PhoneArt - MathArt IOS Application

Aditya Prakash, Noah Huang, Emmett De Bruin, Zachary Letcher
Paul Phillips
Doctor Saturnino Garcia

<<PhoneArt - MathArt IOS Application>>

Project Summary

The MathArt App is an iOS mobile application inspired by the MathArt Playground website (https://mathart.us/ng-playground/). This app seeks to simplify the process of creating visually stimulating images by enabling users to manipulate shapes, colors, and transformations through mathematical functions. Beyond merely providing tools for artistic expression, the app aims to fill an educational gap by making the intersection of art and math more accessible and engaging. By fostering creativity and learning, the MathArt App addresses the challenge of introducing mathematical principles in a visually intuitive and user-friendly manner.

Problem Statement

The MathArt App addresses the challenge of creating accessible and engaging tools that merge art and mathematics. Many existing creative tools have steep learning curves, making them less accessible to educators, students, and hobbyists. The app seeks to solve this by offering an intuitive platform that allows users to generate complex artistic designs with minimal effort. By simplifying interactions with math-based art creation, the app lowers barriers for those with limited technical or mathematical expertise.

The broader goal is to provide a platform for artistic expression and learning that benefits educators, students, and creative hobbyists alike. This project will have a significant impact by making math concepts more tangible and engaging, fostering creativity in educational settings, and providing a unique tool for exploring the interplay of math and art. Ultimately, the app serves as a bridge between two traditionally separate domains, offering practical and educational value.

Stakeholders

Client/Customer Stakeholders:

Paul Phillips

Description: Creator of the MathArt Playground website and conceptual initiator of the MathArt App. While Paul is not directly involved in the app's technical development, his vision drives its goals.

Goals:

- Ensure the app aligns with the website's core functionality and vision.
- Expand the MathArt platform to mobile users.

Pain Points:

- Limited technical expertise in iOS development may challenge collaboration.
- Ensuring the mobile app maintains the quality and scope of the original web platform.

End User (User Roles) Stakeholders:

Hobbyists/Mathematicians

Description: Individuals with an interest in art and math who wish to explore mathematical transformations creatively.
Goals:

- Create visually appealing artwork using mathematical transformations.

Pain Points:

- Limited understanding of technical jargon or tools might hinder usage.
- Ensuring real-time responsiveness of sliders and transformations.

STEAM Academy Instructors

Description: Educators aiming to introduce math concepts through creative tools in their classrooms.
Goals:

- Demonstrate the app's features effectively to enhance student engagement.
- Use the app to connect abstract mathematical concepts with practical application.

Pain Points:

- Students may struggle with the app's features without adequate training materials.
- Difficulty incorporating the app into existing curricula seamlessly.

STEAM Academy Students

Description: Students learning math concepts such as geometric transformations, who will use the app to explore and apply these concepts creatively.
Goals:

- Develop a deeper understanding of math concepts through creative exploration.
- Enjoy the process of creating art while learning.

Pain Points:

- Initial learning curve with the app interface.
- Potential frustration with understanding complex mathematical transformations.

Requirements

Functional Requirements: User Stories

User role: Client

1. As a Client,
   I want to customize artwork parameter with intuitive controls
   so I can adjust elements like size, position, and rotation to create unique MathArt.
   - **Must Have - Completed**

2. As a Client,
   I want access to various geometric shapes
   so I can create different types of MathArt compositions.

   - **Must Have - Completed**

3. As a Client,
   I want to select from a range of colors
   so I can make artwork that is visually appealing and customizable.

   - **Must Have - Completed**

User role: STEAM Academy Instructor

4. As an Instructor,
   I want students to be able to upload their artwork to a shared class folder
   so I can easily access, organize, and review my students' work for assessments and class discussions.

   - **Nice to Have**

5. As an Instructor
   I want to be able to comment on students' artwork directly within the app
   so I can provide constructive feedback and encourage learning.

   - **Nice to Have**

User role: STEAM Academy Students

6. As a Student,
   I want to be able to share my artwork in multiple ways (e.g., social media, print, email)
   so I can show it to my friends and family.

   - **Nice to Have - In Progress**

7. As a Student,
   I want to access a tutorial page that explains the different art translation functions (like Skew, Zoom, etc.),
   so I can better understand how to use these features independently.

   - **Must Have**

8. As a Student,
   I want access to a variety of creative tools and options,
   so I can explore and create different types of artwork.

