CNN-Powered Interactive Travel Guide

by
Onat Özgen

Engineering Project Report

Yeditepe University
Faculty of Engineering
Department of Computer Engineering
2025

CNN-Powered Interactive Travel Guide

APPROVED BY:

Assist. Prof. Dr. Onur Demir (Supervisor)	. . . . .
Prof. Dr. Mert Özkaya	. . . . .
Assoc. Prof. Dr. Dionysis Goularas	. . . . .

DATE OF APPROVAL: .../.../2025

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor ... for his guidance and support throughout my project.

Also I would like to thank my parents for their support and encouragement throughout my education up to the present.

ABSTRACT

CNN-Powered Interactive Travel Guide

This project presents a mobile travel application that encourages users to explore historical sites while engaging in meaningful social interactions. Users can write and share travel guides and explore available routes in cities. The app also provides audio narratives about recognized landmarks and rewards users with digital memory coins for each visit. Additionally, on-site sharings such as comments and photos can be made at historical locations, viewable only to followers who physically visit the same place. With user profiles and a follower system, the platform functions as a social media application that offers an interactive and community-driven travel experience.

ÖZET

CNN Tabanlı İnteraktif Seyahat Rehberi

Bu proje, kullanıcıların tarihi yerleri keşfederken anlamlı sosyal etkileşimler kurmalarını teşvik eden bir mobil seyahat uygulaması sunmaktadır. Kullanıcılar seyahat rehberleri yazıp paylaşabilir ve şehirlerdeki mevcut rotaları keşfedebilirler. Uygulama, tarihi yapıları tanıyarak hakkında sesli anlatımlar sunar ve her ziyaret için kullanıcılara dijital hatıra para kazandırır. Ayrıca, kullanıcılar tarihi mekanlarda yorum ve fotoğraf gibi paylaşımlar yapabilir; bu içerikler yalnızca aynı mekanı fiziksel olarak ziyaret eden takipçiler tarafından görüntülenebilir. Kullanıcı profilleri ve takip sistemi sayesinde platform, sosyal medya uygulaması olarak, etkileşimli ve topluluk odaklı bir seyahat deneyimi sunar.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

ML	Machine Learning
CNN	Convolutional Neural Network
AI	Artificial Intelligence
AR	Augmented Reality
UI	User Interface
UX	User Experience
FPS	Frames Per Second
CPU	Central Processing Unit
GPS	Global Positioning System
JSON	JavaScript Object Notation
NoSQL	Non-Relational Database (Not Only SQL)
URL	Uniform Resource Locator
IDE	Integrated Development Environment
ms	Milliseconds
ReLU	Rectified Linear Unit
ID	Identifier
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface

1. INTRODUCTION

Mobile technologies have revolutionized how people navigate, experience, and share the world in today's more interconnected world. Mobile applications that provide location discovery, route planning, user reviews, and multimedia content have revolutionized travel in particular. This change is indicative of a larger cultural shift: tourists are no longer merely passive consumers of printed guides or carefully planned tours. They now actively contribute to the production, interpretation, and dissemination of travel experiences.

Furthermore, travel itself has been redefined by the emergence of social media. Experiences are increasingly influenced by how we record and communicate our travels to others, in addition to where we go. An important realization is revealed by this cultural trend: travel is about connecting with people and places, not just about moving around.

This changing environment offers a chance to reconsider how travelers interact with space, history, and each other. In addition to improving individual comprehension, a more contextualized and integrated approach to travel information may encourage a sense of shared memory and involvement in cultural heritage preservation.

1.1. Accessing Cultural Knowledge: Traditional and Digital Methods

People have used a range of tools and techniques to access historical knowledge and cultural heritage throughout history. Information is frequently conveyed through signs, plaques, and printed brochures in physical settings like monuments, museums, and archaeological sites. These resources are usually designed to offer crucial information like historical background, important dates, and the importance of a specific place. They act as guiding references that enhance the visitor's experience on the property and foster understanding of its cultural or historical significance.

Guided tours and local experts provide a more individualized and interactive way to learn about culture for those who want to get more involved. These experiences frequently incorporate audience-specific local interpretations, historical context, and storytelling. Guides can share legends, respond to inquiries, and offer context-sensitive commentary that enhances a site's comprehension. These tours are a long-standing component of heritage engagement in many parts of the world and represent the human aspect of cultural transmission.

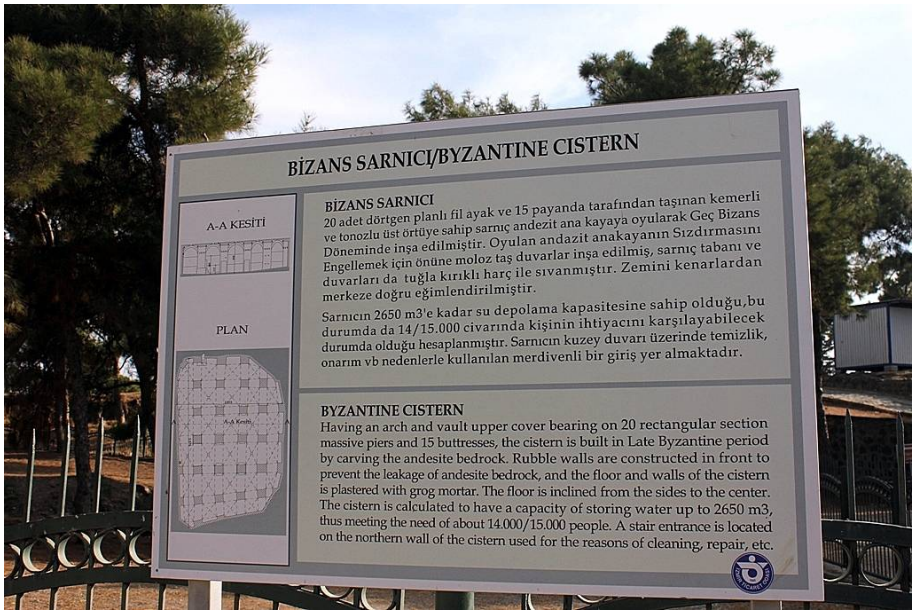

Figure 1.1. An example of traditional on-site signage used to present historical information.

Traveler's access to and engagement with cultural information is increasingly influenced by digital technologies in addition to on-site instruction. A lot of people research places ahead of time by looking through websites, travel blogs, videos, and social media posts. These platforms offer a range of viewpoints, suggestions, and firsthand accounts that enhance a traveler's curiosity and sense of readiness. In addition to visuals, personal narratives, and carefully curated lists that enhance the physical journey, digital content is frequently educational and diverse.

Every one of these methods contributes in a different way to the exploration and comprehension of cultural heritage. Digital sources offer extensive accessibility and personalization, guided tours offer dynamic, narrative-driven engagement, and printed materials offer direct, location-specific reference points. Together, they create a complex learning ecosystem that is influenced by the interaction of personal interests, technology, and physical presence.

This changing environment of cross-cultural interaction informs this project. It takes into account how the traveler's experience is influenced by time, place, and context as well as how new technologies might facilitate situated, real-time learning. By coordinating information delivery with the times and locations where users are most receptive to interaction, the aim is to improve cultural awareness and promote a seamless, integrated process of discovery and comprehension.

1.2. Terms

- *Historical Site Recognition* is a method based on computer vision that allows the mobile application to use the live camera input to identify a historical site. The app enhances the user's experience by triggering extra content, like digital rewards and narration, once it has been identified.
- *Core ML* is the on-device machine learning framework from Apple that allows trained models to be used directly in iOS apps. It guarantees real-time performance without relying on internet connectivity by enabling offline historical site recognition.
- *ResNet (Residual Network)* is an architecture for CNNs intended for image classification applications. In this project, historical landmarks are reliably identified from live camera images using a ResNet-based model.
- *Landmark Coin* is a particular kind of digital memory token that users receive upon successfully visiting and identifying a historical site. The user's cultural journey can be documented by viewing and displaying these coins on their profile.
- *Place Info Panel* is a section of the application that provides comprehensive textual details about a historical location. Along with displaying the relevant landmark coin, this panel has a button that takes the user to Apple Maps, which shows the location of the site.
- *Explore Button* is the main interaction technique used to start the recognition of historical sites. Users can identify nearby landmarks by tapping the Explore button, which activates the camera and processes live image input.
- *Historical Description Panel* is a part of the user interface that controls the audio narration of a historical site. It provides an organized and engaging educational experience by presenting the site's history in sections according to dates or themes.
- *Audio Narration* is spoken material that explains a recognized site's cultural and historical background. For ease of listening, it is divided into historical themes or timelines and is automatically activated upon recognition.
- *Travel Guide* is a user-curated, shareable collection of historical or cultural sites. In addition to written descriptions and route planning, guides can have a theme (such as architecture or religious heritage).
- *On Site Sharings* is a location-restricted feature that only permits users to upload content (such as images or comments) while they are physically present at a historical site. They can also see the shared content that other users have left there.

- *Location Awareness* is the application's capacity to identify the user's GPS location and only unlock content when the user is at or close to a recognized historical site. This guarantees location-specific interaction and authentic content.

1.3. Aims

The goal of this project is to develop a mobile application that combines social and interactive elements with sophisticated historical site recognition to improve cultural exploration. By doing this, it hopes to promote community and engagement around common experiences in addition to giving users relevant, up-to-date information while they are traveling. The specific aims of the project include:

- To encourage people to travel more and explore historical and cultural sites by making the process more interactive and rewarding.
- To provide users with all the essential information they might need during exploration, including historical context, navigation tools, and site-specific content.
- To deliver audio narrations and site descriptions in a way that is informative yet easy to follow, avoiding overwhelming the user with dense or overly academic language.
- To create a friendly UI and a seamless UX that makes the application intuitive, visually appealing, and accessible to all age groups.
- To design a clean and modular screen structure that facilitates smooth navigation between camera, guide creation, and profile features.
- To build a vibrant travel-oriented community by implementing social media-like features such as following, commenting, and sharing guides.
- To offer a personalized and gamified experience through collectible Landmark Coins that are awarded when users visit and recognize a historical site.
- To allow users to view and contribute on-site sharings, making the act of traveling more dynamic and rooted in physical presence.
- To empower users to create and share personalized travel guides that can be followed by others, helping users inspire each other through culturally rich routes.

1.4. Motivation

There has been a discernible change in the way people interact with cultural heritage in recent years. Despite the fact that travel is easier than ever, there are often still few opportunities to interact meaningfully with historical sites. Without completely understanding a place's cultural significance or depth of history, many tourists only stop by once, snap a few photos, and then depart.

By providing a tool that not only recognizes and describes historical sites instantly, but also turns each visit into an opportunity for learning, engagement, and sharing, this project aims to improve that experience. Driven by the rise of digital storytelling, mobile-first exploration habits, and the increasing influence of social platforms on travel, this application seeks to integrate cultural education with modern social interaction.

The contrast between static, conventional information sources (such as plaques or brochures) and dynamic, user-centered digital tools is another powerful motivator. The objective of this project is to close that gap by enabling users to become both explorers and storytellers of their own journeys while providing cultural knowledge at the appropriate time and location.

This project seeks to redefine what it means to explore the past in the digital present by providing users with a compelling reason to visit historical sites, rewarding them for their curiosity, and providing them with tools to connect with like-minded individuals.

1.5. Scope and Limitations

The scope of this project encompasses the development of a mobile application for iOS devices that enables users to recognize historical landmarks, access real-time audio narratives, and share personalized travel guides. The application combines image classification with social interaction features and is developed using the Swift programming language with Firebase as the backend infrastructure. While the application is fully functional within its defined scope, several limitations exist in terms of platform, data, and performance:

1.5.1. Platform-Specific Development

The application is currently available in iOS devices. There is no Android version at this stage, limiting accessibility for a broader user base.

1.5.2. Limited Dataset of Historical Sites

The historical site recognition system is restricted to a predefined dataset consisting of selected landmarks from Istanbul only. The inclusion of historical places from other cities or countries has not yet been implemented.

1.5.3. Image Classification Constraints

The model, based on the ResNet architecture and converted to Core ML format, performs adequately within the bounds of its training data. However, recognition accuracy may decrease for images taken under poor lighting, from unusual angles, or for unregistered landmarks.

1.5.4. Audio-Only Narration

The app provides only audio-based explanations for each recognized site. There is no option for visual or text-based historical summaries, which may limit accessibility in certain contexts.

1.5.5. Single Language Support

As of now, the application supports only the Turkish language. Multilingual support has not yet been added, restricting its usability for international travelers.

1.5.6. Backend Infrastructure and Latency

The application makes use of the Firebase Realtime Database, and the data is stored on US servers. Users who access the app from areas with higher latency to U.S. servers may experience somewhat slower response times as a result.

1.5.7. Offline Usability Limits

While the image recognition model can function offline, some features such as guide sharing and data synchronization require an active internet connection, limiting offline usability.

1.6. Problem Definition

Even though mobile technologies and travel apps are widely used, there are still a number of persistent challenges that travelers must overcome in order to navigate destinations and interact with cultural heritage. An immersive, contextual, and socially relevant experience is frequently not provided by current systems. The primary concerns that this project seeks to resolve are outlined in the following major issues:

1.6.1. Superficial Interaction with Historical Places

Despite the rapid advancement of mobile technology and the abundance of travel-related content, many tourists still only engage with historical and cultural sites on a surface level. Their travels often involve brief activities like taking pictures, reading a few words on signs, or checking off popular tourist spots. As a result, these locations' richer cultural significance and historical narratives are ignored. Rushing through sightseeing routines obscures the opportunity for introspection, emotional connection, and learning. Although informative, the tools that are currently in use, such as printed brochures or museum signage, are static and rarely pique interest or hold attention for long.

1.6.2. Diminished Value of Physical Presence

Being physically present at a cultural site is no longer as unique because digital content is now accessible anywhere, at any time. A sense of overfamiliarity may result from travelers having access to videos, reviews, and anecdotes about a place before they even arrive. The significance of on-site exploration decreases as content becomes less tied to location. By providing the same content regardless of the viewer's physical location, modern social media platforms intensify this effect. This causes a gap between perception and presence; even though tourists are physically present at a landmark, their experience is still routine or mentally remote rather than rich and engaging.

1.6.3. Fragmented and Unstructured Exploration

Rather than providing a cohesive or escorted cultural experience, contemporary travel planning frequently depends on a patchwork of disjointed sources, such as blogs, vlogs, forums, or mapping tools. This fragmentation results in a lack of chronological or thematic flow during exploration, thereby reducing opportunities to discover lesser-known sites or understand the historical connections between them. Exploration can be ineffective and confusing

without well-organized and contextually aware navigation, especially for first-time visitors. Instead of exploring cities or landmarks via carefully considered and important routes, users are faced with time-consuming planning procedures and disorganized suggestions that do not provide a comprehensive understanding of a location.

1.7. Requirements

1.7.1. Functional Requirements

1.7.1.1. Real-Time Historical Site Recognition. The application enables users to recognize historical landmarks in real time using their device's camera. This feature forms the core functionality of the app and allows users to receive instant visual feedback and contextual information as they explore cultural sites. The recognition system is powered by an image classification model integrated through Core ML.

1.7.1.2. Travel Guide Creation and Sharing. Users can create personalized travel guides by combining recognized landmarks, notes, and recommendations. These guides can be saved and shared with the broader community within the app, enabling others to follow curated routes or discover hidden gems. The system supports liking and bookmarking guides, which requires proper backend synchronization and correct functionality to ensure users can reliably access their saved or favorited content.

1.7.1.3. Landmark Coin (Digital Memory Token). Upon visiting and successfully recognizing a historical site, users receive a unique digital collectible called a Landmark Coin. These tokens act as memory badges and can be viewed within the user's profile. This gamification element encourages exploration and helps users visually track the landmarks they have discovered.

1.7.1.4. On-Site Social Sharing. Users are allowed to post comments and share photos, but only when physically present at a recognized landmark. This location-based limitation ensures authenticity in shared content and builds a sense of place-specific community interaction.

1.7.1.5. Place Info Panel. Each recognized site features a dedicated information panel that displays historical descriptions, the image of the earned Landmark Coin, and a quick link to

open the site's location via Apple Maps. This panel offers users both educational content and navigational support in one place.

1.7.1.6. Historical Description Panel. This panel controls the playback of the audio narration, offering segmented historical storytelling. Users can skip between narrated sections to match their interests and pace of exploration.

1.7.1.7. Follow System. To foster a social layer, users can follow others, view their public travel guides, and see their collected Landmark Coins. This system enhances interaction and community-building among culturally curious users.

1.7.2. Non-Functional Requirements

1.7.2.1. User Interface and Experience. Developed using SwiftUI, the app interface is clean, accessible, and user-centric. Careful attention has been given to color schemes, typography, and layout to support readability and a smooth interaction flow.

1.7.2.2. Performance. Real-time operations—especially camera-based recognition and audio playback—are optimized for high FPS rates to maintain responsiveness. All features are designed to run efficiently on modern iOS devices.

1.7.2.3. Accuracy and Robustness. The ResNet-based model used for site recognition was extensively trained and tested under various lighting and environmental conditions to ensure reliable performance in diverse real-world scenarios.

1.7.2.4. Data Synchronization. Firebase services (Firestore, Authentication, and Storage) support real-time synchronization of user-generated content, follow actions, and travel guides. This ensures consistency and reliability across sessions.

1.7.2.5. Offline Accessibility. Once the recognition model is downloaded, core features such as landmark recognition and audio narration can function offline. This increases the app's usability in areas with limited or no internet access.

1.7.3. Software and Hardware

1.7.3.1. Software Stack.

- **Frontend:** Swift & SwiftUI (iOS)
- **Machine Learning:** PyTorch (training), Core ML (deployment)
- **Backend:** Firebase (Firestore, Authentication, Storage)

1.7.3.2. Hardware Requirements.

- iPhone with iOS 15 or later
- Device with camera functionality
- Internet connection (for model download and social features)

1.7.4. Planned Tests

1.7.4.1. Model Accuracy Metrics. The classification model was evaluated using precision, recall, F1-score, and confusion matrices to verify robust and consistent recognition across various historical site categories.

1.7.4.2. Firebase Latency Tests. Read and write latencies for common user actions—such as creating guides, posting comments, and following users—were measured under varying network conditions to evaluate system responsiveness.

1.7.4.3. User Feedback Surveys. A group of ten participants completed a Likert-scale survey (1 to 5) assessing app usability, recognition accuracy, feature usefulness, and overall user satisfaction.

2. RELATED WORK

Mobile applications have completely changed how people travel, explore, and interact with cultural heritage in recent years. This section examines earlier research and technological advancements in the areas of real-time image recognition, social sharing tools, audio tour platforms, and travel apps. It examines the advantages and disadvantages of current solutions like Google Lens, Piri, and Gezibilen and contrasts them with popular applications like TripAdvisor. This section also emphasizes the function of deep learning in mobile environments, specifically the application of image classification models for tasks involving recognition. These technologies help contextualize the project's contribution to mobile cultural exploration and serve as the basis for its core functionalities.

2.1. Travel Applications and Visual Recognition Technologies

By offering location-based recommendations and cultural insights, some of travel-related apps and visual recognition technologies have been created to enhance user experiences.For instance, Google Lens is widely used to identify well-known paintings, artwork, and everyday objects with the help of smartphone cameras. Quick access to online information about recognized items is made possible by its deep learning models. But it isn't designed to tell stories about history or culture, particularly when it comes to visiting heritage sites [1].

Applications like Piri and Gezibilen, which were created locally, concentrate on audio-guided tours specifically designed for Turkish cities and historical sites. Gezibilen offers downloadable tour packages and multilingual support, while Piri employs skilled narrators to tell captivating tales. However, both apps rely on preset routes and passive listening experiences, and neither allows user-generated content, interactive exploration, or real-time recognition. Additionally, they lack gamified components that are necessary for this project, such as collectible items or in-app rewards.

On the other hand, the application created for this project supports a more individualized and interesting exploration approach by introducing on-device historical site recognition with interactive narration and collectible Landmark Coins. Instead of being restricted to static tours, users can explore locations in real time, get immediate feedback, and compile a collection of memories connected to their travels.

2.2. Social Media-Integrated Exploration Platforms

Social media dynamics are increasingly influencing travel content, as sites like YouTube, Instagram, and Tripadvisor allow users to share itineraries, photos, and reviews with audiences around the world. Specifically, Tripadvisor provides a database of user-generated reviews, ratings, and location-based travel recommendations. However, the platform's interaction model is not necessarily linked to location-aware storytelling or real-time presence; rather, it serves primarily as a review aggregator [2].

Instagram and comparable platforms facilitate the sharing of lively travel content, but they are devoid of location-gated visibility, cultural context, and organized guidance systems. In other words, despite providing social validation and motivation, these platforms do not promote real face-to-face interactions or educational engagement. This project bridges that gap by limiting the posting of images and comments to users who are physically present at a historical site, thereby ensuring authenticity and encouraging community engagement around culturally significant sites.

2.3. Deep Learning Models on Mobile Devices

Deep learning models can now be deployed on smartphones with reduced latency and without constant internet access thanks to frameworks like TensorFlow Lite and Apple's Core ML. These technologies enable real-time inference, enabling features like augmented reality, image recognition, and natural language processing, while preserving offline functionality and data privacy [3].

To ensure smooth user interaction even in travel situations with poor connectivity, Core ML was chosen for this project to power the on-device recognition of historical landmarks.

2.4. Image Classification Models: ResNet, MobileNet, and VGGNet

In computer vision tasks, a number of convolutional neural network architectures have gained widespread use. The vanishing gradient issue can be avoided when training very deep networks thanks to ResNet's use of residual connections. For intricate image recognition tasks, such as the classification of intricate architectural structures and historical landmarks, its high accuracy and stability make it perfect [4].

MobileNet, on the other hand, places a strong emphasis on efficiency and lightweight

design, which makes it ideal for low-power devices, though frequently at the expense of somewhat lower accuracy. Despite its reputation for depth and simplicity, VGGNet is computationally costly and less suitable for mobile applications because of memory limitations.

3. ANALYSIS AND DESIGN

3.1. System Architecture

The application's central component is a mobile system that uses three main pillars—intelligent landmark recognition, dynamic guide creation, and location-bound social sharing—to improve the cultural travel experience. Shared modules that control data persistence, authentication, and real-time updates are integrated into each of these sections. To maintain a seamless user experience under various network conditions, all interactions are made to be asynchronous. The following essential architectural elements make up the system:

3.1.1. Image Classification Module

This module uses the device's camera to process user-submitted images. It recognizes historical landmarks using a pre-trained image classification model. After a classification result is generated, it is compared to the pertinent data kept in the local database of the application. This layer is in charge of turning on location-bound features and starting the storytelling process.

3.1.2. Application Interface

The user interface is structured around three main screens.

