PRD Version Two

Authors

Cole Margerum cole.margerum@gmail.com (Group lead)

Michael Amalfitano amalfitano@ucsb.edu (Scribe)

Isaac Zinman zinisaac@gmail.com

Jake Guida jlguida@ucsb.edu

Artem Jivotovski artem.jivotov@gmail.com

Team

'); DROP TABLE TEAMS;--

Project Title

Free Real Estate

Introduction

Our project will allow property managers to avoid the hassle of buying furniture and other decor by staging a property virtually. Currently, property managers have to hire people to move in furniture, decorations, and appliances. This takes a considerable amount of time and money, prevents the property manager from showing anyone the property until it is completely staged, and can't be customized or changed after the fact. Our augmented reality application will allow the user to virtually place furniture and decor in an empty room for prospective tenants to view. In addition to making the property manager's job much cheaper and easier, there is

virtually no time required besides pulling out an iPhone or iPad and downloading our application. Furthermore, prospective renters will be able to view properties decorated in their preferred style. Our application gives the customer the power of choosing how the interior looks while they view the property, in real time. Moving into a new property is a big commitment -- with our app, customers will be able to feel more certain of their investment.

Products that offer similar features are Ikea Place and Amazon's "View in room" feature built into their mobile shopping app. Ikea Place allows the user to put multiple pieces of furniture and other various pieces of home decor in their home. Amazon's augmented reality viewer offers a much wider selection of 3D models to place virtually in a space. Neither of these two applications offer persistent viewing -- the ability to close the application or restart the device and relaunch the app to see the previously placed 3D models in your scene, in their original location. What makes our application so innovative is its ability to not only place multiple 3D models of furniture and home decor in a space but to offer persistent scenes. This is a new feature of Apple's ARKit 2 that was released under a year ago and has not been utilized by any current apps on the market. Yet another new feature within ARKit 2 is the ability for multi-viewer scenes, where two or more users can all see the same scene on their devices in real time. When one user adjusts a 3D model in the space, all users see the adjustment on their individual screens. Similar to persistent scenes, the Ikea Place and Amazon apps do not offer this feature.

Although there are current augmented reality applications on the market that allow users to place furniture like the two mentioned above, none of them are designed for property managers, nor do they feature persistent AR settings. Our biggest innovation will be the

introduction of AR furniture presets designed by the property manager, which will be visible to users once they start touring the property. To do this, the property manager can create a preset by entering the property, adding virtual furniture and decor, and saving it to their account. Once the preset profile is saved, a QR code is generated and emailed to the property manager. They can print this and tape it to a wall. Later, a user can scan the QR code and all of the furniture the property manager previously staged will be shown on the user's screen in its original position. In order to introduce such functionality, we will be leveraging the capabilities built into Apple's ARKit 2. The API features persistent ARWorldMaps, which allow the state of the virtual space to be saved and loaded between sessions. This feature only works under the assumption that the persistent scenes can be stored in a web service like AWS, accessed using a QR code, and accurately loaded and displayed on a user's device without any missing 3D models or lag.

As mentioned above, the second major new feature within Apple's ARKit 2, shared experiences, allows multiple users to collaborate on the same project in real time. Assuming ARKit handles all of the networking components of shared experiences, we should be able to incorporate this into our application with minimal frustration. Our app will allow any of the connected users to place, reposition, or remove a 3D object from the scene, and the change will be reflected on everyone's device in real time.

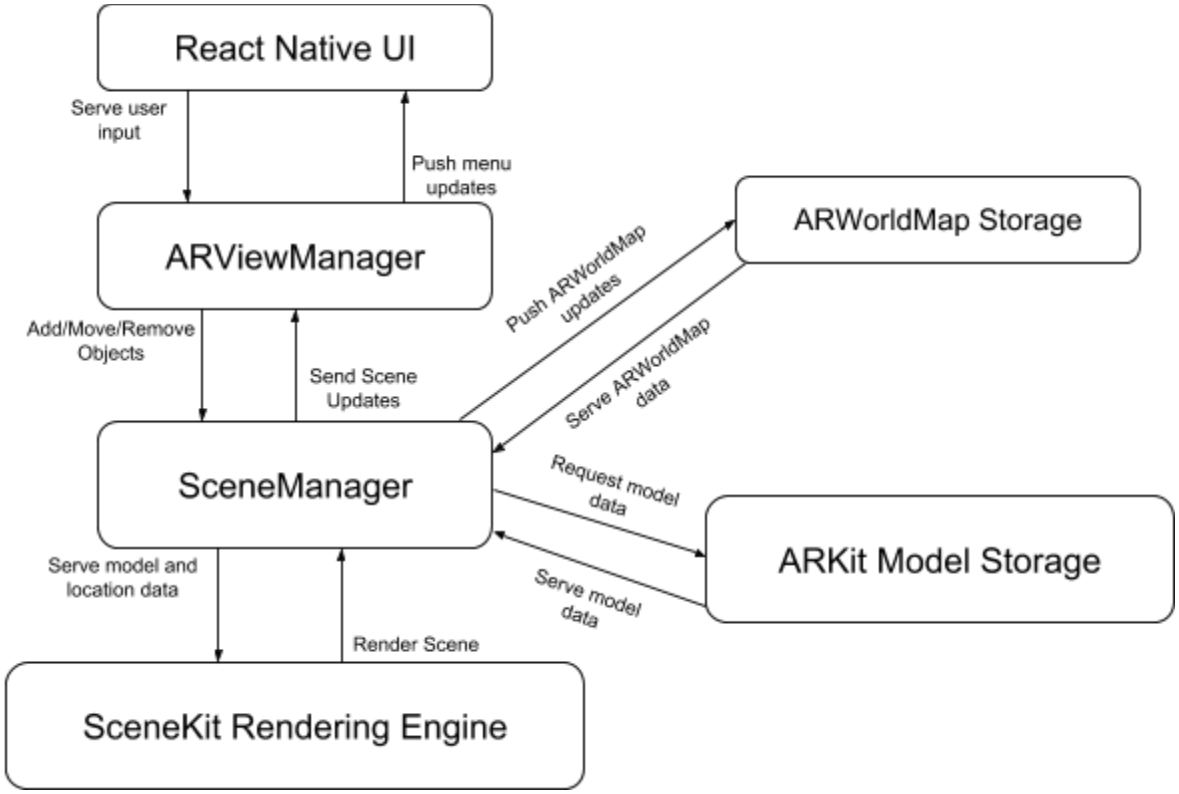
In order to ensure a smooth and consistent user experience, we will develop the frontend utilizing React Native. This framework will enable us to quickly prototype and deploy UI components in order to develop a minimalist but powerful interface. Additionally, React Native features robust APIs for interacting with native Swift code, which ensure that the application's

performance is not bottlenecked by calls to the backend components. The project's backend will be written in Swift to leverage the full capabilities of ARKit and iOS.