   - **Must Have**

Non-Functional Requirements

Current Implementation Status :

Performance:

- Maintains 60 FPS during transformations
- Smooth real-time updates
- Efficient memory management

Reliability:

- Robust error handling for invalid inputs
- State preservation during view updates
- Crash-free operation during testing

Usability:

- Intuitive dual-input controls
- Responsive touch interface
- Clear visual feedback

Coding and Testing:

The app will be developed in Swift, leveraging Xcode for live testing and visualization during development. This approach ensures the app functions smoothly on iOS devices and maintains a user-friendly interface.

Ethical and Accessibility Considerations

Ethical Considerations

For our MathArt App, our primary ethical concern revolves around inclusivity and equal access for all users, regardless of disabilities or unique challenges. We acknowledge that each user, whether considered an “edge case” or not, deserves full engagement and enjoyment from our app. To meet this commitment, we will implement user-centered design practices, such as engaging diverse user groups during regular testing sessions and incorporating feedback loops to continually refine the app's features. By prioritizing inclusivity in every stage of development, we ensure that our app’s design embraces the entire spectrum of users.

Consent and data use are equally important in our ethical framework. While our app does not heavily rely on personal data collection, any data collected will be handled transparently, with clear consent processes in place. Users will be informed when and how their data is collected, including an option to opt-out at any time. We will also provide a simple method for users to request data deletion. For instance, if a user’s artwork is featured in promotional materials, explicit consent will be obtained beforehand, and they will retain the right to revoke permission. These measures emphasize our commitment to respecting user privacy and using data solely to enhance their experience.

Accessibility Considerations

Our MathArt app prioritizes accessibility in several key ways, ensuring adaptability for users with varying abilities. One primary focus is visual accessibility, particularly for users who are colorblind or have other visual impairments. To address this, we will integrate features such as customizable color contrast options, alternative color schemes, and compatibility with Apple's built-in accessibility settings, like VoiceOver and Dynamic Type. Additionally, we will use algorithms to detect and suggest optimal color modes for users based on their preferences.

For users with cognitive impairments, the app will feature a simplified and intuitive interface with large buttons, visual cues, and customizable display options to reduce cognitive load. The onboarding process will include step-by-step tutorials designed to assist first-time users, ensuring they feel confident navigating the app.

To support users with limited vision, we will incorporate text-to-speech functionalities and enlargeable interface elements. Comprehensive user guides will be accessible in multiple formats—text, audio, and video—with walkthroughs to explain app features and functionality. These guides will be regularly updated to ensure alignment with any new app updates, allowing all users to fully utilize the app's potential. By integrating these considerations, MathArt will serve as a universally accessible tool, fostering an inclusive environment for users to explore and enjoy the intersections of math and art.

Design Plan

System Models

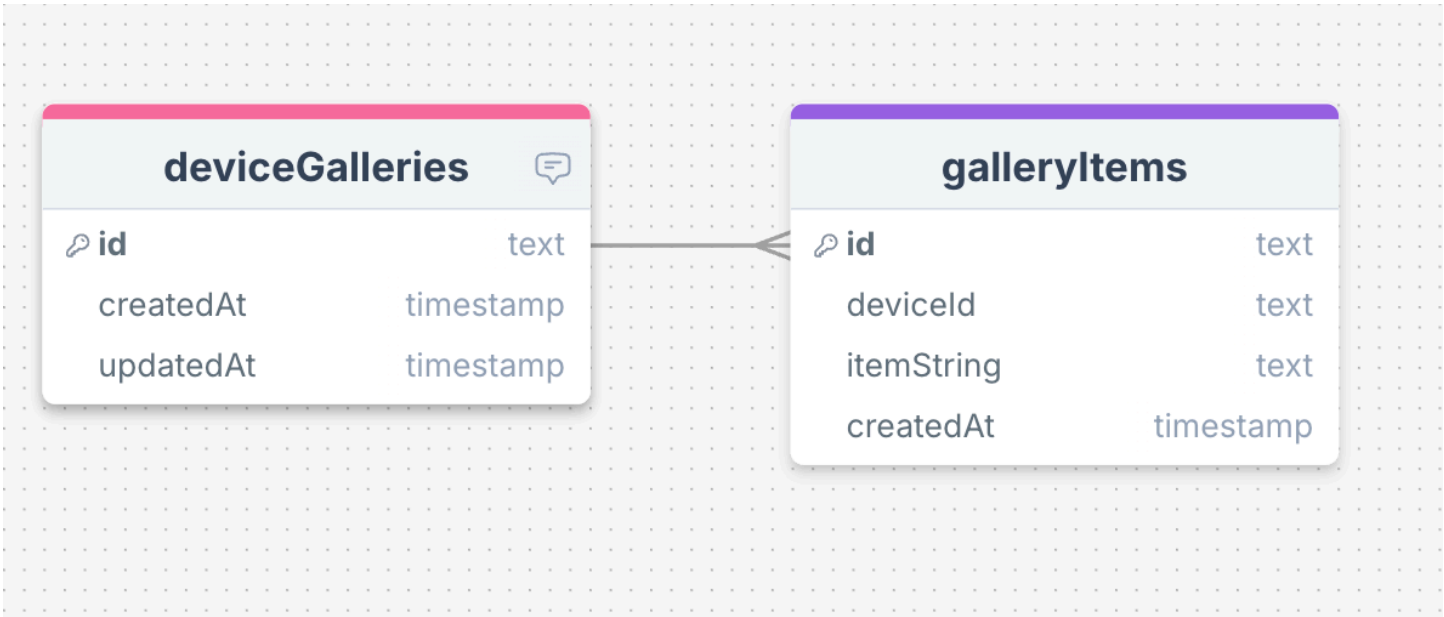
Current Implementation Architecture:

1. View Layer:
    - ContentView: Main container view
    - CanvasView: Handles shape rendering and transformations
    - PropertiesView: Controls panel for shape manipulation
    - ColorPickerView: Color selection interface (planned)

2. Model Layer:
    - ArtworkProperties: Manages transformation values
    - ShapeProperties: Handles shape-specific attributes
    - CanvasState: Maintains current canvas configuration

3. ViewModel Layer:
    - ArtworkViewModel: Manages state and business logic
    - TransformationManager: Handles mathematical calculations

Class Diagram



Note: This diagram includes all of the necessary attributes. Please reference the applicable methods below

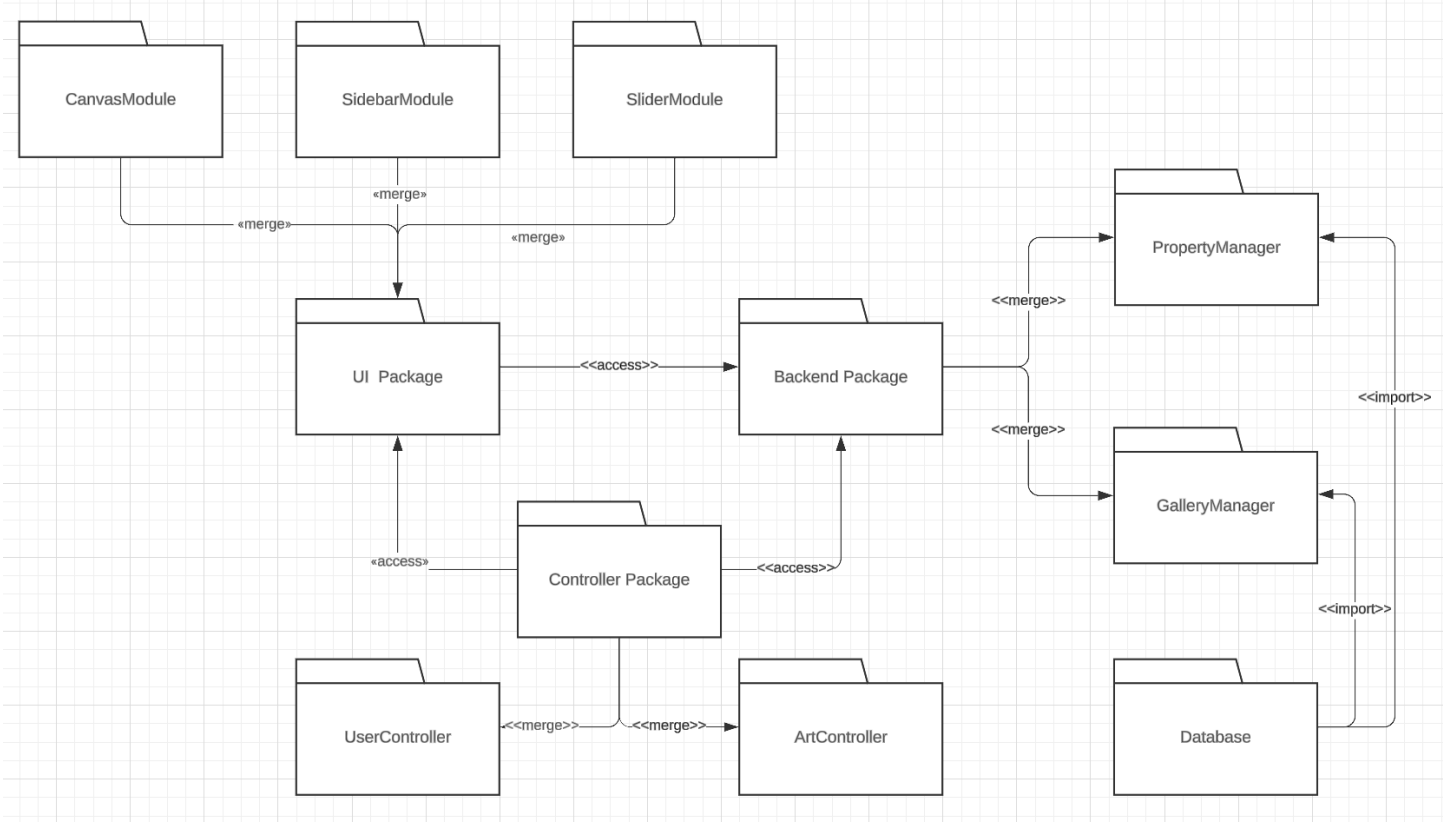
The **Class Diagram** outlines the key classes in the MathArt App and their relationships:

1. **DeviceGallery Class**
    - **Attributes:** id (text), createdAt (timestamp), updatedAt (timestamp)
    - **Methods:** createGallery(), updateGallery()

2. **GalleryItem Class**
    - **Attributes:** id (text), deviceId (text), itemString (text), createdAt (timestamp)
    - **Relationships:** Many-to-One with DeviceGallery (through deviceId)
    - **Methods:** addItem(), removeItem(), getItems()

<<PhoneArt - MathArt IOS Application>>

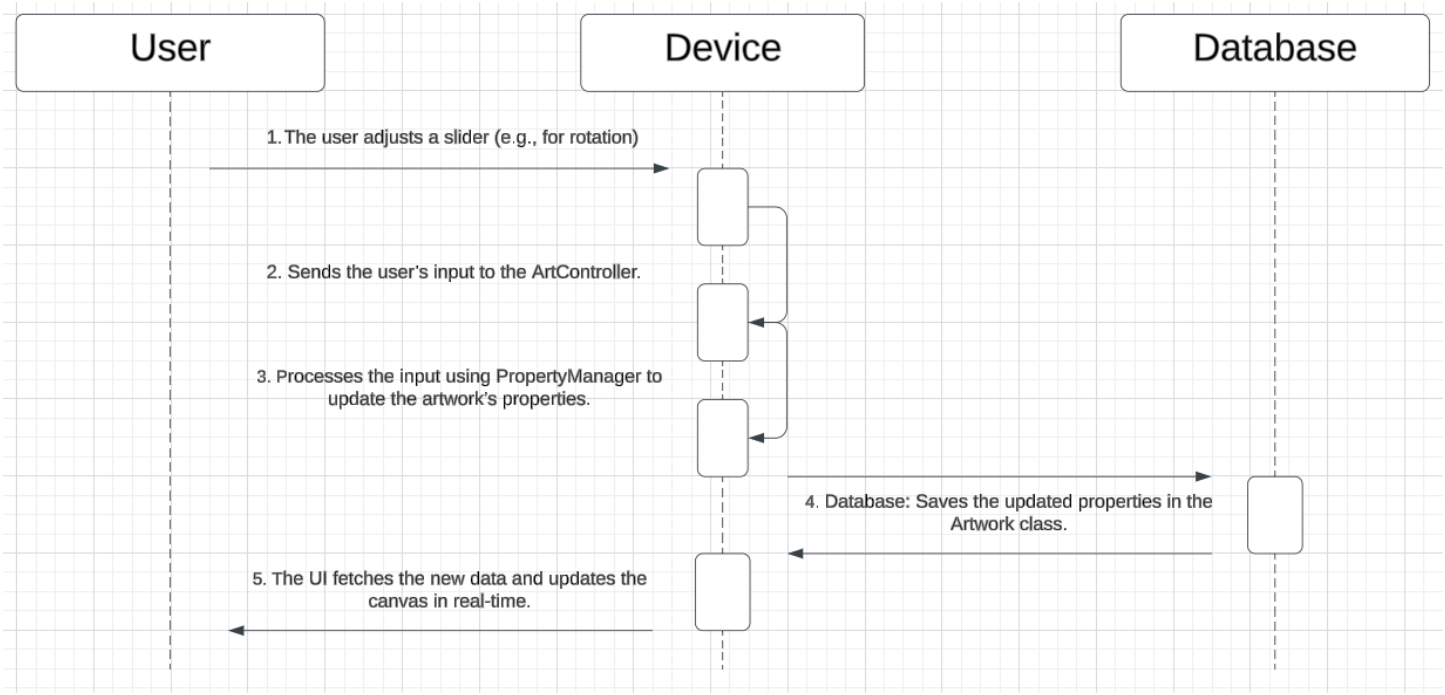
Package Diagram



The **Package Diagram** shows how the app's components are organized:

1. **UI Package**
    a. Includes components like `CanvasModule`, `SidebarModule`, and `SliderModule`.
2. **Controller Package**
    a. Handles interactions between the UI and backend logic.
