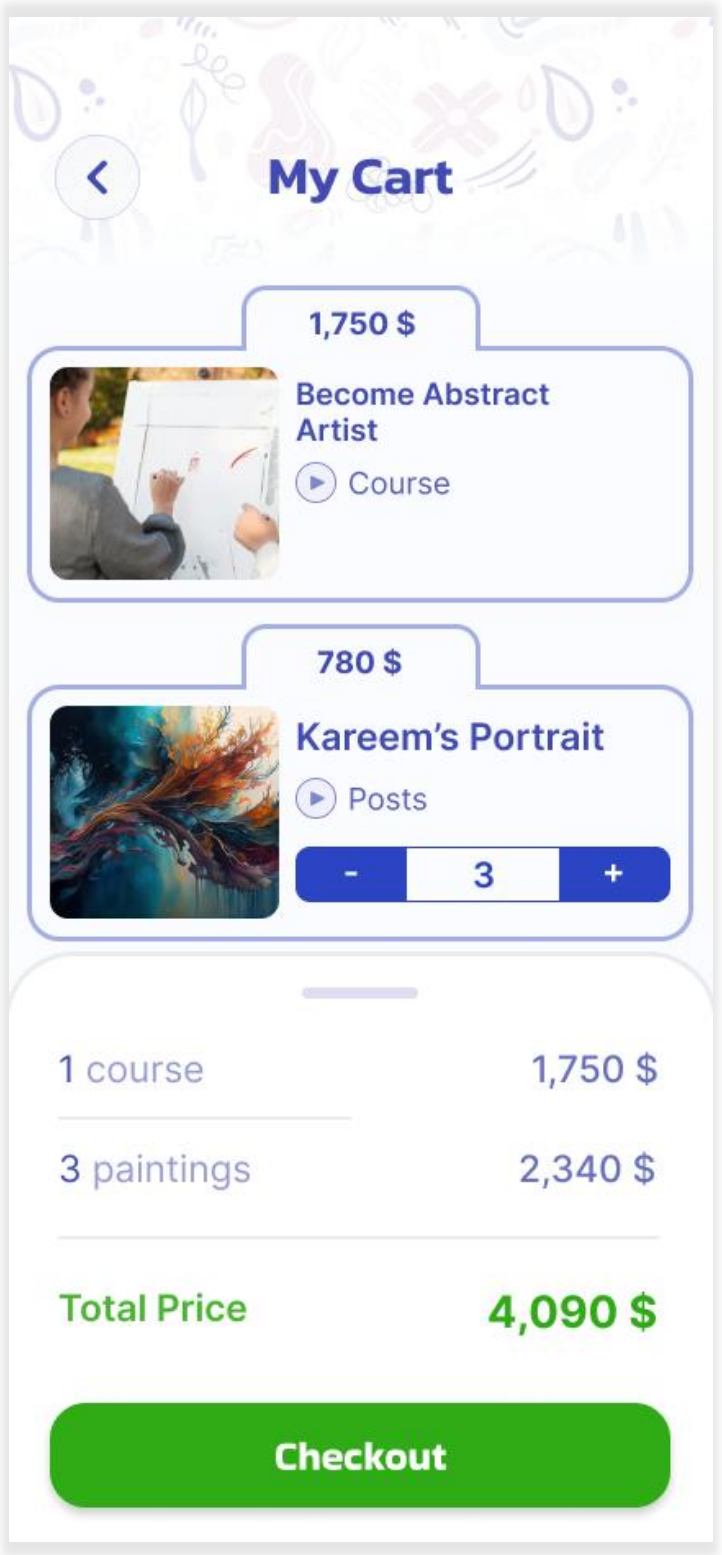


Figure (30) - Client Cart

Admin Pages:

Admin Home Page:

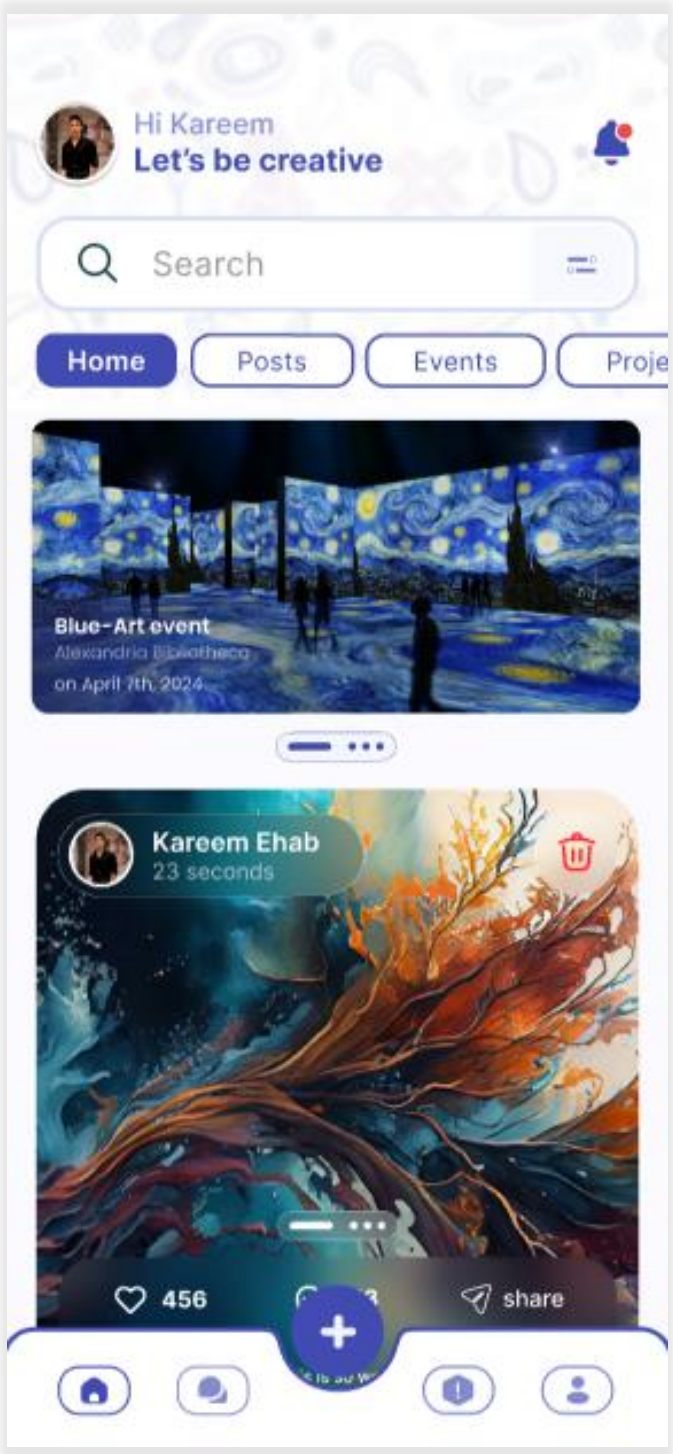


Figure (31) - Admin Home Page

Chat Reports:

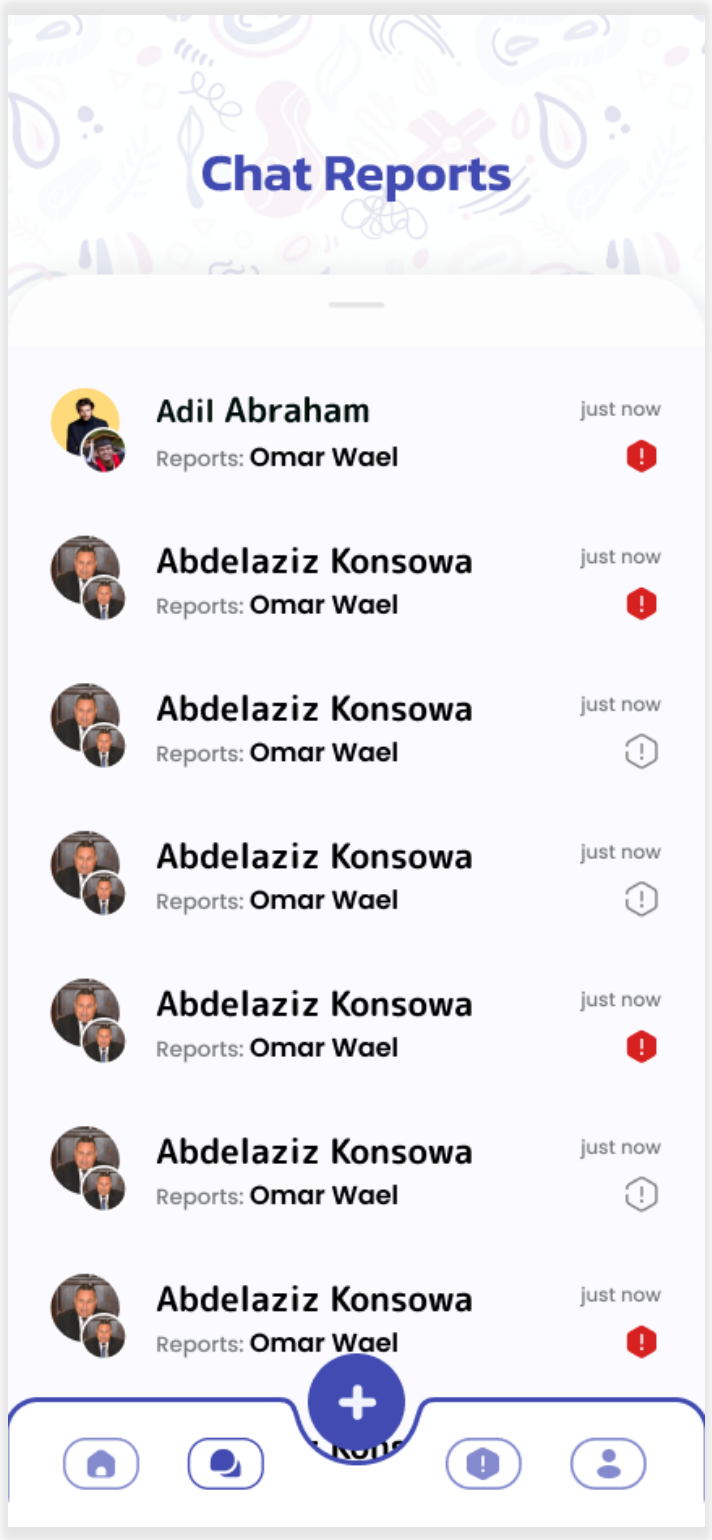


Figure (32) - Chat Reports

Complaint:

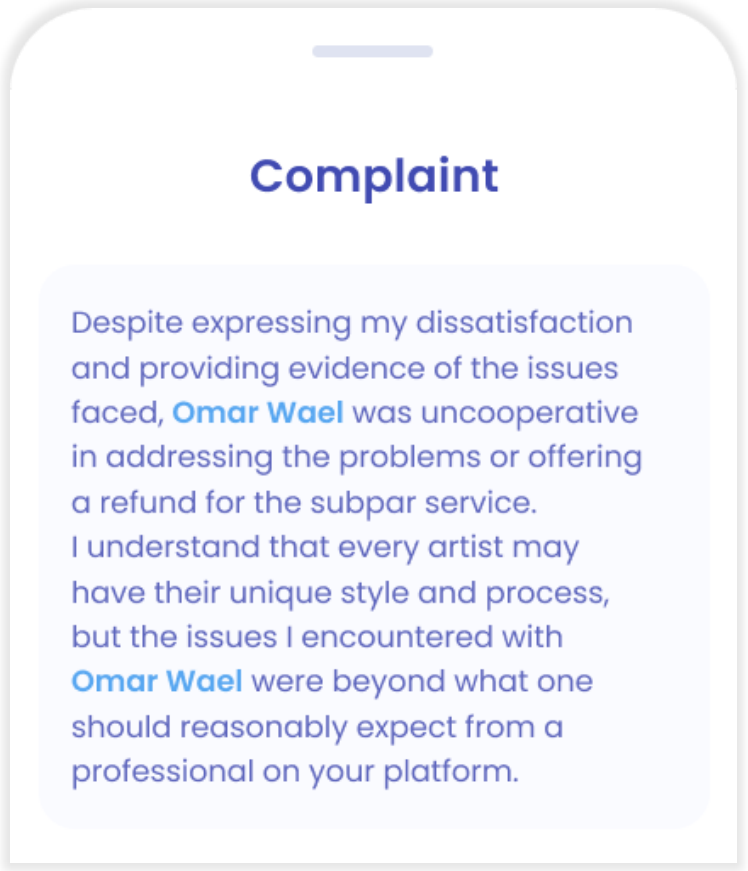


Figure (33) - Complaint

Posts Reports:

Figure (34) - Posts Reports

Orders Reports:

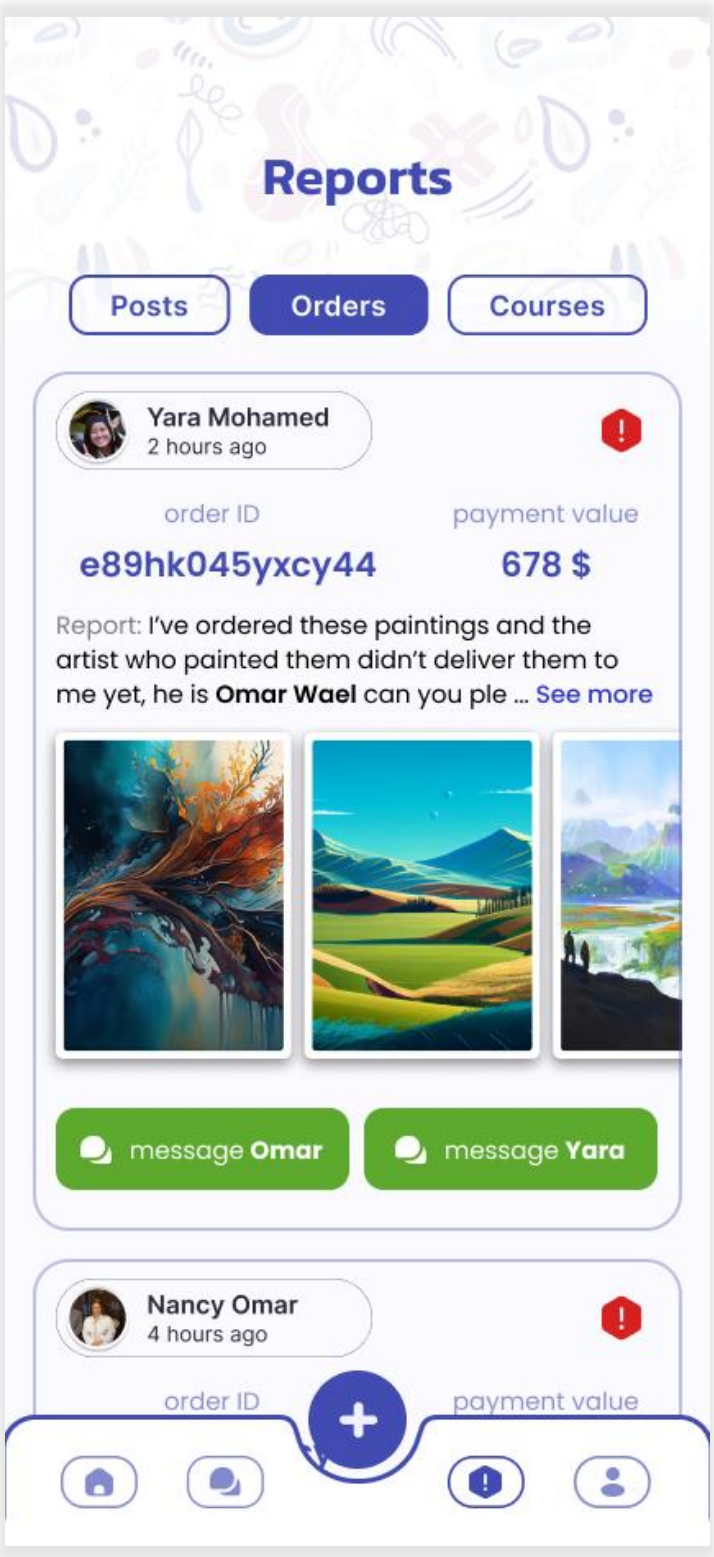


Figure (35) - Orders Reports

Courses Reports:

