Department of Computer Science & Engineering
The University of Texas at Arlington

Architectural Design Specification
CSE 4316: Senior Design I
Summer 2025

AR Wetlands Watchers
Water Pollution Simulator

Adrian Macias
Mauricio Mendoza-Silos
Nooraldeen Alsmady
Yahia Elsaad
Mohamad Nahib Alkhateeb

Revision History

Revision	Date	Author(s)	Description
0.1	07.14.2025	MMS	Document creation
0.2	07.17.2025	AM	Updated all ADS diagrams
1.0	07.23.2025	AM, MMS, NA, YE, MNA	Complete draft
2.0	12.05.2025	AM, MMS	Updated to final product standards

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

The AR Wetlands project is an interactive AR educational system designed to raise awareness about water pollution, runoff, and environmental impacts of everyday life. Sponsored by the U.S. Army Corps of Engineers (USACE), the project combined a physical tabletop exhibit with a mobile AR application to deliver engaging, scenario based learning experiences to a broad audience.

The main experience centers around a tabletop, approximately 2' x 2' which serves as a visual anchor for AR content. Printed markers on the table will be recognized by the mobile device's rear camera, triggering real-time 3D simulations of environmental scenarios such as construction runoff and storm water drainage in residential areas. As users point their AR-enabled Android devices toward different regions of the table, the system overlays animated 3D models and informative narration to explain the flow of pollutants. Enhanced interactivity is achieved through touch gestures, allowing users to tap specific virtual elements to trigger educational audio clips.

The AR application is being developed using Unity, AR Foundations, and ARCore, and is designed to operate fully offline once installed. This offline functionality makes the system highly portable and dependable for use in classrooms, outreach events, and public demonstrations where internet connectivity may not be available. The intended users include students, community members, and non-technical stakeholders who benefit from simplified, visual explanations of environmental systems and infrastructure.

This document outlines the Architectural Design Specification (ADS) for the AR Wetlands system. It details the overall development strategy, system layering, internal subsystems, and interactions between components. The architecture is structured to be modular and maintainable, supporting scalability and future enhancements while ensuring robust performance during demonstrations.

The project scope includes accurate recognition of printed markers, surface detection, placement of 3D content in response to real-world input, smooth AR rendering, intuitive user interaction through taps and gestures, and delivery of scientifically accurate and visually accessible educational material. The final product will run on ARCore compatible Android devices and present a self contained, immersive learning experience that supports the sponsor's goals of outreach, awareness, and public education.

2 System Overview

The AR Wetlands system is designed using a modular, layered architecture that separates hardware interaction, data processing, and application-level logic. This layered approach enhances system scalability, maintainability, and cross-platform adaptability. The primary objective of this system is to deliver an interactive augmented reality experience that educates users about environmental impact scenarios using a printed tabletop map and an Android-based AR mobile application.

The system is organized into three major architectural layers, each with distinct responsibilities

- Device Hardware Layer: Interfaces directly with the devices hardware, including camera, touchscreen, and sensors.
- Data Manager Layer: Serves as the intermediary between raw device input and app-level logic, interpreting and managing core data flows.
- App Layer: Renders visual output, handles user interaction logic, and presents educational content to the user.

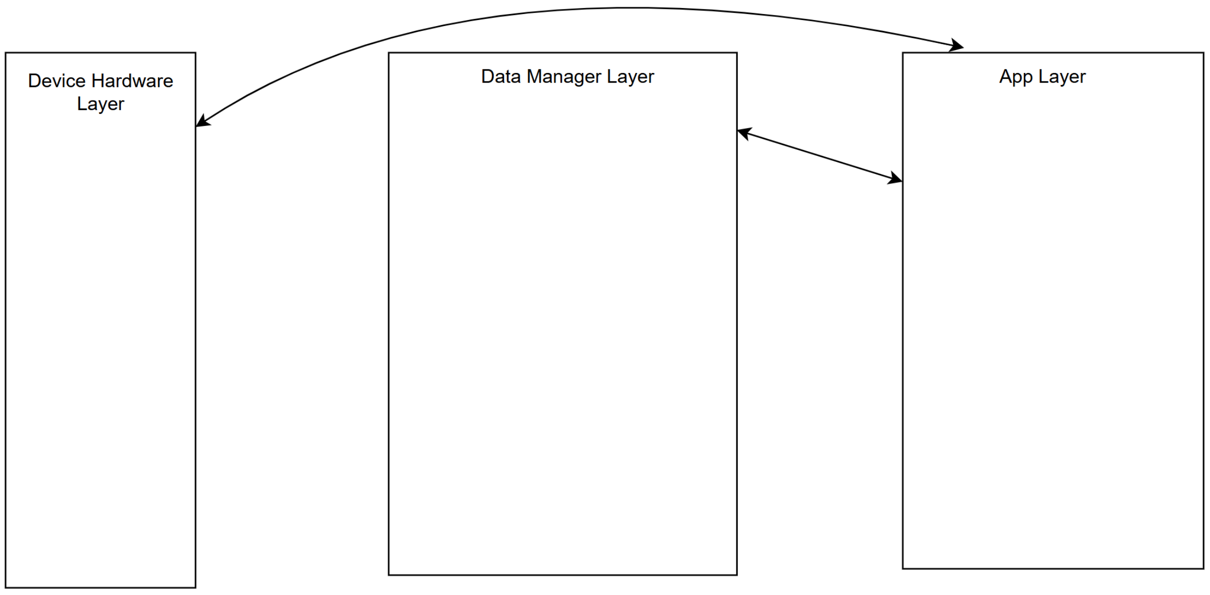

Figure 1: Base architectural layer diagram

2.1 Device Hardware Layer Description

The Device Hardware Layer is responsible for managing all interactions with the mobile device's built-in hardware. This includes receiving input from the capacitive touchscreen, accessing the live camera feed, and leveraging ARCore features such as marker detection. These capabilities serve as the foundation for higher-level AR interactions and real-time visual feedback.

Key Features:

- Touch input capture
- Real-time access to device camera feed
- Marker detection using ARCore

Interfaces: Sends raw camera data to the Surface Detection and Marker Recognition subsystems; transmits touch input to the Data Manager Layer for gesture processing and context validation.

2.2 Data Manager Layer Description

The Data Manager Layer functions as the system's core coordination hub. It receives raw data from the Device Hardware Layer, including touch inputs and surface detection results, and translates them into structured information used by the App Layer. This includes determining when and where to trigger animations, identifying valid marker locations, and forwarding interaction context to the rendering engine.

