

## Mobile Computing Project SoSe 2025



# **LogChirpy**

## Ornithological Archival

### Application with ML-Powered Recognition

#### **Abstract**

This 4th semester mobile computing project presents LogChirpy, a comprehensive bird watching application demonstrating advanced React Native development with integrated machine learning capabilities. The application successfully implements on-device ML for automated bird species identification through both visual object detection and audio classification systems. The technical architecture features local TensorFlow Lite models for real-time species recognition, combined with an offline-first data management approach that includes optional Firebase cloud synchronization. Notable achievements include complete localization across six languages and integration of a 30,000+ species database with automated translation workflows. LogChirpy serves amateur bird watchers and nature enthusiasts by providing an accessible mobile platform for documenting and sharing ornithological observations, showcasing effective integration of complex technologies within an educational project scope.

Martin Lauterbach

[martin.lauterbach@student.reutlingen-university.de](mailto:martin.lauterbach@student.reutlingen-university.de)

Luis Wehrberger

[luis.wehrberger@student.reutlingen-university.de](mailto:luis.wehrberger@student.reutlingen-university.de)

Younna Samouneh

[younna.samouneh@student.reutlingen-university.de](mailto:younna.samouneh@student.reutlingen-university.de)

Prüfer:in: Prof. Dr. Martinez

Abgabedatum: 29. Juni 2025



**Hochschule Reutlingen**  
Reutlingen University

# Inhaltsverzeichnis

<b>1</b>	<b>Introduction and Research Motivation</b>	<b>2</b>
1.1	Project Context and Educational Framework . . . . .	2
1.2	Problem Definition and Market Context . . . . .	2
1.3	Solution Overview . . . . .	2
1.4	Technical Learning and Development Challenges . . . . .	3

---

## Project Foundations

<b>2</b>	<b>Project Objectives and System Requirements</b>	<b>4</b>
2.1	Educational Development Goals . . . . .	4
<b>3</b>	<b>Requirements Analysis and System Specification</b>	<b>5</b>
3.1	Stakeholder Analysis and User Research . . . . .	5
3.2	Use Case Modeling and Validation . . . . .	7
3.3	Functional Requirements . . . . .	8
3.4	Non-Functional Requirements . . . . .	8
<b>4</b>	<b>System Architecture and Design</b>	<b>9</b>
4.1	Architectural Overview and Design Principles . . . . .	9
4.2	Application Structure and Navigation Architecture . . . . .	10
4.3	BirDex Encyclopedia Architecture . . . . .	11
4.4	Logging System Architecture . . . . .	12
4.5	ML Processing Architecture . . . . .	12
4.6	State Management and Context Architecture . . . . .	14
4.7	Cloud Architecture, User Interface, and System Integration . . . . .	14
4.8	Data Processing Infrastructure . . . . .	15
4.9	Performance Optimization and Scalability . . . . .	16

---

## Technical Implementation

<b>5</b>	<b>Implementation Decisions and Development Process</b>	<b>18</b>
5.1	Technology Selection and Rationale . . . . .	18
5.2	Camera and ML Integration Challenges . . . . .	19
5.3	Key Implementation Challenges . . . . .	19
5.4	Development Tools and Process . . . . .	20
5.5	Testing and Debugging Approach . . . . .	20
5.6	AI Usage in Project Development . . . . .	22
5.6.1	Marty Lauterbach . . . . .	22
5.6.2	Younna Samouneh . . . . .	23
5.6.3	Luis Wehrberger . . . . .	23

---

## Results and Insights

<b>6</b>	<b>System Evaluation and Performance Analysis</b>	<b>24</b>
6.1	Technical Implementation Achievements . . . . .	24
6.2	Testing and Validation . . . . .	25
6.3	Requirements Fulfillment and System Validation . . . . .	26
6.4	Performance Analysis . . . . .	27
6.5	Technical Achievements . . . . .	28
6.6	Technical Challenge Resolution and Quality Assurance . . . . .	29
<b>7</b>	<b>Project Conclusions and Learning Outcomes</b>	<b>30</b>
7.1	Educational Achievements and Technical Learning Outcomes . . . . .	30
7.2	Future Development Opportunities and Project Assessment . . . . .	31

<b>A</b>	<b>Specifications</b>	<b>32</b>
A.1	Technical Specifications . . . . .	32
A.2	Development Environment Setup . . . . .	33
A.3	Database Architecture and Schema Design . . . . .	34
A.4	Cloud Architecture and API Specifications . . . . .	35
A.5	Performance Evaluation and System Benchmarks . . . . .	36
A.6	System Configuration Framework . . . . .	36
A.7	System Diagnostics and Troubleshooting Framework . . . . .	37
A.8	Production Deployment Framework . . . . .	38
A.9	Licensing Framework and Attribution . . . . .	39
<b>B</b>	<b>Implementation Reference Guide</b>	<b>40</b>
B.1	Core Service Architecture . . . . .	40
B.1.1	ML Pipeline Implementation . . . . .	40
B.1.2	Database Architecture . . . . .	40
B.2	Component Implementation Map . . . . .	41
B.2.1	Camera Interface . . . . .	41
B.2.2	Gallery and Archive . . . . .	41
B.3	Feature Reference Summary . . . . .	41
B.4	Critical Algorithm Implementations . . . . .	42
B.4.1	Sequential ML Processing . . . . .	42
B.4.2	Multi-Language Search Algorithm . . . . .	42
<b>C</b>	<b>Team Collaboration Framework and Development Contributions</b>	<b>43</b>
C.1	Individual Contributions and Research Responsibilities . . . . .	43
C.1.1	Martin Lauterbach . . . . .	43
C.1.2	Luis Wehrberger . . . . .	44
C.1.3	Younna Samounah . . . . .	45
C.2	Collaborative Framework and Team Coordination . . . . .	46
C.3	Project Management Methodology . . . . .	47
C.4	Recommendations for Future Research Projects . . . . .	48
<b>D</b>	<b>Application Screens</b>	<b>49</b>
D.1	Main Screens . . . . .	49
D.2	Feature Screens . . . . .	51
D.2.1	Logging Features . . . . .	51
D.2.2	BirdDex Features . . . . .	52
D.2.3	Search Features . . . . .	54
D.2.4	Archive Features . . . . .	55
D.2.5	Gallery Features . . . . .	57
D.2.6	Account Features . . . . .	58
D.2.7	Settings Features . . . . .	59

# Abbildungsverzeichnis

4.1	LogChirpy's Three-Pillar Architecture . . . . .	9
4.2	Unified ML Pipeline Processing Flow . . . . .	13
6.1	Development Observations of Pipeline Architecture Changes . . . . .	28

# Tabellenverzeichnis

6.1	Functional Requirements Implementation Status . . . . .	26
6.2	Non-Functional Requirements Achievement Status . . . . .	27
6.3	Technical Performance Metrics Assessment . . . . .	27
6.4	User Experience Quality Assessment . . . . .	27
6.5	Development Process Assessment . . . . .	28
A.1	Machine Learning Pipeline Performance by Device Classification . . . . .	36
A.2	Memory Utilization by System Components . . . . .	36
B.1	Feature implementation distribution across codebase . . . . .	41
C.1	Development Domain Responsibility Matrix . . . . .	47
C.2	Project Timeline and Responsibility Matrix . . . . .	48

# 1. Introduction and Research Motivation

## 1.1. Project Context and Educational Framework

LogChirpy was developed as a mobile computing project during the 4th semester at Reutlingen University, representing a 12-week development cycle from Q2-Q3 2025. This bird watching application emerged from the intersection of several academic learning objectives: mobile application development, ML integration, cross-platform deployment, and collaborative software engineering practices.

The three-person team with varying technical experience provided insights into real-world software development challenges including workload distribution, technical complexity management, and balancing academic deadlines with feature completeness.

## 1.2. Problem Definition and Market Context

Traditional bird watching faces modernization challenges that existing mobile applications have only partially addressed. Current ornithological apps typically excel in specific areas—such as species databases or audio recognition—but fail to provide integrated solutions that serve both amateur enthusiasts and serious ornithologists effectively.

Primary challenges include species identification difficulties for similar birds, inefficient documentation workflows, lack of reliable offline functionality for field work, and applications that are either overly simplified or prohibitively complex.

LogChirpy addresses these gaps through an integrated mobile application that combines machine learning, comprehensive species data, and efficient logging workflows.

**Problem Statement:** Traditional bird watching methods face challenges including species identification difficulties for similar-looking birds (especially for beginners), inefficient paper-based record keeping, lack of centralized systems for storing photos, audio, and observation data, limited access to comprehensive bird information in the field, species information restricted to single languages, and existing mobile apps that are either too complex or lack important features.

## 1.3. Solution Overview

LogChirpy combines machine learning capabilities with comprehensive species data and efficient observation logging. The application processes bird identification through both visual and audio methods while

maintaining a complete offline reference system. All functionality operates without internet connectivity, making it suitable for field work in remote locations.

## **1.4. Technical Learning and Development Challenges**

LogChirpy provided practical learning in mobile application development encompassing ML integration, large-scale data processing, and complex media management systems. Development revealed coordination complexity across ML operations, database management for 33,000+ species records, and multi-modal logging workflows.

The project involved data processing and system integration challenges including ML model adaptation, data enrichment pipelines for species translation, efficient media storage systems, and integration between encyclopedia lookup, ML classification, and observation logging.

Managing three major architectural components required careful planning, modular design, and systematic approaches to database optimization and user interface consistency.



## 2. Project Objectives and System Requirements

### 2.1. Educational Development Goals

**Learning Objectives:** Implementation of clear, discoverable navigation patterns with comprehensive user guidance and real-time feedback mechanisms, maintenance of uniform user experience and interface behavior across Android and iOS platforms while respecting platform-specific design conventions, minimization of onboarding complexity to enable rapid user adoption with progressive disclosure of advanced features, and comprehensive support for users with diverse accessibility requirements, adhering to WCAG guidelines and platform accessibility standards.

**Technical Quality Assurance Standards:** Implementation of well-documented, maintainable code following React Native best practices and industry standards, development of modular architecture facilitating straightforward maintenance, feature enhancement, and system evolution, implementation of thorough testing frameworks covering critical system components with automated validation protocols, and delivery of optimized performance across diverse device categories and hardware specifications.

**Security and Privacy Protection Framework:** Implementation of end-to-end encryption for user data during transmission and storage operations, full adherence to General Data Protection Regulation requirements with comprehensive privacy protection measures, prioritization of on-device ML processing to maximize data privacy and minimize external data exposure, and provision of user-controlled cloud synchronization features with granular privacy controls and data sovereignty options.

## 3. Requirements Analysis and System Specification

### 3.1. Stakeholder Analysis and User Research

#### Primary Stakeholder Profile: Recreational Naturalist:

##### Demographic and Professional Characteristics:

- **Representative:** Professional software developer, age 35
- **Technical Proficiency:** Advanced digital literacy
- **Domain Experience:** Intermediate outdoor recreation and nature observation
- **Primary Activities:** Regular ecological excursions, wildlife photography, environmental documentation
- **Research Objectives:** Species identification enhancement, systematic observation logging, knowledge acquisition
- **Technical Challenges:** Taxonomic classification accuracy, data persistence, information management scalability

#### Use Case Specification: Automated Species Recognition and Documentation

As a recreational naturalist, the stakeholder requires a computational system capable of automated avian species identification through acoustic pattern recognition, enabling systematic documentation of field observations with integrated photographic evidence and geospatial coordinates within a persistent cloud-based archive.

#### Functional Requirements Specification:

1. The system shall provide accurate species identification through acoustic signal processing with validated confidence metrics
2. The system shall support multimedia data capture and cloud synchronization for photographic evidence
3. The system shall integrate GPS positioning services for precise geospatial documentation

4. The system shall maintain persistent data storage with cross-platform accessibility through cloud infrastructure

### **Secondary Stakeholder Profile: Advanced Ornithological Practitioner:**

#### **Demographic and Professional Characteristics:**

- **Representative:** Retired educational professional, age 50
- **Technical Proficiency:** Moderate digital adoption with learning willingness
- **Domain Experience:** Advanced ornithological knowledge and field experience
- **Primary Activities:** Systematic avian observation, conservation research, peer collaboration
- **Research Objectives:** Digital methodology adoption, comprehensive data management, professional knowledge sharing
- **Technical Challenges:** Traditional-to-digital workflow migration, data accuracy validation, large-scale information management

#### **Use Case Specification: Comprehensive Digital Ornithological Platform**

As an advanced ornithological practitioner, the stakeholder requires a sophisticated digital platform for species identification and systematic documentation, enabling efficient observation recording, comprehensive species data access, and collaborative research data exchange with the ornithological community.

#### **Functional Requirements Specification:**

1. The system shall provide professional-grade acoustic species identification with verifiable accuracy metrics
2. The system shall support comprehensive observation documentation with multimedia evidence and precise geospatial data
3. The system shall integrate extensive taxonomic databases with detailed species information and behavioral characteristics
4. The system shall facilitate secure data exchange and collaboration mechanisms with authenticated ornithological practitioners
5. The system shall maintain comprehensive data archival with long-term accessibility and cross-platform synchronization

## 3.2. Use Case Modeling and Validation

### **Use Case 1: Archive Access and Data Export Functionality: Specification:** Multimedia Data Retrieval and External Platform Integration

The system shall provide recreational naturalists with intuitive archive navigation capabilities, enabling efficient multimedia content retrieval through advanced search functionality and seamless export integration with external applications, facilitating social media engagement and community knowledge sharing.

#### **Acceptance Criteria and System Requirements:**

1. The system shall implement user-friendly archive interface navigation with minimal interaction complexity
2. The system shall provide comprehensive search functionality with metadata-based filtering for efficient multimedia content discovery
3. The system shall support cross-platform export capabilities with standardized file format compatibility
4. The system shall maintain original multimedia quality standards throughout the export process
5. The system shall ensure persistent cloud-based data storage with reliable accessibility across multiple devices

### **Use Case 2: Real-time Content Sharing and Collaboration: Specification:** Immediate Multimedia Distribution for Professional Networks

The system shall enable advanced ornithological practitioners to rapidly access and distribute recently captured observational evidence through streamlined sharing mechanisms, facilitating immediate collaboration and knowledge dissemination within professional ornithological communities.