3.1.2.1. Home Screen. Displays a feed of user-generated and carefully chosen travel guides. Users have the option to view, like, and save other people's guides or access their own saved guides. It serves as the center for finding cultural paths and suggestions.

3.1.2.2. Explore Screen. Includes an interactive map that plots historical sites. In order to enable real-time landmark recognition, the user can then tap the explore button to turn on the camera. Audio narration starts and on-site sharing are made available after a recognized location has been confirmed.

3.1.2.3. Profile Screen. Acts as the individual dashboard for the user. Users can manage their social interactions, including friend requests and following, access their personal travel

guides, and view the Landmark Coins they have earned. This screen starts the guide creation process.

3.2. Functionalities

The main features of the mobile application that support its three main goals—improving cultural discovery, facilitating customized travel planning, and encouraging community-based interaction—are described in this section. A particular system behavior or user-facing feature that enhances the overall user experience is described in each subsection. These features, which range from location-bound social sharing to guide creation, authentication procedures, and real-time landmark recognition, constitute the application's operational core. The way these components are organized, activated, and linked to create a unified and adaptable cultural travel guide is explained in the ensuing subsections.

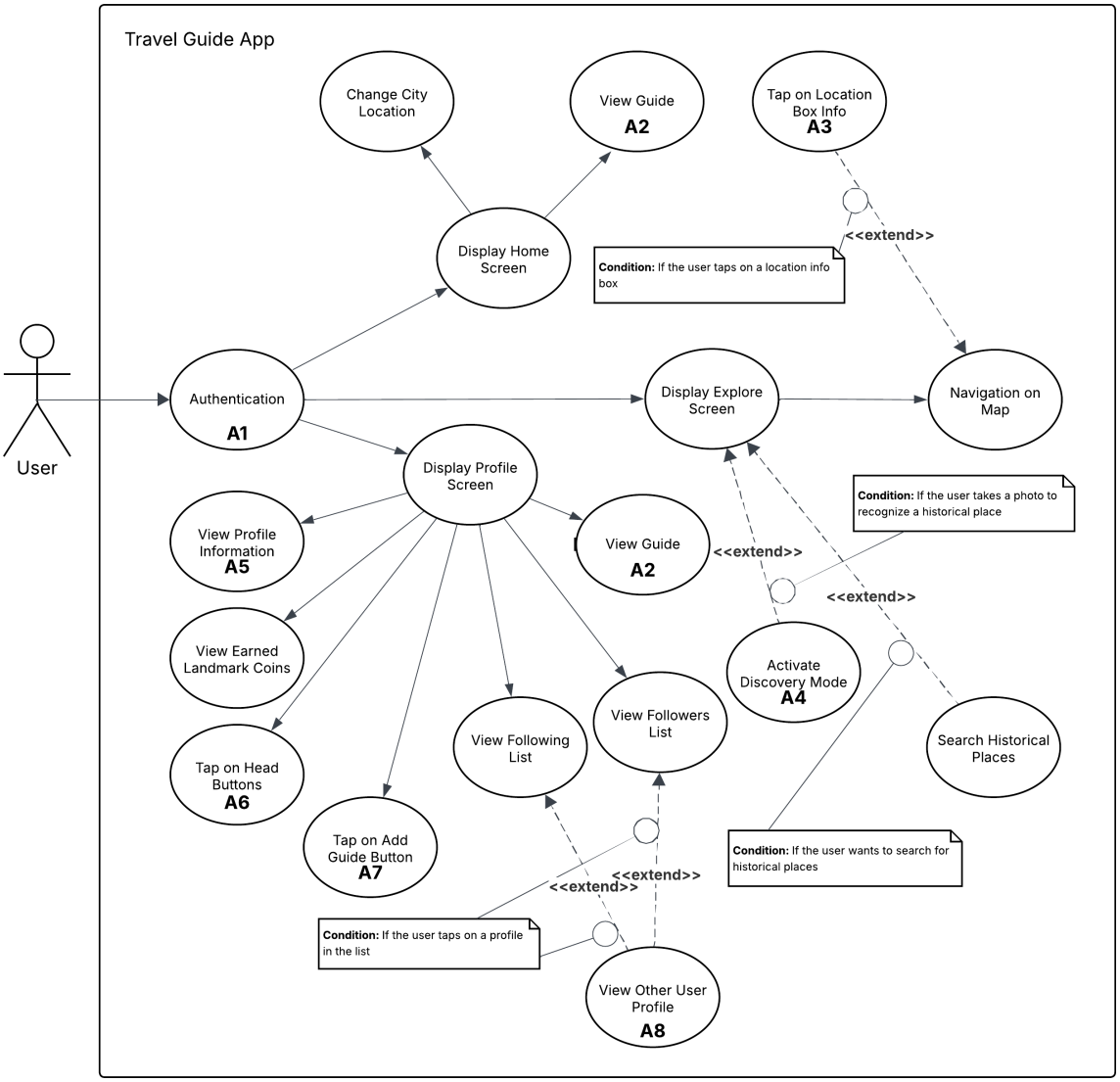

Figure 3.1. General Use Case Diagram of the Application

Figure 3.1 illustrates the overall user interaction flow within the application. It highlights the primary use cases across the three main screens: Home, Explore, and Profile. After completing the authentication process, the user can proceed with a number of tasks, including managing their social connections, viewing guides, creating customized travel guides, and using discovery mode to identify landmarks. Conditional extensions like location tapping, profile navigation, and real-time recognition are also included in the diagram. Together, these use cases highlight the system's primary features and show how users switch between the app's various modules according to their objectives and situation.

3.2.1. Authentication

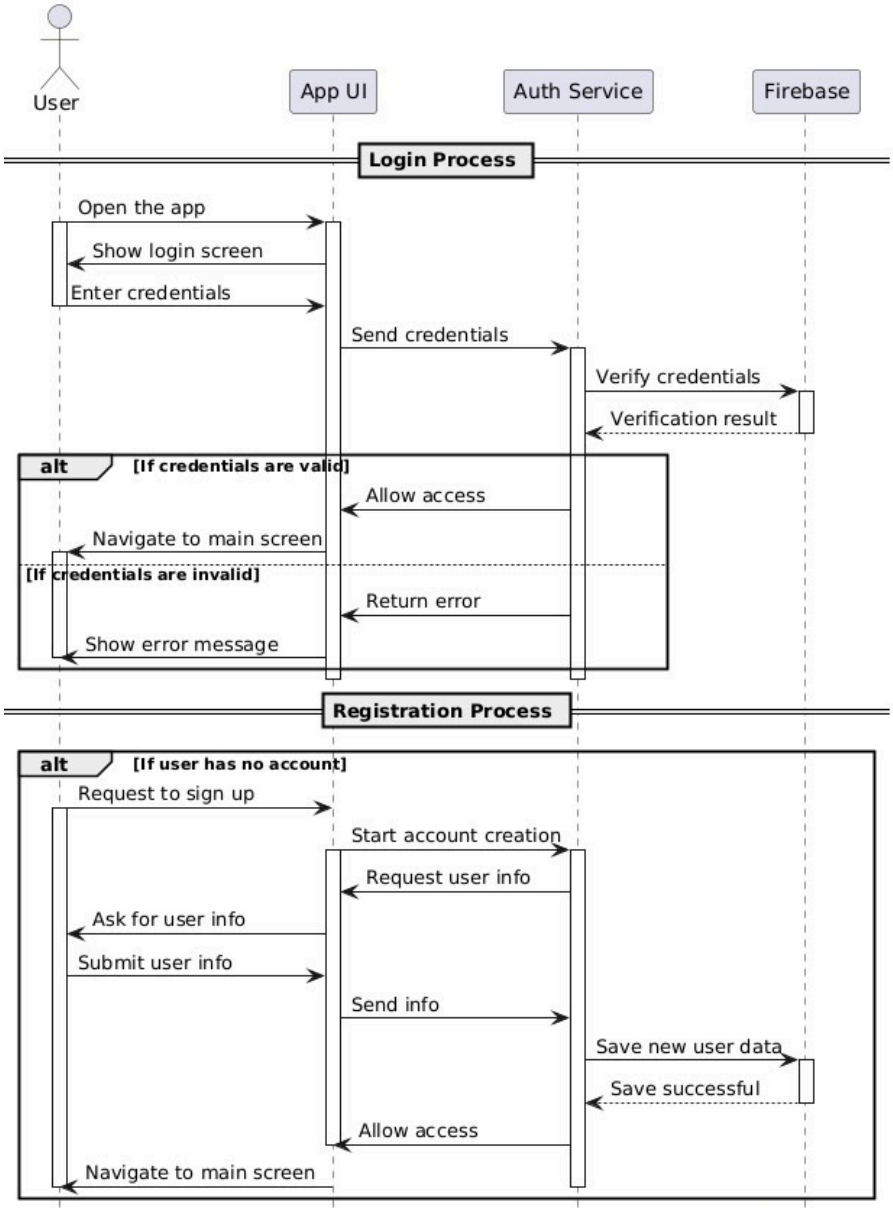

Figure 3.2. Login and Registration Sequence Diagram

Enabling secure access to the application begins with authentication. Users can use this feature to create a new account or log into if already has an account. Credential validation, successful redirection to the main interface, and error handling in the event of failure are all part of the authentication flow.

The Login and Registration Sequence Diagram, which depicts the sequential relationship between the user, application interface, authentication service, and Firebase backend, is shown in Figure 3.2. The system checks user credentials during login, and user data is gathered and stored during registration. If both flows are successful, access to the home screen is granted at the end.

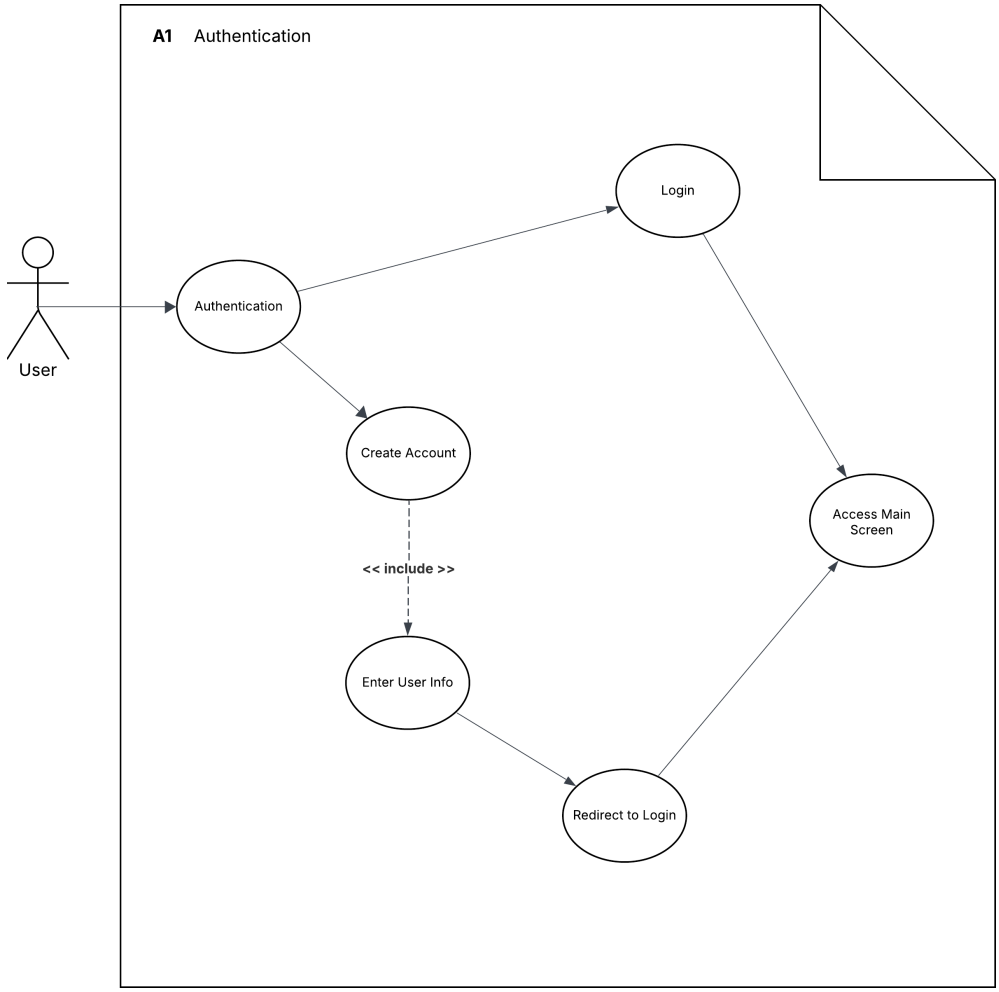

Figure 3.3. Authentication Use Case Diagram

Following authentication, users are taken to the application's main screen, which houses all of its essential features. To guarantee usability and clarity, relevant error messages are shown in the event that input is invalid during login or registration.

The authentication procedure is shown in simplified form in the use case diagram above (Figure 3.3). It describes potential user actions, like creating an account and logging in, and how they result in access to the main application interface.

3.2.2. Viewing Travel Guides

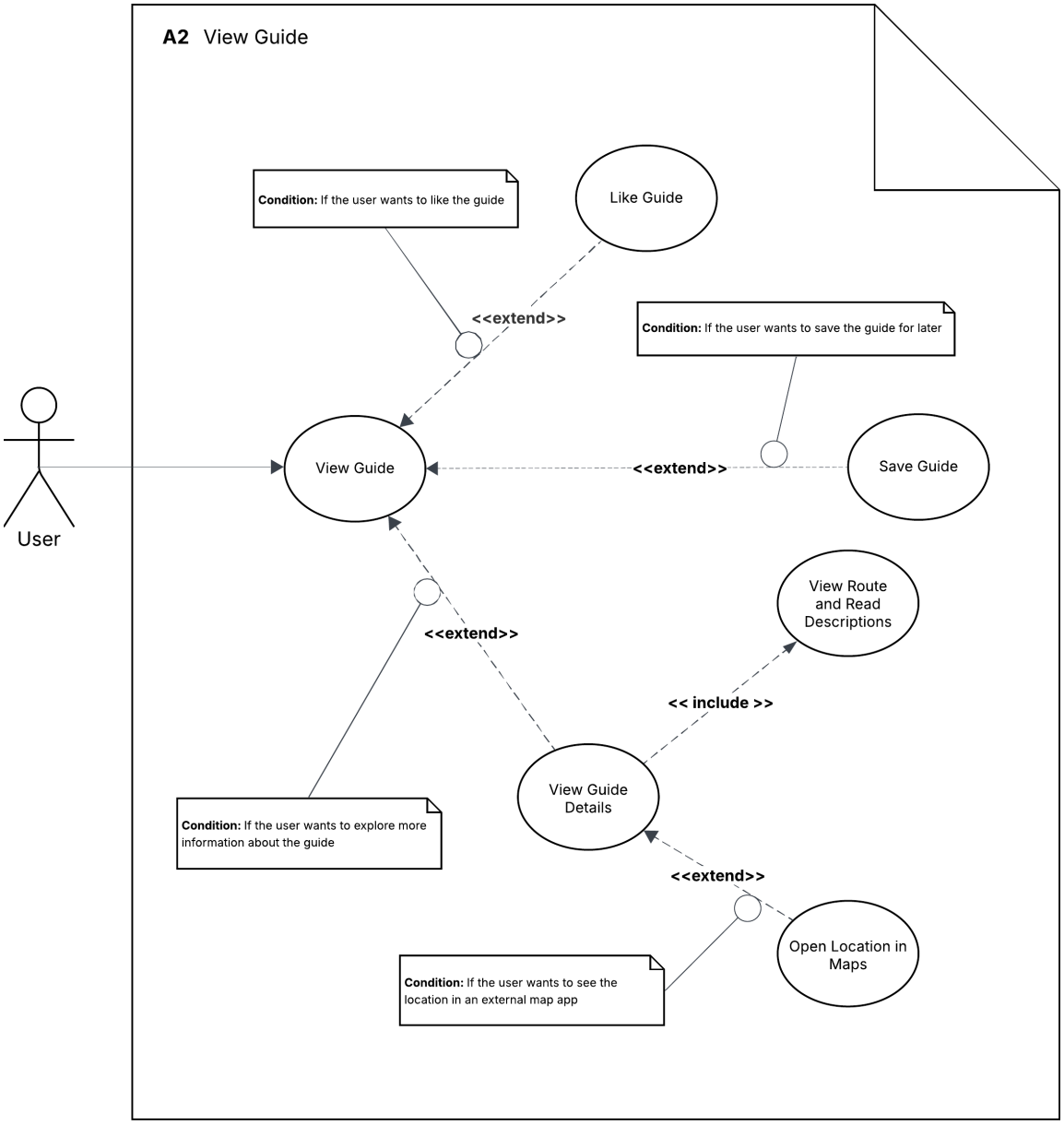

Figure 3.4. Viewing Travel Guide Use Case Diagram

Users can engage with carefully chosen cultural content produced by other users or by themselves within the application while viewing travel guides. Users can access route descriptions, landmark details, and associated navigation tools once they open a guide.

As illustrated in Figure 3.4, after opening a travel guide, users can perform several extended actions such as:

- Liking the guide, if they thought it is helpful
- Saving the guide, to add it to their personal collection for future reference
- Using the ”View Guide Details” extension to view route descriptions and details
- Opening the location in an external map app for navigation assistance

These interactions are structured using an extension model, meaning users only access these actions if they actively request them, thereby keeping the interface clean and focused while still supporting richer exploration when desired.

By facilitating meaningful interaction with community-generated content in addition to passive consumption, the viewing functionality increases user engagement. It creates a feedback loop that highlights popular or useful guides for other users by allowing likes and saves.

3.2.3. Creating Travel Guides

The travel guide creation process allows users to enter personalized cultural routes, save them, and share them with the community (Figure 3.5).

Users start the guide creation process by tapping the ”Add Guide” button on the profile screen. The title, general description, and category of the guide are among the basic details that users can enter in the form interface that is opened by this action.

The user is then taken to the stop editor screen, where they can add more than one route stop. Every stop has a map with the precise location and thorough descriptions. Users are given several choices after completing the content input process:

- Users can save the guide as a draft if they wish to complete it later.
- Users can continue editing from a previously saved draft to update its content.
- Users can share the guide by uploading a cover image and publishing it for others to see.

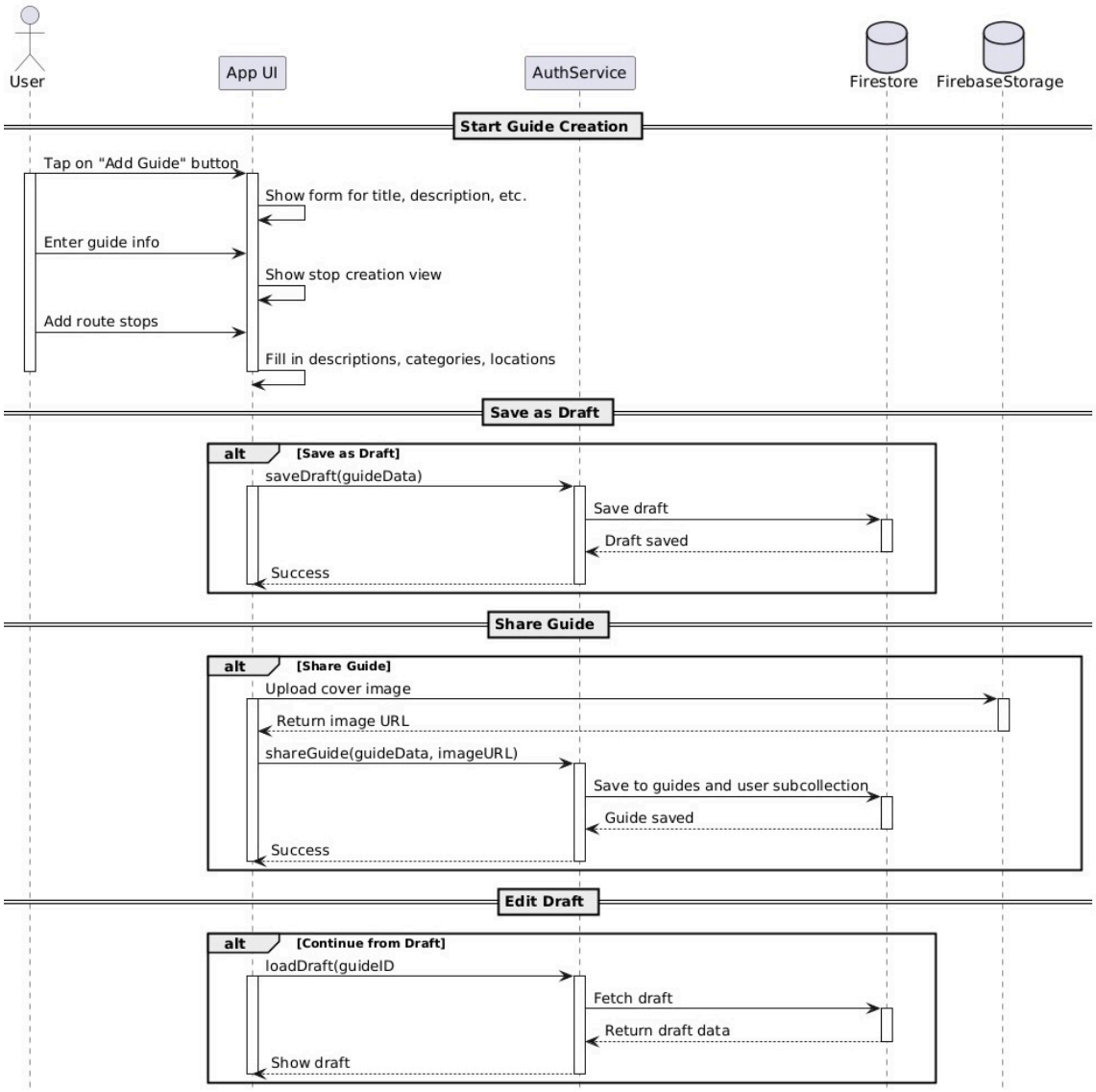

Figure 3.5. Creating Travel Guide Sequence Diagram

Both short-term exploratory users and those who want to add in-depth cultural content to the platform can engage with the interaction. The experience is more user-friendly and appropriate for mobile usage conditions thanks to the flexibility of multi-step editing and draft saving.

The use case diagram for the guide creation module, which shows the included and extended actions in the process, is shown in Figure 3.6.