Our initial goal is to create a minimum viable product that can showcase simple AR functionality. This simply includes the ability to place furniture and small decorative items in a space. Once we have that completed, we will move on to creating preset scenes using ARKit 2's persistent ARWorldMaps. After making these additions, we will work on our stretch goals like enabling multiple devices to view and manipulate the same scene in real time. We hope to achieve such functionality through a responsive and intuitive UI, focused on providing a simple but dynamic user experience. In addition to this, we plan on optimizing the storage and loading of 3D models in order to ensure that transitions between presets are seamless. This will be done by passing the React Native backend a list of all available 3D model names and file paths within the file structure when the app is first launched. That way, when a 3D model is selected to be placed in the scene, the React Native backend already knows it is a valid model and locates it using the file path passed in earlier.

Our primary non-technical goal is to cultivate a dynamic and efficient work environment. We aim to split work up evenly among all five team members, and support each other when needed. We plan to share resources like devices. Not all of our members have an iOS device capable of running ARKit, so we will utilize pair programming for efficiency and to ensure we can efficiently run and test our code. We also want all of our team members to be excited about what they are working on; this includes the tech stack, the product, and individual components in development. The biggest goal for us as a team is to have a presentable project by the end of winter quarter.

System Architecture Overview - High Level Diagram



Functional & Non-Functional Requirements

Use Cases

1. Select a style of furniture
    a. Actors: User 1
    b. Precondition: User opens up application for the first time
    c. Flow of Events - Basic Path
        i. User sees a menu with all of the furniture styles that are selectable

ii. User picks a style

iii. User clicks the select furniture style button

d. Flow of Events - Alternative Path

i. There is nothing the user can do besides blank button to proceed

e. Postcondition: The user can see a blank camera feed

Github commit:

https://github.com/izinman/droptableteams/commit/a8d803861f879535a8a75b8b575ee6504b5d30d2

2. Allow camera access to application

a. Actors: User 1

b. Precondition: use case basic path 1

c. Flow of Events - Basic Path

i. System asks the user for permission to use the iPhone's rear facing camera

ii. User grants the app permission to use the iPhone's rear facing camera

d. Flow of Events - Alternative Path

i. User does not allow the app to use the iPhone's rear facing camera. The app displays a message that in order to use the app the user must go to Settings and allow camera access.

e. Postcondition: User can see a live camera feed on their display.

https://github.com/izinman/droptableteams/commit/234cf849a5327eb26062889458b837c6e608cb45

3. Identify flat surfaces
    a. Actors: User 1
    b. Precondition: uses case 2 basic path
    c. Flow of Events - Basic Path
        i. System renders the live camera feed on the display
        ii. System uses ARKit and SceneKit to identify flat horizontal and vertical planes in the scene
    d. Flow of Events - Alternative Path
        i. User has not yet allowed the application access to the device's camera. An error is shown on the display
    e. Postcondition: System's menu button (+) in the bottom center of the display becomes visible once at least one flat surface is recognized https://github.com/izinman/droptableteams/commit/5fa96c365de927686c9bb48f821736d5da909bcc

4. Overlay on flat surfaces
    a. Actors: User 1
    b. Precondition: uses case 3 basic path
    c. Flow of Events - Basic Path
        i. System overlays a yellow colored plane above flat objects it recognizes on the display

d. Flow of Events - Alternative Path
    i. User has not yet allowed the application access to the device's camera. An error is shown on the display
e. Postcondition: User can see all flat surfaces recognized in the scene.

5. System renders menu with available 3D objects
    a. Actors: User 1
    b. Precondition: uses case 3 postcondition, uses case 4 basic path
    c. Flow of Events - Basic Path
        i. System has a visible menu button (+) in the bottom center of the display
        ii. User taps menu button
        iii. System pulls list of 3D objects from ARKit model storage
    d. Flow of Events - Alternative Path
        i. System is unable to recognize flat surfaces in the scene and the menu button is grayed out
    e. Postcondition: System successfully loads a list of available 3D objects to display in a menu
    https://github.com/izinman/droptableteams/commit/7de4e5d1db6a7fb82218c7f0a5576ceeca6dd409

6. Select a piece of furniture from menu and make it movable in the scene (ready for placement)

a. Actors: User 1
b. Precondition: uses case 5 basic path
c. Flow of Events - Basic Path
    i. System draws a menu window overlaying the live camera render and surface detection
    ii. System displays list of 3D objects in the newly drawn menu
    iii. User selects a 3D model option from the menu by tapping it
d. Flow of Events - Alternative Path
    i. System has no 3D models to load from memory and the menu is empty
e. Postcondition: User successfully selects one 3D object and the menu disappears, revealing the live camera render and recognized flat surfaces

7. Place the piece of furniture selected
    a. Actors: User 1
    b. Precondition: uses case 6 basic path
    c. Flow of Events - Basic Path
        i. System allows user to drag the selected 3D object around the scene
        ii. Once user has moved the 3D object around the scene and it is above a recognized flat surface, the 3D object can be tapped once to drop it in place
    d. Flow of Events - Alternative Path

i. System does not allow user to place an object because the surface the user chooses has not been recognized by ARKit as a flat surface.

ii. The object disappears from the screen and a message is displayed “Unable to place object. Please choose an item from the menu and place it on a recognized flat surface.”

e. Postcondition: System is called to retain the position of the 3D model after the user taps to place it

8. Retain position of placed piece of furniture

a. Actors: User 1

b. Precondition: uses case 7 basic path

c. Flow of Events - Basic Path

i. System calls ARKit’s function to lock an item in place

ii. System uses called function to retain the position of the 3D object once user lifts their finger from the display

d. Flow of Events - Alternative Path

i. System does not allow user to place an object because the surface the user chooses has not been recognized by ARKit as a flat surface.

ii. The object disappears from the screen and a message is displayed “Unable to place object. Please choose an item from the menu and place it on a recognized flat surface.”

e. Postcondition: System retains position of the 3D object as the user moves their device

9. Walk around a piece of furniture and view it from different angles
    a. Actors: User 1
    b. Precondition: uses case 8 basic path
    c. Flow of Events - Basic Path
        i. System retains position of the placed 3D object as the user walks around the object, always pointing their device in the direction of the object
        ii. System renders and displays the 3D object from different positions as the user physically walks around it (front, sides, back, top, etc.)
    d. Flow of Events - Alternative Path
        i. System loses the orientation of the scene and the 3D object disappears, up to ARKit to rerender the object again
    e. Postcondition: System retains position of the 3D object as the user moves their device

10. System has lost orientation of scene
    a. Actors: User 1
    b. Precondition: uses case 9 alternative path
    c. Flow of Events - Basic Path
        i. System renders the live camera feed on the display

ii. System uses ARKit and SceneKit to identify flat horizontal and vertical planes in the scene all over again

d. Flow of Events - Alternative Path

i. System has encountered an error and is unable to identify flat surfaces in the scene, potentially due to a physical obstruction on the device's camera.