Key Features:

- Surface event interpretation
- Marker validation and plane tracking
- Interaction context management
- Input routing to appropriate app subsystems

Interfaces: Receives hardware-level inputs (camera, touch) from the Device Hardware Layer; forwards processed data and event triggers to the App Layer.

2.3 App Layer Description

The App Layer delivers the visual and interactive experience to the user. It renders 3D models, displays overlays and narration, and manages the educational flow. Based on the processed data from the Data Manager Layer, the App Layer determines how to visually respond to user actions and environmental changes, such as instantiating the entire AR scene when the specific marker is recognized.

Key Features:

- 3D model and animation rendering
- Text overlays, narration, and scene transitions
- Real-time feedback based on touch gestures and surface state

Interfaces: Receives input and trigger data from the Data Manager Layer; outputs AR visuals and feedback through the mobile device's screen and speakers.

3 Subsystem Definitions & Data Flow

Below is a detailed breakdown of the system's architecture into functional subsystems, organized across three major layers: *Device Hardware Layer*, *Data Manager Layer*, and *App Layer*. Each subsystem represents a core functional unit of the *AR Wetlands: Water Pollution Simulator*, operating together to deliver real-time, interactive augmented reality experiences on mobile devices.

The data flow diagram below illustrates how these subsystems interact through defined data paths, numbered for traceability. Arrows indicate the direction of data flow, and inter-layer communication is shown explicitly to reveal dependency and system cohesion.

Each subsystem shown in the diagram contributes to the real-time operation of the simulator. From user input and camera sensing to asset fetching, placement, animation, and interaction, this pipeline forms the backbone of the system. This layered, modular design ensures efficiency on mobile devices while maintaining a smooth user experience within Unity's AR Foundation framework.

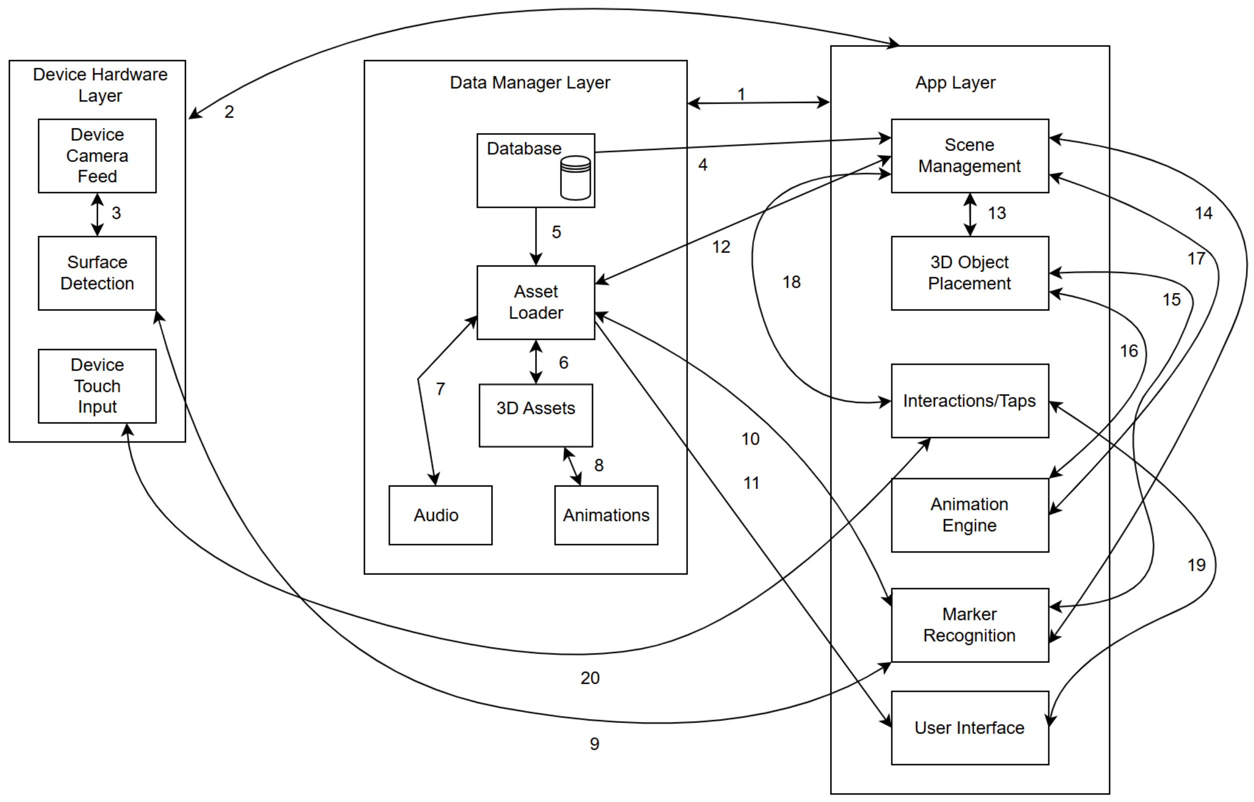

Figure 2: Full ADS data flow diagram

4 Device Hardware Layer Subsystems

This layer comprises the core input capabilities of the AR Wetlands: Water Pollution Simulator. These subsystems are tightly integrated with the mobile device's physical hardware and provide the foundational data streams required for surface tracking, scene augmentation, and user interaction.

4.1 Device Camera Feed

The Device Camera Feed subsystem captures the live video stream from the device's rear-facing camera. It serves as the visual foundation for the AR experience by providing a continuous feed used by other subsystems, especially for recognizing markers and detecting physical surfaces on the table. The real-time camera input is processed and forwarded to both the Surface Detection and Marker Recognition subsystems.

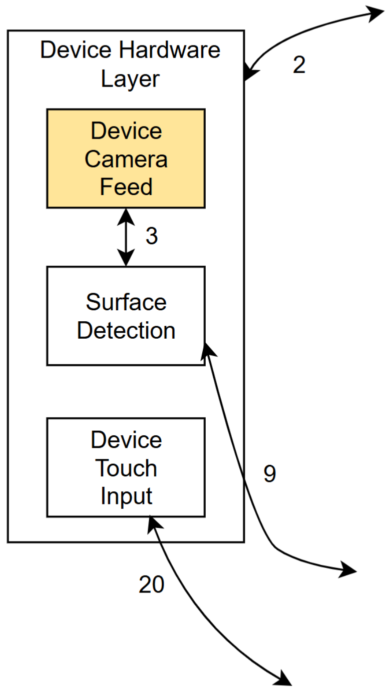

Figure 3: Device Camera Feed subsystem description diagram

4.1.1 Assumptions

- The device has a functional, rear-facing camera compatible with Unity's AR Foundation or ARCore.
- The device maintains low latency in capturing and processing frames.
- Camera access permissions must be granted at runtime for the subsystem to function properly.
- The camera frames can be forwarded in real time to both the Surface Detection subsystem and the Marker Recognition engine in the App Layer.