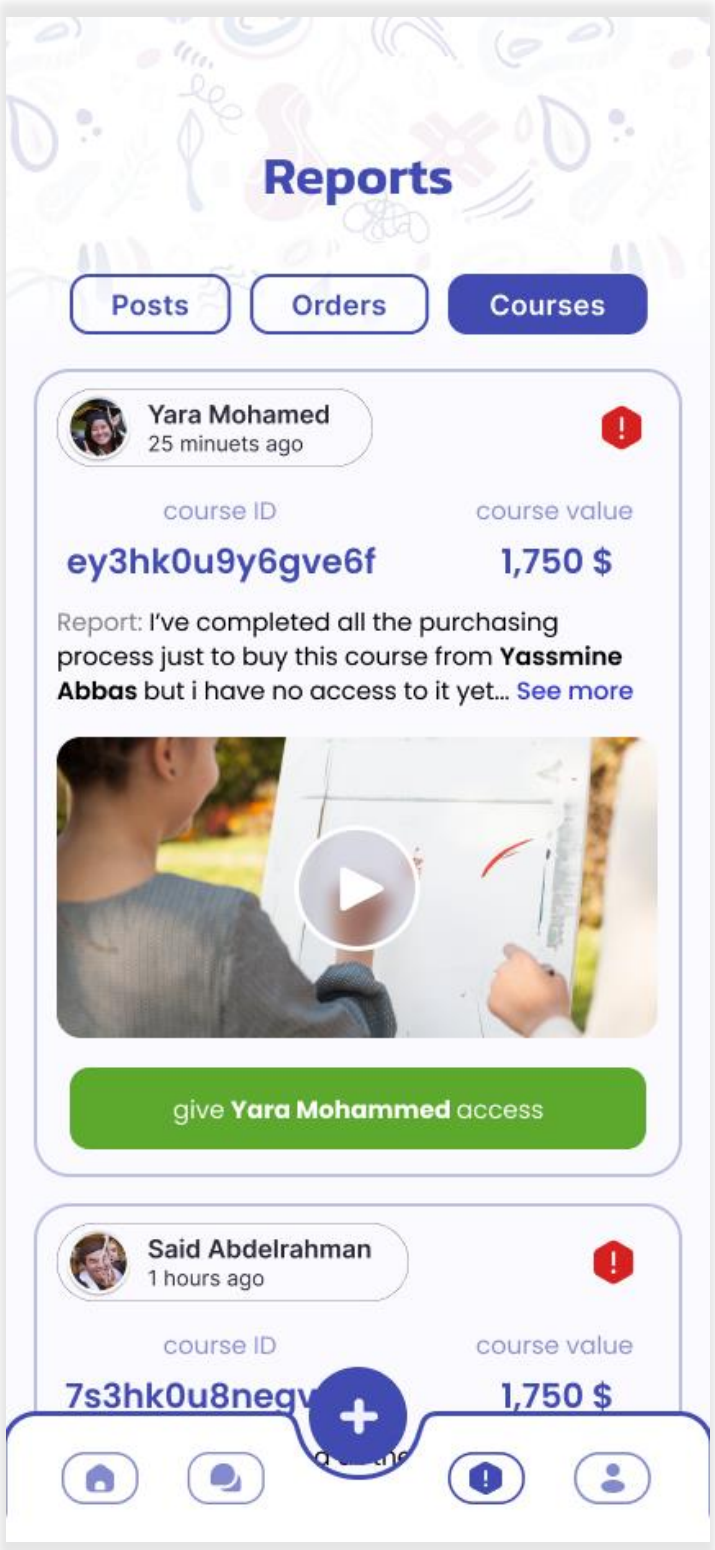


Figure (36) - Courses Reports

List of Users:

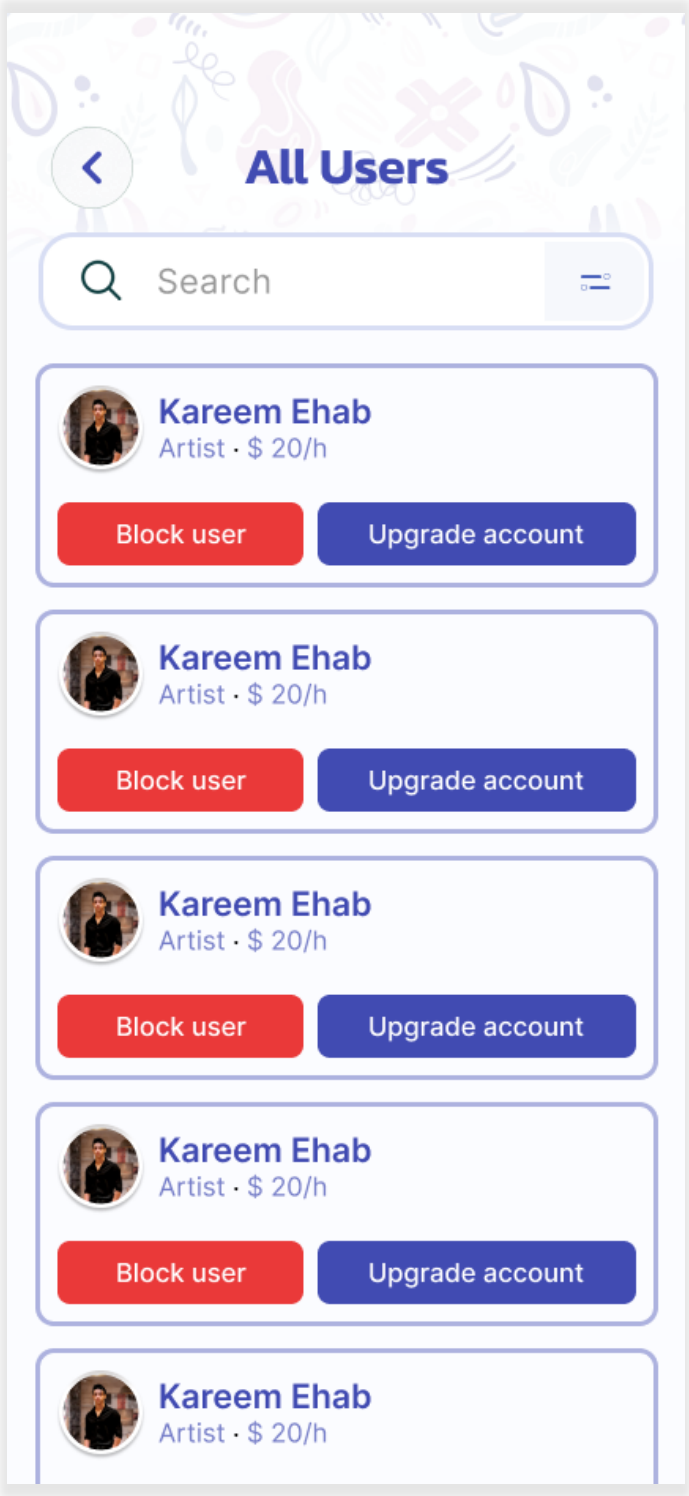


Figure (37) - List of Users

Create Event:

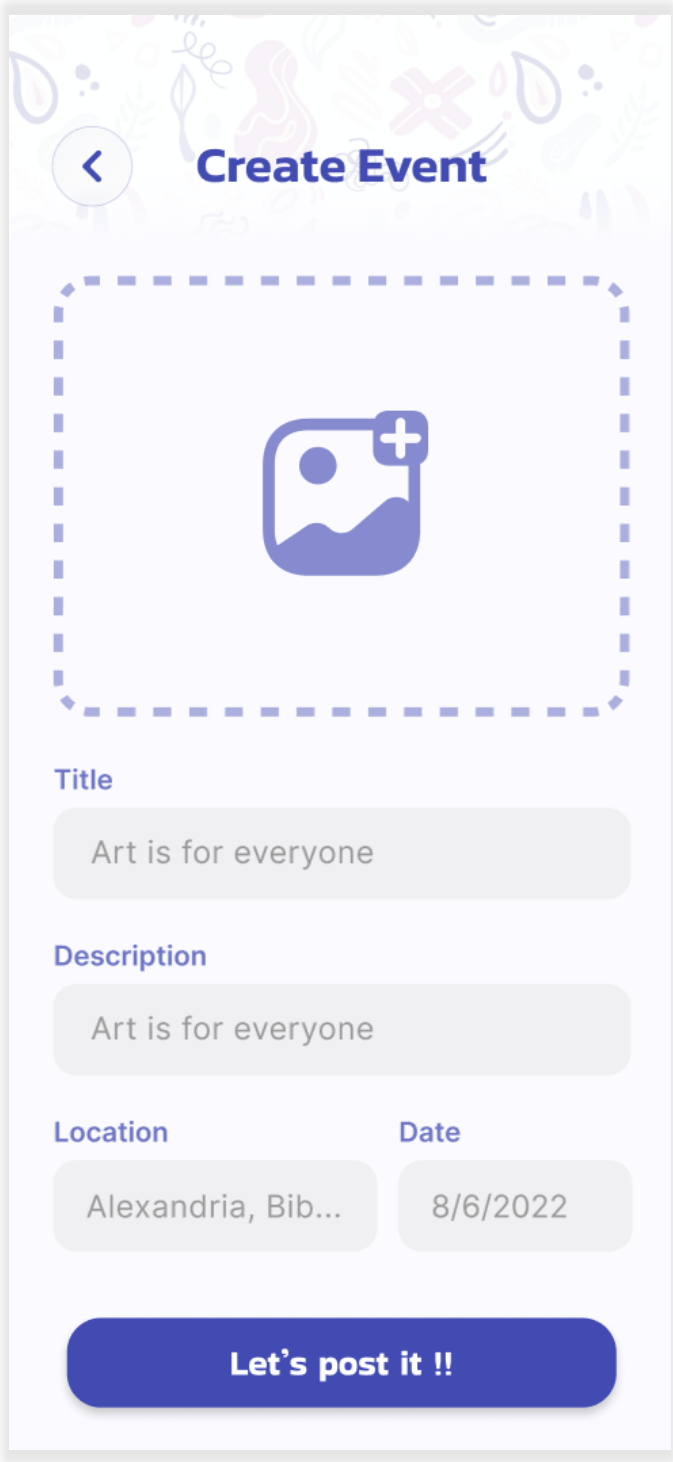


Figure (38) - Create Event

App Database

User:

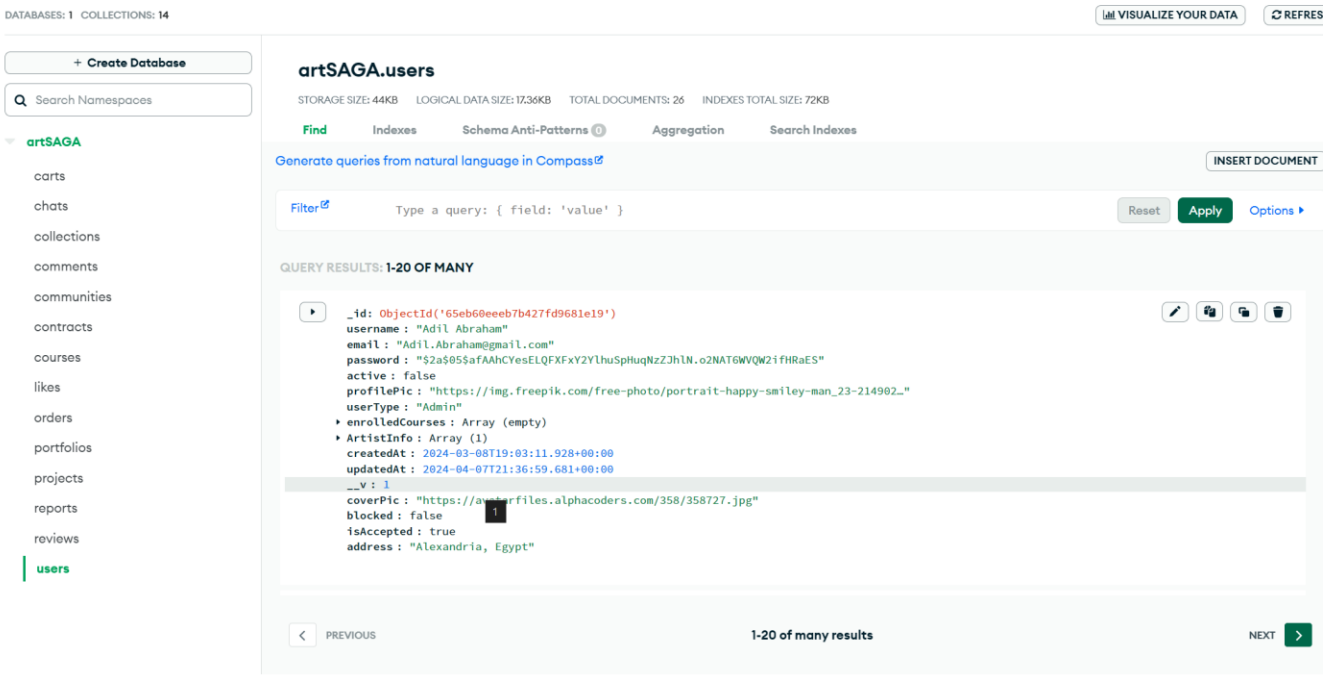


Figure (39) - User Database

Reviews:

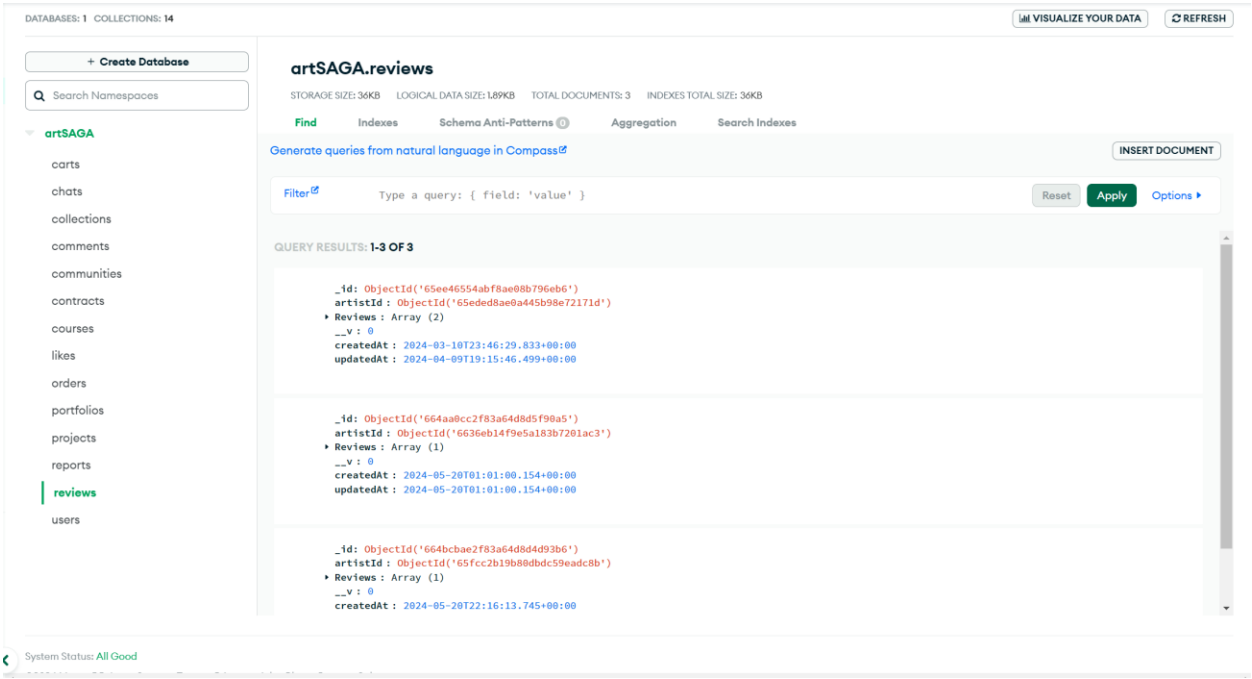


Figure (40) - Reviews Database

Reports:

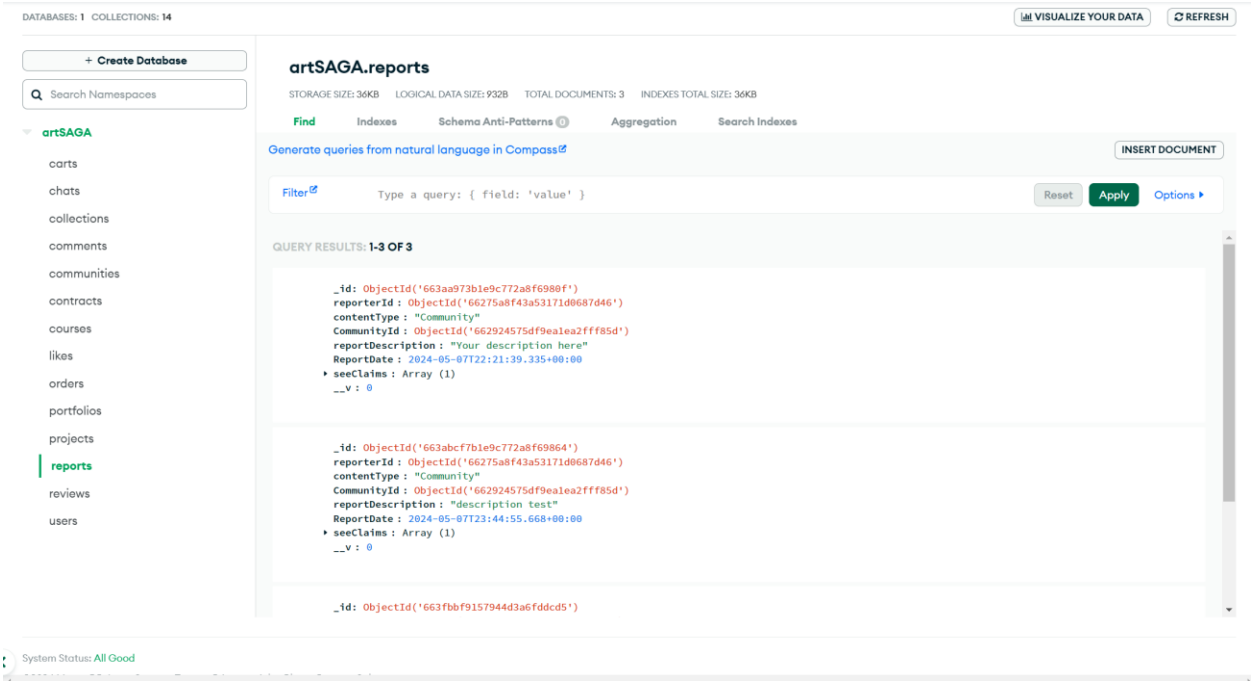


Figure (41) - Reports Database

Projects:

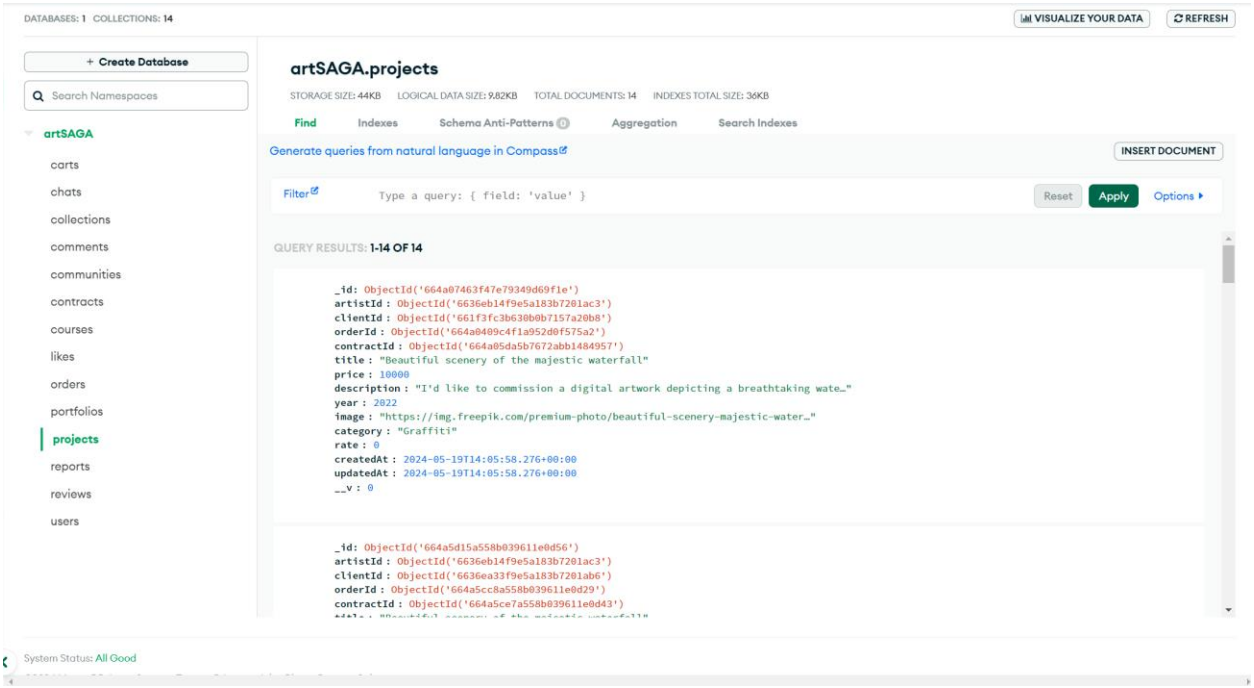


Figure (42) - Projects Database

Portfolio:

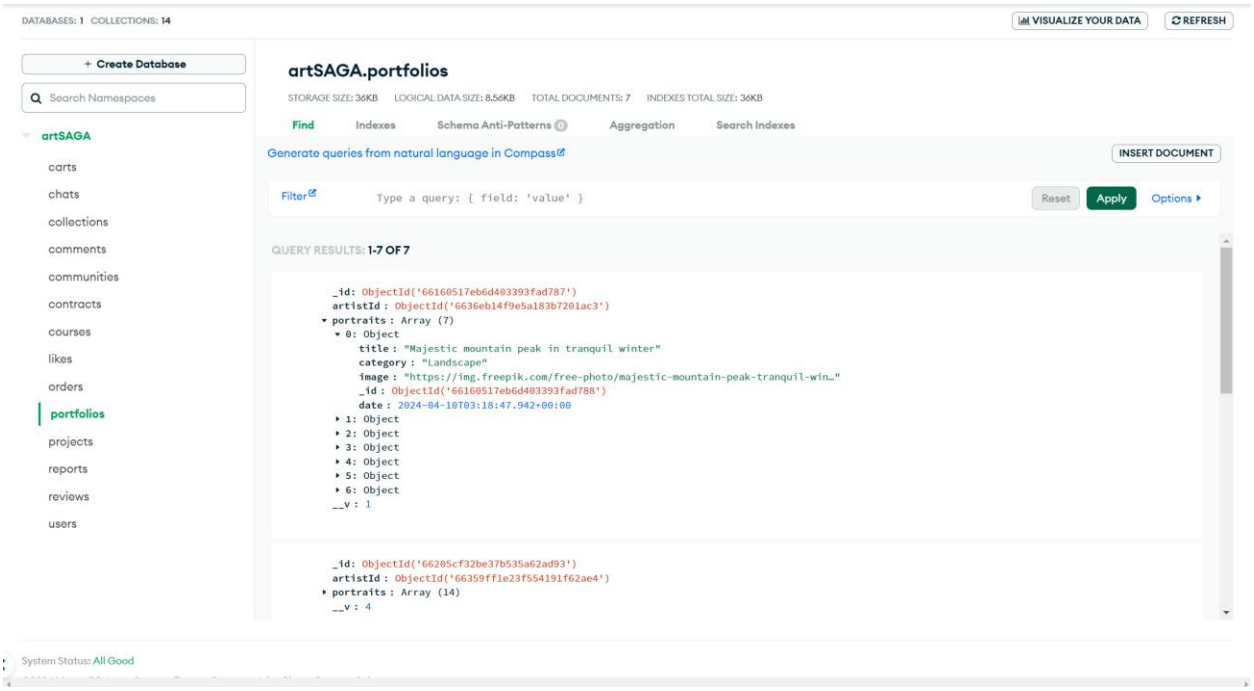


Figure (43) - Portfolio Database

Orders:

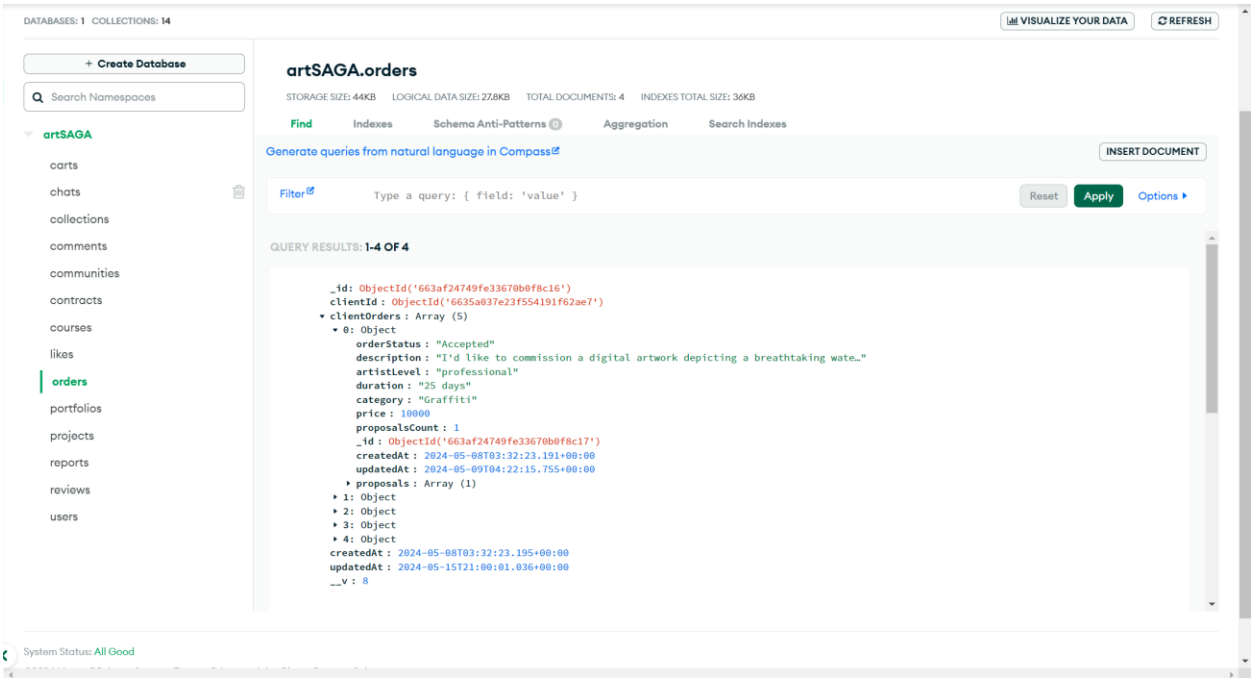


Figure (44) - Orders Database

Likes:

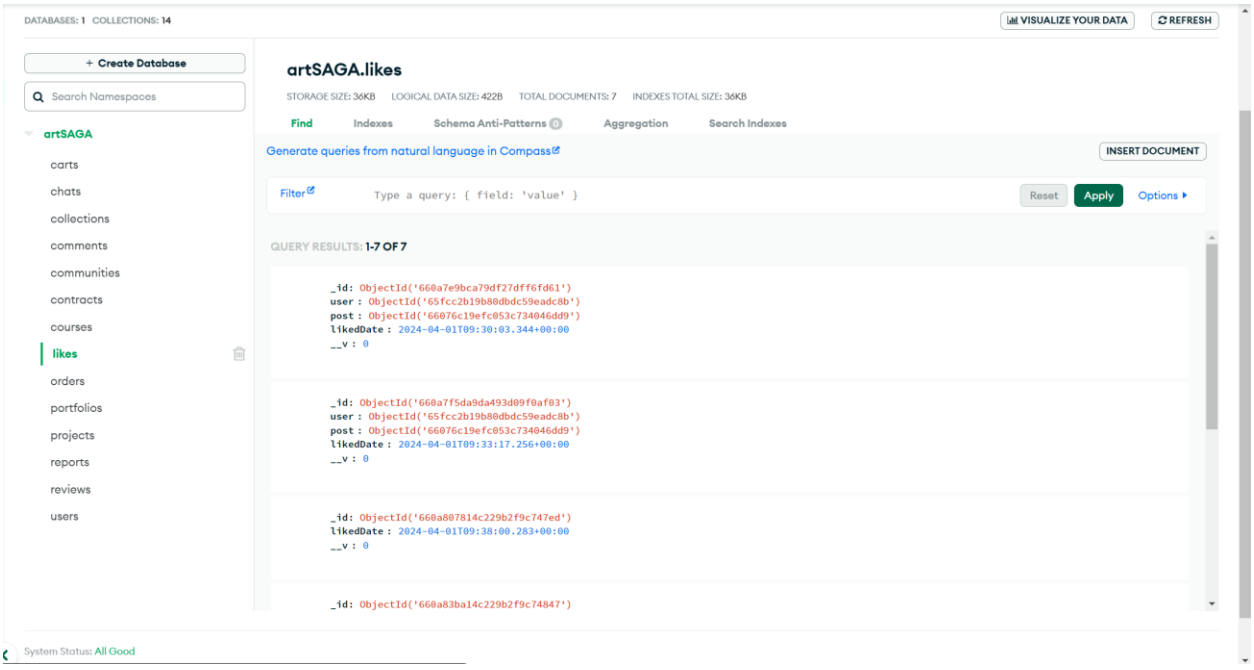


Figure (45) - Likes Database

Courses:

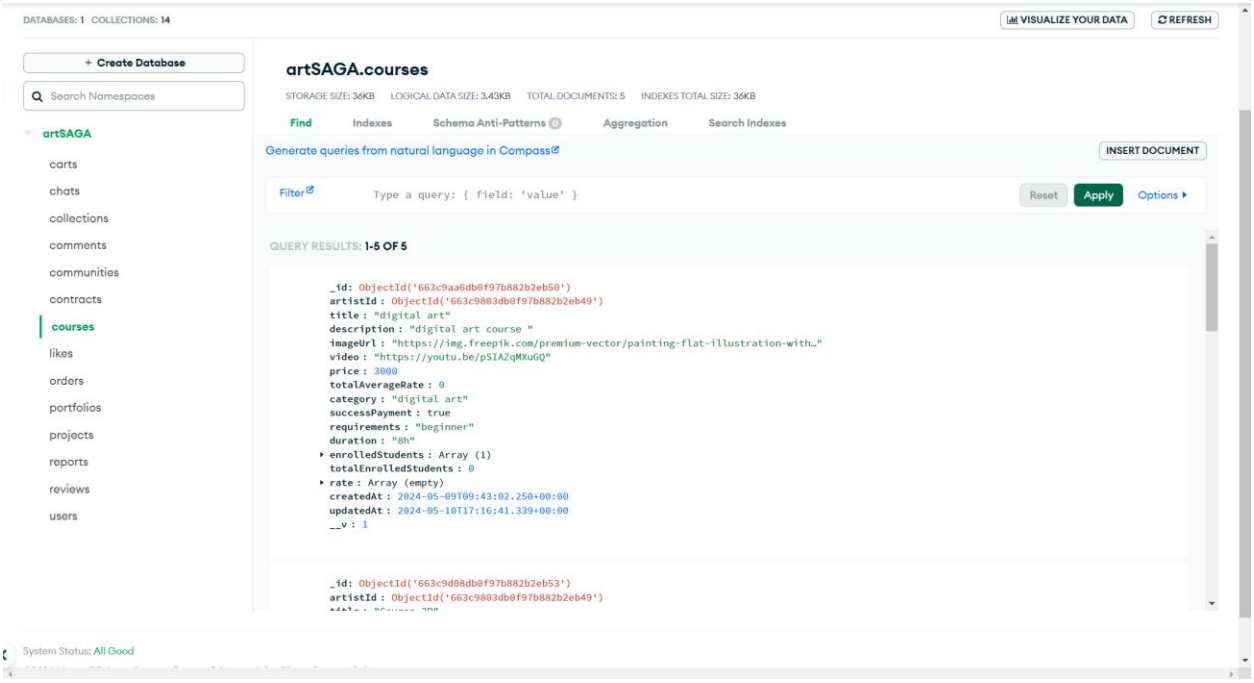


Figure (46) - Courses

Contracts:

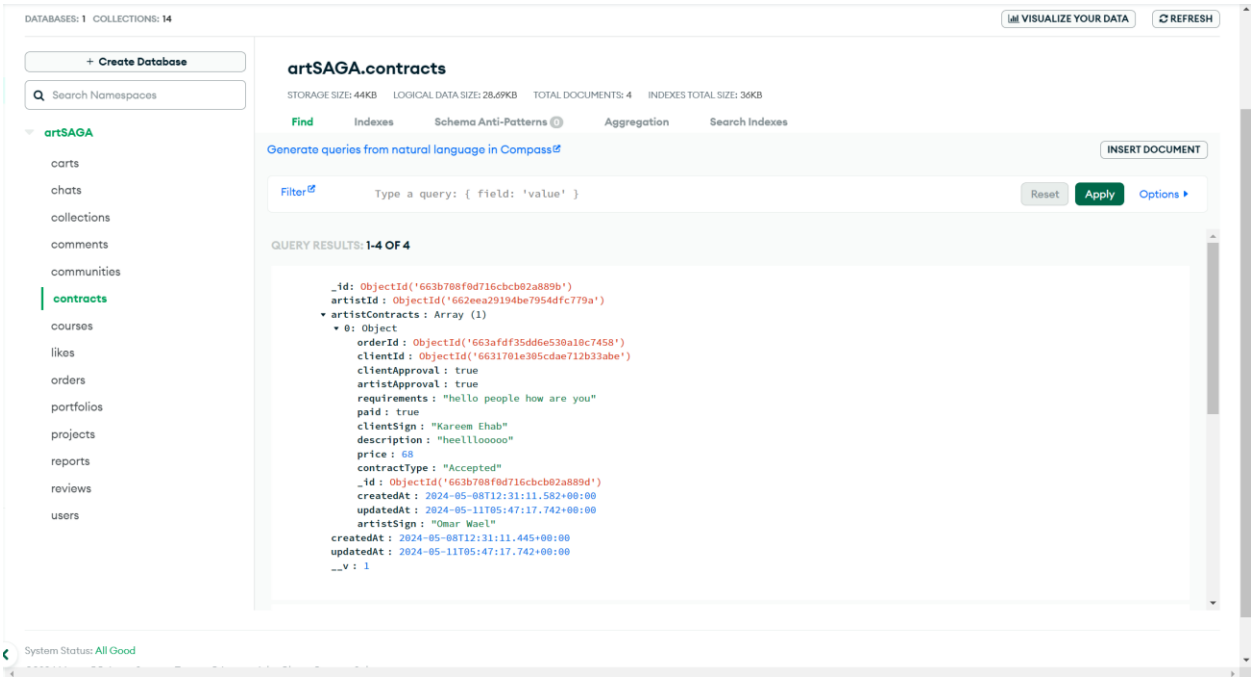


Figure (47) - Contracts

Communities:

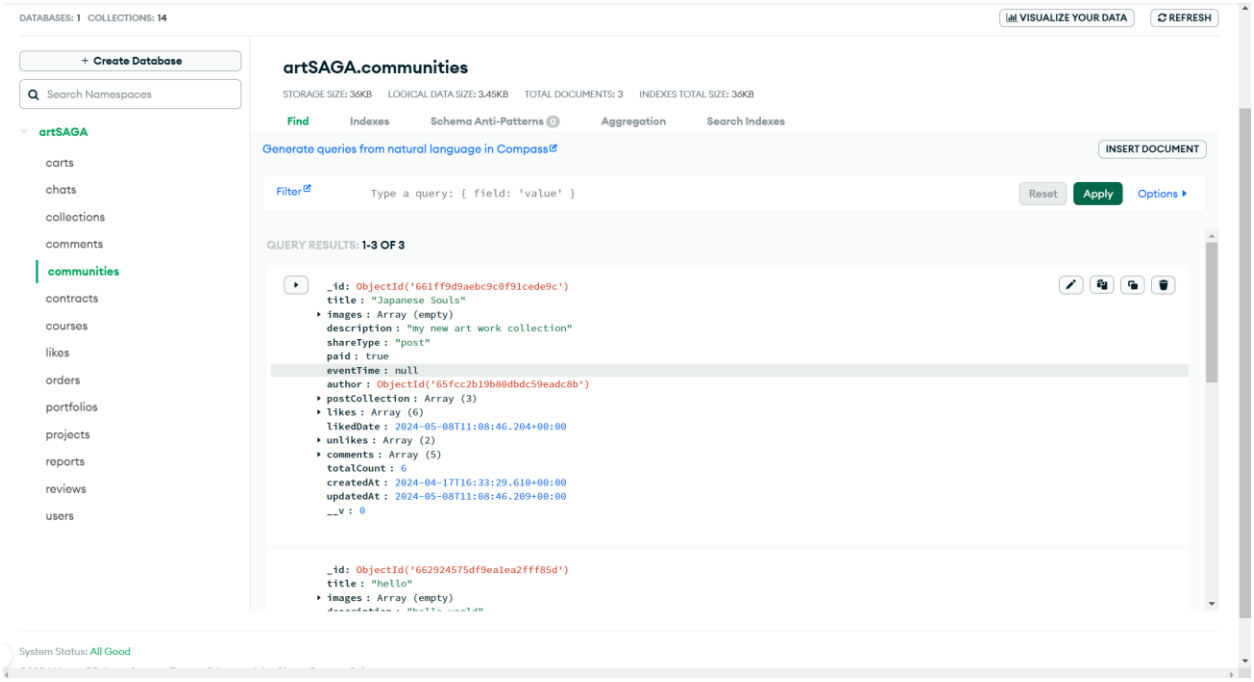


Figure (48) - Communities

Comments:

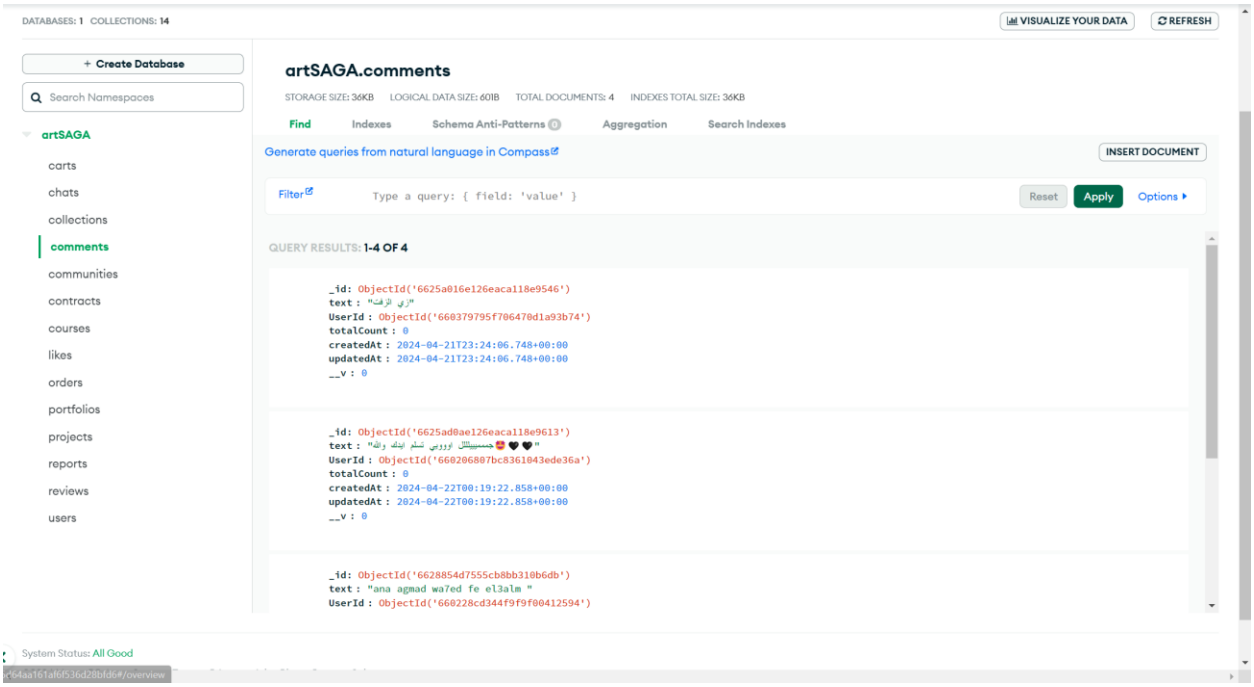


Figure (49) - Comments

Collections:

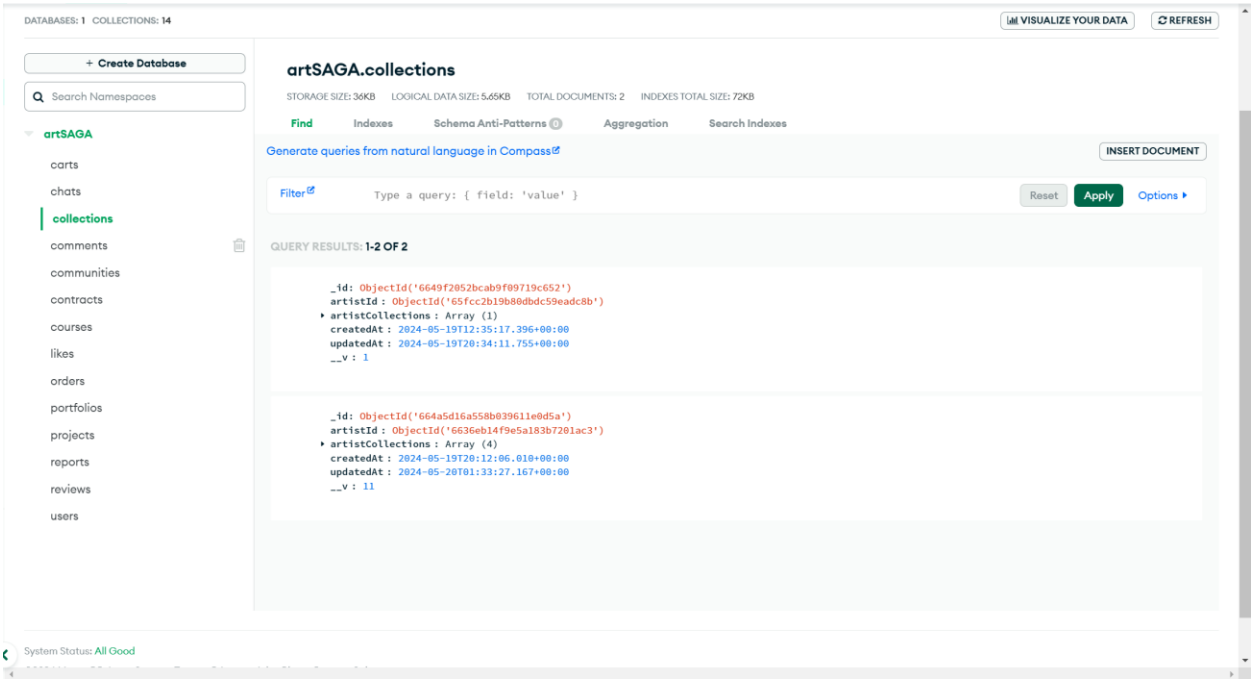


Figure (50) - Collections

Chapter 6

(Conclusion, results and future work)

Conclusion:

Working on our project has taught us a lot. From the initial idea to the final product, we learned about teamwork, effective communication, and managing time, especially under pressure. We want to thank Dr. Yasser Abdel-Ghaffar for his support and valuable guidance. This project not only gave us new knowledge but also built our confidence and teamwork. We believe our Unified Artist Platform will greatly benefit the artistic community by providing innovative tools for networking, learning, and business success.
Thank you for your attention. We welcome your feedback and questions.