e. Postcondition: System's menu button (+) in the bottom center of the display becomes visible once at least one flat surface is recognized again

11. Select a placed piece of furniture to alter its position

a. Actors: User 1

b. Precondition: uses case 9 basic path

c. Flow of Events - Basic Path

i. User taps and holds a previously placed 3D object

ii. System allows user to drag the selected 3D object around the scene

iii. System retains position of 3D object (placed on a recognized flat surface) once user lifts their finger from the display

d. Flow of Events - Alternative Path

i. System does not allow user to move the object because the new surface the user chooses has not been recognized by ARKit as a flat surface.

ii. The object disappears from the screen and a message is displayed "Unable to place object. Please choose an item from the menu and place it on a flat surface."

e. Postcondition: System retains position of the 3D object as the user moves their device

12. Select a placed piece of furniture to remove it
    a. Actors: User 1
    b. Precondition: uses case 9 basic path
    c. Flow of Events - Basic Path
        i. User taps and releases a previously placed 3D object
        ii. System displays a small delete button with a trash icon next to the recently tapped object in the scene
        iii. System removes the 3D object from the scene if the user taps the delete button
    d. Flow of Events - Alternative Path
        i. User taps and releases a previously placed 3D object
        ii. System displays a small delete button with a trash can next to the recently tapped object in the scene
        iii. System does not remove the 3D object from the scene if the user taps anywhere on the screen besides the delete button
    e. Postcondition: The selected/deleted 3D object is removed from the scene and is no longer visible to the user

13. User hosts their scene so it can be viewed on multiple devices

a. Actors: User 1

b. Precondition: uses case 2 basic path

c. Flow of Events - Basic Path

    i. System displays a menu button in the top right corner of the display

    ii. System lists two options when user taps on menu button: “Host multi-user session” & “Join multi-user session”

    iii. System calls ARKit 2’s multi-viewer library which handles object coordinates within the scene

    iv. System displays the message “Users on your local wifi network will now be able to share in your AR session” when the user selects “Host multi-user experience”

d. Flow of Events - Alternative Path

    i. System displays error message if device is not connected to wifi

e. Postcondition: A multi-user scene is created

14. User joins a hosted scene

a. Actors: User 2

b. Precondition: uses case 2 basic path, uses case 13 basic path

c. Flow of Events - Basic Path

    i. System displays a menu button in the top right corner of the display

    ii. System lists two options when user taps on menu button: “Host multi-user session” & “Join multi-user session”

iii. System calls ARKit 2's multi-viewer library

iv. System displays list of available local AR sessions to join when the user selects "Join multi-user experience"

v. System connects to User 1's session

vi. System renders any objects User 1 has placed in the scene and positions them accurately using their coordinates shared by User 1's ARKit 2 multi-viewer library

d. Flow of Events - Alternative Path

i. System displays error message if there are no local AR sessions being hosted

e. Postcondition: A multi-user scene is joined. Each user can place 3D objects and the other opposite user will be able to see the changes in real time on their own device.

15. Persistence

a. Actors: User 1

b. Precondition: uses case 8 basic path

c. Flow of Events - Basic Path

i. Objects that are placed are stored locally on the device

ii. The app is closed

iii. The user returns to the app and

d. Flow of Events - Alternative Path

i. ARKit can’t identify where to place items

e. Postcondition: The objects are still placed in the scene in the same positions

16. View the same scene from multiple devices

a. Actors: User 1, User 2

b. Precondition: uses case 13 basic path, uses case 14 basic path

c. Flow of Events - Basic Path

i. System generates and renders the same scene on two devices by calling ARKit 2’s multi-viewer library

ii. System correctly positions 3D objects on both devices so they appear to be in the same place in the room (using the each object’s stored coordinates

d. Flow of Events - Alternative Path

i. System displays error message if there are no local AR sessions being hosted or if User 1’s wifi connection is lost

e. Postcondition: Both Users view the same SceneKit.

17. Application does not change orientation with user’s movements

a. Actors: User 1

b. Precondition: User is viewing a live scene before or after placing objects

c. Flow of Events - Basic Path

i. User tilts device 90° from the left to the right (from portrait orientation to landscape) so the user’s right hand is holding the bottom of the device

(charge port/home button) and the user's left hand is holding the top of the device (earpiece speaker/camera module)

ii. System does not change layout of menu (+) button or the host/join menu button

iii. System keeps the (+) button centered along the "bottom" edge of the device -- the edge nearest the user's right hand

iv. System keeps the host/join menu button in the same corner as before -- the top corner nearest the user's left hand

d. Flow of Events - Alternative Path

i. User does not tilt the device.

e. Postcondition: User's live camera feed is still working and no functionality has changed. If the user has placed 3D objects in the scene, no matter how they move their device, the object will not move, thanks to the item being anchored to specific coordinates as mentioned in case 7's basic path.

18. Application only changes orientation of menus when they are open

a. Actors: User 1

b. Precondition: uses case 17 basic path

c. Flow of Events - Basic Path

i. User taps the (+) button

ii. System draws the menu shifted 90° to the left so that all of the 3D models displayed are right side up

d. Flow of Events - Alternative Path
    i. User taps the host/join button
    ii. System draws the two menu options “Host multi-user session” & “Join multi-user session” right side up
e. Postcondition: The system retains the position of the menu buttons when the device is tilted BUT once the menus are opened, the data displayed is rotated to correspond with the orientation of the display.

19. Application dynamically fits any size iOS screen resolution
    a. Actors: User 1
    b. Precondition: User launches our app from their home screen for the first time
    c. Flow of Events - Basic Path
        i. System finds width of display by using the JavaScript command Dimensions.get('window').width
        ii. System finds height of display by using the JavaScript command Dimensions.get('window').height
        iii. System draws and places all buttons and menu options to scale so whether the user opens our app on an iPhone 6s, iPhone Xs Max, or iPad, all buttons are correctly displayed in their respective locations
    d. Flow of Events - Alternative Path
        i. System is unable to determine the dimensions of the display using the above JavaScript command.

ii. The application appears distorted with menu buttons out of place.

e. Postcondition: System always draws the (+) menu button in the center of the bottom edge of the display -- the edge nearest the charge port/home button. System always draws the host/join menu button in the corner of the display nearest the rear-facing camera.