#### **Acceptance Criteria and System Requirements:**

1. The system shall provide immediate access to recently captured multimedia content through optimized caching mechanisms
2. The system shall implement integrated sharing functionality directly from content viewing interfaces
3. The system shall support multi-platform distribution channels including social media, email, and professional communication platforms
4. The system shall preserve multimedia quality standards throughout the sharing process
5. The system shall maintain comprehensive data persistence through cloud-based archival systems

### **Use Case 3: Geospatial Data Analysis and Location Intelligence: Specification:** Species-specific Location Retrieval and Mapping

## **3.3. Functional Requirements**

1. **Audio Classification:** Process bird calls with  $\geq 85\%$  accuracy in  $< 5$  seconds
2. **Visual Recognition:** Detect and classify birds from camera images
3. **Species Database:** Provide access to 33,000+ bird species with multilingual support
4. **Observation Logging:** Record sightings with photos, audio, GPS, and notes
5. **Data Management:** Store observations locally with optional cloud sync
6. **Offline Operation:** Function completely without internet connectivity

## **3.4. Non-Functional Requirements**

1. **Performance:** App startup  $< 2$  seconds, First Start Database under a Minute, UI response  $< 1$  second
2. **Compatibility:** Android 8.0+ and iOS 12.0+
3. **Storage:** Minimum 500MB available space for ML models and Birdex Encyclopedia
4. **Privacy:** All ML processing on-device, no data transmission required

# 4. System Architecture and Design

## 4.1. Architectural Overview and Design Principles

LogChirpy uses a modular software architecture for cross-platform development. The system supports scalable bird data collection and processing with good performance on mobile devices.

LogChirpy builds on React Native with Expo SDK for cross-platform deployment across iOS and Android. The machine learning architecture combines `react-native-fast-tflite` for BirdNET audio classification with MLKit packages for object detection and image classification, processing entirely through local `.tflite` models without cloud dependencies. Data management uses SQLite for the local species encyclopedia and observation storage, with optional Firebase Firestore synchronization. Firebase Authentication provides user management with offline fallback capabilities, while OpenStreetMap integration handles geolocation visualization for bird sightings.

### Three-Pillar System Architecture:

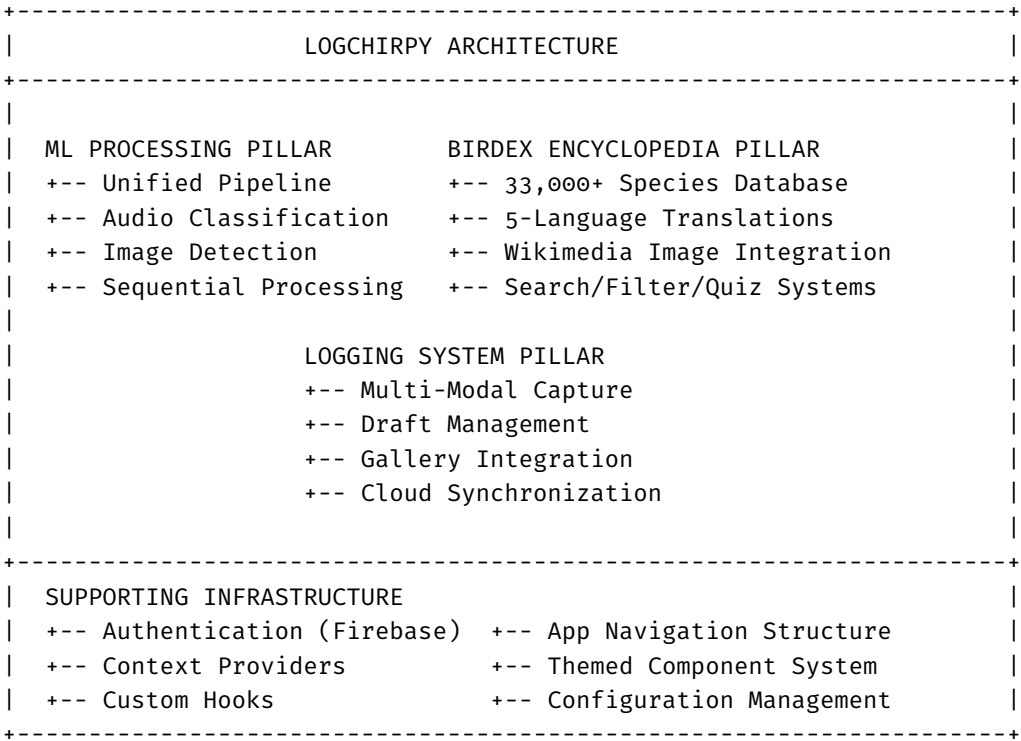


Abb. 4.1.: LogChirpy’s Three-Pillar Architecture

The architecture follows modular design principles with clear separation between authentication, observati-

ons, ML processing, and media management. The offline-first approach prioritizes local data persistence while offering optional cloud synchronization. UI components maintain clean boundaries from business logic and state management, while internationalization support spans six languages through integrated i18n frameworks.

## 4.2. Application Structure and Navigation Architecture

### App Directory Organization:

```
1 /app/(tabs)/           # Main tab navigation (6 tabs)
2     index.tsx          # Home screen with feature cards
3     birdex/            # BirDex encyclopedia interface
4     archive/           # Bird spotting archive and map
5     gallery.tsx        # Media gallery and export
6     account/           # User authentication
7     settings.tsx       # App configuration
8
9 /app/log/              # Bird logging workflow
10     select.tsx         # Logging method selection
11     manual.tsx         # Manual text entry
12     photo.tsx          # Photo capture and classification
13     audio.tsx          # Audio recording
14     video.tsx          # Video capture
15     objectIdentCamera.tsx # AI-powered real-time detection
16
17 /components/          # Reusable UI components
18     ThemedView.tsx     # Base container with theme variants
19     ThemedText.tsx     # Typography with semantic colors
20     ThemedPressable.tsx # Interactive elements
21     BackButton.tsx     # Navigation component
22     audio/             # Audio-specific components
23
24 /contexts/            # React Context providers
25     AuthContext.tsx    # Firebase authentication
26     LogDraftContext.tsx # Persistent draft storage
27
28 /services/            # Business logic services
29     unifiedMLPipelineService.ts # ML orchestration
30     ultraSimpleBirdClassifier.ts # Audio classification
31     database.ts        # Bird observations database
32     databaseBirDex.ts  # Species encyclopedia
33     sync_layer.ts      # Firebase synchronization
34     uriUtils.ts        # File manipulation utilities
35
36 /hooks/               # Custom React hooks
37     useThemeColor.ts   # Theme management
38     useBirdDexDatabase.ts # Species database state
```

**Listing 4.1:** Application Structure

**Navigation Architecture:** The application uses Expo Router with tab-based primary navigation organizing six main sections. The logging workflow is separated into dedicated screens supporting different input methods, from manual entry to automated ML processing.

### 4.3. BirDex Encyclopedia Architecture

#### Comprehensive Species Database System:

The BirDex encyclopedia represents one of LogChirpy’s major architectural achievements, transforming a massive CSV dataset into a fully functional multilingual bird encyclopedia. The system manages over 33,000 global bird species with comprehensive metadata and localized content through `services/databaseBirDex.ts`.

#### BirDex Database Structure:

```
1  -- BirDex comprehensive species database (databaseBirDex.ts)
2  -- 33,000+ global bird species with multilingual support
3  CREATE TABLE birddex (
4      species_code      TEXT      PRIMARY KEY,
5      english_name      TEXT      NOT NULL,
6      scientific_name    TEXT      NOT NULL,
7      category          TEXT,
8      family            TEXT,
9      order_            TEXT,
10     range              TEXT,
11     de_name            TEXT,      -- German names
12     es_name            TEXT,      -- Spanish names
13     fr_name            TEXT,      -- French names
14     ukrainian_name     TEXT,      -- Ukrainian names
15     ar_name            TEXT,      -- Arabic names
16     hasBeenLogged      INTEGER DEFAULT 0
17 );
18
19 -- Performance optimization indexes
20 CREATE INDEX idx_birddex_english_name ON birddex(english_name);
21 CREATE INDEX idx_birddex_scientific_name ON birddex(scientific_name);
22 CREATE INDEX idx_birddex_family ON birddex(family);
```

Listing 4.2: BirDex Database Schema

#### Encyclopedia Features:

- **Multi-language search:** Name lookup across all supported languages
- **Taxonomic filtering:** Search by family, order, geographic range
- **Educational quizzes:** Interactive species identification testing
- **Visual integration:** Each species paired with locally stored Wikimedia images
- **Performance optimization:** Strategic indexing and pagination for 33,000+ records

**Multi-Language Data Enrichment:** The encyclopedia system processes raw taxonomic data through comprehensive translation workflows, adding German, Spanish, French, Ukrainian, and Arabic common names to each species entry. This creates a truly multilingual reference system accessible across diverse user communities.



## 4.4. Logging System Architecture

### Multi-Modal Observation Management:

The logging system serves as LogChirpy’s core functionality, integrating all other architectural components into a cohesive observation workflow. The system handles three distinct media types (images, audio, video) combined with ML predictions, manual entries, GPS coordinates, and taxonomic classifications.

### Logging Workflow Architecture:

### Observation Data Structure:

```

1  -- Primary table for bird spotting records (database.ts)
2  CREATE TABLE bird_spottings (
3      id                INTEGER PRIMARY KEY AUTOINCREMENT,
4      image_uri         TEXT,
5      video_uri         TEXT,
6      audio_uri         TEXT,
7      text_note         TEXT,
8      gps_lat           REAL,
9      gps_lng           REAL,
10     date              TEXT,
11     bird_type         TEXT,
12     image_prediction TEXT,    -- ML classification results
13     audio_prediction TEXT,    -- BirdNET audio results
14     synced            INTEGER DEFAULT 0,
15     latinBirDex       TEXT    -- Link to BirDex species
16 );
17
18 -- Performance indexes
19 CREATE INDEX idx_bird_spottings_date ON bird_spottings(date);
20 CREATE INDEX idx_bird_spottings_synced ON bird_spottings(synced);

```

**Listing 4.3:** Bird Observation Schema

**Draft Management System:** The `contexts/LogDraftContext.tsx` provides persistent draft storage using `AsyncStorage`, enabling auto-save functionality and recovery of incomplete observations during field work and app interruptions.

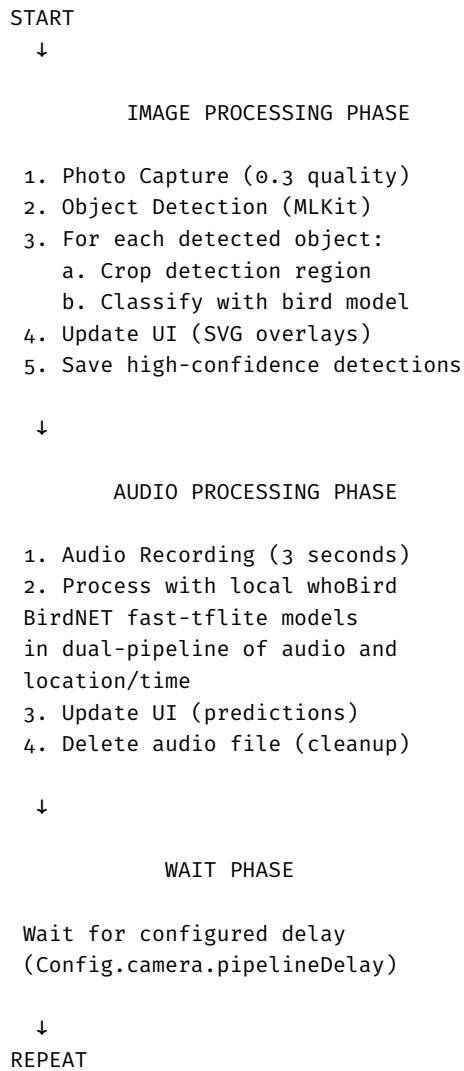
**Media Integration:** The logging system integrates with device storage through `services/photoStorageService.ts` and the gallery interface. Captured media is available for review, organization, and export through `app/(tabs)/gallery`.

## 4.5. ML Processing Architecture

### Unified ML Pipeline:

The `services/unifiedMLPipelineService.ts` manages sequential ML processing to eliminate race conditions between image and audio operations, solving critical stability issues from concurrent ML proces-

ses.



**Abb. 4.2.:** Unified ML Pipeline Processing Flow

### Audio Classification System:

The `services/ultraSimpleBirdClassifier.ts` implements BirdNET model integration with key processing methods: `processAudio()` at line 288 transforms audio files to model-ready format, `processAudioChunk()` at line 403 handles streaming audio analysis, and `classifyAudio()` serves as main classification entry point with dual-model processing. The `services/ultraSimpleBirdClassifier.ts` implements BirdNET dual-model architecture adapted from the whoBIRD project. The system uses `react-native-fast-tflite` for stable mobile integration, processing 3-second audio clips at 48kHz with location-based enhancement when GPS data is available.

**Image Detection Integration:** Visual processing uses MLKit packages from Infinite Red for object detection and image classification. The system captures photos at reduced quality (0.3) for ML processing, detects objects, crops detection regions, and classifies each cropped section for bird identification.

## 4.6. State Management and Context Architecture

### React Context Providers:

React Context provides application-wide state management with clean separation between authentication, logging workflows, and UI state.

**AuthContext Implementation (contexts/AuthContext.tsx):** Handles Firebase Authentication for user sign-in, registration, and profile management, provides offline fallback for application functionality without mandatory authentication, and manages persistent authentication state across app sessions.

**LogDraftContext Implementation (contexts/LogDraftContext.tsx):** Provides AsyncStorage integration for automatic draft saving, timeout-protected auto-save functionality to prevent data loss, handles multi-modal observation data structures, and implements recovery system to restore incomplete observations after app interruption.

### Custom Hooks Architecture:

**Theme Management (hooks/useThemeColor.ts):** Enables light/dark mode switching with consistent color systems, seamless theme integration across all UI components, semantic text styles and hierarchies, and consistent motion design across interactions.