Future Work:

- **Enhancing Technical Infrastructure:** Improve the app's technical capabilities to handle a larger number of users and ensure smooth performance.
- **Website Development:** Develop a web version of the platform to complement the mobile application and provide more accessibility options.
- **Expanding Course Offerings:** Include more diverse courses and workshops, covering a wider range of artistic disciplines.
- **Job and Training Suggestions:** Implement features to suggest relevant job opportunities and training programs to artists based on their profiles and interests.
- **Event and Course Recommendations:** Develop algorithms to recommend events and courses to users based on their preferences and activity.
- **Collaboration with Educational Institutions:** Partner with universities and art schools to provide exclusive content and opportunities for students and alumni.
- **Improved Design and User Experience:** Continuously enhance the user interface for better navigation and user experience, making it more intuitive and user-friendly.
- **Enhanced Security Measures:** Strengthen security protocols to protect user data and ensure safe transactions on the platform.
- **Global Expansion:** Extend the platform's reach beyond local artists to include international artists, fostering a global community.

- **Integration with More Payment Systems:** Add support for additional payment methods to accommodate users from different regions.
- **Freelance Milestones:** milestones serve as predefined checkpoints for payment, allowing for the full payment to be made at the beginning of the project and transferred to the app. Upon project completion, the artist receives their funds.
- **Exclusive Course Certificates:** Unlock limitless job opportunities for aspiring artists with our exclusive course certificates! Crafted by seasoned artists, each certificate not only validates your skills but also opens doors to a world of creative opportunities. Elevate your career potential and showcase your talent with pride!
- **AI for Religious Compliance in Art:** We aim to integrate AI into our art application to identify and filter out any artworks that conflict with religious principles, ensuring that the content remains respectful and appropriate.

References

https://bit.ly/45aLPhQ

https:// figma.com/

https://flutter.dev/

https://dart.dev/

https://firebase.google.com/

https://pub.dev/packages/bloc

https://pub.dev/packages

https://nodejs.org/en/

https://www.npmjs.com/

https://www.javascript.com/

https://www.mongodb.com/

https://www.mongodb.com/products/platform/atlas-database/

https://visualstudio.microsoft.com/

https://tensorflow.org/

https://keras.io/

https://numpy.org/

https://matplotlib.org/

https://opencv.org/

https://docs.python.org/3/library/os.html

https://stackoverflow.com/

https://www.edrawmax.com/

https://www.drawio.com/

Appendix: App Code Snapshots:

Front-End:

```dart
class Projects extends StatefulWidget {
  const Projects({
    super.key,
    required this.category,
    required this.index,
    required this.contexts,
  });

  final String category;
  final int index;
  final BuildContext contexts;

  @override
  State<Projects> createState() => _ProjectsState();
}

class _ProjectsState extends State<Projects> {
  late ScrollPhysics physics;
  late List<bool> isExpandedList;
  bool display = false;
  @override
  void initState() {
    super.initState();
    if (widget.index == 0) {
      physics = const NeverScrollableScrollPhysics();
    } else {
      physics = const BouncingScrollPhysics();
    }
    ProjectsCubit.get(context).getAllProjects(widget.category);
  }

  @override
  Widget build(BuildContext context) {
    return BlocConsumer<ProjectsCubit, ProjectsState>(
      listener: (context, state) {
        if (state is GetAllProjectSuccessState) {
          display = !display;
          isExpandedList = List.generate(
              ProjectsCubit.get(context).projectsList.length, (index) => false);
        } else if(state is CreateProjectSuccessState){
          isExpandedList.insert(0,false);
        }
      },
      builder: (context, state) {
        return ConditionalBuilder(
          condition: display,
          builder: (context) {
            return Padding(
              padding: EdgeInsets.symmetric(horizontal: 20.w),
              child: ProjectsCubit.get(context).projectsList.isEmpty
                  ? Center(
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                          SvgPicture.asset(
                            'images/empty_project.svg',
                            width: 350.w,
                          ),
                          Text(
                            'Empty Projects !!',
                            style: TextStyle(
                              color: primaryColor,
                              fontWeight: FontWeight.w800,
                              fontSize: 28.sp,
                            ),
                          ),
                        ],
                      ),
                    )
```

Figure (51) - Projects UI

```dart
// ignore_for_file: avoid_print

import 'package:elysium_2/constants/cache.dart';
import 'package:elysium_2/constants/dio_helper.dart';
import 'package:elysium_2/constants/texts.dart';
import 'package:elysium_2/cubit/auth/signup/sign_up_state.dart';
import 'package:elysium_2/models/user_model.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class SignUpCubit extends Cubit<SignUpState> {
  SignUpCubit() : super(SignUpInitialState());

  static SignUpCubit get(context) => BlocProvider.of(context);

  void userSignUp({
    required String username,
    required String email,
    required String password,
    required String confirmPass,
    required String userType,
    required String userAddress,
    List<Map<String, dynamic>>? artistInfo,
  }) {
    emit(SignUpLoadingState());
    Map<String, dynamic> data = {
      'username': username,
      'email': email,
      'password': password,
      'cpassword': confirmPass,
      'address': userAddress,
      'userType': userType,
    };
    if (userType == 'Artist' && artistInfo != null) {
      data['ArtistInfo'] = artistInfo;
    }

    DioHelper.postData(
      url: 'auth/signUp',
      data: data,
    ).then((value) async {
      if (!value.data['message'].toString().contains('done')) {
        emit(SignUpErrorState(value.data['message']));
        return;
      }
      currentUserModel = UserModel.fromJson(value.data['saveUser']);
      id = value.data['saveUser']['_id'];
      await CacheHelper.saveData(key: 'id', value: id);
      print('id is $id');
      emit(SignUpSuccessState(value.data['saveUser']['_id']));
    }).catchError((error) {
      print(error.toString());
      emit(SignUpErrorState(error.toString()));
    });
  }
}
```

Figure (52) - Create a new account function

```dart
List<PostsModel> postsList = [];
  List<PostsModel> myPostsList = [];
  List<PostsModel> postsLikesList = [];
  List<PostsCommentsModel> commentsList = [];
  List<int> currentImageIndices = [];
  List<PageController> pageControllers = [];

  Future<void> createPost({
    required String title,
    required String description,
    List<int>? price,
    required List<XFile> images,
    required bool isPaid,
  }) async {
    emit(CreatePostLoadingState());

    try {
      List<String> imageUrls = await uploadPostImages('Posts/', images);
      List<Map<String, dynamic>> postCollection = [];
      for (int i = 0; i < imageUrls.length; i++) {
        postCollection.add({
          'price': price != null ? price[i] : 0,
          'images': imageUrls[i],
        });
      }

      var response = await DioHelper.postData(
        url: 'Community/CreatePost',
        data: {
          "title": title,
          "description": description,
          "postCollection": postCollection,
          "shareType": "post",
          "paid": isPaid
        },
        token: 'Bearer__$token',
      );

      dynamic responseData = response.data;
      if (responseData is String) {
        responseData = jsonDecode(responseData);
      }

      if (responseData['message'] == "Post Added successfully") {
        var savedPost;
        if (isPaid) {
          savedPost = responseData['savedPost1'];
        } else {
          savedPost = responseData['savedPost2'];
        }
        print(savedPost.toString());
        var updatedPost = PostsModel.fromJson(savedPost);
        pageControllers.add(PageController());
        currentImageIndices.insert(0, 0);
        postsList.insert(0, updatedPost);
        myPostsList.add(updatedPost);
        emit(CreatePostSuccessState());
      } else {
        emit(CreatePostErrorState());
      }
    } catch (e) {
      print('Error creating post: $e');
      emit(CreatePostErrorState());
    }
  }
```

Figure (53) - Create a new community post function

```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class MessageModel {
  String? id;
  String? toId;
  String? fromId;
  String? message;
  String? read;
  String? type;
  Timestamp? createdAt;

  MessageModel({
    required this.id,
    required this.toId,
    required this.fromId,
    required this.message,
    required this.read,
    required this.type,
    required this.createdAt,
  });

  MessageModel.fromJson(Map<String, dynamic> json) {
    id = json['id'] ?? '';
    toId = json['toId'] ?? '';
    fromId = json['fromId'] ?? '';
    message = json['message'] ?? '';
    read = json['read'] ?? '';
    type = json['type'] ?? '';
    createdAt = json['createdAt'];
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'toId': toId,
      'fromId': fromId,
      'message': message,
      'read': read,
      'type': type,
      'createdAt': createdAt,
    };
  }
}
```

Figure (54) - Model for sending the messages through Firebase

```dart
class ReviewsModel {
  String? username;
  String? profilePic;
  String? comment;
  late DateTime createdAt;

  ReviewsModel({
    required this.username,
    required this.profilePic,
    required this.comment,
    required this.createdAt,
  });

  ReviewsModel.fromJson(Map<String, dynamic> json) {
    username = json["username"];
    profilePic = json["profilePic"];
    comment = json["comment"];
    createdAt = DateTime.parse(json["createdAt"]);
  }
}
```

Figure (55) - Model for reviewing specific artist after his work

```dart
Future<void> paypal({
    required String contractId,
  }) async {
    emit(PayPalLoadingState());
    try {
      await DioHelper.postData(
        url: 'paypal/pay',
        data: {
          "payType": "Order",
          "contractIds": contractId,
        },
        token: 'Bearer__$token',
      ).then((value) {
        if (!value.statusCode.toString().contains('200')) {
          emit(PayPalErrorState());
        }

        print(value.data);
        String paymentLink = value.data['payLink'];
        emit(PayPalSuccessState(paymentLink));
      });
    } catch (e) {
      print('Error creating contract: $e');
      emit(PayPalErrorState());
    }
  }
```

Figure (56) - Call PayPal Gateway function to send payments

```dart
class AdminReports extends StatefulWidget {
  const AdminReports({
    super.key,
    required this.scrollController,
  });

  final ScrollController scrollController;

  @override
  State<AdminReports> createState() => _AdminReportsState();
}

class _AdminReportsState extends State<AdminReports> {
  PageController pageController = PageController();
  int selectedTab = 0;

  List<String> communityTabBar = [
    "Posts",
    "Orders",
    "Courses",
  ];

  List<Widget> communityPages = [];

  @override
  void initState() {
    super.initState();
    communityPages = [
      AdminPosts(scrollController: widget.scrollController),
      AdminOrders(scrollController: widget.scrollController),
      AdminCourses(scrollController: widget.scrollController),
    ];
  }
}
```

Figure (57) - Admin Reports UI

```dart
Future<void> upgradeUser(String userId) async {
    emit(UpgradingLoadingState());
    try {
      var response = await DioHelper.patchData(
        url: 'user/upgrader/$userId',
        token: 'Bearer__$token',
        data: {},
      );
      print(response.data);
      if (response.data['success'] != true) {
        emit(UpgradingErrorState());
        return;
      }
      emit(UpgradingSuccessState());
    } catch (error) {
      print(error.toString());
      emit(UpgradingErrorState());
    }
  }

  void filterUsersByName(String name) {
    final filteredUsers = allUsersList.where((user) =>
        user.username!.toLowerCase().startsWith(name.toLowerCase())).toList();
    emit(FilterUsersState(filteredUsers));
  }
```

Figure (58) - Upgrade user function

```dart
class Projects extends StatefulWidget {
  const Projects({super.key});

  @override
  State<Projects> createState() => _ProjectsState();
}

class _ProjectsState extends State<Projects> {
  List<String> projectCategory = ["Sketches", "Digital", "3D"];

  List<bool> isExpandedList = List.generate(5, (index) => false);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(horizontal: 20.w),
      child: ListView.builder(
        shrinkWrap: true,
        padding: EdgeInsets.zero,
        itemCount: 5,
        physics: const BouncingScrollPhysics(),
        itemBuilder: (context, index) {
          return Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            children: [
              if (index == 0)
                SizedBox(
                  height: 15.h,
                ),
              // arrow up button
              Padding(
                padding: EdgeInsets.symmetric(horizontal: 20.w),
                child: Material(
                  color: primaryColor,
                  borderRadius: BorderRadius.only(
                    topRight: Radius.circular(50.r),
                    topLeft: Radius.circular(50.r),
                  ),
                  child: InkWell(
                    borderRadius: BorderRadius.only(
                      topRight: Radius.circular(50.r),
                      topLeft: Radius.circular(50.r),
                    ),
                    onTap: () {
                      setState(() {
                        isExpandedList[index] = !isExpandedList[index];
                      });
                      print(isExpandedList[index]);
                    },
                    child: AnimatedContainer(
                      duration: const Duration(seconds: 1),
                      curve: Curves.easeInOut,
                      width: 40.w,
                      height: isExpandedList[index] ? 26.h : 0,
                      decoration: BoxDecoration(
                        borderRadius: BorderRadius.only(
                          topRight: Radius.circular(50.r),
                          topLeft: Radius.circular(50.r),
                        ),
                      ),
```

Figure (59) - Posts UI

```dart
class Projects extends StatefulWidget {
  const Projects({super.key});

  @override
  State<Projects> createState() => _ProjectsState();
}

class _ProjectsState extends State<Projects> {
  List<String> projectCategory = ["Sketches", "Digital", "3D"];

  List<bool> isExpandedList = List.generate(5, (index) => false);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(horizontal: 20.w),
      child: ListView.builder(
        shrinkWrap: true,
        padding: EdgeInsets.zero,
        itemCount: 5,
        physics: const BouncingScrollPhysics(),
        itemBuilder: (context, index) {
          return Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            children: [
              if (index == 0)
                SizedBox(
                  height: 15.h,
                ),
              // arrow up button
              Padding(
                padding: EdgeInsets.symmetric(horizontal: 20.w),
                child: Material(
                  color: primaryColor,
                  borderRadius: BorderRadius.only(
                    topRight: Radius.circular(50.r),
                    topLeft: Radius.circular(50.r),
                  ),
                  child: InkWell(
                    borderRadius: BorderRadius.only(
                      topRight: Radius.circular(50.r),
                      topLeft: Radius.circular(50.r),
                    ),
                    onTap: () {
                      setState(() {
                        isExpandedList[index] = !isExpandedList[index];
                      });
                      print(isExpandedList[index]);
                    },
                    child: AnimatedContainer(
                      duration: const Duration(seconds: 1),
                      curve: Curves.easeInOut,
                      width: 40.w,
                      height: isExpandedList[index] ? 26.h : 0,
                      decoration: BoxDecoration(
                        borderRadius: BorderRadius.only(
                          topRight: Radius.circular(50.r),
                          topLeft: Radius.circular(50.r),
                        ),
                      ),
```