20. Undo-Replace a removed object

a. Actors: User 1

b. Precondition: uses case 12 basic path

c. Flow of Events - Basic Path

i. User removes an object (use case 12) and decides they want it back, presses undo button in top left corner of display (when held in portrait mode)

d. Flow of Events - Alternative Path

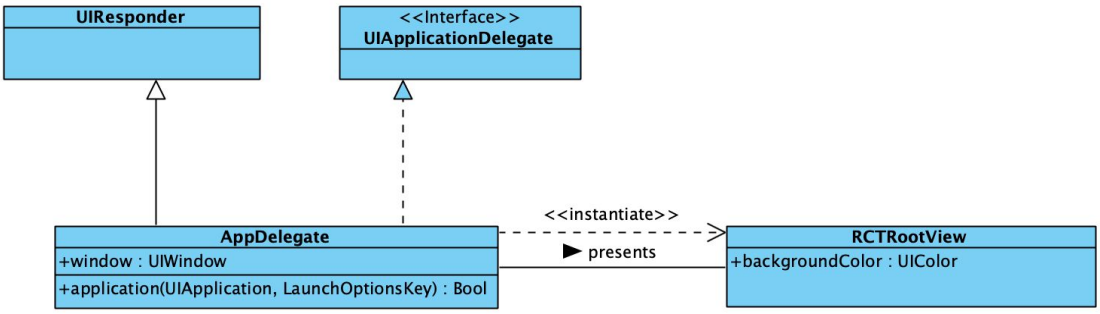
i. The object is not replaced in the scene, the object is different, or it is placed in a different position from where it was supposed to be

e. Postcondition: Object appears back where it was placed, and the undo button is no longer active

UML Class Diagrams

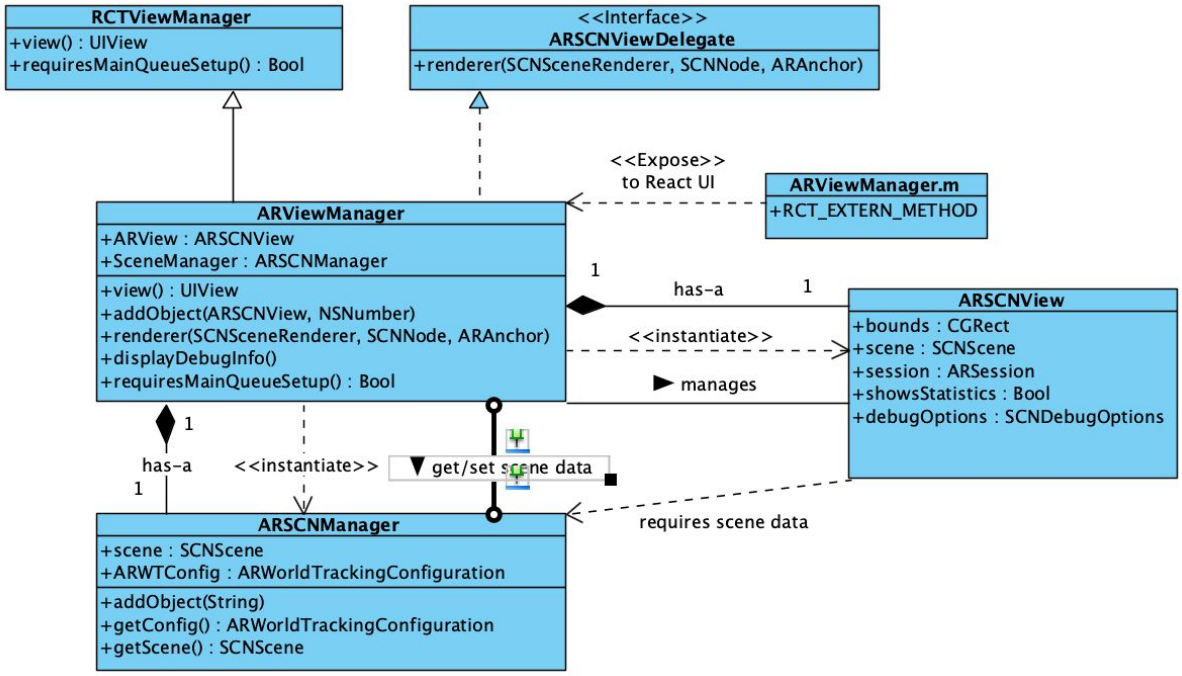
AppDelegate.swift

Creates a React Native root view (RCTRootView) and presents it. This class serves as the entry point of the app, and transfers the control flow to the React Native layer.



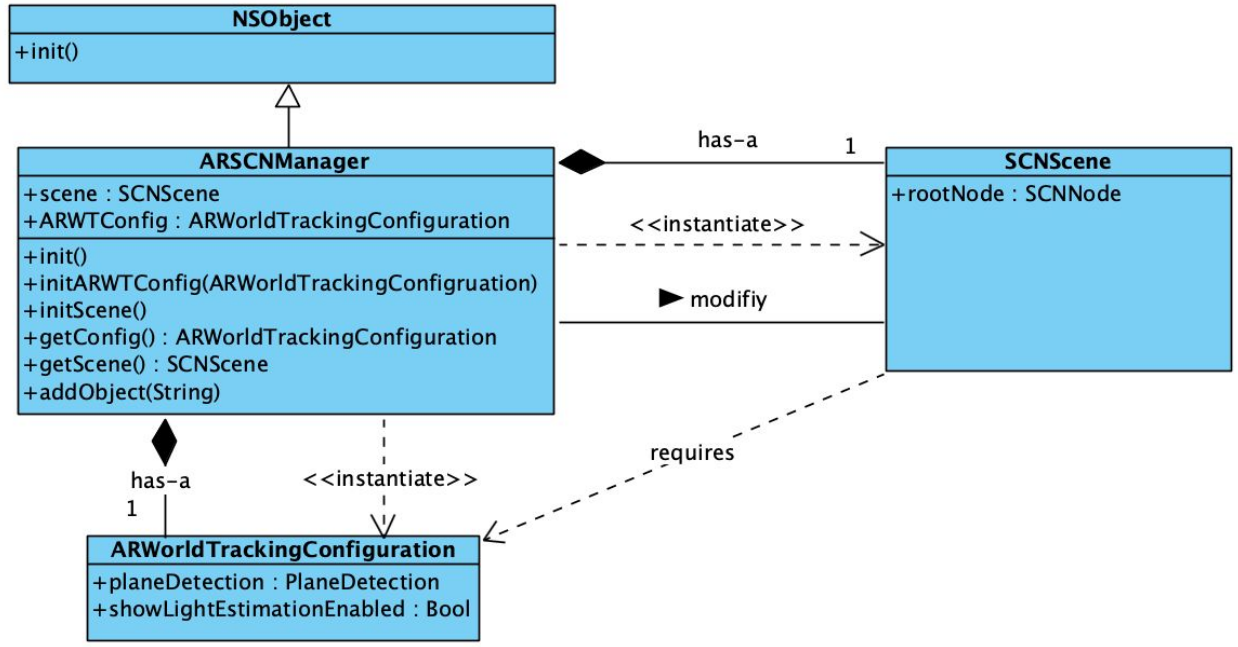
ARViewManager.Swift

This class manages the ARView presented to the user. It renders the scene and displays native UI components to guide the user's object placement. It also acts as an intermediary between the React Native layer and ARSCNManager, passing the user's modifications of the scene to the manager.