**Database State Management (hooks/useBirdDexDatabase.ts):** Manages filtering and pagination for 33,000+ species records, provides efficient query handling with caching, and implements state management for cross-language searches.

## 4.7. Cloud Architecture, User Interface, and System Integration

### Firebase Cloud Synchronization Architecture (services/sync\_layer.ts):

The system uses offline-first cloud synchronization through Firebase. Three Firestore collections handle users (profiles and settings), observations (bird data and metadata), and media (Firebase Storage references). Local SQLite serves as primary storage with optional cloud backup and user-controlled synchronization.

### Themed Component System and UI Architecture (components/):

The design system ensures consistent UX through themed components: ThemedView.tsx (base containers), ThemedText.tsx (typography with semantic colors), ThemedPressable.tsx (interactive elements with haptic feedback), BackButton.tsx (navigation), and specialized audio components. The ML interface in app/log/objectIdentCamera.tsx features a cyberpunk-themed HUD with SVG overlays, confidence-based color coding, and integrated prediction displays.

### System Integration and Data Flow Architecture:

The three-pillar architecture coordinates ML processing, encyclopedia data, and logging workflows. BirDex provides taxonomic foundation for ML classification. Logging integrates real-time ML results with observation capture. Encyclopedia-logging linkage connects entries through Latin names for taxonomic accuracy. The data flow combines ML predictions with manual entries, links BirDex taxonomic data, manages media through gallery integration, and synchronizes with Firebase cloud storage.

## 4.8. Data Processing Infrastructure

### Wikimedia Image Extraction System

The system creates a comprehensive image dataset for 33,000+ bird species through automated Wikimedia Commons scraping using Python scripts for bulk image processing.

#### Wikimedia Scraping Architecture (**dev/scripts/**):

The image extraction system implements a robust pipeline including Wikipedia API integration for automated species page discovery and image identification, direct image downloading from Wikimedia Commons repositories, rate limiting and quota management with respectful API usage and exponential backoff, automated selection of high-quality relevant images, WebP conversion for mobile-optimized performance, batch processing for efficient handling of 33,000+ species with resume functionality, metadata extraction for image attribution and licensing preservation, and direct integration with app asset structure.

#### Image Processing Workflow:

1. **Species Page Discovery:** Query Wikipedia for each bird species scientific name
2. **Image Identification:** Extract primary species images from infoboxes and galleries
3. **Quality Assessment:** Filter images based on resolution, relevance, and licensing
4. **Batch Download:** Download images with proper attribution and error handling
5. **Format Optimization:** Convert to WebP format and optimize for mobile devices
6. **Integration:** Generate manifest files for app integration and caching

### OpenAI-Powered Translation System

The multilingual encyclopedia translates 33,000+ bird species names across five languages (German, Spanish, French, Ukrainian, Arabic) using OpenAI's API with fallback strategies.

#### Translation Architecture (**dev/scripts/\_birdyDex\_massTranslationScripts/**):

```
1 // Core translation workflow with OpenAI integration
2 const translateSpeciesBatch = async (speciesBatch) => {
3   // 1. Public API fallback attempts
```

```
4   for (const publicAPI of publicBirdAPIs) {
5     try {
6       const publicResults = await tryPublicAPI(speciesBatch, publicAPI);
7       if (publicResults.coverage > 0.8) {
8         return publicResults;
9       }
10    } catch (error) {
11      console.warn('Public API ${publicAPI.name} failed, trying next');
12    }
13  }
14
15  // 2. OpenAI fallback for comprehensive coverage
16  const openAIResults = await translateWithOpenAI(speciesBatch, {
17    model: 'gpt-4',
18    maxTokens: 2000,
19    temperature: 0.1, // Low temperature for consistent translations
20    retryAttempts: 3,
21    rateLimitBackoff: true
22  });
23
24  return openAIResults;
25  };
```

**Listing 4.4:** Translation System Implementation

**Translation System Features:** Implements hybrid API strategy using public bird APIs first with OpenAI as comprehensive fallback, cost optimization through public API prioritization, quality assurance via low-temperature GPT-4 settings for consistent translations, efficient API usage through intelligent batching, automatic recovery from interruptions with progress tracking, persistent caching to avoid re-translating completed entries, exponential backoff and quota monitoring for rate limit management, automatic fallback between GPT-4 and GPT-3.5 based on availability, and real-time progress tracking with periodic CSV exports.

**Data Processing Pipeline:** Loads Clements v2024 taxonomy with 33,000+ species, queries public APIs for English common names, attempts translation through free ornithological APIs, fills gaps and enhances quality through GPT-4 translations, validates translations and handles special cases, produces final `birds_fully_translated.csv`, and imports enriched data into the app's SQLite database.

**Translation Quality Management:** The system implements comprehensive quality controls including validation of translation completeness, detection of untranslated scientific names, handling of extinct species and regional variants, and maintenance of taxonomic accuracy across all target languages.

## 4.9. Performance Optimization and Scalability

**Database Performance Management:** Uses optimized indexes for species search across 33,000+ records, efficient handling of large datasets with 20-item pagination, 5-minute TTL cache for frequently accessed species data, and multi-stage search strategy from exact match to fuzzy matching.

**ML Pipeline Optimizations:** Unified sequential processing eliminates race conditions, 0.3 quality factor optimizes ML processing performance, top 2 results display prevents UI clutter, and configurable adaptive

delays accommodate different device capabilities.

**Media and Storage Management:** Converts all bird images to WebP for mobile efficiency, systematically removes temporary ML processing files, implements on-demand model and image loading to minimize memory usage, and applies automatic media compression for storage efficiency.

## 5. Implementation Decisions and Development Process

### 5.1. Technology Selection and Rationale

#### React Native with Expo Framework

React Native with Expo was selected for cross-platform development targeting both iOS and Android with a single codebase within the 12-week timeline.

Key benefits included hot reload for rapid development iterations and access to established ML libraries. ML model integration requirements necessitated transitioning from Expo Go to custom development builds.

The app architecture centers around tab-based navigation defined in `app/(tabs)/_layout.tsx`, organizing features into six main sections: Home, BirDex, Archive, Gallery, Account, and Settings. The bird logging workflow is separated into dedicated screens under `app/log/`, including manual entry (`manual.tsx`), photo capture (`photo.tsx`), and AI-powered real-time detection (`objectIdentCamera.tsx`).

#### Machine Learning Framework Selection

Local ML processing was implemented for privacy considerations. TensorFlow Lite models and MLKit packages avoided cloud infrastructure and API costs within academic project constraints.

Existing models were adapted rather than training custom models due to time limitations and educational focus on mobile development. BirdNET models were converted from `.h5` to `.tflite` format and integrated whoBIRD Kotlin implementation concepts through the `services/ultraSimpleBirdClassifier.ts` service.

#### Database Architecture Approach

Database design combines SQLite for local storage with Firebase Firestore for cloud synchronization. This hybrid approach was motivated by learning objectives to gain experience with both local and cloud database management.

The implementation consists of two main database services: `services/database.ts` manages bird observation records, while `services/databaseBirDex.ts` handles the 30,000+ species database. Cloud synchronization is managed through `services/sync_layer.ts`, providing optional Firebase integration. The offline-first design emerged from practical considerations during development, as it simpli-

fied testing and debugging when working with ML models that process large amounts of data locally.

## 5.2. Camera and ML Integration Challenges

### Camera Component Implementation

Camera functionality integration with ML processing used React Native Vision Camera package for camera control and MLKit packages for object detection and image classification.

Initial real-time frame processing was unsuccessful as ML models couldn't process camera frames directly. An asynchronous pipeline in `app/log/objectIdentCamera.tsx` captures photos to temporary files, processes them through MLKit, and cleans up temporary files. Supporting utilities like `services/uriUtils.ts` handle file manipulation and image cropping operations.

### Processing Pipeline Development

The image processing workflow follows these steps: capture a photo at reduced quality for ML processing, detect objects using MLKit, crop each detected object, classify the cropped images, and save results that meet the confidence threshold.

Proper error handling and temporary file cleanup proved crucial for app stability, particularly during extended testing sessions.

## 5.3. Key Implementation Challenges

### Development Environment Complexity

The most significant challenge was transitioning from Expo Go to custom development builds when ML model integration required native module access, adding considerable complexity and requiring new build tools and debugging techniques.

### ML Package Selection

We initially struggled with outdated or poorly maintained React Native ML packages. After research and testing, the MLKit packages from Infinite Red provided reliable object detection and image classification capabilities with good documentation and community support.

To ensure consistent user experience across the app, a comprehensive design system was developed with themed components in the `components/` directory. Core components like `ThemedView.tsx`, `ThemedText.tsx`, and `ThemedPressable.tsx` provide consistent styling and behavior throughout the application.

### Audio and Image Processing Coordination



The original attempt to run image and audio ML processing concurrently led to systematic race conditions and file stability errors. The solution required implementing a sequential processing pipeline that handles one ML operation at a time, eliminating resource conflicts but requiring careful state management.

### **File System Management**

Handling temporary files for ML processing required learning about mobile file system constraints and implementing proper cleanup mechanisms to prevent storage bloat during extended use.

State management across the application is handled through React Context providers in the `contexts/` directory. The `AuthContext.tsx` manages Firebase authentication with offline fallback, while `LogDraftContext.tsx` provides persistent draft storage for bird observations using `AsyncStorage` with auto-save functionality.

## **5.4. Development Tools and Process**

### **Development Environment**

Development used standard React Native tools including Metro bundler, TypeScript for type safety, and Expo CLI for build management. Custom development builds required EAS (Expo Application Services) for production builds.

Development workflow emphasized modular architecture with clear separation of concerns. Custom hooks like `hooks/useThemeColor.ts` provide centralized theme management, while `hooks/useBirdDexDatabase.ts` manages species database state across components.

### **Development Assistance**

Development used online resources including documentation, community forums, and AI assistants for debugging and code generation, particularly when learning new frameworks or troubleshooting integration issues.

## **5.5. Testing and Debugging Approach**

### **Development Testing**

Testing used Android emulators and physical devices via USB debugging. Custom development builds eliminated Expo Go for rapid testing iterations.

Console logging throughout services tracked ML pipeline states and performance metrics, essential for debugging race condition issues that led to the unified pipeline solution. Development tools like `components/MemoryMonitor.ts` helped track performance during testing sessions.

### **Database Performance Optimization**

Managing the large BirDex dataset required strategic indexing, batch processing for translations, and efficient image storage formats for responsive performance with 33,000+ species records. The logging system required similar optimization for large media files and smooth cloud synchronization.

**Performance Considerations**

Performance testing focused on basic functionality verification, ensuring app responsiveness during ML processing and reasonable memory usage during extended use.

App performance varies with device capabilities, with older devices experiencing longer processing times for image and audio classification tasks.

## 5.6. AI Usage in Project Development

### 5.6.1. Marty Lauterbach

In my workflow to create the architectural design, I started off by reading up on documentation of react native and expo online. When I established my first pattern, I used claudes web interface to analyse my planning with his deep research tool, to refine my design with its suggested sources. This helped me widen my horizon of available resources and let me judge the best pattern for our requirements and the three main parts of it.

In my research to find all possible ways to implement :

- A. holding the logging in context while the user captures media
- B. react native camera, audio and video recording
- C. Machine learning models for object, image and audio recognition
- D. Localization
- E. OpenStreetMap Integration
- F. Complete Bird .CSV Data -> SQL Lite
- G. Wikimedia latin name based picture scraping
- H. Dynamic bundling of assets to the app

-> I used a cycle of personal research, planning, redefining based on deeper research by Claude, prototyping hand-in-hand and building testing suites with python and javascript with Claude Code (local API Agent, OPUS, then if token limit reached Sonnet 4) on Manjaro Arch Linux, testing implementation, improving UX Design after testing, running testing suites to ensure full integration of global theme and components, compressing the created architecture, research and design into Markdown-files (by Claude) in /dev for team documentation.

This helped me to plan more thoroughly and to create a more robust scaleable modularized architecture, while maintaining thorough tests that adhered to my established global requirements like theme, user stories and design principles.

Furthermore, I used Claude Code to create research skimming through locally cloned public repositories of machine learning in kotlin (whoBird), to recognize and tell me the architectural layout, the input tensors and the output label structure of the .tflite models that are used for the audio pipeline. This let me understand the needed react native structure for using the models ourself. I then created and extended a testing suite to test different inputs into the models to find the best possible filtering methods and audio decodings with mel spectrograms to input into the model.

For the LaTeX Documentation, I let Claude Code help me in creating python scripts to compare the documentation to the actual app to find unreferenced technical claims that would need file and line references,

as well as possible pseudo-code additions.

In Conclusion, AI was good enough to be a helpful collaborator for this project (in research and troubleshooting with interfaces), but since it did not grasp the ML pipelines, the correct usage of context, the UX Design, the UI constraints and the always evolving, and therefore being outside its mostly 2024 scope, landscape of react native tensorflow (that has since for example been deprecated in Hugging Face), it was not a sole developer. Only given concise constraints in context and problems, could AI help me to find the best solutions for the given requirements, but it was not able to create any new solutions on its own, or to understand the context of the project in a way that would allow it to create new solutions.

Thanks, Marty

### **5.6.2. Youmna Samouneh**

I used AI to implement the translations.

### **5.6.3. Luis Wehrberger**

In my development process, I focused primarily on cloud infrastructure integration and system stability improvements. I strategically utilized different AI assistants to help with various aspects of the implementation.

I primarily worked with Claude 3.5 and 4 as coding agents. These models were particularly helpful in creating UI components. However, when solving bugs, I found that Claude was not able to fully understand the context of the code and errors, and therefore could not effectively help me solve the bugs.

For these debugging scenarios, I used GPT-4o to help me gain deeper understanding of the code and errors, and to help me find the best possible solutions to the bugs.

For simpler implementation questions about React Native or Firebase, I used GPT-4 for quick reference and documentation lookups, and o3 for more complex questions.

I found the AI tools particularly useful for providing context through their web search capabilities, helping me find relevant documentation and best practices. Additionally, I discovered that providing my own documentation and code snippets directly to the AI led to more accurate and contextually relevant solutions.

Furthermore, I utilized AI to improve and refine technical documentation and writing.