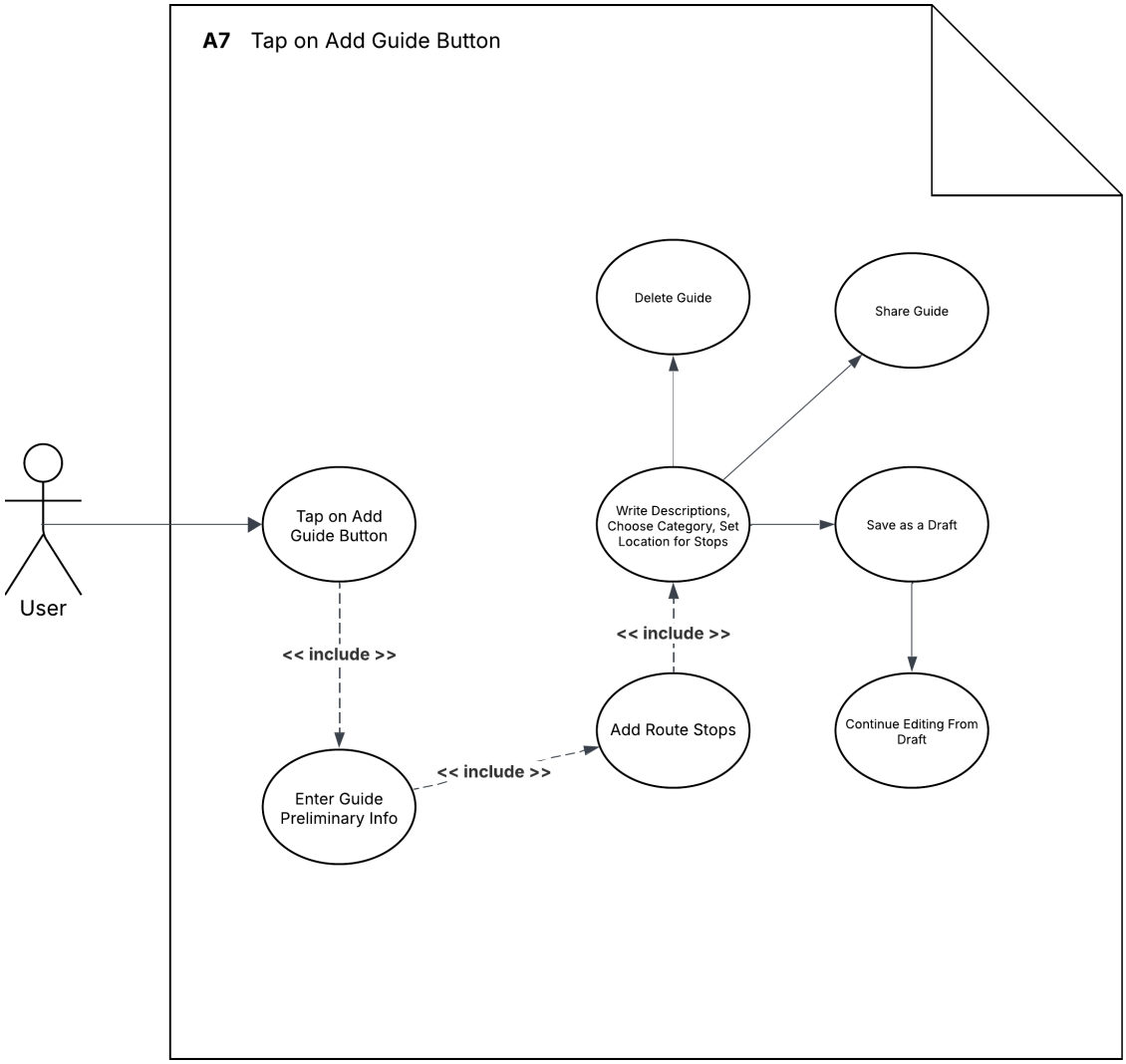

Figure 3.6. Add Guide Button Use Case Diagram

3.2.4. Maps and Location Information

Users can explore historical locations based on their current or chosen city using the interactive map, which is a key interface in the application. The map screen's visual layout emphasizes notable locations, each of which is indicated by a clickable location information box. These boxes act as entry points for more detailed information.

A specific information view that shows the landmark's historical background and descriptive metadata is displayed when a user taps on a location info box. This enables users to explore the site without having to exit the map or change views. In order to facilitate navigation to the destination, the user can also choose to extend the action by opening the landmark's coordinates directly in an external map application.

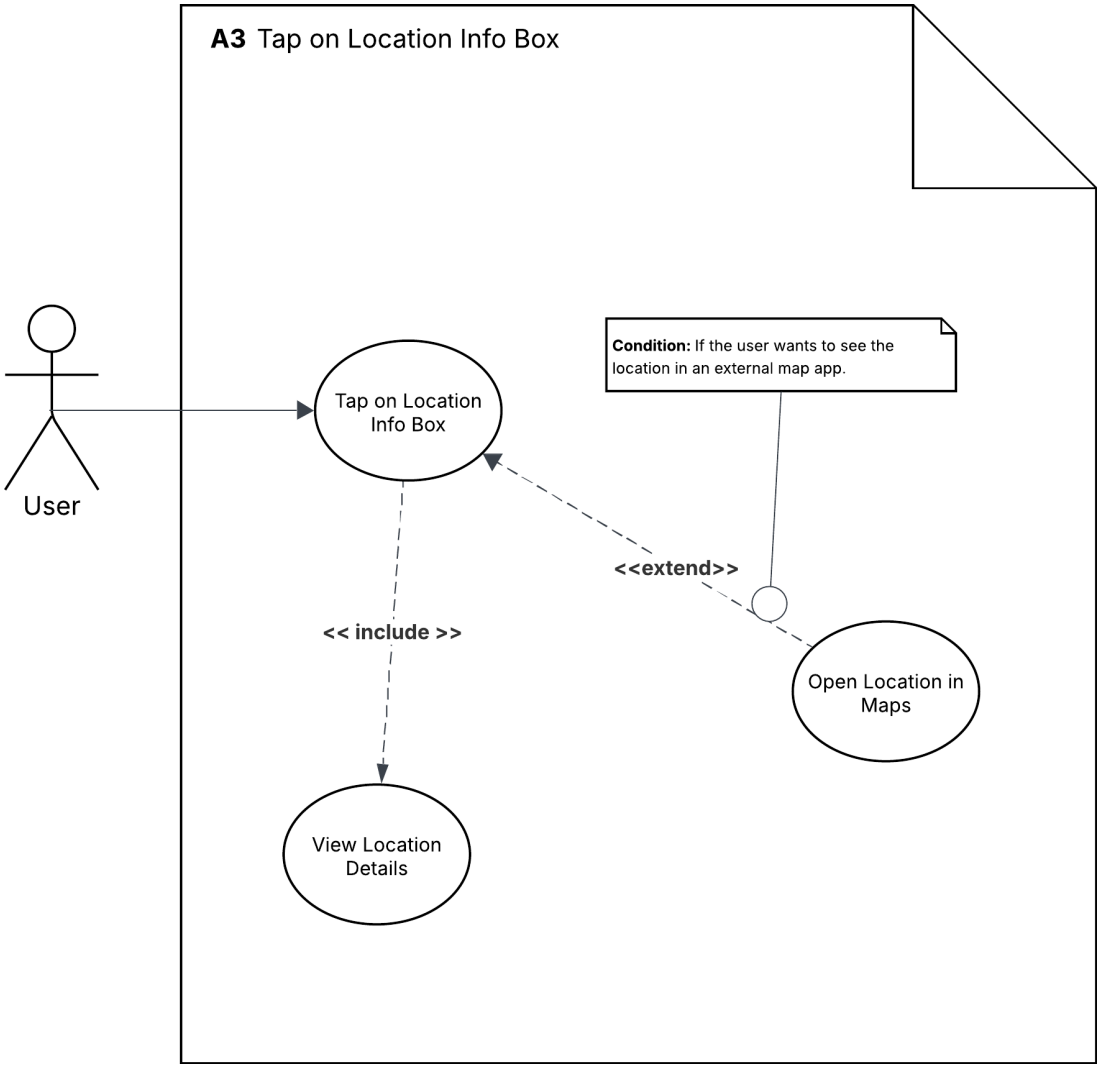

Figure 3.7. Tap on Location Info Box Use Case Diagram

Figure 3.7 shows the use case for location info box interaction.

Exploration and usability are balanced in this modular approach. Without interfering with the app's natural flow, it guarantees that users are both empowered and informed to make location-based travel decisions in real time.

3.2.5. Landmark Recognition

The landmark recognition process begins when the user taps the "Activate Discovery Mode" button on the Explore screen. This allows the user to take a picture of a nearby historical landmark by opening the device's camera view. After receiving the captured image, the onboard image classification model attempts to match it with a predefined list of identi-

fiable historical sites. To confirm that the user is physically near the predicted location, the system obtains the user's GPS coordinates after the classification result is generated (shown in Figure 3.8).

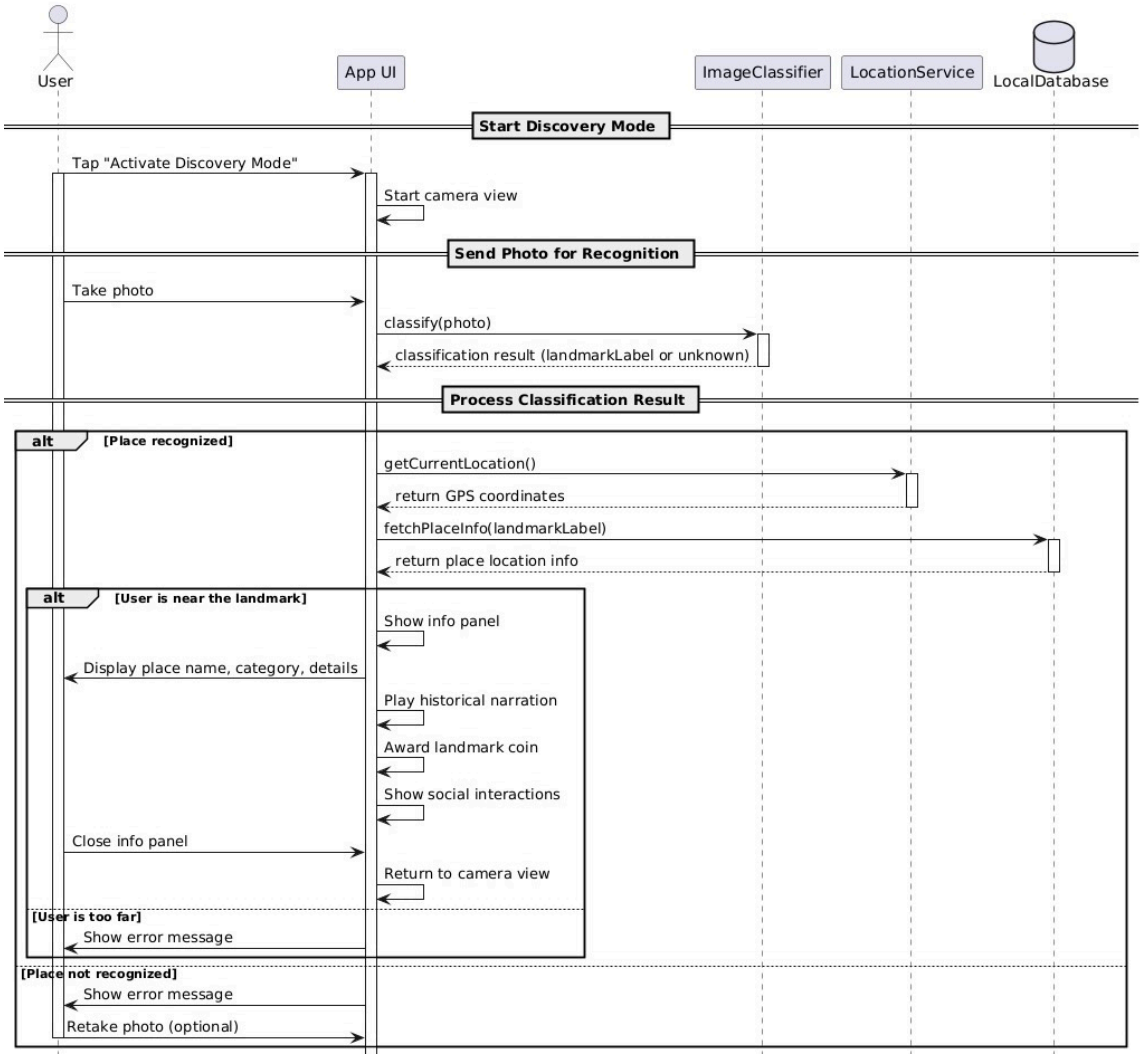

Figure 3.8. Landmark Recognition Sequence Diagram

An interactive information panel with the site's name, type, and historical details is displayed if the visual match and proximity check are successful. A segmented audio narration of the site's history starts at the same time the user receives a Landmark Coin. Additionally, the panel displays location-bound social media content, such as other people's shared images and comments. The user can exit the panel after interacting with the content and go back to the camera interface to continue exploring.

The appropriate error messages are displayed if the model is unable to identify the image or if the user is too far away from the location. The user might be asked to take another

picture or approach the landmark in these situations.

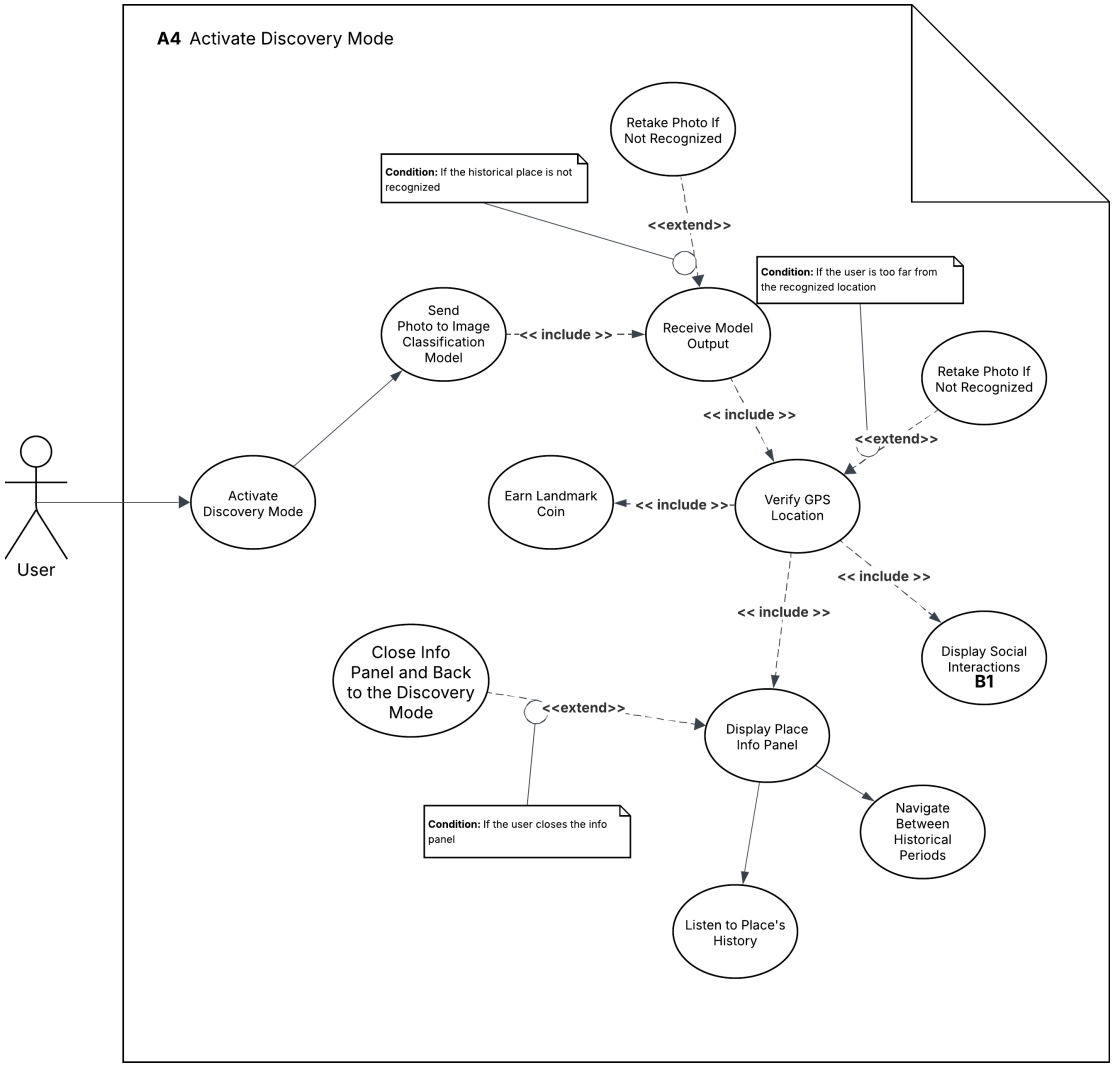

Figure 3.9. Landmark Recognition Process Use Case Diagram

The use case diagram illustrates the recognition process's conditional logic and decision flow, as seen in Figure 3.9. In order to guarantee strong user feedback and seamless re-engagement, it incorporates extensions for frequent failure conditions (such as misclassification or GPS mismatch). From taking the picture to showing the historical narrative and giving out coins, every stage in the diagram is designed to optimize location-based interactivity and cultural engagement.

3.2.6. On-Site Sharing System

The on-site sharing system introduces a original feature that ties user-generated content directly to physical presence at a historical location. Unlike traditional social media platforms

where content can be viewed from anywhere, this system ensures that shared posts are only visible to followers who also visit the same location. Sharing is given emotional significance and authenticity by this restriction, which turns it from a passive broadcast into an immersive exchange.

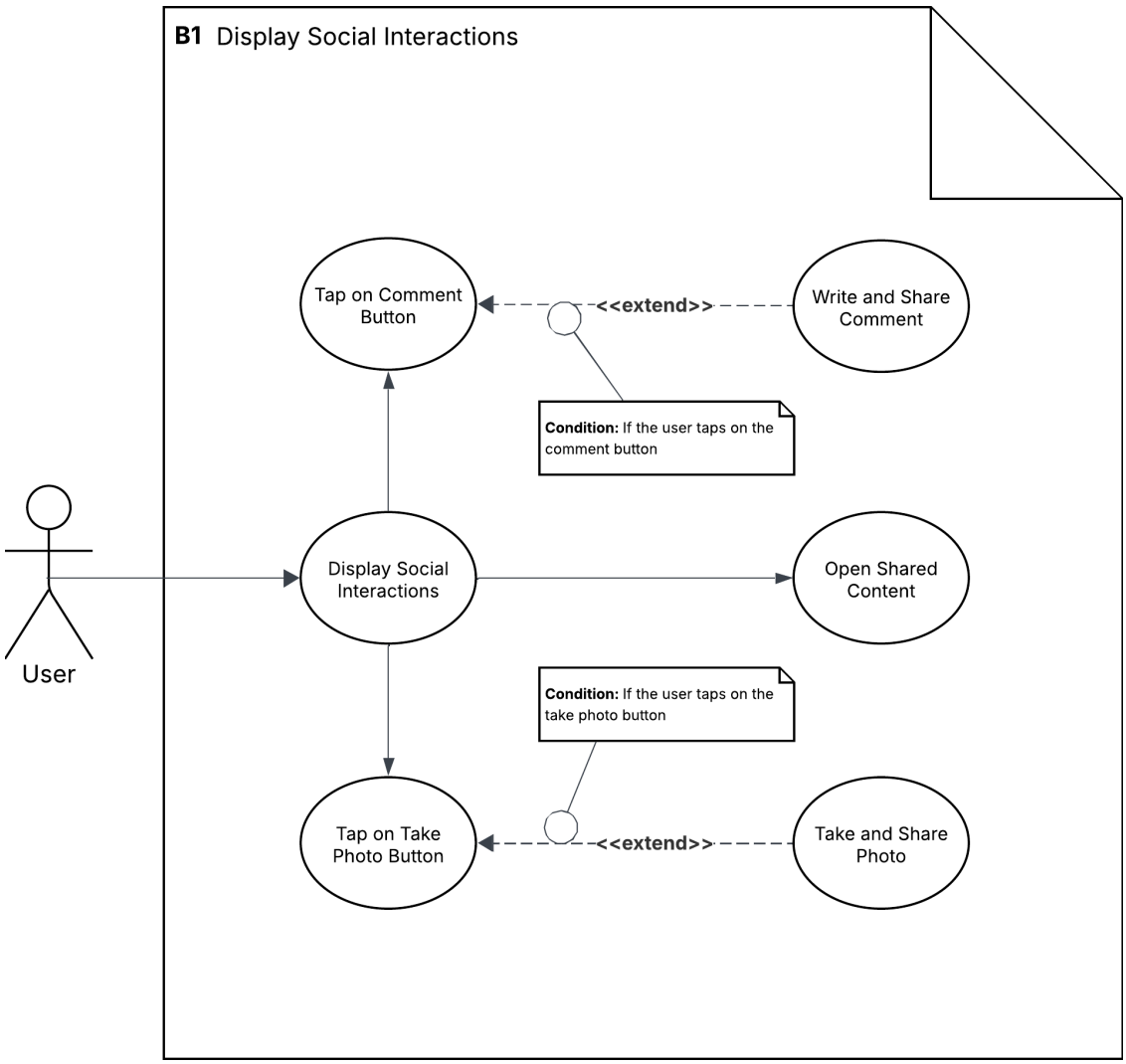

Figure 3.10. On-Site Social Interactions Use Case Diagram

Users can engage with existing shared content by opening comments or photos associated with a specific location while they are at a recognized historical site. They can also take part by tapping on the write comment or take photo buttons. The ability to both consume and produce content on-site fosters a more meaningful connection between users and the cultural site, as well as among members of the user's social circle.

As shown in Figure 3.10, the system flow begins with displaying social interactions for a recognized location. Users can extend this action by commenting or posting a photo, but

only if they are physically present. In addition to maintaining the content's relevance, this presence requirement improves digital travel storytelling's sense of reward, engagement, and authenticity.

3.2.7. Audio Narration and Historical Description

By turning passive sightseeing into an engaging storytelling session, the audio narration and historical description feature aims to enhance users' cultural experiences. The system starts playing location-specific historical content in a narration panel after identifying a historical landmark and confirming that the user is physically present at the location. Users can move between various time periods or topics because this content is organized into chronological or thematic segments rather than being presented as a single, continuous track.

This modular structure allows users to personalize their listening experience—skipping certain parts, revisiting others, or pausing as needed. For instance, one section might discuss the site's architectural past, while another might examine its significance in a historical event. This kind of segmentation guarantees that the information is both comprehensible and pertinent to the context, which helps users stay focused, particularly in outdoor settings where distractions are frequent.

The narration encourages the user to pay attention to specifics of the landmark while they listen, going beyond merely imparting information. This improves knowledge retention and emotional engagement by forging a deeper bond between the story and the surroundings.

The info panel interface and the landmark recognition module are closely linked with the narration system. The historical storytelling feature is activated once the system verifies the identity of the landmark and the user's GPS proximity. This guarantees that users only encounter accurate and pertinent content when they are ready to interact with it both physically and contextually.

In addition to encouraging self-directed exploration, this feature can be used in place of conventional guided tours, providing accessibility, customization, and immediacy in a seamless experience.