Figure (60) - Events UI

```dart
class Courses extends StatefulWidget {
  const Courses({
    Key? key,
    this.scrollController,
  }) : super(key: key);

  final ScrollController? scrollController;

  @override
  State<Courses> createState() => _CoursesState();
}

class _CoursesState extends State<Courses> {
  late DateTime currentTime;

  @override
  void initState() {
    super.initState();
    // Save the current time when the widget is initialized
    currentTime = DateTime.now();
    context.read<CoursesCubit>().fetchCourses();
  }

  String calculateTimeDifference(String createdAtString) {
    final createdAt = DateTime.parse(createdAtString);
    final difference = currentTime.difference(createdAt);
    // Format the time difference as desired
    if (difference.inDays > 0) {
      return '${difference.inDays}d ago';
    } else if (difference.inHours > 0) {
      return '${difference.inHours}h ago';
    } else if (difference.inMinutes > 0) {
      return '${difference.inMinutes}m ago';
    } else {
      return 'Just now';
    }
  }

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<CoursesCubit, CoursesState>(
      builder: (context, state) {
        if (state is CoursesLoading) {
          return Center(
            child: CircularProgressIndicator(),
          );
        } else if (state is CoursesLoaded) {
          return ListView.builder(
            padding: EdgeInsets.zero,
            controller: widget.scrollController,
            physics: const BouncingScrollPhysics(),
            itemCount: state.courses.length,
            shrinkWrap: true,
            itemBuilder: (context, courseIndex) {
              final course = state.courses[courseIndex];
              return Column(
                children: [
                  if (courseIndex == 0) SizedBox(height: 10.h),
                  SizedBox(
                    height: 10.h,
                  ),
```

Figure (61) - Courses UI

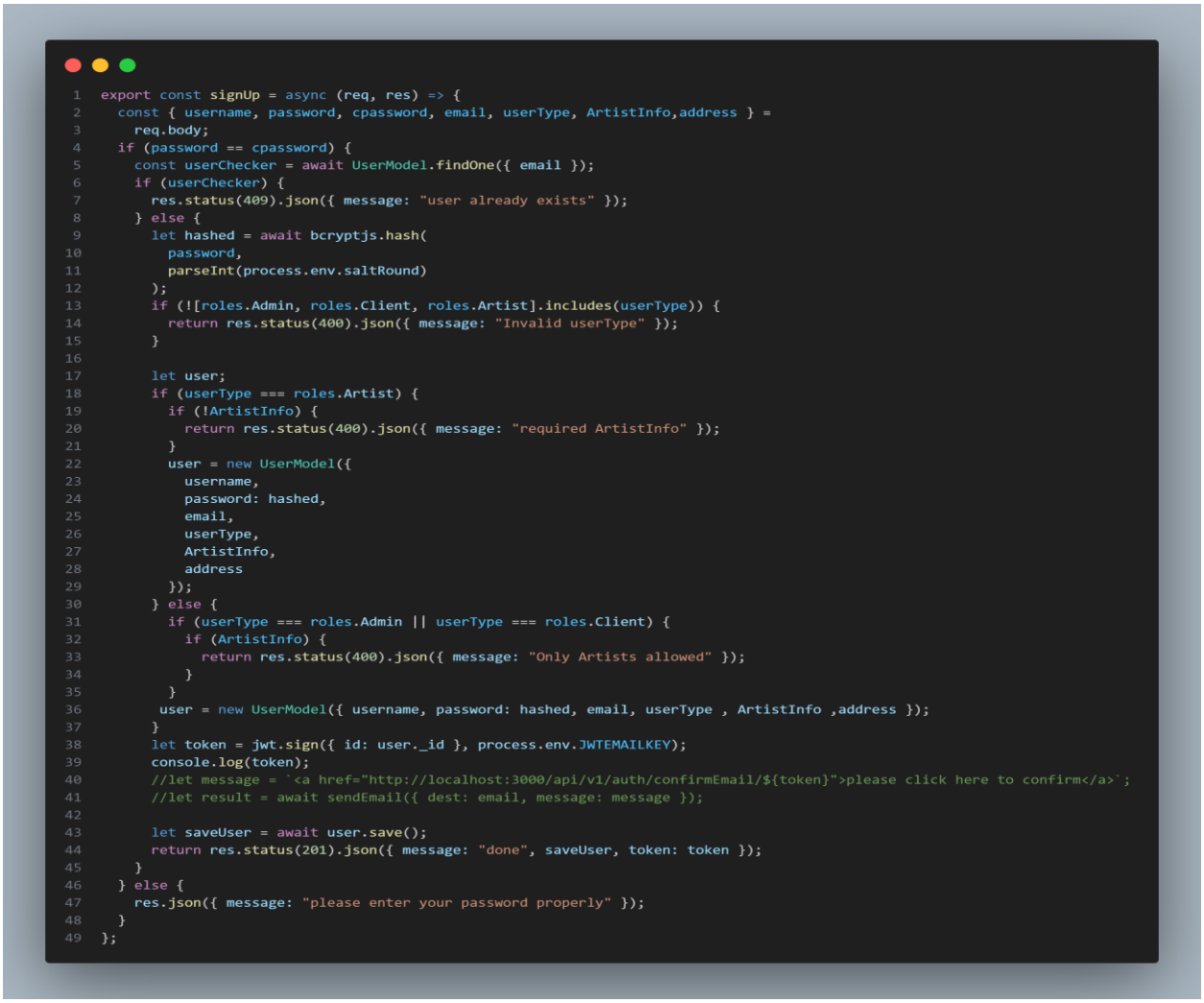
```dart
uture<void> createEvent({
    required String title,
    required String description,
    required String place,
    required String location,
    required DateTime eventDate,
    required String city,
    required DateTime time,
    required List<XFile> images,
  }) async {
    List imageUrls = await uploadEventImages('Events/', images);
    emit(CreateEventLoadingState());
    await DioHelper.postData(
      url: 'Community/CreatePost',
      data: {
        'title': title,
        'description': description,
        'eventDate': eventDate.toString(), // Convert DateTime to string
        'eventPlace': place,
        'eventLocation': location,
        'eventCity': city,
        'eventTime': time.toString(),
        'images': imageUrls,
        'shareType': 'event',
      },
      token: 'Bearer__$token',
    ).then((value) {
      print(value.data);
      if (value.data['message'] != 'Event Added successfully') {
        emit(CreateEventErrorState());
        return;
      }
      var updatedEvent = EventsModel.fromJson(value.data['savedEvent']);
      eventsList.add(updatedEvent);
      myEventsList.add(updatedEvent);
      emit(CreateEventSuccessState());
    }).catchError((e) {
      print('Error creating event: $e');
      emit(CreateEventErrorState());
    });
  }
```

Figure (62) - Create Event Function

```dart
class CoursesCubit extends Cubit<CoursesState> {
  CoursesCubit() : super(CoursesInitial());
  Future<void> fetchCourses() async {
    emit(CoursesLoading());
    try {
      Response response = await DioHelper.getData(
        url: 'course/all',
        token: 'Bearer__$token',
      );

      if (response.data != null && response.data is Map<String, dynamic>) {
        // Extract the 'allCourses' list from the Map
        var allCourses = response.data['allCourses'];

        // Check if 'allCourses' is a List
        if (allCourses is List) {
          // Map each course data to a Course object
          List<Course> courses = allCourses
              .map((courseData) => Course.fromJson(courseData))
              .toList();

          // Emit the CoursesLoaded event with the list of courses
          emit(CoursesLoaded(courses));
        } else {
          emit(CoursesError('Error: Invalid data format'));
        }
      } else {
        emit(CoursesError('Error: Empty or invalid response data'));
      }
    } catch (e) {
      emit(CoursesError(e.toString()));
      print(e);
    }
  }

  void deleteCourse(Course course) async {
    await DioHelper.deleteData(
      url: 'course/${course.id}/delete',
      token: 'Bearer__$token',
    );
    emit(CourseDeleted(course.id ?? ""));
  }
}

```

Figure (63) - Courses Cubit

Back-End:



```
export const signUp = async (req, res) => {
  const { username, password, cpassword, email, userType, ArtistInfo,address } =
    req.body;
  if (password == cpassword) {
    const userChecker = await UserModel.findOne({ email });
    if (userChecker) {
      res.status(409).json({ message: "user already exists" });
    } else {
      let hashed = await bcryptjs.hash(
        password,
        parseInt(process.env.saltRound)
      );
      if (![roles.Admin, roles.Client, roles.Artist].includes(userType)) {
        return res.status(400).json({ message: "Invalid userType" });
      }

      let user;
      if (userType === roles.Artist) {
        if (!ArtistInfo) {
          return res.status(400).json({ message: "required ArtistInfo" });
        }
        user = new UserModel({
          username,
          password: hashed,
          email,
          userType,
          ArtistInfo,
          address
        });
      } else {
        if (userType === roles.Admin || userType === roles.Client) {
          if (ArtistInfo) {
            return res.status(400).json({ message: "Only Artists allowed" });
          }
        }
       user = new UserModel({ username, password: hashed, email, userType , ArtistInfo ,address });
      }
      let token = jwt.sign({ id: user._id }, process.env.JWTEMAILKEY);
      console.log(token);
      //let message = `<a href="http://localhost:3000/api/v1/auth/confirmEmail/${token}">please click here to confirm</a>`;
      //let result = await sendEmail({ dest: email, message: message });

      let saveUser = await user.save();
      return res.status(201).json({ message: "done", saveUser, token: token });
    }
  } else {
    res.json({ message: "please enter your password properly" });
  }
};
```

Figure (64) - User Sign Up with role-based registration and authentication

```
const { paid } = req.body;

    if (paid === true) {
      let { price } = req.body;

      let post = new CommunityModel({
        title,
        description,
        author,
        postCollection,
        price,
        paid,
        shareType,
      });

      post.populate({
        path: "author",
        select: "username profilePic userType",
      });

      let savedPost1 = await post.save();

      if (!savedPost1) {
        res
          .status(400)
          .json({ message: "Error while inserting to the Database" });
      } else {
        res
          .status(201)
          .json({ message: "Post Added successfully", savedPost1 });
      }
    } else {
      let post = new CommunityModel({
        title,
        description,
        author,
        images,
        shareType,
      });
      post.populate({
        path: "author",
        select: "username profilePic userType",
      });

      let savedPost2 = await post.save();

      if (!savedPost2) {
        res
          .status(400)
          .json({ message: "Error while inserting to the Database" });
      } else {
        res
          .status(201)
          .json({ message: "Post Added successfully", savedPost2 });
      }
    }
  }
  } catch (error) {
    console.log(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
};

```

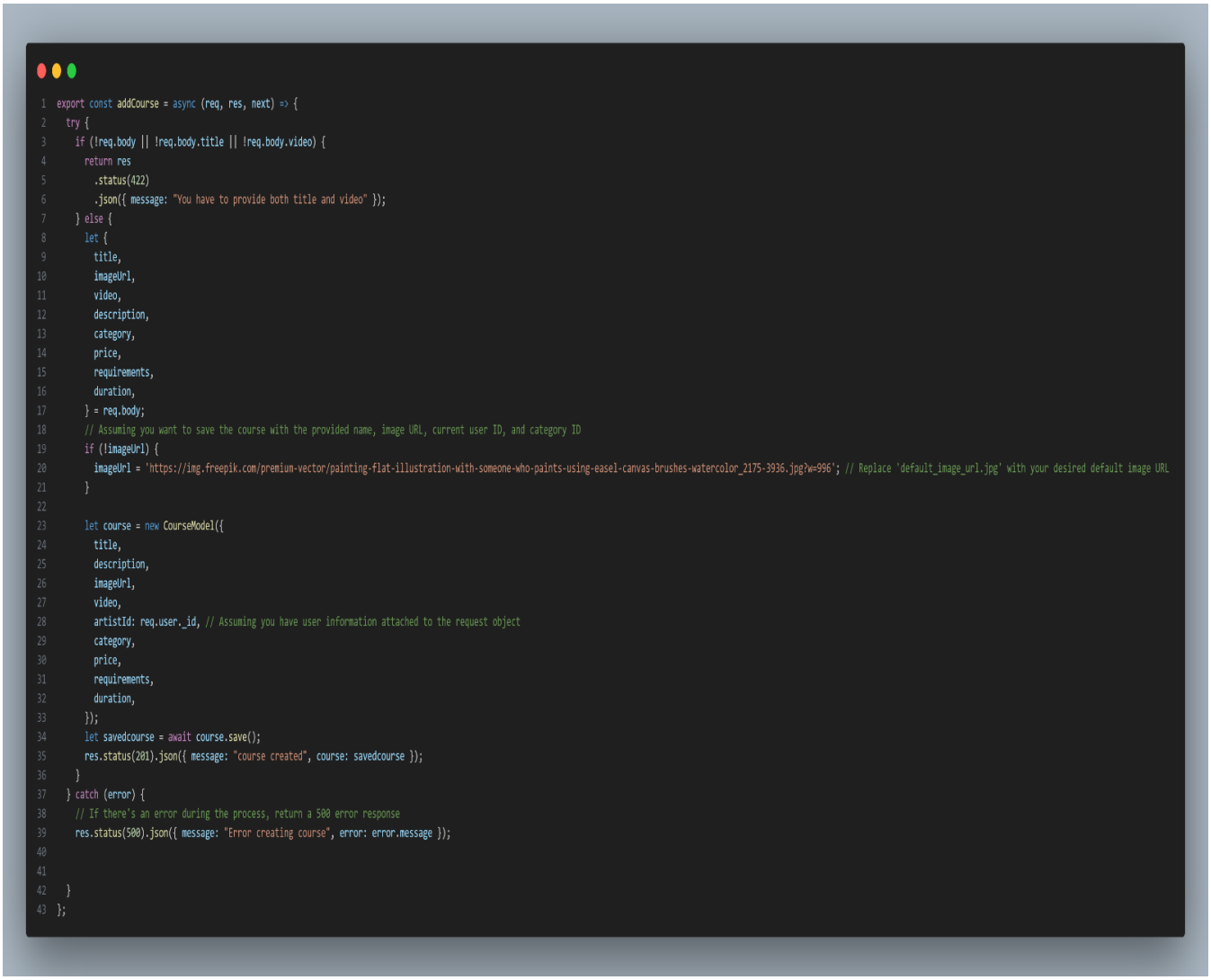
Figure (65) - Create post function

```
export const createArtistPost = async (req, res) => {
  try {
    const author = req.user._id;
    let { images, title, description, shareType, postCollection } = req.body;

    if (shareType === "event") {
      let { eventDate, eventLocation, eventPlace, eventTime, eventCity } =
        req.body;
      let event = new CommunityModel({
        title,
        description,
        images,
        author,
        eventDate,
        eventPlace,
        eventTime,
        eventCity,
        eventLocation,
        shareType,
      });
      event.populate({
        path: "author",
        select: "username profilePic userType",
      });

      let savedEvent = await event.save();

      if (!savedEvent) {
        res
          .status(400)
          .json({ message: "Error while inserting to the Database" });
```