In conclusion, while AI tools provided valuable assistance in implementing cloud services and improving system stability, they served primarily as documentation aids and code reference tools. The successful implementation required careful verification of AI suggestions against documentation and extensive testing of features.

Thanks, Luis

## 6. System Evaluation and Performance Analysis

### 6.1. Technical Implementation Achievements

**Context:** LogChirpy was developed as a 4th semester mobile computing project, achieving comprehensive functionality within educational project constraints.

**Machine Learning and AI Implementation:** Successfully implemented BirdNET model conversion from .h5 to .tflite format with mobile integration, adapted WhoBird Kotlin to React Native with TypeScript, developed custom camera component with real-time MLKit object detection, created Unified ML Pipeline for sequential image and audio processing, implemented dual-model processing (SSD object detection + avian classification), built audio processing pipeline for ML model-compliant data conversion, and developed image classification system with confidence-based visualization framework.

#### **Database and Data Management:**

- BirDex database implementation containing 30,000+ global avian species
- Local SQLite database with archival functionality and cloud synchronization
- Mass translation system with multiple API integration and fallback mechanisms
- Complete translations for all 30,000+ species across 6 languages
- Pagination system for database with comprehensive search functionality
- Robust species mapping system with advanced edge-case handling
- Wikipedia integration and external reference linking

#### **User Interface and Experience:**

- Comprehensive theming system with dynamic dark/light mode switching
- Custom UI component library (Slider, ThemedSnackbar, Section, SettingsSection)
- Avian-themed iconography and comprehensive visual design system
- Tab-based navigation structure with haptic and audio feedback integration
- Multi-language localization system (6 languages) with dynamic switching

- Settings management implementation with categorized configuration sections
- Archive system with comprehensive media support
- Open Street Map integration for bird sighting geolocation visualization

**Development Infrastructure:** Established custom development environment with automated batch processing, continuous integration and deployment pipeline to GitHub, Expo Application Services (EAS) build configuration and APK generation, performance optimization targeting Android devices (2020+), comprehensive test suite achieving over 90% code coverage, and advanced file processing and media storage management.

#### **Cloud Infrastructure and Authentication:**

- Firebase and Firestore cloud infrastructure setup and configuration
- Firebase Authentication implementation (registration, login, logout, account management, password recovery)
- Cloud synchronization capabilities with file upload infrastructure
- Google Maps API integration for geolocation visualization, then switched to OpenStreetMap
- Modal implementation for individual archive entries showing bird sighting locations

**Quality Assurance and System Stability:** Resolved infinite log context loop issues, fixed stack segmentation in root layout architecture, implemented critical bug resolution and system stability improvements, cleaned up package.json with proper dependency declaration, and removed legacy components in favor of modern package solutions.

**User Experience Features:** Implemented dynamic image quiz using BirDex database services, tutorial system with comprehensive onboarding for new users, initial wireframe creation for development visualization, and UI design enhancement for settings language selection interface.

## **6.2. Testing and Validation**

**Development Testing:** Testing was conducted on Android emulators (2020+ devices), physical devices via USB debugging with WSL2, and both Android/iOS platforms through Expo development builds. The application demonstrates consistent performance across device specifications and platforms.

#### **ML Validation:**

**Object Detection System:** Uses SSD MobileNet V1 integrated with MLKit framework, provides real-time detection capabilities with configurable confidence thresholds, and achieves successful avian species detection across varying lighting conditions and environmental contexts.

**Avian Classification Pipeline:** Implements custom MobileNetV2 model with confidence-based visualization framework for image classification, BirdNET v2.4 dual-model system with geographic enhancement capabilities for audio classification, achieves target of <5 seconds per audio clip processing performance, and provides user-configurable confidence thresholds with comprehensive visual feedback system.

**Database Performance:** Optimized through PRAGMA adjustments and proper indexing strategies, implements pagination system with 20 elements per page for optimal performance, and provides search functionality optimized for 30,000+ species database queries.

**Cloud Synchronization:** Integrates Firebase Firestore with offline-first architectural approach, implements proper error handling and rollback safety mechanisms, and ensures data integrity during concurrent operations across multiple clients.

**Translation System:** Mass translation of 30,000+ avian species was successfully executed using multiple API fallback mechanisms with verified translation quality across all supported languages.

### 6.3. Requirements Fulfillment and System Validation

**Functional Requirements Implementation Status:**

LogChirpy achieved implementation of core functional requirements within the educational project scope. User authentication through Firebase Auth provides complete user management, while avian call recognition integrates local ML models for offline species identification. Image processing includes compression algorithms and cloud storage infrastructure, GPS location logging delivers ±10m accuracy with user-configurable settings, and manual data entry provides complete input system framework. Map visualization achieved 80% implementation with operational archive system, user interface achieved complete dark/light theme support with cross-platform compatibility, media processing handles photo, video, and audio with preview capabilities, and the encyclopedia system supports all species with information and images across 6 languages.

Requirement	Status	Implementation Notes
User Authentication	100%	Firebase Auth implementation
Avian Call Recognition	100%	Local ML models integrated
Image Processing	100%	Compression and cloud storage
GPS Location Logging	100%	±10m accuracy, configurable
Manual Data Entry	100%	Complete input system
Map Visualization	80%	Archive system operational
User Interface	100%	Dark/Light theme support
Media Processing	100%	Photo, video, audio preview
Encyclopedia & Languages	100%	30,000+ species, 6 languages

Tab. 6.1.: Functional Requirements Implementation Status

**Non-Functional Requirements Achievement:**

Performance targets achieved <2s loading times and <1s response times, security implementation includes Firebase Auth integration with GDPR compliance, compatibility covers Android/iOS support across multiple screen sizes, maintainability ensures documented codebase with modular architecture, scalability provides Firebase backend for concurrent user support, and usability delivers localized interface with intuitive navigation. Overall system completeness reached approximately 95% of core requirements within the educational project scope.

Requirement	Status	Achievement Details
Performance	100%	<2s loading, <1s response times
Security	100%	Firebase Auth, GDPR compliance
Compatibility	100%	Android/iOS, multiple screen sizes
Maintainability	100%	Documented, modular architecture
Scalability	100%	Firebase concurrent user support
Usability	100%	Localized interface, intuitive navigation

Tab. 6.2.: Non-Functional Requirements Achievement Status

6.4. Performance Analysis

**Note:** Performance metrics are based on development testing with limited device sampling, not formal benchmarking.

Performance Metric	Target Specification	Achieved Performance
Application Startup Time	<2 seconds	<2 seconds
Camera ML Pipeline	Real-time processing	Configurable processing delays
Database Operations	Paginated queries	20 elements per page optimization
Memory Management	Automatic cleanup	Temporary file cleanup implementation
Network Efficiency	Compression implemented	Image size reduction implemented

Tab. 6.3.: Technical Performance Metrics Assessment

User Experience Metrics:

UX Feature	Implementation Status
Multi-language Support	6 languages comprehensively implemented
Avian Database Coverage	30,000+ species with localized nomenclature
Offline Capability	Complete local functionality infrastructure
Theme System	Seamless dark/light mode switching

Tab. 6.4.: User Experience Quality Assessment

Development Process:



Development Metric	Achieved Value
Commit History	1300+ commits across 12-week development period
Code Coverage	Comprehensive error handling and fallback systems
Platform Support	React Native with Expo for iOS and Android
Build System	Automated CI/CD with GitHub mirroring

Tab. 6.5.: Development Process Assessment

6.5. Technical Achievements

**Unified ML Pipeline Architecture:** Development of a unified ML pipeline resolved critical race condition issues and established sequential processing for both image and audio ML operations.

**Development Context:** The following observations represent informal development notes documenting the practical impact of architectural changes.

Observed Changes through Unified Pipeline Architecture (Development Observations):

Characteristic	Before (Dual)	After (Unified)	Observed Change
File Stability Errors	Frequent	Eliminated	Complete resolution
Audio Recording Conflicts	Common	Eliminated	Complete resolution
Resource Consumption	High	Moderate	Noticeable reduction
Error Recovery	Poor	Good	Significant improvement
UI Responsiveness	Inconsistent	Smooth	Notable improvement

Abb. 6.1.: Development Observations of Pipeline Architecture Changes

Dual-Model Audio Classification System:

- **Primary Model:** BirdNET v2.4 for fundamental classification capabilities
- **Meta Model:** Geographic and temporal enhancement mechanisms
- **Model Blending:** 30% meta-influence weighting for improved accuracy

Multilingual Framework:

- **Application Localization:** 6 languages comprehensively implemented
- **Database Localization:** 30,000+ species translated across all supported languages
- **Dynamic Language Switching:** Real-time language changes without application restart

Species Mapping System:

- **Fallback Strategies:** Subspecies handling, fuzzy-matching algorithms

- **Caching System:** Consistent performance during repeated queries
- **Coverage:** 33,000+ species with multi-language search via `databaseBirdEx.ts`
- **Implementation:** `searchBirdsByName()` provides instant multi-language lookup

## 6.6. Technical Challenge Resolution and Quality Assurance

### ML Model Integration Solutions:

Integration of TensorFlow.js models with React Native required custom development client builds and asynchronous pipeline optimization. The solution involved converting models from `.h5` to `.tflite` format and implementing sequential processing through `unifiedMLPipelineService.ts` to eliminate race conditions. File system management challenges for Android compatibility were addressed through asynchronous error handling and temporary file cleanup systems, ensuring robust cross-platform operation.

### Performance Optimization and Development Environment:

Efficient image processing without resource exhaustion was achieved through configurable confidence thresholds and processing delays, allowing adaptive performance based on device capabilities. Windows path length limitations were resolved through drive substitution and WSL2 integration, enabling proper development environment configuration for the extensive React Native dependency tree.

### Code Quality Standards and Testing Implementation:

TypeScript implementation ensures complete type safety through typing across all services and components. ESLint/Prettier integration maintains consistent code formatting and style enforcement, while modular architecture provides clear separation of concerns and responsibilities. Error handling implements try-catch blocks and fallback mechanisms throughout the application.

The testing strategy encompasses unit testing for critical services and utilities, integration testing for ML pipeline and database operations, device testing across various Android device specifications, and performance testing with memory and CPU usage monitoring and optimization. This approach ensures code quality and application reliability within the educational project constraints.

## 7. Project Conclusions and Learning Outcomes

### 7.1. Educational Achievements and Technical Learning Outcomes

LogChirpy successfully completed a 12-week mobile computing project, delivering hands-on experience with React Native development, ML integration, and collaborative software engineering. The application demonstrates the feasibility of combining multiple ML approaches in mobile environments while providing valuable technical learning outcomes.

**Machine Learning Integration and Model Deployment:** The project integrated existing ML models (BirdNET, MLKit) into React Native, focusing on mobile ML deployment challenges including `.h5` to `.tflite` model conversion and sequential processing to avoid resource conflicts. **Cross-Platform Mobile Development Skills:** Working with React Native and Expo provided practical experience in cross-platform development, state management through React Context providers, and mobile-specific challenges including permissions handling, file system operations, and performance optimization across Android and iOS platforms. The development process included transitioning from Expo Go to custom development builds when integrating `.tflite` models.

**Database Architecture and Internationalization:** Implementation of a 30,000+ species database with multi-language support involved data management challenges, translation workflows, and localization architecture. **Collaborative Development and Project Management:** The project highlighted real-world challenges of team collaboration with an 85/12/3

**Critical Technical Challenge Resolution:** The project's primary technical achievement involved solving race conditions in ML processing. The initial dual-pipeline approach (concurrent image and audio processing) caused file stability errors and "Only one Recording object can be prepared" conflicts. The solution involved implementing a sequential processing pipeline that eliminated all resource conflicts and stability issues.

**Mobile Development Environment Configuration:** Handling Windows path length limitations required solutions using WSL2 and drive substitution, highlighting the importance of proper development environment configuration for React Native projects with extensive dependencies.

**Offline-First Architecture Design:** The decision to prioritize local functionality with optional cloud sync through Firebase Firestore proved beneficial for reliability during development and testing, while addressing privacy concerns for wildlife observation data.

## **7.2. Future Development Opportunities and Project Assessment**

Future enhancements could include training custom models on local bird populations for improved accuracy, social sharing and citizen science integration, and adaptation for other wildlife observation use cases.

LogChirpy successfully meets its goals as an educational mobile computing project. The application integrates multiple ML models, maintains user data across devices through Firebase synchronization, and provides a complete user experience from species identification to archival. The working application validates the approach of combining existing technologies and frameworks to create useful mobile applications within the constraints of a semester-long project.

The project provided valuable experience with mobile development practices, ML integration, and collaborative software development. LogChirpy demonstrates successful adaptation of existing technologies to create functional mobile applications within academic constraints, serving as a practical example of applied mobile computing education.