3.2.8. Profile Interactions

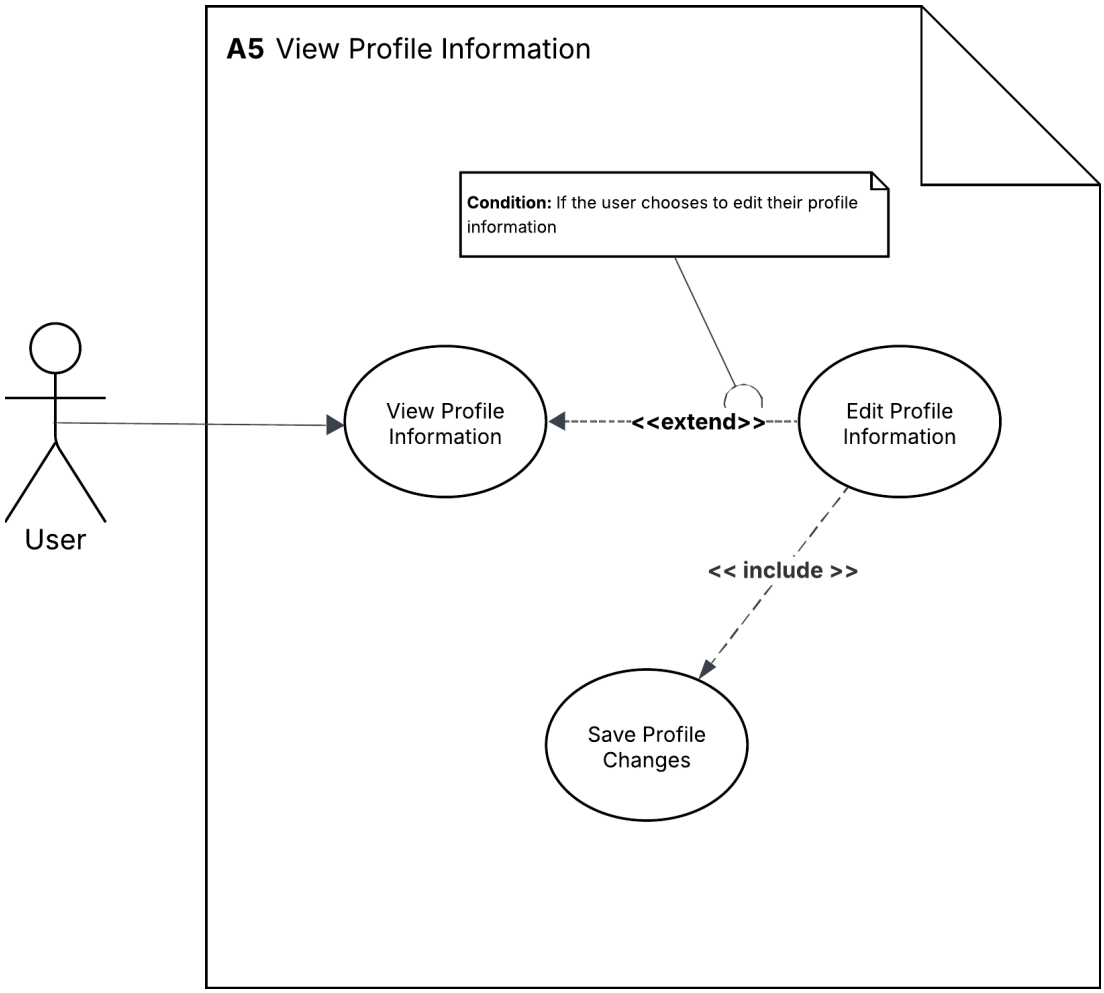

Figure 3.11. Viewing and Editing Profile Information Use Case Diagram

Essential interactions occur on the user's profile screen, which serves as a personal hub. Users can view and modify their profile photo,country and username. The system enables the user to make changes and save them instantly if they choose to update any information (see Figure 3.11). To guarantee uniform identity presentation, these changes are synchronized throughout sessions.

Using the hamburger menu on the profile screen, users can also log out of the system. Because it is conditionally triggered, the logout option is only shown when it is relevant. The most pertinent actions are easily accessible, resulting in a seamless and concentrated user experience.

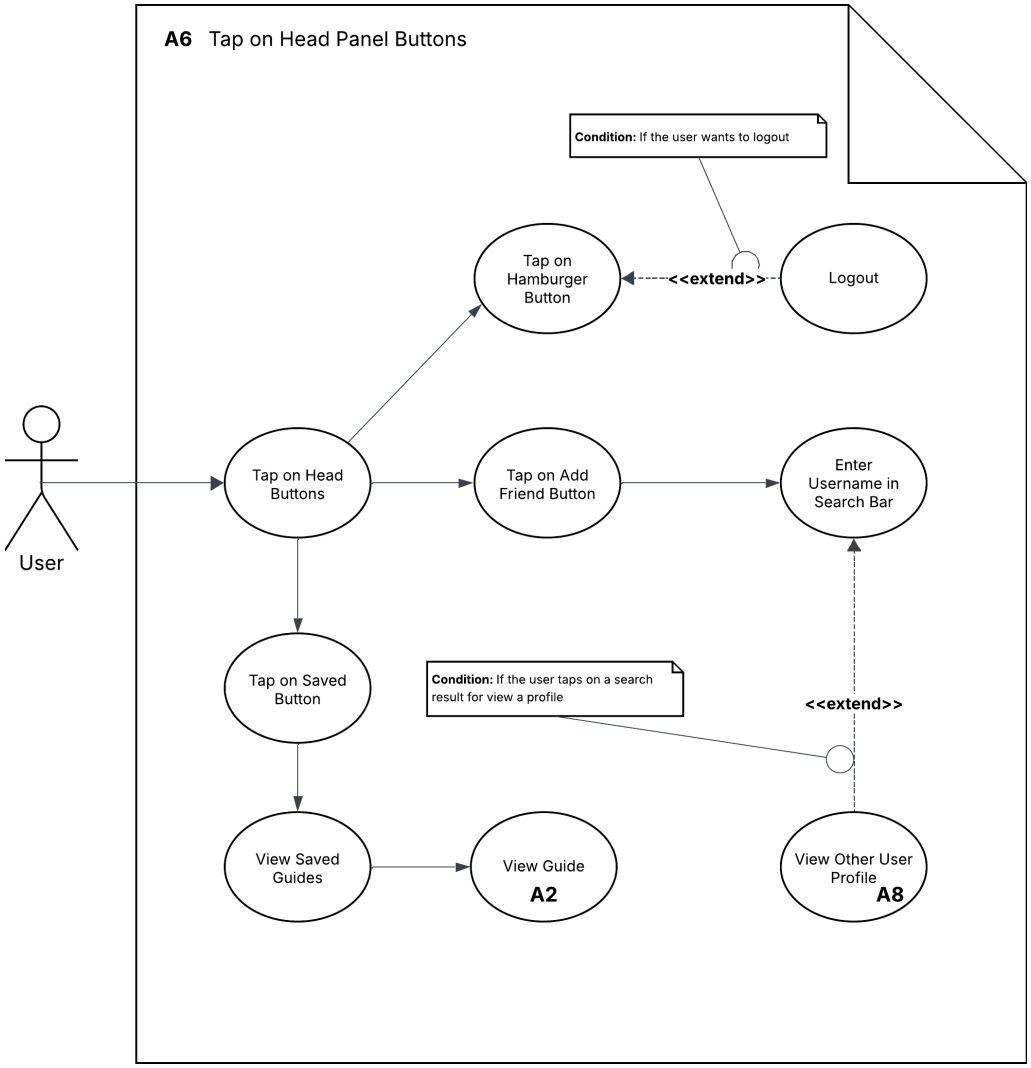

Figure 3.12. Profile Menu and Social Actions Use Case Diagram

The profile interface allows users to manage personal information and carry out various social actions (see Figure 3.12). These include adding other users as friends, viewing saved guides, and regaining access to shared guides. The user is taken to the profile of the chosen person, where they can see their shared content and social media information, after entering their username in the search bar. A richer and more connected experience within the application ecosystem is made possible by each of these actions, which branch out from the core interaction with the profile.

3.2.9. Interaction with Other Users

The ability to view and interact with other users' profiles is one of the application's primary social features. The profile page of the user is accessed when a user taps on another

user's profile, either from the home feed or from their followers or following list. They have access to public interaction data, including likes and on-site posts, as well as shared travel guides and Landmark Coins.

The Following feature is a crucial component of this interaction. A follow button will show up if the user is not already following the profile. Tapping it will give them access to additional content, like on-site shares linked to particular locations. The user also has the option to unfollow if they are already following. This feature allows context-specific content visibility in addition to fostering connections; users can only see specific posts if they have followed the user and visited the same place in person.

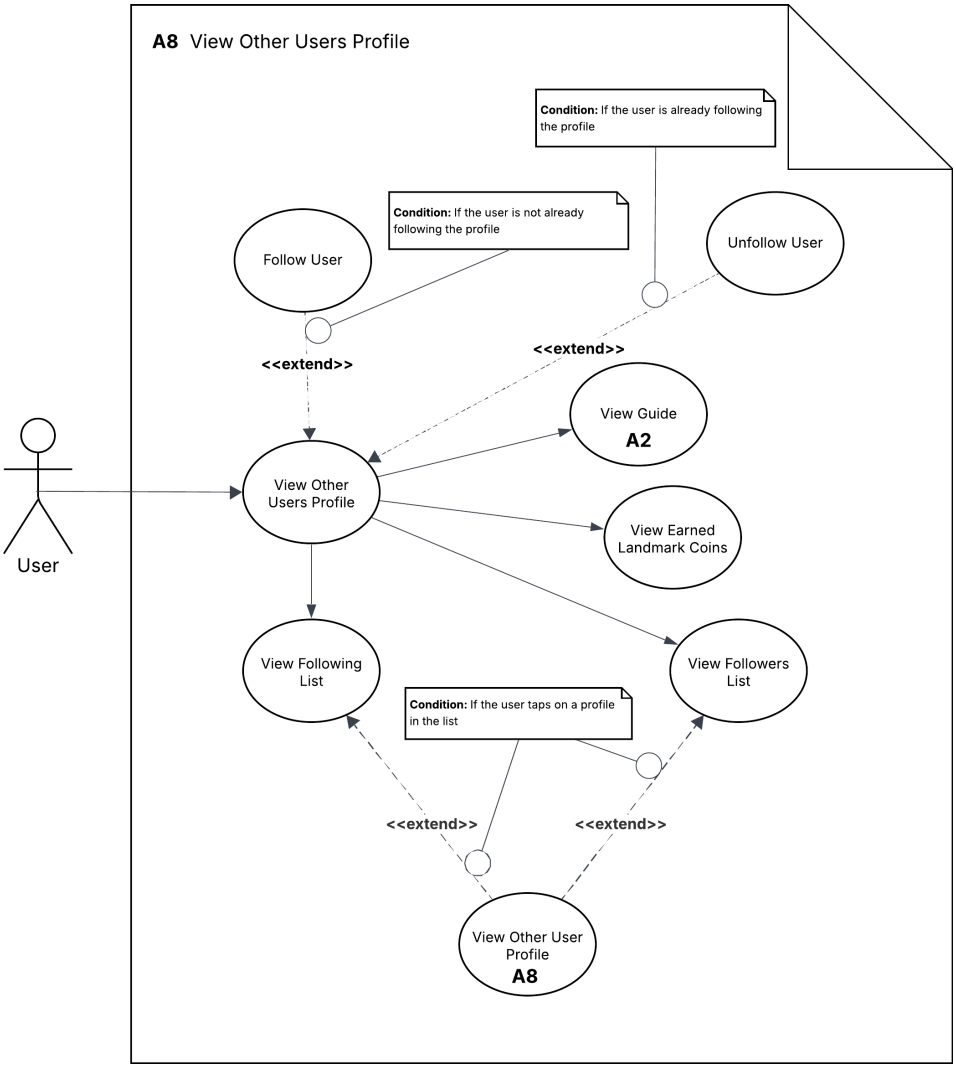

Figure 3.13. Viewing and Interacting with Other Users Use Case Diagram

Users can also look through any public profile's lists of followers and followers. They can recursively view more public profiles by tapping on any name in the list if they so choose.

Users with similar travel interests can interact more easily and exchange cultures thanks to this interconnected structure.

The primary user actions for interacting with other profiles, such as follow/unfollow and content viewing privileges, are captured in the use case diagram, as shown in Figure 3.13.

3.3. ER Diagram

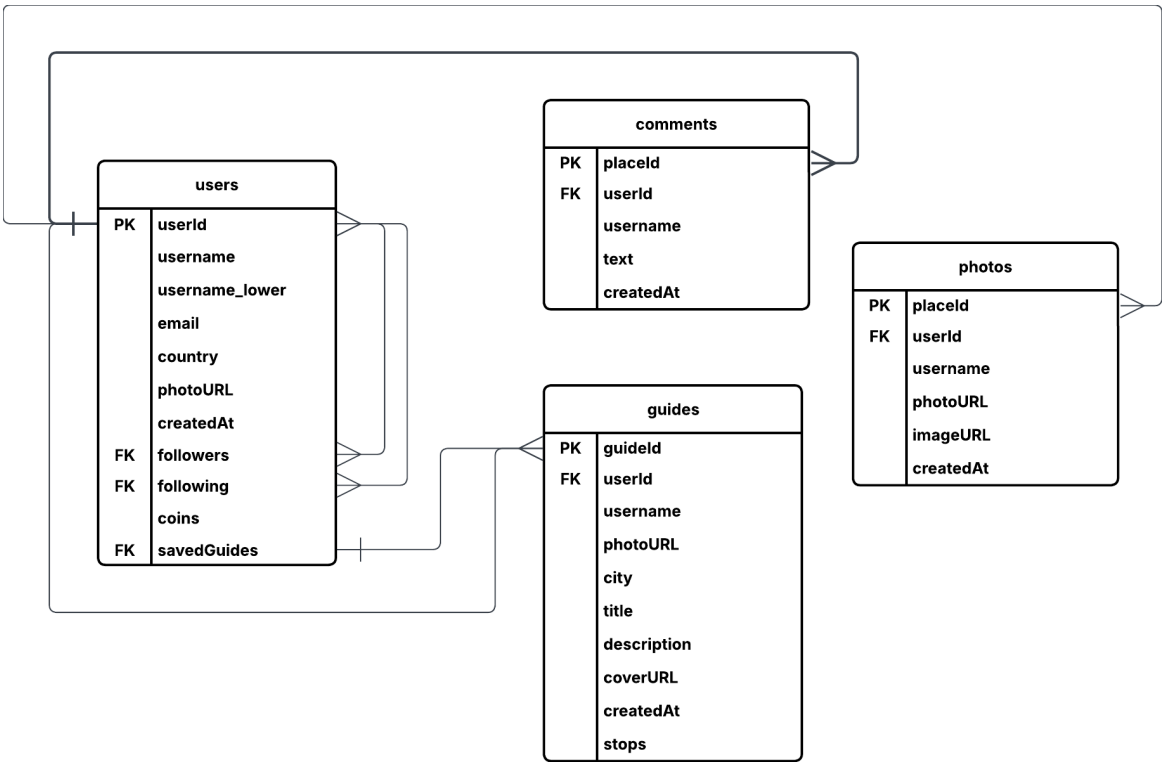

Figure 3.14. This diagram illustrates the structure of the main entities in the application, such as users, guides, comments, and photos, along with their relationships. Although the actual implementation uses Firebase Firestore, the model is presented here in a traditional relational database style for clarity and analysis purposes.

The application's main data structure is depicted in the ER Diagram in Figure 3.3, which includes important entities like users, guides, comments, and images. Each user has the ability to create multiple guides and engage with landmarks via posts that include images and comments. The relationships place a strong emphasis on social interactions, like saving guides and following other users. Foreign keys like userId and placeId are used to manage entity associations, guaranteeing referential integrity between tables.

3.4. Class Diagrams

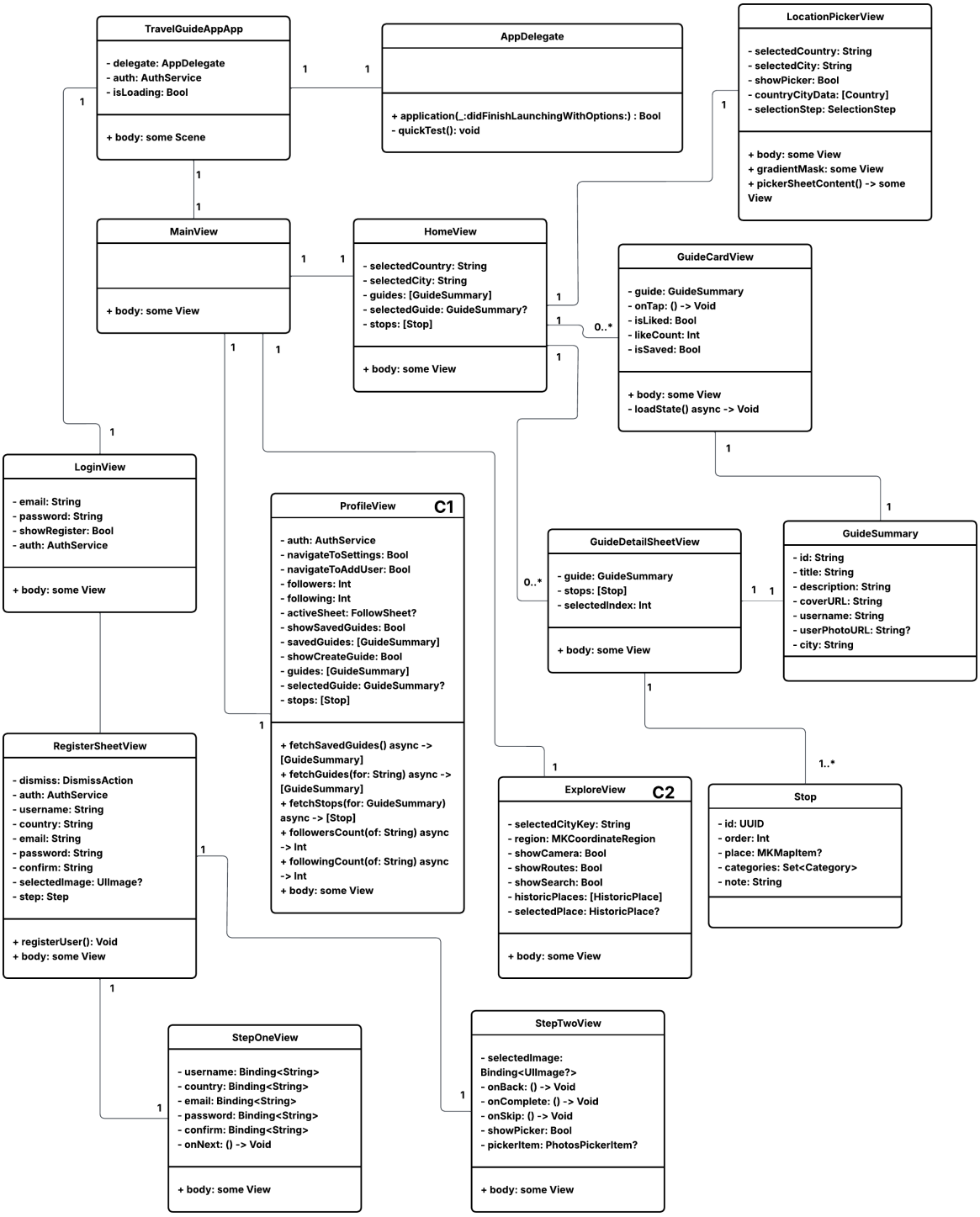

Figure 3.15. App Structure and Authentication

The application's overall architecture, with an emphasis on the onboarding and authentication process, is depicted in Figure 3.15. It illustrates how views such as StepOneView and StepTwoView, LoginView, and RegisterSheetView communicate with the AuthService

to register and log in users. The main app lifecycle (TravelGuideAppApp and AppDelegate) leads into the MainView, which directs the user to various app sections, as the diagram also illustrates. Following successful authentication, a unified entry experience is created by integrating location-based city filtering (LocationPickerView), data display (HomeView, GuideCardView), and guide details (GuideDetailSheetView).

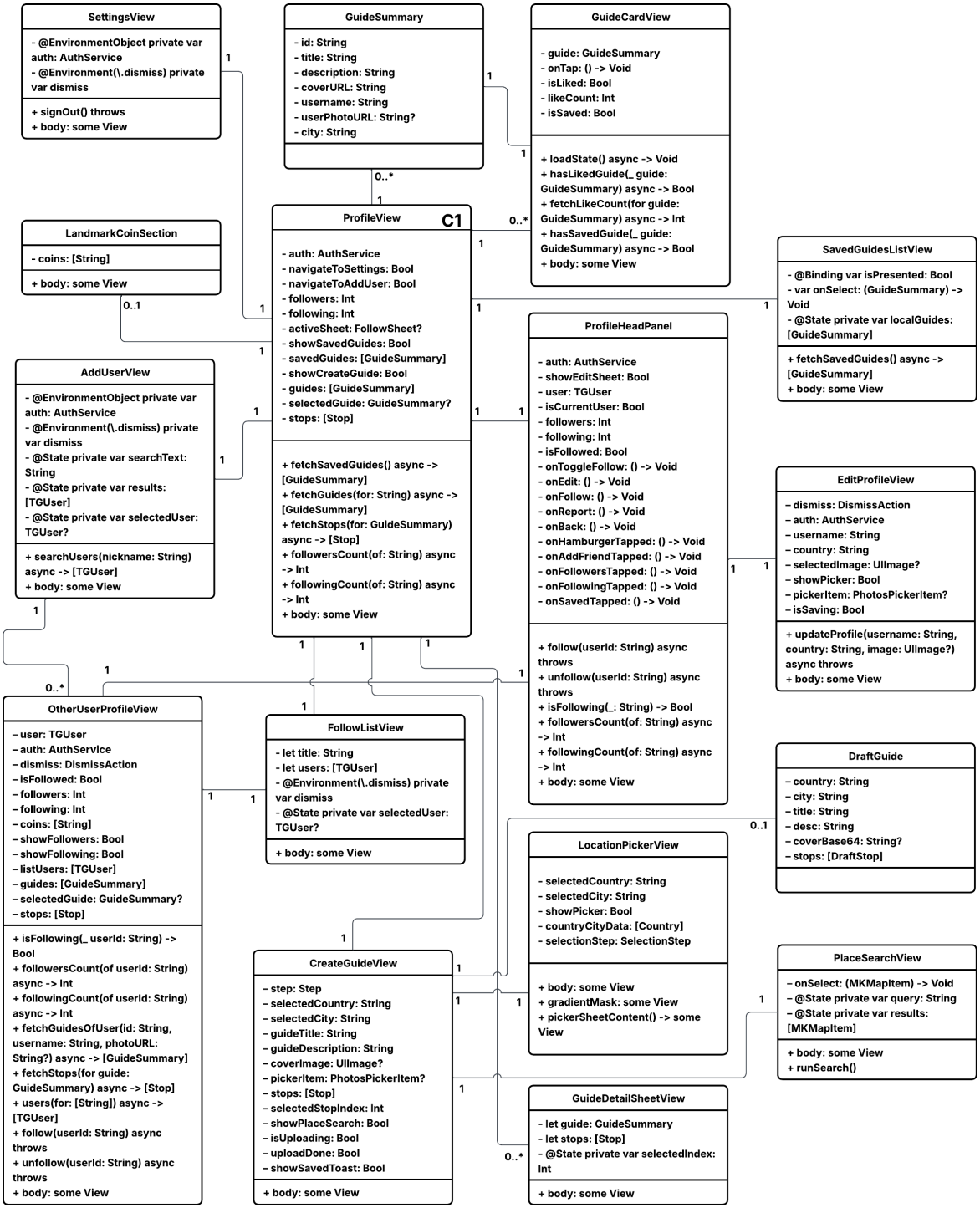

Figure 3.16. Profile Management, Social Interactions and Guide System

The architecture of the guide creation system, social interactions, and user profiles is shown in Figure 3.16. Users can view and manage their guides, followers, and saved content from the ProfileView, which acts as a central hub. Rich social features, like following and viewing other users' profiles, are made possible by components like FollowListView, AddUserView, and OtherUserProfileView. By choosing stops and adding cover photos, users can author travel guides using the CreateGuideView class. A modular and interactive social-travel ecosystem is formed by the flow of guide-related data through reusable views such as GuideCardView, GuideDetailSheetView, and SavedGuidesListView, as shown in this figure.