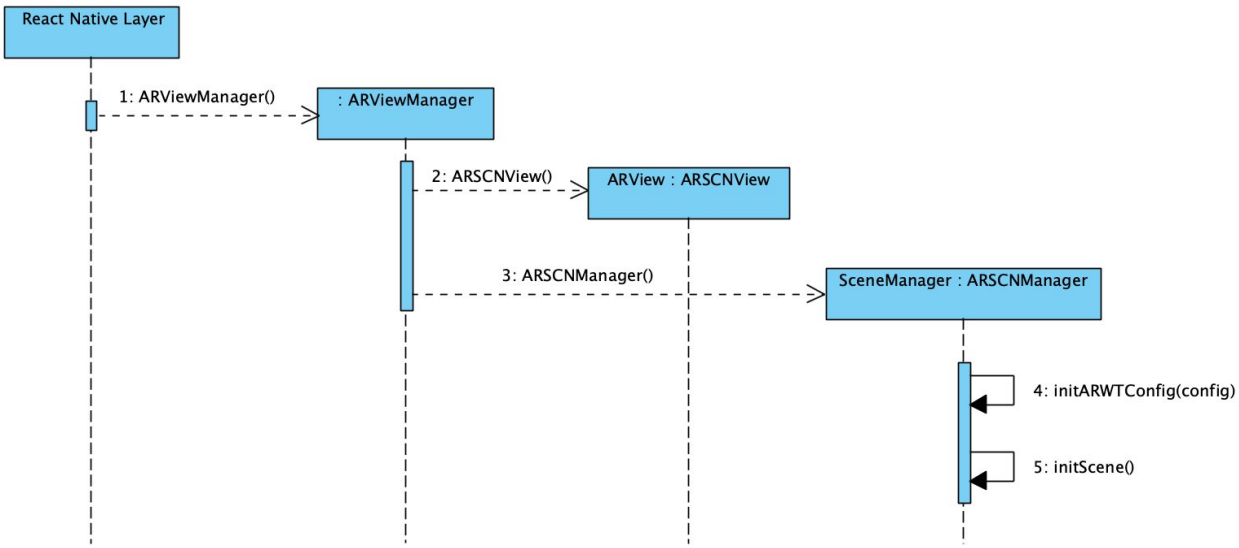
ARSCNManager.swift

Manages the state of the scene, including object placement, manipulation, and removal. Stores changes to an ARWorldTrackingConfigration for preset management.