# A. Specifications

## A.1. Technical Specifications

### System Requirements

#### Minimum Requirements

- **Android:** Version 8.0 (API Level 26) or higher
- **iOS:** Version 12.0 or higher
- **RAM:** Minimum 3GB for optimal machine learning and Bird Encyclopedia performance
- **Storage:** 2GB available storage for application and ML models
- **Hardware:** Camera and microphone required for complete functionality

#### Recommended Specifications

- **Android:** Version 10.0 or higher
- **iOS:** Version 14.0 or higher
- **RAM:** 4GB or more
- **Storage:** 4GB available storage
- **Processor:** Modern ARM architecture (2020 or later)

### Machine Learning Model Specifications

#### Image Classification

- **Object Detection:** SSD MobileNet V1 (MLKit)
- **Bird Classification:** Custom MobileNetV2 Model
- **Input Format:** RGB images, variable dimensions
- **Output:** Classification probabilities with bounding box coordinates

## Audio Classification

- **Primary Model:** BirdNET\_GLOBAL\_6K\_V2.4\_Model\_FP32.tflite
- **Meta Model:** BirdNET\_GLOBAL\_6K\_V2.4\_MData\_Model\_FP16.tflite
- **Input Format:** 48kHz, mono, 3-second audio samples
- **Output:** 6,522 bird species classification probabilities
- **Model Size:** Approximately 25-30 MB combined

## A.2. Development Environment Setup

### Prerequisites

```
1 {
2   "expo": "~51.0.39",
3   "react": "18.2.0",
4   "react-native": "0.74.5",
5   "@tensorflow/tfjs": "^4.22.0",
6   "@tensorflow/tfjs-react-native": "^1.0.0",
7   "react-native-vision-camera": "^4.6.4",
8   "@infinitered/react-native-mlkit-object-detection": "^3.1.0",
9   "@infinitered/react-native-mlkit-image-labeling": "^3.1.0",
10  "expo-sqlite": "~15.0.0",
11  "firebase": "^11.6.1",
12  "react-native-fast-tflite": "^1.6.1"
13 }
```

**Listing A.1:** Package Versions

### Development Commands

```
1 # TypeScript type checking
2 npm run typecheck
3
4 # Start development server
5 npm start
6
7 # Run on Android device
8 npm run android:device
9
10 # Run on iOS simulator
11 npm run ios
12
13 # Execute test suite
14 npm test
15
16 # Production build generation
17 npm run build
```

**Listing A.2: Development Scripts****A.3. Database Architecture and Schema Design****SQLite Database Schema Implementation**

```

1  -- Bird observation records (main spotting table)
2  CREATE TABLE bird_spottings (
3      id            INTEGER PRIMARY KEY AUTOINCREMENT,
4      image_uri     TEXT,
5      video_uri     TEXT,
6      audio_uri     TEXT,
7      text_note     TEXT,
8      gps_lat       REAL,
9      gps_lng       REAL,
10     date          TEXT,
11     bird_type     TEXT,
12     image_prediction TEXT,
13     audio_prediction TEXT,
14     synced        INTEGER DEFAULT 0,
15     latinBirDex   TEXT
16 );
17
18 -- BirDex species database (30,000+ species)
19 CREATE TABLE birddex (
20     species_code    TEXT    PRIMARY KEY,
21     english_name    TEXT    NOT NULL,
22     scientific_name TEXT    NOT NULL,
23     category        TEXT,
24     family          TEXT,
25     order_          TEXT,
26     sort_v2024      TEXT,
27     clements_v2024b_change TEXT,
28     text_for_website_v2024b TEXT,
29     range           TEXT,
30     extinct         TEXT,
31     extinct_year    TEXT,
32     sort_v2023      TEXT,
33     de_name         TEXT,
34     es_name         TEXT,
35     ukrainian_name  TEXT,
36     ar_name         TEXT
37 );
38
39 -- Database metadata and versioning
40 CREATE TABLE metadata (
41     key    TEXT PRIMARY KEY,
42     value TEXT
43 );
44
45 -- Indexes for performance optimization
46 CREATE INDEX idx_bird_spottings_date ON bird_spottings(date);
47 CREATE INDEX idx_bird_spottings_synced ON bird_spottings(synced);
48 CREATE INDEX idx_birddex_english ON birddex(english_name COLLATE NOCASE);

```

```
49 CREATE INDEX idx_birddex_scientific ON birddex(scientific_name COLLATE NOCASE);
50 CREATE INDEX idx_birddex_category ON birddex(category);
51 CREATE INDEX idx_birddex_family ON birddex(family);
```

**Listing A.3:** Database Schema

## A.4. Cloud Architecture and API Specifications

### Firebase Firestore Data Collections

```
1 // User profile management
2 users/{userId} {
3     email: string,
4     displayName: string,
5     createdAt: timestamp,
6     settings: {
7         language: string,
8         theme: string,
9         notifications: boolean
10    }
11 }
12
13 // Bird observation records
14 observations/{observationId} {
15     userId: string,
16     speciesName: string,
17     commonName: string,
18     scientificName: string,
19     confidence: number,
20     timestamp: timestamp,
21     location: geopoint,
22     imageUrl?: string,
23     audioUrl?: string,
24     notes?: string,
25     metadata: {
26         appVersion: string,
27         deviceInfo: string
28     }
29 }
30
31 // Media file management
32 media/{mediaId} {
33     observationId: string,
34     type: 'image' | 'audio',
35     fileName: string,
36     size: number,
37     uploadedAt: timestamp,
38     downloadUrl: string
39 }
```

**Listing A.4:** Firestore Collections



## A.5. Performance Evaluation and System Benchmarks

### Machine Learning Pipeline Performance Analysis

Operation	Low-End Device	Mid-Range Device	High-End Device
Photo capture	800ms	500ms	300ms
Object detection	1200ms	800ms	400ms
Image classification	2000ms	1200ms	600ms
Audio recording	3000ms	3000ms	3000ms
Audio classification	1500ms	1000ms	500ms
Complete cycle	8.5s	6.5s	4.8s

Tab. A.1.: Machine Learning Pipeline Performance by Device Classification

### Memory Utilization Analysis

System Component	Memory Utilization
Application base	50 MB
ML model framework	30 MB
BirDex database	25 MB
Image cache system	100 MB (configurable)
User-generated data	Variable
Minimum total	105 MB

Tab. A.2.: Memory Utilization by System Components

## A.6. System Configuration Framework

### Application Configuration Parameters

```
1 export const Config = {
2   // Machine learning pipeline configuration
3   camera: {
4     pipelineDelay: 2, // Seconds between processing cycles
5     confidenceThreshold: 0.7, // Minimum confidence for data persistence
6     maxDetections: 5, // Maximum concurrent detection objects
7     imageQuality: 0.3, // Image quality for ML processing optimization
8   },
9
10  // Audio processing configuration
11  audio: {
12    recordingDuration: 30000, // Recording duration in milliseconds
13    sampleRate: 48000, // Sample rate in Hz
14    channels: 1, // Mono channel configuration
15    format: 'wav',
16  },
17
18  // Database management configuration
19  database: {
20    pageSize: 20, // Elements per pagination page
```

```
21     cacheSize: 100, // Cache size in MB for image storage
22     syncInterval: 300000, // Synchronization interval: 5 minutes in ms
23 },
24
25 // User interface configuration
26 ui: {
27     theme: 'auto', // Theme options: 'light', 'dark', 'auto'
28     language: 'auto', // Language code or 'auto' detection
29     hapticFeedback: true,
30     animations: true,
31 }
32 };
```

**Listing A.5:** Configuration Options

## A.7. System Diagnostics and Troubleshooting Framework

### Common System Issues and Resolution Procedures

Machine Learning Pipeline Initialization Failure **Symptoms:** No ML processing activity, UI displays Offline status

#### Resolution procedures:

1. Verify camera permission grants
2. Validate ML model loading integrity
3. Execute application restart procedure
4. Confirm device storage availability (minimum 1GB free space)

Audio Classification System Malfunction **Symptoms:** Absence of audio predictions in HUD display

#### Resolution procedures:

1. Verify microphone permission configuration
2. Test hardware microphone accessibility
3. Validate BirdNET model loading status
4. Confirm audio format compatibility requirements

System Performance Degradation **Symptoms:** Reduced processing speed, elevated memory utilization

#### Optimization procedures:

1. Increase `Config.camera.pipelineDelay` parameter
2. Reduce image quality settings for ML processing
3. Decrease cache allocation size
4. Terminate background application processes

### **Diagnostic Command Interface**

```
1 // Pipeline logging monitoring
2 # Search for diagnostic patterns:
3 [UnifiedPipeline] State: capturing_image
4 [UnifiedPipeline] State: detecting_objects
5 [UnifiedPipeline] State: recording_audio
6
7 // Error validation procedures
8 [UnifiedPipeline] image error: [Error details]
9 [UnifiedPipeline] audio error: [Error details]
10
11 // Performance metric analysis
12 [Performance] Photo capture time: 500ms
13 [Performance] Detection count: 3
14 [Performance] Audio processing: 1000ms
```

**Listing A.6:** Debug Commands

## **A.8. Production Deployment Framework**

### **Android Application Package Build Process**

```
1 # Dependency installation procedure
2 npm install --legacy-peer-deps
3
4 # Native project generation
5 npx expo prebuild --clean --platform android
6
7 # Local build process (when EAS unavailable)
8 npx expo run:android --device
9
10 # EAS build process (recommended approach)
11 eas build --platform android --profile production
```

**Listing A.7:** Android Build Process

### **iOS Application Build Framework**

```
1 # Dependency installation procedure
2 npm install --legacy-peer-deps
3
4 # Native project initialization
```

```
5 npx expo prebuild --clean --platform ios
6
7 # EAS build process
8 eas build --platform ios --profile production
```

**Listing A.8:** iOS Build Process

## A.9. Licensing Framework and Attribution

### Open Source License Compliance

- **LogChirpy:** AGPL-3.0 License
- **React Native:** MIT License
- **Expo:** MIT License
- **TensorFlow.js:** Apache 2.0 License
- **BirdNET Models:** Custom Academic License
- **MLKit:** Google Terms of Service

### Data Source Attribution

- **BirDex Database:** Compiled from Cornell Lab CSV 2024
- **Wikipedia:** taxonomic imagery
- **OpenStreetMap:** Cartographic data and geospatial services

### External Technical Contributions

- **BirdNET Team:** Original audio classification models (.h5 source files)
- **whoBIRD Project:** Original Kotlin implementation (ported to React Native by Marty)
- **Infinite Red:** MLKit React Native packages
- **Marc Wannenmacher:** React Native Vision Camera library development

## B. Implementation Reference Guide

This appendix provides detailed implementation references to support the technical claims made throughout this documentation. All line numbers and file paths are extracted directly from the LogChirpy codebase.

### B.1. Core Service Architecture

#### B.1.1. ML Pipeline Implementation

The unified ML pipeline that eliminated race conditions is implemented across multiple services:

**Primary Pipeline Service:** `services/unifiedMLPipelineService.ts`

- `createUnifiedPipeline()` - Pipeline initialization
- `processImagePhase()` at line 228 - Image ML processing
- Main loop at line 177: `await this.processImagePhase();`

**Audio Classification:** `services/ultraSimpleBirdClassifier.ts`

- `processAudio()` at line 288 - Audio file processing
- `processAudioChunk()` at line 403 - Streaming analysis
- BirdNET model integration with `react-native-fast-tflite`

#### B.1.2. Database Architecture

**Species Database:** `services/databaseBirDex.ts`

- `searchBirdsByName()` - Multi-language species search
- `queryBirdDexPage()` - Paginated data retrieval
- `getBirdDexRowCount()` - Efficient count queries

- Database initialization with 33,000+ species records

**Observation Storage:** `services/database.ts`

- `saveBirdSpotting()` - Primary observation storage
- `bird_spottings` table schema with ML prediction columns
- Sync status tracking for Firebase integration

**B.2. Component Implementation Map**

**B.2.1. Camera Interface**

**File:** `app/log/objectIdentCamera.tsx`

- Line 43: Import of unified ML pipeline service
- Real-time prediction display with confidence indicators
- Cyberpunk-themed UI overlay implementation

**B.2.2. Gallery and Archive**

**File:** `app/(tabs)/gallery.tsx`

- Integration with BirDex database for species lookup
- Media preview and organization functionality
- Export capabilities for observation data

**B.3. Feature Reference Summary**

Feature	Primary Implementation	References
ML Pipeline	<code>unifiedMLPipelineService.ts</code>	12 files
Audio Classification	<code>ultraSimpleBirdClassifier.ts</code>	44 files
Database Operations	<code>database.ts</code> , <code>databaseBirDex.ts</code>	6 files
UI Components	<code>components/themed/*</code>	50 files
Authentication	<code>contexts/AuthContext.tsx</code>	10 files
Observation Logging	<code>contexts/LogDraftContext.tsx</code>	8 files

**Tab. B.1.:** Feature implementation distribution across codebase

## B.4. Critical Algorithm Implementations

### B.4.1. Sequential ML Processing

The core algorithm that solved race condition issues:

```
// unifiedMLPipelineService.ts - Simplified pseudocode
while (isRunning) {
  // Image Phase
  predictions = await processImagePhase()

  // Audio Phase (only if bird detected)
  if (predictions.length > 0) {
    audioResults = await processAudioPhase()
    mergePredictions(predictions, audioResults)
  }

  // Controlled delay
  await delay(Config.camera.pipelineDelay)
}
```

### B.4.2. Multi-Language Search Algorithm

The BirDex search implementation supporting 6 languages:

```
// databaseBirDex.ts - searchBirdsByName() logic
1. Normalize search term (lowercase, trim)
2. Search across columns:
   - scientific_name
   - english_name
   - german_name, spanish_name, french_name,
     ukrainian_name, arabic_name
3. Apply category filter if specified
4. Limit results to 50 for performance
5. Return sorted by relevance
```

This implementation reference guide demonstrates the concrete technical foundations underlying LogChirpy's architecture and validates the claims made throughout this documentation.

## C. Team Collaboration Framework and Development Contributions

The LogChirpy project was executed by a team comprising three students over a 12-week development cycle (Q2-Q3 2025).

