SnapSphere

Immersive Photo Experience

Final Year Project Report
29.04.2025

SUPERVISOR
Dr. Sinead O'Neill
Sinead.ONeill@setu.ie
South East Technological University
School of Science
Department of Computing and Maths

AUTHOR
Yuzhe Shi
20108862@mail.wit.ie
South East Technological University

Contents

1. Introduction 3
1.1. Project Overview 3
2. Technology Stack and Tools 4
2.1. Mobile App Development 4
2.1.1. Core Technologies 4
2.1.2. Key Features 5
2.1.3. Development Tools 5
2.2. Frontend Development 5
2.2.1. Core Technologies 5
2.2.2. Key Features 6
2.2.3. Development Tools 6
2.3. Backend Development 6
2.3.1. Core Technologies 6
2.3.1.1. Cloud Services 7
2.3.2. Key Features 7
2.3.3. Development Tools 7
2.4. Development Infrastructure 8
2.4.1. Version Control 8
2.4.2. CI/CD Pipeline 8
2.4.3. Monitoring and Analytics 8
2.4.4. Codegen 9
3. System Design 10
3.1. Overall Architecture 10
3.1.1. Core Technologies 10
3.1.2. System Architecture 10
3.2. Module Breakdown 10
3.2.1. Core Modules 10
3.2.2. Geographic Modules 10
3.2.3. Content Management 10
3.2.4. Consensus Modules 10
3.2.5. Utility Modules 11
3.3. Database Design 12
3.3.1. Database schema 12
3.3.2. Core Tables 13
3.3.2.1. Cloud Anchor Table 13
3.3.2.2. Geo Object Table 13
3.3.2.3. Scene Table 13
3.3.2.4. User Table 14
3.3.2.5. Role Table 14
3.3.2.6. Scene Managers User Table 14
3.3.2.7. Scene Labels Label Table 14
3.3.2.8. Label Table 15
3.3.2.9. File Table 15
3.3.2.10. Image Copyright Table 15
3.3.3. Foreign Key Constraints 16
3.3.4. Indexes 17
3.3.5. Database Features 18

3.3.5.1. Spatial Data Types 18
3.3.5.2. Spatial Query 18
3.3.6. Point-in-Rectangle Search 18
3.3.7. Radius Search 18
3.3.8. K-Nearest Neighbors 19
3.3.9. Spatial Indexing 19
3.4. Key Technologies and Implementation 19
3.5. AR Object 19
3.6. Cloud Anchor 20
3.7. Positioning 22
3.7.1. Scene Graph 22
3.7.2. Relative Positioning 23
3.7.3. Absolute Positioning Fallback 24
3.8. Scene Management 24
3.9. Invisible Watermarking 24
3.9.1. Embedding Process 24
3.9.2. Extraction Process 27
3.9.3. Technical Considerations 27
3.10. Blockchain-based Copyright Protection 27
3.10.1. On-chain Evidence 27
3.10.2. Embedded Identifier Extraction Enhancement 29
3.11. Image Similarity Search 29
3.11.1. Perceptual Hashing 29
3.11.2. Image Similarity Search 30
3.12. Blockchain-based Zero-Knowledge Proof for Photo Ownership . . 31
3.12.1. Introduction to Zero-Knowledge Proofs 31
3.12.2. Mathematical Foundations 31
3.12.2.1. Finite Fields and Elliptic Curves 31
3.12.2.2. Poseidon Hash Function 31
3.12.2.3. Groth16 Proof System 32
3.12.3. Circuit Design 32
3.12.3.1. Ownership Proof Circuit 32
3.12.3.2. Circuit Constraints 32
3.12.3.3. Comparison with RSA 32
3.12.3.4. Note 33
3.12.3.4.1. Comparison with hash Only 33
3.12.4. System Architecture 33
3.12.4.1. Data Flow 33
3.12.5. Security Analysis 34
3.12.5.1. Zero-Knowledge Property 34
3.12.5.2. Completeness 34
3.12.5.3. Practicality 34
3.12.6. Implementation Details 34
3.12.6.1. Key Management 34
3.12.6.2. On-chain Storage 34
3.12.7. Performance Optimization 34
3.12.7.1. Proof Generation 34
3.12.7.2. On-chain Queries 34

3.13. Web Management Interface 35
3.13.1. Frontend Architecture 35
3.13.2. UI/UX Design 35
3.13.3. Performance Optimization 35
3.13.4. State Management 35
3.13.5. Development Tools 35
3.14. Authentication System 35
3.14.1. JWT Authentication 35
3.14.1.1. Token Structure 35
3.14.1.2. Security Features 35
3.14.1.3. Implementation 35
3.14.1.4. Token Revocation 36
3.14.1.5. Redis Implementation 36
3.14.1.6. Security Considerations 36
3.14.2. Access Control with Nest-Access-Control 36
3.14.2.1. Resource-based Access Control and Attribute-based Access Control 36
3.14.2.2. Permission Definition 36
3.14.2.3. Resource Protection 37
3.14.2.4. Dynamic Access Control 37
3.15. Special Features 38
3.15.1. Interpage Animations 38
4. Unimplemented or Future Features 39
4.1. Scripting 39
4.2. Offline Mode 39
5. Development Process 40
5.1. Methodology 40
5.2. Version Control 40
6. Deployment 41
6.1. Mobile Application Deployment 41
6.2. Web Deployment 41
6.3. Backend Deployment 41
6.4. Database 42
7. User Manual 44
7.1. Mobile App Usage 44
7.1.1. Scanning and Hosting Cloud Anchors 44
7.1.1.1. Scanning a Cloud Anchor 44
7.1.1.2. Hosting a Cloud Anchor 44
7.1.2. Viewing 45
7.1.3. Create a 3D object 45
7.1.4. Edit a 3D object 46
7.2. User Panel Guide 48
7.2.1. Login/Register 48
7.2.2. User Profile 49
7.2.3. Scene Management 49
7.2.4. Cloud Anchor Management 51
7.2.5. Photo Management 53
7.2.6. Comment Management 54

7.2.7. Label Management 56
7.2.8. Verify Image Copyright 57
7.2.9. Zero-Knowledge Proof 59
8. Conclusion 64
8.1. Project Summary 64
8.2. Future Work 64
9. Appendix 66
9.1. A Simple Example of Zero-Knowledge Proof 66
9.1.1. The Magic Cave Story[1] 66
9.1.2. The Challenge 66
9.1.3. The Proof Process 67
9.1.4. Demonstrating ZKP Properties 67
9.2. Why using Incomplete Zero-Knowledge Proofs? 67
9.3. Semester 1 Project Proposal 68
9.3.1. Motivation and Background 68
9.3.2. Objectives and Scope 69
9.3.2.1. General Users 69
9.3.2.2. Content Creators 70
9.3.2.3. Technical Scope 70
9.3.3. Feasibility Study 70
9.3.4. Requirements Analysis 71
9.3.4.1. Functional Requirements 71
9.3.4.1.1. User Authentication 71
9.3.4.1.2. Content Creation 71
9.3.4.1.3. Web Content Management 71
9.3.4.1.4. Backend Review and Analytics 71
9.3.4.1.5. Non-Functional Requirements 71
9.3.4.1.6. Response Time 71
9.3.4.1.7. Data Consistency 72
9.3.4.1.8. Privacy Protection 72
9.3.4.1.9. Security Requirements 72
9.3.4.1.10. Scalability Requirements 72
9.3.4.1.11. Usability Requirements 72
9.3.5. Risk Management 72
9.3.5.1. Technical Challenges 72
9.3.5.1.1. AR Framework Selection 72
9.3.5.1.2. Algorithm Implementation 73
9.3.5.1.3. Development and Debugging 74
9.3.5.2. Risk Mitigation Strategies 74
9.3.5.2.1. Technical Risk Management 74
9.3.5.2.2. Development Process 74
9.3.5.2.3. Quality Assurance 74
Bibliography 75

Yuzhe Shi
20108862@mail.wit.ie

This project is a part of my final year project in
South East Technological University.

This project is open source and available on GitHub:

- Frontend: https://github.com/bkmashiro/fyp-next
- Backend: https://github.com/bkmashiro/fyp-backend
- Demo App: https://github.com/bkmashiro/fyp-next-unity
- Landing Page: https://fl.yuzhes.com
- Production Demo: https://fyp.yuzhes.com
- Report: https://github.com/bkmashiro/fyp-report

TL;DR:

The most interesting part:

- Invisible watermarking (Section 3.9)
- Blockchain-based copyright protection (Section 3.10)
- Fast full-database image search (Section 3.11)
- Zero-knowledge proof for photo ownership (Section 3.12)

Statement of Original Authorship

I hereby declare that this proposal, entitled *SnapSphere - Immersive photo experience - Final Year Project Report*, is the result of my own independent work, and all sources of information and assistance have been fully acknowledged. I confirm that, to the best of my knowledge, this work has not been previously submitted to any other academic institution for a degree or diploma, either in part or in whole.

Any external contributions or collaborative input have been cited appropriately, and all necessary permissions for the use of copyrighted materials have been obtained.

I understand the implications of academic integrity policies and am aware that any breach of these policies may result in academic penalties.

Signature (Seal): ..... **Date:** Apr.29, 2025

1. Introduction

1.1. Project Overview

This project consists of a **mobile app** and **a web management interface** and a **backend service**. The mobile app allows users to capture and view photos, send comments and likes in Augmented Reality (AR). The web management interface allows administrators to manage the scenes and the photos. The backend service provides a RESTful and websocket API for the mobile app and the web management interface to interact with each other.

The system provides users with an immersive augmented reality experience, allowing them to enter a virtual AR world where they can load and explore various scenes. Within these scenes, users can view and interact with 3D elements, create and place new AR content, and leave comments to share their thoughts. The web interface complements this experience by enabling users to discover other people's creations, manage their own scenes, and engage with the broader community. This creates a dynamic ecosystem where users can both experience and contribute to the AR environment, fostering creative expression and social interaction.

Figure 1: Conceptual diagram of the system

The project implements a comprehensive security framework that combines machine learning, blockchain, invisible watermarking, and zero-knowledge proofs to create a robust system for data security and privacy protection. This multi-layered approach ensures tamper-proof ownership records, efficient copyright protection, and privacy-preserving verification while maintaining user data confidentiality.

Note that the content covered in the Proposal has been moved to Section 9.3

2. Technology Stack and Tools

2.1. Mobile App Development

2.1.1. Core Technologies

- Unity 6.0[2]
  - A game engine that makes it easy to create 3D and AR applications
  - Provides tools for building apps for both iOS and Android
  - Handles complex graphics and physics calculations

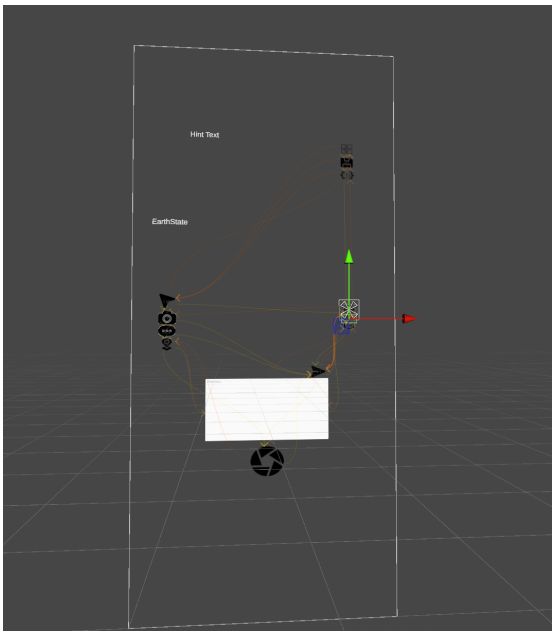


Figure 2: Unity Editor - App UI

Figure 3: Unity Editor - Hierarchy

- AR Foundation[3]
  - Unity's framework for building AR applications
  - Works with both ARCore (Android) and ARKit (iOS)
  - Manages camera, tracking, and AR features
- C# Programming Language[4]
  - The main programming language used in Unity
  - Easy to learn and use
  - Good for game and app development
- Android/iOS Platform Support
  - ARCore[5] for Android devices
  - ARKit[6] for iOS devices
  - Ensures AR works on both major mobile platforms

2.1.2. Key Features

- Cross-platform AR capabilities
  - Same code works on both Android and iOS
  - Consistent AR experience across devices
  - Easy to maintain and update
- Cloud anchor integration
  - Lets multiple users see AR content in the same place
  - Works with Google's ARCore
  - Makes shared AR experiences possible
- Spatial content management
  - Organizes AR objects in 3D space
  - Handles object placement and movement
  - Manages interactions between objects
- Real-time user interactions
  - Instant response to user actions
  - Smooth animations and transitions
  - Natural user experience

2.1.3. Development Tools

- Unity Editor
  - Visual tool for building AR scenes
  - Built-in debugging tools
- Visual Studio Code
  - Text editor

2.2. Frontend Development

2.2.1. Core Technologies

- Nuxt.js 3[7]
  - A framework for building modern web applications
  - Makes Vue.js development easier
  - Handles routing and server-side rendering

- TypeScript[8]
  - Adds type checking to JavaScript, better safety
- Tailwind CSS[9]
  - A utility-first CSS framework
  - Makes styling faster and easier
  - Responsive design made simple
- Vue 3 with Composition API[10]
  - Better code organization
  - More flexible

2.2.2. Key Features

- Server-side rendering (SSR)
  - Render in the server, then send to the client, better performance
  - Good for Search Engine Optimization (SEO)
  - Faster initial page load
- Responsive design
  - Works on all screen sizes
  - Mobile-friendly
  - Consistent experience
- View transitions
  - Smooth page transitions
  - Modern feel

2.2.3. Development Tools

- VSCode[11]
  - Popular code editor
  - Great TypeScript support
  - Helpful extensions
- Chrome DevTools[12]
  - Debug web applications
  - Check performance
  - Test responsive design
- ESLint[13]
  - Finds code problems
  - Enforces coding style
  - Helps maintain code quality
- Prettier[14]
  - Automatically formats code
  - Consistent code style
  - Saves time on formatting

2.3. Backend Development

2.3.1. Core Technologies

- NestJS[15]
  - A framework for building server applications
  - Uses TypeScript

    - Quick and robust
- TypeScript
    - Type safety
    - Better code organization
- PostgreSQL[16]
    - Powerful database
    - Reliable and fast
    - Support spatial data (PostGIS[17])
- Redis[18]
    - Fast in-memory database

2.3.1.1. Cloud Services

- Cloudflare[19]
    - Protects against attacks
    - Speeds up content delivery (Content Delivery Network, CDN)
    - Manages DNS
- AWS[20]
    - Stores media files
    - Provides PostgreSQL as a service
- Google Cloud[21]
    - Provides geospatial services
    - Handles location data
- Hedera Hashgraph[22]
    - Blockchain platform

2.3.2. Key Features

- RESTful API
    - Standard way to communicate
- WebSocket support
    - Real-time communication
    - Good for chat and notifications (actively send messages)
- Optimized database design
    - Fast queries
    - Better performance using indexed and cached data

2.3.3. Development Tools

- VSCode
- Postman[23]
    - Tests API endpoints
    - Good for debugging
- pgAdmin[24]
    - Manages PostgreSQL database
    - Visual interface
    - Export database schema diagram

2.4. Development Infrastructure

2.4.1. Version Control

- Git[25]
  - Tracks code changes
  - Helps with collaboration
  - Keeps history of changes
- GitHub[26]
  - Hosts Git repositories
  - Good for team work
  - Many helpful features

2.4.2. CI/CD Pipeline

- GitHub Actions
  - Automates testing and deployment
  - Runs on code changes on main branch
- Automated testing
  - Checks code quality
- Deployment automation
  - Deploys code automatically (frontend and backend)
- Environment management
  - Handles different environments (development, production)
  - Keeps settings organized
- Vercel[27]
  - Deploys frontend
  - Good performance
  - Free plan
- Heroku[28]
  - Deploys backend
  - Free with limited resources

2.4.3. Monitoring and Analytics

- Performance monitoring
  - Tracks app speed
  - Finds slow parts
  - Helps improve performance
- Error tracking
  - Finds and reports errors
  - Helps fix problems
  - Improves reliability
- Usage analytics
  - Shows how users use the app
  - Helps make improvements
  - Guides development

- Security auditing
  - Checks for security problems
  - Helps keep data safe
  - Important for user trust

2.4.4. Codegen

The project implements a code generation workflow to ensure type safety and consistency between the backend and frontend. This process involves several key components:

1. OpenAPI Documentation Generation
   - NestJS backend automatically generates OpenAPI (Swagger) documentation
   - API endpoints are documented with TypeScript decorators
   - Request/response schemas are automatically inferred
   - Documentation is available at `/api/docs` endpoint, a json file is generated for client code generation
2. Client Code Generation
   - OpenAPI specification is used to generate TypeScript client code, using @hey-api/openapi-ts[29]
   - Generated code includes:
     - Type-safe API client
     - Request/response interfaces
     - Validation schemas
     - Error handling types
3. Nuxt Integration
   - Generated client code is automatically injected into Nuxt
   - API client is available through Nuxt's global context (automatically injected)
   - TypeScript support for full type safety
   - Automatic error handling and response typing

This approach provides several benefits:

- Ensures API contract consistency
- Reduces manual coding errors
- Improves development efficiency
- Enables better IDE support (type safety)

3. System Design

3.1. Overall Architecture

3.1.1. Core Technologies

- Backend Framework: NestJS
- Database: PostgreSQL with PostGIS
- ORM: TypeORM[30]
- Authentication: JWT + Passport
- Blockchain: Hedera Hashgraph
- File Storage: AWS S3[31]

3.1.2. System Architecture

- Modular Design
- Dependency Injection
- Middleware Pipeline
- Security Architecture
- Deployment Architecture

3.2. Module Breakdown

3.2.1. Core Modules

- AppModule: Application entry point
- AuthModule: Authentication and authorization
- UserModule: User management
- RoleModule: Role-based access control

3.2.2. Geographic Modules

- GeoObjectModule: Geographic object management, provides a base Create, Read, Update, Delete (CRUD) interface for all geographic objects
- GeoImageModule: Geographic image management, provides a extension function of GeoObjectModule for image objects
- CloudAnchorModule: AR cloud anchor management, including Cloud Anchor creation, expire time (ET/TTL) management, state tracking, etc.

3.2.3. Content Management

- SceneModule: Scene management, manages a group of GeoObjects as a scene
- StoryboardModule: Storyboard management
- LabelModule: Content labeling, provides the functionality to add labels to scenes or query scenes by labels

3.2.4. Consensus Modules

- ConsensusModule: Distributed consensus, provides the basic functionality of Hedera Hashgraph Blockchain manipulations
- ImageCopyrightModule: Copyright management, based on the consensus module, provides the functionality to create or query image copyright information

3.2.5. Utility Modules

- FileModule: File management, a adapter-pattern-designed module that provides a unified interface for file operations, currently supports local file system and AWS S3
- StatisticsModule: Analytics, provides the functionality to collect and analyze system data
- ZkModule: Zero-knowledge proofs, provides a interface of an external zero knowledge proof service

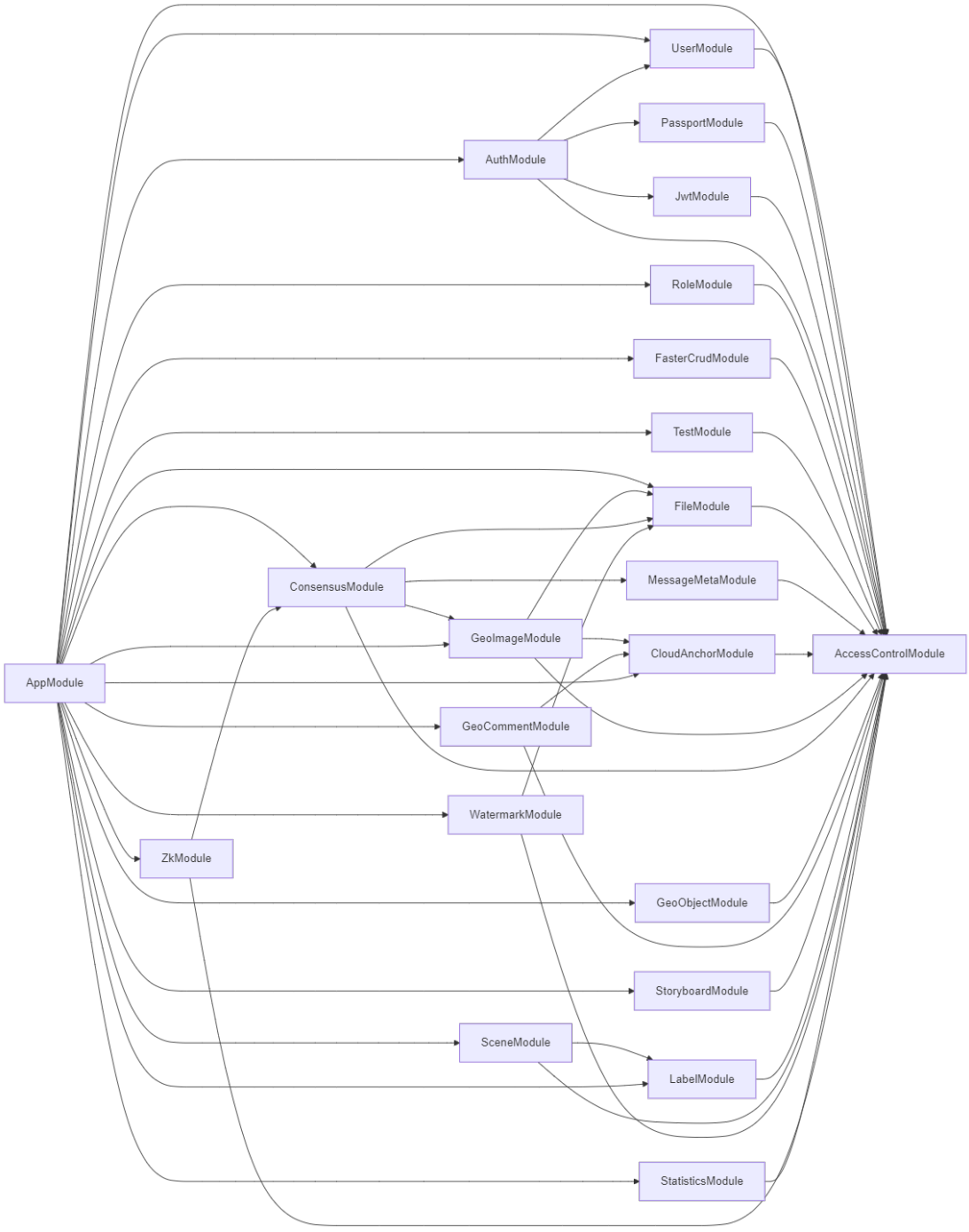


Figure 4: System architecture[1]