Figure (66) - Create event function

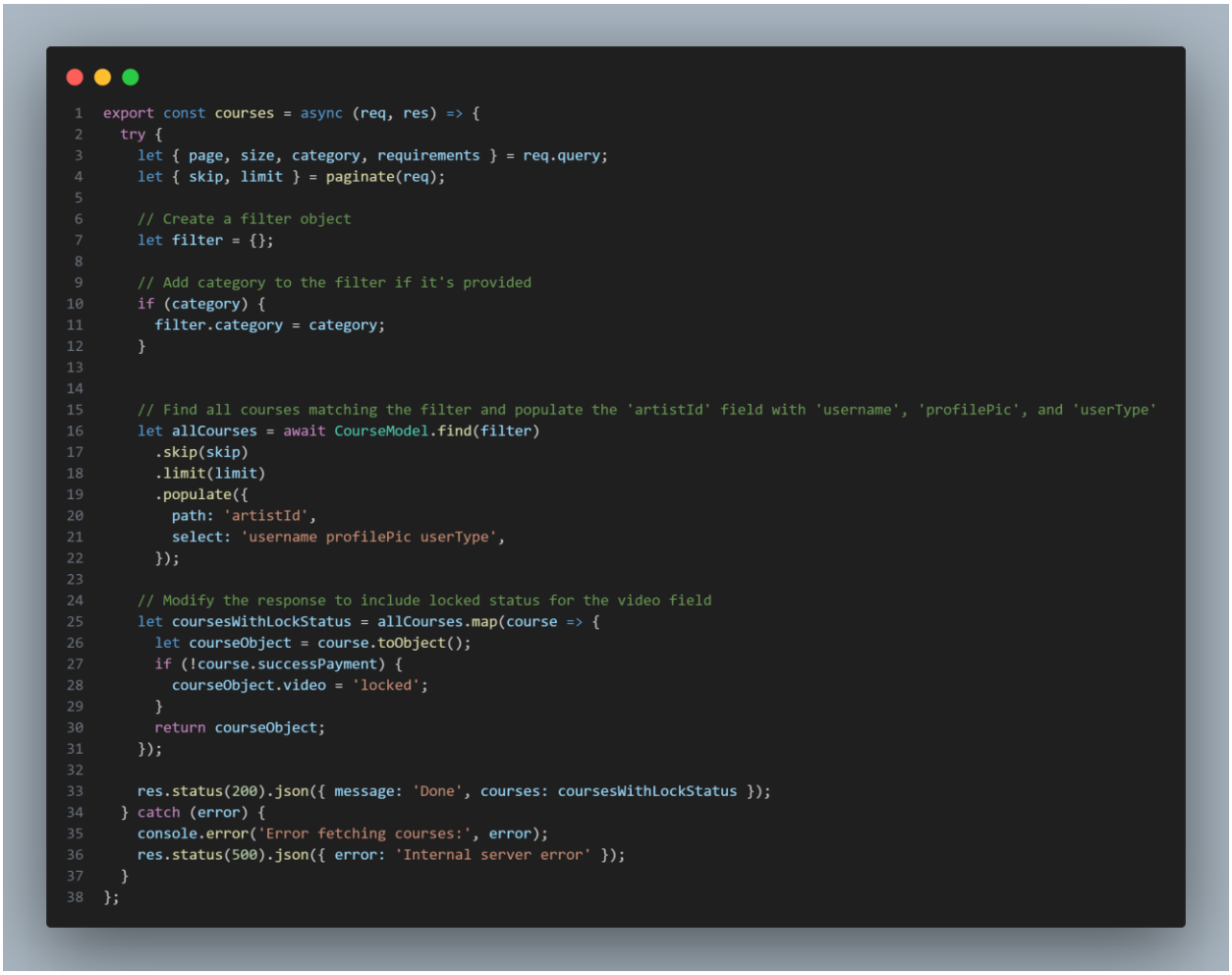
```
export const addComment = async (req, res) => {
  try {
    let { text } = req.body;
    let UserId = req.user._id;
    let { id } = req.params;
    const find = await CommunityModel.findById(id, UserId);
    if (!find) {
      return res.status(404).json({ message: "Post not found" });
    } else {
      let comment = await CommentsModel({ text, UserId });
      let savedComment = await comment.save();
      const data = await CommunityModel.findByIdAndUpdate(
        id,
        {
          $push: { comments: savedComment._id },
        },
        { new: true }
      )
        .populate({
          path: "comments",
          select: "text UserId likes unlikes createdAt",
          populate: {
            path: "UserId",
            select: "username profilePic userType",
          },
        })
        .populate({
          path: "author",
          select: "username profilePic userType",
        })
        .populate({
          path: "likes",
          select: "username profilePic userType",
        })
        .populate({
          path: "unlikes",
          select: "username profilePic userType",
        });
      return res.status(201).json({ message: "Added", data: data });
    }
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: "Error" });
  }
};

```

Figure (67) - Adding comments function



```
export const addCourse = async (req, res, next) => {
  try {
    if (!req.body || !req.body.title || !req.body.video) {
      return res
        .status(422)
        .json({ message: "You have to provide both title and video" });
    } else {
      let {
        title,
        imageUrl,
        video,
        description,
        category,
        price,
        requirements,
        duration,
      } = req.body;
      // Assuming you want to save the course with the provided name, image URL, current user ID, and category ID
      if (!imageUrl) {
        imageUrl = 'https://img.freepik.com/premium-vector/painting-flat-illustration-with-someone-who-paints-using-easel-canvas-brushes-watercolor_2175-3936.jpg?w=996'; // Replace 'default_image_url.jpg' with your desired default image URL
      }

      let course = new CourseModel({
        title,
        description,
        imageUrl,
        video,
        artistId: req.user._id, // Assuming you have user information attached to the request object
        category,
        price,
        requirements,
        duration,
      });
      let savedcourse = await course.save();
      res.status(201).json({ message: "course created", course: savedcourse });
    }
  } catch (error) {
    // If there's an error during the process, return a 500 error response
    res.status(500).json({ message: "Error creating course", error: error.message });


  }
};
```

Figure (68) - Add course function



```
export const courses = async (req, res) => {
  try {
    let { page, size, category, requirements } = req.query;
    let { skip, limit } = paginate(req);

    // Create a filter object
    let filter = {};

    // Add category to the filter if it's provided
    if (category) {
      filter.category = category;
    }


    // Find all courses matching the filter and populate the 'artistId' field with 'username', 'profilePic', and 'userType'
    let allCourses = await CourseModel.find(filter)
      .skip(skip)
      .limit(limit)
      .populate({
        path: 'artistId',
        select: 'username profilePic userType',
      });

    // Modify the response to include locked status for the video field
    let coursesWithLockStatus = allCourses.map(course => {
      let courseObject = course.toObject();
      if (!course.successPayment) {
        courseObject.video = 'locked';
      }
      return courseObject;
    });

    res.status(200).json({ message: 'Done', courses: coursesWithLockStatus });
  } catch (error) {
    console.error('Error fetching courses:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
};
```

Figure (69) - Get all courses function

```
export const enrollInCourse = async (req, res) => {
  try {
    const { userId, courseId } = req.body;

    // Find the user by userId
    const user = await UserModel.findById(userId);
    if (!user) {
      return res.status(404).json({ error: "User not found." });
    }

    // Find the course by courseId and populate the artistId field with artist details
    const course = await CourseModel.findById(courseId)
      .populate({
        path: "artistId",
        select: "username profilePic" // Select only the username and profilePic fields of the artist
      });

    if (!course) {
      return res.status(404).json({ error: "Course not found." });
    }

    // Check if the user is already enrolled in the course
    if (course.enrolledStudents.includes(userId)) {
      return res.status(400).json({ error: "User is already enrolled in the course." });
    }

    // Add the user to the course's list of enrolled students
    course.enrolledStudents.push(userId);
    await course.save();

    res.status(200).json({
      message: "User enrolled in the course successfully.",
      user: {
        userId: user._id,
        username: user.username,
        profilePic: user.profilePic
      },
      course: course // Return the entire course object with populated artist details
    });

  } catch (error) {
    console.error("Error while enrolling user in the course:", error);
    res.status(500).json({ error: "Internal server error." });
  }
};
```

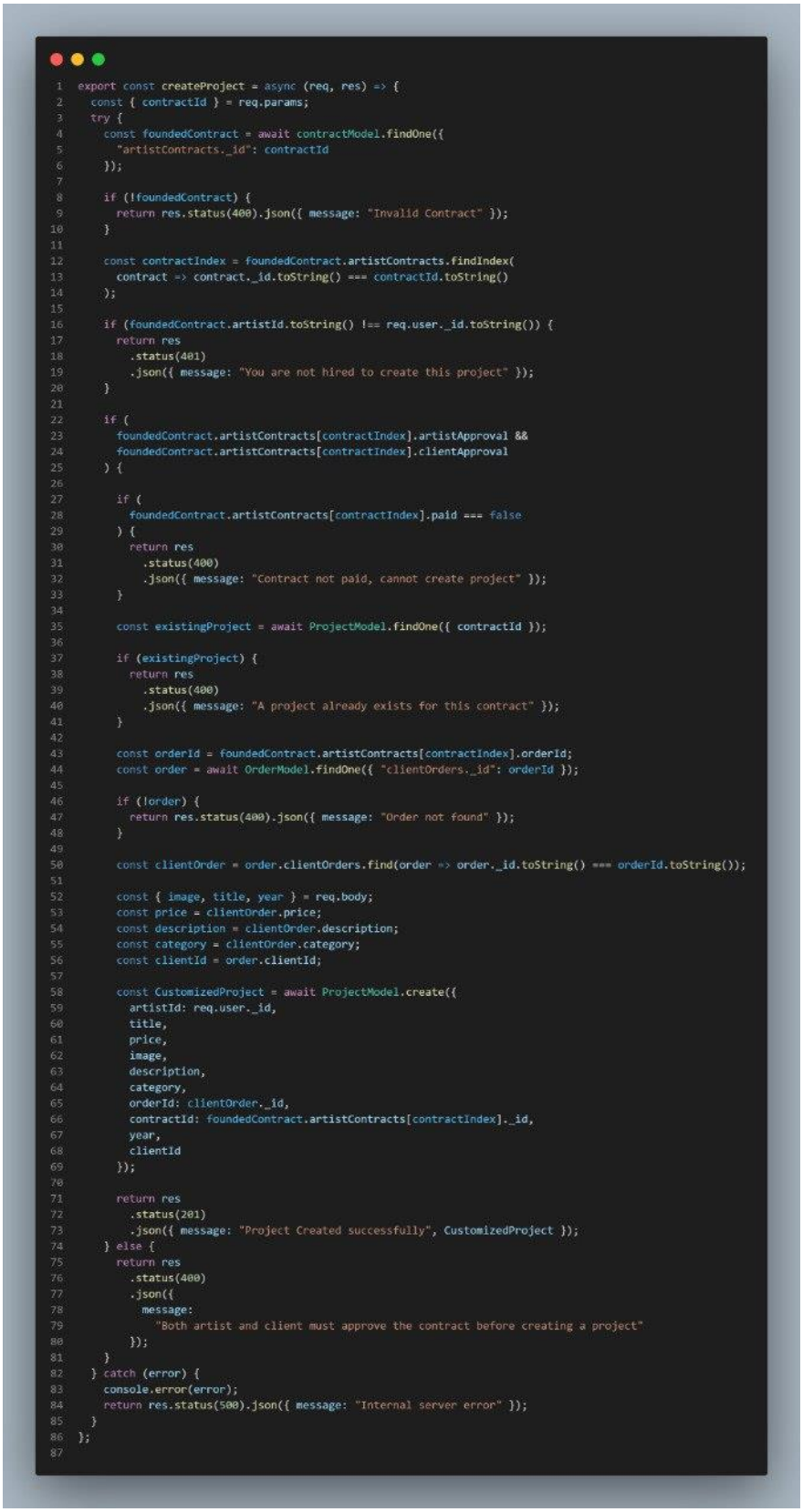
Figure (70) - Enroll in course function

```
export const createOrder = async (req, res) => {
  try {
    let { description, artistLevel, duration, category, price, attachment } = req.body;

    const validCategories = [
      "Landscape",
      "Calligraphy",
      "3D",
      "Anime/Manga",
      "Graffiti",
      "Digital",
      "Sketching",
      "Surreal",
      "abstract",
    ];

    if (!validCategories.includes(category)) {
      return res.status(400).json({ message: "Invalid category" });
    }

    const validArtistLevels = ["Beginner", "Intermediate", "professional"];

    if (!validArtistLevels.includes(artistLevel)) {
      return res.status(400).json({ message: "Invalid artist level" });
    }

    const clientId = req.user._id;

    let client = await OrderModel.findOne({ clientId }).populate({
      path: 'clientId',
      select: '_id username profilePic address',
    });

    if (!client) {
      const user = await UserModel.findById(clientId);
      const createdOrder = await OrderModel.create({
        clientId: req.user._id,
        clientOrders: [{
          description,
          artistLevel,
          duration,
          category,
          price,
          attachment,
        }],
      });

      const savedOrder = await createdOrder.save();

```

Figure (71) - Create Order

```
export const createContract = async (req, res) => {
  const { orderId } = req.params;
  const { requirements } = req.body;

  try {
    const foundedOrder = await OrderModel.findOne({
      "clientOrders._id": orderId,
    });

    if (!foundedOrder) {
      return res.json({ message: "Order not found" });
    }

    const clientOrder = foundedOrder.clientOrders.find(
      (order) => order._id.toString() === orderId
    );
    if (foundedOrder.clientId.toString() !== req.user._id.toString()) {
      return res.json({
        message: "You are not allowed to create this Contract",
      });
    }

    const hiredArtist = clientOrder.proposals.find(
      (proposal) => proposal.hired === true
    );

    if (!hiredArtist) {
      return res.json({ message: "No artist is hired for this order yet" });
    }

    const artistId = hiredArtist.artistId;
    const existingContract = await contractModel.findOne({
      artistContracts: {
        $elemMatch: {
          orderId: orderId,
          contractType: { $nin: ["Rejected"] }
        }
      }
    });

    if (existingContract) {
      return res.json({ message: "Contract already exists" });
    }

    let artistContract = await contractModel.findOne({ artistId });

    if (!artistContract) {
      artistContract = await contractModel.create({
        artistId: artistId,
        artistContracts: [],
      });
    }

    const clientDetails = await UserModel.findById(foundedOrder.clientId);

    const newContract = {
      orderId,
      clientId: foundedOrder.clientId,
      clientApproval: true,
      requirements,
      clientSign: clientDetails.username,
      description: clientOrder.description,
      price: clientOrder.price,
    };

    artistContract.artistContracts.push(newContract);

    const savedContract = await artistContract.save();

    const newContractId = artistContract.artistContracts[artistContract.artistContracts.length - 1]._id;
    newContract.contractId = newContractId ;

    if (!savedContract) {
      return res
        .status(400)
        .json({ message: "Error while inserting to the Database" });
    }
    res
      .status(200)
      .json({ message: "Contract created successfully", newContract });
  } catch (error) {
    console.error("Error creating contract:", error);
    res.status(500).json({ message: "Internal server error" });
  }
};