    b. Example: `UserController`, `ArtController`.
3. **Backend Package**
    a. Manages the database and logic for artwork generation and storage.
    b. Example: `GalleryManager`, `PropertyManager`.

Sequence Diagram

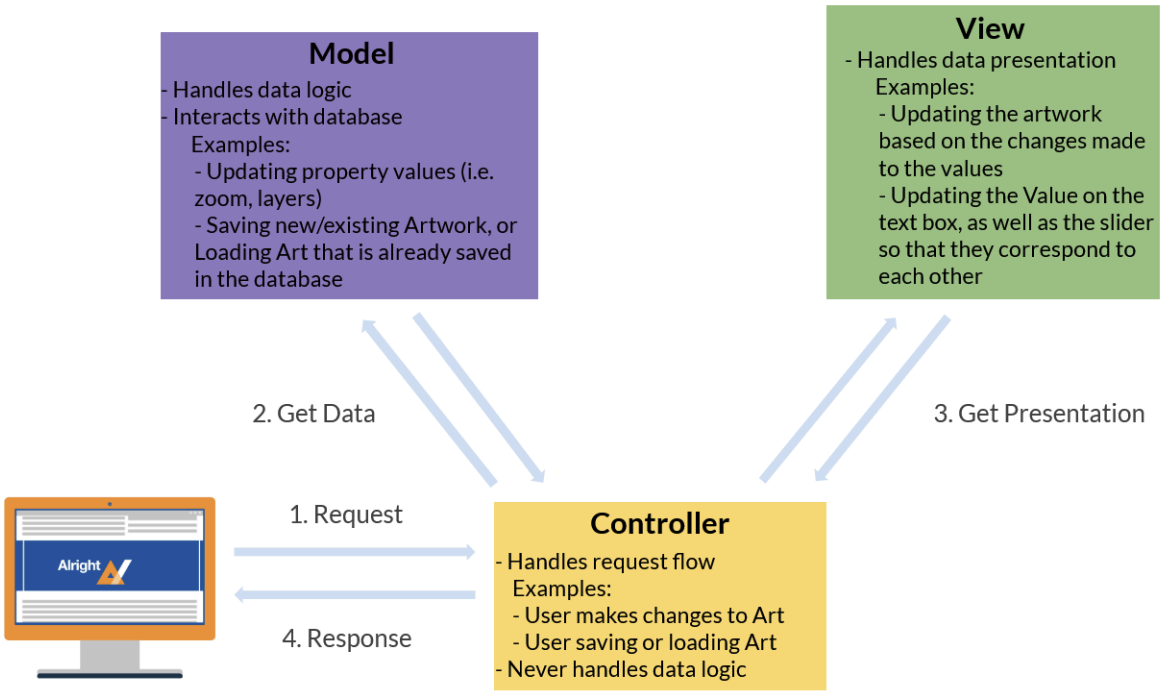


The **Sequence Diagram** demonstrates the interaction flow when a user modifies artwork:

1. **User Action:** The user adjusts a slider (e.g., for rotation).
2. **Controller:** Sends the user's input to the `ArtController`.
3. **Model:** Processes the input using `PropertyManager` to update the artwork's properties.
4. **Database:** Saves the updated properties in the `Artwork` class.
5. **View Update:** The `UI` fetches the new data and updates the canvas in real-time.