[1] https://github.com/bkmashiro/fyp-report/blob/main/images/diagrams/modules.svg

3.3. Database Design

3.3.1. Database schema

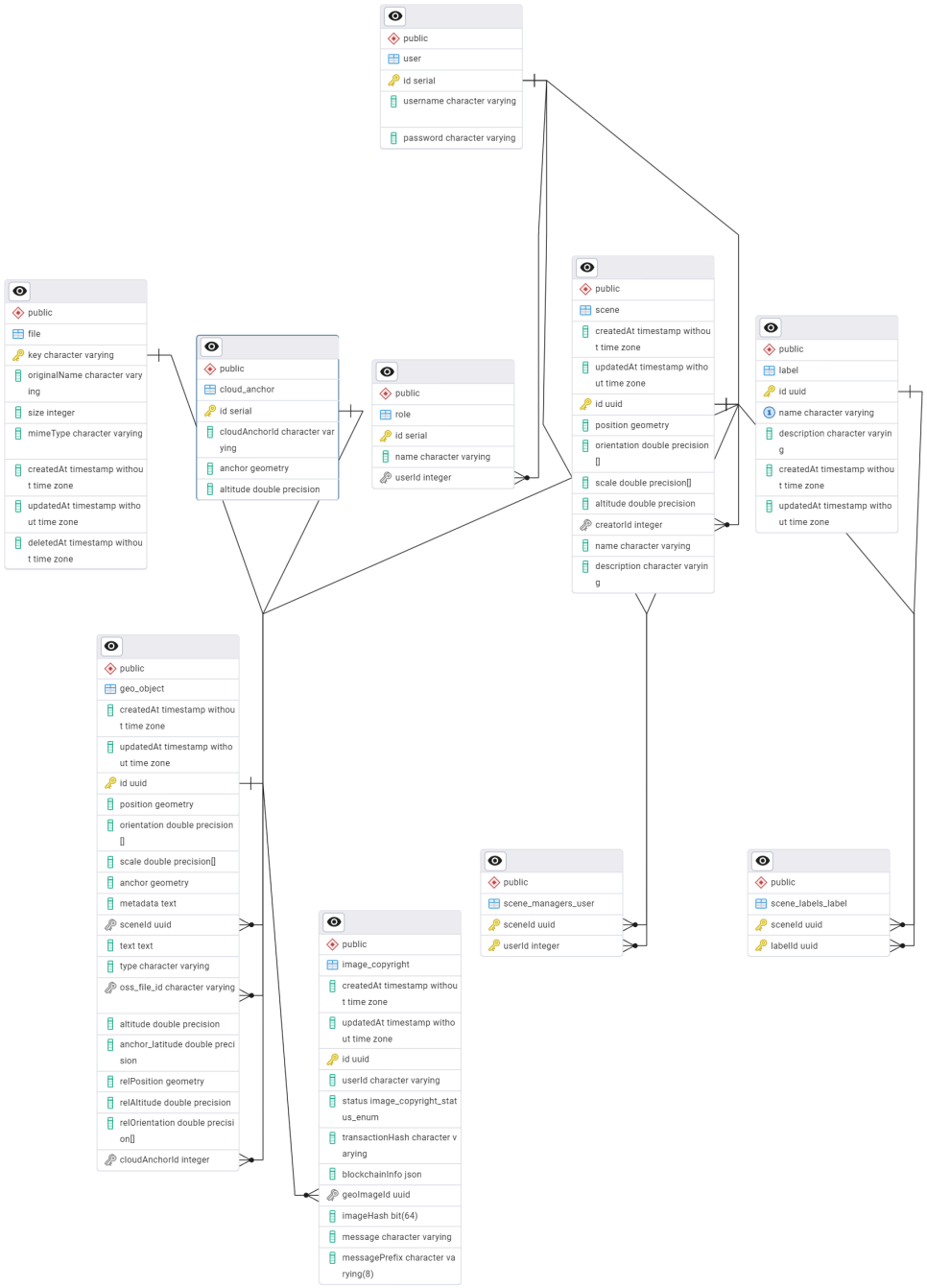


Figure 5: Database schema

3.3.2. Core Tables

3.3.2.1. Cloud Anchor Table

```sql
CREATE TABLE IF NOT EXISTS public.cloud_anchor
(
    id serial NOT NULL,
    "cloudAnchorId" character varying COLLATE pg_catalog."default" NOT NULL,
    anchor geometry NOT NULL,
    altitude double precision NOT NULL DEFAULT '0'::double precision,
    CONSTRAINT "PK_d02bf38fbcf9b99243290c6705f" PRIMARY KEY (id)
);
```

3.3.2.2. Geo Object Table

```sql
CREATE TABLE IF NOT EXISTS public.geo_object
(
    "createdAt" timestamp without time zone NOT NULL DEFAULT now(),
    "updatedAt" timestamp without time zone NOT NULL DEFAULT now(),
    id uuid NOT NULL DEFAULT uuid_generate_v4(),
    "position" geometry NOT NULL,
    orientation double precision[] NOT NULL DEFAULT '{0,0,0,1}'::double precision[],
    scale double precision[] NOT NULL DEFAULT '{1,1,1}'::double precision[],
    anchor geometry NOT NULL DEFAULT st_geomfromtext('POINT(0 0)'::text, 4326),
    metadata text COLLATE pg_catalog."default",
    "sceneId" uuid,
    text text COLLATE pg_catalog."default",
    type character varying COLLATE pg_catalog."default" NOT NULL,
    oss_file_id character varying COLLATE pg_catalog."default",
    altitude double precision NOT NULL DEFAULT '0'::double precision,
    anchor_latitude double precision NOT NULL DEFAULT '0'::double precision,
    "relPosition" geometry NOT NULL,
    "relAltitude" double precision NOT NULL DEFAULT '0'::double precision,
    "relOrientation" double precision[] NOT NULL DEFAULT '{0,0,0,1}'::double precision[],
    "cloudAnchorId" integer,
    CONSTRAINT "PK_8e888d4c5652cc76865762e44fb" PRIMARY KEY (id),
    CONSTRAINT "UQ_e1f3af9fe8ca8527f2cd3b04f78" UNIQUE (oss_file_id)
);
```

3.3.2.3. Scene Table

```sql
CREATE TABLE IF NOT EXISTS public.scene
(
```

```
    "createdAt" timestamp without time zone NOT NULL DEFAULT now(),
    "updatedAt" timestamp without time zone NOT NULL DEFAULT now(),
    id uuid NOT NULL DEFAULT uuid_generate_v4(),
    "position" geometry NOT NULL,
    orientation double precision[] NOT NULL DEFAULT '{0,0,0,1}'::double precision[],
    scale double precision[] NOT NULL DEFAULT '{1,1,1}'::double precision[],
    altitude double precision NOT NULL DEFAULT '0'::double precision,
    "creatorId" integer,
    name character varying COLLATE pg_catalog."default" NOT NULL DEFAULT 'untitled'::character varying,
    description character varying COLLATE pg_catalog."default" NOT NULL DEFAULT 'No description provided'::character varying,
    CONSTRAINT "PK_680b182e0d3bd68553f944295f4" PRIMARY KEY (id)
);
```

3.3.2.4. User Table

```sql
CREATE TABLE IF NOT EXISTS public."user"
(
    id serial NOT NULL,
    username character varying COLLATE pg_catalog."default" NOT NULL,
    password character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "PK_cace4a159ff9f2512dd42373760" PRIMARY KEY (id)
);
```

3.3.2.5. Role Table

```sql
CREATE TABLE IF NOT EXISTS public.role
(
    id serial NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    "userId" integer,
    CONSTRAINT "PK_b36bcfe02fc8de3c57a8b2391c2" PRIMARY KEY (id)
);
```

3.3.2.6. Scene Managers User Table

```sql
CREATE TABLE IF NOT EXISTS public.scene_managers_user
(
    "sceneId" uuid NOT NULL,
    "userId" integer NOT NULL,
    CONSTRAINT "PK_4f7ad96966b4111e05841edf938" PRIMARY KEY ("sceneId", "userId")
);
```

3.3.2.7. Scene Labels Label Table

```sql
CREATE TABLE IF NOT EXISTS public.scene_labels_label
```

```
2  (
3      "sceneId" uuid NOT NULL,
4      "labelId" uuid NOT NULL,
5      CONSTRAINT "PK_df95847ac1415d50fd81b90b568" PRIMARY KEY ("sceneId",
       "labelId")
6  );
```

3.3.2.8. Label Table

```
1  CREATE TABLE IF NOT EXISTS public.label                    SQL
2  (
3      id uuid NOT NULL DEFAULT uuid_generate_v4(),
4      name character varying COLLATE pg_catalog."default" NOT NULL,
5      description character varying COLLATE pg_catalog."default",
6      "createdAt" timestamp without time zone NOT NULL DEFAULT now(),
7      "updatedAt" timestamp without time zone NOT NULL DEFAULT now(),
8      CONSTRAINT "PK_5692ac5348861d3776eb5843672" PRIMARY KEY (id),
9      CONSTRAINT "UQ_972f95f212512a35e838562ea30" UNIQUE (name)
10 );
```

3.3.2.9. File Table

```
1  CREATE TABLE IF NOT EXISTS public.file                     SQL
2  (
3      key character varying COLLATE pg_catalog."default" NOT NULL,
4      "originalName" character varying COLLATE pg_catalog."default" NOT
       NULL,
5      size integer NOT NULL,
6      "mimeType" character varying COLLATE pg_catalog."default" NOT NULL,
7      "createdAt" timestamp without time zone NOT NULL DEFAULT now(),
8      "updatedAt" timestamp without time zone NOT NULL DEFAULT now(),
9      "deletedAt" timestamp without time zone,
10     CONSTRAINT "PK_e4a453ce0a609a5f94c66afb6ca" PRIMARY KEY (key)
11 );
```

3.3.2.10. Image Copyright Table

```
1  CREATE TABLE IF NOT EXISTS public.image_copyright          SQL
2  (
3      "createdAt" timestamp without time zone NOT NULL DEFAULT now(),
4      "updatedAt" timestamp without time zone NOT NULL DEFAULT now(),
5      id uuid NOT NULL DEFAULT uuid_generate_v4(),
6      "userId" character varying COLLATE pg_catalog."default" NOT NULL,
7      status image_copyright_status_enum NOT NULL DEFAULT
       'pending'::image_copyright_status_enum,
8      "transactionHash" character varying COLLATE pg_catalog."default",
9      "blockchainInfo" json,
10     "geoImageId" uuid,
11     "imageHash" bit(64) NOT NULL,
```

```
    message character varying COLLATE pg_catalog."default",
    "messagePrefix" character varying(8) COLLATE pg_catalog."default",
    CONSTRAINT "PK_d10f2e5d0f87bad5df843dee258" PRIMARY KEY (id)
);
```

3.3.3. Foreign Key Constraints

SQL

```
ALTER TABLE IF EXISTS public.geo_object
    ADD CONSTRAINT "FK_0543e1e184417cc6845d1a4c8fe" FOREIGN KEY
("cloudAnchorId")
    REFERENCES public.cloud_anchor (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.geo_object
    ADD CONSTRAINT "FK_3fbb7d2d54ddf4167f623c5b428" FOREIGN KEY
("sceneId")
    REFERENCES public.scene (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.geo_object
    ADD CONSTRAINT "FK_e1f3af9fe8ca8527f2cd3b04f78" FOREIGN KEY
(oss_file_id)
    REFERENCES public.file (key) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.scene
    ADD CONSTRAINT "FK_6f050bf6fb5415311d0c7e888cb" FOREIGN KEY
("creatorId")
    REFERENCES public."user" (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.role
    ADD CONSTRAINT "FK_3e02d32dd4707c91433de0390ea" FOREIGN KEY
("userId")
    REFERENCES public."user" (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.scene_managers_user
    ADD CONSTRAINT "FK_325064f9bfbbbedaa975cb054ba" FOREIGN KEY
("sceneId")
    REFERENCES public.scene (id) MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE;
```

```
36
37  ALTER TABLE IF EXISTS public.scene_managers_user
38      ADD CONSTRAINT "FK_4da77eb5c0f9a6c7f78cd73191f" FOREIGN KEY
    ("userId")
39      REFERENCES public."user" (id) MATCH SIMPLE
40      ON UPDATE CASCADE
41      ON DELETE CASCADE;
42
43  ALTER TABLE IF EXISTS public.scene_labels_label
44      ADD CONSTRAINT "FK_18597e747df9f6b6ef788e335fa" FOREIGN KEY
    ("labelId")
45      REFERENCES public.label (id) MATCH SIMPLE
46      ON UPDATE NO ACTION
47      ON DELETE NO ACTION;
48
49  ALTER TABLE IF EXISTS public.scene_labels_label
50      ADD CONSTRAINT "FK_b78bf773912eb343d75fbed6c0e" FOREIGN KEY
    ("sceneId")
51      REFERENCES public.scene (id) MATCH SIMPLE
52      ON UPDATE CASCADE
53      ON DELETE CASCADE;
54
55  ALTER TABLE IF EXISTS public.image_copyright
56      ADD CONSTRAINT "FK_90d5d6b183e25fed552dbab4c2e" FOREIGN KEY
    ("geoImageId")
57      REFERENCES public.geo_object (id) MATCH SIMPLE
58      ON UPDATE NO ACTION
59      ON DELETE NO ACTION;
```

3.3.4. Indexes

```
1   CREATE INDEX IF NOT EXISTS "UQ_e1f3af9fe8ca8527f2cd3b04f78"                SQL
2       ON public.geo_object(oss_file_id);
3
4   CREATE INDEX IF NOT EXISTS "IDX_325064f9bfbbbedaa975cb054b"
5       ON public.scene_managers_user("sceneId");
6
7   CREATE INDEX IF NOT EXISTS "IDX_4da77eb5c0f9a6c7f78cd73191"
8       ON public.scene_managers_user("userId");
9
10  CREATE INDEX IF NOT EXISTS "IDX_18597e747df9f6b6ef788e335f"
11      ON public.scene_labels_label("labelId");
12
13  CREATE INDEX IF NOT EXISTS "IDX_b78bf773912eb343d75fbed6c0"
14      ON public.scene_labels_label("sceneId");
```

3.3.5. Database Features

- PostGIS for geospatial data
- Spatial indexing
- Data integrity constraints
- Performance optimization
- Regular maintenance

3.3.5.1. Spatial Data Types

- Point (geometry, SRID: 4326)
- LineString (geometry, SRID: 4326)
- Polygon (geometry, SRID: 4326)
- MultiPoint (geometry, SRID: 4326)
- MultiLineString (geometry, SRID: 4326)
- MultiPolygon (geometry, SRID: 4326)

3.3.5.2. Spatial Query

PostgreSQL with PostGIS provides several powerful functions for spatial queries, particularly for point-based searches. Here are some commonly used functions with examples:

3.3.6. Point-in-Rectangle Search

The `ST_Contains` function checks if a geometry is completely contained within another geometry. For rectangular searches:

```sql
-- Find all points within a rectangular area
SELECT *
FROM geo_object
WHERE ST_Contains(
    ST_MakeEnvelope(
        min_lon, min_lat,  -- bottom-left corner
        max_lon, max_lat,  -- top-right corner
        4326               -- SRID for WGS84
    ),
    position
);
```

3.3.7. Radius Search

The `ST_DWithin` function finds all geometries within a specified distance of another geometry:

```sql
-- Find all points within 1000 meters of a given point
SELECT *
FROM geo_object
WHERE ST_DWithin(
    position::geography,  -- Cast to geography for meter-based distance
    ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geography,
    1000  -- Distance in meters
);
```

3.3.8. K-Nearest Neighbors

The `ST_Distance` function combined with `ORDER BY` and `LIMIT` can find the k-nearest points:

```sql
-- Find 10 nearest points to a given location
SELECT *
FROM geo_object
ORDER BY ST_Distance(
    position::geography,
    ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geography
)
LIMIT 10;
```

3.3.9. Spatial Indexing

For better performance, spatial indexes should be created:

```sql
-- Create a spatial index on the position column
CREATE INDEX idx_geo_object_position ON geo_object USING GIST (position);

-- Create a spatial index on the anchor column
CREATE INDEX idx_geo_object_anchor ON geo_object USING GIST (anchor);
```

These spatial queries are essential for location-based features in the application, such as finding nearby AR content or filtering scenes by geographic area.

3.4. Key Technologies and Implementation

3.5. AR Object

AR Object is a 3D object that can be placed in the real world. It can be a photo, a video, a 3D model, or a interactive object, etc.

In the system database design, AR Object table is a STI (Single Table Inheritance) table, which means it has only one column for the object type, and the rest of the columns are inherited from the superclass.

The following shows the base schema of the AR Object table:

Field	Type	Description
id	uuid	Object identifier
type	varchar	Object type identifier
anchor	Point (geometry, SRID: 4326)	Geographic anchor position
anchor_latitude	float (default: 0)	Anchor latitude
metadata	text (nullable)	Metadata information
cloudAnchor	CloudAnchor (many-to-one)	Cloud anchor association
relPosition	Point (geometry, SRID: 4326)	Relative position
relAltitude	float (default: 0)	Relative altitude
relOrientation	float[] (default: [0, 0, 0, 1])	Relative orientation
createdAt	Date (auto-generated)	Creation time

updatedAt	Date (auto-updated)	Update time
scene	Scene (many-to-one)	Scene association

This is the inheritance schema of the subclass `SpatialImage` table:

Field	Type	Description
type	constant(“SpatialImage”)	Object type identifier
ossFile	File (one-to-one)	OSS file

This is the inheritance schema of the subclass `SpatialComment` table:

Field	Type	Description
type	constant(“SpatialComment”)	Object type identifier
content	text	Comment content

`spatial_image` and `spatial_comment` are the only two subclasses of `spatial_object`, they’re being stored in the `spatial_object` table, and the `type` field is used to identify the object type. This is used to simplify the database design of derived classes.[2]

3.6. Cloud Anchor

In AR, an anchor is a fixed point in space used to keep virtual objects in a consistent location in the real world. A Cloud Anchor is an anchor that is hosted in the cloud, making it accessible across devices.

A Google Cloud Anchor is a feature provided by ARCore, Google’s platform for building AR experiences. Cloud Anchors allow AR content to be shared across devices by enabling users to place and anchor virtual objects in the same real-world location, even on different devices or at different times. [32]

A Cloud Anchor captures **local environmental features** (such as depth and color information) to mark an absolute position in space. Unlike GPS-based positioning, it does not rely on latitude and longitude coordinates. Instead, it uses the unique visual characteristics of the surrounding environment to establish precise spatial references. This approach enables **highly accurate positioning** with an error margin of less than 10 centimeters.

The process works by:

1. **Scanning** the immediate environment to extract distinctive visual features
2. Creating **a unique fingerprint** of the local space using these features
3. **Storing** this fingerprint in the cloud along with the anchor’s position
4. Allowing other devices to **match** these features to locate the same position

[2] Note that when type number is large, a STI is not a best practice.