4.1.2 Responsibilities

- Continuously stream video feed from the mobile device's camera.

- Provide input for the Surface Detection subsystem for recognizing flat areas for AR object placement.
- The Marker Recognition for detecting predefined table-top markers.

4.1.3 Subsystem Interfaces

Table 2: Device Camera Feed Subsystem Interfaces

ID	Description	Inputs	Outputs
#1.1	Camera feed to Surface Detection	Device Camera	Raw camera frames
#1.2	Camera feed to Marker Recognition (App Layer)	Device Camera	Real-time image stream
#1.3	Request from Scene Manager (App Layer)	Trigger to initialize/restart camera	Live video stream

4.2 Surface Detection

The Surface Detection subsystem is responsible for analyzing the video input from the Device Camera Feed to identify real-world horizontal surfaces based on the markers. These surfaces serve as the anchors for placing 3D models in the augmented environment. This subsystem relies on AR frameworks to detect planes based on visual feature tracking (Markings) and provides the Scene Management and Object Placement modules with updated surface geometry.

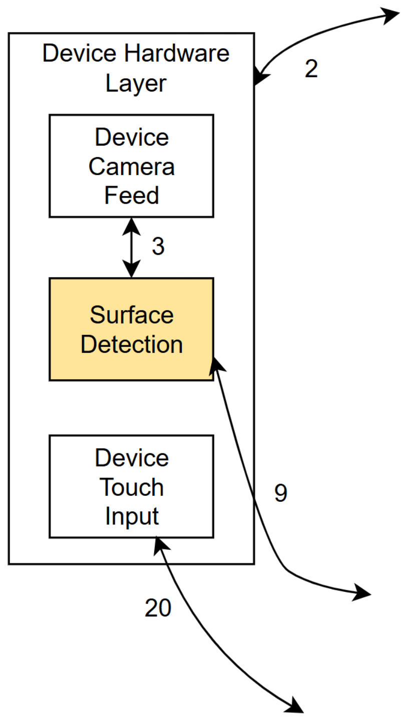

Figure 4: Surface Detection subsystem description diagram

4.2.1 Assumptions

- The device is AR-capable and supports surface detection via AR APIs such as ARCore.
- The environment is assumed to be well-lit and textured enough to enable effective feature tracking.
- It is assumed that the device's camera feed is uninterrupted and synchronized with the processing engine.
- The subsystem assumes that plane detection is enabled and accessible via Unity's AR framework and that performance optimizations are in place for mobile execution.

4.2.2 Responsibilities

- Process camera frames and detect horizontal planes suitable for object placement via the markers placed.
- These planes are communicated to the 3D Object Placement and Scene Management subsystems for further use.
- Ensures that only valid and stable and marked surfaces are considered, minimizing false positives.
- This subsystem actively updates surface boundaries as the user moves the device, allowing for dynamic adaptation of the virtual content to the real-world scene.

4.2.3 Subsystem Interfaces

Table 3: Surface Detection Subsystem Interfaces

ID	Description	Inputs	Outputs
#2.1	Camera feed input from Device Camera Feed	Raw video frames	Processed surface data
#2.2	Surface detection result to Scene Management	Processed surface data	Detected planes/surfaces
#2.3	Surface detection result to 3D Object Placement	Processed surface data	Surface transform coordinates

4.3 Device Touch Input

The Device Touch Input subsystem is responsible for capturing and interpreting all touch-based input from the mobile device. As shown in Figure 5, this subsystem sits within the Device Hardware Layer and interacts closely with the Surface Detection component to ensure accurate touch localization in the AR environment.

The touch input serves as the first point of interaction for users, enabling them to tap on the screen to trigger system responses such as starting animations, or triggering educational pop-ups. The Device Touch Input subsystem translates raw gesture data into structured input events and forwards them to the appropriate subsystems in the App Layer (such as the Interactions/Taps subsystem).

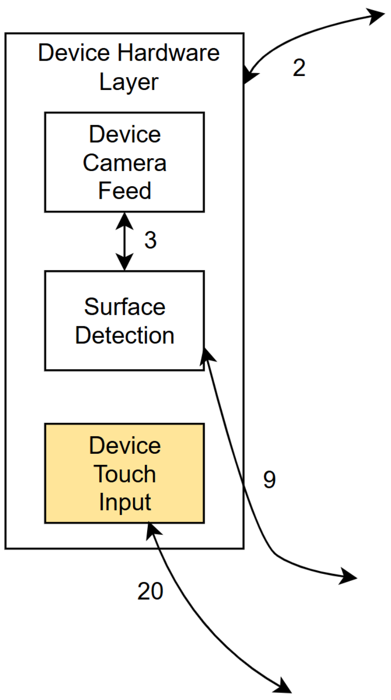

Figure 5: Device Touch Input subsystem description diagram

4.3.1 Assumptions

- The mobile device reliably detects both single and multi-touch gestures using its built-in capacitive touchscreen.
- The operating system is expected to provide stable and timely access to touch event APIs, ensuring that input data is delivered without delays.
- Touch inputs are assumed to occur only during active AR sessions to prevent unintended or misinterpreted interactions.
- The Surface Detection subsystem is expected to continuously provide up-to-date plane recognition data to support accurate and context-sensitive interpretation of touch gestures.
- Users will interact with the screen while holding the device in a standard orientation, either portrait or landscape, which enables consistent mapping of screen coordinates to the AR environment.

4.3.2 Responsibilities

- Detecting user touch events on the screen, identifying their location and type (e.g., tap, swipe, long press), and translating this data into actionable events for other subsystems.
- Provides raw touch coordinates and gestures to the Interactions/Taps subsystem for context-sensitive responses, such as object placement or scene transitions.
- Communicates with Surface Detection to ensure that user taps are registered on valid surfaces and not in empty or undetected regions.

- Ensures that only intentional and interpretable touch inputs are passed forward, filtering out accidental multi-touch or palm contact when possible.
- The subsystem operates in real time to maintain responsiveness and accuracy, which are critical to user engagement in an AR environment.