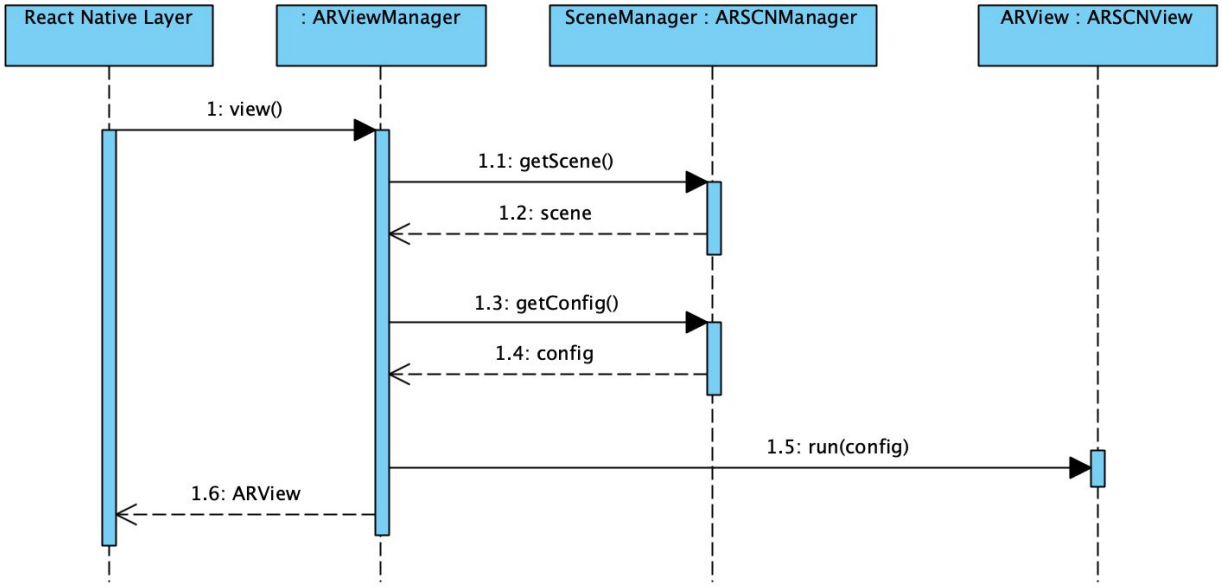


Class Interaction Sequence Diagrams

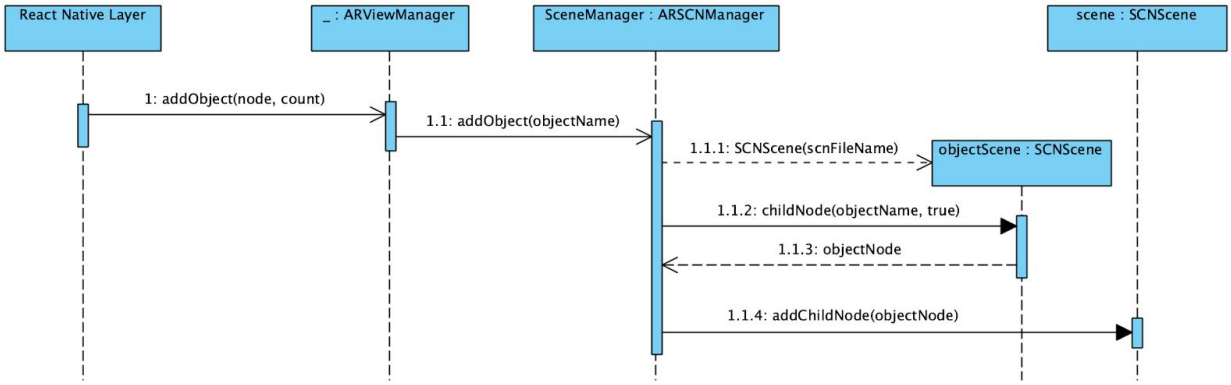
Initialize ARViewManager



Render ARView

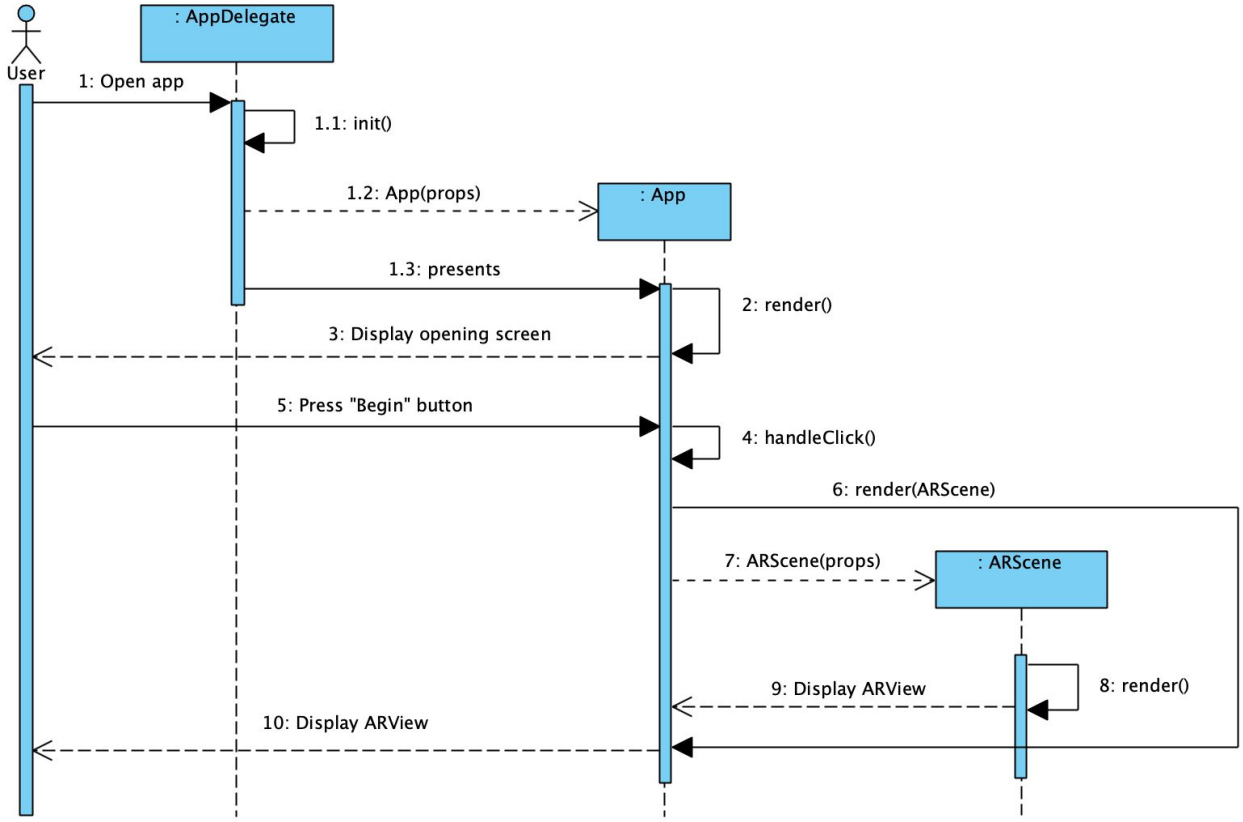


Add object

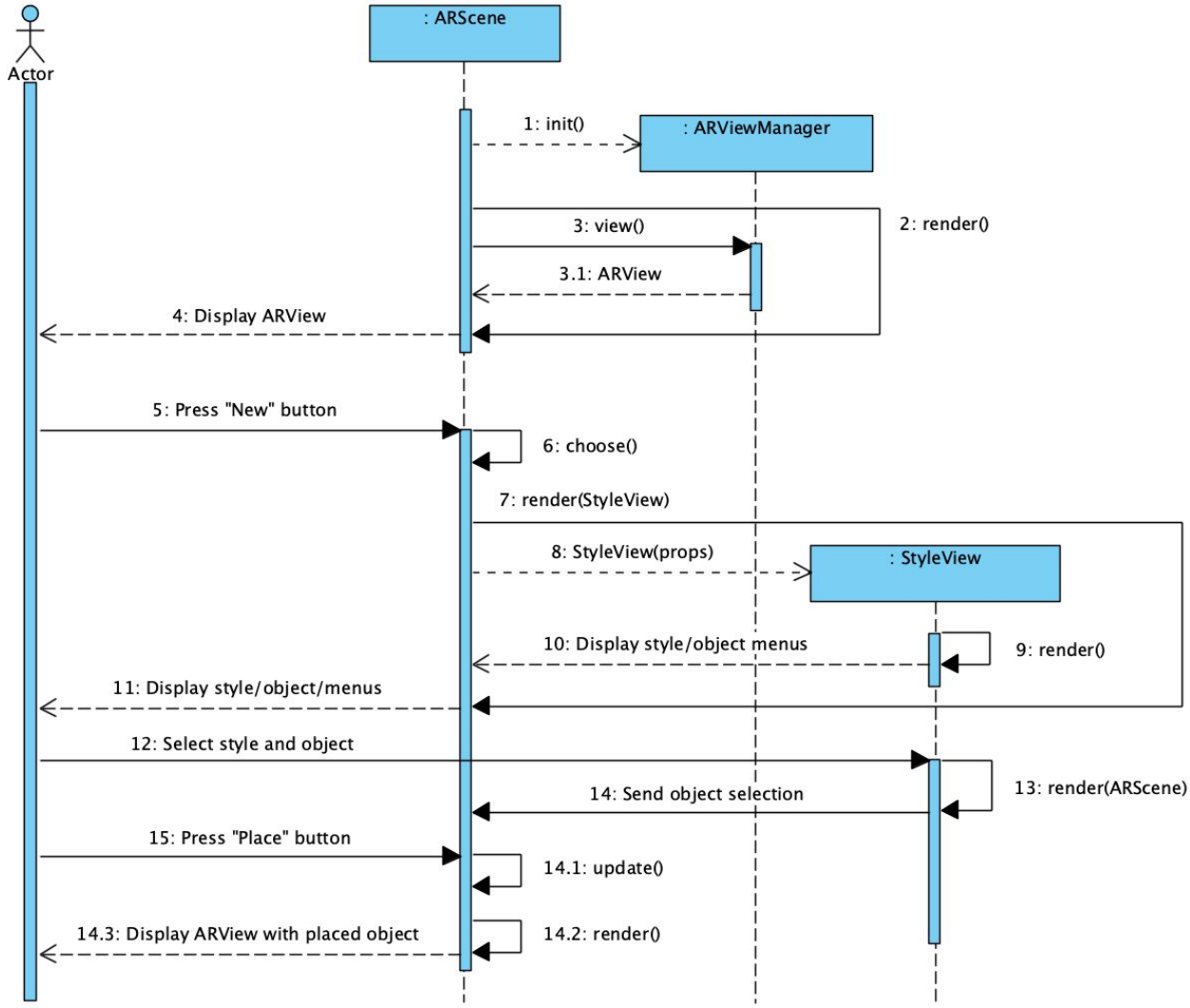


User Interaction Sequence Diagrams

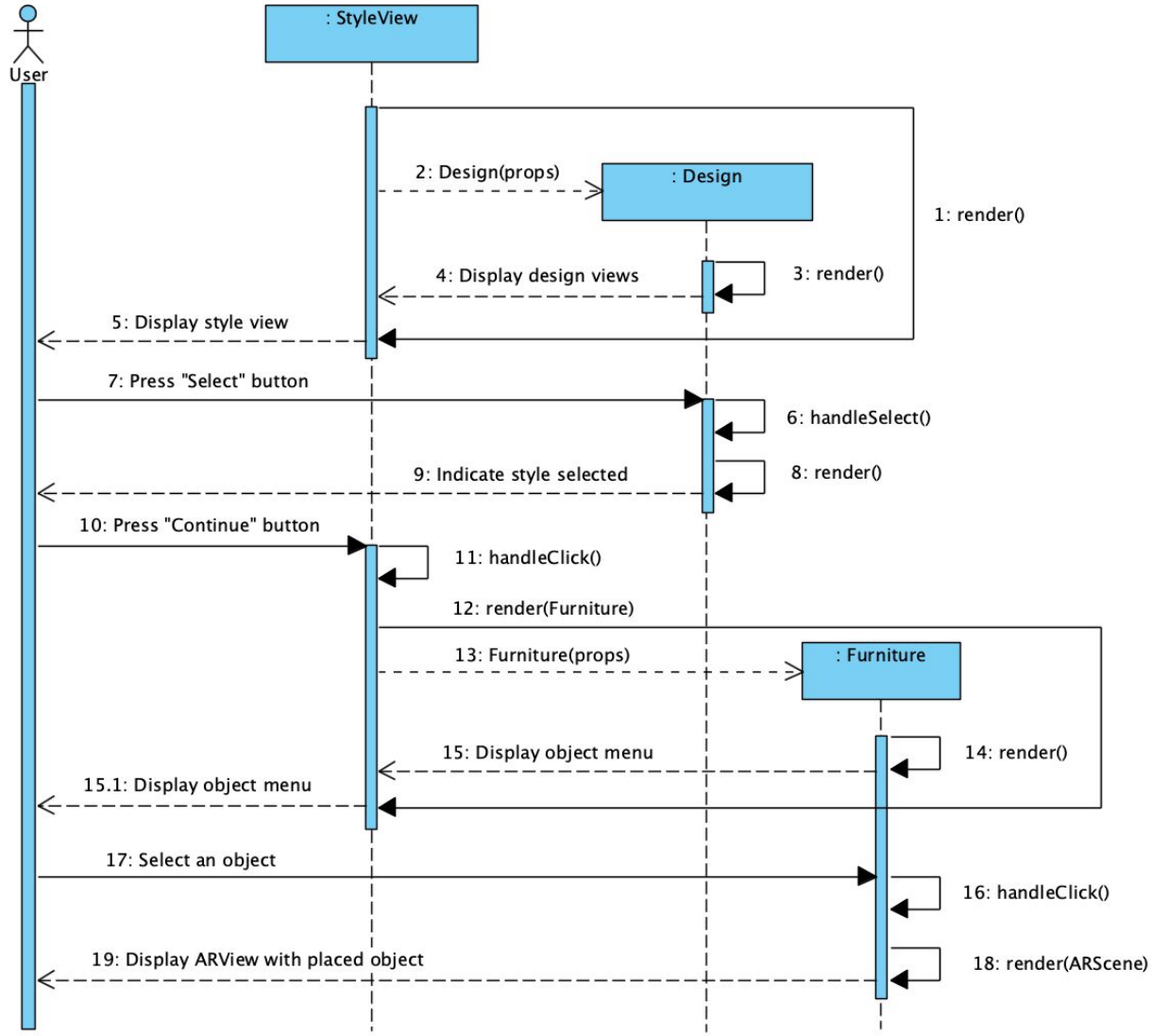
Open app



Place object



Select style and object to place



Prototyping Code, Tests, & Metrics

https://github.com/izinman/droptableteams/

https://trello.com/b/Cj62CfFf/freerealestate

At this juncture, Jake is working primarily in React Native for UI elements. The rest of us are working on various AR-specific tasks like surface recognition, importing and texturing 3D models, and placing objects.

Testing Frameworks

There are multiple aspects of this project that make it difficult to test. First of all, the interaction between the React layer and the Swift layer adds complexity in that both halves have to be able to be tested independently as well as when working together - essentially, unit tests and integration tests, respectively. Furthermore, the app's main function is to serve a visual component in the ARView, whose qualitative properties are hard to measure programmatically. With all of this in mind, we have developed a system to provide solid code coverage and functional validation for our uses.

- Native (Swift)

  https://github.com/izinman/droptableteams/commit/6e70ccac28857eda62179499f85a91e82b3d7b57