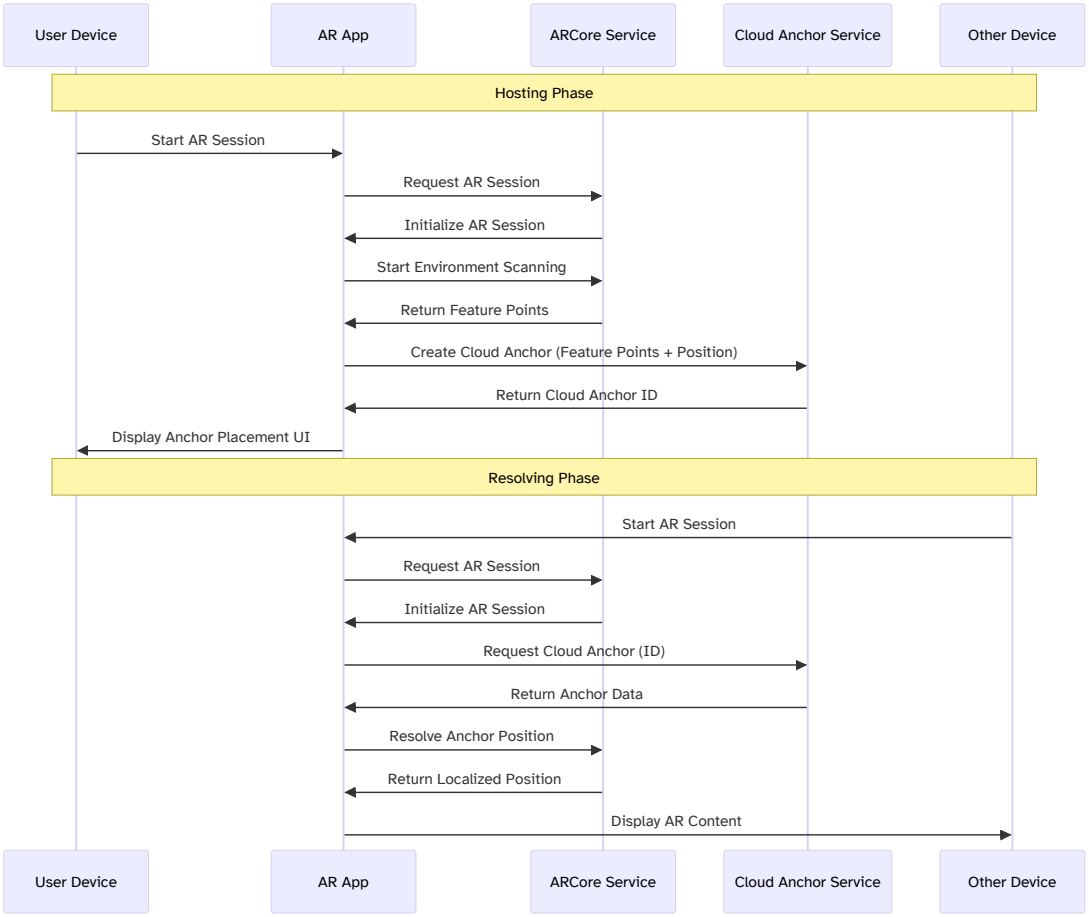


Figure 6: A device hosting an anchor

This feature-based approach offers several advantages:

- Works in GPS-denied environments (indoors, urban canyons)
- Provides sub-meter accuracy
- Enables persistent AR experiences across multiple sessions
- Supports multi-user experiences in the same physical space

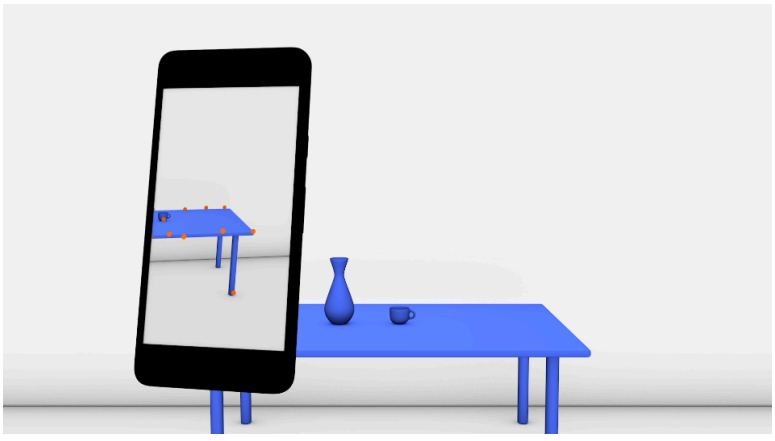


Figure 7: A device hosting an anchor

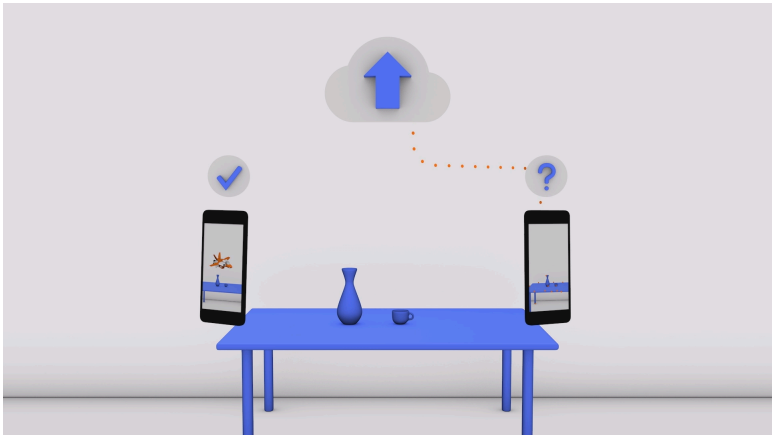


Figure 8: Another device locating the same anchor

3.7. Positioning

The system's prior positioning method is **relative positioning**, which is a more accurate and reliable approach for indoor AR applications, when it's unavailable, the system will fall back to absolute positioning using Global Positioning System (GPS).

3.7.1. Scene Graph

In the application, virtual objects are organized in a **hierarchical structure** known as a **scene graph**. This tree-like organization provides several key benefits:

1. **Hierarchical Organization**
   - Each object maintains its position relative to its parent
   - Objects can be grouped together by making them children of a common parent
   - When a parent moves, all its children maintain their relative positions
   - Moving a group only cause one edit to the root element of the group
2. **Root Anchor Requirement**
   - The root of the scene graph must be an Anchor
   - This establishes the connection between virtual content and the real world
   - It's children objects are positioned relative to this root anchor
3. **Grouping Benefits**
   - Related objects can be moved together as a unit
   - Complex arrangements can be created and manipulated easily
   - Hierarchical organization simplifies scene management

3.7.2. Relative Positioning

Relative positioning is the preferred approach in this project for several key reasons:

1. **Precision in Indoor Environments**
   - Achieves decimeter-level accuracy indoors
   - Significantly outperforms GPS-based positioning which can have errors of tens of meters
   - Particularly important for AR applications requiring precise spatial alignment
2. **Enhanced Sharing Capabilities**
   - As discussed in the Cloud Anchor section, enables seamless sharing of AR experiences
   - Multiple users can interact with the same virtual content in the same physical space
   - Maintains spatial consistency across different devices and sessions
3. **Coordinate System Design**
   - Uses Cloud Anchor as the origin point (0,0,0) of the coordinate system
   - Establishes a local reference frame that's independent of global coordinates
   - Enables precise relative measurements between virtual objects
4. **Implementation Benefits**
   - Reduces dependency on external positioning systems
   - Provides more stable and reliable positioning in indoor environments
   - Supports complex multi-user interactions with minimal latency

This approach aligns perfectly with the project's requirements for high-precision indoor AR experiences and collaborative features.

In 3D computer graphics, a rotation and position of an object is represented by a 4×4 matrix:

$$\begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & \boldsymbol{1} \end{pmatrix}$$

where $\boldsymbol{R}$ is a 3×3 rotation matrix, $\boldsymbol{t}$ is a 3×1 translation vector, and $\boldsymbol{0}^T$ is a 1×3 zero vector.

We want a relative transform:

$$\boldsymbol{T}_{\text{relative}} = \boldsymbol{T}_{\text{origin}}^{-1} \cdot \boldsymbol{T}_{\text{target}} = \begin{pmatrix} \boldsymbol{R}_{\text{origin}}^T & -\boldsymbol{R}_{\text{origin}}^T \cdot \boldsymbol{t}_{\text{origin}} \\ \boldsymbol{0}^T & \boldsymbol{1} \end{pmatrix} \cdot \boldsymbol{T}_{\text{target}}$$

That is:

$$\textbf{Position}_{\text{relative}} = \boldsymbol{R}_{\text{origin}}^T \cdot \left(\boldsymbol{t}_{\text{target}} - \boldsymbol{t}_{\text{origin}} \right)$$

$$\textbf{Rotation}_{\text{relative}} = \boldsymbol{R}_{\text{origin}}^T \cdot \boldsymbol{R}_{\text{target}}$$

C# code to convert a target transform to a local transform relative to an origin transform:

```
1 public (Vector3, Quaternion) ConvertToLocalTransform(Transform
  target, Transform origin)                                    C#
2   {
```

```
3       var localPos = Quaternion.Inverse(origin.rotation) * (target.position
        - origin.position);
4       var localRot = Quaternion.Inverse(origin.rotation) * target.rotation;
5
6       return (localPos, localRot);
7   }
```

3.7.3. Absolute Positioning Fallback

While the relative positioning system provides excellent indoor performance, we also implement an absolute positioning fallback using GPS coordinates. This system uses latitude, longitude, and altitude to determine an object's position in the real world.

However, GPS-based positioning has several limitations:

- Varying accuracy across different devices (typically 4.9 meters[33])
- Signal interference from buildings, trees, and other obstacles
- Multi-path effects in indoor environments causing signal reflections [34]
- Atmospheric conditions affecting signal propagation
- Limited vertical accuracy for altitude measurements

These factors make GPS positioning **less reliable** for precise indoor AR applications, especially in urban environments or buildings with dense structures. The system always tries to use relative positioning first, and then falls back to absolute positioning if the relative positioning is not available. If both are not available, the system will use inertial navigation system, all features are unavailable at this point.

3.8. Scene Management

A **Scene** is a collection of AR objects (Section 3.5), which provides access control and scene management.

When a user loads a scene, the system will retrieve all the cloud anchors that related to the scene, and ask the user to scan the environment to find at least one anchor. Once the anchor is found, the system will use the anchor to position the AR objects.

3.9. Invisible Watermarking

This feature has a demo at https://github.com/bkmashiro/watermark

The invisible watermarking system implements a robust digital watermarking algorithm that embeds information into images **without visible artifacts** (i.e. the watermark is almost invisible to human eyes). The process consists of two main phases: **embedding** and **extraction**.

3.9.1. Embedding Process

The embedding process takes an input image and watermark information (which can be an image, string, or bitstream[3]) and follows these steps:

1. Color Space Conversion
   - Convert RGB to YUV color space
   - Only modify the Y (luminance) channel as human eyes are more sensitive to brightness changes

[3] Eventually they are processed as bitstream

2. Multi-resolution Analysis
    - Apply Discrete Wavelet Transform (DWT)[4] to each channel
    - Extract low-frequency (ca) and high-frequency (hvd) components
    - Focus on low-frequency components for better stability against common image processing operations

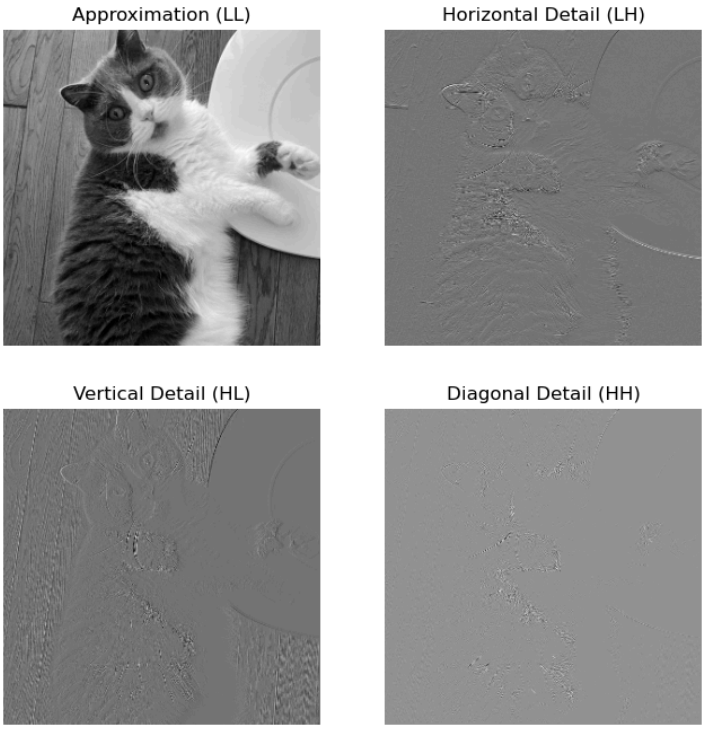


Figure 9: DWT decomposition

3. Block-based Processing
    - Divide the low-frequency component (LL) into 4×4 pixel blocks
    - Apply Discrete Cosine Transform (DCT)[5] to each block
    - Most image information concentrates in the top-left corner
4. Security and Embedding
    - Generate pseudo-random sequence using password_img
    - Shuffle DCT blocks using the sequence for security
    - Apply Singular Value Decomposition (SVD)[6] to each block
    - Modify singular values (s[0], s[1]) to embed 1-bit information
5. Reconstruction
    - Apply inverse SVD
    - Apply inverse DCT
    - Merge blocks
    - Apply inverse DWT
    - Convert back to RGB color space

[4] DWT decomposes an image into different frequency bands, providing multi-resolution analysis

[5] DCT transforms a image from spatial domain to frequency domain, which focus the energy in the top-left corner, if we modify the coefficients in frequencies that not sensitive to human eyes, the image quality is not affected

[6] SVD decomposes a matrix into three matrices(U, Sigma, V), which is a powerful tool for image processing, the Sigma matrix is a diagonal matrix, that is very robust to noise and modify Sigma affects little to the image quality

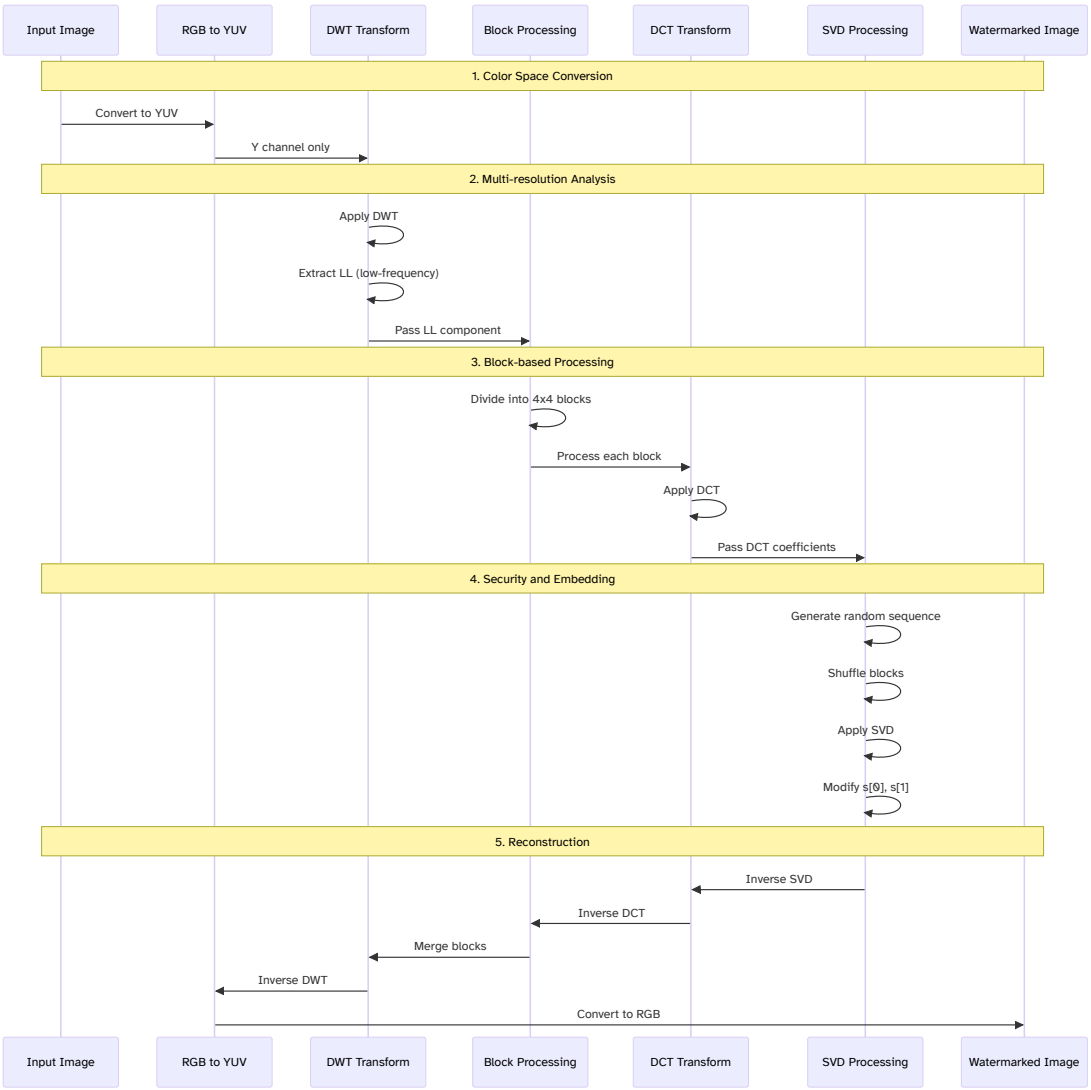
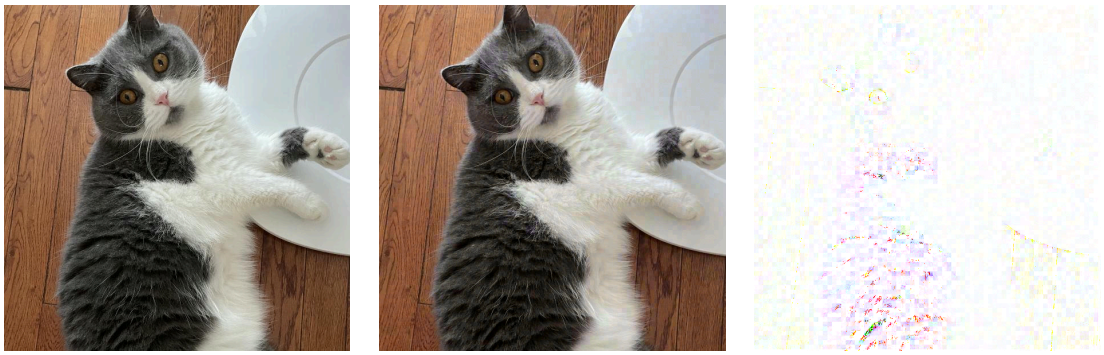


Figure 10: Embedding process



Original Embedded Residual

Figure 11: Embedding has no visible effect (hidden information: `ilovecat`)

3.9.2. Extraction Process

The extraction process can recover the watermark without requiring the original image:

1. Preprocessing
   - Apply DWT and DCT to the input image
   - Use the same password_img to generate the shuffling sequence
2. Information Recovery
   - Apply SVD to each block
   - Extract singular values s[0] and s[1]
   - Decode the embedded bit (0 or 1)
3. Optional Enhancement
   - If extract_with_kmeans is enabled, use 1D k-means clustering to automatically determine the extraction threshold
4. Final Output
   - Reconstruct the bitstream
   - Convert to image or string format

3.9.3. Technical Considerations

Technique	Purpose	Reason
DWT	Multi-resolution analysis	High-frequency components are unstable; embedding in low-frequency provides better stability
DCT	Energy concentration	Most image information is in the top-left corner; modifying coefficients minimally affects quality
SVD	Stability	Singular values are stable; modifications don't cause significant block distortion
Shuffling	Security	Prevents direct analysis of embedding regions

3.10. Blockchain-based Copyright Protection

3.10.1. On-chain Evidence

The blockchain-based copyright protection system combines perceptual hashing with author identity to create a tamper-proof record of image ownership. Here's how it works:

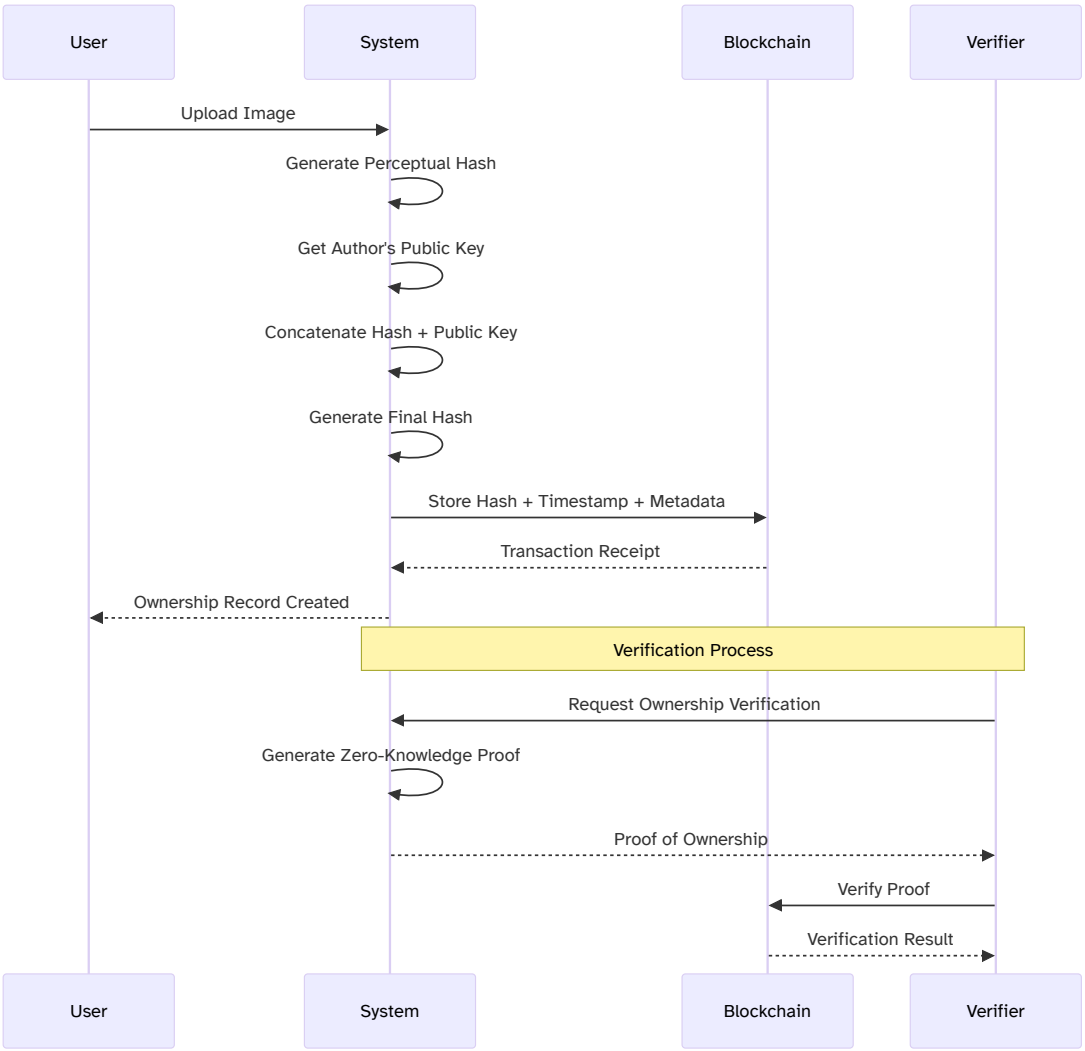


Figure 12: Blockchain-based copyright protection process

1. Image Fingerprinting
   - Generate perceptual hash of the image
   - Concatenate with author's public key
   - Hash the combined data to create a unique **identifier**
2. Blockchain Storage
   - Store the hash on the blockchain
   - Include timestamp and metadata
   - Create immutable ownership record
3. Verification (challenge) Process
   - Anyone can verify image ownership
   - Challenge the system with the combination of image hash and author's public key
   - The system will verify the ownership record on the blockchain and tell `true` or `false`

The system provides several key benefits:

1. Privacy Protection
   - Original image not stored on chain
   - Cannot directly infer the author's identity from the blockchain records
   - Only hash values are public
2. Tamper Resistance
   - Immutable ownership records
   - Timestamp verification
   - Cryptographic proof of authenticity
3. Efficient Verification
   - Cached quick ownership checks
4. Decentralized Storage
   - Ownership records stored on blockchain
   - Independent of backend infrastructure
   - Permanent and immutable records
   - Trustless verification possible

This approach ensures that while the ownership information is publicly verifiable, the actual content and author details remain private until explicitly shared.