4.3.3 Subsystem Interfaces

Table 4: Device Touch Input Subsystem interfaces

ID	Description	Inputs	Outputs
#3.1	Interface between Device Touch Input and Interactions/Taps subsystem	Screen touch coordinates, gestures	Interaction event trigger
#3.2	Communication with Surface Detection for touch validation	Surface data	Placement-valid gesture data
#3.3	Forwarded touch gestures to App Layer subsystems	Gesture events	Tap commands, scene triggers

5 Data Manager Layer Subsystems

The Data Manager Layer is responsible for providing all structured content used by the AR Wetlands system during runtime. It includes internal data management subsystems that collectively support asset delivery, visual rendering, and scenario construction. These subsystems operate offline and serve as the foundation for dynamic scene generation and user interaction. This layer ensures that 3D models, animations, audio, and educational content are preloaded and retrievable within the Unity environment. Each subsystem in this layer contributes to resource availability, playback synchronization, and environmental simulation accuracy. The components described below form the internal backbone of the application's runtime behavior, supplying the App Layer with critical scene elements on demand.

5.1 Database

The Database subsystem is responsible for managing the application's structured internal data, including scene configurations, environmental parameters, and relationships between AR objects and their corresponding markers. This subsystem serves as a local storage system embedded within the Unity application, ensuring that all necessary information is accessible offline. It supports key operations by providing scene metadata, marker-to-object mappings, and references to educational content. During runtime, it supplies the Scene Management and Asset Loader subsystems with the data required to accurately construct and populate AR scenarios.

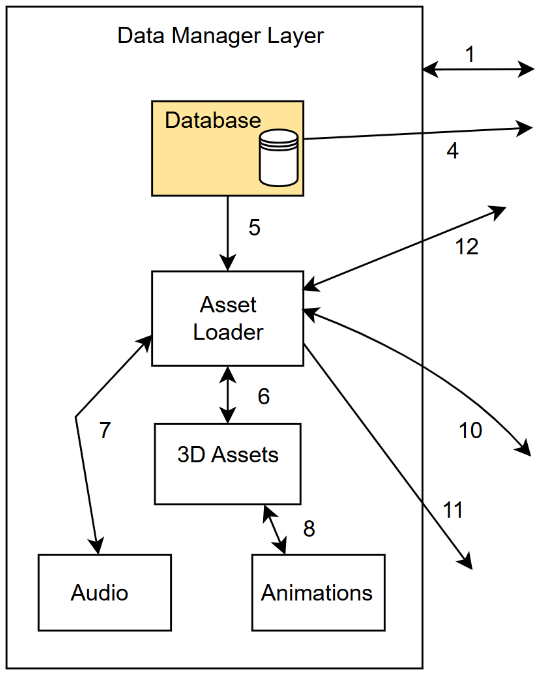

Figure 6: Database subsystem description diagram

5.1.1 Assumptions

- The database will be embedded into the Unity application.
- The subsystem is expected to operate entirely offline, without runtime network connectivity.

- Data access patterns will be read-only after installation, and the structure of the database will remain consistent throughout the life cycle of the app.
- Interfacing subsystems, such as Scene Management, are expected to use a fixed schema when retrieving configuration data.

5.1.2 Responsibilities

- Providing all stored configuration data necessary to build each scene.
- Linking printed markers to AR content, defining interaction logic, and supplying contextual educational content.
- Support Asset Loader operations by identifying which 3D models and audio elements are required for a given marker or scenario.
- Aids Scene Management in determining which elements are visible or interactive within a particular context.

5.1.3 Subsystem Interfaces

Table 5: Database Subsystem interfaces

ID	Description	Inputs	Outputs
#4.1	Load scene metadata	N/A	Scene parameters for Scene Management
#4.2	Load asset references	N/A	Asset tags for Asset Loader

5.2 Asset Loader

The Asset Loader subsystem serves as the central mechanism for loading 3D models, audio clips, and animation files into the AR scene. It is responsible for retrieving these resources based on identifiers provided by the database or marker recognition. This subsystem is highly performance-sensitive, ensuring that assets are streamed or preloaded efficiently for optimal runtime responsiveness.

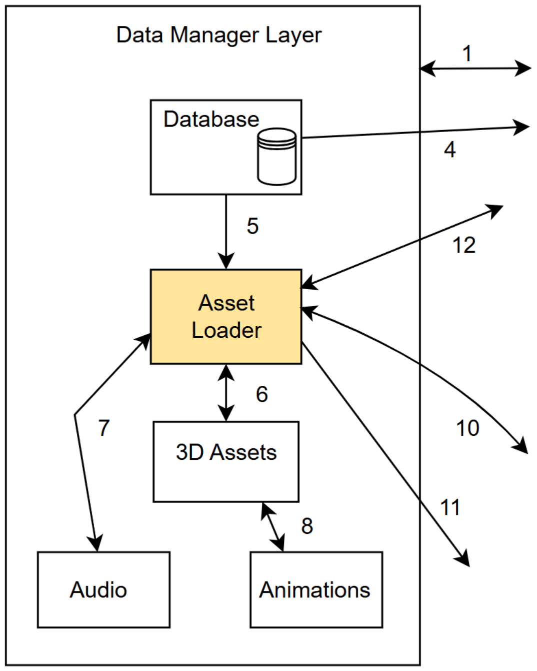

Figure 7: Asset Loader subsystem description diagram

5.2.1 Assumptions

- All assets will be packaged with the application or pre-downloaded to the device before use.
- The subsystem relies on Unity's asset loading systems to retrieve models and media efficiently.
- Marker recognition and user interface systems are assumed to wait for successful asset retrieval before proceeding with display or interaction.

5.2.2 Responsibilities

- Identifying, locating, and loading assets referenced by the database or triggered through user interaction.
- Provides assets to the Scene Management subsystem to populate scenarios and to the Marker Recognition subsystem to associate content with identified markers.
- Supports the User Interface by supplying icons, thumbnails, or other graphical elements as needed.

5.2.3 Subsystem Interfaces

Table 6: Asset Loader Subsystem interfaces

ID	Description	Inputs	Outputs
#5.1	Load asset references from database	Asset paths	Loaded objects to memory
#5.2	Asset request from marker recognition	Marker IDs	Associated asset instances
#5.3	Provide assets to UI	N/A	Icons or visuals
#5.4	Supply asset links to Scene Management	Asset bundles	Render-ready models
#5.5	Load 3D asset from 3D Assets subsystem	Asset name or ID	3D model prefab
#5.6	Load audio clip from Audio subsystem	Audio clip ID	Audio file for playback

5.3 3D Assets

The 3D Assets subsystem manages the physical representations of models used within the AR scenes. These include houses, drainage infrastructure, construction equipment, and environmental props. Models are designed to be mobile-friendly, optimized for performance and quick rendering on ARCore compatible devices.