Software Architecture

Our group has chosen to adopt a MVC architecture as its independent component structure – Model, View, and Controller – is a good fit for developing our mobile application. This separation will allow us to better maintain and test each aspect of our application, which also promotes scalability as the components work independently from each other. The MVC model also facilitates code reuse as we may have multiple viewing formats for the same model. A good example of this is how the user may visualize the properties for the artwork in the application. The same numerical data will be displayed in the form of a slider, as well as a box, that may be manually updated via the on screen keyboard. This change would then be reflected on the slider accordingly



Here is a brief example of how our application will interact with the different components of the MVC:
Controller - The End User changes the values of the art properties, for this example let's say that the user changes the number of layers that the art has.
The Controller will then fetch this request to Model, which will handle the data logic, as well as interact with the Database, when needed. Then it will
Model - The model will take the request and update the values using the mathematical logic that the backend runs under. Once all of the backend values have been updated by the model, it will then send the greenlight to the Controller to now communicate with the View to visually represent the changes made to the data. This performs the updates to reflect user input, such as calculations for artwork rotations, layers and other sliders.
View: The View will now gather the updated values, and update the UI to represent the changes requested by the user. The View will then fetch the presentation to the Controller. Finally the Controller will respond to the End User's request with the updated artwork in real-time, ensuring a responsive and an engaging experience.

Data Architecture

External / Pre-existing Data Sources

- **Data Source 1: mathart.us**
    - **Description:** The mathart.us page includes the web-version of PhoneArt, which we will be referencing as we begin to design and implement the project. The page also includes other projects that our client has worked on, which may be referenced for functional inspiration . Finally, mathart.us will be used test our implementation to ensure that the mathematics being coded is adjusting the artwork as intended (i.e. rotating a shape by 90 degrees clockwise).
    - **Type:** Structured. All the data is well presented and set up for use.

User Interaction Architecture

We will be using a touch-based modality since we will be releasing the app for IOS devices. The easiest way to use these apps is through tapping, dragging, swiping, and potentially a pinch to zoom function.

1. CanvasModule: Currently implemented drag functionality with bounce effects, center point reference for transformations, and for future updates planned: Multi-touch support.
2. ColorModule: Users will have the ability to choose colors for their artwork to show a palette that they can view or that other users prefer. Choosing this option will in real-time change the appearance of the on screen display.
3. PropertiesPanel: Will allow users to change size, rotational axis, the number of objects in the art, and many other settings. This will allow the user to have a larger range of control to give precise display changes and more freedom to explore math and art.
4. Built in a simple, intuitive interface; with quick access to all transformation controls, and plan for a gallery integration.
5. PropertiesPanel: Added in: dual input methods (sliders and text fields), real-time value validation, collapsible panel design, and a clear visual feedback..
6. ShapesPanel: Rich selection of shape options, which include: circle, square, triangle, hexagon, star, rectangle, oval, and more. Includes an intuitive grid-based interface for easy shape selection, which ties in seamlessly with other property panels for complete shape customization(colors and movement).

Our interface will enable users to access tools, features and changes quickly to save more time for exploration and use of the app. Keeping our touch modules smooth and with quick response time will help the user have a better experience while not just having fun, but learning more about math and art.

Proposed Development Environments

Languages:

Comparing Objective-C, Dart(via Flutter), and Swift

- Objective-C
    - Pros:
        - Mature and stable, having a long standing in Apple development and includes a large amount of legacy frameworks and libraries