3.10.2. Embedded Identifier Extraction Enhancement

If the image is **embedded** with the **identifier** (mentioned in Section 3.10.1, Image Fingerprinting, step 3)[7], the system will extract the identifier from the image.

Mentioned in the Section 3.9 section[8], the embedding identifier can be extracted from the image, even if the image is edited slightly (including compression, rotation, cropping, etc.).

Then the system will query the blockchain to find and verify the ownership record of the image.

3.11. Image Similarity Search

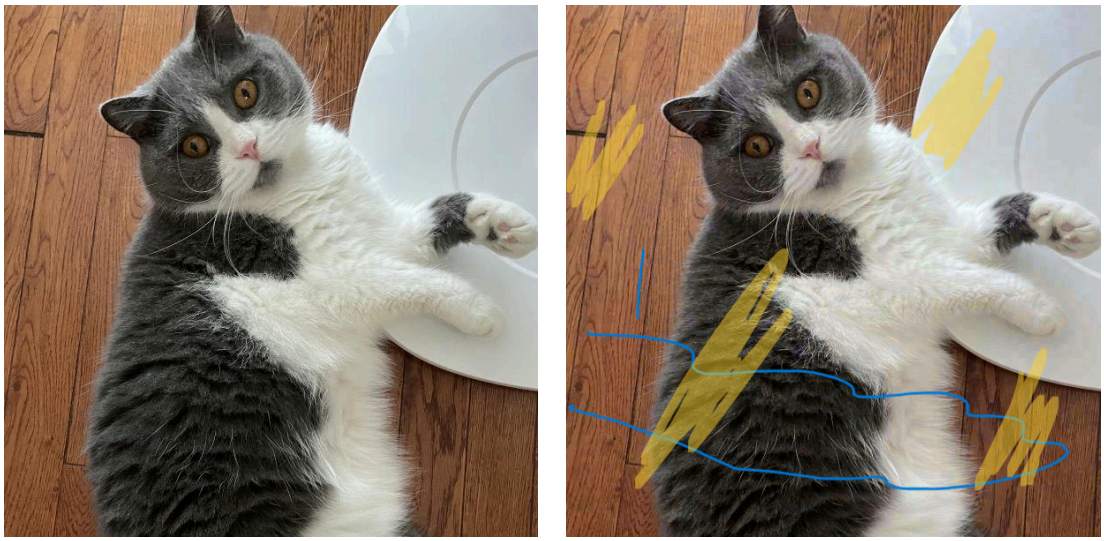
3.11.1. Perceptual Hashing

Perceptual hashing is a technique that hashes an image into a fixed-length fingerprint. It is a type of hash function that is designed to be robust to common image processing operations, such as compression, rotation, and scaling. [35]

Image perceptual hashing has been proposed to identify or authenticate image contents in a robust way against distortions caused by compression, noise, common signal processing and geometrical modifications, while still holding a good discriminability for different ones in sense of human perception. A method called "Block Mean Value Based Image Perceptual Hashing" is used to generate the hash value. [36]

[7] Due to space and query efficiency consideration, only the first 64 bits of the identifier are embedded.

[8] This is combining the Section 3.9 and the primitive way of Section 3.10.



p-hash: 070f0f0d1f05ce06 p-hash: 070f0f0d1f058c8e[9]

Figure 13: Similar images have similar perceptual hash values

3.11.2. Image Similarity Search

A 64-bit hash value is generated for each image, which is used to represent the image.

For two images, the Hamming distance between their hash values is calculated. The smaller the Hamming distance, the more similar the two images are.

$$\text{Hamming distance} = \sum(\boldsymbol{x}_i \oplus \boldsymbol{y}_i)$$

Here is the PostgreSQL query to calculate the Hamming distance of a given image hash with all the images in the database:

```
1 SELECT                                                    PostgreSQL
2   *,
3   (1 - (bit_count(imageCopyright.imageHash # B'${bitHash}')::float / 64))
    as similarity
4 FROM
5   image_copyright imageCopyright
6 WHERE
7   bit_count(imageCopyright.imageHash # B'${bitHash}') <= :maxDistance
```

A 84% threshold is used to determine if two images are similar. A 92% threshold is used to determine if a photo is a duplicate[10].

[9] Fun fact:The hidden information can still be extracted if the image is edited

[10] There are small differences between same-sourced photos due to compression and other factors, so a 8% tolerance is applied

3.12. Blockchain-based Zero-Knowledge Proof for Photo Ownership

> ⚠ **Warning**
>
> The following sections require advanced knowledge in:
> - Mathematics (Finite Fields, Elliptic Curves)
> - Cryptography (Zero-Knowledge Proofs, Hash Functions)
> - Blockchain (Hedera)
> - Circuit Design (R1CS, Groth16)

3.12.1. Introduction to Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs)[11] are cryptographic protocols that allow one party (the prover) to prove to another party (the verifier) that a statement is true without revealing any additional information[12]. In the mean time, the verifier, even after having become convinced of the statement's truth, should nonetheless remain unable to prove the statement to further third parties.

For example:

- Traditional way: To prove you own a photo, you need to show the photo
  - Or using RSA for example, the photo is signed with the owner's private key, so the verifier can verify the ownership by using the owner's public key. But here the photo is revealed, and the owner's identity is exposed.
- Zero-knowledge way: You only need to provide a proof that verifies your ownership, without revealing the photo content and the owner's identity. The verifier is convinced, but cannot prove the statement to further third parties.

Summary:

- Prover does not reveal any information
- Verifier is convinced of the statement's truth
- Verifier cannot prove the statement to further third parties

3.12.2. Mathematical Foundations

3.12.2.1. Finite Fields and Elliptic Curves

- Using BN128 elliptic curve: $E : y^2 = x^3 + 3$
- Defined over finite field F_p, where p is a large prime
- Points on the curve form a cyclic group G

3.12.2.2. Poseidon Hash Function

- Hash function based on permutation networks
- Input: two elements $(x, y) \in F_p^2$
- Output: one element $h \in F_p$
- Mathematical representation: $h = \text{Poseidon}(x, y)$

[11] A analogy story is shown in Section 9.1.1 to illustrate the concept of ZKPs

[12] That is to say, in the scenario of photo ownership, the prover can prove to the verifier that they own a photo, **without revealing the photo content or the owner's identity**, but in the system, it's simplified, that the owner's address is revealed (actively informed to the verifier). In theory this can be improved using randomized `artistId`

3.12.2.3. Groth16 Proof System

- Zero-knowledge proof system based on R1CS (Rank-1 Constraint System)
- Prover knows witness w satisfying $Aw \times Bw = Cw$
- Where A, B, C are constraint matrices, $\times$ denotes element-wise multiplication

3.12.3. Circuit Design

3.12.3.1. Ownership Proof Circuit

Circom is a language to compile circuits, which is a group of addition and multiplication gates.[37]

```circom
template Ownership() {
    signal input sigHash;      // Signature hash
    signal input artHash;      // Artwork hash
    signal output commitment;  // Commitment value

    component hash = Poseidon(2);
    hash.inputs[0] <== sigHash;
    hash.inputs[1] <== artHash;

    commitment <== hash.out;
}
```

3.12.3.2. Circuit Constraints

- Input constraints: $\text{sigHash}, \text{artHash} \in F_p$
- Output constraints: $\text{commitment} = \text{Poseidon}(\text{sigHash}, \text{artHash})$
- Witness[13]: private inputs, internal variables

3.12.3.3. Comparison with RSA

The zero-knowledge proof system and RSA (Rivest-Shamir-Adleman) are both cryptographic systems, but they serve different purposes and have distinct characteristics:

Aspect	Zero-Knowledge Proof	RSA
Purpose	Prove knowledge without revealing it	Encryption and digital signatures
Key Components	Prover, Verifier, Witness	Public key, Private key
Security Basis	Computational complexity of mathematical problems	Integer factorization problem
Privacy Level	Complete privacy (zero-knowledge)	Partial privacy (reveals some information)
Verification Process	Interactive or non-interactive proof verification	Direct signature verification
Use Case	Proving ownership without revealing content	Encrypting messages and signing documents

Key differences in implementation:

[13]The witness is a set of values that satisfies the circuit constraints. Only when a proper witness is provided, a proof can be generated

1. **Privacy Level**
   - ZKP: Proves knowledge without revealing any information
   - RSA: Reveals some information during verification
2. **Verification Process**
   - ZKP: Requires complex proof generation and verification
   - RSA: Simple signature verification process
3. **Use Cases**
   - ZKP: Ideal for privacy-preserving verification
   - RSA: Better suited for traditional encryption and signing
4. **Implementation Complexity**
   - ZKP: Requires specialized circuit design
   - RSA: Standardized implementation available

3.12.3.4. Note

In the context of “Copyright Protection”, it makes little sense to use ZKP to prove only the ownership of the artwork, without revealing the artwork content / owner’s identity.

The idea here is to reveal little information (user’s indenty hash or artwork signature) to the verifier, while ZK protects the full artwork content from being revealed.

3.12.3.4.1. Comparison with hash Only

Current Web3 platforms typically use direct hashing: storing artwork hash on-chain. This approach has privacy risks:

- Hash collisions (theoretical)
- Brute force attacks on small files
- Reverse matching for popular content

ZK-enhanced registration improves privacy by:

- Storing only hash and ZK proof on-chain
- Proving knowledge without revealing content
- Preventing content speculation
- Blocking reverse matching attempts
- Securing small/popular works

In short:

- Hash only: the third party cannot know the content, temporarily
- ZK-enhanced: without actively revealing the content, the third party cannot know anything about the content

3.12.4. System Architecture

3.12.4.1. Data Flow

1. Artwork Registration Process:
   - User uploads artwork file
   - System computes artwork hash (Keccak256)
   - Generates user key pair
   - Signs artwork hash with private key
   - Generates zero-knowledge proof
   - Stores proof and metadata on-chain

2. Ownership Verification Process:
    - User provides artwork hash and claimed owner address
    - System queries related records from chain
    - Verifies zero-knowledge proof
    - Checks address match
    - Returns verification result

3.12.5. Security Analysis

3.12.5.1. Zero-Knowledge Property

- Proof pi does not reveal $\mathrm{sigHash}$ and $\mathrm{artHash}$
- Verifier only knows commitment value $\mathrm{commitment}$
- Original data cannot be recovered from proof

3.12.5.2. Completeness

- Only correct $\mathrm{sigHash}$ and $\mathrm{artHash}$ can generate valid proof
- Forging proof is computationally infeasible
- Commitment value $\mathrm{commitment}$ is bound to specific input pair

3.12.5.3. Practicality

- Fixed proof size, independent of input size
- Constant verification time, suitable for blockchain environment
- Can prove artwork ownership without revealing content

3.12.6. Implementation Details

3.12.6.1. Key Management

- Generate-and-dispose by the backend system, only the user will process the private and public keys
- User can generate key pairs on device, no need to transfer keys to the backend, which is more secure

3.12.6.2. On-chain Storage

- Only proofs and hash values stored
- No original artwork data stored

3.12.7. Performance Optimization

3.12.7.1. Proof Generation

- Using Groth16 proof system for faster generation
- Fixed proof size suitable for on-chain storage
- Support for batch verification

3.12.7.2. On-chain Queries

- Query performance optimized with indexes
- Support for time-range queries
- Caching of common query results

3.13. Web Management Interface

3.13.1. Frontend Architecture

- Built with Nuxt.js 3 for optimal performance
- Server-side rendering (SSR) for fast initial load
- Automatic code splitting and lazy loading
- Progressive enhancement support

3.13.2. UI/UX Design

- Tailwind CSS for responsive design
- Animations between pages
- Mobile-first approach

3.13.3. Performance Optimization

- Image optimization with Nuxt Image
- Route-based code splitting
- Component lazy loading
- Caching strategies
- Bundle size optimization

3.13.4. State Management

- Pinia for state management
- Persistent storage (user sensitive data, e.g. private keys)
- Real-time updates
- Error handling

3.13.5. Development Tools

- TypeScript support
- ESLint configuration
- Prettier formatting

3.14. Authentication System

3.14.1. JWT Authentication

3.14.1.1. Token Structure

- Header: Algorithm and token type
- Payload: User data
- Signature: HMAC SHA256 verification

3.14.1.2. Security Features

- Token expiration (exp)
- Issuer verification (iss)
- Audience validation (aud)
- Subject identification (sub)

3.14.1.3. Implementation

- Backend generates JWT upon successful login
- Frontend stores token in local storage
- Token included in Authorization header
- Automatic token refresh mechanism

3.14.1.4. Token Revocation

- Implement Redis-based token blacklist
- Store revoked tokens with expiration time
- Check blacklist during token validation
- Automatic cleanup of expired blacklist entries

3.14.1.5. Redis Implementation

- Use Redis SET data structure for blacklist
- Key format: “blacklist:token”
- Value: timestamp of revocation
- TTL set to match token expiration

3.14.1.6. Security Considerations

- Prevent reuse of revoked tokens
- Maintain blacklist size through TTL

3.14.2. Access Control with Nest-Access-Control

3.14.2.1. Resource-based Access Control and Attribute-based Access Control

The system implements a hybrid access control model combining Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) to manage user permissions and resource access. This model consists of four key components:

1. Role Definition
   - Predefined roles with hierarchical inheritance
   - Each role has specific permissions
   - Roles can be assigned to users
2. Permission Management
   - Fine-grained permission definitions
   - Resource-based access control
   - Action-based permission checks
3. Access Control Implementation
   - Decorator-based permission guards
   - Controller-level access control
   - Resource ownership validation
4. Dynamic Access Control
   - Context-aware permission checks
   - Runtime permission evaluation
   - Custom permission logic support

3.14.2.2. Permission Definition

The system defines permissions through a structured approach:

1. Permission Enums
   - Create permission enums for different actions
   - Define resource-specific permissions
   - Map permissions to roles
2. Resource Actions (`nest-access-control`[38])
   - Read: View resource content

- Create: Add new resources
- Update: Modify existing resources
- Delete: Remove resources

3. Permission Guards
   - Implement permission validation
   - Check role-based access
   - Handle unauthorized access

3.14.2.3. Resource Protection

Resources are protected through multiple layers:

1. Guard Implementation
   - Use `@UseGuards` decorator
   - Apply permission checks
   - Validate resource ownership
2. Access Control
   - Controller-level protection
   - Method-level restrictions
   - Resource-specific rules
3. Error Handling
   - Unauthorized access responses
   - Permission denied messages
   - Access violation logging

3.14.2.4. Dynamic Access Control

The system supports dynamic access control through:

1. Context-based Permissions
   - User context evaluation
   - Resource context consideration
   - Environment-based rules
2. Runtime Permission Checks
   - Dynamic permission evaluation
   - Conditional access rules
   - Custom permission logic
3. Access Control Extensions
   - Custom permission handlers
   - Extended validation rules
   - Flexible permission models

3.15. Special Features

3.15.1. Interpage Animations

The system implements smooth page transitions using the View Transitions API[39], a new web standard that enables seamless animations between page states.

1. Implementation

```css
/* Enable view transitions */
@view-transition {
  navigation: auto;
}

/* Define transition styles */
.page-transition {
  view-transition-name: page;
}

/* Customize animation */
::view-transition-old(page) {
  animation: fade-out 0.3s ease-out;
}

::view-transition-new(page) {
  animation: fade-in 0.3s ease-in;
}

@keyframes fade-in {
  from { opacity: 0; }
  to { opacity: 1; }
}

@keyframes fade-out {
  from { opacity: 1; }
  to { opacity: 0; }
}
```

2. Benefits
   - Improved user experience
   - Consistent animations

4. Unimplemented or Future Features

4.1. Scripting

The system was planned to support Lua scripting for dynamic behavior adjustment in Unity, but this feature was not implemented due to the following reasons:

1. Implementation Complexity
   - Runtime behavior modification
2. Development Effort
   - Security considerations
   - Performance optimization
3. Alternative Solutions
   - Predefined behavior patterns
   - Configuration-based adjustments

This feature remains as a potential future enhancement for more flexible behavior customization.

4.2. Offline Mode

The offline mode feature was initially considered but ultimately not implemented due to the following reasons:

1. High Implementation Cost
   - Complex synchronization logic
   - Local data storage management
   - Conflict resolution mechanisms
2. Limited Business Value
   - Most operations require real-time data
   - Network availability is generally good
   - Minimal user demand for offline functionality
3. Technical Challenges
   - Data consistency maintenance

Given these factors, the development team decided to focus resources on core features that provide more immediate value to users.

5. Development Process

5.1. Methodology

The project adopted an Agile development methodology, specifically Scrum, to ensure efficient and flexible development. The team followed a structured approach with two-week sprint cycles, allowing for regular feedback and adaptation. Daily stand-up meetings were conducted to track progress and address any blockers, while sprint reviews and retrospectives provided opportunities for continuous improvement.

Development practices were centered around quality and collaboration. The team implemented test-driven development (TDD) to ensure code reliability, while continuous integration helped maintain a stable codebase. Code reviews were mandatory for all pull requests, and complex features often involved pair programming sessions to share knowledge and improve code quality.

Project management was streamlined using modern tools. GitHub Projects served as the primary task tracking system, while Jira was used for more detailed issue management. Regular stakeholder meetings ensured alignment with project goals, and comprehensive progress tracking helped maintain transparency throughout the development process.

5.2. Version Control

The project's version control strategy was built around Git and GitHub, providing a robust foundation for collaborative development. The branching strategy followed a modified Git Flow approach, with main and develop branches serving as the primary integration points. Feature branches were created for new development work, while release branches helped manage versioning and deployment cycles.

The development workflow emphasized quality and collaboration. Every code change required a pull request review, ensuring that multiple team members reviewed and approved changes. Automated testing and code quality checks were integrated into the workflow, while documentation updates were treated as an integral part of the development process.

6. Deployment

6.1. Mobile Application Deployment

The mobile application deployment process was streamlined using Unity's comprehensive build system. Unity Cloud Build was utilized to automate the build process, generating platform-specific builds for both iOS[14] and Android. This automation significantly reduced the time required for deployment while ensuring consistent build quality.

6.2. Web Deployment

The web application was deployed on Vercel, taking advantage of its modern deployment infrastructure. The platform's seamless GitHub integration enabled automatic deployments, with preview deployments for pull requests allowing for easy testing of new features. The edge network optimization provided by Vercel ensured fast content delivery worldwide.

Performance optimization was a key focus of the web deployment strategy. Automatic HTTPS implementation and global CDN distribution helped maintain security and speed. Serverless functions were utilized for dynamic content, while integrated analytics provided valuable insights into user behavior and application performance.

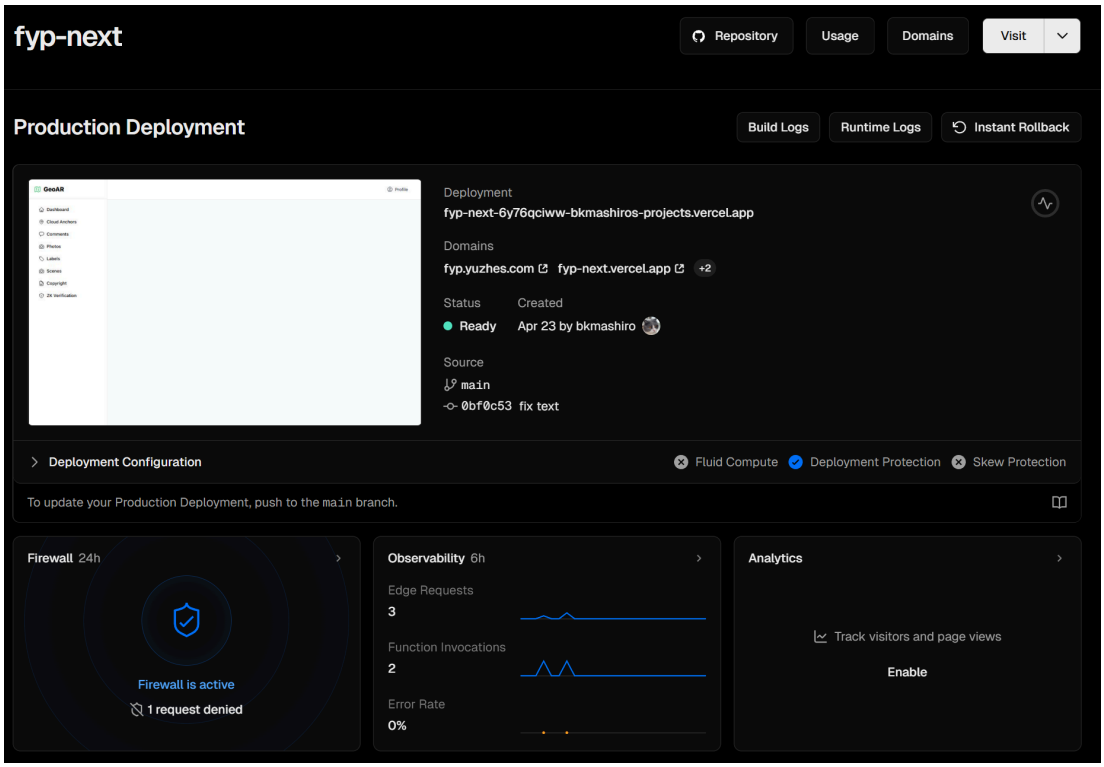


Figure 14: Web application deployment using Vercel

6.3. Backend Deployment

The backend infrastructure was built on Heroku with AWS integration, creating a robust and scalable deployment environment. Heroku dynos provided the application hosting platform, while AWS RDS PostgreSQL handled database operations. Redis was imple-

[14] the iOS app is not available in AppStore due to Apple's policy

mented for caching, and AWS S3 was used for file storage, creating a comprehensive cloud infrastructure.