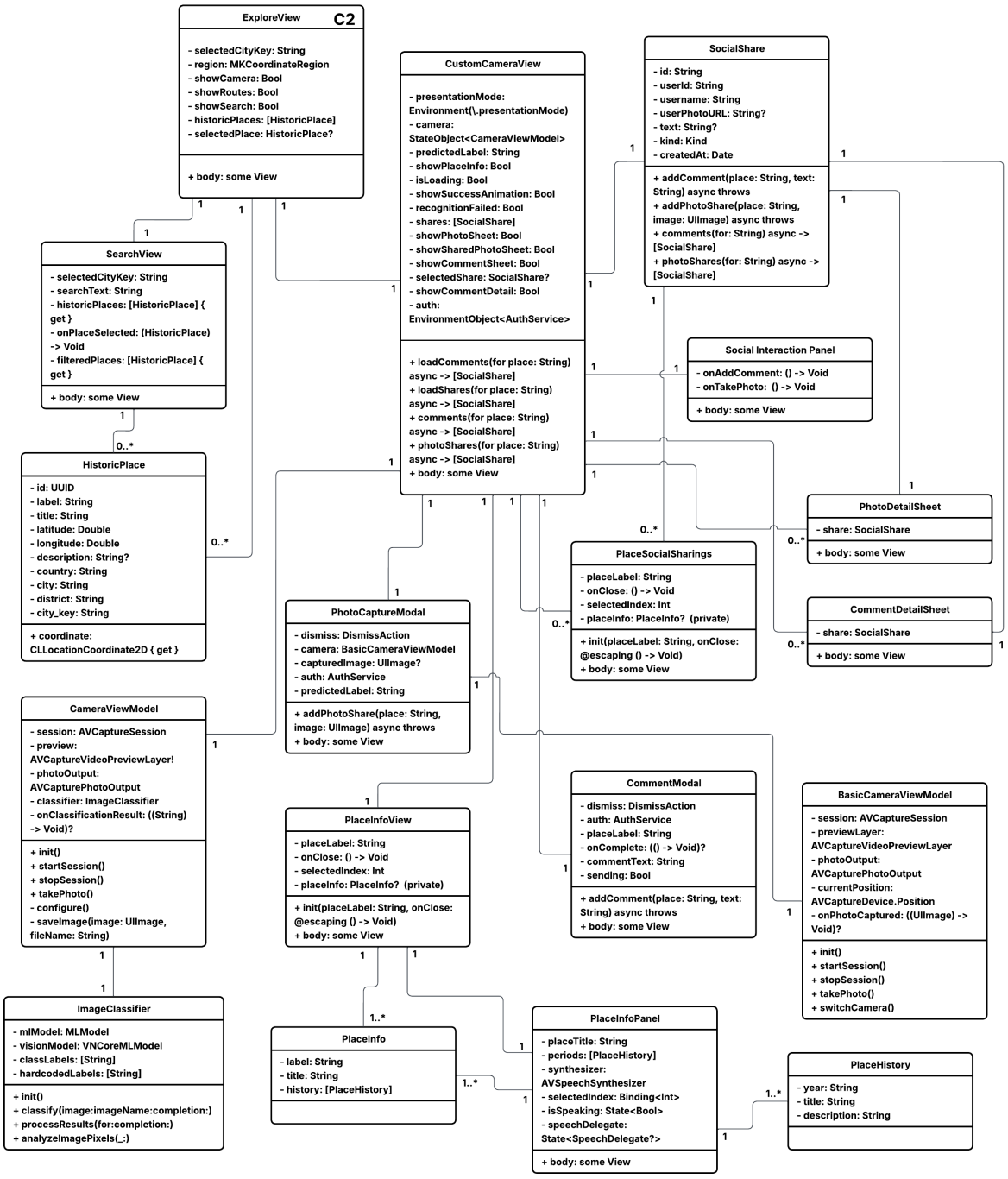

Figure 3.17. Landmark Recognition and On-Site Sharing

The system architecture enabling on-site sharing and landmark recognition is shown in Figure 3.17. The CustomCameraView serves as the main interface for taking pictures of historical sites, collaborating with the CameraViewModel and ImageClassifier to use machine learning to identify landmarks. After being recognized, users can interact with place-specific content using PlaceInfoView and PlaceInfoPanel, where related PlaceHistory entries offer historical context. Users can post comments or images using the CommentModal and PhotoCaptureModal, which are connected to the SocialShare model, thanks to the close integration of social sharing. Contextual sheets like PhotoDetailSheet and CommentDetailSheet allow users to view these shares, enhancing the interactive and location-aware experience.

4. IMPLEMENTATION

This chapter describes the tools, technologies, and development processes used in the actual implementation of the mobile travel application. It starts by outlining the fundamental software libraries and technologies that underpin the frontend and backend systems. The application's modular organization and architectural structure are shown in the ensuing sections, which also provide information on the arrangement of the logic and user interface. The creation and integration of the deep learning model for real-time landmark recognition, including dataset preparation, model training, and deployment, takes up a significant amount of this chapter. The chapter concludes with a feature-by-feature tour of the system's implementation, bolstered by illustrations and descriptions of the main features of the application.

4.1. Technologies, Libraries, and Tools

4.1.1. Programming Languages

4.1.1.1. Swift. The application was developed using Swift, Apple's modern and type-safe programming language, offering high performance and native support for iOS devices. Swift's concise syntax and safety features enabled clean, efficient, and maintainable code for both UI and logic components. Swift was used throughout the project to develop the mobile application's entire structure, including interface views, camera integration, and Firebase connectivity.

4.1.1.2. Python. Python was used for developing and training the deep learning model due to its rich ecosystem of machine learning libraries. It allowed efficient data preprocessing, model experimentation, and performance visualization before integration into the iOS application. Python also facilitated dataset handling, data augmentation, training pipelines, and model evaluation.

4.1.2. Development Frameworks and Libraries

4.1.2.1. SwiftUI. The user interface was built using SwiftUI, Apple's declarative UI framework. SwiftUI provides reusable components, reactive data flow, and seamless adaptation to various device sizes and orientations, enabling a scalable and consistent user experience. All views were created with SwiftUI to maintain visual consistency.

4.1.2.2. CoreML. Core ML was used to integrate a pre-trained deep learning model into the iOS application for on-device image classification. This framework ensures fast inference and privacy by eliminating the need for internet-based model queries. It provides GPU-accelerated performance and works smoothly with Apple's ecosystem. The model was trained externally and converted into .mlmodel format for deployment using coremltools. Core ML powers the real-time recognition of historical landmarks from photos taken within the app's Explore feature.

4.1.2.3. MapKit. For map-based navigation and historical site exploration, Apple MapKit was used. It enables rendering of interactive maps within the app, supports location-based annotations, and allows users to tap on historical site markers for detailed information. MapKit's built-in features like routing and region tracking supported the development of a more immersive exploration experience. It was primarily used in the Explore section of the app where historical locations are displayed.

4.1.2.4. Lottie. To improve UI responsiveness and add visually appealing transitions and animations, Lottie was employed. It enables rendering of JSON-based vector animations and provides lightweight alternatives to traditional animation techniques. Animations were especially helpful for visual responses to user actions such as landmark recognition and loading spinner.

4.1.3. Machine Learning Components,Frameworks and Architectures

4.1.3.1. PyTorch. The historical landmark recognition model was developed and trained using PyTorch, an open-source machine learning framework. PyTorch was selected due to its dynamic computation graph, flexibility, and wide support for vision tasks. After training the model with a custom dataset, it was exported and converted for iOS deployment using Core ML. PyTorch served as the main platform for training and evaluating the classification model.

4.1.3.2. ResNet18. ResNet18, a convolutional neural network with residual connections, was chosen for its balance between accuracy and computational efficiency. It successfully classified historical landmarks and was later converted for mobile deployment. This architecture underpinned the application's real-time recognition system by accurately identifying site images taken by the user.

4.1.3.3. NumPy. NumPy was employed during the model training phase to handle numerical operations and data manipulation efficiently. It was used in conjunction with PyTorch to prepare and process input data batches and support loss function calculations.

4.1.3.4. Matplotlib. Matplotlib supported visualization of training metrics such as confusion matrices and accuracy/loss curves, facilitating performance evaluation and comparison. It was particularly useful in generating visual reports of model validation results.

4.1.3.5. coremltools. he trained PyTorch model was converted to Core ML format using coremltools, ensuring compatibility with Apple's on-device inference framework. This step enabled seamless integration of the trained ResNet18 model into the iOS app.

4.1.4. Cloud and Backend Services

4.1.4.1. Firebase Authentication. To handle user registration, login, and session management, Firebase Authentication was integrated. It provides secure authentication workflows with email/password methods and session tracking. The platform's integration with other Firebase services makes it ideal for seamless user state management throughout the application. It was used in the Authentication module of the app.

4.1.4.2. Cloud Firestore. The app stores and retrieves user-generated content, such as travel guides, comments, and profile data, using Cloud Firestore. Firestore's real-time update capabilities and NoSQL document structure allow for flexible data modeling and live synchronization between users. Its scalability and offline support also enhance the app's reliability during travel in low-connectivity regions. It was the primary database used across the app.

4.1.4.3. Firebase Storage. Firebase Storage handles image uploads for travel guides, profile pictures, and social sharing. It is optimized for handling large binary objects and integrates easily with Firestore by referencing file paths and download URLs. This provides a secure and scalable way to store multimedia assets while maintaining performance and access control. It was used in profile photo, guide creation and on-site sharing systems.

4.1.5. Supporting Tools

4.1.5.1. Xcode. Development, testing, and deployment were carried out using Xcode, Apple's official IDE. Xcode provided interface previews, real-device testing, integrated simulators, and memory usage profiling. Instruments within Xcode were used to monitor performance metrics, such as CPU and battery consumption, ensuring efficient operation of both camera-based and offline recognition features.

4.1.5.2. Apache JMeter. Apache JMeter was utilized to perform performance and load testing on the Firebase backend services. It allowed simulation of concurrent users performing tasks such as authentication, guide creation, and data retrieval. This helped assess the system's scalability, response time, and stability under various levels of demand.

4.1.5.3. Bing Image Downloader. This Python-based tool was used to programmatically download images from Bing to create a large, diverse dataset of historical landmarks. The automated image collection process significantly accelerated dataset preparation for training the image classification model.

4.2. Overview of System Architecture and Development Flow

The application follows a modular architecture, carefully separating the user interface, backend interactions, and machine learning components to ensure clarity, maintainability, and ease of future updates. In addition to expediting development, this modular approach permits separate enhancements or extensions in particular domains without interfering with the system as a whole.

Users are presented with an interesting home screen when they first launch the app, which features carefully chosen travel guides and highlights popular locations. This screen incorporates a prominent search function that enables users to find hidden cultural treasures, filter guides by city or interest, and explore new historical sites.

The primary interface also makes it simple to access key features, like a user profile section where users can curate their own digital presence, manage personal information, and display their travel memories. This encourages communication and a sense of community because they can see how many people follow and follow them.

By adding numerous stops to record their travels, sharing images, and crafting thorough descriptions, users can quickly and simply create a customized travel guide. The ease with which each travel guide can be shared with others within the app promotes social interaction and group exploration.

Through a dedicated button, users can access the app's camera-based historical site recognition feature, which opens a live camera view that instantly identifies historical sites using a ResNet-based image classification model. For a more engaging and educational experience, the app uses a text-to-speech engine to narrate historical and cultural facts in audio-based descriptions that are automatically presented when a site is recognized.

Users receive digital landmark coins —virtual currency honoring their visits to historical locations—to further increase engagement. In addition to fostering a feeling of accomplishment, these tokens can be shown off graphically in the profile section, giving the trip a gamified touch.

Users can share their travel guides, images from historical sites, and personal experiences with their followers or larger audiences thanks to the smooth integration of social sharing tools. Individual explorations are transformed into collective cultural narratives by the dynamic, community-driven environment created by comments, likes, and location-specific interactions.

4.2.1. Modular SwiftUI Architecture and File Structure

The SwiftUI file structure used in the project is depicted in a visual diagram in this section. It draws attention to the modular design that divides shared components, Views, and ViewModels. Scalability and easier maintenance are ensured by this modular approach as features are added or modified.To maintain a distinct division of responsibilities, each View has a ViewModel that handles state and business logic. To guarantee uniform styling and interaction patterns throughout the application, reusable components (such as buttons and cards) are arranged in a special directory.

This structure enhances code reusability and testing while also making it easier for development teams to collaborate. For instance, the Helpers and Models directories contain essential features like camera operations, location verification, and classification logic, which are available from various perspectives without duplication. Similar to this, developers can iterate on particular features without causing regressions in unrelated areas of the app by

keeping social and guide-related UI elements separate under the Components folder. The project maintains its stability, scalability, and adaptability to upcoming platform extensions or enhancements by following this structured and modular architecture.

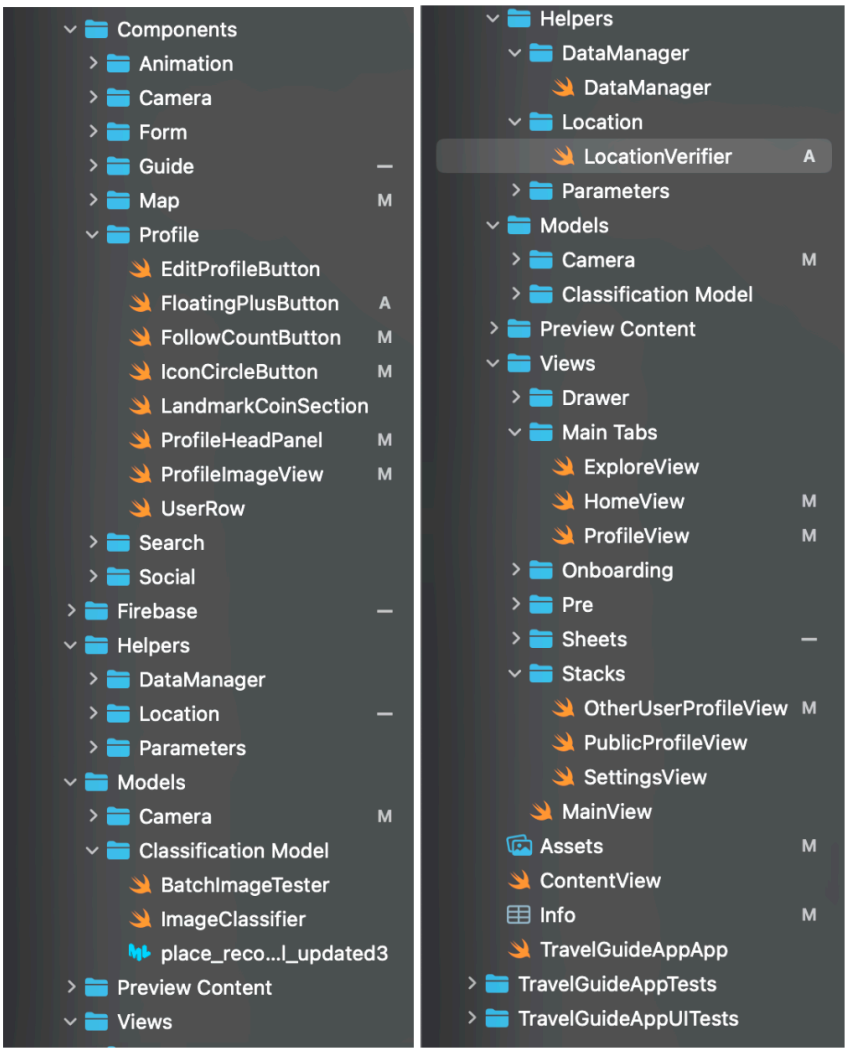

Figure 4.1. SwiftUI File and Folder Structure of the Travel Guide Application

4.3. Deep Learning Model Development and Integration

4.3.1. Dataset and Data Preprocessing

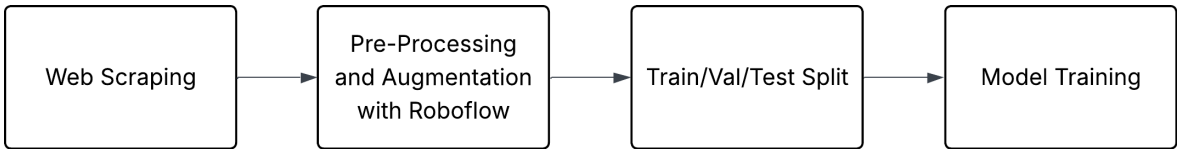

Figure 4.2. Dataset Collection and Preparation Workflow

The dataset used in this project comprises images of historical sites, curated to represent a diverse set of classes relevant to the application's core functionality. The dataset includes images of 14 different historical landmarks, ensuring the model can generalize across various visual styles and perspectives. Data augmentation techniques, such as rotation, flipping, and cropping, were applied to enhance the robustness of the dataset and mitigate overfitting by simulating a wider range of real-world conditions. The dataset was partitioned into training, validation, and testing subsets, typically with a split of 70% for training, 20% for validation, and 10% for testing, to provide a balanced approach for model development and evaluation.

The Bing Image Downloader Python library was used to gather an extensive collection of images for every historical location. By using pertinent keywords (like "Yerebatan Sarnıcı Medusa") and downloading a lot of images, this web scraping tool made it possible to automatically collect images and ensured that the dataset was diverse and rich. Only photos that faithfully captured the aesthetic qualities of the historical sites were kept, and the procedure was closely watched to guarantee data quality and relevance. This method increased dataset diversity, reduced manual labor, and expedited the data collection process.

To further increase model robustness, the images were downloaded and then uploaded to the Roboflow platform, where they underwent additional pre-processing and data augmentation procedures.The pre-processing involved auto-orientation of pixel data and resizing all images to a uniform size of 224x224 pixels by stretching them to fit. Data augmentation techniques were used to create three different versions of each source image in order to further improve the dataset and simulate various real-world conditions. These augmentation techniques included random rotations between -6 and +6 degrees, a 50% chance of horizontal flip, and random brightness level adjustments between -20 and +20 percent.

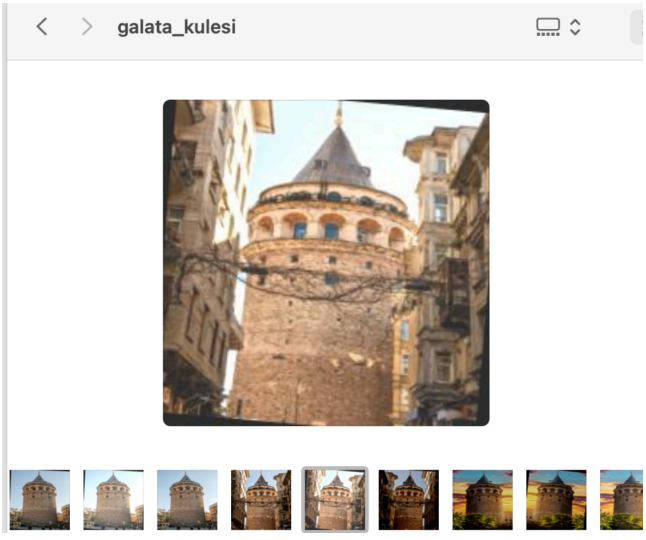

Figure 4.3. Example of an Augmented Image from dataset

4.3.2. Model Comparison and Baselines

A comparison with two other well-known deep learning architectures, MobileNetV2 and VGG16, was done in order to confirm the selection of ResNet18 as the main model for historical site classification. To ensure that variations in performance could only be ascribed to the model architectures themselves, each model was trained and assessed using the same dataset under the same conditions.

All three models' training loss plots showed consistent convergence throughout the training process. The validation loss plots, however, showed clear variations. ResNet18 and MobileNetV2 demonstrated strong generalization abilities and training process stability by maintaining consistently low validation losses across epochs, as shown in the graphs.On the other hand, VGG16 showed noticeably greater and more variable validation losses, indicating a propensity for overfitting or trouble extrapolating from the training data.

The following were the final test accuracy results in terms of test performance: The best test accuracy was 99.31% for MobileNetV2, 98.61% for ResNet18, and 85.42% for VGG16. The comparative bar chart graphically illustrates these test accuracy scores, emphasizing the ResNet18 and MobileNetV2 architectures' superior performance over VGG16.