        - Powerful running time allows for dynamic typing and message passing for devs to have larger control over the system and runtime changes
        - Supports older IOS versions consistently, which is ideal for continuing upkeep on legacy applications down to IOS 3
    - Cons:
        - Excessive syntax that is complex, making it harder for newer devs and less accessible compared to other languages
        - Lacks modern language features found in swift, such as improved error handling and less support from apple as they now promote swift
- Dart(via Flutter)
    - Pros:
        - Includes cross platform capabilities, allowing development for android, ios and others with only one codebase
        - Higher-performance UI offers flexibility and customization, ensuring easier visually appealing applications
        - Ability to see changes in real-time as you enter new sections of code, speeding up productivity and the development process
    - Cons:
        - Does not include the same feeling of native iOS apps as it is cross platform, interrupting user experience
        - Larger app size as it would include compatibility for multiple platforms all in, requiring a larger download/updates
- Swift
    - Pros:
        - Apple's preferred language for iOS development, includes strong features including memory management, security and quick performance
        - Comes with concise, easily readable syntax and has modern features built-in for an easier learning curve for beginners
        - Easily compatible with Apple development tools in xcode, giving a more smooth iOS experience in xcode for uploading and updates
    - Cons:
        - Swift is only for apple platforms, limiting its use
        - Still growing but is a new language compared to the others, with smaller amount of libraries
        - Less of a back compatibility for older versions of iOS, makes it challenging for legacy apps/devices

We will be using swift as it is the base default language for IOS app development. Swift is efficient, powerful, and gives us a lot of safety features like type safety and memory management. Since it is supported by Apple, we can ensure with any updates that it will still be compatible and although it has fewer libraries than the established languages, the gap is shrinking constantly. Although it comes with no cross-platform compatibility, we have chosen to only develop for Apple, so it is not an issue for us.

Frameworks:

Proposed Frameworks: SwiftUI, UIKit, Core Graphics

- UIKit
  - Pros:
    - Hosts an extensive library and has a mature framework, guaranteed stability
    - Most commonly used for complex/highly customized UI designs that include interactive controls
    - Supports a further backwards compatibility across IOS versions back to IOS 9
  - Cons:
    - Requires a more extensive boilerplate code, which causes a longer/slower development process
    - Steeper learning curve, especially for those who have never used swift before
- SwiftUI
  - Pros:
    - Declarative syntax allows for streamlined code and enhances usability and readability for newer teams
    - Easier state management, creating more efficient prototyping and testing
    - Integrates simply with swift, reducing the boilerplate and includes faster UI updates with real time code to display updates
  - Cons:
    - Very limited compatibility for older IOS versions, only supports IOS 13 and above
    - Lacks the more advanced UI customization found in UIKit
- Core Graphics
  - Pros:
    - Low-level rendering gives a more precise and quicker updates for 2D implementation
    - Ideal for apps that require advanced animations or specific graphic changes
  - Cons:
    - Higher complexity and a huge learning curve for just using even simple graphics
    - Time-consuming to learn, implement and test compared to other high-level frameworks(i.e. UIKit/SwiftUI)

After reviewing these three high-level frameworks, we have noted that SwiftUI would be in our best interest to implement. It has an easier learning curve for beginner users, and although it lacks the more advanced features, it still includes all features we would need for our project. The other con being “limited compatibility” is true if any user has an IOS version between 9-13, but as IOS is now onto IOS 18, we won’t see anything prior to IOS 13 often.

APIs:

Comparing Firebase, Icloud, and SQLite for our Artwork Storage

- Firebase
  - Pros:
    - Real-time database gives instant updates and cross-device synchronization
    - Very scalable, including user authentication, giving a more enhanced security and user experience
    - Gives availability with a free tier, suitable for smaller-scale app development that has a potential for growth

    - Cons:
        - Requires Internet connection to save to the cloud, stopping offline saving
        - Potential cost increases at extremely high levels due to its tiered pricing list
- iCloud
    - Pros:
        - Includes seamless integration for IOS apps, giving reliable usage for offline syncing
        - No added dependencies since it comes integrated into IOS/macOS download environments
    - Cons:
        - Limited to only Apple usage, missing out for cross-platform sharing
        - Could be challenging to change for our goal of real-time updates since it has delays in data syncing
- SQLite
    - Pros:
        - Lightweight and efficient, requiring no server, which is ideal for storing offline and on-device data
        - Supports SQL, giving familiar capabilities in queries for relational data
    - Cons:
        - Does not include real-time syncing or cross-device compatibility, limited for users with multiple devices
        - Not ideal for apps that require data sharing or cloud backups

We want to give users the option to save and bring back their artwork from a gallery. This can be fixed with using Firebase as an API, this would be used to authenticate users and give storage of their data and artwork. We prefer Firebase over iCloud for the real-time updates available and Firebase over SQLite for the cross-device compatibility for users with iPhones and iPad.