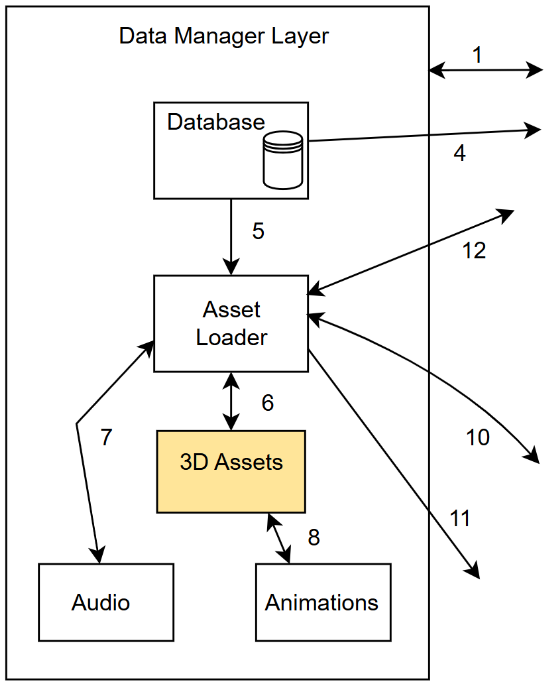

Figure 8: 3D Assets subsystem description diagram

5.3.1 Assumptions

- All models are prebuilt, tested for AR compatibility, and bundled with the application.
- Assets are expected to include appropriate metadata such as collider data and anchor points.

- Each model is designed to visually align with environmental scenarios and can be animated or interacted with as required.

5.3.2 Responsibilities

- Provides prefabricated models to the scene, either directly or via the Asset Loader.
- These models must support interaction detection, spatial anchoring, and animation integration.
- Form the core visual content that users engage with throughout the AR experience.

5.3.3 Subsystem Interfaces

Table 7: 3D Assets Subsystem interfaces

ID	Description	Inputs	Outputs
#6.1	Asset spawn request	Asset Loader	3D model prefabs
#6.2	Connect to animation system	Model reference	Animation-ready object

5.4 Audio

The Audio subsystem handles playback of environmental sounds, voice narration, and audio effects triggered by interactions. All audio is localized and packaged with the application to ensure offline capability.

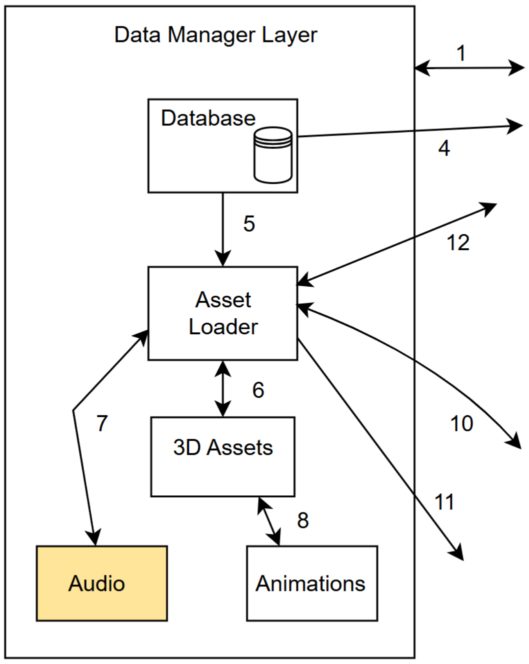

Figure 9: Audio subsystem description diagram

5.4.1 Assumptions

- All required audio files will be included with the application, using efficient formats such as MP3.
- The playback will be driven by Unity's audio components and managed in sync with user interactions or scenario progression.
- Audio files are expected to match the content being presented visually or narratively.

5.4.2 Responsibilities

- Delivering sound that enhances the educational message.
- Playing narration when a user taps on a model, triggering ambient sounds during animations, or delivering instructional cues.
- Ensures that audio playback is synchronized with animations or scene transitions for immersive feedback.

5.4.3 Subsystem Interfaces

Table 8: Audio Subsystem interfaces

ID	Description	Inputs	Outputs
#7.1	Request audio from loader	Audio asset ID	Audio clip to device
#7.2	Trigger playback	Interaction ID	Sound output

5.5 Animations

The Animations subsystem enables movement, transformations, and effects that visually represent environmental processes such as water runoff, flooding, or chemical dispersion. Animations are pre-made within Unity and tied to specific objects or scenarios.

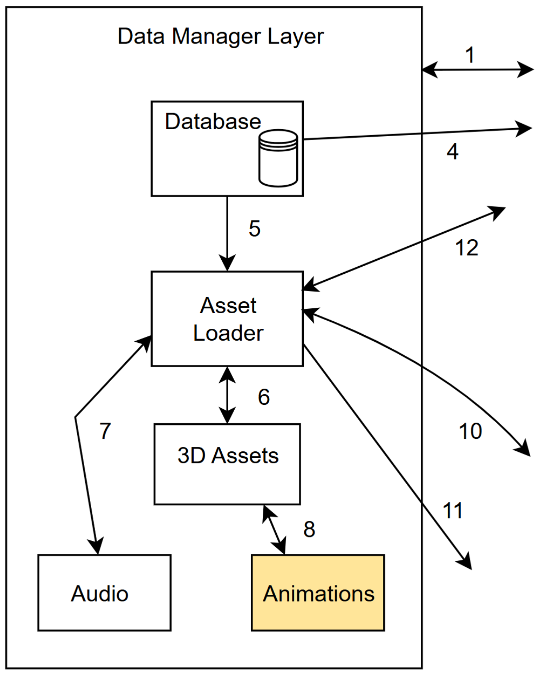

Figure 10: Animations subsystem description diagram

5.5.1 Assumptions

- All animations will be timeline or script-driven and pre-integrated with corresponding 3D assets.
- Relies on events or triggers from the Scene Management and Interactions subsystems to activate animations at the appropriate time.

5.5.2 Responsibilities

- Bringing environmental dynamics to life.
- Visualizing the flow of water, the movement of trash, or other key actions in each scenario.
- Must coordinate with Scene Management to play the right sequence, and with Interactions to respond to user taps or system events.

5.5.3 Subsystem Interfaces

Table 9: Animations Subsystem interfaces

ID	Description	Inputs	Outputs
#8.1	Link animation to object prefab	Asset reference	Animator controller applied
#8.2	Trigger animation from scene manager	Scenario ID	Animated 3D visual effect

6 App Layer Subsystems

The App Layer is responsible for everything the user directly sees and interacts with in the AR application. It serves as the main interface between the system's internal data and the user, ensuring that the experience is intuitive, responsive, and educational. This layer is designed with user-friendliness as a core priority, making it easy for users to understand and engage with the AR scenes. The App Layer will allow users to view an interactive AR scene projected onto a physical table or surface. Users can tap on objects within the scene to receive visual or animated responses, such as polluted water becoming clean or trash disappearing. This interaction supports the educational goals of the project by helping users understand the impact of pollution in a visual and engaging way.