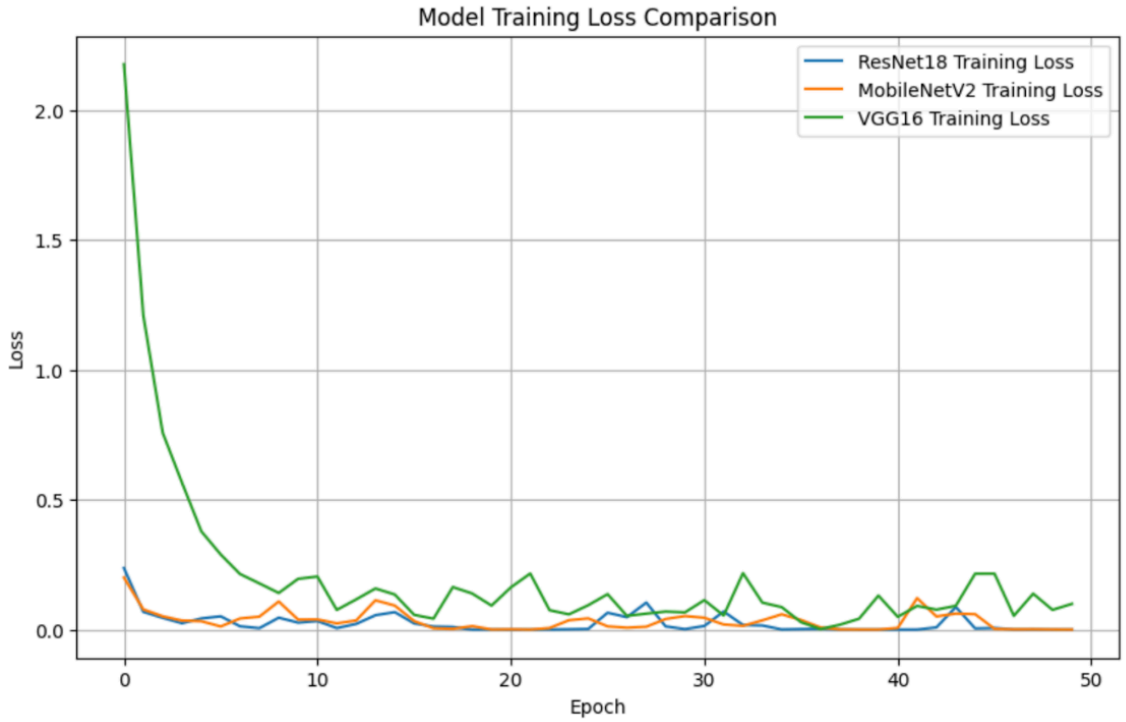

Figure 4.4. Comparison of Training Loss Across ResNet18, MobileNetV2, and VGG16

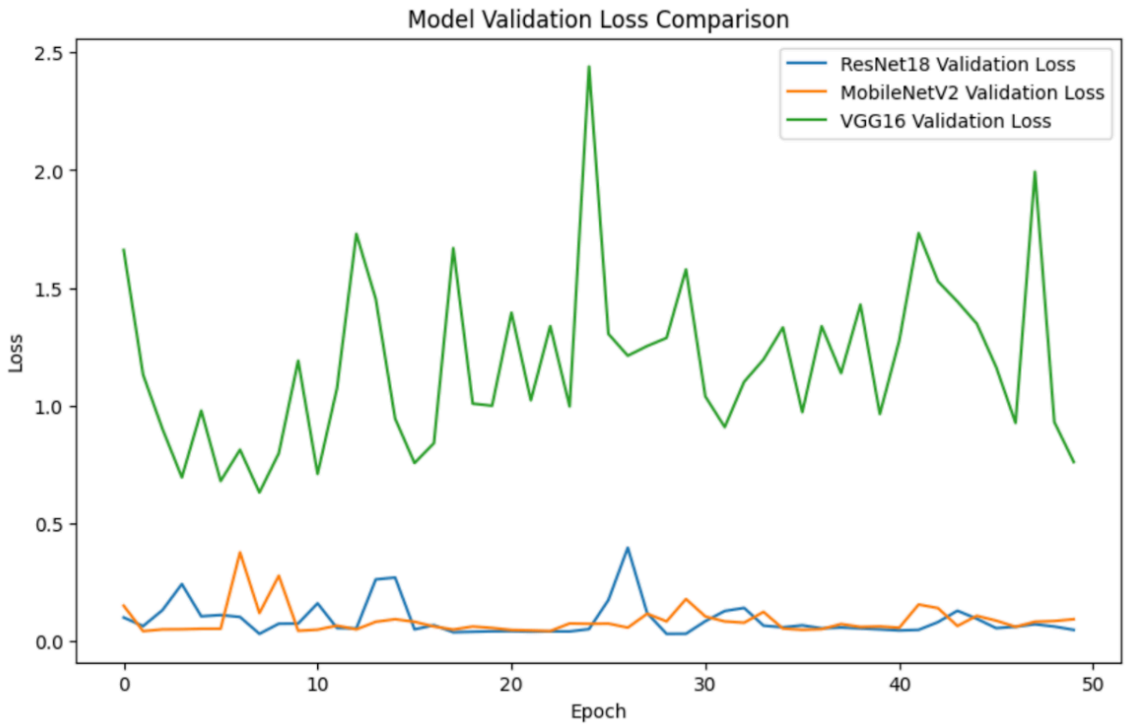

Figure 4.5. Comparison of Validation Loss Across ResNet18, MobileNetV2, and VGG16

Beyond accuracy, inference latency is a crucial factor for real-time applications, particularly on mobile devices. Measurements of inference latency per image revealed that ResNet18 achieved an average latency of 0.14 ms, while MobileNetV2 required approximately 0.30 ms per image, and VGG16 recorded a faster latency of 0.09 ms. Despite VGG16's lower latency, its significantly lower accuracy rendered it unsuitable for this application.

Classification reports further validated the superior performance of MobileNetV2 and ResNet18 in addition to accuracy and latency. While VGG16's performance was more inconsistent, with lower precision and recall in multiple classes, both models achieved nearly perfect precision, recall, and F1-scores across the majority of historical site classes.

Based on these findings, ResNet18 was selected as the final model for deployment in the mobile application. Although MobileNetV2 demonstrated slightly higher accuracy, ResNet18 offered a compelling balance of accuracy, generalization ability, and faster inference time compared to MobileNetV2. This balance is particularly important in real-time, resource-constrained environments such as mobile applications. Moreover, ResNet18's residual connections help mitigate vanishing gradient issues in deeper networks, ensuring more stable convergence during training.

Model	Test Accuracy (%)	Validation Loss Stability	Inference Latency (ms/image)	Macro F1-score	Weighted F1-score
ResNet18	98.61	Low, stable	0.14	0.99	0.99
MobileNetV2	99.31	Low, stable	0.30	0.99	0.99
VGG16	85.42	High, fluctuating	0.09	0.83	0.85

Table 4.1. Comparison of different CNN architectures in terms of test accuracy, validation stability, latency, and F1-scores

Overall, the comparative analysis emphasizes how crucial it is to choose a model carefully that balances computational efficiency and deployment viability in addition to achieving high accuracy. ResNet18 was determined to be the best model for the goals of this project due to its near-perfect accuracy and low latency.

4.3.3. Model Selection and Justification

Because it can successfully handle the vanishing gradient issue, which can impede the training of deep neural networks, the ResNet architecture (Residual Networks) was selected for this project. Because of ResNet's creative use of residual connections, information can pass through some layers, allowing the model to learn more intricate patterns without losing important characteristics. ResNet18 was chosen for this project because it balanced classification accuracy with computational efficiency. This decision is especially crucial for applications running on mobile devices, where resource constraints and performance must be carefully taken into account. ResNet is a good fit for iOS deployment because of its demonstrated compatibility with Core ML, which guarantees effective and instantaneous inference capabilities on the target platform. Additionally, comparisons with other cutting-edge models, such as MobileNetV2 and VGG16, showed that ResNet18 offered faster inference latency (0.14 ms/image vs. 0.30 ms/image) and nearly equal test accuracy (98.61%) to MobileNetV2 (99.31%). The choice of ResNet18 as the best model for the project's objectives and real-time performance requirements was ultimately influenced by this trade-off between high accuracy and computational speed.

4.3.4. Model Architecture

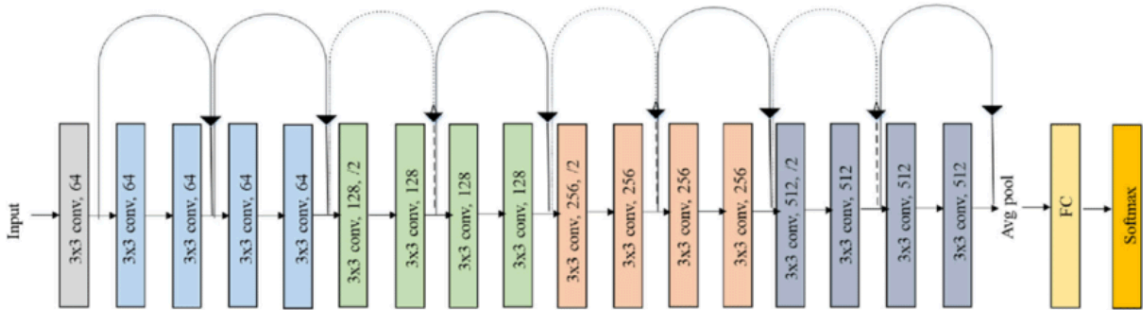

Figure 4.6. ResNet18 architecture[5]

The use of residual blocks, which include skip connections that enable the direct propagation of input data across layers, is what distinguishes the ResNet model. The model can learn intricate representations thanks to this structure, which also reduces the possibility of performance deterioration as the network gets deeper. Each residual block consists of ReLU activation functions to add non-linearity, batch normalization layers to stabilize learning, and convolutional layers to extract hierarchical image features. With 18 convolutional layers and roughly 11.7 million parameters, the architecture of the particular version used for this project, ResNet18, effectively balances classification accuracy and computational efficiency. The first layer uses a 7x7 convolutional filter with a stride of 2 and a 3x3 max pooling operation to reduce spatial dimensions before entering the sequence of residual blocks. The final output layer creates probability scores for the target number of classes, in this case the number of historical sites in the dataset, using a softmax activation function. ResNet18 is particularly well-suited for deployment on mobile devices where resource and performance constraints are critical because of its general architecture, which balances accuracy and computational demands.

4.3.5. Training Details

Standard deep learning optimization methods were used to train the model. With a learning rate of 0.001, the Adam optimizer was specifically chosen to fine-tune model weights based on the error gradients. The model was trained over 50 epochs with a batch size of 32. Through iterative experimentation, these hyperparameters were carefully adjusted to guarantee that the model performed well without overfitting.

The difference between the true class labels and the model's predicted class probabilities was measured using the cross-entropy loss function. The dataset was separated into batches

and the model was placed in training mode for each epoch. To benefit from accelerated computation, the input images and associated labels were loaded onto the GPU for every batch. At the beginning of every batch, the optimizer's gradients were reset. The output probabilities were then obtained by a forward pass through the ResNet18 architecture. The model's ability to distinguish between the historical site classes was then improved by backpropagating the computed cross-entropy loss to update the model's weights.

After each epoch, validation was carried out using a different validation dataset, which revealed information about the model's capacity for generalization. To make sure that batch normalization and dropout behaviors matched inference, the model was placed in evaluation mode during validation. The validation loss was averaged across the whole dataset, and no gradient updates were made during this phase. In order to visualize the model's convergence behavior and identify any indications of overfitting, these validation losses were closely tracked and plotted alongside training losses.

Training was carried out on a workstation equipped with GPU acceleration, significantly reducing training time compared to CPU-based training and allowing for multiple iterations of hyperparameter tuning.

Hyperparameter	**Value**
Optimizer	Adam
Learning Rate	0.001
Batch Size	32
Number of Epochs	50
Loss Function	Cross-Entropy

Table 4.2. Hyperparameters used during model training.

4.3.6. Model Conversion and Integration

Using PyTorch's versatility for deep learning research and prototyping, the trained ResNet18 model was first created in this framework. The model was transformed into a Core ML format in order to facilitate deployment on iOS devices and guarantee compatibility with the Core ML ecosystem. Two steps were taken to complete this conversion:

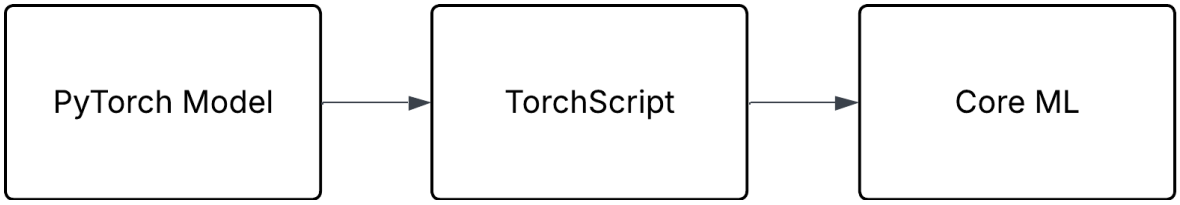

Figure 4.7. Conversion pipeline

To create a scripted model that was optimized for deployment, the trained PyTorch model was first traced using TorchScript. TorchScript enables effective and repeatable conversion by capturing both the learned weights and the model architecture. Static analysis was made possible by the scripted model, which acted as a transitional representation while maintaining the computational graph.

Using coremltools, a specialized library for connecting PyTorch and Apple's machine learning ecosystem, the TorchScript model was transformed into Core ML format in the second step. In order to guarantee that the output of the Core ML model could be directly interpreted within the application, a ClassifierConfig was defined with the entire list of class labels during this conversion. Furthermore, input preprocessing parameters were specifically set to match the data preprocessing used during training, including pixel normalization and color bias adjustment. This preserved accuracy and dependability by guaranteeing that the Core ML model would receive inputs in line with the initial training pipeline.

A normalization inconsistency was discovered during this conversion, though, as the Core ML model produced unnormalized logits rather than probabilities. Consequently, predictions were not initially displayed as normalized confidence scores (i.e., not in the [0,1] range). This was fixed by implementing a custom softmax function on the Swift side of the classification module. By ensuring that the final predictions in the iOS app were normalized probabilities, this step preserved the integrity of the model's predictions and fully aligned with the expected classification output.

The Core ML model in the mobile app operates fully on the device, removing the need for constant server communication and improving user experience by classifying historical site photos instantly and smoothly. The Core ML model's seamless integration with the iOS ecosystem attests to its suitability and demonstrates the project's technical maturity in integrating deep learning frameworks with practical mobile applications.

Furthermore, utilizing the VNCoreMLModel and VNCoreMLRequest APIs, a special-

ized Swift-based classification module was created, guaranteeing close integration with the Core ML model and utilizing the sophisticated image analysis features of the Vision framework. To replicate the training pipeline and preserve classification accuracy, this module incorporates strong preprocessing steps, such as resizing input images to 224×224 pixels and using consistent normalization. Additionally, it employs a confidence threshold mechanism that designates predictions as ”unknown” if their confidence score is less than 75%, enhancing dependability and user confidence. Additionally, the list of class labels is dynamically retrieved from the Core ML model itself by the Swift code, guaranteeing deployment flexibility and reverting to a predefined list when necessary. By removing server dependencies and providing a fully on-device, real-time classification experience, this thorough integration greatly improves the mobile application's overall user experience.

4.3.7. Limitations of the Model

The ResNet-based model performed well, but a number of drawbacks were noted. Overfitting could impair the model's capacity to generalize to fresh and varied photos of historical locations due to the comparatively small dataset used for training. Additionally, there were clear disparities in precision and recall for a few classes in the dataset. For instance, the model occasionally incorrectly classified other historical sites as ”medusa,” even though the ”medusa” class had perfect recall and relatively low precision. These incorrect classifications imply that the model may become confused by visually similar architectural details, especially in difficult lighting situations or when there are partial occlusions.

Additionally, variations in environmental factors such as outdoor lighting conditions, camera angle, and device camera quality can significantly impact recognition accuracy. These limitations highlight the importance of further expanding the dataset to include a broader variety of historical sites and environmental scenarios. Incorporating additional data from different weather, lighting, and viewing angles would improve the model's robustness and overall performance in real-world applications. Future improvements could also focus on augmenting the dataset for underperforming classes, as well as exploring lightweight model variants or fine-tuning the feature extraction layers to address misclassification in visually similar landmark classes.

4.4. Feature-Level Implementation in the Application

4.4.1. User Authentication: Login and Registration

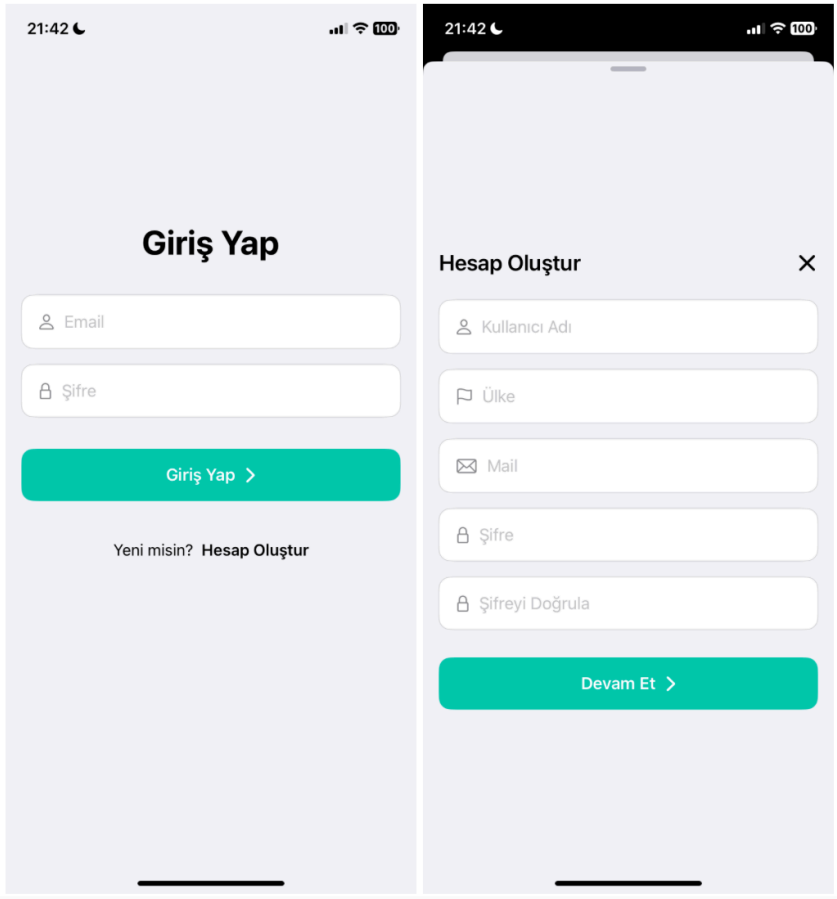

Figure 4.8. Login Screen – Users enter email and password to access the app.
Figure 4.9. Registration Screen – User inputs username, country, email, password, and an optional profile photo.

Using Firebase Authentication, the authentication flow was put into place, offering a scalable and safe way to handle user registration and login. Session restoration, user information retrieval, profile photo registration, and country selection are all handled by the AuthService class.

After successfully completing the Firebase validation process, users are taken to the main application interface after entering their email address and password in the login interface. A SwiftUI sheet view is used to display a two-step form during registration. Users enter their email address, password, username, and country in the first step. Users can use the native PhotosPicker component to upload a profile picture in the second step. The user document

in Firestore is linked to this image, which has been uploaded to Firebase Storage.

Throughout the authentication process, reusability and clarity are guaranteed by the modular use of views (StepOneView, StepTwoView) and state management. Animations and loading indicators are used to improve the user experience, with a focus on simplicity and guidance for novice users.

Centralized access control and user data handling are made possible by Firebase Authentication's smooth integration with other Firebase services (Firestore, Storage). The root view changes to the main content screen after a user logs in, and the authentication status is continuously tracked. The overlay and progress view components of SwiftUI are used to gracefully handle all error messages and loading states.

4.4.2. Home Screen: Guide Feed with Location-Aware Filtering

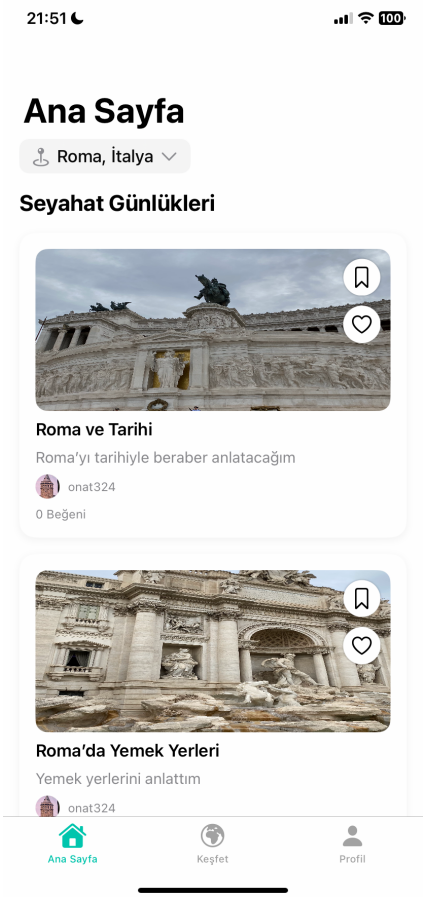

Figure 4.10. Home screen displaying city-specific travel guides with options to like and save.

The user's main feed for perusing shared travel guides is the home screen. The LocationPickerView, which is synchronized throughout the app by @AppStorage, allows users to choose a particular nation and city at the top. The screen dynamically displays the travel guides that are pertinent to the city you have chosen.

A cover photo, title, brief description, author username and profile picture, and interaction buttons for liking and saving are all displayed on each guide card (GuideCardView). Real-time Firestore queries that represent the user's preferences support these buttons.

When a user taps on a guide card, a detailed view in a modal sheet with the trip's stops fetched asynchronously from Firestore is displayed.

To keep the home feed and the map in sync, the chosen city also chooses the focus point in the Explore (Map) section of the app. By allowing users to plan their visits more contextually based on other people's experiences, this interconnected design promotes location-based exploration.

4.4.3. Guide Detail View: Interactive Stop-Based Exploration

Users are shown a modal sheet (GuideDetailSheetView) that provides an immersive breakdown of the journey through several consecutive stops after choosing a guide from the home screen. Users can easily move between locations within the same travel guide thanks to the numbered bubbles at the top that represent each stop.

The stops related to the guide are fetched asynchronously using the function fetchStops(). Each stop includes metadata such as the order, name, categories, coordinates, and a user-written note. These stops are displayed horizontally using circular step indicators. When a user taps on a step, the selectedIndex state is updated, and the UI refreshes to show the details of the corresponding stop.

For each stop, the following information is displayed:

- Stop Name: The specific name of the place.
- Categories: Contextual tags shown as stylized labels.
- Map Integration: A dedicated button that opens the corresponding location directly in Apple Maps via MKMapItem.openInMaps().

- Narrative Note: A personalized description from the guide's author detailing their experience or observations at that stop.

Travelers taking the same route will find the guide more actionable and more engaging thanks to its stop-by-stop storytelling format. The system guarantees a simple and engaging viewing experience and supports several categories per stop.