The Swift unit testing framework which comes built-in is called XCTest and works similarly to JUnit or other popular unit testing frameworks. It has the advantage of being able to directly affect private members of the class in question simply by importing the desired module with a `@testable` preprocessor directive. The tests are organized by source file, with each source file corresponding to a test class with coverage of all its exposed methods and properties. Files

automatically generated by the React Native project template are not tested, as they are covered by a built-in test framework.

- React (JavaScript)

  https://github.com/izinman/droptableteams/commit/a7a66afd9475c9cf6aa3ba4b82d3e711f9d075e4

The React unit testing frameworks we chose to leverage are Jest [https://jestjs.io/] and Enzyme [https://github.com/airbnb/enzyme]. Using these together allows the tester to manipulate props and state directly in the React code, analogous to the capabilities of the Swift testing framework above. It furthermore utilizes the concept of snapshots, which save the output of a rendered component as a JSON file and compare the previously generated version with the currently generated one. This allows the frontend developer to track the effects of their changes on code and ensure differences in rendered output are intended.

- Integration tests

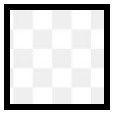
To ensure the React side is playing well with the Swift side, we make sure the app runs correctly and responds to user input. This is primarily a task for the Javascript side as it controls the entry point into the application, so the same frameworks as above are utilized. Of course, visual hands-on testing is still an important aspect of our process, but thankfully the Hot Refresh functionality of React Native allows UI changes to be reflected immediately, reducing wait times significantly.