6.1 Scene Management

Regarding the AR water pollution visualization app, the Scene Management initializes the AR session, loads the appropriate 3D assets (such as trash items, clean water models, or animations), and coordinates their placement based on user interaction and surface detection. It also ensures that once a marker or surface is detected, the correct AR content is triggered and displayed without delay.

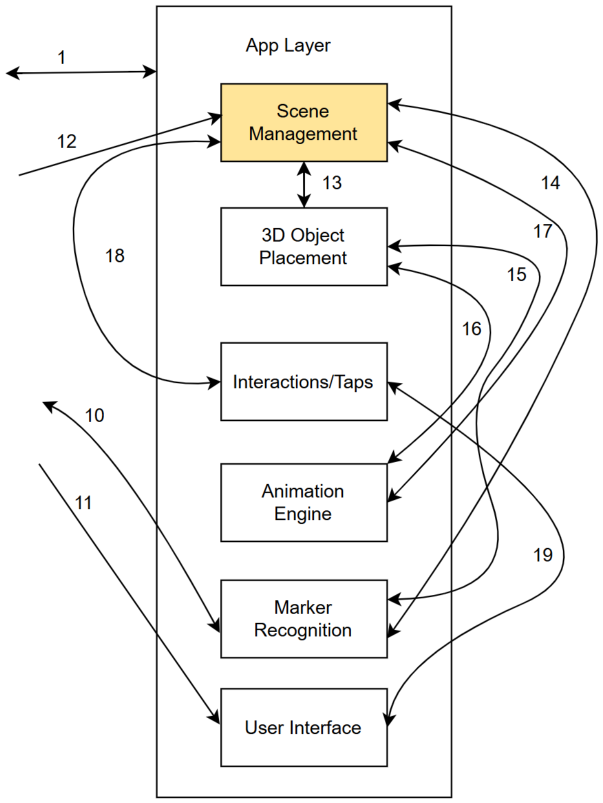

Figure 11: Scene Management subsystem description diagram

6.1.1 Assumptions

- The Device Hardware Layer (camera, surface detection, and touch input) functions correctly and consistently provides real-time input to the App Layer.
- The Asset Loader from the Data Manager Layer successfully loads all required assets.

- The Marker Recognition subsystem accurately detects and identifies markers.
- Users will operate the application in an appropriate environment (e.g., well-lit area with a flat surface), enabling reliable surface and marker detection.
- The mobile device used will have sufficient hardware resources (e.g., RAM, CPU, battery life) to support smooth scene initialization and transitions.

6.1.2 Responsibilities

The Scene Management subsystem is responsible for controlling the flow, visibility, and life cycle of AR content presented to the user

- **Scene Initialization:** Prepares the AR session by loading required assets, setting up surface detection, and waiting for marker or surface recognition.
- **Scene Transitions:** Manages transitions between different stages of the experience (e.g., intro screen, AR view, end screen).
- **Trigger Handling:** Responds to input from marker recognition and user interactions (e.g., taps) to update or reload scene elements accordingly.

6.1.3 Subsystem Interfaces

Table 10: Scene Management Subsystem interfaces

ID	Description	Inputs	Outputs
#9.1	Interface from Asset Loader to receive 3D objects, audio, and animations	Loaded 3D assets Audio Animations	Renderable scene content
#9.2	Interface from User Interface and Interaction subsystem for handling user input	Touch input Tap commands	Scene transitions State changes
#9.3	Interface with Marker Recognition subsystem to identify and respond to visual markers	Marker ID Marker position	Scene activation Object placement

6.2 3D Object Placement

The 3D Object Placement subsystem is responsible for accurately positioning virtual objects in the user's physical environment, based on surface detection data from the device's camera feed and input from the Scene Management and Marker Recognition subsystems. This subsystem takes the 3D assets provided by the Asset Loader and places them into the scene at positions determined by detected image markers.

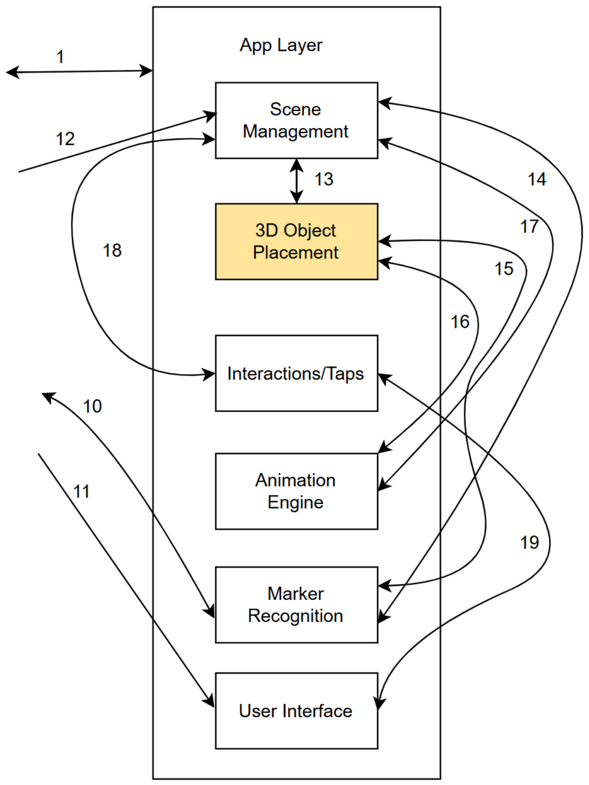

Figure 12: 3D object Placement subsystem description diagram

6.2.1 Assumptions

- There has already been solidified valid planes before placement occurs.
- The Asset Loader provides fully loaded and correctly formatted 3D models ready for placement.
- The user interface delivers accurate tap or interaction data when requesting object placement.
- Marker Recognition will reliably detect and identify markers prior to placement being triggered.
- The device's camera feed and AR tracking are functioning without lag or significant drift.

6.2.2 Responsibilities

- Responsible for placing 3D assets on detected surfaces based on user interaction or marker detection.
- Receives user tap or interaction input to determine placement coordinates.
- Communicates with the Asset Loader to receive 3D objects for display.
- Coordinates with Scene Management to store object position, orientation, and scene updates.
- Works with Marker Recognition to enable placement of 3D assets on top of detected AR markers.
- Ensures correct scaling and orientation of placed objects based on real-world surfaces.