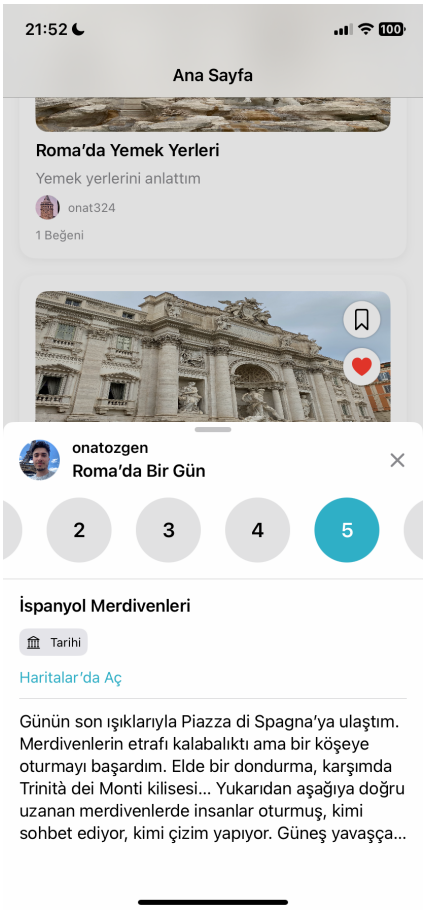

Figure 4.11. A detailed travel guide view where users navigate through sequential stops and access map integration for each.

4.4.4. Explore Screen – Map-Based Discovery

Through an interactive map powered by MapKit, users can visually explore historical landmarks within a city using the Explore screen. The application centers the map region using MKCoordinateRegion after determining the chosen city from UserDefaults as soon as the screen loads.

MapAnnotation is used to display each historical location on the map after it has been

loaded using loadHistoricPlaces(). A unique user interface element (MiniPlaceBar) is used to identify each landmark, displaying its name and icon. A bottom sheet (InformationView) with a brief description and the option to open the location in Apple Maps appears when a user taps a location on the map.

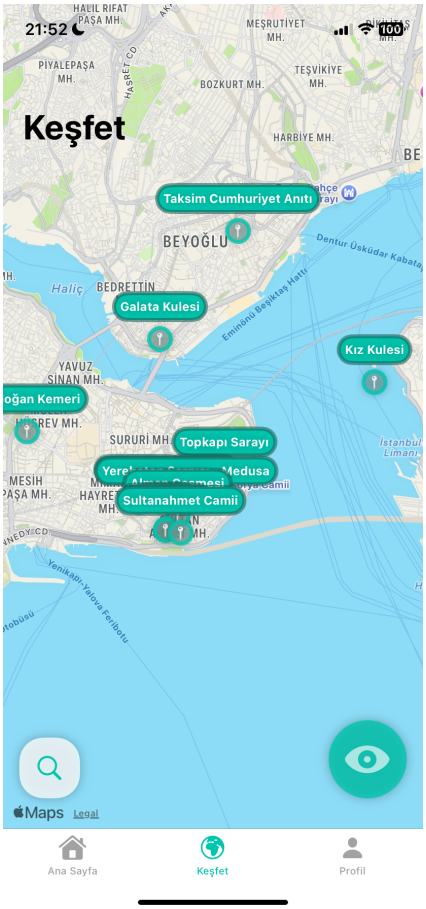

Figure 4.12. Explore screen allows users to navigate the city map, tap on historical landmarks for brief information, and access Discovery Mode via the camera button to recognize landmarks.

Users can browse and filter the city's landmarks in the search view that is opened by clicking the magnifying glass button (ExploreBottomPanel) in the bottom-left corner. When you select a result, the map region is updated to zoom in on that location. Discovery Mode (CustomCameraView) is accessed via a camera-eye button on the bottom-right. In this mode, the user's camera is turned on, and the historical landmark being viewed is identified using an on-device Core ML classification model.

4.4.5. Historical Landmark Search and Information Views

With the help of the Search View and the Information View, users can easily find historical sites in a chosen city and gain more knowledge about each one.

Users can peruse every landmark connected to the city they are currently selecting in the Search View. Each landmark is shown in the view as a scrollable list with its name, location, brief description, and a symbolic landmark coin. Keywords that match the title or description can be used to filter results using the search bar. Swift's.filter and localizedCaseInsensitive-Contains are used for this filtering, and the data is loaded by filtering loadHistoricPlaces()'s output according to the chosen city key kept in @AppStorage.

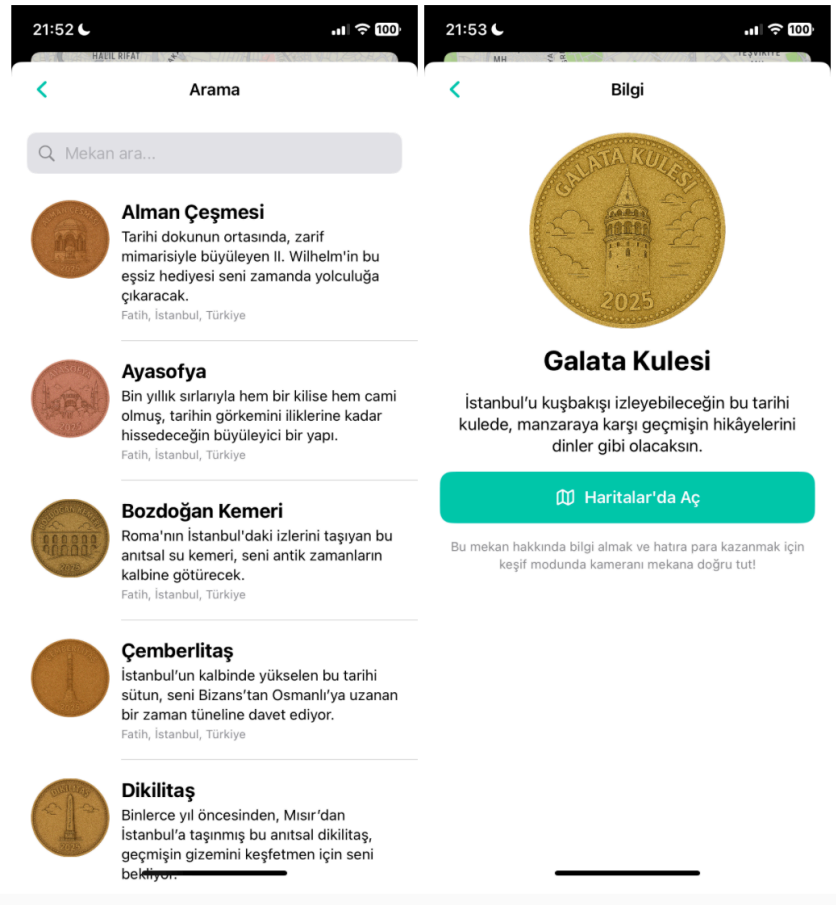

Figure 4.13. The Search View screen displaying landmark coins and brief descriptions for historical places in the selected city.

Figure 4.14. The Information View of the Galata Tower, where users can read a detailed description, see the landmark coin, and access directions via Apple Maps.

The Information View appears when a user taps on a landmark entry. A detailed view of

the landmark is shown on this screen, along with a larger image of the landmark coin, the name and an expanded description, a button that uses MKMapItem.openInMaps() to open the location in Apple Maps, and a reminder that scanning the landmark in discovery mode will yield additional rewards.

The InformationView composes the user interface using SwiftUI views and coordinate binding using MapKit. The visual design supports both exploration and educational goals by hierarchically highlighting significant data.

4.4.6. Discovery Mode: A CNN-Based Cultural Exploration System

The Discovery Mode of the mobile application transforms travel into a smart, interactive, and social experience. By using real-time image recognition, location verification, and social features, it allows users to engage with cultural landmarks in a much deeper way.

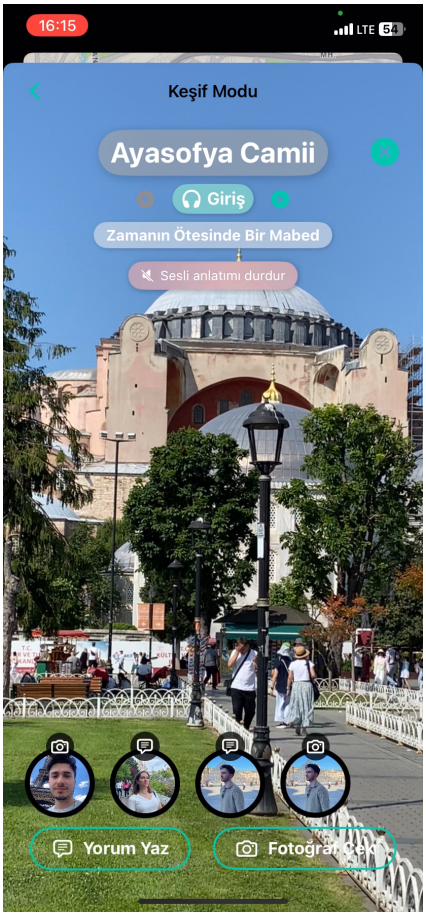

Figure 4.15. The Discovery Mode interface showing the real-time recognition of Hagia Sophia.

4.4.6.1. Landmark Recognition. To match the expected input dimensions of the Core ML model, the captured image is resized using UIGraphicsImageRenderer to a fixed size of 224x224 pixels. Regardless of the initial resolution, this preprocessing guarantees a consistent input format. The compiled.mlmodelc file is wrapped by Vision's VNCoreMLRequest for optimal inference on iOS devices during the classification process. The top result and its confidence score are used directly if the model output is in VNClassificationObservation format; if not, raw logits from VNCoreMLFeatureValueObservation are subjected to a fallback softmax computation.

Through the LocationVerifier class, which publishes the user's current GPS coordinates, CLLocationManager performs the location check. CLLocation calculates the haversine distance between the user's location and the coordinates of the predicted landmark.distance(from:), and only in the event that the outcome is less than 200 meters, the label is confirmed.

4.4.6.2. Earning Landmark Coins. Firestore's FieldValue.arrayUnion() is used by the function AuthService.addCoin(label) to append the historic label to the user's coins field. Duplicate entries are automatically avoided, ensuring idempotent updates. The app uses a @Published var coins: [String] property to listen to the user's profile after the coin is added, allowing the user interface to update instantly. To guarantee non-blocking user interface interaction, the logic is asynchronous and carried out through Swift's Task API. Users are encouraged to visit and gather coins from new locations by this gamified system.

4.4.6.3. Audio Narration. A PlaceInfoView panel displays after a landmark has been identified. It depicts several historical eras associated with the location, such as the Byzantine and Ottoman periods. There is a title and description for every period.

The system provides text-to-speech narration in Turkish using AVSpeechSynthesizer. Using navigation buttons, users can pause the narration at any moment and move between different time periods. As a result, the app has the feel of an audio tour guide in real time.

4.4.6.4. On-Site Social Interactions. Using whereField("userId", in: [...]) to filter only followed users, the application uses parallel asynchronous Firestore queries to retrieve social content, one for comments and another for photos. A SocialShare object containing metadata like the username, userId, photoURL, and creation time is mapped to each result. Using putDataAsync(), images are uploaded to Firebase Storage, and the URLs of those images are stored in the places/label/photos subcollection. Places/label/comments is where textual com-

ments are kept. PlaceSocialSharings, a unique horizontally scrollable SwiftUI view, is then used to display these. Depending on the share type, tapping an item modally displays either the PhotoDetailSheet or the CommentDetailSheet using.sheet(isPresented:) with dynamic content. To preserve contextual integrity, each post is linked to the current place label.

This feature makes social sharing location-aware and exclusive:

- Viewing Follower Content: Comments and images posted by their followed users are only visible to users who are physically present at the same landmark. PlaceSocialSharings is a horizontal scroll view that displays this.
- Making a Share: Users have the option to take and share a photo (PhotoCaptureModal) or write a comment (CommentModal). These have been uploaded to Firebase Storage and Firebase Firestore, respectively. Photos are located at /places/label/photos, and comments are located at /places/label/comments.
- Post Details: When a user taps on a shared item, a preview of the photos opens in PhotoDetailSheet and comments open in CommentDetailSheet.

This system ensures that shared content is authentic, tied to physical presence, and socially relevant.

4.4.6.5. Feedback Error Alerts. A @State flag, like recognitionFailed or showDistanceAlert, is used to implement error feedback. This causes a red translucent overlay to appear via SwiftUI conditional rendering. This overlay uses DispatchQueue.main.asyncAfter to automatically dismiss the message ”Place could not be verified” after a 2-second lag. Both the location distance threshold and classification confidence (default: $\geq$0.75) are strict conditions, and failing either resets the recognition pipeline. Additionally, a centralized handler inside onClassificationResult protects the recognition state machine from recurrent API calls or transitions.

4.4.7. User Profile and Social Layer

A customized dashboard is available in the User Profile section, where users can manage their profiles, view collected items like travel guides and landmark coins, and engage with social features like followers, following, and saved content. Secure data handling and real-time updates are guaranteed by the close integration of all interactions with Firebase Firestore

and Firebase Storage.

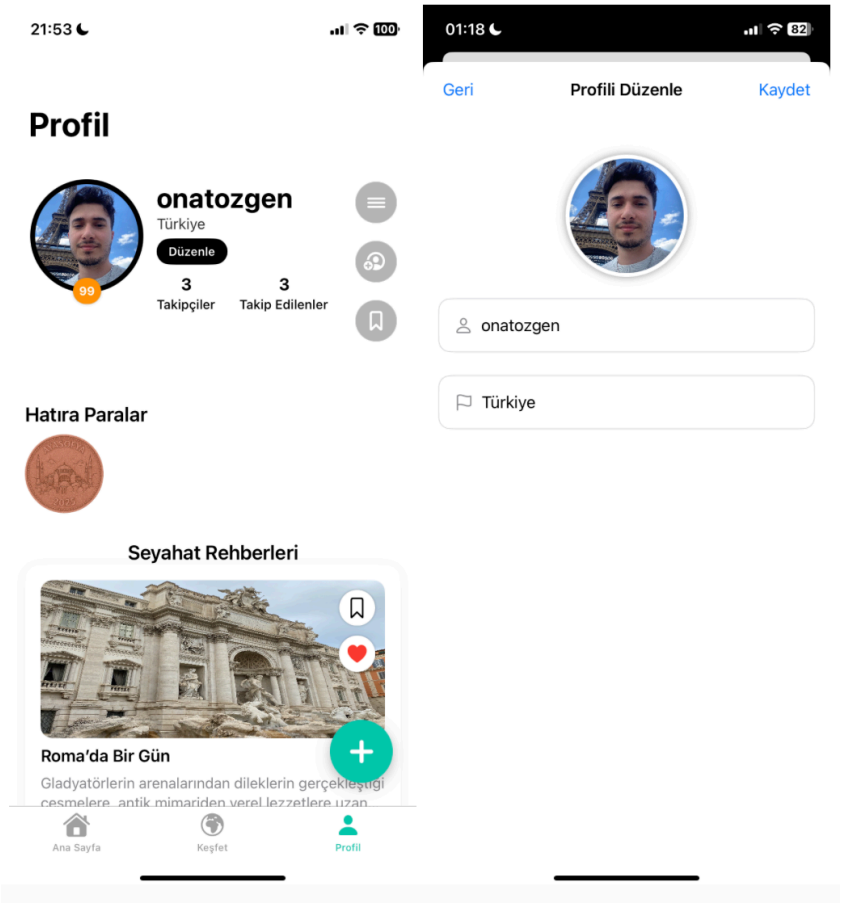

Figure 4.16. The profile screen

Figure 4.17. The profile editing sheet

4.4.7.1. Displaying User Information. The application uses the AuthService to retrieve pertinent information from Firestore, including the username, country, profile image URL, and coin collection, when a user accesses their profile.the fetchUser(uid:) function. Using URLSession, the profile picture is downloaded asynchronously and displayed in a circular frame. This data and follow statistics are shown in the ProfileHeadPanel view. Using followersCount(of:) and followingCount(of:), the follower and following counts are dynamically retrieved from the Firestore users document.

The quantity of landmark coins the user has gathered is displayed by a tiny badge above the profile picture. Using a @Published property coin in the AuthService class, this badge is updated in real-time and offers a gamified visual cue.

4.4.7.2. Editing Profile Information. The EditProfileView opens as a sheet when you tap the ”Edit” button. Users can change their profile picture, username, and country in this view. The updateProfile(username:country:image:) method is used to write the updated data back to Firestore. A new image chosen by the user through the PhotosPicker is uploaded to Firebase Storage with the path avatars/uid.jpg. After that, the generated URL is saved to the user’s Firestore document.

As soon as the upload is finished, the @Published profile image and local TGUser model are updated to reflect the UI changes.

4.4.7.3. Landmark Coin Collection. Verified landmarks are represented by a list of coin images that can be scrolled horizontally in the LandmarkCoinSection. The coin image names are kept in the coins array of the user’s Firestore document. The assets in Xcode’s Asset Catalog, which are rendered using Image(label) in a ForEach loop, are represented by each entry in this array. These pictures act as a progress tracker for the user’s travels as well as a memory archive.

4.4.7.4. User’s Travel Guides. The GuideCardView component is used to display the user’s personal travel guides beneath the coin section. These guides are obtained using fetchGuidesOfUser(id:username:photoURL:) from the user’s subcollection users/uid/guides. A title, description, city, and the URL of the cover image are among the metadata included in each guide.

FetchStops(forGuide:) is used to retrieve the stops contained in a selected guide. Within the guide document, each stop contains geolocation, categories, and user notes that are stored as a JSON-like array. A customized travel schedule can be saved and viewed by others thanks to this system.

4.4.7.5. Saved Guides. Users can access previously saved guides by clicking on the bookmark icon on the profile. Both the old (string ID) and new (ID + city) formats are supported by fetchSavedGuides(), which is called by the system when it is tapped. To enable users to revisit content they found useful, guides are pulled from the global collection guides/city/items/guideID and displayed in a scrollable sheet (SavedGuidesListView).

The savedGuides array in the Firestore users document is updated by the saving mechanism itself using the saveGuide() and unsaveGuide() methods. Using hasSavedGuide(), the

application can also determine if a guide has been saved.

4.4.7.6. Followers and Following System. The application implements a bi-directional social system. The follower and following arrays are stored in the user document (followers, following). When users tap on these values, the app fetches the related user IDs and then loads their public profiles in chunks of 10 using Firestore's in filter, respecting the query limits. The resulting data is shown in FollowListView.

The follow/unfollow operations are handled by batch updates in Firestore using follow(userId:) and unfollow(userId:). This ensures atomic updates across both the current user and the target user's documents.

The state is kept in sync with Firestore using a snapshot listener (startFollowingListener()), which updates the @Published followersIds and followingIds arrays automatically.

4.4.7.7. Creating New Guides. The floating action button in the bottom-right corner opens a sheet where users can create new guides. This launches CreateGuideView, which lets users set a title, description, cover image, and list of stops.

4.4.7.8. User Search and Logout. Users can log out by using the signOut() method in AuthService, which is triggered by the hamburger icon in the top-right corner. This ensures a clean logout process by removing the current user from the session and turning off all snapshot listeners.

Users can look for others by username using the AddUserView discovery interface, which is opened by clicking on the friend icon. This enables prefix-based matching by using Firestore queries with range conditions on the username.

4.4.8. Guide Creation System

Users can create and share personalized travel routes made up of consecutive stops using the guide creation system's dynamic interface. With its two-stage flow and offline support, it offers technical robustness through Firebase backend services and enables both novice and expert users to create meaningful travel content.

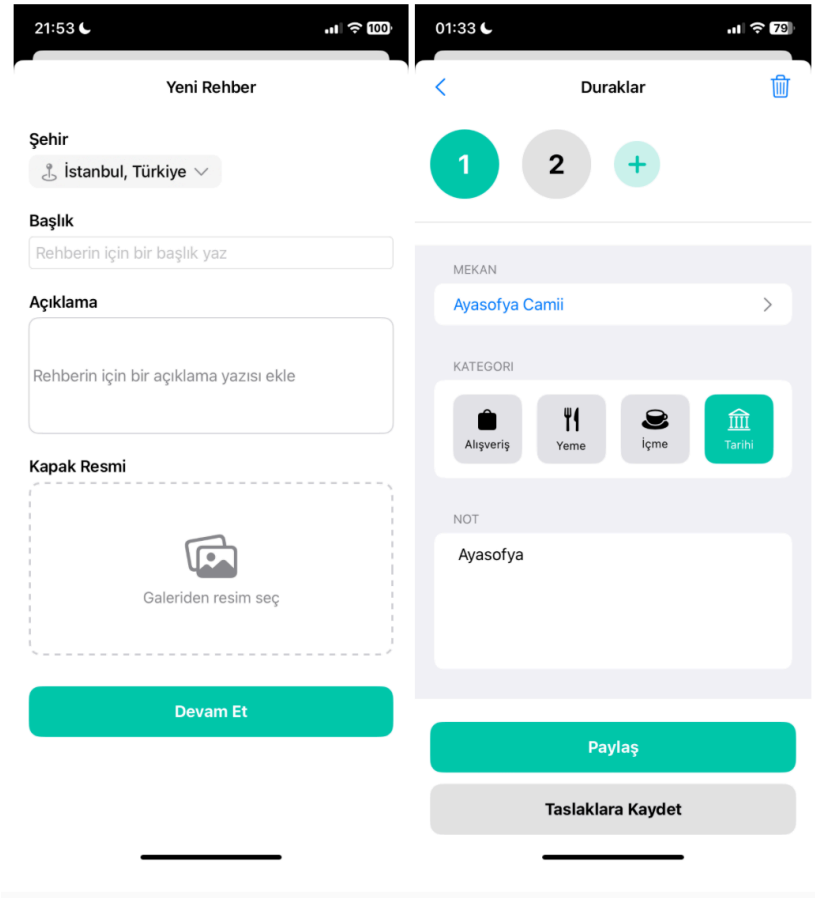

Figure 4.18. The initial form of guide creation

Figure 4.19. Second step of guide creation where individual stops are defined with location, category, and notes.

4.4.8.1. General Information Entry. Users input the necessary details for their travel guide in the first step of the guide creation process. This entails deciding on a city, adding a cover photo, writing a description, and entering a title. To guarantee consistency between sessions, the chosen city and nation are obtained from persistent @AppStorage keys. While the description uses a TextEditor to enable multiline input with a custom placeholder implementation, the title field records a brief identifier for the guide. When users use PhotosPicker to choose a cover image from their photo collection, the image is transformed into a UIImage and saved locally. Until the guide is published or saved as a draft, this data is kept on file. This first step serves as the foundation for the travel guide, setting the tone and providing contextual background for the route that will follow.

4.4.8.2. Stop Management Interface. Detailed stops along the travel route are defined in the second step. A numbered bubble designates each stop, and a horizontal scroll interface

lets users navigate between them. A built-in search modal (MKLocalSearch) allows users to choose a location for each stop and returns MKMapItem objects with placemarks and geographic coordinates. Predefined tags—Shopping, Food, Drink, and Historical—can be used to group stops. Each tag is displayed as an enum with visually distinct toggle buttons. Users can add personal notes or advice for each location in a freeform text field. The Stop struct is used to internally model these stops, which are then saved as an array. The chosen stop index determines how the user interface changes dynamically. This step allows for the personalization of each location, making the guide more informative and engaging.