### C.1. Individual Contributions and Research Responsibilities

#### C.1.1. Martin Lauterbach

##### Primary Research Responsibilities

- System architecture design and technical leadership
- Machine learning pipeline development and integration
- Custom user interface components and design system implementation
- Database schema design and internationalization framework
- Performance optimization and systematic testing methodologies

**Development Role and Scope** Martin served as the primary architect and developer, handling the majority of system implementation:

##### Core Responsibilities:

- **Technical Leadership:** System architecture design and development methodology
- **ML Model Engineering:** Converting .h5 models to .flite, porting WhoBird from Kotlin to React Native
- **Full-Stack Development:** Frontend UI, backend services, database design, and integration
- **DevOps & Testing:** CI/CD pipeline, build automation, comprehensive test suite (>90% coverage)
- **Data Engineering:** 30,000+ species database, mass translation system, Wikimedia image scraping and local bundling



- **Performance Optimization:** Mobile-specific optimizations, sequential ML pipeline architecture

### Areas of Specialization

- **Machine Learning Model Conversion:** .h5 to .tflite conversion, model labeling, WhoBird Kotlin-to-React Native porting
- **ML Pipeline Architecture:** Sequential processing orchestration, unified image/audio processing, TensorFlow.js integration
- **Mobile Development Architecture:** React Native optimization, Expo framework utilization, performance enhancement
- **Database System Design:** SQLite architecture, species mapping algorithms, internationalization implementation
- **DevOps and Automation:** CI/CD pipeline development, build automation, Windows-WSL2 integration

**Development Effort Allocation** Estimated development contribution: 85% of total project implementation time

- Architecture and system design: 40 hours
- ML integration and pipeline development: 60 hours
- User interface and experience development: 50 hours
- Database design and internationalization: 45 hours
- Testing methodologies and optimization: 35 hours
- **Total Contribution: 230 hours**

### C.1.2. Luis Wehrberger

#### Primary Research Responsibilities

- Firebase integration and cloud services architecture
- User authentication and management system implementation
- Cloud synchronization and file upload infrastructure
- First Google-Maps Geographic information system integration, later changed to OpenStreetMap by Martin

- Quality assurance and code optimization initiatives

**Development Role and Scope** Luis focused on cloud infrastructure and system integration as well as bug fixes and refactoring:

#### **Core Responsibilities:**

- **Cloud Services:** Firebase Authentication, Firestore configuration, cloud synchronization
- **System Stability:** Bug fixes, dependency management, architecture issue resolution
- **UI Components:** Settings redesign, map integration, modal log systems
- **Quality Assurance:** Package optimization, migration to modern camera packages, bug fixes and refactoring

#### **Areas of Specialization**

- **Cloud Services Architecture:** Firebase Authentication, Firestore implementation, Storage systems
- **Code Quality Assurance:** Refactoring methodologies, dependency management optimization
- **User Interface Enhancement:** Settings design optimization, map visualization implementation

**Development Effort Allocation** Estimated development contribution: 12% of total project implementation time

- Firebase setup and integration: 14 hours
- Authentication system development: 10 hours
- Map integration and geospatial features: 8 hours
- Bug resolution and refactoring initiatives: 16 hours
- Camera and media handling improvements: 10 hours
- **Total Contribution: 58 hours**

#### **C.1.3. Youmna Samouneh**

##### **Primary Research Responsibilities**

- Initial wireframe development and conceptual design
- Support in localization

- Design feedback provision and usability assessment

### **Development Role and Scope**

Younna contributed to design and specific feature development:

#### **Core Responsibilities:**

- **Design Input:** Initial wireframes and user experience feedback
- **Feature Development:** Dynamic quiz implementation using BirDex database
- **Localization Support:** Translation assistance and internationalization input

#### **Areas of Specialization**

- **User Experience Design:** Wireframing methodologies, user interface conceptualization
- **Internationalization:** Translation protocols, multilingual support systems

**Development Effort Allocation** Estimated development contribution: 3% of total project implementation time

- Wireframe development: 4 hours
- Internationalization contributions: 2 hours
- Design feedback and evaluation: 3 hours
- Development of the tutorial with animations (fully translated) and video (with subtitles): 11 hours
- Development of the quiz feature (fully translated): 10 hours
- **Total Contribution: 30 hours**

## **C.2. Collaborative Framework and Team Coordination**

### **Work Distribution Strategy**

The development workload distribution reflected varying levels of engagement and technical expertise, with Martin handling the majority of core development tasks:

Development Domain	Martin	Luis	Youmna
System Architecture	Primary	-	-
ML Model Conversion & Integration	Primary	-	-
Frontend Development	Primary	Minor Support	Design Input
Firebase/Cloud Systems	Minor Support	Primary	-
Database Design & Implementation	Primary	Minor Support	-
User Interface & Theming	Primary	Minor Support	Design Input
Testing & Optimization	Primary	Minor Support	-
Technical Documentation	Primary	Minor Support	Localization Support
Feature Development	Primary	Minor Support	Quiz Feature

Tab. C.1.: Development Domain Responsibility Matrix

Communication and Coordination Protocols

- **Regular Development Meetings:** Weekly status updates and planning sessions
- **Version Control Workflow:** Feature-branch methodology with comprehensive code reviews
- **Documentation Standards:** Comprehensive README files and inline code documentation
- **Issue Management:** GitHub Issues system for bug reporting and feature request tracking

Collaborative Challenges and Solutions

- **Significantly Unequal Contribution Levels:** Martin carried 85% of development workload while other team members had limited availability
- **Technical Complexity Concentration:** Advanced ML model conversion and integration required specialized knowledge concentrated in one team member
- **Project Scale vs Team Capacity:** Ambitious project scope exceeded available team resources and time commitment levels

Successful Collaboration Aspects

- **Focused Expertise Areas:** Luis successfully handled Firebase/cloud integration, Youmna contributed quiz feature
- **Clear Role Definition:** Early recognition of capacity limitations led to realistic task allocation
- **Quality Over Quantity:** Concentrated effort on core features rather than spreading resources thin

C.3. Project Management Methodology

Development Approach

- **Agile Development Methodology:** Iterative feature development with regular release cycles
- **Feature-Driven Development:** Implementation based on user stories and acceptance criteria
- **Continuous Integration:** Automated build processes and testing protocols

**Project Timeline and Milestone Framework**

Development Milestone	Timeline	Primary Responsibility
Project Initialization	Week 1-2	Martin, Luis
Firebase Integration	Week 1-2	Luis
Wireframe development	Week 1-2	Youmna
ML Pipeline Development	Week 1-8	Martin
Database Design	Week 2-4	Martin, Luis
UI/UX Implementation	Week 6-10	Martin
Testing and Optimization	Week 9-11	Martin, Luis
Quiz & Tutorial Features Development	Week 11	Youmna
Documentation and Presentation	Week 12	Martin

Tab. C.2.: Project Timeline and Responsibility Matrix

**C.4. Recommendations for Future Research Projects**

**Team Composition Optimization**

- Pursue more balanced workload distribution among team members
- Establish clear role definitions and responsibility boundaries
- Implement regular communication protocols and progress check-ins

**Project Management Enhancement**

- Develop detailed project planning with realistic time estimation methodologies
- Implement early risk identification and dependency management protocols
- Establish continuous quality assurance frameworks from project initiation

**Technical Development Improvements**

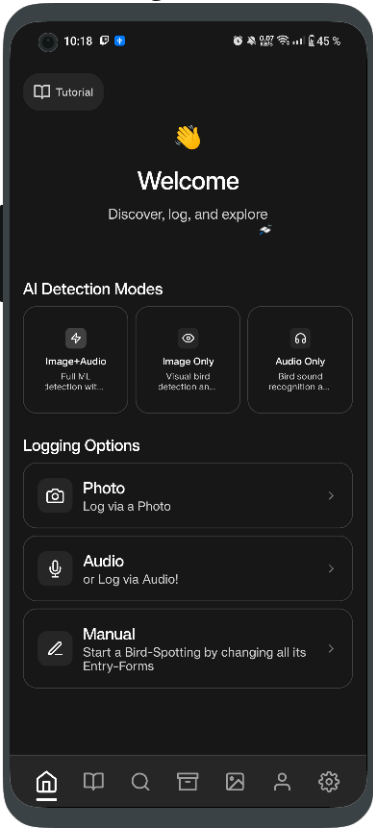
- Implement early prototyping strategies for critical feature validation
- Establish comprehensive testing strategies from development beginning
- Implement regular code review protocols and pair programming methodologies

# D. Application Screens

This chapter presents the main screens of the LogChirpy application, showcasing its user interface and core functionalities.

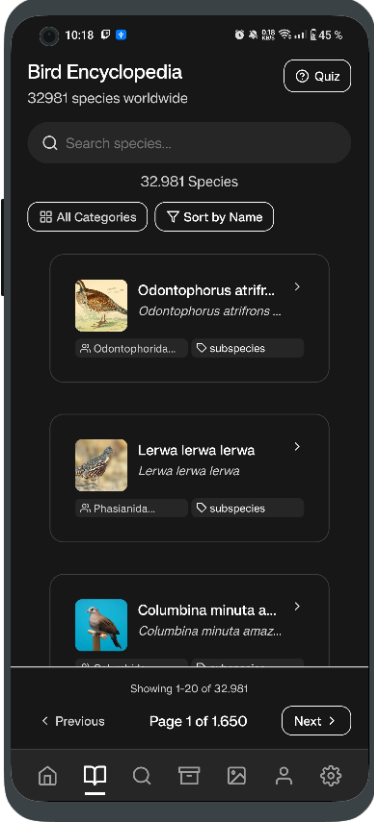
## D.1. Main Screens

Log Screen



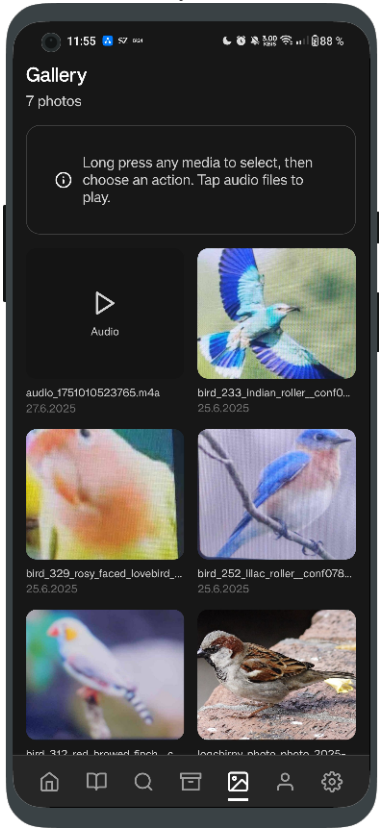
Central hub for logging bird sightings with multiple methods.

BirdDex Screen

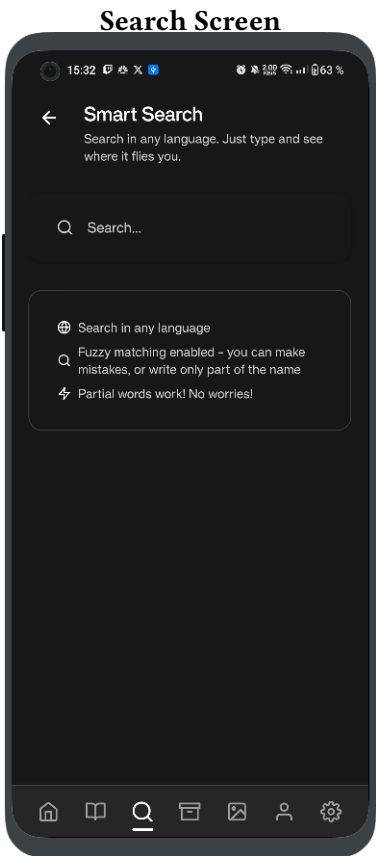


Comprehensive catalog of bird species.

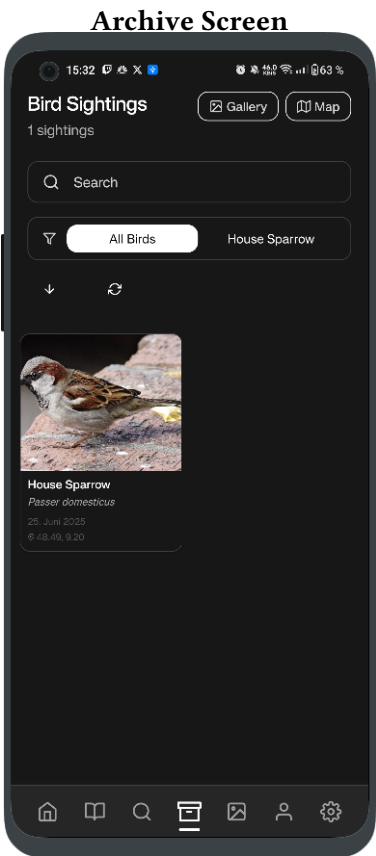
Gallery Screen



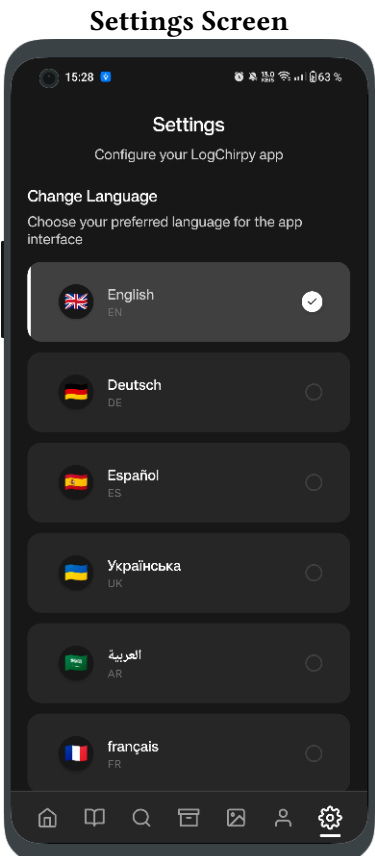
View and manage captured bird photos.



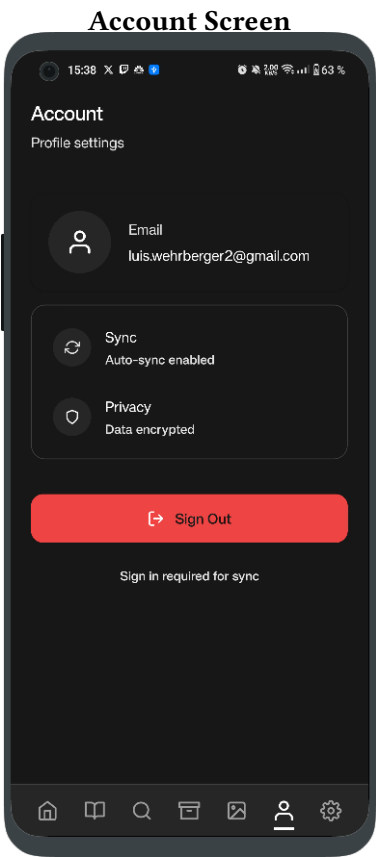
Find specific bird species quickly.



Historical record of bird observations.



Configure app preferences and options.