6.2.3 Subsystem Interfaces

Table 11: 3D Object Placement Subsystem interfaces

ID	Description	Inputs	Outputs
#10.1	Interface with the Asset Loader for retrieving 3D models to place	Object ID, asset request	3D object model
#10.2	Interface with Marker Recognition to trigger placement on recognized markers	Marker ID	Placement command to scene

6.3 Interactions/Taps

The Interactions/Taps subsystem is responsible for capturing and interpreting user input, primarily through screen taps. This subsystem plays a key role in enabling users to interact with virtual elements within the augmented environment. It detects taps on the screen, interprets their purpose (such as selecting, placing, or animating a 3D object), and passes the appropriate data to other connected subsystems for further processing.

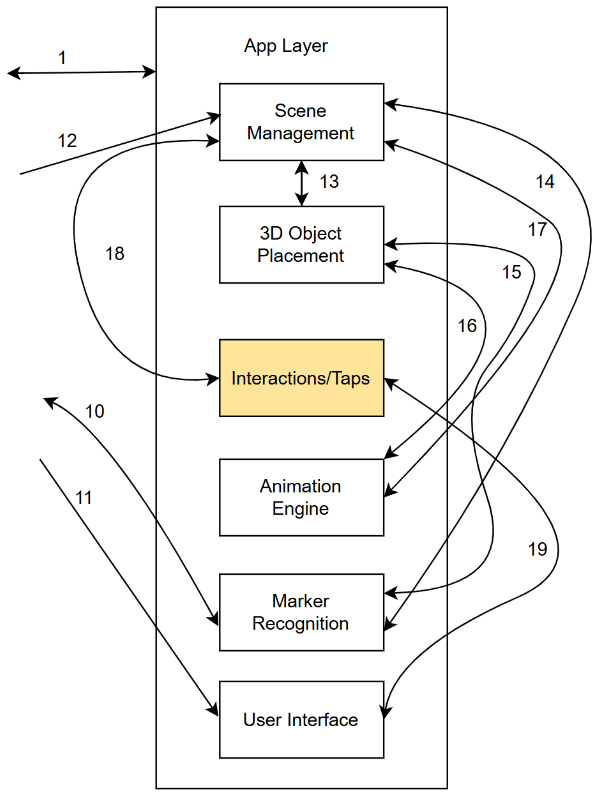

Figure 13: Interactions/Taps subsystem description diagram

6.3.1 Assumptions

- It is assumed that the User Interface subsystem will successfully detect tap events and forward them to the Interactions/Taps subsystem without delay.
- The App Layer will be consistently responsive, allowing real-time updates and feedback from tap interactions.
- All subsystems such as Scene Management, 3D Object Placement, and Animation Engine are available and functional when tapped interactions are triggered.
- Marker Recognition will have already processed and identified any markers on the screen prior to tap interactions, ensuring accurate context for response.

6.3.2 Responsibilities

- Capture and interpret screen tap gestures from the user interface.
- Determine the context of a tap event (e.g., object selection, animation trigger, etc.).
- Relay interpreted actions to the appropriate subsystems (Scene Management, 3D Object Placement, or Animation Engine).
- Coordinate with the App Layer to ensure user interactions are intuitive and reflect in real time.

6.3.3 Subsystem Interfaces

Table 12: Interactions/Taps Subsystem interfaces

ID	Description	Inputs	Outputs
#11.1	Communication between User Interface and Interactions/Taps subsystem. Detects tap gestures.	Tap coordinates, gesture signals	Interaction trigger
#11.2	Data sent from Interactions/Taps to Scene Management to update AR scene state	User-selected object ID	Scene update request

6.4 Animation Engine

The Animation Engine is responsible for managing and executing all visual animations in response to user interactions or system events. It receives input triggers and converts them into smooth, real-time animations within the augmented reality environment. The engine ensures consistency in transitions, visual feedback, and timing. It communicates closely with the Scene Management, Interactions/Taps, and User Interface subsystems to provide a cohesive and immersive user experience.

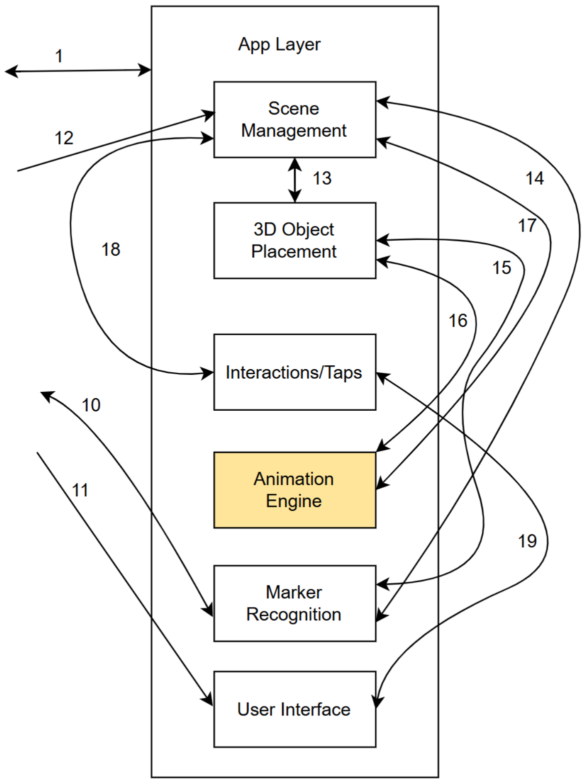

Figure 14: Animation Engine subsystem description diagram

6.4.1 Assumptions

- Animations are triggered through tap gestures, scene transitions, or predefined timelines managed by Scene Management.
- Animations rely on the position and state of 3D objects in the AR scene for context (e.g., flood level, trash location).
- Unity's animation tools (Animator Controller) are used as the runtime execution engine.
- Timing, transitions are handled within Unity's update loop or event system.

6.4.2 Responsibilities

- Execute animations such as flooding, flowing water, trash accumulation, or erosion effects based on interaction input or scene logic.
- Coordinate visual transitions across multiple assets for scene consistency.
- Manage animation states and transitions in sync with user actions.
- Communicate feedback to Scene Management or UI.