The deployment pipeline was automated using GitHub Actions, ensuring consistent and reliable deployments. Automated testing was integrated into the pipeline, while database migrations were handled through a systematic approach. Environment configuration was managed through secure environment variables, maintaining separation between development and production environments.

Monitoring and observability were critical components of the backend deployment strategy. Application performance monitoring tools provided real-time insights into system health, while error tracking systems helped identify and resolve issues quickly. Log aggregation and resource utilization metrics enabled proactive system management and capacity planning.

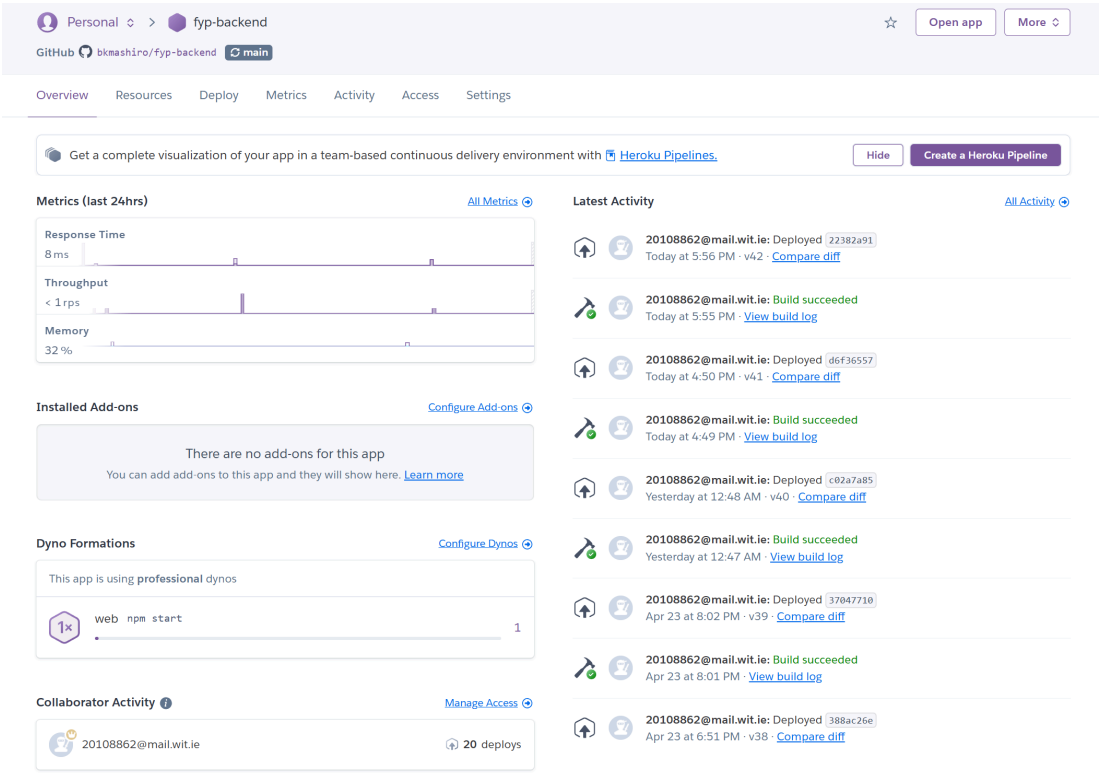


Figure 15: Backend deployment using Heroku

6.4. Database

The database was deployed on AWS RDS PostgreSQL, taking advantage of its modern deployment infrastructure. The platform provides a web interface for managing the database.

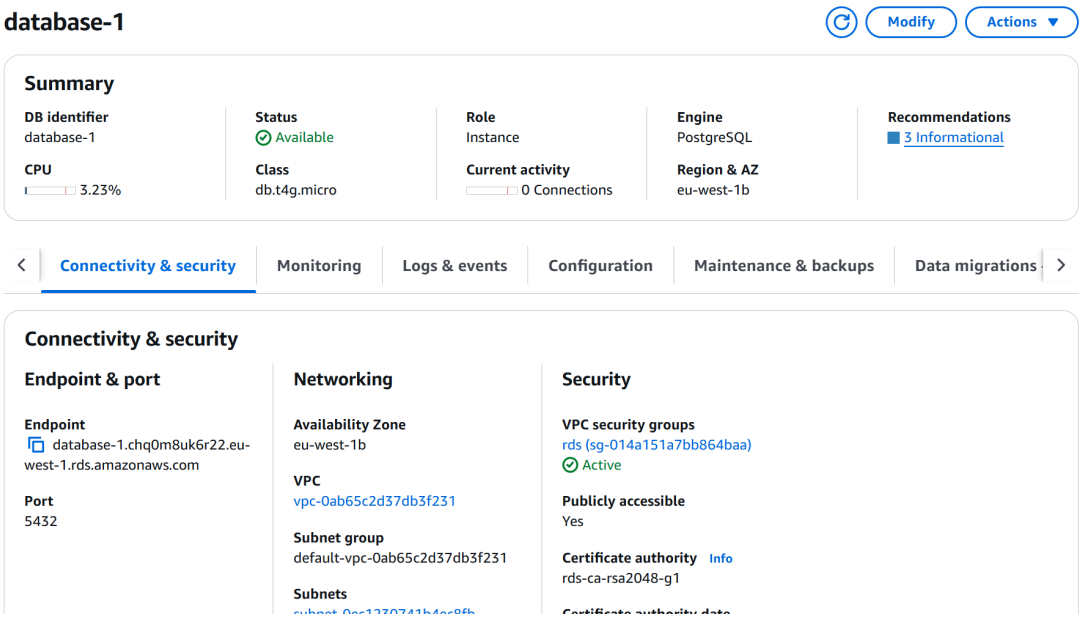


Figure 16: Database deployment using AWS RDS

7. User Manual

7.1. Mobile App Usage

7.1.1. Scanning and Hosting Cloud Anchors

Cloud anchors serve as the foundation of the positioning system and are essential for initiating the application.

7.1.1.1. Scanning a Cloud Anchor

The scanning process automatically activates when a cloud anchor is detected nearby. As users navigate their surroundings, the application reconstructs a 3D map of the environment. Once sufficient environmental details are captured, the anchor is automatically resolved.

7.1.1.2. Hosting a Cloud Anchor

Figure 17: Host a cloud anchor

To host a new cloud anchor, users should:

1. Select the “Anchor” tool
2. Tap on the screen to designate the desired anchor location

A quality indicator will then appear, displaying color-coded bars:

- Green: Optimal quality
- Yellow: Acceptable quality
- Red: Poor quality

For successful hosting, users should:

1. Move around the target point in a 180-degree arc
2. Continue until the overall quality reaches an acceptable level
3. The anchor will be hosted automatically once quality requirements are met

7.1.2. Viewing

Once the anchor is resolved, the app will automatically load the scene related to the anchor.

7.1.3. Create a 3D object

To take a photo or leave a comment, users should:

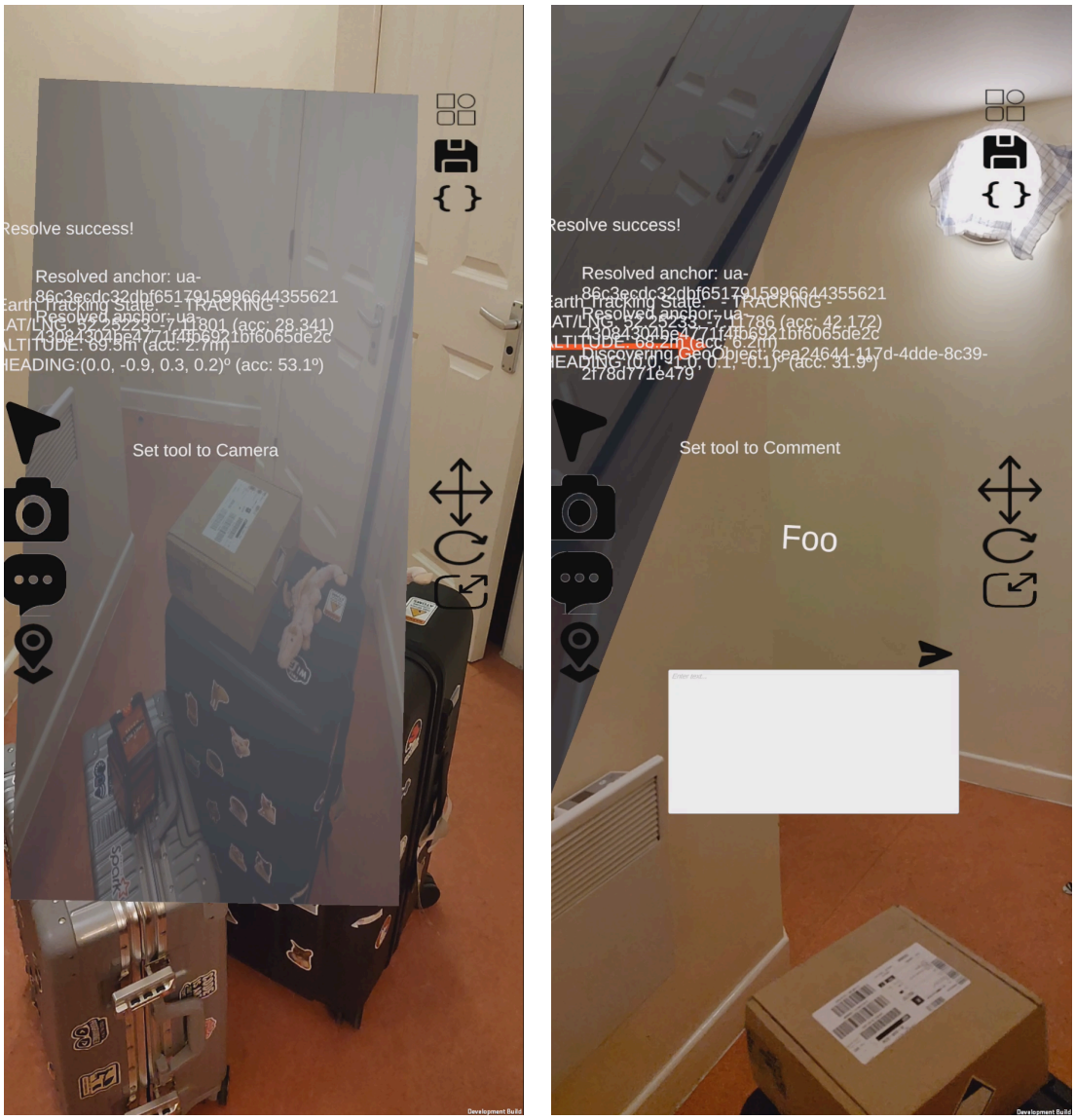
1. Switch to the desired mode using the mode selector:
    - Camera mode: For capturing photos
    - Comment mode: For leaving text comments

In camera mode:

1. Tap the center of the screen to capture a photo
2. The photo will appear as a 3D object in the scene

In comment mode:

1. Tap the center of the screen to open the keyboard
2. Type your comment
3. Tap the arrow button to send
4. The comment will appear as a 3D text object in the scene



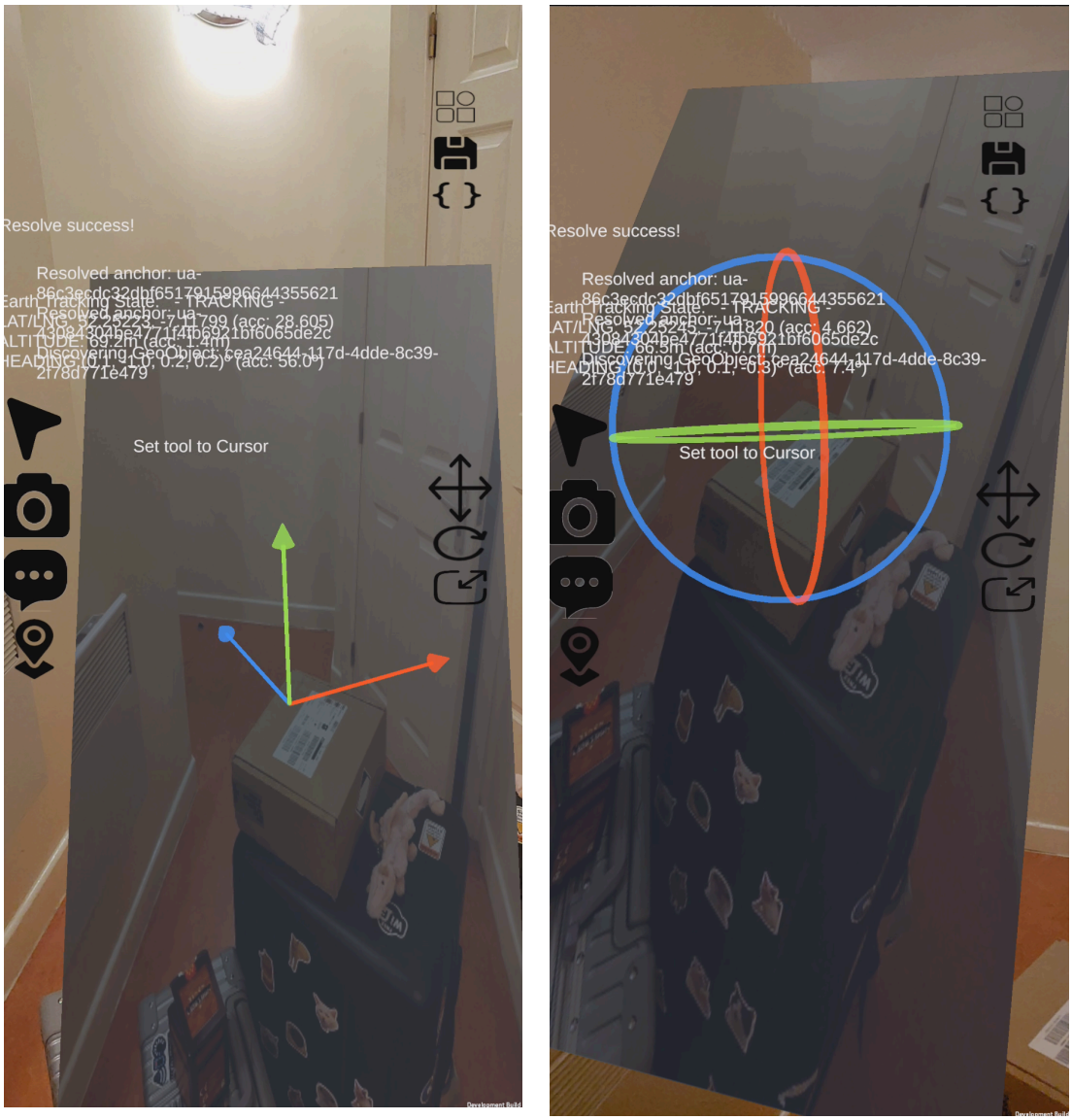
Camera Mode Comment Mode

Figure 18: Taking a photo and leaving a comment

7.1.4. Edit a 3D object

To edit a 3D object, users should:

1. Select the object by tapping on it
2. Choose the desired operation mode from the side panel:
    - Translation: Move the object in 3D space
    - Rotation: Rotate the object around its axes
    - Scale: Resize the object proportionally



Translate

Rotate

Figure 19: Edit a 3D object

The selected object will be highlighted, and users can:

1. Drag to move/rotate/scale the object
2. Drag the arcball to rotate the object
3. Drag the scale handle to resize the object

7.2. User Panel Guide

7.2.1. Login/Register

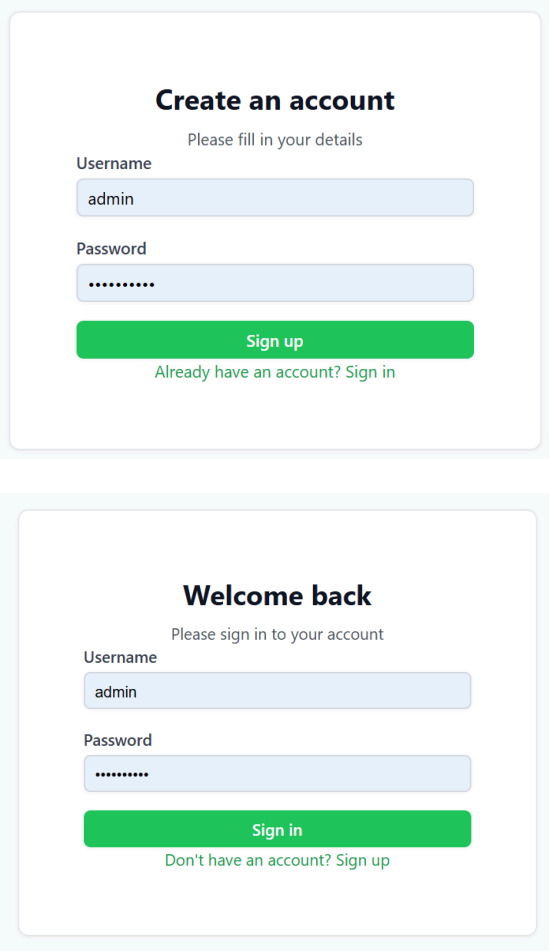


Figure 20: Login/Register interface

Users can register or login to the system using their username and password. The registration process requires:

1. A unique username
2. A secure password

7.2.2. User Profile

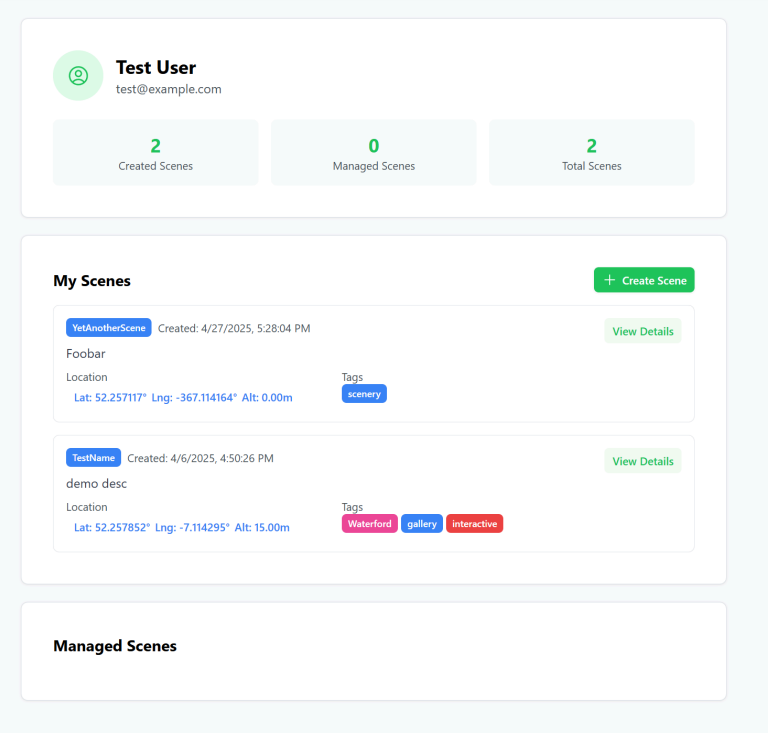


Figure 21: User profile

Users can view their profile information, including:

1. Username
2. Email
3. Profile picture
4. Scenes (created and managed by the user)

7.2.3. Scene Management

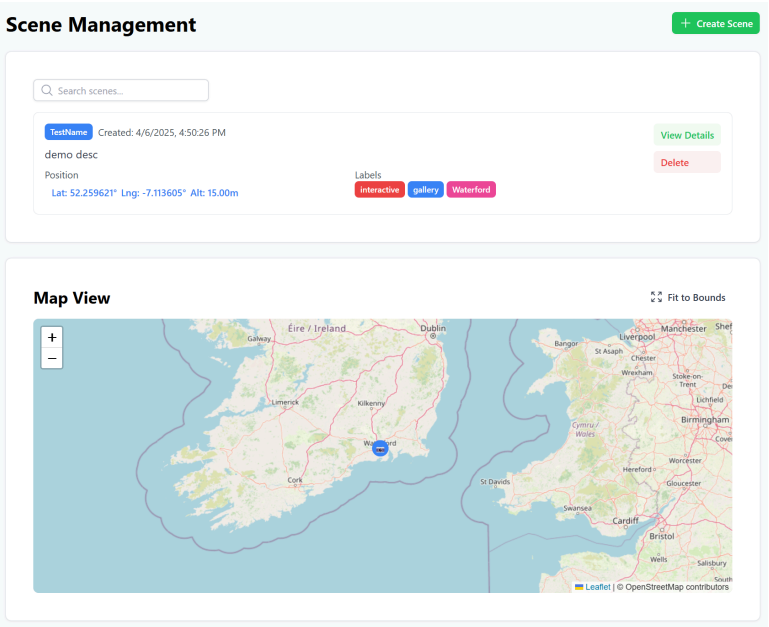


Figure 22: Scene Management Interface

The scene management interface provides users with comprehensive control over their virtual scenes. Key features include:

1. Scene Creation: Users can create new scenes with customizable names and descriptions
2. Scene Editing: Existing scenes can be modified to update their properties
3. Scene Deletion: Unwanted scenes can be removed from the system
4. Label Management: Each scene supports multiple descriptive labels for better organization

A global map view is integrated below the scene list, displaying the geographic distribution of all scenes across the platform.