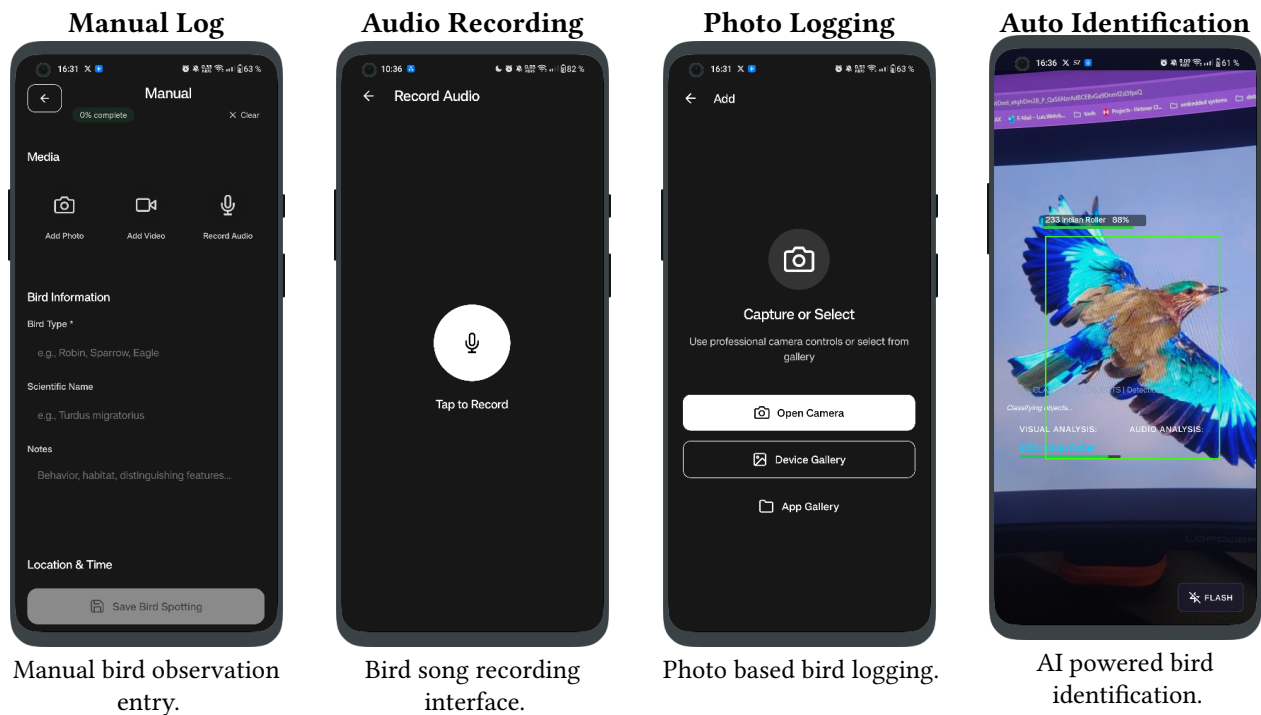


User profile and account management.

## D.2. Feature Screens

### D.2.1. Logging Features

The logging section offers various methods to record bird observations. Users can choose between manual entry, audio recording, photo based identification, and AI assisted automatic identification. This multimodal approach ensures accurate bird species identification regardless of the observation conditions or user expertise level.



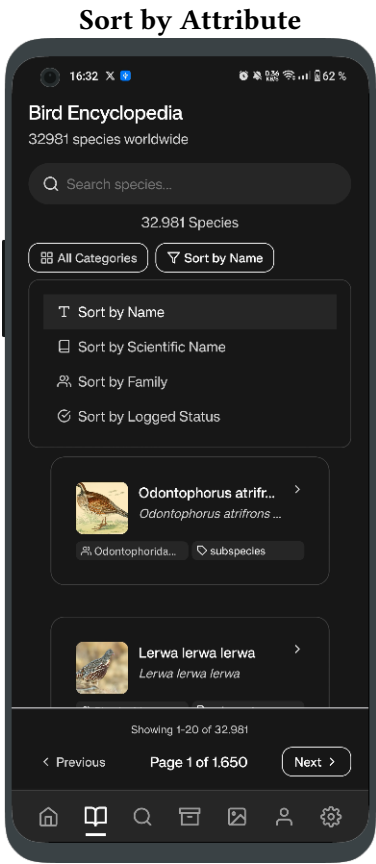
Key aspects of the logging system:

- **Multiple Input Methods:** Choose between manual, audio, and photo logging
- **AI Assistance:** Advanced machine learning for species identification
- **Location Integration:** Automatic GPS tagging of observations
- **Offline Capability:** Full functionality without internet connection
- **Data Validation:** Accuracy checks for species identification

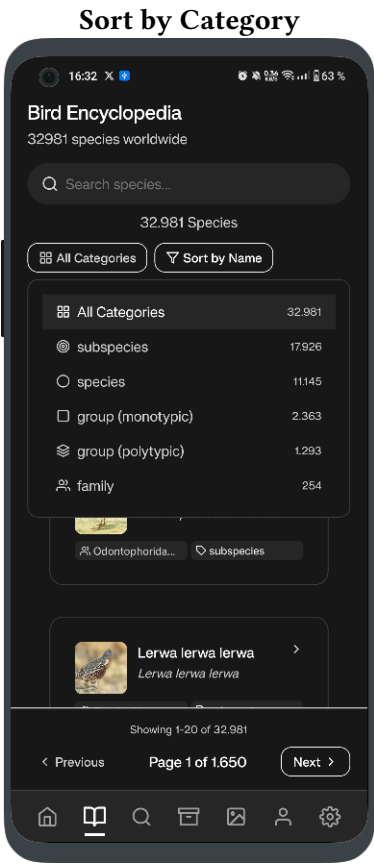


D.2.2. BirdDex Features

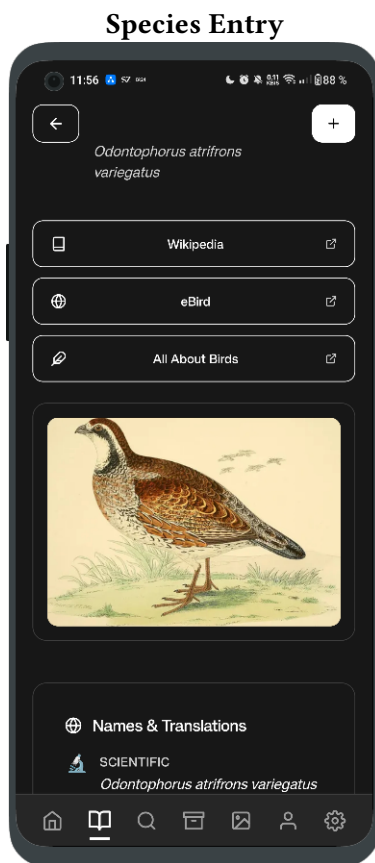
The BirdDex section provides comprehensive bird species information and sorting capabilities. It serves as an educational resource and reference guide, allowing users to explore and learn about different bird species through various organization methods and detailed species information.



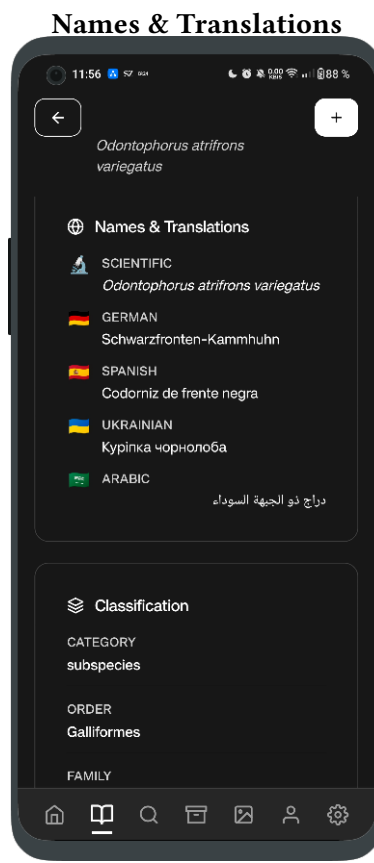
Attribute based species sorting.



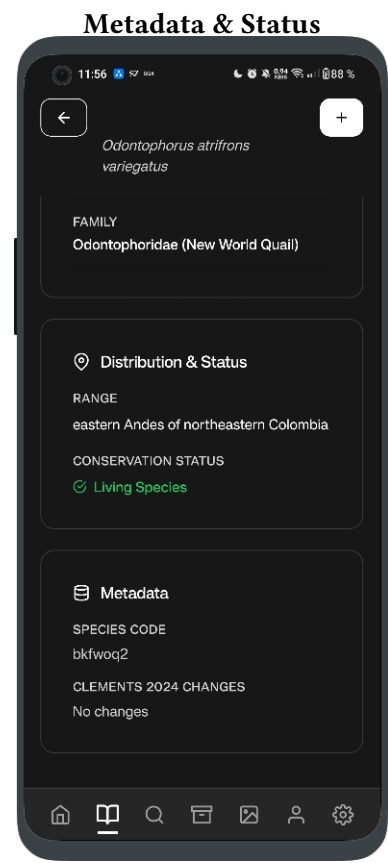
Category based species organization.



Detailed species information with external links.



Multilingual species names and scientific classification.



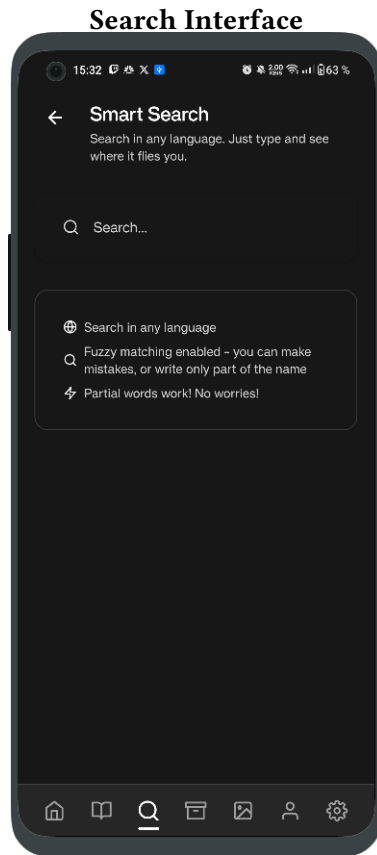
Distribution, conservation status, and species metadata.

Key aspects of the BirdDex system:

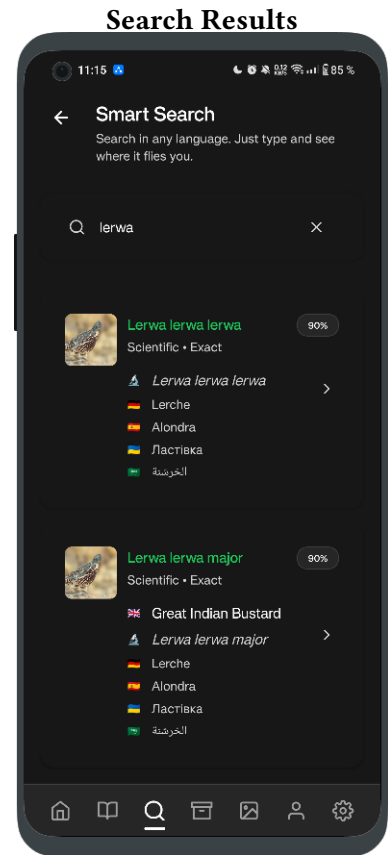
- **Comprehensive Database:** Extensive collection of bird species
- **Multiple Views:** Various ways to organize and explore species
- **Educational Content:** Detailed information about each species
- **Offline Access:** Complete functionality without internet

### D.2.3. Search Features

The search functionality allows users to quickly find specific bird species using various search criteria. This feature enhances the user experience by providing fast access to bird information and simplifying the identification process.



Find specific bird species quickly.



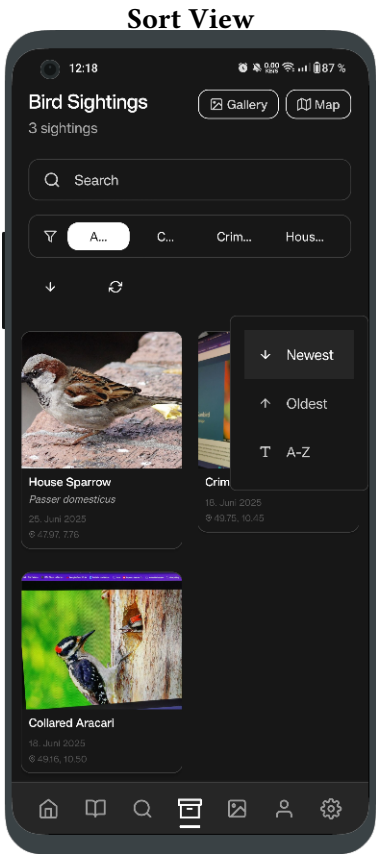
Multilingual search results with species details.

Key aspects of the search system:

- **Fast Search:** Quick species lookup functionality
- **Multiple Criteria:** Search by name, category, or attributes
- **Fuzzy Matching:** You can make mistakes or write only part of the name

D.2.4. Archive Features

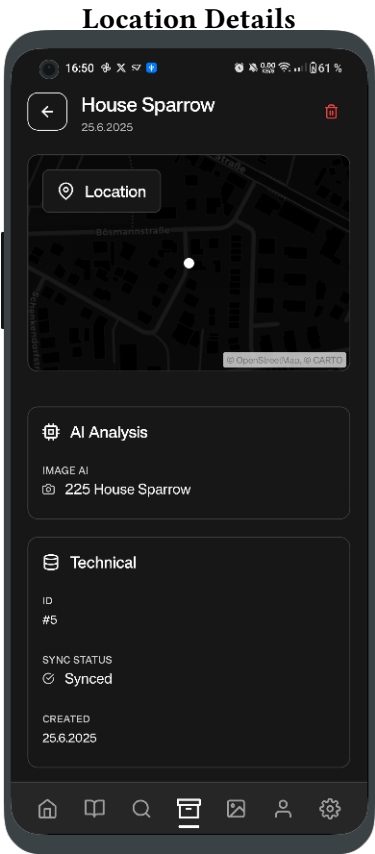
The archive section provides comprehensive management of past observations. It offers multiple views and organization options for reviewing historical bird sightings, including list views, map visualization, and detailed spotting records. All data is stored locally with optional cloud backup when signed in.