6.4.3 Subsystem Interfaces

Table 13: Animation Engine Subsystem interfaces

ID	Description	Inputs	Outputs
#12.1	Interaction triggers (tap, scene transition) initiate specific animation events.	Tap ID, Scene Event	Animation Start Signal
#12.2	Receives object positioning data for context-sensitive animations .	Object Coordinates, Model Type	Adjusted Animation States
#12.3	Sends completion signal to UI/Scene Manager to indicate animation is done.	N/A	Completion Callback, Visual Feedback
#12.4	Receives animation state transitions from Scene Manager.	Scene Flags, Mode	Animation Layer Config

6.5 Marker Recognition

The Marker Recognition subsystem is responsible for detecting predefined visual marker placed in the center of the AR table. This marker serves as an anchor to initiate the entire AR experience once identified by the mobile device's camera. The system relies on the marker to register the table's surface and trigger the display of the full animated environment.

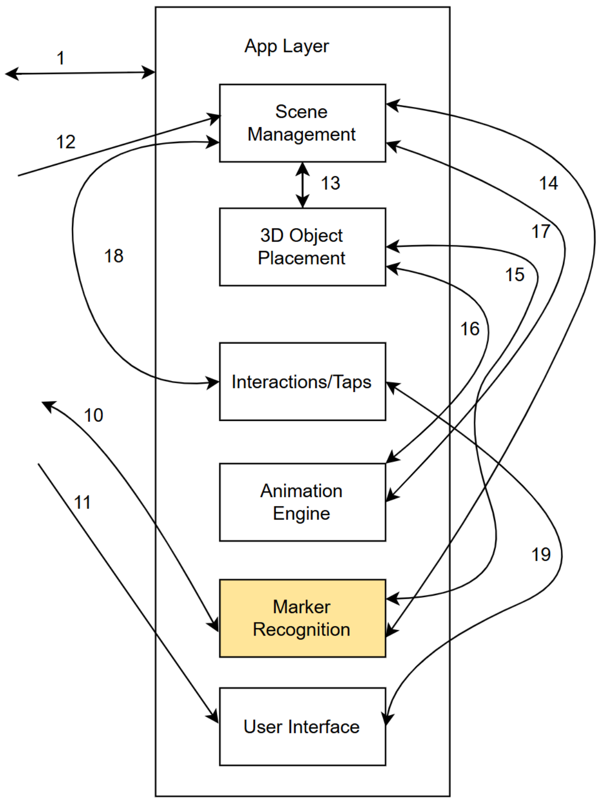

Figure 15: Marker Recognition subsystem description diagram

6.5.1 Assumptions

- Marker tracking is implemented using Unity's AR Foundation tools or compatible AR libraries.
- The table's physical dimensions and marker positions are consistent and fixed throughout all usage scenarios.
- Recognition of center marker is sufficient to register the entire table plane and initiate the AR scene.
- Lighting conditions will be controlled to ensure reliable marker detection using standard mobile device cameras.

6.5.2 Responsibilities

- Detect and identify visual marker placed on the center of the table using the device camera feed.
- Trigger the AR scene initialization process once a marker is successfully recognized.
- Maintain alignment between the virtual environment and the physical table surface using anchor transforms.
- Reconfirm marker presence periodically to prevent drift or misalignment of the AR content.

6.5.3 Subsystem Interfaces

Table 14: Marker Recognition Subsystem interfaces

ID	Description	Inputs	Outputs
#13.1	Captures live camera feed and scans for center marker.	Camera Image Stream	Marker Detection Signal
#13.2	Sends marker detection result to Scene Manager to begin AR scene playback.	Marker ID	Scene Activation Command
#13.3	Sends spatial transform data for alignment of AR content to the Animation Engine.	Marker Pose	Anchor Transform
#13.4	Receives configuration data defining marker patterns and physical placement.	Marker Config File	N/A

6.6 User Interface

The User Interface (UI) subsystem is responsible for presenting information, supporting limited user interaction, and guiding users through the AR experience. Although the system is designed primarily for passive viewing, it does support basic interactions such as taps or gestures to initiate or advance scenes. The interface provides visual overlays, labels, and optional prompts.

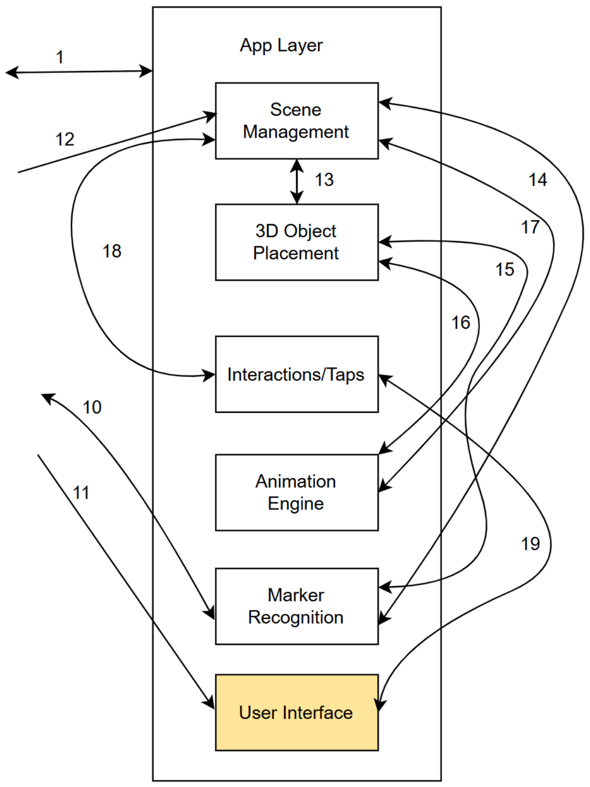

Figure 16: User Interface subsystem description diagram

6.6.1 Assumptions

- Users may interact with the system through simple touch gestures or taps to advance scenes or trigger animations.
- The mobile device running the AR application will have a responsive touch screen and sufficient display size.
- Most of the user experience will still be guided automatically through audio and animation sequences.

6.6.2 Responsibilities

- Present clear visual overlays, labels, or highlights related to AR content (e.g., construction zones, storm drains).
- Manage the application flow: Displaying the Start Screen, presenting the Terms of Service modal, requesting Android Camera permissions, and rendering floating audio buttons over the active AR scene.
- Synchronize visual UI prompts with animation and narration timing.
- Provide access to a hidden admin interface for system testing or resetting.
- Ensure all on-screen content remains legible and accessible in real-world lighting conditions.

6.6.3 Subsystem Interfaces

Table 15: User Interface Subsystem interfaces

ID	Description	Inputs	Outputs
#14.1	Floating UI buttons overlaying the AR scene for audio narration control.	Scene Activation Signal	Audio Buttons Visibility
#14.2	User interactions for App Start, Terms of Service acceptance, and Audio toggles.	Single Tap Input	Scene Initialization, Audio Playback
#14.3	Permission handling for device camera access upon startup.	Permission Grant (User 'Allow')	Activate AR Camera Feed

References