```

Figure (72) - Create Contract



```
export const createProject = async (req, res) => {
  const { contractId } = req.params;
  try {
    const foundedContract = await contractModel.findOne({
      "artistContracts._id": contractId
    });

    if (!foundedContract) {
      return res.status(400).json({ message: "Invalid Contract" });
    }

    const contractIndex = foundedContract.artistContracts.findIndex(
      contract => contract._id.toString() === contractId.toString()
    );

    if (foundedContract.artistId.toString() !== req.user._id.toString()) {
      return res
        .status(401)
        .json({ message: "You are not hired to create this project" });
    }

    if (
      foundedContract.artistContracts[contractIndex].artistApproval &&
      foundedContract.artistContracts[contractIndex].clientApproval
    ) {

      if (
        foundedContract.artistContracts[contractIndex].paid === false
      ) {
        return res
          .status(400)
          .json({ message: "Contract not paid, cannot create project" });
      }

      const existingProject = await ProjectModel.findOne({ contractId });

      if (existingProject) {
        return res
          .status(400)
          .json({ message: "A project already exists for this contract" });
      }

      const orderId = foundedContract.artistContracts[contractIndex].orderId;
      const order = await OrderModel.findOne({ "clientOrders._id": orderId });

      if (!order) {
        return res.status(400).json({ message: "Order not found" });
      }

      const clientOrder = order.clientOrders.find(order => order._id.toString() === orderId.toString());

      const { image, title, year } = req.body;
      const price = clientOrder.price;
      const description = clientOrder.description;
      const category = clientOrder.category;
      const clientId = order.clientId;

      const CustomizedProject = await ProjectModel.create({
        artistId: req.user._id,
        title,
        price,
        image,
        description,
        category,
        orderId: clientOrder._id,
        contractId: foundedContract.artistContracts[contractIndex]._id,
        year,
        clientId
      });

      return res
        .status(201)
        .json({ message: "Project Created successfully", CustomizedProject });
    } else {
      return res
        .status(400)
        .json({
          message:
            "Both artist and client must approve the contract before creating a project"
        });
    }
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: "Internal server error" });
  }
};
```

Figure (73) - Create Project

```
export const createPortfolio = async (req, res) => {
  try {
    const existingPortfolio = await PortfolioModel.findOne({ artistId: req.user._id });
    if (existingPortfolio) {
      return res.status(400).json({ message : 'Portfolio already exists for this artist' });
    }
    const { portraits } = req.body;
    if (!portraits || !Array.isArray(portraits) || portraits.length === 0) {
      return res.status(400).json({ message : 'Portraits array is required' });
    }

    const validCategories = [
      "Landscape",
      "Calligraphy",
      "3D",
      "Anime/Manga",
      "Graffiti",
      "Digital",
      "Sketching",
      "Surreal",
      "abstract",
    ];

    const invalidPortraits = portraits.filter(portrait => !validCategories.includes(portrait.category));
    if (invalidPortraits.length > 0) {
      return res.status(400).json({ message : 'Invalid category found in portraits array' });
    }

    const portfolio = new PortfolioModel({
      artistId: req.user._id,
      portraits: portraits.map(portrait => ({
        title: portrait.title,
        category: portrait.category,
        image: portrait.image
      }))
    });

    await portfolio.save();
    res.status(201).json({ message: 'Portfolio created successfully', portfolio });
  } catch (error) {
    console.error('Error creating portfolio:', error);
    res.status(500).json({ message : 'Internal Server Error' });
  }
};
```

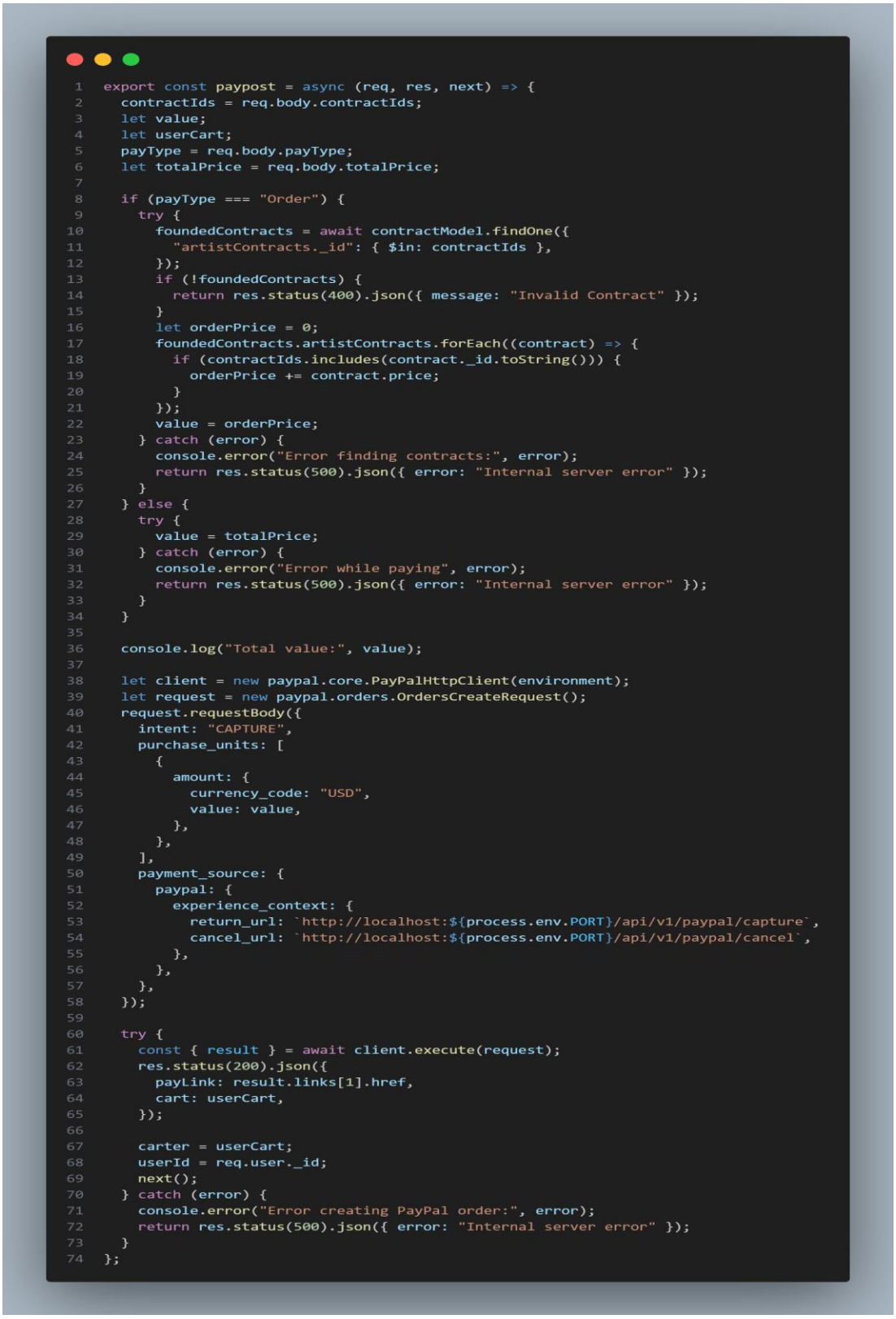
Figure (74) - Create Portfolio

```
export const deletePortfolio = async (req, res) => {
  try {
    const { portfolioId } = req.params;
    if (!portfolioId) {
      return res.status(400).json({ message : 'Portfolio ID is required' });
    }
    const portfolio = await PortfolioModel.findOne({
      _id: portfolioId,
     // artistId: req.user._id,
    });

    if (!portfolio) {
      return res.status(404).json({ message : 'Portfolio not found' });
    }

    if (portfolio.artistId.toString() !== req.user._id.toString()) {
      return res.json({ message: "you are not authorized to delete this portfolio" });
    }

    const deletedPortfolio = await PortfolioModel.findOneAndDelete({ _id: portfolioId });
    await PortfolioModel.deleteOne({ _id: portfolioId });

    res.json({ message: 'Portfolio deleted successfully', deletedPortfolio });
  } catch (error) {
    console.error('Error deleting portfolio:', error);
    res.status(500).json({ message : 'Internal Server Error' });
  }
};
```

Figure (75) - Delete Portfolio

```
export const getAllReports = async (req, res) => {
  try {
    const { contractId } = req; // Accessing contractId from req
    console.log(contractId);
    const reports = await reportModel.find({}).populate([
      {
        path: "reporterId",
        select: "username profilePic",
      },
      {
        path: "CommunityId",
        populate: { path: "author" },
      },
      {
        path: "ContractId",
        populate: {
          path: "artistId",
          select: "username profilePic",
        },
      },
      {
        path: "CourseId",
        populate: {
          path: "artistId",
          select: "username profilePic",
        },
      },
    ]);

    if (!reports) {
      return res.status(404).json({ message: "No reports found" });
    }
    console.log(contractIds);
    reports.map((report) => {
      if (report.ContractId && report.ContractId.artistContracts) {
        let contract;
        for (const contractId of contractIds) {
          contract = report.ContractId.artistContracts.find(
            (contract) => contract._id.toString() === contractId
          );
        }
        report.ContractId.artistContracts = contract ? [contract] : [];
      }
    });

    return res.status(200).json({ message: "successful", Reports: reports });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
};

```

Figure (76) - Get all reports function



```
export const paypost = async (req, res, next) => {
  contractIds = req.body.contractIds;
  let value;
  let userCart;
  payType = req.body.payType;
  let totalPrice = req.body.totalPrice;

  if (payType === "Order") {
    try {
      foundedContracts = await contractModel.findOne({
        "artistContracts._id": { $in: contractIds },
      });
      if (!foundedContracts) {
        return res.status(400).json({ message: "Invalid Contract" });
      }
      let orderPrice = 0;
      foundedContracts.artistContracts.forEach((contract) => {
        if (contractIds.includes(contract._id.toString())) {
          orderPrice += contract.price;
        }
      });
      value = orderPrice;
    } catch (error) {
      console.error("Error finding contracts:", error);
      return res.status(500).json({ error: "Internal server error" });
    }
  } else {
    try {
      value = totalPrice;
    } catch (error) {
      console.error("Error while paying", error);
      return res.status(500).json({ error: "Internal server error" });
    }
  }

  console.log("Total value:", value);

  let client = new paypal.core.PayPalHttpClient(environment);
  let request = new paypal.orders.OrdersCreateRequest();
  request.requestBody({
    intent: "CAPTURE",
    purchase_units: [
      {
        amount: {
          currency_code: "USD",
          value: value,
        },
      },
    ],
    payment_source: {
      paypal: {
        experience_context: {
          return_url: `http://localhost:${process.env.PORT}/api/v1/paypal/capture`,
          cancel_url: `http://localhost:${process.env.PORT}/api/v1/paypal/cancel`,
        },
      },
    },
  });

  try {
    const { result } = await client.execute(request);
    res.status(200).json({
      payLink: result.links[1].href,
      cart: userCart,
    });

    carter = userCart;
    userId = req.user._id;
    next();
  } catch (error) {
    console.error("Error creating PayPal order:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
};
```

Figure (77) - Payment processing with PayPal Integration using NPM

AI Model:

```
Click here to ask Blackbox to help you code faster
from tensorflow.keras.metrics import Precision, BinaryAccuracy, Recall  # type: ignore


Click here to ask Blackbox to help you code faster
pre = Precision() # how pecise
acc = BinaryAccuracy() # how classification is going
re = Recall() #how the recognized data and pixel filters are doing throughout the neural network
```

Figure (78) - extraction of confusion matrix elements

```
Click here to ask Blackbox to help you code faster
# 1)filter numbers    2)3*3 pixels in size for each filter 3)stride by 1
#                !   @ @  #
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape = (256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())


model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Figure (79) - Overview of Convolutional Neural Network Architecture for Image Classification

```
[ ]

Click here to ask Blackbox to help you code faster
model.compile('adam', loss = tf.losses.BinaryCrossentropy(), metrics= ['accuracy'])
[ ]

Click here to ask Blackbox to help you code faster
model.summary()
[ ]
```

Figure (80) - Detailed CNN Architecture with Adam Optimizer and Learning Rate Configuration

```
Click here to ask Blackbox to help you code faster
fig = plt.figure()
plt.plot(hist.history['loss'],color = 'green', label = 'loss')
plt.plot(hist.history['val_loss'], color = 'purple', label = 'val_loss')
fig.suptitle('Total Loss', fontsize = 20)
plt.legend(loc ="upper left")
plt.show()
[ ]

Click here to ask Blackbox to help you code faster
fig = plt.figure()
plt.plot(hist.history['accuracy'],color = 'teal', label = 'accuracy')
plt.plot(hist.history['val_accuracy'], color = 'orange', label = 'val_accuracy')
fig.suptitle('Accuracy', fontsize = 20)
plt.legend(loc ="upper left")
plt.show()
```

Figure (81) - Graphing Model Performance Over Time Using Matplotlib's Pyplot Module

```
Click here to ask Blackbox to help you code faster
if yhat > 0.5:
    print(f'predicted class has a human presence')
else:
    print(f'predicted class has no human presence')
```

Saving the model

```
Click here to ask Blackbox to help you code faster
#API and stuffing
from tensorflow.keras.models import load_model # type: ignore
```

Figure (82) - Binary Prediction Inference Using Model Prediction Variable yhat

```
model_path ='models/converted_v2_fine_tuned.tflite'

interpreter = tf.lite.Interpreter(model_path)
interpreter.resize_tensor_input(0, [1, 255, 255, 3])
interpreter.allocate_tensors()
interpreter.set_tensor(0, image)

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print (input_details)
print (output_details)
input_shape = input_details[0]['shape']
print(input_shape)

input_data = image

interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

output_data_tflite = interpreter.get_tensor(output_details[0]['index'])
print(output_data_tflite)

#print(output_data_tflite[0][1])

average = output_data_tflite[0][0] + output_data_tflite[0][1] + output_data_tflite[0][2]/3

print (average)

if (average > 0.85):
    print("there is human presence")
else:
    print("no human presence")
```

Figure (83) - Implementing Average Inference Logic in Python and Java with Gradle Integration

```
Click here to ask Blackbox to help you code faster
import numpy as np
from matplotlib import pyplot as plt
[ ]

Click here to ask Blackbox to help you code faster
data = tf.keras.utils.image_dataset_from_directory('data')
[ ]
```

Data iterating and how we are going to iterate and pass from batch to another using numpy

```
Click here to ask Blackbox to help you code faster
data_iterator = data.as_numpy_iterator()
[ ]

Click here to ask Blackbox to help you code faster
batch = data_iterator.next()
[ ]

Click here to ask Blackbox to help you code faster
batch
[ ]

Click here to ask Blackbox to help you code faster
batch[1]
```

Figure (84) - Batch Iteration process using NumPy iterator module

```
Click here to ask Blackbox to help you code faster
import tensorflow as tf
import os

Run and Debug (Ctrl+Shift+D)

Click here to ask Blackbox to help you code faster
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
[ ]

Click here to ask Blackbox to help you code faster
import cv2
import imghdr
[ ]

Click here to ask Blackbox to help you code faster
data_dir = 'data'
[ ]

Click here to ask Blackbox to help you code faster
os.listdir(data_dir)
[ ]

Click here to ask Blackbox to help you code faster
image_exts = ['jpeg','jpg','bmp','png']
```

Figure (85) - The initialization of data pipeline for training