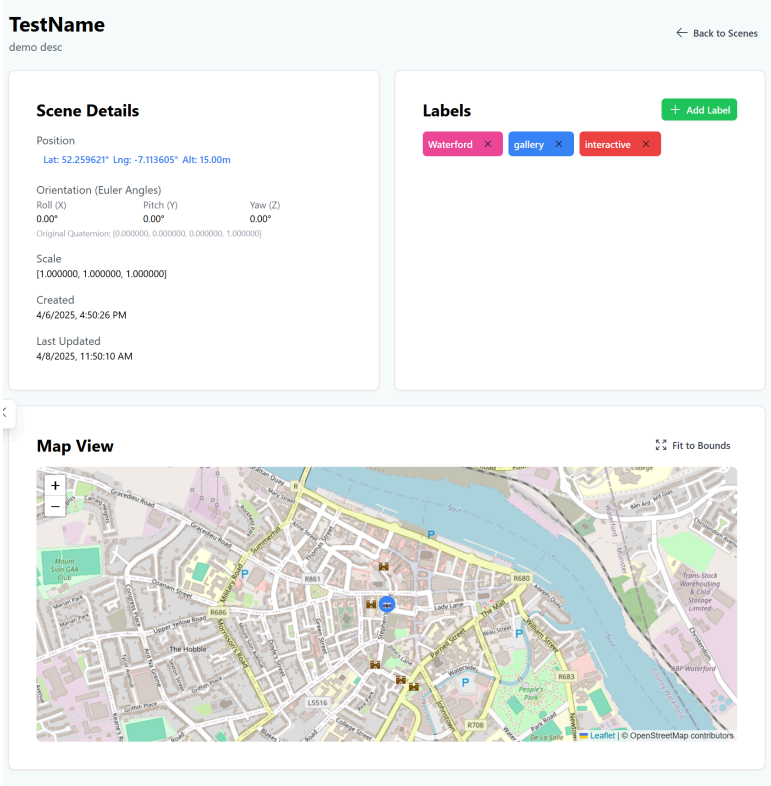


Figure 23: Scene Details View

The scene detail view offers an in-depth look at each scene's properties and contents:

1. Basic Information: Scene name and detailed description
2. Geographic Data: Location-specific details and coordinates
3. Label Management: Interactive panel for viewing and modifying scene labels
4. Spatial Visualization: An integrated map view displaying all geo-referenced objects within the scene

This comprehensive interface enables users to effectively manage and interact with their virtual scenes while maintaining spatial context through the integrated mapping system.

7.2.4. Cloud Anchor Management

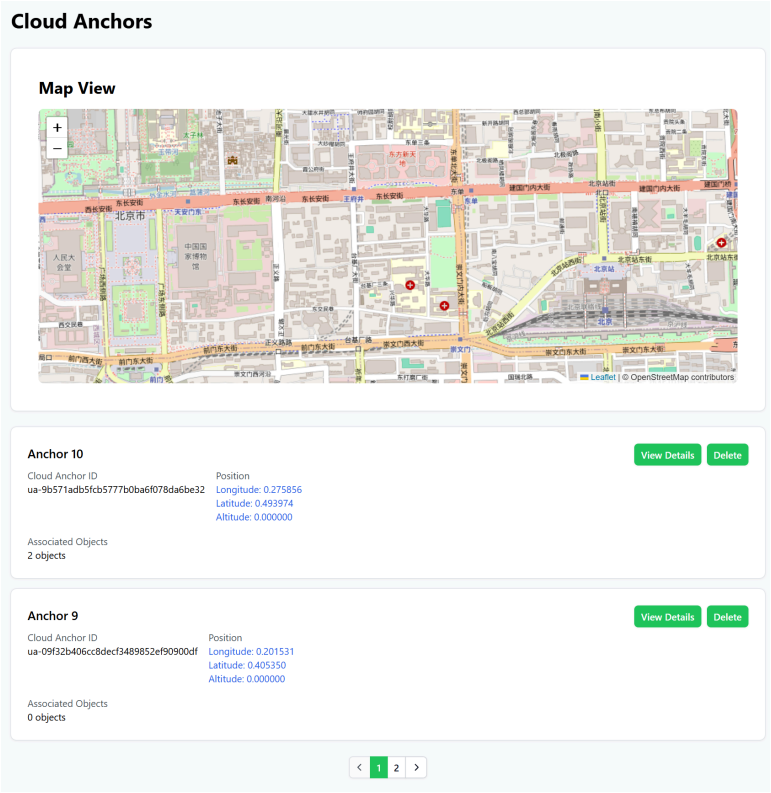


Figure 24: Cloud Anchor Management Interface

The cloud anchor management interface allows users to:

1. View all available cloud anchors in the system in map view
2. View the details of a specific cloud anchor
3. Delete a cloud anchor
4. View paginated list of cloud anchors

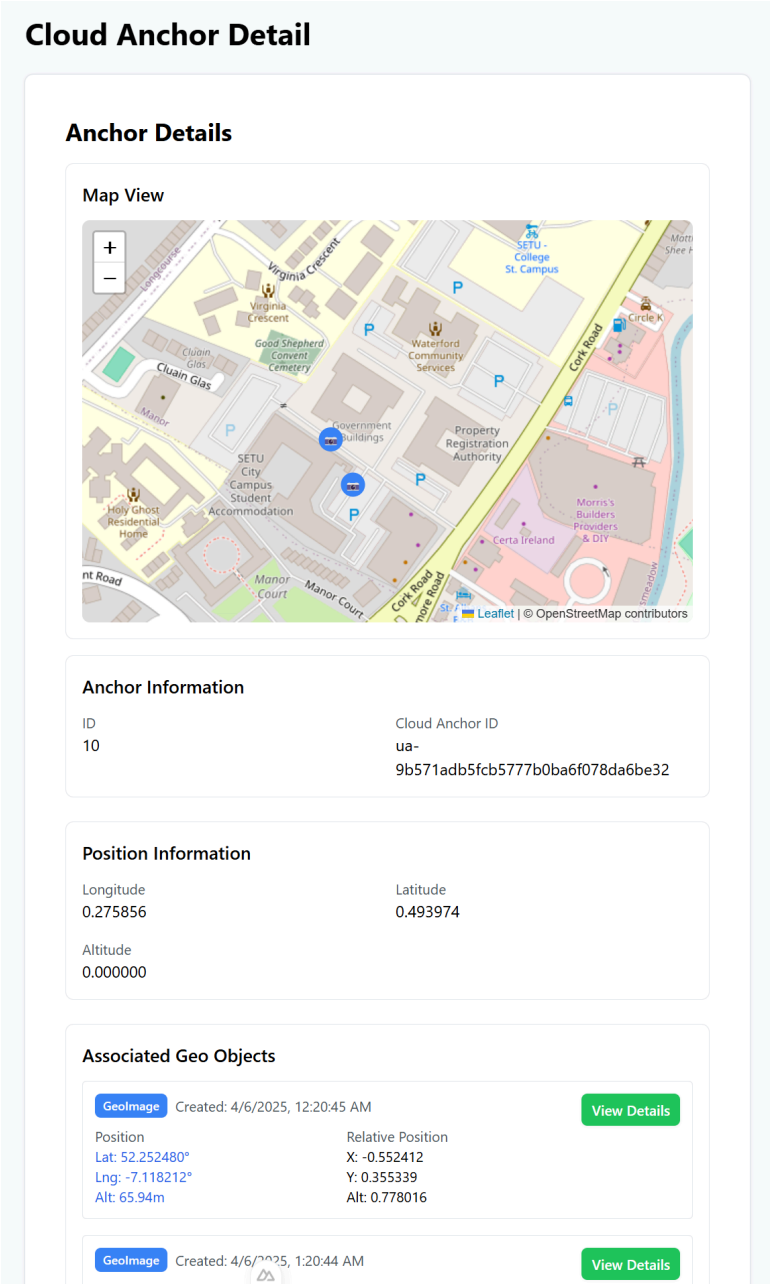


Figure 25: Cloud Anchor Details View

The cloud anchor details view provides comprehensive information about a specific cloud anchor:

1. Basic Information: Anchor name and description
2. Geographic Data: Location-specific details and coordinates
3. Object Associated: List of objects associated with the anchor, including a link to the object details

7.2.5. Photo Management

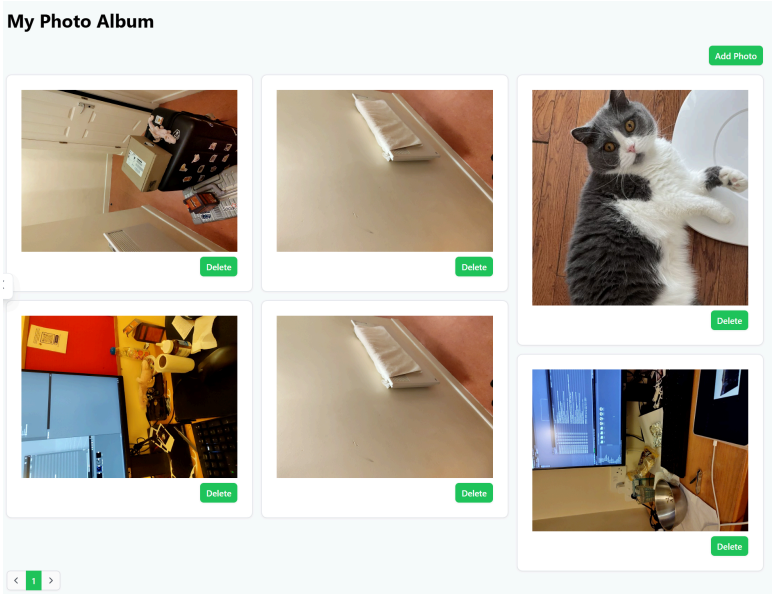


Figure 26: Photo Management Interface

The photo management interface allows users to:

1. View the details of a specific photo
2. Delete a photo
3. View paginated list of photos

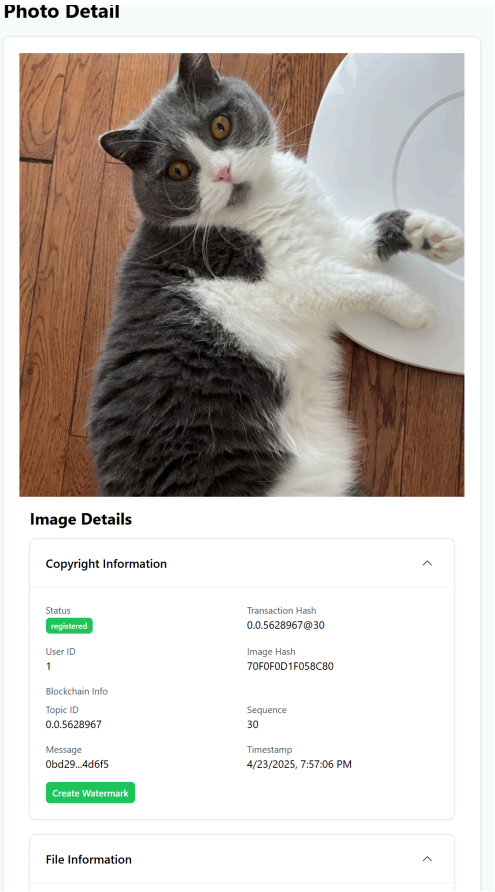


Figure 27: Photo Details View

The photo details view provides comprehensive information about a specific photo:

1. Basic Information: Photo name and description
2. Geographic Data: Location-specific details and coordinates
3. Parent Scene: Link to the parent scene
4. Creation Date: Date and time of photo creation
5. Copyright Information: Copyright information/blockchain information of the photo

In the Copyright Information section, users can download the blind-watermarked photo, which can be verified in the Section 7.2.8.

7.2.6. Comment Management

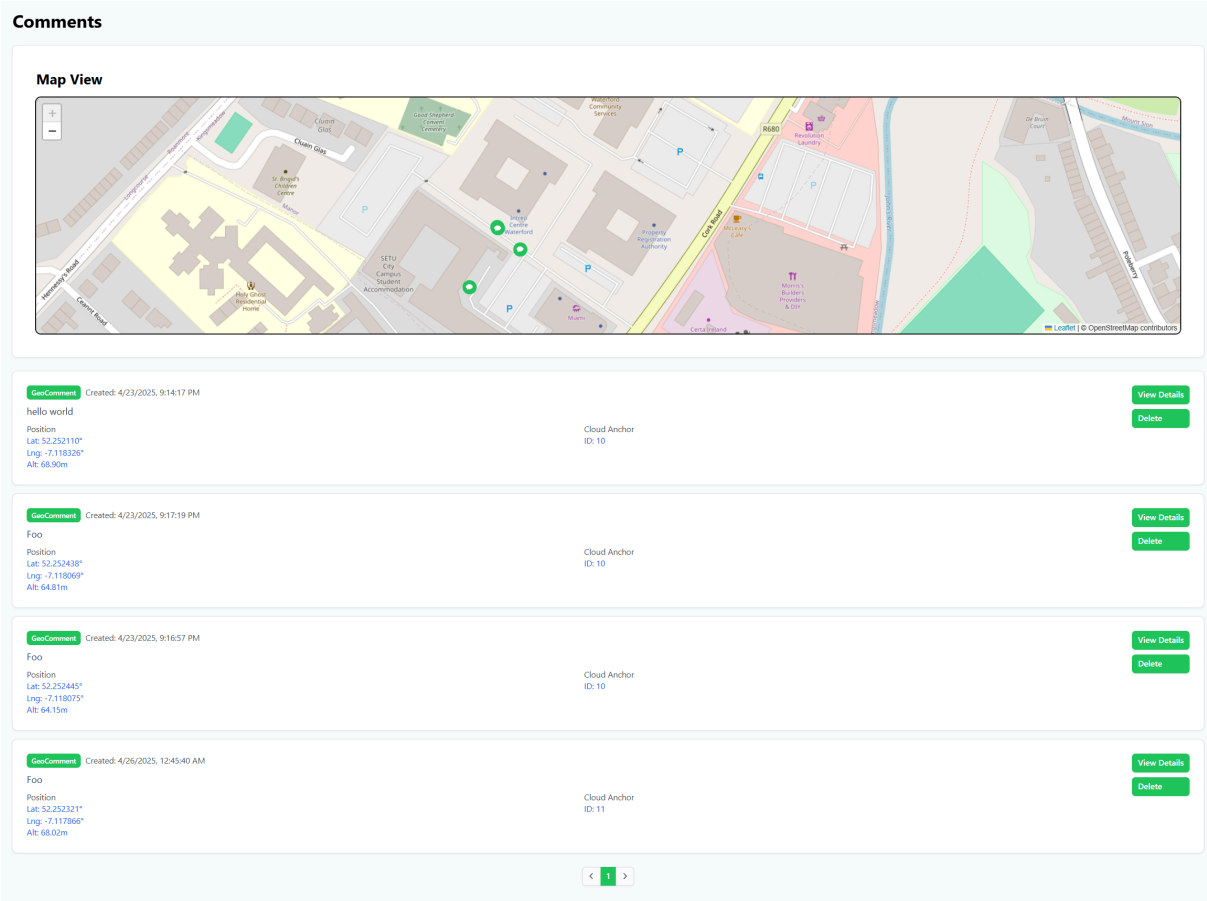


Figure 28: Comment Management Interface

The comment management interface allows users to:

1. View all comments in the system in map view
2. View the details of a specific comment
3. Delete a comment
4. View paginated list of comments

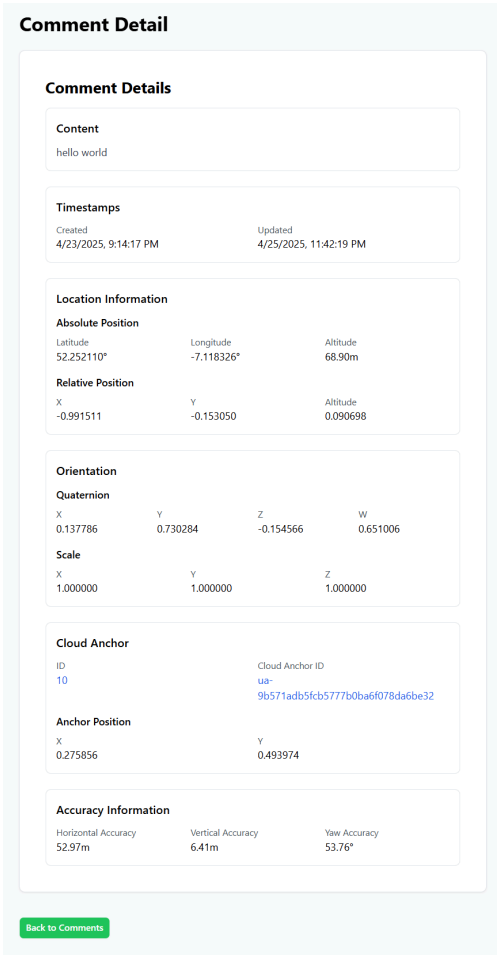


Figure 29: Comment Details View

The comment details view provides comprehensive information about a specific comment:

1. Basic Information: Comment name and description
2. Geographic Data: Location-specific details and coordinates
3. Creation Date: Date and time of comment creation

7.2.7. Label Management

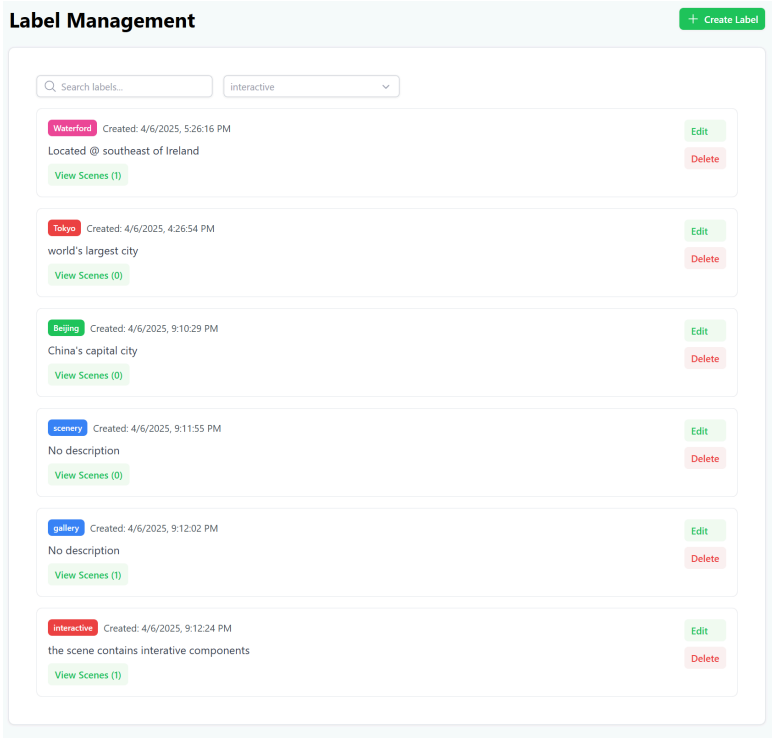
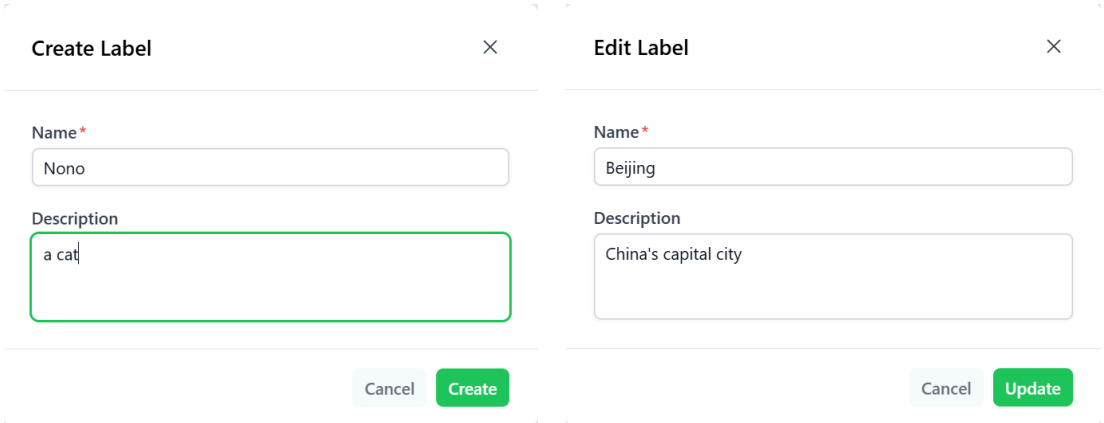


Figure 30: Label Management Interface

The label management interface allows users to:

1. View the details of a specific label
2. View all scenes associated with a label
3. Search for a label
4. Add / Edit / Delete a label