Primary UI Mockup

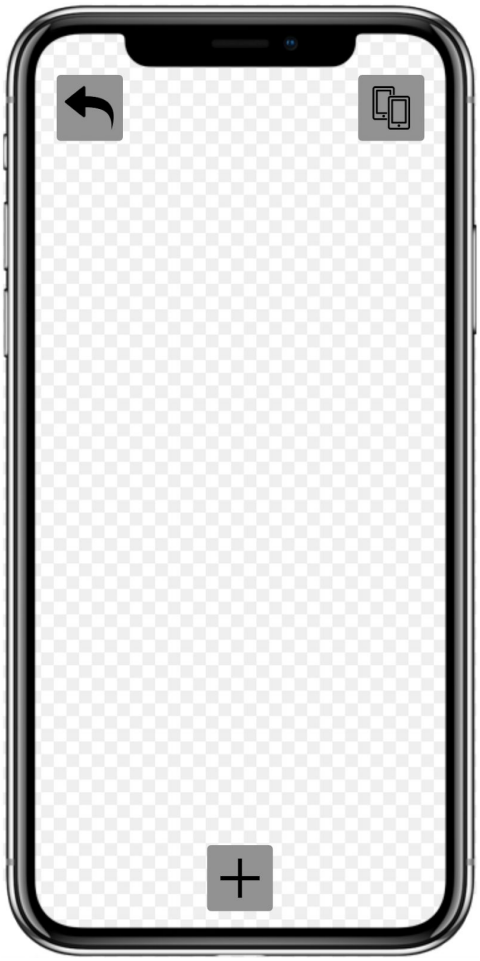
Menu button to launch a menu to find and select a new 3D model to place in the scene

Network button: see use cases 13 and 14.

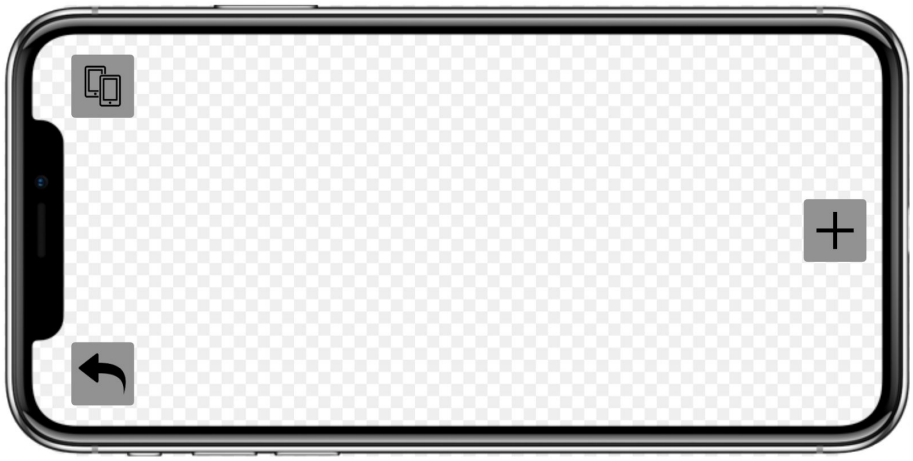
Undo button: see use case 20.



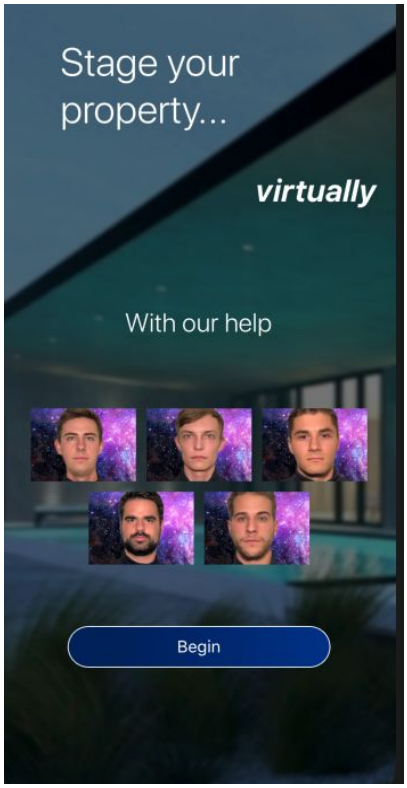
Live camera feed with ARview



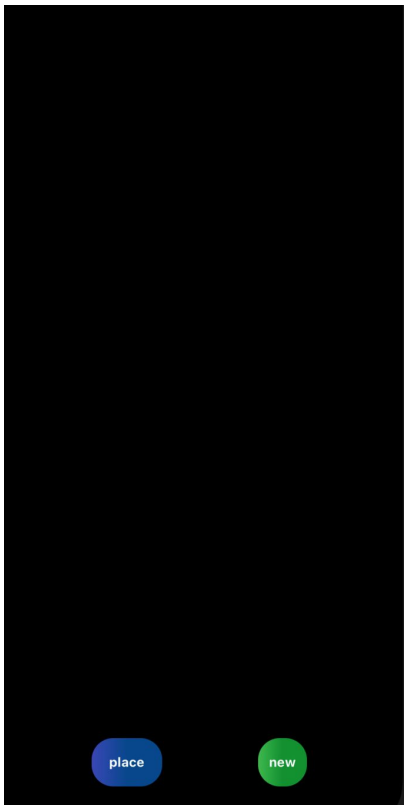
Portrait Orientation



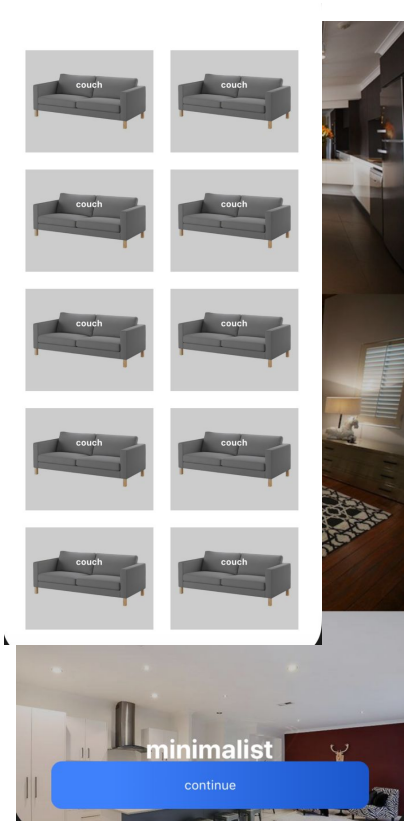
Landscape Orientation (the button location does not change, but the orientation of the icon adjusts)



When the app is launched for the first time



Once the (new) button is pressed this AR view appears. When the user taps on a style of decor, they will be taken to another screen to choose a style of furniture



Once a user selects a style they can hit continue to choose a piece of furniture

After touching a piece of furniture they return to the AR view

Appendix - Technologies Employed

- Apple's Swift APIs:
    - ARKit 2
        - API for identifying flat surfaces, placing and viewing 3D virtual objects, creating persistent scenes between sessions, and utilizing shared experiences for multi-user viewing.
    - SceneKit
        - API that keeps state of the 3D objects, used in conjunction with ARKit 2.
        - All 3D models will be stored locally on each device. The preset scenes will be stored in AWS. Once a preset file is loaded from AWS it will tell the device where to place certain 3D models after the QR code has been scanned and the room is recognized by SceneKit.
- React Native
    - Framework we will use to develop most of the UI components like menus and buttons.