4.4.8.3. Draft System and Offline Support. A powerful draft-saving feature that facilitates offline use is part of the guide creation process. Codable is used to serialize all form data into a unique DraftGuide model. When users decide to save a draft, the DraftStorage.save() method is used to encode the data into JSON and store it in UserDefaults. When a draft is detected during view load, it is automatically decoded and brought back into the user interface. This guarantees that user progress is not lost as a result of unintentional app closures or interruptions. After sharing is successful, the draft can be cleared manually or automatically. Usability is greatly enhanced by this offline-first design, particularly in situations with spotty internet connectivity.

4.4.8.4. Guide Upload to Firestore. After a guide is complete, users can use Firebase Firestore to upload it to the cloud. The cover image is initially uploaded to Firebase Storage by the system. The list of stops is then converted to JSON format, which includes structured fields like categories, note, latitude, longitude, placeName, order, and placeName. The entire guide data is stored in two Firestore locations: /users/uid/guides/guideId for user-specific access and /guides/city/items/guideId for public listing. By chunking data client-side, the system also manages Firestore limitations, such as query filters. This architecture guarantees low latency, dependable, traceable, and efficient uploads.

4.4.8.5. Sharing and Saving Experience. At the end of the form, users are presented with two options—Share or Save as Draft. The “Share” button triggers the full Firebase upload process asynchronously, while the “Save as Draft” option allows users to pause and return later. UI states such as isUploading, uploadDone, and showSavedToast manage user feedback through overlays and animations. The sharing experience is responsive and user-friendly, ensuring smooth transitions and a clear understanding of the guide’s status. The system is tightly integrated with SwiftUI’s state-driven design, allowing real-time updates and error handling without disrupting the user flow.

4.4.9. User Search and Follow System

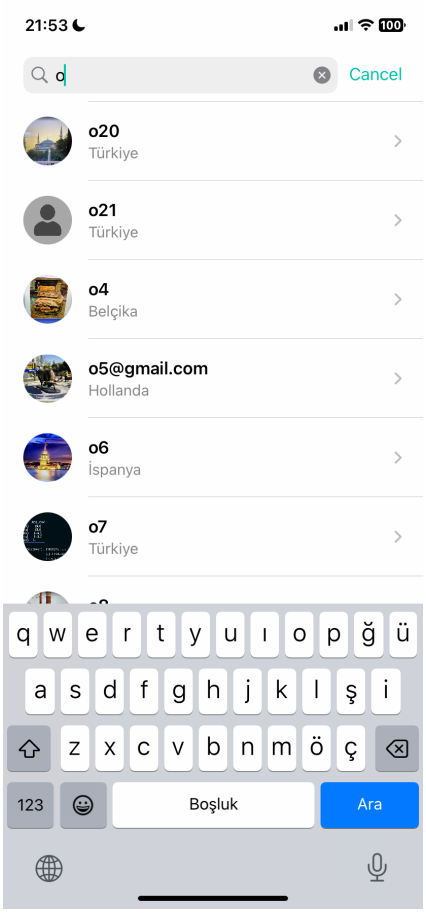

Figure 4.20. Search interface allowing users to look up other members by username prefix.

With the help of this feature, users can look up other people by entering their username and see their public profiles. Real-time search capabilities, responsive user interface feedback, and smooth follow and unfollow integration are all features of the system's design.

The application runs a prefix-based query on Firestore's ”users” collection when a user types text into the search bar. The AuthService class's searchUsers(nickname:) method is in charge of this. In order to filter results, the method employs two range-based conditions: one to determine whether the username is greater than or equal to the nickname that was entered, and another to make sure it is less than or equal to the nickname that is followed by the special character f8ff. All usernames that start with the input string are essentially returned by this. SwiftUI's.searchable modifier, which is connected to a state variable called searchText, is used to implement the search bar on the interface side. An asynchronous fetch operation is started each time the search text is modified. The corresponding users are then shown in a scrollable list, with each item containing the user's username, country for context, and profile

picture (loaded using AsyncImage if it's available, or a default placeholder if not).

Each search result is wrapped in a NavigationLink, allowing the user to tap and view the selected profile in detail through the OtherUserProfileView. On this profile screen, the visitor can see the selected user's shared guides and choose to follow or unfollow them.

The users are modeled as instances of the TGUser struct, which includes identifiers and metadata such as username, country, photoURL, and a list of landmark coins.

The follow system uses Firestore's array operations to update "following" and "followers" fields inside user documents. These updates are handled through Firestore batch operations. When a user follows another, their ID is added to the other user's "followers" list, and vice versa for "following". This ensures atomic updates and reduces latency.

In the background, a snapshot listener keeps the local AuthService state updated with real-time changes to the follow lists. This makes it possible to reflect current follow states dynamically across the app's views without manual refresh. This system, which is closely linked with Firebase's real-time capabilities and SwiftUI's state-driven rendering, promotes social interaction within the application. It facilitates social interaction and community exploration.

5. TEST AND RESULTS

5.1. Landmark Recognition Accuracy Tests

The reserved testing dataset was used to evaluate the trained model's classification performance and generalization abilities. To give a thorough picture of the model's advantages and shortcomings, standard performance metrics were calculated. The model's strong ability to distinguish between a wide variety of landmark classes was highlighted by the final test accuracy, which was roughly 91.99%.

Class	Precision	Recall	F1-score	Support
alman_cesmesi	1.00	0.94	0.97	17
ayasofya	1.00	0.96	0.98	23
bozdogan_kemeri	0.80	0.73	0.76	11
cemberlitas	1.00	0.86	0.93	22
dikilitas	0.90	0.90	0.90	20
galata_kulesi	1.00	0.78	0.88	23
kadikoy_boga_heykeli	0.86	0.75	0.80	16
kiz_kulesi	0.86	1.00	0.92	18
medusa	0.42	1.00	0.59	5
ortakoy_cami	0.97	0.97	0.97	58
sultanahmet	0.83	0.83	0.83	18
taksim_cumhuriyet_aniti	0.96	0.97	0.97	71
topkapi_sarayi	0.93	0.93	0.93	14
yilanli_sutun	0.91	1.00	0.95	21
Accuracy			**0.92**	337
Macro avg	0.89	0.90	0.88	337
Weighted avg	0.93	0.92	0.92	337

Table 5.1. Classification report showing precision, recall, F1-score, and support for each class in the historical site recognition task.

Precision, recall, and F1-score values for every class were highlighted in a comprehensive classification report that was produced. Although the majority of classes performed well on these metrics, some deviations suggested possible areas for improvement. The ”medusa” class, for example, had a perfect recall (1.00) but a lower precision (0.42), indicating that other classes were occasionally misclassified as ”medusa.” However, categories like ”tak-

sim_cumhuriyet_aniti" and "ortakoy_cami" continuously showed high recall and precision, indicating the model's potent discriminative power in those situations.

Precision, recall, and F1-score values for every class were highlighted in a comprehensive classification report that was produced. Although the majority of classes performed well on these metrics, some deviations suggested possible areas for improvement. The model correctly identified all real "medusa" images (no false negatives), but it also mistakenly labeled some images from other classes as "medusa" (false positives). For example, the "medusa" class had a perfect recall (1.00) but a lower precision (0.42). On the other hand, categories like "taksim_cumhuriyet_aniti" and "ortakoy_cami" continuously showed high precision and recall, indicating the model's potent discriminative power and well-rounded performance in those situations.

The distribution of true positives, false positives, false negatives, and true negatives across all classes was visualized using a normalized confusion matrix to better depict these dynamics. Areas where the model displayed confusion, such as pictures of "galata_kulesi" that were occasionally mistakenly classified as "kiz_kulesi," were clearly indicated by the matrix. These incorrect classifications are prime examples of false positives for "kiz_kulesi" and false negatives for "galata_kulesi." These observations offer important information for comprehending the model's shortcomings and offer focused approaches for enhancement, such as improving feature extraction in the model architecture and augmenting data for visually similar classes.

Although the majority of false positives were negligible, the analysis also showed that they tended to group together around classes with comparable visual or architectural characteristics. Similar to this, difficult lighting or occlusion situations where distinguishing characteristics were partially hidden frequently resulted in false negatives. In order to guarantee reliable performance in real-world deployments, this pattern emphasizes the significance of both dataset diversity and real-world scenario testing.

Qualitative analyses of samples that were correctly and incorrectly classified were carried out in addition to quantitative metrics. An analysis of these instances showed that partial occlusions, camera angles, and changes in lighting frequently led to classification errors. In order to increase model robustness, future data collection and augmentation strategies can benefit from an understanding of these patterns.

To assess whether real-time deployment in a mobile application is feasible, inference

latency per image was also measured. The ResNet18 model ensured effective real-time performance appropriate for mobile environments with an average inference latency of roughly 0.14 milliseconds per image.

Collectively, these thorough analyses verify that the ResNet18 model maintains computational efficiency suitable for mobile application deployment while simultaneously delivering strong accuracy and robust generalization on the test dataset. The results indicate that data collection and model refinement will be improved in the future, guaranteeing that the application will continue to be flexible enough to accommodate changing user requirements and technological breakthroughs.

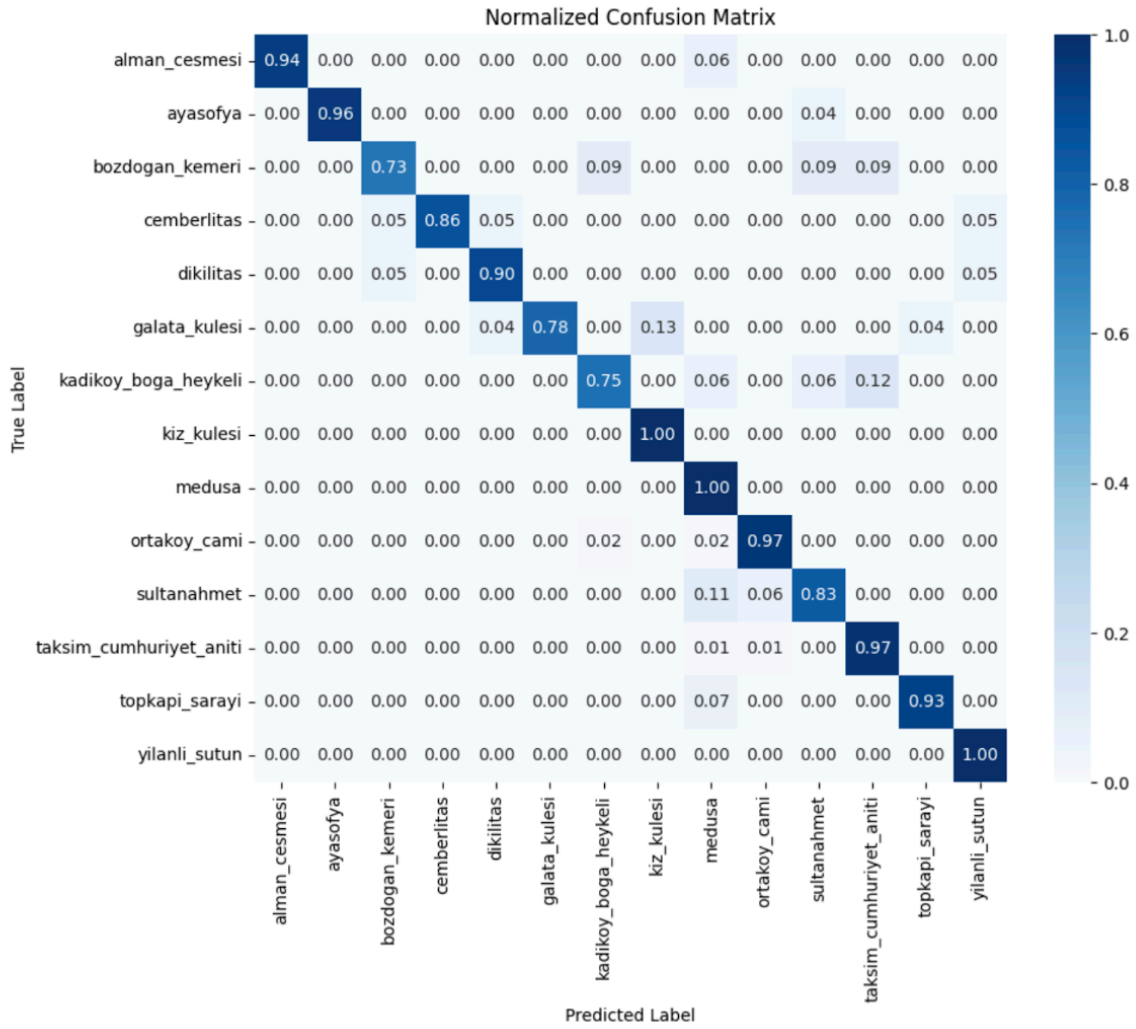

Figure 5.1. ResNet18 normalized confusion matrix on the custom test set

5.2. Load Testing

Using Apache JMeter, a number of load tests were conducted to make sure the Firebase-built backend system could withstand typical usage patterns and scale under pressure. By mimicking user interactions common to the travel guide application, these tests assessed read operations in the Firestore database.

5.2.1. Test Methodology

The first step in the performance testing process was using Postman to obtain authentication tokens, which were subsequently included in HTTP headers in JMeter. The following was the layout of the testing setup:

- **Token Acquisition:** Access tokens were manually retrieved from Firebase via Postman to authenticate requests.
- **JMeter Configuration:** Each load scenario was implemented as a separate thread group in JMeter. Every thread group contained:
  - An HTTP Request Sampler targeting Firebase endpoints,
  - An HTTP Header Manager for authentication and content-type headers.
- **Endpoint Tested:**
  - **Read Test:** This endpoint retrieves travel guide data from Firestore for the Istanbul. Read tests simulate user access to this content under varying load conditions.

5.2.2. Load Scenarios

The system was evaluated under varying numbers of concurrent virtual users to simulate different load levels:

- **Read Tests:** Conducted for 10, 100, and 1000 users. Each test used 50, 500, and 5000 samples respectively, with valid token-based authentication. The latency results reflect per-request measurements.

5.2.3. Test Results

The key observations from the test results are as follows:

- At 10 users, the average latency was 640 ms, with a minimum of 174 ms and maximum of 1577 ms.
- At 100 users, latency increased slightly to 746 ms, with consistent throughput of 36.93 requests/sec.
- At 1000 users, the system maintained responsiveness with 796 ms average latency and a peak throughput of 337.38 requests/sec.
- In all cases, the error rate was 0%, indicating Firebase handled concurrent read loads reliably within tested limits.

Users	Samples	Average (ms)	Min	Max	Std. Dev.	Throughput
10	50	640	174	1577	367.13	3.95
100	500	746	172	1992	356.90	36.93
1000	5000	796	180	2609	325.63	337.38

Table 5.2. Simplified read performance results for different user loads.

5.2.4. Evaluation and Conclusion

Firebase's capacity to facilitate responsive and real-time read operations under both typical and moderately high loads is confirmed by the load tests. Up to 1000 concurrent users, the system showed steady performance with no errors and stable latency.

All things considered, the Firebase Firestore API provided dependable read performance, demonstrating its appropriateness for the travel guide application's practical implementation. These results imply that even during periods of high access, the system can provide a smooth user experience for reading content.

5.3. System Usability Scale

5.3.1. Purpose and Method

A structured usability study was carried out to assess the developed mobile travel application's functional usability and user experience. A modified version of the System Usability Scale (SUS), tailored to the feature set of the app, was used for the evaluation. To guarantee diversity of viewpoints, twelve participants in all, representing a range of age groups and technical backgrounds, were included.

The survey used a 5-point Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree), designed to collect both quantitative ratings and qualitative impressions across aspects such as interface design, user flow, feature utility, and overall satisfaction.

5.3.2. Evaluation Form Structure

General Usability Statements (SUS-inspired):

- I would like to use this application regularly.
- The system felt unnecessarily complicated.
- I found the app interface easy to work with.
- I believe technical assistance would be needed to operate this app.
- The integration between different parts of the app worked well.
- The application showed signs of inconsistency.
- I think new users could quickly learn how to use this system.
- The app felt difficult and awkward to use.
- I felt confident navigating the system.
- There was a steep learning curve before I could effectively use the app.

Feature-Oriented Feedback Items:

- How smooth was the process of creating a travel guide?
- How accurate was the landmark recognition system?
- How effective was the on-site sharing mechanism?
- How well did the social and profile-related features perform?
- How would you rate the overall interface design and usability?

Open Feedback Prompts:

- Which feature did you enjoy using the most?
- Which aspects of the system could be improved further?
- Any additional thoughts or suggestions?

5.3.3. Collected Data Overview

SUS Score Classification:

- Excellent (80–100): 5 participants
- Good (70–79): 4 participants
- Average (60–69): 2 participants
- Below Average (Under 60): 1 participant

Average Ratings per Feature (1 to 5):

- Travel Guide Authoring Experience: 4.5
- Recognition Accuracy for Historical Places: 4.6
- Location-Based Sharing Function: 4.8
- Profile and Social System Usability: 4.3
- Interface and Design Satisfaction: 4.7
- Overall User Experience: 4.65

5.3.4. Thematic Findings from Feedback

Positive Highlights: Participants frequently complimented the application's usability and accessibility. The process of creating the guides was praised by many users as seamless and enriching. The remarkable accuracy and capacity to enrich cultural experiences made the historical site recognition feature stand out. One innovative and genuine layer of interaction was the location-based sharing feature, which requires users to be physically present in order to share comments or images. Customers valued the sense of personalization it gave their travel experiences.

Critiques and Suggestions: Performance optimization in this area is suggested by some users' reports of sporadic delays during image uploads. Others suggested adding more landmarks from various nations to the image recognition dataset. The need for improved offline functionality was also brought up, especially for use when traveling abroad or in areas with poor connectivity.

5.3.5. Conclusion and Recommendations

Key Strengths:

- High usability ratings across all categories
- Clean, minimalistic design that supports intuitive navigation
- Guide creation feature encourages exploration and content sharing
- Accurate image recognition system
- Innovative location-restricted sharing mechanism fosters authenticity and trust

Improvement Opportunities:

- Expand recognition dataset to include more global historical landmarks
- Improve offline usability for travelers without consistent internet access
- Enhance image upload performance, especially for high-resolution media

User Comments (Selected Highlights):

- "Users appreciated how this feature enriched their travel experiences."
- "Users highlighted its impressive accuracy and how it enhanced visits to cultural landmarks."
- "Users found this feature unique and engaging, adding a social and authentic dimension to the experience."
- "Users expressed a positive overall experience with the app."
- "Expand dataset with more global historical sites and improve offline capabilities."

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

This project effectively illustrated the possibilities of combining social media-inspired features with deep learning-based historical site recognition in a mobile travel application. High accuracy was attained by the ResNet-based image classification model, allowing for dependable real-time historical landmark recognition even in a variety of environmental circumstances. The application offered users interested in cultural exploration a smooth and captivating experience by utilizing cutting-edge computer vision techniques within an intuitive SwiftUI interface.

A community-centered approach to travel was promoted by the incorporation of social media features like user profiles, following systems, and location-based pop-up sharing, which promoted genuine interactions and shared experiences. Notably, the use of memory tokens as virtual collectibles increased the incentive and reward for users to visit historical locations. But there were also some areas that needed work. It was difficult to generalize to a wider variety of historical landmarks because of the small dataset and the number of historical site classes. In addition, some discrepancies in precision in specific classes—like the ”medusa” class—emphasized the necessity of more thorough data coverage and focused enhancements to model training.

The project also demonstrated how crucial it is to handle normalization carefully when converting models to Core ML. The Core ML outputs were unnormalized logits instead of probabilities, indicating a normalization inconsistency. This was fixed by adding a custom softmax function to the Swift-based classification module, which made sure that the app's final predictions were comprehensible and normalized. The necessity of paying close attention to details when implementing deep learning models in actual mobile environments was highlighted by this experience.

Lastly, throughout the development process, ethical issues like location-based sharing and user data privacy were taken into account. Building trust and upholding adherence to data privacy best practices depend on making sure that user interactions and sensitive location data are kept safe and transparent.

All things considered, the project achieved its main goals by combining cutting-edge

computer vision methods with creative social features to produce a reliable, approachable, and morally sound platform that enhances the journey for contemporary adventurers.

6.2. Future Work

Building on this strong foundation, a number of important areas for future research have been identified to improve the application's scalability, functionality, and ethical alignment. Adding more historical sites to the dataset, both locally and internationally, is one of the immediate goals. By identifying and narrating websites from around the globe, this will not only enhance the model's generalization abilities but also enable the application to serve a wider audience.

When it comes to feature development, adding augmented reality (AR) technology offers a big chance to improve the trip. For example, AR might make it possible for immersive storytelling to visually superimpose historical narratives or reconstructions onto actual scenes, strengthening users' ties to the culture they are learning about. The creation of an AI-powered recommendation system that makes tailored travel route recommendations based on user preferences, location, and behavior is another possible feature. By enabling users to find new places that suit their particular interests, this system would enhance the application's standing as a reliable travel companion.

Additionally, by providing collectible mementos or "memory coins" connected to visited locations, there is a chance to bring the idea of digital memory tokens into the real world. In addition to bridging the gap between digital and physical exploration, this would give users concrete reminders of their experiences and generate new revenue streams for long-term growth.

Examining lightweight model architectures like MobileNetV3 or pruning strategies for ResNet from a technical standpoint could improve performance on less powerful devices and guarantee that the application is usable and effective on a variety of smartphones.

Future updates will continue to prioritize ethical considerations. As social sharing features and location-based interactions grow, protecting user data privacy and handling location-based content securely will continue to be a top priority. Sustaining user trust and adhering to changing data privacy standards will require clear opt-in procedures, open data practices, and user control over sharing.

When combined, these upcoming additions and improvements will guarantee that the program not only keeps identifying historical sites with accuracy and dependability, but also develops to satisfy the various demands and preferences of contemporary tourists. The project is positioned to have a long-lasting effect on how people find, engage with, and share cultural heritage in the digital age by fusing technological innovation with user-focused design and ethical responsibility.

Bibliography

[1] Google. "Google Lens." Retrieved June 13, 2025. (2023), [Online]. Available: https://lens.google.

[2] R. Filieri, S. Alguezaui, and F. McLeay, "Why do travelers trust tripadvisor? antecedents of trust towards consumer-generated media and its influence on recommendation adoption and word of mouth," *Annals of Tourism Research*, vol. 51, pp. 28–45, 2015. DOI: 10.1016/j.annals.2014.12.004.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, *et al.*, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, https://arxiv.org/abs/1704.04861, 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://arxiv.org/abs/1512.03385.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Figure retrieved from https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248, 2016, pp. 770–778. [Online]. Available: https://arxiv.org/abs/1512.03385.