Create Edit

Figure 31: Add / Edit form

7.2.8. Verify Image Copyright

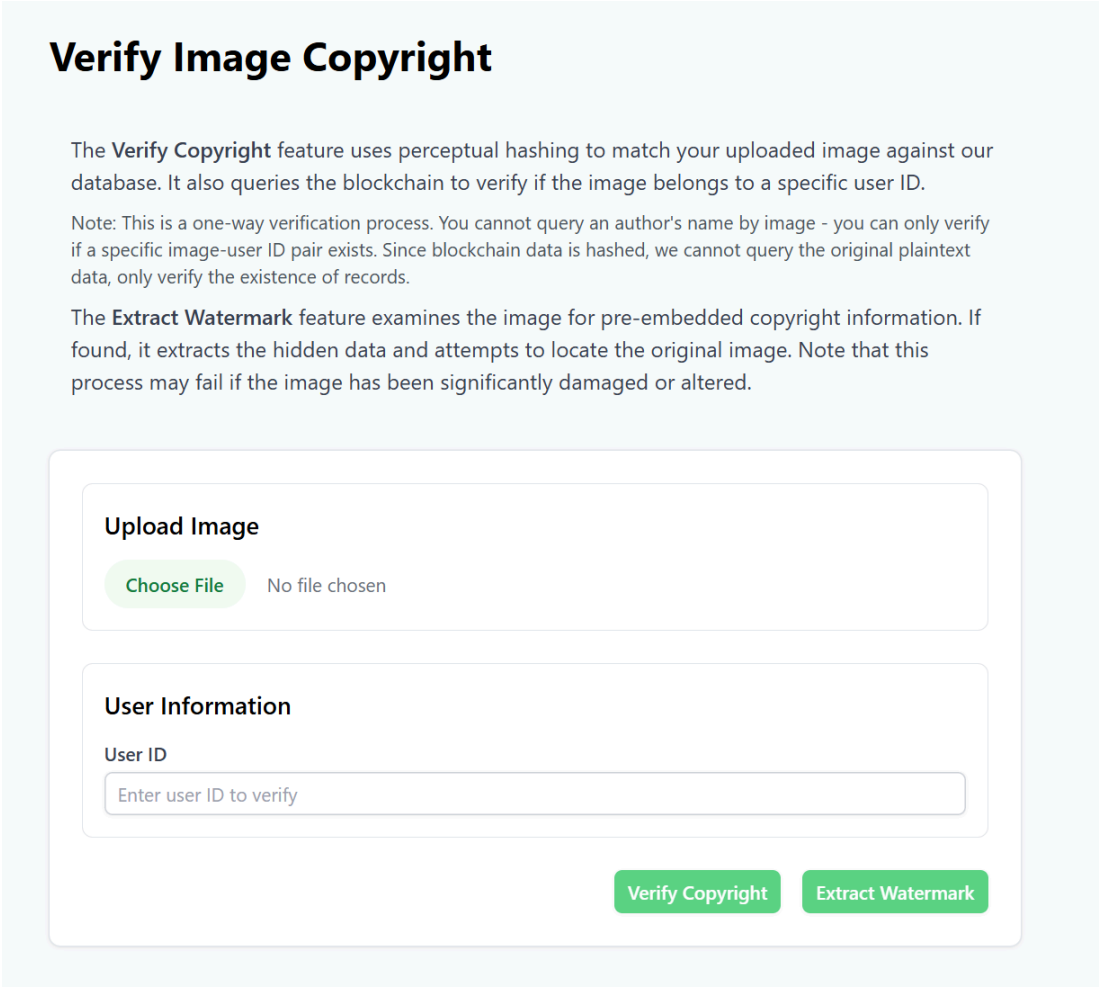


Figure 32: Verify Image Copyright Interface

The verify image copyright page provides two main functionalities for image verification and copyright protection:

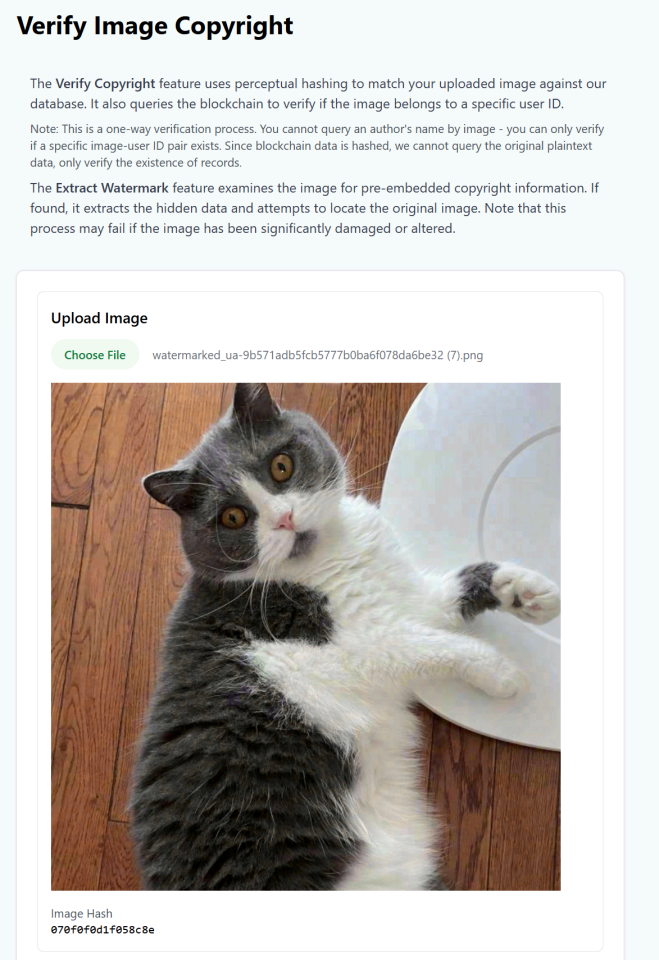
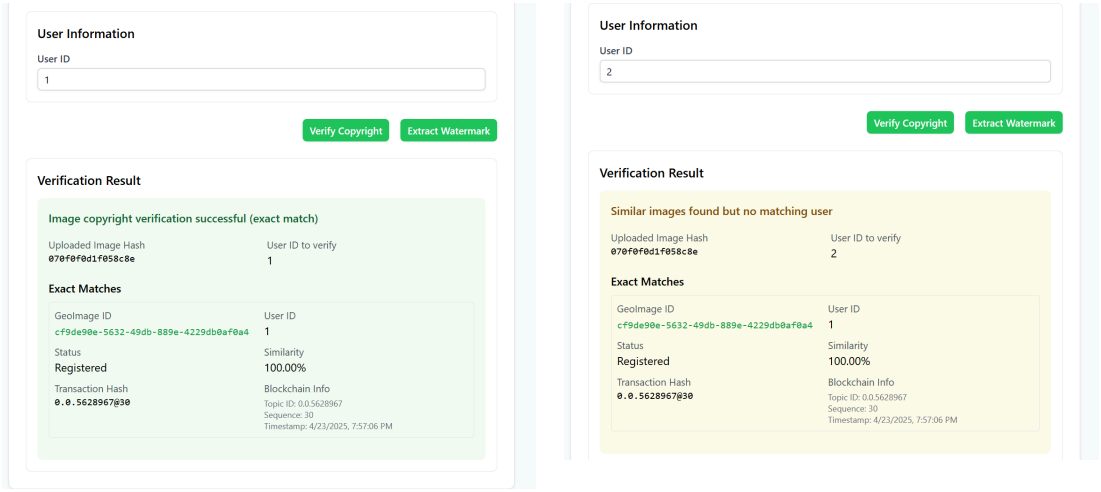


Figure 33: Upload Image

1. **Image Copyright Verification**
   - Upload an image and enter a user ID
   - System performs perceptual hashing to match against database
   - Searches for similar images and corresponding geoImages
   - Verifies if the image was created by the specified user
   - Returns both potential and exact matches
2. **Watermark Extraction**
   - Extracts embedded copyright information from the image
   - Searches database cache and blockchain for matching records
   - Note: Process may fail if image has been significantly altered

The system implements one-way verification for privacy protection:

- Can verify if an image belongs to a specific user ID
- Cannot query author information by image
- Blockchain data remains hashed and secure



Exact match Suspicious

Figure 34: Verify Image Copyright Successful Results

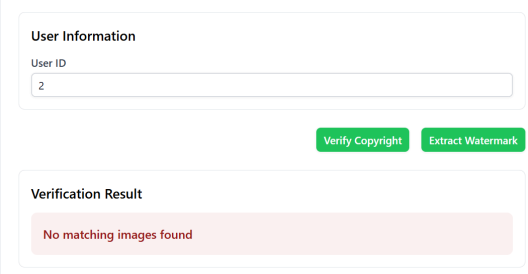
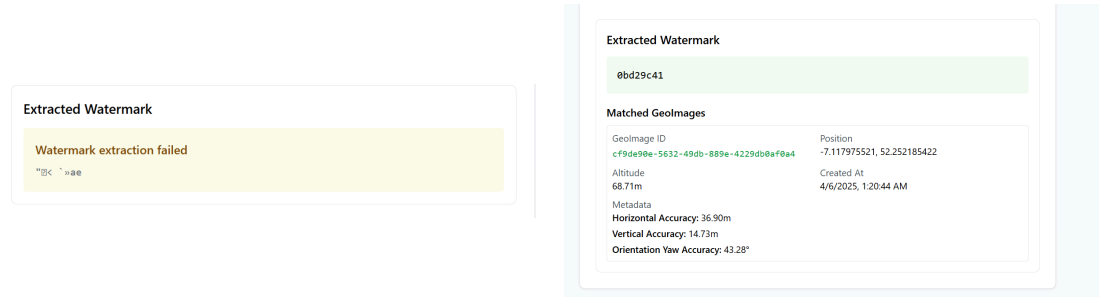


Figure 35: Verify Image Copyright Failed Results



Broken Watermark Extracted and Found

Figure 36: Verify Image Copyright Blind Watermark

7.2.9. Zero-Knowledge Proof

The zero-knowledge copyright verification system enables users to create and verify copyright proofs in a privacy-preserving manner. The process involves several key components:

1. **Proof Generation**
   - User generates key pair locally (never leaves device)
   - Creates commitment from artwork hash and signature
   - Generates zero-knowledge proof using Groth16
   - Optionally stores proof and commitment on blockchain

2. **Verification Modes**
    - Blockchain-based verification:
        - Verifier computes artwork hash
        - Searches blockchain for matching commitment
        - Verifies zero-knowledge proof
        - Optionally checks owner address
    - Direct commitment verification:
        - Verifier provides commitment value
        - System verifies zero-knowledge proof
        - Optionally checks owner address
3. **Privacy Features**
    - Owner address verification is optional
    - Original artwork content remains private
    - Only commitment values are stored on-chain
    - Proof verification reveals no additional information

This implementation ensures that copyright verification can be performed without compromising the privacy of the artwork owner or revealing sensitive information about the artwork itself.

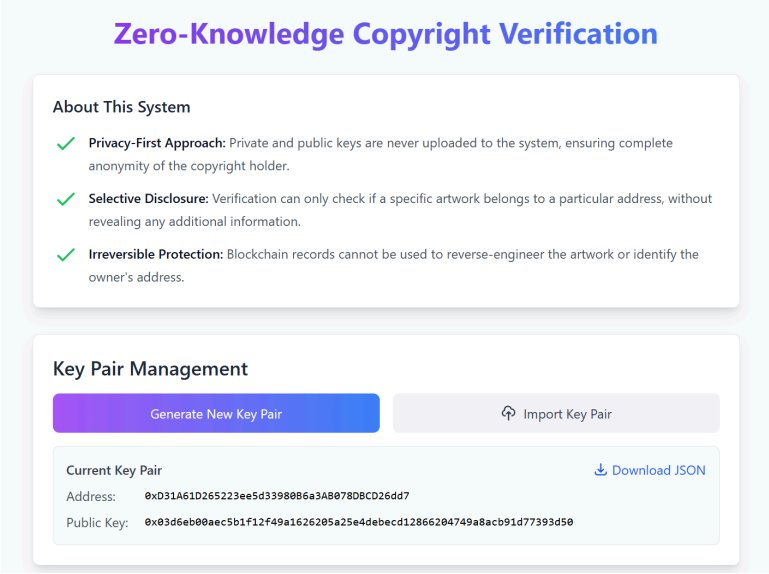


Figure 37: Zero-Knowledge Proof Key Pair

Click “Generate New Key Pair” to generate a new key pair.

Or click “Import Key Pair” to import an existing key pair[15].

Click “Download JSON” to download the key pair as a JSON file.

Click on Address / Public Key to copy the address / public key to the clipboard[16].

[15] This allows users to generate a key pair offline by themselves.

[16] It’s recommended to not to share public key and address to anyone unless you intend to reveal your identity.

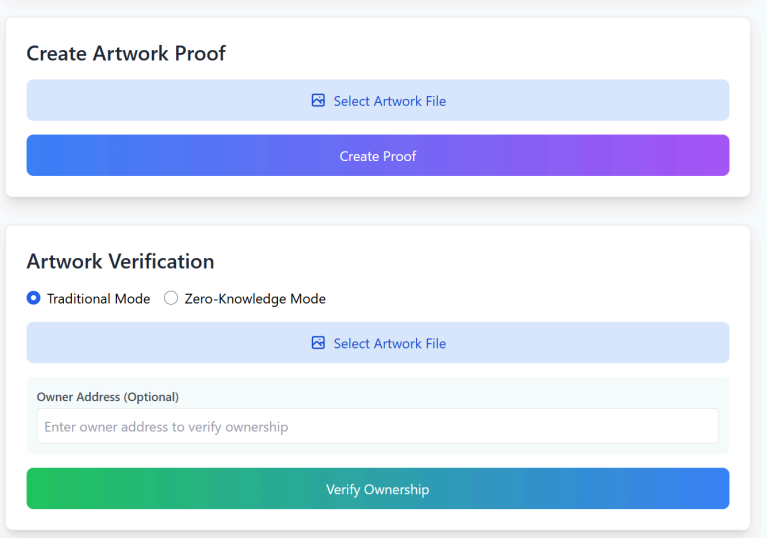


Figure 38: Zero-Knowledge Proof Interface

Click “Select Artwork File” to upload an artwork file.

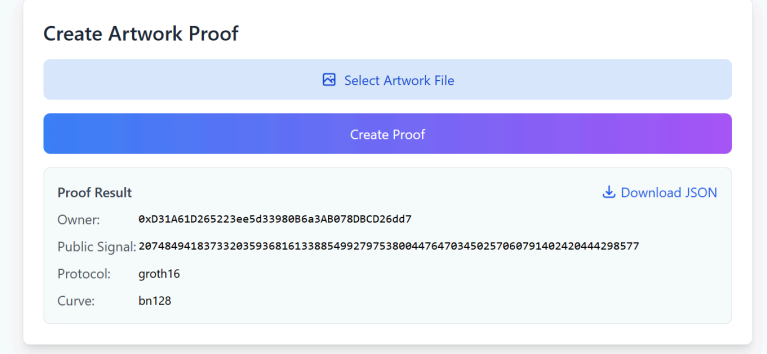


Figure 39: Zero-Knowledge Proof Created

A proof is created with details as below:

- Owner Address: The address of the owner of the artwork
- Public Signal: The first public signal of the artwork
- Protocol: The protocol used to create the proof
- Curve: The curve used to create the proof

Click “Download Proof” to download the proof as a JSON file.

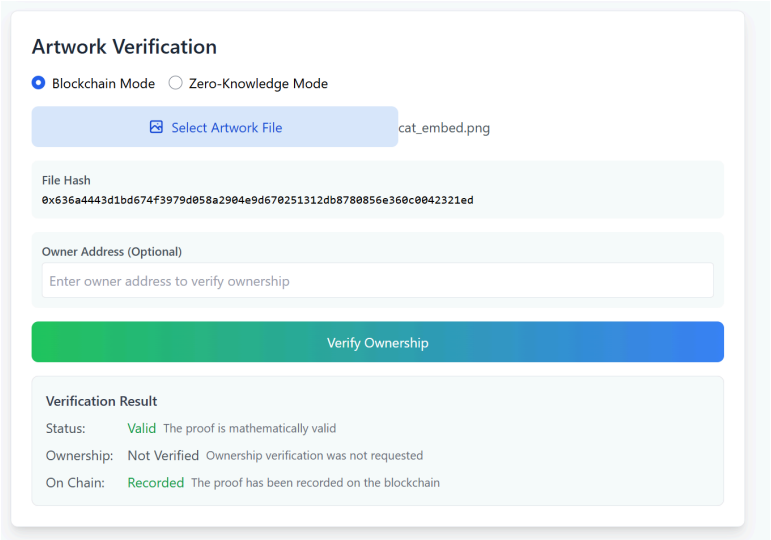


Figure 40: Zero-Knowledge Proof Blockchain Verification

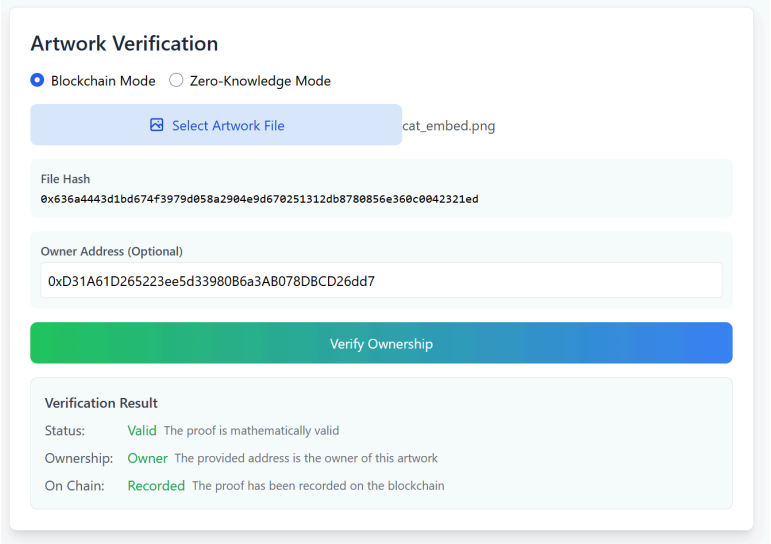


Figure 41: Zero-Knowledge Proof Blockchain Verification with Owner Address

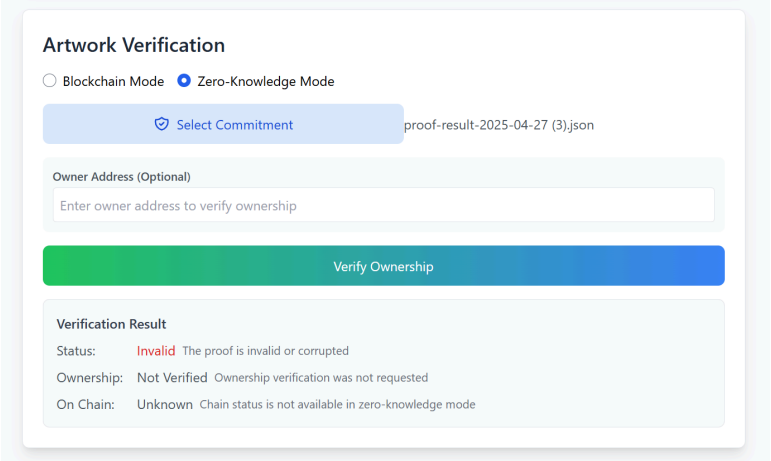


Figure 42: Zero-Knowledge Proof Broken Proof

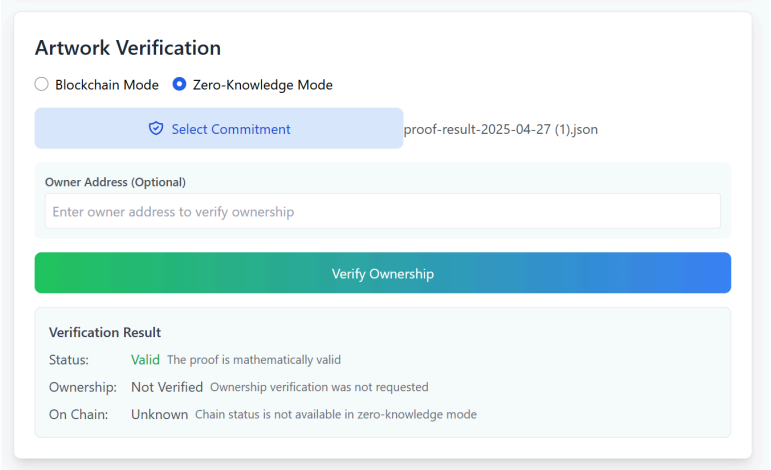


Figure 43: Zero-Knowledge Proof without Owner Verification

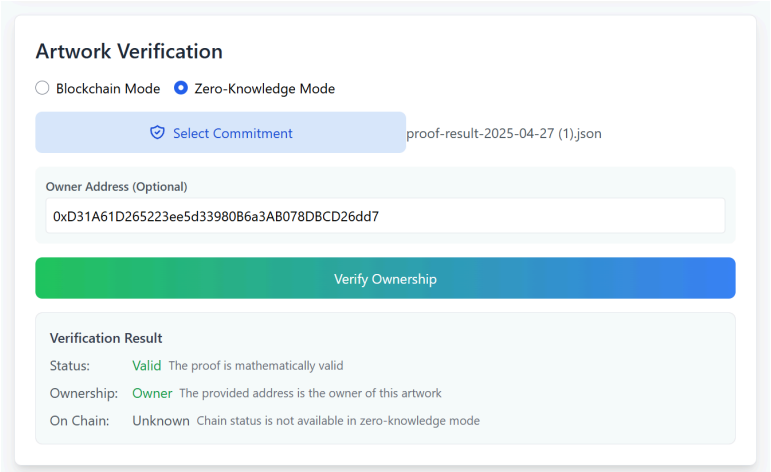


Figure 44: Zero-Knowledge Proof with Owner Verification

8. Conclusion

8.1. Project Summary

The SnapSphere project represents a significant advancement in the field of immersive photo experiences, combining cutting-edge technologies to create a unique platform for digital content sharing and protection. The project successfully integrated augmented reality, blockchain technology, and advanced security features to deliver a comprehensive solution for modern digital content management.

Key achievements of the project include:

1. **Innovative Technology Integration**
   - Seamless AR experience with cloud anchors
   - Robust blockchain-based copyright protection
   - Advanced invisible watermarking system
   - Efficient image similarity search
2. **Security and Privacy**
   - Zero-knowledge proof implementation
   - Tamper-proof content protection
   - Privacy-preserving verification
   - Secure data management
3. **User Experience**
   - Intuitive AR interactions
   - Responsive web interface
   - Cross-platform compatibility
   - Real-time social features

The project demonstrates the potential of combining emerging technologies to create innovative solutions for digital content management and social interaction. The successful implementation of complex features like invisible watermarking and zero-knowledge proofs showcases the technical capabilities of the development team.