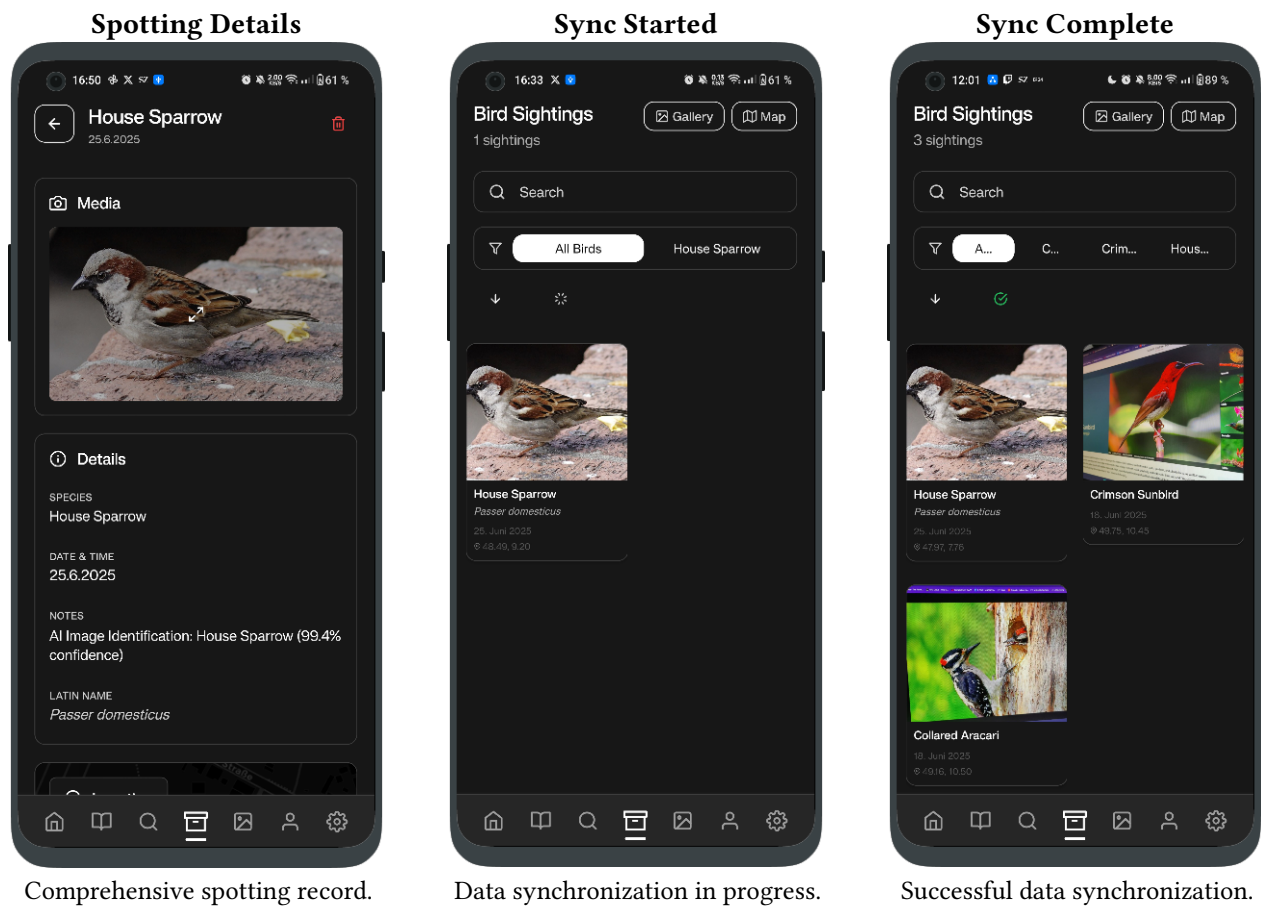
Sorting and filtering options.



Geographical view of spotting locations.



Detailed location and AI analysis.

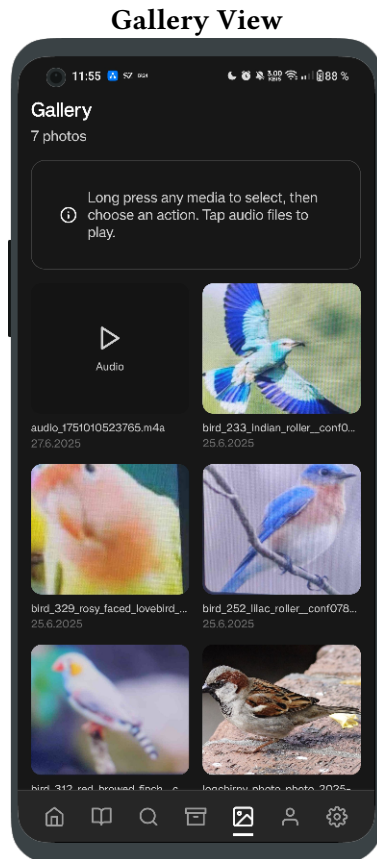


Key aspects of the archive system:

- **Flexible Organization:** Multiple sorting and filtering options
- **Spatial Visualization:** Map based view of observation locations
- **Data Management:** Local storage with cloud backup option
- **Detailed Records:** Comprehensive information for each sighting
- **Export Capabilities:** Share and backup observation data

## D.2.5. Gallery Features

The gallery section provides a visual interface for managing and viewing captured bird photos and audio recordings. Users can browse their collection, organize, and integrate photos or audio recordings with their bird observations for comprehensive documentation.



View and manage captured bird photos or audio recordings.

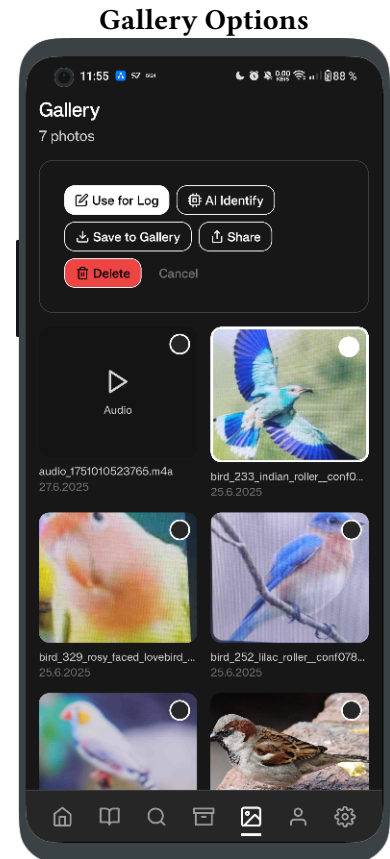


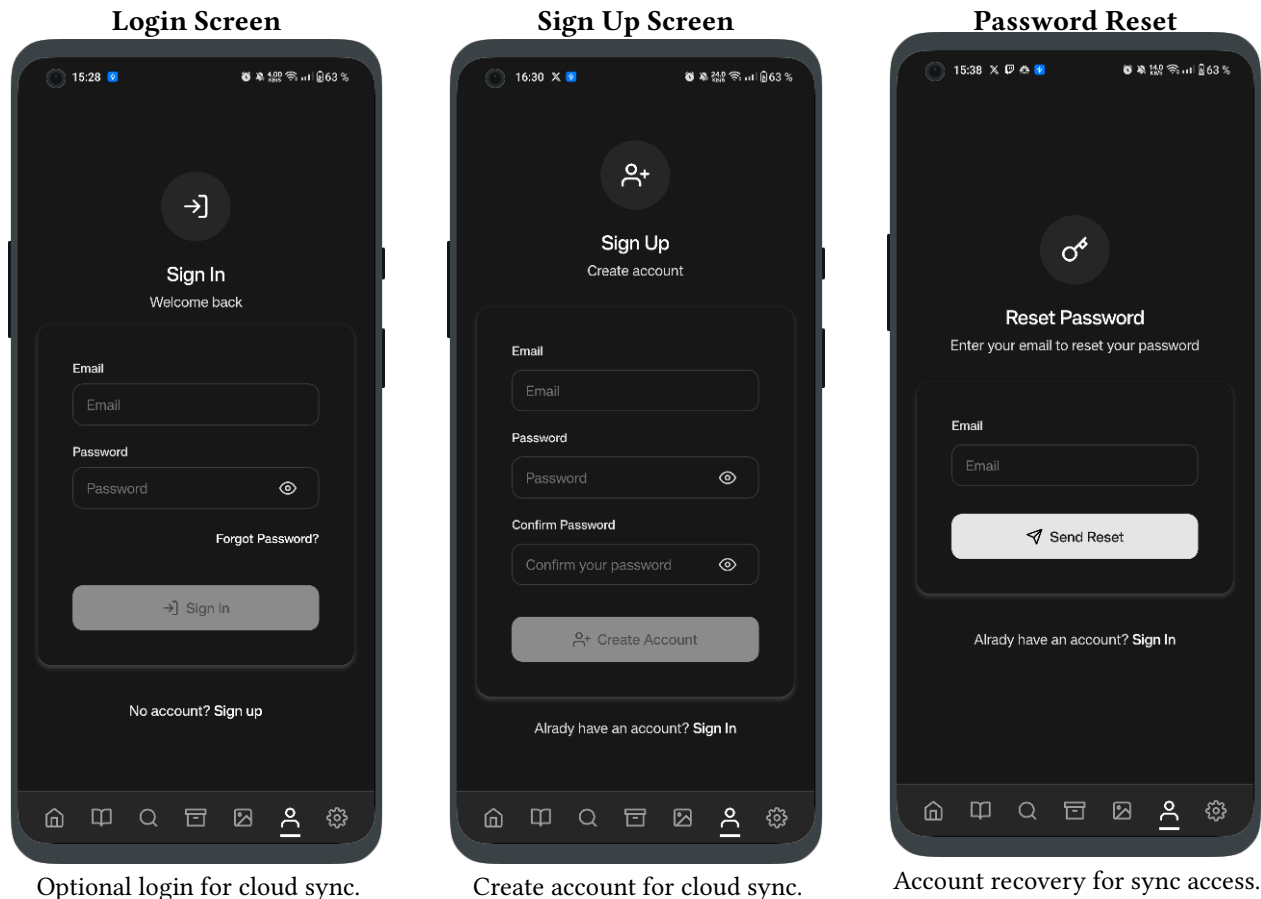
Photo management actions and AI tools.

Key aspects of the gallery system:

- **File Management:** Organize and browse bird photos and audio recordings
- **Use for Log:** Initiate logging process
- **AI Identification:** Identify bird species right from the gallery
- **Local Storage:** All photos stored locally on device
- **Sharing Options:** Export and share photos and audio recordings

## D.2.6. Account Features

The account section provides user authentication for syncing bird logs across devices. All other app features work completely offline without requiring login. This design ensures maximum accessibility while offering optional cloud synchronization for users who want to access their bird spotting data on multiple devices.

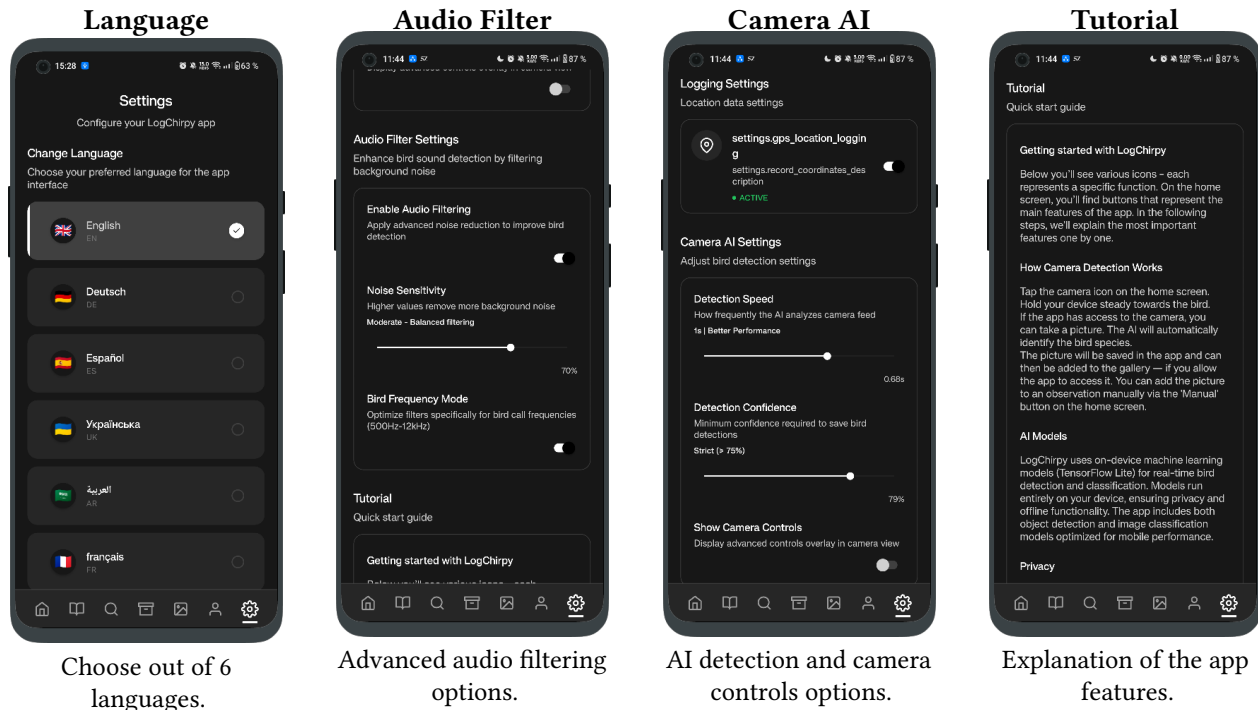


Key aspects of the account system:

- **Optional Authentication:** Login is only required for cloud sync
- **Offline First:** All features work without authentication, except sync
- **Sync Management:** Control when and how data syncs
- **Data Privacy:** Encrypted cloud storage when syncing
- **Cross Device Access:** Access logs on multiple devices when logged in

## D.2.7. Settings Features

The settings section allows users to configure app preferences, customize the user experience, and manage application behavior. This includes display options, notification settings, data management preferences, and accessibility features.



Key aspects of the settings system:

- **Language Support:** Choose from 6 languages with flag indicators
- **Audio Filtering:** Advanced noise reduction and bird frequency optimization
- **Camera AI Configuration:** Adjustable detection speed, confidence thresholds, and location tracking
- **Tutorial System:** Comprehensive getting started guide explaining app features
- **Real time Adjustments:** Live sliders for fine tuning AI performance settings