8.2. Future Work

The project has laid a solid foundation for future development and enhancement. Several areas have been identified for potential improvement and expansion:

1. **Unimplemented Features**
   - Lua scripting support for dynamic behavior
   - Offline mode functionality
   - Enhanced AR interactions
   - Advanced content management tools
2. **UI/UX Improvements**
   - More intuitive AR controls
   - Enhanced visual feedback
   - Improved accessibility features
   - Customizable interface options
3. **Technical Enhancements**
   - Performance optimization
   - Scalability improvements

- Advanced analytics
- Machine learning integration

4. **Integration Opportunities**
   - Social media platform integration
   - Third-party login service support
   - Better cross-platform compatibility
   - API expansion
5. **Security and Privacy**
   - Enhanced encryption methods
   - Advanced privacy controls

These future developments will further enhance the platform's capabilities and user experience, making it an even more powerful tool for digital content creation and sharing.

9. Appendix

9.1. A Simple Example of Zero-Knowledge Proof

9.1.1. The Magic Cave Story[1]

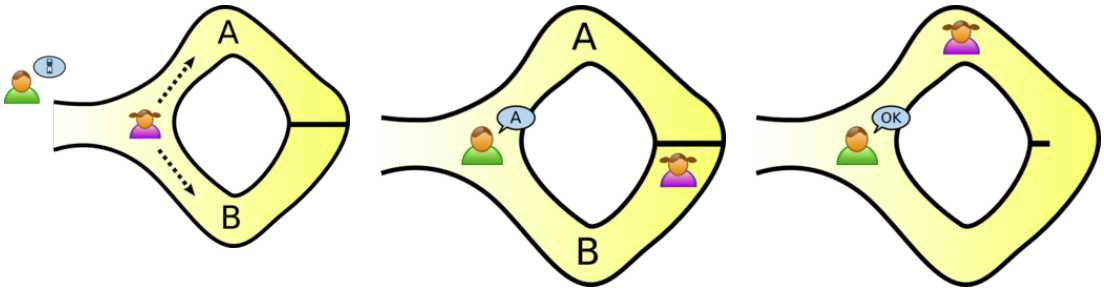


Figure 45: The Magic Cave Story

Once upon a time, you, as Ali, discovered a magical cave with two paths:

- Path A (left)
- Path B (right)

At the heart of the cave, these paths converge at a mysterious door that can only be opened with a secret magic word.

9.1.2. The Challenge

Your friend Xiao Ming is skeptical about your knowledge of the secret word. To prove your knowledge without revealing the word itself, you devise a clever game:

1. **While Xiao Ming waits outside**, you enter the cave through either path
2. Xiao Ming then randomly requests you to exit through Path A or B
3. If you truly know the secret word, you can:
    - Unlock the door
    - Exit through whichever path Xiao Ming requests
4. If you don't know the word, you must:
    - Return through the same path you entered

> **Notification**
>
> The key to this being a "zero-knowledge" proof lies in Xiao Ming not knowing which path you initially entered through.
>
> If you were to let Xiao Ming see you enter through Path A and exit through Path B, he would gain a **fact** about your knowledge of the secret word.
>
> This would violate the **zero-knowledge** property, as Xiao Ming could then prove to others that you know the secret word by recording the process. (If he secretly recorded the process, he could then show it to others.)
>
> The beauty of the zero-knowledge proof is that Xiao Ming becomes convinced of your knowledge, but cannot convince others.

9.1.3. The Proof Process

By repeating this game multiple times (say, 20 times), Xiao Ming becomes increasingly convinced of your knowledge because:

- You consistently exit through the requested path
- The probability of guessing correctly every time is astronomically low
- The secret word itself remains a mystery to Xiao Ming

Most importantly, Xiao Ming cannot prove your knowledge to others. Since he never sees which path you initially enter through, any recording of the process could be easily faked by two people working together.

9.1.4. Demonstrating ZKP Properties

This enchanting story perfectly illustrates the three fundamental properties of zero-knowledge proofs:

Completeness	If you know the word, you always succeed
Soundness	If you don't know the word, you'll likely fail eventually
Zero-Knowledge	Xiao Ming learns nothing about the word itself, just that you know it

9.2. Why using Incomplete Zero-Knowledge Proofs?

From a theoretical perspective of zero-knowledge proofs,we can use **only the commitment** to verify the proof. In fact, we can design a more "zero-knowledge" solution:

The commitment should essentially only contain:

- The output value of Poseidon (sigHash, artHash)

This value serves as the public input. In the circuit, we verify:

- The signer indeed knows the corresponding signature (sigHash)
- This signature is indeed for the specific artwork (artHash)
- The Poseidon hash of these two values equals the commitment

This design has the following advantages:

- More aligned with zero-knowledge principles, as we don't need to expose the owner's address at all
- Simpler verification process, only requiring proof validity check
- Better privacy, as external observers cannot determine who the owner is

However, the current implementation chooses to store pubKeyHash (i.e., owner's address) simultaneously, is due to the following practical considerations:

- Facilitates on-chain tracking and querying
- Supports explicit ownership verification
- Better integration with existing blockchain ecosystems

Therefore, this represents a trade-off between theoretical purity and practicality. From a pure zero-knowledge proof perspective - we could indeed complete the verification using only the commitment, but it's not practical here.

9.3. Semester 1 Project Proposal

9.3.1. Motivation and Background

The motivation behind this project stems from three key observations about the current state of digital media and social interaction:

1. Digital Media Evolution
   - Traditional photo sharing platforms lack spatial context and interactive experiences
   - Users desire more immersive ways to experience digital content
   - AR technology enables new forms of visual storytelling
2. Content Protection Needs
   - Growing concerns about digital content ownership
   - Increasing cases of unauthorized content usage
   - Need for robust copyright protection mechanisms
3. Social Interaction Enhancement
   - Limited ways to share experiences in digital spaces
   - Desire for more meaningful social connections
   - Potential of AR to create shared virtual spaces

The project addresses these challenges by combining several innovative technologies:

1. Augmented Reality Integration
   - Enables spatial photo viewing
   - Creates immersive social experiences
   - Bridges physical and digital worlds
2. Blockchain-based Protection
   - Ensures content ownership
   - Provides tamper-proof records
   - Enables trustless verification
3. Advanced Security Features
   - Invisible watermarking for content tracking
   - Zero-knowledge proofs for privacy
   - Perceptual hashing for content identification

This combination of technologies creates a unique platform that:

- Preserves the spatial context of photos
- Protects creators' rights
- Fosters meaningful social interactions
- Maintains user privacy and security

The project's background is rooted in the convergence of several technological trends:

- The rise of AR/VR technologies
- Growing adoption of blockchain solutions
- Increasing focus on digital privacy
- Evolution of social media platforms

These factors create an ideal environment for developing a platform that reimagines how we share, protect, and experience digital content in an increasingly connected world.

9.3.2. Objectives and Scope

The project aims to create an innovative platform that combines AR technology, blockchain, and social features to revolutionize how we experience and share digital content. The objectives are organized around three main user groups:

9.3.2.1. General Users

Immersive Experience

- Explore photos in AR/MR with spatial context
- View content from original capture perspectives
- Interact through spatial comments and replies

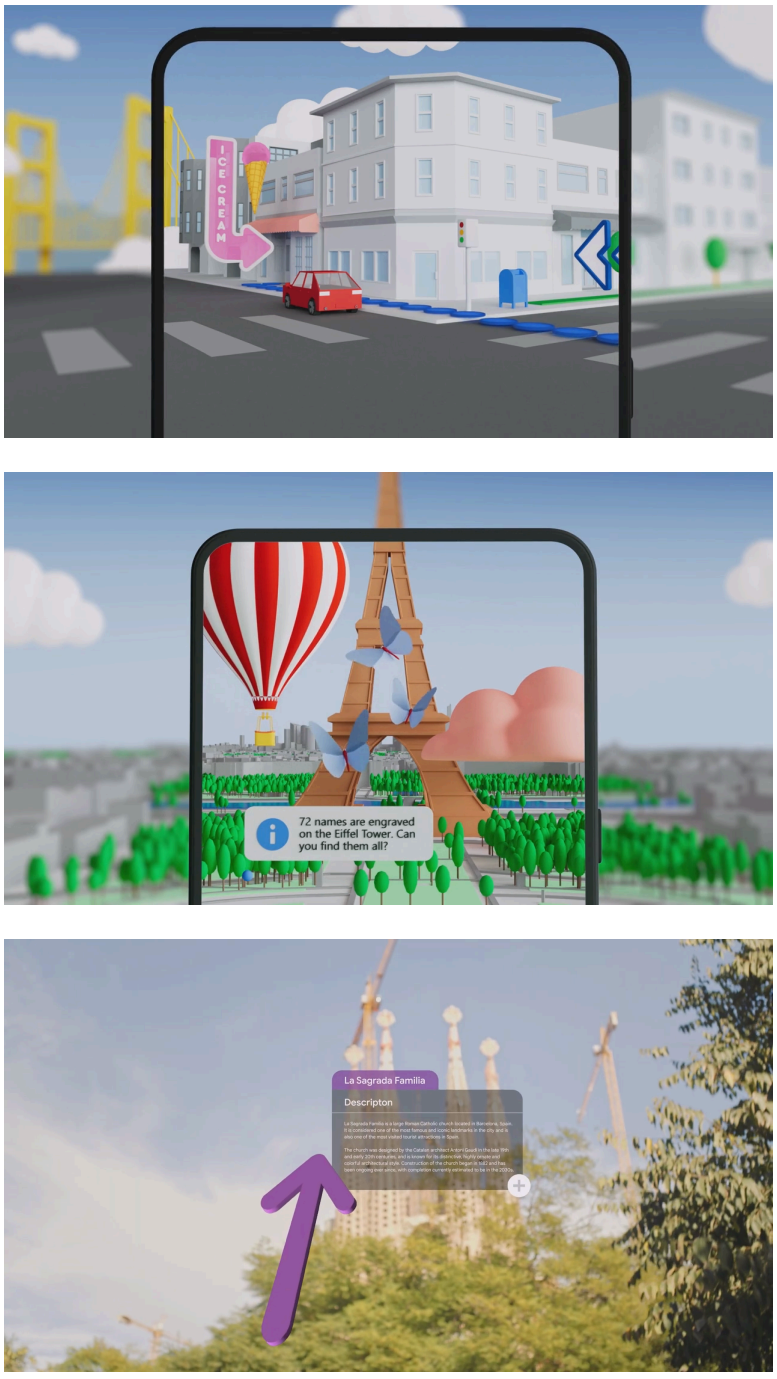


Figure 46: Conceptual diagram of AR/MR experience[5]

Privacy and Discovery

- On-device processing of private data
- Flexible privacy controls for data protection

9.3.2.2. Content Creators

Creative Tools

- Create interactive AR stories with photos and media
- Showcase artwork in immersive environments
- Build time-based narratives and sequences

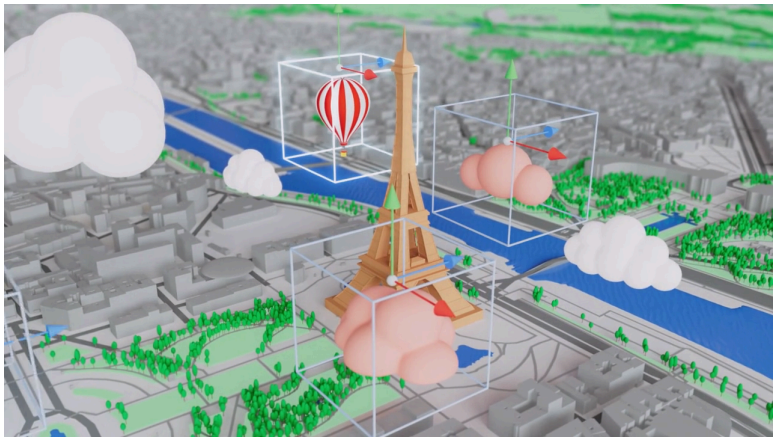


Figure 47: Conceptual diagram of content creator[5]

Content Protection

- Secure copyright management through blockchain
- Decentralized ownership verification
- Tamper-proof content records

9.3.2.3. Technical Scope

Core Features

- AR/MR content viewing and interaction
- Blockchain-based ownership management
- Privacy-preserving technology
- Real-time social features

Technical Limitations

- Device compatibility requirements
- Privacy implementation complexity
- External service dependencies
- Limited offline functionality

9.3.3. Feasibility Study

The feasibility study for SnapSphere was conducted to evaluate the project's viability across three key dimensions: market potential, technical requirements, and operational sustainability.

Market analysis revealed a growing demand for immersive AR experiences and interactive social media platforms, particularly among younger demographics. The project's unique combination of AR technology, blockchain-based content protection, and social features positions it well in the market. User research indicated strong interest in novel

ways to experience and share content, with particular emphasis on privacy and content ownership.

From a technical perspective, the project leverages well-established technologies including AR Foundation, blockchain, and zero-knowledge proofs. The core technologies have been proven in production environments, and the development team possesses the necessary expertise in these areas. Key technical challenges such as device sensor calibration, privacy-preserving architecture, and cross-platform compatibility were identified and addressed through careful planning and implementation.

Operational viability was assessed through a comprehensive evaluation of user experience, technical infrastructure, and resource requirements. The study confirmed that the project can be implemented with existing resources while maintaining high standards of performance and reliability. The combination of cloud services, modern development tools, and robust security measures ensures a sustainable operational model.

The study concluded that SnapSphere is technically feasible and commercially viable, with a clear path to implementation and a strong value proposition for users. The project's innovative approach to digital content sharing and protection addresses current market needs while leveraging proven technologies and best practices.

9.3.4. Requirements Analysis

9.3.4.1. Functional Requirements

9.3.4.1.1. User Authentication

- JWT-based authentication system
- Role-based access control
- Token revocation mechanism
- Secure key management

9.3.4.1.2. Content Creation

- Cloud anchor management
- Spatial positioning system
- Scene-based organization

9.3.4.1.3. Web Content Management

- Scene and content administration
- User management interface
- Content moderation tools
- Analytics dashboard

9.3.4.1.4. Backend Review and Analytics

- Content verification system
- Usage statistics tracking
- Performance monitoring
- Security audit logging

9.3.4.1.5. Non-Functional Requirements

9.3.4.1.6. Response Time

- Real-time AR content loading

- Fast image similarity search
- Efficient blockchain transactions
- Low-latency social interactions

9.3.4.1.7. Data Consistency

- Blockchain-based ownership records
- Synchronized AR content
- Reliable data backup

9.3.4.1.8. Privacy Protection

- Zero-knowledge proof implementation
- Invisible watermarking
- Privacy-preserving content discovery
- Secure data storage

9.3.4.1.9. Security Requirements

- Tamper-proof content protection
- Secure key management
- Access control enforcement
- Vulnerability prevention

9.3.4.1.10. Scalability Requirements

- Distributed content delivery
- Load balancing
- Resource optimization
- Performance monitoring

9.3.4.1.11. Usability Requirements

- Intuitive AR interactions
- Clear content organization
- Responsive web interface
- Cross-platform compatibility

9.3.5. Risk Management

9.3.5.1. Technical Challenges

9.3.5.1.1. AR Framework Selection

Initial Challenges

- Limited open-source AR solutions
- The8thwall's closed-source nature
- AR.js implementation difficulties
- Cross-platform compatibility issues

Solution

- Selected Unity + AR Foundation after evaluating WebXR and The8thwall
- Leveraged Unity's robust AR development tools
- Utilized AR Foundation's cross-platform capabilities
- AR Foundation is a cross-platform framework by nature

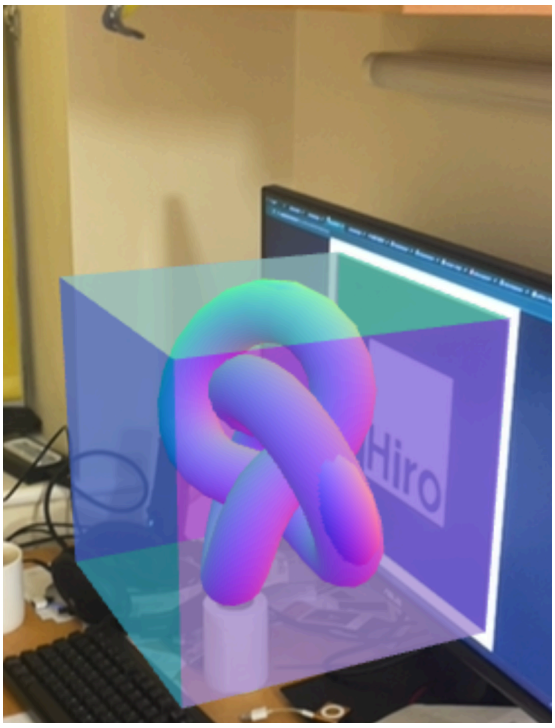


Figure 48: Maker based positioning in AR.js

Marker Tracking displays content when a marker is detected by the camera. While stable, markers are limited in shape, color and size. Common applications include augmented books, flyers and advertising.[40]

Marker Tracking is attempted to be used in the project, but it's requiring a pre-defined marker image, which is not suitable for the project.

Location Based AR uses real-world coordinates to display AR content on user devices. Users can move around outdoors and see AR content anchored to real locations, with content scaling based on distance. Applications include tourist guides, city exploration, point-of-interest discovery, and educational experiences like treasure hunts and situated art. [40]

Location Based AR is not applicable to the project, due to the low accuracy of the location.

Eventually, it's possible to use Cloud Anchor positioning in this project, explained in Section 3.6.

9.3.5.1.2. Algorithm Implementation

Complex Requirements

- Zero-knowledge proof integration
- External software dependencies
  - The watermarking module depends on a python executable.
- Mathematical complexity
  - In the mobile app development, complex 3D environment interactions require extensive knowledge of linear algebra and computer graphics
- Performance optimization
  - The watermarking module is a performance bottleneck

Solution

- Modular design approach
- Python-based watermarking
- Optimized proof generation
- Efficient verification process

9.3.5.1.3. Development and Debugging

Development Challenges

- Long Unity compilation times (~3 minutes)
  - Unity build is full package compilation by default
- Limited debugging capabilities
  - Hard to debug across devices
- Mobile deployment complexity
- Performance monitoring
- Requirement of Secure contexts
  - Some AR features in Browser are not available in non-secure contexts (non-HTTPS)

Solution

- Optimized build process (Incremental Build)
- Enhanced logging system (Logcat[41])
- Performance profiling[17]
- Use HTTPS proxy, e.g. `ngrok`[42]

9.3.5.2. Risk Mitigation Strategies

9.3.5.2.1. Technical Risk Management

- Regular technology evaluation
- Backup solution planning
- Performance optimization
- Cross-platform testing

9.3.5.2.2. Development Process

- Modular architecture
- Continuous integration
- Automated testing
- Performance monitoring

9.3.5.2.3. Quality Assurance

- Comprehensive testing
- User feedback collection
- Performance benchmarking
- Security auditing

[17] It turns out that the performance is good enough, so I didn't do anything about it.

Bibliography

[1] “Zero-knowledge proofs.” [Online]. Available: https://en.wikipedia.org/wiki/Zero-knowledge_proof

[2] “Unity Documentation.” [Online]. Available: https://docs.unity3d.com/

[3] “AR Foundation.” [Online]. Available: https://docs.unity3d.com/Manual/arfoundation.html

[4] “C#.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/

[5] “Google ARCore.” [Online]. Available: https://developers.google.com/ar

[6] “ARKit.” [Online]. Available: https://developer.apple.com/augmented-reality/arkit/

[7] “Nuxt.js.” [Online]. Available: https://nuxt.com/

[8] “TypeScript.” [Online]. Available: https://www.typescriptlang.org/

[9] “Tailwind CSS.” [Online]. Available: https://tailwindcss.com/

[10] “Vue.js.” [Online]. Available: https://vuejs.org/

[11] “VSCode.” [Online]. Available: https://code.visualstudio.com/

[12] “Chrome DevTools.” [Online]. Available: https://developer.chrome.com/docs/devtools

[13] “ESLint.” [Online]. Available: https://eslint.org/

[14] “Prettier.” [Online]. Available: https://prettier.io/

[15] “NestJS.” [Online]. Available: https://nestjs.com/

[16] “PostgreSQL.” [Online]. Available: https://www.postgresql.org/

[17] “PostGIS.” [Online]. Available: https://postgis.net/

[18] “Redis.” [Online]. Available: https://redis.io/

[19] “Cloudflare.” [Online]. Available: https://www.cloudflare.com/

[20] “AWS.” [Online]. Available: https://aws.amazon.com/

[21] “Google Cloud.” [Online]. Available: https://cloud.google.com/

[22] “Hedera Hashgraph.” [Online]. Available: https://www.hedera.com/

[23] “Postman.” [Online]. Available: https://www.postman.com/

[24] “pgAdmin.” [Online]. Available: https://www.pgadmin.org/

[25] “Git.” [Online]. Available: https://git-scm.com/

[26] “GitHub.” [Online]. Available: https://github.com/

[27] “Vercel.” [Online]. Available: https://vercel.com/

[28] “Heroku.” [Online]. Available: https://www.heroku.com/

[29] “Hey API OpenAPI TS.” [Online]. Available: https://github.com/hey-api/openapi-ts

[30] “TypeORM.” [Online]. Available: https://typeorm.io/

[31] “AWS S3.” [Online]. Available: https://aws.amazon.com/s3/

[32] “Google Cloud Anchor.” [Online]. Available: https://developers.google.com/ar/develop/cloud-anchors

[33] “GPS Accuracy.” [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/

[34] “Multipath Propagation.” [Online]. Available: https://en.wikipedia.org/wiki/Multipath_propagation

[35] “Perceptual hashing.” [Online]. Available: https://en.wikipedia.org/wiki/Perceptual_hashing

[36] “Block Mean Value Based Image Perceptual Hashing.”

[37] “Circom.” [Online]. Available: https://github.com/iden3/circom

[38] “Nest-Access-Control.” [Online]. Available: https://github.com/nestjs-community/access-control

[39] “View Transitions API.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/View_Transitions_API

[40] “AR.js.” [Online]. Available: https://ar-js-org.github.io/AR.js-Docs/

[41] “Logcat.” [Online]. Available: https://developer.android.com/studio/debug/logcat

[42] “ngrok.” [Online]. Available: https://ngrok.com/